Arbre de génération pour les permutations et simulation en aveugle

Romaric Duvignau

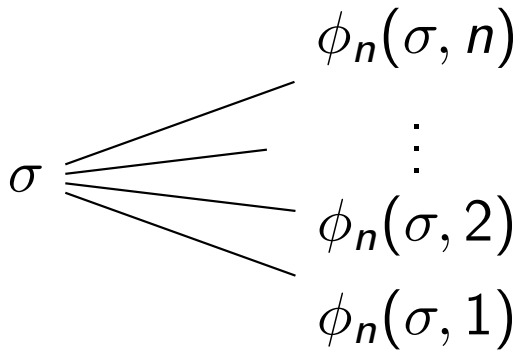
Journées de Combinatoire de Bordeaux

5 février 2015

Travail en commun avec Philippe Duchon

Plan

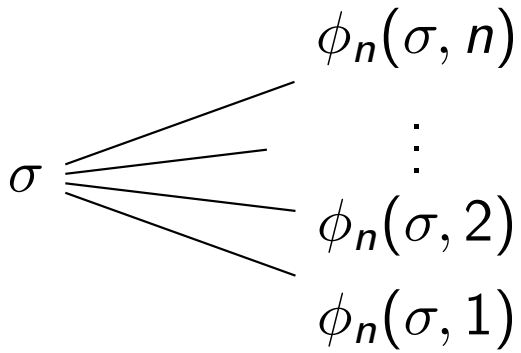
Arbre de génération



Définition : un arbre infini

- Nœuds : permutations et Racine : permutation vide
- Niveau n : $\mathcal{S}_n$ ensemble des permutations de $[n] = \{1, \ldots, n\}$
- Chaque permutation de taille n, a $n + 1$ enfants

Arbre de génération



Définition : un arbre infini

- Nœuds : permutations et Racine : permutation vide
- Niveau n : $\mathcal{S}_n$ ensemble des permutations de $[n] = \{1, \ldots, n\}$
- Chaque permutation de taille n, a $n+1$ enfants

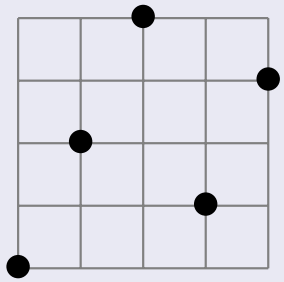
Une description de...

- Pour tout $n \geq 1$, une bijection ϕ_n entre $\mathcal{S}_{n-1} \times [n]$ et $\mathcal{S}_n$
- Algorithme de génération aléatoire uniforme de permutations de taille n :
  - Descente aléatoire dans l'arbre jusqu'au niveau n.

Arbre de génération : Exemple 1

Insertion par décalage : σ' i-ème enfant de σ

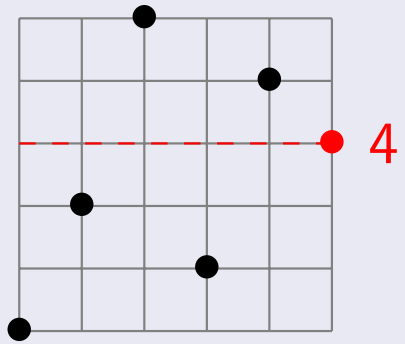
- $\sigma'(n) = i$
- Pour $1 \leq j \leq n-1$,
  - $\sigma'(j) = \sigma(j)$ si $\sigma(j) < i$
  - $\sigma'(j) = \sigma(j) + 1$ si $\sigma(j) \geq i$



Arbre de génération : Exemple 1

Insertion par décalage : σ' i-ème enfant de σ

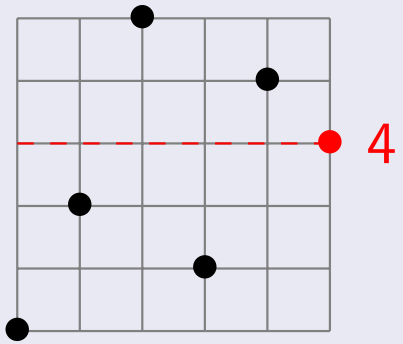
- $\sigma'(n) = i$
- Pour $1 \leq j \leq n-1$,
  - $\sigma'(j) = \sigma(j)$ si $\sigma(j) < i$
  - $\sigma'(j) = \sigma(j) + 1$ si $\sigma(j) \geq i$



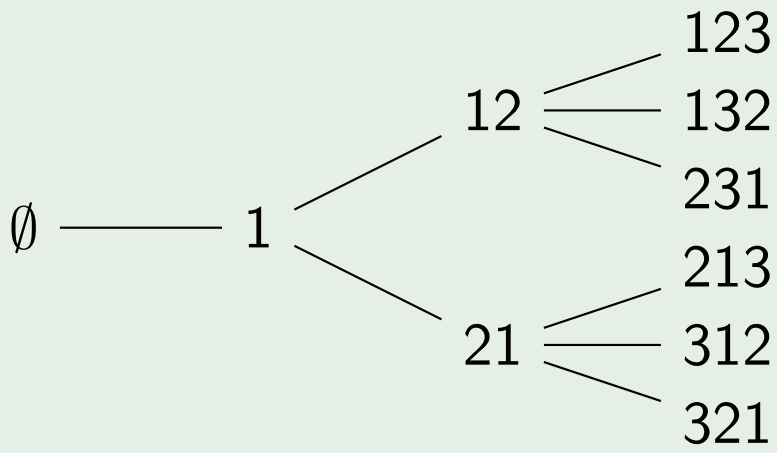
Arbre de génération : Exemple 1

Insertion par décalage : σ' i-ème enfant de σ

- $\sigma'(n) = i$
- Pour $1 \leq j \leq n-1$,
  - $\sigma'(j) = \sigma(j)$ si $\sigma(j) < i$
  - $\sigma'(j) = \sigma(j) + 1$ si $\sigma(j) \geq i$



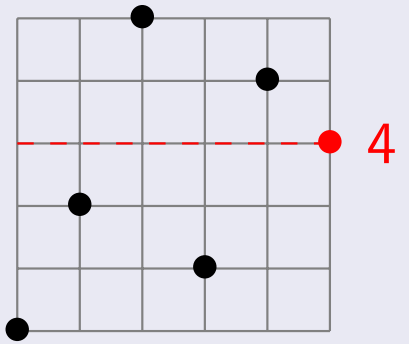
Début de l'arbre



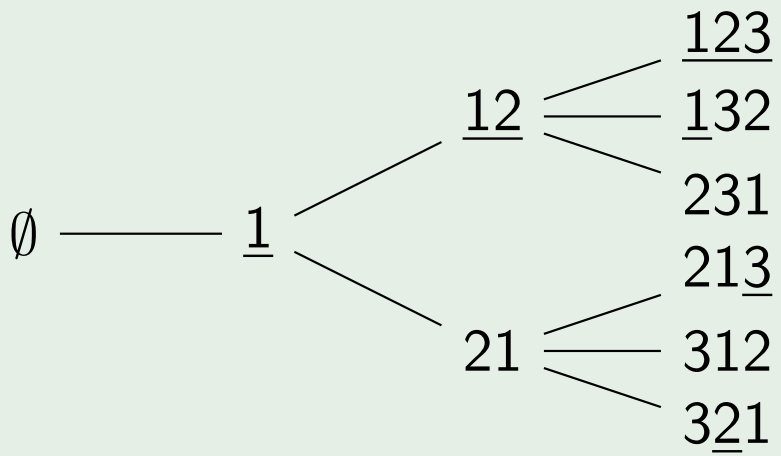
Arbre de génération : Exemple 1

Insertion par décalage : σ' i-ème enfant de σ

- $\sigma'(n) = i$
- Pour $1 \leq j \leq n-1$,
  - $\sigma'(j) = \sigma(j)$ si $\sigma(j) < i$
  - $\sigma'(j) = \sigma(j) + 1$ si $\sigma(j) \geq i$



