CM 02 - Algorithmes de plus courts chemins (DIJKSTRA et BELLMAN-FORD)

De nombreux problèmes concrets peuvent se modéliser par la recherche de plus courts chemins dans un graphe valué. Par exemple : déterminer l'itinéraire le plus rapide entre deux villes (voiture), entre deux stations (métro), entre deux arrêts (bus) ou bien effectuer le routage dans des réseaux de télécommunication. Enfin, certains problèmes d'ordonnancement font aussi appel à la recherche de *plus longs* chemins.

Définitions.

- Soit G = (S, A) un graphe orienté (ou non) valué par une fonction de poids w : A -> R.

- Le poids d'un chemin/chaîne P = (s_0, s_1, s_2 … s_k) est égal à la somme des poids des arcs/arêtes composant le chemin/chaîne, i.e. w(P) = Sigma pour i de 1 à k des w(s_i-1, s_i)

Existence. Etant donnés deux sommets s et t, trois cas peuvent de présenter :

a. il n'y a pas de chemins/chaînes de s à t
b. il existe des chemins/chaînes de s à t mais pas de plus court(e)
c. il existe un ou plusieurs plus courts chemins/chaînes de s à t

S'il existe un plus court chemin de s à t,
alors le poids de ce chemin/chaîne pcc(s, t) = min {w(P) | P chemin de s à t}

Différentes variantes du problème.

- Origine unique s : trouver le plus court chemin d'un sommet source s à tous les autres sommets
  → algorithme de DIJKSTRA / algorithme de BELLMAN-FORD

- Couple unique (s, t) : trouver un plus court chemin entre deux sommets donnés s et t
  → algorithme A*
  → résoudre le problème "origine unique"

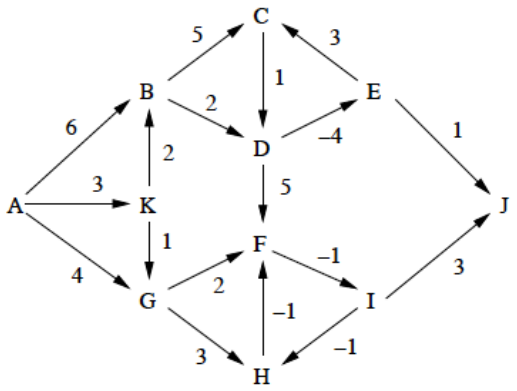
- Tous couples (calcul d'un distancier**) :** trouver les plus courts chemins entre tous les couples de sommets de S → algorithme de FLOYD-WARSHALL

Plan de la présentation

1. Existence d'un ou plusieurs plus courts chemins entre deux sommets
2. Arborescence des plus courts chemins issus d'un sommet source
3. Origine unique : principes communs à DIJKSTRA et BELLMAN-FORD
4. Algorithme de DIJKSTRA
5. Algorithme de BELLMAN-FORD

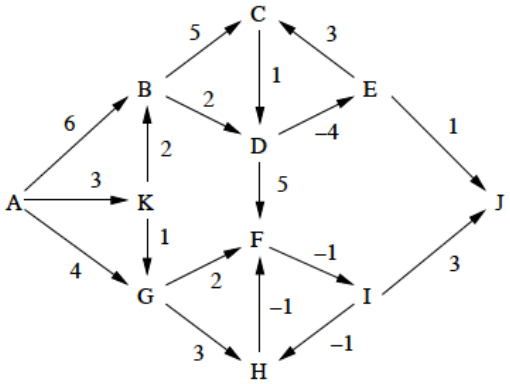
1. Existence d'au moins un plus court chemin entre deux sommets

Exemple #1.



De A à B : il existe un unique plus court chemin (A, K, B).
De A à G : il existe deux plus courts chemins (A, K, G) et (A, G).
De E à A : il n'existe pas de chemins, donc pas de plus courts chemins.

Exemple 2.



De A à E : il existe une infinité de plus courts chemins : (A, K, B, D, E), (A, K, B, D, E, C, D, E), $(A, K, B, D, E, C, D, E, \ldots, C, D, E), \ldots$
De A à J : il existe des chemins mais pas de plus court : les chemins $(A, G, H, F, I, H, F, I, \ldots, H, F, I, J)$ sont arbitrairement courts.

Définition. Un **circuit/cycle absorbant** est un circuit/cycle de poids strictement négatif.

