Chapitre 29

Graphes

Sommaire.

Les propositions marquées de ★ sont au programme de colles.

1 Graphes.

1.1 Définitions.

Définition 1: Graphe

Un **graphe** est la donnée d'un ensemble de relations entre des paires d'éléments.

Définition 2: Graphe Orienté

Un **graphe orienté** est la donnée de deux ensembles :
- un ensemble fini S de **sommets** *(vertices)*.
- un ensemble $A \subset S \times S$ d'**arcs** *(edges)*.

Un tel graphe est noté (S, A), et un arc (s, t) peut être noté $s \to t$.

Notation : Sommets

Soit $u = (s, t)$ un arc d'un graphe orienté.
On appelle s son **origine**, t sa **cible** et s et t ses **extrêmités**.
On dit que t est le **successeur** de s et s est le **prédécesseur** de t.

Soit $v = \{s, t\}$ une arête d'un graphe non-orienté.
On dit que s et t sont **incidents** à l'arête v et on note parfois st ou ts l'arête $\{s, t\}$.
On dit que s est **voisin** de t ou que s et t sont **adjacents**.

Définition 3: Degrés

Dans un graphe non orienté, le **degré** d'un sommet s est le nombre d'arêtes auxquelles il est incident.
Dans un graphe orienté, le **degré entrant** d'un sommet s est le nombre d'arcs dont il est la cible, le **degré sortant** est le nombre d'arcs dont il est l'origine.
On note le degré d'un sommet s par $\delta(s)$, le degré sortant par $\delta_-(s)$ et entrant par $\delta_+(s)$.

Définition 4: Sous-graphes

Soit un graphe $G = (S, A)$.
Le graphe $G' = (S', A')$ est un **sous-graphe** de G si $S' \subset S$ et $A' \subset A$.
Soit $S'' \subset S$, le sous-graphe **induit** de G par S'' est le graphe $G'' = (S'', A'')$ où $A'' = \{(s, t) \in A \mid s, t \in S''\}$.

1.2 Dénombrement.

Proposition 5

Dans un graphe orienté (S, A), la somme des degrés entrants est égale à la somme des degrés sortants et au nombre d'arcs.

$$\sum_{s\in S} \delta_+(s) = \sum_{s\in S} \delta_-(s) = |A|$$

Proposition 6

Dans un graphe non-orienté (S, A), la somme des degrés est égale au double du nombre d'arêtes.

$$\sum_{s\in S} \delta(s) = 2|A|$$

Proposition 7

Dans un graphe orienté (S, A) à n sommets, il y a au plus n^2 arcs.
Dans un graphe non-orienté (S, A) à n sommets, il y a au plus $\frac{n(n-1)}{2}$ arêtes.

Preuve :

Graphe orienté: L'ensemble des arcs est un sous-ensemble de $S \times S$, de cardinal n^2.
Graphe non-orienté: L'ensemble des arêtes est un ensemble de 2-combinaisons de S, de cardinal $\binom{n}{2}$.

Corrolaire 8

En fixant un ensemble de sommets de cardinal n, le nombre de graphes orientés sur cet ensemble est 2^{n^2} et le nombre de graphes non-orientés sur cet ensemble est $2^{\frac{n(n-1)}{2}}$.

Preuve :

Pour chaque arc / arête, on a le choix de le mettre ou non, d'où $2^{|A|}$ possibilités.

Proposition 9

Soit $G = (S, A)$ un graphe G possède 2^n sous-graphes induits.

Preuve :

Pour chaque sommet, on a le choix de le mettre ou non, d'où 2^n possibilités.

1.3 Connexité.

Définition 10: Chemin ★

Dans un graphe orienté, un chemin de longueur n d'un sommet s vers un sommet t est une suite de sommets :

$$(s_i)_{0\leq i<n} \mid s_0 = s,\ s_{n-1} = t,\ \forall i \in [\![0, n-2]\!],\ (s_i, s_{i+1}) \in A$$

On le note $s \rightsquigarrow t$ ou $s \xrightarrow{*} t$.
On prend une définition équivalente pour les graphes non-orientés et on étend les notions d'origines, cibles et extrêmités aux chemins.
Remarque: Il existe toujours un chemin de longueur 0 d'un sommet vers lui-même.

Définition 11: Chemin simple

Un chemin est simple si un arc / une arête n'y apparaît au plus qu'une fois.
Il est élémentaire si un sommet n'y apparaît au plus qu'une fois.
Remarque: Tout chemin élémentaire est simple, la réciproque est fausse.

Proposition 12

Soient deux sommets s, t d'un graphe G orienté ou non.
S'il existe un chemin de s vers t, alors il existe un chemin élémentaire de s vers t qui en est extrait, c'est-à-dire qui est un sous-graphe du chemin.

Preuve :

On se place dans le cas d'un graphe orienté.
Par récurrence sur le longueur du chemin de s vers t :
Cas de base: Le chemin est de longueur 0, donc $s = t$, le chemin est bien élémentaire.
Hérédité: Soit $n \in \mathbb{N}^*$ tel que pour tout $k < n$, la propriété est vraie.
Soit un chemin de longueur n de s vers t. On note u l'avant-dernier sommet sur ce chemin : $s \rightsquigarrow u \to t$.
Cas 1: t apparaît dans le chemin, donc $s \rightsquigarrow t \rightsquigarrow u$. Le chemin $s \rightsquigarrow t$ est de longueur strictement inférieure à n et par hypothèse de récurrence, on peut en extraire un chemin élémentaire de s à t.

Cas 2: t n'apparaît pas dans le chemin, donc $s \rightsquigarrow u$ est un chemin de longueur $n-1$. Par hypothèse de récurrence, on peut en extraire un chemin élémentaire de s à u, puis de s à t en ajoutant l'arc (u,t).
Conclusion: Par récurrence, la propriété est vraie pour tout $n \in \mathbb{N}$.

Définition 13: Connexité

Un graphe non-orienté est **connexe** si pour tout couple de sommets, il existe un chemin entre ces deux sommets.

Définition 14: Forte connexité ★

Un graphe orienté est **fortement connexe** si pour tout couple de sommets (s,t), il existe un chemin de s vers t et un chemin de t vers s.

Proposition 15

Dans un graphe non-orienté, la connexité est une relation d'équivalence.
Dans un graphe orienté, la forte connexité est une relation d'équivalence.

Définition 16: Composantes connexes

Les classes d'équivalences définies par ces relations sont appelées les composantes (fortement) connexes du graphe.

