REMARQUE PRELIMINAIRE SUR LE NIVEAU 4

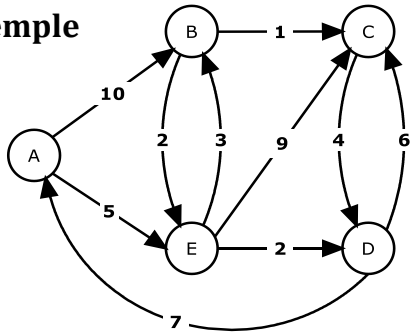
Dans ce niveau, nous abordons un certain nombre d'algorithmes d'optimisation de chemins. L'objectif est de montrer leur diversité et de bien comprendre qu'il y a toujours plusieurs façons d'aborder un problème algorithmique.
Certains de ces algorithmes sont des parcours de graphes assez classiques (Dijkstra, Bellman Ford Kalaba), d'autres sont des approches plus originales (Bellman, Floyd-Warshall).
Le programme est chargé, mais l'évaluation ne portera que sur les méthodes itératives et non sur le pseudo-code ou la programmation Python qui, elle, sera abordée dans la **SAE S2.02 - Exploration algorithmique d'un problème.**

OPTIMISATION DE CHEMINS

NOTION DE GRAPHE VALUE

Un graphe **valué** $G = (S, A, \boldsymbol{f})$ est un graphe $G = (S, A)$ (orienté ou non) muni d'une application $f : A \to \mathbb{R}$.
Cette application permet d'associer des valeurs (ou valuations) à chaque arc du graphe.

Exemple



- $f(A, B) = 10$
- $f(A, E) = 5$
- $f(B, C) = 1$ …

La recherche de **chemins de poids* minimum ou maximum** entre deux sommets (I : sommet initial, F : sommet final) est un problème associé à un graphe valué dans lequel le réel $f(x, y)$ associé à chaque arc (x, y) représentera son **poids**.

**poids : de façon générale on utilisera le terme de « poids », mais on pourra aussi utiliser les termes de « distance », « coût », « gain » selon le contexte et la signification de la valuation $f(x, y)$. On évitera le terme de « longueur » qui est normalement utilisé pour le nombre d'arcs d'un chemin et non la somme de ses valuations.*

Le **poids d'un chemin** $\boldsymbol{C} = (\boldsymbol{x_1}, \ldots, \boldsymbol{x_p})$ dans un graphe valué $G = (S, A, f)$ est bien sûr la somme des poids des arcs constituant le chemin.

$$\textbf{poids}_{\mathbf{f}}(\boldsymbol{C}) = \sum_{i=0}^{p-1} f(x_i, x_{i+1})$$

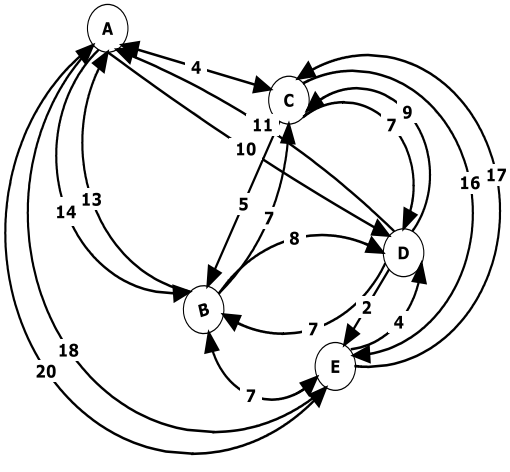
UN PROBLEME MOINS SIMPLE QU'IL N'Y PARAIT…

EXERCICE 1 : NOMBRE DE CHEMINS POUR ALLER D'UN SOMMET VERS UN AUTRE (CORRIGE P12)

Un petit problème de dénombrement…

Dans un graphe orienté sans boucle, combien a-t-on de chemins sans circuit (les circuits ne pouvant que rallonger les chemins) allant d'un sommet A vers un sommet B ?

On pourra commencer dénombrer ces chemins sur un exemple, comme le graphe ci-contre :



BILAN - ASPECT NUMERIQUE

Nombre de chemins allant d'un sommet vers un autre dans un graphe complet, en fonction de l'ordre n du graphe :

- $n = 5 \rightarrow \boldsymbol{Nb_{Chemins} = 16}$
- $n = 10 \rightarrow \boldsymbol{Nb_{Chemins} = 109601}$
- $n = 15 \rightarrow \boldsymbol{Nb_{Chemins} \approx}$ **17 Millions**
- $n = 20 \rightarrow \boldsymbol{Nb_{Chemins} \approx}$ **17 Millions de milliards**

Si, pour chercher un chemin minimum, on devait parcourir tous les chemins possibles, en supposant que le traitement à chaque étape soit de $1\mu s$, il faudrait :

- 4h42 pour un graphe d'ordre 15 (=15 sommets)
- 552 ans pour un graphe d'ordre 20
- Un peu plus de 2 milliards d'années pour un graphe de 25 sommets (rappel : histoire de la terre ≈ 4,5 milliards d'années)

Et si la durée d'un traitement était de 1ns (nanoseconde, $10^{-9}s$), il faudrait :

- 6 mois ½ pour un graphe d'ordre 20
- 2,2 millions d'années pour un graphe d'ordre 25 (Premiers hommes ≈ 2,8 millions d'années)

CONCLUSION

Pour chercher un chemin particulier dans un graphe, de poids minimum par exemple, nous ne pouvons pas envisager de parcourir tous les chemins possibles pour choisir le meilleur, il nous faut adopter des stratégies de algorithmiques qui nous permettent d'ignorer les chemins inutiles.
Dans ce cours, nous abordons quatre algorithmes de recherche de chemins de poids minimum :

- Dijkstra
- Bellman Ford Kalaba
- Bellman
- Floyd Warshall

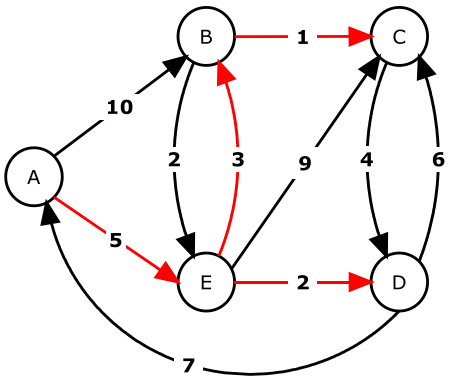
I. ALGORITHME DE DIJKSTRA : UNE PREMIERE STRATEGIE POUR LE CHEMIN DE POIDS MINIMUM, EXPLORATION A PARTIR DU « MEILLEUR »

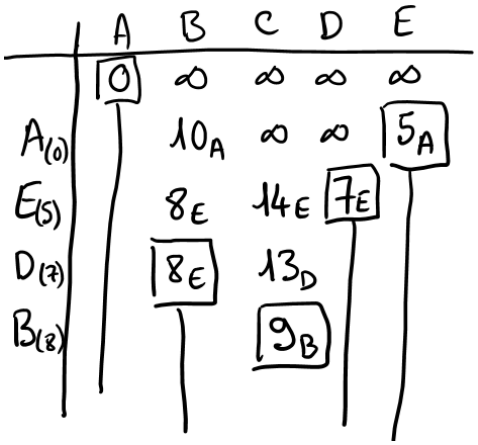
INTRODUCTION

PRINCIPE

- On reprend l'idée d'un parcours de graphe dans la mesure où à chaque étape on regarde les successeurs d'un sommet (sommet traité),
- À l'étape k, le sommet traité a été vu à une distance au plus k du sommet de départ, on est donc plus proche d'un parcours en largeur que d'un parcours en profondeur.
- Par contre, le choix du sommet à traiter ne se fait pas par une file, à chaque étape on va choisir le sommet correspondant au plus cours chemin parcouru jusque-là.

