Résumé

1) Déterminer une sous-structure optimale dont on a besoin pour résoudre le problème

   Fib(n) : $n^{ème}$ nombre de Fibonacci

   OPT(i) : valeur d'une solution optimale en se restreignant aux intervalles 1, …, i.

   X[i, j] : il existe un sous-ensemble de $\{a_1, a_2, \ldots, a_i\}$ dont la somme des éléments est j.

2) Caractériser (par une équation) cette sous-structure optimale

   $Fib(n) = Fib(n-1)+Fib(n-2)$

   $OPT(i) = \max \{ v_i + OPT(der(i)) \ , \ OPT(i-1)\}$

   $X[i, j] = X[i - 1, j]$ ou $X[i-1, j - a_i]$

3) En déduire la valeur d'une solution optimale

   Fib(n) ; OPT(n) ; X[n, W]

4) Ecrire un algorithme de programmation dynamique pour résoudre le problème

   - récursif (avec mémoïsation), ou
   - itératif (approche du bas vers le haut)

5) Pour retrouver la solution optimale à partir de sa valeur : retour en arrière (backtrack)

Plus courts chemins

Problème : *Entrée :* un graphe $G=(S,A)$ orienté et valué (c) ;
sommet origine s ; sommet destination v.
Sortie : un plus court chemin de s à v dans G.
(ou *A(s)*, arborescence des plus courts chemins d'origine s).

On note $d(x)$ la distance de s à x , pour tout $x \in S$.

Propriété : Il existe un plus court chemin entre s et x si et seulement si il n'existe pas de circuit absorbant dans un chemin entre s et x dans G.

Supposons qu'il n'y ait pas de circuit absorbant accessible à partir de s.
Comment calculer *A(s)* ?

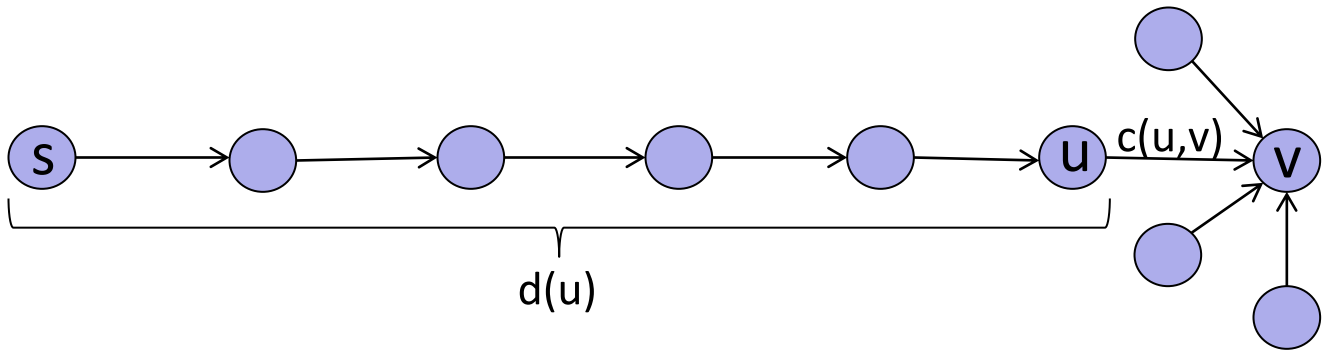
- L'algorithme de Dijkstra ne retourne pas toujours une arborescence des plus courts chemins si les coûts des arcs sont négatifs.
- Augmenter le coût des arcs de façon à avoir seulement des coûts positifs ne marche pas non plus.

Plus courts chemins : programme dynamique

Comment calculer le plus court chemin de s à v ?

On ne connait pas la réponse ? On essaie toutes les solutions (et on prend la meilleure).

Programmation dynamique : récursion + mémoïsation + essais



Quel est le dernier arc du plus court chemin entre s et v ?

On ne sait pas $\Rightarrow$ on les essaie tous.

$$\begin{cases} d(v) = \min\limits_{u \mid (u,v) \in A} \{\ d(u) + c(u,v)\ \} & \text{si } v \neq s \\ d(s) = 0 \end{cases}$$

Plus courts chemins : programme dynamique

On a : $\begin{cases} d(s) = 0 \\ d(v) = \min\limits_{u \mid (u,v) \in A} \{ \ d(u) + c(u,v) \ \} \quad \text{si } v \neq s \end{cases}$

Algorithme récursif :

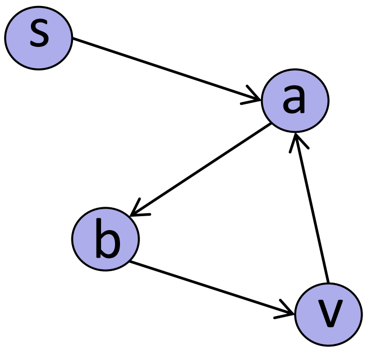
```
d(s,v) :
   si v=s retourner 0
   d_min = + ∞
   pour tout prédécesseur u de v faire
        d_courant = d(s,u) + c(u,v)
        si (d_courant < d_min)
            d_min = d_courant
   retourner d_min
```

