Algorithmique des graphes

Cours 3 – Parcours en largeur

František Kardoš

frantisek.kardos@u-bordeaux.fr

Plan

- Vocabulaire des graphes orientés
- Parcours en largeur
- Sa complexité

Exercices du cours précédent

Comment définir l'isomorphisme pour des graphes pas forcément simples ?

Exercices du cours précédent

Comment définir l'isomorphisme pour des graphes pas forcément simples ?

Comment trouver dans une liste d'entiers données une tranche d'une somme donnée en temps linéaire ?

Graphes orientés

Un graphe orienté $G = (V, E)$ est un couple d'ensembles finis, dont

- V est l'ensemble de *sommets* de G, et
- E est l'ensemble d'*arcs* de G, où

tout arc relie un sommet à un (autre) sommet.

Si l'arc e relie u à v, on écrit $e = uv$, on dit que

- uv est un arc sortant de u,
- v est un voisin sortant / successeur de u,
- uv est un arc entrant à/de v,
- u est un voisin entrant / prédécesseur de v.

Graphes orientés – représentations

- Les listes de successeurs : pour chaque sommet du graphe la liste de ses successeurs ;
- La matrice d'adjacence : pour chaque paire de sommets v_i et v_j il est indiqué s'il existe un arc de v_i à v_j ;
- La matrice d'incidence : pour chaque sommet v_i et pour chaque arc e_j il est indiqué si l'arc e_j est un arc sortant (-1) ou entrant $(+1)$ du sommet v_i.

Graphes orientés : Degrés

Le *degré sortant* $deg^+(v)$ d'un sommet v est la longueur de la liste de successeurs de v.

Le degré sortant d'un sommet v est égal au nombre d'arcs sortants de v.

Graphes orientés : Degrés

Le *degré sortant* $deg^+(v)$ d'un sommet v est la longueur de la liste de successeurs de v.

Le degré sortant d'un sommet v est égal au nombre d'arcs sortants de v.

Le *degré entrant* $deg^-(v)$ d'un sommet v est la longueur de la liste de prédécesseur de v.

Le degré entrant d'un sommet v est égal au nombre d'arcs entrants de v.

Cheminement

Dans un graphe orienté G, un *chemin* de u à v est une séquence d'arcs consécutifs, par exemple $(u_0u_1, u_1u_2, \ldots, u_{k-1}u_k)$, telle que $u = u_0$ et $v = u_k$.

La *longueur* d'un chemin est le nombre d'arcs qui le composent (ici k).

Cheminement

Dans un graphe orienté G, un *chemin* de u à v est une séquence d'arcs consécutifs, par exemple $(u_0u_1, u_1u_2, \ldots, u_{k-1}u_k)$, telle que $u = u_0$ et $v = u_k$.

La *longueur* d'un chemin est le nombre d'arcs qui le composent (ici k).

On dit qu'un sommet v est *accessible* à partir du sommet u s'il existe un chemin de u à v.

Un graphe orienté est dit *fortement connexe* si ses sommets sont tous accessibles entre eux, i.e., pour toute paire de sommets u et v il existe un chemin de u à v et aussi un chemin de v à u.

Exercice

Pour chacune des trois représentations, concevoir un algorithme qui, étant donné un graphe orienté G et un sommet v de G, calcule le degré entrant de v.

Exercice

Pour chacune des trois représentations, concevoir un algorithme qui, étant donné un graphe orienté G et un sommet v de G, calcule le degré entrant de v.

À partir des listes de successeurs :

Exercice

Pour chacune des trois représentations, concevoir un algorithme qui, étant donné un graphe orienté G et un sommet v de G, calcule le degré entrant de v.

À partir des listes de successeurs :
Il faut parcourir toutes les listes et compter le nombre total d'occurrences de v dans l'ensemble des listes.

Exercice

Pour chacune des trois représentations, concevoir un algorithme qui, étant donné un graphe orienté G et un sommet v de G, calcule le degré entrant de v.

À partir des listes de successeurs :
Il faut parcourir toutes les listes et compter le nombre total d'occurrences de v dans l'ensemble des listes.

À partir de la matrice d'adjacence :

Exercice

Pour chacune des trois représentations, concevoir un algorithme qui, étant donné un graphe orienté G et un sommet v de G, calcule le degré entrant de v.

À partir des listes de successeurs :
Il faut parcourir toutes les listes et compter le nombre total d'occurrences de v dans l'ensemble des listes.

À partir de la matrice d'adjacence :
Il suffit de calculer la somme de la colonne du sommet v.

Exercice

Pour chacune des trois représentations, concevoir un algorithme qui, étant donné un graphe orienté G et un sommet v de G, calcule le degré entrant de v.

À partir des listes de successeurs :
Il faut parcourir toutes les listes et compter le nombre total d'occurrences de v dans l'ensemble des listes.

À partir de la matrice d'adjacence :
Il suffit de calculer la somme de la colonne du sommet v.

À partir de la matrice d'incidence :

Exercice

Pour chacune des trois représentations, concevoir un algorithme qui, étant donné un graphe orienté G et un sommet v de G, calcule le degré entrant de v.