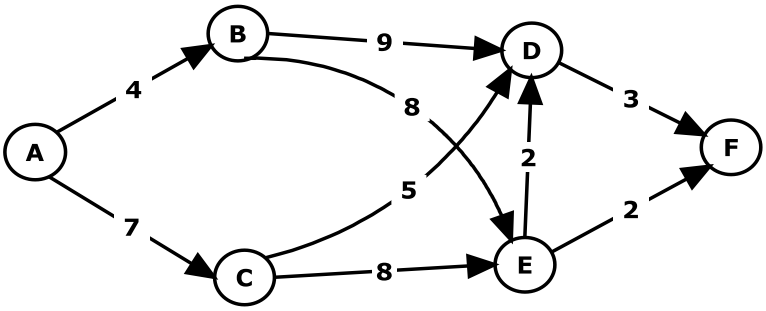
EXEMPLE





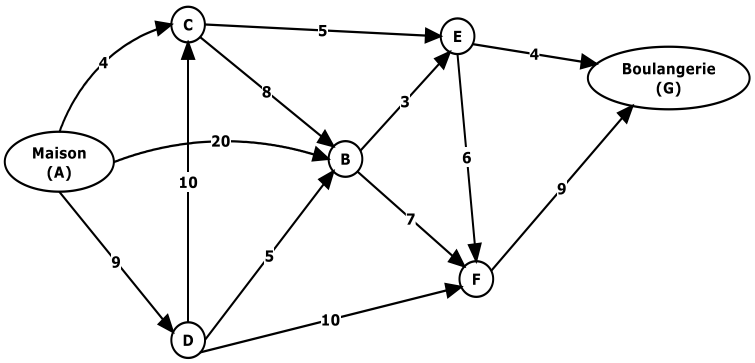
EXERCICE 2 (CORRIGE P13)

Appliquer l'algorithme de Dijkstra pour trouver les chemins minimums en partant du sommet A vers les autres sommets



EXERCICE 3 (CORRIGE P13)

Appliquer l'algorithme de Dijkstra pour trouver le chemin minimum pour aller de la maison à la boulangerie



PSEUDO-CODE ET PYTHON

PREPARATION A LA DESCRIPTION ALGORITHMIQUE

On pourra implémenter le graphe pas une matrice des poids, dans laquelle une absence d'arc se traduit par une valeur infinie.

Par exemple pour le graphe ci-dessus : $Poids = \begin{pmatrix} \infty & \infty & 10 & 8 & \infty \\ \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & 5 \\ \infty & 9 & 1 & \infty & \infty \\ \infty & 1 & \infty & 2 & \infty \end{pmatrix}$

Comme vu dans les exemples précédent, l'algorithme de Dijkstra permet de remplir deux tableaux :

- *Pred* Le tableau des prédécesseurs permettant de reconstituer les chemins minimums
- *Dist,* le tableau des distances minimales pour l'accès à chaque sommet.

***Pred* et *Dist* sont bien sûr des tableaux de dimension n (on utilisera des listes sous Python)**

- On constate également, qu'il n'est pas possible d'utiliser une file pour choisir les sommets à traiter, on utilisera une « liste*» (qui peut être un tableau, un vecteur, une liste… selon le langage utilisé) en précisant à chaque étape quel est le sommet extrait de la liste. On utilisera donc les primitives suivante : *Ajouter(L,s)* et *Enlever(L,s)*

  ** des structures de données plus avancées comme les files de priorité avec les tas binaires, permettent une meilleure implémentation de l'algorithme*

On peut par exemple définir une fonction : **extract_min(L,dist)** qui extrait de **L** le sommet de L ayant la distance minimum et le retourne. Enfin, on suppose également que l'on est capable, par une fonction par exemple, de parcourir tous les successeurs d'un sommet.

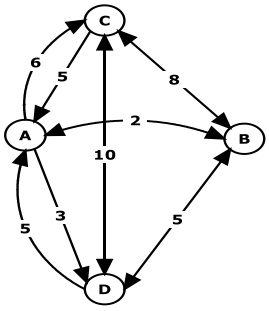
PSEUDO CODE

Initialisation
n ← ordre(Poids)
Pour touts sommet s de S faire
    $Ajouter(L, s)$
    $Dist[s] \leftarrow \infty$
    $Pred[s] \leftarrow Null$
Fin Pour
$s \leftarrow s0$ $(sommet\ de\ départ)$
$Dist[s] = 0$

Traitement (à compléter)
Tant que L est non vide faire
    $s \leftarrow$ **extract_min(L,dist)** (on pourrait dire aussi : $s \leftarrow$ le sommet s de L tel que *Dist[s]* est minimum)

TP PYTHON (FACULTATIF MAIS UTILE POUR LA SAE)

1. **lst** étant une liste, créer une fonction **extract_min(lst)** qui renvoie, non pas la plus petite valeur de la liste **lst**, mais l'indice (sommet) correspondant.
2. **Dans l'algorithme de Dijkstra, les données de départ sont un sommet *s* et une liste de liste définissant la matrice des poids d'un graphe. Par exemple, pour le graphe ci-contre :**




```
poids=[[float('inf'),2,6,3],
   [2,float('inf'),8,5],
   [5,8,float('inf'),10],
   [5,5,10,float('inf')]]
```

   Si A est le sommet de départ, nous prendrons **s=0**
   Les sommets étant caractérisés par leur numérotation en commençant par 0…

   Initialisation de l'algorithme
   En entrée : nous disposons d'une liste de liste **poids** décrivant le graphe pondéré et d'un sommet **s (numéro de 0 à n-1)** pour le sommet de départ.
   Trois listes vont être utilsées : une liste **dist,** une liste **pred,** une liste **a_traiter** de sommets restant à traiter
   a. Initialiser les listes **dist, pred** et **a_traiter** en utilisant bien sûr les dimensions de **poids**
   b. Autre variable, **som**, le sommet courant, initialement égal au sommet de départ **s**.
   Modifier le contenu de **dist, pred** et **a_traiter** sachant que **som** est le sommet de départ.

