Algorithmique des graphes

Cours 3 – Parcours en largeur

František Kardoš

frantisek.kardos@u-bordeaux.fr

Plan

- Parcours en largeur
- Complexité des algorithmes

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)

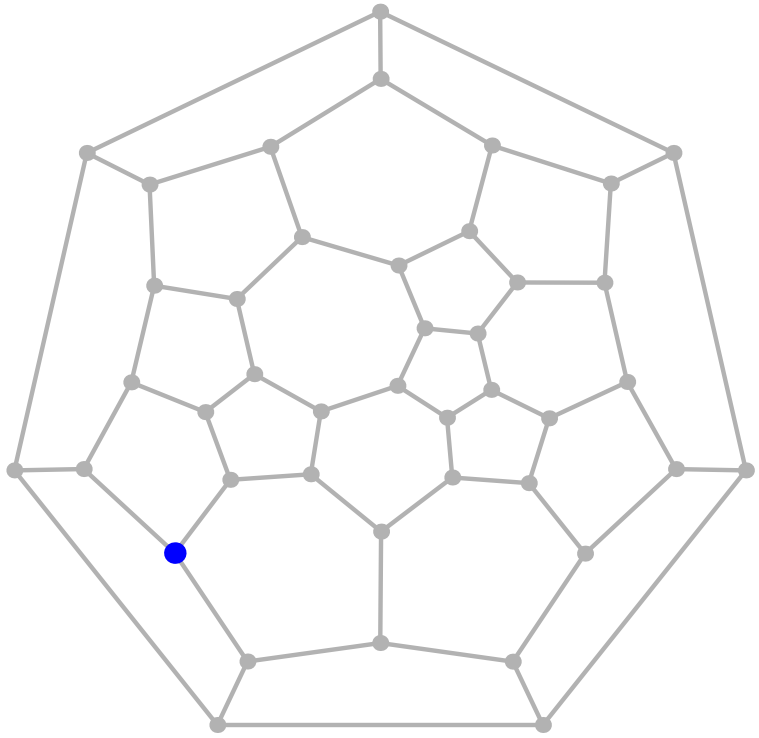
Un parcours en largeur explore le graphe à partir d'un sommet donné (sommet de départ ou sommet source).

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

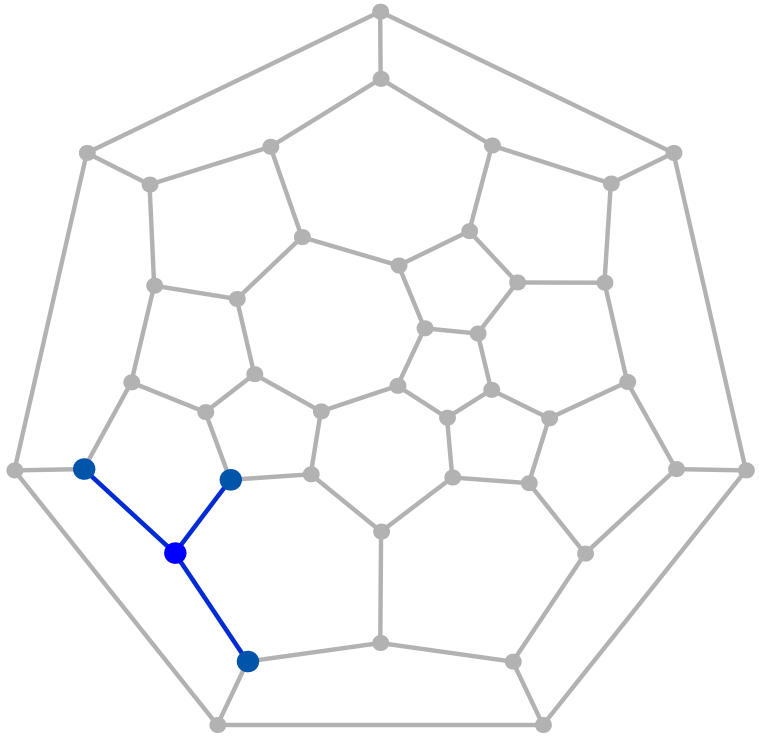
L'algorithme simule la transmission d'un message à partir d'un sommet source, en utilisant l'idée suivante :