À partir des listes de successeurs :
Il faut parcourir toutes les listes et compter le nombre total d'occurrences de v dans l'ensemble des listes.

À partir de la matrice d'adjacence :
Il suffit de calculer la somme de la colonne du sommet v.

À partir de la matrice d'incidence :
Il suffit de calculer le nombre de $+1$ dans la ligne de v.

Exercice

Pour chacune des trois représentations, concevoir un algorithme qui, étant donné un graphe orienté G et un sommet v de G, calcule le degré entrant de v.

À partir des listes de successeurs :
Il faut parcourir toutes les listes et compter le nombre total d'occurrences de v dans l'ensemble des listes.

À partir de la matrice d'adjacence :
Il suffit de calculer la somme de la colonne du sommet v.

À partir de la matrice d'incidence :
Il suffit de calculer le nombre de $+1$ dans la ligne de v.

Quelle est la complexité ?

Parcours en largeur ou BFS (Breadth First Search)

Un parcours en largeur explore le graphe à partir d'un sommet donné (sommet de départ ou sommet source).

L'algorithme permet de calculer les distances de tous les sommets accessibles depuis le sommet source.

Parcours en largeur ou BFS (Breadth First Search)

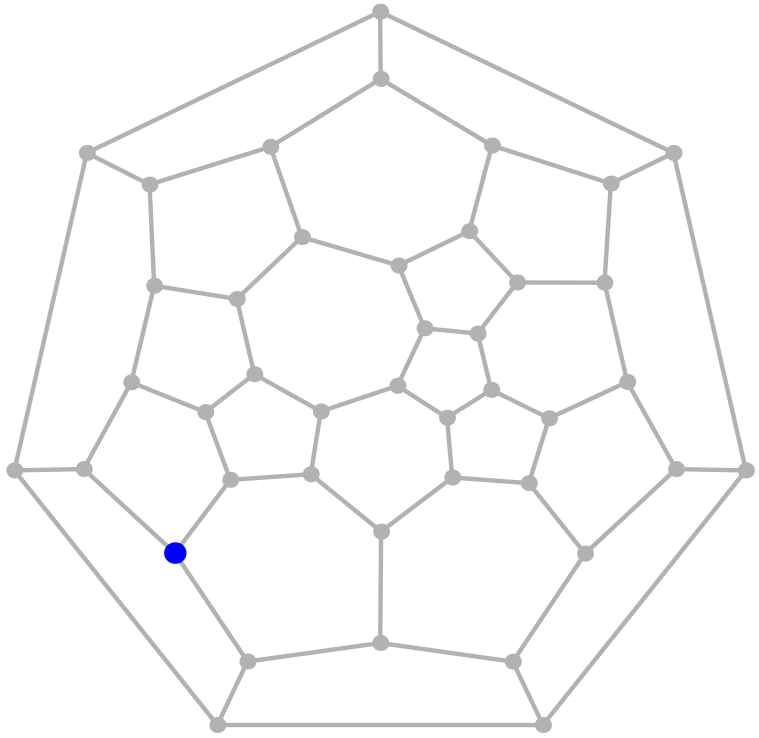
Un parcours en largeur explore le graphe à partir d'un sommet donné (sommet de départ ou sommet source).

L'algorithme permet de calculer les distances de tous les sommets accessibles depuis le sommet source.

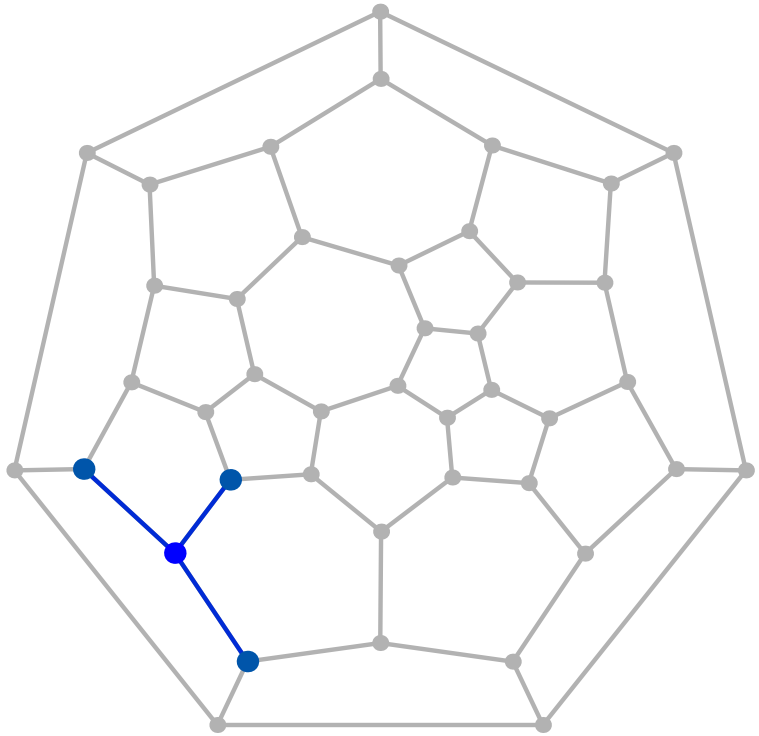
L'algorithme simule la transmission d'un message à partir d'un sommet source, en utilisant l'idée suivante :

Tout sommet qui reçoit le message, le transférera à tous ses voisins qui ne l'auront pas encore reçu.

