M1 Info - Graphes et programmation dynamique

Cours 2 - Plus courts chemins a origine unique

BFS et Dijkstra

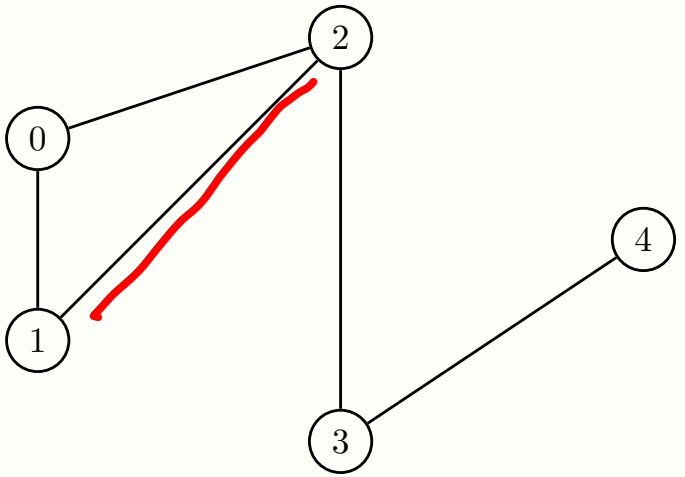
Semestre Automne 2022-2023 - Université Côte D'azur

Christophe Crespelle
christophe.crespelle@univ-cotedazur.fr

UNIVERSITÉ CÔTE D'AZUR

DIGITAL SYSTEMS FOR HUMANS
GRADUATE SCHOOL AND RESEARCH

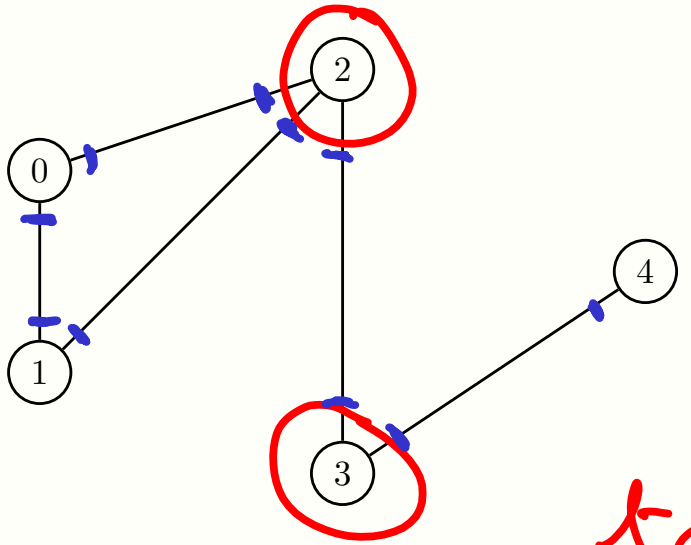
Representation des graphes en memoire



- Matrice d'adjacence

	0	**1**	**2**	**3**	**4**
0	0	1	1	0	0
1	1	0	1	0	0
2	1	1	0	1	0
3	0	0	1	0	1
4	0	0	0	1	0

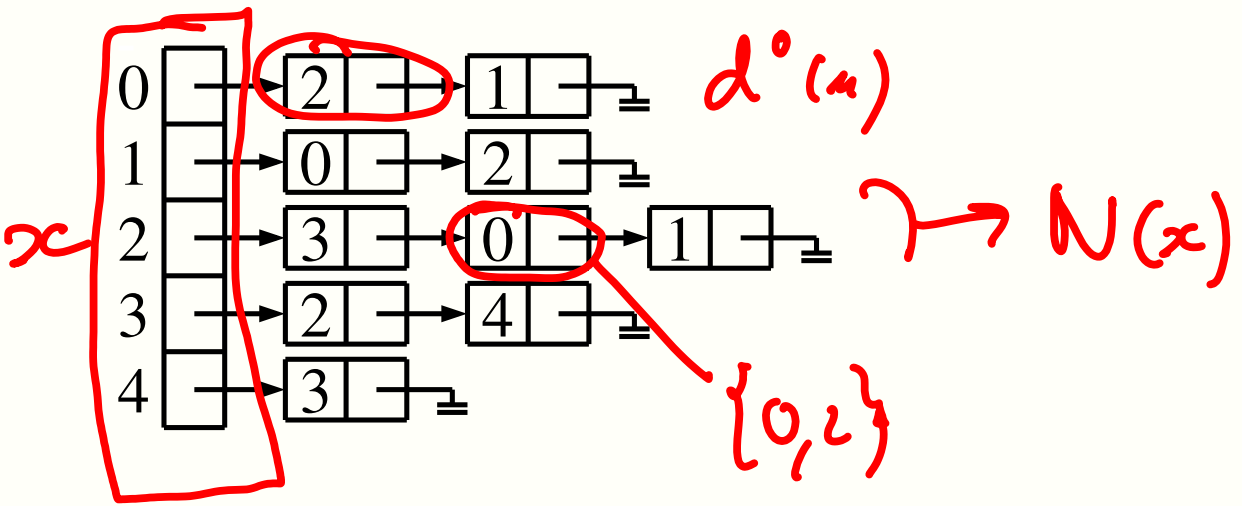
Representation des graphes en memoire



$\sum_{u \in V} d^0(u) = 2m$

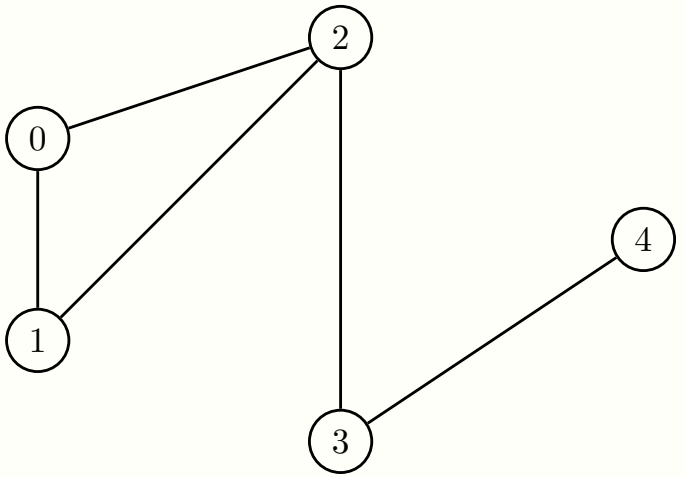
taille : $n + 2m$

- Listes d'adjacence

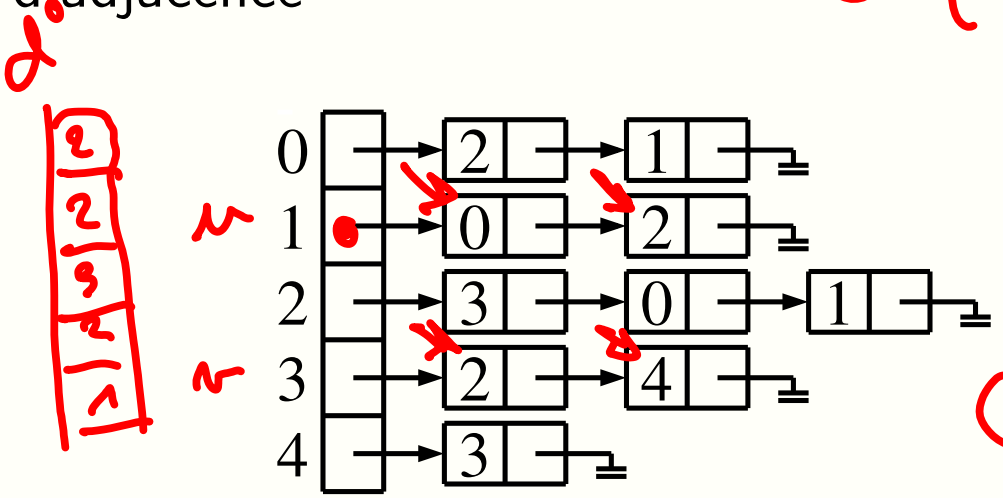


x → $N(x)$

Representation des graphes en memoire



- Listes d'adjacence

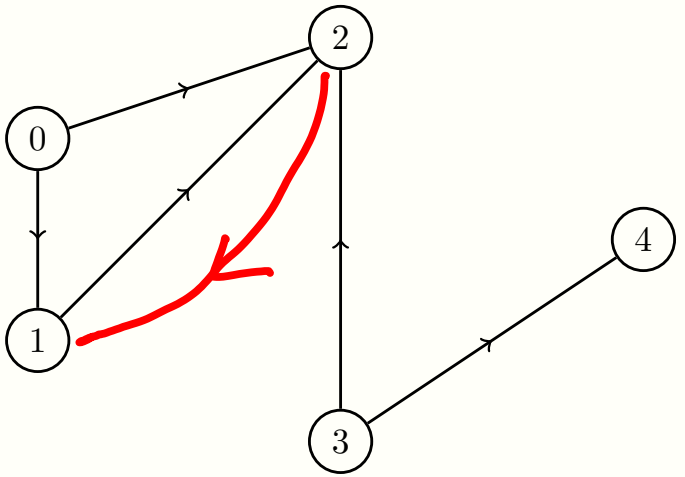


$O(\min\{d^\circ(u), d^\circ(v)\})$

$O(d^\circ(u))$

$O(d^\circ(v))$

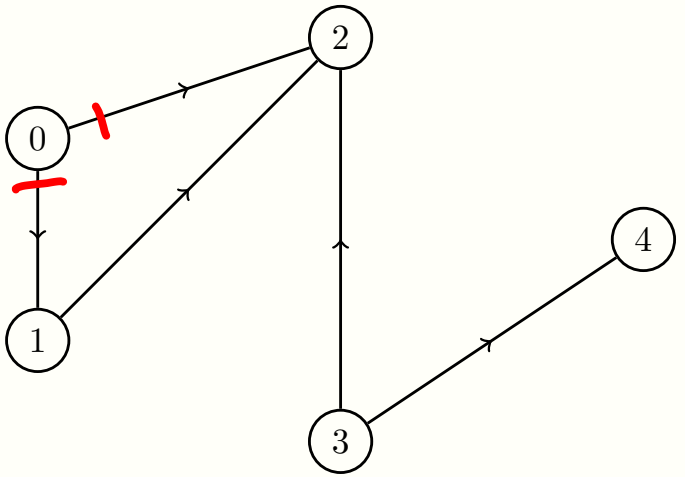
Representation des graphes orientes en memoire