Si un graphe possède un circuit/cycle absorbant, alors il n'existe pas de plus courts chemins/chaînes entre certains de ses sommets.
En effet,
si deux sommets s et t appartiennent à un circuit/cycle de poids strictement négatif, alors w(s, t) n'est pas défini.

Théorème. Soit G un graphe orienté (ou pas) n'ayant pas de circuits/cycles absorbants. Soient s et t deux sommets de G.
S'il existe un chemin allant de s à t, alors il existe (au moins) un plus court chemin de s à t.

2. Arbres des plus courts chemins issus d'un sommet source

On calcule non seulement les poids des plus courts chemins issus d'un sommet source s, mais aussi les sommets présents sur ces plus courts chemins.

Pour représenter ces plus courts chemins, on peut utiliser un tableau (ou une liste) des prédécesseurs :

PRED[s] = nil ;

PRED[s_j] = s_i ssi l'arc (s_j -> s_j) appartient à l'arborescence.

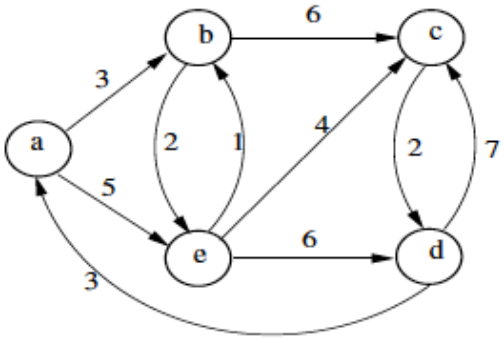
Remarque

pcc(s, t) = pcc(s, PRED[t]) + w(PRED[t], t)

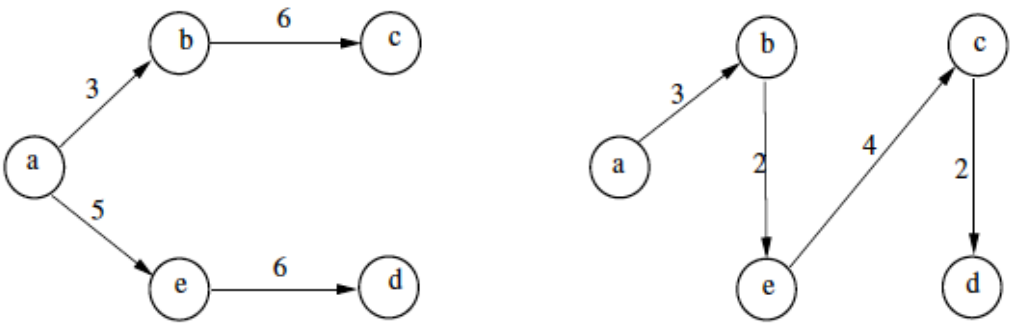
Pour connaitre le plus court chemin entre le sommet source s et un sommet t donné, il faut "remonter" du sommet t jusqu'au sommet s en utilisant l'algorithme suivant :

```
Procédure AfficherPlusCourtChemin(t sommet de G) ;
début
        si t = s
          alors afficher(t)
          sinon
                AfficherPlusCourtChemin(Pred[t]) ;
                Afficher(t)
        fsi
fin
```

L'arbre des plus courts chemins issus d'un sommet source s n'est pas forcément unique, comme le montre le graphe orienté valué ci-dessous en prenant comme sommet source a.



Graphe orienté valué



Deux arbres distincts des plus courts chemins issus du sommet source a

3. Origine unique : principes communs à DIJKSTRA et BELLMAN-FORD

Deux algorithmes permettent de résoudre les problèmes de recherche de plus courts chemins à origine unique :

- l'algorithme de DIJKSTRA lorsque tous les poids sont positifs ou nuls,
- l'algorithme de BELLMAN-FORD lorsque les poids sont positifs, nuls ou négatifs, sous réserve qu'il n'y ait pas de circuit absorbant (de poids strictement négatif).

Ces deux algorithmes procèdent de manière analogue. L'idée est d'associer à chaque sommet s_i de S une valeur DistSource [s_i] qui représente un ***majorant*** du plus court chemin entre s et s_i.

Au départ,

DistSource [s] vaut 0 ;
DistSource [s_i] vaut +inf pour tout sommet s_i ≠ s ;
Le sommet source s ne possède pas de prédécesseur.

