Algorithmique des graphes - Cours 2

Olivier Baudon

Université de Bordeaux

20 septembre 2020

Notations asymptotyques

Notation O ("grand o")

$O(g(n)) = \{f(n) | \exists c \in \mathbb{R}^+, \exists n_0 \in \mathbb{N}, \forall n \geq n_0, 0 \leq f(n) \leq c \times g(n)\}$

Remarque : on ne notera pas $f(n) \in O(g(n))$, *mais*
$f(n) = O(g(n))$
Attention ! Cette notation "=" n'est pas symétrique !

Exemple

$3 * n^2 + 2 * n + 3 = O(n^2)$
Il suffit de prendre $c = 5$ et $n_0 = 2$.
On pourrait aussi dire que $3 * n^2 + 2 * n + 3 = O(n^3)$, *mais l'approximation est moins précise !*

Notations asymptotyques

Notation Ω

$\Omega(g(n)) = \{f(n) | \exists c \in \mathbb{R}^+, \exists n_0 \in \mathbb{N}, \forall n \geq n_0, 0 \leq c \times g(n) \leq f(n)\}$

Notation Θ

$\Theta(g(n)) = \{f(n) | \exists c_1, c_2 \in \mathbb{R}^+, \exists n_0 \in \mathbb{N}, \forall n \geq n_0, 0 \leq c_1 \times g(n) \leq f(n) \leq c_2 \times g(n)\}$

Théorèmes

$f(n) = O(g(n)) \Leftrightarrow g(n) = \Omega(f(n))$
$f(n) = \Theta(g(n)) \Leftrightarrow f(n) = O(g(n))$ et $f(n) = \Omega(g(n))$
$f(n) = \Theta(g(n)) \Leftrightarrow f(n) = O(g(n))$ et $g(n) = O(f(n))$

Taille mémoire selon la représentation

Matrice d'adjacence

La taille mémoire nécessaire pour représenter un graphe (orienté ou non) par sa matrice d'adjacence est en $O(n^2)$.

Matrice d'indicence

La taille mémoire nécessaire pour représenter un graphe (orienté ou non) par sa matrice d'incidence est en $O(n \times m)$.
Attention, si le graphe est orienté et possède une boucle, il est impossible de savoir sur quel sommet est la boucle.

Listes d'adjacences

La taille mémoire nécessaire pour représenter un graphe (orienté ou non) par ses listes d'adjacence est en $O(n + m)$.
Justification : il faut n listes et la somme des longueurs des listes est égal à $2 \times$ le nombre d'arêtes si le graphe est non orienté ou au nombre d'arcs si le graphe est orienté.

Complexité selon la représentation : Matrice d'adjacence

Existence d'une arête ou d'un arc entre deux sommets

A : matrice d'adjacence d'un graphe G orienté ou non
i, j : deux sommets de G
Teste si j est voisin ou successeur de i
SontVoisins(A, i, j**)** :

1: **retourner** $A[i, j] \neq 0$

Complexité : $O(1)$

Complexité selon la représentation - Matrice d'adjacence

Degré d'un sommet

A : matrice d'adjacence d'un graphe G non orienté
i : sommet de G
Degré(A, i) :

```
1: degre ← 0
2: pour j de 1 à n faire
3:     si i ≠ j alors
4:         degre ← degre + A[i, j]
5:     sinon
6:         degre ← degre + 2 × A[i, j]
7:     fin si
8: fin pour
9: retourner degre
```

Complexité : $O(n)$

Complexité selon la représentation - Matrice d'adjacence

Degrés de tous les sommets

A : matrice d'adjacence d'un graphe G non orienté

Degres(A) :

1: **pour** i de 1 à n **faire**
2: $degre[i] \leftarrow 0$
3: **pour** j de 1 à n **faire**
4: **si** $i \neq j$ **alors**
5: $degre[i] \leftarrow degre[i] + A[i,j]$
6: **sinon**
7: $degre[i] \leftarrow degre[i] + 2 \times A[i,j]$
8: **fin si**
9: **fin pour**
10: **fin pour**
11: **retourner** $degre$