Plus courts chemins : programme dynamique

Algorithme récursif avec mémoïsation :

```
pour tout sommet i de S\{s}
mémo[i] = vide
mémo[s] = 0

d(s,v) :
  si mémo[v] est vide
        d_min = + ∞
        pour tout prédécesseur u de v faire
              d_courant = d(s,u) + c(u,v)
              si (d_courant < d_min)
                    d_min = d_courant
        mémo[v]=d_min
  retourner mémo[v]
```

$$d(s) = 0$$
$$d(v) = \min_{u \mid (u,v) \in A} \{ d(u) + c(u,v) \}$$

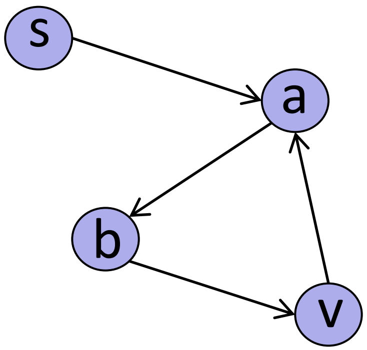


Plus courts chemins : programme dynamique

Algorithme récursif avec mémoïsation :

$$d(s) = 0$$
$$d(v) = \min_{u \mid (u,v) \in A} \{ d(u) + c(u,v) \}$$

```
pour tout sommet i de S\{s}
mémo[i] = vide
mémo[s] = 0

d(s,v) :
  si mémo[v] est vide
        d_min = + ∞
        pour tout prédécesseur u de v faire
                d_courant = d(s,u) + c(u,v)
                si (d_courant < d_min)
                    d_min = d_courant
        mémo[v]=d_min
  retourner mémo[v]
```



Présence d'un circuit : l'algorithme ne termine pas.

Plus courts chemins dans un graphe sans circuit

Cet algorithme est valide s’il n’y a pas de circuit dans le graphe.

```
d(s,v)
  si mémo[v] est vide
      d_min = + ∞
      pour tout prédécesseur u de v faire
          d_courant = d(s,u) + c(u,v)
          si (d_courant < d_min)
              d_min = d_courant
      mémo[v]=d_min
  retourner mémo[v]
```

Quelle est la complexité de cet algorithme ?

```
d(s,v)
  si mémo[v] est vide
       d_min = + ∞
       pour tout prédécesseur u de v faire
              d_min = min{d_min, d(s,u) + c(u,v) }
mémo[v]=d_min
retourner mémo[v]
```

A. O(n)
B. O(n+m)
C. $O(n^2)$
D. O(nm)

Quelle est la complexité de cet algorithme ?

```
d(s,v)
  si mémo[v] est vide
       d_min = + ∞
       pour tout prédécesseur u de v faire
             d_min = min{d_min, d(s,u) + c(u,v) }
mémo[v]=d_min
retourner mémo[v]
```

A. O(n)
B. O(n+m)
C. O(n^2)
D. O(nm)

Complexité de l'appel non mémoïsé de d(v) : $d^-(v) + 1$
Appels non mémoïsés : un par sommet (n sommets)
=> Complexité totale = O(n+m)

Plus courts chemins dans un graphe sans circuit

Version ``du bas vers le haut” de cet algorithme
(on suppose que s est une racine) :

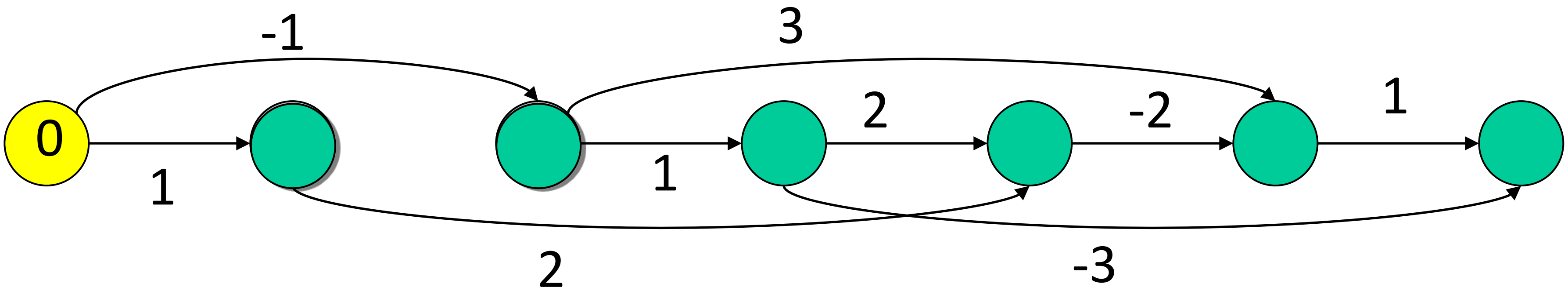
soit L=(s_1,…,s_n) une liste topologique des sommets de G telle que s= s_1
pour tout sommet $x \neq s$ d[x]= $+\infty$; d[s]=0
pour k allant de 1 à n
    pour tout successeur y de s_k faire
        si $d(y) > d(s_k) + c(s_k, y)$ alors
            $d(y) = d(s_k) + c(s_k, y)$

On obtient *A(s)*, une arborescence des plus courts chemins d’origine s.