L'algorithme diminue alors progressivement les valeurs DistSource[s_i] associées aux différents sommets s_i, jusqu'à ce qu'on ne puisse plus les diminuer, i.e. DistSource[s_i] = pcc(s, s_i).
Pour diminuer les valeurs, on va itérativement examiner chaque arc (s_i -> s_j) du graphe, et regarder si, en passant par s_i, on ne peut diminuer la valeur de DistSource[s_j].

Cette opération de diminution du majorant est appelée "relâchement" de l'arc (s_i -> s_j)

```
procédure Relacher (s_i -> s_j : arc) ;
début
        si DistSource [s_i] + w(s_i, s_j) < DistSource [s_j]
            alors       % mieux vaut passer par s_i pour aller en s_j
                DistSource [s_j] := DistSource [s_i] + w(s_i, s_j) ;
                PRED[s_j] := s_i
        finsi
fin
```

Les algorithmes de DIJKSTRA et BELLMAN-FORD procèdent tous les deux par relâchements successifs d'arcs, mais :

- pour DIJKSTRA, chaque arc est relâché une fois et une seule (***algorithme à fixation d'étiquettes***),
- pour BELLMAN-FORD, chaque arc peut être relâché plusieurs fois (***algorithme à correction d'étiquettes).***

4. Algorithme de DIJKSTRA

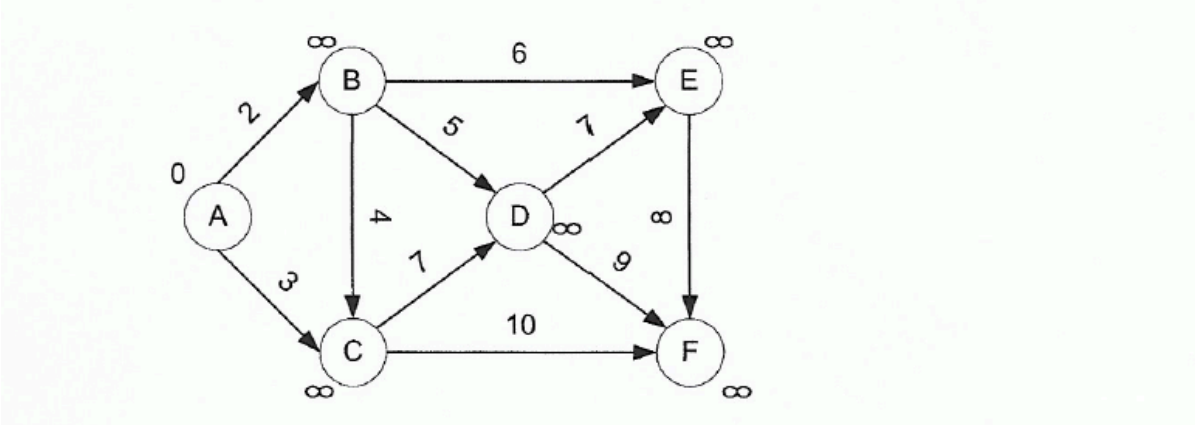
- origine unique : trouver les plus courts chemins d'un sommet s à tous les autres sommets de S
- la fonction de poids w est à valeurs positives (et donc absence de circuit/cycle absorbant)

1. Pseudo-code.

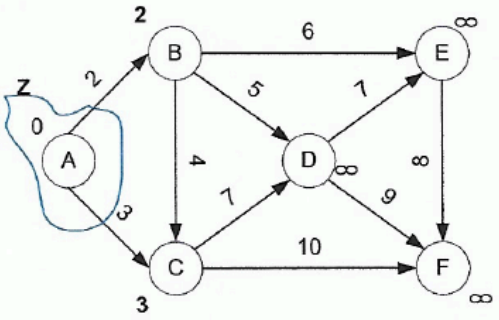
```
Procédure DIJKSTRA (G : graphe valué, s : sommet source) ;
début
DistSource[s] := 0 ;
pour tout sommet u ≠ s faire DistSource[u] := + inf ;
pour tout sommet u faire PRED[u] := nil ;
tantque les sommets ne sont pas tous marqués faire
        soit t0 un sommet non marqué tq DistSource[t0] soit minimale ;
        marquer le sommet t0 ;
        pour tout successeur non marqué t de t0 faire Relacher(t0 -> t) ;
fttque
fin

procédure Relacher (arc u -> v) ;
début
        si DistSource[v] > DistSource[u] + w(u, v)
            alors          % Mieux vaut passer par u pour aller de s en v
                DistSource[v] := DistSource[u] + w(u, v) ;
                PRED[v] := u;
        fsi
fin
```