Définition 17

Un graphe est **connexe** s'il possède une unique composante connexe.
Remarque: La composante connexe d'un sommet s d'un graphe G est le plus grand sous-graphe connexe de G contenant s.

Proposition 18

Soit $G = (S, A)$ un graphe connexe, alors $|A| \geq |S| - 1$.

Preuve :

Algorithme 1 :

Entrées : Un graphe $G = (S, A)$ connexe et un sommet $s \in S$
Sorties : On a coloré toutes les arêtes et tous les sommets différents de s
tant que il existe une arête non colorée incidente à s ou un sommet coloré **faire**
  Colorer cette arête.
  si l'autre extrêmité n'est ni s, ni colorée **alors**
    Colorer l'autre extrêmité.
  fin
fin

On suppose $|S| \geq 2$.
L'algorithme se termine : chaque itération se termine et le nombre d'arêtes non colorées est un variant de boucle.
Montrons sa correction. Soit le prédicat (P) : «si il existe des arêtes non colorées, alors il en existe au moins une dont une extrêmité est colorée ou est s». Montrons que c'est un invariant de boucle.
(P) est vrai avant la boucle car le graphe est connexe, $|S| \geq 2$ donc il existe une arête d'extrêmité s.
Supposons (P) vrai au début d'une itération.
— S'il n'existe pas d'arête non colorée, alors tous les sommets sont colorés par construction : après avoir coloré une arête, l'algorithme fait en sorte que ses deux extrêmités soient colorées, et G est connexe.
— S'il existe au moins une $\{u, v\}$ arête non colorée, il existe $s \rightsquigarrow u$. On suppose sans perte de généralité que ce chemin ne se termine pas par l'arête $\{u, v\}$ (sinon on prend $s \rightsquigarrow v$). On a donc le chemin :

$$s \to s_1 \to s_2 \to ... \to s_{n-1} \to u \to v.$$

Si u est coloré ou $u = s$, c'est bon, sinon si $\{s, s_1\}$ n'est pas colorée, on a l'arête recherchée, sinon, on considère la dernière arête colorée du chemin si l'arête suivante répond à la question.
(P) est donc un invariant: il est vrai en sortie de boucle. La condition de sortie de boucle nous assure qu'à la fin toutes les arêtes sont colorées. L'algorithme nous assure que si toutes les arêtes sont colorées, tous les sommets autres que s le sont aussi (le graphe étant connexe, il n'y a pas de sommet isolé.)
À chaque fois qu'on colorie une arête, on colorie 0 ou 1 sommet, donc $|A| \geq |S| - 1$.

Définition 19: Cycles

Un **cycle** est un chemin simple dont les extrêmités sont identiques.
Un cycle est élémentaire si un sommet ne peut y apparaître plus d'une fois, en dehors des extrêmités.
Une **boucle** dans un graphe orienté est un cycle de longueur 1.

Lemme 20

Soit un graphe non orienté connexe $G = (S, A)$ et une arête st de G.
Soit le graphe $G' = (S, A \setminus \{st\})$. G' possède au plus 2 composantes connexes.

Preuve :

1er cas: st appartient à un cycle $st \rightsquigarrow s$.
Le chemin $t \rightsquigarrow s$ ne contient pas l'arête st par définition d'un cycle. Montrons que G' est connexe.
Soit $u, v \in S$. Comme G est connexe, il existe un chemin élémentaire ρ entre u et v dans G. Si ρ ne contient pas st, alors c'est un chemin de G' et u, v appartiennent à la même composante connexe de G'. Sinon, si ρ contient st, il la contient une unique fois et SPDG, supposons que $\rho : u \rightsquigarrow st \rightsquigarrow v$. Alors $u \rightsquigarrow s \rightsquigarrow t \rightsquigarrow v$ ne contient pas st, c'est donc un chemin de G' et u, v appartiennent à la même composante connexe de G' donc G' est connexe.
2eme cas: st n'appartient à aucun cycle dans G.
Montrons que s et t n'appartiennent pas à la même composante connexe de G'.
Par l'absurde, si s et t appartiennent à la même composante connexe de G', alors il existe un chemin simple $\rho : s \rightsquigarrow t$ dans G'. Alors $\rho \to ts$ est un cycle de G contenant st, d'où contradiction. Il existe donc au moins 2 composantes connexes dans G'.
Montrons qu'il n'y en a pas d'autres.
Soit $u \in S$. G étant connexe, il existe un chemin élémentaire $s \rightsquigarrow u$ dans G. Si la première arête de ce chemin n'est pas st, alors il ne contient pas st car élémentaire, donc c'est un chemin de G' et u appartient à la même composante connexe que s dans G'. Sinon, le chemin s'écrit $st \rightsquigarrow u$ ne contenant s qu'au début car élémentaire. En particulier, $t \rightsquigarrow u$ est un chemin de G' donc u et t appartiennent à la même composante connexe de G'. Par conséquent, G' est formé de deux composantes connexes.

Proposition 21

Soit un graphe connexe $G = (S, A)$. Alors $|A| \geq |S| - 1$.

Preuve :

Par récurrence sur $|S| \in \mathbb{N}^*$.
Cas de base: $|S| = 1$ donc $|A| = 0$ donc $|A| \geq |S| - 1$.
Hérédité: Supposons la propriété vraie pour tout graphe connexe ayant au plus n sommets pour $n \in \mathbb{N}^*$.
Soit $G = (S, A)$ un graphe connexe tel que $|S| = n + 1$ et $s \in S$. On considère G' induit par $S \setminus \{s\}$. On a donc $|S'| = |S| - 1$ et $|A'| = |A| - \delta(s)$. D'après 20, le graphe (S, A') possède au plus $\delta(s) + 1$ composantes connexes dont $(\{s\}, \varnothing)$. Donc G' contient au plus $\delta(s)$ composantes connexes. Notons $G'_i = (S'_i, A'_i)$ avec $1 \leq i \leq k$ et $k \leq \delta(s)$, les composantes connexes de G'. On a pour tout i que $|S'_i| \leq |S'| = n$. On applique alors l'hypothèse à chaque G'_i et $|A'_i| \geq |S'_i| - 1$ pour tout $i \in [\![1, k]\!]$.
On a $\sum_{i=1}^{k} |S'_i| = |S'| = n$ et $\sum_{i=1}^{k} |A'_i| = |A'| = |A| - \delta(s)$.
De plus, $\sum_{i=1}^{k} |A'_i| \geq \sum_{i=1}^{k} (|S'_i| - 1)$ donc $|A| - \delta(s) \geq n - k \geq |S| - 1 - \delta(s)$ car $k \leq \delta(s)$.
Ainsi, $|A| \geq |S| - 1$.

