1 Représentation par triplets

Supposons qu'on ait un graphe G à 15 sommets

et une seule arête $0 \xrightarrow{10} 1$

Si on utilise une matrice d'adjacence pour représenter G, on va stocker $15 \times 15 = 225$ coefficients dont... 224 zéros!

Or, il suffirait de connaître:

15 (nbs)

(0, 1, 10) la seule arête de G

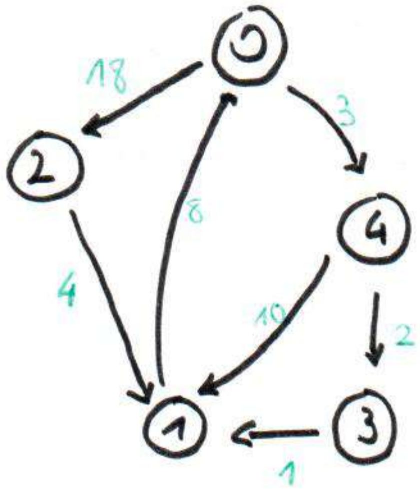
On peut éviter de stocker tous les zéros de la matrice d'adjacence en stockant de cette manière toutes les arêtes dans un vecteur contenant des triplets de la forme

(i, j, w) ⟷ arête de i vers j de poids w dans G

```
typedef struct triplet{
    int i;
    int j;
    int poids;
}triplet;

typedef struct graphe{
    int nbs;
    int nba;
    triplet *aretes;
} graphe;
```

Exemple: Pour G =



Nombre de sommets : 5

Vecteur d'arêtes : (0,4,3), (0,2,18), (1,0,8), (2,1,4), (3,1,1), (4,1,10), (4,3,2)

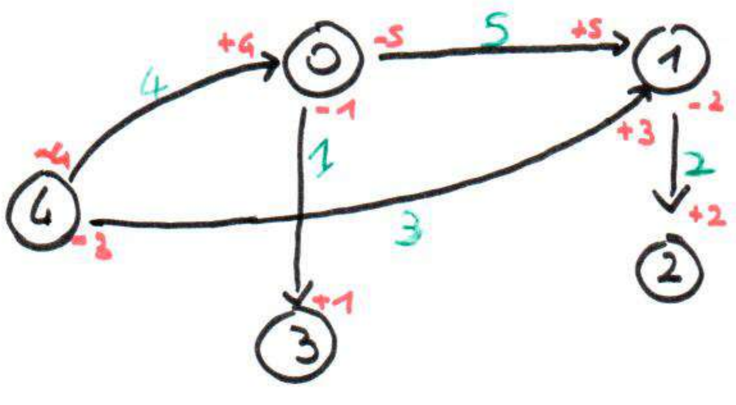
Remarque: Pour déterminer si il y a une arête de i vers j dans G, il suffit de parcourir le vecteur d'arêtes. C'est une opération en $O(a)$ où a est le nombre d'arêtes de G.

2) Représentation par brins

C'est une manière d'encoder un graphe introduite par Cori en 1973 ; très peu de références dans la littérature.

Principe: On considère un graphe, dont on va numéroter les arêtes (si le graphe est pondéré, on peut garder les poids mais il faut des poids distincts)

Ex:



Une arête numérotée n va être séparée en deux parties :

- un brin -n vers le sommet d'où part l'arête.
- un brin +n vers le sommet où arrive l'arête.

On va ensuite associer à chaque sommet un brin (le premier brin), indiqué par le brin arrivant / sortant au / du sommet avec la plus petite étiquette, et gardant son signe.

Ex:

Sommet	0	1	2	3	4
Brin	-1	-2	+2	+1	-3

Ensuite, chaque brin est associé à un brin suivant en fonction du sommet qui lui est associé, ce brin suivant étant le premier qu'on rencontre en tournant autour du sommet dans le sens ↺

On aurait pu choisir l'autre convention, mais il faut être cohérent et garder la même.

Ex:

Brin	-5	-4	-3	-2	-1	1	2	3	4	5
Sommet	0	4	4	1	0	3	2	1	0	1
Brin suivant	+4	-3	-4	+5	-5	+1	+2	-2	-1	3

2° Comment reconstruire un graphe de manière unique à partir de cette donnée?

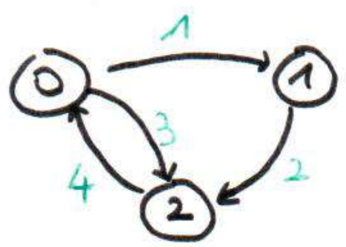
Exemple:

	-4	-3	-2	-1	1	2	3	4
Brim	-4	-3	-2	-1	1	2	3	4
Sommet	2	0	1	0	1	2	2	0
Brim suivant	+2	-1	+1	+4	-2	+3	-4	0

Om va construire um graphe á 3 sommets et 4 arêtes.