2. Exemple. Soit **A** le sommet source.

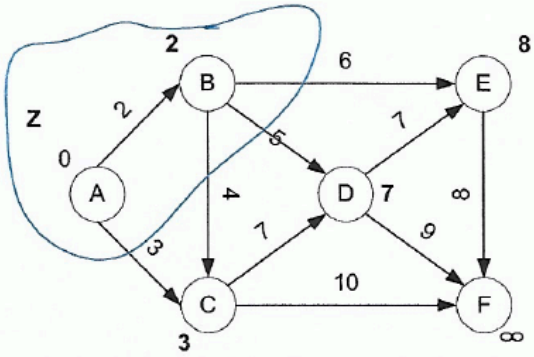


On absorbe A avec une mise à jour de B et C



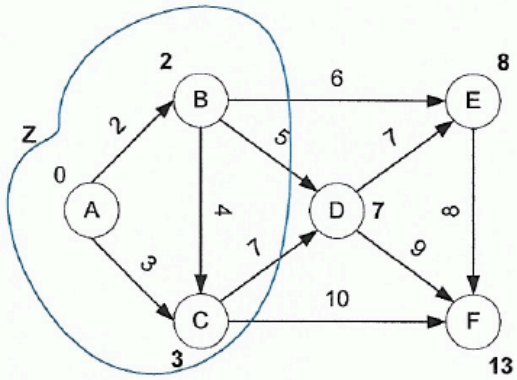
B=0+2, C=0+3

On absorbe B, et on marque C, D, et E



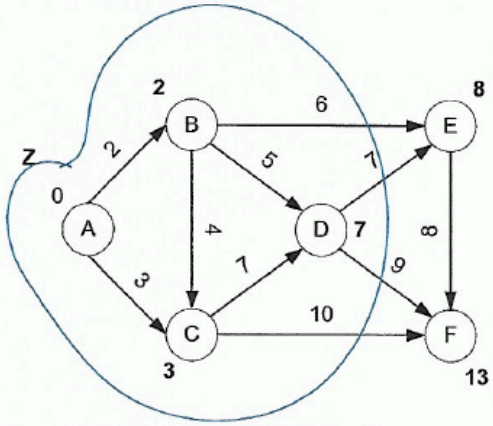
C=min(3, 2+4)=3, D=2+5=7, E=2+6=8

On absorbe C et on marque D et F



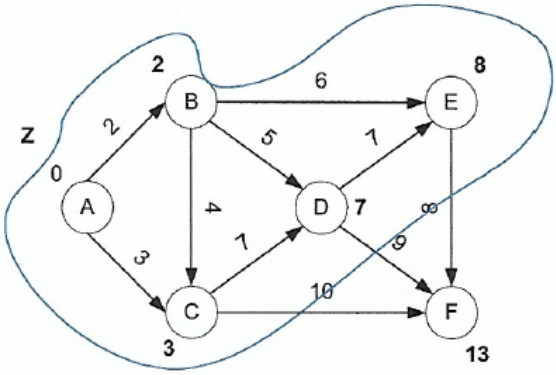
D=min(7,3+7)=7, F=3+10=13

On absorbe D et on marque E et F



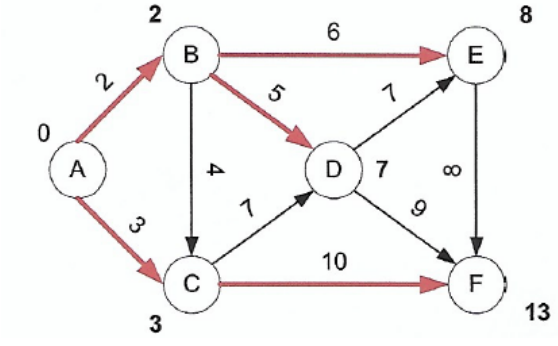
E=min(8,7+7)=8, F=min(13,7+9)=13

On absorbe E et on marque F



F=min(13,8+8)=13

Arbre des plus courts chemins issus de A



5. Algorithme de BELLMAN-FORD

L'algorithme de BELLMAN-FORD permet de trouver les plus courts chemins à origine unique dans le cas où le graphe contient des arcs dont le poids est négatif, sous réserve que le graphe ne contienne pas de circuit absorbant (dans ce cas, l'algorithme détecte l'existence de circuits absorbants).