Complexité : $O(n^2)$

Complexité selon la représentation : Matrice d'incidence

Existence d'une arête

B : matrice d'incidence d'un graphe G non orienté
i, j : deux sommets de G
SontVoisins(B, i, j) :

1: **pour** e de 1 à m **faire**
2: **si** $B[i, e] \neq 0$ **et** $B[j, e] \neq 0$ **alors**
3: **retourner** VRAI
4: **fin si**
5: **fin pour**
6: **retourner** FAUX

Complexité : $O(m)$
Pour tester l'existence d'un arc de i vers j, il suffit de remplacer la condition : $B[i, e] \neq 0$ et $B[j, e] \neq 0$ par : $B[i, e] = -1$ et $B[j, e] = 1$
Par contre, il est impossible avec la matrice d'incidence d'un graphe orienté de tester s'il existe une boucle sur un sommet i donné.

Complexité selon la représentation - Matrice d'incidence

Degré d'un sommet

B : matrice d'incidence d'un graphe G non orienté
i : sommet de G
Degré(B, i) :

1: $degre \leftarrow 0$
2: **pour** e de 1 à m **faire**
3: $\quad degre \leftarrow degre + B[i, e]$
4: **fin pour**
5: **retourner** $degre$

Complexité : $O(m)$

Complexité selon la représentation - Matrice d'incidence

Degrés de tous les sommets

B : matrice d'incidence d'un graphe G non orienté

Degres(A) :

1: **pour** i de 1 à n **faire**
2: $\quad degres[i] \leftarrow 0$
3: $\quad$ **pour** e de 1 à m **faire**
4: $\quad\quad degre[i] \leftarrow degre[i] + B[i, e]$
5: $\quad$ **fin pour**
6: **fin pour**
7: **retourner** $degres$

Complexité : $O(n \times m)$

Complexité selon la représentation : Listes d'adjacence

Existence d'une arête

G : un graphe (orienté ou non)
u, v : deux sommets de G
Teste si v est voisin ou successeur de u
SontVoisins(G, u, v) :

1: **pour tout** $w \in Adj(u)$ **faire**
2: **si** $w = v$ **alors**
3: **retourner** VRAI
4: **fin si**
5: **fin pour**
6: **retourner** FAUX

Complexité : $O(\Delta(G)) = O(n)$

Complexité selon la représentation : Listes d'adjacence

Degré d'un sommet

G : un graphe non orienté
u : un sommet de G
Degré(G, u) :

```
1: degre ← 0
2: pour tout v ∈ Adj(u) faire
3:     si v ≠ u alors
4:         degre(u) ← degre(u) + 1
5:     sinon
6:         degre(u) ← degre(u) + 2
7:     fin si
8: fin pour
9: retourner degre
```

Complexité : $O(\Delta(G)) = O(n)$

Complexité selon la représentation : Listes d'adjacence

Degrés de tous les sommets

G : un graphe non orienté
Degres(G) :

```
1:  pour tout u ∈ V(G) faire
2:      degres(u) ← 0
3:      pour tout v ∈ Adj(u) faire
4:          si v ≠ u alors
5:              degres(u) ← degres(u) + 1
6:          sinon
7:              degres(u) ← degres(u) + 2
8:          fin si
9:      fin pour
10: fin pour
11: retourner degres
```

Complexité : $O(n+m)$

Parcours

Introduction

Il existe principalement deux algorithmes permettant de parcourir un graphe (sommets et arêtes ou arcs) en temps linéaire, c'est à dire en $O(n+m)$:

- le parcours en largeur, que l'on appellera également BFS pour "Breadth-First-Search", basé sur l'utilisation d'un file ;
- le parcours en profondeur, que l'on appellera également DFS pour "Depth-First-Search", basé sur l'utilisation d'un pile.