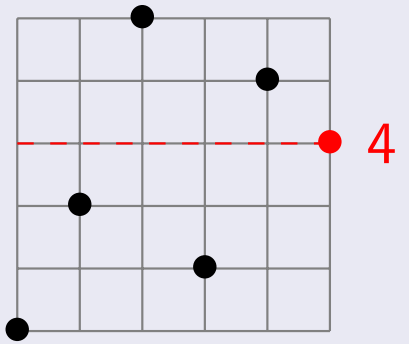
Début de l'arbre



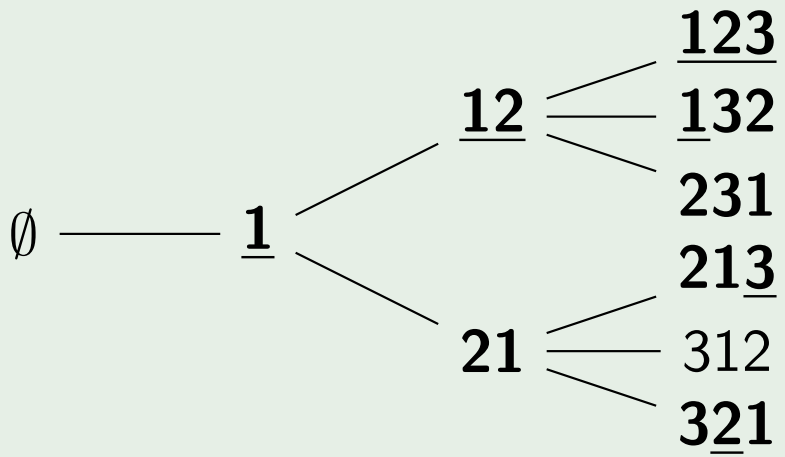
Arbre de génération : Exemple 1

Insertion par décalage : σ' i-ème enfant de σ

- $\sigma'(n) = i$
- Pour $1 \leq j \leq n-1$,
  - $\sigma'(j) = \sigma(j)$ si $\sigma(j) < i$
  - $\sigma'(j) = \sigma(j) + 1$ si $\sigma(j) \geq i$



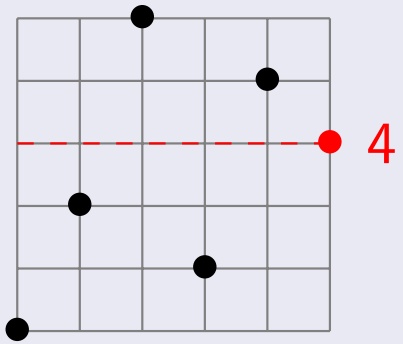
Début de l'arbre



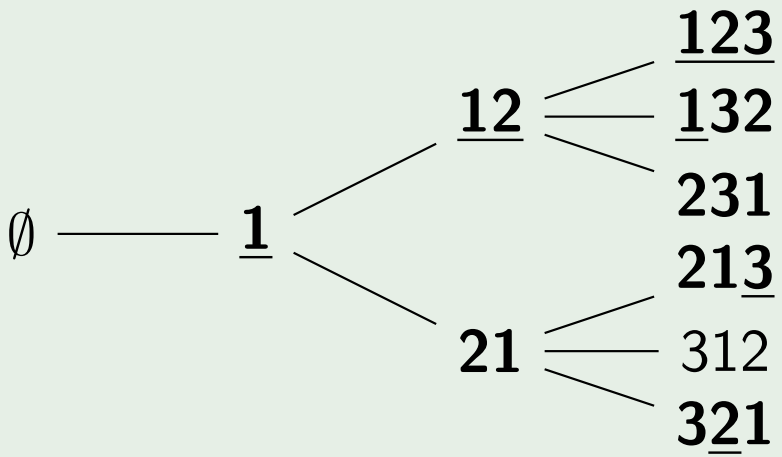
Arbre de génération : Exemple 1

Insertion par décalage : σ' i-ème enfant de σ

- $\sigma'(n) = i$
- Pour $1 \leq j \leq n-1$,
  - $\sigma'(j) = \sigma(j)$ si $\sigma(j) < i$
  - $\sigma'(j) = \sigma(j) + 1$ si $\sigma(j) \geq i$



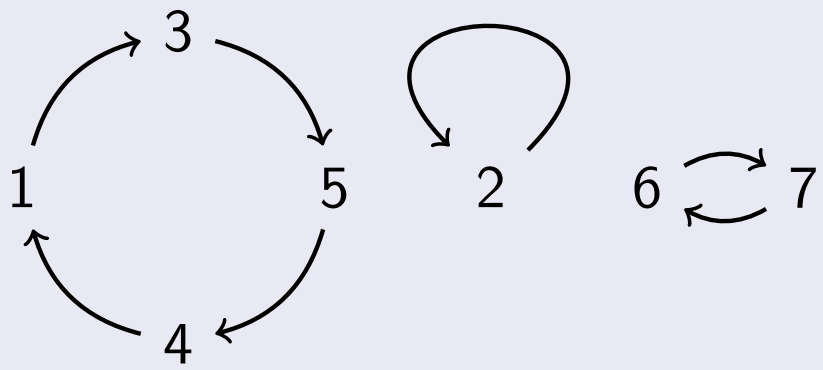
Début de l'arbre



- Le nombre de points fixes peut changer beaucoup
- $\geq (n-1)!$ permutations **changeantes** *i.e.* permutation σ avec $\text{fp}(\sigma) \neq \text{fp}(\text{parent}(\sigma))$

