

# Graphes

## 1 Définitions

### 1.1 Graphes

Un graphe **non orienté** est la donnée d'un couple  $G = (V, E)$ , où  $V$  est un ensemble fini non vide et  $E \subset \{\{x, y\} \mid (x, y) \in V^2\}$ . Un graphe **orienté** est la donnée d'un couple  $H = (S, A)$ , où  $S$  est un ensemble fini non vide et  $A \in \mathcal{P}(S^2)$ .

Les éléments de  $V$  et  $A$  sont appelés **sommets du graphe**. On parlera, pour désigner les éléments de  $E$  dans un graphe non-orienté d'**arêtes** du graphe, et pour un graphe orienté d'**arcs**.

Si  $e = \{x\} \in E$  (avec  $x \in V$ ),  $e$  est une **boucle** sur  $x$  (idem pour  $e = (x, x)$  pour  $x \in S$ ). Pour  $(x, y) \in V^2$ , on dit que  $x$  et  $y$  sont **voisins** ssi  $\{x, y\} \in E$ . Cette notion se précise dans un graphe orienté : on dit que  $x \in S$  est un **successeur** (resp. **prédécesseur**) de  $y$  ssi  $(y, x) \in A$  (resp.  $(x, y) \in A$ ).

On appelle **voisinage** du sommet  $x$  l'ensemble  $\mathcal{V}(x)$  de ses voisins. Le **degré** de  $x$ , noté  $\deg x$ , est le cardinal de ce voisinage.

Pour les graphes orientés, on distingue le **degré sortant** de  $x$ , noté  $\deg^+ x$ , le nombre de successeurs de  $x$ , du **degré entrant** de  $x$ , noté  $\deg^- x$ , le nombre de prédécesseurs de  $x$ .

*On supposera par la suite que l'on travaille avec des graphes sans boucles.*

**Propriété :** Soit  $G = (V, E)$  un graphe. On a  $\sum_{x \in V} \deg(x) = 2 \text{ card}(E)$

▷ On a :

$$\sum_{x \in V} \deg(x) = \sum_{x \in V} \sum_{y \in V} \mathbb{1}_{\{x, y\}} = \sum_{(x, y) \in V^2} \mathbb{1}_{\{x, y\}} = 2 \sum_{\{x, y\} \in V^2} \mathbb{1}_{\{x, y\}} = 2 \text{ card}(E).$$

Car le graphe est sans boucle. Il faudrait sinon ajouter le nombre de boucles présentes dans le graphe.

### 1.2 Accessibilité, connexité

*On fixe  $G = (V, E)$  un graphe non orienté, et  $H = (A, S)$  un graphe orienté.*

Soit  $s = (s_i) \in V^{n+1}$ . On dit que  $s$  est une **chaîne de**  $G$  ssi  $\forall i \in \llbracket 1, n \rrbracket, \{s_i, s_{i+1}\} \in E$ . On dit alors que  $s$  est une chaîne de longueur  $n$  et qui relie  $s_0$  et  $s_n$ .

Soit  $s = (s_i) \in A^{n+1}$ . On dit que  $s$  est un **chemin de**  $G$  ssi  $\forall i \in \llbracket 1, n \rrbracket, \{s_i, s_{i+1}\} \in E$ . On dit alors que  $s$  est une chaîne de longueur  $n$  et qui relie  $s_0$  à  $s_n$ .

On dit alors que  $s_n$  est **accessible** depuis  $s_0$ . Par ailleurs, si  $s_n = s_0$ , on dit que  $s$  est un **cycle** pour un graphe non-orienté, ou un **circuit** dans un graphe orienté.

Si tous les  $(s_i)$  sont distincts, on dit que  $s$  est **élémentaire**.

**Remarque :** Il y a toujours un nombre fini de chaînes élémentaires, mais si  $G$  (resp.  $H$ ) a des cycles (resp. des circuits), il y a un nombre infini de chaînes (il suffit de tourner en rond...).

**Exercice 1:** Définir la relation entre les circuits/chemins d'un graphe, qui met en relation deux circuits/chemins ssi ils relient les mêmes sommets. Est-ce une relation d'équivalence ?