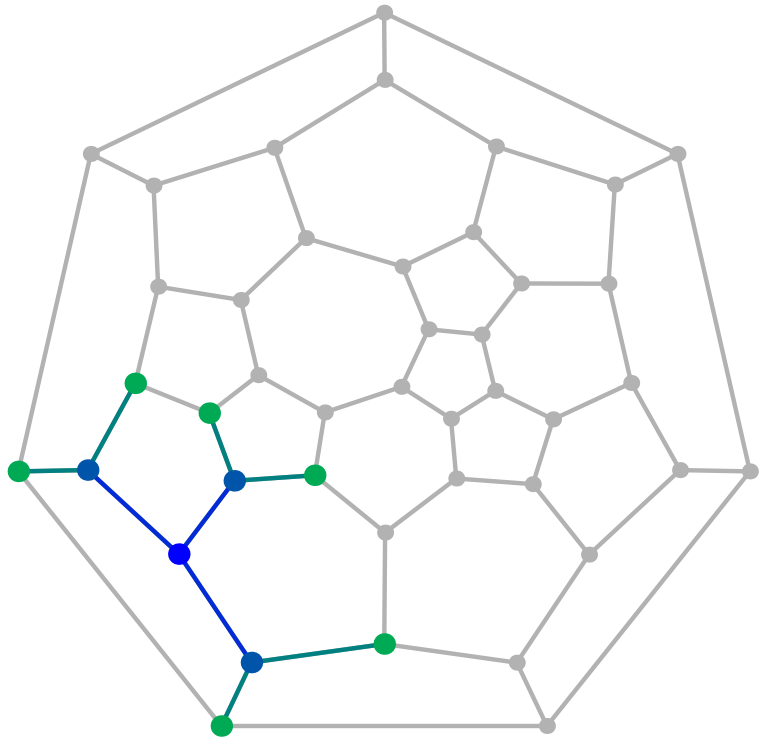
Parcours en largeur



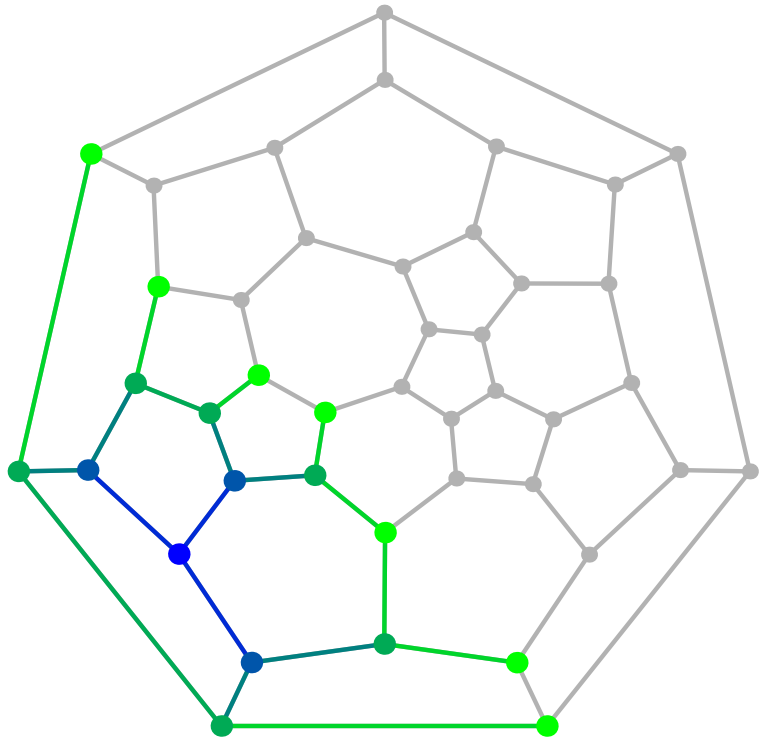
Parcours en largeur



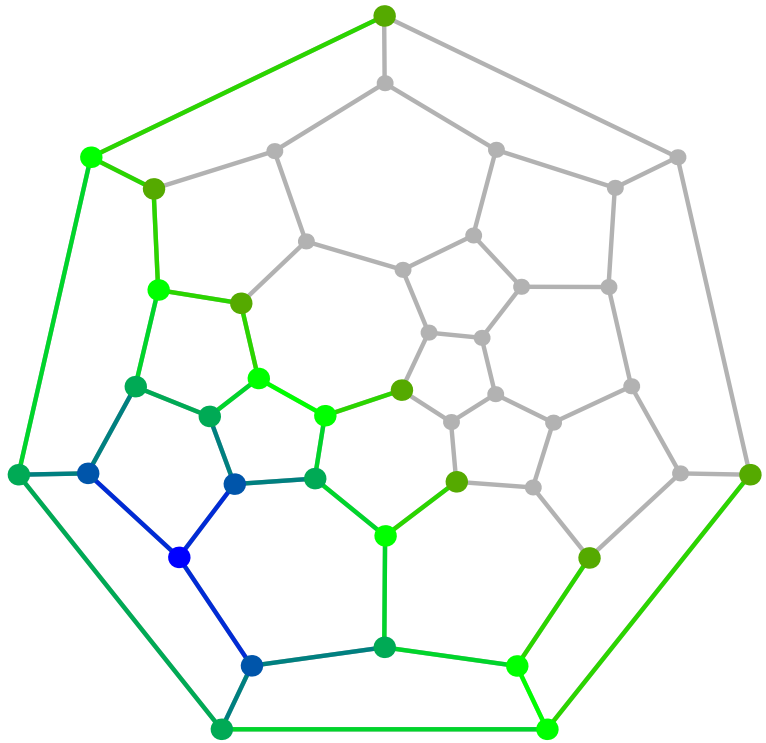
Parcours en largeur



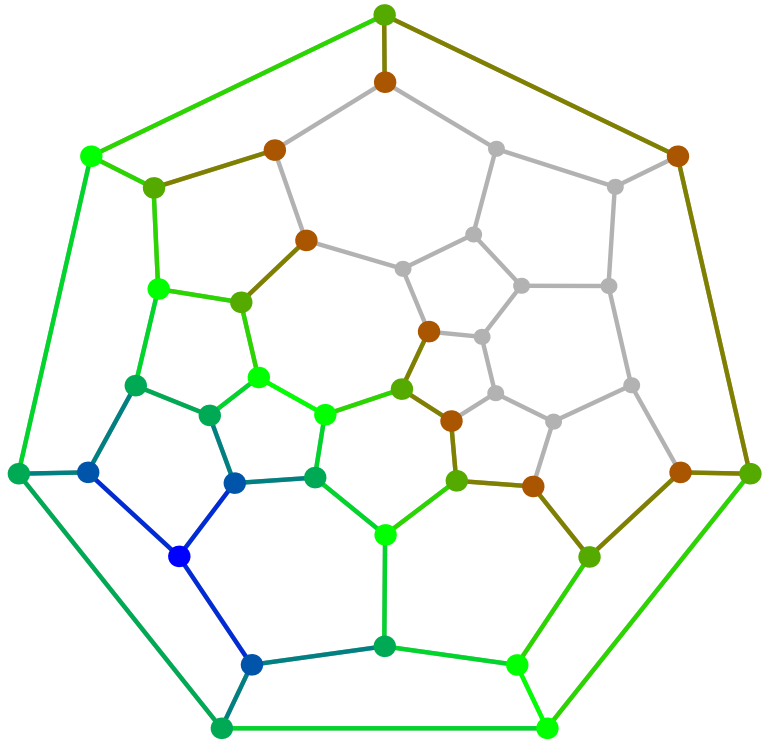
Parcours en largeur



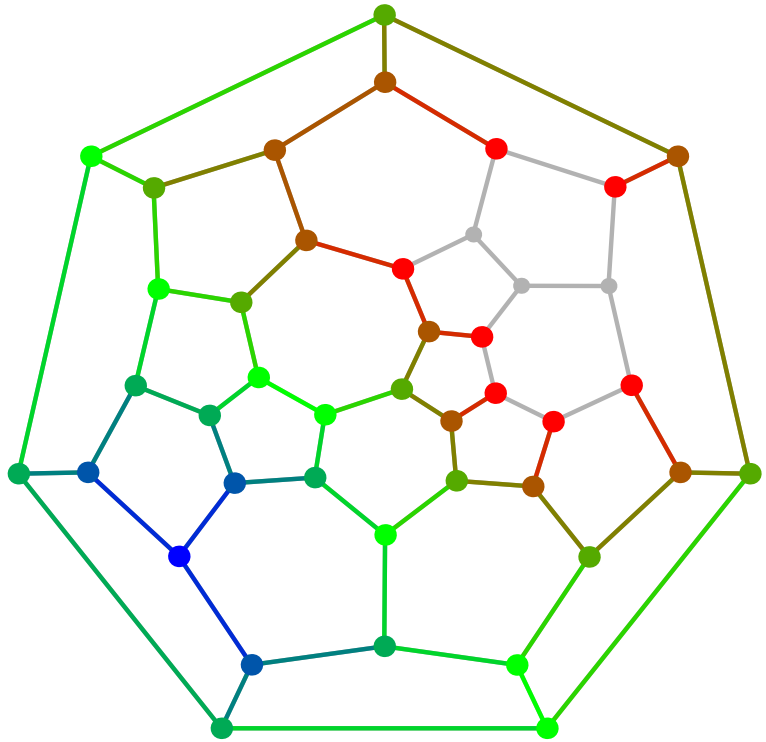
Parcours en largeur



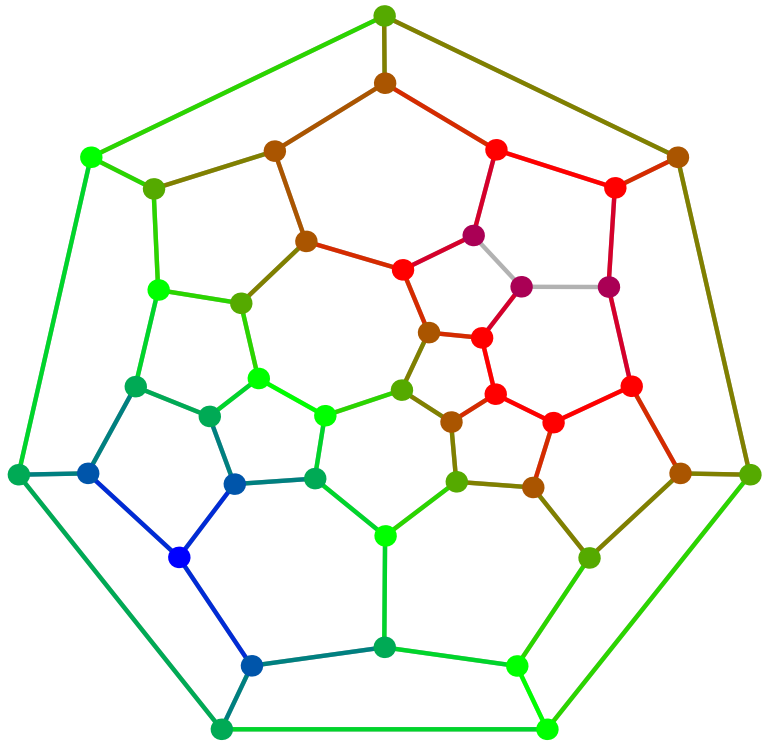
Parcours en largeur



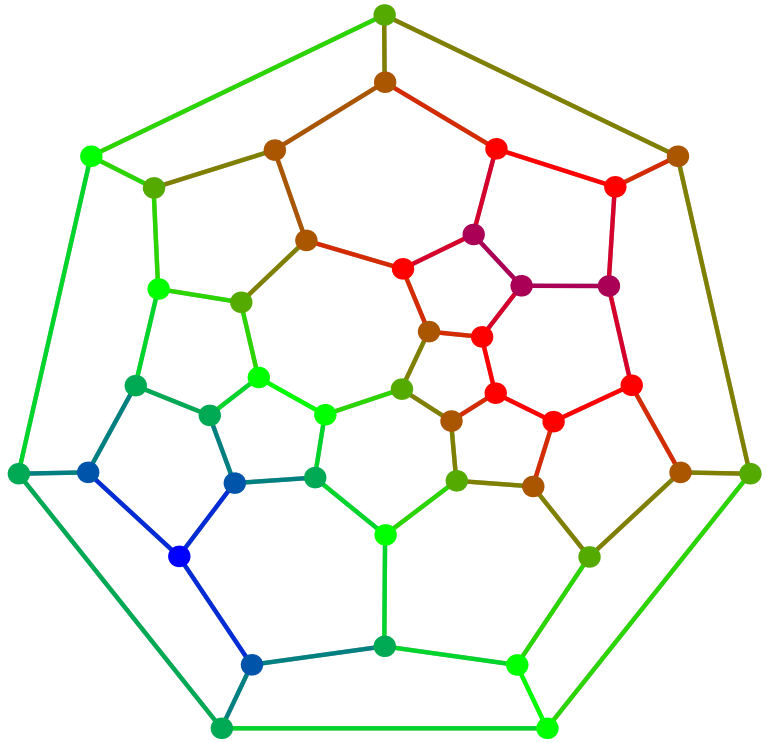
Parcours en largeur



Parcours en largeur



Parcours en largeur



Parcours en largeur

Pour modéliser le fait si un sommet a déjà été visité par le parcours (s'il a reçu le message) ou pas, on utilise le marquage.

