Théorie des graphes: plus court chemin

Algorithmique et structures de données, 2023-2024

P. Kunzli (Cloud) et O. Malaspinas (A401), ISC, HEPIA

2024-05-28

En partie inspirés des supports de cours de P. Albuquerque

Rappel

Comment représente-t-on un graphe?

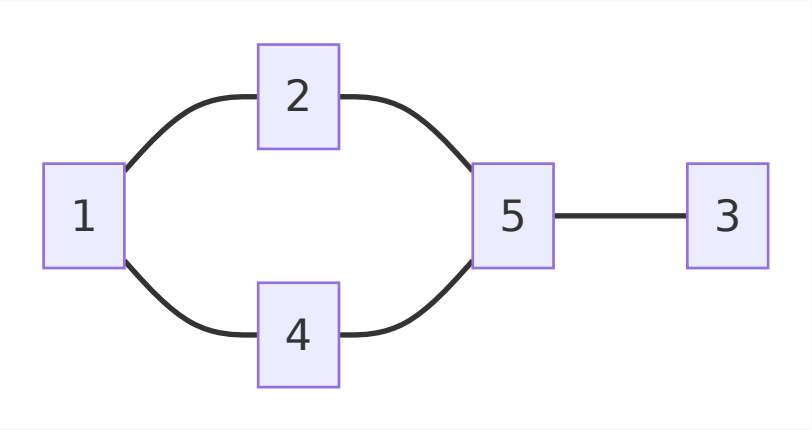
Rappel

Comment représente-t-on un graphe?

- Matrice ou liste d'ajdacence

Rappel: Matrice d'adjacence

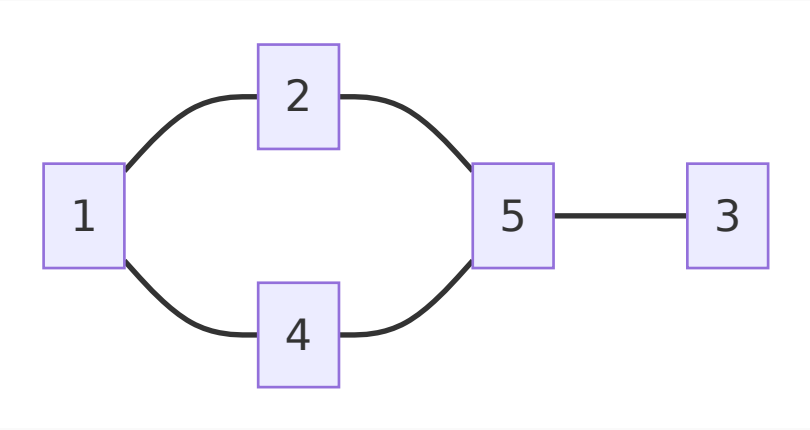
Exemple



Quelle matrice d'adjacence?

Rappel: Matrice d'adjacence

Exemple



Quelle matrice d'adjacence?

	1	2	3	4	5
1	0	1	0	1	0
2	1	0	0	0	1
3	0	0	0	0	1
4	1	0	0	0	1
5	0	1	1	1	0

Rappel: La liste d'adjacence

Exemple

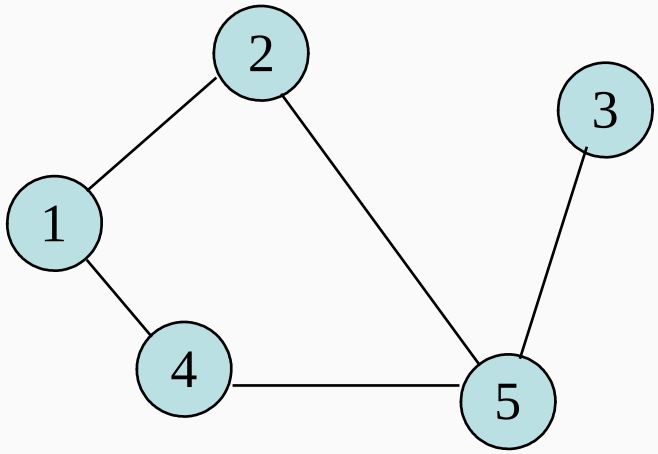


Figure 1: Un graphe non-orienté.

Quelle liste d'adjacence?

Rappel: La liste d'adjacence

Exemple

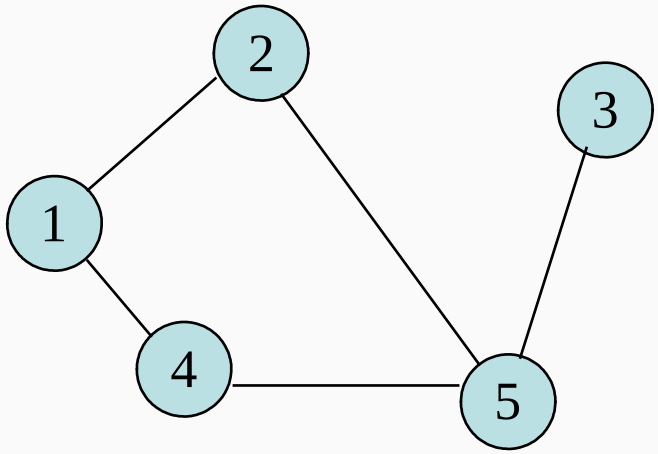


Figure 1: Un graphe non-orienté.

Quelle liste d'adjacence?

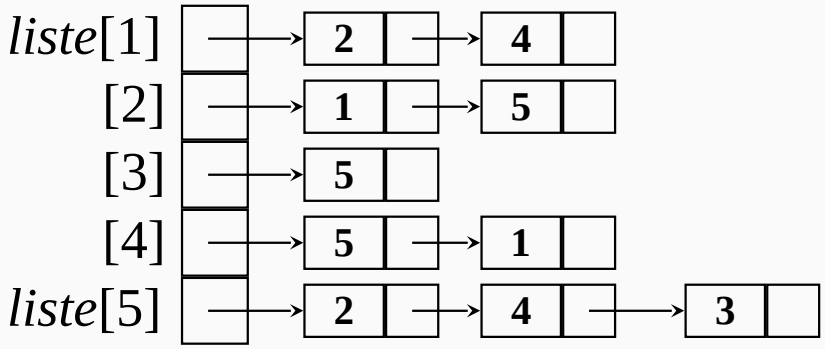


Figure 2: La liste d'adjacence.

Algorithmes de plus courts chemins

Contexte: les réseaux (informatique, transport, etc.)

- Graphe orienté;
- Source: sommet `s`;
- Destination: sommet `t`;
- Les arêtes ont des poids (coût d'utilisation, distance, etc.);
- Le coût d'un chemin est la somme des poids des arêtes d'un chemin.

Problème à résoudre

- Quel est le plus court chemin entre `s` et `t`.

Plus courts chemins à source unique

- Soit un graphe, $G = (V, E)$, une fonction de pondération $w : E \to \mathbb{R}$, et un sommet $s \in V$
  - Trouver pour tout sommet $v \in V$, le chemin de poids minimal reliant s à v.
- Algorithmes standards:
  - Dijkstra (arêtes de poids positif seulement);
  - Bellman-Ford (arêtes de poids positifs ou négatifs, mais sans cycles).
- Comment résoudre le problèmes si tous les poids sont les mêmes?

Plus courts chemins à source unique

- Soit un graphe, $G = (V, E)$, une fonction de pondération $w : E \to \mathbb{R}$, et un sommet $s \in V$
  - Trouver pour tout sommet $v \in V$, le chemin de poids minimal reliant s à v.
- Algorithmes standards:
  - Dijkstra (arêtes de poids positif seulement);
  - Bellman-Ford (arêtes de poids positifs ou négatifs, mais sans cycles).
- Comment résoudre le problèmes si tous les poids sont les mêmes?
- Un parcours en largeur!

Algorithme de Dijkstra

Comment chercher pour un plus court chemin?

Algorithme de Dijkstra

Comment chercher pour un plus court chemin?

```
si distance(u,v) > distance(u,w) + distance(w,v)
    on passe par w plutôt qu'aller directement
```

Algorithme de Dijkstra (1 à 5)

- D est le tableau des distances au sommet 1: $D[7]$ est la distance de 1 à 7.
- Le chemin est pas forcément direct.
- S est le tableau des sommets visités.

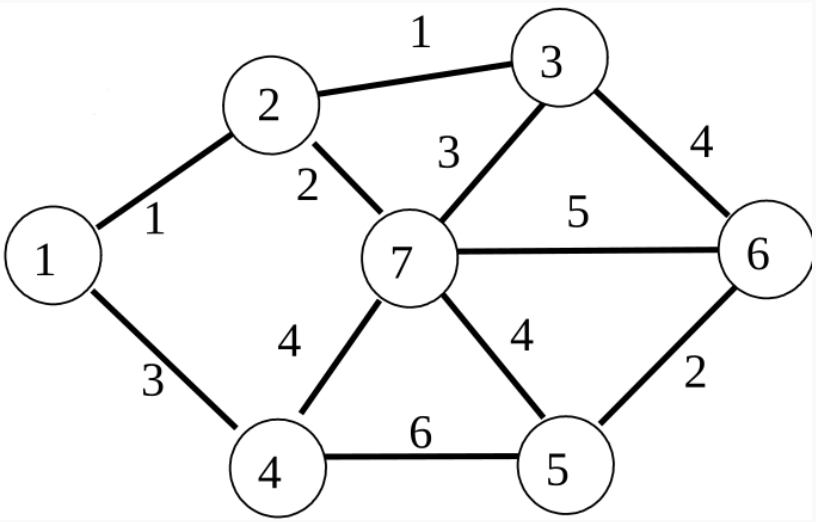


Figure 3: Initialisation.

Algorithme de Dijkstra (1 à 5)

- D est le tableau des distances au sommet 1: $D[7]$ est la distance de 1 à 7.
- Le chemin est pas forcément direct.
- S est le tableau des sommets visités.

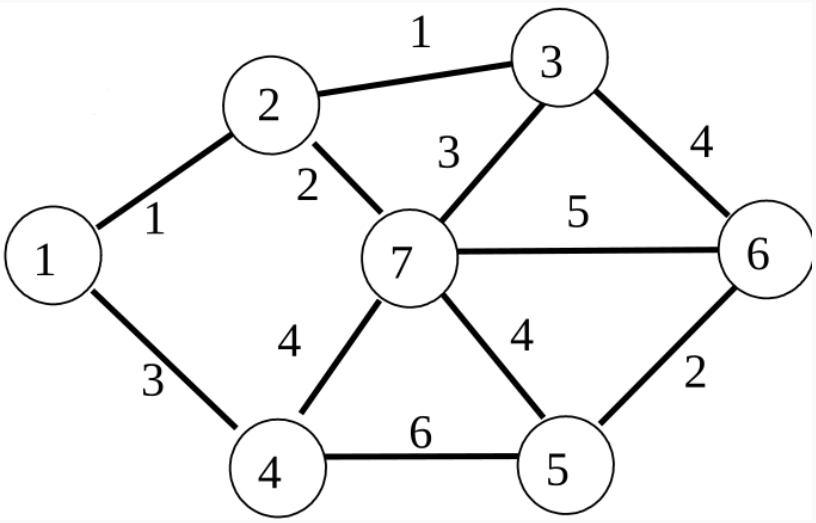


Figure 3: Initialisation.

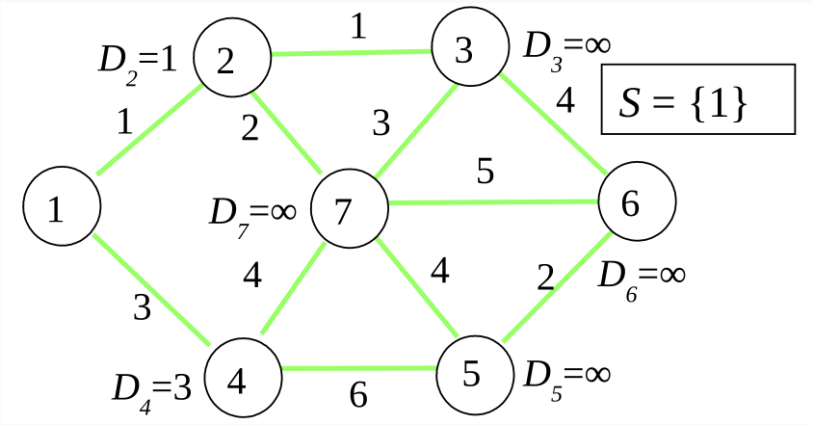


Figure 4: 1 visité, `D[2]=1`, `D[4]=3`.

