Algorithmique du texte

Master Infomatique LaBRI

P. Ferraro
pascal.ferraro@labri.fr

LaBRI, Université Bordeaux I

Algorithmique du texte 2007

Plan

Plan

Plan

Objectifs

Plan

Comment NOUS trouvons l'information: exemples

Trouver l'information

Données en entrée:

C A A A C A G T A T C C C C
C A A A G C C G T C T A C A
C G A G A A G A C A T T T T
A A A A A G T T T C G G C T
C C A C G G T T T C C T T C
C C A C A G T T T C G G G C
C A G A G A G G C A A T G T
A C G C A A C G T G G A T T
G C G C G G G T C C C A A C

Motifs connus

A A A
A A A
A
A A A A A
A
A
A A
A A
A A

Occurence approchée: rotation, changement lettre

Comment NOUS trouvons l'information: exemples

Trouver l'information

Données en entrée:

C A A A C A G T A T C C C C
C A A A G C C G T C T A C A
C G A G A A G A C A T T T T
A A A A A G T T T C G G C T
C C A C G G T T T C C T T C
C C A C A G T T T C G G G C
C A G A G A G G C A A T G T
A C G C A A C G T G G A T T
G C G C G G G T C C C A A C

Motifs connus

```
  A A A
  A A A
    A
A A A A A
    A
    A
  A   A
A       A
```

Occurence approchée: rotation, changement lettre

Comment NOUS trouvons l'information: exemples

Trouver l'information

Données en entrée:

C A A A C A G T A T C C C C
C A A A G C C G T C T A C A
C G A G A A G A C A T T T T
A A A A A G T T T C G G C T
C C A C G G T T T C C T T C
C C A C A G T T T C G G G C
C A G A G A G G C A A T G T
A C G C A A C G T G G A T T
G C G C G G G T C C C A A C

Motifs connus

```
  A A A
  A A A
    A
A A A A A
    A
    A
  A   A
A       A
```

Occurence approchée: rotation, changement lettre

Comment NOUS trouvons l'information: exemples

Trouver l'information

Données en entrée:

C A A A C A G T A T C C C C
C A A A G C C G T C T A C A
C G A G A A G A C A T T T T
A A A A A G T T T C G G C T
C C A C G G T T T C C T T C
C C A C A G T T T C G G G C
C A G A G A G G C A A T G T
A C G C A A C G T G G A T T
G C G C G G G T C C C A A C

Motifs connus

```
  A A A
  A A A
    A
A A A A A     C
    A       C C C
    A     C T T T C
  A   A     T T T
A       A   T T T
```

Occurence approchée: rotation, changement lettre

Comment NOUS trouvons l'information: exemples

Trouver l'information

Données en entrée:

```
C A A A C A G T A T C C C C
C A A A G C C G T C T A C A
C G A G G A G A C A T T T T
A A A A A G T T T C G G C T
C C A C G G T T T C C T T C
C C A C A G T T T C G G G C
C A G A G A G G C A A T G T
A C G C A A C G T G G A T T
G C G C G G G T C C C A A C
```

Motifs connus

```
  A A A
  A A A
    A
A A A A A     C
    A       C C C
    A     C T T T C
  A   A     T T T
A       A   T T T
```

Occurence approchée: rotationOccurence approchée: rotation, changement lettre

Comment NOUS trouvons l'information: exemples

Trouver l'information

Données en entrée:

```
C A A A C A G T A T C C C C
C A A A G C C G T C T A C A
C G A G G A G A C A T T T T
A A A A A G T T T C G G C T
C C A C G G T T T C C T T C
C C A C A G T T T C G G G C
C A G A G A G G C A A T G T
A C G C A A C G T G G A T T
G C G C G G G T C C C A A C
```

Motifs connus

```
  A A A
  A A A
    A
A A A A A     C
    A       C C C
    A     C T T T C
  A   A     T T T
A       A   T T T
```

Occurence approchée: rotation, changement lettre

Comment NOUS trouvons l'information: exemples

Trouver l'information

Données en entrée

Comment NOUS trouvons l'information: exemples

Trouver l'information

Données en entrée

C G A A C G G T A A C C C C
C G A A G C C G A A T A C A
C G A A A A A A A A T T T T
A C A G G A A G G A G G C T
C C A A A T T A A A C T T C
C C A A A A A A A A G G G C
C T A A C C C C A A A T G T
A C A A A A A A A A G A T T
G C G C G G G T C C C A A C

Données en entrée

G C A A A C C G A A T A C A
G C A A T T A C A A A T G T
G C A A A A A A A A T A A A
T G A G G A A G G A C C G A
G G A A A T T A A A A T C G
T T A A A A A A A A A G G A G
G C A A C C C C A A T A G A
G G A A A A A A A A A G T A G
C T A T A C C T A G G T T G

Recherche de similarité

Comment NOUS trouvons l'information: exemples

Trouver l'information

Données en entrée

C G A A C G G T A A C C C C
C G A A G C C G A A T A C A
C G A A A A A A A A T T T T
A C A G G A A G G A G G C T
C C A A A T T A A A C T T C
C C A A A A A A A A G G G C
C T A A C C C C A A A T G T
A C A A A A A A A A G A T T
G C G C G G G T C C C A A C

Données en entrée

G C A A A C C G A A T A C A
G C A A T T A C A A A T G T
G C A A A A A A A A T A A A
T G A G G A A G G A C C G A
G G A A A T T A A A A T C G
T T A A A A A A A A G G A G
G C A A C C C C A A T A G A
G G A A A A A A A A G T A G
C T A T A C C T A G G T T G

Recherche de similarité

Comment NOUS trouvons l'information: exemples

Trouver l'information

Données en entrée

C G A A C G G T A A C C C C
C G A A G C C G A A T A C A
C G A A A A A A A A T T T T
A C A G G A A G G A G G C T
C C A A A T T A A A C T T C
C C A A A A A A A A G G G C
C T A A C C C C A A A T G T
A C A A A A A A A A G A T T
G C G C G G G T C C C A A C

Données en entrée

G C A A A C C G A A T A C A
G C A A T T A C A A A T G T
G C A A A A A A A A T A A A
T G A G G A A G G A C C G A
G G A A A T T A A A A T C G
T T A A A A A A A A G G A G
G C A A C C C C A A T A G A
G G A A A A A A A A G T A G
C T A T A C C T A G G T T G

Recherche de similarité

Comment NOUS trouvons l'information: exemples

Trouver l'information

Données en entrée

C G A A C A A T A A C C C C
C G A A G A A G A A T A C A
C G A A A A A A A A T T T T
A C A G G A A G G A G G C T
C C A A A T T A A A C T T C
C C A A A A A A A A G G G C
C T A A C C C C A A A T G T
A C A A A A A A A A G A T T
G C G C G G G T C C C A A C

Recherche de différences

Comment NOUS trouvons l'information: exemples

Trouver l'information

Données en entrée

C G A A C A A T A A C C C C
C G A A G A A G A A T A C A
C G A A A A A A A A T T T T
A C A G G A A G G A G G C T
C C A A A T T A A A C T T C
C C A A A A A A A A G G G C
C T A A C C C C A A A T G T
A C A A A A A A A A G A T T
G C G C G G G T C C C A A C

Données en entrée

C G A A C G G T A A C C C C
C G A A G C C G A A T A C A
C G A A A A A A A A T T T T
A C A G G A A G G A G G C T
C C A A A T T A A A C T T C
C C A A A A A A A A G G G C
C T A A C C C C A A A T G T
A C A A A A A A A A G A T T
G C G C G G G T C C C A A C

Recherche de différences

Comment NOUS trouvons l'information: exemples

Trouver l'information

Données en entrée

C G A A C A A T A A C C C C
C G A A G A A G A A T A C A
C G A A A A A A A A T T T T
A C A G G A A G G A G G C T
C C A A A T T A A A C T T C
C C A A A A A A A A G G G C
C T A A C C C C A A A T G T
A C A A A A A A A A G A T T
G C G C G G G T C C C A A C

Données en entrée

C G A A C G G T A A C C C C
C G A A G C C G A A T A C A
C G A A A A A A A A T T T T
A C A G G A A G G A G G C T
C C A A A T T A A A C T T C
C C A A A A A A A A G G G C
C T A A C C C C A A A T G T
A C A A A A A A A A G A T T
G C G C G G G T C C C A A C

Recherche de différences

Comment NOUS trouvons l'information: exemples

Trouver l'information

Données en entrée

C G A A C A A T A A C C C C
C G A A G A A G A A T A C A
C G A A A A A A A A T T T T
A C A G G A A G G A G G C T
C C A A A T T A A A C T T C
C C A A A A A A A A G G G C
C T A A C C C C A A A T G T
A C A A A A A A A A G A T T
G C G C G G G T C C C A A C

Données en entrée

C G A A C G G T A A C C C C
C G A A G C C G A A T A C A
C G A A A A A A A A T T T T
A C A G G A A G G A G G C T
C C A A A T T A A A C T T C
C C A A A A A A A A G G G C
C T A A C C C C A A A T G T
A C A A A A A A A A G A T T
G C G C G G G T C C C A A C

Recherche de différences

Plan

Comment trouver du sens à partir d'informations connues

Motifs

- Nous avons des connaissances
- Nous utilisons ces connaissances pour explorer les données
- Pour chaque motif (connaissance), nous recherchons celui-ci dans les données:
  $\Rightarrow$ Recherche de motif
- Possiblité d'erreurs dans les occurences:
  $\Rightarrow$ Recherche approchée de motif

Comment trouver du sens sans information:
Similarité

- Nous disposons d'au moins deux données en entrée
- Nous recherchons des informations communes à ces données:
  $\Rightarrow$ Inférence de motifs

Comment trouver du sens sans information:

- Nous disposons d'au moins deux données en entrée
- Nous recherchons des différences entre ces données:
  $\Rightarrow$ Inférence de motifs

Plan

Recherche exacte de motifs

- Pour une seule recherche on utilisera un algorithme de recherche tel que KMP
- Pour plusieurs recherche on construit un index de ce texte (prochaine section)

Recherche exacte de motifs

- Pour une seule recherche on utilisera un algorithme de recherche tel que KMP
- Pour plusieurs recherche on construit un index de ce texte (prochaine section)

Recherche approchée de motifs

Nous cherchons des occurrences *approchées* d'un motif:
$\Rightarrow$ quelles types d'approximations ?

- Distance de Hamming
- Distance d'édition
- Alignement

Recherche approchée de motifs

Nous cherchons des occurrences *approchées* d'un motif:
$\Rightarrow$ quelles types d'approximations ?

- Distance de Hamming
- Distance d'édition
- Alignement

Recherche approchée de motifs

Nous cherchons des occurrences *approchées* d'un motif:
$\Rightarrow$ quelles types d'approximations ?

- Distance de Hamming
- Distance d'édition
- Alignement

Recherche approchée de motifs

Nous cherchons des occurrences *approchées* d'un motif:
$\Rightarrow$ quelles types d'approximations ?

- Distance de Hamming
- Distance d'édition
- Alignement

Définitions

- On appelera indifférement séquence, texte, mot ou chaîne une suite de symbole sur un alphabet donné Σ.
- Soit t un texte de longueur n. On notera le i^{eme} symbole de t, t_i.
- Soient deux mots u et v de longueurs n et m. On note $u.v$ le mot de longueur $n+m$ resultant de la concaténation de u et v: $u_0 ... u_n v_0 ... v_m$
- w (non nul) est un facteur de t s'il existe deux mots u et v éventuellement de longueurs nulles tels que $u.w.v = t$. Si $u = \epsilon$, alors w est une prefixe. Si $v = \epsilon$, alors w est un suffixe.
- w de longueur m est un facteur de t s'il existe une position $i < n - m$ telle que $\forall j \in [0, m[,\ w_j = t_{i+j}$

Définitions

- On appelera indifférement séquence, texte, mot ou chaîne une suite de symbole sur un alphabet donné Σ.
- Soit t un texte de longueur n. On notera le i^{eme} symbole de t, t_i.
- Soient deux mots u et v de longueurs n et m. On note $u.v$ le mot de longueur $n+m$ resultant de la concaténation de u et v: $u_0 ... u_n v_0 ... v_m$
- w (non nul) est un facteur de t s'il existe deux mots u et v éventuellement de longueurs nulles tels que $u.w.v = t$. Si $u = \epsilon$, alors w est une prefixe. Si $v = \epsilon$, alors w est un suffixe.
- w de longueur m est un facteur de t s'il existe une position $i < n - m$ telle que $\forall j \in [0, m[,\ w_j = t_{i+j}$

Définitions

- On appelera indifférement séquence, texte, mot ou chaîne une suite de symbole sur un alphabet donné Σ.
- Soit t un texte de longueur n. On notera le i^{eme} symbole de t, t_i.
- Soient deux mots u et v de longueurs n et m. On note $u.v$ le mot de longueur $n+m$ resultant de la concaténation de u et v: $u_0 \ldots u_n v_0 \ldots v_m$
- w (non nul) est un facteur de t s'il existe deux mots u et v éventuellement de longueurs nulles tels que $u.w.v = t$. Si $u = \epsilon$, alors w est une prefixe. Si $v = \epsilon$, alors w est un suffixe.
- w de longueur m est un facteur de t s'il existe une position $i < n - m$ telle que $\forall j \in [0, m[,\ w_j = t_{i+j}$

Définitions

- On appelera indifférement séquence, texte, mot ou chaîne une suite de symbole sur un alphabet donné Σ.
- Soit t un texte de longueur n. On notera le i^{eme} symbole de t, t_i.
- Soient deux mots u et v de longueurs n et m. On note $u.v$ le mot de longueur $n+m$ resultant de la concaténation de u et v: $u_0 ... u_n v_0 ... v_m$
- w (non nul) est un facteur de t s'il existe deux mots u et v éventuellement de longueurs nulles tels que $u.w.v = t$. Si $u = \epsilon$, alors w est une prefixe. Si $v = \epsilon$, alors w est un suffixe.
- w de longueur m est un facteur de t s'il existe une position $i < n - m$ telle que $\forall j \in [0, m[,\ w_j = t_{i+j}$

Définitions

- On appelera indifférement séquence, texte, mot ou chaîne une suite de symbole sur un alphabet donné Σ.
- Soit t un texte de longueur n. On notera le i^{eme} symbole de t, t_i.
- Soient deux mots u et v de longueurs n et m. On note $u.v$ le mot de longueur $n+m$ resultant de la concaténation de u et v: $u_0 \ldots u_n v_0 \ldots v_m$
- w (non nul) est un facteur de t s'il existe deux mots u et v éventuellement de longueurs nulles tels que $u.w.v = t$. Si $u = \epsilon$, alors w est une prefixe. Si $v = \epsilon$, alors w est un suffixe.
- w de longueur m est un facteur de t s'il existe une position $i < n - m$ telle que $\forall j \in [0, m[,\ w_j = t_{i+j}$

Distance de Hamming: exemple

Compter le nombre de symboles différents entre deux textes de même taille.

	l	i	s	b	o	a		
Exemple:	v	i	s	i	o	n		
	1	0	0	1	0	1	=	3

Le distance de hamming entre *lisboa* et *vision* est 3

Distance de Hamming: exemple

Compter le nombre de symboles différents entre deux textes de même taille.

	l	i	s	b	o	a	l	i	s	b	o	a		
Exemple:	v	i	s	i	o	n	v	i	s	i	o	n		
							1	0	0	1	0	1	=	3

Le distance de hamming entre *lisboa* et *vision* est 3

Distance de Hamming: exemple

Compter le nombre de symboles différents entre deux textes de même taille.

	l	i	s	b	o	a		l	i	s	b	o	a					
Exemple:	v	i	s	i	o	n		v	i	s	i	o	n					
								1	0	0	1	0	1	=	3			

Le distance de hamming entre *lisboa* et *vision* est 3

Distance de Hamming: exemple

Compter le nombre de symboles différents entre deux textes de même taille.

	l	i	s	b	o	a		l	i	s	b	o	a	
Exemple:	v	i	s	i	o	n		v	i	s	i	o	n	
	1	0	0	1	0	1		1	0	0	1	0	1	= 3

Le distance de hamming entre *lisboa* et *vision* est 3

Distance de Hamming: exemple

Compter le nombre de symboles différents entre deux textes de même taille.

	l	i	s	b	o	a		
Exemple:	v	i	s	i	o	n		
	1	0	0	1	0	1	=	3

Le distance de hamming entre *lisboa* et *vision* est 3

Distance de Hamming: définition

Definition (Distance de Hamming)

Distance de Hamming: exercice

- Écrire une fonction `hamming(s,t)` qui calcule la distance de Hamming entre s et t si ils sont de même taille et retourne *None* sinon.
- Écrire une fonction `hamming_factors(p,t)` qui prend en arguments deux chaînes p et t de taille m et n ($m < n$). Cette fonction fournit la liste des positions i telles que $Ham(p, t_{i \ldots i+m})$ est minimale sur l'ensemble des facteurs de t.
- Quelles sont les complexités en temps de ces deux fonctions.

Distance de Hamming: exercice

- Écrire une fonction `hamming(s,t)` qui calcule la distance de Hamming entre s et t si ils sont de même taille et retourne *None* sinon.
- Écrire une fonction `hamming_factors(p,t)` qui prend en arguments deux chaînes p et t de taille m et n ($m < n$). Cette fonction fournit la liste des positions i telles que $Ham(p, t_{i \ldots i+m})$ est minimale sur l'ensemble des facteurs de t.
- Quelles sont les complexités en temps de ces deux fonctions.

Distance de Hamming: exercice

- Écrire une fonction `hamming(s,t)` qui calcule la distance de Hamming entre s et t si ils sont de même taille et retourne *None* sinon.
- Écrire une fonction `hamming_factors(p,t)` qui prend en arguments deux chaînes p et t de taille m et n ($m < n$). Cette fonction fournit la liste des positions i telles que $Ham(p, t_{i \ldots i+m})$ est minimale sur l'ensemble des facteurs de t.
- Quelles sont les complexités en temps de ces deux fonctions.

Distance d'édition/Levenstein

- La distance de Hamming est trop simple pour comparer deux séquences. En effet elle ne prend pas en compte certains événements tels que l'insertion et la délétion de base:

$$Ham(examples, exxample) = 6 = Ham(examples, exercise)$$

- La distance d'édition entre deux chaînes consiste à trouver une suite d'opérations atomiques (dites d'édition) de coût minimal permettant de transformer la première chaîne en la deuxième.

Distance d'édition/Levenstein

- La distance de Hamming est trop simple pour comparer deux séquences. En effet elle ne prend pas en compte certains événements tels que l'insertion et la délétion de base:

$$Ham(examples, exxample) = 6 = Ham(examples, exercise)$$

- La distance d'édition entre deux chaînes consiste à trouver une suite d'opérations atomiques (dites d'édition) de coût minimal permettant de transformer la première chaîne en la deuxième.

Distance d'édition/Levenstein

3 événements atomiques:

- la substitution: change une lettre de t en une autre
- la délétion: supprime une lettre de t à une position donnée
- l'insertion: insert une lettre dans t à une position donnée

Soient deux séquences t et u, on recherche l'ensemble des suites d'opérations d'édition $S = \{s_1, s_2, \dots\}$ telles que si on applique successivement s_1, s_2 ... à t on obtient u.

On dira que S réalise l'édition de t en u et on notera $t \overset{S}{\rightsquigarrow} u$

Distance d'édition/Levenstein

3 événements atomiques:

- la substitution: change une lettre de t en une autre
- la délétion: supprime une lettre de t à une position donnée
- l'insertion: insert une lettre dans t à une position donnée

Soient deux séquences t et u, on recherche l'ensemble des suites d'opérations d'édition $S = \{s_1, s_2, \dots\}$ telles que si on applique successivement s_1, s_2 ... à t on obtient u.

On dira que S réalise l'édition de t en u et on notera $t \overset{S}{\rightsquigarrow} u$

Distance d'édition/Levenstein

3 événements atomiques:

- la substitution: change une lettre de t en une autre
- la délétion: supprime une lettre de t à une position donnée
- l'insertion: insert une lettre dans t à une position donnée

Soient deux séquences t et u, on recherche l'ensemble des suites d'opérations d'édition $S = \{s_1, s_2, \dots\}$ telles que si on applique successivement s_1, s_2 ... à t on obtient u.

On dira que S réalise l'édition de t en u et on notera $t \overset{S}{\rightsquigarrow} u$

Distance d'édition/Levenstein

- on assigne un coût à chaque opération d'édition: c_{sub}, c_{del} et c_{ins} tels que s_{sub} est une distance et $c_{del}(a) = c_{ins}(a)$
- Le coût $c(S)$ d'une série d'opérations d'édition $S = \{s_1, s_2, \dots\}$ est défini par la somme des coûts de chaque opération.
- La distance d'édition entre u et v est définie par:

$$d(t,u) = min\{c(S)/t \overset{S}{\rightsquigarrow} u\}$$

Levenshtein, *Binary codes capable of correcting deletions, insertions, and reversals*,1965

Distance d'édition/Levenstein

- on assigne un coût à chaque opération d'édition: c_{sub}, c_{del} et c_{ins} tels que s_{sub} est une distance et $c_{del}(a) = c_{ins}(a)$
- Le coût $c(S)$ d'une série d'opérations d'édition $S = \{s_1, s_2, \dots\}$ est défini par la somme des coûts de chaque opération.
- La distance d'édition entre u et v est définie par:

$$d(t,u) = min\{c(S)/t \overset{S}{\rightsquigarrow} u\}$$

Levenshtein, *Binary codes capable of correcting deletions, insertions, and reversals*,1965

Distance d'édition/Levenstein

- on assigne un coût à chaque opération d'édition: c_{sub}, c_{del} et c_{ins} tels que s_{sub} est une distance et $c_{del}(a) = c_{ins}(a)$
- Le coût $c(S)$ d'une série d'opérations d'édition $S = \{s_1, s_2, \dots\}$ est défini par la somme des coûts de chaque opération.
- La distance d'édition entre u et v est définie par:

$$d(t,u) = min\{c(S)/t \overset{S}{\rightsquigarrow} u\}$$

Levenshtein, *Binary codes capable of correcting deletions, insertions, and reversals*,1965

Distance d'édition/Levenstein

Lemma ($d(t,u)$ est une distance)

$d(t,u)$ est une distance si et seulement si c_{sub} est une distance et $c_{ins}(a) = c_{del}(a) > 0$ pour tout $a \in \Sigma$.

Proof.

$\Leftarrow$ Inégalité triangulaire:

Supposons qu'il existe trois mots tels que $d(u,t) > d(u,v) + d(v,t)$:

$u \overset{S_{uv}}{\rightsquigarrow} v$

$v \overset{S_{vt}}{\rightsquigarrow} t$

La série $S = S_{uv}S_{vt}$ transforme u en t et a pour coût $d(u,v) + d(v,t) < d(u,t)$!

□

Distance d'édition/Levenstein

Lemma ($d(t,u)$ est une distance)

$d(t,u)$ est une distance si et seulement si c_{sub} est une distance et $c_{ins}(a) = c_{del}(a) > 0$ pour tout $a \in \Sigma$.

Proof.

$\Rightarrow$ $d(t,u)$ est une distance donc, $\forall a, b \in \Sigma$:

- $d(a,b) = c_{sub(a,b)}$ est une distance
- $d(a,\epsilon) = c_{del}(a) = d(\epsilon,a) = c_{ins}(a)$ (positivité et symétrie)

$\Leftarrow$ Inégalité triangulaire:

Supposons qu'il existe trois mots tels que $d(u,t) > d(u,v) + d(v,t)$:

- $u \overset{S_{uv}}{\rightsquigarrow} v$
- $v \overset{S_{vt}}{\rightsquigarrow} t$
- La série $S = S_{uv}S_{vt}$ transforme u en t et a pour coût $d(u,v) + d(v,t) < d(u,t)$!

Distance d'édition/Levenstein

Lemma ($d(t,u)$ est une distance)

$d(t,u)$ est une distance si et seulement si c_{sub} est une distance et $c_{ins}(a) = c_{del}(a) > 0$ pour tout $a \in \Sigma$.

Proof.

$\Leftarrow$ Positivity, $\forall a, b \in \Sigma$:

- $c_{sub}(a,b) \geq 0$
- $c_{del}(a) > 0$
- $c_{ins}(a) > 0$

$\Leftarrow$ Inégalité triangulaire:

Supposons qu'il existe trois mots tels que $d(u,t) > d(u,v) + d(v,t)$:

- $u \overset{S_{uv}}{\rightsquigarrow} v$
- $v \overset{S_{vt}}{\rightsquigarrow} t$
- La série $S_{uv} \cdot S_{vt}$ transforme u en t et a pour coût

Distance d'édition/Levenstein

Lemma ($d(t,u)$ est une distance)

$d(t,u)$ est une distance si et seulement si c_{sub} est une distance et $c_{ins}(a) = c_{del}(a) > 0$ pour tout $a \in \Sigma$.

Proof.

$\Leftarrow$ Séparation:
Si $t = u$, il est clair que $d(t,u) = 0$. Si $d(t,u) = 0$ alors $t = u$ car seule c_{sub} peut être nulle. $\Leftarrow$ Inégalité triangulaire:
Supposons qu'il existe trois mots tels que $d(u,t) > d(u,v) + d(v,t)$:

- $u \overset{S_{uv}}{\rightsquigarrow} v$
- $v \overset{S_{vt}}{\rightsquigarrow} t$
- La série $S = S_{uv}S_{vt}$ transforme u en t et a pour coût $d(u,v) + d(v,t) < d(u,t)$!

□

Distance d'édition/Levenstein

Lemma ($d(t,u)$ est une distance)

$d(t,u)$ est une distance si et seulement si c_{sub} est une distance et $c_{ins}(a) = c_{del}(a) > 0$ pour tout $a \in \Sigma$.

Proof.

$\Leftarrow$ Symétrie:
c_{sub} est symétrique et $c_{del} = c_{ins}$ donc pour toute série d'opération S qui transforme t en u, on peut construire une série S' qui transforme u en t et qui a le même coût que S. $\Leftarrow$ Inégalité triangulaire:
Supposons qu'il existe trois mots tels que $d(u,t) > d(u,v) + d(v,t)$:

- $u \overset{S_{uv}}{\rightsquigarrow} v$
- $v \overset{S_{vt}}{\rightsquigarrow} t$
- La série $S = S_{uv}S_{vt}$ transforme u en t et a pour coût $d(u,v) + d(v,t) < d(u,t)$!

Distance d'édition/Levenstein

Lemma ($d(t,u)$ est une distance)

$d(t,u)$ est une distance si et seulement si c_{sub} est une distance et $c_{ins}(a) = c_{del}(a) > 0$ pour tout $a \in \Sigma$.

Proof.

$\Leftarrow$ Inégalité triangulaire:
Supposons qu'il existe trois mots tels que $d(u,t) > d(u,v) + d(v,t)$:

- Appelons S_{uv} la série d'opérations qui transforme u en v et qui a pour coût $d(u,v)$.
- $u \overset{S_{uv}}{\rightsquigarrow} v$
- $v \overset{S_{vt}}{\rightsquigarrow} t$
- La série $S = S_{uv}S_{vt}$ transforme u en t et a pour coût $d(u,v) + d(v,t) < d(u,t)$!

Distance d'édition/Levenstein

Lemma ($d(t,u)$ est une distance)

$d(t,u)$ est une distance si et seulement si c_{sub} est une distance et $c_{ins}(a) = c_{del}(a) > 0$ pour tout $a \in \Sigma$.

Proof.

$\Leftarrow$ Inégalité triangulaire:
Supposons qu'il existe trois mots tels que $d(u,t) > d(u,v) + d(v,t)$:

- $u \overset{S_{uv}}{\rightsquigarrow} v$
- Appelons S_{vt} la série d'opérations d'éditions qui transforme v en t et qui a pour coût $d(v,t)$.
- $v \overset{S_{vt}}{\rightsquigarrow} t$
- La série $S = S_{uv}S_{vt}$ transforme u en t et a pour coût $d(u,v) + d(v,t) < d(u,t)$!

Distance d'édition/Levenstein

Lemma ($d(t,u)$ est une distance)

$d(t,u)$ est une distance si et seulement si c_{sub} est une distance et $c_{ins}(a) = c_{del}(a) > 0$ pour tout $a \in \Sigma$.

Proof.

$\Leftarrow$ Inégalité triangulaire:
Supposons qu'il existe trois mots tels que $d(u,t) > d(u,v) + d(v,t)$:

- $u \overset{S_{uv}}{\rightsquigarrow} v$
- $v \overset{S_{vt}}{\rightsquigarrow} t$
- La série $S = S_{uv}S_{vt}$ transforme u en t et a pour coût $d(u,v) + d(v,t) < d(u,t)$!

□

Alignement

Definition (Alignement)

Soient deux séquences u et t de tailles m et n sur Σ.
Un alignement de u et t consiste en un couple de deux séquences u' et t' de même taille l sur $\Sigma \cup \{-\}$ telles que:

- Si on supprime tout les $-$ de u' on obtient u.
- Si on supprime tout les $-$ de t' on obtient t.
- $\forall i$ si $u'_i =' -'$ alors $v'_i \neq' -'$

Exemple: alignement de ”publics” and ”nucleic”:

Alignement

Definition (Alignement)

Soient deux séquences u et t de tailles m et n sur Σ.
Un alignement de u et t consiste en un couple de deux séquences u' et t' de même taille l sur $\Sigma \cup \{-\}$ telles que:

- Si on supprime tout les $-$ de u' on obtient u.
- Si on supprime tout les $-$ de t' on obtient t.
- $\forall i$ si $u'_i =' -'$ alors $v'_i \neq' -'$

Exemple: alignement de ”publics” and ”nucleic”:

Alignement

Definition (Alignement)

Soient deux séquences u et t de tailles m et n sur Σ.
Un alignement de u et t consiste en un couple de deux séquences u' et t' de même taille l sur $\Sigma \cup \{-\}$ telles que:

- Si on supprime tout les $-$ de u' on obtient u.
- Si on supprime tout les $-$ de t' on obtient t.
- $\forall i$ si $u'_i =' -'$ alors $v'_i \neq' -'$

Exemple: alignement de ”publics” and ”nucleic”:

p	u	b	l	-	i	c	s
n	u	c	l	e	i	c	-

Alignement

Definition (Alignement)

Soient deux séquences u et t de tailles m et n sur Σ.
Un alignement de u et t consiste en un couple de deux séquences u' et t' de même taille l sur $\Sigma \cup \{-\}$ telles que:

- Si on supprime tout les $-$ de u' on obtient u.
- Si on supprime tout les $-$ de t' on obtient t.
- $\forall i$ si $u'_i =' -'$ alors $v'_i \neq' -'$

Exemple: alignement de ”publics” and ”nucleic”:

p	-	u	b	-	l	i	c	-	s
-	n	u	c	l	-	e	i	c	-

Alignement

Fonction de coût s: $\forall a, b \in \Sigma$,

$$\begin{array}{lll} s(a,b) & = 2 & \text{if } a = b \\ s(a,b) & = -1 & \text{if } a \neq b \\ s(a,-) & = s(-,a) & = -1 \end{array}$$

Le but est de maximiser le score de l'alignement des deux séquences.

Exemple: alignement de "publics" et "nucleic":

Alignement

Fonction de coût s: $\forall a, b \in \Sigma$,

$$\begin{array}{lll} s(a,b) & = 2 & \text{if } a = b \\ s(a,b) & = -1 & \text{if } a \neq b \\ s(a,-) & = s(-,a) & = -1 \end{array}$$

Le but est de maximiser le score de l'alignement des deux séquences.

Alignement

Fonction de coût s: $\forall a, b \in \Sigma$,

$$\begin{array}{lll} s(a,b) & = 2 & \text{if } a = b \\ s(a,b) & = -1 & \text{if } a \neq b \\ s(a,-) & = s(-,a) & = -1 \end{array}$$

Le but est de maximiser le score de l'alignement des deux séquences.

Exemple: alignement de "publics" et "nucleic":

p	u	b	l	-	i	c	s	
n	u	c	l	e	i	c	-	le score est 4
-1	2	-1	2	-1	2	2	-1	

Alignement

Fonction de coût s: $\forall a, b \in \Sigma$,

$$\begin{array}{lll} s(a,b) & = 2 & \text{if } a = b \\ s(a,b) & = -1 & \text{if } a \neq b \\ s(a,-) & = s(-,a) & = -1 \end{array}$$

Le but est de maximiser le score de l'alignement des deux séquences.

Exemple: alignement de "publics" et "nucleic":

p	-	u	b	-	l	i	c	-	s	
-	n	u	c	l	-	e	i	c	-	le score est -7
-1	-1	2	-1	-1	-1	-1	-1	-1	-1	

Edit vs Alignement

Pour toutes les séries d'opérations d'édition qui transforment u en t, on peut calculer un alignement valide

Edition

- distance
- fournie une distance et une suite d'opérations
- uniquement pour 2 séquences

Alignement

- mesure de similarité
- score + un alignement
- alignement multiple

Si on utilise les fonctions de scores de l'édition dans l'alignement et que l'on essaye de minimiser le score de l'alignement, alors on calcul la distance d'édition.

Edit vs Alignement

Pour toutes les séries d'opérations d'édition qui transforment u en t, on peut calculer un alignement valide

Edition

- distance
- fournie une distance et une suite d'opérations
- uniquement pour 2 séquences

Alignement

- mesure de similarité
- score + un alignement
- alignement multiple

Si on utilise les fonctions de scores de l'édition dans l'alignement et

Edit vs Alignement

Pour toutes les séries d'opérations d'édition qui transforment u en t, on peut calculer un alignement valide

Edition

- distance
- fournie une distance et une suite d'opérations
- uniquement pour 2 séquences

Alignement

- mesure de similarité
- score + un alignement
- alignement multiple

Si on utilise les fonctions de scores de l'édition dans l'alignement et que l'on essaye de minimiser le score de l'alignement, alors on calcul la distance d'édition.

Edit vs Alignement

Pour toutes les séries d'opérations d'édition qui transforment u en t, on peut calculer un alignement valide

Edition

- distance
- fournie une distance et une suite d'opérations
- uniquement pour 2 séquences

Alignement

- mesure de similarité
- score + un alignement
- alignement multiple

Si on utilise les fonctions de scores de l'édition dans l'alignement et que l'on essaye de minimiser le score de l'alignement, alors on calcul la distance d'édition.

Edit vs Alignement

Pour toutes les séries d'opérations d'édition qui transforment u en t, on peut calculer un alignement valide

Edition

Plan

Formule

L'idée principale est de calculer $D(u_{0\ldots i}, t_{0\ldots j})$ à partir des distances $D(u_{0\ldots i-1}, t_{0\ldots j})$, $D(u_{0\ldots i}, t_{0\ldots j-1})$ et $D(u_{0\ldots i-1}, t_{0\ldots j-1})$

$$D(u_{0\ldots i}, t_{0\ldots j}) = Min \begin{cases} D(u_{0\ldots i-1}, t_{0\ldots j-1}) + c_{sub}(u_i, t_j) \\ D(u_{0\ldots i-1}, t_{0\ldots j}) + c_{del}(u_i) \\ D(u_{0\ldots i}, t_{0\ldots j-1}) + c_{ins}(t_j) \end{cases}$$

Quelle est la complexité en temps d'une implémentation naïve de cette formule $O(3^{n+m})$!

Formule

L'idée principale est de calculer $D(u_{0\ldots i}, t_{0\ldots j})$ à partir des distances $D(u_{0\ldots i-1}, t_{0\ldots j})$, $D(u_{0\ldots i}, t_{0\ldots j-1})$ et $D(u_{0\ldots i-1}, t_{0\ldots j-1})$

$$D(u_{0\ldots i}, t_{0\ldots j}) = Min \begin{cases} D(u_{0\ldots i-1}, t_{0\ldots j-1}) + c_{sub}(u_i, t_j) \\ D(u_{0\ldots i-1}, t_{0\ldots j}) + c_{del}(u_i) \\ D(u_{0\ldots i}, t_{0\ldots j-1}) + c_{ins}(t_j) \end{cases}$$

Quelle est la complexité en temps d'une implémentation naïve de cette formule $O(3^{n+m})$!

Formule

L'idée principale est de calculer $D(u_{0\ldots i}, t_{0\ldots j})$ à partir des distances $D(u_{0\ldots i-1}, t_{0\ldots j})$, $D(u_{0\ldots i}, t_{0\ldots j-1})$ et $D(u_{0\ldots i-1}, t_{0\ldots j-1})$

$$D(u_{0\ldots i}, t_{0\ldots j}) = Min \begin{cases} D(u_{0\ldots i-1}, t_{0\ldots j-1}) + c_{sub}(u_i, t_j) \\ D(u_{0\ldots i-1}, t_{0\ldots j}) + c_{del}(u_i) \\ D(u_{0\ldots i}, t_{0\ldots j-1}) + c_{ins}(t_j) \end{cases}$$

Quelle est la complexité en temps d'une implémentation naïve de cette formule $O(3^{n+m})$!

Formule

L'idée principale est de calculer $D(u_{0\ldots i}, t_{0\ldots j})$ à partir des distances $D(u_{0\ldots i-1}, t_{0\ldots j})$, $D(u_{0\ldots i}, t_{0\ldots j-1})$ et $D(u_{0\ldots i-1}, t_{0\ldots j-1})$

$$D(u_{0\ldots i}, t_{0\ldots j}) = Min \begin{cases} D(u_{0\ldots i-1}, t_{0\ldots j-1}) + c_{sub}(u_i, t_j) \\ D(u_{0\ldots i-1}, t_{0\ldots j}) + c_{del}(u_i) \\ D(u_{0\ldots i}, t_{0\ldots j-1}) + c_{ins}(t_j) \end{cases}$$

Quelle est la complexité en temps d'une implémentation naïve de cette formule $O(3^{n+m})$!

Formule

L'idée principale est de calculer $D(u_{0\ldots i}, t_{0\ldots j})$ à partir des distances $D(u_{0\ldots i-1}, t_{0\ldots j})$, $D(u_{0\ldots i}, t_{0\ldots j-1})$ et $D(u_{0\ldots i-1}, t_{0\ldots j-1})$

$$D(u_{0\ldots i}, t_{0\ldots j}) = Min \begin{cases} D(u_{0\ldots i-1}, t_{0\ldots j-1}) + c_{sub}(u_i, t_j) \\ D(u_{0\ldots i-1}, t_{0\ldots j}) + c_{del}(u_i) \\ D(u_{0\ldots i}, t_{0\ldots j-1}) + c_{ins}(t_j) \end{cases}$$

Quelle est la complexité en temps d'une implémentation naïve de cette formule $O(3^{n+m})$!

Formule

L'idée principale est de calculer $D(u_{0\ldots i}, t_{0\ldots j})$ à partir des distances $D(u_{0\ldots i-1}, t_{0\ldots j})$, $D(u_{0\ldots i}, t_{0\ldots j-1})$ et $D(u_{0\ldots i-1}, t_{0\ldots j-1})$

$$D(u_{0\ldots i}, t_{0\ldots j}) = Min \begin{cases} D(u_{0\ldots i-1}, t_{0\ldots j-1}) + c_{sub}(u_i, t_j) \\ D(u_{0\ldots i-1}, t_{0\ldots j}) + c_{del}(u_i) \\ D(u_{0\ldots i}, t_{0\ldots j-1}) + c_{ins}(t_j) \end{cases}$$

Quelle est la complexité en temps d'une implémentation naïve de cette formule $O(3^{n+m})$!

Programmation dynamique

Idée: utiliser un algorithme de programmation dynamique:

- Calculer des problèmes les plus ”petits” vers les problèmes les plus ”gros”.
- Ne jamais recalculer deux fois une même valeur

On remplit une matrice M de taille $(n+1)*(m+1)$ telle que $M[i,j]$ soit égal à $D(u_{0...i-1}, t_{0...j-1})$.

Programmation dynamique

Idée: utiliser un algorithme de programmation dynamique:

- Calculer des problèmes les plus ”petits” vers les problèmes les plus ”gros”.
- Ne jamais recalculer deux fois une même valeur

On remplit une matrice M de taille $(n+1)*(m+1)$ telle que $M[i,j]$ soit égal à $D(u_{0...i-1}, t_{0...j-1})$.

Programmation dynamique

Idée: utiliser un algorithme de programmation dynamique:

- Calculer des problèmes les plus ”petits” vers les problèmes les plus ”gros”.
- Ne jamais recalculer deux fois une même valeur

On remplit une matrice M de taille $(n+1)*(m+1)$ telle que $M[i,j]$ soit égal à $D(u_{0...i-1}, t_{0...j-1})$.

Programmation dynamique

Idée: utiliser un algorithme de programmation dynamique:

- Calculer des problèmes les plus ”petits” vers les problèmes les plus ”gros”.
- Ne jamais recalculer deux fois une même valeur

On remplit une matrice M de taille $(n+1)*(m+1)$ telle que $M[i,j]$ soit égal à $D(u_{0\ldots i-1}, t_{0\ldots j-1})$.

Programmation Dynamique

Exemple: u=”course” and t=”bonus”

		0	1 b	2 o	3 n	4 u	5 s
0							
1	c						
2	o						
3	u						
4	r						
5	s						
6	e						

Programmation Dynamique

Exemple: u=”course” and t=”bonus”

		0	1 b	2 o	3 n	4 u	5 s
0		0					
1	c						
2	o						
3	u						
4	r						
5	s						
6	e						

Commencer avec les cas les plus ”petits”: $D(\epsilon, \epsilon) = 0$

Programmation Dynamique

Exemple: u="course" and t="bonus"

		0	1 b	2 o	3 n	4 u	5 s
0		0					
1	c	1					
2	o	2					
3	u	3					
4	r	4					
5	s	5					
6	e	6					

les plus "petits" cas: $D(u_{0...i}, \epsilon) = i$
exemple: $D(co, \epsilon) = c_{del}(c) + c_{del}(o) + D(\epsilon, \epsilon)$

Programmation Dynamique

Exemple: u="course" and t="bonus"

		0	1 b	2 o	3 n	4 u	5 s
0		0	1	2	3	4	5
1	c	1					
2	o	2					
3	u	3					
4	r	4					
5	s	5					
6	e	6					

les plus "petits" cas: $D(\epsilon, t_{0\ldots j}) = j$
exemple: $D(\epsilon, bon) = c_{ins}(b) + c_{ins}(o) + c_{ins}(n) + D(\epsilon, \epsilon)$

Programmation Dynamique

Exemple: u="course" and t="bonus"

		0	1 b	2 o	3 n	4 u	5 s
0		0	1	2	3	4	5
1	c	1	2				
2	o	2					
3	u	3					
4	r	4					
5	s	5					
6	e	6					

à partir des **valeurs connues** calculer les **valeurs manquantes**:
$D(c, b)$ est $D(c, \epsilon) + c_{ins}(b)$

Programmation Dynamique

Exemple: u="course" and t="bonus"

		0	1 b	2 o	3 n	4 u	5 s
0		0	1	2	3	4	5
1	c	1	2				
2	o	2					
3	u	3					
4	r	4					
5	s	5					
6	e	6					

à partir des **valeurs connues** calculer les **valeurs manquantes**:
$D(c, b)$ ou $D(\epsilon, b) + c_{del}(c)$

Programmation Dynamique

Exemple: u="course" and t="bonus"

			1 b	2 o	3 n	4 u	5 s
0		0	1	2	3	4	5
1	c	1	1				
2	o	2					
3	u	3					
4	r	4					
5	s	5					
6	e	6					

à partir des **valeurs connues** calculer les **valeurs manquantes**:
$D(c, b)$ ou $D(\epsilon, \epsilon) + c_{sub}(c, b)$

Programmation Dynamique

Exemple: u="course" and t="bonus"

		0	1 b	2 o	3 n	4 u	5 s
0		0	1	2	3	4	5
1	c	1	1				
2	o	2	2				
3	u	3					
4	r	4					
5	s	5					
6	e	6					

à partir des **valeurs connues** calculer les **valeurs manquantes**:
$D(co, b) = min\{1 + 1, 1 + 1, 2 + 1\}$

Programmation Dynamique

Exemple: u="course" and t="bonus"

		0	1 b	2 o	3 n	4 u	5 s
0		0	1	2	3	4	5
1	c	1	1	X			
2	o	2	2				
3	u	3	X				
4	r	4					
5	s	5					
6	e	6					

valeurs pouvant être calculées

Programmation Dynamique

Exemple: u="course" and t="bonus"

		0	1 b	2 o	3 n	4 u	5 s
0		0	1	2	3	4	5
1	c	1	1	2			
2	o	2	2				
3	u	3	3				
4	r	4					
5	s	5					
6	e	6					

valeurs pouvant être calculées

Programmation Dynamique

Exemple: u="course" and t="bonus"

		0	1 b	2 o	3 n	4 u	5 s
0		0	1	2	3	4	5
1	c	1	1	2			
2	o	2	2	1?			
3	u	3	3				
4	r	4					
5	s	5					
6	e	6					

$D(co, bo)$ est $D(c, b) + c_{sub}(o, o) = 1 + 0 = 1$

Programmation Dynamique

Exemple: u="course" and t="bonus"

		0	1 b	2 o	3 n	4 u	5 s
0		0	1	2	3	4	5
1	c	1	1	2			
2	o	2	2	3?			
3	u	3	3				
4	r	4					
5	s	5					
6	e	6					

$D(co, bo)$ ou $D(c, bo) + c_{del}(o) = 2 + 1 = 3$

Programmation Dynamique

Exemple: u="course" and t="bonus"

		0	1 b	2 o	3 n	4 u	5 s
0		0	1	2	3	4	5
1	c	1	1	2			
2	o	2	2	3?			
3	u	3	3				
4	r	4					
5	s	5					
6	e	6					

$D(co, bo)$ ou $D(co, b) + c_{ins}(o) = 2 + 1 = 3$

Programmation Dynamique

Exemple: u=”course” and t=”bonus”

		0	1 b	2 o	3 n	4 u	5 s
0		0	1	2	3	4	5
1	c	1	1	2			
2	o	2	2	1			
3	u	3	3				
4	r	4					
5	s	5					
6	e	6					

Programmation Dynamique

Exemple: u="course" and t="bonus"

		0	1 b	2 o	3 n	4 u	5 s
0		0	1	2	3	4	5
1	c	1	1	2	3		
2	o	2	2	1			
3	u	3	3	2			
4	r	4	4				
5	s	5					
6	e	6					

Programmation Dynamique

Exemple: u=”course” and t=”bonus”

		0	1 b	2 o	3 n	4 u	5 s
0		0	1	2	3	4	5
1	c	1	1	2	3	4	5
2	o	2	2	1	2	3	4
3	u	3	3	2	2	2	3
4	r	4	4	3	3	3	3
5	s	5	5	4	4	4	3
6	e	6	6	5	5	5	4

Programmation Dynamique

Exemple: u="course" and t="bonus"

		0	1 b	2 o	3 n	4 u	5 s
0		0	1	2	3	4	5
1	c	1	1	2	3	4	5
2	o	2	2	1	2	3	4
3	u	3	3	2	2	2	3
4	r	4	4	3	3	3	3
5	s	5	5	4	4	4	3
6	e	6	6	5	5	5	4

Resultat

- La distance entre "course" et "bonus" est 4.
- L'espace necessaire est $O(m * n)$
- La complexité en temps est $O(m * n)$
- opérations d'édition ?

Programmation Dynamique

Exemple: u="course" and t="bonus"

		0	1 b	2 o	3 n	4 u	5 s
0		0	1	2	3	4	5
1	c	1	1	2	3	4	5
2	o	2	2	1	2	3	4
3	u	3	3	2	2	2	3
4	r	4	4	3	3	3	3
5	s	5	5	4	4	4	3
6	e	6	6	5	5	5	4

Resultat

- La distance entre "course" et "bonus" est 4.
- L'espace necessaire est $O(m * n)$
- La complexité en temps est $O(m * n)$
- opérations d'édition ?

Programmation Dynamique

Exemple: u="course" and t="bonus"

		0	1 b	2 o	3 n	4 u	5 s
0		0	1	2	3	4	5
1	c	1	1	2	3	4	5
2	o	2	2	1	2	3	4
3	u	3	3	2	2	2	3
4	r	4	4	3	3	3	3
5	s	5	5	4	4	4	3
6	e	6	6	5	5	5	4

Resultat

- La distance entre "course" et "bonus" est 4.
- L'espace necessaire est $O(m * n)$
- La complexité en temps est $O(m * n)$
- opérations d'édition ?

Programmation Dynamique

Exemple: u="course" and t="bonus"

		0	1 b	2 o	3 n	4 u	5 s
0		0	1	2	3	4	5
1	c	1	1	2	3	4	5
2	o	2	2	1	2	3	4
3	u	3	3	2	2	2	3
4	r	4	4	3	3	3	3
5	s	5	5	4	4	4	3
6	e	6	6	5	5	5	4

Resultat

- La distance entre "course" et "bonus" est 4.
- L'espace necessaire est $O(m * n)$
- La complexité en temps est $O(m * n)$
- opérations d'édition ?