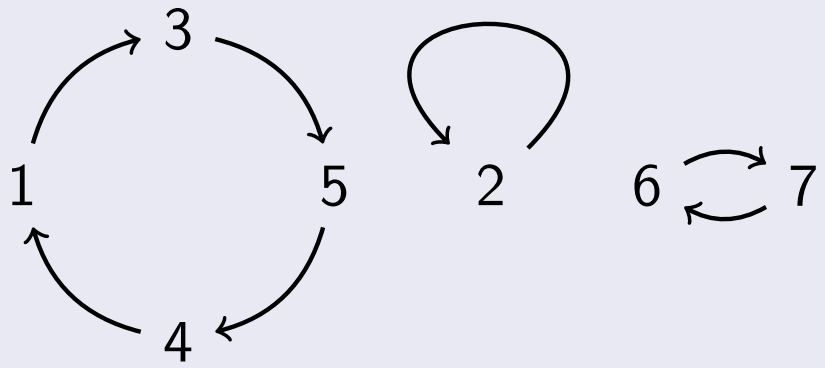
Insertion dans les cycles

Représentation des permutations : ensemble de cycles

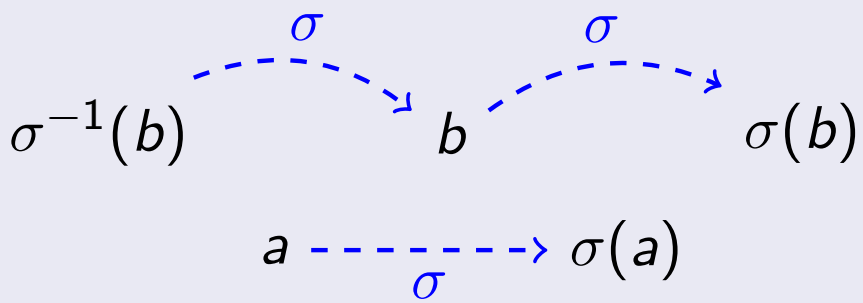


Insertion dans les cycles

Représentation des permutations : ensemble de cycles

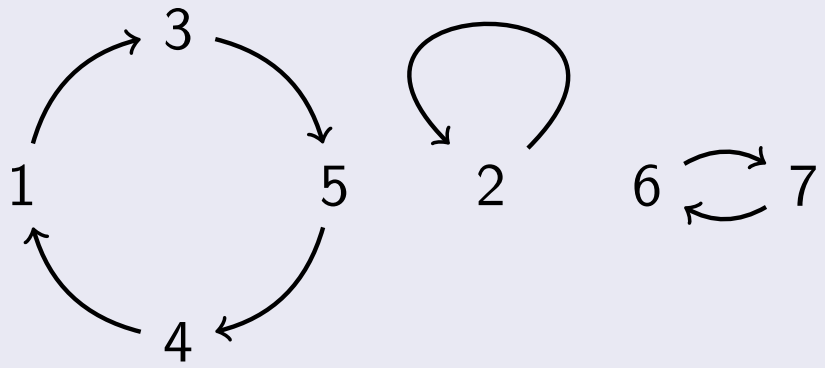


Insérer b après a

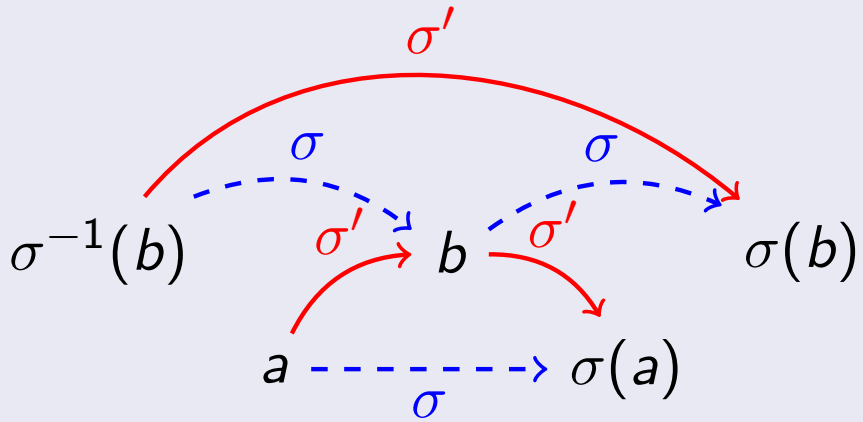


Insertion dans les cycles

Représentation des permutations : ensemble de cycles



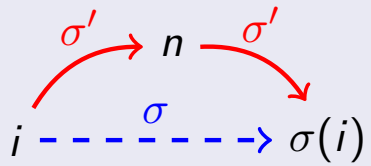
Insérer b après a



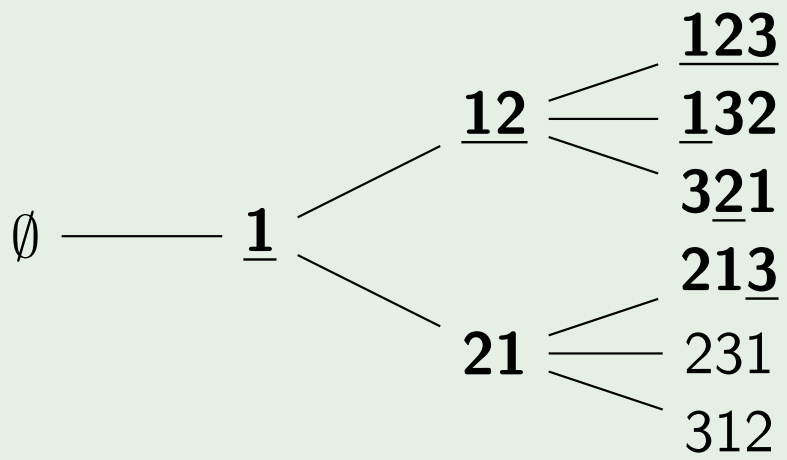
Arbre de génération : Exemple 2

Insertion dans les cycles : σ' i-ème enfant de σ

1. On ajoute n comme point fixe
2. Si $i \neq n$, insérer n après i



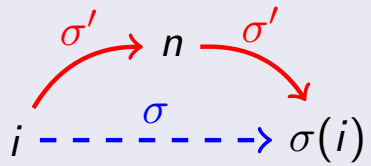
Début de l'arbre



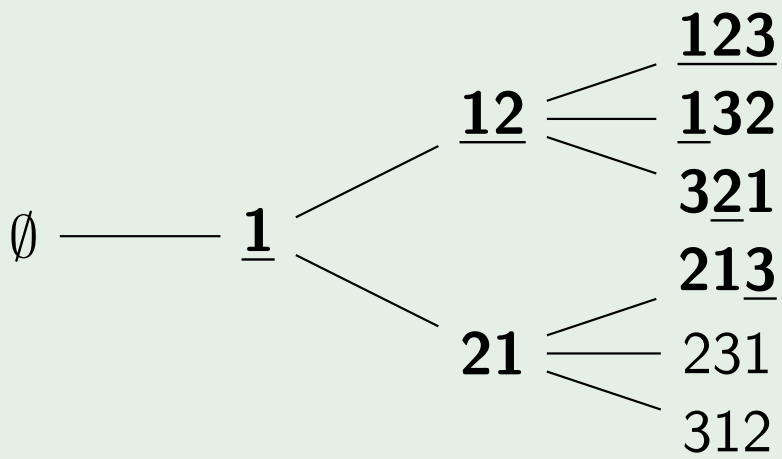
Arbre de génération : Exemple 2

Insertion dans les cycles : σ' i-ème enfant de σ

1. On ajoute n comme point fixe
2. Si $i \neq n$, insérer n après i



Début de l'arbre



- Points fixes :
  - $\mathrm{fp}(\sigma') = \mathrm{fp}(\sigma)$, ou
  - $\mathrm{fp}(\sigma') = \mathrm{fp}(\sigma) \pm 1$
- $2(n-1)!$ permutations *changeantes*

Nombre minimum de permutation changeantes

Proposition

Dans tout arbre de génération de permutations, à chaque niveau n, il y a au moins 2^{n-1} permutations qui n'ont pas le même nombre de points fixes que leur parent dans l'arbre.

Nombre minimum de permutation changeantes

Proposition

Dans tout arbre de génération de permutations, à chaque niveau n, il y a au moins 2^{n-1} permutations qui n'ont pas le même nombre de points fixes que leur parent dans l'arbre.

Preuve

1. On a $T(n,k) = \binom{n}{k} \cdot d_{n-k}$ et $d_n = nd_{n-1} + (-1)^n$.
2. Donc $T(n,k) = n \cdot T(n-1,k) + \binom{n}{k} \cdot (-1)^{n-k}$.
3. Lorsque $n-k$ est pair, au moins $\binom{n}{k}$ des (n,k)-permutations ont un parent qui n'est pas une $(n-1,k)$-permutation.
4. $\sum_{k=0 \mid n-k \text{ pair}}^{n} \binom{n}{k} = 2^{n-1}$.

- Une (n,k)-permutation est une permutation de taille n avec k points fixes
- On note $T(n,k)$ les **rencontres numbers** : le nombre de (n,k)-permutations et $d_n = T(n,0)$ le nombre de dérangements

Notre arbre de génération (propriétés)

On va décrire un arbre de génération qui va *préserver* au maximum le nombre de points fixes, i.e. minimiser le nombre de permutations changeantes de chaque taille.