* Le fait que le brim -4 soit associé au sommet 2 va indiquer que l'arête étiquetée 4 va partir du sommet 2
* Le fait que le sommet 0 admette dans la ligne Brim suivant le +4 indique que ce brim va arriver vers 0

Om a donc une arête $② \xrightarrow{4} ⓪$ et om peut ainsi reconstruire le graphe:




```
typedef struct strand {
    Shu/int mode;
    short mext;
} strand
```

```
typedef struct strandgraph {
    Shu/int mbs;
    Shu/int mba;
    short *mode;      Brim
    strand *mxt; } strandgraphe      Sommet et brim suivant
```

③ D'autres algorithmes de plus court chemim

ⓐ Avec les puissances de la matrice d'adjacence: voir le cas de la page 13 des notes de la Partie I.

Algorithme de Floyd-Warshall

Distance d'un plus court chemin $\delta(u,v)$ entre deux sommets u et v pour toutes les paires (u,v)

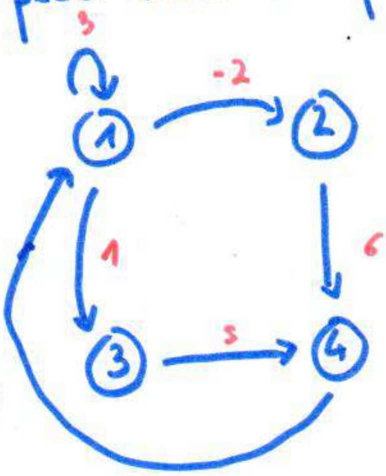
$G = (S, A)$ $\quad S = \{s_1, \dots, s_n\}$

On utilise une matrice $M :=$ $\quad M[i][j] = \begin{cases} 0 & \text{si } i = j \\ w(s_i, s_j) & \text{si } s_i \to s_j \in A \\ \infty & \text{sinon} \end{cases}$

et on veut construire une matrice $(\delta_{i,j})$ telle que $\delta_{i,j} = \delta(s_i, s_j)$

Rem: On peut avoir des poids négatifs

Ex:



$$M = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{pmatrix} 0 & -2 & 1 & \infty \\ \infty & 0 & \infty & 6 \\ \infty & 3 & 0 & 5 \\ 12 & \infty & \infty & 0 \end{pmatrix} \end{array}$$

Idée de l'algo:

$$p : v_0 \to \underbrace{v_1 \to v_2 \cdots \to v_{k-1}}_{\text{Intérieur de } P} \to v_k$$

Un chemin d'intérieur vide est un chemin de la forme

v_k

ou $\quad v_k \to v_{k+1}$

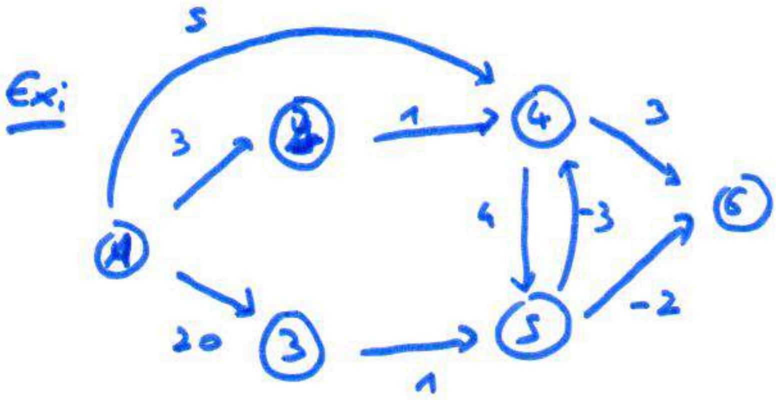
$\delta(s_i, s_j) =$ un chemin dont l'intérieur est potentiellement contenu dans tout S

L'idée est qu'on va chercher des chemins avec des intérieurs petits, et les autoriser à être de plus en plus grands

Soit $I \subseteq S$

Soit $\delta^I(u,v)$ la distance d'un PCC d'intérieur inclus dans I

Ex:



$\delta^{\emptyset}(1,6) = \infty$

$\delta^{\{4\}}(1,6) = 8$

$\delta^{\{2,4\}}(1,6) = 7$

$\delta^{\{2,4,5\}}(1,6) = 6$

$\delta^{\{2,3,4,5\}}(1,6) = 6$

Propriété:

$$\delta^{I \cup \{x\}}(u,v) = \min\left(\delta^I(u,v) ; \ \delta^I(u,x) + \delta^I(x,v) \right)$$

passer par x augmente la longueur du chemin

passer par x améliore le chemin

```
void floyd-warshall (graphe *g)
int n = g->vertices;
int dist[n][n];
:
for (k=0; k<n; k++){
    (for i=0; i<n; i++){
        for (j=0; j<n; j++){
```