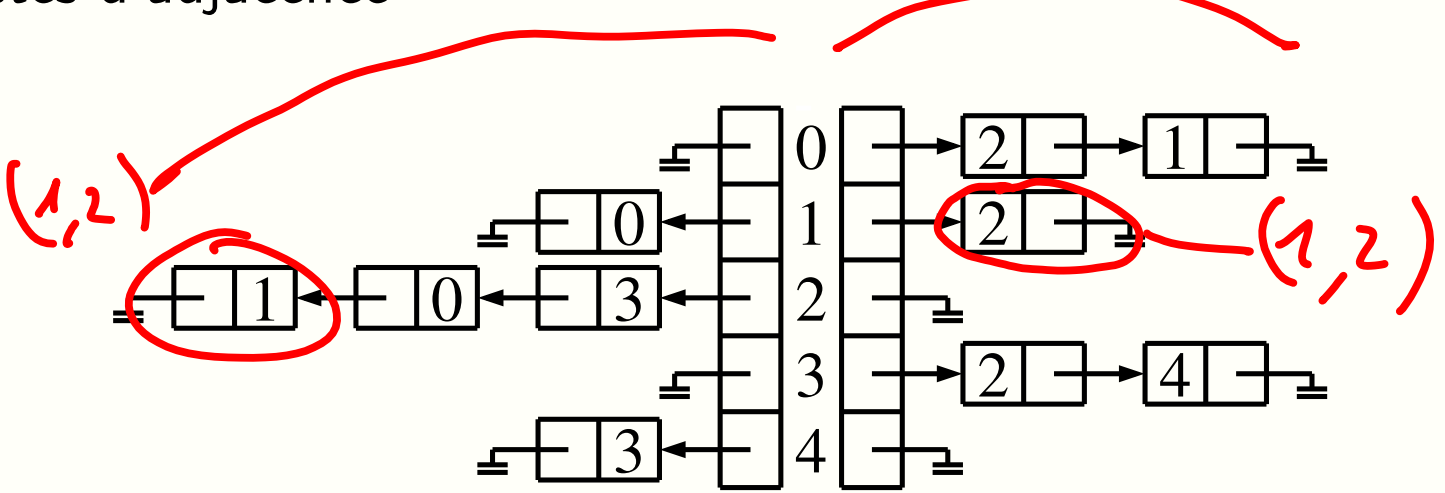
- Matrice d'adjacence

	0	**1**	**2**	**3**	**4**
0	0	1	1	0	0
1	0	0	1	0	0
2	0	0	0	0	0
3	0	0	1	0	1
4	0	0	0	0	0

Representation des graphes orientes en memoire



- Listes d'adjacence



Comparaison des performances des deux representations

Un graphe G a n sommets et m aretes

- En temps
  - Requete d'adjacence : u et v sont-ils adjacents ?

Comparaison des performances des deux representations

Un graphe G a n sommets et m aretes

- En temps
  - Requete d'adjacence : u et v sont-ils adjacents ?
    - Matrice : $O(1)$

Comparaison des performances des deux representations

Un graphe G a n sommets et m aretes

- En temps
  - Requete d'adjacence : u et v sont-ils adjacents ?
    - Matrice : $O(1)$
    - Listes : $O(min\{d(u), d(v)\})$

Comparaison des performances des deux representations

Un graphe G a n sommets et m aretes

- En temps
  - Requete d'adjacence : u et v sont-ils adjacents ?
    - Matrice : $O(1)$
    - Listes : $O(min\{d(u), d(v)\})$
  - Requete de voisinage : tous les voisins de u ?

Comparaison des performances des deux representations

Un graphe G a n sommets et m aretes

- En temps
  - Requete d'adjacence : u et v sont-ils adjacents ?
    - Matrice : $O(1)$
    - Listes : $O(min\{d(u), d(v)\})$
  - Requete de voisinage : tous les voisins de u ?
    - Matrice : $O(n)$

Comparaison des performances des deux representations

Un graphe G a n sommets et m aretes

- En temps
  - Requete d'adjacence : u et v sont-ils adjacents ?
    - Matrice : $O(1)$ optimal
    - Listes : $O(min\{d(u), d(v)\})$
  - Requete de voisinage : tous les voisins de u ?
    - Matrice : $O(n)$
    - Listes : $O(d(u))$ optimal

Comparaison des performances des deux representations

Un graphe G a n sommets et m aretes

- En temps
  - Requete d'adjacence : u et v sont-ils adjacents ?
    - Matrice : $O(1)$
    - Listes : $O(min\{d(u), d(v)\})$
  - Requete de voisinage : tous les voisins de u ?
    - Matrice : $O(n)$
    - Listes : $O(d(u))$
- En espace

Comparaison des performances des deux representations

Un graphe G a n sommets et m aretes

- En temps
  - Requete d'adjacence : u et v sont-ils adjacents ?
    - Matrice : $O(1)$
    - Listes : $O(min\{d(u), d(v)\})$
  - Requete de voisinage : tous les voisins de u ?
    - Matrice : $O(n)$
    - Listes : $O(d(u))$
- En espace
  - Matrice : $O(n^2)$

Comparaison des performances des deux representations

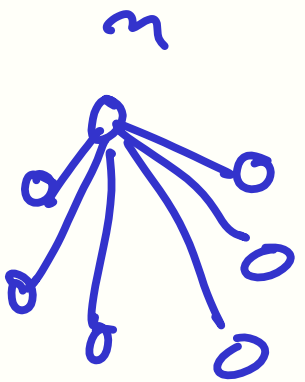
Un graphe G a n sommets et m aretes

n^2 bits

$m \log n$ bits

En moyenne $m \sim \frac{n^2}{4}$

- En temps
  - Requete d'adjacence : u et v sont-ils adjacents ?
    - Matrice : $O(1)$
    - Listes : $O(min\{d(u), d(v)\})$
  - Requete de voisinage : tous les voisins de u ?
    - Matrice : $O(n)$
    - Listes : $O(d(u))$



$O(n^2)$ bits

$O(m)$ entiers $\log n$ bits.