Programmation Dynamique: traceback

Depuis la valeur finale retracer le calcul:

		0	1 b	2 o	3 n	4 u	5 s
0		0	1	2	3	4	5
1	c	1	1	2	3	4	5
2	o	2	2	1	2	3	4
3	u	3	3	2	2	2	3
4	r	4	4	3	3	3	3
5	s	5	5	4	4	4	3
6	e	6	6	5	5	5	4

Traceback:

$D(course, bonus) =$

Programmation Dynamique: traceback

Depuis la valeur finale retracer le calcul:

		0	1 b	2 o	3 n	4 u	5 s
0		0	1	2	3	4	5
1	c	1	1	2	3	4	5
2	o	2	2	1	2	3	4
3	u	3	3	2	2	2	3
4	r	4	4	3	3	3	3
5	s	5	5	4	4	4	3
6	e	6	6	5	5	5	4

Traceback:

$D(course, bonus) =$

- $D(cours, bonus) + c_{del}(e)$

Programmation Dynamique: traceback

Depuis la valeur finale retracer le calcul:

		0	1 b	2 o	3 n	4 u	5 s
0		0	1	2	3	4	5
1	c	1	1	2	3	4	5
2	o	2	2	1	2	3	4
3	u	3	3	2	2	2	3
4	r	4	4	3	3	3	3
5	s	5	5	4	4	4	3
6	e	6	6	5	5	5	4

Traceback:

$D(course, bonus) =$

- $c_{del}(e)$
- $D(cour, bonu) + c_{sub}(s, s)$

Programmation Dynamique: traceback

Depuis la valeur finale retracer le calcul:

		0	1 b	2 o	3 n	4 u	5 s
0		0	1	2	3	4	5
1	c	1	1	2	3	4	5
2	o	2	2	1	2	3	4
3	u	3	3	2	2	2	3
4	r	4	4	3	3	3	3
5	s	5	5	4	4	4	3
6	e	6	6	5	5	5	4

Traceback:

$D(course, bonus) =$

- $c_{del}(e)$
- $c_{sub}(s, s)$
- $D(cou, bonu) + c_{del}(r)$

Programmation Dynamique: traceback

Depuis la valeur finale retracer le calcul:

		0	1 b	2 o	3 n	4 u	5 s
0		0	1	2	3	4	5
1	c	1	1	2	3	4	5
2	o	2	2	1	2	3	4
3	u	3	3	2	2	2	3
4	r	4	4	3	3	3	3
5	s	5	5	4	4	4	3
6	e	6	6	5	5	5	4

Traceback:

$D(course, bonus) =$

- $c_{del}(e)$
- $c_{sub}(s, s)$
- $c_{del}(r)$
- $D(co, bon) + c_{sub}(u, u)$

Programmation Dynamique: traceback

Depuis la valeur finale retracer le calcul:

		0	1 b	2 o	3 n	4 u	5 s
0		0	1	2	3	4	5
1	c	1	1	2	3	4	5
2	o	2	2	1	2	3	4
3	u	3	3	2	2	2	3
4	r	4	4	3	3	3	3
5	s	5	5	4	4	4	3
6	e	6	6	5	5	5	4

Traceback:

Programmation Dynamique: traceback

Depuis la valeur finale retracer le calcul:

		0	1 b	2 o	3 n	4 u	5 s
0		0	1	2	3	4	5
1	c	1	1	2	3	4	5
2	o	2	2	1	2	3	4
3	u	3	3	2	2	2	3
4	r	4	4	3	3	3	3
5	s	5	5	4	4	4	3
6	e	6	6	5	5	5	4

Traceback:

$D(course, bonus) =$

- $c_{del}(e)$
- $c_{sub}(s, s)$
- $c_{del}(r)$
- $c_{sub}(u, u)$
- $c_{ins}(n)$
- $D(c, b) + c_{sub}(o, o)$

Programmation Dynamique: traceback

Depuis la valeur finale retracer le calcul:

		0	1 b	2 o	3 n	4 u	5 s
0		0	1	2	3	4	5
1	c	1	1	2	3	4	5
2	o	2	2	1	2	3	4
3	u	3	3	2	2	2	3
4	r	4	4	3	3	3	3
5	s	5	5	4	4	4	3
6	e	6	6	5	5	5	4

Traceback:

$D(course, bonus) =$

- ●

Programmation Dynamique: traceback

Alignement

c						
b						

Traceback:

$D(course, bonus) =$

- $c_{del}(e)$
- $c_{sub}(s, s)$
- $c_{del}(r)$
- $c_{sub}(u, u)$
- $c_{ins}(n)$
- $c_{sub}(o, o)$
- $D(\epsilon, \epsilon) + s_{sub}(c, b)$

Programmation Dynamique: traceback

Alignement

c	o					
b	o					

Traceback:

$D(course, bonus) =$

Programmation Dynamique: traceback

Alignement

c	o	-				
b	o	n				

Traceback:

$D(course, bonus) =$

- $c_{del}(e)$
- $c_{sub}(s, s)$
- $c_{del}(r)$
- $c_{sub}(u, u)$
- $c_{ins}(n)$
- $c_{sub}(o, o)$
- $D(\epsilon, \epsilon) + s_{sub}(c, b)$

Programmation Dynamique: traceback

Alignement

c	o	-	u			
b	o	n	u			

Traceback:

$D(course, bonus) =$

- $c_{del}(e)$
- $c_{sub}(s, s)$
- $c_{del}(r)$
- $c_{sub}(u, u)$
- $c_{ins}(n)$
- $c_{sub}(o, o)$
- $D(\epsilon, \epsilon) + s_{sub}(c, b)$

Programmation Dynamique: traceback

Alignement

c	o	-	u	r		
b	o	n	u	-		

Traceback:

$D(course, bonus) =$

- $c_{del}(e)$
- $c_{sub}(s, s)$
- $c_{del}(r)$
- $c_{sub}(u, u)$
- $c_{ins}(n)$
- $c_{sub}(o, o)$
- $D(\epsilon, \epsilon) + s_{sub}(c, b)$

Programmation Dynamique: traceback

Alignement

c	o	-	u	r	s	
b	o	n	u	-	s	

Traceback:

$D(course, bonus) =$

- $c_{del}(e)$
- $c_{sub}(s, s)$
- $c_{del}(r)$
- $c_{sub}(u, u)$
- $c_{ins}(n)$
- $c_{sub}(o, o)$
- $D(\epsilon, \epsilon) + s_{sub}(c, b)$

Programmation Dynamique: traceback

Alignement

c	o	-	u	r	s	e
b	o	n	u	-	s	-

Traceback:

$D(course, bonus) =$

- $c_{del}(e)$
- $c_{sub}(s, s)$
- $c_{del}(r)$
- $c_{sub}(u, u)$
- $c_{ins}(n)$
- $c_{sub}(o, o)$
- $D(\epsilon, \epsilon) + s_{sub}(c, b)$

Réduire la complexité en espace

Seulement une colonne (ou ligne) est requise pour calculer la distance:

		0	1 b	2 o	3 n	4 u	5 s
0		0					
1	c	1					
2	o	2					
3	u	3					
4	r	4					
5	s	5					
6	e	6					

- tableau
- temporary variable 1
- temporary variable 2
- anciennes valeurs

Réduire la complexité en espace

Seulement une colonne (ou ligne) est requise pour calculer la distance:

		0	1 b	2 o	3 n	4 u	5 s
0		0					
1	c	1					
2	o	2					
3	u	3					
4	r	4					
5	s	5					
6	e	6					

- tableau
- temporary variable 1
- temporary variable 2
- anciennes valeurs

Réduire la complexité en espace

Seulement une colonne (ou ligne) est requise pour calculer la distance:

		0	1 b	2 o	3 n	4 u	5 s
0		0	1				
1	c	1					
2	o	2					
3	u	3					
4	r	4					
5	s	5					
6	e	6					

- tableau
- temporary variable 1
- temporary variable 2
- anciennes valeurs

Réduire la complexité en espace

Seulement une colonne (ou ligne) est requise pour calculer la distance:

		0	1 b	2 o	3 n	4 u	5 s
0		0	1				
1	c	1	1				
2	o	2					
3	u	3					
4	r	4					
5	s	5					
6	e	6					

- tableau
- temporary variable 1
- temporary variable 2
- anciennes valeurs

Réduire la complexité en espace

Seulement une colonne (ou ligne) est requise pour calculer la distance:

		0	1 b	2 o	3 n	4 u	5 s
0		0	1				
1	c	1	1				
2	o	2	2				
3	u	3					
4	r	4					
5	s	5					
6	e	6					

- tableau
- temporary variable 1
- temporary variable 2
- anciennes valeurs

Réduire la complexité en espace

Seulement une colonne (ou ligne) est requise pour calculer la distance:

		0	1 b	2 o	3 n	4 u	5 s
0		0	1				
1	c	1	1				
2	o	2	2				
3	u	3	3				
4	r	4					
5	s	5					
6	e	6					

- tableau
- temporary variable 1
- temporary variable 2
- anciennes valeurs

Réduire la complexité en espace

Seulement une colonne (ou ligne) est requise pour calculer la distance:

		0	1 b	2 o	3 n	4 u	5 s
0		0	1				
1	c	1	1				
2	o	2	2				
3	u	3	3				
4	r	4	4				
5	s	5					
6	e	6					

- tableau
- temporary variable 1
- temporary variable 2
- anciennes valeurs

Réduire la complexité en espace

Seulement une colonne (ou ligne) est requise pour calculer la distance:

		0	1 b	2 o	3 n	4 u	5 s
0		0	1				
1	c	1	1				
2	o	2	2				
3	u	3	3				
4	r	4	4				
5	s	5	5				
6	e	6					

- tableau
- temporary variable 1
- temporary variable 2
- anciennes valeurs

Réduire la complexité en espace

Seulement une colonne (ou ligne) est requise pour calculer la distance:

		0	1 b	2 o	3 n	4 u	5 s
0		0	1				
1	c	1	1				
2	o	2	2				
3	u	3	3				
4	r	4	4				
5	s	5	5				
6	e	6	6				

- tableau
- temporary variable 1
- temporary variable 2
- anciennes valeurs

Réduire la complexité en espace

Seulement une colonne (ou ligne) est requise pour calculer la distance:

		0	1 b	2 o	3 n	4 u	5 s
0		0	1				
1	c	1	1				
2	o	2	2				
3	u	3	3				
4	r	4	4				
5	s	5	5				
6	e	6	6				

- tableau
- temporary variable 1
- temporary variable 2
- anciennes valeurs

$\Rightarrow$ la complexité mémoire est $O(min(m, n))$

Exercice

- Écrire une fonction `edit_matrix(u,t)` qui calcule la matrice d'édition pour deux mots *u* et *t*.
- Écrire une fonction `trace_back(m,u,t)` qui fournit la liste des opérations (lettres 's', 'd' et 'i') à partir d'une matrice *m* et de deux mots *u* et *t*.
- Écrire un programme qui prend en argument un dictionnaire *d* (fichier contenant un mot par ligne) et un mot *w* et affiche le mot de *d* le plus proche de *w* au sens de l'édition.
- Modifier le programme pour qu'il affiche aussi l'alignement des deux mots.

Exercice

- Écrire une fonction `edit_matrix(u,t)` qui calcule la matrice d'édition pour deux mots u et t.
- Écrire une fonction `trace_back(m,u,t)` qui fournit la liste des opérations (lettres 's', 'd' et 'i') à partir d'une matrice m et de deux mots u et t.
- Écrire un programme qui prend en argument un dictionnaire d (fichier contenant un mot par ligne) et un mot w et affiche le mot de d le plus proche de w au sens de l'édition.
- Modifier le programme pour qu'il affiche aussi l'alignement des deux mots.

Exercice

- Écrire une fonction `edit_matrix(u,t)` qui calcule la matrice d'édition pour deux mots u et t.
- Écrire une fonction `trace_back(m,u,t)` qui fournit la liste des opérations (lettres 's', 'd' et 'i') à partir d'une matrice m et de deux mots u et t.
- Écrire un programme qui prend en argument un dictionnaire d (fichier contenant un mot par ligne) et un mot w et affiche le mot de d le plus proche de w au sens de l'édition.
- Modifier le programme pour qu'il affiche aussi l'alignement des deux mots.

Exercice

- Écrire une fonction `edit_matrix(u,t)` qui calcule la matrice d'édition pour deux mots *u* et *t*.
- Écrire une fonction `trace_back(m,u,t)` qui fournit la liste des opérations (lettres 's', 'd' et 'i') à partir d'une matrice *m* et de deux mots *u* et *t*.
- Écrire un programme qui prend en argument un dictionnaire *d* (fichier contenant un mot par ligne) et un mot *w* et affiche le mot de *d* le plus proche de *w* au sens de l'édition.
- Modifier le programme pour qu'il affiche aussi l'alignement des deux mots.

Edition: variantes

Problème: Best Fit

Soient deux séquences q et t de taille m et n ($m < n$), trouver toutes les positions $p_1 \ldots p_l$ telles que:

$$D(q, t_{p_1 \ldots p_1+m_1}) = D(q, t_{p_2 \ldots p_2+m_2}) \ldots$$

et qu'il n'exite pas de position j telle que

$$D(q, t_{j \ldots j+m_j}) <= D(q, t_{p_1 \ldots p_1+m_1})$$

Best Fit

		0 ϵ	1 A	2 C	3 C	4 T	5 A	6 T	7 G	8 G	9 C	10 T	11 A	12 C
0	ϵ	0												
1	T	1												
2	A	2												
3	C	3												
4	G	4												
5	C	5												
6	T	6												

Best Fit

		0 ϵ	1 A	2 C	3 C	4 T	5 A	6 T	7 G	8 G	9 C	10 T	11 A	12 C
0	ϵ	0	0	0	0	0	0	0	0	0	0	0	0	0
1	T	1												
2	A	2												
3	C	3												
4	G	4												
5	C	5												
6	T	6												

Best Fit

		0 ϵ	1 A	2 C	3 C	4 T	5 A	6 T	7 G	8 G	9 C	10 T	11 A	12 C
0	ϵ	0	0	0	0	0	0	0	0	0	0	0	0	0
1	T	1	1											
2	A	2												
3	C	3												
4	G	4												
5	C	5												
6	T	6												

Best Fit

		0 ϵ	1 A	2 C	3 C	4 T	5 A	6 T	7 G	8 G	9 C	10 T	11 A	12 C
0	ϵ	0	0	0	0	0	0	0	0	0	0	0	0	0
1	T	1	1											
2	A	2	1											
3	C	3												
4	G	4												
5	C	5												
6	T	6												

Best Fit

		0 ϵ	1 A	2 C	3 C	4 T	5 A	6 T	7 G	8 G	9 C	10 T	11 A	12 C
0	ϵ	0	0	0	0	0	0	0	0	0	0	0	0	0
1	T	1	1	1										
2	A	2	1	2										
3	C	3	2	1										
4	G	4	3	2										
5	C	5	4	3										
6	T	6	5	4										

Best Fit

		0 ϵ	1 A	2 C	3 C	4 T	5 A	6 T	7 G	8 G	9 C	10 T	11 A	12 C
0	ϵ	0	0	0	0	0	0	0	0	0	0	0	0	0
1	T	1	1	1										
2	A	2	1	2										
3	C	3	2	1										
4	G	4	3	2										
5	C	5	4	3										
6	T	6	5	4										

Best Fit

		0 ϵ	1 A	2 C	3 C	4 T	5 A	6 T	7 G	8 G	9 C	10 T	11 A	12 C
0	ϵ	0	0	0	0	0	0	0	0	0	0	0	0	0
1	T	1	1	1										
2	A	2	1	2										
3	C	3	2	1										
4	G	4	3	2										
5	C	5	4	3										
6	T	6	5	4										

Best Fit

		0 ϵ	1 A	2 C	3 C	4 T	5 A	6 T	7 G	8 G	9 C	10 T	11 A	12 C
0	ϵ	0	0	0	0	0	0	0	0	0	0	0	0	0
1	T	1	1	1										
2	A	2	1	2										
3	C	3	2	1										
4	G	4	3	2										
5	C	5	4	3										
6	T	6	5	4										

Best Fit

		0 ϵ	1 A	2 C	3 C	4 T	5 A	6 T	7 G	8 G	9 C	10 T	11 A	12 C
0	ϵ	0	0	0	0	0	0	0	0	0	0	0	0	0
1	T	1	1	1										
2	A	2	1	2										
3	C	3	2	1										
4	G	4	3	2										
5	C	5	4	3										
6	T	6	5	4										

Best Fit

		0 ϵ	1 A	2 C	3 C	4 T	5 A	6 T	7 G	8 G	9 C	10 T	11 A	12 C
0	ϵ	0	0	0	0	0	0	0	0	0	0	0	0	0
1	T	1	1	1										
2	A	2	1	2										
3	C	3	2	1										
4	G	4	3	2										
5	C	5	4	3										
6	T	6	5	4										

Best Fit

		0 ϵ	1 A	2 C	3 C	4 T	5 A	6 T	7 G	8 G	9 C	10 T	11 A	12 C
0	ϵ	0	0	0	0	0	0	0	0	0	0	0	0	0
1	T	1	1	1										
2	A	2	1	2										
3	C	3	2	1										
4	G	4	3	2										
5	C	5	4	3										
6	T	6	5	4										

Best Fit

		0 ϵ	1 A	2 C	3 C	4 T	5 A	6 T	7 G	8 G	9 C	10 T	11 A	12 C
0	ϵ	0	0	0	0	0	0	0	0	0	0	0	0	0
1	T	1	1	1										
2	A	2	1	2										
3	C	3	2	1										
4	G	4	3	2										
5	C	5	4	3										
6	T	6	5	4										

Best Fit

		0 ϵ	1 A	2 C	3 C	4 T	5 A	6 T	7 G	8 G	9 C	10 T	11 A	12 C
0	ϵ	0	0	0	0	0	0	0	0	0	0	0	0	0
1	T	1	1	1	1									
2	A	2	1	2	2									
3	C	3	2	1	2									
4	G	4	3	2	2									
5	C	5	4	3	2									
6	T	6	5	4	3									

Best Fit

		0 ϵ	1 A	2 C	3 C	4 T	5 A	6 T	7 G	8 G	9 C	10 T	11 A	12 C
0	ϵ	0	0	0	0	0	0	0	0	0	0	0	0	0
1	T	1	1	1	1	0	1	0	1	1	1	0	1	1
2	A	2	1	2	2	1	0	1	1	2	2	1	0	1
3	C	3	2	1	2	2	1	1	2	2	2	2	1	0
4	G	4	3	2	2	3	2	2	1	2	3	3	2	1
5	C	5	4	3	2	3	3	3	2	2	2	3	3	2
6	T	6	5	4	3	2	3	3	3	3	3	2	3	3

Best Fit

		0 ϵ	1 A	2 C	3 C	4 T	5 A	6 T	7 G	8 G	9 C	10 T	11 A	12 C
0	ϵ	0	0	0	0	0	0	0	0	0	0	0	0	0
1	T	1	1	1	1	0	1	0	1	1	1	0	1	1
2	A	2	1	2	2	1	0	1	1	2	2	1	0	1
3	C	3	2	1	2	2	1	1	2	2	2	2	1	0
4	G	4	3	2	2	3	2	2	1	2	3	3	2	1
5	C	5	4	3	2	3	3	3	2	2	2	3	3	2
6	T	6	5	4	3	2	3	3	3	3	3	2	3	3

Best Fit

		0 ϵ	1 A	2 C	3 C	4 T	5 A	6 T	7 G	8 G	9 C	10 T	11 A	12 C
0	ϵ	0	0	0	0	0	0	0	0	0	0	0	0	0
1	T	1	1	1	1	0	1	0	1	1	1	0	1	1
2	A	2	1	2	2	1	0	1	1	2	2	1	0	1
3	C	3	2	1	2	2	1	1	2	2	2	2	1	0
4	G	4	3	2	2	3	2	2	1	2	3	3	2	1
5	C	5	4	3	2	3	3	3	2	2	2	3	3	2
6	T	6	5	4	3	2	3	3	3	3	3	2	3	3

A	C	C	T	A	T	G	G	C	T	A	C

Best Fit

		0 ϵ	1 A	2 C	3 C	4 T	5 A	6 T	7 G	8 G	9 C	10 T	11 A	12 C
0	ϵ	0	0	0	0	0	0	0	0	0	0	0	0	0
1	T	1	1	1	1	0	1	0	1	1	1	0	1	1
2	A	2	1	2	2	1	0	1	1	2	2	1	0	1
3	C	3	2	1	2	2	1	1	2	2	2	2	1	0
4	G	4	3	2	2	3	2	2	1	2	3	3	2	1
5	C	5	4	3	2	3	3	3	2	2	2	3	3	2
6	T	6	5	4	3	2	3	3	3	3	3	2	3	3

A	C	C	T	A	T	G	G	C	T	A	C
			T								

Best Fit

		0 ϵ	1 A	2 C	3 C	4 T	5 A	6 T	7 G	8 G	9 C	10 T	11 A	12 C
0	ϵ	0	0	0	0	0	0	0	0	0	0	0	0	0
1	T	1	1	1	1	0	1	0	1	1	1	0	1	1
2	A	2	1	2	2	1	0	1	1	2	2	1	0	1
3	C	3	2	1	2	2	1	1	2	2	2	2	1	0
4	G	4	3	2	2	3	2	2	1	2	3	3	2	1
5	C	5	4	3	2	3	3	3	2	2	2	3	3	2
6	T	6	5	4	3	2	3	3	3	3	3	2	3	3

A	C	C	T	A	T	G	G	C	T	A	C
		C	T								

Best Fit

		0 ϵ	1 A	2 C	3 C	4 T	5 A	6 T	7 G	8 G	9 C	10 T	11 A	12 C
0	ϵ	0	0	0	0	0	0	0	0	0	0	0	0	0
1	T	1	1	1	1	0	1	0	1	1	1	0	1	1
2	A	2	1	2	2	1	0	1	1	2	2	1	0	1
3	C	3	2	1	2	2	1	1	2	2	2	2	1	0
4	G	4	3	2	2	3	2	2	1	2	3	3	2	1
5	C	5	4	3	2	3	3	3	2	2	2	3	3	2
6	T	6	5	4	3	2	3	3	3	3	3	2	3	3

A	C	-C	T	A	T	G	G	C	T	A	C
		GC	T								

Best Fit

		0 ϵ	1 A	2 C	3 C	4 T	5 A	6 T	7 G	8 G	9 C	10 T	11 A	12 C
0	ϵ	0	0	0	0	0	0	0	0	0	0	0	0	0
1	T	1	1	1	1	0	1	0	1	1	1	0	1	1
2	A	2	1	2	2	1	0	1	1	2	2	1	0	1
3	C	3	2	1	2	2	1	1	2	2	2	2	1	0
4	G	4	3	2	2	3	2	2	1	2	3	3	2	1
5	C	5	4	3	2	3	3	3	2	2	2	3	3	2
6	T	6	5	4	3	2	3	3	3	3	3	2	3	3

A	C	-C	T	A	T	G	G	C	T	A	C
	C	GC	T								

Best Fit

		0 ϵ	1 A	2 C	3 C	4 T	5 A	6 T	7 G	8 G	9 C	10 T	11 A	12 C
0	ϵ	0	0	0	0	0	0	0	0	0	0	0	0	0
1	T	1	1	1	1	0	1	0	1	1	1	0	1	1
2	A	2	1	2	2	1	0	1	1	2	2	1	0	1
3	C	3	2	1	2	2	1	1	2	2	2	2	1	0
4	G	4	3	2	2	3	2	2	1	2	3	3	2	1
5	C	5	4	3	2	3	3	3	2	2	2	3	3	2
6	T	6	5	4	3	2	3	3	3	3	3	2	3	3

A	C	-C	T	A	T	G	G	C	T	A	C
A	C	GC	T								

Best Fit

		0 ϵ	1 A	2 C	3 C	4 T	5 A	6 T	7 G	8 G	9 C	10 T	11 A	12 C
0	ϵ	0	0	0	0	0	0	0	0	0	0	0	0	0
1	T	1	1	1	1	0	1	0	1	1	1	0	1	1
2	A	2	1	2	2	1	0	1	1	2	2	1	0	1
3	C	3	2	1	2	2	1	1	2	2	2	2	1	0
4	G	4	3	2	2	3	2	2	1	2	3	3	2	1
5	C	5	4	3	2	3	3	3	2	2	2	3	3	2
6	T	6	5	4	3	2	3	3	3	3	3	2	3	3

-A	C	-C	T	A	T	G	G	C	T	A	C
TA	C	GC	T								

Best Fit

		0	1	2	3	4	5	6	7	8	9	10	11	12
		ϵ	A	C	C	T	A	T	G	G	C	T	A	C
0	ϵ	0	0	0	0	0	0	0	0	0	0	0	0	0
1	T	1	1	1	1	0	1	0	1	1	1	0	1	1
2	A	2	1	2	2	1	0	1	1	2	2	1	0	1
3	C	3	2	1	2	2	1	1	2	2	2	2	1	0
4	G	4	3	2	2	3	2	2	1	2	3	3	2	1
5	C	5	4	3	2	3	3	3	2	2	2	3	3	2
6	T	6	5	4	3	2	3	3	3	3	3	2	3	3

-A	C	-C	T	A	T	G	-G	C	T	A	C	
TA	C	GC	T		T	A	CG	C	T			

Exercices

Exercices

Trouver quel facteur de *ACCACGTGTCA* à la plus petite distance d'édition avec *AGGTC* avec la même fonction de score que dans l'exemple précédent.

Alignement: Formule

L'idée principale est de calculer $A(u_{0\ldots i}, t_{0\ldots j})$ à partir des scores $A(u_{0\ldots i-1}, t_{0\ldots j})$, $A(u_{0\ldots i}, t_{0\ldots j-1})$ et $A(u_{0\ldots i-1}, t_{0\ldots j-1})$: Exemple de fonction de coût:
cost(a,a)=2, cost(a,b)=-1, cost(a,'-')=-1, cost('-',a)=-1

S. Needleman, C. Wunsch,A general method applicable to the search for similarities in the amino acid sequence of two proteins, J Mol Biol. 48(3):443-53.

Alignement: Formule

L'idée principale est de calculer $A(u_{0\ldots i}, t_{0\ldots j})$ à partir des scores $A(u_{0\ldots i-1}, t_{0\ldots j})$, $A(u_{0\ldots i}, t_{0\ldots j-1})$ et $A(u_{0\ldots i-1}, t_{0\ldots j-1})$:

$$A(u_{0\ldots i}, t_{0\ldots j}) = Max \left\{ \begin{array}{lll} A(u_{0\ldots i-1} & , t_{0\ldots j-1}) & + \quad cost(u_i, t_j) \end{array} \right.$$

Exemple de fonction de coût:
cost(a,a)=2, cost(a,b)=-1, cost(a,'-')=-1, cost('-',a)=-1

S. Needleman, C. Wunsch,A general method applicable to the search for similarities in the amino acid sequence of two proteins, J Mol Biol. 48(3):443-53.

Alignement: Formule

L'idée principale est de calculer $A(u_{0\ldots i}, t_{0\ldots j})$ à partir des scores $A(u_{0\ldots i-1}, t_{0\ldots j})$, $A(u_{0\ldots i}, t_{0\ldots j-1})$ et $A(u_{0\ldots i-1}, t_{0\ldots j-1})$:

$$A(u_{0\ldots i}, t_{0\ldots j}) = Max \begin{cases} A(u_{0\ldots i-1}, t_{0\ldots j-1}) + cost(u_i, t_j) \\ A(u_{0\ldots i-1}, t_{0\ldots j}) + cost(u_i, '-') \end{cases}$$

Exemple de fonction de coût:
cost(a,a)=2, cost(a,b)=-1, cost(a,'-')=-1, cost('-',a)=-1

S. Needleman, C. Wunsch,A general method applicable to the search for similarities in the amino acid sequence of two proteins, J Mol Biol. 48(3):443-53.

Alignement: Formule

L'idée principale est de calculer $A(u_{0\ldots i}, t_{0\ldots j})$ à partir des scores $A(u_{0\ldots i-1}, t_{0\ldots j})$, $A(u_{0\ldots i}, t_{0\ldots j-1})$ et $A(u_{0\ldots i-1}, t_{0\ldots j-1})$:

$$A(u_{0\ldots i}, t_{0\ldots j}) = Max \begin{cases} A(u_{..i-1} \end{cases}$$

Alignement: Formule

L'idée principale est de calculer $A(u_{0\ldots i}, t_{0\ldots j})$ à partir des scores $A(u_{0\ldots i-1}, t_{0\ldots j})$, $A(u_{0\ldots i}, t_{0\ldots j-1})$ et $A(u_{0\ldots i-1}, t_{0\ldots j-1})$:

$$A(u_{0\ldots i}, t_{0\ldots j}) = Max \begin{cases} A(u_{0\ldots i-1}, t_{0\ldots j-1}) + cost(u_i, t_j) \\ A(u_{0\ldots i-1}, t_{0\ldots j}) + cost(u_i, '-') \\ A(u_{0\ldots i}, t_{0\ldots j-1}) + cost('-', t_j) \end{cases}$$

$$\begin{array}{|ccc|c|} \hline u_0 & \ldots & u_{i-1} & u_i \\ \hline t_0 & \ldots & t_{i-1} & t_j \\ \hline \end{array} = Max \begin{cases} \begin{array}{|ccc|} \hline u_0 & \ldots & u_{i-1} \\ \hline t_0 & \ldots & t_{j-1} \\ \hline \end{array} & \begin{array}{c} u_i \\ t_j \end{array} \end{cases}$$

Exemple de fonction de coût:
cost(a,a)=2, cost(a,b)=-1, cost(a,'-')=-1, cost('-',a)=-1
S. Needleman, C. Wunsch,A general method applicable to the search for similarities in the amino acid sequence of two proteins, J Mol Biol. 48(3):443-53.

Alignement: Formule

L'idée principale est de calculer $A(u_{0\ldots i}, t_{0\ldots j})$ à partir des scores $A(u_{0\ldots i-1}, t_{0\ldots j})$, $A(u_{0\ldots i}, t_{0\ldots j-1})$ et $A(u_{0\ldots i-1}, t_{0\ldots j-1})$:

$$A(u_{0\ldots i}, t_{0\ldots j}) = Max \begin{cases} A(u_{0\ldots i-1}, t_{0\ldots j-1}) + cost(u_i, t_j) \\ A(u_{0\ldots i-1}, t_{0\ldots j}) + cost(u_i, '-') \\ A(u_{0\ldots i}, t_{0\ldots j-1}) + cost('-', t_j) \end{cases}$$

$$\begin{array}{|ccc|c|} \hline u_0 & \ldots & u_{i-1} & u_i \\ \hline t_0 & \ldots & t_{i-1} & t_j \\ \hline \end{array} = Max \begin{cases} \begin{array}{|ccc|} \hline u_0 & \ldots & u_{i-1} \\ \hline t_0 & \ldots & t_{j-1} \\ \hline \end{array} & u_i \\ & t_j \\ \begin{array}{|ccc|} \hline u_0 & \ldots & u_{i-1} \\ \hline t_0 & \ldots & t_j \\ \hline \end{array} & u_i \\ & - \end{cases}$$

Exemple de fonction de coût:
cost(a,a)=2, cost(a,b)=-1, cost(a,'-')=-1, cost('-',a)=-1
S. Needleman, C. Wunsch,A general method applicable to the search for similarities in the amino acid sequence of two proteins, J Mol Biol.

Alignement: Formule

L'idée principale est de calculer $A(u_{0\ldots i}, t_{0\ldots j})$ à partir des scores $A(u_{0\ldots i-1}, t_{0\ldots j})$, $A(u_{0\ldots i}, t_{0\ldots j-1})$ et $A(u_{0\ldots i-1}, t_{0\ldots j-1})$:

$$A(u_{0\ldots i}, t_{0\ldots j}) = Max \begin{cases} A(u_{0\ldots i-1}, t_{0\ldots j-1}) + cost(u_i, t_j) \\ A(u_{0\ldots i-1}, t_{0\ldots j}) + cost(u_i, '-') \\ A(u_{0\ldots i}, t_{0\ldots j-1}) + cost('-', t_j) \end{cases}$$

$$\begin{array}{|ccc|c|} \hline u_0 & \ldots & u_{i-1} & u_i \\ \hline t_0 & \ldots & t_{i-1} & t_j \\ \hline \end{array} = Max \begin{cases} \begin{array}{|ccc|} \hline u_0 & \ldots & u_{i-1} \\ \hline t_0 & \ldots & t_{j-1} \\ \hline \end{array} & \begin{array}{c} u_i \\ t_j \end{array} \\ \begin{array}{|ccc|} \hline u_0 & \ldots & u_{i-1} \\ \hline t_0 & \ldots & t_j \\ \hline \end{array} & \begin{array}{c} u_i \\ - \end{array} \\ \begin{array}{|ccc|} \hline u_0 & \ldots & u_i \\ \hline t_0 & \ldots & t_{j-1} \\ \hline \end{array} & \begin{array}{c} - \\ t_j \end{array} \end{cases}$$

Exemple de fonction de coût:
cost(a,a)=2, cost(a,b)=-1, cost(a,'-')=-1, cost('-',a)=-1

S. Needleman, C. Wunsch,A general method applicable to the search for

Alignement: Formule

L'idée principale est de calculer $A(u_{0\ldots i}, t_{0\ldots j})$ à partir des scores $A(u_{0\ldots i-1}, t_{0\ldots j})$, $A(u_{0\ldots i}, t_{0\ldots j-1})$ et $A(u_{0\ldots i-1}, t_{0\ldots j-1})$:

$$A(u_{0\ldots i}, t_{0\ldots j}) = Max \begin{cases} A(u_{0\ldots i-1}, t_{0\ldots j-1}) + cost(u_i, t_j) \\ A(u_{0\ldots i-1}, t_{0\ldots j}) + cost(u_i, '-') \\ A(u_{0\ldots i}, t_{0\ldots j-1}) + cost('-', t_j) \end{cases}$$

Exemple de fonction de coût:
cost(a,a)=2, cost(a,b)=-1, cost(a,'-')=-1, cost('-',a)=-1

S. Needleman, C. Wunsch,A general method applicable to the search for similarities in the amino acid sequence of two proteins, J Mol Biol. 48(3):443-53.

Alignement: exemple

		0	1 C	2 A	3 G	4 A	5 T	6 C	7 G	8 A	9 T
0		0									
1	A										
2	T										
3	T										
4	G										
5	C										
7	A										
8	T										
9	C										

- cost(a,a)=2
- cost(a,b)=-1
- cost(a,'-')=-1
- cost('-',a)=-1

Alignement: exemple

		0	1 C	2 A	3 G	4 A	5 T	6 C	7 G	8 A	9 T
0		0									
1	A	-1									
2	T	-2									
3	T	-3									
4	G	-4									
5	C	-5									
7	A	-6									
8	T	-7									
9	C	-8									

- cost(a,a)=2
- cost(a,b)=-1
- cost(a,'-')=-1
- cost('-',a)=-1

Alignement: exemple

		0	1 C	2 A	3 G	4 A	5 T	6 C	7 G	8 A	9 T
0		0	-1	-2	-3	-4	-5	-6	-7	-8	-9
1	A	-1									
2	T	-2									
3	T	-3									
4	G	-4									
5	C	-5									
7	A	-6									
8	T	-7									
9	C	-8									

- cost(a,a)=2
- cost(a,b)=-1
- cost(a,'-')=-1
- cost('-',a)=-1

Alignement: exemple

		0	1 C	2 A	3 G	4 A	5 T	6 C	7 G	8 A	9 T
0		0	-1	-2	-3	-4	-5	-6	-7	-8	-9
1	A	-1	-1								
2	T	-2									
3	T	-3									
4	G	-4									
5	C	-5									
7	A	-6									
8	T	-7									
9	C	-8									

- cost(a,a)=2
- cost(a,b)=-1
- cost(a,'-')=-1
- cost('-',a)=-1

Alignement: exemple

		0	1 C	2 A	3 G	4 A	5 T	6 C	7 G	8 A	9 T
0		0	-1	-2	-3	-4	-5	-6	-7	-8	-9
1	A	-1	-1	1	0	-1	-2	-3	-4	-5	-6
2	T	-2	-2								
3	T	-3	-3								
4	G	-4	-4								
5	C	-5	-2								
7	A	-6	-3								
8	T	-7	-4								
9	C	-8	-5								

- cost(a,a)=2
- cost(a,b)=-1
- cost(a,'-')=-1
- cost('-',a)=-1

Alignement: exemple

		0	1 C	2 A	3 G	4 A	5 T	6 C	7 G	8 A	9 T
0		0	-1	-2	-3	-4	-5	-6	-7	-8	-9
1	A	-1	-1	1	0	-1	-2	-3	-4	-5	-6
2	T	-2	-2	0	0	-1	1	0	-1	-2	-3
3	T	-3	-3	-1	-1	-1	1	0	-1	-2	0
4	G	-4	-4	-2	1	0	0	0	2	1	0
5	C	-5	-2	-3	0	0	-1	2	1	1	0
7	A	-6	-3	0	-1	2	1	1	1	3	2
8	T	-7	-4	-1	-1	1	4	3	2	2	5
9	C	-8	-5	-2	-2	0	3	6	5	4	4

- cost(a,a)=+2

Alignement: exemple

		0	1 C	2 A	3 G	4 A	5 T	6 C	7 G	8 A	9 T
0		0	-1	-2	-3	-4	-5	-6	-7	-8	-9
1	A	-1	-1	1	0	-1	-2	-3	-4	-5	-6
2	T	-2	-2	0	0	-1	1	0	-1	-2	-3
3	T	-3	-3	-1	-1	-1	1	0	-1	-2	0
4	G	-4	-4	-2	1	0	0	0	2	1	0
5	C	-5	-2	-3	0	0	-1	2	1	1	0
7	A	-6	-3	0	-1	2	1	1	1	3	2
8	T	-7	-4	-1	-1	1	4	3	2	2	5
9	C	-8	-5	-2	-2	0	3	6	5	4	4

- cost(a,a)=+2

Alignement: exemple

		0	1 C	2 A	3 G	4 A	5 T	6 C	7 G	8 A	9 T
0		0	-1	-2	-3	-4	-5	-6	-7	-8	-9
1	A	-1	-1	1	0	-1	-2	-3	-4	-5	-6
2	T	-2	-2	0	0	-1	1	0	-1	-2	-3
3	T	-3	-3	-1	-1	-1	1	0	-1	-2	0
4	G	-4	-4	-2	1	0	0	0	2	1	0
5	C	-5	-2	-3	0	0	-1	2	1	1	0
7	A	-6	-3	0	-1	2	1	1	1	3	2
8	T	-7	-4	-1	-1	1	4	3	2	2	5
9	C	-8	-5	-2	-2	0	3	6	5	4	4

C	A	G	A	T	-	C	G	A	T	-
-	A	T	-	T	G	C	-	A	T	C

- cost(a,a)=+2
- cost(a,b)=-1
- cost(a,'-')=-1
- cost('-',a)=-1

Alignement: exemple

		0	1 C	2 A	3 G	4 A	5 T	6 C	7 G	8 A	9 T
0		0	-1	-2	-3	-4	-5	-6	-7	-8	-9
1	A	-1	-1	1	0	-1	-2	-3	-4	-5	-6
2	T	-2	-2	0	0	-1	1	0	-1	-2	-3
3	T	-3	-3	-1	-1	-1	1	0	-1	-2	0
4	G	-4	-4	-2	1	0	0	0	2	1	0
5	C	-5	-2	-3	0	0	-1	2	1	1	0
7	A	-6	-3	0	-1	2	1	1	1	3	2
8	T	-7	-4	-1	-1	1	4	3	2	2	5
9	C	-8	-5	-2	-2	0	3	6	5	4	4

C	A	G	A	T	C	G	-	A	T	-
-	-	-	A	T	T	G	C	A	T	C

- cost(a,a)=+2
- cost(a,b)=-1
- cost(a,'-')=-1
- cost('-',a)=-1

Alignement: Variantes

Problème: alignement local

Soient deux séquences q et t de longueur m et n, trouver les facteurs de q et t qui ont le meilleur score d'alignement.

Alignement Local

L'idée est que l'on peut commencer l'alignement à partir de n'importe où dans la matrice:

$$A(u_{0\ldots i}, t_{0\ldots j}) = Max \begin{cases} A(u_{0\ldots i-1}, t_{0\ldots j-1}) + cost(u_i, t_j) \\ A(u_{0\ldots i-1}, t_{0\ldots j}) + cost(u_i, '-') \\ A(u_{0\ldots i}, t_{0\ldots j-1}) + cost('-', t_j) \\ 0 \end{cases}$$

Alignement Local

		0	1 C	2 A	3 G	4 A	5 T	6 C	7 G	8 A
0		0								
1	A									
2	T									
3	T									
4	A									
5	C									
7	A									
8	T									
9	C									

Matrice:

	A	C	G	T	-
A	2	-1	-1	-1	-1
C	-1	2	-1	-1	-1
G	-1	-1	2	-1	-1
T	-1	-1	-1	2	-1
-	-1	-1	-1	-1	

Alignement Local

		0	1 C	2 A	3 G	4 A	5 T	6 C	7 G	8 A
0		0	0	0	0	0	0	0	0	0
1	A	0								
2	T	0								
3	T	0								
4	A	0								
5	C	0								
7	A	0								
8	T	0								
9	C	0								

Matrice:

	A	C	G	T	-
A	2	-1	-1	-1	-1
C	-1	2	-1	-1	-1
G	-1	-1	2	-1	-1
T	-1	-1	-1	2	-1
-	-1	-1	-1	-1	

Alignement Local

		0	1 C	2 A	3 G	4 A	5 T	6 C	7 G	8 A
0		0	0	0	0	0	0	0	0	0
1	A	0	0	2	1	2	1	0	0	2
2	T	0								
3	T	0								
4	A	0								
5	C	0								
7	A	0								
8	T	0								
9	C	0								

Matrice:

	A	C	G	T	-
A	2	-1	-1	-1	-1
C	-1	2	-1	-1	-1
G	-1	-1	2	-1	-1
T	-1	-1	-1	2	-1
-	-1	-1	-1	-1	

Alignement Local

		0	1 C	2 A	3 G	4 A	5 T	6 C	7 G	8 A
0		0	0	0	0	0	0	0	0	0
1	A	0	0	2	1	2	1	0	0	2
2	T	0	0	0	1	1	4	3	2	1
3	T	0								
4	A	0								
5	C	0								
7	A	0								
8	T	0								
9	C	0								

Matrice:

	A	C	G	T	-
A	2	-1	-1	-1	-1
C	-1	2	-1	-1	-1
G	-1	-1	2	-1	-1
T	-1	-1	-1	2	-1
-	-1	-1	-1	-1	

Alignement Local

		0	1 C	2 A	3 G	4 A	5 T	6 C	7 G	8 A
0		0	0	0	0	0	0	0	0	0
1	A	0	0	2	1	2	1	0	0	2
2	T	0	0	0	1	1	4	3	2	1
3	T	0	0	0	0	0	3	3	2	1
4	A	0								
5	C	0								
7	A	0								
8	T	0								
9	C	0								

Matrice:

	A	C	G	T	-
A	2	-1	-1	-1	-1
C	-1	2	-1	-1	-1
G	-1	-1	2	-1	-1
T	-1	-1	-1	2	-1
-	-1	-1	-1	-1	

Alignement Local

		0	1 C	2 A	3 G	4 A	5 T	6 C	7 G	8 A
0		0	0	0	0	0	0	0	0	0
1	A	0	0	2	1	2	1	0	0	2
2	T	0	0	0	1	1	4	3	2	1
3	T	0	0	0	0	0	3	3	2	1
4	A	0	0	2	1	2	2	2	2	4
5	C	0								
7	A	0								
8	T	0								
9	C	0								

Matrice:

	A	C	G	T	-
A	2	-1	-1	-1	-1
C	-1	2	-1	-1	-1
G	-1	-1	2	-1	-1
T	-1	-1	-1	2	-1
-	-1	-1	-1	-1	

Alignement Local

		0	1 C	2 A	3 G	4 A	5 T	6 C	7 G	8 A
0		0	0	0	0	0	0	0	0	0
1	A	0	0	2	1	2	1	0	0	2
2	T	0	0	0	1	1	4	3	2	1
3	T	0	0	0	0	0	3	3	2	1
4	A	0	0	2	1	2	2	2	2	4
5	C	0	2	1	1	1	1	4	3	3
7	A	0								
8	T	0								
9	C	0								

Matrice:

	A	C	G	T	-
A	2	-1	-1	-1	-1
C	-1	2	-1	-1	-1
G	-1	-1	2	-1	-1
T	-1	-1	-1	2	-1
-	-1	-1	-1	-1	

Alignement Local

		0	1 C	2 A	3 G	4 A	5 T	6 C	7 G	8 A
0		0	0	0	0	0	0	0	0	0
1	A	0	0	2	1	2	1	0	0	2
2	T	0	0	0	1	1	4	3	2	1
3	T	0	0	0	0	0	3	3	2	1
4	A	0	0	2	1	2	2	2	2	4
5	C	0	2	1	1	1	1	4	3	3
7	A	0	1	4	3	3	2	3	3	5
8	T	0								
9	C	0								

Matrice:

	A	C	G	T	-
A	2	-1	-1	-1	-1
C	-1	2	-1	-1	-1
G	-1	-1	2	-1	-1
T	-1	-1	-1	2	-1
-	-1	-1	-1	-1	

Alignement Local

		0	1 C	2 A	3 G	4 A	5 T	6 C	7 G	8 A
0		0	0	0	0	0	0	0	0	0
1	A	0	0	2	1	2	1	0	0	2
2	T	0	0	0	1	1	4	3	2	1
3	T	0	0	0	0	0	3	3	2	1
4	A	0	0	2	1	2	2	2	2	4
5	C	0	2	1	1	1	1	4	3	3
7	A	0	1	4	3	3	2	3	3	5
8	T	0	0	3	3	2	5	4	3	4
9	C	0								

Matrice:

	A	C	G	T	-
A	2	-1	-1	-1	-1
C	-1	2	-1	-1	-1
G	-1	-1	2	-1	-1
T	-1	-1	-1	2	-1
-	-1	-1	-1	-1	

Alignement Local

		0	1 C	2 A	3 G	4 A	5 T	6 C	7 G	8 A
0		0	0	0	0	0	0	0	0	0
1	A	0	0	2	1	2	1	0	0	2
2	T	0	0	0	1	1	4	3	2	1
3	T	0	0	0	0	0	3	3	2	1
4	A	0	0	2	1	2	2	2	2	4
5	C	0	2	1	1	1	1	4	3	3
7	A	0	1	4	3	3	2	3	3	5
8	T	0	0	3	3	2	5	4	3	4
9	C	0	2	2	2	2	4	7	6	5

Matrice:

	A	C	G	T	-
A	2	-1	-1	-1	-1
C	-1	2	-1	-1	-1
G	-1	-1	2	-1	-1
T	-1	-1	-1	2	-1
-	-1	-1	-1	-1	

Alignement Local

		0	1 C	2 A	3 G	4 A	5 T	6 C	7 G	8 A
0		0	0	0	0	0	0	0	0	0
1	A	0	0	2	1	2	1	0	0	2
2	T	0	0	0	1	1	4	3	2	1
3	T	0	0	0	0	0	3	3	2	1
4	A	0	0	2	1	2	2	2	2	4
5	C	0	2	1	1	1	1	4	3	3
7	A	0	1	4	3	3	2	3	3	5
8	T	0	0	3	3	2	5	4	3	4
9	C	0	2	2	2	2	4	7	6	5

Matrice:

	A	C	G	T	-
A	2	-1	-1	-1	-1
C	-1	2	-1	-1	-1
G	-1	-1	2	-1	-1
T	-1	-1	-1	2	-1
-	-1	-1	-1	-1	

Alignement Local

		0	1 C	2 A	3 G	4 A	5 T	6 C	7 G	8 A
0		0	0	0	0	0	0	0	0	0
1	A	0	0	2	1	2	1	0	0	2
2	T	0	0	0	1	1	4	3	2	1
3	T	0	0	0	0	0	3	3	2	1
4	A	0	0	2	1	2	2	2	2	4
5	C	0	2	1	1	1	1	4	3	3
7	A	0	1	4	3	3	2	3	3	5
8	T	0	0	3	3	2	5	4	3	4
9	C	0	2	2	2	2	4	7	6	5

Matrice:

	A	C	G	T	-
A	2	-1	-1	-1	-1
C	-1	2	-1	-1	-1
G	-1	-1	2	-1	-1
T	-1	-1	-1	2	

Alignement Local

		0	1 C	2 A	3 G	4 A	5 T	6 C	7 G	8 A
0		0	0	0	0	0	0	0	0	0
1	A	0	0	2	1	2	1	0	0	2
2	T	0	0	0	1	1	4	3	2	1
3	T	0	0	0	0	0	3	3	2	1
4	A	0	0	2	1	2	2	2	2	4
5	C	0	2	1	1	1	1	4	3	3
7	A	0	1	4	3	3	2	3	3	5
8	T	0	0	3	3	2	5	4	3	4
9	C	0	2	2	2	2	4	7	6	5

Matrice:

	A	C	G	T	-
A	2	-1	-1	-1	-1
C	-1	2	-1	-1	-1
G	-1	-1	2	-1	-1
T	-1	-1	-1	2	-1
-	-1	-1	-1	-1	

Alignement Local

		0	1 C	2 A	3 G	4 A	5 T	6 C	7 G	8 A
0		0	0	0	0	0	0	0	0	0
1	A	0	0	2	1	2	1	0	0	2
2	T	0	0	0	1	1	4	3	2	1
3	T	0	0	0	0	0	3	3	2	1
4	A	0	0	2	1	2	2	2	2	4
5	C	0	2	1	1	1	1	4	3	3
7	A	0	1	4	3	3	2	3	3	5
8	T	0	0	3	3	2	5	4	3	4
9	C	0	2	2	2	2	4	7	6	5

Matrice:

	A	C	G	T	-
A	2	-1	-1	-1	-1
C	-1	2	-1	-1	-1
G	-1	-1	2	-1	-1
T	-1	-1	-1	2	-1
-	-1	-1	-1	-1	

Alignement Local

		0	1 C	2 A	3 G	4 A	5 T	6 C	7 G	8 A
0		0	0	0	0	0	0	0	0	0
1	A	0	0	2	1	2	1	0	0	2
2	T	0	0	0	1	1	4	3	2	1
3	T	0	0	0	0	0	3	3	2	1
4	A	0	0	2	1	2	2	2	2	4
5	C	0	2	1	1	1	1	4	3	3
7	A	0	1	4	3	3	2	3	3	5
8	T	0	0	3	3	2	5	4	3	4
9	C	0	2	2	2	2	4	7	6	5

Matrice:

	A	C	G	T	-
A	2	-1	-1	-1	-1
C	-1	2	-1	-1	-1
G	-1	-1	2	-1	-1
T	-1	-1	-1	2	-1
-	-1	-1	-1	-1	

Alignement Local

		0	1 C	2 A	3 G	4 A	5 T	6 C	7 G	8 A
0		0	0	0	0	0	0	0	0	0
1	A	0	0	2	1	2	1	0	0	2
2	T	0	0	0	1	1	4	3	2	1
3	T	0	0	0	0	0	3	3	2	1
4	A	0	0	2	1	2	2	2	2	4
5	C	0	2	1	1	1	1	4	3	3
7	A	0	1	4	3	3	2	3	3	5
8	T	0	0	3	3	2	5	4	3	4
9	C	0	2	2	2	2	4	7	6	5

Matrice:

	A	C	G	T	-
A	2	-1	-1	-1	-1
C	-1	2	-1	-1	-1
G	-1	-1	2	-1	-1
T	-1	-1	-1	2	-1
-	-1	-1	-1	-1	

Exercice

Exercice

Calculer l'alignement local de *ACGCACGT* *CAGGCTGG* avec la même fonction de score que l'exemple précédent.

LCS (plus longue sous-séquence commune)

Definition (Sous-séquence)

La séquence $s = s_0 \ldots s_{l-1}$ est une sous séquence de u (de longueur n) si l'on peut construire s à partir de u en retirant lui $n - l$ symboles.

LCS (plus longue sous-séquence commune)

Definition (Sous-séquence)

La séquence $s = s_0 \ldots s_{l-1}$ est une sous séquence de u (de longueur n) si l'on peut construire s à partir de u en retirant lui $n - l$ symboles.