Parcours en largeur

Principe

On examine les sommets du graphe à partir d'un sommet source s. Tous les sommets accessibles depuis s par une chaîne ou un chemin seront visités, et seulement ceux-là.
Les sommets sont examinés un par un. Ils sont blancs tant qu'ils n'ont pas été visités, puis gris quand ils sont présents dans la file d'attente, puis noirs après avoir été sortis de la file d'attente.
Quand on examine un sommet, on ajoute tous ses voisins ou successeurs dans la file d'attente et l'on met à jour leur distance par rapport au sommet source.

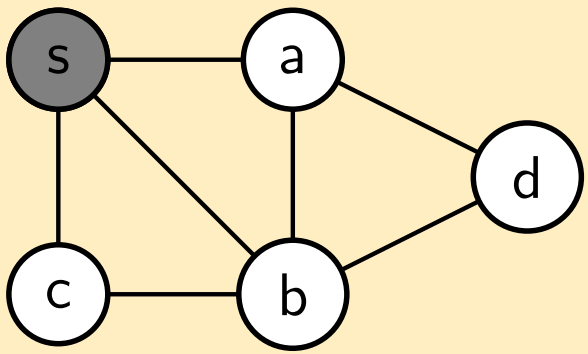
Énoncé

Voir le document "Algorithmes de graphes" page 2.

Parcours en largeur

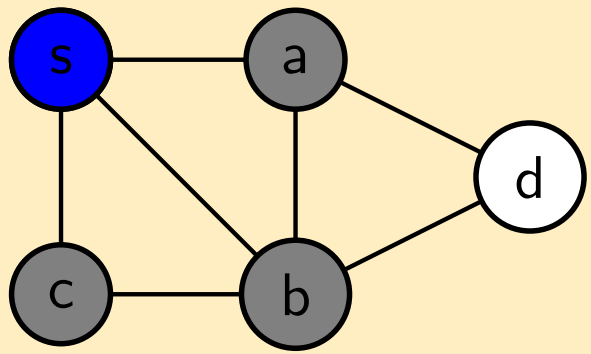
Exemple dans le cas non orienté

La couleur "NOIR" sera affichée avec du bleu afin de laisser l'étiquette lisible.