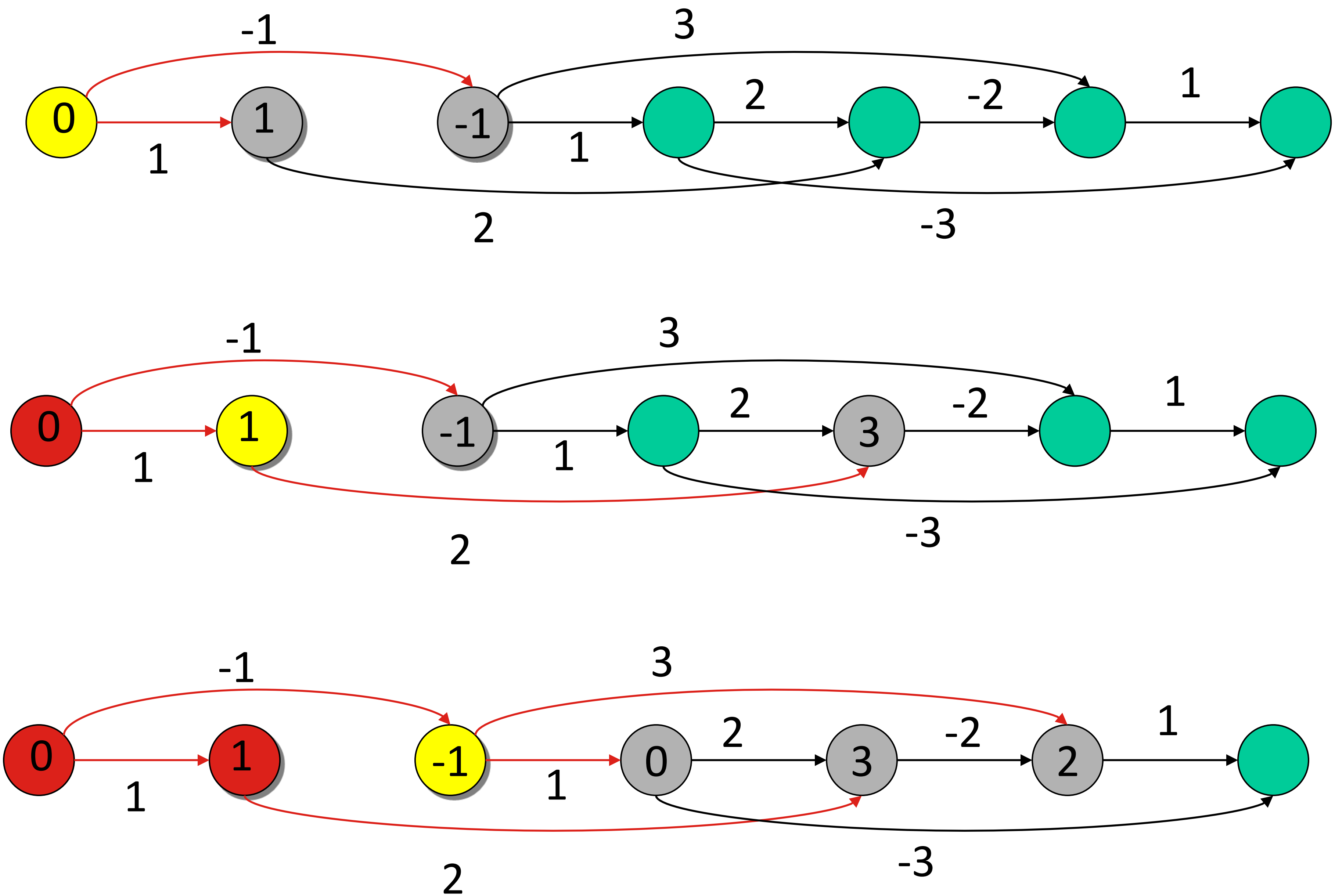
C’est l’algorithme de Bellman.