Algorithme de Dijkstra (1 à 5)

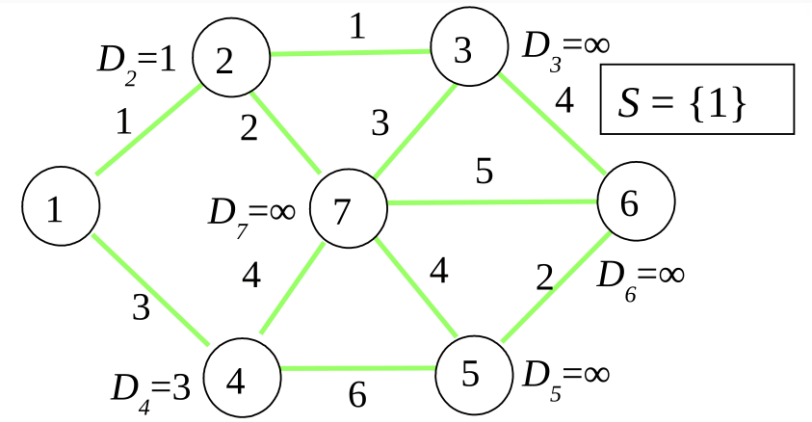


Figure 5: Plus court est 2.

Algorithme de Dijkstra (1 à 5)

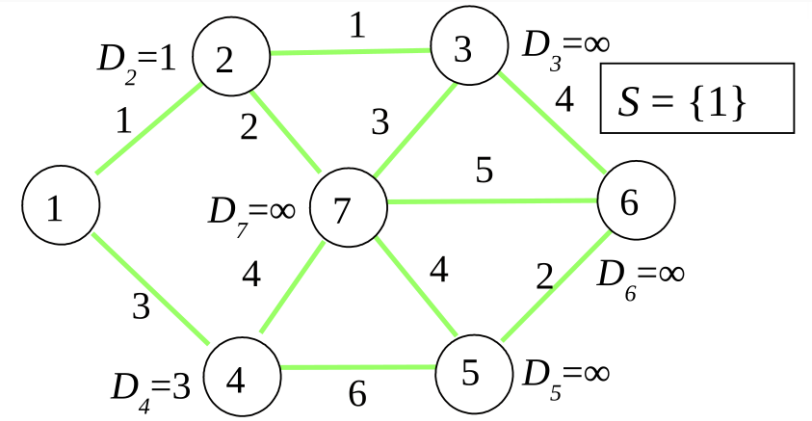


Figure 5: Plus court est 2.

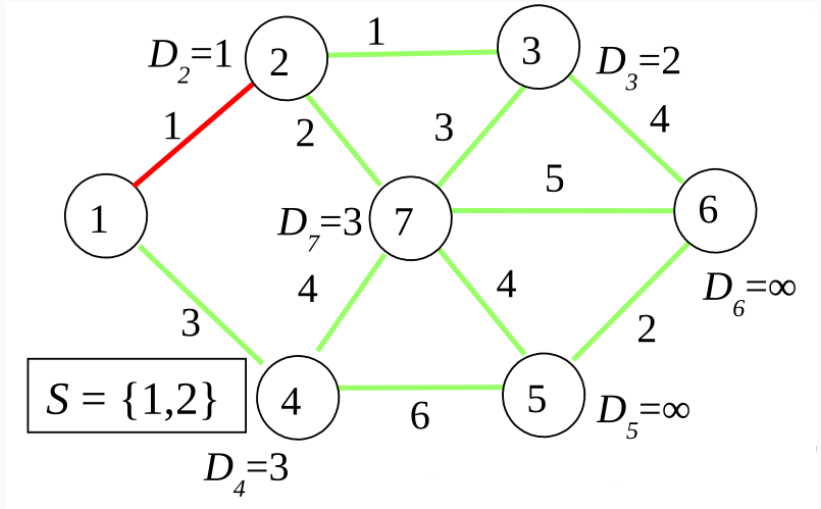


Figure 6: 2 visité, `D[3]=2`, `D[7]=3`.

Algorithme de Dijkstra (1 à 5)

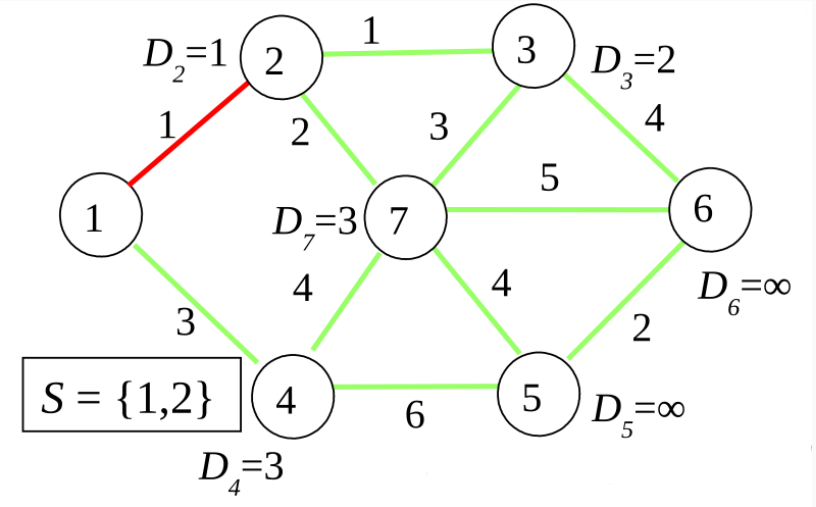


Figure 7: Plus court est 3.

Algorithme de Dijkstra (1 à 5)

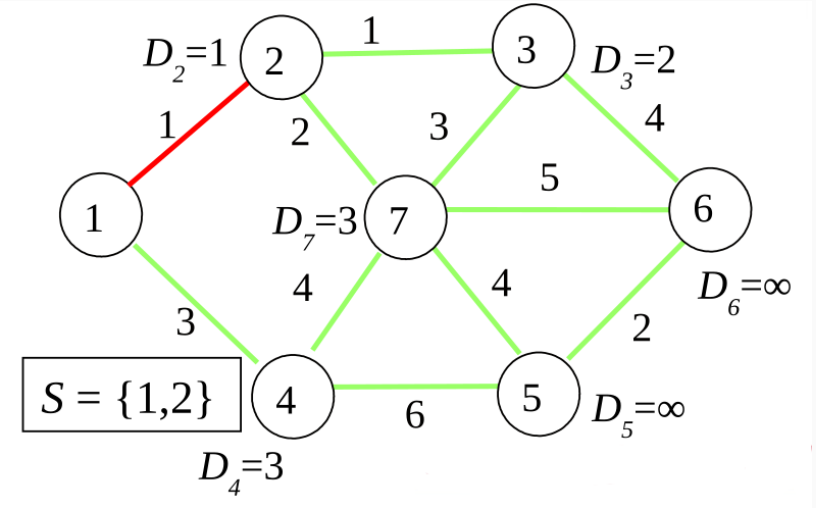


Figure 7: Plus court est 3.

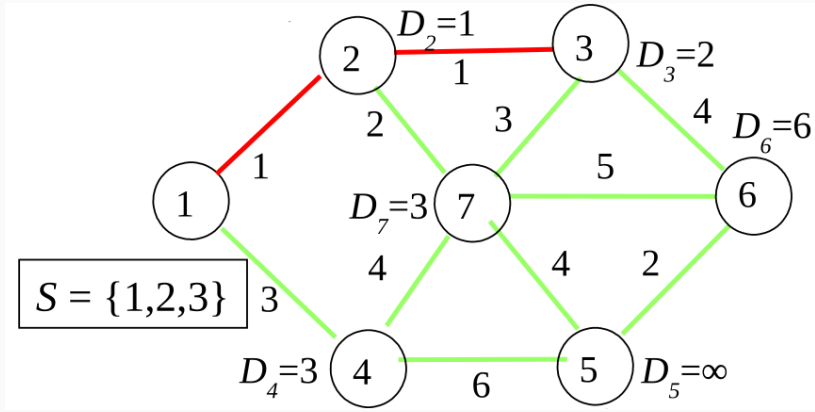


Figure 8: 3 visité, `D[7]`=3 inchangé, `D[6]`=6.

Algorithme de Dijkstra (1 à 5)

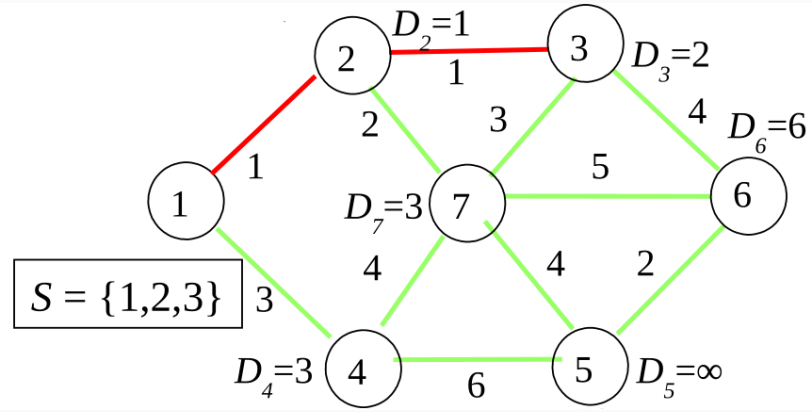


Figure 9: Plus court est 4 ou 7.

Algorithme de Dijkstra (1 à 5)

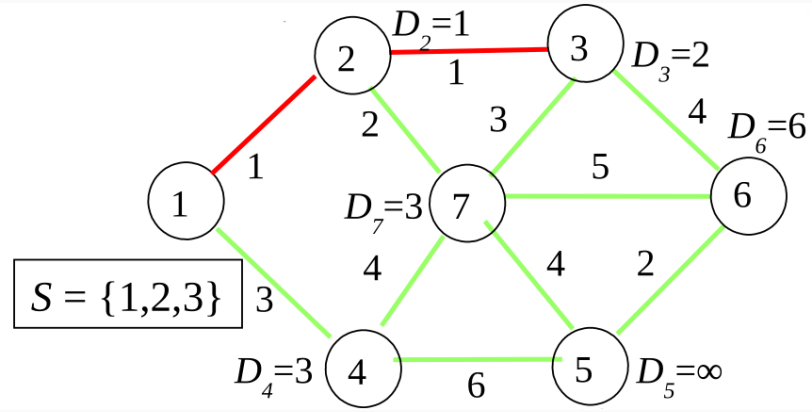


Figure 9: Plus court est 4 ou 7.

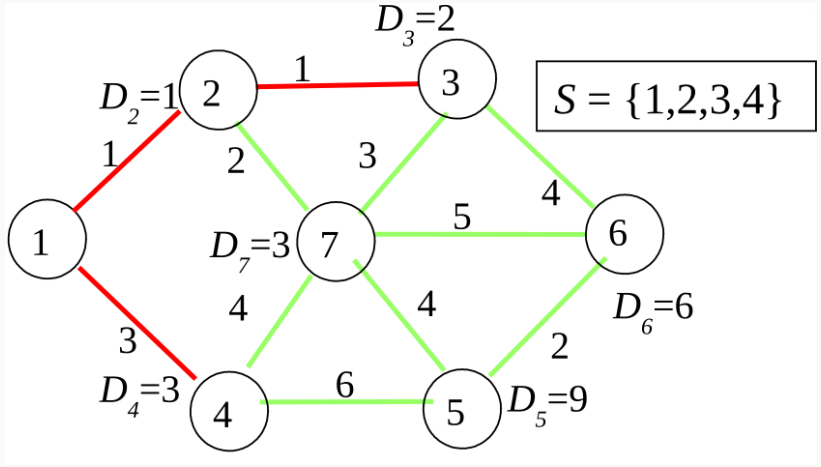


Figure 10: 4 visité, `D[7]`=3 inchangé, `D[5]`=9.