$F = \{s\}$

sommet v	*s*	*a*	*b*	*c*	*d*
$d(v)$	0	∞	∞	∞	∞

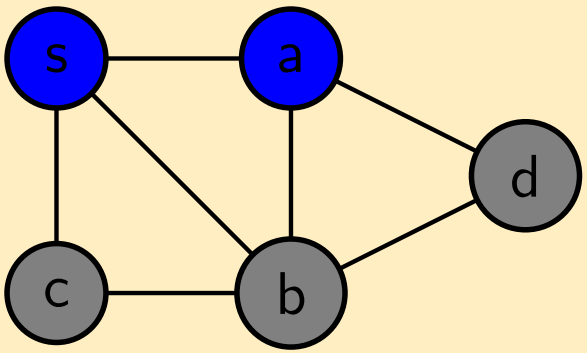


$F = \{a, b, c\}$

sommet v	*s*	*a*	*b*	*c*	*d*
$d(v)$	0	1	1	1	∞

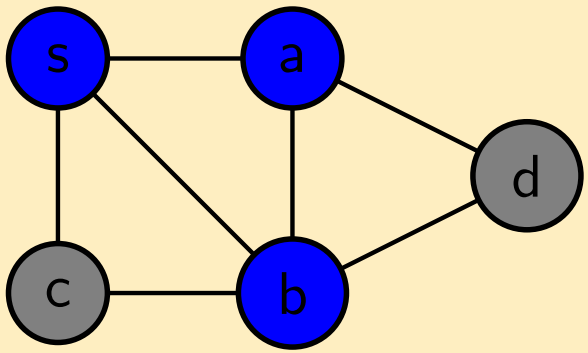
Parcours en largeur

Exemple dans le cas non orienté



$F = \{b, c, d\}$

sommet v	s	a	b	c	d
$d(v)$	0	1	1	1	2

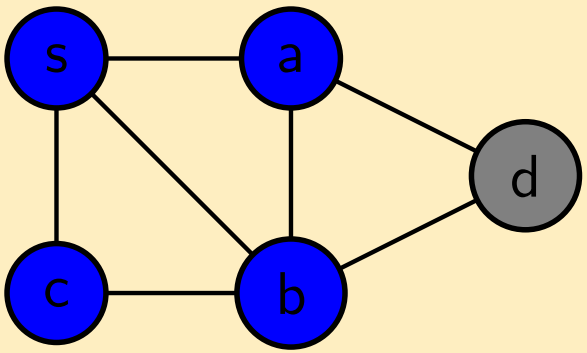


$F = \{c, d\}$

sommet v	s	a	b	c	d
$d(v)$	0	1	1	1	2

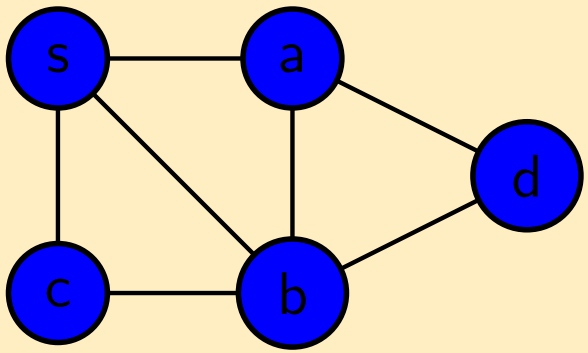
Parcours en largeur

Exemple dans le cas non orienté



$F = \{d\}$

sommet v	s	a	b	c	d
$d(v)$	0	1	1	1	2

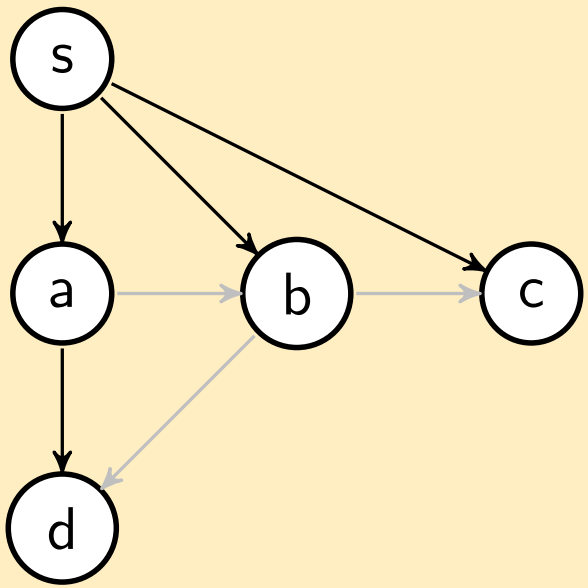


$F = \{\}$

sommet v	s	a	b	c	d
$d(v)$	0	1	1	1	2

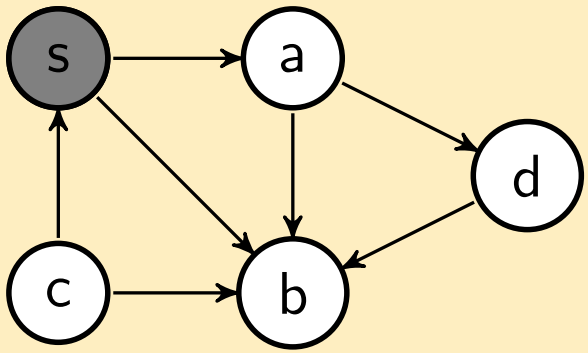
Parcours en largeur

Arborescence du parcours en largeur



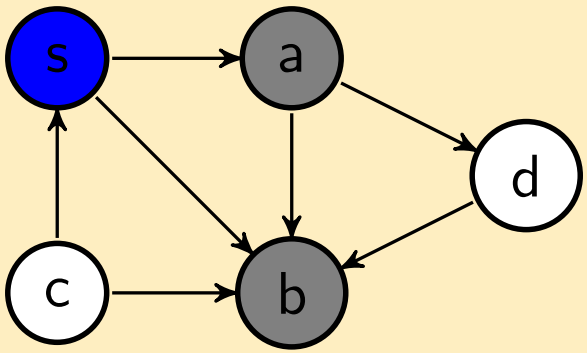
Parcours en largeur

Exemple dans le cas orienté



$F = \{s\}$

sommet v	s	a	b	c	d
$d(v)$	0	∞	∞	∞	∞

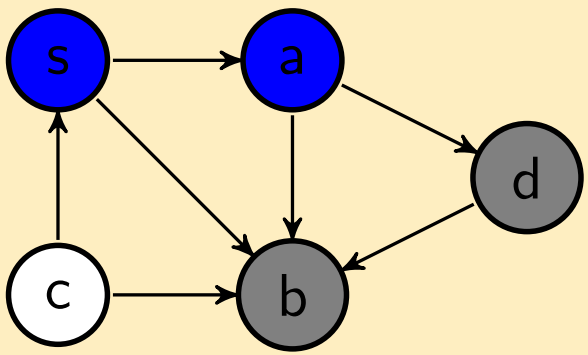


$F = \{a, b\}$

sommet v	s	a	b	c	d
$d(v)$	0	1	1	∞	∞

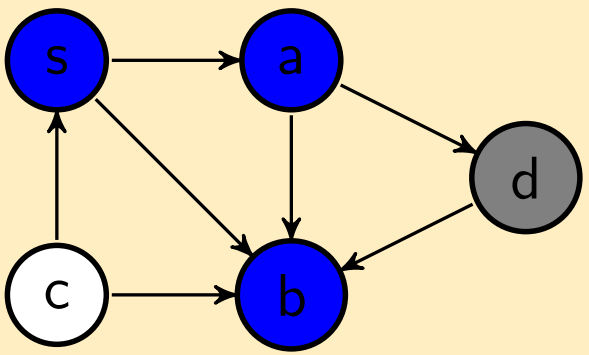
Parcours en largeur

Exemple dans le cas orienté



$F = \{b, d\}$

sommet v	s	a	b	c	d
$d(v)$	0	1	1	∞	2

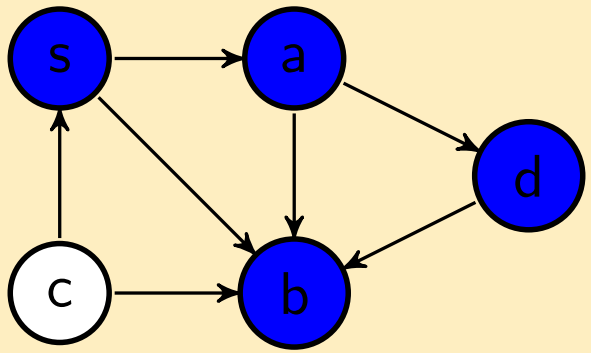


$F = \{d\}$