Pour stocker l'ensemble de sommets ayant reçu le message, mais qui ne l'ont pas encore propagé, on utilise une file (FIFO).

Parcours en largeur (brut)

Algorithme : Parcours en largeur BFS(G, s)

Données : graphe G, sommet de départ s
File II (initialisée à vide), marque des sommets (initialisé à Faux)

```
début
    marque[s] ← Vrai ;
    enfiler s à la fin de II ;
    tant que II non vide faire
        u ← tête(II) ;
        pour chaque v voisin de u faire
            si v non marqué alors
                marque[v] ← Vrai ;
                enfiler v à la fin de II ;
            fin
        fin
        défiler u de la tête de II ;
    fin
fin
```

Parcours en largeur – calcul de distances

Algorithme : Parcours en largeur BFS(G, s)

Données : graphe G, sommet de départ s
File II (initialisée à vide), marque des sommets (initialisé à Faux),
distance *dist* des sommets depuis s (initialisée à infinie)

```
début
    marque[s] ← Vrai;
    enfiler s à la fin de II ;
    dist[s] ← 0 ;
    tant que II non vide faire
        u ← tête(II) ;
        pour chaque v voisin de u faire
            si v non marqué alors
                marque[v] ← Vrai ;
                dist[v] ← dist[u] + 1 ;
                enfiler v à la fin de II ;
            fin
        fin
        défiler u de la tête de II ;
    fin
fin
```