plus compacte en moyenne

- En espace
  - Matrice : $O(n^2)$ bits
  - Listes : $O(n+m)$ entiers

$m \log n$

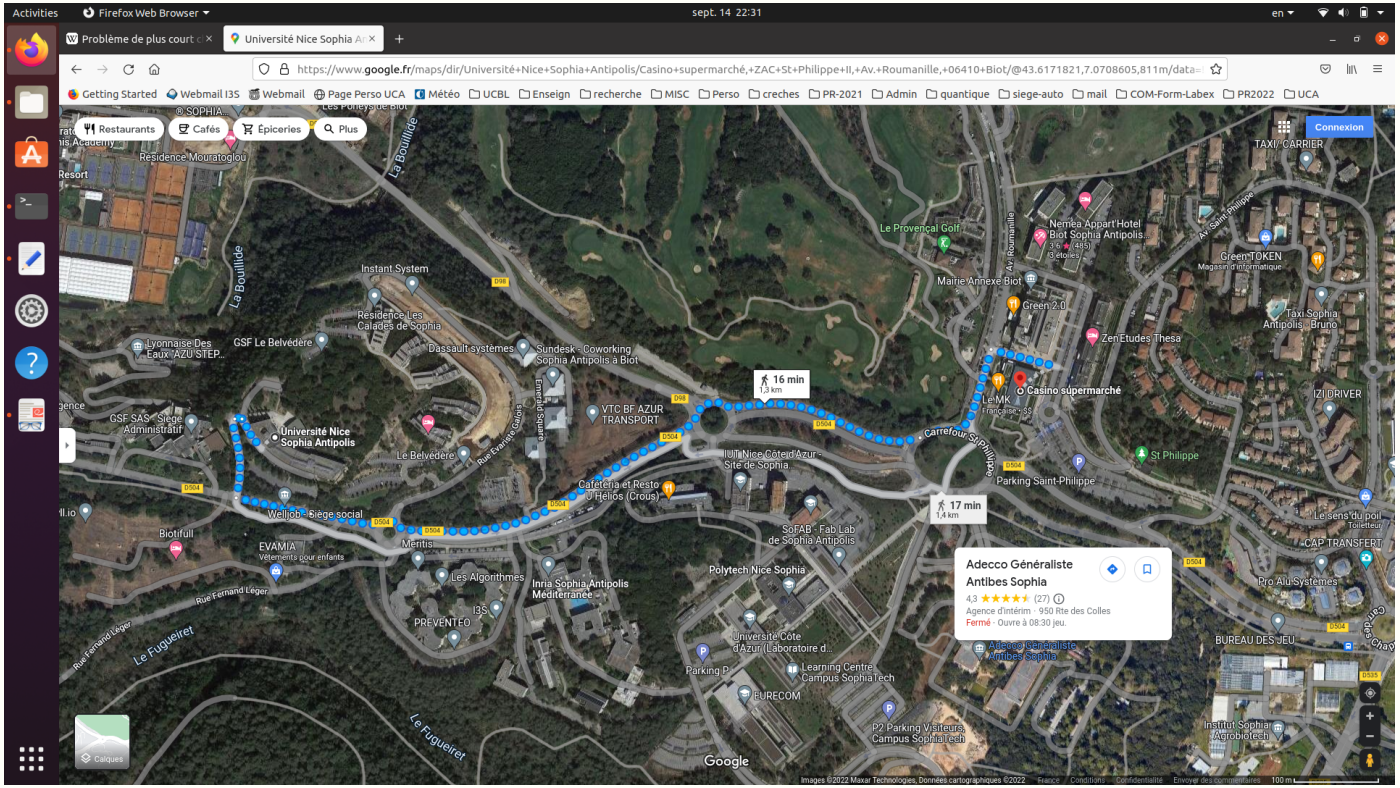
$m + \frac{1}{2} n^2$

$m \leq \frac{(n-1)\,n}{2} = O(n^2)$

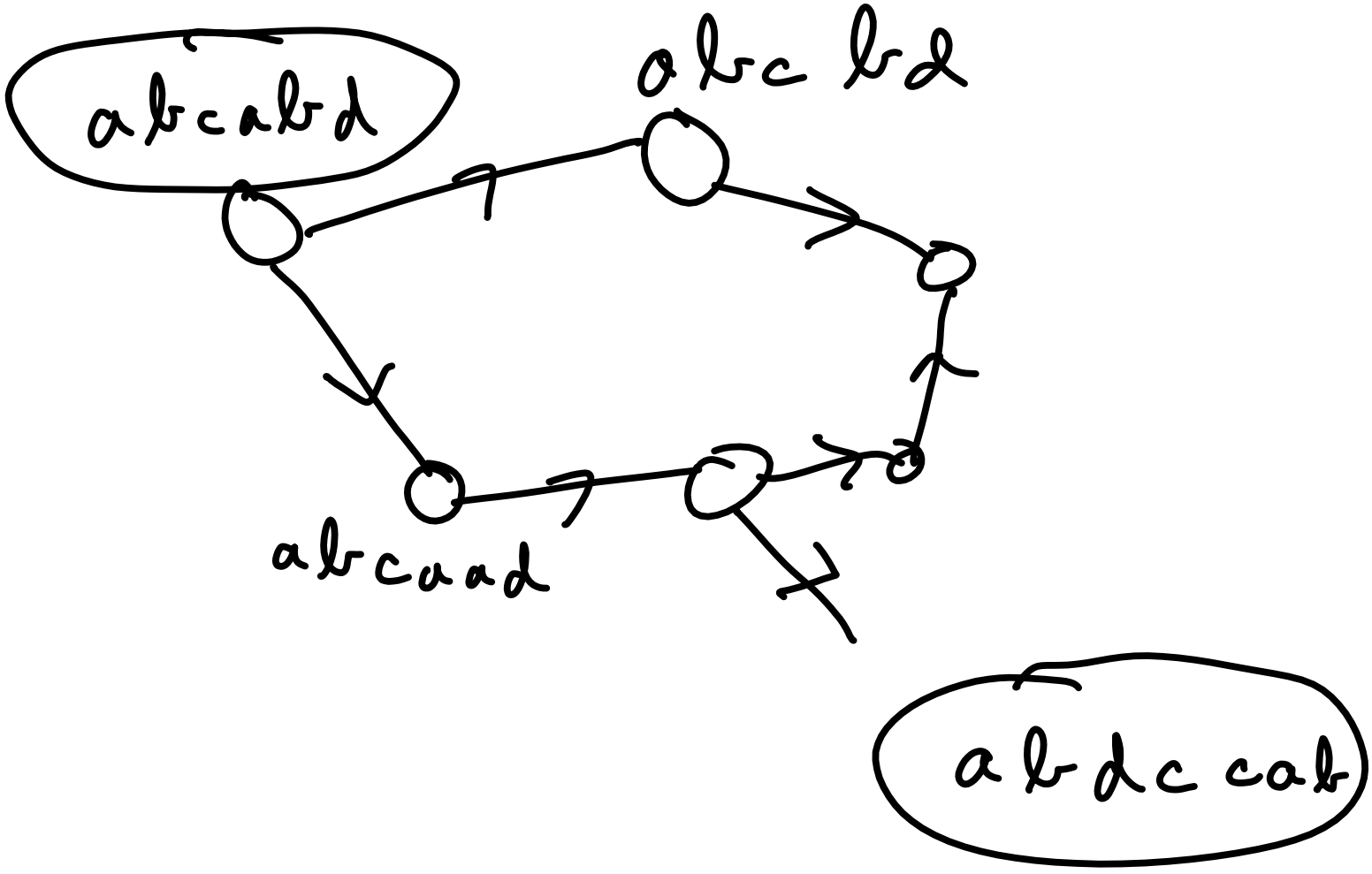
$m = n-1$

Plus court chemin dans les graphes

- **Entrée :** un graphe $G = (V, E)$, deux sommets $s, t \in V$



- **Sortie :** un plus court chemin de s a t dans G


abcabd
abc bd
abcaad
abdc cab

Graphes non orientes et non ponderes

Plus courts chemins a origine unique

Notion de distance dans les graphes

- **Cadre :** graphes non-orientes et non-ponderes

Plus courts chemins a origine unique

Notion de distance dans les graphes

- **Cadre :** graphes non-orientes et non-ponderes
- **Longueur d'un chemin** C **entre** u **et** v **:** nombre d'aretes sur le chemin C

Plus courts chemins a origine unique

Notion de distance dans les graphes

- **Cadre :** graphes non-orientes et non-ponderes
- **Longueur d'un chemin** C **entre** u **et** v **:** nombre d'aretes sur le chemin C
- **Distance** $dist(u, v)$ **entre** u **et** v **:** longueur du plus court chemin entre u et v

Plus courts chemins a origine unique

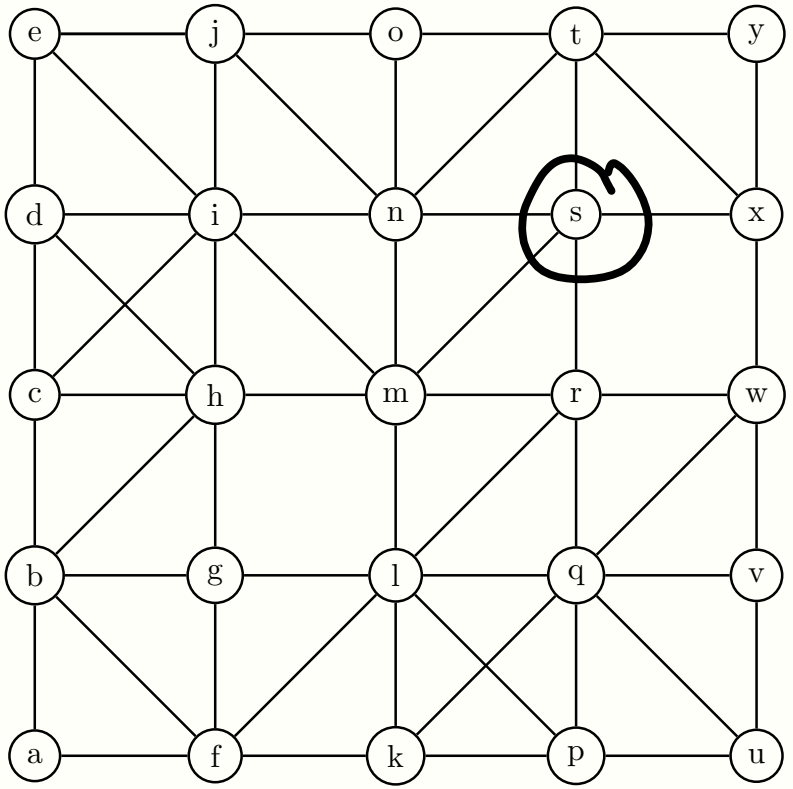
Notion de distance dans les graphes

- **Cadre :** graphes non-orientes et non-ponderes
- **Longueur d'un chemin** C **entre** u **et** v **:** nombre d'aretes sur le chemin C
- **Distance** $dist(u, v)$ **entre** u **et** v **:** longueur du plus court chemin entre u et v

Probleme des plus courts chemins a origine unique

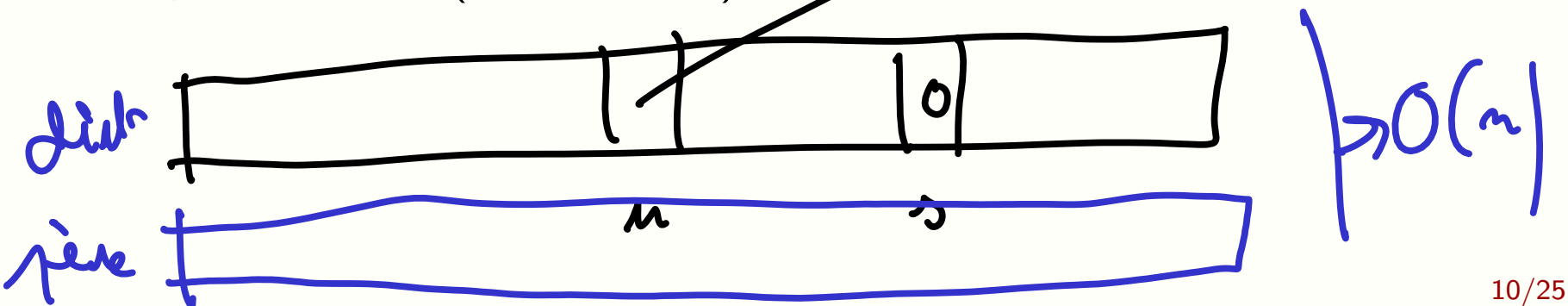
- **Entrée :** un graphe $G = (V, E)$ non oriente et non pondere, un sommet $s \in V$
- **Sortie :** la distance $dist(s, u)$ pour tout les sommets $u \in V$

Exemple



- Representation (en memoire) de la sortie ?

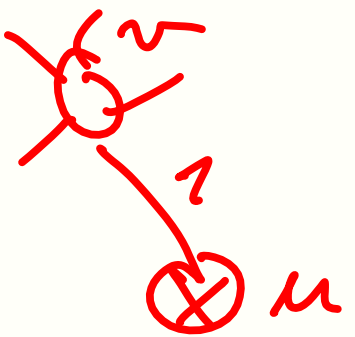
dist(s,u)



Algorithme : parcours en largeur (BFS)

Algorithme 1 : Plus courts chemins a origine unique **BFS(G,s)**

```
1  Un tableau dist de taille n ; F ← ∅ une file;
2  pour u de 0 a n − 1 faire
3  |   dist[u] ← +∞ ; coul[u] ← blanc;
4  fin
5  dist[s] ← 0 ; enfiler(s, F) ; coul[s] ← gris;
6  tant que F ≠ ∅ faire
7  |   u ← premier(F);
8  |   pour v ∈ N(u) faire
9  |   |   si coul[v] = blanc alors
10 |   |   |   dist[v] ← dist[u] + 1 ; enfiler(v, F) ; coul[v] = gris;
11 |   |   fin
12 |   fin
13 |   defiler(F);
14 fin
```



coul[v] ← noir.

Algorithme : parcours en largeur (BFS)

Algorithme 1 : Plus courts chemins a origine unique **BFS(G,s)**

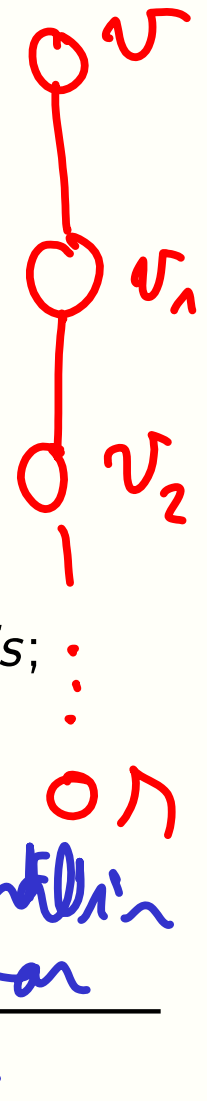
```
1  Un tableau dist de taille n ; F ← ∅ une file;
2  pour u de 0 a n − 1 faire
3  |   dist[u] ← +∞ ; coul[u] ← blanc;
4  fin
5  dist[s] ← 0 ; enfiler(s, F) ; coul[s] = gris;
6  tant que F ≠ ∅ faire
7  |   u ← premier(F);
8  |   pour v ∈ N(u) faire
9  |   |   si coul[v] = blanc alors
10 |   |   |   dist[v] ← dist[u] + 1 ; enfiler(v, F) ; coul[v] = gris;
11 |   |   fin
12 |   fin
13 |   defiler(F);
14 fin
```

Algorithme : parcours en largeur (BFS)

Algorithme 1 : Plus courts chemins a origine unique **BFS(G,s)**

```
1  Un tableau dist de taille n ; F ← ∅ une file;
2  pour u de 0 a n − 1 faire
3  |   dist[u] ← +∞ ; coul[u] ← blanc;
4  fin
5  dist[s] ← 0 ; enfiler(s, F) ; coul[s] = gris;
6  tant que F ≠ ∅ faire
7  |   u ← premier(F);
8  |   pour v ∈ N(u) faire
9  |   |   si coul[v] = blanc alors
10 |   |   |   dist[v] ← dist[u] + 1 ; enfiler(v, F) ; coul[v] = gris;
11 |   |   fin
12 |   fin
13 |   defiler(F);
14 fin
```