3. **Processus itératif : on répète le même traitement tant qu'il y a des sommets à traiter**
   a. Pour choisir, à chaque étape le sommet que l'on va traiter (exploration à partir du « meilleur ») on utilise une fonction du type **extrac_min** mais qu'il faut ici modifier.
   Nous devrons notamment ajouter un paramètre en entrée (liste de sommets à traiter ?)
   Précisez.
   b. Écrire sous Python la boucle permettant de compléter les vecteurs **pred** et **dist**.

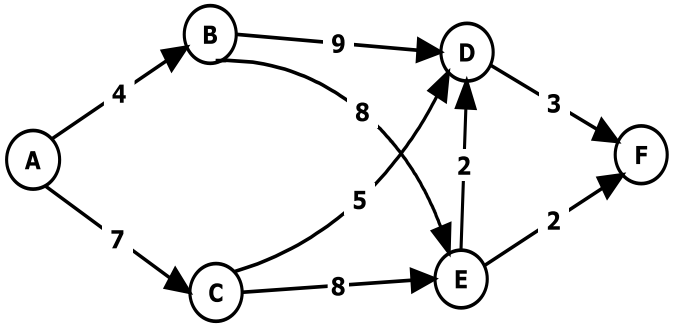
4. Sous Python, écrire une fonction **Dijkstra(poids, som_dep)** permettant de déterminer, dans le graphe pondéré décrit par la liste de liste **poids,** les chemins de longueur minimum du sommet **som_dep** vers tous les autres sommets.

II. BELLMAN-FORD-KALABA : CHEMINS DE POIDS MINIMUM OU MAXIMUM, DETECTION DE CIRCUITS « ABSORBANTS »

L'algorithme de Bellman-Ford-Kalaba fonctionne sur tout type de graphes, à valuations positives comme négatives. Il peut être aussi adapté à une recherche de chemins de poids maximums.
Son temps de calcul est un peu plus long que Dijkstra car à chaque étape, il est possible de revenir sur un calcul précédent : l'algorithme n'est pas glouton.

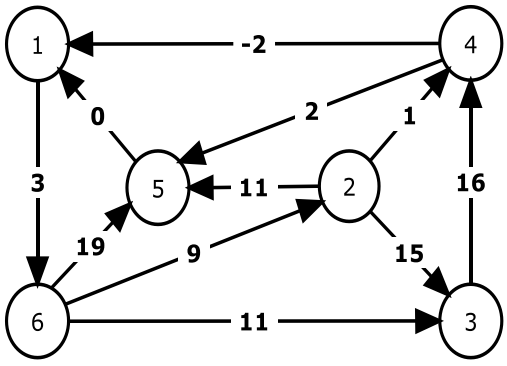
Son principe est assez simple, ce qui en fait son intérêt aussi.
L'itération se fait sur la **longueur des chemins**, c'est-à-dire leur nombre d'arcs. L'idée n'est pas cependant de regarder tous les chemins de longueur 1, 2,… mais seuls ceux susceptibles d'apporter une amélioration.

La vidéo ci-dessous, présente l'algorithme
En traitant l'exemple ci-contre :



Chemins de longueur ≤ k	Sommet S	A	B	C	D	E	F
k=0		0	∞	∞	∞	∞	∞
k=1	$A_{(0)}$	0	4_A	7_A	∞	∞	∞
k=2	$B_{(4)}$	0	4_A	7_A	13_B	12_B	∞
	$C_{(7)}$	0	4_A	7_A	12_C	12_B	
k=3	$D_{(12)}$	0	4_A	7_A	13_B	12_B	15_D
	$E_{(12)}$	0	4_A	7_A	12_C	12_B	14_E
k=4	$F_{(14)}$	0	4_A	7_A	12_C	12_B	14_E

AUTRE EXEMPLE

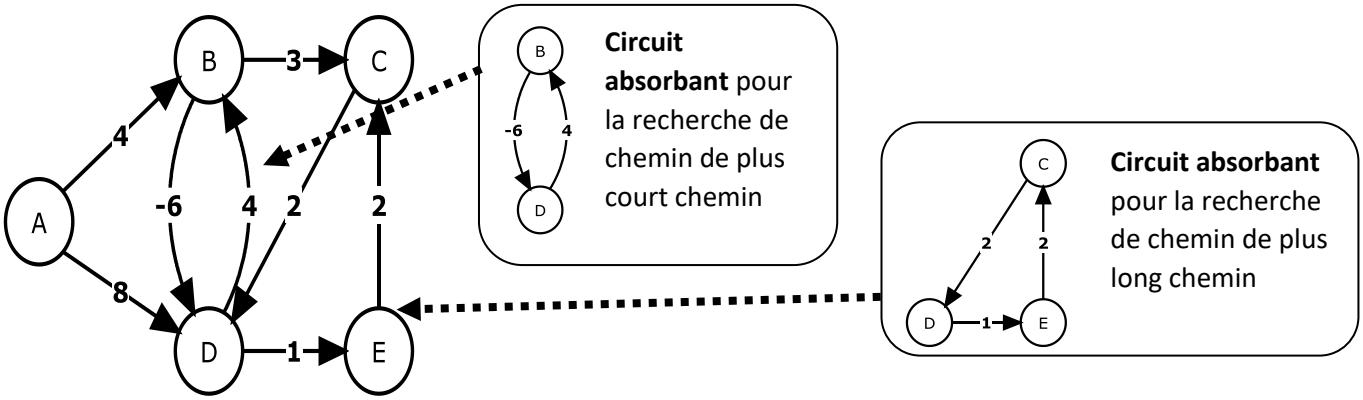


DIST							
Chemin de longueur ≤k	S	1	2	3	4	5	6
k=0		∞	0	∞	∞	∞	∞
k=1	2	∞	0	15_2	1_2	11_2	∞
k=2	3	∞	0	15_2	1_2	11_2	∞
	4	-1_4	0	15_2	1_2	3_4	∞
	5	-1_4	0	15_2	1_2	3_4	∞
k=3	1	-1_4	0	15_2	1_2	3_4	2_1
	5	-1_4	0	15_2	1_2	3_4	2_1
k=4	6	-1_4	0	13_6	1_2	3_4	2_1
k=5	3	-1_4	0	13_6	1_2	3_4	2_1

ARRET

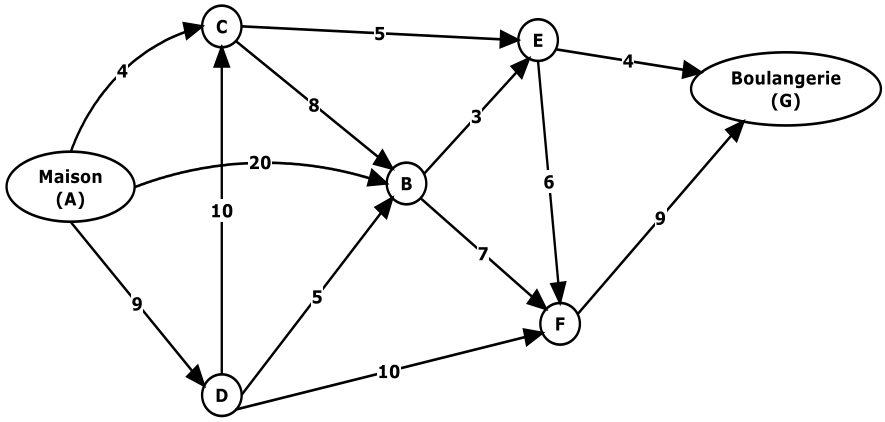
L'algorithme recherche à chaque étape les chemins de poids minimum de longueurs (nombre d'arcs) inférieures à $1,2 \ldots n-1$ à s'arrête lorsqu'il n'y a pas eu de modification entre l'étape k et l'étape $k+1$ ou lorsque $k = n-1$. En effet, nous savons qu'à partir de la longueur n, les chemins contiennent des circuits.