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

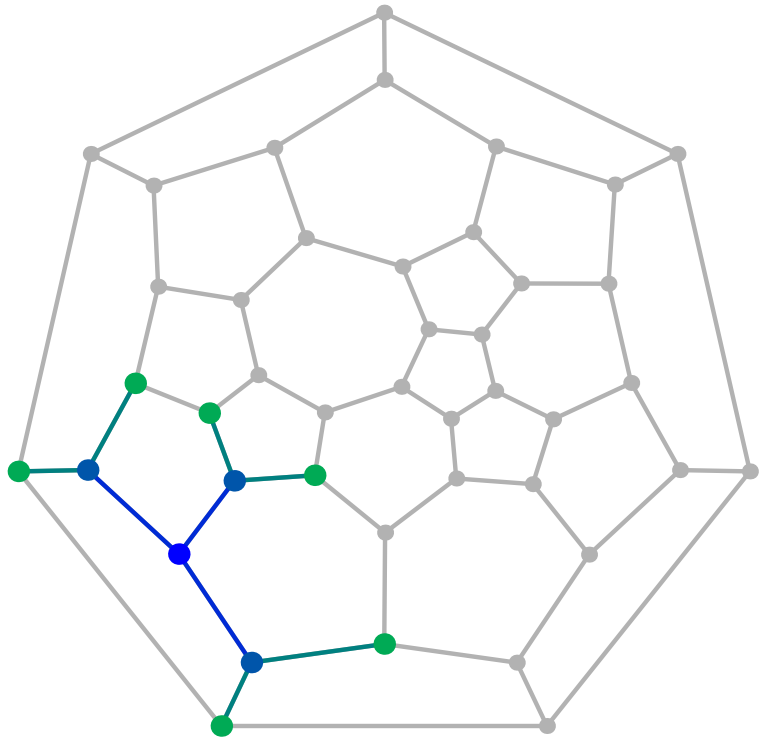
Parcours en largeur



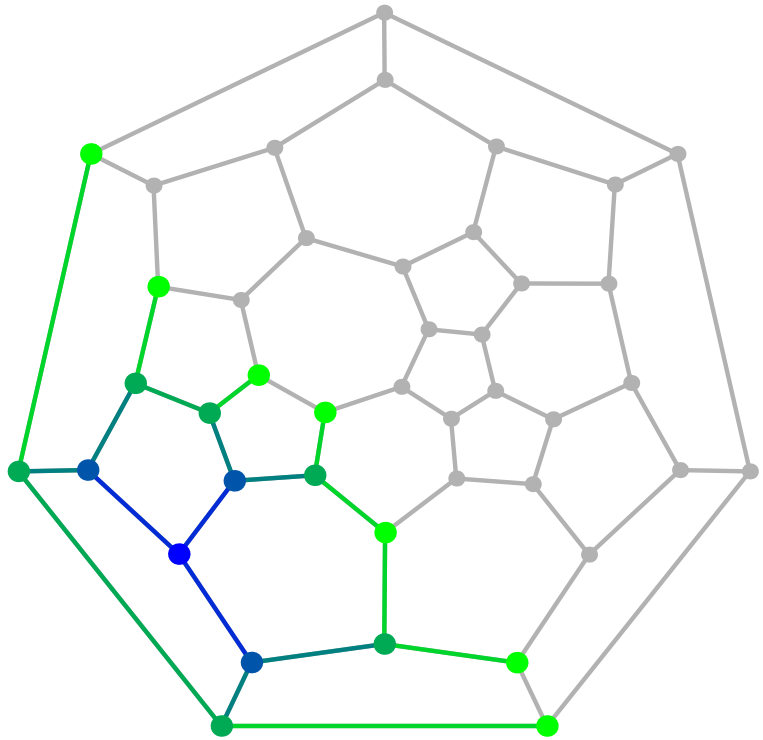
Parcours en largeur