Définition 22: Graphe acyclique

Un graphe est **acyclique** s'il ne possède pas de cycle.
Remarque: Un graphe orienté est acyclique ssi toutes ses composantes fortement connexes sont restreintes à des sommets.

Proposition 23

Tout graphe orienté acyclique possède au moins un sommet de degré entrant (resp. sortant) nul.

Preuve :

Le graphe possède un nombre fini de chemins élémentaires, en particulier, il possède un chemin élémentaire de longueur maximale, soit $\rho : u \rightsquigarrow v$. Si $\delta_+(i) = 0$, alors la propriété est vraie. Sinon, soit $w \to u$ une arête du graphe. Si w n'apparaît pas dans ρ, alors $w \to u \rightsquigarrow v$ est un chemin élémentaire strictement plus long que ρ, ce qui est impossible. Si w apparaît dans ρ, $w \to u \rightsquigarrow w$ est un cycle, ce qui est impossible.

Définition 24: Source

Dans un graphe orienté, on appelle une **source** (resp. un **puit**) un sommet dont le degré entrant (resp. sortant) est nul.
Remarque: Avec les mêmes arguments, on montre que tout graphe non-orienté non-vide possède au moins un sommet de degré au plus 1.

Proposition 25: Équivalences ★

Soit $G = (S, A)$ un graphe non-orienté. Les propriétés suivantes sont équivalentes :

1. G est connexe et acyclique.
2. G est connexe et $|A| = |S| - 1$.
3. G est connexe minimal (= on ne peut enlever une arête de G et préserver la connexité)
4. Si on considère deux sommets distincts s, t de G, il existe un unique chemin élémentaire $s \rightsquigarrow t$.
5. G est acyclique et $|A| = |S| - 1$.
6. G est acylique maximal (= si on ajoute un arête à G, on perd l'acyclicité).

Preuve :

On sait $1 \Rightarrow 2, 2 \Rightarrow 3, 5 \Rightarrow 6$.

$\boxed{3 \Rightarrow 4}$. Supposons G connexe minimal. Soient $s, t \in S$ distincts. Il existe un chemin élémentaire entre s et t car G est connexe. Montrons qu'il est unique.
Supposons qu'il en existe deux $\rho : u_0 - ... - u_k$ et $\rho' : v_0 - ... - v_l$ où $u_0 = v_0 = s$ et $u_k = v_l = t$.
Alors $\exists i \in [\![0, k]\!] \forall j \in [\![0, i-1]\!]$ $u_j = v_j$ et $u_i \neq v_i$.
Soit h le plus petit entier tel que $h \geq i$ et u_h est sommet de $v_i - ... - v_l$, il existe car c'est au moins u_k.
Alors $\exists! g > i \quad u_h = v_g$ et donc $u_{i-1} - u_i - ... - u_k = v_g - ... - v_i$ est un cycle, ce qui contredit l'hypothèse.

$\boxed{4 \Rightarrow 5}$. Montrons que G est acylique par l'absurde. Soit un cycle de G et s, t deux sommets distincts de ce cycle. Le cycle donne deux chemins simples entre s et t avec aucune arête commune. On extrait donc deux chemin élémentaires, différents de ces chemins, ce qui contredit l'hypothèse.
Donc G est acyclique et connexe $|A| = |S| - 1$

$\boxed{6 \Rightarrow 1}$. Soit G acyclique maximal. Montrons que G est connexe.
Soient s, t distincts dans S. Si $st \in A$, il y a bien un chemin entre s et t. Sinon, le graphe $G' = (S, A \cup \{st\})$ contient un cycle par hypothèse, ce cycle contient l'arête st. Il existe donc un chemin qui ne contient pas st (celui du cycle, sans st), on en déduit que G est connexe.

1.4 Compléments.

Proposition 26

Soit un graphe non-vide, non-orienté, acyclique $G = (S, A)$, on a $|A| \leq |S| - 1$.

Preuve :

Par récurrence sur $|S| \in \mathbb{N}^*$.
Cas de base: Si $|S| = 1$, $|A| = 0$ donc $|A| \leq |S| - 1$.
Hérédité: Supposons la propriété vraie pour tout graphe de taille n.
Soit $G = (S, A)$ tel que $|S| = n + 1$. Soit $s \in S$ de degré au plus 1.
Soit G' le sous graphe de G induit par $S \setminus \{s\}$ d'où $|S'| = n$ et $|A'| = |A| - \delta(s)$.
G' étant acyclique, on lui applique l'hypothèse de récurrence : $|A'| \leq |S'| - 1$.
On a $|A| \leq |A'| + 1 \leq |S'| = |S| - 1$.
Par principe de récurrence, la propriété est donc vraie pour tout graphe non-orienté acyclique non-vide.

Définition 27: Arbres, forêts.

Un graphe acyclique et connexe est appelé un **arbre**.
Un graphe acyclique est appelé une **forêt**.

Définition 28: Ordre topologique

Un **ordre topologique** sur un graphe orienté est un ordre total sur les sommets de ce graphe qui est compatible avec les arcs. Si $s \to t$ est un arc, alors $s < t$.

Proposition 29: Tri de Kahn. ★

Un graphe orienté admet un ordre topologique si et seulement si il est acyclique.

Preuve :

L'implication directe est immédiate car l'existence d'un cycle interdit tout ordre topologique.

Algorithme 2 : Tri topologique de Kahn

Entrées : $G = (S, A)$ un graphe acyclique orienté (DAG).
Sorties : Ordre topologique sur S.

```
1  sources ← file vide.
2  resultats ← file vide.
3  pour tous les s ∈ S faire
4  |  si s est une source alors
5  |  |  enfiler(sources, s).
6  |  fin
7  fin
8  tant que sources ≠ ∅ faire
9  |  s ← défiler(sources).
10 |  enfiler(resultats, s).
11 |  pour tous les t ∈ S tel que s → t faire
12 |  |  supprimer s → t de A.
13 |  |  si t est une source alors
14 |  |  |  enfiler(sources, t)
15 |  |  fin
16 |  fin
17 |  supprimer s de S.
18 fin
```

Terminaison: Lignes 3 et 11 : chaque itération se termine, il y en a un nombre fini. Ligne 8 : chaque itération se termine, le nombre de sommets non défilés décroît strictement car un sommet ne peut être enfilé au plus qu'un fois dans sources, d'où le nombre fini d'itérations.