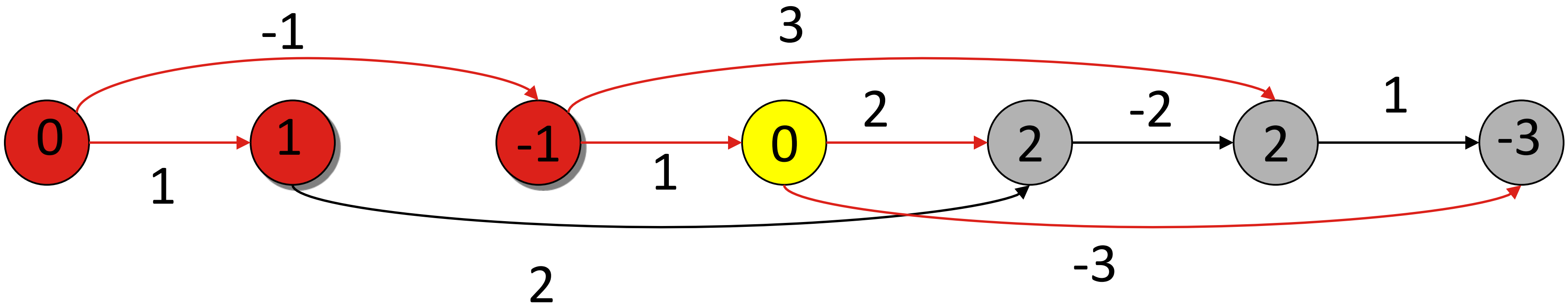
Complexité totale = O(n+m)

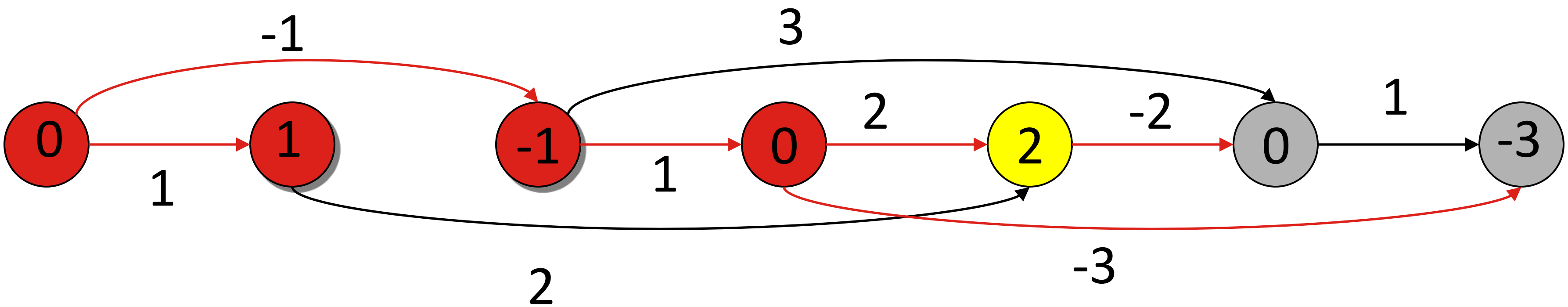
Exemple :

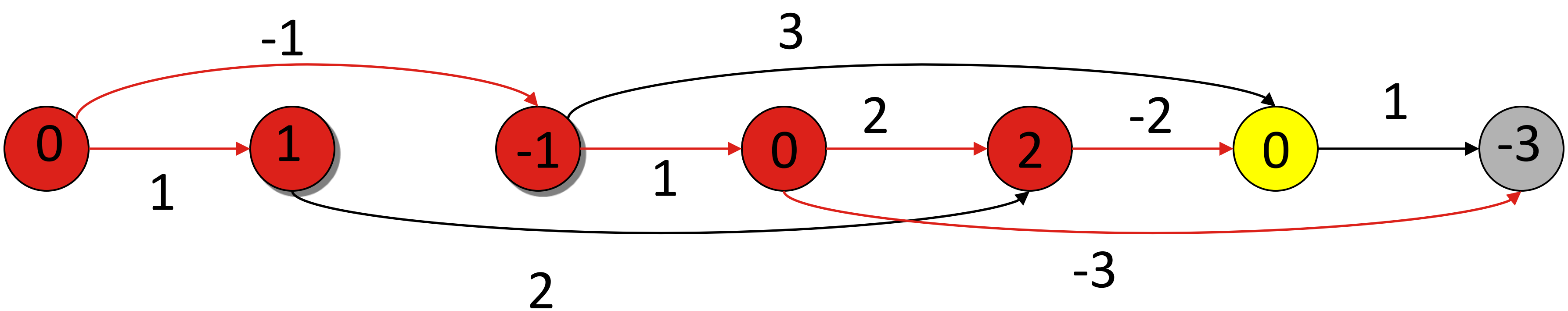


Exemple :




-1
3
2
-2
1
0
1
-1
0
2
2
-3
1
1
2
-3


-1
3
2
-2
1
0
1
-1
0
2
0
-3
1
1
2
-3


-1
3
2
-2
1
0
1
-1
0
2
0
-3
1
1
2
-3

Plus courts chemins dans le cas général

Les dépendances entre sous-problèmes doivent être acycliques.

Partant d'un graphe avec circuit, il faut trouver un moyen de le rendre acyclique…

Solution : on duplique n fois le graphe (n niveaux). A chaque fois que l'on suit un arc, on va vers un sommet du graphe du niveau suivant.

Cette transformation rend tout graphe acyclique.