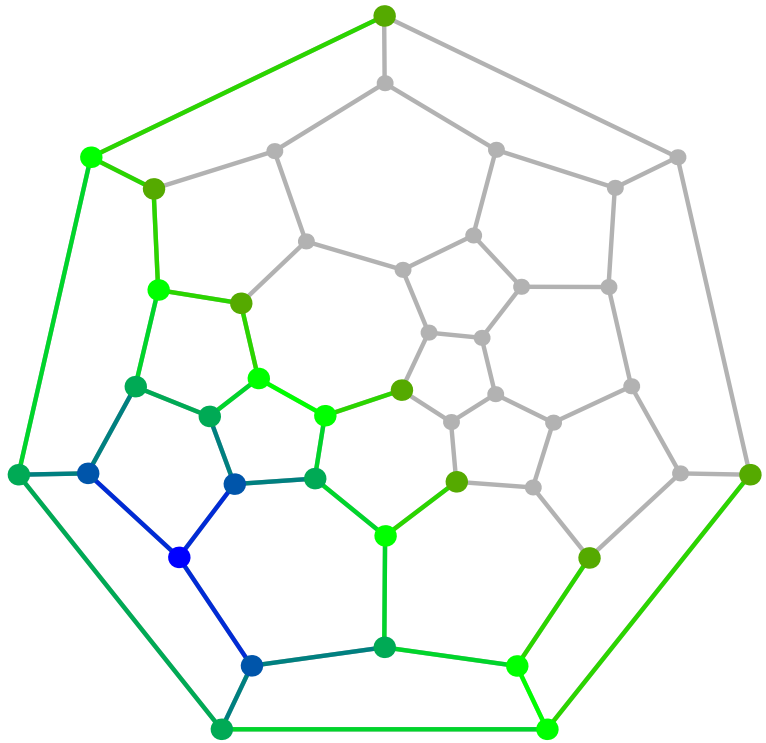
Parcours en largeur



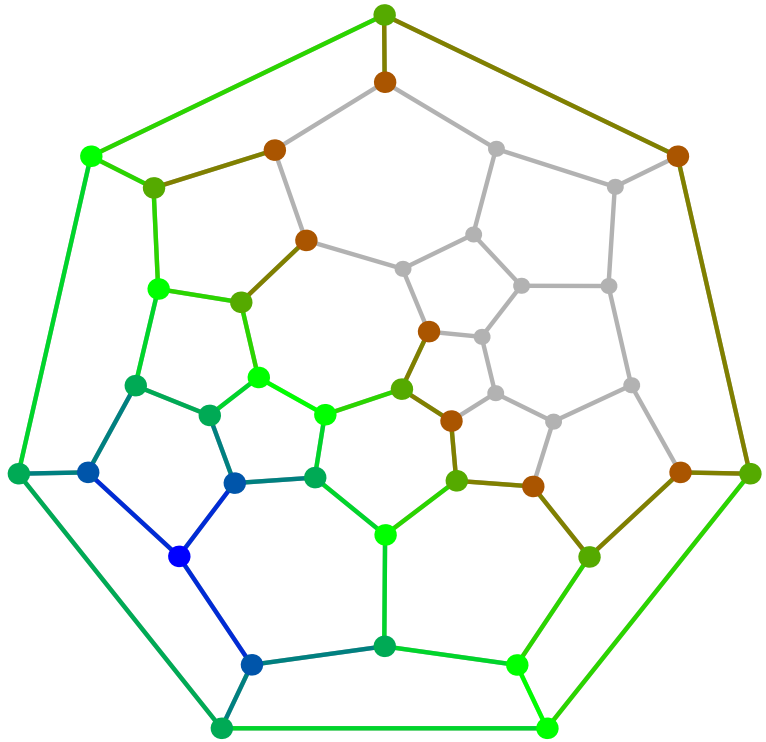
Parcours en largeur



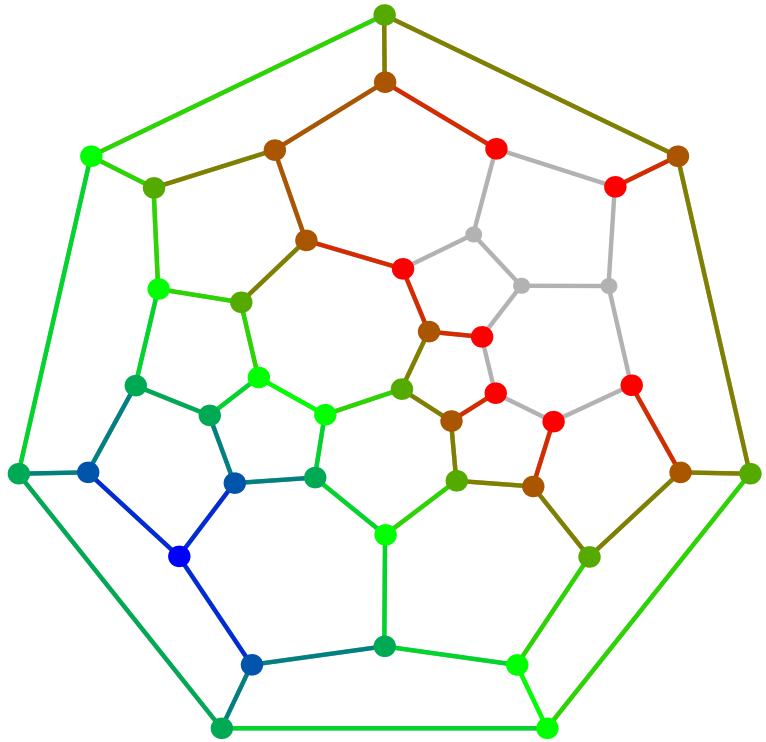
Parcours en largeur