Parcours en largeur – calcul d'un arbre couvrant

Algorithme : Parcours en largeur BFS(G, s)

Données : graphe G, sommet de départ s
File II (vide), marque (faux) et distance (infinie) des sommets,
père π des sommets (initialisée à null)

début
  marque[s] $\leftarrow$ Vrai;
  enfiler s à la fin de II ;
  $dist[s] \leftarrow 0$;
  tant que II *non vide* **faire**
    $u \leftarrow$ tête(II) ;
    pour chaque v *voisin de* u **faire**
      si v *non marqué* **alors**
        marque[v] $\leftarrow$ Vrai ;
        $dist[v] \leftarrow dist[u] + 1$;
        $\pi[v] \leftarrow u$;
        enfiler v à la fin de II ;
      fin
    fin
    défiler u de la tête de II ;
  fin
fin

Parcours en largeur – exemple

u	voisins de u	marque	distance	père
a	b, d, d	Faux	∞	null
b	a, d, e, f	Faux	∞	null
c	e, f	Faux	∞	null
d	a, a, b, f	Faux	∞	null
e	b, c, f	Faux	∞	null
f	b, c, d, e	Faux	∞	null

II :

Parcours en largeur – exemple

u	voisins de u	marque	distance	père
a	b, d, d	Vrai	0	null
b	a, d, e, f	Faux	∞	null
c	e, f	Faux	∞	null
d	a, a, b, f	Faux	∞	null
e	b, c, f	Faux	∞	null
f	b, c, d, e	Faux	∞	null

II : a

Parcours en largeur – exemple

u	voisins de u	marque	distance	père
a	b, d, d	Vrai	0	null
b	a, d, e, f	Faux	∞	null
c	e, f	Faux	∞	null
d	a, a, b, f	Faux	∞	null
e	b, c, f	Faux	∞	null
f	b, c, d, e	Faux	∞	null

II : a

Parcours en largeur – exemple

u	voisins de u	marque	distance	père
a	b, d, d	Vrai	0	null
b	a, d, e, f	Vrai	1	a
c	e, f	Faux	∞	null
d	a, a, b, f	Faux	∞	null
e	b, c, f	Faux	∞	null
f	b, c, d, e	Faux	∞	null

II : a, b

Parcours en largeur – exemple

u	voisins de u	marque	distance	père
a	b, d, d	Vrai	0	null
b	a, d, e, f	Vrai	1	a
c	e, f	Faux	∞	null
d	a, a, b, f	Vrai	1	a
e	b, c, f	Faux	∞	null
f	b, c, d, e	Faux	∞	null

$\mathrm{II} : a, b, d$

Parcours en largeur – exemple

u	voisins de u	marque	distance	père
a	b, d, d	Vrai	0	null
b	a, d, e, f	Vrai	1	a
c	e, f	Faux	∞	null
d	a, a, b, f	Vrai	1	a
e	b, c, f	Faux	∞	null
f	b, c, d, e	Faux	∞	null

$\mathrm{II} : a, b, d$

Parcours en largeur – exemple

u	voisins de u	marque	distance	père
a	b, d, d	Vrai	0	null
b	a, d, e, f	Vrai	1	a
c	e, f	Faux	∞	null
d	a, a, b, f	Vrai	1	a
e	b, c, f	Faux	∞	null
f	b, c, d, e	Faux	∞	null

II : b, d

Parcours en largeur – exemple

u	voisins de u	marque	distance	père
a	b, d, d	Vrai	0	null
b	a, d, e, f	Vrai	1	a
c	e, f	Faux	∞	null
d	a, a, b, f	Vrai	1	a
e	b, c, f	Faux	∞	null
f	b, c, d, e	Faux	∞	null

II : b, d

À compléter !

Parcours en largeur – exemple

u	voisins de u	marque	distance	père
a	b, d, d	Vrai	0	null
b	a, d, e, f	Vrai	1	a
c	e, f	Vrai	3	e
d	a, a, b, f	Vrai	1	a
e	b, c, f	Vrai	2	b
f	b, c, d, e	Vrai	2	b

II :

Parcours en largeur – vocabulaire

Découvrir un sommet = sommet reçoit le message

Visiter un sommet = sommet retransmet le message

Parcours en largeur – vocabulaire

Découvrir un sommet = sommet reçoit le message

Visiter un sommet = sommet retransmet le message

BLANC – sommet non découvert encore = sommet n'ayant pas encore reçu le message

GRIS – sommet découvert, mais pas visité = sommet ayant reçu le message, mais ne l'ayant pas encore retransmis

NOIR – sommet visité = somme ayant reçu et retransmis le message

Parcours en largeur

Proposition

Soit G un graphe et s un sommet de G. Pendant l'exécution de $PL(G, s)$, un sommet v est découvert (et puis visité) si et seulement si v est accessible depuis le sommet de départ s.

Les sommets sont découverts dans l'ordre croissant par rapport à la distance depuis s. D'ailleurs, la valeur $d(v)$ calculée pendant l'exécution de $PL(G, s)$ est la distance entre s et v.