LCS:

Soient deux séquences u et v. Trouver la plus longue sous-séquence commune à u et v

LCS

		0	1 C	2 A	3 G	4 A	5 T	6 C	7 G	8 A
0		0								
1	A									
2	T									
3	G									
4	A									
5	C									
7	A									
8	T									
9	G									

Matrice:

	A	C	G	T	-
A	1	0	0	0	0
C	0	1	0	0	0
G	0	0	1	0	0
T	0	0	0	1	0
-	0	0	0	0	

LCS

		0	1 C	2 A	3 G	4 A	5 T	6 C	7 G	8 A
0		0	0	0	0	0	0	0	0	0
1	A	0								
2	T	0								
3	G	0								
4	A	0								
5	C	0								
7	A	0								
8	T	0								
9	G	0								

Matrice:

	A	C	G	T	-
A	1	0	0	0	0
C	0	1	0	0	0
G	0	0	1	0	0
T	0	0	0	1	0
-	0	0	0	0	

LCS

		0	1 C	2 A	3 G	4 A	5 T	6 C	7 G	8 A
0		0	0	0	0	0	0	0	0	0
1	A	0	0	1	0	1	0	0	0	1
2	T	0								
3	G	0								
4	A	0								
5	C	0								
7	A	0								
8	T	0								
9	G	0								

Matrice:

	A	C	G	T	-
A	1	0	0	0	0
C	0	1	0	0	0
G	0	0	1	0	0
T	0	0	0	1	0
-	0	0	0	0	

LCS

		0	1 C	2 A	3 G	4 A	5 T	6 C	7 G	8 A
0		0	0	0	0	0	0	0	0	0
1	A	0	0	1	0	1	0	0	0	1
2	T	0	0	1	1	1	2	2	2	2
3	G	0								
4	A	0								
5	C	0								
7	A	0								
8	T	0								
9	G	0								

Matrice:

	A	C	G	T	-
A	1	0	0	0	0
C	0	1	0	0	0
G	0	0	1	0	0
T	0	0	0	1	0
-	0	0	0	0	

LCS

		0	1 C	2 A	3 G	4 A	5 T	6 C	7 G	8 A
0		0	0	0	0	0	0	0	0	0
1	A	0	0	1	0	1	0	0	0	1
2	T	0	0	1	1	1	2	2	2	2
3	G	0	0	1	2	2	2	2	3	3
4	A	0								
5	C	0								
7	A	0								
8	T	0								
9	G	0								

Matrice:

	A	C	G	T	-
A	1	0	0	0	0
C	0	1	0	0	0
G	0	0	1	0	0
T	0	0	0	1	0
-	0	0	0	0	

LCS

		0	1 C	2 A	3 G	4 A	5 T	6 C	7 G	8 A
0		0	0	0	0	0	0	0	0	0
1	A	0	0	1	0	1	0	0	0	1
2	T	0	0	1	1	1	2	2	2	2
3	G	0	0	1	2	2	2	2	3	3
4	A	0	0	1	2	3	3	3	3	4
5	C	0								
7	A	0								
8	T	0								
9	G	0								

Matrice:

	A	C	G	T	-
A	1	0	0	0	0
C	0	1	0	0	0
G	0	0	1	0	0
T	0	0	0	1	0
-	0	0	0	0	

LCS

		0	1 C	2 A	3 G	4 A	5 T	6 C	7 G	8 A
0		0	0	0	0	0	0	0	0	0
1	A	0	0	1	0	1	0	0	0	1
2	T	0	0	1	1	1	2	2	2	2
3	G	0	0	1	2	2	2	2	3	3
4	A	0	0	1	2	3	3	3	3	4
5	C	0	1	1	2	3	3	4	4	4
7	A	0								
8	T	0								
9	G	0								

Matrice:

	A	C	G	T	-
A	1	0	0	0	0
C	0	1	0	0	0
G	0	0	1	0	0
T	0	0	0	1	0
-	0	0	0	0	

LCS

		0	1 C	2 A	3 G	4 A	5 T	6 C	7 G	8 A
0		0	0	0	0	0	0	0	0	0
1	A	0	0	1	0	1	0	0	0	1
2	T	0	0	1	1	1	2	2	2	2
3	G	0	0	1	2	2	2	2	3	3
4	A	0	0	1	2	3	3	3	3	4
5	C	0	1	1	2	3	3	4	4	4
7	A	0	1	2	2	3	3	4	4	5
8	T	0								
9	G	0								

Matrice:

	A	C	G	T	-
A	1	0	0	0	0
C	0	1	0	0	0
G	0	0	1	0	0
T	0	0	0	1	0
-	0	0	0	0	

LCS

		0	1 C	2 A	3 G	4 A	5 T	6 C	7 G	8 A
0		0	0	0	0	0	0	0	0	0
1	A	0	0	1	0	1	0	0	0	1
2	T	0	0	1	1	1	2	2	2	2
3	G	0	0	1	2	2	2	2	3	3
4	A	0	0	1	2	3	3	3	3	4
5	C	0	1	1	2	3	3	4	4	4
7	A	0	1	2	2	3	3	4	4	5
8	T	0	1	2	2	3	4	4	4	5
9	G	0								

Matrice:

	A	C	G	T	-
A	1	0	0	0	0
C	0	1	0	0	0
G	0	0	1	0	0
T	0	0	0	1	0
-	0	0	0	0	

LCS

		0	1 C	2 A	3 G	4 A	5 T	6 C	7 G	8 A
0		0	0	0	0	0	0	0	0	0
1	A	0	0	1	0	1	0	0	0	1
2	T	0	0	1	1	1	2	2	2	2
3	G	0	0	1	2	2	2	2	3	3
4	A	0	0	1	2	3	3	3	3	4
5	C	0	1	1	2	3	3	4	4	4
7	A	0	1	2	2	3	3	4	4	5
8	T	0	1	2	2	3	4	4	4	5
9	G	0	1	2	3	3	4	4	5	5

Matrice:

	A	C	G	T	-
A	1	0	0	0	0
C	0	1	0	0	0
G	0	0	1	0	0
T	0	0	0	1	0
-	0	0	0	0	

LCS

		0	1 C	2 A	3 G	4 A	5 T	6 C	7 G	8 A
0		0	0	0	0	0	0	0	0	0
1	A	0	0	1	0	1	0	0	0	1
2	T	0	0	1	1	1	2	2	2	2
3	G	0	0	1	2	2	2	2	3	3
4	A	0	0	1	2	3	3	3	3	4
5	C	0	1	1	2	3	3	4	4	4
7	A	0	1	2	2	3	3	4	4	5
8	T	0	1	2	2	3	4	4	4	5
9	G	0	1	2	3	3	4	4	5	5

Matrice:

	A	C	G	T	-
A	1	0	0	0	0
C	0	1	0	0	0
G	0	0	1	0	0
T	0	0	0	1	0
-	0	0	0	0	

LCS

		0	1 C	2 A	3 G	4 A	5 T	6 C	7 G	8 A
0		0	0	0	0	0	0	0	0	0
1	A	0	0	1	0	1	0	0	0	1
2	T	0	0	1	1	1	2	2	2	2
3	G	0	0	1	2	2	2	2	3	3
4	A	0	0	1	2	3	3	3	3	4
5	C	0	1	1	2	3	3	4	4	4
7	A	0	1	2	2	3	3	4	4	5
8	T	0	1	2	2	3	4	4	4	5
9	G	0	1	2	3	3	4	4	5	5

Matrice:

	A	C	G	T	-
A	1	0	0	0	0
C	0	1	0	0	0
G	0	0	1	0	0
T	0	0	0	1	0
-	0	0	0	0	

LCS

		0	1 C	2 A	3 G	4 A	5 T	6 C	7 G	8 A
0		0	0	0	0	0	0	0	0	0
1	A	0	0	1	0	1	0	0	0	1
2	T	0	0	1	1	1	2	2	2	2
3	G	0	0	1	2	2	2	2	3	3
4	A	0	0	1	2	3	3	3	3	4
5	C	0	1	1	2	3	3	4	4	4
7	A	0	1	2	2	3	3	4	4	5
8	T	0	1	2	2	3	4	4	4	5
9	G	0	1	2	3	3	4	4	5	5

Matrice:

	A	C	G	T	-
A	1	0	0	0	0
C	0	1	0	0	0
G	0	0	1	0	0
T	0	0	0	1	0
-	0	0	0	0	

LCS

		0	1 C	2 A	3 G	4 A	5 T	6 C	7 G	8 A
0		0	0	0	0	0	0	0	0	0
1	A	0	0	1	0	1	0	0	0	1
2	T	0	0	1	1	1	2	2	2	2
3	G	0	0	1	2	2	2	2	3	3
4	A	0	0	1	2	3	3	3	3	4
5	C	0	1	1	2	3	3	4	4	4
7	A	0	1	2	2	3	3	4	4	5
8	T	0	1	2	2	3	4	4	4	5
9	G	0	1	2	3	3	4	4	5	5

Matrice:

	A	C	G	T	-
A	1	0	0	0	0
C	0	1	0	0	0
G	0	0	1	0	0
T	0	0	0	1	0
-	0	0	0	0	

LCS

		0	1 C	2 A	3 G	4 A	5 T	6 C	7 G	8 A
0		0	0	0	0	0	0	0	0	0
1	A	0	0	1	0	1	0	0	0	1
2	T	0	0	1	1	1	2	2	2	2
3	G	0	0	1	2	2	2	2	3	3
4	A	0	0	1	2	3	3	3	3	4
5	C	0	1	1	2	3	3	4	4	4
7	A	0	1	2	2	3	3	4	4	5
8	T	0	1	2	2	3	4	4	4	5
9	G	0	1	2	3	3	4	4	5	5

Matrice:

	A	C	G	T	-
A	1	0	0	0	0
C	0	1	0	0	0
G	0	0	1	0	0
T	0	0	0	1	0
-	0	0	0	0	

LCS

		0	1 C	2 A	3 G	4 A	5 T	6 C	7 G	8 A
0		0	0	0	0	0	0	0	0	0
1	A	0	0	1	0	1	0	0	0	1
2	T	0	0	1	1	1	2	2	2	2
3	G	0	0	1	2	2	2	2	3	3
4	A	0	0	1	2	3	3	3	3	4
5	C	0	1	1	2	3	3	4	4	4
7	A	0	1	2	2	3	3	4	4	5
8	T	0	1	2	2	3	4	4	4	5
9	G	0	1	2	3	3	4	4	5	5

Matrice:

	A	C	G	T	-
A	1	0	0	0	0
C	0	1	0	0	0
G	0	0	1	0	0
T	0	0	0	1	0
-	0	0	0	0	

A propos des gaps

Fonction de gap affine:

- Coût d'ouverture de gap
- Coût d'extension

A propos des gaps

Reformulation de l'alignement:

Alignement avec fonction de gap affine

Soient deux séquences u et v et deux valeurs o et e. On cherche un alignement de u' et v' des deux séquences. On désigne le symbole '-' par "espace". Toute suite d'espace dans u' ou v' sera appelée "gap". Le problème consiste à maximiser le score

$$\sum_i [cost(u'_i, v'_i) + \#gap * o + \#espace * e]$$

avec $cost(a,' -') = cost('-', a) = 0$

Fonction de gap affine

Le calcul du score repose sur l'utilisation de trois matrices:

- $V(i,j)$: contient la valeur du meilleur alignement entre u_i et v_j.
- $E(i,j)$: contient la valeur de l'alignement de u_i et v_j avec u en court d'extension de gap
- $F(i,j)$: contient la valeur de l'alignement de u_i et v_j avec v en court d'extension de gap

Conditions initiales:

- $V(i,0) = E(i,0) = o + i * e$
- $V(0,j) = F(0,j) = o + j * e$

Fonction de gap affine

Conditions initiales:

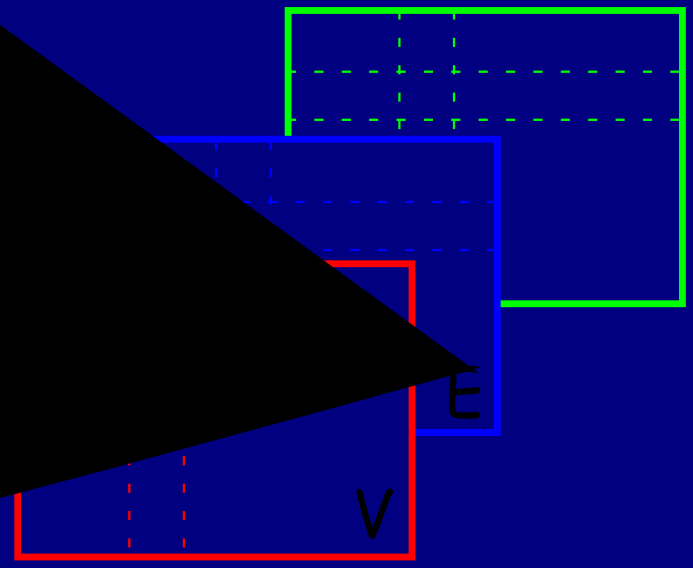
- $V(i,0) = E(i,0) = o + i * e$
- $V(0,j) = F(0,j) = o + j * e$

$$V(i,j) = Max \begin{cases} E(i,j) \\ F(i,j) \\ V(i-1,j-1) + cost(u_i, v_j) \end{cases}$$

$$E(i,j) = Max \begin{cases} E(i,j-1) + e \\ V(i,j-1) + o + e \end{cases}$$

$$F(i,j) = Max \begin{cases} F(i-1,j) + e \\ V(i-1,j) + o + e \end{cases}$$

ap affine



E

$F(i,j)$

$\ldots\ 1,j) + o + e$

Fonction de gap affine

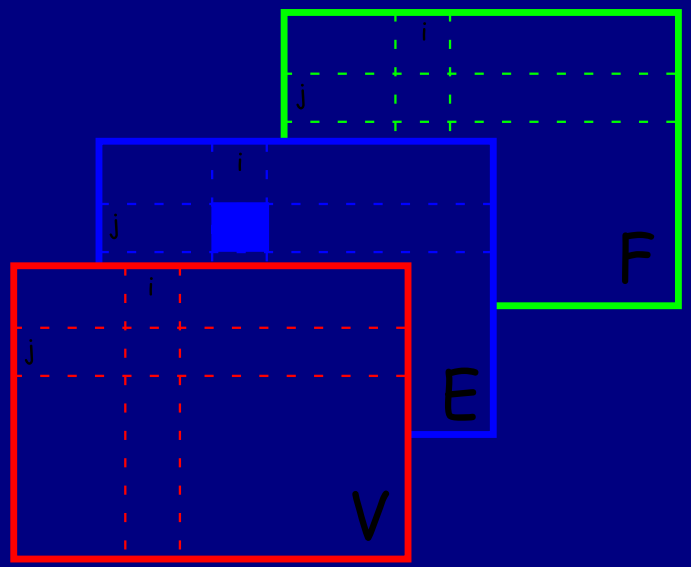
Conditions initiales:

- $V(i,0) = E(i,0) = o + i * e$
- $V(0,j) = F(0,j) = o + j * e$

$$V(i,j) = Max \begin{cases} E(i,j) \\ F(i,j) \\ V(i-1,j-1) + cost(u_i, v_j) \end{cases}$$

$$E(i,j) = Max \begin{cases} E(i,j-1) + e \\ V(i,j-1) + o + e \end{cases}$$

$$F(i,j) = Max \begin{cases} F(i-1,j) + e \\ V(i-1,j) + o + e \end{cases}$$



Fonction de gap affine

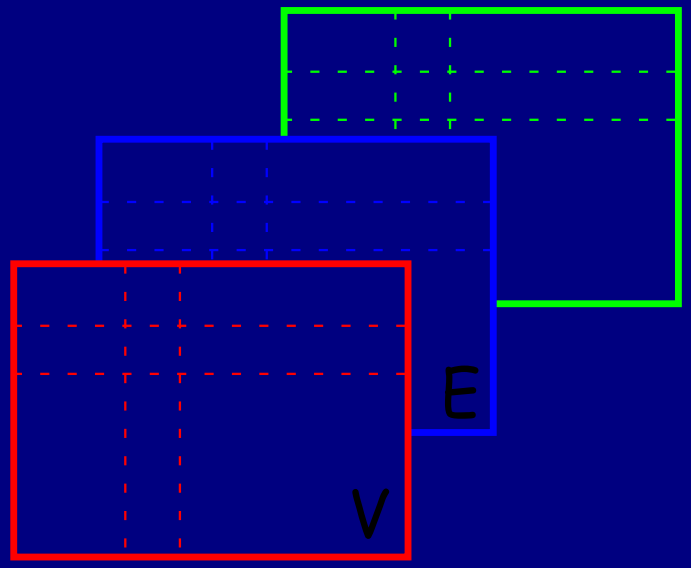
Conditions initiales:

- $V(i,0) = E(i,0) = o + i * e$
- $V(0,j) = F(0,j) = o + j * e$

$$V(i,j) = Max \begin{cases} E(i,j) \\ F(i,j) \\ V(i-1,j-1) + cost(u_i, v_j) \end{cases}$$

$$E(i,j) = Max \begin{cases} E(i,j-1) + e \\ V(i,j-1) + o + e \end{cases}$$

$$F(i,j) = Max \begin{cases} F(i-1,j) + e \\ V(i-1,j) + o + e \end{cases}$$



Fonction de gap affine

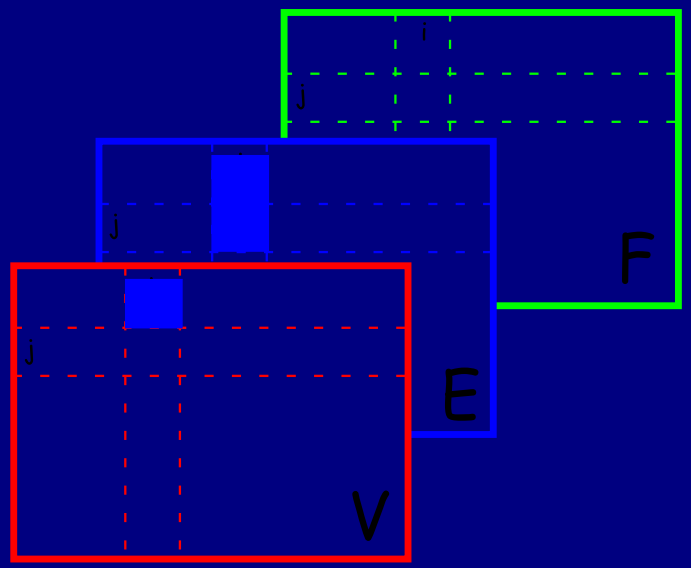
Conditions initiales:

- $V(i,0) = E(i,0) = o + i * e$
- $V(0,j) = F(0,j) = o + j * e$

$$V(i,j) = Max \begin{cases} E(i,j) \\ F(i,j) \\ V(i-1,j-1) + cost(u_i, v_j) \end{cases}$$

$$E(i,j) = Max \begin{cases} E(i,j-1) + e \\ V(i,j-1) + o + e \end{cases}$$

$$F(i,j) = Max \begin{cases} F(i-1,j) + e \\ V(i-1,j) + o + e \end{cases}$$



Fonction de gap affine

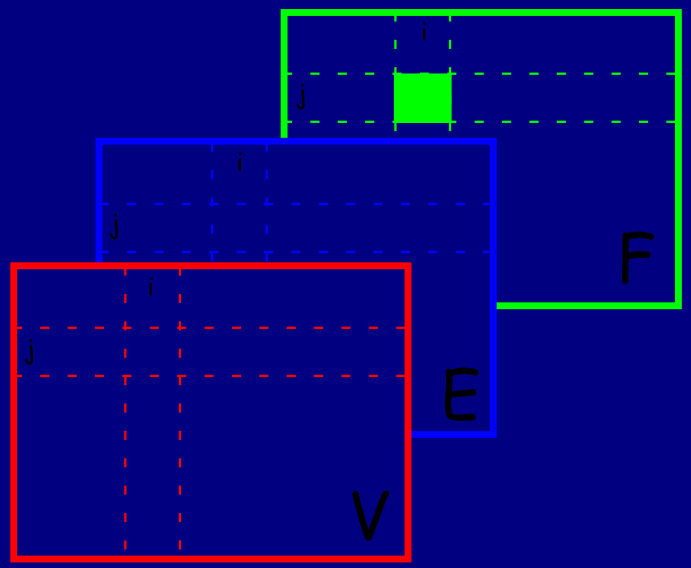
Conditions initiales:

- $V(i,0) = E(i,0) = o + i * e$
- $V(0,j) = F(0,j) = o + j * e$

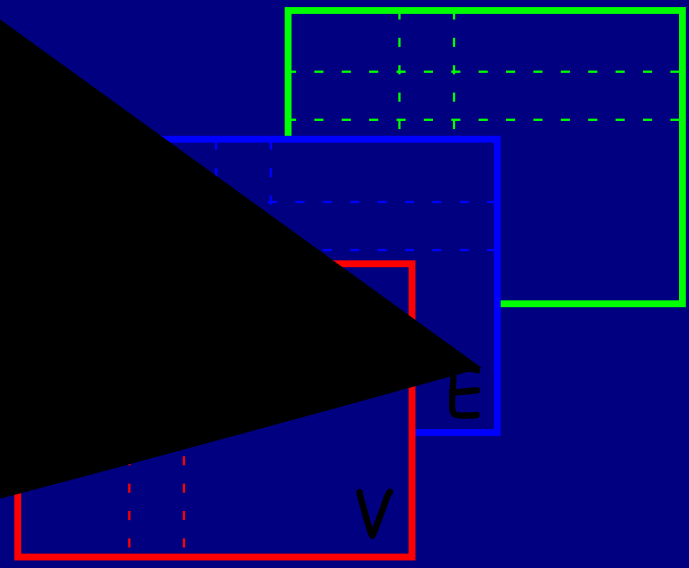
$$V(i,j) = Max \begin{cases} E(i,j) \\ F(i,j) \\ V(i-1,j-1) + cost(u_i, v_j) \end{cases}$$

$$E(i,j) = Max \begin{cases} E(i,j-1) + e \\ V(i,j-1) + o + e \end{cases}$$

$$F(i,j) = Max \begin{cases} F(i-1,j) + e \\ V(i-1,j) + o + e \end{cases}$$



ap affine



E

$F(i,j)$

$+1,j) + o + e$

Fonction de gap affine

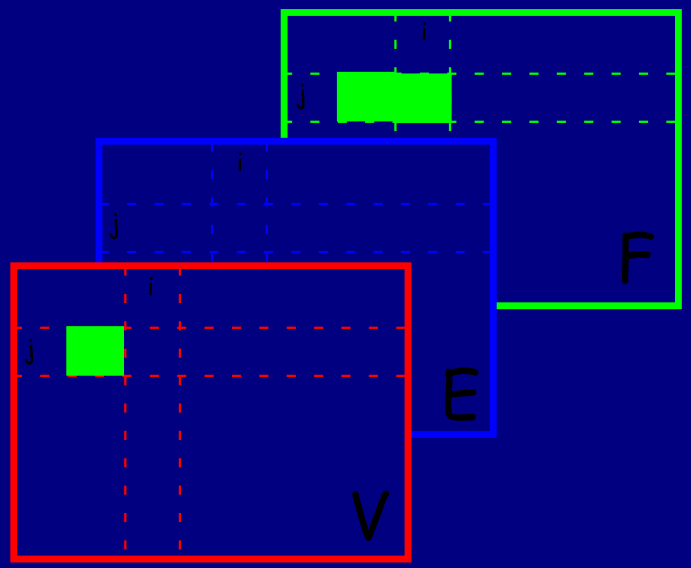
Conditions initiales:

- $V(i,0) = E(i,0) = o + i * e$
- $V(0,j) = F(0,j) = o + j * e$

$$V(i,j) = Max \begin{cases} E(i,j) \\ F(i,j) \\ V(i-1,j-1) + cost(u_i, v_j) \end{cases}$$

$$E(i,j) = Max \begin{cases} E(i,j-1) + e \\ V(i,j-1) + o + e \end{cases}$$

$$F(i,j) = Max \begin{cases} F(i-1,j) + e \\ V(i-1,j) + o + e \end{cases}$$



Fonction de gap affine

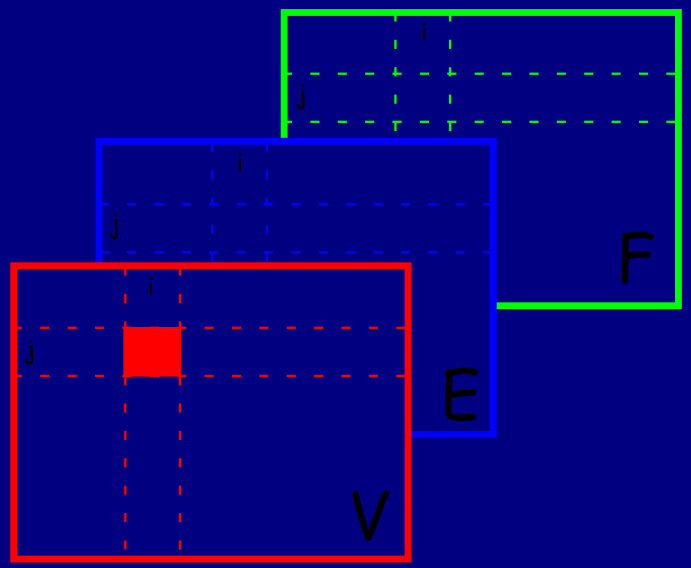
Conditions initiales:

- $V(i,0) = E(i,0) = o + i * e$
- $V(0,j) = F(0,j) = o + j * e$

$$V(i,j) = Max \begin{cases} E(i,j) \\ F(i,j) \\ V(i-1,j-1) + cost(u_i, v_j) \end{cases}$$

$$E(i,j) = Max \begin{cases} E(i,j-1) + e \\ V(i,j-1) + o + e \end{cases}$$

$$F(i,j) = Max \begin{cases} F(i-1,j) + e \\ V(i-1,j) + o + e \end{cases}$$



Fonction de gap affine

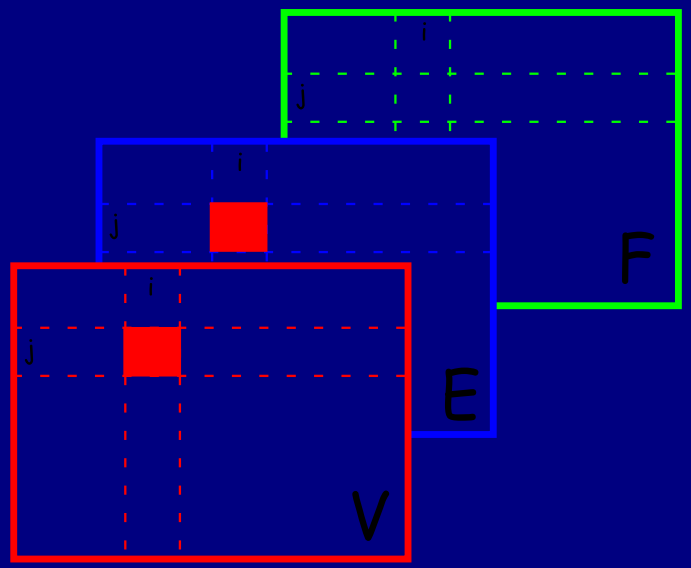
Conditions initiales:

- $V(i,0) = E(i,0) = o + i * e$
- $V(0,j) = F(0,j) = o + j * e$

$$V(i,j) = Max \begin{cases} E(i,j) \\ F(i,j) \\ V(i-1,j-1) + cost(u_i, v_j) \end{cases}$$

$$E(i,j) = Max \begin{cases} E(i,j-1) + e \\ V(i,j-1) + o + e \end{cases}$$

$$F(i,j) = Max \begin{cases} F(i-1,j) + e \\ V(i-1,j) + o + e \end{cases}$$



Fonction de gap affine

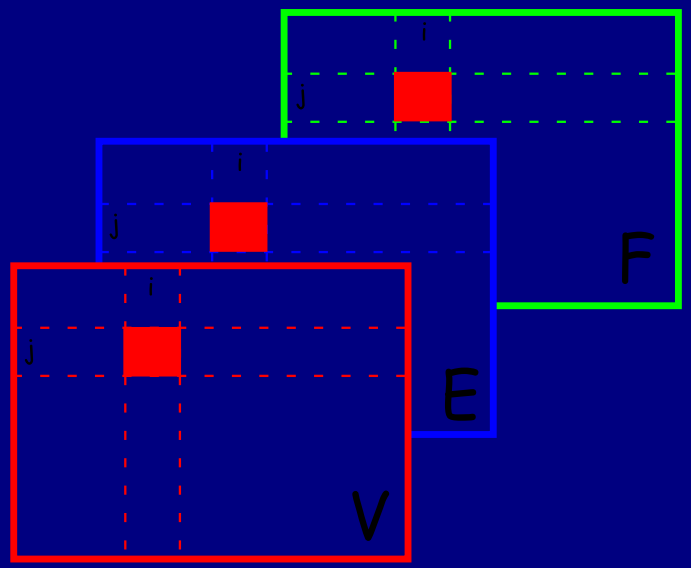
Conditions initiales:

- $V(i,0) = E(i,0) = o + i * e$
- $V(0,j) = F(0,j) = o + j * e$

$$V(i,j) = Max \begin{cases} E(i,j) \\ F(i,j) \\ V(i-1,j-1) + cost(u_i, v_j) \end{cases}$$

$$E(i,j) = Max \begin{cases} E(i,j-1) + e \\ V(i,j-1) + o + e \end{cases}$$

$$F(i,j) = Max \begin{cases} F(i-1,j) + e \\ V(i-1,j) + o + e \end{cases}$$



Fonction de gap affine

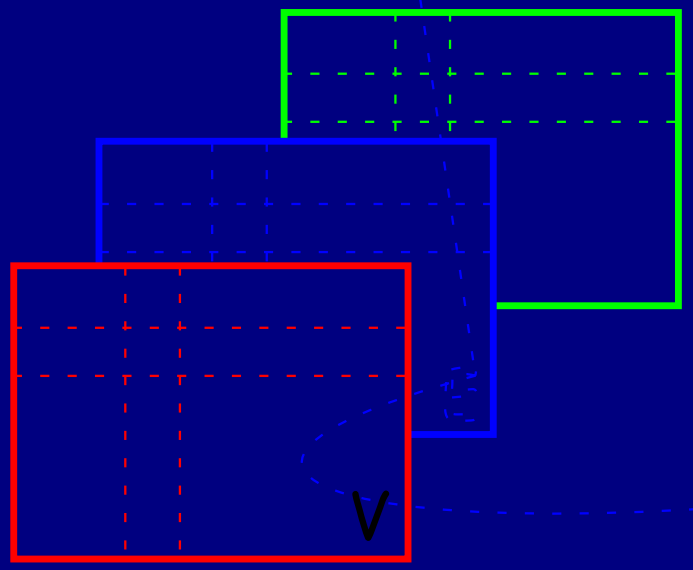
Conditions initiales:

- $V(i,0) = E(i,0) = o + i * e$
- $V(0,j) = F(0,j) = o + j * e$

$$V(i,j) = Max \begin{cases} E(i,j) \\ F(i,j) \\ V(i-1,j-1) + cost(u_i, v_j) \end{cases}$$

$$E(i,j) = Max \begin{cases} E(i,j-1) + e \\ V(i,j-1) + o + e \end{cases}$$

$$F(i,j) = Max \begin{cases} F(i-1,j) + e \\ V(i-1,j) + o + e \end{cases}$$



Fonction de gap affine

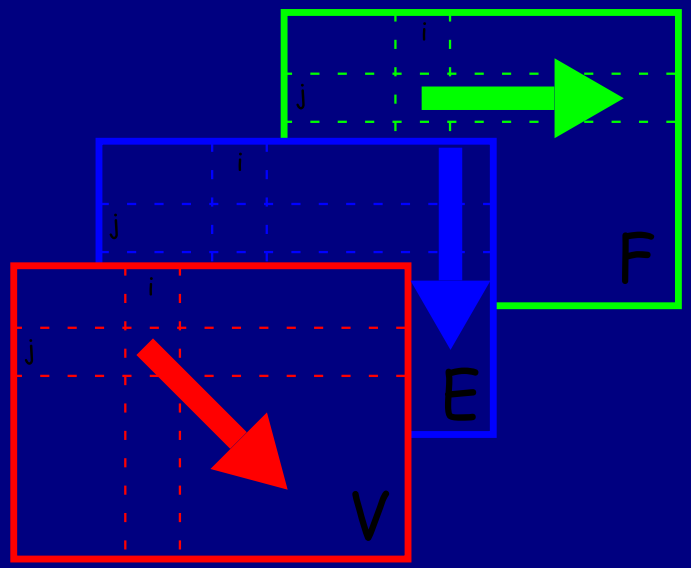
Conditions initiales:

- $V(i,0) = E(i,0) = o + i * e$
- $V(0,j) = F(0,j) = o + j * e$

$$V(i,j) = Max \begin{cases} E(i,j) \\ F(i,j) \\ V(i-1,j-1) + cost(u_i, v_j) \end{cases}$$

$$E(i,j) = Max \begin{cases} E(i,j-1) + e \\ V(i,j-1) + o + e \end{cases}$$

$$F(i,j) = Max \begin{cases} F(i-1,j) + e \\ V(i-1,j) + o + e \end{cases}$$



Exemple

$V(i,j) =$

$Max\{E(i,j), F(i,j), V(i-1,j-1)+c\}$

		0	1 C	2 A	3 G
0		0			
1	C				
2	A				
3	T				
4	T				
5	T				
6	G				

$cost(a,a) = +1$

$E(i,j) =$

$Max\{E(i,j-1)+e, V(i,j-1)+o+e\}$

	0	1 C	2 A	3 G
	X			
C				
A				
T				
T				
T				
G				

$cost(a,b) = -1$

$F(i,j) =$

$Max\{F(i-1,j)+e, V(i-1,j)+o+e\}$

	0	1 C	2 A	3 G
	X			
C				
A				
T				
T				
T				
G				

$o = -1$ $e = -1$

Exemple

$V(i,j) =$

$Max\{E(i,j), F(i,j), V(i-1,j-1)+c\}$

		0	1 C	2 A	3 G
0		0			
1	C				
2	A				
3	T				
4	T				
5	T				
6	G				

$cost(a,a) = +1$

$E(i,j) =$

$Max\{E(i,j-1)+e, V(i,j-1)+o+e\}$

	0	1 C	2 A	3 G
	X	X	X	X
C				
A				
T				
T				
T				
G				

$cost(a,b) = -1$

$F(i,j) =$

$Max\{F(i-1,j)+e, V(i-1,j)+o+e\}$

	0	1 C	2 A	3 G
	X			
C				
A				
T				
T				
T				
G				

$o = -1$ $e = -1$

Exemple

$V(i,j) =$

$Max\{E(i,j), F(i,j), V(i-1,j-1)+c\}$

		0	1 C	2 A	3 G
0		0			
1	C				
2	A				
3	T				
4	T				
5	T				
6	G				

$cost(a,a) = +1$

$E(i,j) =$

$Max\{E(i,j-1)+e, V(i,j-1)+o+e\}$

	0	1 C	2 A	3 G
	X	X	X	X
C				
A				
T				
T				
T				
G				

$cost(a,b) = -1$

$F(i,j) =$

$Max\{F(i-1,j)+e, V(i-1,j)+o+e\}$

	0	1 C	2 A	3 G
	X	-2		
C				
A				
T				
T				
T				
G				

$o = -1$ $e = -1$

Exemple

$V(i,j) =$

$Max\{E(i,j), F(i,j), V(i-1,j-1)+c\}$

		0	1 C	2 A	3 G
0		0	-2		
1	C				
2	A				
3	T				
4	T				
5	T				
6	G				

$E(i,j) =$

$Max\{E(i,j-1)+e, V(i,j-1)+o+e\}$

	0	1 C	2 A	3 G
	X	X	X	X
C				
A				
T				
T				
T				
G				

$F(i,j) =$

$Max\{F(i-1,j)+e, V(i-1,j)+o+e\}$

	0	1 C	2 A	3 G
	X	-2		
C				
A				
T				
T				
T				
G				

$cost(a,a) = +1$ $cost(a,b) = -1$ $o = -1$ $e = -1$

Exemple

$V(i,j) =$

$Max\{E(i,j), F(i,j), V(i-1,j-1)+c\}$

		0	1 C	2 A	3 G
0		0	-2		
1	C				
2	A				
3	T				
4	T				
5	T				
6	G				

$cost(a,a) = +1$

$E(i,j) =$

$Max\{E(i,j-1)+e, V(i,j-1)+o+e\}$

	0	1 C	2 A	3 G
	X	X	X	X
C				
A				
T				
T				
T				
G				

$cost(a,b) = -1$

$F(i,j) =$

$Max\{F(i-1,j)+e, V(i-1,j)+o+e\}$

	0	1 C	2 A	3 G
	X	-2	-3	
C				
A				
T				
T				
T				
G				

$o = -1$ $e = -1$

Exemple

$V(i,j) =$

$Max\{E(i,j), F(i,j), V(i-1,j-1)+c\}$

		0	1 C	2 A	3 G
0		0	-2	-3	
1	C				
2	A				
3	T				
4	T				
5	T				
6	G				

$E(i,j) =$

$Max\{E(i,j-1)+e, V(i,j-1)+o+e\}$

	0	1 C	2 A	3 G
	X	X	X	X
C				
A				
T				
T				
T				
G				

$F(i,j) =$

$Max\{F(i-1,j)+e, V(i-1,j)+o+e\}$

	0	1 C	2 A	3 G
	X	-2	-3	
C				
A				
T				
T				
T				
G				

$cost(a,a) = +1$ $cost(a,b) = -1$ $o = -1$ $e = -1$

Exemple

$V(i,j) =$

$Max\{E(i,j), F(i,j), V(i-1,j-1)+c\}$

		0	1 C	2 A	3 G
0		0	-2	-3	
1	C				
2	A				
3	T				
4	T				
5	T				
6	G				

$E(i,j) =$

$Max\{E(i,j-1)+e, V(i,j-1)+o+e\}$

	0	1 C	2 A	3 G
	X	X	X	X
C				
A				
T				
T				
T				
G				

$F(i,j) =$

$Max\{F(i-1,j)+e, V(i-1,j)+o+e\}$

	0	1 C	2 A	3 G
	X	-2	-3	-4
C				
A				
T				
T				
T				
G				

$cost(a,a) = +1$ $cost(a,b) = -1$ $o = -1$ $e = -1$

Exemple

$V(i,j) =$

$Max\{E(i,j), F(i,j), V(i-1,j-1)+c\}$

	0	1 C	2 A	3 G
0	0	-2	-3	-4

Exemple

$V(i,j) =$

$Max\{E(i,j), F(i,j), V(i-1,j-1)+c\}$

		0	1 C	2 A	3 G
0		0	-2	-3	-4
1	C	-2			
2	A	-3			
3	T	-4			
4	T	-5			
5	T	-6			
6	G	-7			

$E(i,j) =$

$Max\{E(i,j-1)+e, V(i,j-1)+o+e\}$

	0	1 C	2 A	3 G
	X	X	X	X
C	-2			
A	-3			
T	-4			
T	-5			
T	-6			
G	-7			

$F(i,j) =$

$Max\{F(i-1,j)+e, V(i-1,j)+o+e\}$

	0	1 C	2 A	3 G
	X	-2	-3	-4
C	X			
A	X			
T	X			
T	X			
T	X			
G	X			

$cost(a,a) = +1$ $cost(a,b) = -1$ $o = -1$ $e = -1$

Exemple

$V(i,j) =$

$Max\{E(i,j), F(i,j), V(i-1,j-1)+c\}$

		0	1 C	2 A	3 G
0		0	-2	-3	-4
1	C	-2			
2	A	-3			
3	T	-4			
4	T	-5			
5	T	-6			
6	G	-7			

$E(i,j) =$

$Max\{E(i,j-1)+e, V(i,j-1)+o+e\}$

	0	1 C	2 A	3 G
	X	X	X	X
C	-2	-4		
A	-3			
T	-4			
T	-5			
T	-6			
G	-7			

$F(i,j) =$

$Max\{F(i-1,j)+e, V(i-1,j)+o+e\}$

	0	1 C	2 A	3 G
	X	-2	-3	-4
C	X			
A	X			
T	X			
T	X			
T	X			
G	X			

$cost(a,a) = +1$ $cost(a,b) = -1$ $o = -1$ $e = -1$

Exemple

$V(i,j) =$

$Max\{E(i,j), F(i,j), V(i-1,j-1)+c\}$

		0	1 C	2 A	3 G
0		0	-2	-3	-4
1	C	-2			
2	A	-3			
3	T	-4			
4	T	-5			
5	T	-6			
6	G	-7			

$E(i,j) =$

$Max\{E(i,j-1)+e, V(i,j-1)+o+e\}$

	0	1 C	2 A	3 G
	X	X	X	X
C	-2	-4		
A	-3			
T	-4			
T	-5			
T	-6			
G	-7			

$F(i,j) =$

$Max\{F(i-1,j)+e, V(i-1,j)+o+e\}$

	0	1 C	2 A	3 G
	X	-2	-3	-4
C	X	-4		
A	X			
T	X			
T	X			
T	X			
G	X			

$cost(a,a) = +1$ $cost(a,b) = -1$ $o = -1$ $e = -1$

Exemple

$V(i,j) =$

$Max\{E(i,j), F(i,j), V(i-1,j-1)+c\}$

		0	1 C	2 A	3 G
0		0	-2	-3	-4
1	C	-2	1		
2	A	-3			
3	T	-4			
4	T	-5			
5	T	-6			
6	G	-7			

$E(i,j) =$

$Max\{E(i,j-1)+e, V(i,j-1)+o+e\}$

	0	1 C	2 A	3 G
	X	X	X	X
C	-2	-4		
A	-3			
T	-4			
T	-5			
T	-6			
G	-7			

$F(i,j) =$

$Max\{F(i-1,j)+e, V(i-1,j)+o+e\}$

	0	1 C	2 A	3 G
	X	-2	-3	-4
C	X	-4		
A	X			
T	X			
T	X			
T	X			
G	X			

$cost(a,a) = +1$ $cost(a,b) = -1$ $o = -1$ $e = -1$

Exemple

$V(i,j) =$

$Max\{E(i,j), F(i,j), V(i-1,j-1)+c\}$

		0	1 C	2 A	3 G
0		0	-2	-3	-4
1	C	-2	1		
2	A	-3			
3	T	-4			
4	T	-5			
5	T	-6			
6	G	-7			

$cost(a,a) = +1$

$E(i,j) =$

$Max\{E(i,j-1)+e, V(i,j-1)+o+e\}$

	0	1 C	2 A	3 G
	X	X	X	X
C	-2	-4		
A	-3	-1		
T	-4			
T	-5			
T	-6			
G	-7			

$cost(a,b) = -1$

$F(i,j) =$

$Max\{F(i-1,j)+e, V(i-1,j)+o+e\}$

	0	1 C	2 A	3 G
	X	-2	-3	-4
C	X	-4		
A	X			
T	X			
T	X			
T	X			
G	X			

$o = -1$ $e = -1$

Exemple

$V(i,j) =$

$Max\{E(i,j), F(i,j), V(i-1,j-1)+c\}$

		0	1 C	2 A	3 G
0		0	-2	-3	-4
1	C	-2	1		
2	A	-3			
3	T	-4			
4	T	-5			
5	T	-6			
6	G	-7			

$E(i,j) =$

$Max\{E(i,j-1)+e, V(i,j-1)+o+e\}$

	0	1 C	2 A	3 G
	X	X	X	X
C	-2	-4		
A	-3	-1		
T	-4			
T	-5			
T	-6			
G	-7			

$F(i,j) =$

$Max\{F(i-1,j)+e, V(i-1,j)+o+e\}$

	0	1 C	2 A	3 G
	X	-2	-3	-4
C	X	-4		
A	X	-5		
T	X			
T	X			
T	X			
G	X			

$cost(a,a) = +1$ $cost(a,b) = -1$ $o = -1$ $e = -1$

Exemple

$V(i,j) =$

$Max\{E(i,j), F(i,j), V(i-1,j-1)+c\}$

		0	1 C	2 A	3 G
0		0	-2	-3	-4
1	C	-2	1		
2	A	-3	-1		
3	T	-4			
4	T	-5			
5	T	-6			
6	G	-7			

$E(i,j) =$

$Max\{E(i,j-1)+e, V(i,j-1)+o+e\}$

	0	1 C	2 A	3 G
	X	X	X	X
C	-2	-4		
A	-3	-1		
T	-4			
T	-5			
T	-6			
G	-7			

$F(i,j) =$

$Max\{F(i-1,j)+e, V(i-1,j)+o+e\}$

	0	1 C	2 A	3 G
	X	-2	-3	-4
C	X	-4		
A	X	-5		
T	X			
T	X			
T	X			
G	X			

$cost(a,a) = +1$ $cost(a,b) = -1$ $o = -1$ $e = -1$

Exemple

$V(i,j) =$

$Max\{E(i,j), F(i,j), V(i-1,j-1)+c\}$

		0	1 C	2 A	3 G
0		0	-2	-3	-4
1	C	-2	1		
2	A	-3	-1		
3	T	-4			
4	T	-5			
5	T	-6			
6	G	-7			

$cost(a,a) = +1$

$E(i,j) =$

$Max\{E(i,j-1)+e, V(i,j-1)+o+e\}$

	0	1 C	2 A	3 G
	X	X	X	X
C	-2	-4		
A	-3	-1		
T	-4	-2		
T	-5			
T	-6			
G	-7			

$cost(a,b) = -1$

$F(i,j) =$

$Max\{F(i-1,j)+e, V(i-1,j)+o+e\}$

	0	1 C	2 A	3 G
	X	-2	-3	-4
C	X	-4		
A	X	-5		
T	X			
T	X			
T	X			
G	X			

$o = -1$ $e = -1$

Exemple

$V(i,j) =$

$Max\{E(i,j), F(i,j), V(i-1,j-1)+c\}$

		0	1 C	2 A	3 G
0		0	-2	-3	-4
1	C	-2	1		
2	A	-3	-1		
3	T	-4			
4	T	-5			
5	T	-6			
6	G	-7			

$E(i,j) =$

$Max\{E(i,j-1)+e, V(i,j-1)+o+e\}$

	0	1 C	2 A	3 G
	X	X	X	X
C	-2	-4		
A	-3	-1		
T	-4	-2		
T	-5			
T	-6			
G	-7			

$F(i,j) =$

$Max\{F(i-1,j)+e, V(i-1,j)+o+e\}$

	0	1 C	2 A	3 G
	X	-2	-3	-4
C	X	-4		
A	X	-5		
T	X	-6		
T	X			
T	X			
G	X			

$cost(a,a) = +1$ $cost(a,b) = -1$ $o = -1$ $e = -1$

Exemple

$V(i,j) =$

$Max\{E(i,j), F(i,j), V(i-1,j-1)+c\}$

		0	1 C	2 A	3 G
0		0	-2	-3	-4
1	C	-2	1		
2	A	-3	-1		
3	T	-4	-2		
4	T	-5			
5	T	-6			
6	G	-7			

$E(i,j) =$

$Max\{E(i,j-1)+e, V(i,j-1)+o+e\}$

	0	1 C	2 A	3 G
	X	X	X	X
C	-2	-4		
A	-3	-1		
T	-4	-2		
T	-5			
T	-6			
G	-7			

$F(i,j) =$

$Max\{F(i-1,j)+e, V(i-1,j)+o+e\}$

	0	1 C	2 A	3 G
	X	-2	-3	-4
C	X	-4		
A	X	-5		
T	X	-6		
T	X			
T	X			
G	X			

$cost(a,a) = +1$ $cost(a,b) = -1$ $o = -1$ $e = -1$

Exemple

$V(i,j) =$

$Max\{E(i,j), F(i,j), V(i-1,j-1)+c\}$

		0	1 C	2 A	3 G
0		0	-2	-3	-4
1	C	-2	1		
2	A	-3	-1		
3	T	-4	-2		
4	T	-5			
5	T	-6			
6	G	-7			

$cost(a,a) = +1$

$E(i,j) =$

$Max\{E(i,j-1)+e, V(i,j-1)+o+e\}$

	0	1 C	2 A	3 G
	X	X	X	X
C	-2	-4		
A	-3	-1		
T	-4	-2		
T	-5	-3		
T	-6			
G	-7			

$cost(a,b) = -1$

$F(i,j) =$

$Max\{F(i-1,j)+e, V(i-1,j)+o+e\}$

	0	1 C	2 A	3 G
	X	-2	-3	-4
C	X	-4		
A	X	-5		
T	X	-6		
T	X			
T	X			
G	X			

$o = -1$ $e = -1$

Exemple

$V(i,j) =$

$Max\{E(i,j), F(i,j), V(i-1,j-1)+c\}$

		0	1 C	2 A	3 G
0		0	-2	-3	-4
1	C	-2	1		
2	A	-3	-1		
3	T	-4	-2		
4	T	-5			
5	T	-6			
6	G	-7			

$E(i,j) =$

$Max\{E(i,j-1)+e, V(i,j-1)+o+e\}$

	0	1 C	2 A	3 G
	X	X	X	X
C	-2	-4		
A	-3	-1		
T	-4	-2		
T	-5	-3		
T	-6			
G	-7			

$F(i,j) =$

$Max\{F(i-1,j)+e, V(i-1,j)+o+e\}$

	0	1 C	2 A	3 G
	X	-2	-3	-4
C	X	-4		
A	X	-5		
T	X	-6		
T	X	-7		
T	X			
G	X			

$cost(a,a) = +1$ $cost(a,b) = -1$ $o = -1$ $e = -1$

Exemple

$V(i,j) =$

$Max\{E(i,j), F(i,j), V(i-1,j-1)+c\}$

		0	1 C	2 A	3 G
0		0	-2	-3	-4
1	C	-2	1		
2	A	-3	-1		
3	T	-4	-2		
4	T	-5	-3		
5	T	-6			
6	G	-7			

$E(i,j) =$

$Max\{E(i,j-1)+e, V(i,j-1)+o+e\}$

	0	1 C	2 A	3 G
	X	X	X	X
C	-2	-4		
A	-3	-1		
T	-4	-2		
T	-5	-3		
T	-6			
G	-7			

$F(i,j) =$

$Max\{F(i-1,j)+e, V(i-1,j)+o+e\}$

	0	1 C	2 A	3 G
	X	-2	-3	-4
C	X	-4		
A	X	-5		
T	X	-6		
T	X	-7		
T	X			
G	X			

$cost(a,a) = +1$ $cost(a,b) = -1$ $o = -1$ $e = -1$

Exemple

$V(i,j) =$

$Max\{E(i,j), F(i,j), V(i-1,j-1)+c\}$

		0	1 C	2 A	3 G
0		0	-2	-3	-4
1	C	-2	1		
2	A	-3	-1		
3	T	-4	-2		
4	T	-5	-3		
5	T	-6			
6	G	-7			

$E(i,j) =$

$Max\{E(i,j-1)+e, V(i,j-1)+o+e\}$

	0	1 C	2 A	3 G
	X	X	X	X
C	-2	-4		
A	-3	-1		
T	-4	-2		
T	-5	-3		
T	-6	-4		
G	-7			

$F(i,j) =$

$Max\{F(i-1,j)+e, V(i-1,j)+o+e\}$

	0	1 C	2 A	3 G
	X	-2	-3	-4
C	X	-4		
A	X	-5		
T	X	-6		
T	X	-7		
T	X			
G	X			

$cost(a,a) = +1$ $cost(a,b) = -1$ $o = -1$ $e = -1$

Exemple

$V(i,j) =$

$Max\{E(i,j), F(i,j), V(i-1,j-1)+c\}$

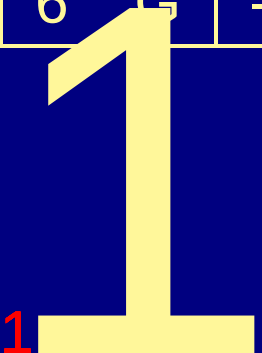
		0	1 C	2 A	3 G
0		0	-2	-3	-4
1	C	-2	1		
2	A	-3	-1		
3	T	-4	-2		
4	T	-5	-3		
5	T	-6			
6	G	-7			