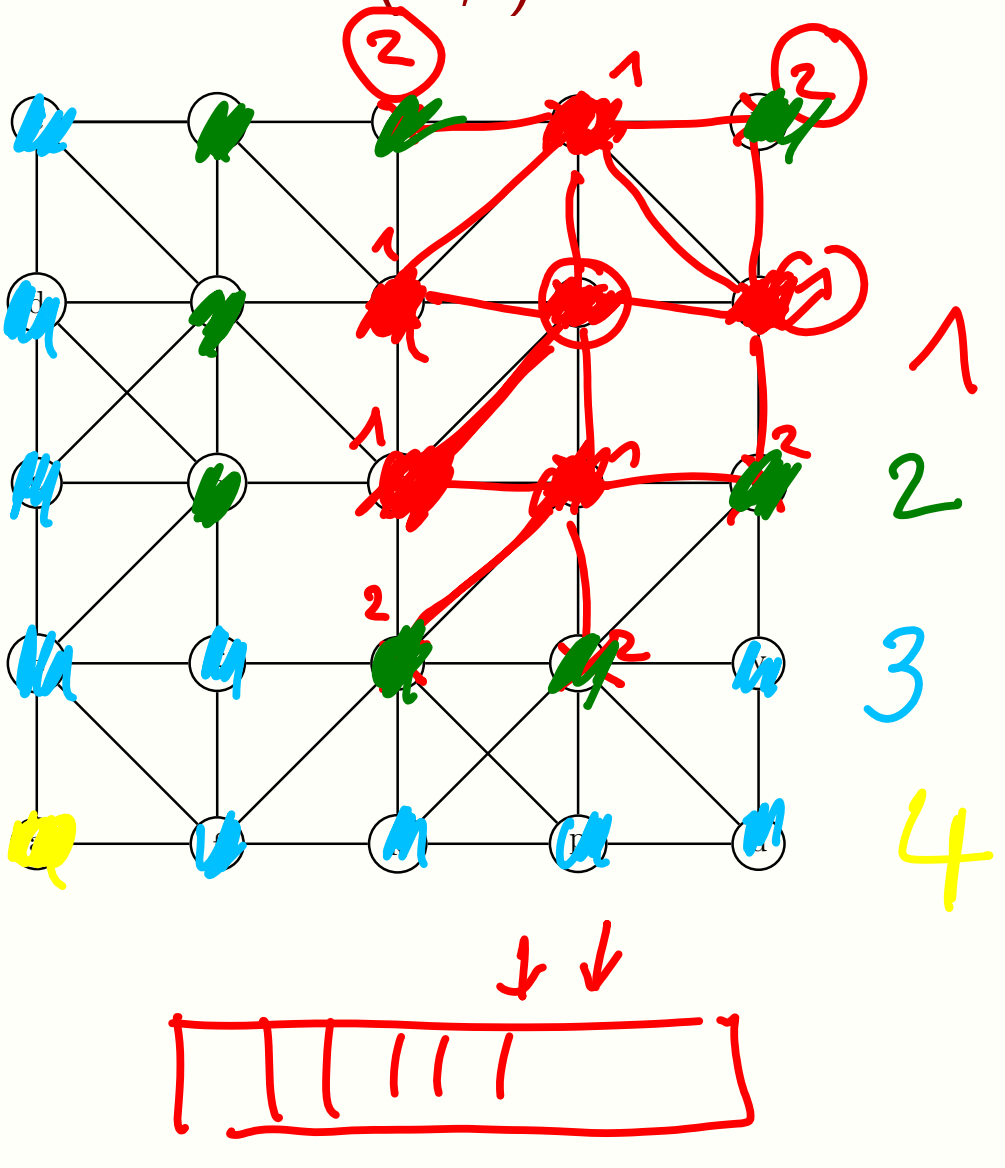
pere(n)



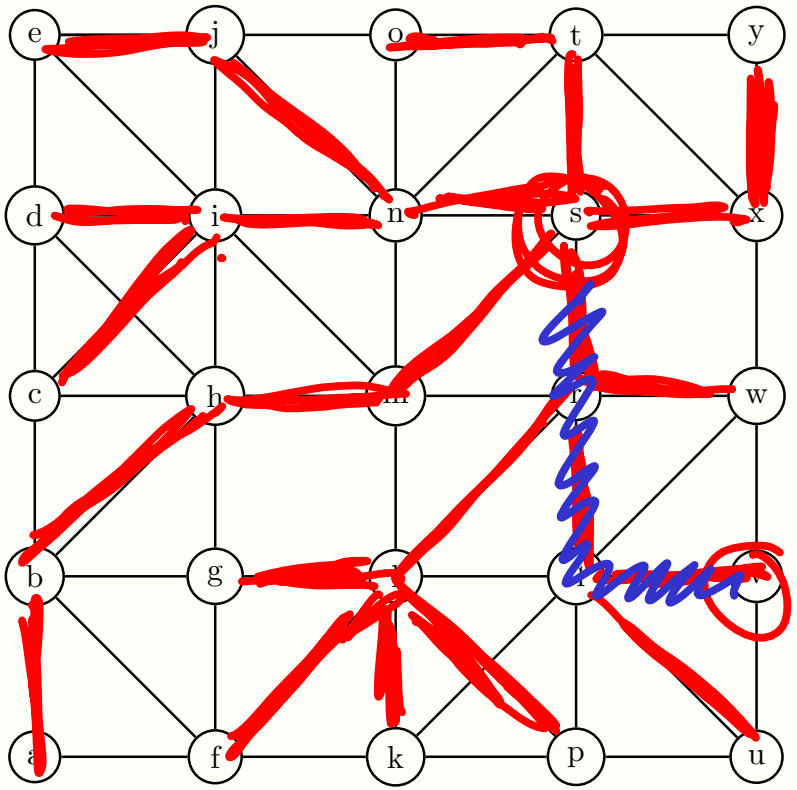
pere(v) ← pere(n)

→ la représentation d'un arbre par un tableau.

Exemple d'execution de BFS(G,s)



Exemple d'execution de BFS(G,s)



Preuve de correction de l'algorithme

Theoreme : A la fin de l'algorithme, la distance $dist[u]$ affectee a chaque sommet u est exactement la distance $dist(s, u)$ entre u et s.

Preuve de correction de l'algorithme

Theoreme : A la fin de l'algorithme, la distance $dist[u]$ affectee a chaque sommet u est exactement la distance $dist(s, u)$ entre u et s.

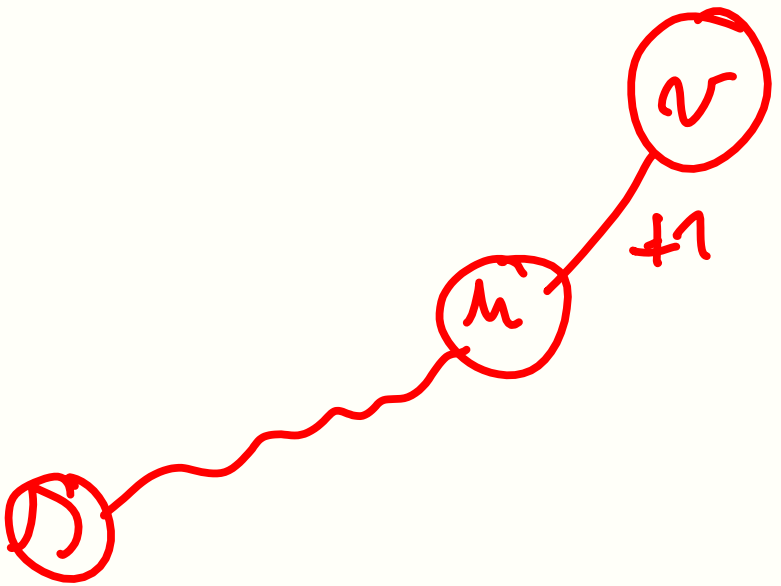
Propriete 1 : La distance $dist[v]$ est initialement $+\infty$ et si v est accessible depuis s alors $dist[v]$ devient finie au cours de l'algorithme.

Preuve de correction de l'algorithme

Theoreme : A la fin de l'algorithme, la distance $dist[u]$ affectee a chaque sommet u est exactement la distance $dist(s, u)$ entre u et s.

Propriete 1 : La distance $dist[v]$ est initialement $+\infty$ et si v est accessible depuis s alors $dist[v]$ devient finie au cours de l'algorithme.

Propriete 2 : Lorsque $dist[v] \leftarrow dist[u] + 1$ a la ligne 10, v est accessible depuis s par un chemin de longueur $dist[v]$.



Preuve de correction de l'algorithme

Theoreme : A la fin de l'algorithme, la distance $dist[u]$ affectee a chaque sommet u est exactement la distance $dist(s, u)$ entre u et s.

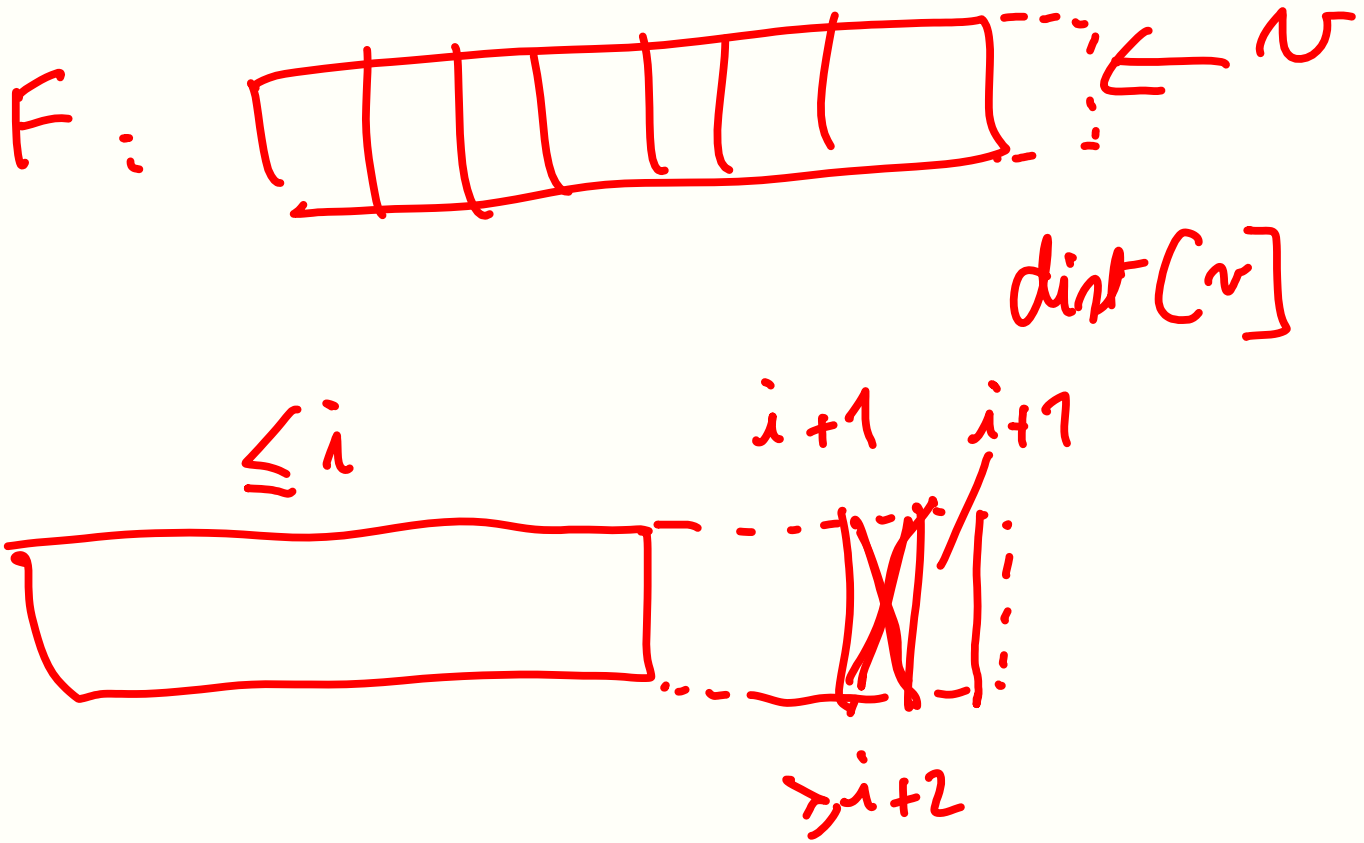
Propriete 1 : La distance $dist[v]$ est initialement $+\infty$ et si v est accessible depuis s alors $dist[v]$ devient finie au cours de l'algorithme.