Propriétés de notre arbre

- Exactement 2^{n-1} permutations **changeantes** de taille n
- Être non-changeante est héréditaire
- Le nombre de points fixes n'évolue que d'un (au plus) entre une permutation et ses enfants

Notre arbre de génération (propriétés)

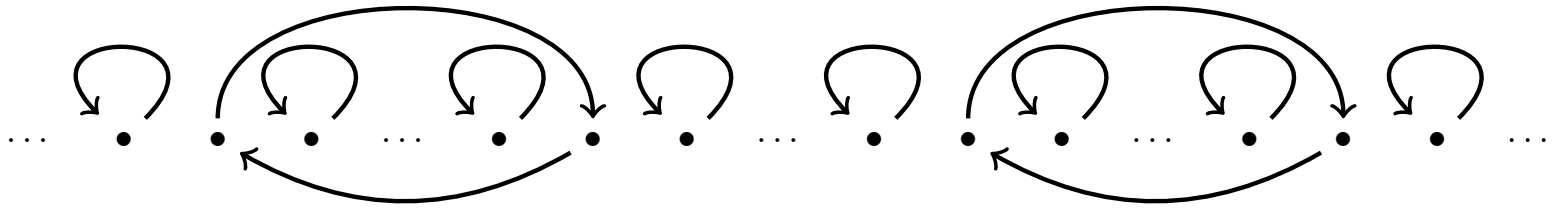
On va décrire un arbre de génération qui va *préserver* au maximum le nombre de points fixes, i.e. minimiser le nombre de permutations changeantes de chaque taille.

Propriétés de notre arbre

- Exactement 2^{n-1} permutations **changeantes** de taille n
- Être non-changeante est héréditaire
- Le nombre de points fixes n'évolue que d'un (au plus) entre une permutation et ses enfants

- Ce seront nos permutations **spéciales**.

Permutations spéciales

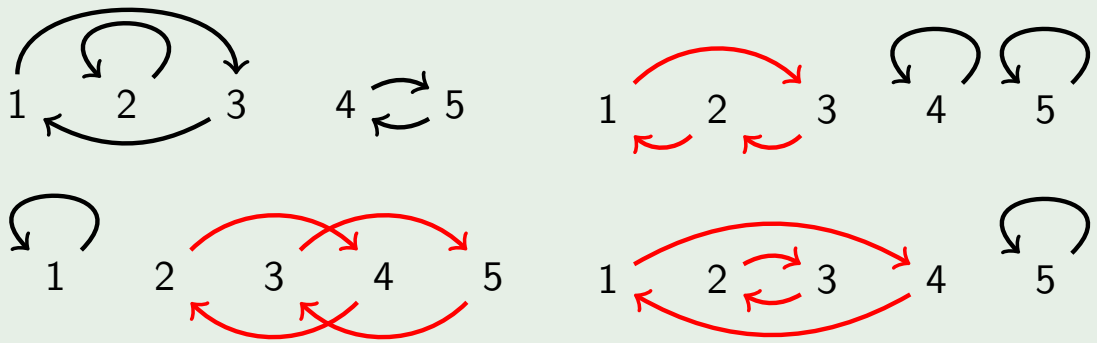


Définition

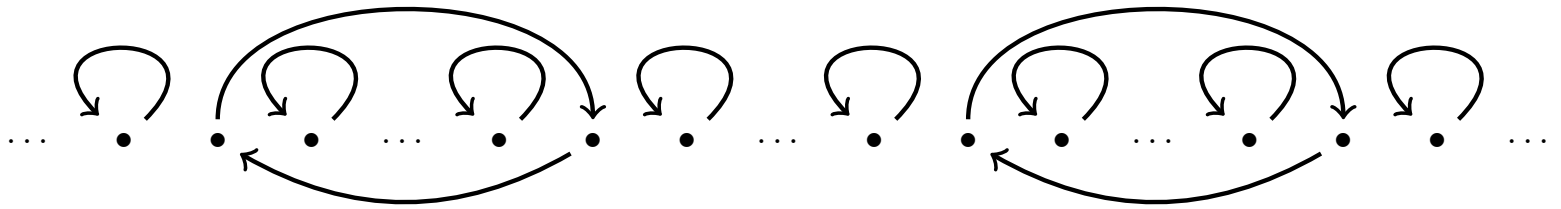
Une permutation σ de taille n est spéciale si, mis à part ses points fixes, elle ne contient que des cycles de taille 2, non-imbriqués et non-chevauchants, i.e. :

- Pour tout $i \in [n]$, $\sigma^2(i) = i$ et pour tout $i < j < \sigma(i)$, $\sigma(j) = j$

Exemples



Permutations spéciales



Définition

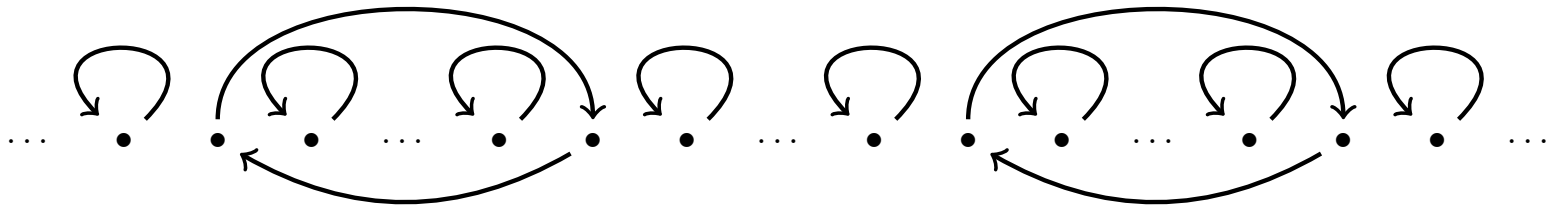
Une permutation σ de taille n est spéciale si, mis à part ses points fixes, elle ne contient que des cycles de taille 2, non-imbriqués et non-chevauchants, i.e. :

- Pour tout $i \in [n]$, $\sigma^2(i) = i$ et pour tout $i < j < \sigma(i)$, $\sigma(j) = j$

Dénombrement

Il existe autant de permutations spéciales de taille n que de sous-ensembles pairs de $[n]$, i.e. 2^{n-1}.

Permutations spéciales



Définition

Une permutation σ de taille n est spéciale si, mis à part ses points fixes, elle ne contient que des cycles de taille 2, non-imbriqués et non-chevauchants, i.e. :

- Pour tout $i \in [n]$, $\sigma^2(i) = i$ et pour tout $i < j < \sigma(i)$, $\sigma(j) = j$

Dénombrement

Il existe autant de permutations spéciales de taille n que de sous-ensembles pairs de $[n]$, i.e. 2^{n-1}.

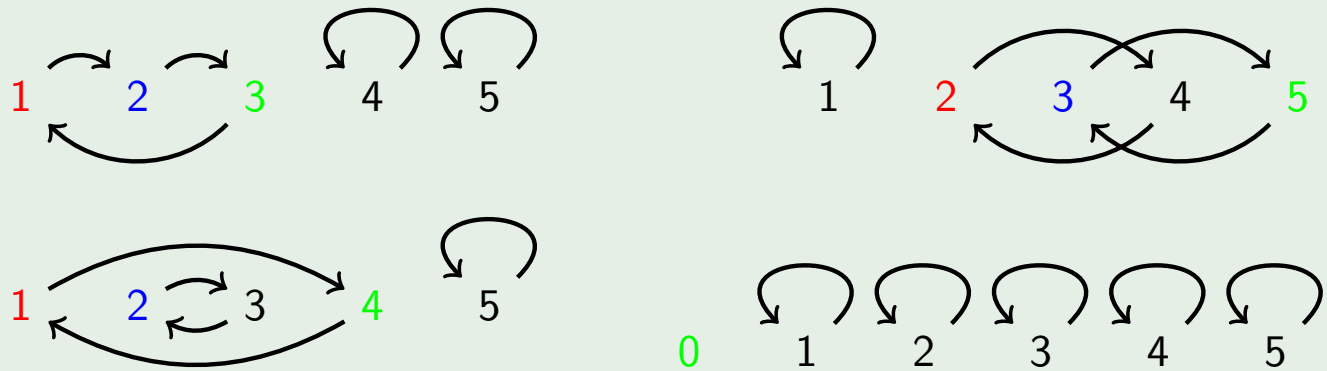
- Uniques permutations **changeantes** de notre arbre.