Si nous continuons à l'étape n et si celle-ci donne un résultat différent de l'étape $n-1$ cela signifie que le graphe présente un **CIRCUIT ABSORBANT, C'EST-A-DIRE DE POIDS NEGATIF,** et que le problème n'a pas de solution (l'algorithme détecte un circuit absorbant et l'arrêt à $k = n$ lui évite de boucler indéfiniment).



EXERCICE 4 (CORRIGE P14)

Appliquer l'algorithme de Bellman Ford Kalaba pour déterminer les chemins de poids minimum et maximum pour aller de la maison à la Boulangerie :



III. BELLMAN : CHEMINS DE POIDS MINIMUM OU MAXIMUM, DETECTION DE CIRCUITS « ABSORBANTS »

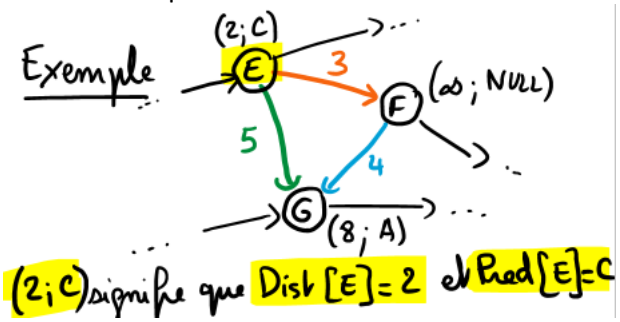
L'algorithme de Bellman repose sur la notion de **relâchement d'un arc** que nous avons déjà utilisé, sans le nommer ainsi, dans les algorithmes précédents.
Dans ces algorithmes, relâcher un arc (s, s') consiste à regarder si en passant par l'arc (s, s') nous obtenons un accès au sommet s' de poids inférieur à celui que nous avions jusque-là, et le cas échéant à définir un nouveau chemin minimum en changeant $dist[s]$ et $pred[s]$.

Exemple dans l'algorithme de Dijkstra :

Relâchement de l'arc (s,s')

Traitement
Tant que L est non vide faire
  $s \leftarrow$ **extract_min(L,dist)** (on pourrait dire aussi : $s \leftarrow$ le sommet s de L tel que *Dist[s]* est minimum)
  Pour tout successeur s' de s dans L faire
    Si $Poids[s, s'] + Dist[s] < dist[s']$ alors
      $Dist[s'] = Poids[Som, s'] + Dist[s]$
      $Pred[s'] \leftarrow s$
    Fin Si
  Fin Pour
Fin Tant que

Plus concrètement, considérons l'extrait du graphe ci-dessous, dans lequel un couple $(\boldsymbol{dist}[\boldsymbol{s}], \boldsymbol{pred}[\boldsymbol{s}])$ est associé à chaque sommet $\boldsymbol{s}$.



Relâchement de (E,G)
- Dist[E] + Poids(E,G) = 2 + 5 = 7] 7 < 8
- Dist[G] = 8

⇓
{ Dist[G] ← 7
{ Pred[G] ← E

Relâchement de (F,G)
- Dist[F] + Poids(F,G) = ∞ + 4 = ∞] ∞ > 8
- Dist[G] = 8

⇒ Aucun changement

Relâchement de (E,F)
- Dist[E] + Poids(E,F) = 2 + 3 = 5] 5 < +∞ ⇒ { Dist[F] ← 5 ; { Pred[F] ← E
- Dist[F] = +∞

ALGORITHME DE BELLMAN

Pour trouver les chemins de poids minimum, l'algorithme de Bellman propose **« relâcher » tous les arcs du graphe n-1 fois.**

APPLICATION SUR UN EXEMPLE

JUSTIFICATION DE L'ALGORITHME

Quel que soit l'ordre dans lequel les arcs sont relâchés, on est sûr que :

- A la première étape : tous les **premiers** arcs des plus courts chemins ont été relâchés
- A la deuxième étape : tous les **deuxièmes** arcs des plus cours chemins ont été relâchés
- …
- A la n-1 ème étape tous les n-1èmes arcs (s'ils existent) des plus courts chemins ont été relâchés.

Les chemins de poids minimum étant de longueur inférieure ou égale à n-1, on est donc certain d'avoir déterminé ces chemins à la fin de la n-1ème étape.

On peut bien sûr utiliser cet algorithme pour détecter des circuits absorbants, il suffit pour cela de pousser jusqu'à l'étape n et d'observer, ou non, une variation entre les étapes n et n-1

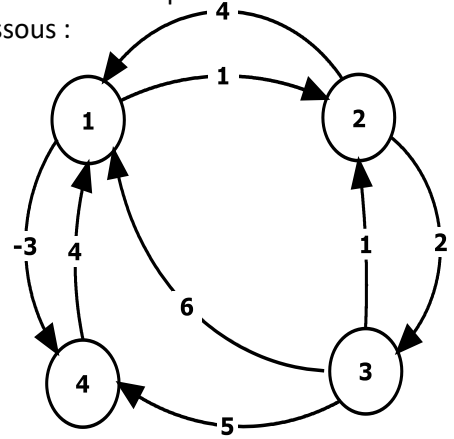
PSEUDO-CODE

Initialisation
n ← ordre(Poids)
Pour tout sommet s de S faire
    $Dist[s] \leftarrow \infty$
    $Pred[s] \leftarrow Null$
Fin Pour
$s \leftarrow s0\ (sommet\ de\ départ)$
$Dist[s] = 0$

Traitement
Pour $i = 1$ jusqu'à $n - 1$
    Pour tout arc (s, s') de G faire
        $Relâchement(s, s')$
    Fin Pour
Fin Pour

EXERCICE 5 (CORRIGE P15)

Appliquer l'algorithme de Bellman pour la recherche de chemins de poids minimums partant du sommet 2 dans le graphe ci-dessous :



BILAN INTERMEDIAIRE – INTRODUCTION A FLOYD WARSHALL

Les trois algorithmes précédents permettent d'obtenir tous les chemins minimums en partant d'un sommet du graphe :

- **Dijkstra** ne fonctionne que sur des graphes à valuations **positives**.
- **Bellman et Bellman-Ford-Kalaba** : fonctionnent sur des graphes à valuations **positives ou négatives**, ils peuvent aussi être adaptés à des recherches de chemins maximum.