$E(i,j) =$

$Max\{E(i,j-1)+e, V(i,j-1)+o+e\}$

	0	1 C	2 A	3 G
	X	X	X	X
C	-2	-4		
A	-3	-1		
T	-4	-2		
T	-5	-3		
T	-6	-4		
G	-7			

$F(i,$



Exemple

$V(i,j) =$

$Max\{E(i,j), F(i,j), V(i-1,j-1)+c\}$

		0	1 C	2 A	3 G
0		0	-2	-3	-4
1	C	-2	1		
2	A	-3	-1		
3	T	-4	-2		
4	T	-5	-3		
5	T	-6	-4		
6	G	-7			

$E(i,j) =$

$Max\{E(i,j-1)+e, V(i,j-1)+o+e\}$

	0	1 C	2 A	3 G
	X	X	X	X
C	-2	-4		
A	-3	-1		
T	-4	-2		
T	-5	-3		
T	-6	-4		
G	-7			

$F(i,j) =$

$Max\{F(i-1,j)+e, V(i-1,j)+o+e\}$

	0	1 C	2 A	3 G
	X	-2	-3	-4
C	X	-4		
A	X	-5		
T	X	-6		
T	X	-7		
T	X	-8		
G	X			

$cost(a,a) = +1$ $cost(a,b) = -1$ $o = -1$ $e = -1$

Exemple

$V(i,j) =$

$Max\{E(i,j), F(i,j), V(i-1,j-1)+c\}$

		0	1 C	2 A	3 G
0		0	-2	-3	-4
1	C	-2	1		
2	A	-3	-1		
3	T	-4	-2		
4	T	-5	-3		
5	T	-6	-4		
6	G	-7	-5		

$E(i,j) =$

$Max\{E(i,j-1)+e, V(i,j-1)+o+e\}$

	0	1 C	2 A	3 G
	X	X	X	X
C	-2	-4		
A	-3	-1		
T	-4	-2		
T	-5	-3		
T	-6	-4		
G	-7	-5		

$F(i,j) =$

$Max\{F(i-1,j)+e, V(i-1,j)+o+e\}$

	0	1 C	2 A	3 G
	X	-2	-3	-4
C	X	-4		
A	X	-5		
T	X	-6		
T	X	-7		
T	X	-8		
G	X	-9		

$cost(a,a) = +1$ $cost(a,b) = -1$ $o = -1$ $e = -1$

Exemple

$V(i,j) =$

$Max\{E(i,j), F(i,j), V(i-1,j-1)+c\}$

		0	1 C	2 A	3 G
0		0	-2	-3	-4
1	C	-2	1	-1	
2	A	-3	-1		
3	T	-4	-2		
4	T	-5	-3		
5	T	-6	-4		
6	G	-7	-5		

$E(i,j) =$

$Max\{E(i,j-1)+e, V(i,j-1)+o+e\}$

	0	1 C	2 A	3 G
	X	X	X	X
C	-2	-4	-5	
A	-3	-1		
T	-4	-2		
T	-5	-3		
T	-6	-4		
G	-7	-5		

$F(i,j) =$

$Max\{F(i-1,j)+e, V(i-1,j)+o+e\}$

	0	1 C	2 A	3 G
	X	-2	-3	-4
C	X	-4	-1	
A	X	-5		
T	X	-6		
T	X	-7		
T	X	-8		
G	X	-9		

$cost(a,a) = +1$ $cost(a,b) = -1$ $o = -1$ $e = -1$

Exemple

$V(i,j) =$

$Max\{E(i,j), F(i,j), V(i-1,j-1)+c\}$

		0	1 C	2 A	3 G
0		0	-2	-3	-4
1	C	-2	1	-1	
2	A	-3	-1	2	
3	T	-4	-2		
4	T	-5	-3		
5	T	-6	-4		
6	G	-7	-5		

$E(i,j) =$

$Max\{E(i,j-1)+e, V(i,j-1)+o+e\}$

	0	1 C	2 A	3 G
	X	X	X	X
C	-2	-4	-5	
A	-3	-1	-3	
T	-4	-2		
T	-5	-3		
T	-6	-4		
G	-7	-5		

$F(i,j) =$

$Max\{F(i-1,j)+e, V(i-1,j)+o+e\}$

	0	1 C	2 A	3 G
	X	-2	-3	-4
C	X	-4	-1	
A	X	-5	-3	
T	X	-6		
T	X	-7		
T	X	-8		
G	X	-9		

$cost(a,a) = +1$ $cost(a,b) = -1$ $o = -1$ $e = -1$

Exemple

$V(i,j) =$

$Max\{E(i,j), F(i,j), V(i-1,j-1)+c\}$

		0	1 C	2 A	3 G
0		0	-2	-3	-4
1	C	-2	1	-1	
2	A	-3	-1	2	
3	T	-4	-2	0	
4	T	-5	-3		
5	T	-6	-4		
6	G	-7	-5		

$E(i,j) =$

$Max\{E(i,j-1)+e, V(i,j-1)+o+e\}$

	0	1 C	2 A	3 G
	X	X	X	X
C	-2	-4	-5	
A	-3	-1	-3	
T	-4	-2	0	
T	-5	-3		
T	-6	-4		
G	-7	-5		

$F(i,j) =$

$Max\{F(i-1,j)+e, V(i-1,j)+o+e\}$

	0	1 C	2 A	3 G
	X	-2	-3	-4
C	X	-4	-1	
A	X	-5	-3	
T	X	-6	-4	
T	X	-7		
T	X	-8		
G	X	-9		

$cost(a,a) = +1$ $cost(a,b) = -1$ $o = -1$ $e = -1$

Exemple

$V(i,j) =$

$Max\{E(i,j), F(i,j), V(i-1,j-1)+c\}$

		0	1 C	2 A	3 G
0		0	-2	-3	-4
1	C	-2	1	-1	
2	A	-3	-1	2	
3	T	-4	-2	0	
4	T	-5	-3	-1	
5	T	-6	-4		
6	G	-7	-5		

$E(i,j) =$

$Max\{E(i,j-1)+e, V(i,j-1)+o+e\}$

	0	1 C	2 A	3 G
	X	X	X	X
C	-2	-4	-5	
A	-3	-1	-3	
T	-4	-2	0	
T	-5	-3	-1	
T	-6	-4		
G	-7	-5		

$F(i,j) =$

$Max\{F(i-1,j)+e, V(i-1,j)+o+e\}$

	0	1 C	2 A	3 G
	X	-2	-3	-4
C	X	-4	-1	
A	X	-5	-3	
T	X	-6	-4	
T	X	-7	-5	
T	X	-8		
G	X	-9		

$cost(a,a) = +1$ $cost(a,b) = -1$ $o = -1$ $e = -1$

Exemple

$V(i,j) =$

$Max\{E(i,j), F(i,j), V(i-1,j-1)+c\}$

		0	1 C	2 A	3 G
0		0	-2	-3	-4
1	C	-2	1	-1	
2	A	-3	-1	2	
3	T	-4	-2	0	
4	T	-5	-3	-1	
5	T	-6	-4	-2	
6	G	-7	-5		

$E(i,j) =$

$Max\{E(i,j-1)+e, V(i,j-1)+o+e\}$

	0	1 C	2 A	3 G
	X	X	X	X
C	-2	-4	-5	
A	-3	-1	-3	
T	-4	-2	0	
T	-5	-3	-1	
T	-6	-4	-2	
G	-7	-5		

$F(i,j) =$

$Max\{F(i-1,j)+e, V(i-1,j)+o+e\}$

	0	1 C	2 A	3 G
	X	-2	-3	-4
C	X	-4	-1	
A	X	-5	-3	
T	X	-6	-4	
T	X	-7	-5	
T	X	-8	-6	
G	X	-9		

$cost(a,a) = +1$ $cost(a,b) = -1$ $o = -1$ $e = -1$

Exemple

$V(i,j) =$

$Max\{E(i,j), F(i,j), V(i-1,j-1)+c\}$

		0	1 C	2 A	3 G
0		0	-2	-3	-4
1	C	-2	1	-1	
2	A	-3	-1	2	
3	T	-4	-2	0	
4	T	-5	-3	-1	
5	T	-6	-4	-2	
6	G	-7	-5	-3	

$E(i,j) =$

$Max\{E(i,j-1)+e, V(i,j-1)+o+e\}$

	0	1 C	2 A	3 G
	X	X	X	X
C	-2	-4	-5	
A	-3	-1	-3	
T	-4	-2	0	
T	-5	-3	-1	
T	-6	-4	-2	
G	-7	-5	-3	

$F(i,j) =$

$Max\{F(i-1,j)+e, V(i-1,j)+o+e\}$

	0	1 C	2 A	3 G
	X	-2	-3	-4
C	X	-4	-1	
A	X	-5	-3	
T	X	-6	-4	
T	X	-7	-5	
T	X	-8	-6	
G	X	-9	-7	

$cost(a,a) = +1$ $cost(a,b) = -1$ $o = -1$ $e = -1$

Exemple

$V(i,j) =$

$Max\{E(i,j), F(i,j), V(i-1,j-1)+c\}$

		0	1 C	2 A	3 G
0		0	-2	-3	-4
1	C	-2	1	-1	-2
2	A	-3	-1	2	0
3	T	-4	-2	0	1
4	T	-5	-3	-1	-1
5	T	-6	-4	-2	-2
6	G	-7	-5	-3	-1

$E(i,j) =$

$Max\{E(i,j-1)+e, V(i,j-1)+o+e\}$

	0	1 C	2 A	3 G
	X	X	X	X
C	-2	-4	-5	-6
A	-3	-1	-3	-4
T	-4	-2	0	-2
T	-5	-3	-1	-1
T	-6	-4	-2	-2
G	-7	-5	-3	-3

$F(i,j) =$

$Max\{F(i-1,j)+e, V(i-1,j)+o+e\}$

	0	1 C	2 A	3 G
	X	-2	-3	-4
C	X	-4	-1	-2
A	X	-5	-3	0
T	X	-6	-4	-2
T	X	-7	-5	-3
T	X	-8	-6	-4
G	X	-9	-7	-5

$cost(a,a) = +1$ $cost(a,b) = -1$ $o = -1$ $e = -1$

Exemple

$V(i,j) =$

$Max\{E(i,j), F(i,j), V(i-1,j-1)+c\}$

		0	1 C	2 A	3 G
0		0	-2	-3	-4
1	C	-2	1	-1	-2
2	A	-3	-1	2	0
3	T	-4	-2	0	1
4	T	-5	-3	-1	-1
5	T	-6	-4	-2	-2
6	G	-7	-5	-3	-1

$E(i,j) =$

$Max\{E(i,j-1)+e, V(i,j-1)+o+e\}$

	0	1 C	2 A	3 G
	X	X	X	X
C	-2	-4	-5	-6
A	-3	-1	-3	-4
T	-4	-2	0	-2
T	-5	-3	-1	-1
T	-6	-4	-2	-2
G	-7	-5	-3	-3

$F(i,j) =$

$Max\{F(i-1,j)+e, V(i-1,j)+o+e\}$

	0	1 C	2 A	3 G
	X	-2	-3	-4
C	X	-4	-1	-2
A	X	-5	-3	0
T	X	-6	-4	-2
T	X	-7	-5	-3
T	X	-8	-6	-4
G	X	-9	-7	-5

$cost(a,a) = +1$ $cost(a,b) = -1$ $o = -1$ $e = -1$

Exemple

$V(i,j) =$

$Max\{E(i,j), F(i,j), V(i-1,j-1)+c\}$

		0	1 C	2 A	3 G
0		0	-2	-3	-4
1	C	-2	1	-1	-2
2	A	-3	-1	2	0
3	T	-4	-2	0	1
4	T	-5	-3	-1	-1
5	T	-6	-4	-2	-2
6	G	-7	-5	-3	-1

$E(i,j) =$

$Max\{E(i,j-1)+e, V(i,j-1)+o+e\}$

	0	1 C	2 A	3 G
	X	X	X	X
C	-2	-4	-5	-6
A	-3	-1	-3	-4
T	-4	-2	0	-2
T	-5	-3	-1	-1
T	-6	-4	-2	-2
G	-7	-5	-3	-3

$F(i,j) =$

$Max\{F(i-1,j)+e, V(i-1,j)+o+e\}$

	0	1 C	2 A	3 G
	X	-2	-3	-4
C	X	-4	-1	-2
A	X	-5	-3	0
T	X	-6	-4	-2
T	X	-7	-5	-3
T	X	-8	-6	-4
G	X	-9	-7	-5

$cost(a,a) = +1$ $cost(a,b) = -1$ $o = -1$ $e = -1$

Exemple

$V(i,j) =$

$Max\{E(i,j), F(i,j), V(i-1,j-1)+c\}$

		0	1 C	2 A	3 G
0		0	-2	-3	-4
1	C	-2	1	-1	-2
2	A	-3	-1	2	0
3	T	-4	-2	0	1
4	T	-5	-3	-1	-1
5	T	-6	-4	-2	-2
6	G	-7	-5	-3	-1

$E(i,j) =$

$Max\{E(i,j-1)+e, V(i,j-1)+o+e\}$

	0	1 C	2 A	3 G
	X	X	X	X
C	-2	-4	-5	-6
A	-3	-1	-3	-4
T	-4	-2	0	-2
T	-5	-3	-1	-1
T	-6	-4	-2	-2
G	-7	-5	-3	-3

$F(i,j) =$

$Max\{F(i-1,j)+e, V(i-1,j)+o+e\}$

	0	1 C	2 A	3 G
	X	-2	-3	-4
C	X	-4	-1	-2
A	X	-5	-3	0
T	X	-6	-4	-2
T	X	-7	-5	-3
T	X	-8	-6	-4
G	X	-9	-7	-5

$cost(a,a) = +1$ $cost(a,b) = -1$ $o = -1$ $e = -1$

Exemple

$V(i,j) =$

$Max\{E(i,j), F(i,j), V(i-1,j-1)+c\}$

		0	1 C	2 A	3 G
0		0	-2	-3	-4
1	C	-2	1	-1	-2
2	A	-3	-1	2	0
3	T	-4	-2	0	1
4	T	-5	-3	-1	-1
5	T	-6	-4	-2	-2
6	G	-7	-5	-3	-1

$E(i,j) =$

$Max\{E(i,j-1)+e, V(i,j-1)+o+e\}$

	0	1 C	2 A	3 G
	X	X	X	X
C	-2	-4	-5	-6
A	-3	-1	-3	-4
T	-4	-2	0	-2
T	-5	-3	-1	-1
T	-6	-4	-2	-2
G	-7	-5	-3	-3

$F(i,j) =$

$Max\{F(i-1,j)+e, V(i-1,j)+o+e\}$

	0	1 C	2 A	3 G
	X	-2	-3	-4
C	X	-4	-1	-2
A	X	-5	-3	0
T	X	-6	-4	-2
T	X	-7	-5	-3
T	X	-8	-6	-4
G	X	-9	-7	-5

$cost(a,a) = +1$ $cost(a,b) = -1$ $o = -1$ $e = -1$

Exemple

$V(i,j) =$

$Max\{E(i,j), F(i,j), V(i-1,j-1)+c\}$

		0	1 C	2 A	3 G
0		0	-2	-3	-4
1	C	-2	1	-1	-2
2	A	-3	-1	2	0
3	T	-4	-2	0	1
4	T	-5	-3	-1	-1
5	T	-6	-4	-2	-2
6	G	-7	-5	-3	-1

$E(i,j) =$

$Max\{E(i,j-1)+e, V(i,j-1)+o+e\}$

	0	1 C	2 A	3 G
	X	X	X	X
C	-2	-4	-5	-6
A	-3	-1	-3	-4
T	-4	-2	0	-2
T	-5	-3	-1	-1
T	-6	-4	-2	-2
G	-7	-5	-3	-3

$F(i,j) =$

$Max\{F(i-1,j)+e, V(i-1,j)+o+e\}$

	0	1 C	2 A	3 G
	X	-2	-3	-4
C	X	-4	-1	-2
A	X	-5	-3	0
T	X	-6	-4	-2
T	X	-7	-5	-3
T	X	-8	-6	-4
G	X	-9	-7	-5

$cost(a,a) = +1$ $cost(a,b) = -1$ $o = -1$ $e = -1$

Exemple

$V(i,j) =$

$Max\{E(i,j), F(i,j), V(i-1,j-1)+c\}$

		0	1 C	2 A	3 G
0		0	-2	-3	-4
1	C	-2	1	-1	-2
2	A	-3	-1	2	0
3	T	-4	-2	0	1
4	T	-5	-3	-1	-1
5	T	-6	-4	-2	-2
6	G	-7	-5	-3	-1

$E(i,j) =$

$Max\{E(i,j-1)+e, V(i,j-1)+o+e\}$

	0	1 C	2 A	3 G
	X	X	X	X
C	-2	-4	-5	-6
A	-3	-1	-3	-4
T	-4	-2	0	-2
T	-5	-3	-1	-1
T	-6	-4	-2	-2
G	-7	-5	-3	-3

$F(i,j) =$

$Max\{F(i-1,j)+e, V(i-1,j)+o+e\}$

	0	1 C	2 A	3 G
	X	-2	-3	-4
C	X	-4	-1	-2
A	X	-5	-3	0
T	X	-6	-4	-2
T	X	-7	-5	-3
T	X	-8	-6	-4
G	X	-9	-7	-5

$cost(a,a) = +1$ $cost(a,b) = -1$ $o = -1$ $e = -1$

Exemple

$V(i,j) =$

$Max\{E(i,j), F(i,j), V(i-1,j-1)+c\}$

		0	1 C	2 A	3 G
0		0	-2	-3	-4
1	C	-2	1	-1	-2
2	A	-3	-1	2	0
3	T	-4	-2	0	1
4	T	-5	-3	-1	-1
5	T	-6	-4	-2	-2
6	G	-7	-5	-3	-1

$E(i,j) =$

$Max\{E(i,j-1)+e, V(i,j-1)+o+e\}$

	0	1 C	2 A	3 G
	X	X	X	X
C	-2	-4	-5	-6
A	-3	-1	-3	-4
T	-4	-2	0	-2
T	-5	-3	-1	-1
T	-6	-4	-2	-2
G	-7	-5	-3	-3

$F(i,j) =$

$Max\{F(i-1,j)+e, V(i-1,j)+o+e\}$

	0	1 C	2 A	3 G
	X	-2	-3	-4
C	X	-4	-1	-2
A	X	-5	-3	0
T	X	-6	-4	-2
T	X	-7	-5	-3
T	X	-8	-6	-4
G	X	-9	-7	-5

$cost(a,a) = +1$ $cost(a,b) = -1$ $o = -1$ $e = -1$

Exemple

$V(i,j) =$

$Max\{E(i,j), F(i,j), V(i-1,j-1)+c\}$

		0	1 C	2 A	3 G
0		0	-2	-3	-4
1	C	-2	1	-1	-2
2	A	-3	-1	2	0
3	T	-4	-2	0	1
4	T	-5	-3	-1	-1
5	T	-6	-4	-2	-2
6	G	-7	-5	-3	-1

$E(i,j) =$

$Max\{E(i,j-1)+e, V(i,j-1)+o+e\}$

	0	1 C	2 A	3 G
	X	X	X	X
C	-2	-4	-5	-6
A	-3	-1	-3	-4
T	-4	-2	0	-2
T	-5	-3	-1	-1
T	-6	-4	-2	-2
G	-7	-5	-3	-3

$F(i,j) =$

$Max\{F(i-1,j)+e, V(i-1,j)+o+e\}$

	0	1 C	2 A	3 G
	X	-2	-3	-4
C	X	-4	-1	-2
A	X	-5	-3	0
T	X	-6	-4	-2
T	X	-7	-5	-3
T	X	-8	-6	-4
G	X	-9	-7	-5

$cost(a,a) = +1$ $cost(a,b) = -1$ $o = -1$ $e = -1$

Exercice

Exercice

Calculer l'alignement global de *TCCTATTC* avec *TCAGGTAC*, avec $cost(a, a) = 1$,$cost(a, b) = -1$, ouverture de -1 et extension de -1.

A propos des matrices

Matrice Unitaire:

	A	C	G	T	-
A	1	0	0	0	0
C	0	1	0	0	0
G	0	0	1	0	0
T	0	0	0	1	0
-	0	0	0	0	

Transition/Transversion:

	A	C	G	T	-
A	-	β	α	β	-
C	β	-	β	α	-
G	α	β	-	β	-
T	β	α	β	-	-
-	-	-	-	-	

Transitions:

- Purines: A-G
- Pyrimidines: C-T

A propos des matrices

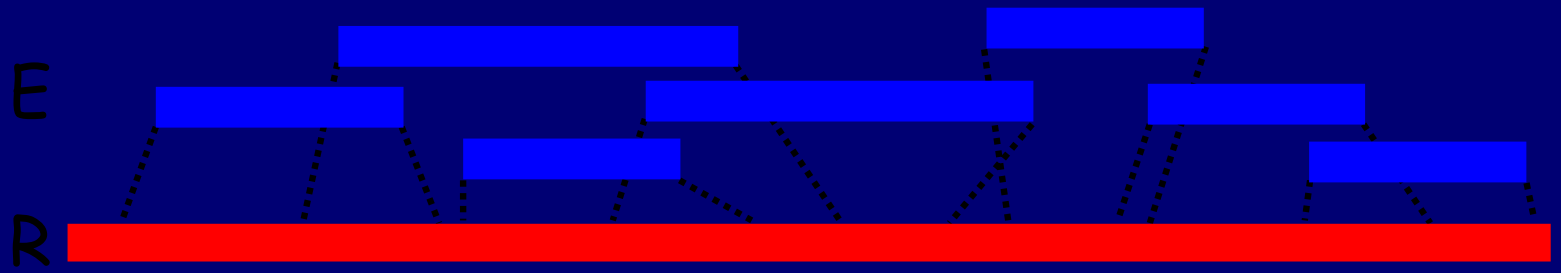
BLAST:

	A	C	G	T	-
A	1	-3	-3	-3	-5/-2
C	-3	1	-3	-3	-5/-2
G	-3	-3	1	-3	-5/-2
T	-3	-3	-3	1	-5/-2
-	-5/-2	-5/-2	-5/-2	-5/-2	

Chaînage à une dimension

Problème

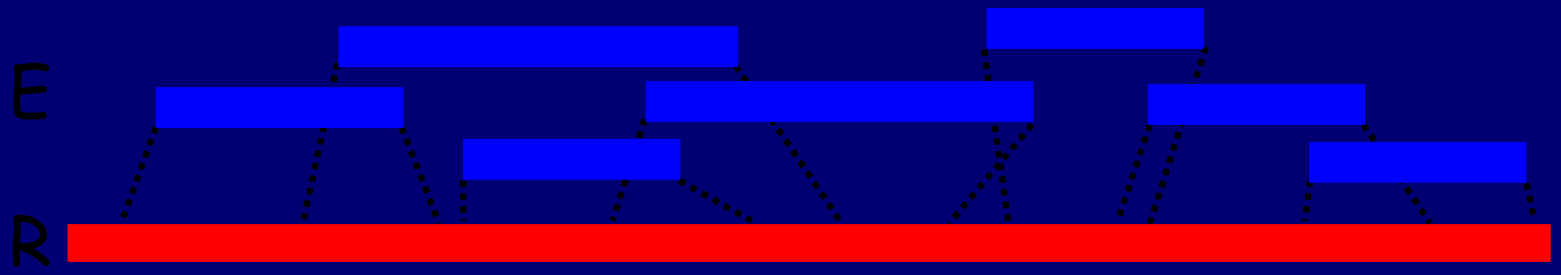
On dispose d'une séquence d'ADN R d'un gène eucaryote et d'un ensemble d'exons $E_1, E_2 \ldots$ candidats pour ce gène. Pour chaque exons, on calcul le facteur de R de plus forte similarité.



Chaînage à une dimension

Problème

On dispose d'une séquence d'ADN R d'un gène eucaryote et d'un ensemble d'exons $E_1, E_2 \ldots$ candidats pour ce gène. Pour chaque exons, on calcul le facteur de R de plus forte similarité.



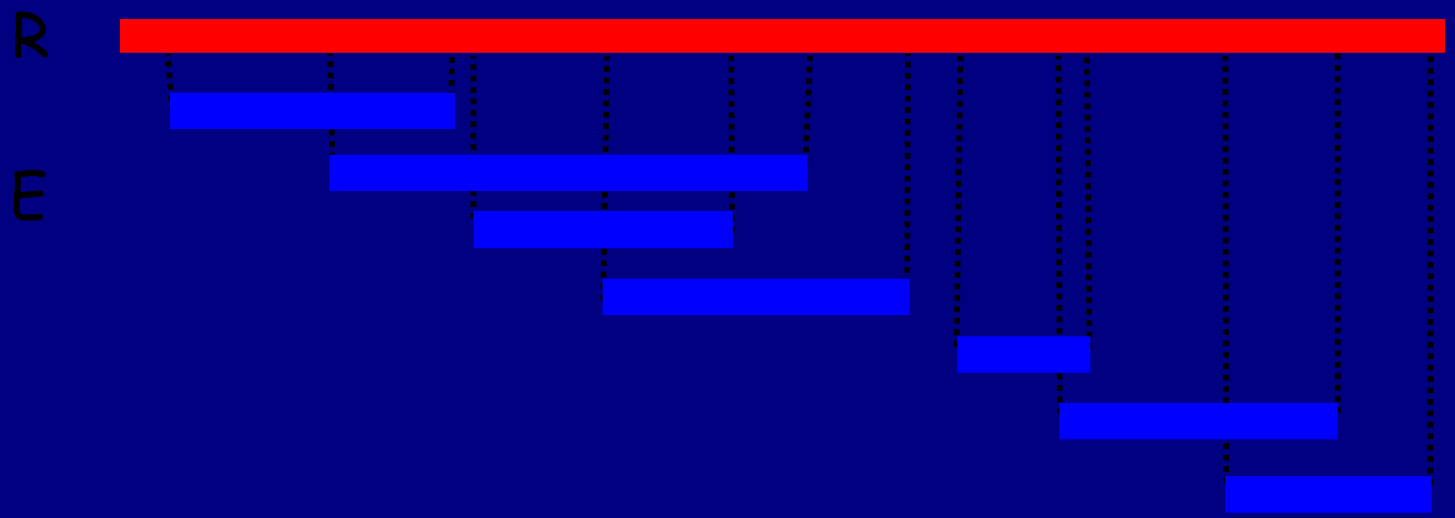
Problème: trouver la suite d'exons dont les occurences sur R ne se chevauchent pas et la somme des scores est maximale (recouvrement maximal).

Chaînage à une dimension

Problème

Chaînage: algorithme

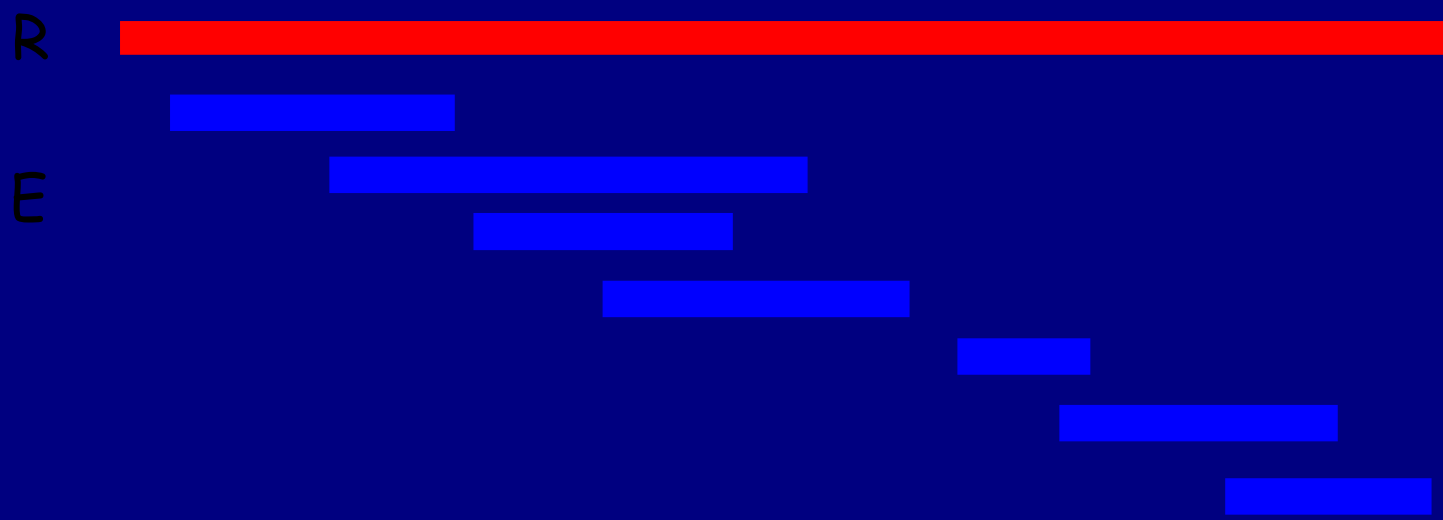
On pourrait tout a fait calculer le chaînage en utilisant un graphe orienté sans cylcle tel que chaque exons soit représenté par un sommet et deux sommets sont reliés s'ils sont compatibles:



Recherche du plus long chemin de S à T : $O(n^2)$

Chaînage: algorithme

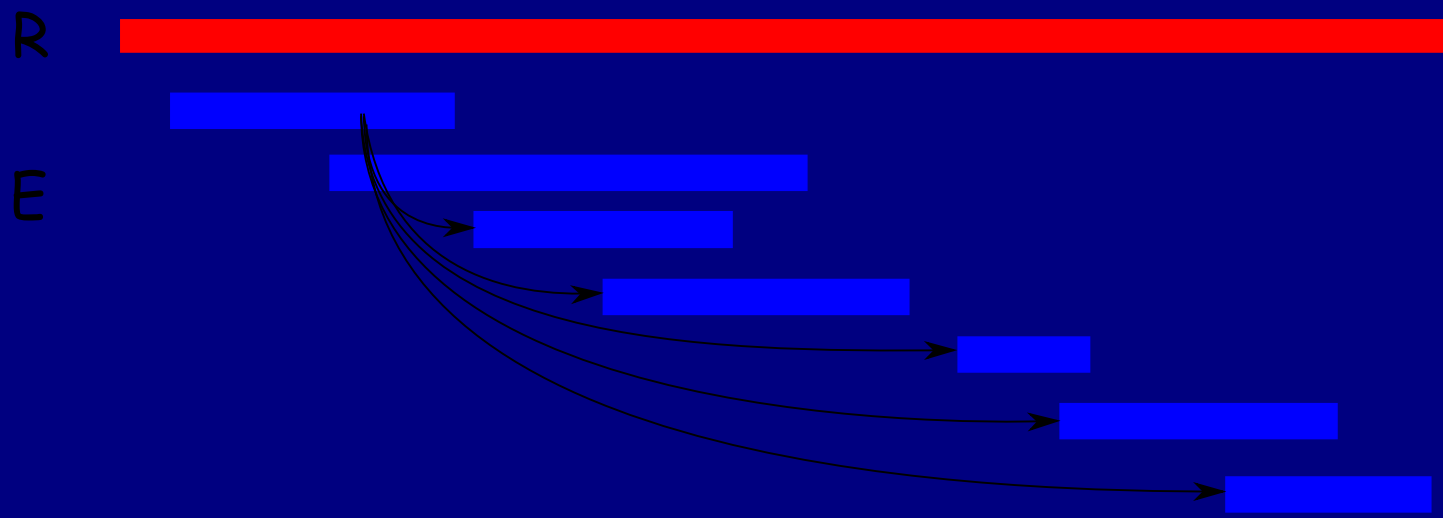
On pourrait tout a fait calculer le chaînage en utilisant un graphe orienté sans cylcle tel que chaque exons soit représenté par un sommet et deux sommets sont reliés s'ils sont compatibles:



Recherche du plus long chemin de S à T : $O(n^2)$

Chaînage: algorithme

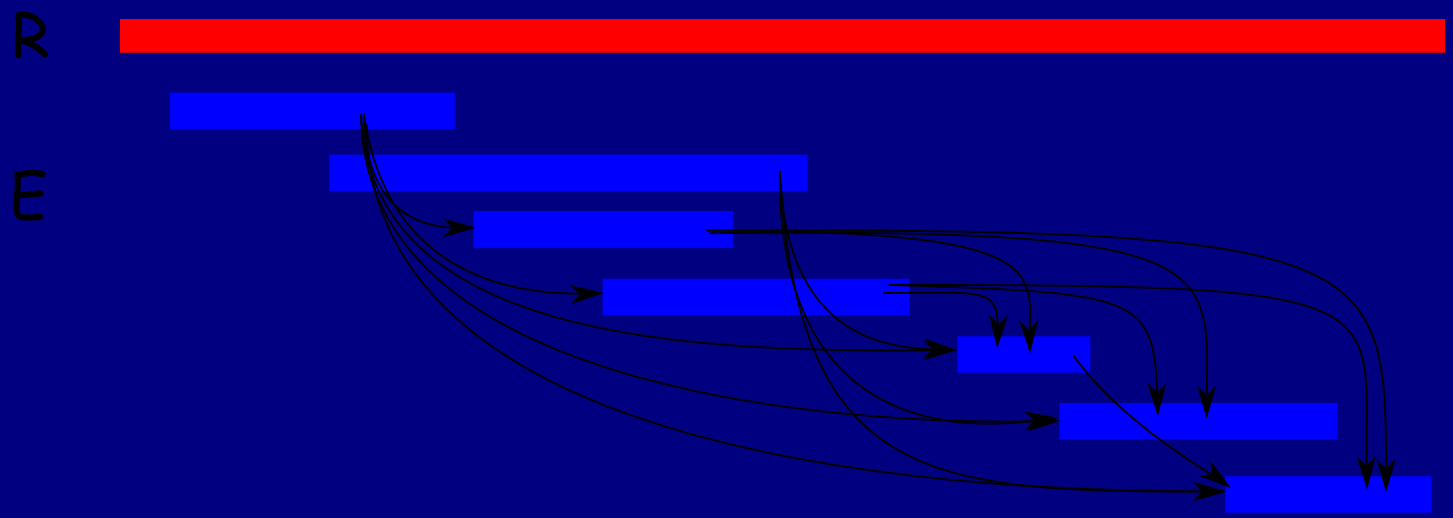
On pourrait tout a fait calculer le chaînage en utilisant un graphe orienté sans cylcle tel que chaque exons soit représenté par un sommet et deux sommets sont reliés s'ils sont compatibles:



Recherche du plus long chemin de S à T: $O(n^2)$

Chaînage: algorithme

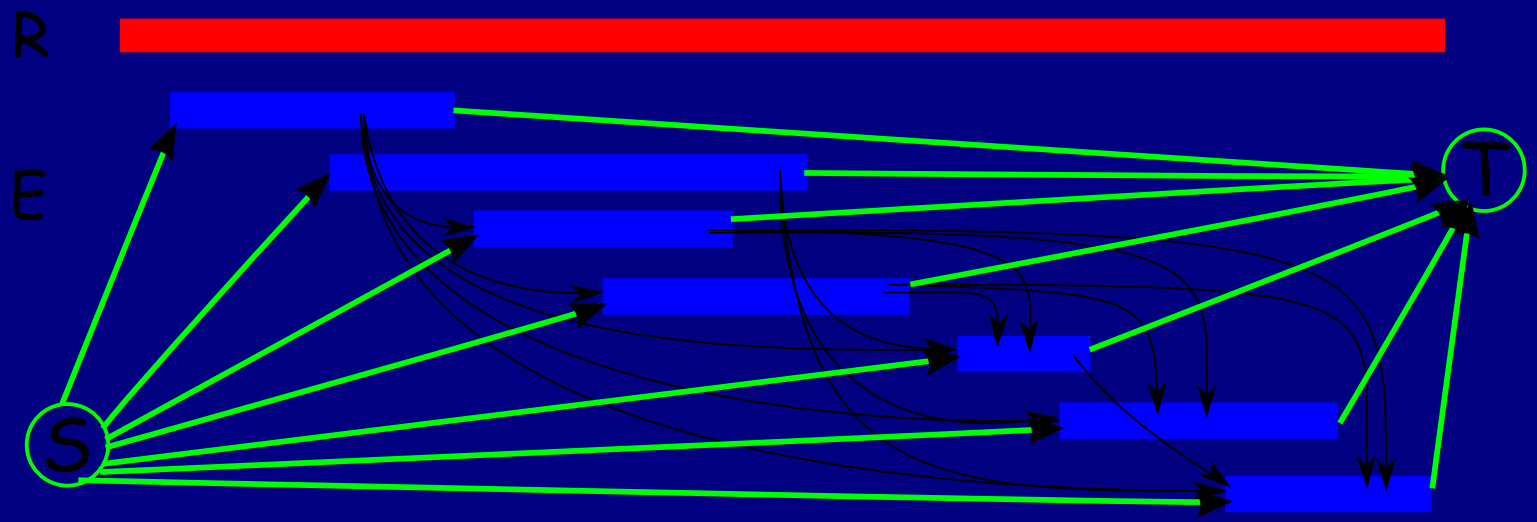
On pourrait tout a fait calculer le chaînage en utilisant un graphe orienté sans cylcle tel que chaque exons soit représenté par un sommet et deux sommets sont reliés s'ils sont compatibles:



Recherche du plus long chemin de S à T: $O(n^2)$

Chaînage: algorithme

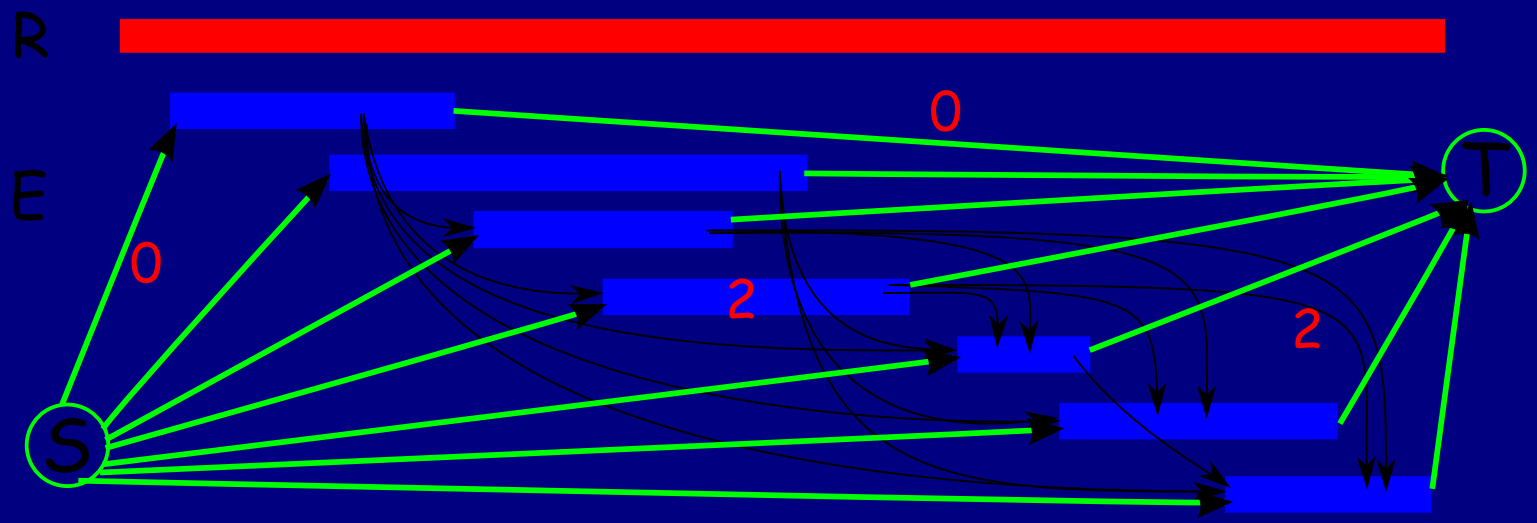
On pourrait tout a fait calculer le chaînage en utilisant un graphe orienté sans cylcle tel que chaque exons soit représenté par un sommet et deux sommets sont reliés s'ils sont compatibles:



Les arcs sont valués par le score du sommet d'origine Recherche du plus long chemin de S à T : $O(n^2)$

Chaînage: algorithme

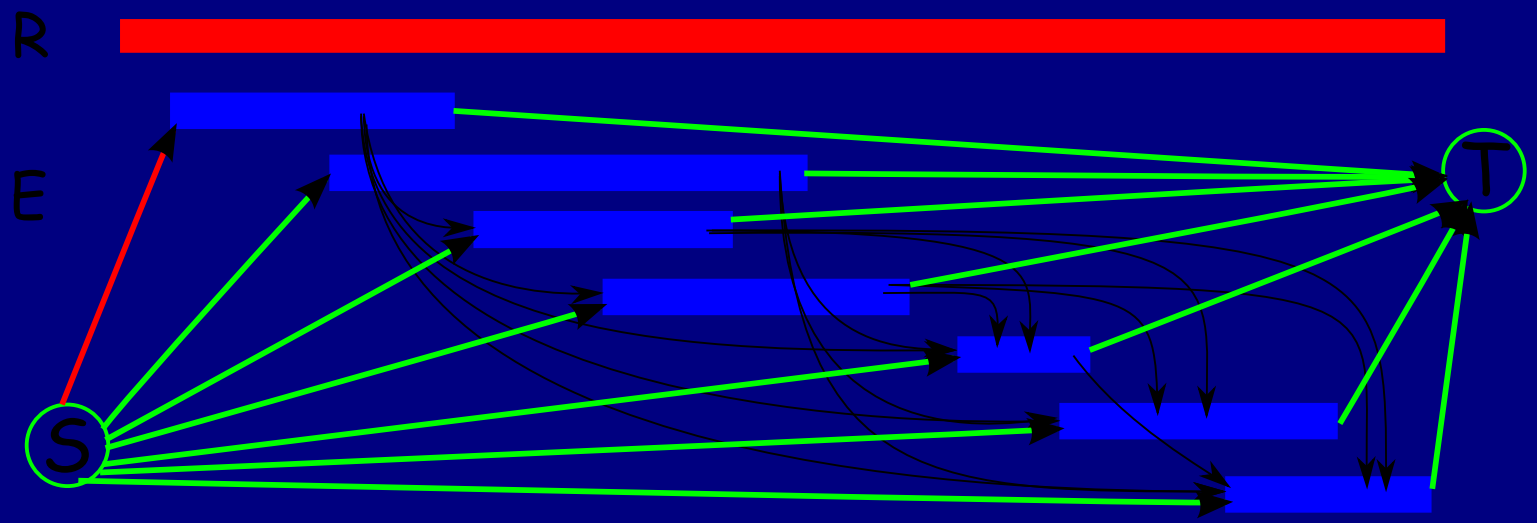
On pourrait tout a fait calculer le chaînage en utilisant un graphe orienté sans cylcle tel que chaque exons soit représenté par un sommet et deux sommets sont reliés s'ils sont compatibles:



Recherche du plus long chemin de S à T: $O(n^2)$

Chaînage: algorithme

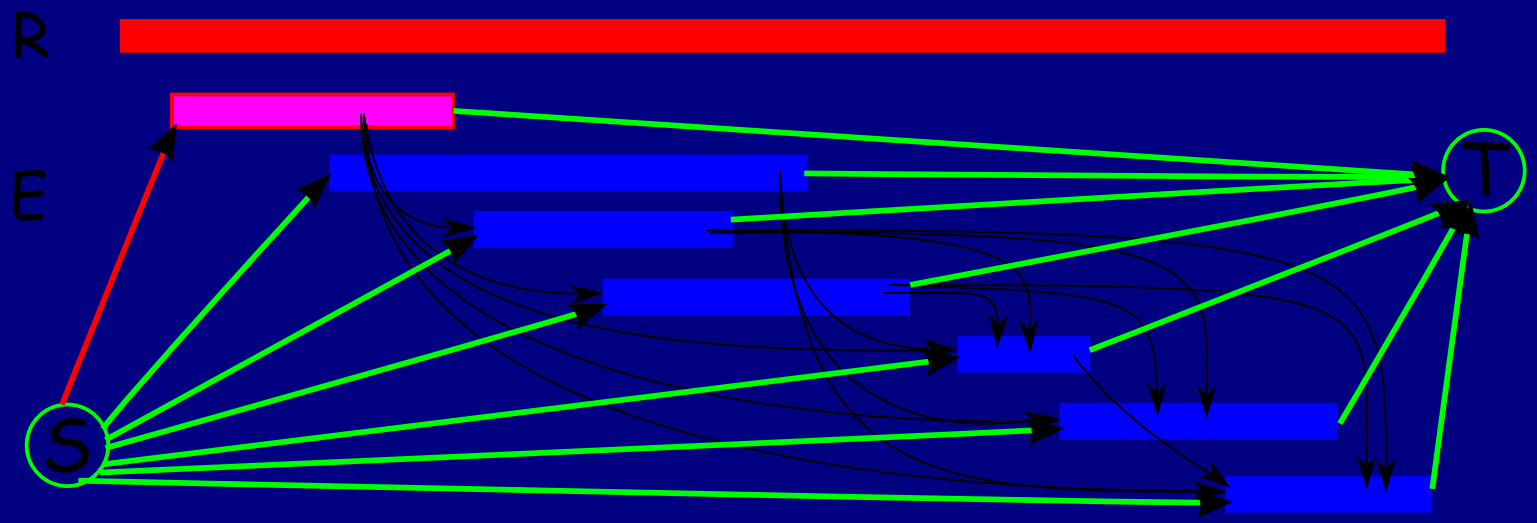
On pourrait tout a fait calculer le chaînage en utilisant un graphe orienté sans cylcle tel que chaque exons soit représenté par un sommet et deux sommets sont reliés s'ils sont compatibles:



Recherche du plus long chemin de S à T: $O(n^2)$

Chaînage: algorithme

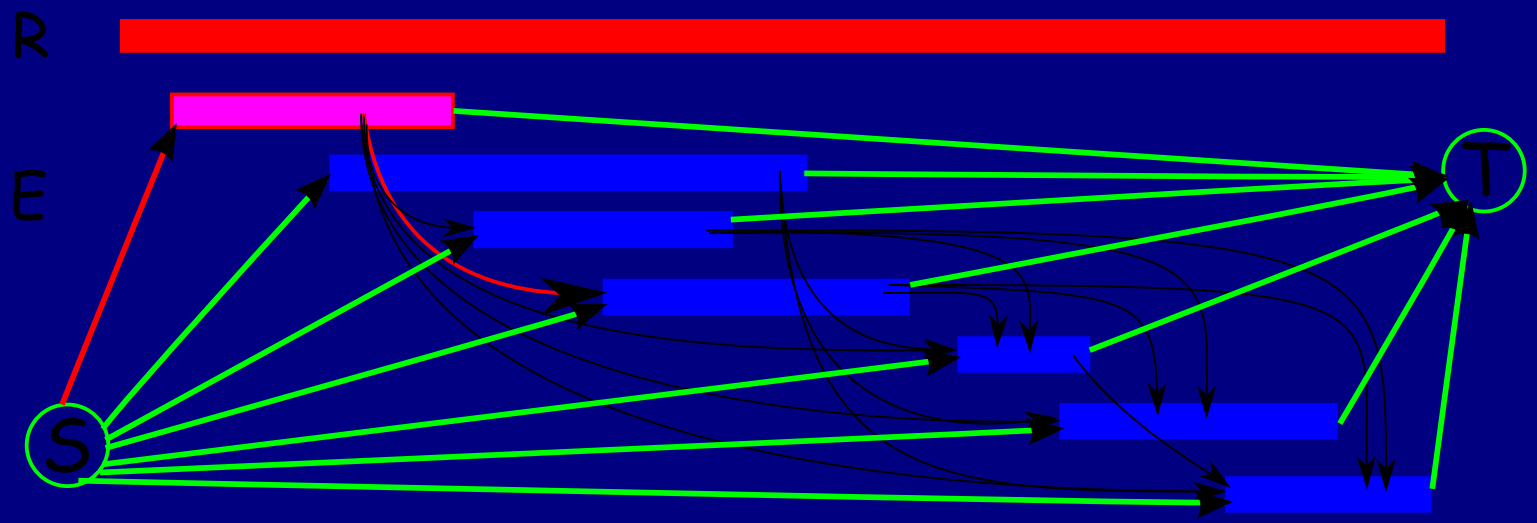
On pourrait tout a fait calculer le chaînage en utilisant un graphe orienté sans cylcle tel que chaque exons soit représenté par un sommet et deux sommets sont reliés s'ils sont compatibles:



Recherche du plus long chemin de S à T: $O(n^2)$

Chaînage: algorithme

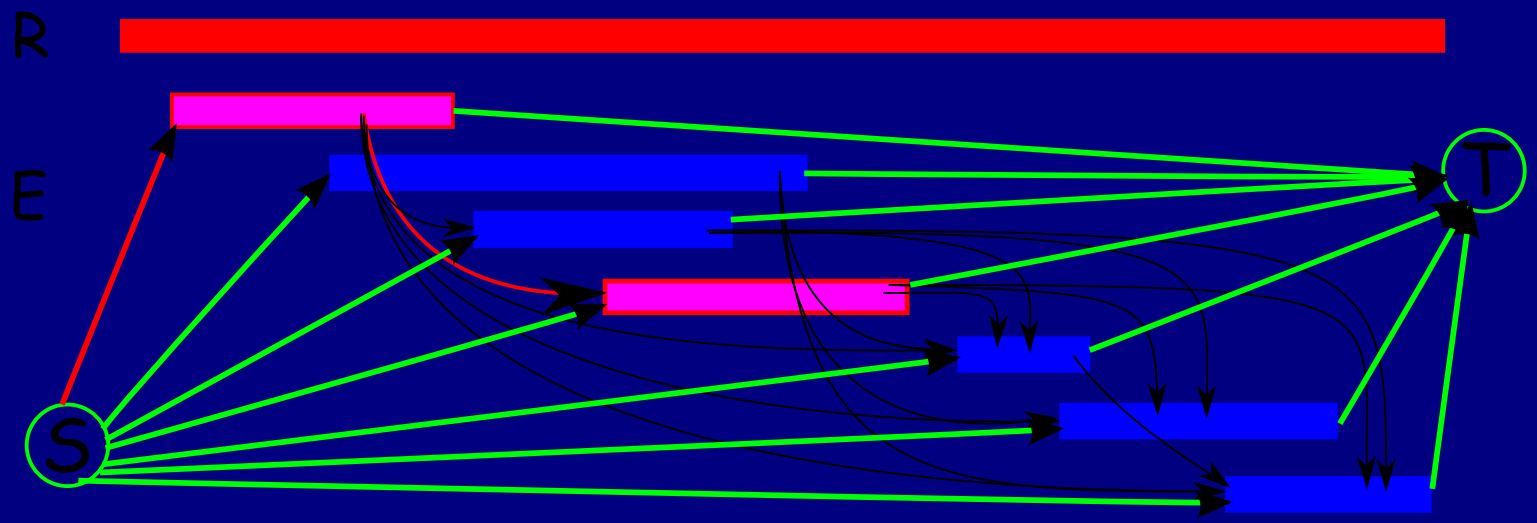
On pourrait tout a fait calculer le chaînage en utilisant un graphe orienté sans cylcle tel que chaque exons soit représenté par un sommet et deux sommets sont reliés s'ils sont compatibles:



Recherche du plus long chemin de S à T: $O(n^2)$

Chaînage: algorithme

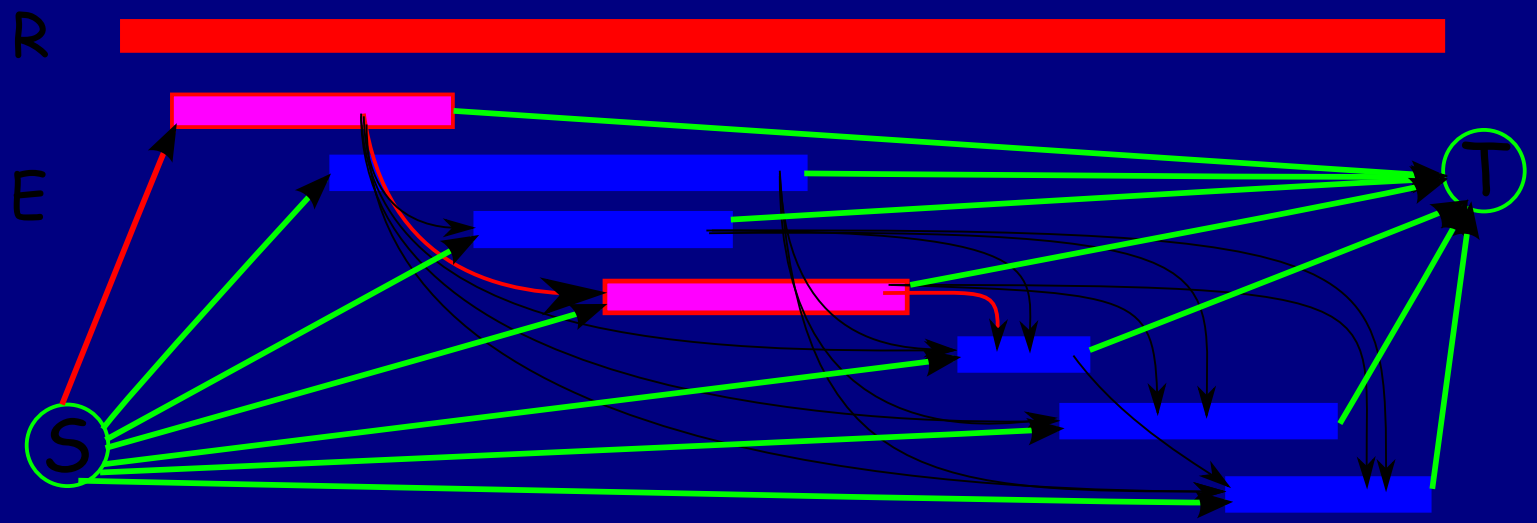
On pourrait tout a fait calculer le chaînage en utilisant un graphe orienté sans cylcle tel que chaque exons soit représenté par un sommet et deux sommets sont reliés s'ils sont compatibles:



Recherche du plus long chemin de S à T: $O(n^2)$

Chaînage: algorithme

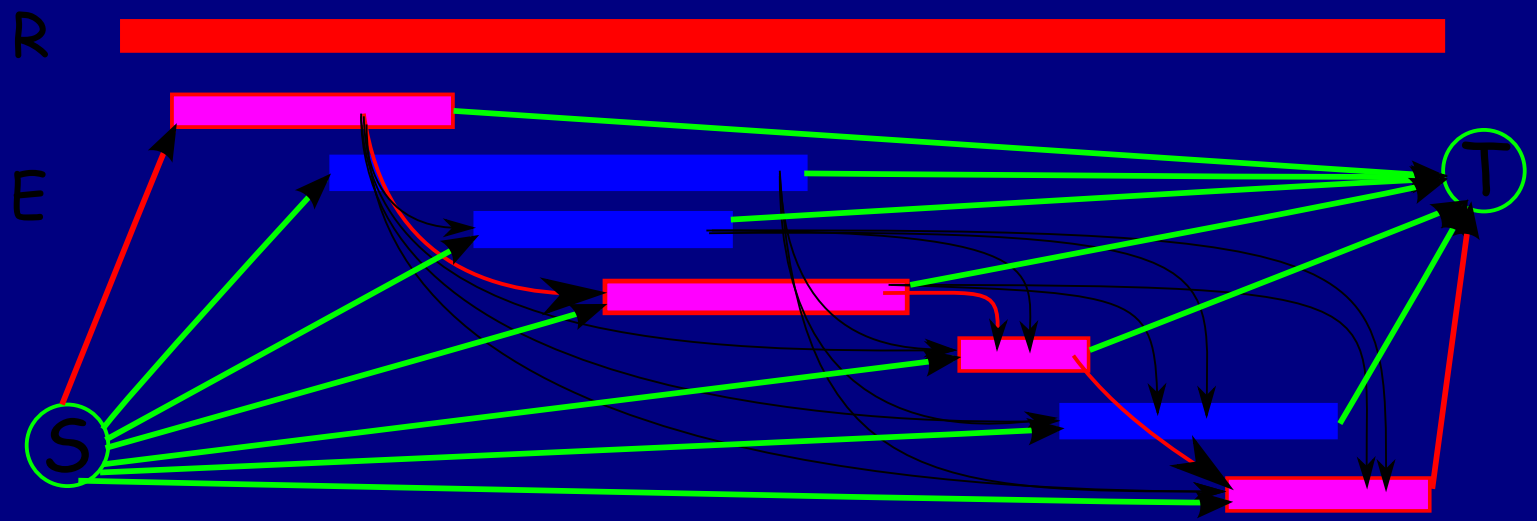
On pourrait tout a fait calculer le chaînage en utilisant un graphe orienté sans cylcle tel que chaque exons soit représenté par un sommet et deux sommets sont reliés s'ils sont compatibles:



Recherche du plus long chemin de S à T: $O(n^2)$

Chaînage: algorithme

On pourrait tout a fait calculer le chaînage en utilisant un graphe orienté sans cylcle tel que chaque exons soit représenté par un sommet et deux sommets sont reliés s'ils sont compatibles:



Recherche du plus long chemin de S à T: $O(n^2)$

Chaînage 1D: algorithme

Une autre solution est d'utiliser un algorithme de type "programmation dynamique".