Algorithme de Dijkstra (1 à 5)

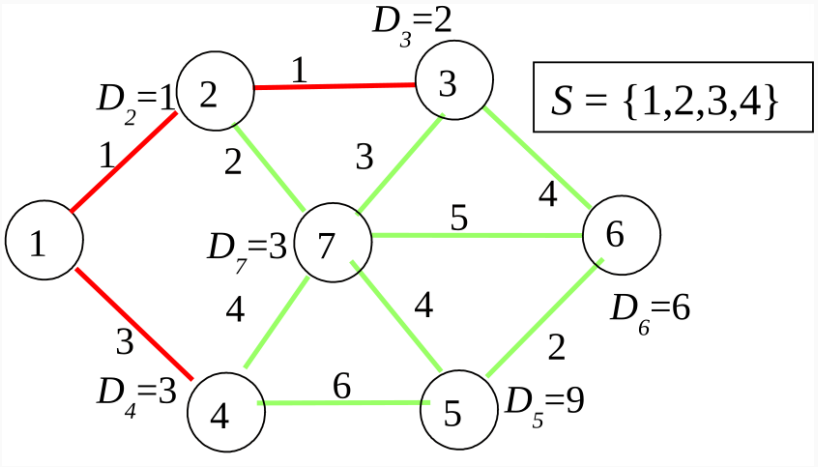


Figure 11: Plus court est 7.

Algorithme de Dijkstra (1 à 5)

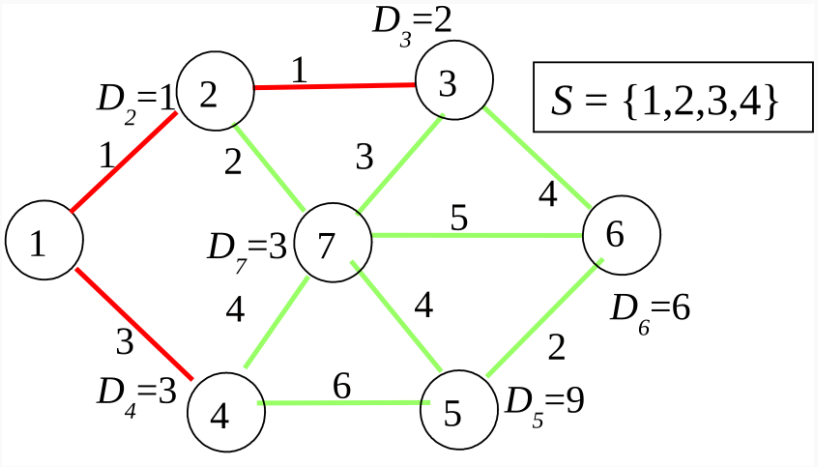


Figure 11: Plus court est 7.

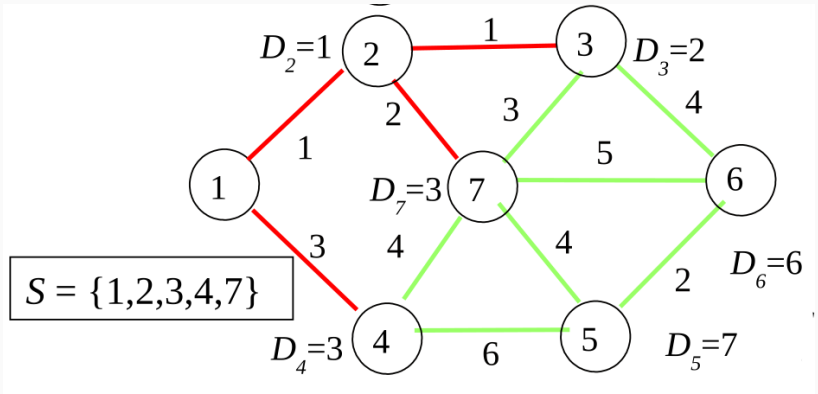


Figure 12: 7 visité, `D[5]=7`, `D[6]=6` inchangé.

Algorithme de Dijkstra (1 à 5)

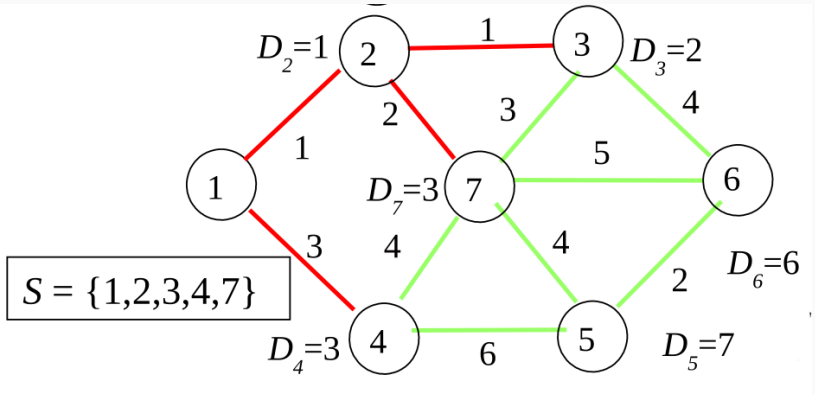


Figure 13: Plus court est 6.

Algorithme de Dijkstra (1 à 5)

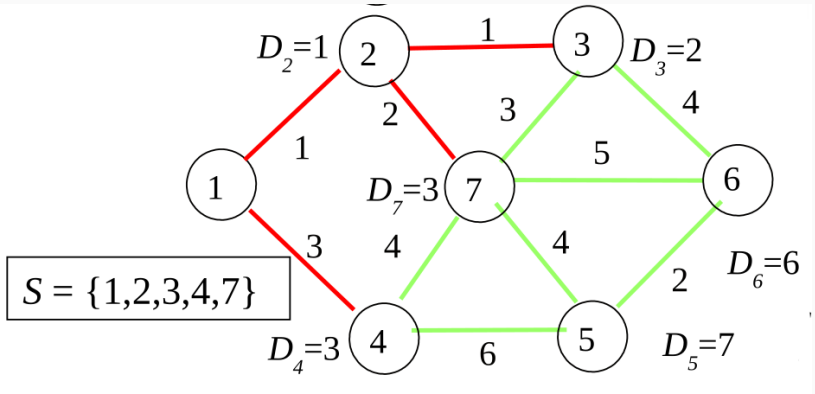


Figure 13: Plus court est 6.

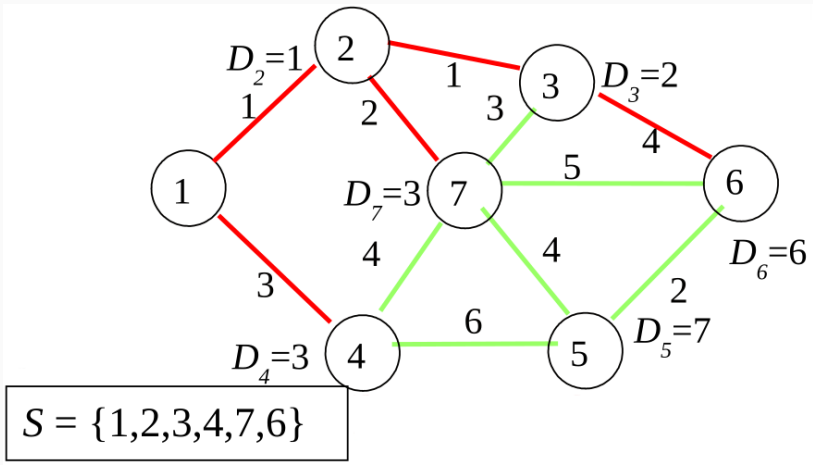


Figure 14: 6 visité, `D[5]=7` inchangé.

Algorithme de Dijkstra (1 à 5)

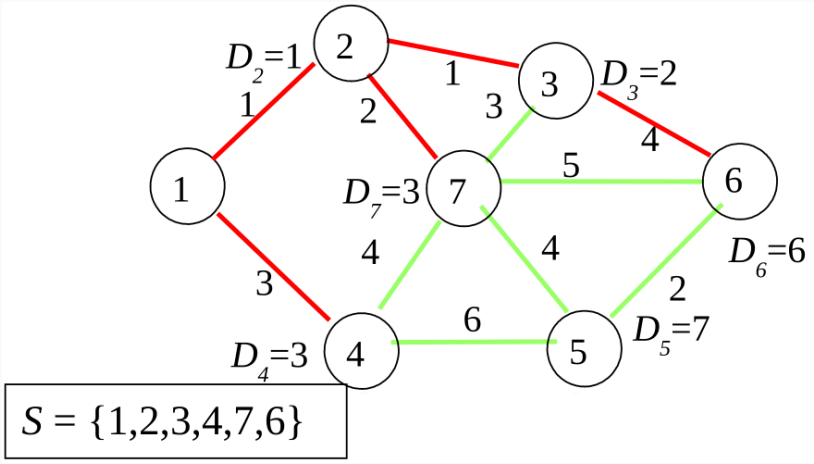


Figure 15: Plus court est 5 et c'est la cible.

Algorithme de Dijkstra (1 à 5)

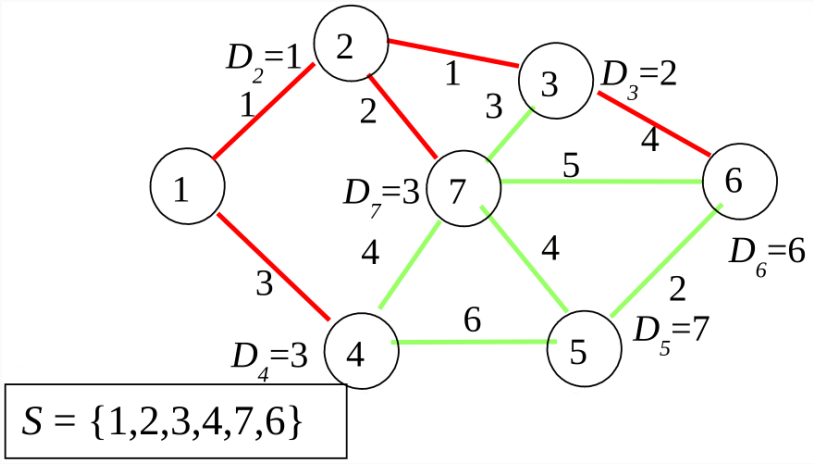


Figure 15: Plus court est 5 et c'est la cible.

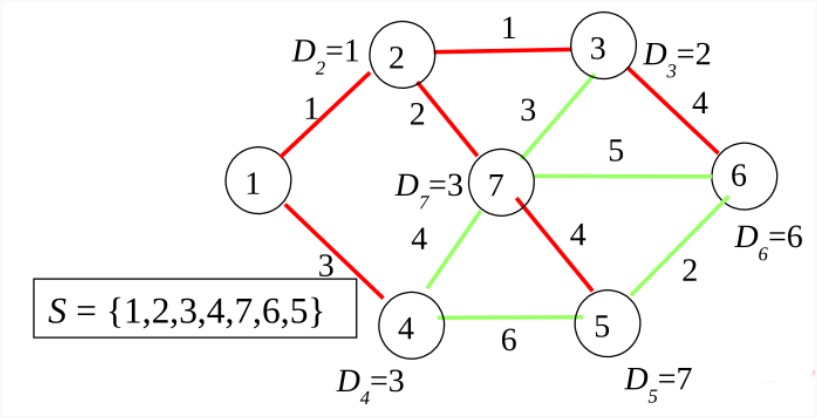


Figure 16: The end, tous les sommets ont été visités.