Correction: Montrons que les propriétés suivantes forment un invariant de la boucle ligne 8 :
(i) $A \subseteq S \times S$.
(ii) (S, A) est un graphe orienté acyclique.
(iii) La file sources est exactement constituée des sources de G.
(iv) {S, résultat} est un recouvrement disjoint de S_0, $S \cap$ resultat$= \varnothing$.
(v) $s \to t \in A \iff (s \to t \in A_0$ et $s \notin$ resultats).
(vi) La liste resultats est un tri topologique du sous-graphe de G_0 induit par les sommets dans resultat.

Avant la boucle, tout est immédiat.
Notation: on ajout des prime pour ce qu'on obtient en fin d'itération; A_0 est pour A avant la boucle.
Supposons les conditions vraies au début d'une itération.
(i) Soit s supprimé dans l'itération. Alors $\delta_+(s) = 0$, les arcs dont s est l'origine sont supprimés. Juste avant de supprimer s, son degré sortant est nul. Alors $A' \subseteq (S \setminus \{s\}) \times (S \setminus \{S\}) = S' \times S'$.
(ii) Immédiat.
(iii) En enlevant s, pas de changements pour ses non successeurs, décrémentation du degré entrant de ses successeurs, s'il devient 0, on l'ajoute aux sources. Alors s est supprimé des sources et du graphe, donc (iii) est vérifié en fin d'itération.
(iv) Immédiat car l'élément ajouté à resultats est le sommet supprimé de G.
(v) Un arc existe ssi il existait dans A_0 et n'a pas été supprimé. Or un arc est supprimé ssi son origine l'est et un sommet est supprimé ssi il est dans resultats.
(vi) resultats est par hypothèse un tri topologique sur son sous-graphe induit. resultats' = resultats $\cup \{s\}$ avec s dans sources. S'il existait $t \in$ resultats tel que $s \to t \in A_0$, alors $t \neq s$ donc $t \in$ resultats et $t \notin S$ d'après (iv) donc $s \to t \notin A$ d'après (i).
On a donc bien (vi) en fin d'itération. On sait qu'un DAG non vide contient au moins une source. D'après (iii), en sortie de boucle, sources est vide donc $S = \varnothing$. D'après (iv), resultats contient tous les sommets de G_0. Alors (vi) permet de conclure que l'algorithme est correct.

2 Implémentation de graphes.

2.1 Matrices d'adjacence.

Définition 30: Matrice d'adjacence ★

Soit un graphe $G = (S, A)$ possédant n sommets $S = \{s_1, ...s_n\}$. La matrice d'adjacence de G est carrée de taille $n \times n$ sur les entiers dont le coefficient (i, j) vaut 1 ssi $s_i \to s_j \in A$ si orienté, $s_i - s_j \in A$ si non orienté et 0 sinon.
Remarque: Dans le cas non-orienté, la matrice d'adjacence est symétrique et sa diagonale ne contient que des 0.

Proposition 31

Soit M la matrice d'adjacence d'un graphe et $n \in \mathbb{N}$, pour tout couple de sommets s, t, $M^n_{s,t}$ donne le nombre de chemins de longueur n de s vers t.

Preuve :

On note $C_n(s, t)$ le nombre de chemins de longueur n de s vers t.
Par récurrence sur n.
Cas de base:, $M^0 = I$ et il existe un chemin de longueur 0 de s vers t ssi $t = s$.
Cas de base:, $M_{s,t} = C_1(s, t)$ pour tout s et t par construction de M.
Hérédité: Supposons que pour un certain $n \geq 0$, on a $\forall s, t \in S$ $M^n_{s,t} = C_n(s, t)$.

Alors $C_{n+1}(s,t) = \sum_{u\in S_s} C_n(u,t) = \sum_{u\in S_s} C_1(s,u)C_n(u,t) = \sum_{u\in S_s} M^1_{s,u}M^n_{u,t} = M^{n+1}_{s,t}$ avec S_s l'ensemble des successeurs de s.

Proposition 32

La fonction chemin a une correction totale.

Preuve :

Terminaison: Le variant d'appel récursif est len, entier positif diminuant strictement à chaque appel.
Le variant de boucle est $M.n - k$. Ces deux variants assurent la terminaison de l'appel car les autres instructions se terminent.
Correction: Par récurrence sur len.
Cas de base: Si $len = 0$, correct par la ligne 15.
Hérédité: Supposons que tout appel à chemin soit corect pour un certain $len \geq 0$. Considérons un appel avec $len + 1$. Montrons l'invariant : «Il n'existe pas de chemin de longueur $len + 1$ de i vers j qui commence par un arc $i \to k'$ pour $k' < k$.
Avant la boucle, l'ensemble des $\{k' \mid k' < k\}$ est vide donc l'invariant est vrai.
Supposons l'invariant vrai en début d'une itération l, si on atteint la fin de l'itération, il n'existe pas de chemin de longueur $len + 1$ $i \to k' \rightsquigarrow j$ avec $k' < k$ par hypothèse et il n'existe pas de chemin $i \to k \rightsquigarrow j$ car sinon on passerait à la ligne 19 par hypothèse. Donc l'invariant est vrai en fin d'itération.
Si on sort en ligne 19, l'hypothèse nous assure l'existence du chemin, sinon on sort ligne 22 et l'invariant assure l'inexistence du chemin.

Proposition 33

La complexité temporelle de la fonction chemin est en $\Theta(n^l)$ avec $n = |S|$ et $l = len$.

Preuve :

On note $T(l)$ le nombre d'opérations élémentaires de chemin quand $len = l$.
On a $T(l) \leq nT(l-1) + n + 1$ et $nT(l-1) \leq n^2T(l-2) + n^2 + n$ ... et $n^{l-1}T(1) \leq n^lT(0) + n^l + n^{l-1}$.
Alors $T(l) \leq n^lT(0) + 2\frac{1-n^{l+1}}{1-n} = O(n^l)$.