L'algorithme Floyd-Warshall permet de calculer des chemins de **poids minimum** en partir **de tous les sommets** d'un graphe aux valuations **positives ou négatives.**

IV. FLOYD-WARSHALL : CHEMINS MINIMUMS DE TOUS LES SOMMETS VERS TOUS LES SOMMETS

EXEMPLE DE RESULTAT

	Calgary	Edmonton	Halifax	Montréal	Québec	Toronto	Vancouver
Calgary							
Edmonton	290						
Halifax	5200	5250					
Montréal	3500	3600	1330				
Québec	3780	3830	1050	280			
Toronto	3400	3470	1790	560	800		
Vancouver	970	1150	6200	4900	5180	4380	

Toutes les distances ont été calculées, de tous les sommets vers tous les autres sommets

PRINCIPE

L'originalité de cet algorithme est d'étudier, de façon itérative, les chemins de poids minimum passant par le 1er sommet, les 2 premiers, les 3 premiers…
Comme nous cherchons tous les chemins partant de **tous** les sommets, le résultat attendu n'est pas un vecteur ***dist*** et un vecteur ***pred*** comme pour les algorithmes précédents, mais ***n vecteurs dist*** et ***n vecteurs pred***, ce qui peut se représenter dans des matrices dont chaque ligne correspond à un sommet de départ.
La détermination de ces matrices se fera de façon itérative par le calcul de matrices intermédiaires :

- $M_0, M_1, \dots, M_n$ pour les poids
- $P_0, P_1, \dots, P_n$ pour les prédécesseurs

NOTATIONS

- $\boldsymbol{M_k}$: matrice des poids des chemins minimums passant par des sommets numérotés de 1 à k.
  $\boldsymbol{M_n}$ sera donc la matrice des poids minimums attendue
  Conventions :
  - 0 dans la diagonale, qu'il y ait une boucle ou non sur le sommet, on estime que le poids pour aller d'un sommet à lui-même est 0.
  - ∞ si on n'a pas encore trouvé de chemin reliant les sommets considérés
- $\boldsymbol{P_k}$: matrice des prédécesseurs dans les chemins minimums passant par des sommets numérotés de 1 à k.
  $\boldsymbol{P_n}$ sera donc la matrice des prédécesseurs des chemins minimums de $\boldsymbol{M_n}$
  Convention : $N\ ou\ Null$ dans la diagonale ou si on n'a pas encore trouvé de chemin reliant les sommets considérés

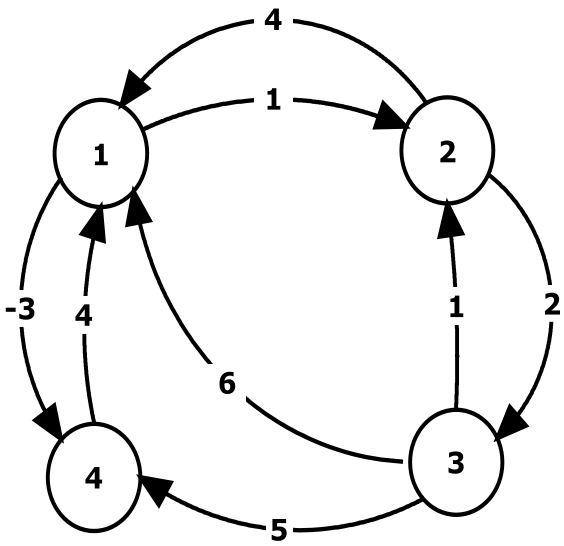
EXEMPLE

$$M_2 = \begin{pmatrix} 0 & 1 & 3 & -4 \\ 4 & 0 & 2 & \infty \\ 6 & 1 & 0 & 5 \\ 4 & \infty & 1 & 0 \end{pmatrix}$$

$$\boldsymbol{P_2} = \begin{pmatrix} \boldsymbol{N} & \boldsymbol{1} & \boldsymbol{2} & \boldsymbol{1} \\ \boldsymbol{2} & \boldsymbol{N} & \boldsymbol{2} & \boldsymbol{N} \\ \boldsymbol{3} & \boldsymbol{3} & \boldsymbol{N} & \boldsymbol{3} \\ \boldsymbol{4} & \boldsymbol{4} & \boldsymbol{2} & \boldsymbol{N} \end{pmatrix}$$

- $M_2(2,4) = \infty \Rightarrow$ il n'y a pas d'arc direct ou de chemin passant par 1 ou 2 allant de 2 vers 4
- $M_2(4,3) = 1 \Rightarrow$ le poids minimum des chemins allant de 4 vers 3 et passant par les sommets 1 et/ou 2 est de 1.
- $P_2(2,3) = 2 \Rightarrow$ le prédécesseur de 3 dans le chemin minimum (arc ou passant par 1 ou 2) allant de 2 vers 3 est 2 : ce chemin est donc l'arc (2,3) qui a un poids de -3 ($M_2(2,3) = -3$)
- $P_2(4,3) = 2\ et\ P_2(4,2) = 4 \Rightarrow$ dans le chemin de poids minimum 2 allant de 4 vers 3, le prédécesseur de 3 est 2 et le prédécesseur de 2 est 4 : le chemin est donc (4,2,3)

EXEMPLE



INITIALISATION

Matrice des poids : $M_0 = \begin{pmatrix} 0 & 1 & \infty & -3 \\ 4 & 0 & 2 & \infty \\ 6 & 1 & 0 & 5 \\ 4 & \infty & \infty & 0 \end{pmatrix}$

Matrice des prédécesseurs : $P_0 = \begin{pmatrix} N & 1 & N & 1 \\ 2 & N & 2 & N \\ 3 & 3 & N & 3 \\ 4 & N & N & N \end{pmatrix}$

ETAPE 1

$M_0 = \begin{pmatrix} 0 & 1 & \infty & -3 \\ 4 & 0 & 2 & \infty \\ 6 & 1 & 0 & 5 \\ 4 & \infty & \infty & 0 \end{pmatrix}$

COLONNE 1 : chemin allant vers 1
- (2,1) de poids 4
- (3,1) de poids 6
- (4,1) de poids 4

LIGNE 1 : chemin partant de 1
- (1,2) de poids 1
- (1,4) de poids -4

3 x 2 chemins passant par 1 :
- (2,1,2) : ne nous intéresse pas (circuit de 2 vers 2)
- (2,1,4) de poids $\mathbf{4-3=1}$ **mieux que** ∞
- (3,1,2) de poids $\mathbf{6+1=7}$ **pas mieux que 1**
- (3,1,4) de poids $\mathbf{6-3=3}$ **mieux que 5**
- (4,1,2) de poids $\mathbf{4+1=5}$ **mieux que** ∞
- (4,1,4) : ne nous intéresse pas (circuit de 4 vers 4

$$M_1 = \begin{pmatrix} 0 & \mathbf{1} & \infty & \mathbf{-3} \\ \mathbf{4} & 0 & 2 & \mathbf{1} \\ \mathbf{6} & 1 & 0 & \mathbf{3} \\ \mathbf{4} & \mathbf{5} & \infty & 0 \end{pmatrix}$$

$$\boldsymbol{M_1(i,j) = \inf(M_0(i,1) + M_0(1,j), M_0(i,j))}$$

$$P_1 = \begin{pmatrix} N & 1 & N & 1 \\ 2 & N & 2 & \mathbf{1} \\ 3 & 3 & N & \mathbf{1} \\ 4 & \mathbf{1} & N & N \end{pmatrix}$$