sommet v	s	a	b	c	d
$d(v)$	0	1	1	∞	2

Parcours en largeur

Exemple dans le cas orienté

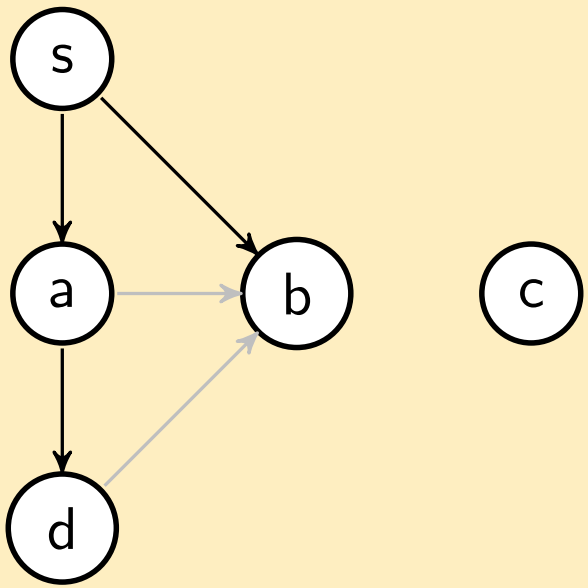


$F = \{\}$

sommet v	s	a	b	c	d
$d(v)$	0	1	1	∞	2

Parcours en largeur

Arborescence du parcours en largeur



Parcours en largeur

Complexité

L'algorithme de parcours en largeur a une complexité en $O(n+m)$. En effet, chaque sommet est introduit au plus une fois dans la file (tous uniquement si tous les sommets sont accessibles depuis s par une chaîne ou un chemin) et pour chaque sommet extrait de la file, on parcourt toutes ses arêtes incidentes (ou arc sortant).

Parcours en largeur

Validité

L'objectif du parcours en largeur est de calculer $d(v)$ qui contiendra en sortie $d(s, v)$: la distance dans le graphe G entre s et tout sommet v du graphe (∞ s'il n'existe pas de chaîne ou de chemin de s à v).

Lemme

Pour toute arête ou arc uv, $d(s, v) \leq d(s, u) + 1$.
En effet, si u est accessible à partir de s, alors le plus court chemin de s à v ne peut être plus long que le plus court chemin de s à u prolongé de uv.
Si u n'est pas accessible, alors $d(s, u) = \infty$ et dans tous les cas $d(s, v) \leq \infty + 1$.

Parcours en largeur