Soit $G = (S, A)$ où $S = \{s_1, ..., s_n\}$ et $A = \{s_i \to s_j \mid i, j \in [\![0, n-2]\!]\}$.
L'appel à chemin$(M, 0, n-1, l)$ vérifie $U(l) = (n-1)(1+T(l-1))$ et par la même somme téléscopique, on obtient $U(l) = (n-1)^l = U(n^l)$.
D'où $T(l) = \Omega(n^l)$.
On a donc cherché une majoration toujours valide, puis une complexité pour une famille particulière.

2.2 Listes d'adjacences.

Définition 34: Listes d'adjacences ★

Soit un graphe $G = (S, A)$ à n sommets $\{s_1, ...s_n\}$. La représentation par listes d'adjacences de G est la donnée de n listes telles que la $i$ème liste contient les sommets de G qui sont voisins de s_i.

3 Parcours de graphes.

3.1 Parcours générique.

Définition 35: Algorithme générique. ★

Algorithme 3 : Parcours générique.

Entrées : Graphe G, sommet s_0 de G.
Sorties : Un parcours des sommets de G atteignables à partir de s_0.

```
1  découverts ← sac contenant s_0.
2  tant que découverts non vide faire
3  |  s ← sommet de découverts (retiré).
4  |  si s non marqué alors
5  |  |  marquer s.
6  |  |  pour chaque t voisin de s faire
7  |  |  |  ajouter t dans découverts.
8  |  |  fin
9  |  fin
10 fin
```

Terminaison: La boucle (6) de se termine.
Pour la boucle (2), $|S|^2+|\text{decouverts}|+|\text{marqués}|\times(|-|S|-1|)$ est un variant de boucle car si le test (4) est faux, cette quantité diminue de 1, si il est vrai, elle diminue au moins de 1. Cette boucle se termine.

Proposition 36: Correction totale.

L'algorithme de parcours générique marque exactement les sommets atteignables à partir de s_0.
De plus, l'ensemble {{s, parent(s)} | parent(s) existe} est un arbre dont les sommets sont exactement les sommets atteignables à partir de s_0.

Preuve :

1. Montrons que tout sommet atteignable à partir de s_0 est marqué, par récurrence sur la distance à s_0.
Cas de base. $\text{dist}(s_0, s) = 0$ alors $s = s_0$ et s_0 est marqué à la première itération.
Hérédité. Supposons que tout sommet à distance $n \geq 0$ de s_0 sont marqués. Soit t un sommet à distance $n+1$ de s_0 et un chemin de s_0 à t de longueur $n+1$.
Soit s le dernier sommet avant t sur ce chemin, à distance n de s_0. Par le même chemin, on atteint s à partir de s_0 en n arêtes. S'il existait un chemin plus court de s_0 à s strictement plus court, alors t serait à distance strictement inférieure à $n+1$ de s_0, ce qui est absurde. Par hypothèse de récurrence, s est marqué par l'algorithme. Dans cette même itération, t est ajouté à découverts. Comme on sort de la boucle, t est extrait de découverts et on est sûr qu'il est marqué par l'algorithme.

2. Montrons que l'algorithme ne marque que des sommets atteignables à partir de s_0.
Considérons le graphe orienté H dont les arcs sont les (s, parent(s)) quand parent(s)$\neq \varnothing$, chacun de ces arcs correspond à une arête de G. Montrons que le graphe non-orienté sous-jacent à H est connexe. On peut montrer que tout chemin de H : $s \to$parent(s)$\to$parent(parent(s))$\to$ ... s'arrête sur s_0 par récurrence sur la distance à s_0.
Donc le graphe non-orienté sous-jacent à H est un arbre d'après la proposition 29.6. Si G est non-orienté, les arêtes de ce graphe sont des arêtes de G et on a le résultat, s'il est orienté, les arcs de H sont des arcs inversés de G et on a le résultat car dans H il existe pour tout s un chemin vers s_0.

Définition 37: Arbre couvrant.

Soit G un graphe. Un arbre couvrant de G est un arbre qui possède les mêmes sommets que G et est un sous-graphe de G.

3.2 Parcours en largeur.

Définition 38: Parcours en largeur.

Le parcours obtenu en prenant une file pour sac s'appelle un **parcours en largeur**.
Sa complexité est en $O(|A|)$ sur un graphe $G = (S, A)$.

Proposition 39

Soient deux sommets u et v atteignables à partir de s_0. Lors d'un parcours en largeur à partir de s_0, on a :
- Si u est marqué avant v, alors $\text{dist}(s_0, u) \leq \text{dist}(s_0, v)$.
- Si $\text{dist}(s_0, u) < \text{dist}(s_0, v)$ alors u est marqué avant v.

Preuve :

On numérote les sommets selon leur ordre de marquage dans le parcours : $s_0, s_1, ..., s_n$.
On a l'invariant :

(i) La file découvert contient dans l'ordre : $s_k, ...s_{k+l}$ avec $k, l \in \mathbb{N}$. De plus, il existe $k' \in [\![k, k+l]\!]$ tel que les sommets les plus anciens non marqués sont à une distance $d_{k'}$ de s_0 et les plus récents non marqués à une distance $d_{k'} + 1$ de s_0.

(ii) Si $d_i < d_{k'}$, alors s_i est déjà marqué.

(iii) Si $d_i = d_{k'}$, alors s_i est découvert ou marqué.

(iv) Si $d_i = d_{k'} + 1$, alors s_i est découvert ou non marqué.

(v) Si $d_i > d_{k'} + 1$, alors s_i n'est ni dans découvert ni marqué.

Avant la boucle, l'invariant est vrai. Supposons l'invariant en début d'itération.
La ligne (3) permet de récupérer s_k. S'il est marqué, on ne fait rien et les propriétés sont conservées.
Si s_k n'est pas marqué, $d_k = d_{k'}$, on marque s_k donc (iii) est vrai. Les successeurs de s_k sont soit marqués, soit à distance $d_k + 1$ donc (iv) est conservé.

Définition 40: Parcours en largeur.

Ici, le poids est la distance à s_0.

Algorithme 4 : Parcours en largeur.

Entrées : Graphe G, sommet s_0 de G.

```
1  découverts ← file vide.
2  enfiler(s_0, découverts).
3  enfiler(★, découverts).
4  tant que découverts non vide faire
5  |  s ← défiler(découverts).
6  |  si s est une ★ et la file est non-vide alors
7  |  |  enfiler(★, découverts)
8  |  fin
9  |  pour chaque arc s → v faire
10 |  |  si s → v tendu alors
11 |  |  |  relâcher(u → v).
12 |  |  |  enfiler(v, découverts).
13 |  |  fin
14 |  fin
15 fin
```

On groupe les itérations de la boucle tant que entre deux défilement d'étoiles, qu'on appelle phase.