**Propriété :** La relation  $\leftrightarrow$  définie sur  $V^2$  par  $x \leftrightarrow y$  ssi  $x$  est accessible depuis  $y$  est une relation d'équivalence.

- ▷ Soit  $x \in V$ . On a bien  $x \leftrightarrow x$  : la chaîne de longueur  $n = 0$   $s = (x)$  convient.
- ▷ Soit  $(x, y) \in V^2$ , tel que  $x \leftrightarrow y$ . Alors par définition il existe  $s = (s_0, \dots, s_n) \in V^{n+1}$  tel que  $s_0 = x$  et  $s_n = y$ , et  $\forall i \in \llbracket 0, n \rrbracket, \{s_i, s_{i+1}\} \in E$ . Considérons  $s' = (s_n, s_{n-1}, \dots, s_1, s_0)$ .  $s'$  est une chaîne reliant  $y$  et  $x$ . En effet,  $s'_0 = s_n = y$  et  $s'_n = s_0 = x$ , et  $\forall i \in \llbracket 0, n \rrbracket, \{s'_i, s'_{i+1}\} = \{s_{n-i}, \{n-i-1\} = \{s_k, s_{k+1}\}\} \in E$  en posant  $k = n - i - 1 \in \llbracket 0, n \rrbracket$ .
- ▷ Soit  $(x, y, z) \in V^3$  tel que  $x \leftrightarrow y$  et  $y \leftrightarrow z$ . Comme  $x \leftrightarrow y$ , il existe  $s = (s_0, \dots, s_n) \in V^{n+1}$  une chaîne avec  $s_0 = x$  et  $s_n = y$ . Comme  $y \leftrightarrow z$ , il existe  $t = (t_0, \dots, t_m) \in V^{m+1}$  une chaîne avec  $t_0 = y$  et  $t_m = z$ . Considérons  $u = (s_0, \dots, s_n, t_1, \dots, t_m)$ .  $u$  est bien une chaîne car  $\forall i \in \llbracket 0, n+m \rrbracket$ , soit  $i \in \llbracket 0, n \rrbracket$  et dans ce cas on a  $\{u_i, u_{i+1}\} = \{s_i, s_{i+1}\} \in A$ , soit  $i = n$  et on a  $\{u_i, u_{i+1}\} = \{t_0, t_1\} \in E$ , soit  $i \in \llbracket n, n+m \rrbracket$  et  $\{u_i, u_{i+1}\} = \{t_{i-n}, t_{i-n+1}\} \in E$ . On a par ailleurs  $u_0 = x$  et  $u_{n+m} = z$ , donc  $x \leftrightarrow z$ .

**Exercice 2:** Définir une relation d'équivalence similaire pour  $H$ , où l'on doit avoir un chemin dans chaque sens entre deux points en relation.

Une **composante connexe** de  $G$  est une classe d'équivalence pour la relation d'équivalence définie ci-dessus. Si  $G$  n'admet qu'une composante connexe, on dit que  $G$  est un **graphe connexe**. Dans le cas de la relation d'équivalence sur les graphes orientés, on appelle les classes d'équivalences **composante fortement connexe**.

Soit  $W \subset V$  avec  $W \neq \emptyset$ .  $W$  est **connexe** ssi  $\forall (x, y) \in W^2$ , il existe une chaîne reliant  $x$  et  $y$ .

**Propriété :**  $W$  est une composante connexe ssi  $W$  est connexe minimal, c'est à dire si  $\forall W' \subset V \setminus \{W\}, W \subset W', W'$  n'est pas connexe.

Soit  $G' = (V', E')$  un graphe.  $G'$  est un **sous-graphe** ssi  $V' \subset V, E' \subset E$ .

Soit  $V' \subset V$  Le **graphe induit par  $G$  sur  $V'$**  est  $G' = (V', \{\{x, y\} \in E \mid (x, y) \in V'^2\})$ .

**Propriété :** Un ensemble de sommets est connexe ssi le graphe qu'il induit est connexe.

- ▷ En exercice.

## 1.3 Types de graphes

Un graphe non-orienté (resp. orienté) est dit **acyclique** s'il ne contient aucun cycle élémentaire (resp. aucun circuit).