On va chercher à calculer pour chaque exons le meilleur chaînage se terminant par cet exon.

On parcours toutes la séquence de gauche à droite:

Chaînage 1D: algorithme

Une autre solution est d'utiliser un algorithme de type "programmation dynamique".

On va chercher à calculer pour chaque exons le meilleur chaînage se terminant par cet exon.

On parcours toutes la séquence de gauche à droite:

- À chaque fois que l'on rencontre le début d'un exon, on le chaîne avec le meilleur chaînage déjà calculé.
- Si l'on atteint le fin d'un exons alors on regarde si son chaînage est le meilleur chaînage.

Chaînage 1D: algorithme

Une autre solution est d'utiliser un algorithme de type "programmation dynamique".

On va chercher à calculer pour chaque exons le meilleur chaînage se terminant par cet exon.

On parcours toutes la séquence de gauche à droite:

- À chaque fois que l'on rencontre le début d'un exon, on le chaîne avec le meilleur chaînage déjà calculé.
- Si l'on atteint le fin d'un exons alors on regarde si son chaînage est le meilleur chaînage.

Chaînage 1D: algorithme

Une autre solution est d'utiliser un algorithme de type "programmation dynamique".

On va chercher à calculer pour chaque exons le meilleur chaînage se terminant par cet exon.

On parcours toutes la séquence de gauche à droite:

- À chaque fois que l'on rencontre le début d'un exon, on le chaîne avec le meilleur chaînage déjà calculé.
- Si l'on atteint le fin d'un exons alors on regarde si son chaînage est le meilleur chaînage.

Chaînage 1D: algorithme

Une autre solution est d'utiliser un algorithme de type "programmation dynamique".

On va chercher à calculer pour chaque exons le meilleur chaînage se terminant par cet exon.

On parcours toutes la séquence de gauche à droite:

- À chaque fois que l'on rencontre le début d'un exon, on le chaîne avec le meilleur chaînage déjà calculé.
- Si l'on atteint le fin d'un exons alors on regarde si son chaînage est le meilleur chaînage.

Chaînage 1D: algorithme

E_1 (2) :									*	*	*
E_2 (4) :	*	*	*	*	*						
E_3 (1) :		*	*								
E_4 (3) :				*	*	*	*	*			
E_5 (3) :			*	*	*	*					
E_6 (2) :						*	*	*			
R :	0	1	2	3	4	5	6	7	8	9	10

Deux tableaux: I (positions) et V (meilleur score par exons)

position:	8	10	0	4	1	2	3	7	2	5	5	7
exon:	1_ℓ	1_r	2_ℓ	2_r	3_ℓ	3_r	4_ℓ	4_r	5_ℓ	5_r	6_ℓ	6_r

exons:	1	2	3	4	5	6
maximum:	0	0	0	0	0	0

$max = 0$

Chaînage 1D: algorithme

E_1 (2) :									*	*	*
E_2 (4) :	*	*	*	*	*						
E_3 (1) :		*	*								
E_4 (3) :				*	*	*	*	*			
E_5 (3) :			*	*	*	*					
E_6 (2) :						*	*	*			
R :	0	1	2	3	4	5	6	7	8	9	10

Trier I

position:	0	1	2	2	3	4	5	5	7	7	8	10
exon:	2_ℓ	3_ℓ	3_r	5_ℓ	4_ℓ	2_r	5_r	6_ℓ	6_r	4_r	1_ℓ	1_r

exons:	1	2	3	4	5	6
maximum:	0	0	0	0	0	0

$max = 0$

Chaînage 1D: algorithme

E_1 (2) :									*	*	*
E_2 (4) :	*	*	*	*	*						
E_3 (1) :		*	*								
E_4 (3) :				*	*	*	*	*			
E_5 (3) :			*	*	*	*					
E_6 (2) :						*	*	*			
R :	0	1	2	3	4	5	6	7	8	9	10

Parcourir I, Si position gauche de E_j alors $V[j] = max + v_j$

position:	0	1	2	2	3	4	5	5	7	7	8	10
exon:	2_ℓ	3_ℓ	3_r	5_ℓ	4_ℓ	2_r	5_r	6_ℓ	6_r	4_r	1_ℓ	1_r

exons:	1	2	3	4	5	6
maximum:	0	4	0	0	0	0

$max = 0$

Chaînage 1D: algorithme

E_1 (2) :									*	*	*
E_2 (4) :	*	*	*	*	*						
E_3 (1) :		*	*								
E_4 (3) :				*	*	*	*	*			
E_5 (3) :			*	*	*	*					
E_6 (2) :						*	*	*			
R :	0	1	2	3	4	5	6	7	8	9	10

Parcourir I, Si position gauche de E_j alors $V[j] = max + v_j$

position:	0	1	2	2	3	4	5	5	7	7	8	10
exon:	2_ℓ	3_ℓ	3_r	5_ℓ	4_ℓ	2_r	5_r	6_ℓ	6_r	4_r	1_ℓ	1_r

exons:	1	2	3	4	5	6
maximum:	0	4	1	0	0	0

$max = 0$

Chaînage 1D: algorithme

E_1 (2) :									*	*	*
E_2 (4) :	*	*	*	*	*						
E_3 (1) :		*	*								
E_4 (3) :				*	*	*	*	*			
E_5 (3) :			*	*	*	*					
E_6 (2) :						*	*	*			
R :	0	1	2	3	4	5	6	7	8	9	10

Parcourir I, Si position droite de E_j alors max=max(max,$V[j]$)

position:	0	1	2	2	3	4	5	5	7	7	8	10
exon:	2_ℓ	3_ℓ	3_r	5_ℓ	4_ℓ	2_r	5_r	6_ℓ	6_r	4_r	1_ℓ	1_r

exons:	1	2	3	4	5	6
maximum:	0	4	1	0	0	0

$max = 1$

Chaînage 1D: algorithme

E_1 (2) :									*	*	*
E_2 (4) :	*	*	*	*	*						
E_3 (1) :		*	*								
E_4 (3) :				*	*	*	*	*			
E_5 (3) :			*	*	*	*					
E_6 (2) :						*	*	*			
R :	0	1	2	3	4	5	6	7	8	9	10

Parcourir I, Si position gauche de E_j alors $V[j] = max + v_j$

position:	0	1	2	2	3	4	5	5	7	7	8	10
exon:	2_ℓ	3_ℓ	3_r	5_ℓ	4_ℓ	2_r	5_r	6_ℓ	6_r	4_r	1_ℓ	1_r

exons:	1	2	3	4	5	6
maximum:	0	4	1	0	3+1=4	0

$max = 1$

Chaînage 1D: algorithme

	0	1	2	3	4	5	6	7	8	9	10
E_1 (2) :									*	*	*
E_2 (4) :	*	*	*	*	*						
E_3 (1) :		*	*								
E_4 (3) :				*	*	*	*	*			
E_5 (3) :			*	*	*	*					
E_6 (2) :						*	*	*			
R :	0	1	2	3	4	5	6	7	8	9	10

Parcourir I, Si position gauche de E_j alors $V[j] = max + v_j$

position:	0	1	2	2	3	4	5	5	7	7	8	10
exon:	2_ℓ	3_ℓ	3_r	5_ℓ	4_ℓ	2_r	5_r	6_ℓ	6_r	4_r	1_ℓ	1_r

exons:	1	2	3	4	5	6
maximum:	0	4	1	4	4	0

$max = 1$

Chaînage 1D: algorithme

E_1 (2) :									*	*	*
E_2 (4) :	*	*	*	*	*						
E_3 (1) :		*	*								
E_4 (3) :				*	*	*	*	*			
E_5 (3) :			*	*	*	*					
E_6 (2) :						*	*	*			
R :	0	1	2	3	4	5	6	7	8	9	10

Parcourir I, Si position droite de E_j alors max=max(max,$V[j]$)

position:	0	1	2	2	3	4	5	5	7	7	8	10
exon:	2_ℓ	3_ℓ	3_r	5_ℓ	4_ℓ	2_r	5_r	6_ℓ	6_r	4_r	1_ℓ	1_r

exons:	1	2	3	4	5	6
maximum:	0	4	1	4	4	0

$max = 4$

Chaînage 1D: algorithme

E_1 (2) :									*	*	*
E_2 (4) :	*	*	*	*	*						
E_3 (1) :		*	*								
E_4 (3) :				*	*	*	*	*			
E_5 (3) :			*	*	*	*					
E_6 (2) :						*	*	*			
R :	0	1	2	3	4	5	6	7	8	9	10

Parcourir I, Si position droite de E_j alors max=max(max,$V[j]$)

position:	0	1	2	2	3	4	5	5	7	7	8	10
exon:	2_ℓ	3_ℓ	3_r	5_ℓ	4_ℓ	2_r	5_r	6_ℓ	6_r	4_r	1_ℓ	1_r

exons:	1	2	3	4	5	6
maximum:	0	4	1	4	4	0

$max = 4$

Chaînage 1D: algorithme

E_1 (2) :									*	*	*
E_2 (4) :	*	*	*	*	*						
E_3 (1) :		*	*								
E_4 (3) :				*	*	*	*	*			
E_5 (3) :			*	*	*	*					
E_6 (2) :						*	*	*			
R :	0	1	2	3	4	5	6	7	8	9	10

Parcourir I, Si position gauche de E_j alors $V[j] = max + v_j$

position:	0	1	2	2	3	4	5	5	7	7	8	10
exon:	2_ℓ	3_ℓ	3_r	5_ℓ	4_ℓ	2_r	5_r	6_ℓ	6_r	4_r	1_ℓ	1_r

exons:	1	2	3	4	5	6
maximum:	0	4	1	4	4	6

$max = 4$

Chaînage 1D: algorithme

E_1 (2) :									*	*	*
E_2 (4) :	*	*	*	*	*						
E_3 (1) :		*	*								
E_4 (3) :				*	*	*	*	*			
E_5 (3) :			*	*	*	*					
E_6 (2) :						*	*	*			
R :	0	1	2	3	4	5	6	7	8	9	10

Parcourir I, Si position droite de E_j alors max=max(max,$V[j]$)

position:	0	1	2	2	3	4	5	5	7	7	8	10
exon:	2_ℓ	3_ℓ	3_r	5_ℓ	4_ℓ	2_r	5_r	6_ℓ	6_r	4_r	1_ℓ	1_r

exons:	1	2	3	4	5	6
maximum:	0	4	1	4	4	6

$max = 6$

Chaînage 1D: algorithme

E_1 (2) :									*	*	*
E_2 (4) :	*	*	*	*	*						
E_3 (1) :		*	*								
E_4 (3) :				*	*	*	*	*			
E_5 (3) :			*	*	*	*					
E_6 (2) :						*	*	*			
R :	0	1	2	3	4	5	6	7	8	9	10

Parcourir I, Si position droite de E_j alors max=max(max,$V[j]$)

position:	0	1	2	2	3	4	5	5	7	7	8	10
exon:	2_ℓ	3_ℓ	3_r	5_ℓ	4_ℓ	2_r	5_r	6_ℓ	6_r	4_r	1_ℓ	1_r

exons:	1	2	3	4	5	6
maximum:	0	4	1	4	4	6

$max = 6$

Chaînage 1D: algorithme

E_1 (2) :									*	*	*
E_2 (4) :	*	*	*	*	*						
E_3 (1) :		*	*								
E_4 (3) :				*	*	*	*	*			
E_5 (3) :			*	*	*	*					
E_6 (2) :						*	*	*			
R :	0	1	2	3	4	5	6	7	8	9	10

Parcourir I, Si position gauche de E_j alors $V[j] = max + v_j$

position:	**0**	**1**	**2**	**2**	**3**	**4**	**5**	**5**	**7**	**7**	**8**	**10**
exon:	2_ℓ	3_ℓ	3_r	5_ℓ	4_ℓ	2_r	5_r	6_ℓ	6_r	4_r	1_ℓ	1_r

exons:	**1**	**2**	**3**	**4**	**5**	**6**
maximum:	**8**	**4**	**1**	**4**	**4**	**6**

$max = 6$

Chaînage 1D: algorithme

E_1 (2) :									*	*	*
E_2 (4) :	*	*	*	*	*						
E_3 (1) :		*	*								
E_4 (3) :				*	*	*	*	*			
E_5 (3) :			*	*	*	*					
E_6 (2) :						*	*	*			
R :	0	1	2	3	4	5	6	7	8	9	10

Parcourir I, Si position droite de E_j alors max=max(max,$V[j]$)

position:	0	1	2	2	3	4	5	5	7	7	8	10
exon:	2_ℓ	3_ℓ	3_r	5_ℓ	4_ℓ	2_r	5_r	6_ℓ	6_r	4_r	1_ℓ	1_r

exons:	1	2	3	4	5	6
maximum:	8	4	1	4	4	6

$max = 8$

Chaînage 1D: algorithme

E_1 (2) :									*	*	*
E_2 (4) :	*	*	*	*	*						
E_3 (1) :		*	*								
E_4 (3) :				*	*	*	*	*			
E_5 (3) :			*	*	*	*					
E_6 (2) :						*	*	*			
R :	0	1	2	3	4	5	6	7	8	9	10

Le résultat est dans *max*

position:	0	1	2	2	3	4	5	5	7	7	8	10
exon:	2_ℓ	3_ℓ	3_r	5_ℓ	4_ℓ	2_r	5_r	6_ℓ	6_r	4_r	1_ℓ	1_r

exons:	1	2	3	4	5	6
maximum:	8	4	1	4	4	6

$max = 8$

Chainage 1D: Exercices

Exercices

- Comment gérer le chevauchement?
- Quelle est la complexité de l'algorithme?
- Comment faire pour obtenir la liste des exons retenus?
- Calculer le chaînage optimal pour les intervals suivants:

$$[(8, 10, 2); (10, 11, 1); (6, 9, 3); (3, 7, 1); (1, 4, 2); (0, 2, 2)]$$

- Écrire le pseudo-code (ou le code python) permettant de calculer le chaînage optimal (valeur + éléments). Le paramètre en entrée est une liste de triplets $(debut, fin, score)$.

Chaînage à deux dimensions

Problème

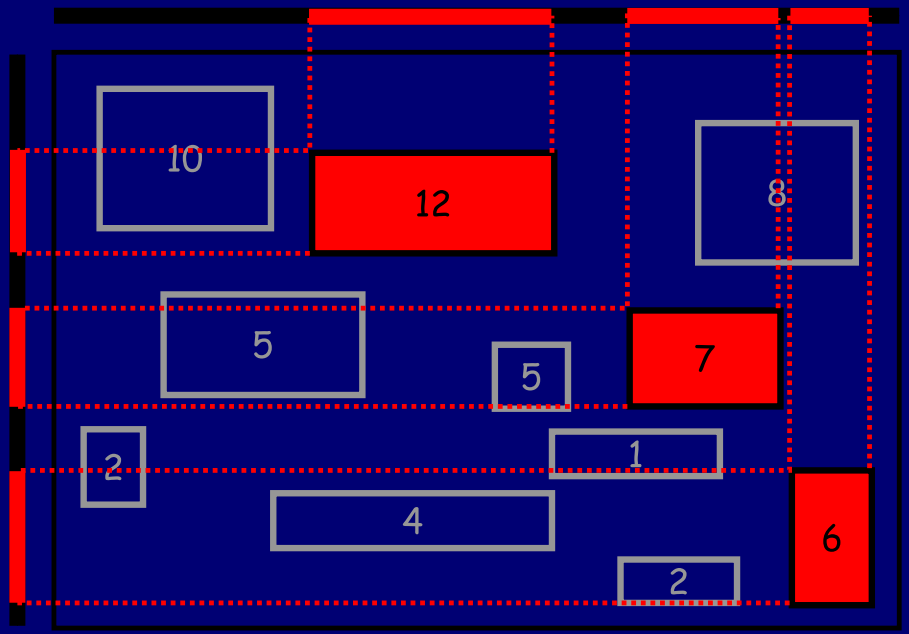
Chaînage à deux dimensions

Problème

Chaînage à deux dimensions

Problème

On dispose de deux séquences d'ADN S_1 et S_2 ainsi qu'un ensemble $\mathcal{R}$ de couple de sous-chaînes de S_1 et S_2 de fortes similarités. À chaque couple est associé un score.



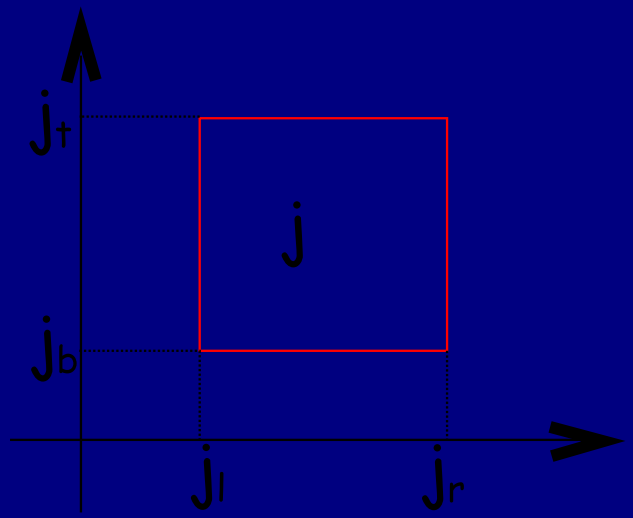
Problème: Trouver un sous-ensemble de $\mathcal{R}$ tel que les zones ne soient pas chevauchantes et la somme des scores retenus soit maximale.

Chaînage 2D: Algorithme

L'algorithme est similaire au cas à une dimension

Chaque "rectangle" j est composé de 5 valeurs:

- j_l: valeur gauche
- j_r: valeur droite
- j_t: valeur haute
- j_b: valeur basse
- v_j: score



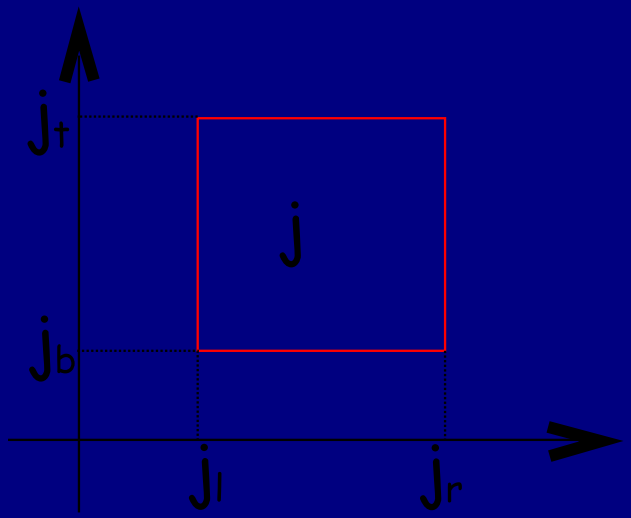
On commence par trier dans un tableau $\mathcal{I}$ l'ensemble des valeurs j_g, j_d pour tous les $j \in \mathcal{R}$.

Pour chaque valeur de $\mathcal{I}$ on conserve le numéro de rectangle correspondant.

Chaînage 2D: Algorithme

L'algorithme est similaire au cas à une dimension
Chaque "rectangle" j est composé de 5 valeurs:

- j_l: valeur gauche
- j_r: valeur droite
- j_t: valeur haute
- j_b: valeur basse
- v_j: score

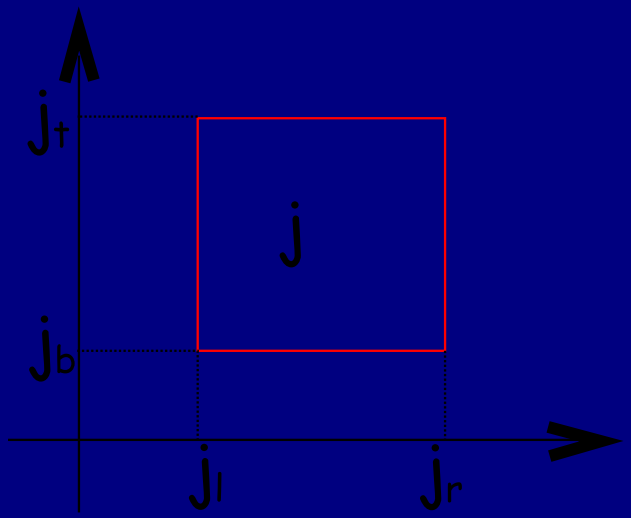


On commence par trier dans un tableau $\mathcal{I}$ l'ensemble des valeurs j_g, j_d pour tous les $j \in \mathcal{R}$.
Pour chaque valeur de $\mathcal{I}$ on conserve le numéro de rectangle correspondant.

Chaînage 2D: Algorithme

L'algorithme est similaire au cas à une dimension
Chaque "rectangle" j est composé de 5 valeurs:

- j_l: valeur gauche
- j_r: valeur droite
- j_t: valeur haute
- j_b: valeur basse
- v_j: score

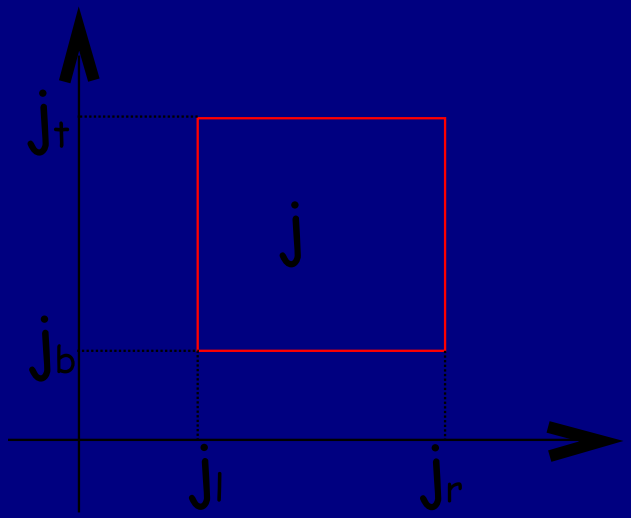


On commence par trier dans un tableau $\mathcal{I}$ l'ensemble des valeurs j_g, j_d pour tous les $j \in \mathcal{R}$.
Pour chaque valeur de $\mathcal{I}$ on conserve le numéro de rectangle correspondant.

Chaînage 2D: Algorithme

L'algorithme est similaire au cas à une dimension
Chaque "rectangle" j est composé de 5 valeurs:

- j_l: valeur gauche
- j_r: valeur droite
- j_t: valeur haute
- j_b: valeur basse
- v_j: score



On commence par trier dans un tableau $\mathcal{I}$ l'ensemble des valeurs j_g, j_d pour tous les $j \in \mathcal{R}$.
Pour chaque valeur de $\mathcal{I}$ on conserve le numéro de rectangle correspondant.

Chaînage 2D: Algorithme

On va aussi maintenir une liste $\mathcal{L}$ de triplets $(j_b, V[j], j)$ où

- j_b est la coordonnée basse du rectangle j
- $V[j]$ est la valeur de meilleur chaînage se terminant par j.

La liste $\mathcal{L}$ sera triée selon les positions basses décroissante. Le triplet en début de liste est le rectangle le plus haut.

Chaînage 2D: Algorithme

On va aussi maintenir une liste $\mathcal{L}$ de triplets $(j_b, V[j], j)$ où

- j_b est la coordonnée basse du rectangle j
- $V[j]$ est la valeur de meilleur chaînage se terminant par j.

La liste $\mathcal{L}$ sera triée selon les positions basses décroissante. Le triplet en début de liste est le rectangle le plus haut.

Chaînage 2D: Algorithme

$\mathcal{L}$ est vide, pour chaque position i de $\mathcal{I}$:

- Si $\mathcal{I}[i]$ est la position gauche du rectangle k:
  - Trouver dans $\mathcal{L}$ le dernier triplet $(j_b, V[j], j)$ tel que $j_b > k_t$ (j est juste au dessus de k).
  - Positionner $V[k]$ à $v_k + V[j]$.
- Sinon $\mathcal{I}[i]$ est la position droite de k:
  - Rechercher le dernier triplet $(j_b, V[j], j)$ de $\mathcal{L}$ tq $j_b \geq k_b$
  - si $V[k] > V[j]$
    - Ajouter $(k_b, V(k), k)$ dans $\mathcal{L}$
  - Retirer de $\mathcal{L}$ les triplets j' tq $j'_b \leq k_b$ et $V[k] > V[j']$ (rectangles plus bas avec un score plus faible).

Chaînage 2D: Algorithme

$\mathcal{L}$ est vide, pour chaque position i de $\mathcal{I}$:

- Si $\mathcal{I}[i]$ est la position gauche du rectangle k:
  - Trouver dans $\mathcal{L}$ le dernier triplet $(j_b, V[j], j)$ tel que $j_b > k_t$ (j est juste au dessus de k).
  - Positionner $V[k]$ à $v_k + V[j]$.
- Sinon $\mathcal{I}[i]$ est la position droite de k:
  - Rechercher le dernier triplet $(j_b, V[j], j)$ de $\mathcal{L}$ tq $j_b \geq k_b$
  - si $V[k] > V[j]$
    - Ajouter $(k_b, V(k), k)$ dans $\mathcal{L}$
  - Retirer de $\mathcal{L}$ les triplets j' tq $j'_b \leq k_b$ et $V[k] > V[j']$ (rectangles plus bas avec un score plus faible).

Chaînage 2D: Algorithme

$\mathcal{L}$ est vide, pour chaque position i de $\mathcal{I}$:

- Si $\mathcal{I}[i]$ est la position gauche du rectangle k:
  - Trouver dans $\mathcal{L}$ le dernier triplet $(j_b, V[j], j)$ tel que $j_b > k_t$ (j est juste au dessus de k).
  - Positionner $V[k]$ à $v_k + V[j]$.
- Sinon $\mathcal{I}[i]$ est la position droite de k:
  - Rechercher le dernier triplet $(j_b, V[j], j)$ de $\mathcal{L}$ tq $j_b \geq k_b$
  - si $V[k] > V[j]$
    - Ajouter $(k_b, V(k), k)$ dans $\mathcal{L}$
  - Retirer de $\mathcal{L}$ les triplets j' tq $j'_b \leq k_b$ et $V[k] > V[j']$ (rectangles plus bas avec un score plus faible).

Chaînage 2D: Algorithme

$\mathcal{L}$ est vide, pour chaque position i de $\mathcal{I}$:

- Si $\mathcal{I}[i]$ est la position gauche du rectangle k:
  - Trouver dans $\mathcal{L}$ le dernier triplet $(j_b, V[j], j)$ tel que $j_b > k_t$ (j est juste au dessus de k).
  - Positionner $V[k]$ à $v_k + V[j]$.
- Sinon $\mathcal{I}[i]$ est la position droite de k:
  - Rechercher le dernier triplet $(j_b, V[j], j)$ de $\mathcal{L}$ tq $j_b \geq k_b$
  - si $V[k] > V[j]$
    - Ajouter $(k_b, V(k), k)$ dans $\mathcal{L}$
  - Retirer de $\mathcal{L}$ les triplets j' tq $j'_b \leq k_b$ et $V[k] > V[j']$ (rectangles plus bas avec un score plus faible).

Chaînage 2D: Algorithme

$\mathcal{L}$ est vide, pour chaque position i de $\mathcal{I}$:

- Si $\mathcal{I}[i]$ est la position gauche du rectangle k:
  - Trouver dans $\mathcal{L}$ le dernier triplet $(j_b, V[j], j)$ tel que $j_b > k_t$ (j est juste au dessus de k).
  - Positionner $V[k]$ à $v_k + V[j]$.
- Sinon $\mathcal{I}[i]$ est la position droite de k:
  - Rechercher le dernier triplet $(j_b, V[j], j)$ de $\mathcal{L}$ tq $j_b \geq k_b$
  - si $V[k] > V[j]$
    - Ajouter $(k_b, V(k), k)$ dans $\mathcal{L}$
  - Retirer de $\mathcal{L}$ les triplets j' tq $j'_b \leq k_b$ et $V[k] > V[j']$ (rectangles plus bas avec un score plus faible).

Chaînage 2D: Algorithme

$\mathcal{L}$ est vide, pour chaque position i de $\mathcal{I}$:

- Si $\mathcal{I}[i]$ est la position gauche du rectangle k:
  - Trouver dans $\mathcal{L}$ le dernier triplet $(j_b, V[j], j)$ tel que $j_b > k_t$ (j est juste au dessus de k).
  - Positionner $V[k]$ à $v_k + V[j]$.
- Sinon $\mathcal{I}[i]$ est la position droite de k:
  - Rechercher le dernier triplet $(j_b, V[j], j)$ de $\mathcal{L}$ tq $j_b \geq k_b$
  - si $V[k] > V[j]$
    - Ajouter $(k_b, V(k), k)$ dans $\mathcal{L}$
  - Retirer de $\mathcal{L}$ les triplets j' tq $j'_b \leq k_b$ et $V[k] > V[j']$ (rectangles plus bas avec un score plus faible).

Chaînage 2D: Algorithme

$\mathcal{L}$ est vide, pour chaque position i de $\mathcal{I}$:

- Si $\mathcal{I}[i]$ est la position gauche du rectangle k:
  - Trouver dans $\mathcal{L}$ le dernier triplet $(j_b, V[j], j)$ tel que $j_b > k_t$ (j est juste au dessus de k).
  - Positionner $V[k]$ à $v_k + V[j]$.
- Sinon $\mathcal{I}[i]$ est la position droite de k:
  - Rechercher le dernier triplet $(j_b, V[j], j)$ de $\mathcal{L}$ tq $j_b \geq k_b$
  - si $V[k] > V[j]$
    - Ajouter $(k_b, V(k), k)$ dans $\mathcal{L}$
  - Retirer de $\mathcal{L}$ les triplets j' tq $j'_b \leq k_b$ et $V[k] > V[j']$ (rectangles plus bas avec un score plus faible).

Chaînage 2D: Algorithme

$\mathcal{L}$ est vide, pour chaque position i de $\mathcal{I}$:

- Si $\mathcal{I}[i]$ est la position gauche du rectangle k:
  - Trouver dans $\mathcal{L}$ le dernier triplet $(j_b, V[j], j)$ tel que $j_b > k_t$ (j est juste au dessus de k).
  - Positionner $V[k]$ à $v_k + V[j]$.
- Sinon $\mathcal{I}[i]$ est la position droite de k:
  - Rechercher le dernier triplet $(j_b, V[j], j)$ de $\mathcal{L}$ tq $j_b \geq k_b$
  - si $V[k] > V[j]$
    - Ajouter $(k_b, V(k), k)$ dans $\mathcal{L}$
  - Retirer de $\mathcal{L}$ les triplets j' tq $j'_b \leq k_b$ et $V[k] > V[j']$ (rectangles plus bas avec un score plus faible).

Chaînage 2D: Exemple

Trois rectangles: 1:(0,2,2,4,2), 2:(1,3,1,5,4), 3:(3,4,0,1,1)

$\mathcal{I}$:

0	2	1	3	3	4
1_g	1				

Chaînage 2D: Exemple

Trois rectangles: 1:(0,2,2,4,2), 2:(1,3,1,5,4), 3:(3,4,0,1,1)

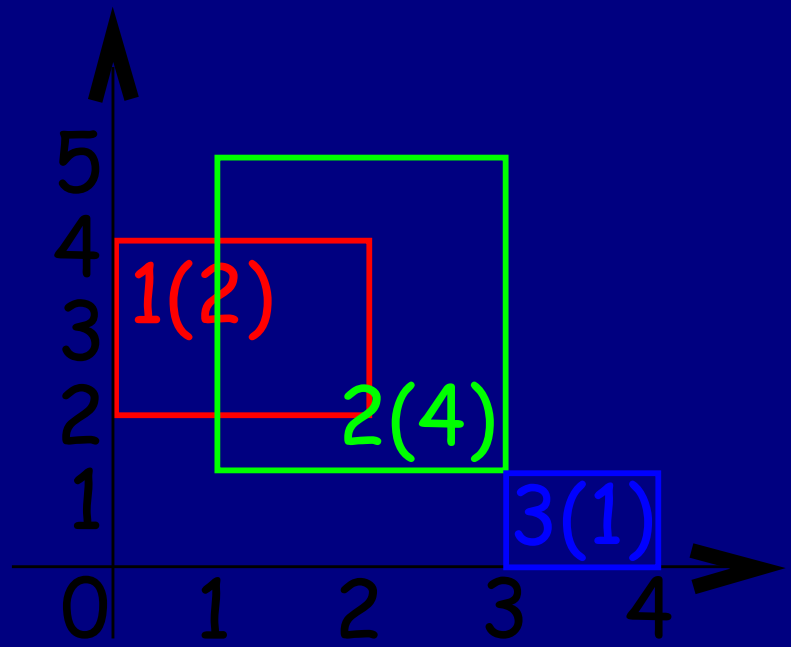
$\mathcal{I}$:

0	1	2	3	3	4
1_g	2_g	1_d	3_g	2_d	3_d

V:

1	2	3

$\mathcal{L}$:



Chaînage 2D: Exemple

Trois rectangles: 1:(0,2,2,4,2), 2:(1,3,1,5,4), 3:(3,4,0,1,1)

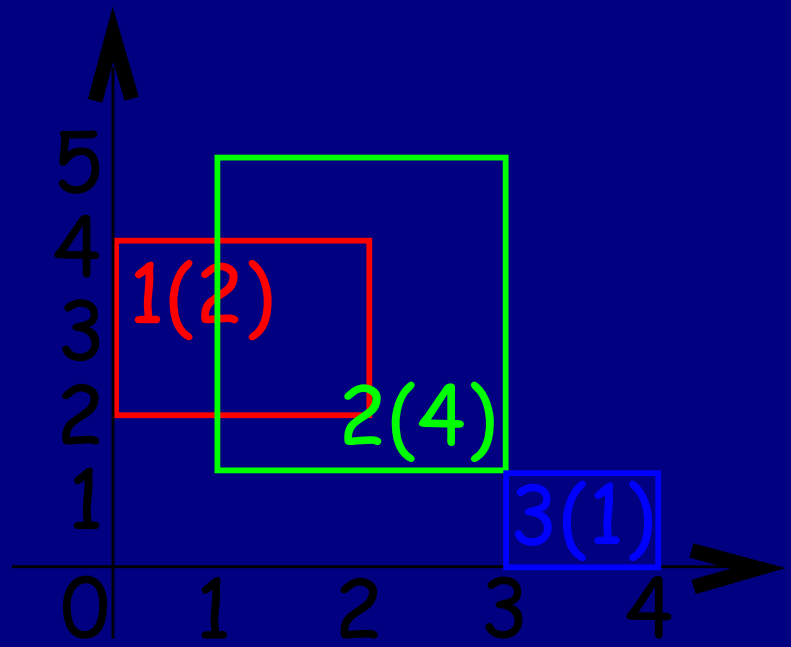
$\mathcal{I}$:

0	1	2	3	3	4
1_g	2_g	1_d	3_g	2_d	3_d

V:

1	2	3

$\mathcal{L}$:



Chaînage 2D: Exemple

Trois rectangles: 1:(0,2,2,4,2), 2:(1,3,1,5,4), 3:(3,4,0,1,1)

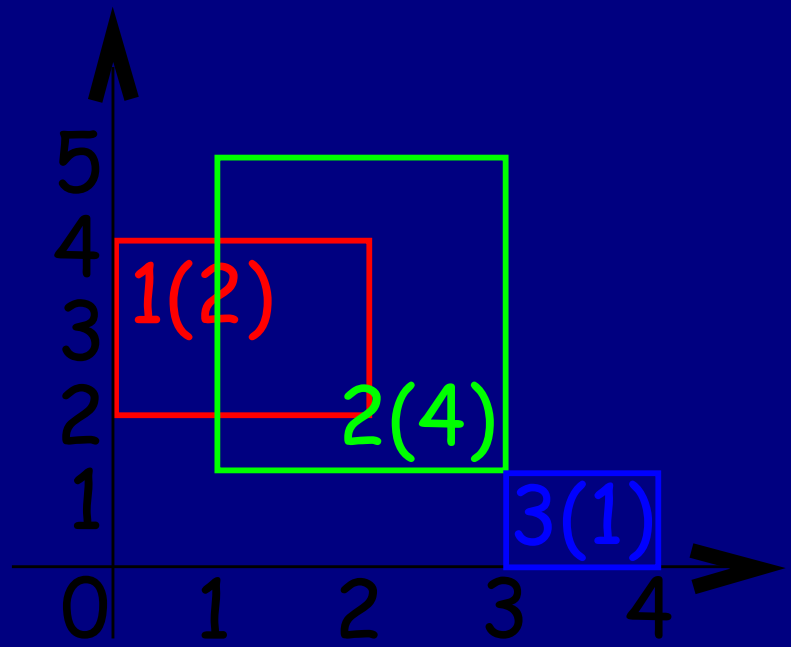
$\mathcal{I}$:

0	1	2	3	3	4
1_g	2_g	1_d	3_g	2_d	3_d

V:

1	2	3
2		

$\mathcal{L}$:



Chaînage 2D: Exemple

Trois rectangles: 1:(0,2,2,4,2), 2:(1,3,1,5,4), 3:(3,4,0,1,1)

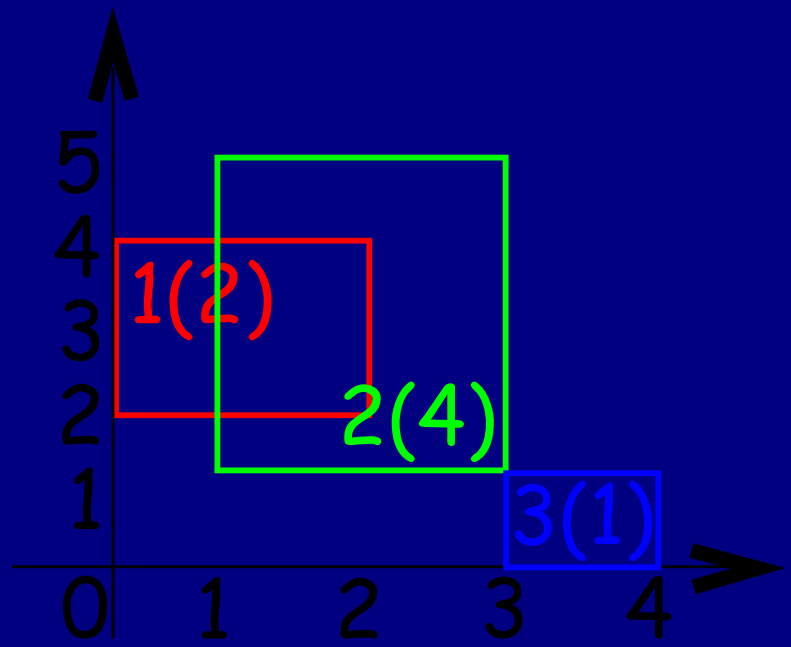
$\mathcal{I}$:

0	1	2	3	3	4
1_g	2_g	1_d	3_g	2_d	3_d

V:

1	2	3
2		

$\mathcal{L}$:



Chaînage 2D: Exemple

Trois rectangles: 1:(0,2,2,4,2), 2:(1,3,1,5,4), 3:(3,4,0,1,1)

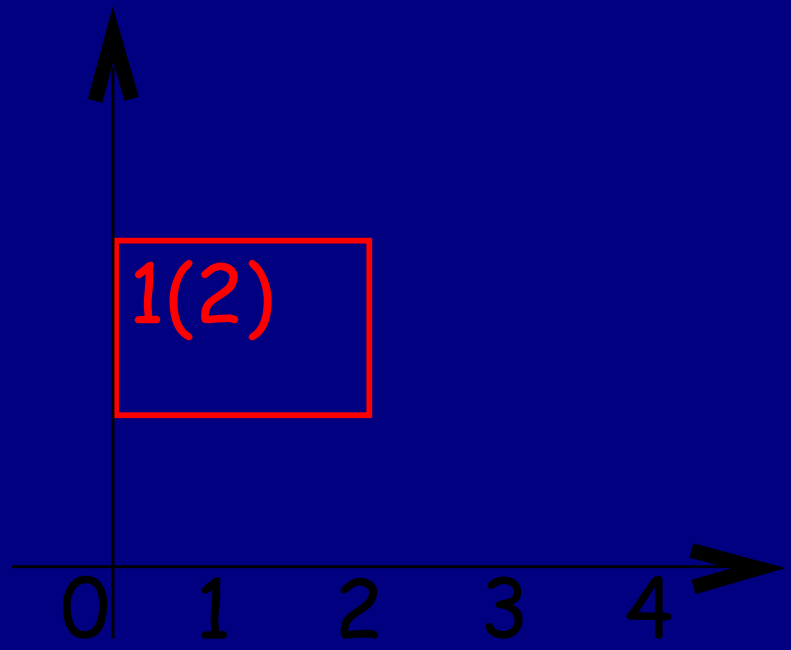
$\mathcal{I}$:

0	1	2	3	3	4
1_g	2_g	1_d	3_g	2_d	3_d

V:

1	2	3
2	4	

$\mathcal{L}$:



Chaînage 2D: Exemple

Trois rectangles: 1:(0,2,2,4,2), 2:(1,3,1,5,4), 3:(3,4,0,1,1)

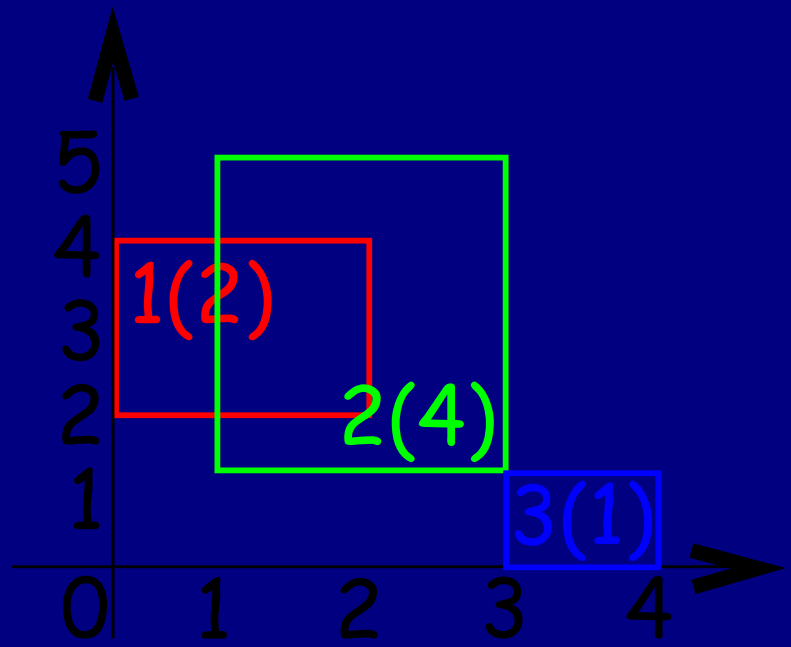
$\mathcal{I}$:

0	1	2	3	3	4
1_g	2_g	1_d	3_g	2_d	3_d

V:

1	2	3
2	4	

$\mathcal{L}$:



Chaînage 2D: Exemple

Trois rectangles: 1:(0,2,2,4,2), 2:(1,3,1,5,4), 3:(3,4,0,1,1)

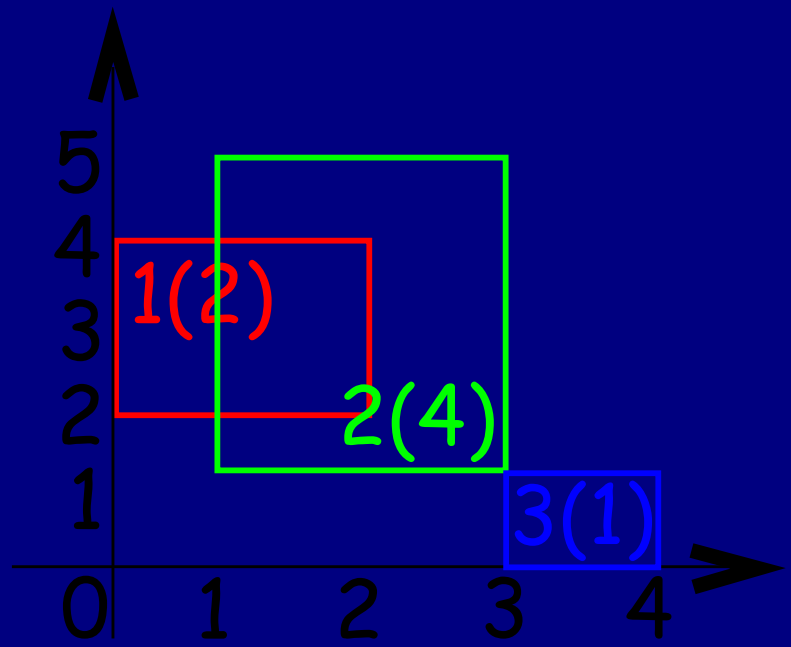
$\mathcal{I}$:

0	1	2	3	3	4
1_g	2_g	1_d	3_g	2_d	3_d

V:

1	2	3
2	4	

$\mathcal{L}$: (2,2,1)



Chaînage 2D: Exemple

Trois rectangles: 1:(0,2,2,4,2), 2:(1,3,1,5,4), 3:(3,4,0,1,1)

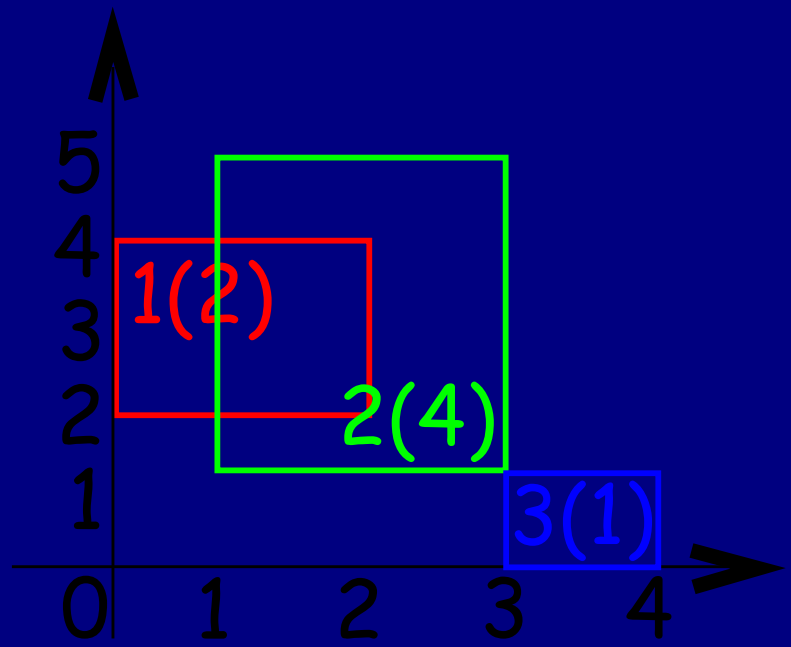
$\mathcal{I}$:

0	1	2	3	3	4
1_g	2_g	1_d	3_g	2_d	3_d

V:

1	2	3
2	4	

$\mathcal{L}$: (2,2,1)



Chaînage 2D: Exemple

Trois rectangles: 1:(0,2,2,4,2), 2:(1,3,1,5,4), 3:(3,4,0,1,1)

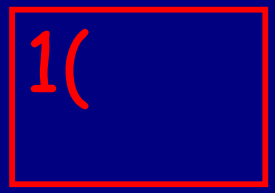
$\mathcal{I}$:

0	1	2	3	3	4
1_g	2_g	1_d	3_g	2_d	3_d

V:

1	2	3
2	4	

$\mathcal{L}$: (2,2,1) (2>1)



Chaînage 2D: Exemple

Trois rectangles: 1:(0,2,2,4,2), 2:(1,3,1,5,4), 3:(3,4,0,1,1)

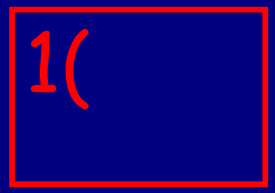
$\mathcal{I}$:

0	1	2	3	3	4
1_g	2_g	1_d	3_g	2_d	3_d

V:

1	2	3
2	4	3= 1+2

$\mathcal{L}$: (2,2,1)



Chaînage 2D: Exemple

Trois rectangles: 1:(0,2,2,4,2), 2:(1,3,1,5,4), 3:(3,4,0,1,1)

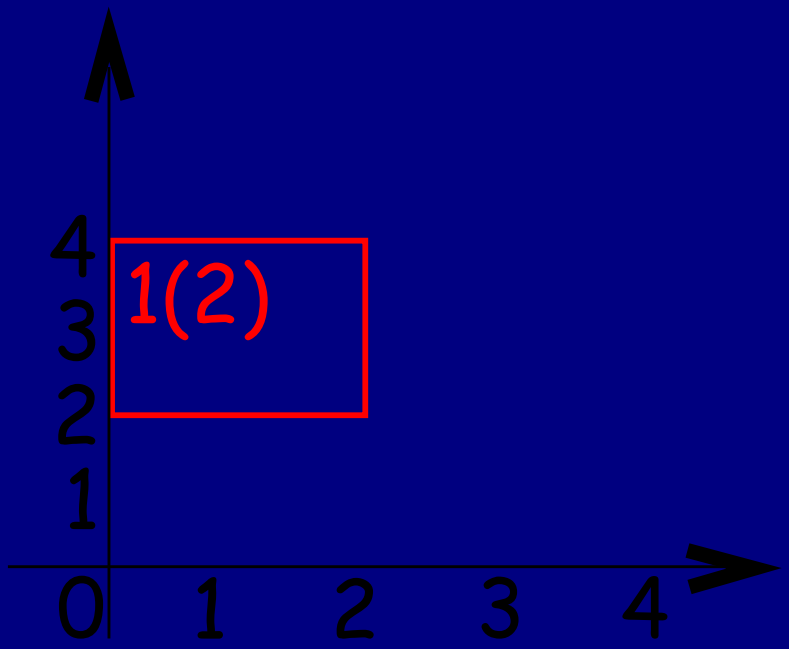
$\mathcal{I}$:

0	1	2	3	3	4
1_g	2_g	1_d	3_g	2_d	3_d

V:

1	2	3
2	4	3

$\mathcal{L}$: (2,2,1) (1,4,2)



Chaînage 2D: Exemple

Trois rectangles: 1:(0,2,2,4,2), 2:(1,3,1,5,4), 3:(3,4,0,1,1)

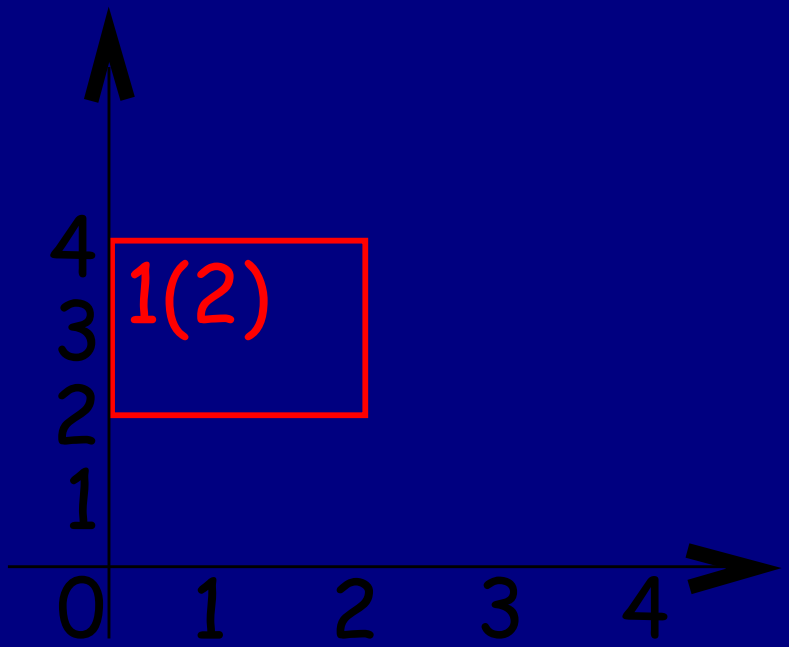
$\mathcal{I}$:

0	1	2	3	3	4
1_g	2_g	1_d	3_g	2_d	3_d

V:

1	2	3
2	4	3

$\mathcal{L}$: (2,2,1) (1,4,2)



Chaînage 2D: Exemple

Trois rectangles: 1:(0,2,2,4,2), 2:(1,3,1,5,4), 3:(3,4,0,1,1)

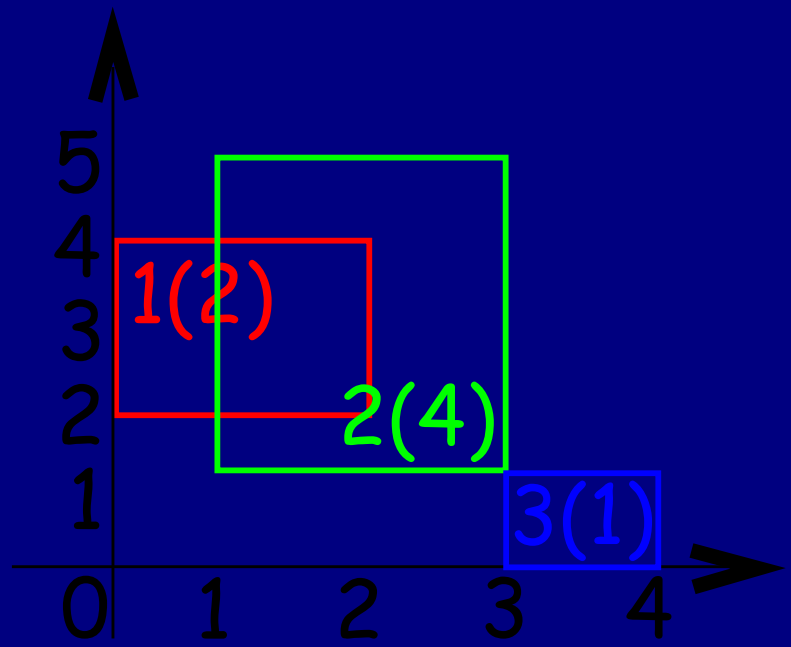
$\mathcal{I}$:

0	1	2	3	3	4
1_g	2_g	1_d	3_g	2_d	3_d

V:

1	2	3
2	4	3

$\mathcal{L}$: (2,2,1) (1,4,2)



Chaînage 2D: Exemple

Trois rectangles: 1:(0,2,2,4,2), 2:(1,3,1,5,4), 3:(3,4,0,1,1)

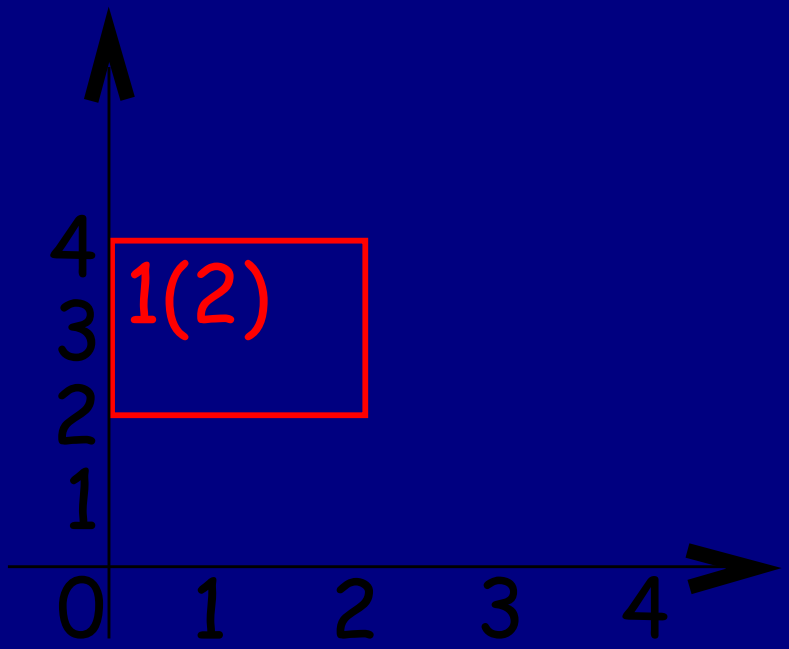
$\mathcal{I}$:

0	1	2	3	3	4
1_g	2_g	1_d	3_g	2_d	3_d

V:

1	2	3
2	4	3

$\mathcal{L}$: (2,2,1) (1,4,2)



Chaînage 2D: Exemple

Trois rectangles: 1:(0,2,2,4,2), 2:(1,3,1,5,4), 3:(3,4,0,1,1)

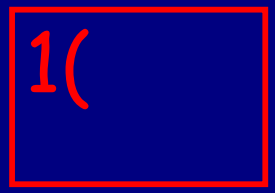
$\mathcal{I}$:

0	1	2	3	3	4
1_g	2_g	1_d	3_g	2_d	3_d

V:

1	2	3
2	4	3

$\mathcal{L}$: (2,2,1) (1,4,2)



Chainage 2D: Complexité

On remarque tout d'abord que les triplets de $\mathcal{L}$ sont implicitement triés selon les scores croissants.