Lemme 41

Si on numérote les phases par $i \geq 0$, alors pour tout $i \geq 0$, et tout sommet v, à la fin de la $i^{\text{ème}}$ phase, soit poids$(v) = +\infty$, soit poids$(v) \leq i$. De plus, v est dans découverts ssi poids$(v) = i$.

Preuve :

Avant la boucle, c'est immédiat.
Supposons le lemme vrai en début de phase i, la file contient tous les sommets de poids i, suivis de l'★.
On défile chaque sommet u tel que poids$(u) = i$. Si v est successeur de u, :
• Si $u \to v$ est tendu, alors poids$(v) >$ poids$(u) + \omega(u \to v) = i + 1$.
Par hypothèse, poids$(v) \leq i$ ou égal à $+\infty$, donc poids$(v) = +\infty$, donc il n'existe pas $w \to v$ avec poids$(w) < i$.
Donc à la fin de la $i^{\text{ème}}$ étape, poids$(v) = i + 1$ et v est dans découverts.

• Si $u \to v$ n'est pas tendu, alors poids$(v) \leq i$.
À la fin de la $i^{\text{ème}}$ étape, poids$(v) \leq i < i + 1$ et v n'est pas dans découverts.

Théorème 42

À la fin de l'algorithme de parcours en largeur, poids(v) est la longueur d'un plus court chemin de s à v dans G pour tout v.

Preuve :

Soit un sommet v et un chemin $s = v_0 \to v_1 \to ... \to v_l = v$.
On note $d : S \to \mathbb{N},\ v \mapsto$ la valeur finale de poids(v).
Montrons que pour tout $j \in [\![0, l]\!]$, on a $d(v_j) \leq j$ par récurrence sur j.
Cas de base: Si $j = 0$, on a $d(v_0) = d(s) = 0$, vrai.
Hérédité: Supposons $d(v_j) \leq j$ pour $j \in [\![0, l-1]\!]$.
Au moment où on atteint la valeur finale de poids(v_j), on extrait v_j de la file.
On regarde $v_j \to v_{j+1}$.
• S'il n'est pas tendu, poids$(v_{j+1}) \leq$ poids$(v_j) + \omega(v_j \to v_{j+1}) = d(v_j) + 1 \leq j + 1$
Donc $d(v_{j+1}) \leq$ poids$(v_{j+1}) \leq j + 1$.
• S'il est tendu, alors poids(v_{j+1}) prend la valeur poids$(v_j) + \omega(v_j \to v_{j+1}) = d(v_j) + 1$.
Donc poids$(v_{j+1}) \leq j + 1$.

On applique le résultat à $j = l$:
La longueur du chemin initial est supérieure à la longueur du chemin trouvé par l'algorithme. Comme on n'a pas mis de condition sur le chemin initial, le chemin trouvé par l'algorithme est le chemin le plus court.

3.3 Parcours en profondeur.

Définition 43: Parcours en profondeur.

Le parcours obtenu en prenant une pile pour sac s'appelle un **parcours en profondeur.**
Sa complexité est en $O(|A|)$ sur un graphe $G = (S, A)$.

Algorithme 5 : Parcours en profondeur récursif.

Entrées : Graphe G, sommet s de G.
Sorties : Un parcours des sommets de G atteignables à partir de s.

```
1  marquer s.
2  pour chaque t voisin de s non marqué faire
3  |  parcours_en_profondeur(G, t).
4  fin
```

3.4 Graphes non orientés non connexes.

Définition 44

On peut vouloir parcourir tous les sommets, pas uniquement la composante connexe d'un sommet donné.

Algorithme 6 : Générique de parcours de graphe.

Entrées : un graphe G.
Sorties : Un parcours de tous les sommets de G, nombre de composantes connexes.

```
1 n ← 0.
2 pour chaque sommet s de G faire
3 |   si s n'est pas marqué alors
4 |   |   n ← n + 1.
5 |   |   algorithme_générique(G, s).
6 |   fin
7 fin
8 retourner n.
```

4 Enrichir les graphes.

4.1 Graphe bipartis.

Définition 45: Graphe biparti.

Un graphe $G = (S, A)$ est **biparti** s'il existe un recouvrement disjoint de S par deux ensembles S_1 et S_2 tel que toute arête possède une extrémité dans S_1 et l'autre dans S_2.

Proposition 46

Sur les graphes bipartis, il y a un lien entre :
- La couverture minimum par sommets.
- Les couplages parfaits : extraire un ensemble d'arêtes tel que deux arêtes ne partagent pas d'extrêmité commune et tout sommet du graphe est une extrêmité d'une des arêtes.

Les deux sont de même tailles.

Théorème 47: Caractérisation.

Soit G un graphe, il y a équivalence entre :
1. G est biparti.
2. G ne contient pas de cycle de longueur impaire.

Preuve :

$\boxed{\Longrightarrow}$ Supposons G biparti. Par l'absurde, on suppose l'existence d'un cycle $s_0 \to ... \to s_{2k+1} = s_0$ de longueur impaire. Sans perte de généralité, $s_0 \in S_1$.
Alors pour tout i pair, $s_i \in S_1$ et pour tout i impair, $s_i \in S_2$ donc $s_0 \in S_1 \cap S_2$, absurde.

$\boxed{\Longleftarrow}$ Supposons G sans cycle de longueur impaire.
1er cas: pas de cycles. Alors G est une forêt et on colorie les sommets de chaque arbre en alternant par parité du niveau, donc G est biparti.
2ème cas: il existe un cycle. On prend une composante connexe de G, notée H. Alors H possède un arbre couvrant, on fabrique S_1 et S_2 à partir de cet arbre.
Soit uv une arête de H. Montrons que u et v sont dans des ensembles différents.
Si uv appartient à l'arbre couvrant, c'est correct.
Si uv n'y appartient pas, alors il existe un chemin de u à v dans H, on en extrait un sous-chemin élémentaire $u \rightsquigarrow v$. En ajoutant l'arête uv, on obtient un cycle, de longueur paire par hypothèse, donc $u \rightsquigarrow v$ est de longueur impaire. On en déduit que u et v sont dans des ensembles différents.

4.2 Graphes pondérés.

4.2.1 Définitions.

Définition 48: Graphe pondéré.