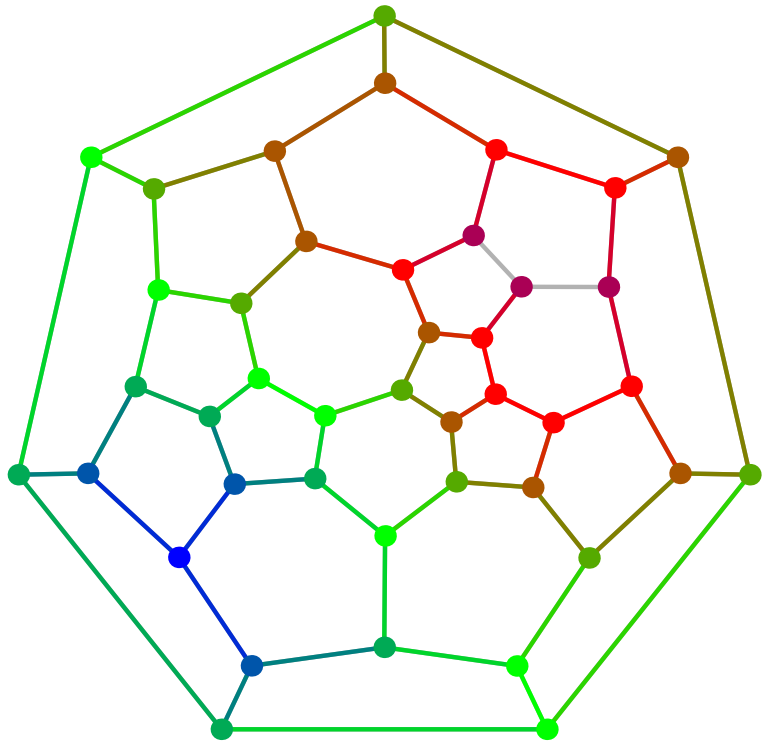
Parcours en largeur



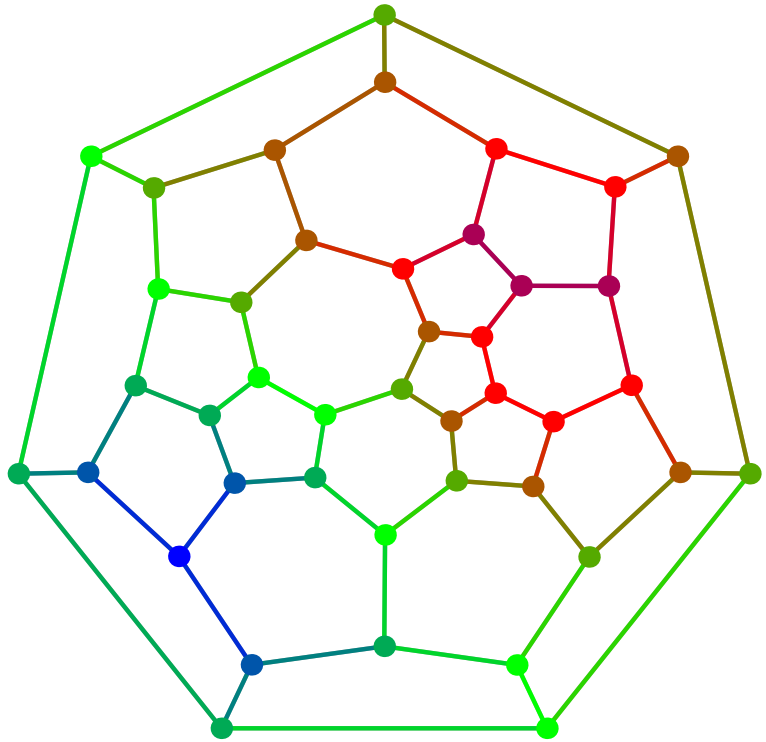
Parcours en largeur



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 1 : Parcours en largeur BFS(G, s)