Propriete 2 : Lorsque $dist[v] \leftarrow dist[u] + 1$ a la ligne 10, v est accessible depuis s par un chemin de longueur $dist[v]$.

Corollaire : A la fin de l'algorithme, pour tout u, on a $dist[u]$ est finie ssi u est accessible depuis s, et on a de plus $dist(s, u) \leq dist[u]$.

Preuve de correction de l'algorithme

Propriete 3 : Au cours de l'algorithme, la valeur $dist[v]$ du sommet v qui est enfiler dans F (ligne 10) est croissante. De plus, a chaque instant de l'algorithme la difference entre les valeur $dist[.]$ associee au premier $prem$ et du dernier $dern$ sommet de F est au plus 1 : $|dist[prem] - dist[dern]| \leq 1$.



Preuve de correction de l'algorithme

Theoreme : A la fin de l'algorithme, la distance $dist[u]$ affectee a chaque sommet u est exactement la distance $dist(s, u)$ entre u et s.

Démonstration. par l'absurde.
Considerons la premiere fois au cours de l'algorithme que la distance affectee a un sommet v (ligne 10) n'est pas correcte.

dis

□

Preuve de correction de l'algorithme

Theoreme : A la fin de l'algorithme, la distance $dist[u]$ affectee a chaque sommet u est exactement la distance $dist(s, u)$ entre u et s.

Démonstration.

Considerons la premiere fois au cours de l'algorithme que la distance affectee a un sommet v (ligne 10) n'est pas correcte.

D'apres le corollaire, on a $dist[v] > dist(s, v)$

□

Preuve de correction de l'algorithme

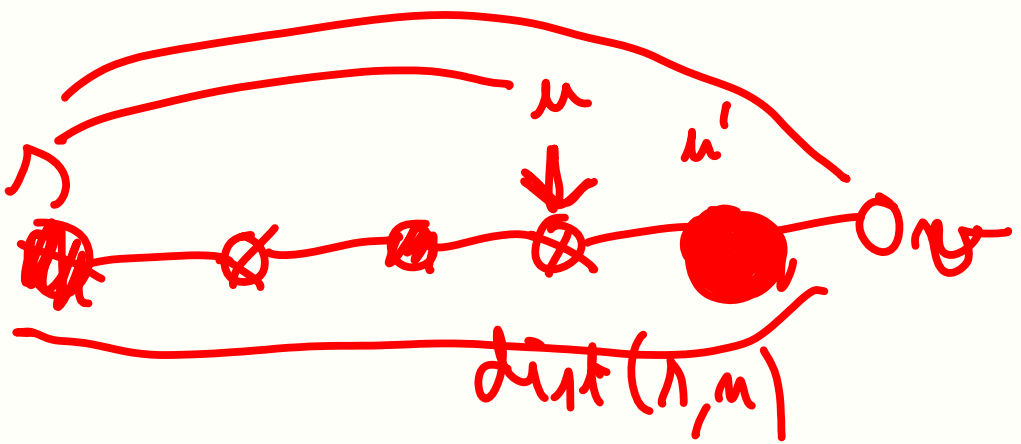
Theoreme : A la fin de l'algorithme, la distance $dist[u]$ affectee a chaque sommet u est exactement la distance $dist(s, u)$ entre u et s.

Démonstration.

Considerons la premiere fois au cours de l'algorithme que la distance affectee a un sommet v (ligne 10) n'est pas correcte.

D'apres le corollaire, on a $dist[v] > dist(s, v)$

Considerons un plus court chemin de s a v, et u le dernier sommet non blanc sur ce chemin. On a $dist[u] = dist(s, u)$. Et comme $dist(s, v) < dist[v]$, on a $dist[u] = dist(s, u) < dist(s, v) < dist[v]$.



□

Preuve de correction de l'algorithme

Theoreme : A la fin de l'algorithme, la distance $dist[u]$ affectee a chaque sommet u est exactement la distance $dist(s, u)$ entre u et s.

Démonstration.

Considerons la premiere fois au cours de l'algorithme que la distance affectee a un sommet v (ligne 10) n'est pas correcte.

D'apres le corollaire, on a $dist[v] > dist(s, v)$

Considerons un plus court chemin de s a v, et u le dernier sommet non blanc sur ce chemin. On a $dist[u] = dist(s, u)$. Et comme $dist(s, v) < dist[v]$, on a $dist[u] = dist(s, u) < dist(s, v) < dist[v]$.

Or d'apres la propriete 3, les sommets entrent dans F (et donc en sortent) a $dist[.]$ croissant. u est donc noir et ne peut avoir de voisin blanc : contradiction. □

Analyse de complexite de l'algorithme

- toutes les instructions s'executent en temps $O(1)$
- boucle ligne 2 : $O(n)$
- Boucle ligne 6 : au plus n fois
  - car u est premier de F (ligne 7) au plus une fois
- boucle ligne 8 : $d^\circ(u)$ fois
- au total, la boucle ligne 6 prend un temps $\leq O(n) + O(\sum_{u \in V} d^\circ(u)) = O(n+m)$
- complexite totale de l'algo : $O(n+m)$

linéaire.

en taille que les listes d'adjacence.

Graphes orientes (toujours non ponderes)

- **Cadre :** graphes orientes et non-ponderes
- **Longueur d'un chemin oriente C de u a v :** nombre d'arcs sur le chemin C
- **Distance $dist(u, v)$ entre u et v :** longueur du plus court chemin oriente de u a v
  - Note : $dist(u, v) \neq dist(v, u)$

Probleme des plus courts chemins orientes a origine unique

- **Entrée :** un graphe oriente $G = (V, A)$ non pondere, un sommet $s \in V$
- **Sortie :** la distance $dist(s, u)$ pour tout les sommets $u \in V$

Graphes orientes (toujours non ponderes)

Meme solution mais $N(u)$ devient $N^+(u)$

Algorithme 2 : BFS(G,s) dans un graphe oriente

```
1  Un tableau dist de taille n ; F ← ∅ une file;
2  pour u de 0 a n − 1 faire
3  |   dist[u] ← +∞ ; coul[u] ← blanc;
4  fin
5  dist[s] ← 0 ; enfiler(s, F) ; coul[s] = gris;
6  tant que F ≠ ∅ faire
7  |   u ← premier(F);
8  |   pour v ∈ N+(u) faire
9  |   |   si coul[v] = blanc alors
10 |   |   |   dist[v] ← dist[u] + 1 ; enfiler(v, F) ; coul[v] = gris;
11 |   |   fin
12 |   fin
13 |   defiler(F);
14 fin
```

Graphes orientes (toujours non ponderes)

Meme solution mais $N(u)$ devient $N^+(u)$

Algorithme 2 : BFS(G,s) dans un graphe oriente

```
1  Un tableau dist de taille n ; F ← ∅ une file;
2  pour u de 0 a n − 1 faire
3  |   dist[u] ← +∞ ; coul[u] ← blanc;
4  fin
5  dist[s] ← 0 ; enfiler(s, F) ; coul[s] = gris;
6  tant que F ≠ ∅ faire
7  |   u ← premier(F);
8  |   pour v ∈ N+(u) faire
9  |   |   si coul[v] = blanc alors
10 |   |   |   dist[v] ← dist[u] + 1 ; enfiler(v, F) ; coul[v] = gris;
11 |   |   fin
12 |   fin
13 |   defiler(F);
14 fin
```