Validité : pour tout sommet $v, d(v) \geq d(s, v)$

On utilise une récurrence sur le nombre d'insertion de sommets dans F.
Au départ, seul s est introduit dans F et $d(s) = d(s, s) = 0$.
Soit w un sommet blanc, voisin d'un sommet u. Lorsque les voisins de u sont étudiés, par hypothèse, $d(u) \geq d(s, u)$.
Donc, quand w va être inséré dans F,
$d(w) = d(u) + 1 \geq d(s, u) + 1$. Or d'après le lemme précédent, on sait que $d(s, w) \leq d(s, u) + 1$. Donc $d(w) \geq d(s, w)$.

Parcours en largeur

Validité : Si $F = \{v_1, \ldots, v_r\}$, alors pour tout i, $d(v_i) \leq d(v_{i+1})$ et $d(v_r) \leq d(v_1) + 1$

On utilise une récurrence sur les opérations sur la file F.

- Au départ, $F = \{s\}$ et l'hypothèse est vérifiée.
- Quand on retire le sommet v_1 en tête de file, v_2 devient la tête de file. Par hypothèse,
  - $d(v_1) \leq d(v_2) \leq \ldots \leq d(v_r)$
  - $d(v_r) \leq d(v_1) + 1$

  donc
  - $d(v_r) \leq d(v_2) + 1$ car $d(v_r) \leq d(v_1) + 1$ et $d(v_1) \leq d(v_2)$
  - $d(v_2) \leq \ldots \leq d(v_r)$.

Parcours en largeur

Validité : Si $F = \{v_1, \ldots, v_r\}$, alors pour tout i, $d(v_i) \leq d(v_{i+1})$ et $d(v_r) \leq d(v_1) + 1$

On utilise une récurrence sur les opérations sur la file F.