Exemple :

Plus courts chemins dans le cas général

Soit $d_k(v)$ le coût d'un plus court chemin de s à v qui utilise au plus k arcs ($0 \leq k \leq n-1$).

$$\begin{cases} d_k(v) = \min\limits_{u \mid (u,v)\in A} \{ d_{k-1}(u) + c(u,v) \} & \text{si } v \neq s \\ d_k(s) = 0 & \end{cases}$$

But : calcul de la valeur $d_{n-1}(v)$ (plus court chemin élémentaire).

Algorithme de Bellman-Ford : algorithme récursif avec mémoïsation correspondant à la relation de récurrence ci-dessus.

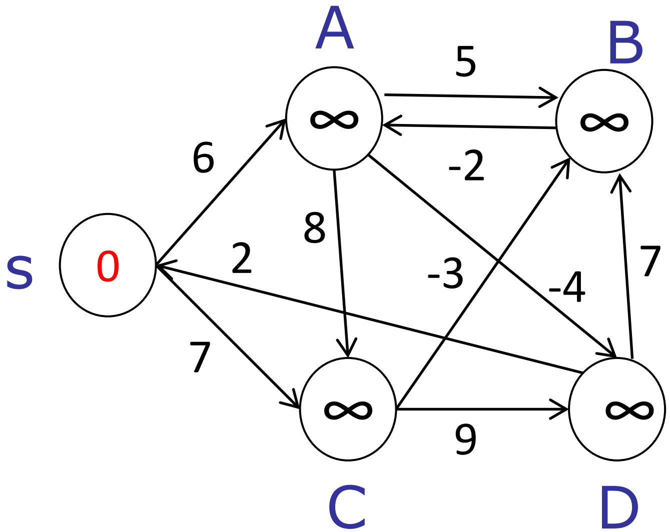
Complexité de l'appel non mémoïsé de $d_k(v)$: $d^-(v) + 1$