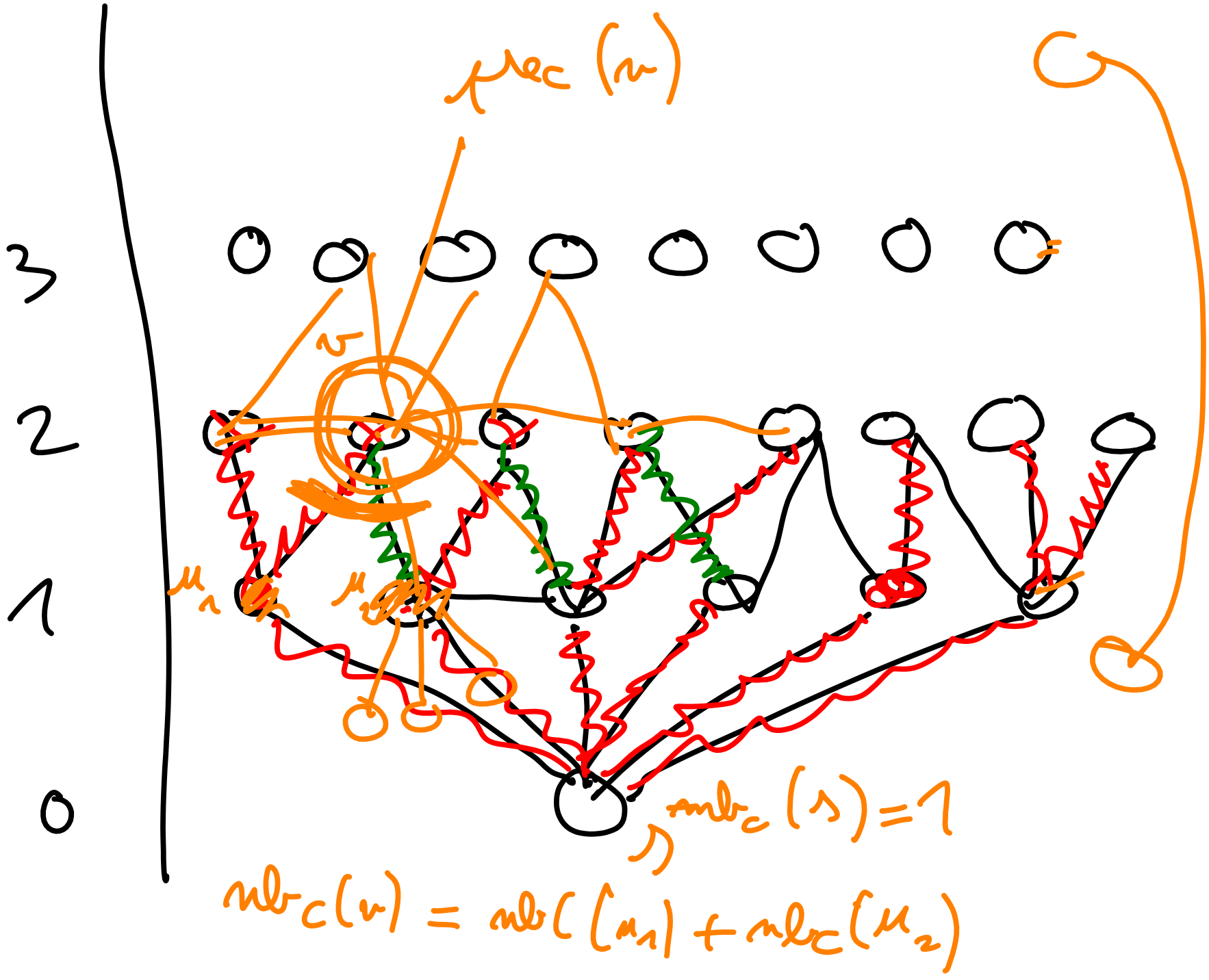
prec, (v)

prec(v) ← prec(u) ∪ {u}

pere(v) ← u

si coul[v] = gris et dist[v] = dist[u]+1

alors Prec(v) ← Prec(v) ∪ {u}


3
2
1
0
$\mu_{rec}(v)$
v
μ_1
μ_2
$nb_C(1)=1$
$nb_C(v) = nb(\mu_1) + nb_C(\mu_2)$

Graphes ponderes positivement

Plus courts chemins a origine unique

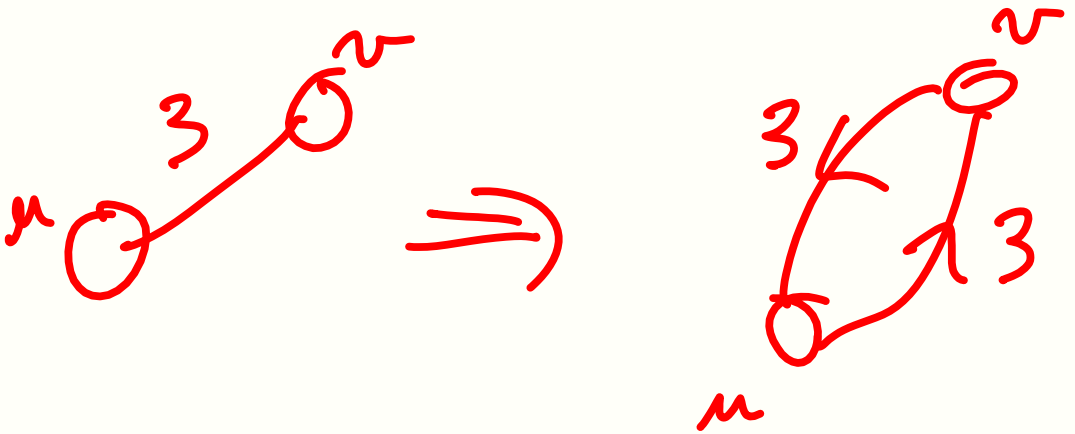
Notion de distance dans les graphes ponderes positivement

- **Cadre :** graphes orientes et ponderes positivement
  - une application $w : A \to \mathbb{R}_+$

Plus courts chemins a origine unique

Notion de distance dans les graphes ponderes positivement

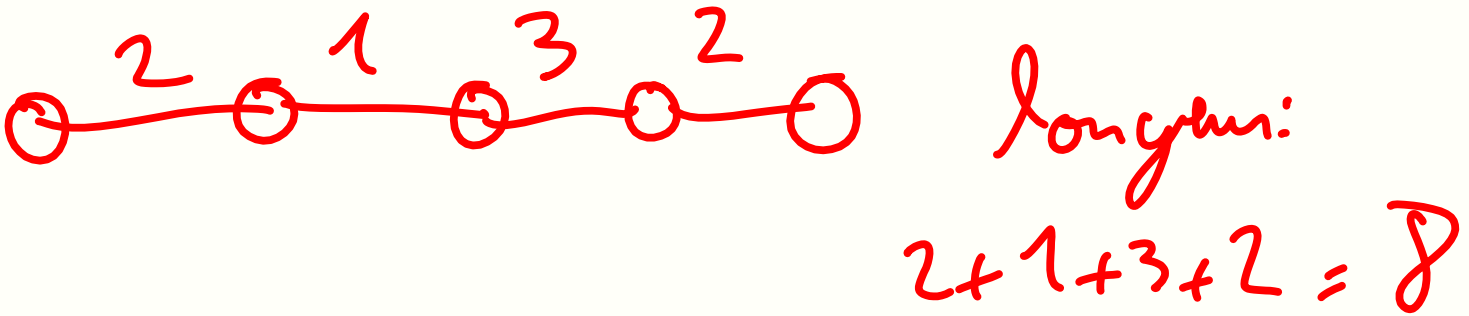
- **Cadre :** graphes orientes et ponderes positivement
  - une application $w : A \to \mathbb{R}_+$
  - graphes non orientes obtenus comme cas particulier : graphes orientes symetriques avec fonction de poids symetrique $\forall (u, v) \in A, w(u, v) = w(v, u)$.



Plus courts chemins a origine unique

Notion de distance dans les graphes ponderes positivement

- **Cadre :** graphes orientes et ponderes positivement
  - une application $w : A \to \mathbb{R}_+$
  - graphes non orientes obtenus comme cas particulier : graphes orientes symetriques avec fonction de poids symetrique $\forall (u, v) \in A, w(u, v) = w(v, u)$.
- **Longueur d'un chemin oriente pondere C de u a v :** somme des poids des arcs du chemin oriente C



Plus courts chemins a origine unique

Notion de distance dans les graphes ponderes positivement

- **Cadre :** graphes orientes et ponderes positivement
  - une application $w : A \to \mathbb{R}_+$
  - graphes non orientes obtenus comme cas particulier : graphes orientes symetriques avec fonction de poids symetrique $\forall (u, v) \in A, w(u, v) = w(v, u)$.
- **Longueur d'un chemin oriente pondere C de u a v :** somme des poids des arcs du chemin oriente C
- **Distance $dist(u, v)$ de u a v :** longueur du plus court chemin oriente pondere de u a v

Plus courts chemins a origine unique

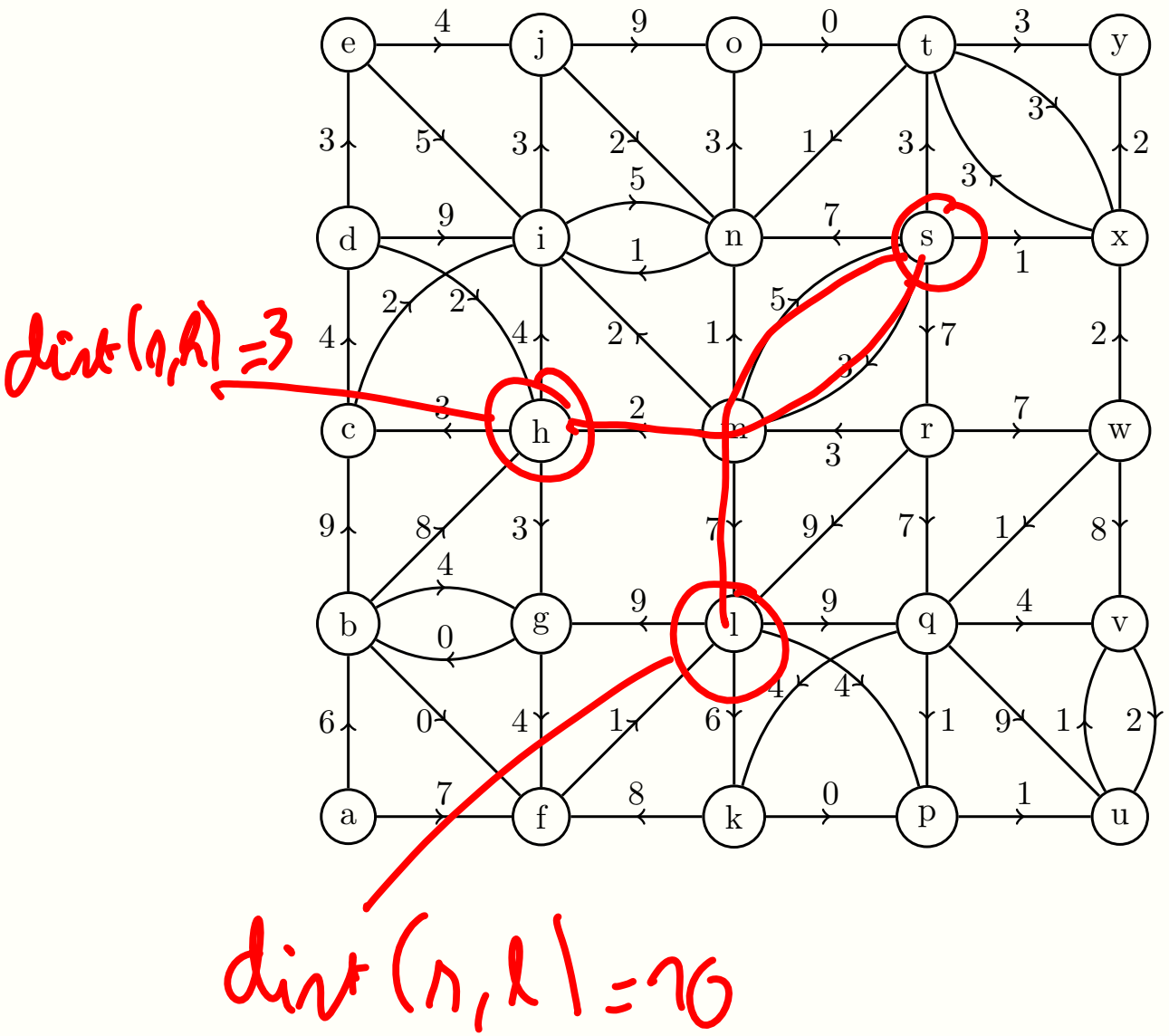
Notion de distance dans les graphes ponderes positivement