Ici il faut construire la matrice initiale

```
if (dist[i][k] + dist[k][j]
        < dist[i,j])
    dist[i][j] = dist[i][k]
        + dist[k][j];
```

(c) Algorithme de Bellman - Ford

C'est un algorithme de plus court chemin qui autorise d'avoir des arêtes avec des poids négatifs; mais pas de cycles de poids négatifs, par ex

→ Le cycle 1 → 2 → 3 → 1 amène un chemin de longueur -2 de 1 vers lui-même. Ça n'a donc pas de sens de parler de plus court chemin de 0 vers 1 !

Une partie importante de l'algo consiste à vérifier qu'il n'y a pas de cycle négatif; on s'en passera ici car on garde notre hypothèse de poids positifs!

<u>Idée</u>: On va calculer les longueurs des chemins les plus courts de longueur k, pour $k{+}{+}$.

<u>observation</u>:

- Les chemins les plus courts auront au plus $n-1$ arêtes! (où n = nb de sommets)

  <u>En effet</u>: Si il y a plus de n arêtes, on passe par un sommet au moins 2 fois donc il y a forcément un cycle. Si ce cycle est de poids ≥ 0, il est mauvais pour le plus court chemin donc on l'oublie. Il ne peut pas être de poids $<= 0$.

- De plus, un chemin de longueur k de $u \to v$ est $\begin{cases} \text{un chemin de longueur } k-1 \text{ de } u \to w \text{ (où } w \text{ est un voisin de } v) \\ + \text{ une arête } w \to v \end{cases}$

Notons $dist^k[v]$ la longueur du plus court chemin de src = u vers v avec au plus k arêtes

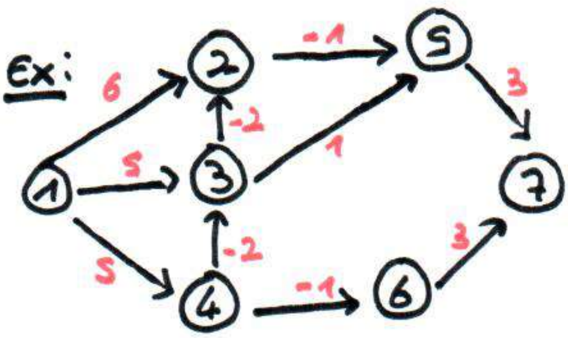
on a alors :

$$\boxed{dist^k[v] = \min\left(dist^{k-1}[v],\ dist^{k-1}[w] + poids(w \to v) \text{ pour tout voisin } w \text{ de } v\right)}$$

Formule de récurrence !

__Initialisation__ : $dist^0[u] = 0$ $dist^0[w] = INF$ pour tout autre sommet

Ex :



Longueur chemin \ Sommet	1	2	3	4	5	6	7
0	0	INF	INF	INF	INF	INF	INF
1	0	6	5	5	INF	INF	INF
2	0	3	3	5	5	4	INF
3	0	1	3	5	2	4	7
4	0	1	3	5	0	4	5
5	0	1	3	5	0	4	3
6	0	1	3	5	0	4	3

Pour la représentation triplets :

```
void bellmanford(graphe *graph, int src){
    int n = graph->nbs;
    int a = graph->nba;
    int u, v, poids;
    int *dist = malloc(n * sizeof(int))
    for (int k=0; k<n; k++)
        dist[k] = INF;
    dist[src] = 0;
        ⋮
```

```
:
for (int k=0; k<m; k++){
    for (int l=0; l<a; l++){
        u = graph -> aretes [l]. i ;
        v = graph -> aretes [l]. j ;
        poids = graph -> aretes [l]. poids;
        if (poids != 0  && dist [u] + poids < dist [v])
                dist [v] = dist [u] + poids
    }
}
print_dist (dist, m, src);
free (dist);
}
```

4 Complexités

Floyd - Warshall $O(m^3)$

Bellman - Ford:
- $O(m^3)$ avec matrice d'adjacence
- $O(m \times a)$ avec liste d'adjacence
- $O(m \times a)$ avec triplets

	Matrice Adj.	Liste Adj.	Triplets
Dijkstra	$O(2m^2)$	$O(m^2 + ma)$	$O(m^2 + ma)$
Dijkstra avec tas binaire	$O(m\log(m) + m^2)$	$O((m+a)\log(m))$	$O(m\log(m) + am$