Algorithme de Dijkstra

Idée générale

- On assigne à chaque noeud une distance 0 pour s, ∞ pour les autres.
- Tous les noeuds sont marqués non-visités.
- Depuis du noeud courant, on suit chaque arête du noeud vers un sommet non visité et on calcule le poids du chemin à chaque voisin et on met à jour sa distance si elle est plus petite que la distance du noeud.
- Quand tous les voisins du noeud courant ont été visités, le noeud est mis à visité (il ne sera plus jamais visité).
- Continuer avec le noeud à la distance la plus faible.
- L'algorithme est terminé losrque le noeud de destination est marqué comme visité, ou qu'on a plus de noeuds qu'on peut visiter et que leur distance est infinie.

Algorithme de Dijkstra

Pseudo-code (5min, matrix)

Algorithme de Dijkstra

Pseudo-code (5min, matrix)

```
tab dijkstra(graph, s, t)
    pour chaque v dans graphe
        distance[v] = infini
        q = ajouter(q, v)
    distance[s] = 0
    tant que non_vide(q)
    // sélection de u t.q. la distance dans q est min
        u = min(q, distance)
        si u == t // on a atteint la cible
            retourne distance
        q = remove(q, u)
        // voisin de u encore dans q
        pour chaque v dans voisinage(u, q)
        // on met à jour la distance du voisin en passant par u
            n_distance = distance[u] + w(u, v)
            si n_distance < distance[v]
                distance[v] = n_distance
    retourne distance
```

Algorithme de Dijkstra

- Cet algorithme, nous donne le plus court chemin mais...
- ne nous donne pas le chemin!

Comment modifier l'algorithme pour avoir le chemin?

Algorithme de Dijkstra

- Cet algorithme, nous donne le plus court chemin mais...
- ne nous donne pas le chemin!

Comment modifier l'algorithme pour avoir le chemin?

- Pour chaque nouveau noeud à visiter, il suffit d'enregistrer d'où on est venu!
- On a besoin d'un tableau `precedent`.

Modifier le pseudo-code ci-dessus pour ce faire (3min matrix)

Algorithme de Dijkstra

```
tab, tab dijkstra(graph, s, t)
    pour chaque v dans graphe
        distance[v] = infini
        precedent[v] = indéfini
        q = ajouter(q, v)
    distance[s] = 0
    tant que non_vide(q)
    // sélection de u t.q. la distance dans q est min
        u = min(q, distance)
        si u == t
            retourne distance
        q = remove(q, u)
        // voisin de u encore dans q
        pour chaque v dans voisinage(u, q)
            n_distance = distance[u] + w(u, v)
            si n_distance < distance[v]
                distance[v] = n_distance
                precedent[v] = u
    retourne distance, precedent
```

Algorithme de Dijkstra

Comment reconstruire un chemin ?

Algorithme de Dijkstra

Comment reconstruire un chemin ?

```
pile parcours(precedent, s, t)
    sommets = vide
    u = t
    // on a atteint t ou on ne connait pas de chemin
    si u != s && precedent[u] != indéfini
        tant que vrai
            sommets = empiler(sommets, u)
            u = precedent[u]
            si u == s // la source est atteinte
                retourne sommets
    retourne sommets
```

Algorithme de Dijkstra amélioré

On peut améliorer l'algorithme

- Avec une file de priorité!

Une file de priorité est

- Une file dont chaque élément possède une priorité,
- Elle existe en deux saveurs: `min` ou `max`:
  - File `min`: les éléments les plus petits sont retirés en premier.
  - File `max`: les éléments les plus grands sont retirés en premier.
- On regarde l'implémentation de la `max`.

Comment on fait ça?

Algorithme de Dijkstra amélioré

On peut améliorer l'algorithme

- Avec une file de priorité!

Une file de priorité est

- Une file dont chaque élément possède une priorité,
- Elle existe en deux saveurs: `min` ou `max`:
  - File `min`: les éléments les plus petits sont retirés en premier.
  - File `max`: les éléments les plus grands sont retirés en premier.
- On regarde l'implémentation de la `max`.

Comment on fait ça?

- On insère les éléments à haute priorité tout devant dans la file!

Les files de priorité

Trois fonction principales

```
booléen est_vide(element) // triviale
element enfiler(element, data, priorite)
data defiler(element)
rien changer_priorite(element, data, priorite)
nombre priorite(element) // utilitaire
```

Pseudo-implémentation: structure (1min)

Les files de priorité

Trois fonction principales

```
booléen est_vide(element) // triviale
element enfiler(element, data, priorite)
data defiler(element)
rien changer_priorite(element, data, priorite)
nombre priorite(element) // utilitaire
```

Pseudo-implémentation: structure (1min)

```
struct element
    data
    priorite
    element suivant
```

Les files de priorité

Pseudo-implémentation: enfiler (2min)

Les files de priorité

Pseudo-implémentation: enfiler (2min)

```
element enfiler(element, data, priorite)
    n_element = creer_element(data, priorite)
    si est_vide(element)
        retourne n_element
    si priorite(n_element) > priorite(element)
        n_element.suivant = element
        retourne n_element
    sinon
        tmp = element
        prec = element
        tant que !est_vide(tmp) && priorite < priorite(tmp)
            prec = tmp
            tmp = tmp.suivant
        prev.suivant = n_element
        n_element.suivant = tmp
        retourne element
```

Les files de priorité

Pseudo-implémentation: defiler (2min)

Les files de priorité

Pseudo-implémentation: defiler (2min)

```
data, element defiler(element)
    si est_vide(element)
        retourne AARGL!
    sinon
        tmp = element.data
        n_element = element.suivant
        liberer(element)
        retourne tmp, n_element
```

Algorithme de Dijkstra avec file de priorité min

```
distance, precedent dijkstra(graphe, s, t):
    distance[source] = 0
    fp = file_p_vide()
    pour v dans sommets(graphe)
        si v != s
            distance[v] = infini
            precedent[v] = indéfini
        fp = enfiler(fp, v, distance[v])
    tant que !est_vide(fp)
        u, fp = defiler(fp)
        pour v dans voisinage de u
            n_distance = distance[u] + w(u, v)
            si n_distance < distance[v]
                distance[v] = n_distance
                precedent[v] = u
                fp = changer_priorite(fp, v, n_distance)
    retourne distance, precedent
```

Algorithme de Dijkstra avec file

```
distance dijkstra(graphe, s, t)
-------------------------------------------------------
    pour v dans sommets(graphe)
O(V)    si v != s
            distance[v] = infini
O(V)        fp = enfiler(fp, v, distance[v]) // notre impl est nulle
-----------------O(V * V)-------------------------------
    tant que !est_vide(fp)
O(1)    u, fp = defiler(fp)
-------------------------------------------------------
O(E)    pour v dans voisinage de u
            n_distance = distance[u] + w(u, v)
            si n_distance < distance[v]
                distance[v] = n_distance
O(V)            fp = changer_priorite(fp, v, n_distance)
-------------------------------------------------------
    retourne distance
```

- Total: $\mathcal{O}(|V|^2 + |E| \cdot |V|)$:
  - Graphe dense: $\mathcal{O}(|V|^3)$
  - Graphe peu dense: $\mathcal{O}(|V|^2)$

Algorithme de Dijkstra avec file

On peut faire mieux

- Avec une meilleure implémentation de la file de priorité:
  - Tas binaire: $\mathcal{O}(|V| \log |V| + |E| \log |V|)$.
  - Tas de Fibonnacci: $\mathcal{O}(|V| + |E| \log |V|)$
- Graphe dense: $\mathcal{O}(|V|^2 \log |V|)$.
- Graphe peu dense: $\mathcal{O}(|V| \log |V|)$.

Algorithme de Dijkstra (exercice, 5min)

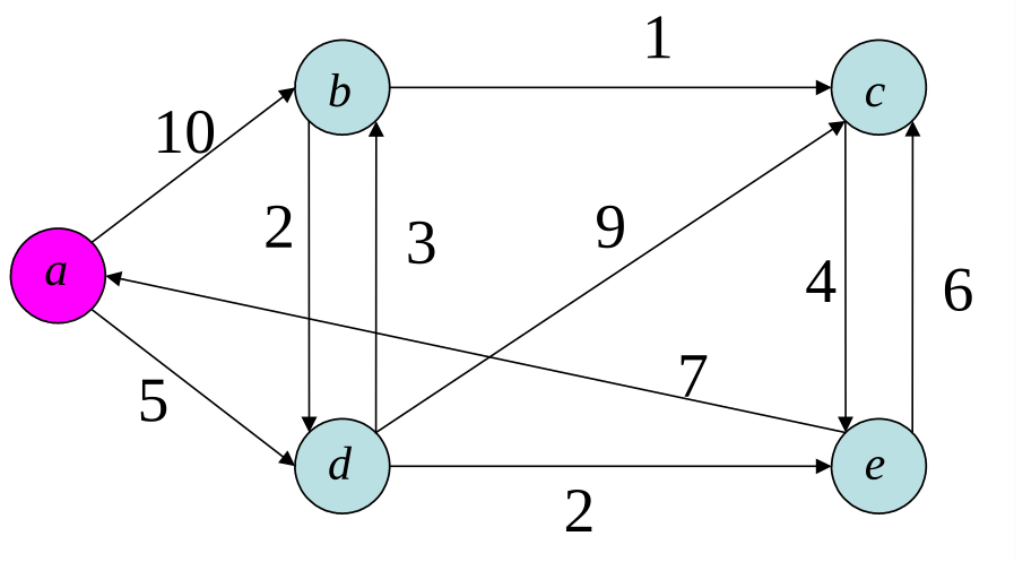


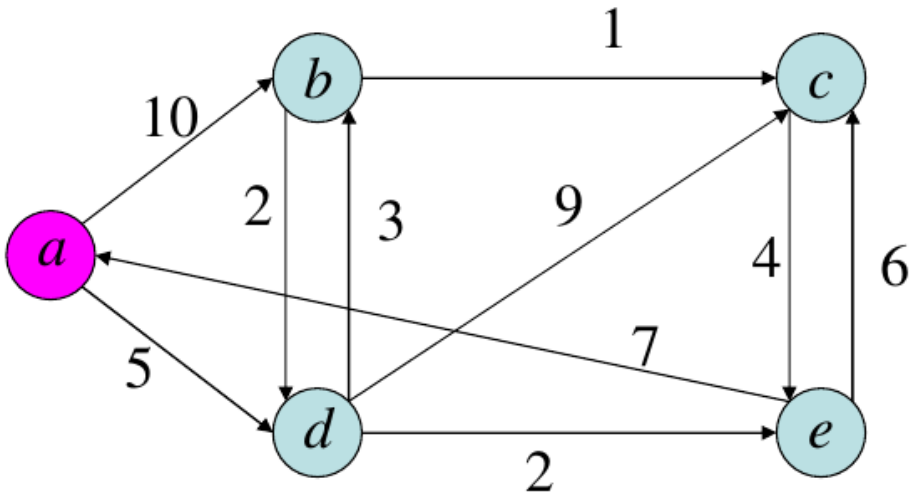
Figure 17: L'exercice.

- Donner la liste de priorité, puis...

A chaque étape donner:

- Le tableau des distances à a;
- Le tableau des prédécesseurs;
- L'état de la file de priorité.