La complexité pour le tri de $\mathcal{I}$ est $O(nlog(n))$.

Pour les $2n$ positions (gauche/droites), la recherche l'insertion et la délétion dans $\mathcal{L}$ se fait en $O(log(n))$ (ABR).

La complexité totale est $O(nlog(n))$.

possibilité d'extension avec des coûts pour les gaps

Chainage 2D: Complexité

On remarque tout d'abord que les triplets de $\mathcal{L}$ sont implicitement triés selon les scores croissants.

La complexité pour le tri de $\mathcal{I}$ est $O(nlog(n))$.

Pour les $2n$ positions (gauche/droites), la recherche l'insertion et la délétion dans $\mathcal{L}$ se fait en $O(log(n))$ (ABR).

La complexité totale est $O(nlog(n))$.

possibilité d'extension avec des coûts pour les gaps

Chainage 2D: Complexité

On remarque tout d'abord que les triplets de $\mathcal{L}$ sont implicitement triés selon les scores croissants.

La complexité pour le tri de $\mathcal{I}$ est $O(nlog(n))$.

Pour les $2n$ positions (gauche/droites), la recherche l'insertion et la délétion dans $\mathcal{L}$ se fait en $O(log(n))$ (ABR).

La complexité totale est $O(nlog(n))$.

possibilité d'extension avec des coûts pour les gaps

Chainage 2D: Complexité

On remarque tout d'abord que les triplets de $\mathcal{L}$ sont implicitement triés selon les scores croissants.

La complexité pour le tri de $\mathcal{I}$ est $O(nlog(n))$.

Pour les $2n$ positions (gauche/droites), la recherche l'insertion et la délétion dans $\mathcal{L}$ se fait en $O(log(n))$ (ABR).

La complexité totale est $O(nlog(n))$.

possibilité d'extension avec des coûts pour les gaps

Chainage 2D: Complexité

On remarque tout d'abord que les triplets de $\mathcal{L}$ sont implicitement triés selon les scores croissants.

La complexité pour le tri de $\mathcal{I}$ est $O(nlog(n))$.

Pour les $2n$ positions (gauche/droites), la recherche l'insertion et la délétion dans $\mathcal{L}$ se fait en $O(log(n))$ (ABR).

La complexité totale est $O(nlog(n))$.

possibilité d'extension avec des coûts pour les gaps

Plan

Le concept d'index

- Soit une donnée D de grande taille
- Nous cherchons des *parties* de S qui sont égale à certains motifs (pas encore connus).
- Sans pré-traitement sur D, chaque recherche demandera à parcourir D entièrement.
- $\Rightarrow$ Idée: pré-traiter D pour que chaque requête aille plus vite.
- $\Rightarrow$ Construire une structure de donnée que l'on appelle un index de D.

Le concept d'index

- Soit une donnée D de grande taille
- Nous cherchons des *parties* de S qui sont égale à certains motifs (pas encore connus).
- Sans pré-traitement sur D, chaque recherche demandera à parcourir D entièrement.
- $\Rightarrow$ Idée: pré-traiter D pour que chaque requête aille plus vite.
- $\Rightarrow$ Construire une structure de donnée que l'on appelle un index de D.

Le concept d'index

- Soit une donnée D de grande taille
- Nous cherchons des *parties* de S qui sont égale à certains motifs (pas encore connus).
- Sans pré-traitement sur D, chaque recherche demandera à parcourir D entièrement.
- $\Rightarrow$ Idée: pré-traiter D pour que chaque requête aille plus vite.
- $\Rightarrow$ Construire une structure de donnée que l'on appelle un index de D.

Le concept d'index

- Soit une donnée D de grande taille
- Nous cherchons des *parties* de S qui sont égale à certains motifs (pas encore connus).
- Sans pré-traitement sur D, chaque recherche demandera à parcourir D entièrement.
- $\Rightarrow$ Idée: pré-traiter D pour que chaque requête aille plus vite.
- $\Rightarrow$ Construire une structure de donnée que l'on appelle un index de D.

Le concept d'index

- Soit une donnée D de grande taille
- Nous cherchons des *parties* de S qui sont égale à certains motifs (pas encore connus).
- Sans pré-traitement sur D, chaque recherche demandera à parcourir D entièrement.
- $\Rightarrow$ Idée: pré-traiter D pour que chaque requête aille plus vite.
- $\Rightarrow$ Construire une structure de donnée que l'on appelle un index de D.

Regroupement des données

- Extraire de D toutes les *parties* possibles.
- Ordonner ces éléments de telle façon que chaque requête prend un temps proportionnel à la taille de celle-ci $\Rightarrow$ construire un index
- Compresser l'index afin d'éliminer la redondance.

Regroupement des données

- Extraire de D toutes les *parties* possibles.
- Ordonner ces éléments de telle façon que chaque requête prend un temps proportionnel à la taille de celle-ci $\Rightarrow$ construire un index
- Compresser l'index afin d'éliminer la redondance.

Regroupement des données

- Extraire de D toutes les *parties* possibles.
- Ordonner ces éléments de telle façon que chaque requête prend un temps proportionnel à la taille de celle-ci $\Rightarrow$ construire un index
- Compresser l'index afin d'éliminer la redondance.

Plan

L'arbre des suffixes

- T est un *large* texte
- Nous allons chercher des facteurs dans T
- On extrait de T tous les facteurs possibles
- On indexe ces facteurs dans un arbre digital (lexical)

Arbre digital

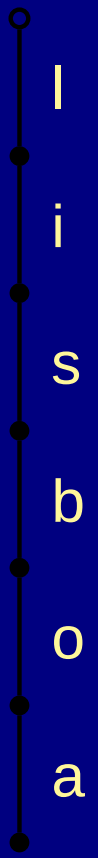
Arbre digital du mot ”lisboa”

Arbre digital

Arbre digital du mot ”lisboa”

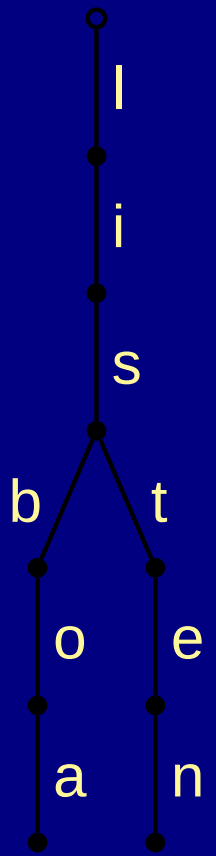
Arbre digital

Arbre digital du mot ”lisboa”



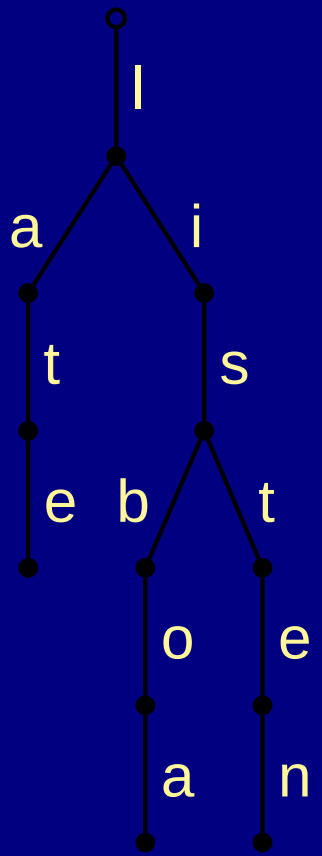
Arbre digital

Arbre digital des mots ”lisboa” and ”listen”



Arbre digital

Arbre digital des mots ”lisboa”, ”listen” and ”late”



Arbre digital des facteurs

Pour le mot ”cacao” tous les facteurs possibles sont $\{c, a, c, a, o, ca, \ldots, cacao\}$.

On retire les doublons $\{c, a, o\}$, $\{ca, ac, ao\}$, $\{cac, aca, cao\}$, $\{caca, acao\}$,$\{cacao\}$

Inserons les dans un arbre digital

Arbre digital des facteurs

Pour le mot ”cacao” tous les facteurs possibles sont $\{c, a, c, a, o, ca, \ldots, cacao\}$.

On retire les doublons $\{c, a, o\}$, $\{ca, ac, ao\}$, $\{cac, aca, cao\}$, $\{caca, acao\}$,$\{cacao\}$

Inserons les dans un arbre digital

Arbre digital des facteurs

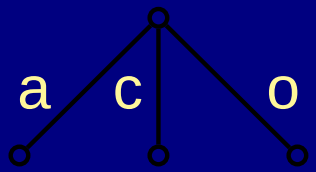
Pour le mot ”cacao” tous les facteurs possibles sont $\{c, a, c, a, o, ca, \ldots, cacao\}$.

On retire les doublons $\{c, a, o\}$, $\{ca, ac, ao\}$, $\{cac, aca, cao\}$, $\{caca, acao\}$,$\{cacao\}$

Inserons les dans un arbre digital

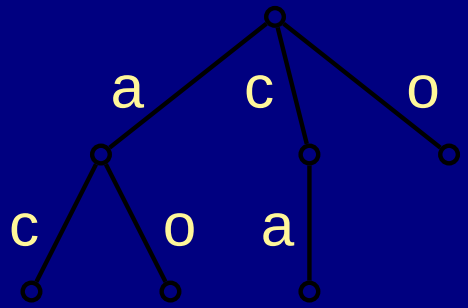
Arbre digital des facteurs

$\{c, a, o\}$, $\{ca, ac, ao\}$, $\{cac, aca, cao\}$, $\{caca, acao\}$,$\{cacao\}$.



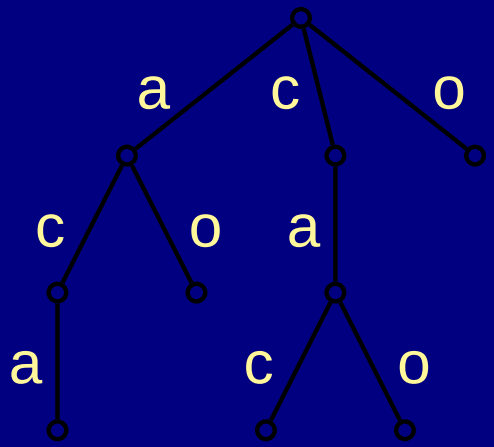
Arbre digital des facteurs

$\{c, a, o\}$, $\{ca, ac, ao\}$, $\{cac, aca, cao\}$, $\{caca, acao\}$,$\{cacao\}$.



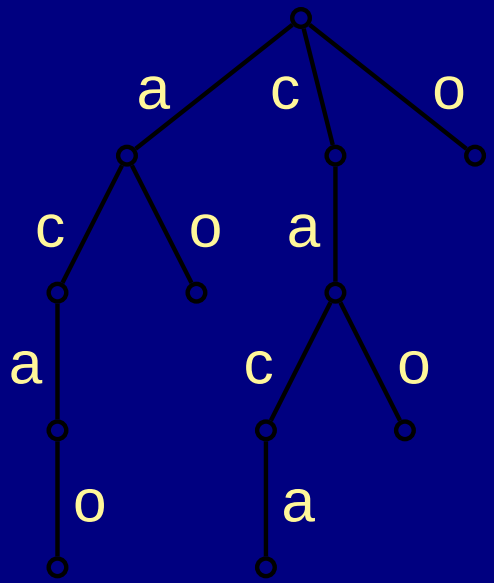
Arbre digital des facteurs

$\{c, a, o\}$, $\{ca, ac, ao\}$, $\{cac, aca, cao\}$, $\{caca, acao\}$,$\{cacao\}$.



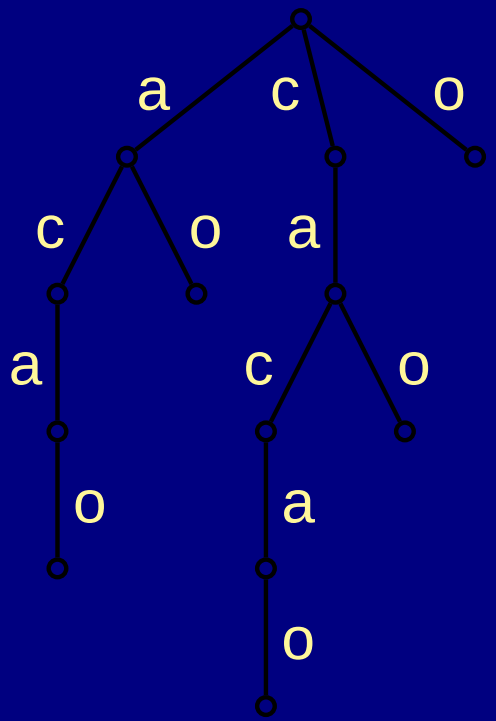
Arbre digital des facteurs

$\{c, a, o\}$, $\{ca, ac, ao\}$, $\{cac, aca, cao\}$, $\{caca, acao\}$,$\{cacao\}$.



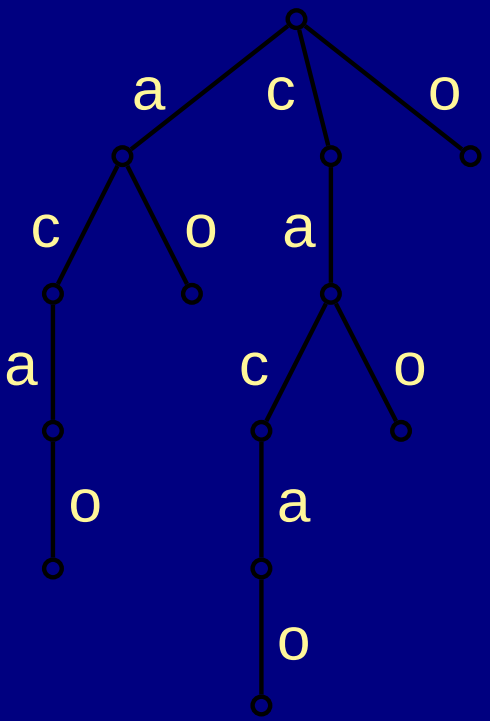
Arbre digital des facteurs

$\{c, a, o\}$, $\{ca, ac, ao\}$, $\{cac, aca, cao\}$, $\{caca, acao\}$, $\{cacao\}$.



Arbre digital des facteurs

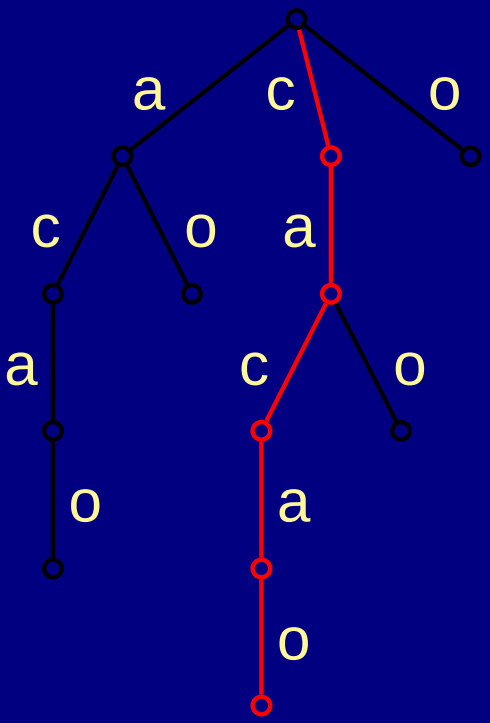
$\{c, a, o\}$, $\{ca, ac, ao\}$, $\{cac, aca, cao\}$, $\{caca, acao\}$, $\{cacao\}$.



Regardons les mots eppelés depuis la racine vers les feuilles:

Arbre digital des facteurs

$\{c, a, o\}$, $\{ca, ac, ao\}$, $\{cac, aca, cao\}$, $\{caca, acao\}$, $\{cacao\}$.

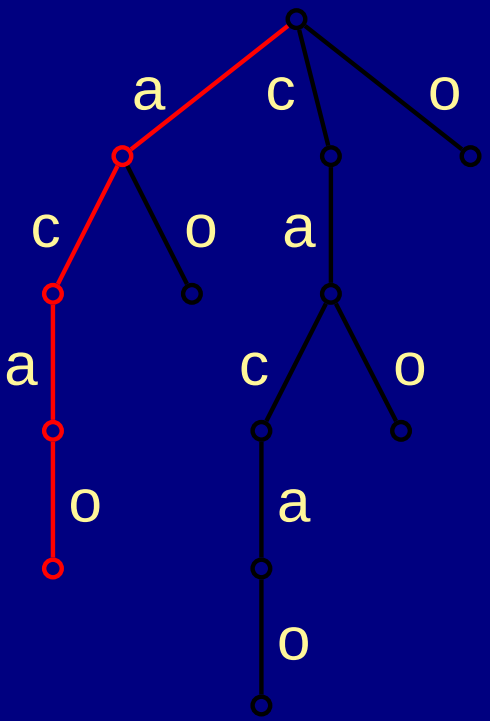


Regardons les mots eppelés depuis la racine vers les feuilles:

- cacao

Arbre digital des facteurs

$\{c, a, o\}$, $\{ca, ac, ao\}$, $\{cac, aca, cao\}$, $\{caca, acao\}$, $\{cacao\}$.

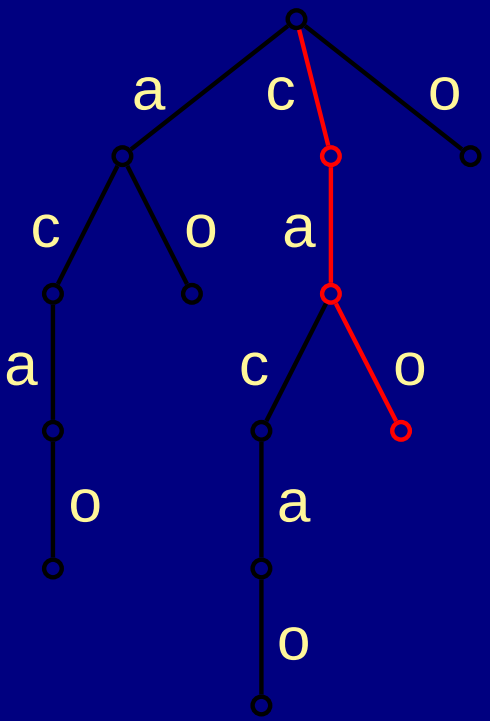


Regardons les mots eppelés depuis la racine vers les feuilles:

- cacao
- acao

Arbre digital des facteurs

$\{c, a, o\}$, $\{ca, ac, ao\}$, $\{cac, aca, cao\}$, $\{caca, acao\}$, $\{cacao\}$.

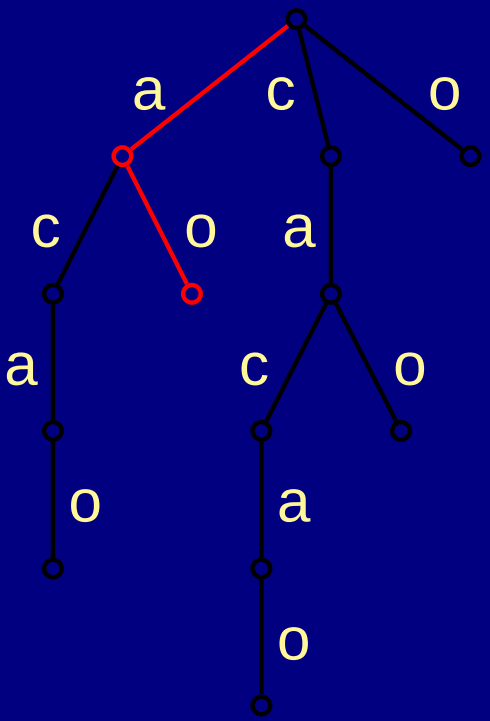


Regardons les mots eppelés depuis la racine vers les feuilles:

- cacao
- acao
- cao

Arbre digital des facteurs

$\{c, a, o\}$, $\{ca, ac, ao\}$, $\{cac, aca, cao\}$, $\{caca, acao\}$, $\{cacao\}$.

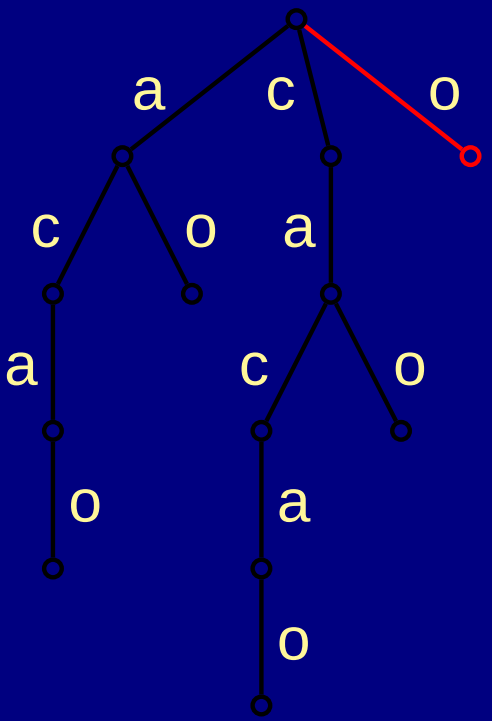


Regardons les mots eppelés depuis la racine vers les feuilles:

- cacao
- acao
- cao
- ao

Arbre digital des facteurs

$\{c, a, o\}$, $\{ca, ac, ao\}$, $\{cac, aca, cao\}$, $\{caca, acao\}$, $\{cacao\}$.

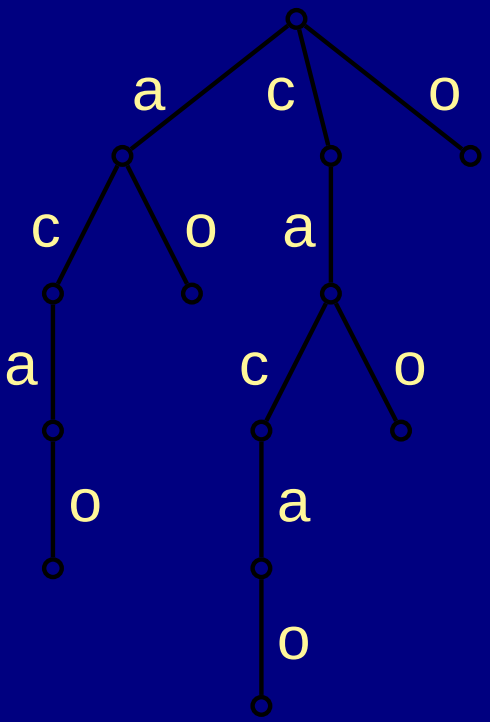


Regardons les mots eppelés depuis la racine vers les feuilles:

- cacao
- acao
- cao
- ao
- o

Arbre digital des facteurs

$\{c, a, o\}$, $\{ca, ac, ao\}$, $\{cac, aca, cao\}$, $\{caca, acao\}$, $\{cacao\}$.



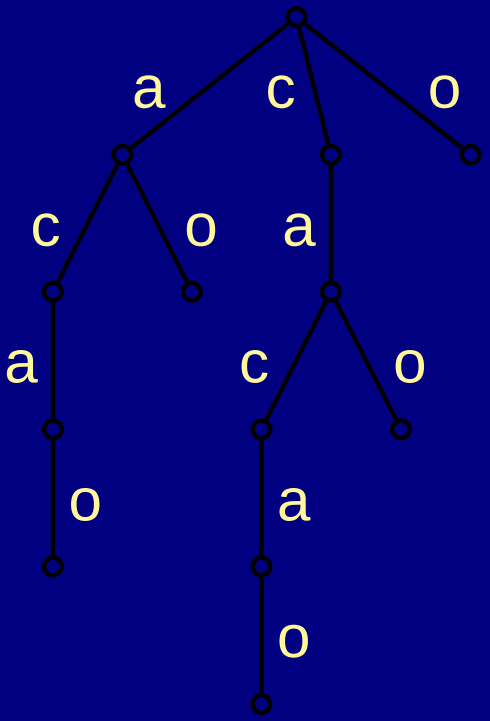
Regardons les mots eppelés depuis la racine vers les feuilles:

- cacao
- acao
- cao
- ao
- o

$\Rightarrow$ Nous avons indexer tous les suffixes de ”cacao”.

Arbre digital des facteurs

$\{c, a, o\}$, $\{ca, ac, ao\}$, $\{cac, aca, cao\}$, $\{caca, acao\}$, $\{cacao\}$.



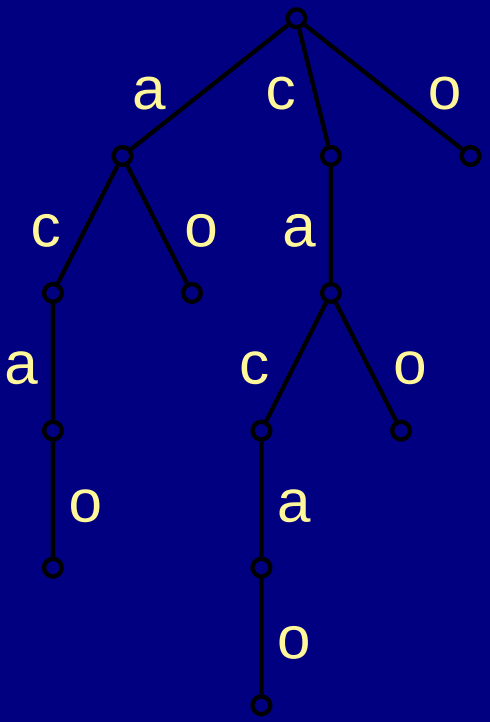
Regardons les mots eppelés depuis la racine vers les feuilles:

- cacao
- acao
- cao
- ao
- o

$\Rightarrow$ Nous avons indexer tous les suffixes de ”cacao”.
(Tous les facteurs sont prefixes d’un suffixe)

Arbre digital des facteurs

$\{c, a, o\}$, $\{ca, ac, ao\}$, $\{cac, aca, cao\}$, $\{caca, acao\}$, $\{cacao\}$.



Complexité

- Quelle est la complexité en espace dans le pire des cas ?
- Quel est le temps necessaire à la construction de l'arbre ?
- Quelle est la complexité pour savoir si un mot est présent dans le texte?
- Quelle est la complexité pour énumérer toutes les occurences d'un mot dans le texte?

Complexité de l'arbre des suffixes

Soit t le texte de taille n sur Σ indexé dans l'arbre des suffixes.

Complexité

Complexité de l'arbre des suffixes

Soit t le texte de taille n sur Σ indexé dans l'arbre des suffixes.

Complexité

- Quelle est la complexité en espace dans le pire des cas?
- Quel est le temps necessaire à la construction de l'arbre ?
- Quel est le temps necessaire à la construction de l'arbre ?
- Quelle est la complexité pour savoir si un mot est présent dans le texte?
- Quelle est la complexité pour énumérer toutes les occurences d'un mot

Complexité de l'arbre des suffixes

- La somme des longueurs des suffixes: $\sum_{i=1}^{i=n} i = O(n^2)$
- Chaque insertion dans l'arbre prend un temps proportionnel à sa longueur $\longrightarrow O(n^2)$
- Chaque insertion dans l'arbre prend un temps proportionnel à sa longueur $\longrightarrow O(n^2)$
- Soit m la taille du motif. Le temps requit est $O(m)$.
- Dans le pire des cas, nous devons traverser tous l'arbre qui contient $O(n^2)$ noeuds $\Rightarrow O(m + n^2)$

Complexité de l'arbre des suffixes

Soit t le texte de taille n sur Σ indexé dans l'arbre des suffixes.

Complexité

- Quelle est la complexité en espace dans le pire des cas?
- Quel est le temps necessaire à la construction de l'arbre ?
- Quelle est la complexité pour savoir si un mot est présent dans le texte?
- Quelle est la complexité pour savoir si un mot est présent dans le texte?
- Quelle est la complexité pour énumérer toutes les occurences d'un mot

Complexité de l'arbre des suffixes

- La somme des longueurs des suffixes: $\sum_{i=1}^{i=n} i = O(n^2)$
- Chaque insertion dans l'arbre prend un temps proportionnel à sa longueur $\longrightarrow O(n^2)$
- Soit m la taille du motif. Le temps requit est $O(m)$.
- Soit m la taille du motif. Le temps requit est $O(m)$.
- Dans le pire des cas, nous devons traverser tous l'arbre qui contient $O(n^2)$ noeuds $\Rightarrow O(m + n^2)$

Complexité de l'arbre des suffixes

Soit t le texte de taille n sur Σ indexé dans l'arbre des suffixes.

Complexité

- Quelle est la complexité en espace dans le pire des cas?
- Quel est le temps necessaire à la construction de l'arbre ?
- Quelle est la complexité pour savoir si un mot est présent dans le texte?
- Quelle est la complexité pour énumérer toutes les occurences d'un mot dans le texte?

Complexité de l'arbre des suffixes

- La somme des longueurs des suffixes: $\sum_{i=1}^{i=n} i = O(n^2)$
- Chaque insertion dans l'arbre prend un temps proportionnel à sa longueur $\longrightarrow O(n^2)$
- Soit m la taille du motif. Le temps requit est $O(m)$.
- Dans le pire des cas, nous devons traverser tous l'arbre qui contient $O(n^2)$ noeuds $\Rightarrow O(m + n^2)$

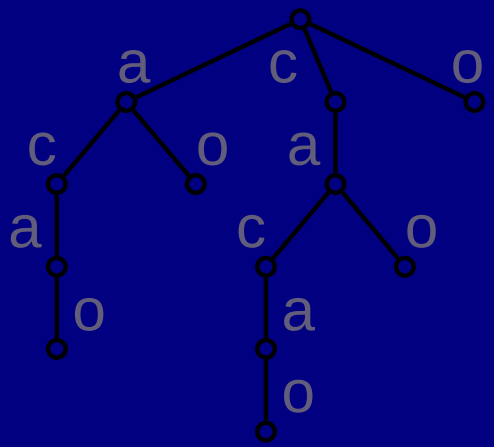
Compacter l'arbre des suffixes

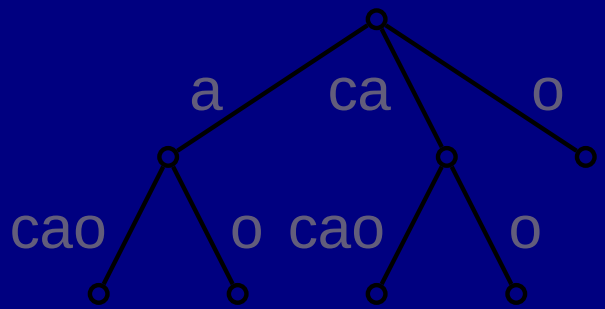
Idée

Pour réduire le temps pour trouver toutes les occurences d'un motif dans le texte nous devons réduire le nombre de noeuds dans l'arbre

Solution

Supprimer tous les noeuds n'aillant qu'un fils. Les arcs sont alors étiquetés sur Σ^*





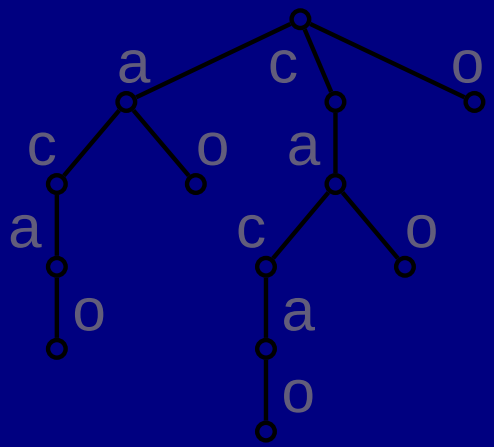
Compacter l'arbre des suffixes

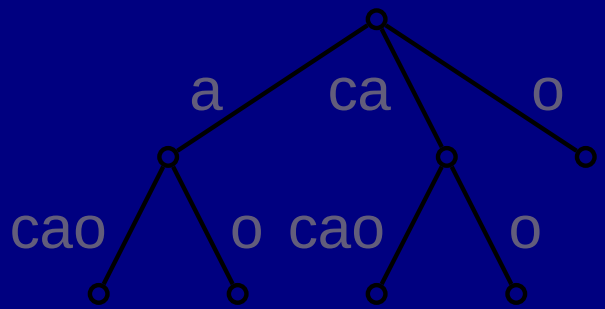
Idée

Pour réduire le temps pour trouver toutes les occurences d'un motif dans le texte nous devons réduire le nombre de noeuds dans l'arbre

Solution

Supprimer tous les noeuds n'aillant qu'un fils. Les arcs sont alors étiquetés sur Σ^*





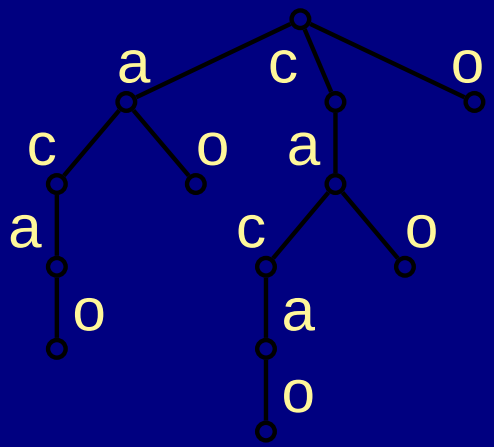
Compacter l'arbre des suffixes

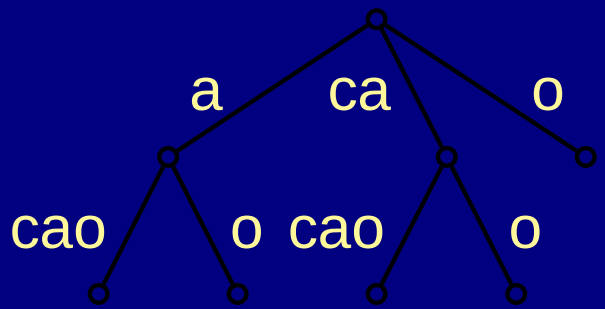
Idée

Pour réduire le temps pour trouver toutes les occurences d'un motif dans le texte nous devons réduire le nombre de noeuds dans l'arbre

Solution

Supprimer tous les noeuds n'aillant qu'un fils. Les arcs sont alors étiquetés sur Σ^*





Arbre compact des suffixes

Complexité en espace

L'arbre ne comporte par plus de n feuilles. Dans le pire des cas chaque noeud interne est de degré 2.

$\Rightarrow$ Il y a au plus $2n$ noeuds dans l'arbre

$\Rightarrow$ Trouver toutes les occurences d'un motif de taille m prend un temps proportionnel au nombre d'occurence L de ce motif: $O(m + L)$ ($O(m + n)$ dans le pire des cas)

Les étiquettes des arcs sont sur Σ^*

$\Rightarrow$ La complexité en espace est toujours $O(n^2)$

Idée

Si l'on veut que l'espace occupé soit linéaire en la taille du texte, il faut que les étiquettes occupent un espace constant.

Arbre compact des suffixes

Complexité en espace

L'arbre ne comporte par plus de n feuilles. Dans le pire des cas chaque noeud interne est de degré 2.

$\Rightarrow$ Il y a au plus $2n$ noeuds dans l'arbre

$\Rightarrow$ Trouver toutes les occurences d'un motif de taille m prend un temps proportionnel au nombre d'occurence L de ce motif: $O(m + L)$ ($O(m + n)$ dans le pire des cas)

Les étiquettes des arcs sont sur Σ^*

$\Rightarrow$ La complexité en espace est toujours $O(n^2)$

Idée

Si l'on veut que l'espace occupé soit linéaire en la taille du texte, il faut que les étiquettes occupent un espace constant.

Arbre compact des suffixes

Complexité en espace

L'arbre ne comporte par plus de n feuilles. Dans le pire des cas chaque noeud interne est de degré 2.
$\Rightarrow$ Il y a au plus $2n$ noeuds dans l'arbre
$\Rightarrow$ Trouver toutes les occurences d'un motif de taille m prend un temps proportionnel au nombre d'occurence L de ce motif: $O(m + L)$ ($O(m + n)$ dans le pire des cas)
Les étiquettes des arcs sont sur Σ^*
$\Rightarrow$ La complexité en espace est toujours $O(n^2)$

Idée

Si l'on veut que l'espace occupé soit linéaire en la taille du texte, il faut que les étiquettes occupent un espace constant.

Arbre compact des suffixes

Complexité en espace

L'arbre ne comporte par plus de n feuilles. Dans le pire des cas chaque noeud interne est de degré 2.
$\Rightarrow$ Il y a au plus $2n$ noeuds dans l'arbre
$\Rightarrow$ Trouver toutes les occurences d'un motif de taille m prend un temps proportionnel au nombre d'occurence L de ce motif: $O(m + L)$ ($O(m + n)$ dans le pire des cas)
Les étiquettes des arcs sont sur Σ^*
$\Rightarrow$ La complexité en espace est toujours $O(n^2)$

Idée

Si l'on veut que l'espace occupé soit linéaire en la taille du texte, il faut que les étiquettes occupent un espace constant.

Arbre compact des suffixes

Complexité en espace

L'arbre ne comporte par plus de n feuilles. Dans le pire des cas chaque noeud interne est de degré 2.
$\Rightarrow$ Il y a au plus $2n$ noeuds dans l'arbre
$\Rightarrow$ Trouver toutes les occurences d'un motif de taille m prend un temps proportionnel au nombre d'occurence L de ce motif: $O(m + L)$ ($O(m + n)$ dans le pire des cas)
Les étiquettes des arcs sont sur Σ^*
$\Rightarrow$ La complexité en espace est toujours $O(n^2)$

Idée

Si l'on veut que l'espace occupé soit linéaire en la taille du texte, il faut que les étiquettes occupent un espace constant.

Arbre compact des suffixes

Complexité en espace

L'arbre ne comporte par plus de n feuilles. Dans le pire des cas chaque noeud interne est de degré 2.
$\Rightarrow$ Il y a au plus $2n$ noeuds dans l'arbre
$\Rightarrow$ Trouver toutes les occurences d'un motif de taille m prend un temps proportionnel au nombre d'occurence L de ce motif: $O(m + L)$
($O(m + n)$ dans le pire des cas)
Les étiquettes des arcs sont sur Σ^*
$\Rightarrow$ La complexité en espace est toujours $O(n^2)$

Idée

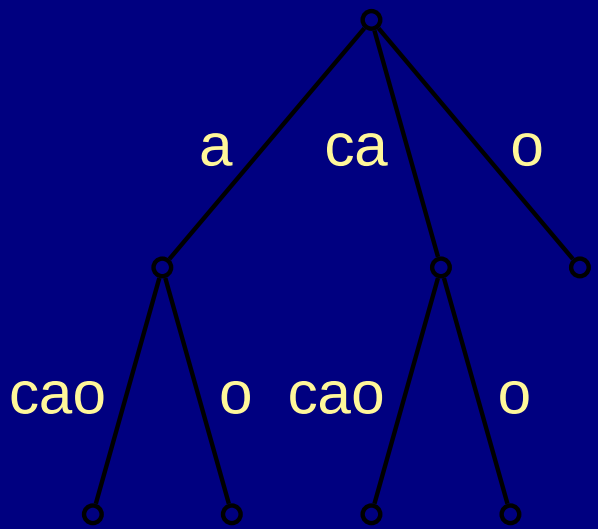
Si l'on veut que l'espace occupé soit linéaire en la taille du texte, il faut que les étiquettes occupent un espace constant.

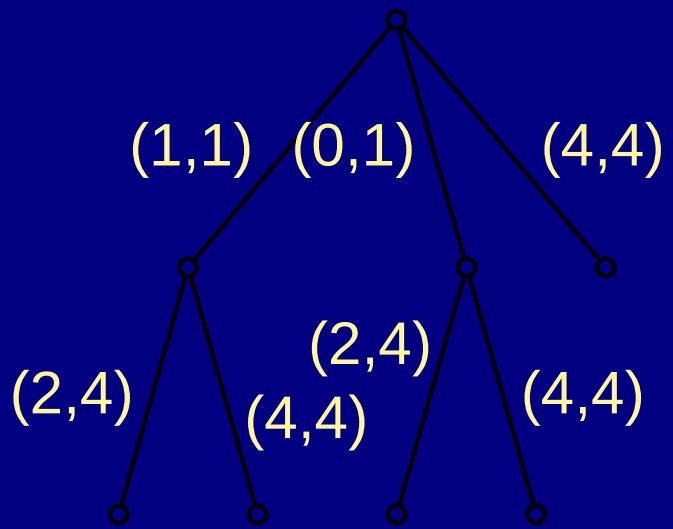
L’arbre des suffixes

c	a	c	a	o
0	1	2	3	4

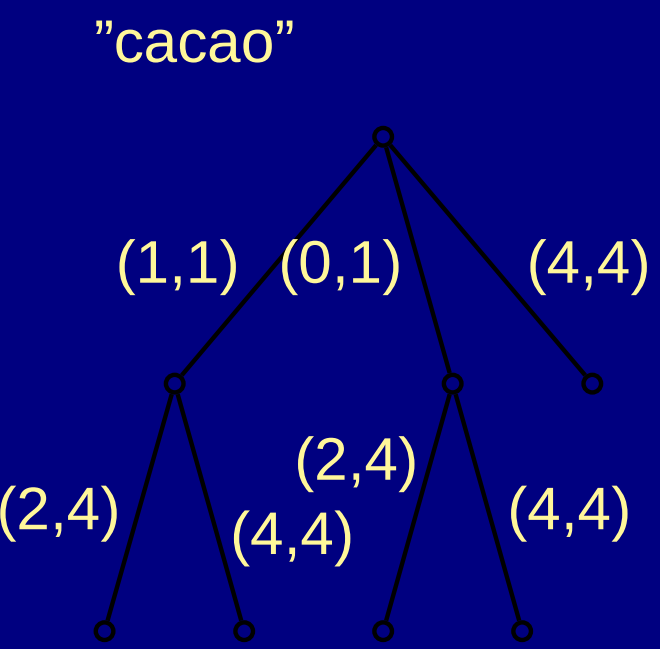
L'arbre des suffixes

c	a	c	a	o
0	1	2	3	4





L'arbre des suffixes

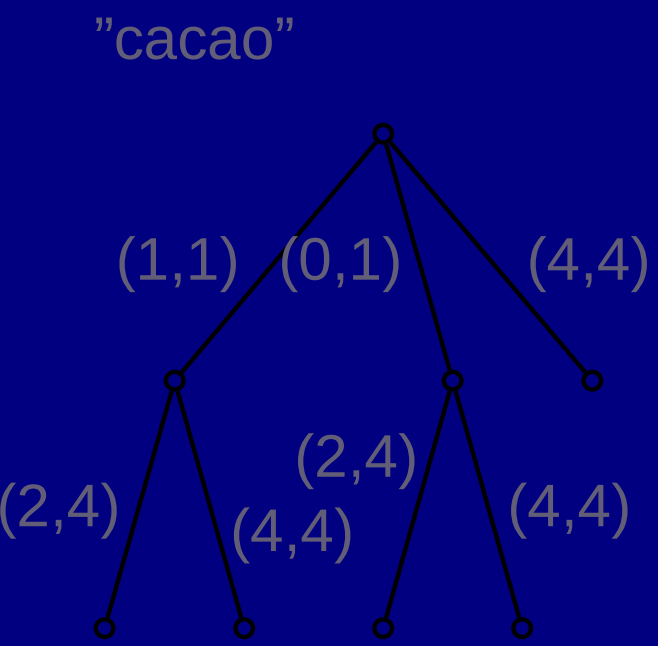


L'arbre implicite des suffixes

Soit t un texte de taille n sur Σ.
L'arbre des suffixes S de t est un arbre dont les étiquettes sont sur Σ^*.
Chaque mot eppelé depuis la racine vers une feuille est un suffixe de t.
Tous les noeuds internes ont un degré ≥ 2 et les étiquettes des arcs sortant débutent avec un caractère différent.
Si la dernière lettre de t apparait ailleur dans t alors S est appelé arbre **implicite** des suffixes.

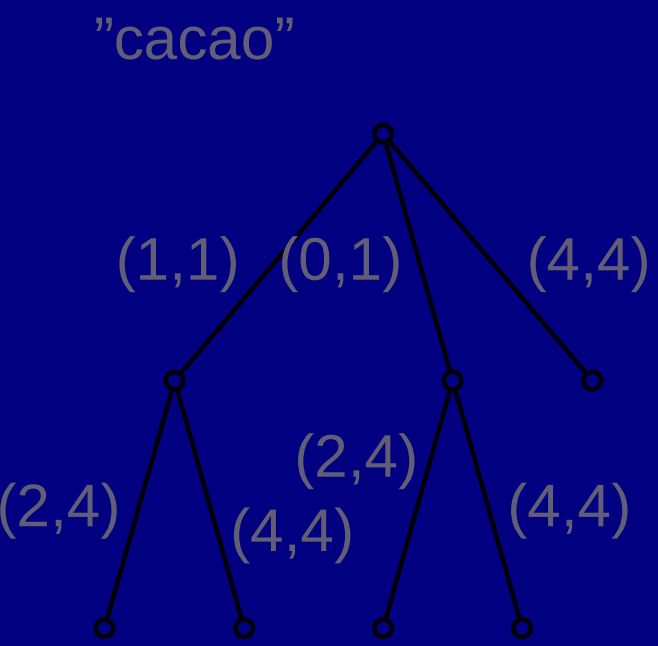
L’arbre des suffixes

L’arbre implicite des suffixes



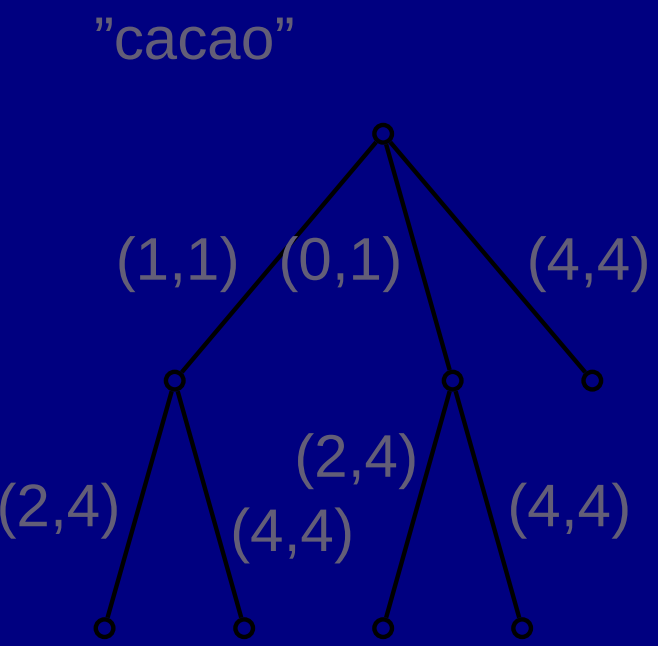
L’arbre des suffixes

L’arbre implicite des suffixes

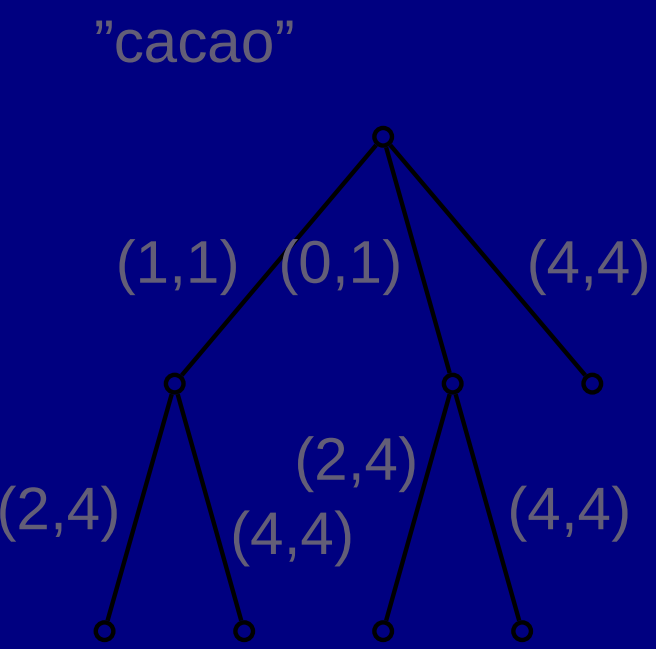


L’arbre des suffixes

L’arbre implicite des suffixes



L'arbre des suffixes



L'arbre implicite des suffixes

Soit t un texte de taille n sur Σ.
L'arbre des suffixes S de t est un arbre dont les étiquettes sont sur Σ^*.
Chaque mot eppelé depuis la racine vers une feuille est un suffixe de t.
Tous les noeuds internes ont un degré ≥ 2 et les étiquettes des arcs sortant débutent avec un caractère différent.
Si la dernière lettre de t apparait ailleur dans t alors S est appelé arbre **implicite** des suffixes.