Un **arbre** est un graphe connexe acyclique. (cf TD pour caractérisation).

Un graphe acyclique décomposé en ses composantes connexes (qui sont donc des arbres), est appelé **forêt**.

Un graphe non-orienté  $(V, E)$  est dit **biparti** ssi il existe une partition  $\{W_1, W_2\}$  de  $V$  telle que toutes les arêtes de  $E$  aient une extrémité dans  $W_1$  et l'autre dans  $W_2$ .

## 2 Parcours

### 2.1 Définitions

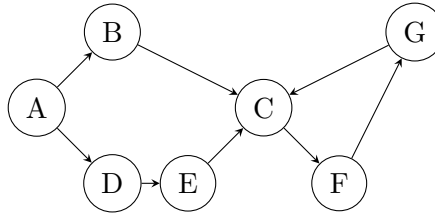
On définit, pour  $W \subset V$  la bordure de  $W$  par :

$$\mathcal{B}(W) = \{y \in V \setminus W \mid \exists x \in W, \{x, y\} \in E\}$$

Dans le cadre d'un graphe orienté, pour  $T \subset S$  :

$$\mathcal{B}(T) = \{y \in V \setminus T \mid \forall x \in T, y \in \mathcal{V}(x)\}$$

On dit que  $L \in V^n$  (ou  $S^n$  pour un graphe orienté) est un **parcours** ssi  $\forall i \in \llbracket 1, n \rrbracket, L_i \in \mathcal{B}(\{L_j \mid j \in \llbracket 1, i \rrbracket\})$  ou  $\mathcal{B}(\{L_j \mid j \in \llbracket 1, i \rrbracket\})$  est vide.

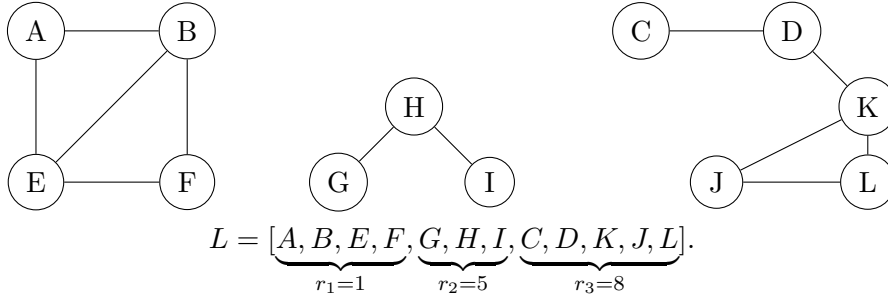


Un parcours du graphe orienté ci-dessus est  $L = [F, G, C, E, A, B, D]$

De plus, on dit que  $L_i$  est un **point de régénération du parcours** ssi  $\mathcal{B}(\{L_j \mid j \in \llbracket 1, i \rrbracket\}) = \emptyset$ .

Enfin, en notant  $\mathcal{R}$  l'ensemble des points de régénération de  $L$ , on dit que  $F = (V, P)$  est une forêt d'arborescences associée au parcours  $L$  ssi  $F$  respecte les trois propriétés suivantes :

1.  $\forall i \in \llbracket 1, n \rrbracket, L_i \in \mathcal{R}$  ou  $\exists j \in \llbracket 1, i \rrbracket, L_i \in \mathcal{V}(L_j), (L_j, L_i) \in P$  ;
2.  $\forall u \in V \setminus R, \exists! w \in V, (w, u) \in P$  (on dit alors que  $w$  est le **père** de  $u$ ) ;
3.  $F = (V, P)$ .  $P$  est minimal parmi les ensemble vérifiant 1.



**Propriété :** Soit  $G = (V, E)$  un graphe non-orienté. Soit  $W \subset V$ . Si  $\mathcal{B}(W) = \emptyset$ , alors il n'existe aucune chaîne reliant un sommet de  $W$  et un sommet de  $V \setminus W$ .