Algorithme de Dijkstra (corrigé)



sommet	a	b	c	d	e
D	0	∞	∞	∞	∞
pred	-	-	-	-	-

$QP = \{ a, b, c, d, e \}$

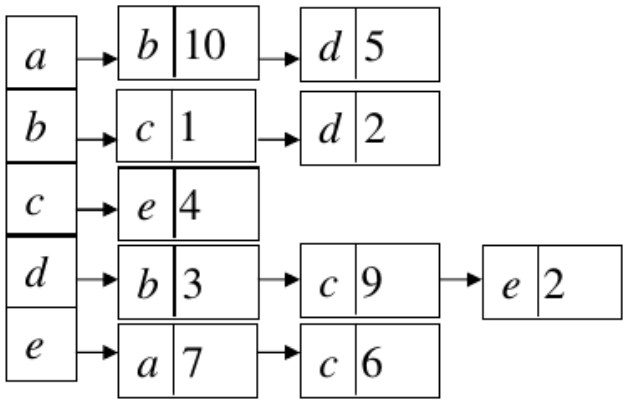
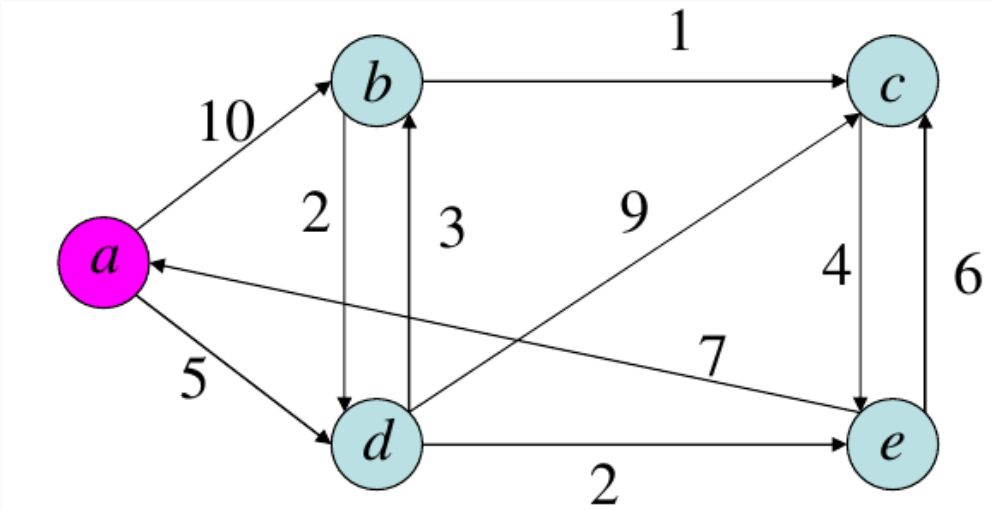


Figure 18: Le corrigé partie 1.

Algorithme de Dijkstra (corrigé)



sommet	*a*	*b*	*c*	*d*	*e*
D	0	10	∞	5	∞
pred	-	*a*	-	*a*	-

$QP = \{b, c, d, e\}$

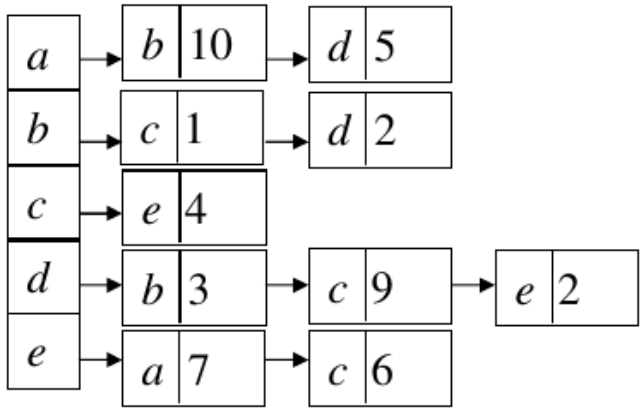
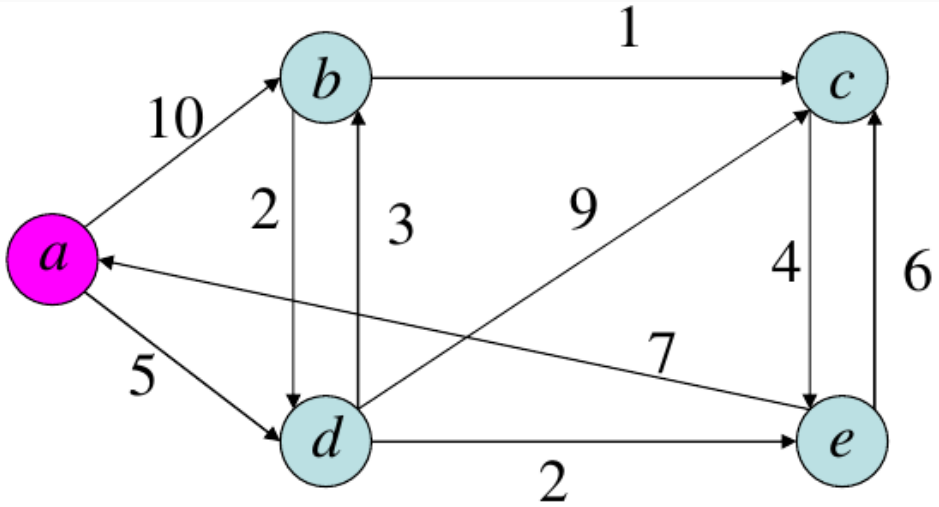


Figure 19: Le corrigé partie 2.

Algorithme de Dijkstra (corrigé)



sommet	*a*	*b*	*c*	*d*	*e*
D	0	8	14	5	7
pred	-	*d*	*d*	*a*	*d*

$QP = \{b, c, e\}$

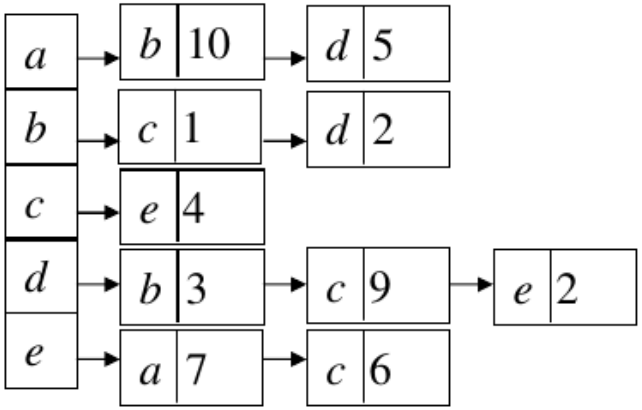
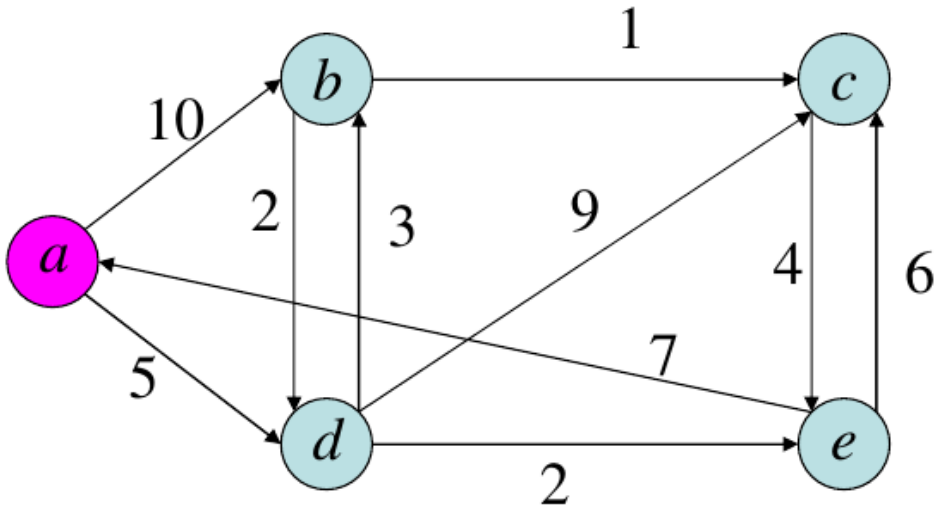


Figure 20: Le corrigé partie 3.

Algorithme de Dijkstra (corrigé)



sommet	*a*	*b*	*c*	*d*	*e*
D	0	8	13	5	7
pred	-	*d*	*e*	*a*	*d*

$QP = \{b, c\}$

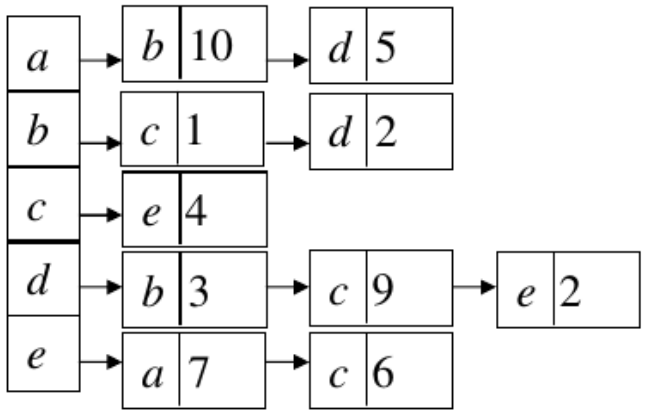
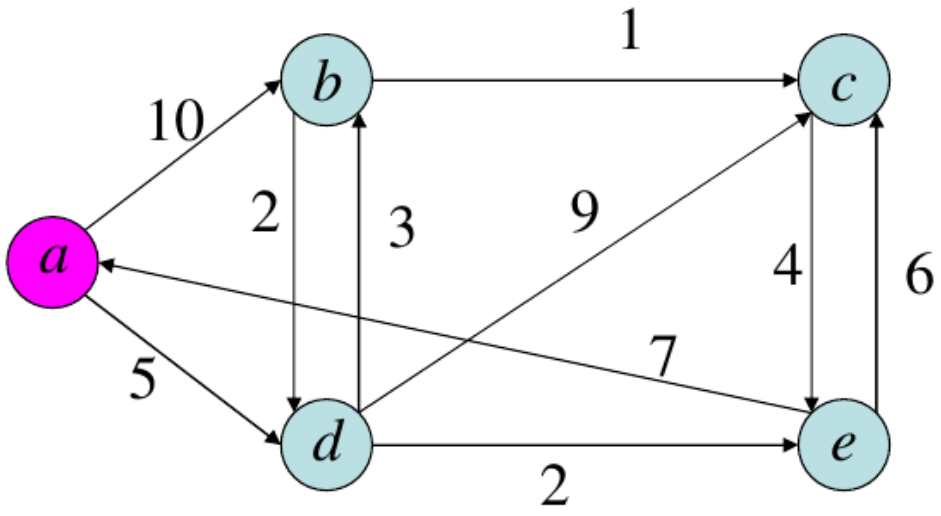


Figure 21: Le corrigé partie 4.

Algorithme de Dijkstra (corrigé)



sommet	*a*	*b*	*c*	*d*	*e*
D	0	8	9	5	7
pred	-	*d*	*b*	*a*	*d*

$QP = \{c\}$

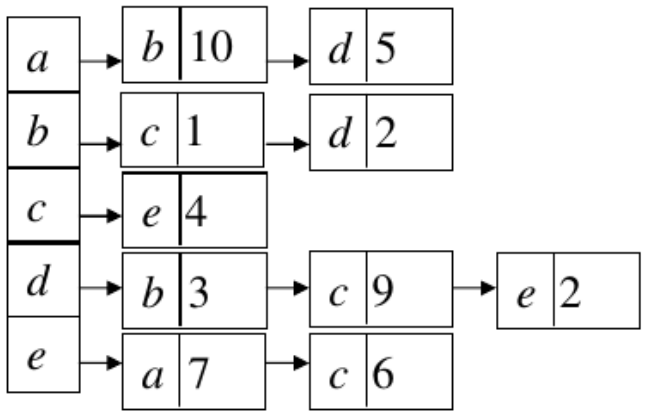
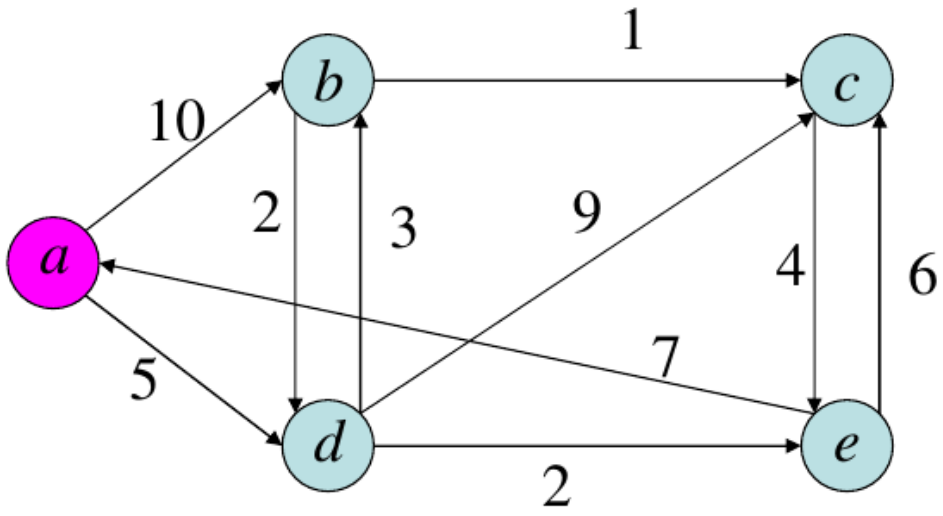


Figure 22: Le corrigé partie 5.