Le prédécesseur des sommets dans les nouveaux chemins est 1

Etape 1

ETAPE 2

$$M_1 = \begin{pmatrix} 0 & \mathbf{1} & \infty & -3 \\ \mathbf{4} & 0 & 2 & \mathbf{1} \\ \mathbf{6} & 1 & 0 & \mathbf{3} \\ \mathbf{4} & \mathbf{5} & \infty & 0 \end{pmatrix}$$

$$M_1(i,j) = \inf(M_0(i,1) + M_0(1,j), M_0(i,j))$$

$$P_1 = \begin{pmatrix} N & 1 & N & 1 \\ 2 & N & 2 & \mathbf{1} \\ 3 & 3 & N & \mathbf{1} \\ 4 & \mathbf{1} & N & N \end{pmatrix}$$

Le prédécesseur des sommets dans les nouveaux chemins est 1

Etape 1

$$M_2 = \begin{pmatrix} 0 & 1 & \mathbf{3} & -3 \\ 4 & 0 & 2 & 1 \\ \mathbf{5} & 1 & 0 & \mathbf{2} \\ 4 & 5 & \mathbf{7} & 0 \end{pmatrix}$$

$$M_2(i,j) = \inf(M_1(i,2) + M_1(2,j), M_1(i,j))$$

$$P_2 = \begin{pmatrix} N & 1 & ? & 1 \\ 2 & N & 2 & 1 \\ ? & 3 & N & ? \\ 4 & 1 & ? & N \end{pmatrix}$$

Le prédécesseur des sommets dans les nouveaux chemins n'est pas forcément 2 …

Etape 2

- **Chemin de 1 vers 3 : (1,2) et (2,3) → (1,2,3)**
  Le prédécesseur de 3 est bien 2
- **Chemin de 3 vers 1 : (3,2) et (2,1) → (3,2,1)**
  Le prédécesseur de 1 est bien 2
- **Chemin de 4 vers 3 : (4,2) et (2,3) → (4,2,3)**
  Le prédécesseur de 3 est bien 2
- **Chemin de 3 vers 4 : (3,2) et (2,1,4) → (3,2,1,4)**

Le prédécesseur de 4 n'est pas 2 mais 1 : le prédécesseur de 4 dans ce nouveau chemin est le correspond au **prédécesseur de 4 dans le chemin qui vient de 2**

$$\Rightarrow P_2 = \begin{pmatrix} N & 1 & 2 & 1 \\ 2 & N & 2 & \mathbf{1} \\ \mathbf{2} & 3 & N & \mathbf{1} \\ 4 & 1 & \mathbf{2} & N \end{pmatrix}$$

De façon générale :

Si un nouveau chemin a été trouvé *($M_2(i,j)$ modifié)*

$$P_2(i,j) = P_1(2,j)$$

$$M_k(i,j) \text{ modifié} \Rightarrow P_k(i,j) = P_{k-1}(k,j)$$

ETAPES 3 ET 4 : PAS DE CHANGEMENT

$$M_2 = \begin{pmatrix} 0 & 1 & \mathbf{3} & -3 \\ 4 & 0 & 2 & 1 \\ \mathbf{5} & 1 & 0 & \mathbf{2} \\ 4 & 5 & \mathbf{7} & 0 \end{pmatrix} = M_3$$

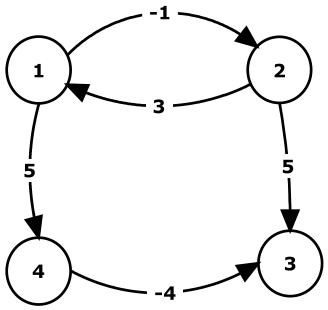
$$P_2 = \begin{pmatrix} N & 1 & 2 & 1 \\ 2 & N & 2 & \mathbf{1} \\ \mathbf{2} & 3 & N & \mathbf{1} \\ 4 & 1 & \mathbf{2} & N \end{pmatrix} = P_3$$

$$M_3 = \begin{pmatrix} 0 & 1 & \mathbf{3} & -3 \\ 4 & 0 & 2 & 1 \\ \mathbf{5} & 1 & 0 & \mathbf{2} \\ 4 & 5 & \mathbf{7} & 0 \end{pmatrix} = M_4$$

$$P_3 = \begin{pmatrix} N & 1 & 2 & 1 \\ 2 & N & 2 & \mathbf{1} \\ \mathbf{2} & 3 & N & \mathbf{1} \\ 4 & 1 & \mathbf{2} & N \end{pmatrix} = P_4$$

EXERCICE 6 (CORRIGE P16)

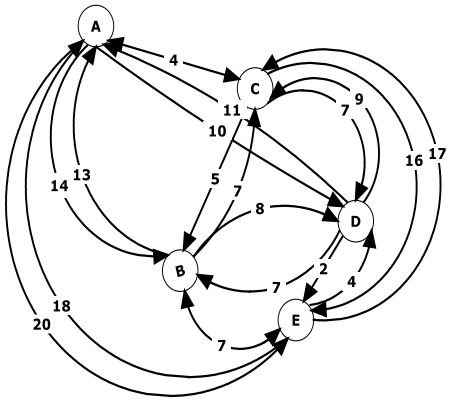
Appliquer l'algorithme de Floyd-Warshall au graphe ci-contre :



CORRIGES

EXERCICE 1

Imaginons un graphe composé de n=5 sommets comportant tous les liens possibles à l'exception des boucles : il y a 5 sommets et 4 arcs partant de chacun, donc 5x4=20 arcs.



Nombre de chemins sans circuit allant du sommet A vers le sommet B dans un graphe d'ordre 5 :

$$\mathbf{1 + 3 + 3 \times 2 + 3 \times 2 \times 1 = 16}$$

On s'intéresse au nombre de chemins sans circuit (les circuits ne pouvant que rallonger les chemins) allant du **sommet A** vers le **sommet B**. On dénombre les chemins selon leur longueur :

- **Chemins de longueur 1 :** 1 seul
- **Chemins de longueur 2**
  3 chemins (il faut choisir le sommet par lequel va passer le chemin : (A,C,B),(A,D,B),(A,E,B)
- **Chemins de longueur 3 :**
  3x2 chemins (il faut choisir les deux sommets par lesquel va passer le chemin) : (A,C,D,B), (A,C,E,B), (A,D,C,B), (A,D,E,B), (A,E,C,B), (A,E,D,B)
- **Chemins de longueur 4 :**
  3x2x1 chemins : (A,C,D,E,B), (A,C,E,D,B), (A,D,C,E,B), (A,D,E,C,B), (A,E,C,D,B), (A,E,D,C,B)