- Au départ, $F = \{s\}$ et l'hypothèse est vérifiée.
- Quand on retire le sommet v_1 en tête de file, v_2 devient la tête de file. Par hypothèse,
  - $d(v_1) \leq d(v_2) \leq \ldots \leq d(v_r)$
  - $d(v_r) \leq d(v_1) + 1$

  donc
  - $d(v_r) \leq d(v_2) + 1$ car $d(v_r) \leq d(v_1) + 1$ et $d(v_1) \leq d(v_2)$
  - $d(v_2) \leq \ldots \leq d(v_r)$.

Parcours en largeur

Validité : Si deux sommets u et v sont enfilés dans F, et u est enfilé avant v, alors $d(u) \leq d(v)$.

Immédiat d'après le lemme précédent et en remarquant que la valeur $d(x)$ n'est jamais modifiée pour tout sommet x une fois qu'elle a été initialisée à une valeur finie.

Parcours en largeur

Validité : Après l'exécution de PL(G, s), tout sommet accessible à partir de s est visité, et pour tout sommet v, $d(v) = d(s, v)$. De plus, pour tout sommet v accessible à partir de s, alors un des plus courts chemins de s à v est un plus court chemin de s à $pere(v)$ complété par l'arête ou l'arc $pere(v)v$.

Supposons qu'il existe un sommet v tel que $d(v) \neq d(s, v)$.
Considérons le sommet v dans ce cas avec la plus petite valeur $d(v)$ possible.
Il est clair que $v \neq s$.
D'après un lemme précédent, on sait que $d(v) \geq d(s, v)$. Donc $d(v) > d(s, v)$. De plus, v est accessible depuis s, sinon $d(v) = \infty = d(s, v)$.
Soit u un sommet précédent v sur un plus court chemin de s à v.
Comme $d(s, u) = d(s, v) - 1$, on a bien $d(u) = d(s, u)$.
Donc $d(v) > d(s, v) = d(s, u) + 1 = d(u) + 1$.

Suite de la preuve

Quand u devient tête de file de F, si v est blanc, on aura $d(v) = d(u) + 1$ ce qui contredit l'hypothèse $d(v) > d(u) + 1$.
Si v est noir, alors $d(v) \leq d(u)$ ce qui contredit $d(v) > d(u) + 1$.
Si v est gris, alors il a été inséré à partir d'un sommet w inséré avant u et donc $d(w) \leq d(u)$ et $d(v) = d(w) + 1$. Donc $d(v) \leq d(u) + 1$ ce qui contredit $d(v) > d(u) + 1$.
Conclusion, v n'existe pas et donc pour tout sommet, $d(v) = d(s, v)$.
Tout sommet v accessible sera découvert, sinon on aurait $d(v) = \infty > d(s, v)$. De plus, si $pere(v) = u$ alors $d(v) = d(u) + 1$ et on aura bien un plus court chemin de s à v en prenant le plus court chemin de s à u complété de l'arête ou de l'arc uv.

Parcours en largeur

Arbre du parcours en largeur

Soit T le sous-graphe partiel de G avec :

- $V(T) = \{v \in V(G) | pere(v) \neq NIL\} \cup \{s\}$
- $E(T) = \{uv | u = pere(v), v \in V(T), v \neq s\}$

T est l'arbre de parcours en largeur de G.

Arêtes du co-arbre

Si uv est une arête de G n'appartenant pas à l'arbre du parcours en largeur, alors $d(u) - 1 \leq d(v) \leq d(u) + 1\}$.

Graphe biparti

Graphe biparti

Un graphe G non orienté est dit biparti si et seulement si il existe une partition de $V(G)$ en deux sous-ensembles A et B tels que les sous-graphes induits G_A et G_B ne contiennent aucune arête.
En d'autres termes, pour toute arête de G
$uv \in E(G) \Rightarrow (u \in A \Leftrightarrow v \in B)$

Parcours en largeur d'un graphe biparti

Si G est biparti, alors pour tout arête uv de G, $d(u) \neq d(v)$. En d'autres termes, les arêtes du co-arbre de l'arbre de parcours en profondeur reliront toujours deux sommets de niveau consécutifs.