Exercice

- Construire l'arbre des suffixes de "london"
- Construire l'arbre des suffixes de "london$"

Algorithmes de construction

Il existe trois algorithmes qui construisent l'arbre des suffixes en un temps lineaire en la taille du text:

- Weiner (73)
- Mc Creight (76)
- Ukkonen (95), online

Algorithmes de construction

Il existe trois algorithmes qui construisent l'arbre des suffixes en un temps lineaire en la taille du text:

- Weiner (73)
- Mc Creight (76)
- Ukkonen (95), online

Ukkonen

La construction se compose de n phases. Chaque phase i de 0 à $n-1$ consiste en l'insertion de tous les suffixes de $t_{0\ldots i}$ dans l'arbre.

phase i				i		
text t	t_0	t_1	$\ldots$	t_i	$\ldots$	t_{n-1}
étape	t_0	t_1 t_1	$\ldots$ $\ldots$ $\ddots$	t_i t_i $\vdots$ t_i		

$\Rightarrow$ Cet algorithme construit l'arbre car à la fin tout les suffixes de t sont inserés!

$\Rightarrow$ Quelle est la complexité en temps? $\sum_{i=0}^{n-1} \sum_{j=0}^{j=i} j \rightarrow O(n^3)$

Ukkonen

La construction se compose de n phases. Chaque phase i de 0 à $n-1$ consiste en l'insertion de tous les suffixes de $t_{0\ldots i}$ dans l'arbre.

phase i				i		
text t	t_0	t_1	$\ldots$	t_i	$\ldots$	t_{n-1}
étape	t_0	t_1 t_1	$\ldots$ $\ldots$ $\ddots$	t_i t_i $\vdots$ t_i		

$\Rightarrow$ Cet algorithme construit l'arbre car à la fin tout les suffixes de t sont inserés!

$\Rightarrow$ Quelle est la complexité en temps? $\sum_{i=0}^{n-1} \sum_{j=0}^{j=i} j \rightarrow O(n^3)$

Ukkonen

La construction se compose de n phases. Chaque phase i de 0 à $n-1$ consiste en l'insertion de tous les suffixes de $t_{0\ldots i}$ dans l'arbre.

phase i				i		
text t	t_0	t_1	$\ldots$	t_i	$\ldots$	t_{n-1}
étape	t_0	t_1 t_1	$\ldots$ $\ldots$ $\ddots$	t_i t_i $\vdots$ t_i		

$\Rightarrow$ Cet algorithme construit l'arbre car à la fin tout les suffixes de t sont inserés!

$\Rightarrow$ Quelle est la complexité en temps? $\sum_{i=0}^{n-1} \sum_{j=0}^{j=i} j \rightarrow O(n^3)$

Ukkonen

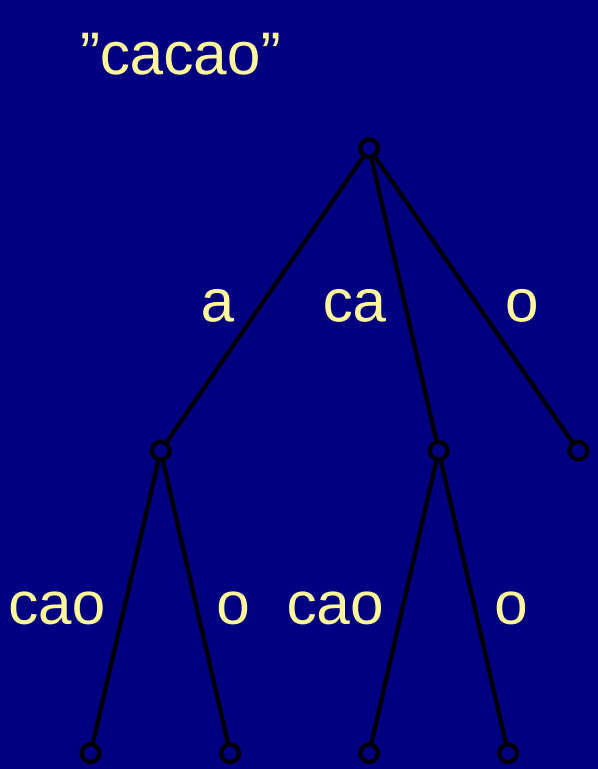
La construction se compose de n phases. Chaque phase i de 0 à $n-1$ consiste en l'insertion de tous les suffixes de $t_{0\ldots i}$ dans l'arbre.

phase i				i		
text t	t_0	t_1	$\ldots$	t_i	$\ldots$	t_{n-1}
étape	t_0	t_1 t_1	$\ldots$ $\ldots$ $\ddots$	t_i t_i $\vdots$ t_i		

$\Rightarrow$ Cet algorithme construit l'arbre car à la fin tout les suffixes de t sont inserés!

$\Rightarrow$ Quelle est la complexité en temps? $\sum_{i=0}^{n-1} \sum_{j=0}^{j=i} j \rightarrow O(n^3)$

3 cas lors de l'insertion



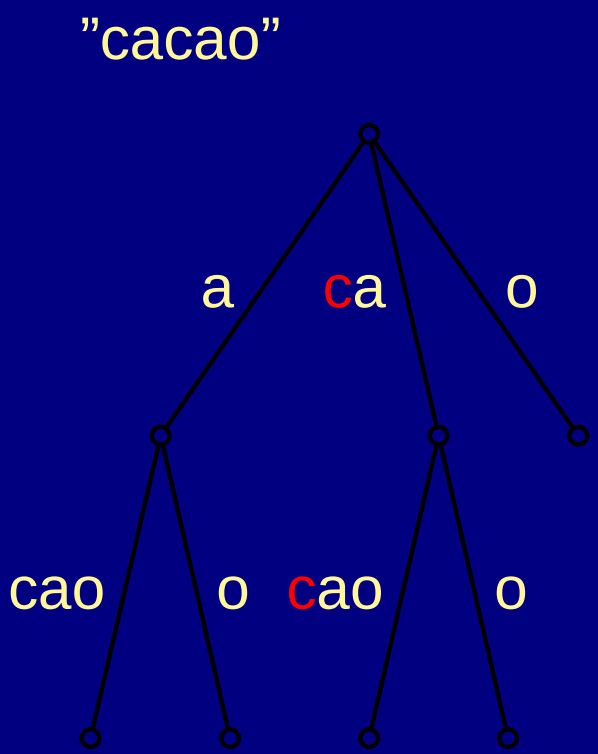
Insertion d'un mot dans l'arbre

On insert $w = vu$, v est le plus long prefixe déjà présent dans l'arbre:

- u est vide $\Rightarrow$ rien à faire
- v mène à une feuille $\Rightarrow$ extension
- création de noeud:
  - v mène à un noeud
  - v mène à un arc

Exemples

3 cas lors de l'insertion



Insertion d'un mot dans l'arbre

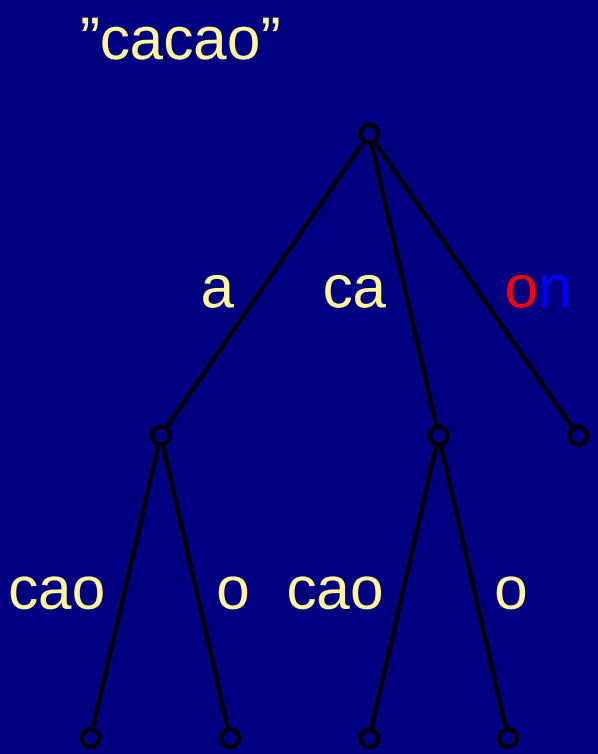
On insert $w = vu$, v est le plus long prefixe déjà présent dans l'arbre:

- u est vide $\Rightarrow$ rien à faire
- v mène à une feuille $\Rightarrow$ extension
- création de noeud:
  - v mène à un noeud
  - v mène à un arc

Exemples

$w = cac$, $v = cac$, $u = \epsilon$

3 cas lors de l'insertion



Insertion d'un mot dans l'arbre

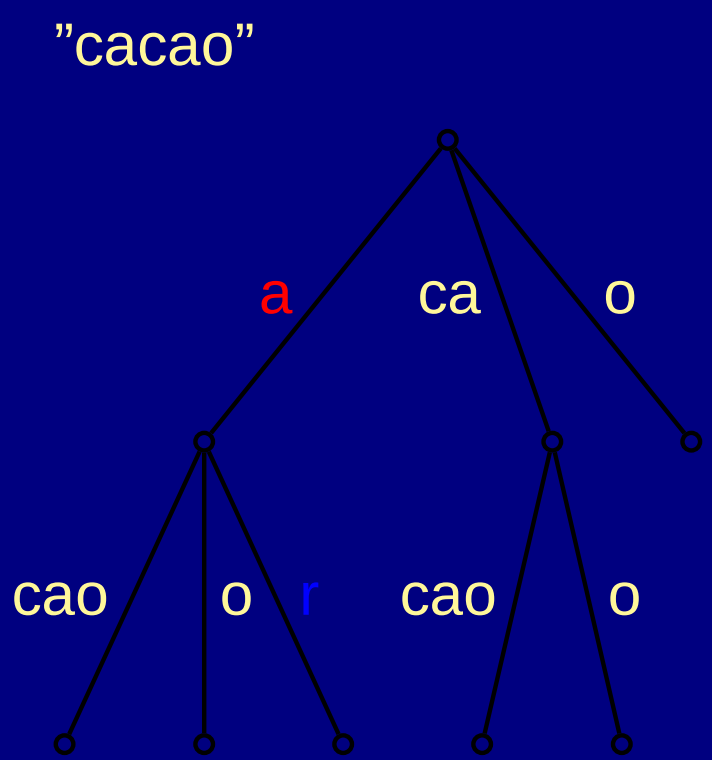
On insert $w = vu$, v est le plus long prefixe déjà présent dans l'arbre:

- u est vide $\Rightarrow$ rien à faire
- v mène à une feuille $\Rightarrow$ extension
- création de noeud:
  - v mène à un noeud
  - v mène à un arc

Exemples

$v = on$, $v = o$, $u = n$

3 cas lors de l'insertion



Insertion d'un mot dans l'arbre

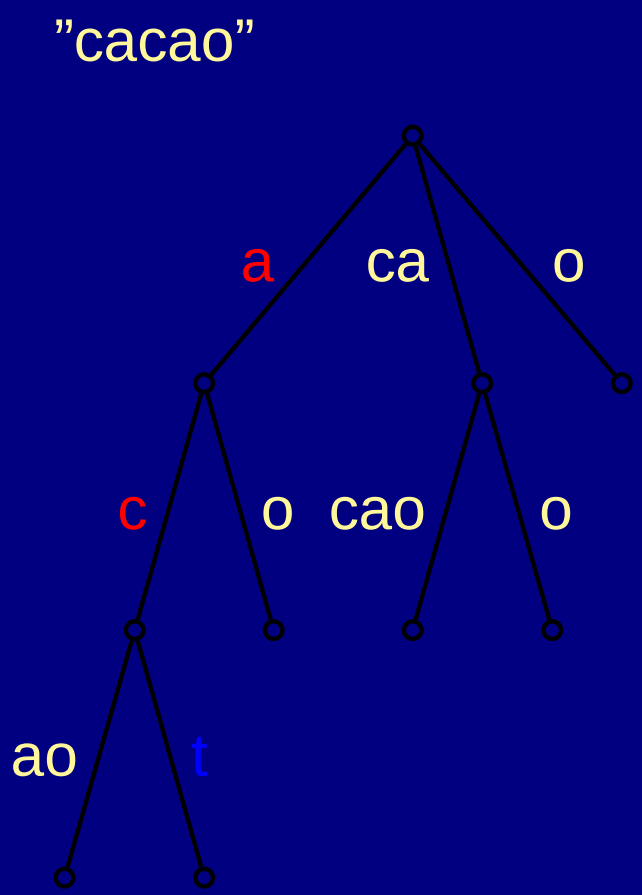
On insert $w = vu$, v est le plus long prefixe déjà présent dans l'arbre:

- u est vide $\Rightarrow$ rien à faire
- v mène à une feuille $\Rightarrow$ extension
- création de noeud:
  - v mène à un noeud
  - v mène à un arc

Exemples

$v = ar$, $v = a$, $u = r$

3 cas lors de l'insertion



Insertion d'un mot dans l'arbre

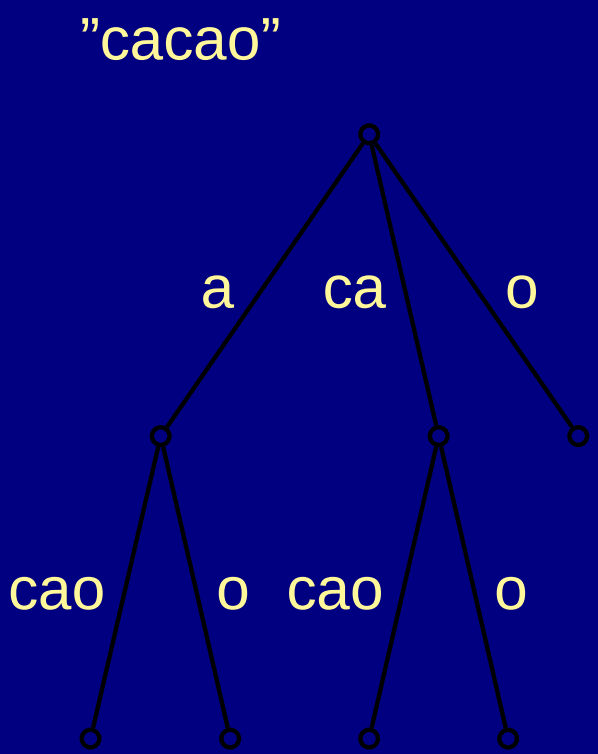
On insert $w = vu$, v est le plus long prefixe déjà présent dans l'arbre:

- u est vide $\Rightarrow$ rien à faire
- v mène à une feuille $\Rightarrow$ extension
- création de noeud:
  - v mène à un noeud
  - v mène à un arc

Exemples

$v = act$, $v = ac$, $u = t$

Les liens suffixes

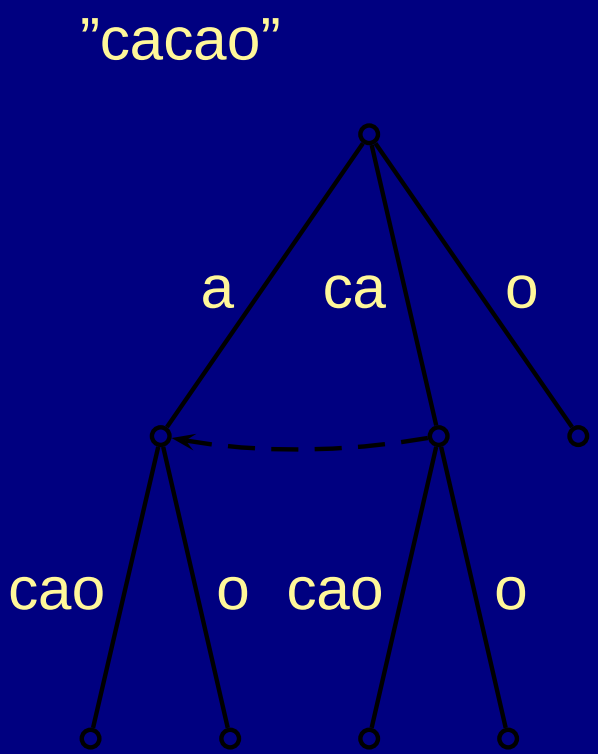


Lien suffixes

Liens entre les noeuds de l’arbre:
on note $sl(a)$ le lien suffixe de a
on note $p(a)$ le mot eppelé depuis la racine à a
$p(sl(a)) = p(a)_{1...}$
Exemple:

- $p(a) = ca, p(sl(a)) = a$
- $p(a) = acao, p(sl(a)) = cao$

Les liens suffixes

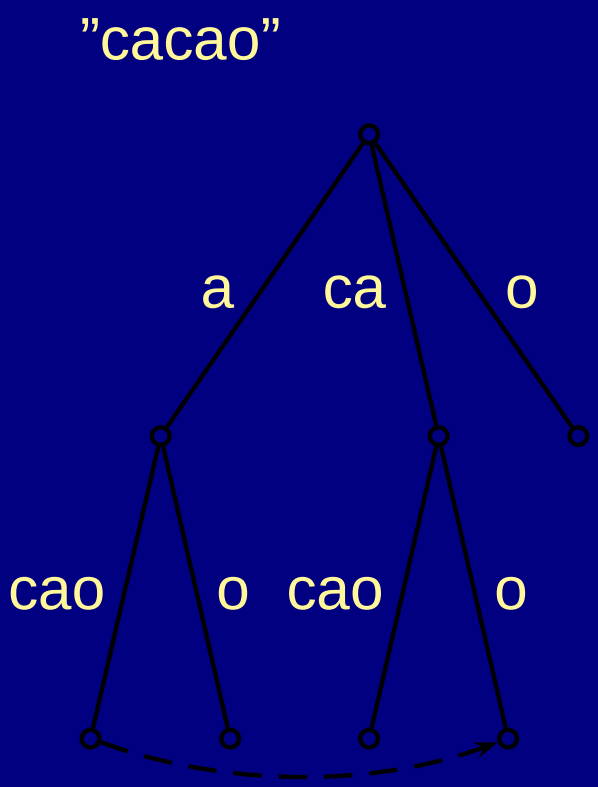


Lien suffixes

Liens entre les noeuds de l'arbre:
on note $sl(a)$ le lien suffixe de a
on note $p(a)$ le mot eppelé depuis la racine à a
$p(sl(a)) = p(a)_{1...}$
Exemple:

- $p(a) = ca, p(sl(a)) = a$
- $p(a) = acao, p(sl(a)) = cao$

Les liens suffixes

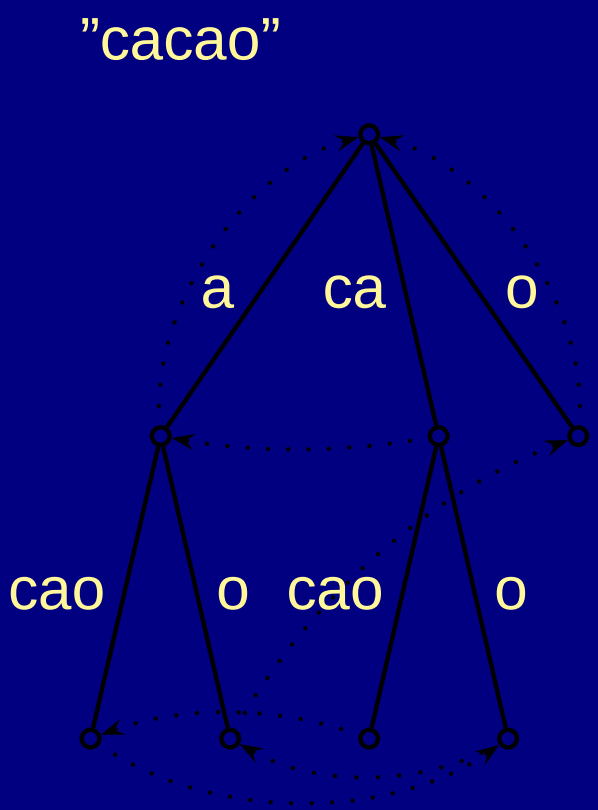


Lien suffixes

Liens entre les noeuds de l'arbre:
on note $sl(a)$ le lien suffixe de a
on note $p(a)$ le mot eppelé depuis la racine à a
$p(sl(a)) = p(a)_{1...}$
Exemple:

- $p(a) = ca, p(sl(a)) = a$
- $p(a) = acao, p(sl(a)) = cao$

Les liens suffixes



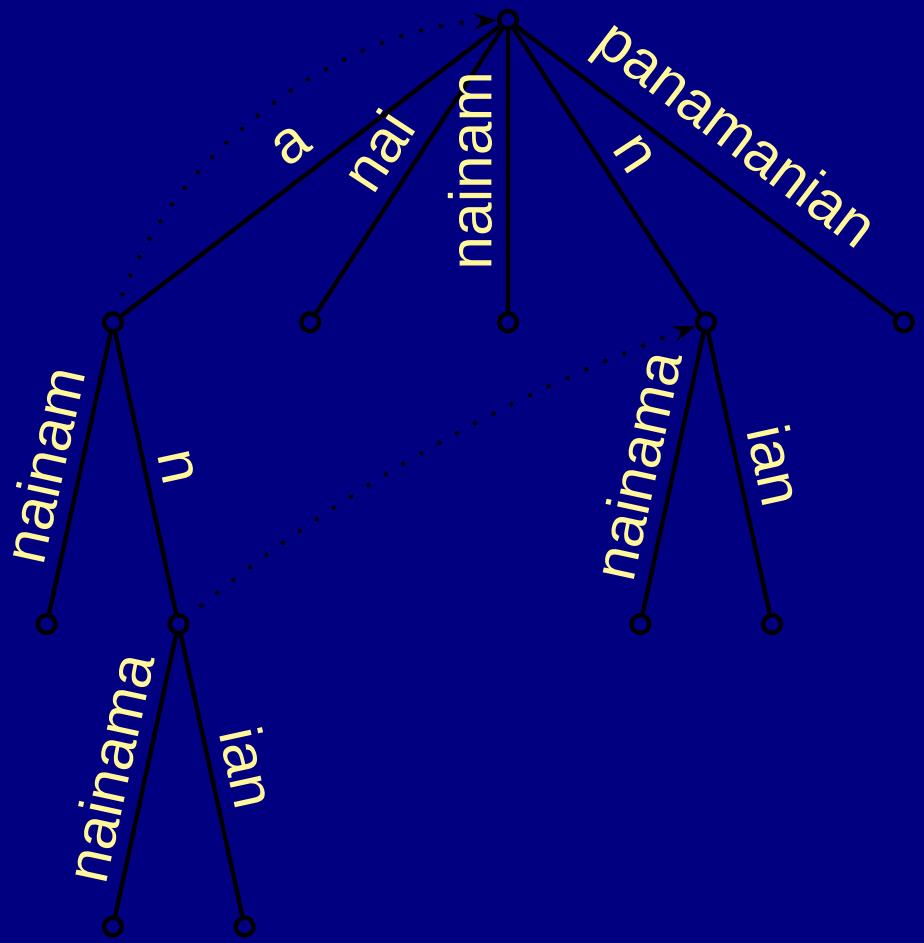
Lien suffixes

Liens entre les noeuds de l’arbre:
on note $sl(a)$ le lien suffixe de a
on note $p(a)$ le mot eppelé depuis la racine à a
$p(sl(a)) = p(a)_{1...}$
Exemple:

- $p(a) = ca, p(sl(a)) = a$
- $p(a) = acao, p(sl(a)) = cao$

Insertion rapide

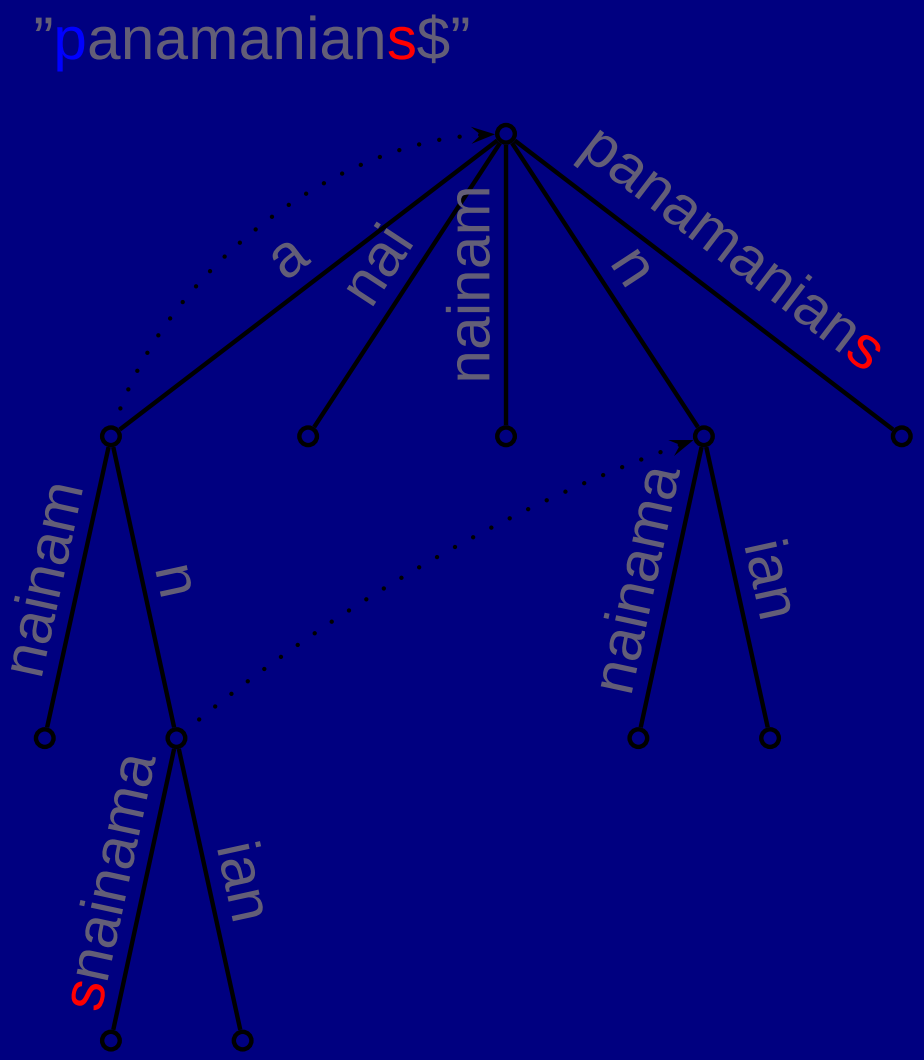
”panamanians$”



Insertion rapide

- Nous avons l’arbre de ”panamanian”
- Insertion des suffixes de ”panamanians”
- on ajoute ”panamanians” et ”anamanians”
- insertion rapide de ”namanians” (saut)
- L’insertion prend un temps proportionnel au nombre de noeuds traversés.

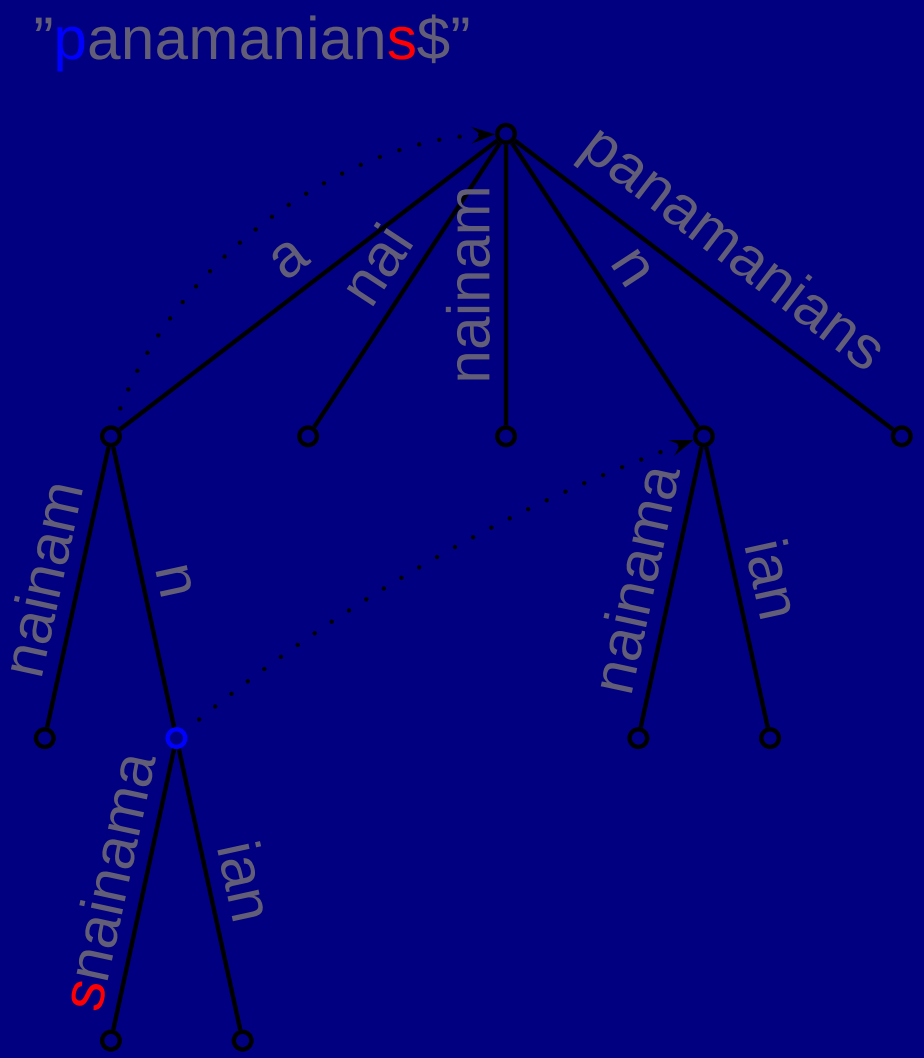
Insertion rapide



Insertion rapide

- Nous avons l’arbre de ”panamanian”
- Insertion des suffixes de ”panamanians”
- on ajoute ”panamanians” et ”anamanians”
- insertion rapide de ”namanians” (saut)
- L’insertion prend un temps proportionnel au nombre de noeuds traversés.

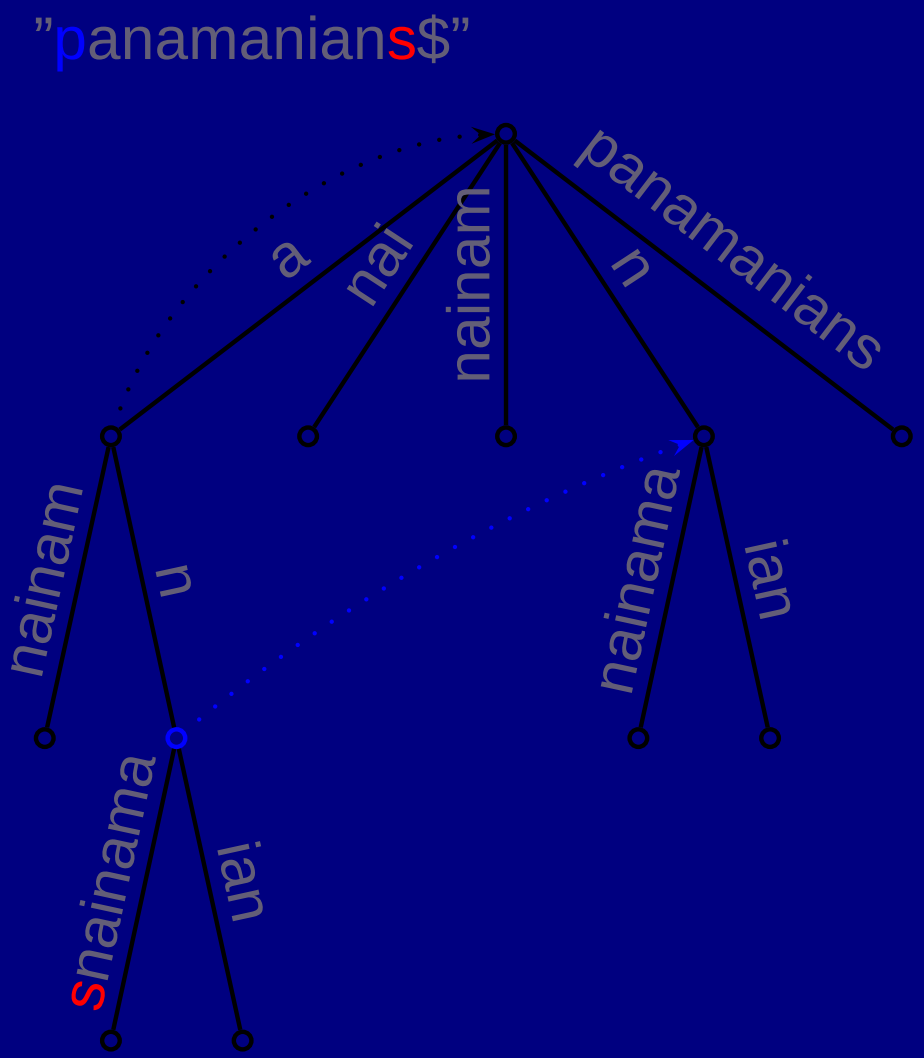
Insertion rapide



Insertion rapide

- Nous avons l’arbre de ”panamanian”
- Insertion des suffixes de ”panamanians”
- on ajoute ”panamanians” et ”anamanians”
- insertion rapide de ”namanians” (saut)
- L’insertion prend un temps proportionnel au nombre de noeuds traversés.

Insertion rapide

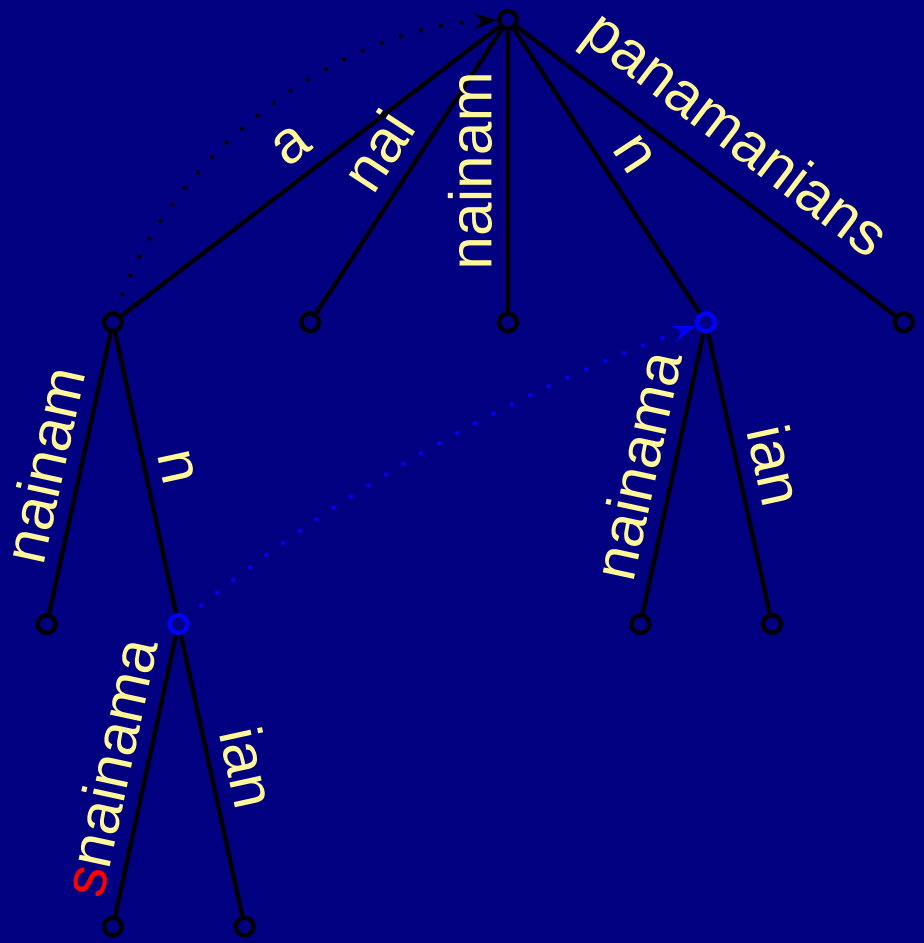


Insertion rapide

- Nous avons l’arbre de ”panamanian”
- Insertion des suffixes de ”panamanians”
- on ajoute ”panamanians” et ”anamanians”
- insertion rapide de ”namanians” (saut)
- L’insertion prend un temps proportionnel au nombre de noeuds traversés.

Insertion rapide

”panamanians$”

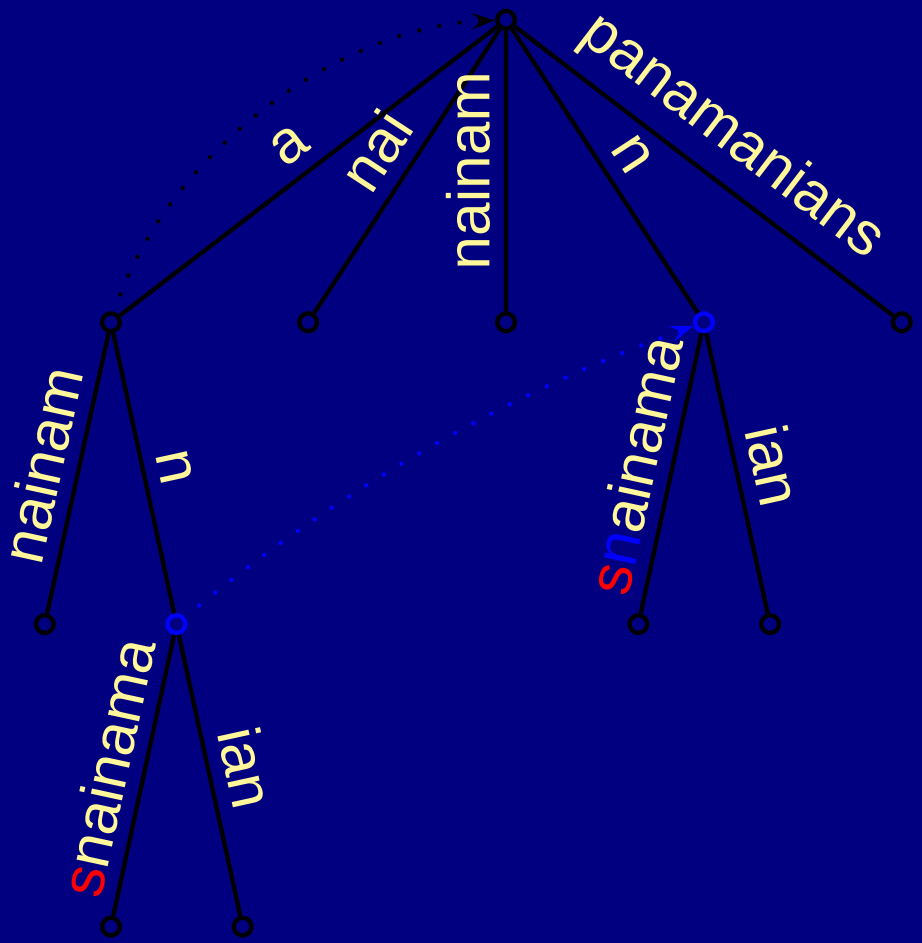


Insertion rapide

- Nous avons l’arbre de ”panamanian”
- Insertion des suffixes de ”panamanians”
- on ajoute ”panamanians” et ”anamanians”
- insertion rapide de ”namanians” (saut)
- L’insertion prend un temps proportionnel au nombre de noeuds traversés.

Insertion rapide

”panamanians$”

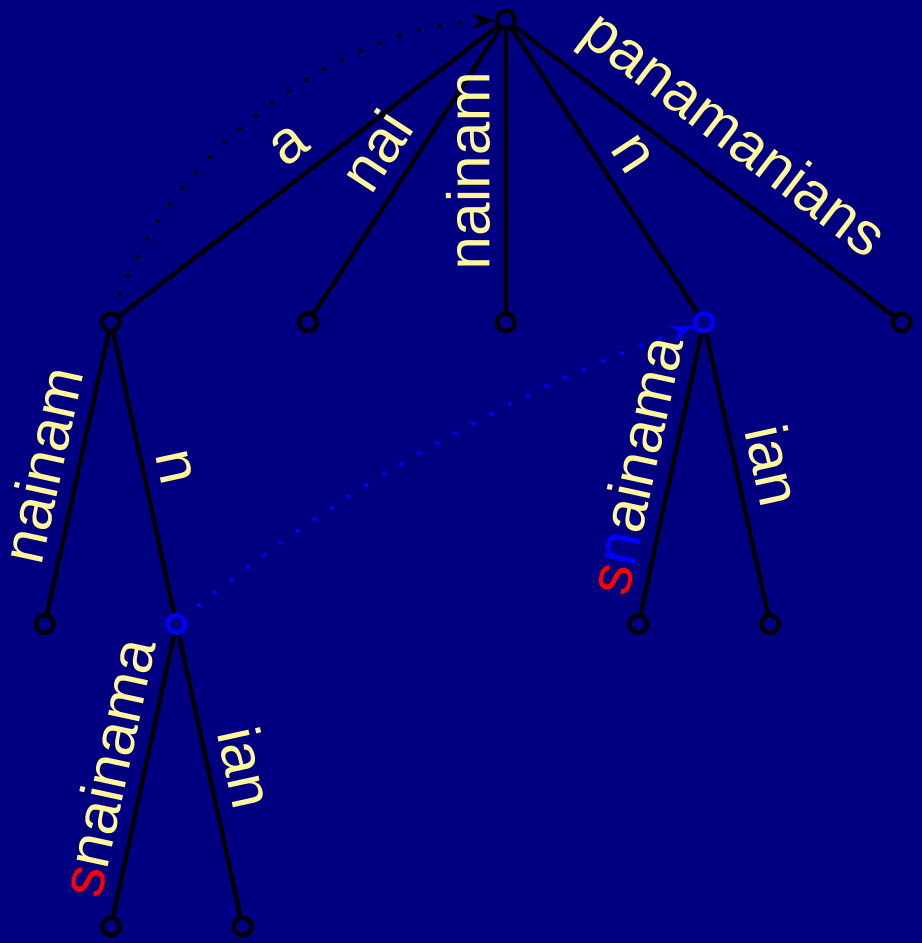


Insertion rapide

- Nous avons l’arbre de ”panamanian”
- Insertion des suffixes de ”panamanians”
- on ajoute ”panamanians” et ”anamanians”
- insertion rapide de ”namanians” (saut)
- L’insertion prend un temps proportionnel au nombre de noeuds traversés.

Insertion rapide

”panamanians$”

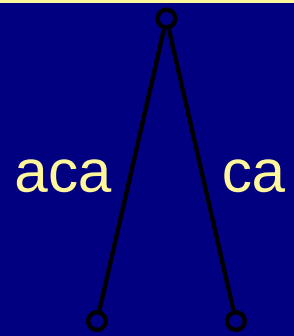


Insertion rapide

- Nous avons l’arbre de ”panamanian”
- Insertion des suffixes de ”panamanians”
- on ajoute ”panamanians” et ”anamanians”
- insertion rapide de ”namanians” (saut)
- L’insertion prend un temps proportionnel au nombre de noeuds traversés.

Extension automatique

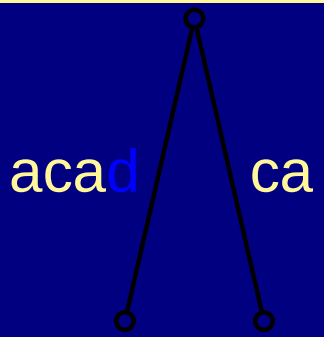
a	c	a	d	e	m	y
0	1	2	3	4	5	6
		i-1				



Extension automatique

Extension automatique

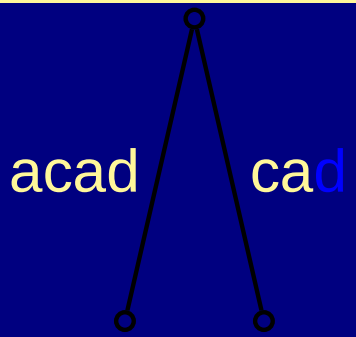
a	c	a	d	e	m	y
0	1	2	3	4	5	6
			i			



Extension automatique

Extension automatique

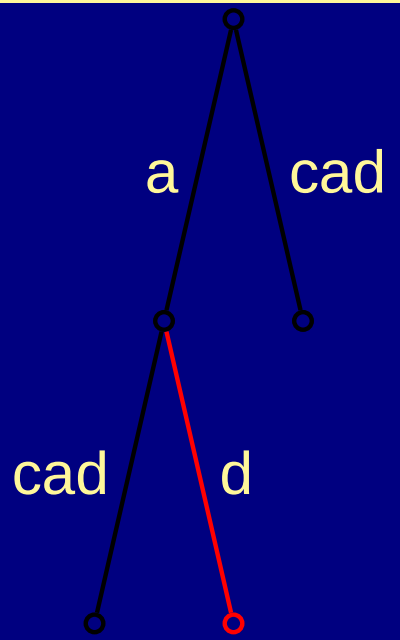
a	c	a	d	e	m	y
0	1	2	3	4	5	6
			i			



Extension automatique

Extension automatique

a	c	a	d	e	m	y
0	1	2	3	4	5	6
		j	i			

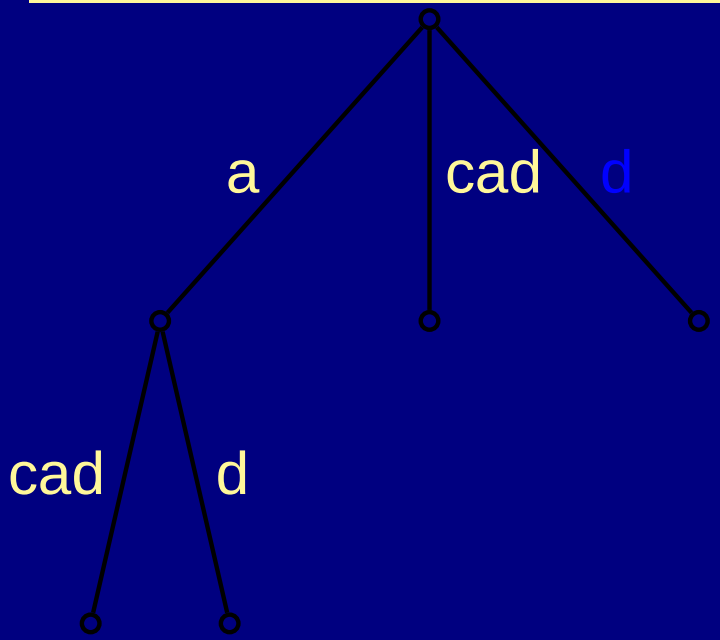


Extension automatique

- À la phase i, étape j on a inséré $t_{j\ldots i}$ et créé une feuille.
- En phase $i+1$, l'insertion de $t_{j\ldots i+1}$ mène à . . .
- une extension
- l'indice de fin passe de i à $i+1$
- $\Rightarrow$ l'indice de fin est égale à l'indice de la phase en court

Extension automatique

a	c	a	d	e	m	y
0	1	2	3	4	5	6
		j	i			

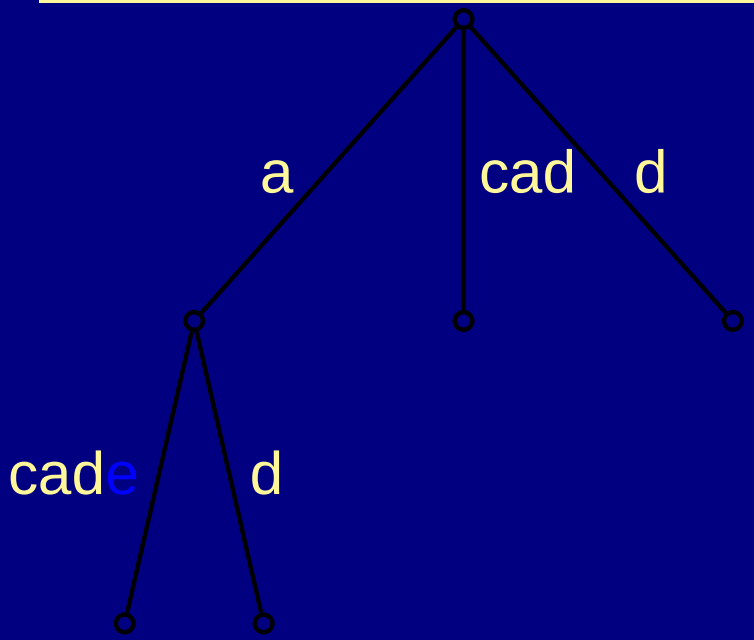


Extension automatique

- À la phase i, étape j on a inséré $t_{j\ldots i}$ et créé une feuille.
- En phase $i+1$, l'insertion de $t_{j\ldots i+1}$ mène à . . .
- une extension
- l'indice de fin passe de i à $i+1$
- $\Rightarrow$ l'indice de fin est égale à l'indice de la phase en court

Extension automatique

a	c	a	d	e	m	y
0	1	2	3	4	5	6
		j	i	i+1		

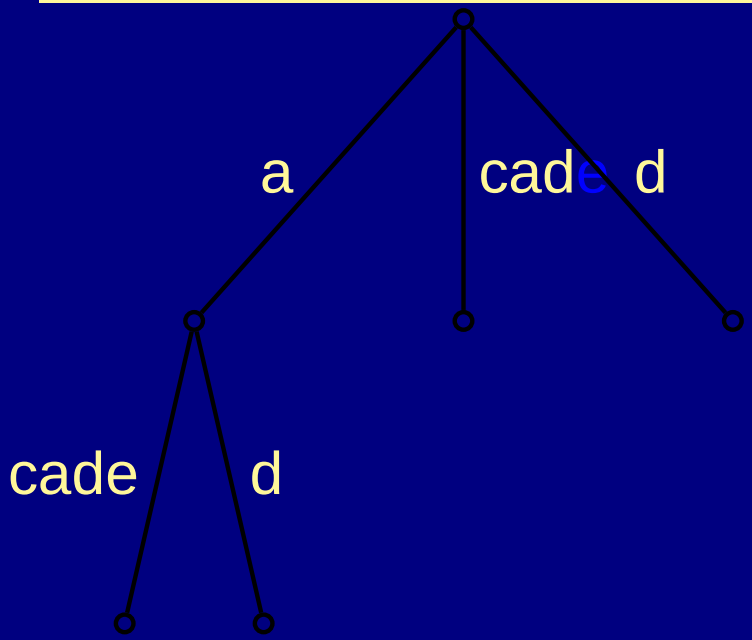


Extension automatique

- À la phase i, étape j on a inséré $t_{j\ldots i}$ et créé une feuille.
- En phase $i+1$, l'insertion de $t_{j\ldots i+1}$ mène à . . .
- une extension
- l'indice de fin passe de i à $i+1$
- $\Rightarrow$ l'indice de fin est égale à l'indice de la phase en court