Notre arbre de génération (quelques dernières notations)

Notations

- On note $\gamma(\sigma)$ le plus grand non-point fixe de σ (ou 0 si σ est l'identité).
- Pour les permutations non-spéciales, on note $p(\sigma)$ le plus petit élément qui est soit
  - dans un cycle de taille ≥ 3, ou
  - tel qu'il existe un non-point fixe entre lui et son image.
- On note $p'(\sigma)$ le plus petit non-point fixe plus grand que $p(\sigma)$.

Exemples : $p(\sigma)$ $p'(\sigma)$ $\gamma(\sigma)$

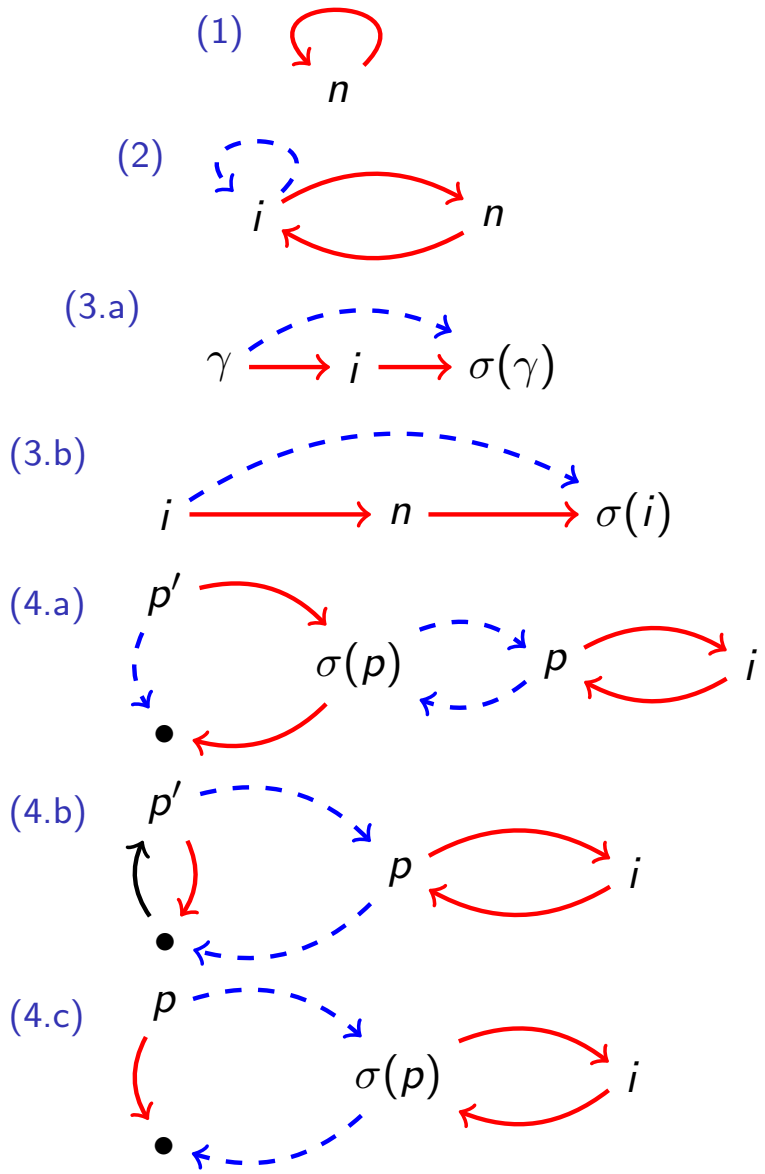


Notre arbre de génération : i-ème enfant de σ de taille $n-1$

On part de σ' obtenue en ajoutant n en point fixe à σ, puis :

1. Si σ est spéciale et $i = n$: rien.
2. Si σ est spéciale et $\gamma < i < n$:
   on insère n après i.
3. Si $i \leq \gamma$:
   a) Si $i = \sigma(i)$: on insère i après γ.
   b) Si $i \neq \sigma(i)$: on insère n après i.
4. Si σ est non-speciale et $i > \gamma$:
   a) Si $p \in$ 2-cycle : on insère p après i et on insère $\sigma(p)$ après p'.
   b) Si $p \in$ 3-cycle et $\sigma^{-1}(p) = p'$: on insère p après i.
   c) Sinon : on insère $\sigma(p)$ après i.

Ici, $\gamma = \gamma(\sigma)$, $p = p(\sigma)$ et $p' = p'(\sigma)$.



Notre arbre de génération : preuve

On peut facilement *inverser* la construction.

Permutations spéciales

Les permutations spéciales ne sont issues que des règles (1) et (2) :

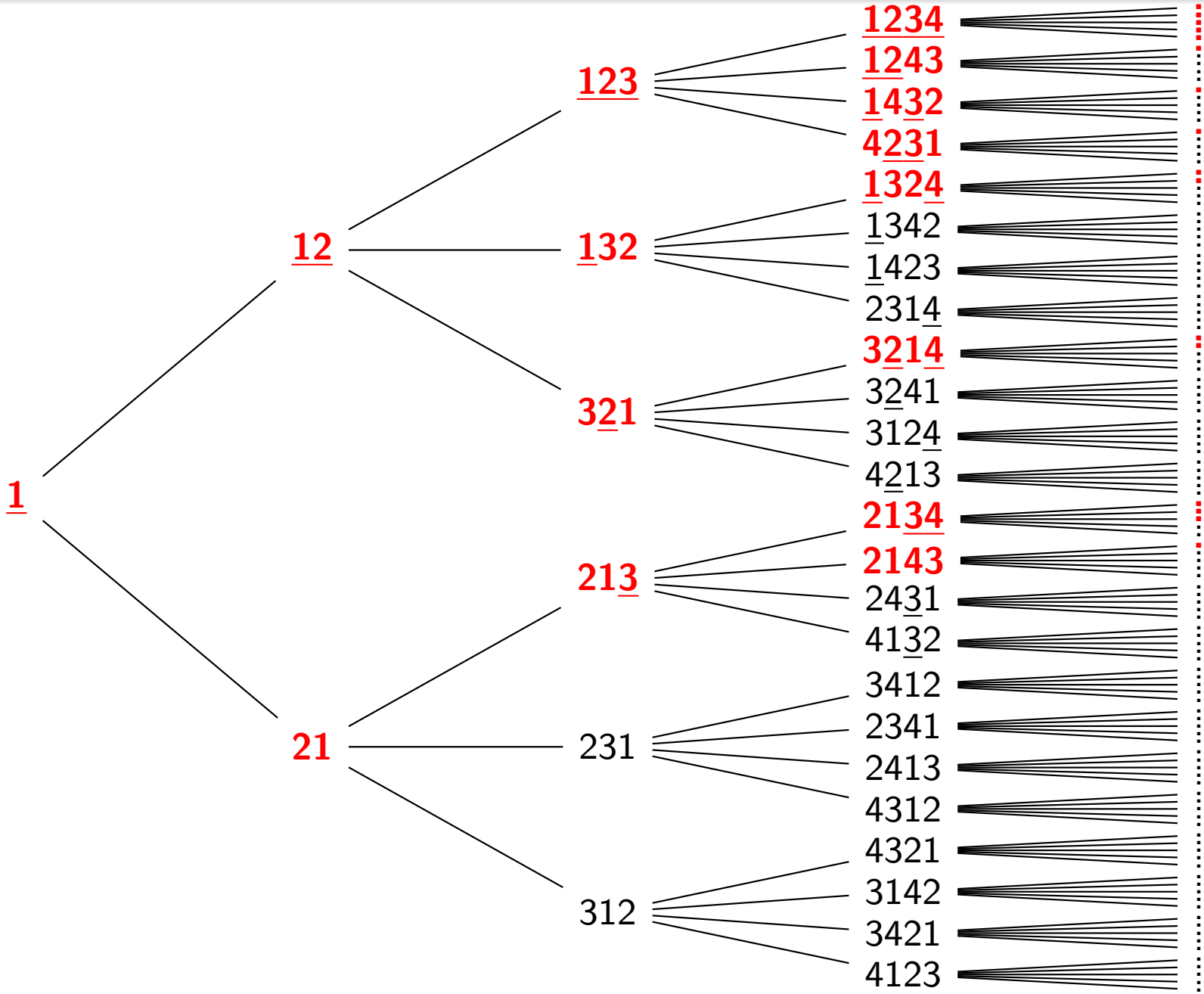
- (3) on crée un cycle de taille ≥ 3
- (4) p et p' ne forment toujours pas un *2-cycle*