Parcours en largeur

Proposition

Soit G un graphe et s un sommet de G. Soit C la composante connexe de G contenant s. Après une exécution de $PL(G, s)$, considérons l'ensemble d'arêtes

$$E = \{(u, pere(u)) | u \in V(C), u \neq s\}.$$

Le graphe $H = (V(C), E)$ est un arbre couvrant de C.

Complexité algorithmique

Étant donné deux algorithmes différents pour résoudre le même problème, pour en choisir le meilleur on cherche des moyens de comparer leur performance.

Une des moyens de mesure l'efficacité d'un algorithme (un programme) est la *complexité*.

La complexité est définie comme la quantité de ressources (temps et/ou espace de mémoire) nécessaire pour résoudre un problème algorithmique.

Complexité algorithmique

Lorsque la complexité d'un algorithme est étudié, pour déterminer le temps de calcul nécessaire, on considère

- le nombre d'exécutions des opérations élémentaires (telles que affectations de valeur à une variable, opérations arithmétiques, tests de comparaison, appels à des fonctions, etc.)
- au pire cas
- en fonction de la taille de l'entrée
- de point de vue asymptotique

Exemple : Complexité de parcours en largeur

Algorithme : Parcours en largeur BFS(G, s)

Données : graphe G à n sommets et m arêtes, sommet de départ s

File II (initialisée à vide), marque des sommets (initialisé à Faux)

début

- marque[s] $\leftarrow$ Vrai ;
- enfiler s à la fin de II ;
- **tant que** II *non vide* **faire**
  - $u \leftarrow$ tête(II) ;
  - **pour chaque** v *voisin de* u **faire**
    - **si** v *non marqué* **alors**
      - marque[v] $\leftarrow$ Vrai ;
      - enfiler v à la fin de II ;
    - **fin**
  - **fin**
  - défiler u de la tête de II ;
- **fin**

fin

Exemple : Complexité de parcours en largeur

Algorithme : Parcours en largeur BFS(G, s)

Données : graphe G à n sommets et m arêtes, sommet de départ s

File II (initialisée à vide), marque des sommets (initialisé à Faux)

début

- marque[s] $\leftarrow$ Vrai ;
- enfiler s à la fin de II ;
- **tant que** II *non vide* **faire**
  - $u \leftarrow$ tête(II) ;
  - **pour chaque** v *voisin de* u **faire**
    - **si** v *non marqué* **alors**
      - marque[v] $\leftarrow$ Vrai ;
      - enfiler v à la fin de II ;
    - **fin**
  - **fin**
  - défiler u de la tête de II ;
- **fin**

fin

Initialisation (ligne 0) : une opération par sommet

Exemple : Complexité de parcours en largeur

Algorithme : Parcours en largeur BFS(G, s)

Données : graphe G à n sommets et m arêtes, sommet de départ s

File II (initialisée à vide), marque des sommets (initialisé à Faux)

```
1  début
2      marque[s] ← Vrai ;
3      enfiler s à la fin de II ;
       tant que II non vide faire
           u ← tête(II) ;
           pour chaque v voisin de u faire
               si v non marqué alors
                   marque[v] ← Vrai ;
                   enfiler v à la fin de II ;
               fin
           fin
           défiler u de la tête de II ;
       fin
   fin
```

Lignes 2,3 : complexité constante

Exemple : Complexité de parcours en largeur

Algorithme : Parcours en largeur BFS(G, s)

Données : graphe G à n sommets et m arêtes, sommet de départ s

File II (initialisée à vide), marque des sommets (initialisé à Faux)

```
1  début
2      marque[s] ← Vrai ;
3      enfiler s à la fin de II ;
4      tant que II non vide faire
5          u ← tête(II) ;
6          pour chaque v voisin de u faire
7              si v non marqué alors
8                  marque[v] ← Vrai ;
9                  enfiler v à la fin de II ;
10             fin
11         fin
12         défiler u de la tête de II ;
13     fin
14 fin
```

Ligne 4 : test exécuté au maximum n fois

Exemple : Complexité de parcours en largeur

Algorithme : Parcours en largeur BFS(G, s)

Données : graphe G à n sommets et m arêtes, sommet de départ s

File II (initialisée à vide), marque des sommets (initialisé à Faux)

```
1  début
2      marque[s] ← Vrai ;
3      enfiler s à la fin de II ;
4      tant que II non vide faire
5          u ← tête(II) ;
6          pour chaque v voisin de u faire
7              si v non marqué alors
8                  marque[v] ← Vrai ;
9                  enfiler v à la fin de II ;
10             fin
11         fin
12         défiler u de la tête de II ;
13     fin
14 fin
```

Lignes 5 et 12 : chaque sommet est visité au plus une fois

Exemple : Complexité de parcours en largeur

Algorithme : Parcours en largeur BFS(G, s)