GENERALISATION

On compte le nombre de chemins entre deux nœuds

- 1 chemins de longueur 1
  Remarque : $1 = A_{n-2}^0$ (0, parce que le chemin ne passe par aucun sommet)
- $n-2$ chemins de longueur 2 (il faut choisir le troisième sommet par lequel ils passent).
  Remarque : $n-2 = A_{n-2}^1$
- $(n-2)(n-3)$ chemins de longueur 3 (il faut choisir les deux sommets par lesquels passent ces chemins, parmi n-2).
  Remarque : $(n-2)(n-3) = A_{n-2}^2$
- $(n-2)(n-3)(n-4)$ chemins de longueur 3.
  Remarque : $(n-2)(n-3)(n-4) = A_{n-2}^3$
- …
- $(n-2)!$ chemins de longueur n.
  Remarque : $(n-2)! = A_{n-2}^{n-2}$

Au total :

$$\boldsymbol{Nb_{Chemins} = A_{n-2}^0 + A_{n-2}^1 + A_{n-2}^2 + A_{n-2}^3 + \cdots + A_{n-2}^{n-2} = \sum_{k=0}^{n-2} A_{n-2}^k}$$

Ou encore :

$$\boldsymbol{Nb_{Chemins} = 1 + (n-2) + (n-2)(n-3) + (n-2)(n-3)(n-4) + \cdots + (n-2)(n-3)(n-4) \ldots 2.1}$$

Sous Python par exemple

```
def nbchem(n):
    A=n-2
    nbc=1
    l=list(range(0,n-2))
    l.sort(reverse=True)
```

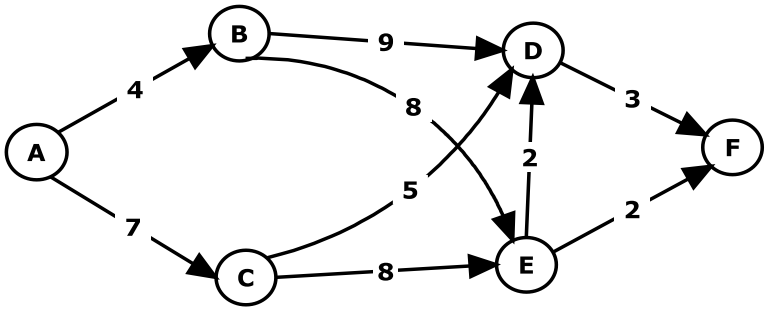
La deuxième expression suggère une méthode récursive pour calculer $Nb_{Chemins}$ (on part de $(n-2)$, puis on ajoute $(n-2)$ multiplié par $(n-3)$, puis on ajoute le produit précédent multiplié par $(n-4)$…

EXERCICE 2 (CORRIGE P)

Appliquer l'algorithme de Dijkstra pour trouver les chemins minimums en partant du sommet A vers les autres sommets

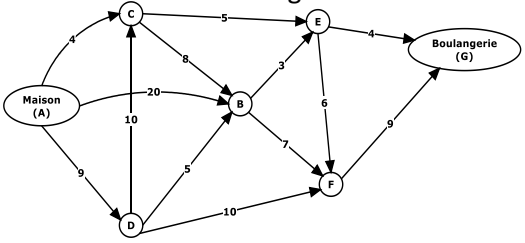
Algorithme de Dijkstra

	A	*B*	*C*	*D*	*E*	*F*
	0	∞	∞	∞	∞	∞
A	•	**4A**	7A	∞	∞	∞
B	•	•	7A	13B	12B	∞
C	•	•	•	12C	12B	∞
D	•	•	•	•	12B	15D
E	•	•	•	•	•	14E



EXERCICE 3

Appliquer l'algorithme de Dijkstra pour déterminer les chemins de poids minimum et maximum pour aller de la maison à la boulangerie :



Dijkstra

	A	B	C	D	E	F	G
	6	∞	∞	∞	∞	∞	∞
$A_{(0)}$		20_A	4_A	9_A	∞	∞	∞
$C_{(4)}$		12_C		9_A	9_C	∞	∞
$D_{(9)}$		12_C			9_C	19_D	∞
$E_{(9)}$		12_C				15_E	13_E
B_{12}						15_E	13_E
G_{13}						15_E	

	A	B	C	D	E	F	G
Dist	0	12	4	9	9	15	13
Pred	X	C	A	A	C	E	E

EXERCICE 4

Appliquer l'algorithme de Bellman Ford Kalaba pour déterminer les chemins de poids minimum et maximum pour aller de la maison à la Boulangerie (même graphe que pour l'exo 2):

Bellman Ford Kalaba (Chemins de poids minimum)

Chemins de longueur ≤ k	Sommet	A	B	C	D	E	F	G
		0	∞	∞	∞	∞	∞	∞
k = 1	A (0)	0	20$_A$	4$_A$	9$_A$	∞	∞	∞
k = 2	B (20)	0	20$_A$	4$_A$	9$_A$	23$_B$	27$_B$	∞
	C (4)	0	12$_C$	4$_A$	9$_A$	9$_C$	27$_B$	∞
	D (9)	0	12$_C$	4$_A$	9$_A$	9$_C$	19$_D$	∞
k = 3	B (12)	0	12$_C$	4$_A$	9$_A$	9$_C$	19$_D$ ou 19$_B$	∞
	E (9)	0	12$_C$	4$_A$	9$_A$	9$_C$	15$_E$	13$_E$
	F (19)	0	12$_C$	4$_A$	9$_A$	9$_C$	15$_E$	13$_E$
k = 4	F (15)	0	12$_C$	4$_A$	9$_A$	9$_C$	15$_E$	13$_E$
	G (13)	0	12$_C$	4$_A$	9$_A$	9$_C$	15$_E$	13$_E$