Permutations non spéciales

Il y a 2 cas suivant la taille du cycle de $\gamma(\sigma')$:

- $\gamma(\sigma') \in \geq$ 3-cycle : il n'y a que la règle 3 qui a pu être utilisée
  - (3.a) Si n est point fixe
  - (3.b) Si n n'est pas point fixe
- $\gamma(\sigma') \in$ 2-cycle : il n'y a que la règle 4 qui a pu être utilisée
  - (4.a) Si $\gamma(\sigma')$ est en couple avec p et $p' \in \geq$ 3-cycle
  - (4.b) Si $\gamma(\sigma')$ est en couple avec p et $p' \in$ 2-cycle
  - (4.c) Si $\gamma(\sigma')$ n'est pas en couple avec p

Notre arbre de génération : les premiers niveaux



Plan

1. Présentation de l'arbre de génération
2. **Deux applications probabilistes de notre arbre**
3. Simulation en aveugle

Application 1 : Génération aléatoire de dérangements

État de l'art

- Comme $d_n = \lceil n!/e \rfloor$, une simple méthode par rejet utilise en moyenne $en + O(1/(n-1)!)$ tirages (appels à `Random`(k))
- Amélioration possible par un rejet anticipé avec en moyenne $(e-1)n + o(n)$ tirages en utilisant le *mélange de Fisher-Yates*
- Martínez, Panholzer et Prodinger (2008) proposent un équivalent du mélange de Fisher-Yates pour les dérangements (temps linéaire) qui a besoin de $2n + o(n)$ tirages
- Possibilité d'utiliser une bijection sur les dérangements pour obtenir au plus $(3/2)n$ tirages

Application 1 : Génération aléatoire de dérangements

État de l'art

- Comme $d_n = \lceil n!/e \rfloor$, une simple méthode par rejet utilise en moyenne $en + O(1/(n-1)!)$ tirages (appels à `Random(k)`)
- Amélioration possible par un rejet anticipé avec en moyenne $(e-1)n + o(n)$ tirages en utilisant le *mélange de Fisher-Yates*
- Martínez, Panholzer et Prodinger (2008) proposent un équivalent du mélange de Fisher-Yates pour les dérangements (temps linéaire) qui a besoin de $2n + o(n)$ tirages
- Possibilité d'utiliser une bijection sur les dérangements pour obtenir au plus $(3/2)n$ tirages

Notre algorithme pour générer un dérangement de taille n

- Utilise en moyenne $n + O(1)$ tirages (idem complexité temps)

Application 1 : Génération aléatoire de dérangements

UniformDerangement(n)

1. **Répéter :**
   - Descendre aléatoirement dans l'arbre jusqu'à obtenir une permutation σ non-spéciale ou atteindre le niveau n

   jusqu'à ce que σ soit un dérangement
2. Terminer la descente jusqu'au niveau n

Application 1 : Génération aléatoire de dérangements

UniformDerangement(n)

1. **Répéter :**
   - Descendre aléatoirement dans l'arbre jusqu'à obtenir une permutation σ non-spéciale ou atteindre le niveau n

   jusqu'à ce que σ soit un dérangement
2. Terminer la descente jusqu'au niveau n

Preuve (OK, rejet anticipé) et coût

- Le coût moyen d'une étape 1 est $\leq \sum_{m=1}^{\infty} \frac{2^{m-1}}{m!} = (e^2 - 1)/2$.
- La probabilité de succès de l'étape 1 est de $1/e + O(1/n!)$.
- On utilise toujours moins de $n - 3$ tirages pour l'étape 2.
- Donc le coût moyen total est plus petit que $n + 5.69$ (numériquement $n + 5.5$).

Application 1 : Génération aléatoire de dérangements

UniformDerangement(n)

1. **Répéter :**
   - Descendre aléatoirement dans l'arbre jusqu'à obtenir une permutation σ non-spéciale ou atteindre le niveau n

   jusqu'à ce que σ soit un dérangement
2. Terminer la descente jusqu'au niveau n

Preuve (OK, rejet anticipé) et coût

- Le coût moyen d'une étape 1 est $\leq \sum_{m=1}^{\infty} \frac{2^{m-1}}{m!} = (e^2 - 1)/2$.
- La probabilité de succès de l'étape 1 est de $1/e + O(1/n!)$.
- On utilise toujours moins de $n - 3$ tirages pour l'étape 2.
- Donc le coût moyen total est plus petit que $n + 5.69$ (numériquement $n + 5.5$).

- Soit $n \log_2(n) + o(n \log_2(n))$ bits aléatoires.

Application 2 : Simulation de Poisson(1)

Simulation de la loi de Poisson

- Plusieurs algorithmes efficaces existent mais utilisent des nombres réels
- Proposition : algorithme combinatoire pour simuler une Poisson(1) n'utilisant que des petits nombres entiers

Application 2 : Simulation de Poisson(1)

Simulation de la loi de Poisson

- Plusieurs algorithmes efficaces existent mais utilisent des nombres réels
- Proposition : algorithme combinatoire pour simuler une Poisson(1) n'utilisant que des petits nombres entiers

L'algorithme

- Descendre dans l'arbre jusqu'à obtenir une permutation non-spéciale
- Retourner son nombre de points fixes

Application 2 : Simulation de Poisson(1)

Simulation de la loi de Poisson

- Plusieurs algorithmes efficaces existent mais utilisent des nombres réels
- Proposition : algorithme combinatoire pour simuler une Poisson(1) n'utilisant que des petits nombres entiers

L'algorithme

- Descendre dans l'arbre jusqu'à obtenir une permutation non-spéciale
- Retourner son nombre de points fixes

- La distribution du nombre de point fixe d'une permutation uniforme de taille n converge en loi lorsque n tend vers l'infini vers la Poisson de paramètre 1.

Application 2 : Simulation de Poisson(1)

Algorithm 1 Poisson

$n \leftarrow 2$, $g \leftarrow 0$, $k \leftarrow 1$
loop
  $i \leftarrow$ **Random**(n)
  if $i = n$ **then**
    $k \leftarrow k + 1$
  else if $i > g$ **then**
    $k \leftarrow k - 1$, $g \leftarrow n$
  else
    return k
  end if
  $n \leftarrow n + 1$
end loop

Coût (en bits aléatoires)

- 6.9 bits (l'optimal est ≤ 3.89 d'après Knuth et Yao (1976))

Plan

1. Présentation de l'arbre de génération
2. 2 applications probabilistes de notre arbre
3. **Simulation en aveugle**

Le problème d'origine

Contexte

- Réseau P2P modélisé par un graphe aléatoire dynamique
- Objectif : maintenir **exactement** une distribution de graphes donnée après n'importe quelle séquence de mises à jour (insertion/suppression de sommets)
- **RandomVertex()** : uniforme sur l'ensemble des nœuds

Le problème d'origine

Contexte

- Réseau P2P modélisé par un graphe aléatoire dynamique
- Objectif : maintenir **exactement** une distribution de graphes donnée après n'importe quelle séquence de mises à jour (insertion/suppression de sommets)
- **RandomVertex()** : uniforme sur l'ensemble des nœuds

Modèle graphes k-sortants uniformes

- Pour les graphes k-sortants uniformes (voisinages sortants de taille k uniformes indépendants), le problème de maintenance peut être résolu sans connaître la taille du réseau si on arrive à simuler une Binomiale$(n, k/n)$ pour $n \geq k$.