De manière analogue à DIJKSTRA, on associe à chaque sommet u une valeur DistSource[u] qui représente un majorant du poids du plus court chemin entre s et u. L'algorithme diminue de manière progressive ces valeurs en relâchant les arcs.

Contrairement à DIJKSTRA, chaque arc va être relâché plusieurs fois : on relâche une première fois tous les arcs ; tous les plus courts chemins de longueur 1, issus de s, sont trouvés. On relâche alors une 2nde fois tous les arcs ; tous les plus courts chemins de longueur 2, issus de s, sont trouvés, et ainsi de suite ...

- Si le graphe ne comporte pas de circuit absorbant, un plus court chemin sera de longueur inf. ou égale à (n-1). Donc, au bout de (n-1) passages, on aura trouvé tous les plus courts chemins issus de s.

- Si le graphe possède un circuit absorbant, après (n-1) passages, il restera au moins un arc (u -> v) dont le relâchement permettrait de diminuer la valeur de DistSource[v]. L'algorithme utilise cette propriété pour détecter la présence de circuits absorbants.

Pseudo-code (v1)

```
Procédure BELLMAN-FORD_v1 (G : graphe valué, s : sommet source) ;
début
DistSource[s] := 0 ;
pour tout sommet u ≠ s faire DistSource[u] := + inf ;
pour tout sommet u faire PRED[u] := nil ;
pour k allant de 1 à (n-1) faire
        pour tout arc (u -> v) de A faire Relacher(u -> v) ;
fpour
si existeCycleAbsorbant(G) alors Afficher(« existence d'un circuit absorbant »)
fin

Fonction existeCycleAbsorbant(G : graphe) → Booléen ;
début
        Existe := False ;
        pour tout arc (u -> v) de A faire
                si DistSource [v] > DistSource [u] + w(u, v) alors Existe := True
        fpour
        retourner Existe
fin
```

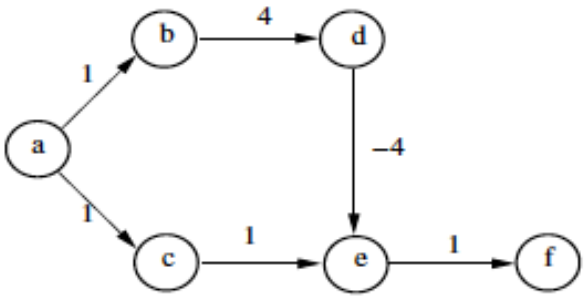
Améliorations possibles.

- ***(v2)*** Mémoriser, lors de chaque itération, l'ensemble des sommets t pour lesquels DistSource[t] a été modifiée, afin de ne relâcher que les arcs partant de ces sommets lors de l'itération suivante.

- ***(v3)*** Arrêter l'algorithme dès qu'aucune valeur DistSource[t] n'a été modifiée lors d'une itération.

Exemple #1 : quid de DIJKSTRA en présence de poids négatifs ?

On considère le graphe orienté valué G ci-dessous dont les sommets sont {a, b, c, d, e, f}.

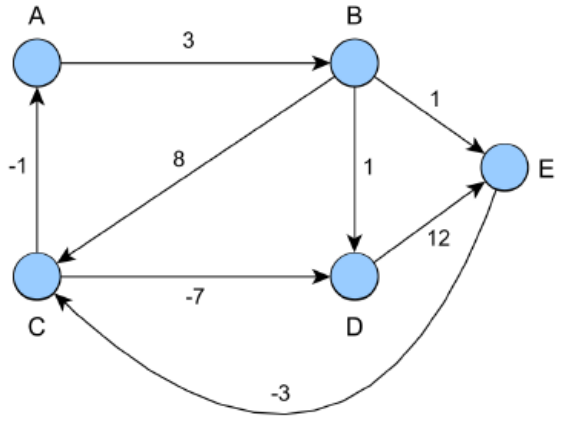


a) Appliquer l'algorithme de DIJKSTRA au graphe G avec comme sommet source a.

b) Quel est le chemin (obtenu par l'algorithme de DIJKSTRA) entre le sommet source a et le sommet f ? Est-il optimal ? Justifier.

Exemple #2 : : BELLMAN-FORD sans circuit absorbant

On considère le graphe orienté G ci-dessous avec A comme sommet source.