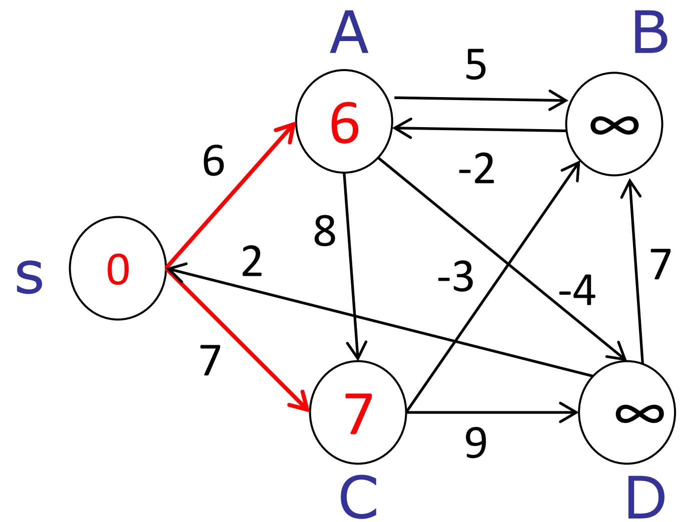
Complexité : $O(n^2 + nm)$

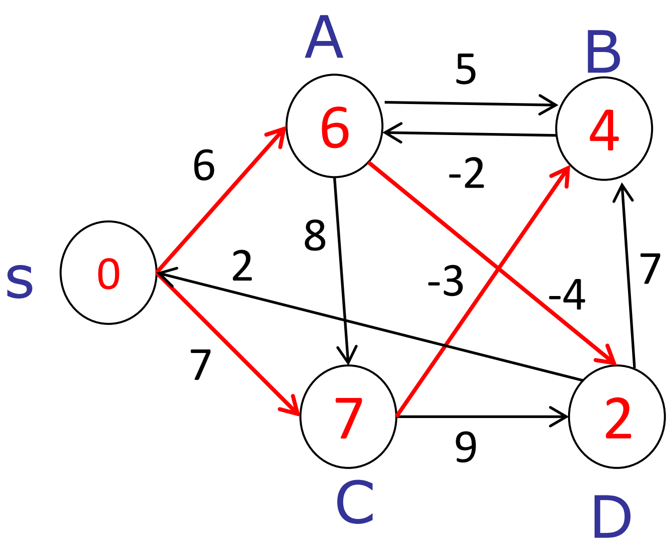
Algorithme de Bellman-Ford

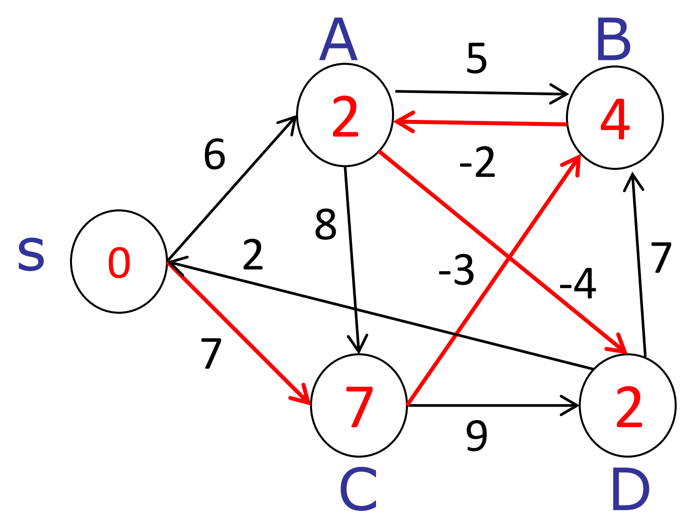
```
// Initialisation
pour tout sommet x faire
    si x=s alors d(x) = 0; pred=null
    sinon d(x) = +∞ ; pred=null

// Boucle principale
pour k = 1 à n-1 faire
      pour chaque arc (x,y) faire
            si d(y) > d(x) + c(x,y)  alors
                        d(y) = d(x) + c(x,y)
                        pred(y) = x

// Détection de circuit absorbant
pour chaque arc (x,y) faire
      si d(y) > d(x) + c(x,y)  alors
            erreur “il n'existe pas de plus court chemin entre s et x”
```

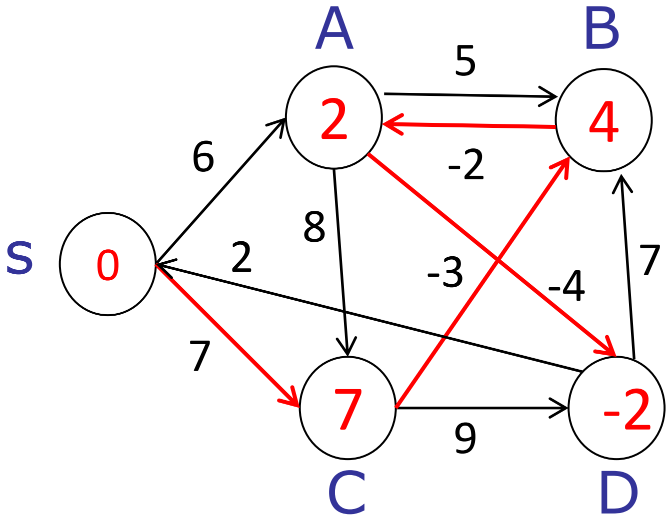
Exemple

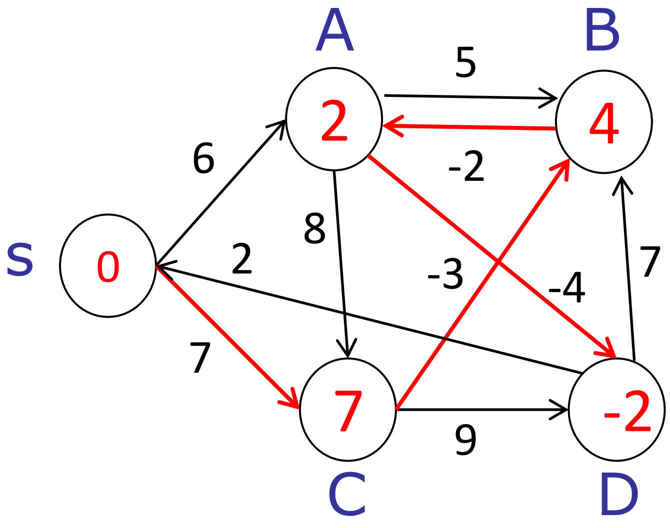

A
B
s
C
D
0
∞
∞
∞
∞
5
-2
6
8
2
-3
-4
7
7
9


A
B
s
C
D
0
6
∞
7
∞
5
-2
6
8
2
-3
-4
7
7
9


A
B
s
C
D
0
6
4
7
2
5
-2
6
8
2
-3
-4
7
7
9


A
B
s
C
D
0
2
4
7
2
5
-2
6
8
2
-3
-4
7
7
9


A
B
s
C
D
0
2
4
7
-2
6
5
-2
8
2
-3
-4
7
7
9


A
B
s
C
D
0
2
4
7
-2
6
5
-2
8
2
-3
-4
7
7
9

Validité

Propriété 1 : A tout moment après l'étape d'initialisation,
si $d(x) \neq \infty$, d(x) est égal à la longueur d'un chemin de s à x.

Preuve (récurrence sur *le nombre i de mises à jour de valeurs d(.)*) :

- Pour i=0, la propriété est vérifiée.
- Supposons que cette propriété soit vérifiée jusqu'à la (i-1)-ème mise à jour, et que la i-ème mise à jour consiste à modifier d(x) après l'examen de l'arc (y,x).

  Par hypothèse de récurrence, d(y) est la longueur d'un chemin µ, de s à y. Ainsi $d(x)= d(y)+c(y,x)$ est la longueur du chemin de s à x qui est µ.(y,x).