Reformulation en terme de sac de boules

Sac de n boules

$\longrightarrow X \sim \rho_n$

Simulation sous contraintes

- Sac de n boules distinctes avec n **inconnu**
- Possibilité de tirer des boules *avec remise*
- But : Générer $X \sim \rho_n$
- On suppose qu'on a le droit de tirer à pile ou face

Reformulation en terme de sac de boules

Sac de n boules

$\longrightarrow X \sim \rho_n$

Simulation sous contraintes

- Sac de n boules distinctes avec n **inconnu**
- Possibilité de tirer des boules *avec remise*
- But : Générer $X \sim \rho_n$
- On suppose qu'on a le droit de tirer à pile ou face

- Notre arbre de génération nous permet de simuler efficacement la Binomiale$(n, 1/n)$.

Binomiale($n, 1/n$) : algorithme du 1er doublon

Algorithme du premier doublon

- Tirer des boules jusqu'à obtenir un doublon
- Retourner l'ensemble des points fixes d'une permutation uniforme des couleurs vues
- Cet ensemble est un *ensemble binomial* sur le sac de boules :
  - Chaque boule est retournée avec probabilité $1/n$ indépendamment des autres

Binomiale($n, 1/n$) : algorithme du 1er doublon

Algorithme du premier doublon

- Tirer des boules jusqu'à obtenir un doublon
- Retourner l'ensemble des points fixes d'une permutation uniforme des couleurs vues
- Cet ensemble est un *ensemble binomial* sur le sac de boules :
  - Chaque boule est retournée avec probabilité $1/n$ indépendamment des autres

Preuve

- Calcul simple

Binomiale$(n, 1/n)$: algorithme du 1er doublon

Algorithme du premier doublon

- Tirer des boules jusqu'à obtenir un doublon
- Retourner l'ensemble des points fixes d'une permutation uniforme des couleurs vues
- Cet ensemble est un *ensemble binomial* sur le sac de boules :
  - Chaque boule est retournée avec probabilité $1/n$ indépendamment des autres

Preuve

- Calcul simple

- Coût moyen $\mathcal{O}(\sqrt{n})$ (paradoxe des anniversaires)

Simulation de la Binomiale$(n, 1/n)$

Algorithme

- Descendre dans l'arbre de génération et à chaque descente tirer une boule du sac jusqu'à
  1. Obtenir une permutation non-spéciale : retourner son nombre de points fixes, ou
  2. Obtenir un doublon : retourner le nombre de points fixes du père de la permutation courante
- S'obtient en adaptant Poisson(1)

Simulation de la Binomiale$(n, 1/n)$

Algorithme

- Descendre dans l'arbre de génération et à chaque descente tirer une boule du sac jusqu'à
  1. Obtenir une permutation non-spéciale : retourner son nombre de points fixes, ou
  2. Obtenir un doublon : retourner le nombre de points fixes du père de la permutation courante
- S'obtient en adaptant Poisson(1)

Correction

- Le nombre renvoyé est distribué comme le nombre de points fixes d'une permutation des boules tirées jusqu'au 1^{er} doublon

Simulation de la Binomiale$(n, 1/n)$

Algorithme

- Descendre dans l'arbre de génération et à chaque descente tirer une boule du sac jusqu'à
  1. Obtenir une permutation non-spéciale : retourner son nombre de points fixes, ou
  2. Obtenir un doublon : retourner le nombre de points fixes du père de la permutation courante
- S'obtient en adaptant Poisson(1)

Correction

- Le nombre renvoyé est distribué comme le nombre de points fixes d'une permutation des boules tirées jusqu'au 1^{er} doublon

Coût moyen (en tirage de boules)

- $\leq$ au coût pour obtenir une permutation non-spéciale, i.e. $(e^2 - 1)/2 \approx 3.1945$ tirages de boules.

Quid de Binomiale$(n, k/n)$?

Simulation de la Binomiale$(n, k/n)$

- On a un algorithme en $\mathcal{O}(k)$ se basant sur notre procédure pour simuler Binomiale$(n, 1/n)$...

Quid de Binomiale$(n, k/n)$?

Simulation de la Binomiale$(n, k/n)$

- On a un algorithme en $\mathcal{O}(k)$ se basant sur notre procédure pour simuler Binomiale$(n, 1/n)$...
- qui n'est pas encore écrit !

Conclusion

Arbre de génération

- Efficace en nombre de tirages aléatoires pour générer des dérangements
- Peut être utilisé pour générer d'autres types de permutations (nombre de points fixes donné, pair, ...)
- Fournit un algorithme combinatoire et simple pour Poisson(1)

Conclusion

Arbre de génération

- Efficace en nombre de tirages aléatoires pour générer des dérangements
- Peut être utilisé pour générer d'autres types de permutations (nombre de points fixes donné, pair, ...)
- Fournit un algorithme combinatoire et simple pour Poisson(1)

Simulation en aveugle

- Temps moyen constant pour Binomiale$(n, 1/n)$
- $\mathcal{O}(k^2)$ pour Binomiale$(n, k/n)$ (asymptotiquement $\mathcal{O}(k)$)

Merci

Merci pour votre attention.
Avez-vous des questions ?

Algorithme du 1er doublon : Preuve

Preuve

Soient

- $\mathcal{U} \subseteq V$ un ensemble de taille k
- D l'ensemble de taille L des couleurs tirées par l'algorithme
- A_u l'évènement "u est retourné par l'algorithme"

On a :

1. $\mathbb{P}(L = \ell) = \frac{(n)_\ell}{n^\ell} \frac{\ell}{n} = \frac{n!}{(n-\ell)!} \frac{\ell}{n^{\ell+1}}$
2. Pour $\ell \geq k$, $\mathbb{P}(\mathcal{U} \subseteq D \mid L = \ell) = \binom{n-k}{\ell-k} / \binom{n}{\ell} = \frac{(n-k)!}{(\ell-k)!} \frac{\ell!}{n!}$
3. Pour $\ell \geq k$, $\mathbb{P}(\bigcap_{u \in \mathcal{U}} A_u \mid \mathcal{U} \subseteq D, L = \ell) = (\ell - k)!/\ell!$

Algorithme du 1er doublon : Preuve

Preuve

Soient

- $\mathcal{U} \subseteq V$ un ensemble de taille k
- D l'ensemble de taille L des couleurs tirées par l'algorithme
- A_u l'évènement "u est retourné par l'algorithme"

On a :

1. $\mathbb{P}(L = \ell) = \frac{(n)_\ell}{n^\ell} \frac{\ell}{n} = \frac{n!}{(n-\ell)!} \frac{\ell}{n^{\ell+1}}$
2. Pour $\ell \geq k$, $\mathbb{P}(\mathcal{U} \subseteq D \mid L = \ell) = \binom{n-k}{\ell-k} / \binom{n}{\ell} = \frac{(n-k)!}{(\ell-k)!} \frac{\ell!}{n!}$
3. Pour $\ell \geq k$, $\mathbb{P}(\bigcap_{u \in \mathcal{U}} A_u \mid \mathcal{U} \subseteq D, L = \ell) = (\ell - k)!/\ell!$

$$\mathbb{P}(\bigcap_{u \in \mathcal{U}} A_u) = \sum_{\ell=k}^{n} (1) \cdot (2) \cdot (3)$$