Algorithme de Dijkstra (corrigé)



sommet	*a*	*b*	*c*	*d*	*e*
D	0	8	9	5	7
pred	-	*d*	*b*	*a*	*d*

$QP = \emptyset$

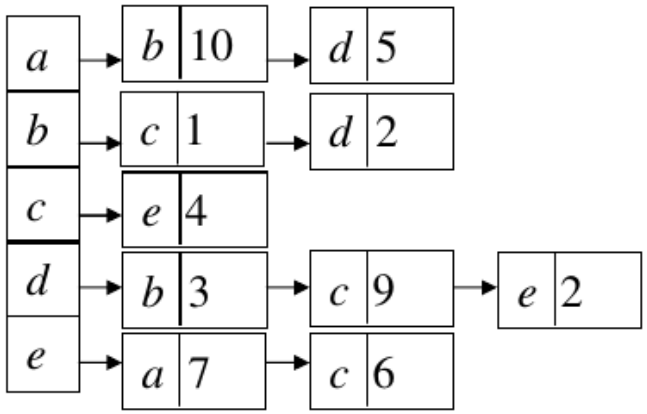


Figure 23: Le corrigé partie 6.

Limitation de l'algorithme de Dijkstra

Que se passe-t-il pour?

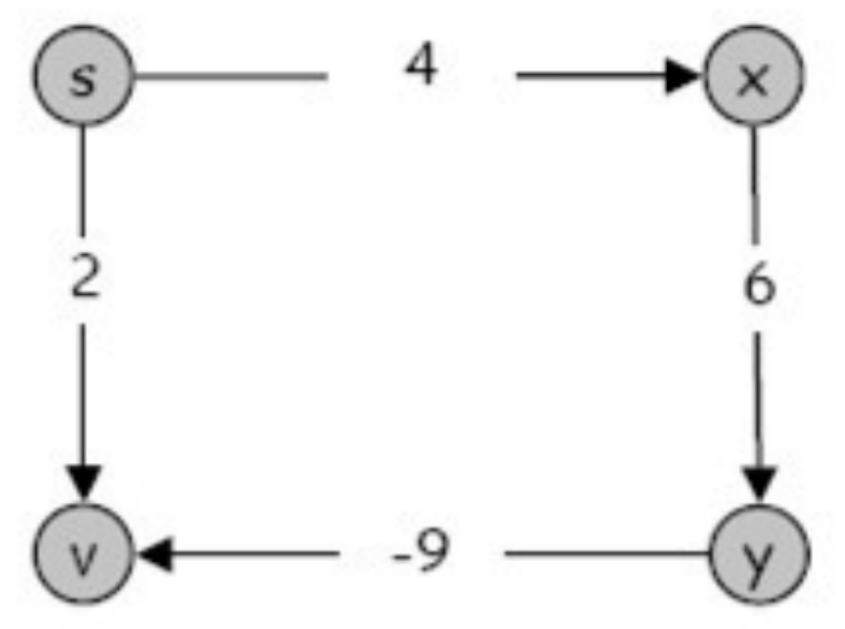


Figure 24: Exemple.

Quel est le problème?

Limitation de l'algorithme de Dijkstra

Que se passe-t-il pour?

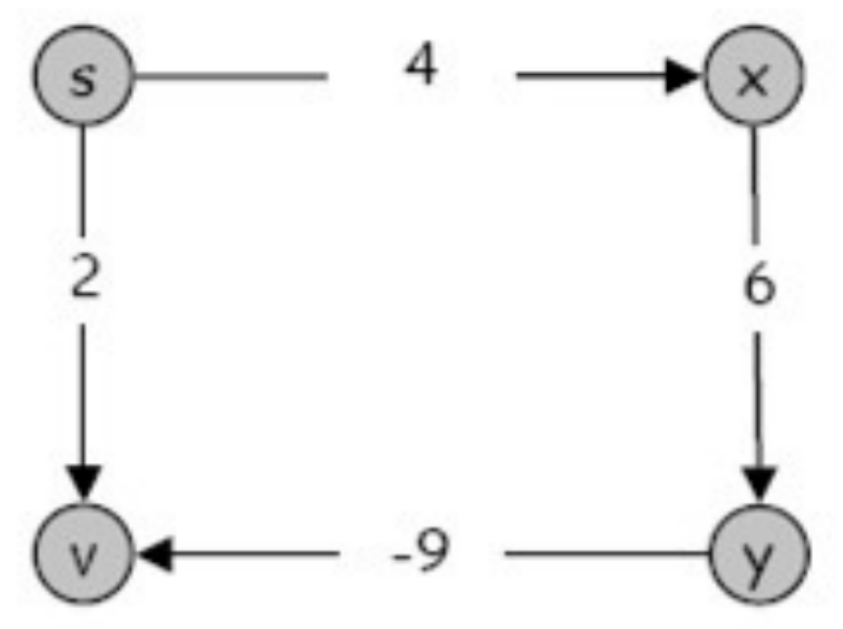


Figure 24: Exemple.

Quel est le problème?

- L'algorithme n'essaiera jamais le chemin `s->x->y->v` et prendra direct `s->v`.
- Ce problème n'apparaît que s'il y a des poids négatifs.

Plus cours chemin pour toute paire de sommets

Comment faire pour avoir toutes les paires?

Plus cours chemin pour toute paire de sommets

Comment faire pour avoir toutes les paires?

- Appliquer Dijkstra sur tous les sommets d'origine.
- Complexité:
  - Graphe dense: $\mathcal{O}(|V|)\mathcal{O}(|V|^2 \log |V|) = \mathcal{O}(|V|^3 \log |V|)$.
  - Graphe peu dense: $\mathcal{O}(|V|)\mathcal{O}(|V| \log |V|) = \mathcal{O}(|V|^2 \log |V|)$.

Plus cours chemin pour toute paire de sommets

Comment faire pour avoir toutes les paires?

- Appliquer Dijkstra sur tous les sommets d'origine.
- Complexité:
  - Graphe dense: $\mathcal{O}(|V|)\mathcal{O}(|V|^2 \log |V|) = \mathcal{O}(|V|^3 \log |V|)$.
  - Graphe peu dense: $\mathcal{O}(|V|)\mathcal{O}(|V| \log |V|) = \mathcal{O}(|V|^2 \log |V|)$.

Solution alternative: Floyd–Warshall

- Pour toutes paires de sommets $u, v \in V$, trouver le chemin de poids minimal reliant u à v.
- Complexité $\mathcal{O}(|V|^3)$, indiqué pour graphes denses.
- Fonctionne avec la matrice d'adjacence.

Algorithme de Floyd–Warshall

Idée générale

- Soit l'ensemble de sommets $V = \{1, 2, 3, 4, ..., n\}$.
- Pour toute paire de sommets, i, j, on considère tous les chemins passant par les sommets intermédiaires $\in \{1, 2, ..., k\}$ avec $k \leq n$.
- On garde pour chaque k la plus petite valeur.

Principe

- A chaque étape, k, on vérifie s'il est plus court d'aller de i à j en passant par le sommet k.
- Si à l'étape $k-1$, le coût du parcours est p, on vérifie si p est plus petit que $p_1 + p_2$, le chemin de i à k, et k à j respectivement.

Algorithme de Floyd–Warshall

The algorithme

Soit $d_{ij}(k)$ le plus court chemin de i à j passant par les sommets $\in \{1, 2, ..., k\}$

$$d_{ij}(k) = \begin{cases} w(i,j), & \text{si } k = 0, \\ \min(d_{ij}(k-1), d_{ik}(k-1) + d_{kj}(k-1)), & \text{sinon.} \end{cases}$$

Algorithme de Floyd–Warshall (exemple)

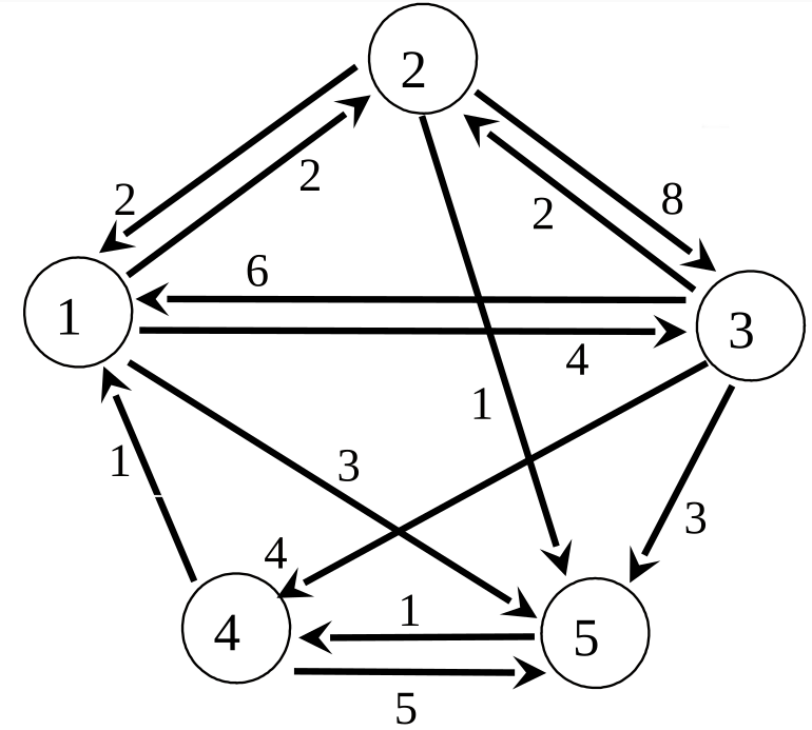


Figure 25: Le graphe, $D = w$.

Que vaut $D^{(0)}$ (3min)?

Algorithme de Floyd–Warshall (exemple)

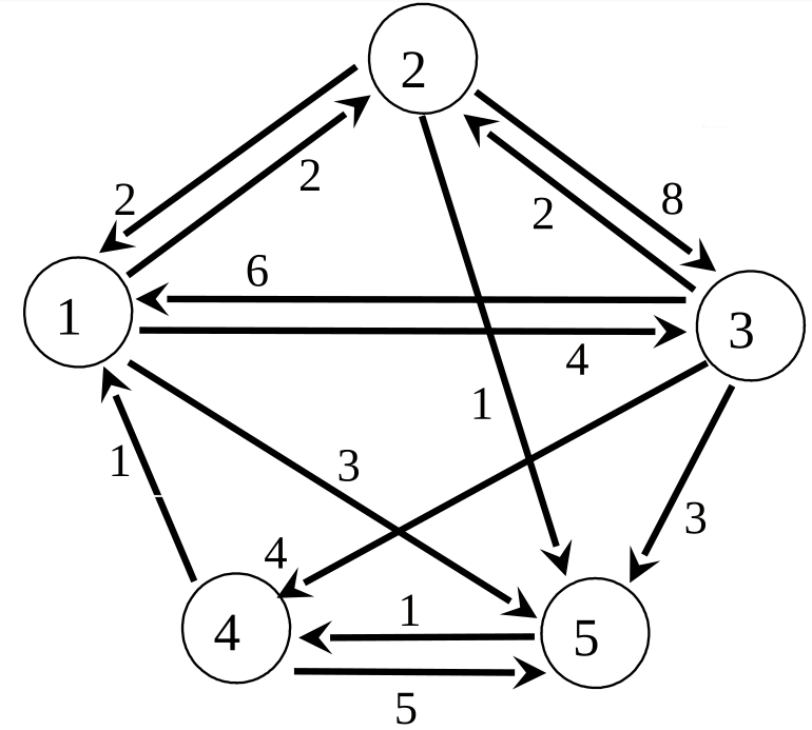


Figure 25: Le graphe, $D = w$.