Validité

Propriété 2 : (invariant de boucle)

Après i itérations de la boucle "pour" principale : s'il existe un chemin de s à x d'au plus i arcs, alors d(x) est inférieur ou égal à la longueur du plus court chemin ayant au plus i arcs de s à x.

Preuve (récurrence sur *i*)

- Pour i=0, l'invariant est vérifié.
- Supposons qu'il soit vérifié jusqu'au rang i-1

  - Soit μ un plus court chemin (pcc) d'au plus i arcs de s à x. Soit y le dernier sommet avant x sur μ : le chemin de s à y est un pcc de s à y d'au plus (i-1) arcs. Par hyp. d'induction, après (i-1) itérations, d(y) est ≤ à la longueur de ce chemin.

  - A l'itération i, d(x) est comparé à d(y)+c(x,y) et mise à jour à cette valeur si cela diminue d(x). Donc, à la fin de cette itération, d(x) est ≤ à la longueur de μ.

Validité

Propriété 1 : A tout moment après l'étape d'initialisation, si $d(x) \neq \infty$, d(x) est égal à la longueur d'un chemin de s à x.

Propriété 2 :
Après i itérations de la boucle "pour" principale : s'il existe un chemin de s à x d'au plus i arcs, alors d(x) est inférieur ou égal à la longueur d'un plus court chemin ayant au plus i arcs de s à x.

=> à la fin de la boucle principale, pour tout sommet x, s'il existe un chemin de s à x dans G, alors d(x) est égal à la longueur d'un plus court chemin de s à x.

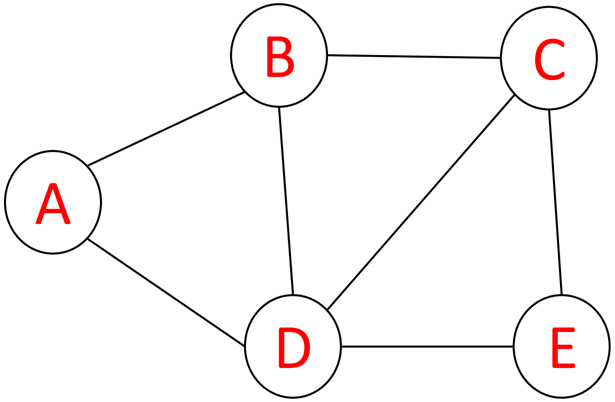
Complexité

```
// Initialisation
// Boucle principale :
pour k = 1 à n-1 faire
     pour chaque arc (x,y) faire
          si d(y) > d(x) + c(x,y)  alors
                    d(y)= d(x) + c(x,y)
                    pred(y)=x
// Détection de circuit absorbant
pour chaque arc (x,y) faire
     si d(y) > d(x) + c(x,y)  alors
          erreur "il n'existe pas de plus court chemin entre s et x"
```

- Complexité : O(nm)
- Remarque : si à l'itération i aucune valeur d(.) n'est modifiée, l'algorithme peut s'arrêter.

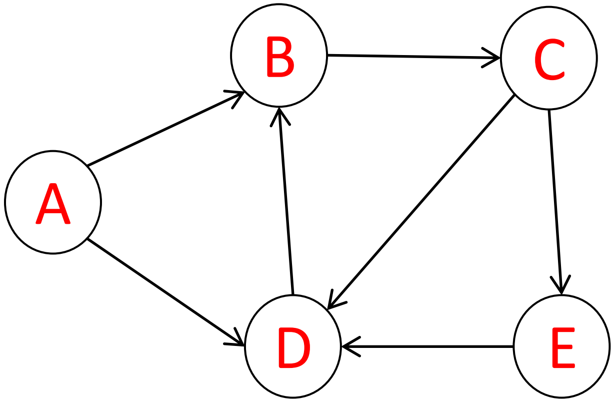
Révisions : Quiz

Dans le graphe suivant, {A,B} est :



A. Un arc.
B. Une arête.
C. Un chemin.

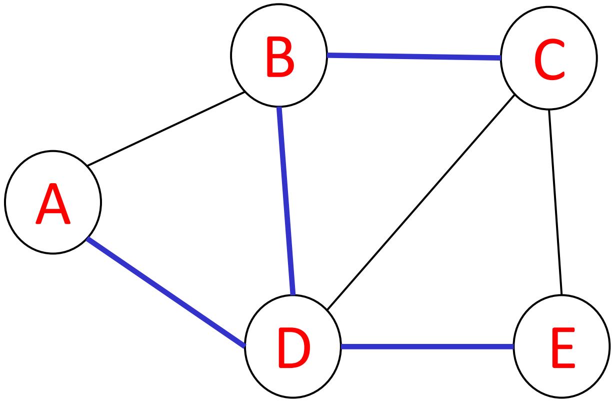
Dans le graphe suivant, {B,C,D,B} est :



A. Un cycle.

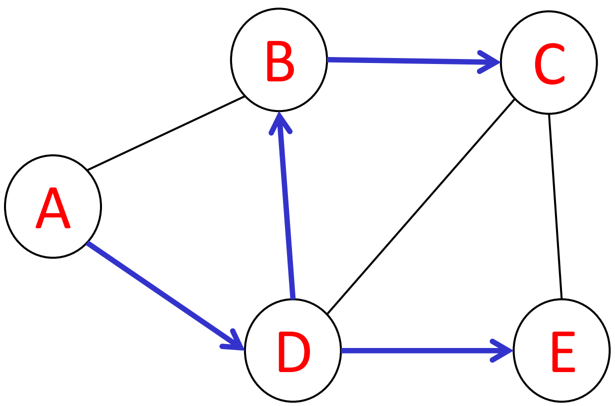
B. Un circuit.