Données : graphe G, sommet de départ s
File $\amalg$ (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 2 : 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 3 : Parcours en largeur BFS(G, s)

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

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

$\mathrm{II} : a, b$

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 C. 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 4 : 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
```

Exemple : Complexité de parcours en largeur

Algorithme 5 : 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 6 : 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 2,3 : complexité constante

Exemple : Complexité de parcours en largeur

Algorithme 7 : 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 8 : 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 9 : 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 10 : 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 6 : or, . . .

Exemple : Complexité de parcours en largeur

Algorithme 11 : 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 6 : chaque arête est considérée au plus deux fois, donc

Exemple : Complexité de parcours en largeur

Algorithme 12 : 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 6 : l'affectation est exécutée au maximum $2m$ fois

Exemple : Complexité de parcours en largeur

Algorithme 13 : 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 7 : test exécuté au maximum $2m$ fois

Exemple : Complexité de parcours en largeur

Algorithme 14 : 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 15 : 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 16 : 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 : $O(n+m)$

Complexité – éléments de notation

La notation grand O de Landau (O comme "Ordnung") signifie intuitivement que une fonction ne croît pas plus vite qu'une autre.

Généralement, on exprime la complexité d'un algorithme $T(n)$ (le nombre précis d'opérations élémentaires) par une fonction générique $f(n)$ portant l'information sur l'ordre de grandeur de la vitesse de croissance de $T(n)$.

On note $T(n) = O(f(n))$ s'il existe des constantes N et c telles que

$$\forall n > N \quad T(n) \leq c \cdot f(n).$$

Exemples

$$3 + 5n + 4m = O(n + m)$$

$$2n^2 + 3n + 5 = O(n^2)$$

$$2n^2 + 1000000 = O(n^2)$$

$$2n^2 + 1000000n = O(n^2)$$

$$n^2 = O(n^3)$$

$$n^3 \neq O(n^2)$$

$$\sqrt{n} = O(n)$$

$$n \neq O(\sqrt{n})$$

$$\sqrt{3n + 1} = O(\sqrt{n})$$

D'autres notations

Une fonction $T(n)$ est en $O(f(n))$, si l'ordre de croissance de $f(n)$ est une borne supérieur sur l'ordre de croissance de $T(n)$. Plus précisément, la limite

$$\lim_{n \to \infty} \frac{T(n)}{f(n)}$$

existe et elle est finie.

D'autres notations

Une fonction $T(n)$ est en $O(f(n))$, si l'ordre de croissance de $f(n)$ est une borne supérieur sur l'ordre de croissance de $T(n)$. Plus précisément, la limite

$$\lim_{n\to\infty} \frac{T(n)}{f(n)}$$

existe et elle est finie.

Une fonction $T(n)$ est en $o(f(n))$, si l'ordre de croissance de $T(n)$ est strictement plus petit que celui de $f(n)$. Plus précisément,

$$\lim_{n\to\infty} \frac{T(n)}{f(n)} = 0.$$

D'autres notations

Une fonction $T(n)$ est en $\Omega(f(n))$, si l'ordre de croissance de $f(n)$ est une borne inférieur sur l'ordre de croissance de $T(n)$. Plus précisément,

$$\lim_{n\to\infty} \frac{T(n)}{f(n)} > 0.$$

D'autres notations

Une fonction $T(n)$ est en $\Omega(f(n))$, si l'ordre de croissance de $f(n)$ est une borne inférieur sur l'ordre de croissance de $T(n)$. Plus précisément,

$$\lim_{n\to\infty} \frac{T(n)}{f(n)} > 0.$$

Finalement, une fonction $T(n)$ est en $\Theta(f(n))$, si l'ordre de croissance de $f(n)$ est asymptotiquement le même que celui de $T(n)$. Plus précisément,

$$0 < \lim_{n\to\infty} \frac{T(n)}{f(n)} < \infty.$$