Que vaut $D^{(0)}$ (3min)?

$$D^{(0)} = \begin{bmatrix} 0 & 2 & 4 & \infty & 3 \\ 2 & 0 & 8 & \infty & 1 \\ 6 & 2 & 0 & 4 & 3 \\ 1 & \infty & \infty & 0 & 5 \\ \infty & \infty & \infty & 1 & 0 \end{bmatrix}$$

Algorithme de Floyd–Warshall (exemple)

On part de $D^{(0)}$?

$$D^{(0)} = \begin{bmatrix} 0 & 2 & 4 & \infty & 3 \\ 2 & 0 & 8 & \infty & 1 \\ 6 & 2 & 0 & 4 & 3 \\ 1 & \infty & \infty & 0 & 5 \\ \infty & \infty & \infty & 1 & 0 \end{bmatrix}$$

Que vaut $D^{(1)}$ (3min)?

Algorithme de Floyd–Warshall (exemple)

On part de $D^{(0)}$?

$$D^{(0)} = \begin{bmatrix} 0 & 2 & 4 & \infty & 3 \\ 2 & 0 & 8 & \infty & 1 \\ 6 & 2 & 0 & 4 & 3 \\ 1 & \infty & \infty & 0 & 5 \\ \infty & \infty & \infty & 1 & 0 \end{bmatrix}$$

Que vaut $D^{(1)}$ (3min)?

$$D^{(0)} = \begin{bmatrix} 0 & 2 & 4 & \infty & 3 \\ 2 & 0 & \mathbf{6} & \infty & 1 \\ 6 & 2 & 0 & 4 & 3 \\ 1 & \mathbf{3} & \mathbf{5} & 0 & \mathbf{4} \\ \infty & \infty & \infty & 1 & 0 \end{bmatrix}$$

Algorithme de Floyd–Warshall (exemple)

On part de $D^{(0)}$

$$D^{(0)} = \begin{bmatrix} 0 & 2 & 4 & \infty & 3 \\ 2 & 0 & 8 & \infty & 1 \\ 6 & 2 & 0 & 4 & 3 \\ 1 & \infty & \infty & 0 & 5 \\ \infty & \infty & \infty & 1 & 0 \end{bmatrix}$$

Que vaut $D^{(1)}$ (3min)?

Algorithme de Floyd–Warshall (exemple)

On part de $D^{(0)}$

$$D^{(0)} = \begin{bmatrix} 0 & 2 & 4 & \infty & 3 \\ 2 & 0 & 8 & \infty & 1 \\ 6 & 2 & 0 & 4 & 3 \\ 1 & \infty & \infty & 0 & 5 \\ \infty & \infty & \infty & 1 & 0 \end{bmatrix}$$

Que vaut $D^{(1)}$ (3min)?

$$D^{(1)} = \begin{bmatrix} 0 & 2 & 4 & \infty & 3 \\ 2 & 0 & \mathbf{6} & \infty & 1 \\ 6 & 2 & 0 & 4 & 3 \\ 1 & \mathbf{3} & \mathbf{5} & 0 & \mathbf{4} \\ \infty & \infty & \infty & 1 & 0 \end{bmatrix}$$

Exemple

$$D^{(1)}_{42} = D^{(0)}_{41} + D^{(0)}_{12} = 1 + 2 < \infty.$$

Algorithme de Floyd–Warshall (exemple)

On part de $D^{(1)}$

$$D^{(1)} = \begin{bmatrix} 0 & 2 & 4 & \infty & 3 \\ 2 & 0 & 6 & \infty & 1 \\ 6 & 2 & 0 & 4 & 3 \\ 1 & 3 & 5 & 0 & 4 \\ \infty & \infty & \infty & 1 & 0 \end{bmatrix}$$

Que vaut $D^{(2)}$ (3min)?

Algorithme de Floyd–Warshall (exemple)

On part de $D^{(1)}$

$$D^{(1)} = \begin{bmatrix} 0 & 2 & 4 & \infty & 3 \\ 2 & 0 & 6 & \infty & 1 \\ 6 & 2 & 0 & 4 & 3 \\ 1 & 3 & 5 & 0 & 4 \\ \infty & \infty & \infty & 1 & 0 \end{bmatrix}$$

Que vaut $D^{(2)}$ (3min)?

$$D^{(2)} = \begin{bmatrix} 0 & 2 & 4 & \infty & 3 \\ 2 & 0 & 6 & \infty & 1 \\ \mathbf{4} & 2 & 0 & 4 & 3 \\ 1 & 3 & 5 & 0 & 4 \\ \infty & \infty & \infty & 1 & 0 \end{bmatrix}$$

Algorithme de Floyd–Warshall (exemple)

On part de $D^{(2)}$

$$D^{(2)} = \begin{bmatrix} 0 & 2 & 4 & \infty & 3 \\ 2 & 0 & 6 & \infty & 1 \\ 4 & 2 & 0 & 4 & 3 \\ 1 & 3 & 5 & 0 & 4 \\ \infty & \infty & \infty & 1 & 0 \end{bmatrix}$$

Que vaut $D^{(3)}$ (3min)?

Algorithme de Floyd–Warshall (exemple)

On part de $D^{(2)}$

$$D^{(2)} = \begin{bmatrix} 0 & 2 & 4 & \infty & 3 \\ 2 & 0 & 6 & \infty & 1 \\ 4 & 2 & 0 & 4 & 3 \\ 1 & 3 & 5 & 0 & 4 \\ \infty & \infty & \infty & 1 & 0 \end{bmatrix}$$

Que vaut $D^{(3)}$ (3min)?

$$D^{(3)} = \begin{bmatrix} 0 & 2 & 4 & \mathbf{8} & 3 \\ 2 & 0 & 6 & \mathbf{10} & 1 \\ 4 & 2 & 0 & 4 & 3 \\ 1 & 3 & 5 & 0 & 4 \\ \infty & \infty & \infty & 1 & 0 \end{bmatrix}$$

Algorithme de Floyd–Warshall (exemple)

On part de $D^{(3)}$

$$D^{(3)} = \begin{bmatrix} 0 & 2 & 4 & 8 & 3 \\ 2 & 0 & 6 & 10 & 1 \\ 4 & 2 & 0 & 4 & 3 \\ 1 & 3 & 5 & 0 & 4 \\ \infty & \infty & \infty & 1 & 0 \end{bmatrix}$$

Que vaut $D^{(4)}$ (3min)?

Algorithme de Floyd–Warshall (exemple)

On part de $D^{(3)}$

$$D^{(3)} = \begin{bmatrix} 0 & 2 & 4 & 8 & 3 \\ 2 & 0 & 6 & 10 & 1 \\ 4 & 2 & 0 & 4 & 3 \\ 1 & 3 & 5 & 0 & 4 \\ \infty & \infty & \infty & 1 & 0 \end{bmatrix}$$

Que vaut $D^{(4)}$ (3min)?

$$D^{(4)} = \begin{bmatrix} 0 & 2 & 4 & 8 & 3 \\ 2 & 0 & 6 & 10 & 1 \\ 4 & 2 & 0 & 4 & 3 \\ 1 & 3 & 5 & 0 & 4 \\ \mathbf{2} & \mathbf{4} & \mathbf{6} & 1 & 0 \end{bmatrix}$$

Algorithme de Floyd–Warshall (exemple)

On part de $D^{(4)}$

$$D^{(4)} = \begin{bmatrix} 0 & 2 & 4 & 8 & 3 \\ 2 & 0 & 6 & 10 & 1 \\ 4 & 2 & 0 & 4 & 3 \\ 1 & 3 & 5 & 0 & 4 \\ 2 & 4 & 6 & 1 & 0 \end{bmatrix}$$

Que vaut $D^{(5)}$ (3min)?

Algorithme de Floyd–Warshall (exemple)

On part de $D^{(4)}$

$$D^{(4)} = \begin{bmatrix} 0 & 2 & 4 & 8 & 3 \\ 2 & 0 & 6 & 10 & 1 \\ 4 & 2 & 0 & 4 & 3 \\ 1 & 3 & 5 & 0 & 4 \\ 2 & 4 & 6 & 1 & 0 \end{bmatrix}$$

Que vaut $D^{(5)}$ (3min)?

$$D^{(5)} = \begin{bmatrix} 0 & 2 & 4 & \mathbf{4} & 3 \\ 2 & 0 & 6 & \mathbf{2} & 1 \\ 4 & 2 & 0 & 4 & 3 \\ 1 & 3 & 5 & 0 & 4 \\ 2 & 4 & 6 & 1 & 0 \end{bmatrix}$$

Algorithme de Floyd–Warshall

The pseudo-code (10min)

- Quelle structure de données?
- Quelle initialisation?
- Quel est le code pour le calcul de la matrice D?

Algorithme de Floyd–Warshall

The pseudo-code

- Quelle structure de données?

```
int distance[n][n];
```

Algorithme de Floyd–Warshall

The pseudo-code

- Quelle structure de données?

```
int distance[n][n];
```

- Quelle initialisation?

```
matrice ini_floyd_warshall(distance, n, w)
    pour i de 1 à n
        pour j de 1 à n
            distance[i][j] = w(i,j)
    retourne distance
```

Algorithme de Floyd–Warshall

The pseudo-code

- Quel est le code pour le calcul de la matrice D?

```
matrice floyd_warshall(distance, n, w)
    pour k de 1 à n
        pour i de 1 à n
            pour j de 1 à n
                distance[i][j] = min(distance[i][j],
                    distance[i][k] + distance[k][j])
    retourne distance
```

Algorithme de Floyd–Warshall

La matrice de précédence

- On a pas encore vu comment reconstruire le plus court chemin!
- On définit, $P_{ij}^{(k)}$, qui est le prédécesseur du sommet j depuis i avec les sommets intermédiaires $\in \{1, 2, ..., k\}$.

$$P_{ij}^{(0)} = \begin{cases} \text{vide}, & \text{si } i = j, \text{ ou } w(i,j) = \infty \\ i, & \text{sinon.} \end{cases}$$

- Mise à jour

$$P_{ij}^{(k)} = \begin{cases} P_{\mathbf{i}j}^{(k-1)}, & \text{si } d_{ij}^{(k)} \leq d_{ik}^{(k-1)} + d_{kj}^{(k-1)} \\ P_{\mathbf{k}j}^{(k-1)}, & \text{sinon.} \end{cases}$$

Algorithme de Floyd–Warshall

La matrice de précédence

- On a pas encore vu comment reconstruire le plus court chemin!
- On définit, $P_{ij}^{(k)}$, qui est le prédécesseur du sommet j depuis i avec les sommets intermédiaires $\in \{1, 2, ..., k\}$.

$$P_{ij}^{(0)} = \begin{cases} \text{vide}, & \text{si } i = j, \text{ ou } w(i,j) = \infty \\ i, & \text{sinon.} \end{cases}$$

- Mise à jour

$$P_{ij}^{(k)} = \begin{cases} P_{\mathbf{i}j}^{(k-1)}, & \text{si } d_{ij}^{(k)} \leq d_{ik}^{(k-1)} + d_{kj}^{(k-1)} \\ P_{\mathbf{k}j}^{(k-1)}, & \text{sinon.} \end{cases}$$

- Moralité: si le chemin est plus court en passant par k, alors il faut utiliser son prédécesseur!