- ▷ Par l'absurde, supposons qu'il existe une chaîne  $\gamma$  de longueur  $l$  telle que  $\gamma_0 \in W$  et  $\gamma_l \in V \setminus W$ . On a  $\gamma_0 \neq \gamma_l$  donc  $l > 0$ . On peut alors définir  $i_0 = \min \{i \in \llbracket 1, l \rrbracket \mid \gamma_i \notin W\}$ . Par définition de  $i_0$ ,  $\gamma_{i_0-1} \in W$ . Par définition, une chaîne  $\{\gamma_{i_0}, \gamma_{i_0}\} \in E$ , autrement dit  $\gamma_{i_0} \in \mathcal{V}(\gamma_{i_0-1})$ . Donc  $\gamma_{i_0} \in \mathcal{B}(W)$  : absurde.

**Propriété :** Soit  $G = (V, E)$  un graphe non-orienté, et  $L = (L_i)_{i \in \llbracket 1, n \rrbracket}$  un parcours de  $G$ . Si l'ensemble des points de régénération s'écrit  $R = \{L_{r_k} \mid k \in \llbracket 1, K \rrbracket\}$  avec  $(r_k)$  croissant, alors  $G$  admet  $K$  composantes connexes, à savoir les  $(C_k)_{k \in \llbracket 1, K \rrbracket}$  définis par  $C_k = \{L_i \mid i \in \llbracket r_k, r_{k+1} \rrbracket\}$  avec  $r_{K+1} = n + 1$ .

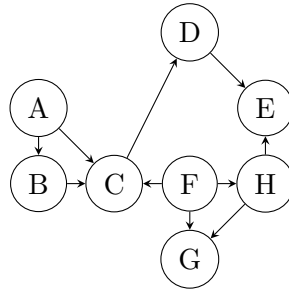
- ▷ Soit  $k \in \llbracket 1, K \rrbracket$ . Montrons que  $C_k$  est connexe maximal.
- ▷ Si  $C_k = V$ , il est trivialement connexe. Sinon, soit  $u \in V \setminus C_k$ . MQ  $\hat{C}_k = C_k \cup \{u\}$  n'est pas connexe. Par définition d'un parcours. Il existe  $i_u \in \llbracket 1, n \rrbracket$  tq  $u = L_{i_u}$ . Comme  $u \notin C_k$ ,  $i_u \notin \llbracket r_k, r_{k+1} \rrbracket$ . Si  $i_u < r_k$ , on pose  $W = \{L_i \mid i \in \llbracket 1, r_k \rrbracket\}$ . Ainsi,  $L_{i_u} \in W$  et  $\mathcal{B} = \emptyset$  car  $L_{i_k}$  est point de régénération. D'après le lemme, il n'existe aucun chemin reliant  $L_{i_u}$  et  $L_{i_k} \notin W$  donc  $\hat{C}_k$  n'est pas connexe. Autre cas... Ainsi  $C_k$  est maximal.
- ▷ Par l'absurde, on suppose que  $C_k$  n'est pas connexe. On peut donc dire qu'il existe  $i \in \llbracket r_k, r_{k+1} \rrbracket$  tel que  $L_{r_k} \not\leftrightarrow$  On considère alors  $i_0 = \min \{j \in \llbracket r_k, r_{k+1} \rrbracket \mid L_j \not\leftrightarrow L_{r_k}\}$ . Par minimalité de  $i_0$ ,  $\forall j \in \llbracket r_k, i_0 \rrbracket, L_{r_k} \leftrightarrow L_j$ . donc  $\forall j \in \llbracket r_k, i_0 \rrbracket, L_j \not\leftrightarrow L_{i_0}$  (sinon on aurait  $L_{i_0} \leftrightarrow L_j \leftrightarrow L_{r_k}$ ). De plus, puisque  $L_{r_k} \in R$ ,  $\mathcal{B}(\{L_j \mid j \in \llbracket 1, r_k \rrbracket\}) = \emptyset$  Donc d'après le lemme,  $\forall j \in \llbracket 1, r_k \rrbracket, L_{i_0} \not\leftrightarrow L_j$ . Donc  $L_{i_0} \notin \mathcal{B}(\{L_j \mid j \in \llbracket 1, i_0 \rrbracket\})$ . Par définition d'un parcours on a nécessairement  $\mathcal{B}(\dots) = \emptyset$  donc  $L_{i_0} \in R$  : Absurde, car  $L_{r_k}$  et  $L_{r_{k+1}}$  sont 2 points de régénération consécutifs.