Un **graphe pondéré** est la donnée d'un graphe $G = (S, A)$ et d'une fonction $w : A \to \mathbb{R}$.
On attribue parfois le poids $+\infty$ à une arête pour signifier qu'elle n'existe pas.

Définition 49: Poids d'un chemin.

Le **poids** d'un chemin est la somme des poids de ses arêtes.

4.2.2 Chemins de poids minimal.

Proposition 50

Soit G un graphe orienté pondéré sans cycle de poids négatif, s un sommet de G et G' induit par l'ensemble des sommets atteignables à partir de s. Le graphe G' possède un arbre couvrant de racine s contenant les chemins de poids minimaux de s vers les autres sommets de G'.

Définition 51: Initialisation de poids.

Principe : on accumule en maintenant à jour les meilleures données «jusque-là» :

- Le poids minimal d'un chemin de s à chaque sommet pour l'instant.
- Le prédecesseur de chaque sommet sur un chemin de poids minimal.

Algorithme 7 : Initialisation de poids.

Entrées : un graphe G et un sommet s.
Sorties : Initialisation des poids et des prédecesseurs.

```
1 poids(s) ← 0.
2 pred(s) ← ∅.
3 pour chaque t ≠ s faire
4 |  poids(t) ← +∞.
5 |  pred(t) ← ∅.
6 fin
```

Définition 52: Arc tendu.

On dit qu'un arc $u \to v$ est tendu si $\text{poids}(u) + \omega(u \to v) < \text{poids}(v)$.
Si $u \to v$ est tendu, alors l'inégalité triangulaire n'est pas vérifiée.

Définition 53: Relâcher un arc.

Algorithme 8 : Relâcher un arc.

Entrées : poids, pred, G, $u \to v$ tendu.
Sorties : poids, pred mis à jour ($u \to v$ n'est plus tendu).

```
1 poids(v) ←poids(u) + ω(u → v).
2 pred(v) ← u.
```

Définition 54: Algorithme de Ford. ★

Algorithme 9 : Algorithme de Ford.

Entrées : un graphe G et un sommet s.
Sorties : Pour tout sommet t, un chemin de poids minimal $s \rightsquigarrow t$.

```
1 initialisation_de_poids(G, s).
2 tant que il reste un arc tendu u → v faire
3 |  relâcher(u → v).
4 fin
```

Remarque: Les poids associés à un sommet diminuent pendant le déroulement de l'algorithme.

- À tout moment, pour un sommet v, soit $\text{poids}(v)$ vaut $+\infty$, soit c'est le poids d'un chemin de s à v.
- En l'absence de cycle de poids négatif, $\text{poids}(v)$ vaut $+\infty$ ou le poids d'un chemin élémentaire de s à v.
- Pour tout sommet v, $s \to ... \to \text{pred}(\text{pred}(v)) \to \text{pred}(v) \to v$ est un chemin de G.
- Si aucun arc n'est tendu, alors pour tout v, $\text{poids}(v)$ est le poids de ce chemin.
- Ce chemin est de poids minimal de s vers v.

4.3 Algorithme de Dijkstra.

Définition 55: Algorithme de Dijkstra. ★

Algorithme 10 : Algorithme de Dijkstra.

Entrées : Graphe G pondéré sans cycle de poids strictement négatif, s sommet de G.
Sorties : Pour tout t, $s \rightsquigarrow t$ de poids minimal.

```
1  initialisation_de_poids(G, s).
2  découverts ← file de priorité vide.
3  enfiler((s, 0), découverts).
4  tant que découverts non vide faire
5  |  u ← défiler(découverts).
6  |  pour chaque u → v tendu faire
7  |  |  relâcher(u → v).
8  |  |  si v est dans découverts alors
9  |  |  |  actualiser(v, découverts).
10 |  |  fin
11 |  |  sinon
12 |  |  |  enfiler((v, poids(v)), découverts).
13 |  |  fin
14 |  fin
15 fin
```

Proposition 56

Pendant l'exécution de l'algorithme, l'arc $u \to v$ est tendu ssi u est dans découverts ou si c'est le sommet le plus récent extrait de découverts.

Notation

Pour $i \in \mathbb{N}$, on note :

- u_i le sommet extrait à l'itération i.
- d_i le poids associé à u_i juste après l'extraction de u_i.

Lemme 57

Pour un graphe G de poids positifs, on a $i < j \implies d_i \leq d_j$.

Preuve :

Il suffit de montrer le résultat pour $j = i + 1$.
Si au début de la $i^{\text{ème}}$ itération, u_{i+1} est déjà dans découverts, alors au moment de l'extraction de u_i, puisque c'est u_i qui est extrait : poids$(u_i) \leq$ poids(u_{i+1}). Si $u_i \to u_{i+1}$ n'existe pas ou n'est pas tendu, alors poids(u_{i+1}) n'est pas modifié et $d_{i+1} =$ poids(u_{i+1}).
Si $u_i \to u_{i+1}$ est tendu, on le relâche, alors poids(u_{i+1}) devient poids$(u_i) + \omega(u_i \to u_{i+1}) = d_{i+1}$.
Si u_{i+1} n'est pas dans découverts au début de la $i^{\text{ème}}$ itération, il est nécessairement ajouté à découverts pendant la $i^{\text{ème}}$ itération : le poids qui lui est associé est alors $d_{i+1} =$ poids$(u_i) + \omega(u_i \to u_{i+1})$.

Lemme 58

Chaque sommet est extrait de découverts au plus une fois.

Preuve :

Par l'absurde, supposons qu'il existe v extrait deux fois.
Alors $\exists i, j \in \mathbb{N},\ i < j \mid v = u_i = u_j$, donc $d_i \leq d_j$, or le poids diminue strictement quand on l'ajoute à découverts.
D'où contradiction.

Théorème 59

L'algorithme de Dijkstra possède une correction totale sur les graphes à pondération positive.