Algorithme de Floyd–Warshall

La matrice de précédence (pseudo-code, 3min)

Algorithme de Floyd–Warshall

La matrice de précédence (pseudo-code, 3min)

```
matrice, matrice floyd_warshall(distance, n, w)
    pour k de 1 à n
        pour i de 1 à n
            pour j de 1 à n
                n_distance = distance[i][k] + distance[k][j]
                if n_distance < distance[i][j]
                    distance[i][j] = n_distance
                    précédence[i][j] = précédence[k][j]
    retourne distance, précédence
```

Algorithme de Floyd–Warshall (exercice)

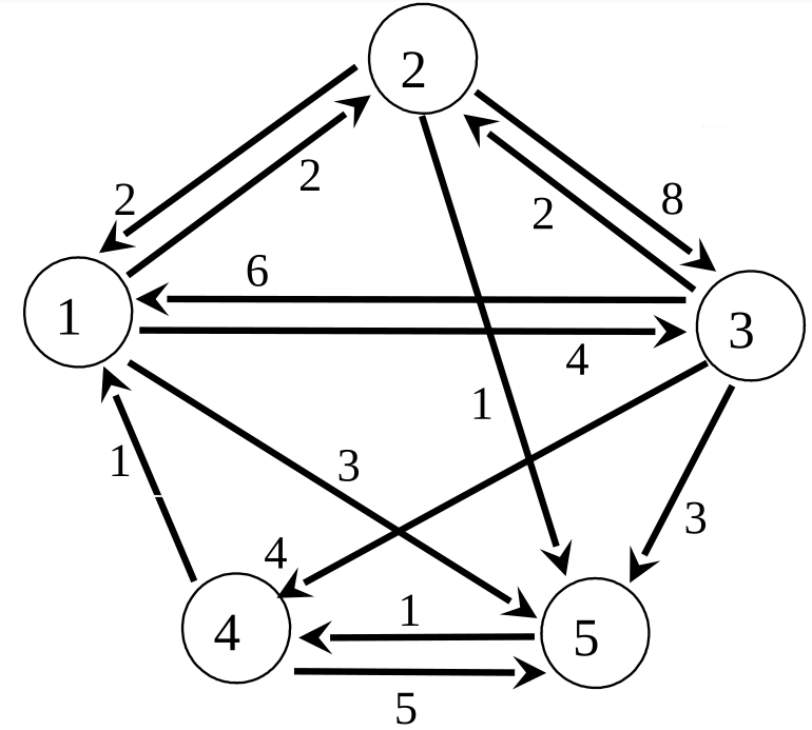


Figure 26: Le graphe, $D = w$.

Que vaut $P^{(0)}$ (3min)?

Algorithme de Floyd–Warshall (exercice)

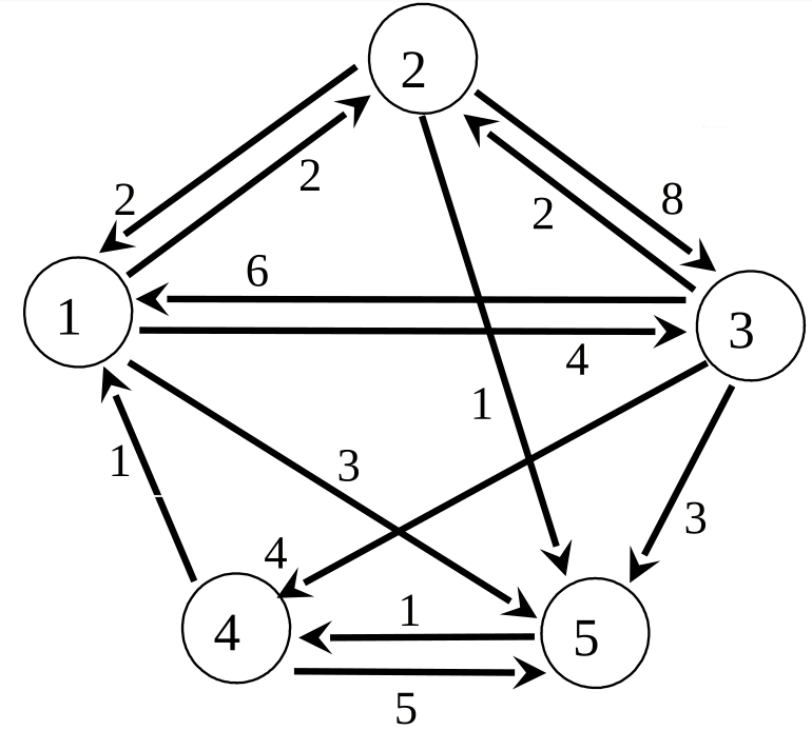


Figure 26: Le graphe, $D = w$.

Que vaut $P^{(0)}$ (3min)?

$$P^{(0)} = \begin{bmatrix} - & 1 & 1 & - & 1 \\ 2 & - & 2 & - & 2 \\ 3 & 3 & - & 3 & 3 \\ 4 & - & - & - & 4 \\ - & - & - & 5 & - \end{bmatrix}$$

Algorithme de Floyd–Warshall (exercice)

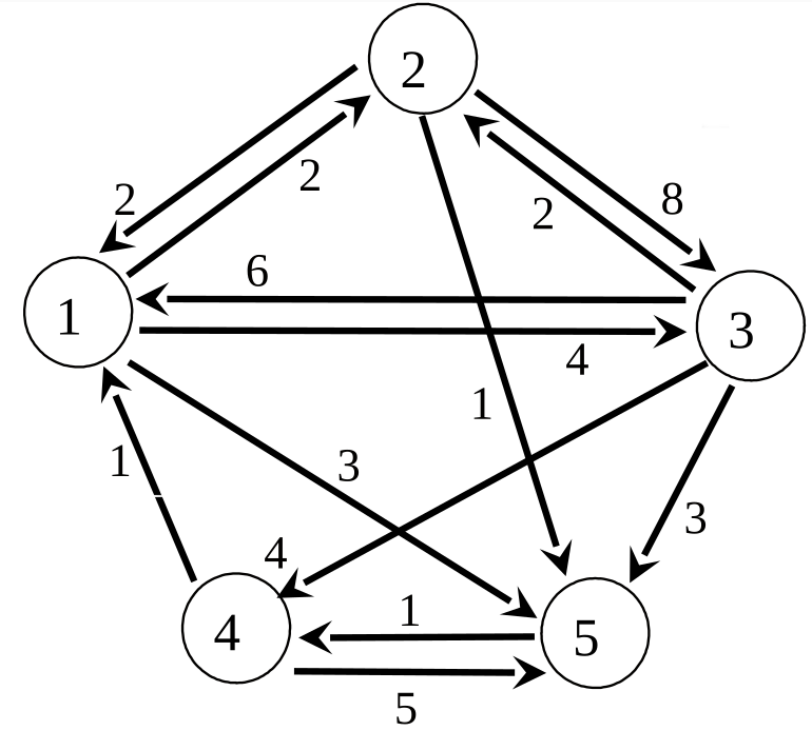


Figure 27: Le graphe, $D = w$.

Que vaut $P^{(5)}$ (10min)?

Algorithme de Floyd–Warshall (exercice)

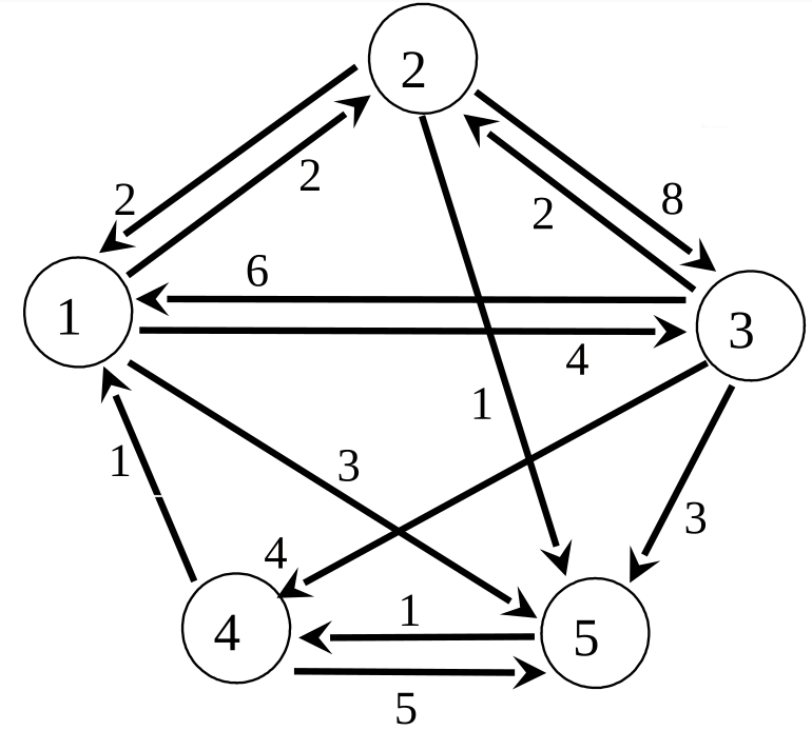


Figure 27: Le graphe, $D = w$.

Que vaut $P^{(5)}$ (10min)?

$$P^{(5)} = \begin{bmatrix} - & 1 & 1 & 5 & 1 \\ 2 & - & 1 & 5 & 2 \\ 2 & 3 & - & 3 & 3 \\ 4 & 1 & 1 & - & 1 \\ 4 & 1 & 1 & 5 & - \end{bmatrix}$$

Exercice: retrouver le chemin entre 1 et 4 (5min)

$$P = \begin{bmatrix} - & 1 & 1 & 5 & 1 \\ 2 & - & 1 & 5 & 2 \\ 2 & 3 & - & 3 & 3 \\ 4 & 1 & 1 & - & 4 \\ 4 & 1 & 1 & 5 & - \end{bmatrix}$$

Exercice: retrouver le chemin entre 1 et 4 (5min)

$$P = \begin{bmatrix} - & 1 & 1 & 5 & 1 \\ 2 & - & 1 & 5 & 2 \\ 2 & 3 & - & 3 & 3 \\ 4 & 1 & 1 & - & 4 \\ 4 & 1 & 1 & 5 & - \end{bmatrix}$$

Solution

- Le sommet $5 = P_{14}$, on a donc, $5 \to 4$, on veut connaître le prédécesseur de 5.
- Le sommet $1 = P_{15}$, on a donc, $1 \to 5 \to 4$. The end.

Exercice complet

Appliquer l'algorithme de Floyd–Warshall au graphe suivant

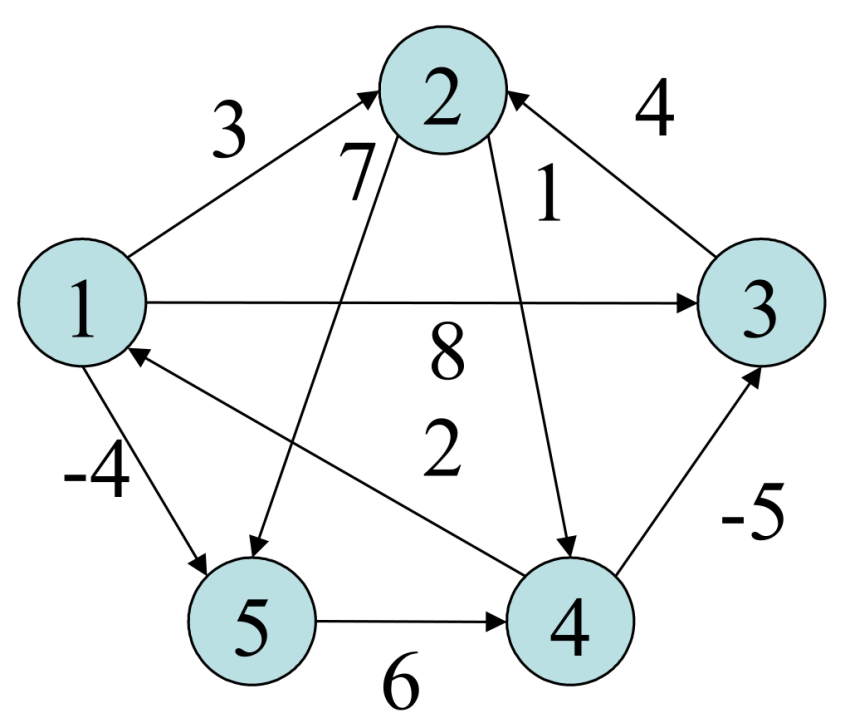


Figure 28: The exorcist.

- Bien indiquer l'état de D et P à chaque étape!
- Ne pas oublier de faire la matrice d'adjacence évidemment...