Invariance entre l'étape 3 et l'étape 4

	A	B	C	D	E	F	G
Dist	0	12	4	9	9	15	13
Pred	X	C	A	A	C	E	E

Bellman Ford Kalaba - Recherche de chemins de poids maximum

Chemins de longueur ≤ k	Sommet	A	B	C	D	E	F	G
		0	-∞	-∞	-∞	-∞	-∞	-∞
k = 1	A (0)	0	20$_A$	4$_A$	9$_A$	-∞	-∞	-∞
k = 2	B (20)	0	20$_A$	4$_A$	9$_A$	23$_B$	27$_B$	-∞
	C (4)	0	20$_A$	4$_A$	9$_A$	23$_B$	27$_B$	-∞
	D (9)	0	20$_A$	19$_D$	9$_A$	23$_B$	27$_B$	-∞
k = 3	C (19)	0	27$_C$	19$_D$	9$_A$	24$_C$	27$_B$	-∞
	E (23)	0	27$_C$	19$_D$	9$_A$	24$_C$	29$_E$	27$_E$
	F (27)	0	27$_C$	19$_D$	9$_A$	24$_C$	29$_E$	36$_F$
k = 4	B (27)	0	27$_C$	19$_D$	9$_A$	30$_B$	34$_F$	36$_F$
	E (24)	0	27$_C$	19$_D$	9$_A$	30$_B$	34$_F$	36$_F$
	F (29)	0	27$_C$	19$_D$	9$_A$	30$_B$	34$_F$	38$_F$
	G (36)	0	27$_C$	19$_D$	9$_A$	30$_B$	34$_F$	38$_F$
k = 5	E (30)	0	27$_C$	19$_D$	9$_A$	30$_B$	36$_E$	38$_F$
	F (34)	0	27$_C$	19$_D$	9$_A$	30$_B$	36$_E$	43$_F$
	G (38)	0	27$_C$	19$_D$	9$_A$	30$_B$	36$_F$	43$_F$
k = 6	F (36)	0	27$_C$	19$_D$	9$_A$	30$_B$	36$_F$	45$_F$
	G (43)	0	27$_C$	19$_D$	9$_A$	30$_B$	36$_F$	45$_F$
k = 7	G (45)	0	27$_C$	19$_D$	9$_A$	30$_B$	36$_F$	45$_F$

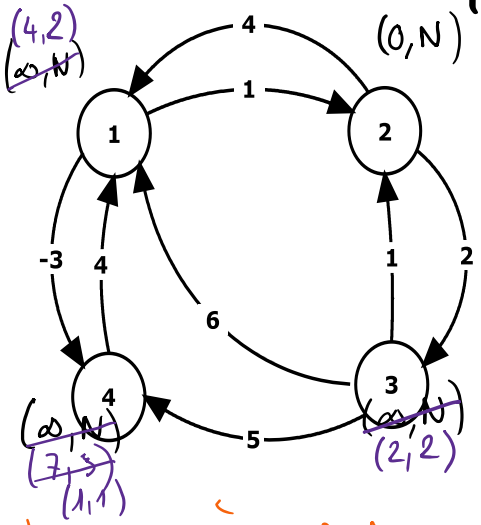
Invariance entre les étapes 6 et 7 → solution.

EXERCICE 5

Appliquer l'algorithme de Bellman pour la recherche de chemins de poids minimums partant du sommet 2 dans le graphe ci-dessous :

$A = \{(1,2), (2,1), (2,3), (3,2), (3,4), (3,1), (4,1), (1,4)\}$

Nous traiterons les sommets dans cet ordre



Etape 1 : 1er relâchement de tous les arcs

(1,2) → pas de changement
(2,1) → sommet 1 : dist[1] = 4 pred[1] = 2
(2,3) → sommet 3 : dist[3] = 2 pred[3] = 2
(3,2) → pas de changement
(3,4) → sommet 4 : dist[4] = 7 pred[4] = 3
(3,1) → pas de changement
(4,1) → pas de changement
(1,4) → sommet 4 : dist[4] = 1 pred[4] = 1

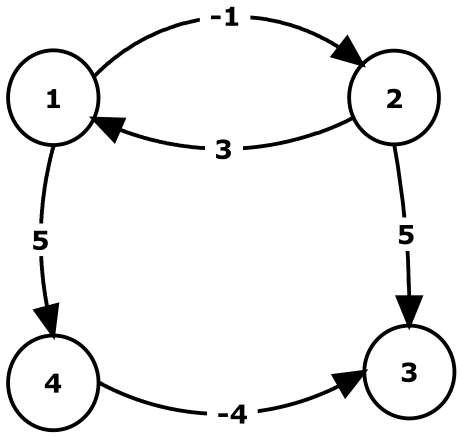
Etape 2 : 2ème relâchement de tous les arcs

(1,2) → pas de changement
(2,1) → pas de changement
(2,3) → pas de changement
(3,2) → pas de changement
(3,4) → pas de changement
(3,1) → pas de changement
(4,1) → pas de changement
(1,4) → pas de changement

Pas de changement
C'est terminé !

EXERCICE 6

Appliquer l'algorithme de Floyd-Warshall au graphe ci-dessous :



Matrice des poids : $M_0 = \begin{pmatrix} 0 & -1 & \infty & 5 \\ 3 & 0 & 5 & \infty \\ \infty & \infty & 0 & \infty \\ \infty & \infty & -4 & 0 \end{pmatrix}$

Matrice des prédécesseurs : $P_0 = \begin{pmatrix} N & 2 & N & 1 \\ 1 & N & 1 & N \\ N & N & N & N \\ N & N & 1 & N \end{pmatrix}$

Suite à l'étape 1

Matrice des poids : $M_1 = \begin{pmatrix} 0 & -1 & \infty & 5 \\ 3 & 0 & 5 & \mathbf{8} \\ \infty & \infty & 0 & \infty \\ \infty & \infty & -4 & 0 \end{pmatrix}$

Matrice des prédécesseurs : $P_1 = \begin{pmatrix} N & 1 & N & 1 \\ 2 & N & 2 & \mathbf{1} \\ N & N & N & N \\ N & N & 4 & N \end{pmatrix}$

Suite à l'étape 2

Matrice des poids : $M_2 = \begin{pmatrix} 0 & -1 & \mathbf{3} & 5 \\ 3 & 0 & 5 & \mathbf{8} \\ \infty & \infty & 0 & \infty \\ \infty & \infty & -4 & 0 \end{pmatrix}$

Matrice des prédécesseurs : $P_2 = \begin{pmatrix} N & 1 & \mathbf{2} & 1 \\ 2 & N & 2 & \mathbf{1} \\ N & N & N & N \\ N & N & 4 & N \end{pmatrix}$

Suite à l'étape 3

Matrice des poids : $M_3 = M_2 = \begin{pmatrix} 0 & -1 & \mathbf{3} & 5 \\ 3 & 0 & 5 & \mathbf{8} \\ \infty & \infty & 0 & \infty \\ \infty & \infty & -4 & 0 \end{pmatrix}$

Matrice des prédécesseurs : $P_3 = P_2 = \begin{pmatrix} N & 1 & \mathbf{2} & 1 \\ 2 & N & 2 & \mathbf{1} \\ N & N & N & N \\ N & N & 4 & N \end{pmatrix}$

Conclusion

Matrice des poids : $M_4 = \begin{pmatrix} \mathbf{0} & \mathbf{-1} & \mathbf{1} & \mathbf{5} \\ \mathbf{3} & \mathbf{0} & \mathbf{4} & \mathbf{8} \\ \infty & \infty & \mathbf{0} & \infty \\ \infty & \infty & \mathbf{-4} & \mathbf{0} \end{pmatrix}$

Matrice des prédécesseurs : $P_4 = \begin{pmatrix} \boldsymbol{N} & \mathbf{1} & \mathbf{4} & \mathbf{1} \\ \mathbf{2} & \boldsymbol{N} & \mathbf{4} & \mathbf{1} \\ \boldsymbol{N} & \boldsymbol{N} & \boldsymbol{N} & \boldsymbol{N} \\ \boldsymbol{N} & \boldsymbol{N} & \mathbf{4} & \boldsymbol{N} \end{pmatrix}$