Les arêtes en bleu forment :



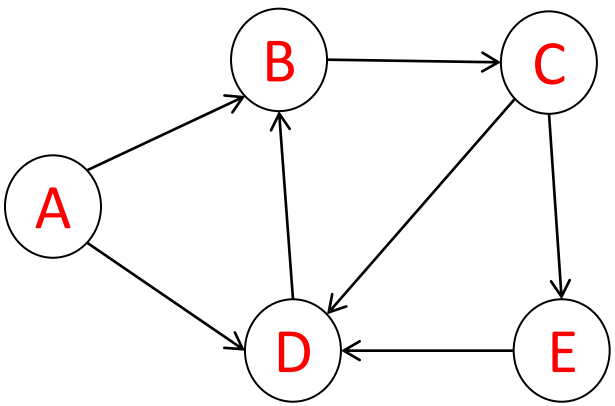
A. Un arbre.
B. Une arborescence de racine A.

Les arcs en bleu forment :



A. Un arbre.
B. Une arborescence de racine A.

Combien de sommets y a-t-il dans le graphe réduit du graphe suivant?



A. 1

B. 2

C. 3

D. 5

Soit G un graphe fortement connexe et soit G_R le graphe réduit de G. Quelle proposition est fausse ?

A. G_R est un graphe sans circuit.
B. G_R contient un seul sommet.
C. Le nombre de sommets de G_R dépend de G.

Soient A et B deux algorithmes résolvant le même problème pour un graphe G non orienté.
A est en $O(n+m)$ et B est en $O(n \log n)$.
Quel algorithme utiliser si G est un graphe complet ?

A. Algorithme A
B. Algorithme B

Soient A et B deux algorithmes résolvant le même problème pour un graphe G non orienté.
A est en $O(n+m)$ et B est en $O(n \log n)$.
Quel algorithme utiliser si G est un arbre ?

A. Algorithme A
B. Algorithme B

Quel algorithme utiliser pour détecter un circuit dans un graphe orienté ?

A. Parcours en largeur
B. Parcours en profondeur
C. Algorithme de Kruskal
D. Algorithme de Dijkstra
E. Algorithme de Bellman Ford

Soit G un graphe connexe, non orienté et valué. Soient u et v deux sommets. Existe-t-il nécessairement une plus courte chaîne entre u et v ?

A. Oui
B. Non

Soit G un graphe connexe, non orienté et valué. Existe-t-il nécessairement un arbre couvrant de coût minimum ?

A. Oui
B. Non

Supposons que l’on ait calculé l’arborescence des plus courts chemins de racine s d’un graphe G valué. Si on modifie le graphe tel que le coût de chaque arête est le double de son coût original. L’arborescence des plus courts chemins reste inchangée (seul le coût des chemins change).

A. Vrai
B. Faux

Quel algorithme utiliser pour trouver une plus courte chaîne dans un graphe où toutes les arêtes ont le même coût (positif) ?

A. Parcours en largeur
B. Algorithme de Dijkstra
C. Algorithme de Bellman
D. Algorithme de Bellman Ford

Quel algorithme utiliser pour trouver une plus courte chaîne dans un graphe sans circuit où les coûts des arêtes sont positifs ?

A. Parcours en largeur
B. Algorithme de Dijkstra
C. Algorithme de Bellman
D. Algorithme de Bellman Ford

Parmi les algorithmes suivants, lequel n'est pas un algorithme glouton ?

A. Algorithme de Prim
B. Algorithme de Djikstra
C. Algorithme de Huffman
D. Algorithme de Bellman Ford
E. Algorithme de Kruskal

Soient A et B deux algorithmes de programmation dynamique basés sur la même relation de récurrence. A est un algorithme récursif avec mémoïsation, et B un algorithme itératif.

A et B ont-ils la même complexité temporelle ?

A. Oui

B. Non

Un groupe de TD comporte n personnes (n est pair). Pour le projet, chacun doit se mettre en binôme avec un autre étudiant. Combien y-a-t-il de façons (f(n)) de former des groupes différents ?

(exemple: si n=4, il faut retourner 3).

A. $f(n) = 2 f(n - 1)$
B. $f(n) = (n - 1) f(n - 2)$
C. $f(n) = f(n - 1) + (n - 1) f(n - 2)$
D. $f(n) = 2^n$