Preuve :

Soit v un sommet de G et un chemin $s : v_0 \to v_1 \to ... \to v_l = v$.
On note ω_j le poids du préfixe $v_0 \to v_1 \to ... \to v_j$.
On montre par récurrence sur j qu'à la fin de l'algorithme de Dijkstra, on a poids$(v_j) \leq \omega_j$ pour tout j.
Cas de base: $j = 0$, on a poids$(v_0) = 0 = \omega_0$.
Hérédité: Supposons que pour un certain $j \geq 0$, on a poids$(v_j) \leq \omega_j$.
On considère $v_j \to v_{j+1}$.
• Si $v_j \to v_{j+1}$ n'est pas tendu, alors poids$(v_{j+1}) \leq$ poids$(v_j) + \omega(v_j \to v_{j+1}) = \omega_j + \omega(v_j \to v_{j+1}) = \omega_{j+1}$.
• Si $v_j \to v_{j+1}$ est tendu, alors poids$(v_{j+1}) = \omega_j + \omega(v_j \to v_{j+1}) = \omega_{j+1}$.
Donc poids$(v_{j+1}) \leq \omega_{j+1}$.

Théorème 60

La complexité de l'algorithme de Dijkstra sur un graphe $G = (S, A)$ à poids positifs est en $O(|A| \log(|S|))$.

Définition 61: Dijkstra à pondération positive.

Algorithme 11 : Algorithme de Dijkstra

Entrées : G à poids positifs, s dans G.
Sorties : Pour tout t, $s \rightsquigarrow t$ de poids minimal.
`initialisation_de_poids`(G, s).
découverts $\leftarrow$ file de priorité vide.
pour chaque sommet v de G **faire**
  `enfiler`$((v, \omega(v))$, découverts).
  tant que découverts n'est pas vide **faire**
    $u \leftarrow$ `défiler`(découverts).
    pour chaque $u \to v$ tendu **faire**
      `relâcher`$(u \to v)$.
      `actualiser`$(v$, découverts).
    fin
  fin
fin

4.4 Chemin de poids minimal entre tous les couples de sommets.

4.4.1 Programmation dynamique.

Définition 62

Pour un graphe pondéré $G = (S, A)$ sans cycle de poids négatif.

$$\text{poids}(u, v) = \begin{cases} 0 & \text{si} \quad u = v \\ \min\limits_{x | x \to v \in A} (\text{poids}(u, x) + \omega(x \to v)) & \text{sinon.} \end{cases}$$

permet d'écrire du code qui s'arrête uniquement si le graphe G ne possède pas de cycle.

Soit $\text{poids}(u, v, l)$ le poids minimal de u vers v de longueur au plus l. En l'absence de cycles de poids strictement négatif, on sait qu'il existe un chemin élémentaire parmi les chemins de poids minimal de u à v pour tout $u, v \in S$. Donc la valeur recherchée est $\text{poids}(u, v, |S| - 1)$.
On a alors :

$$\text{poids}(u, v, l) = \begin{cases} 0 & \text{si} \quad u = v \\ +\infty & \text{si} \quad u \neq v \quad \text{et} \quad l = 0 \\ \min\limits_{x | x \to v \in A} & (\text{poids}(u, x, l-1) + \omega(x \to v)) \quad \text{sinon.} \end{cases}$$

Définition 63

Algorithme 12 : Poids minimal pour toutes les paires.

Entrées : G pondéré sans cycles de poids négatif.
Sorties : Chemin de poids minimal entre tous les couples.
pour chaque sommet u **faire**
  pour chaque sommet v **faire**
    si $u = v$ **alors**
      $\text{poids}(u, v, 0) \leftarrow 0$.
    fin
    sinon
      $\text{poids}(u, v, 0) \leftarrow +\infty$.
    fin
  fin
fin
pour $l \leftarrow 1$ **à** $|S| - 1$ **faire**
  pour chaque sommet u **faire**
    pour chaque sommet v **faire**
      $\text{poids}(u, v, l) \leftarrow +\infty$.
      pour chaque arc $x \to v$ **faire**
        si $\text{poids}(u, x, l-1) + \omega(x \to v) < \text{poids}(u, v, l)$ **alors**
          $\text{poids}(u, v, l) \leftarrow \text{poids}(u, x, l-1) + \omega(x \to v)$.
        fin
      fin
    fin
  fin
fin

4.4.2 Diviser pour régner.

Définition 64

On suppose $l = 2^k$ pour un certain $k \in \mathbb{N}$.

$$\text{poids}(u, v, l) = \begin{cases} \omega(u \to v) & \text{si} \quad l = 1 \\ \min\limits_{x \in S} (\text{poids}(u, x, l/2) + \text{poids}(x, v, l/2)) & \text{sinon.} \end{cases}$$

Définition 65

On prend l'exposant comme troisième paramètre de poids.

Algorithme 13 : Poids minimal pour toutes les paires.

Entrées : G à poids positifs, s dans G.
Sorties : Pour tout t, $s \rightsquigarrow t$ de poids minimal.

```
pour chaque sommet u faire
    poids(u, u, 0) ← 0.
    pour chaque sommet v ≠ v faire
        poids(u, v, 0) ← ω(u → v)
    fin
fin
pour k = 1 à ⌈log_2 |S|⌉ faire
    pour chaque sommet u faire
        pour chaque sommet v faire
            poids(u, v, k) ← +∞.
            pour chaque sommet x faire
                si poids(u, x, k − 1)+poids(x, v, k − 1) <poids(u, v, k) alors
                    poids(u, v, k) ←poids(u, x, k − 1)+poids(x, v, k − 1).
                fin
            fin
        fin
    fin
fin
```

4.4.3 Floyd-Warshall.

Définition 66: Algorithme de Floyd-Warshall.

On numérote les sommets de 1 à S.
Soit poids(u, v, r) le poids minimal d'un chemin de u à v dont les sommets intermédiaires appartiennent à $[\![1, r]\!]$.

$$\text{poids}(u,v,r) = \begin{cases} 0 \quad \text{si} \quad u = v \\ \omega(u \to v) \quad \text{si} \quad r = 0 \\ \min\left(\text{poids}(u,v,r-1), \text{poids}(u,r,r-1) + \text{poids}(r,v,r-1)\right) \quad \text{sinon.} \end{cases}$$

On cherche poids$(u, v, |S|)$.

Algorithme 14 : Algorithme de Floyd-Warshall.

Entrées : Graphe G à poids positifs.
Sorties : Pour tout u, v, $u \rightsquigarrow v$ de poids minimal.

```
pour chaque sommet u faire
    poids(u, u, 0) ← 0.
    pour chaque sommet v faire
        poids(u, v, 0) ← ω(u → v).
    fin
fin
pour r = 1 à |S| faire
    pour chaque sommet u faire
        pour chaque sommet v faire
            poids(u, v, r) ← min (poids(u, v, r − 1), poids(u, r, r − 1) + poids(r, v, r − 1))
        fin
    fin
fin
```