Extension automatique

a	c	a	d	e	m	y
0	1	2	3	4	5	6
		j	i	i+1		

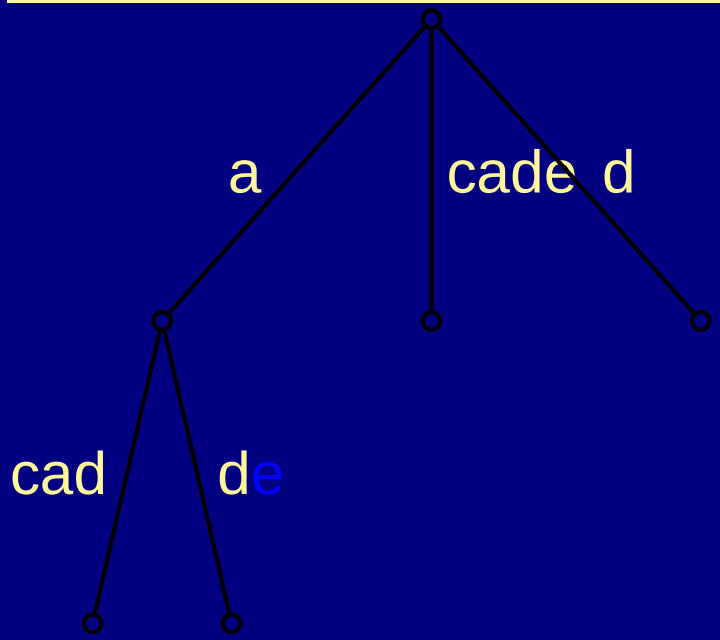


Extension automatique

- À la phase i, étape j on a inséré $t_{j\ldots i}$ et créé une feuille.
- En phase $i+1$, l'insertion de $t_{j\ldots i+1}$ mène à ...
- une extension
- l'indice de fin passe de i à $i+1$
- $\Rightarrow$ l'indice de fin est égale à l'indice de la phase en court

Extension automatique

a	c	a	d	e	m	y
0	1	2	3	4	5	6
		j	i	i+1		

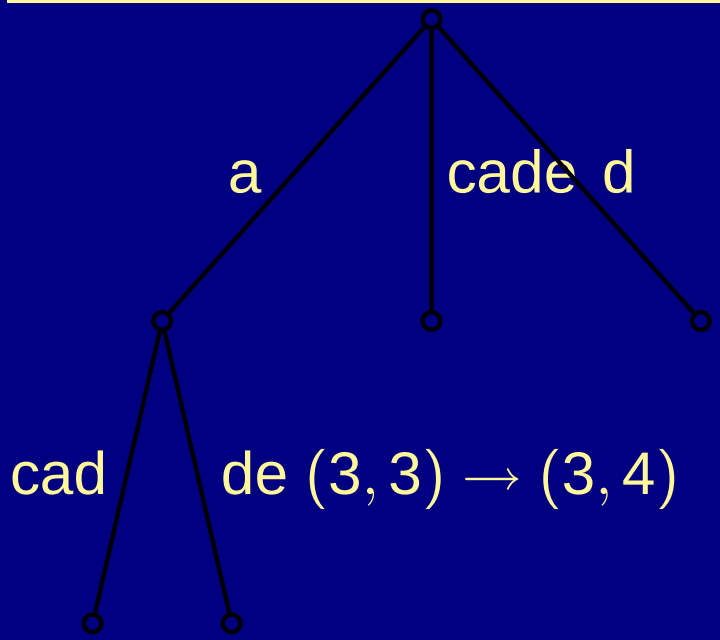


Extension automatique

- À la phase i, étape j on a inséré $t_{j\ldots i}$ et créé une feuille.
- En phase $i+1$, l'insertion de $t_{j\ldots i+1}$ mène à ...
- une extension
- l'indice de fin passe de i à $i+1$
- $\Rightarrow$ l'indice de fin est égale à l'indice de la phase en court

Extension automatique

a	c	a	d	e	m	y
0	1	2	3	4	5	6
		j	i	i+1		

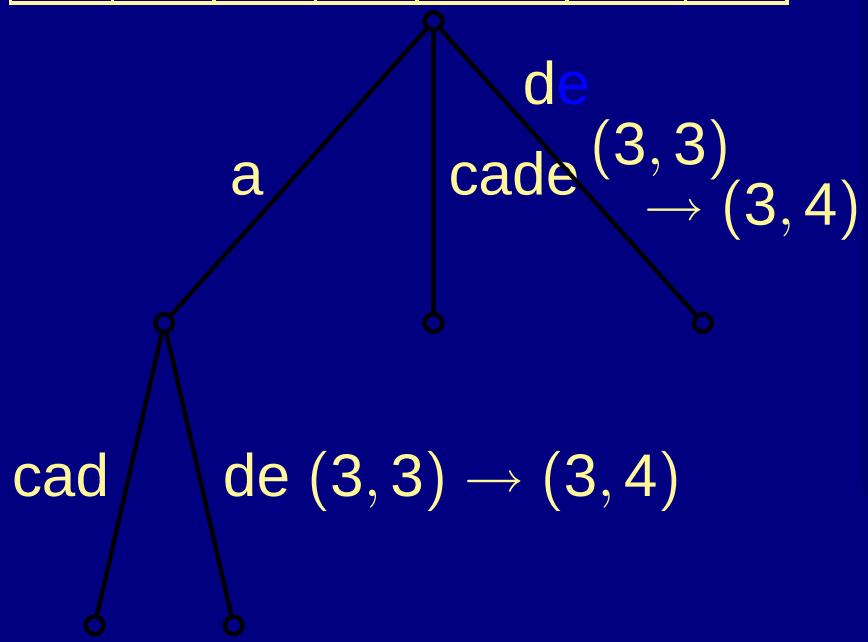


Extension automatique

- À la phase i, étape j on a inséré $t_{j\ldots i}$ et créé une feuille.
- En phase $i+1$, l'insertion de $t_{j\ldots i+1}$ mène à ...
- une extension
- l'indice de fin passe de i à $i+1$
- $\Rightarrow$ l'indice de fin est égale à l'indice de la phase en court

Extension automatique

a	c	a	d	e	m	y
0	1	2	3	4	5	6
		j	i	i+1		

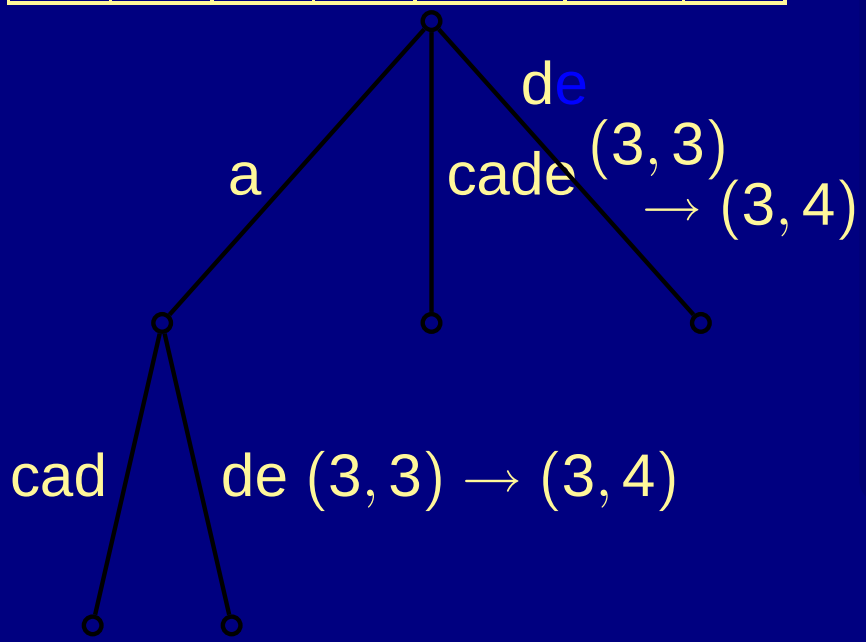


Extension automatique

- À la phase i, étape j on a inséré $t_{j\ldots i}$ et créé une feuille.
- En phase $i+1$, l'insertion de $t_{j\ldots i+1}$ mène à ...
- une extension
- l'indice de fin passe de i à $i+1$
- $\Rightarrow$ l'indice de fin est égale à l'indice de la phase en court

Extension automatique

a	c	a	d	e	m	y
0	1	2	3	4	5	6
		j	i	i+1		

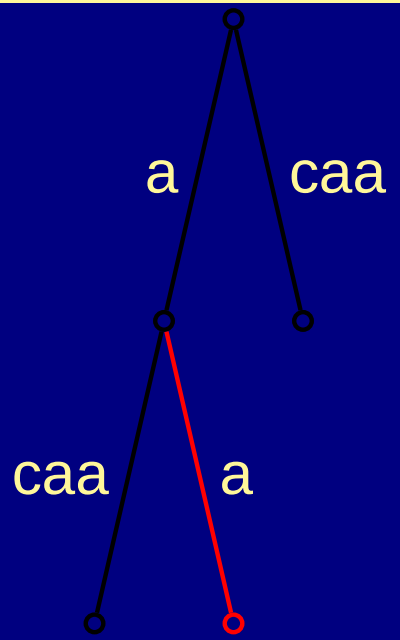


Extension automatique

- À la phase i, étape j on a inséré $t_{j...i}$ et créé une feuille.
- En phase $i + 1$, l'insertion de $t_{j...i+1}$ mène à ...
- une extension
- l'indice de fin passe de i à $i + 1$
- $\Rightarrow$ l'indice de fin est égale à l'indice de la phase en court

Extension automatique

a	c	a	a	e
0	1	2	3	4
		j	i	



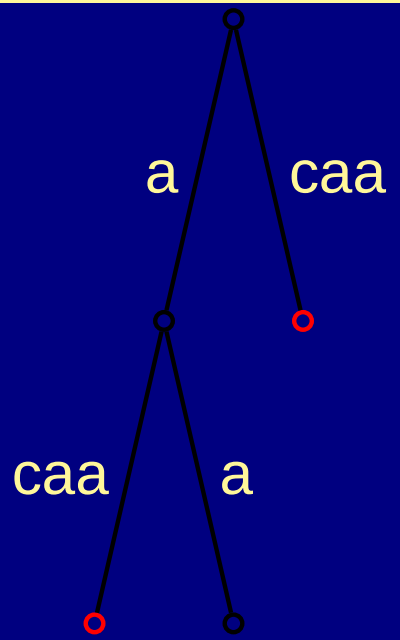
Extension automatique

Phase i:

- Si $t_{j\ldots i}$ créé une feuille
- $\forall k < j, t_{k\ldots i}$ mène à une feuille
- Suppose $t_{j+1\ldots i}$ ne crée pas de feuille

Extension automatique

a	c	a	a	e
0	1	2	3	4
k		j	i	



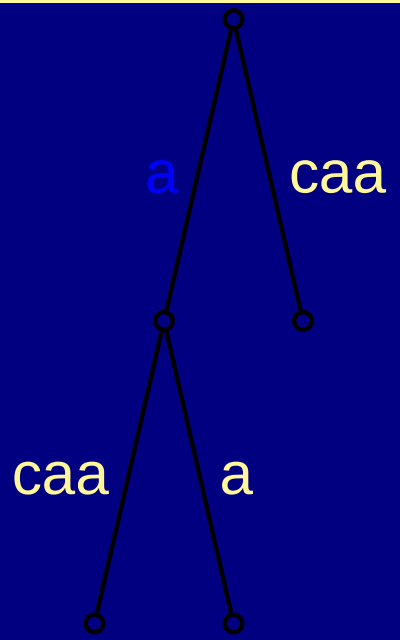
Extension automatique

Phase i:

- Si $t_{j\ldots i}$ créé une feuille
- $\forall k < j, t_{k\ldots i}$ mène à une feuille
- Suppose $t_{j+1\ldots i}$ ne crée pas de feuille

Extension automatique

a	c	a	a	e
0	1	2	3	4
		j	i	



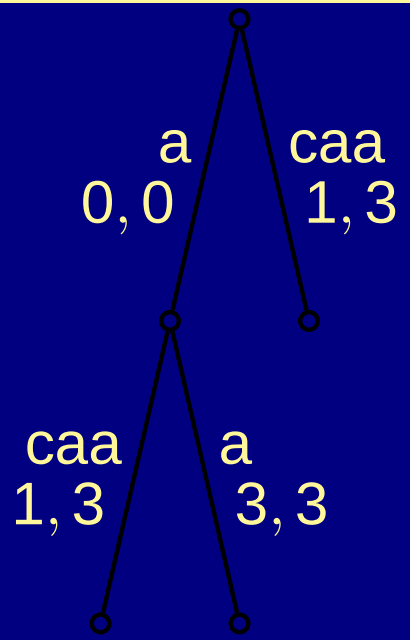
Extension automatique

Phase i:

- Si $t_{j\ldots i}$ créé une feuille
- $\forall k < j, t_{k\ldots i}$ mène à une feuille
- Suppose $t_{j+1\ldots i}$ ne crée pas de feuille

Extension automatique

a	c	a	a	e
0	1	2	3	4
		j	i	



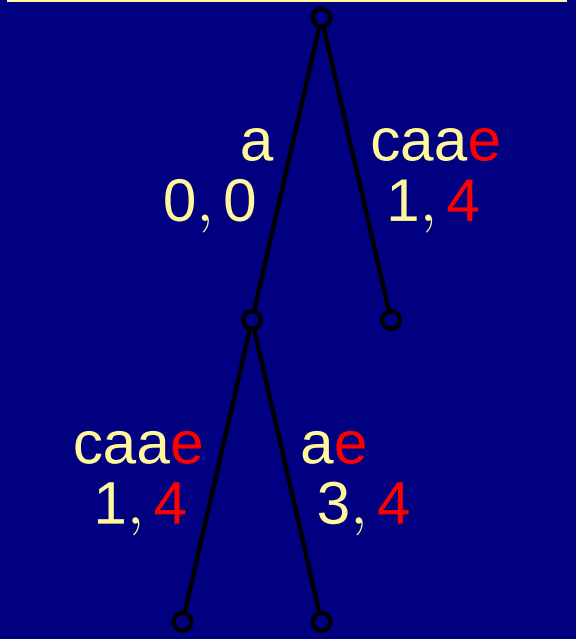
Extension automatique

Phase i:

- Si $t_{j\ldots i}$ créé une feuille
- $\forall k < j, t_{k\ldots i}$ mène à une feuille
- Suppose $t_{j+1\ldots i}$ ne crée pas de feuille

Extension automatique

a	c	a	a	e
0	1	2	3	4
		j		i+1



Extension automatique

Phase i:

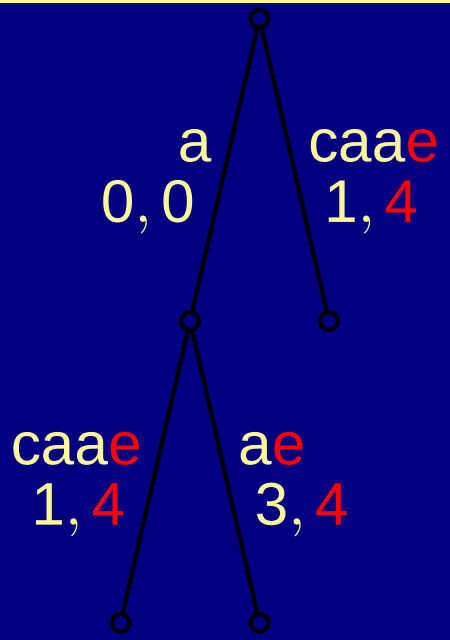
- Si $t_{j\ldots i}$ créé une feuille
- $\forall k < j, t_{k\ldots i}$ mène à une feuille
- Suppose $t_{j+1\ldots i}$ ne crée pas de feuille

Phase $i + 1$:

- $\forall k \leq j, t_{k\ldots i+1}$ déjà insérés
- Reprise à $t_{j+1\ldots i+1}$
- insertion **rapide** depuis la dernière feuille

Extension automatique

a	c	a	a	e
0	1	2	3	4
k				i+1



Extension automatique

Phase i:

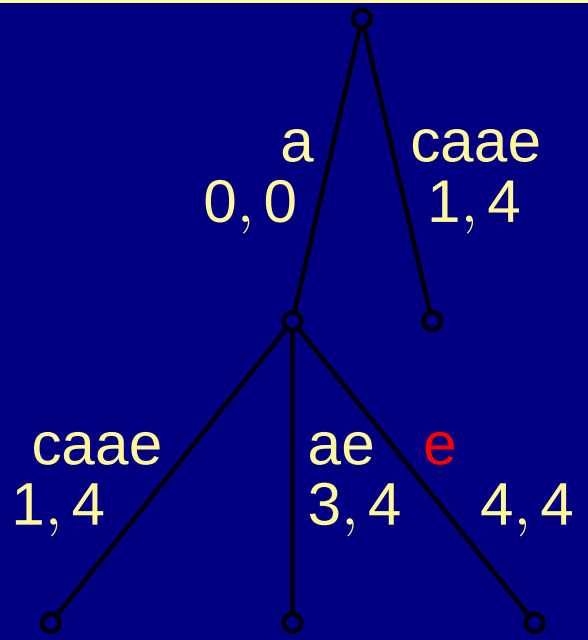
- Si $t_{j\ldots i}$ créé une feuille
- $\forall k < j, t_{k\ldots i}$ mène à une feuille
- Suppose $t_{j+1\ldots i}$ ne crée pas de feuille

Phase $i + 1$:

- $\forall k \leq j, t_{k\ldots i+1}$ déjà insérés
- Reprise à $t_{j+1\ldots i+1}$
- insertion **rapide** depuis la dernière feuille

Extension automatique

a	c	a	a	e
0	1	2	3	4
....			j+1	i+1



Extension automatique

Phase i:

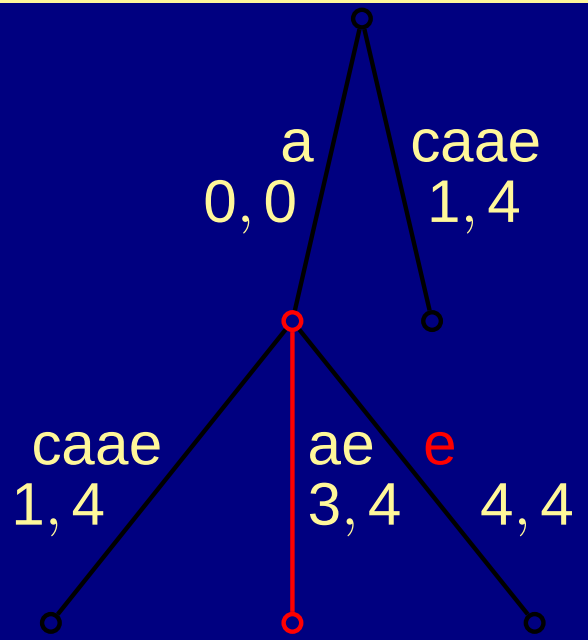
- Si $t_{j\ldots i}$ créé une feuille
- $\forall k < j, t_{k\ldots i}$ mène à une feuille
- Suppose $t_{j+1\ldots i}$ ne crée pas de feuille

Phase $i + 1$:

- $\forall k \leq j, t_{k\ldots i+1}$ déjà insérés
- Reprise à $t_{j+1\ldots i+1}$
- insertion **rapide** depuis la dernière feuille

Extension automatique

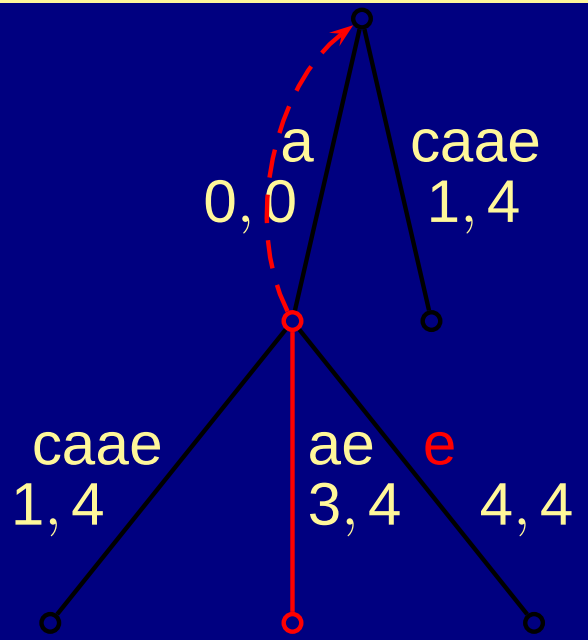
a	c	a	a	e
0	1	2	3	4
....			j+1	i+1



Extension automatique

Extension automatique

a	c	a	a	e
0	1	2	3	4
....			j+1	i+1



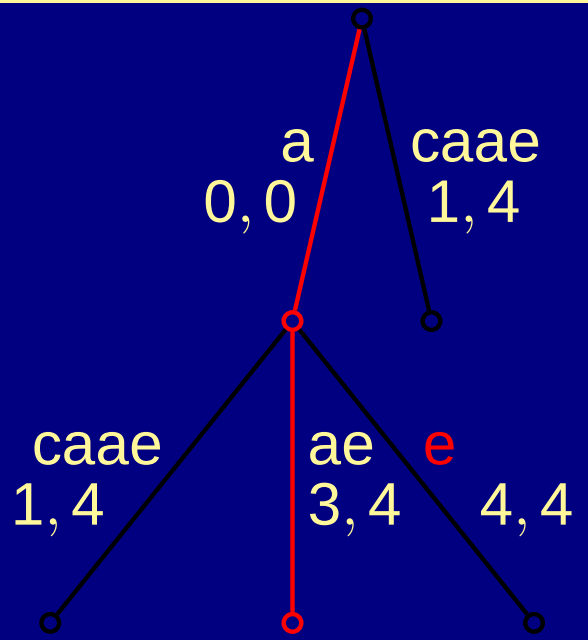
Extension automatique

Phase i:

- Si $t_{j...i}$ créé une feuille
- $\forall k < j, t_k$

Extension automatique

a	c	a	a	e
0	1	2	3	4
....			j+1	i+1



Exemple complet

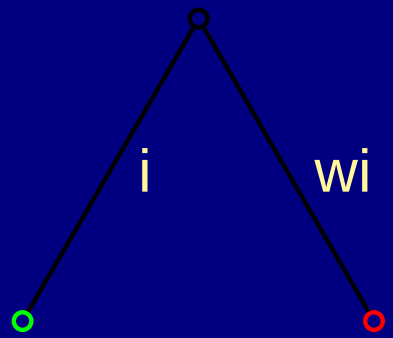
text:	w	i	n	i	n	g	$
phase:							
step:							

Exemple complet

text:	w	i	n	i	n	g
phase:	i					
step:	j					

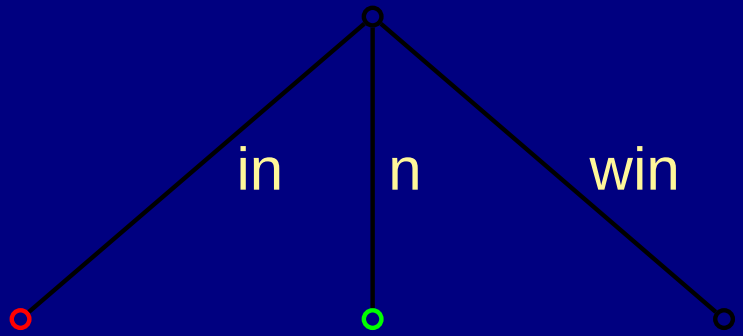
Exemple complet

text:	w	i	n	i	n	g
phase:		i				
step:		j				



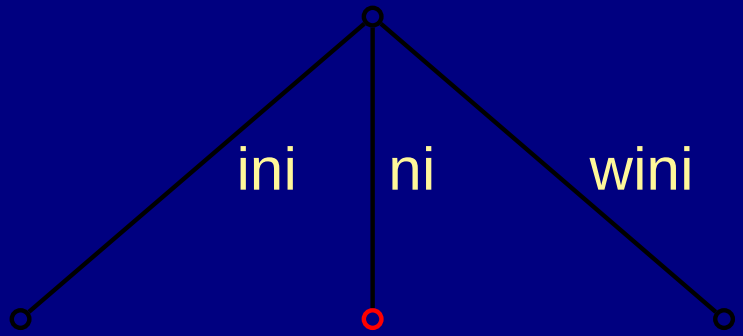
Exemple complet

text:	w	i	n	i	n	g
phase:			i			
step:			j			



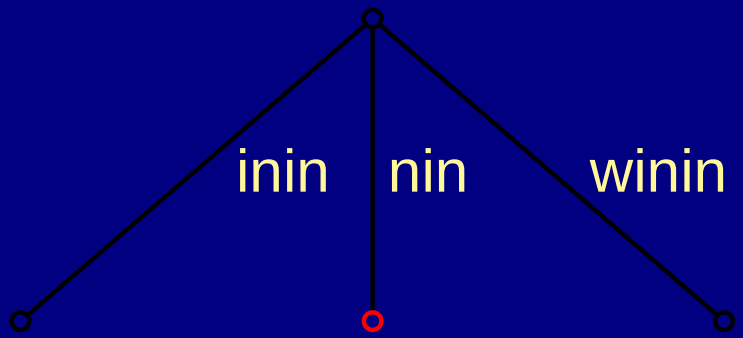
Exemple complet

text:	w	i	n	i	n	g
phase:				i		
step:			j			



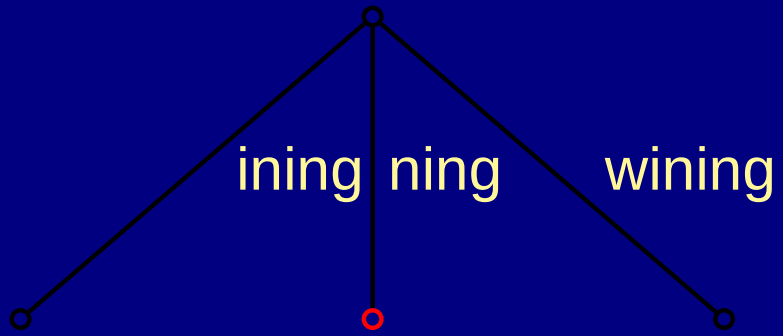
Exemple complet

text:	w	i	n	i	n	g
phase:					i	
step:			j			



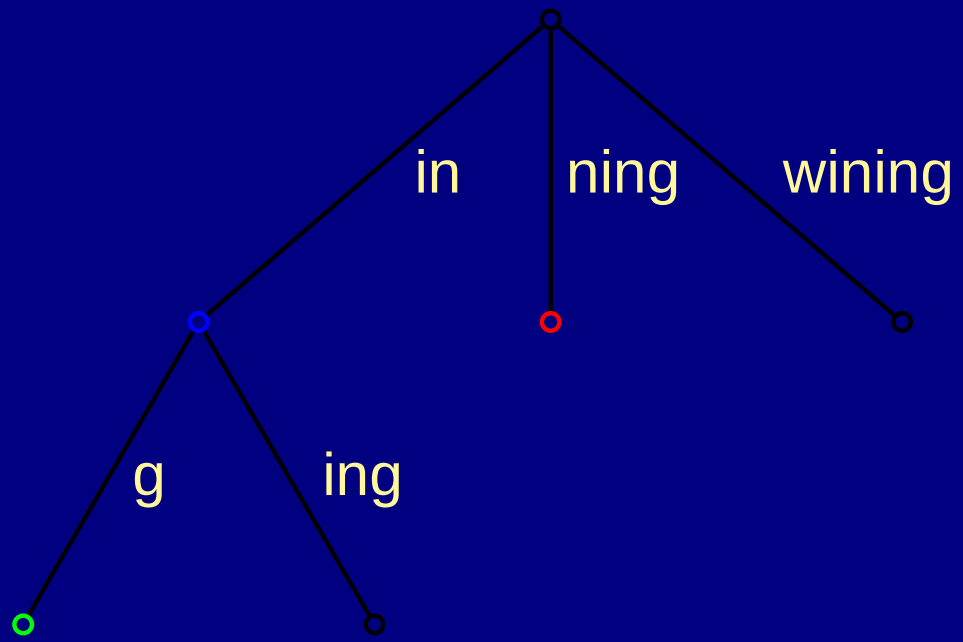
Exemple complet

text:	w	i	n	i	n	g
phase:						i
step:			j			



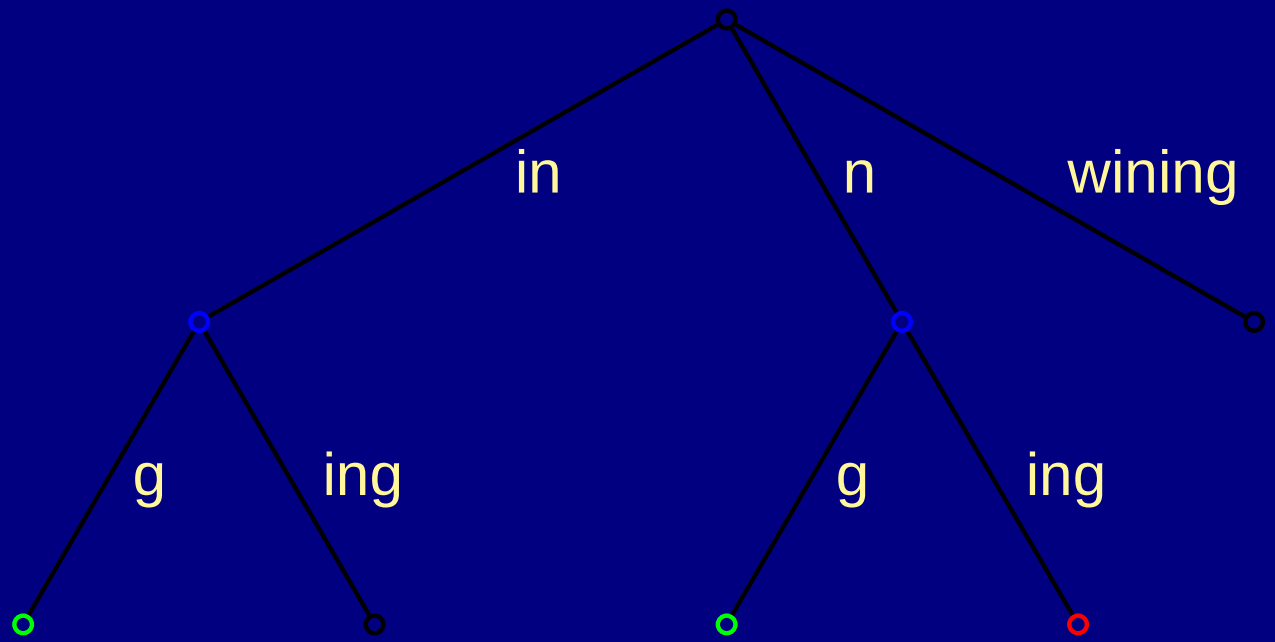
Exemple complet

text:	w	i	n	i	n	g
phase:						i
step:				j		



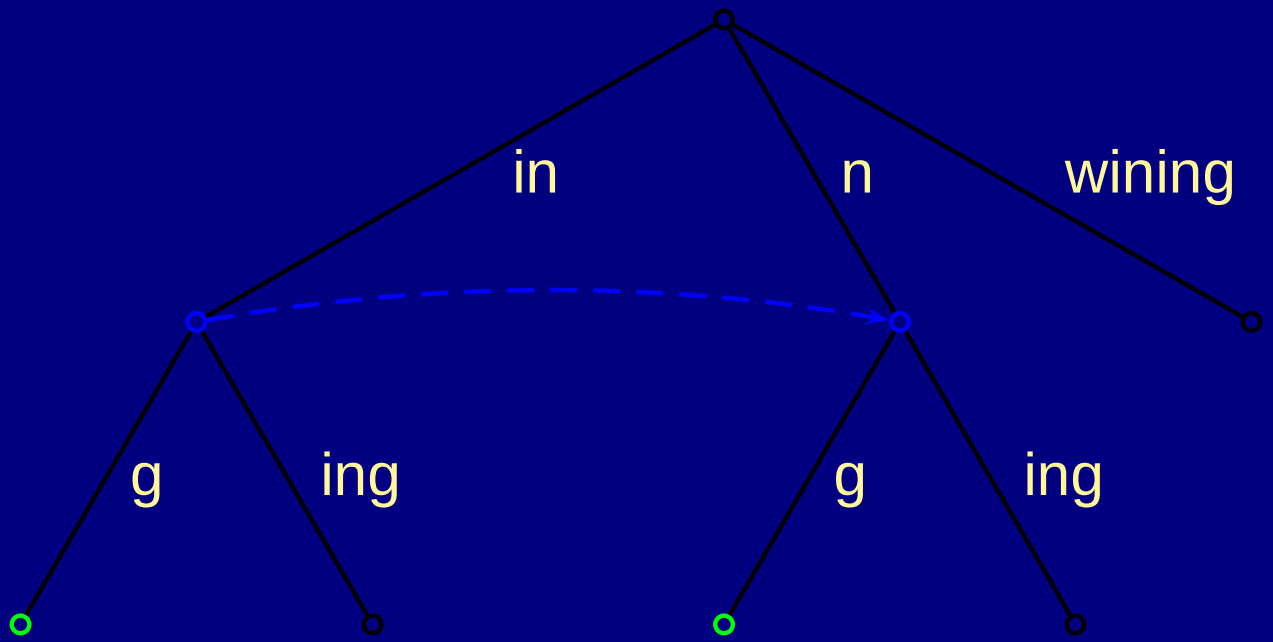
Exemple complet

text:	w	i	n	i	n	g
phase:						i
step:					j	



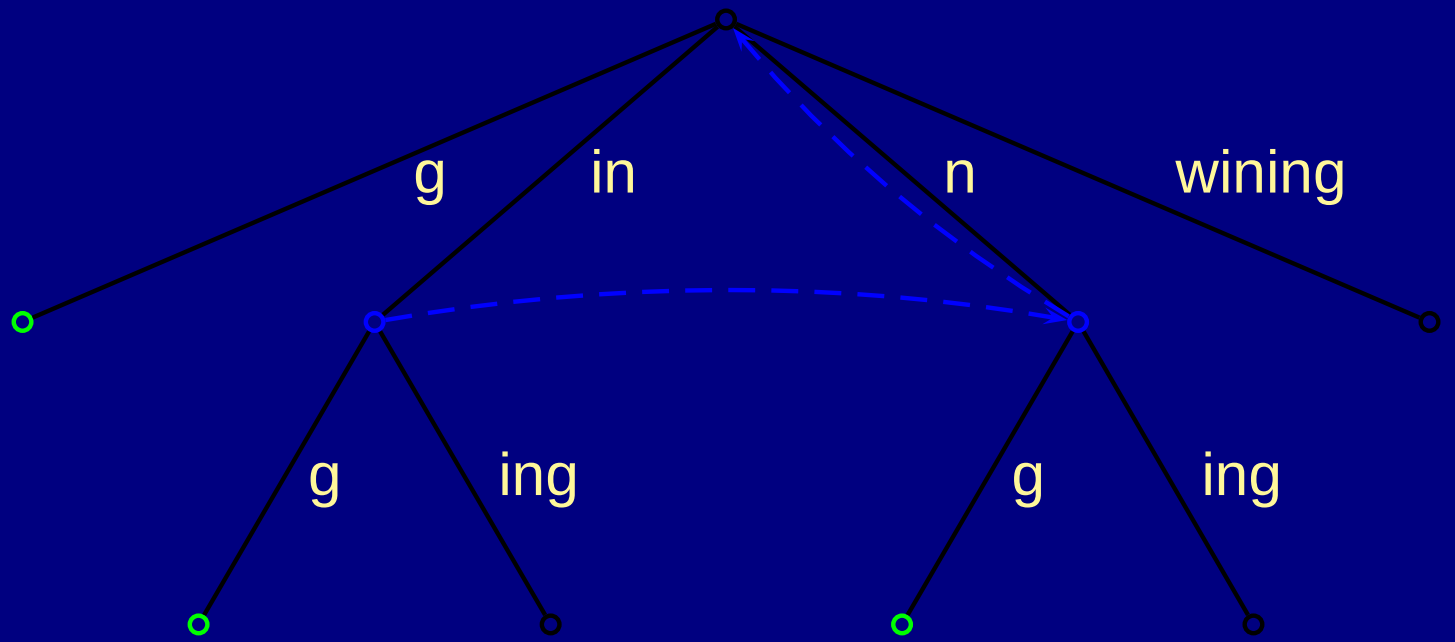
Exemple complet

text:	w	i	n	i	n	g
phase:						i
step:					j	



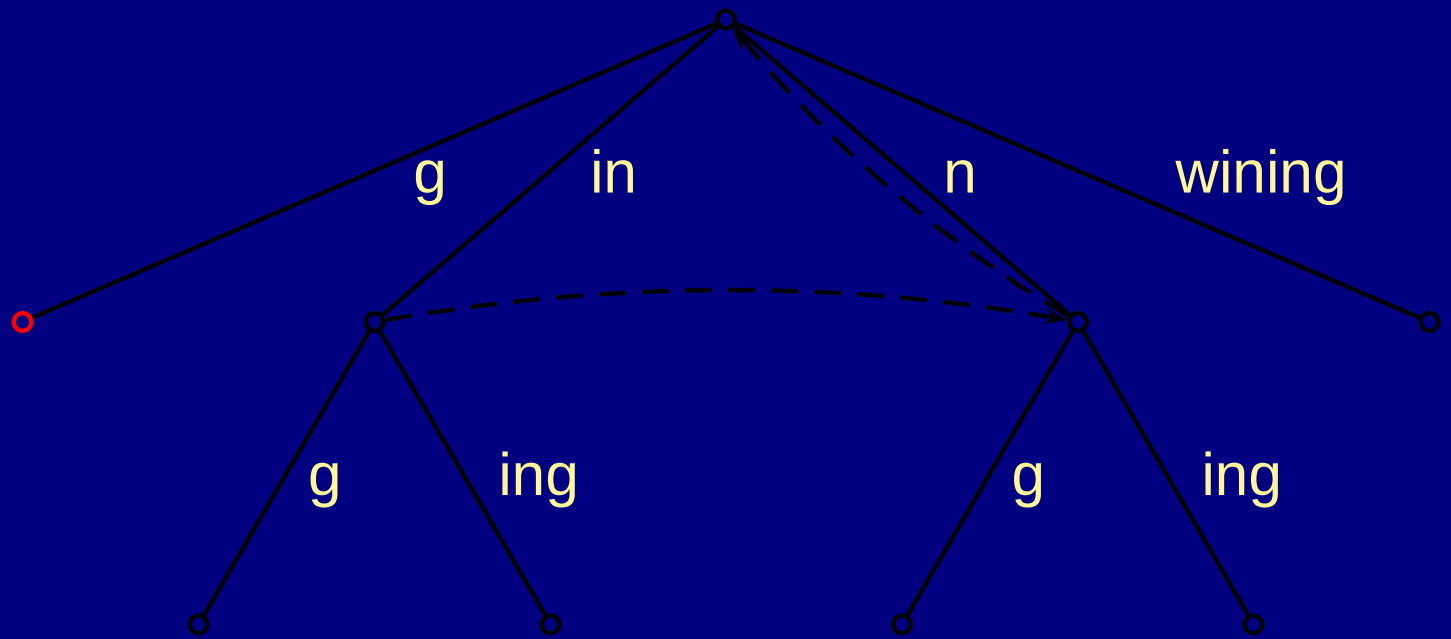
Exemple complet

text:	w	i	n	i	n	g
phase:						i
step:						j



Exemple complet

text:	w	i	n	i	n	g	x
phase:							i
step:						j	



Complexité en temp

Lemme

Soit $depth(N)$ le nombre noeud entre N et la racine.
Pour chaque noeud N de l'arbre des suffixes:

$$depth(N) \leq depth(sl(N)) + 1$$

Complexité en temp

Lemme

$depth(N) \leq depth(sl(N)) + 1$

Complexité

- Pour chaque phase, l'extension automatique prend un temps constant, toutes ces insertions coûtent $O(n)$
- Soit j l'indice de l'insertion explicite. j ne décrémente jamais mais peut rester inchanger entre deux phases $\Rightarrow O(2 * n)$ insertions.
- insertion: La profondeur en cours dans l'arbre reponte d'au plus 2 et puis redescend.
- La profondeur max. est n, $\Rightarrow O(n)$ noeuds traversés.

Complexité en temp

Lemme

Complexité en temp

Lemme

$depth(N) \leq depth(sl(N)) + 1$

Complexité

- Pour chaque phase, l'extension automatique prend un temps constant, toutes ces insertions coûtent $O(n)$
- Soit j l'indice de l'insertion explicite. j ne décrémente jamais mais peut rester inchanger entre deux phases $\Rightarrow O(2 * n)$ insertions.
- insertion: La profondeur en cours dans l'arbre reponte d'au plus 2 et puis redescend.
- La profondeur max. est n, $\Rightarrow O(n)$ noeuds traversés.

Complexité en temp

Lemme

$depth(N) \leq depth(sl(N)) + 1$

Complexité

- Pour chaque phase, l'extension automatique prend un temps constant, toutes ces insertions coûtent $O(n)$
- Soit j l'indice de l'insertion explicite. j ne décrémente jamais mais peut rester inchanger entre deux phases $\Rightarrow O(2 * n)$ insertions.
- insertion: La profondeur en cours dans l'arbre reponte d'au plus 2 et puis redescend.
- La profondeur max. est n, $\Rightarrow O(n)$ noeuds traversés.

Extension

Generalized suffix tree

Given N sequences $s_1 \dots s_N$, the generalized suffix tree of these sequences correspond to the superposition of the suffix tree of each sequence.

This tree is built in $O(\sum_i |s_i|)$ time and its space requierement is also in $O(\sum_i |s_i|)$ ($|s_i|$ is the length of s_i).

Allow to ask the problem: ”find the longest common word of N sequences” or ”output all word that occurs in at least q sequences”.

Extension

Generalized suffix tree

Given N sequences $s_1 \dots s_N$, the generalized suffix tree of these sequences correspond to the superposition of the suffix tree of each sequence.

This tree is built in $O(\sum_i |s_i|)$ time and its space requierement is also in $O(\sum_i |s_i|)$ ($|s_i|$ is the length of s_i).

Allow to ask the problem: ”find the longest common word of N sequences” or ”output all word that occurs in at least q sequences”.

Extension

Generalized suffix tree

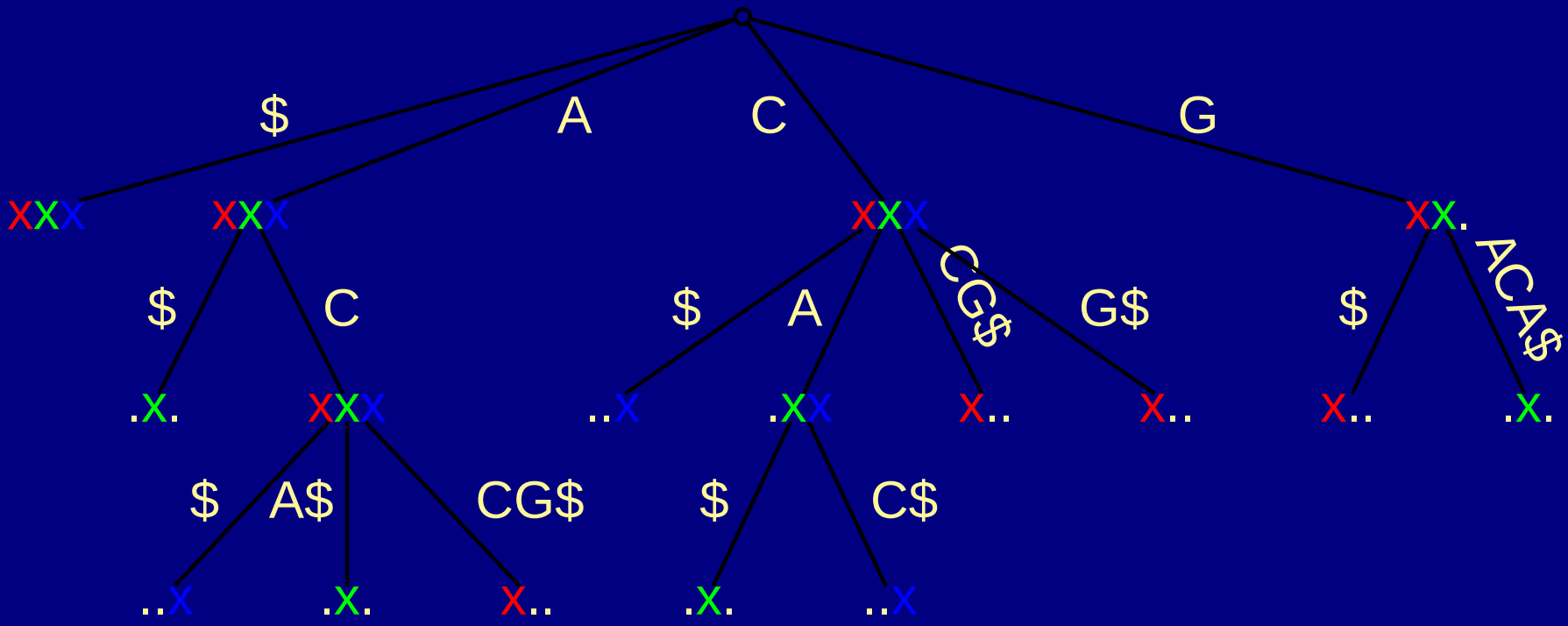
Given N sequences $s_1 \dots s_N$, the generalized suffix tree of these sequences correspond to the superposition of the suffix tree of each sequence.

This tree is built in $O(\sum_i |s_i|)$ time and its space requierement is also in $O(\sum_i |s_i|)$ ($|s_i|$ is the length of s_i).

Allow to ask the problem: ”find the longest common word of N sequences” or ”output all word that occurs in at least q sequences”.

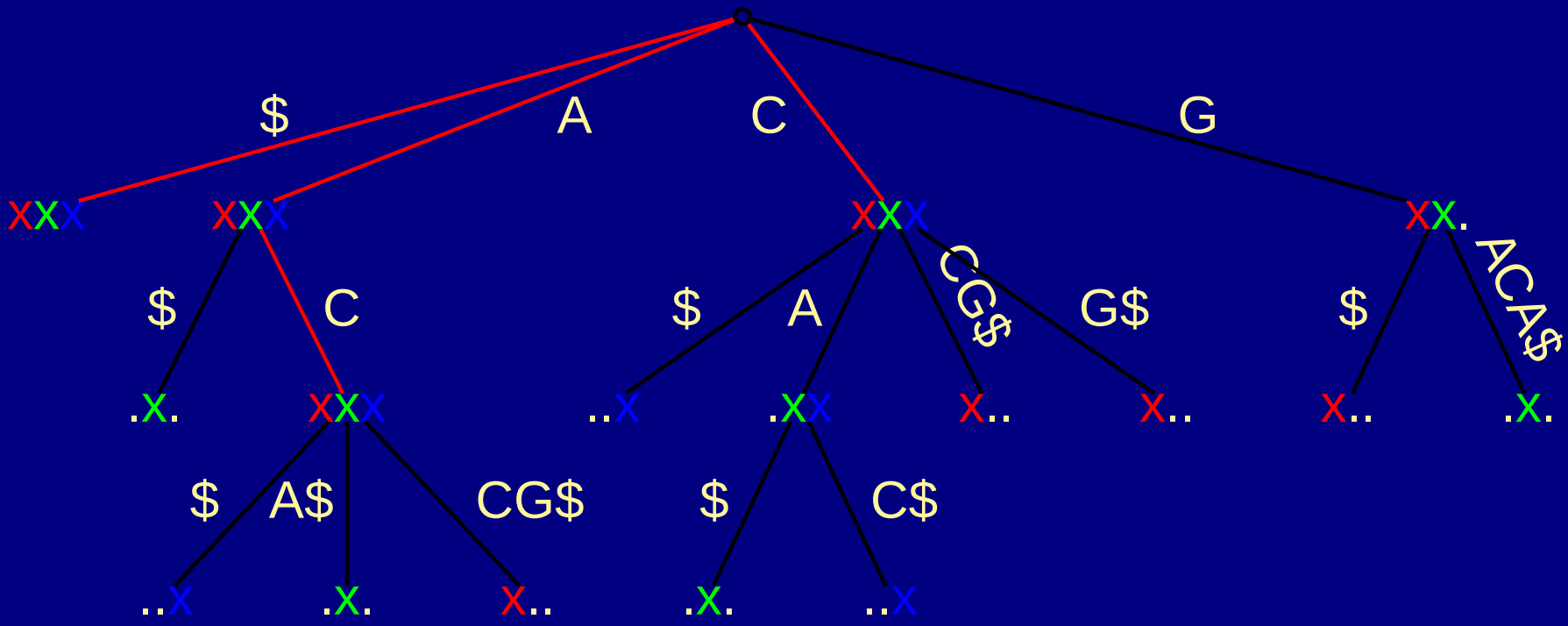
Generalized suffix tree: example

Generalized suffix tree of "ACCG$", "GACA$" and "CAC$":



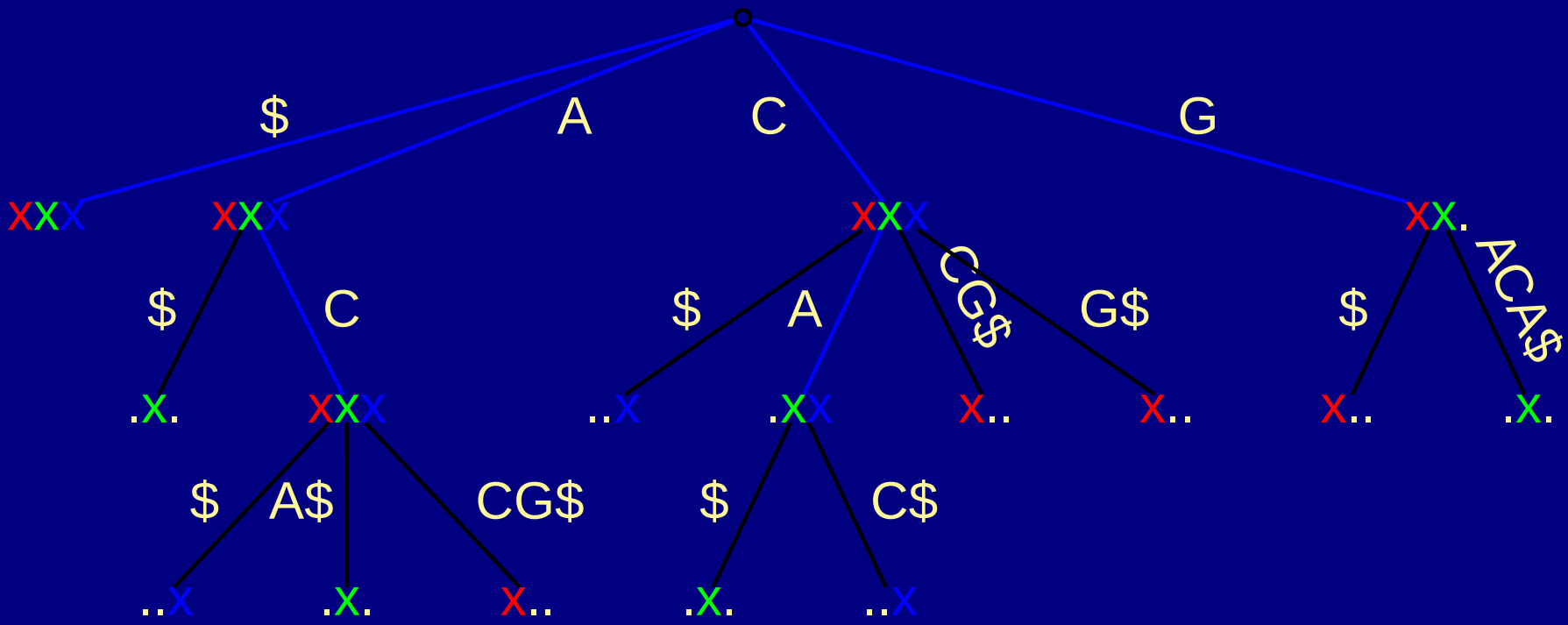
Generalized suffix tree: example

Generalized suffix tree of "ACCG$", "GACA$" and "CAC$":



Generalized suffix tree: example

Generalized suffix tree of ”ACCG$”, ”GACA$” and ”CAC$”:



Plan

The Suffix Array

Main idea

Sorting the suffix of a text into alphabetical order. Store this order into a table.

example

suffix array of ”cacao”

c	a	c	a	o
0	1	2	3	4

acao
ao
cacao
cao
o

The Suffix Array

Main idea

Sorting the suffix of a text into alphabetical order. Store this order into a table.

example

suffix array of ”cacao”

c	a	c	a	o
0	1	2	3	4

0	1	acao
1	3	ao
2	0	cacao
3	2	cao
4	4	o

The Suffix Array

Main idea

Sorting the suffix of a text into alphabetical order. Store this order into a table.

example

suffix array of ”cacao”

c	a	c	a	o
0	1	2	3	4

0	1
1	3
2	0
3	2
4	4

acao
ao
cacao
cao
o

The Suffix Array

Main idea

Sorting the suffix of a text into alphabetical order. Store this order into a table.

example

suffix array of ”cacao”

c	a	c	a	o
0	1	2	3	4

0	1	acao
1	3	ao
2	0	cacao
3	2	cao
4	4	o

Question: How to build suffix array in linear time?

Plan

Resume

The time/memory balance