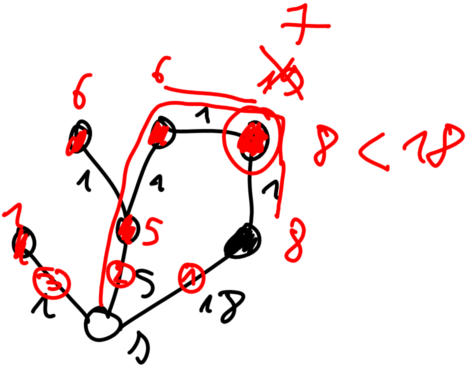
- **Cadre :** graphes orientes et ponderes positivement
  - une application $w : A \to \mathbb{R}_+$
  - graphes non orientes obtenus comme cas particulier : graphes orientes symetriques avec fonction de poids symetrique $\forall (u, v) \in A, w(u, v) = w(v, u)$.
- **Longueur d'un chemin oriente pondere C de u a v :** somme des poids des arcs du chemin oriente C
- **Distance $dist(u, v)$ de u a v :** longueur du plus court chemin oriente pondere de u a v

Probleme des plus courts chemins a origine unique

- **Entrée :** un graphe $G = (V, E)$ oriente et pondere positivement, un sommet $s \in V$
- **Sortie :** la distance $dist(s, u)$ pour tout les sommets $u \in V$

Exemple

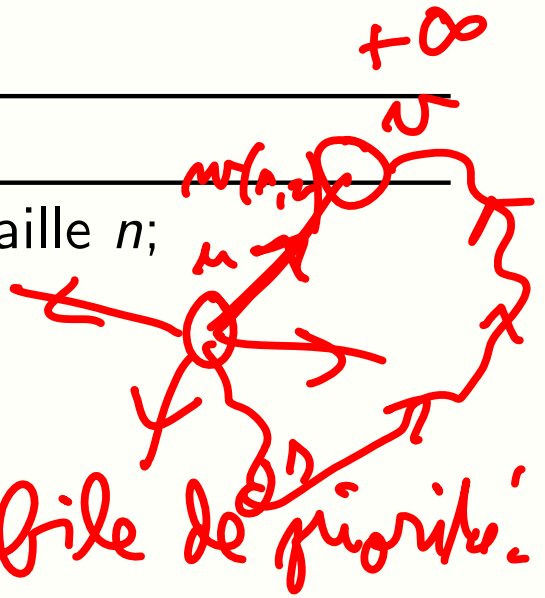



7
6
6
8 < 18
1
1
1
1
5
8
5
1
18
1

Algorithme : Dijkstra

Algorithme 3 : Dijkstra(G,s)

```
1  Un tableau dist de taille n ; Un tableau coul de taille n;
2  pour u de 0 a n − 1 faire
3  |   dist[u] ← +∞ ; coul[u] ← blanc;
4  fin
5  dist[s] ← 0 ; Q ← {s} ; coul[s] = gris;
6  tant que Q ≠ ∅ faire
7  |   u ← min_dist(Q) ; Q ← Q \ {u} ; coul[u] ← noir;
8  |   pour v ⊂ N+(u) faire
9  |   |   si dist[u] + w(u, v) < dist[v] alors
10 |   |   |   dist[v] ← dist[u] + w(u, v);
11 |   |   |   si coul[v]=blanc alors
12 |   |   |   |   Q ← Q ∪ {v} ; coul[v] ← gris;
13 |   |   fin
14 |   fin
15 fin
```

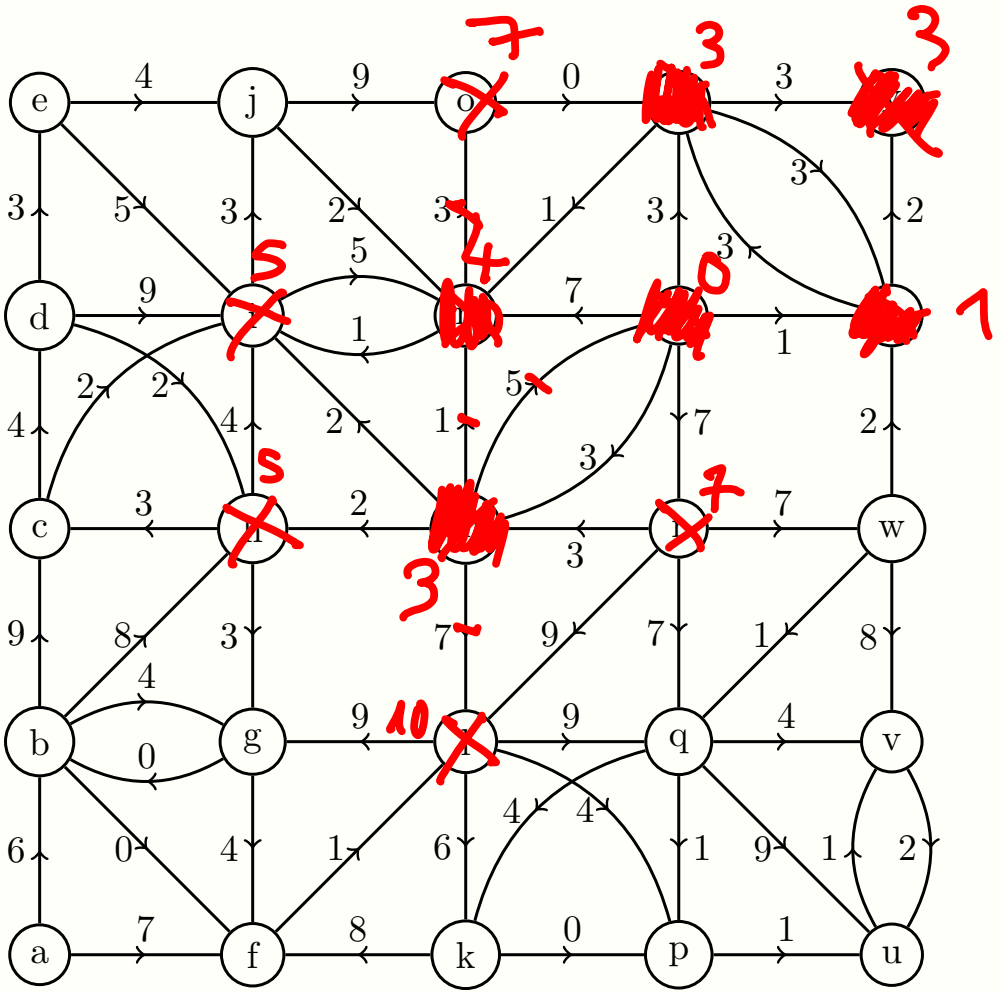


Algorithme : Dijkstra

Algorithme 3 : Dijkstra(G,s)

```
1  Un tableau dist de taille n ; Un tableau coul de taille n;
2  pour u de 0 a n − 1 faire
3  |   dist[u] ← +∞ ; coul[u] ← blanc;
4  fin
5  dist[s] ← 0 ; Q ← {s} ; coul[s] ← gris;
6  tant que Q ≠ ∅ faire
7  |   u ← min_dist(Q) ; Q ← Q \ {u} ; coul[u] ← noir;
8  |   pour v ∈ N+(u) faire
9  |   |   si dist[u] + w(u, v) < dist[v] alors
10 |   |   |   dist[v] ← dist[u] + w(u, v);
11 |   |   |   si coul[v]=blanc alors
12 |   |   |   |   Q ← Q ∪ {v} ; coul[v] ← gris;
13 |   |   fin
14 |   fin
15 fin
```

Exemple d'execution de Dijkstra(G,s)



Preuve de correction de l'algorithme

Theoreme : A la fin de l'algorithme, la distance $dist[u]$ affectee a chaque sommet u est exactement la distance $dist(s, u)$ de s a u.

Preuve de correction de l'algorithme

Theoreme : A la fin de l'algorithme, la distance $dist[u]$ affectee a chaque sommet u est exactement la distance $dist(s, u)$ de s a u.

Lemme : A la ligne 7 de l'algorithme, $dist[u]$ est exactement la distance de s a u dans G.

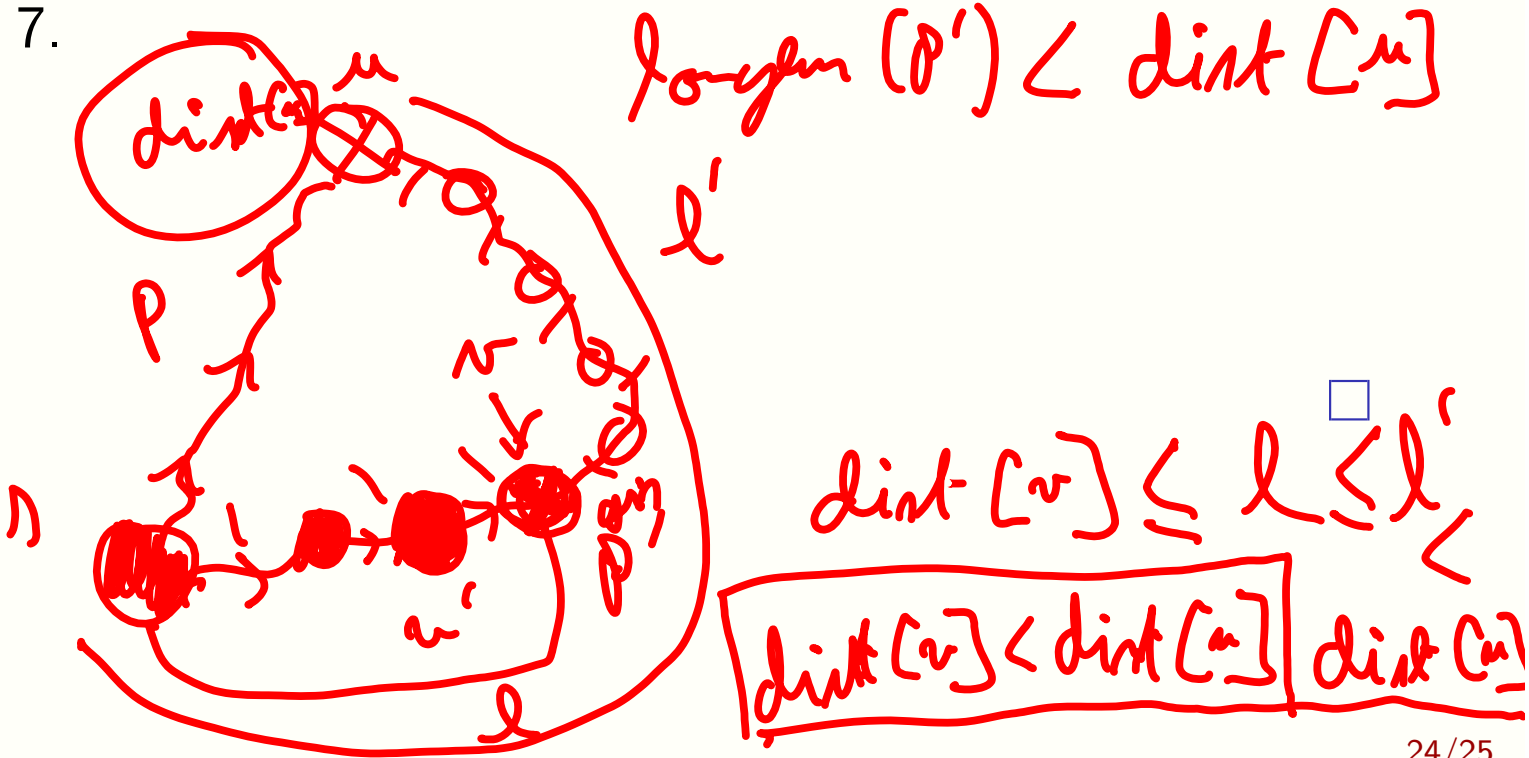
Preuve de correction de l'algorithme

Theoreme : A la fin de l'algorithme, la distance $dist[u]$ affectee a chaque sommet u est exactement la distance $dist(s, u)$ de s a u.