Soit  $L = (L_i)_{i \in \llbracket 1, n \rrbracket}$  un parcours de  $G$ . Pour  $k \in \llbracket 0, n \rrbracket$ , on appelle **sommet ouvert à l'étape  $k$**  (dans  $L$ ) un sommet de  $O_k = \{L_j \mid j \in \llbracket 1, k \rrbracket \text{ et } \mathcal{V}(L_j) \not\subset \{L_i \mid i \in \llbracket 1, k \rrbracket\}\}$ .

$L$  est un **parcours en largeur** (resp. profondeur) de  $G$  ssi  $\forall k \in \llbracket 1, n \rrbracket$ ,  $L_k$  est un point de régénération ou bien  $L_k \in \mathcal{V}(L_{i_0})$  où  $i_0 = \min \{i \in \llbracket 1, n \rrbracket \mid L_i \in O_k\}$  (resp. max). Autrement, chaque sommet du parocurs est point de régénération ou bien voisin du premier (resp.dernier) sommet ouvert.

**Remarque :** Lorsque l'on s'intéresse à la forêt d'arborescence associée à un parcours en largeur / profondeur, on choisira comme paire d'un sommet  $L_k$  le premier / dernier sommet ouvert à l'étape  $k$ . Autrement dit, la forêt justifiera que le parcours est en largeur / profondeur.

**Exemple :** Un parcours en largeur du graphe ci-dessous est ABCDEFHG, et un parcours en profondeur de ACDEBFHG.



### 3 Algorithmes de parcours

#### 3.1 Détection de composantes connexes.

$G = (V, E)$  avec  $V = \llbracket 1, n \rrbracket$  un graphe non orienté. L'algorithme consiste à associer à chaque sommet du graphe le numéro de la composante connexe associée.

- $O$  contient les éléments de la composante connexe actuelle que l'on est en train de traiter (les éléments "Ouverts" afin de les étudier) ;
- $n_c$  contient le numéro de la composante connexe que l'on est en train d'explorer ;
- $F$  contient tous les sommets traités (les sommets **F**ermés).
- $T_{res}$  contient les numéros de parties de chaque sommet.

```

1  n <- nb_sommets(G)
2  O <- ensemble de sommets, initialement vide
3  F <- ensemble de sommets, initialement vide
4  T_res <- tableau indicé par  $\llbracket 1, n \rrbracket$ , initialisé à -1.
5  n_c <- 0
6  racine <- 1
7  T_res[racine] <- n_c
8  O.ajouter(racine)
9  i <- 0
10 Invariants:
11   - les éléments de F sont dans n_c composantes connexes différentes;
12   - il y a i éléments différents de -1 dans T; ce sont les éléments de F;
13   - les éléments portant le même numéro dans T sont accessibles entre eux.
14 Tant que i < n:
15   Si O est vide:
16     racine <- numéro d'un sommet S tel que T_res[s] = -1
17     n_c <- n_c + 1
18     O.ajouter(racine)
19     T_res[racine] <- n_c
20   u <- un sommet extraît de O
21   Pour tout v voisin de u
22     Si T_res[v] = -1:
23       O.ajouter(v)
24       T_res[v] <- n_c
25   F.ajouter(u)
26   i++
27 Renvoyer T_res

```

### 3.2 Détection de graphe biparti.

**Algorithme pour décider si un graphe est biparti :** Considérons un graphe  $G$ . On parcourt récursivement (à partir d'un point de régénération, puis ses successeurs...) les composantes connexes du graphe en associant à chaque sommet une couleur (ici représentée par 0 ou 1). Si jamais il y a conflit entre deux associations de couleur (un sommet qui s'est vu associer la couleur 0 auquel on essaierait d'associer la couleur 1 par exemple), le graphe n'est pas biparti. Si par contre on arrive à parcourir tout le graphe sans conflit, le graphe est bien biparti.