Algorithme du 1er doublon : Preuve

Preuve

Soient

- $\mathcal{U} \subseteq V$ un ensemble de taille k
- D l'ensemble de taille L des couleurs tirées par l'algorithme
- A_u l'évènement "u est retourné par l'algorithme"

On a :

1. $\mathbb{P}(L = \ell) = \frac{(n)_\ell}{n^\ell} \frac{\ell}{n} = \frac{n!}{(n-\ell)!} \frac{\ell}{n^{\ell+1}}$
2. Pour $\ell \geq k$, $\mathbb{P}(\mathcal{U} \subseteq D \mid L = \ell) = \binom{n-k}{\ell-k} / \binom{n}{\ell} = \frac{(n-k)!}{(\ell-k)!} \frac{\ell!}{n!}$
3. Pour $\ell \geq k$, $\mathbb{P}(\bigcap_{u \in \mathcal{U}} A_u \mid \mathcal{U} \subseteq D, L = \ell) = (\ell - k)!/\ell!$

$$\mathbb{P}(\bigcap_{u \in \mathcal{U}} A_u) = \sum_{\ell=k}^{n} \frac{n!}{(n-\ell)!} \frac{\ell}{n^{\ell+1}} \cdot \frac{(n-k)!}{(\ell-k)!} \frac{\ell!}{n!} \cdot \frac{(\ell-k)!}{\ell!}$$

Algorithme du 1er doublon : Preuve

Preuve

Soient

- $\mathcal{U} \subseteq V$ un ensemble de taille k
- D l'ensemble de taille L des couleurs tirées par l'algorithme
- A_u l'évènement "u est retourné par l'algorithme"

On a :

1. $\mathbb{P}(L = \ell) = \frac{(n)_\ell}{n^\ell} \frac{\ell}{n} = \frac{n!}{(n-\ell)!} \frac{\ell}{n^{\ell+1}}$
2. Pour $\ell \geq k$, $\mathbb{P}(\mathcal{U} \subseteq D \mid L = \ell) = \binom{n-k}{\ell-k} / \binom{n}{\ell} = \frac{(n-k)!}{(\ell-k)!} \frac{\ell!}{n!}$
3. Pour $\ell \geq k$, $\mathbb{P}(\bigcap_{u \in \mathcal{U}} A_u \mid \mathcal{U} \subseteq D, L = \ell) = (\ell - k)!/\ell!$

$$\mathbb{P}(\bigcap_{u \in \mathcal{U}} A_u) = \sum_{\ell=k}^{n} \frac{\cancel{n!}}{(n-\ell)!} \frac{\ell}{n^{\ell+1}} \cdot \frac{(n-k)!}{\cancel{(\ell-k)!}} \frac{\cancel{\ell!}}{\cancel{n!}} \cdot \frac{\cancel{(\ell-k)!}}{\cancel{\ell!}}$$

Algorithme du 1er doublon : Preuve

Preuve

Soient

- $\mathcal{U} \subseteq V$ un ensemble de taille k
- D l'ensemble de taille L des couleurs tirées par l'algorithme
- A_u l'évènement "u est retourné par l'algorithme"

On a :

1. $\mathbb{P}(L = \ell) = \frac{(n)_\ell}{n^\ell} \frac{\ell}{n} = \frac{n!}{(n-\ell)!} \frac{\ell}{n^{\ell+1}}$
2. Pour $\ell \geq k$, $\mathbb{P}(\mathcal{U} \subseteq D \mid L = \ell) = \binom{n-k}{\ell-k} / \binom{n}{\ell} = \frac{(n-k)!}{(\ell-k)!} \frac{\ell!}{n!}$
3. Pour $\ell \geq k$, $\mathbb{P}(\bigcap_{u \in \mathcal{U}} A_u \mid \mathcal{U} \subseteq D, L = \ell) = (\ell - k)!/\ell!$

$$\mathbb{P}(\bigcap_{u \in \mathcal{U}} A_u) = \sum_{\ell=k}^{n} \frac{(n-k)!}{(n-\ell)!} \frac{\ell}{n^{\ell+1}}$$

Algorithme du 1er doublon : Preuve

Preuve

Soient

- $\mathcal{U} \subseteq V$ un ensemble de taille k
- D l'ensemble de taille L des couleurs tirées par l'algorithme
- A_u l'évènement "u est retourné par l'algorithme"

On a :

1. $\mathbb{P}(L = \ell) = \frac{(n)_\ell}{n^\ell} \frac{\ell}{n} = \frac{n!}{(n-\ell)!} \frac{\ell}{n^{\ell+1}}$
2. Pour $\ell \geq k$, $\mathbb{P}(\mathcal{U} \subseteq D \mid L = \ell) = \binom{n-k}{\ell-k} / \binom{n}{\ell} = \frac{(n-k)!}{(\ell-k)!} \frac{\ell!}{n!}$
3. Pour $\ell \geq k$, $\mathbb{P}(\bigcap_{u \in \mathcal{U}} A_u \mid \mathcal{U} \subseteq D, L = \ell) = (\ell - k)!/\ell!$

$$\mathbb{P}(\bigcap_{u \in \mathcal{U}} A_u) = \frac{1}{n^k} \cdot \sum_{\ell=k}^{n} \frac{(n-k)!}{(n-\ell)!} \frac{\ell}{n^{\ell-k+1}}$$

Algorithme du 1er doublon : Preuve

Preuve

Soient

- $\mathcal{U} \subseteq V$ un ensemble de taille k
- D l'ensemble de taille L des couleurs tirées par l'algorithme
- A_u l'évènement "u est retourné par l'algorithme"

On a :

1. $\mathbb{P}(L = \ell) = \frac{(n)_\ell}{n^\ell} \frac{\ell}{n} = \frac{n!}{(n-\ell)!} \frac{\ell}{n^{\ell+1}}$
2. Pour $\ell \geq k$, $\mathbb{P}(\mathcal{U} \subseteq D \mid L = \ell) = \binom{n-k}{\ell-k} / \binom{n}{\ell} = \frac{(n-k)!}{(\ell-k)!} \frac{\ell!}{n!}$
3. Pour $\ell \geq k$, $\mathbb{P}(\bigcap_{u \in \mathcal{U}} A_u \mid \mathcal{U} \subseteq D, L = \ell) = (\ell - k)!/\ell!$

$$\mathbb{P}(\bigcap_{u \in \mathcal{U}} A_u) = \frac{1}{n^k} \cdot \underbrace{\sum_{\ell=k}^{n} \frac{(n-k)!}{(n-\ell)!} \frac{\ell}{n^{\ell-k+1}}}_{\mathbb{P}(L=\ell \mid L \geq k)}$$

Algorithme du 1er doublon : Preuve

Preuve

Soient

- $\mathcal{U} \subseteq V$ un ensemble de taille k
- D l'ensemble de taille L des couleurs tirées par l'algorithme
- A_u l'évènement "u est retourné par l'algorithme"

On a :

1. $\mathbb{P}(L = \ell) = \frac{(n)_\ell}{n^\ell} \frac{\ell}{n} = \frac{n!}{(n-\ell)!} \frac{\ell}{n^{\ell+1}}$
2. Pour $\ell \geq k$, $\mathbb{P}(\mathcal{U} \subseteq D \mid L = \ell) = \binom{n-k}{\ell-k} / \binom{n}{\ell} = \frac{(n-k)!}{(\ell-k)!} \frac{\ell!}{n!}$
3. Pour $\ell \geq k$, $\mathbb{P}(\bigcap_{u \in \mathcal{U}} A_u \mid \mathcal{U} \subseteq D, L = \ell) = (\ell - k)!/\ell!$

$$\mathbb{P}(\bigcap_{u \in \mathcal{U}} A_u) = \left(\frac{1}{n}\right)^k$$