Données : graphe G à n sommets et m arêtes, sommet de départ s

File II (initialisée à vide), marque des sommets (initialisé à Faux)

début

```
    marque[s] ← Vrai ;
    enfiler s à la fin de II ;
    tant que II non vide faire
        u ← tête(II) ;
        pour chaque v voisin de u faire
            si v non marqué alors
                marque[v] ← Vrai ;
                enfiler v à la fin de II ;
            fin
        fin
        défiler u de la tête de II ;
    fin
```

fin

Ligne 6 : pour un u fixe, il y a $d(u)$ voisins, donc au plus $n - 1$

Exemple : Complexité de parcours en largeur

Algorithme : Parcours en largeur BFS(G, s)

Données : graphe G à n sommets et m arêtes, sommet de départ s

File II (initialisée à vide), marque des sommets (initialisé à Faux)

```
début
    marque[s] ← Vrai ;
    enfiler s à la fin de II ;
    tant que II non vide faire
        u ← tête(II) ;
        pour chaque v voisin de u faire
            si v non marqué alors
                marque[v] ← Vrai ;
                enfiler v à la fin de II ;
            fin
        fin
        défiler u de la tête de II ;
    fin
fin
```

Ligne 6 : or, . . .

Exemple : Complexité de parcours en largeur

Algorithme : Parcours en largeur BFS(G, s)

Données : graphe G à n sommets et m arêtes, sommet de départ s

File II (initialisée à vide), marque des sommets (initialisé à Faux)

```
début
    marque[s] ← Vrai ;
    enfiler s à la fin de II ;
    tant que II non vide faire
        u ← tête(II) ;
        pour chaque v voisin de u faire
            si v non marqué alors
                marque[v] ← Vrai ;
                enfiler v à la fin de II ;
            fin
        fin
        défiler u de la tête de II ;
    fin
fin
```

Ligne 6 : chaque arête est considérée au plus deux fois, donc

Exemple : Complexité de parcours en largeur

Algorithme : Parcours en largeur BFS(G, s)

Données : graphe G à n sommets et m arêtes, sommet de départ s

File II (initialisée à vide), marque des sommets (initialisé à Faux)

début

```
    marque[s] ← Vrai ;
    enfiler s à la fin de II ;
    tant que II non vide faire
        u ← tête(II) ;
        pour chaque v voisin de u faire
            si v non marqué alors
                marque[v] ← Vrai ;
                enfiler v à la fin de II ;
            fin
        fin
        défiler u de la tête de II ;
    fin
```

fin

Ligne 6 : l'affectation est exécutée au maximum $2m$ fois

Exemple : Complexité de parcours en largeur

Algorithme : Parcours en largeur BFS(G, s)

Données : graphe G à n sommets et m arêtes, sommet de départ s

File II (initialisée à vide), marque des sommets (initialisé à Faux)

```
1  début
2      marque[s] ← Vrai ;
3      enfiler s à la fin de II ;
4      tant que II non vide faire
5          u ← tête(II) ;
6          pour chaque v voisin de u faire
7              si v non marqué alors
8                  marque[v] ← Vrai ;
9                  enfiler v à la fin de II ;
10             fin
11         fin
12         défiler u de la tête de II ;
13     fin
14 fin
```

Ligne 7 : test exécuté au maximum $2m$ fois

Exemple : Complexité de parcours en largeur

Algorithme : Parcours en largeur BFS(G, s)

Données : graphe G à n sommets et m arêtes, sommet de départ s

File II (initialisée à vide), marque des sommets (initialisé à Faux)

```
1  début
2      marque[s] ← Vrai ;
3      enfiler s à la fin de II ;
4      tant que II non vide faire
5          u ← tête(II) ;
6          pour chaque v voisin de u faire
7              si v non marqué alors
8                  marque[v] ← Vrai ;
9                  enfiler v à la fin de II ;
10             fin
11         fin
12         défiler u de la tête de II ;
13     fin
14 fin
```

Lignes 8 et 9 : exécutées au maximum n fois

Exemple : Complexité de parcours en largeur

Algorithme : Parcours en largeur BFS(G, s)

Données : graphe G à n sommets et m arêtes, sommet de départ s

File II (initialisée à vide), marque des sommets (initialisé à Faux)

début

```
    marque[s] ← Vrai ;
    enfiler s à la fin de II ;
    tant que II non vide faire
        u ← tête(II) ;
        pour chaque v voisin de u faire
            si v non marqué alors
                marque[v] ← Vrai ;
                enfiler v à la fin de II ;
            fin
        fin
        défiler u de la tête de II ;
    fin
```

fin

Nombre d'opérations élémentaires : $3 + 5n + 4m$ au maximum

Exemple : Complexité de parcours en largeur

Algorithme : Parcours en largeur BFS(G, s)

Données : graphe G à n sommets et m arêtes, sommet de départ s

File II (initialisée à vide), marque des sommets (initialisé à Faux)

début

- marque[s] $\leftarrow$ Vrai ;
- enfiler s à la fin de II ;
- **tant que** II *non vide* **faire**
  - $u \leftarrow$ tête(II) ;
  - **pour chaque** v *voisin de* u **faire**
    - **si** v *non marqué* **alors**
      - marque[v] $\leftarrow$ Vrai ;
      - enfiler v à la fin de II ;
    - **fin**
  - **fin**
  - défiler u de la tête de II ;
- **fin**

fin

Complexité de l'algorithme : $O(n + m)$