Lemme : A la ligne 7 de l'algorithme, $dist[u]$ est exactement la distance de s a u dans G.

Démonstration.

Par l'absurde, supposons que cela ne soit pas vrai pour le sommet u de la ligne 7.



Preuve de correction de l'algorithme

Theoreme : A la fin de l'algorithme, la distance $dist[u]$ affectee a chaque sommet u est exactement la distance $dist(s, u)$ de s a u.

Lemme : A la ligne 7 de l'algorithme, $dist[u]$ est exactement la distance de s a u dans G.

Démonstration.

Par l'absurde, supposons que cela ne soit pas vrai pour le sommet u de la ligne 7.

Considerons un plus court chemin de s a u et le premier sommet non noir v sur ce chemin.

□

Preuve de correction de l'algorithme

Theoreme : A la fin de l'algorithme, la distance $dist[u]$ affectee a chaque sommet u est exactement la distance $dist(s, u)$ de s a u.

Lemme : A la ligne 7 de l'algorithme, $dist[u]$ est exactement la distance de s a u dans G.

Démonstration.

Par l'absurde, supposons que cela ne soit pas vrai pour le sommet u de la ligne 7.

Considerons un plus court chemin de s a u et le premier sommet non noir v sur ce chemin.

Alors v est gris et $dist[v] < dist[u]$

□

Preuve de correction de l'algorithme

Theoreme : A la fin de l'algorithme, la distance $dist[u]$ affectee a chaque sommet u est exactement la distance $dist(s, u)$ de s a u.

Lemme : A la ligne 7 de l'algorithme, $dist[u]$ est exactement la distance de s a u dans G.

Démonstration.

Par l'absurde, supposons que cela ne soit pas vrai pour le sommet u de la ligne 7.

Considerons un plus court chemin de s a u et le premier sommet non noir v sur ce chemin.

Alors v est gris et $dist[v] < dist[u]$

$\Longrightarrow$ contradiction : l'algo aurait du choisir v plutot que u.

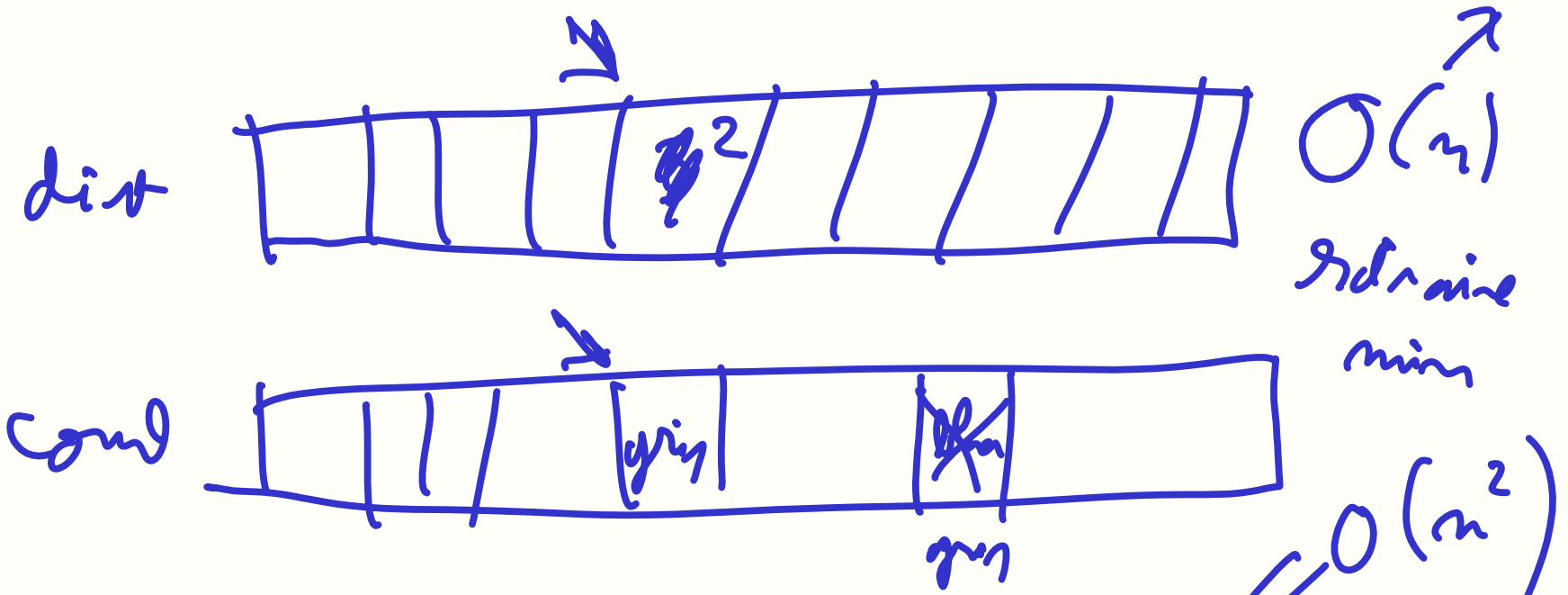
□

Analyse de complexite de l'algorithme

- Toutes les instructions sont en temps $O(1)$, sauf
  - $u \leftarrow min_{dist}(Q)$, a la ligne 7
  - $dist[v] \leftarrow dist[u] + w(u, v)$, a la ligne 10
  - qui dependent de la structure de donnees pour extraire le min

Analyse de complexite de l'algorithme

- Toutes les instructions sont en temps $O(1)$, sauf
  - $u \leftarrow min_{dist}(Q)$, a la ligne 7
  - $dist[v] \leftarrow dist[u] + w(u, v)$, a la ligne 10
  - qui dependent de la structure de donnees pour extraire le min
- au total : $O(n + m)$ + temps utilisation structure de donnees



Analyse de complexite de l'algorithme

- Toutes les instructions sont en temps $O(1)$, sauf
  - $u \leftarrow min_{dist}(Q)$, a la ligne 7
  - $dist[v] \leftarrow dist[u] + w(u, v)$, a la ligne 10
  - qui dependent de la structure de donnees pour extraire le min
- au total : $O(n + m)$ + temps utilisation structure de donnees

Differentes implementations de la file de priorite

- Tableau des poids : $O(n^2)$
  - Total algo : $O(n^2)$

Analyse de complexite de l'algorithme

- Toutes les instructions sont en temps $O(1)$, sauf
  - $u \leftarrow min_{dist}(Q)$, a la ligne 7
  - $dist[v] \leftarrow dist[u] + w(u, v)$, a la ligne 10
  - qui dependent de la structure de donnees pour extraire le min
- au total : $O(n + m)$ + temps utilisation structure de donnees

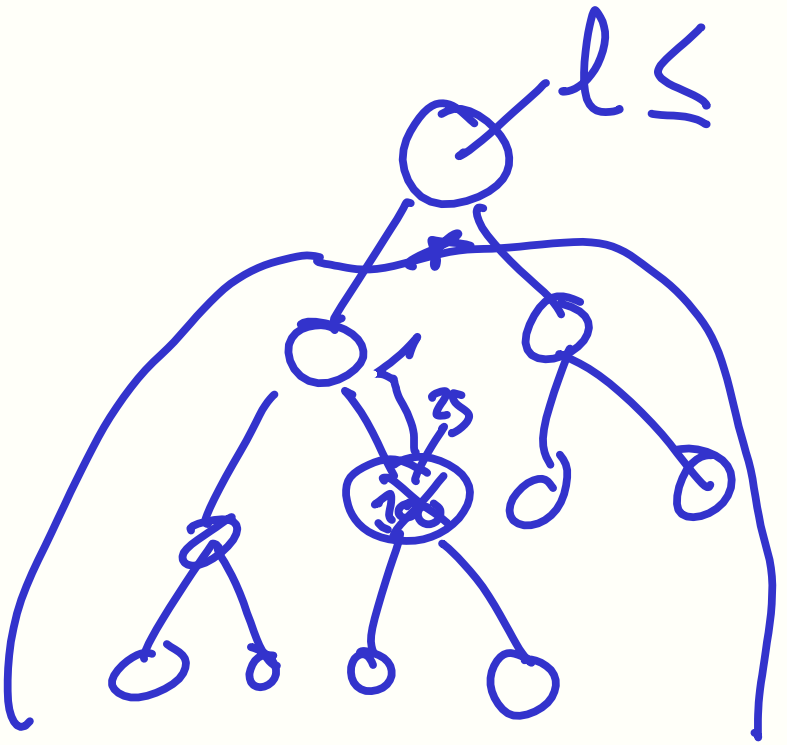
Differentes implementations de la file de priorite

- Tableau des poids : $O(n^2)$
  - Total algo : $O(n^2)$
- Tas binaire : $O((n + m) \log n)$
  - Total algo : $O((n + m) \log n)$

extraire min : $O(n)$ n

adapter : $O(\log n)$ m

ajouter : $O(\log n)$ n



Analyse de complexite de l'algorithme

- Toutes les instructions sont en temps $O(1)$, sauf
  - $u \leftarrow min_{dist}(Q)$, a la ligne 7
  - $dist[v] \leftarrow dist[u] + w(u, v)$, a la ligne 10
  - qui dependent de la structure de donnees pour extraire le min
- au total : $O(n + m)$ + temps utilisation structure de donnees

Differentes implementations de la file de priorite

- Tableau des poids : $O(n^2)$
  - Total algo : $O(n^2)$
- Tas binaire : $O((n + m) \log n)$
  - Total algo : $O((n + m) \log n)$
- Tas fibonacci : $O(m + n \log n)$
  - Total algo : $O(m + n \log n)$

m actualisations : $O(1)$
n extraction min : $O(\log n)$