(1) On effectue la phase d'initialisation, puis

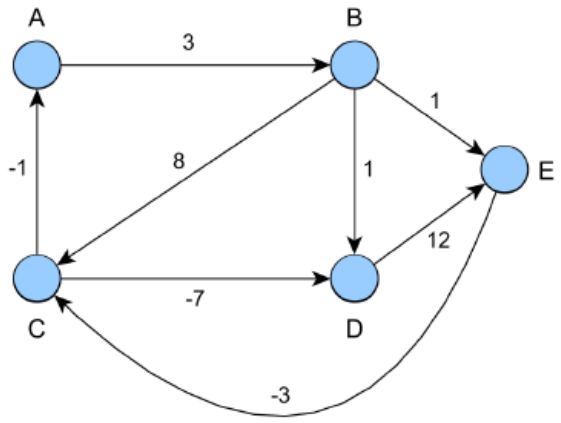
Puis, on envisage les chemins de longueur 1 issus du sommet source A.

DistSource [1..5]

	A	B	C	D	E
init	0	inf	inf	inf	inf
1		3			
2					
3					
4					

PRED [1..5]

	A	B	C	D	E
init	nil	nil	nil	nil	nil
1		A			
2					
3					
4					



(2) On continue avec les chemins de longueur 2 issus du sommet source A. Pour cela, on envisage les successeurs de B, i.e. C, puis D, puis E.

DistSource [1..5]

	A	B	C	D	E
init	0	inf	inf	inf	inf
1		3			
2			**11**	**4**	**4**
3			**1**		
4				**-6**	

PRED [1..5]

	A	B	C	D	E
init	nil	nil	nil	nil	nil
1		A			
2			**B**	**B**	**B**
3			**E**		
4				**C**	

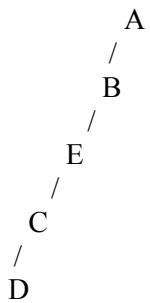
(3) On continue avec les chemins de longueur 3 issus du sommet source A.
Pour cela, on envisage successivement

- les successeurs de C (sommet D) → ABCD poids 4
- puis les successeurs de D (sommet E) → ABDE poids 16
- puis les successeurs de E (sommet C) → ABEC poids 1 (seule amélioration)

(4) On termine avec les chemins de longueur 4 issus de A → ABECD de poids -6 (seule amélioration)

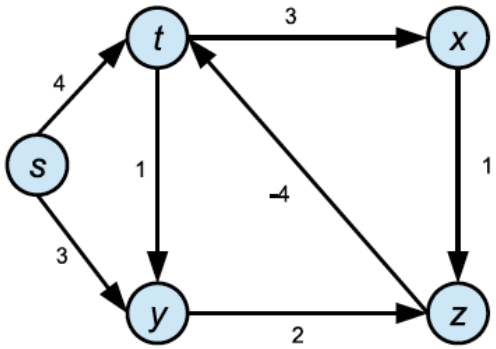
Il existe toujours un PCC entre 2 sommets car le graphe ne possède aucun circuit absorbant. Donc, les PCC issus de la source forment un arbre.

Ici, il se trouve qu'ici l'arbre est un peu « particulier » car chaque nœud (sauf la racine A) possède un unique fils.



Exemple #3 : BELLMAN-FORD en présence d'un circuit absorbant

On considère le graphe orienté valué suivant :



1. Montrer que ce graphe possède un circuit absorbant

2. Appliquer l'algorithme de BELLMAN-FORD sur ce graphe

Contrairement à DIJKSTRA, chaque arc va être relâché plusieurs fois : on relâche une première fois tous les arcs ; tous les plus courts chemins de longueur 1, issus de s, sont trouvés.

On relâche alors une 2nde fois tous les arcs ; tous les plus courts chemins de longueur 2, issus de s, sont trouvés, et ainsi de suite .../...

3. Ce graphe possède-t-il un arbre de PCC ?

Eléments de solution

(1) Iter=0 représente la phase d'initialisation.

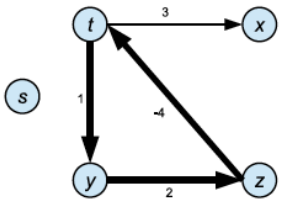
(2) Evolution des 2 tableaux au fur et à mesure des 4 (5-1) itérations

Iter	s	t	y	x	z
0	0	+∞	+∞	+∞	+∞
1	0	4	3	+∞	+∞
2	0	4	3	7	5
3	0	1	3	7	5
4	0	1	2	4	5

DistSource[u]

s	t	y	x	z
-1	z	t	t	y

PRED[u]



Circuit absorbant (poids -1)