Dans le cas où  $G = (V, E)$  avec  $V = \llbracket 1, n \rrbracket$  :

- $T$  est un tableau contenant la couleur du sommet  $i$ , -1 si aucune couleur ne lui a été assignée;
- $O$  désigne l'ensemble des sommets déjà colorés desquels on colorie les voisins.

```
1  n <- nb_sommets(G)
2  T <- tableau d'entiers indicé par  $\llbracket 1, n \rrbracket$  initialisé à -1
3  O <- {}
4  couleur <- 0
5  T[1] <- couleur
6  Invariants:
7      - le sous graphe induit par G sur  $V' = \{i \in \llbracket 1, n \rrbracket \mid T[i] \neq -1\}$  est biparti;
8      - il y a  $i$  éléments différents de -1 dans T.
9  Pour i allant de 1 à n:
10     Si O est vide:
11         u <- élément de  $\llbracket 1, n \rrbracket$  tel que  $T[u] = -1$ 
12         T[u] = couleur
13     Sinon:
14         u <- élément extrait de O
15         couleur = 1 - T[u]
16     Pour v voisin de u:
17         Si T[v] = -1
18             T[v] <- couleur
19             ajouter v à O
20         Sinon:
21             Si T[v] ≠ couleur:
22                 Renvoyer Faux
23 Renvoyer Vrai
```

### 3.3 Plus court chemin en nombre d'arcs.

Dans un graphe non-orienté, on appelle **distance entre les sommets  $a$  et  $b$** , notée  $\text{dist}(a, b)$ , la longueur minimale d'une chaîne reliant  $a$  et  $b$  (et  $+\infty$  s'il n'en existe pas). On généralise cette notion aux graphes orientés, mais il se s'agit alors plus vraiment d'une distance : la distance de  $a$  à  $b$  n'est pas forcément égale à la distance de  $b$  à  $a$  !

Pour cet algorithme, on considère  $G = (S, A)$  graphe orienté avec  $S = \llbracket 1, n \rrbracket$  et  $(a, b) \in S^2$ . On cherche la distance en nombre d'arcs de  $a$  à  $b$ . On parcourt récursivement le graphe à partir de  $a$ , et dès que l'on rencontre un sommet, on lui assigne sa distance à  $a$  - alors minimale - jusqu'à atteindre  $b$ . Pour garder une trace du chemin optimal, on stocke pour chaque élément son prédécesseur.

- $O_1$  représente la liste des sommets à traiter,  $O_2$  la liste de leurs successeurs.
- **prof** est le "numéro de la génération" : le premier sommet dont on part représente la génération 0, ses successeurs la génération 1, et ainsi de suite.
- $D$  contient les distances des différents sommets du graphe par rapport au point  $a$ .
- $P[i]$  est le numéro du point précédent le sommet numéro  $i$  dans l'éventuel plus court chemin.

- $F$  est l'ensemble des sommets traités par l'algorithme dont les successeurs ont aussi été traités.

```

1  n <- nb_sommets(G)
2  O1 <- pile d'entiers initialisée vide
3  O2 <- pile d'entiers initialisée vide
4  F <- ensemble initialisé vide
5  D <- tableau d'entiers indicé par  $\llbracket 1, n \rrbracket$  initialisé à  $+\infty$ 
6  P <- tableau d'entiers indicé par  $\llbracket 1, n \rrbracket$  initialisé à  $+\infty$ 
7  prof <- 0
8  O1.ajouter(a)
9  D[a] <- 0
10 P[a] <- a
11
12 Invariants:
13   - pour tout sommet  $f$  de  $F$ ,  $d(a, f) = D[a] \leq \text{prof}$  ;
14   -  $F$  est inclus dans la composante connexe  $W$  contenant  $a$ ;
15   -  $O_2$  est la bordure de  $F$ .
16 Tant que  $O_1$  et  $O_2$  ne sont pas vides:
17   Si  $O_1$  est vide:
18     Transvaser  $O_2$  dans  $O_1$ 
19     prof = prof + 1
20   u <- O1.extraire_sommet
21   Pour v successeur de u:
22     Si  $D[v] = +\infty$ :
23       D[v] <- D[u] + 1
24       P[v] <- u
25       O2.ajouter(v)
26   F.ajouter(u)
27 Renvoyer D[b]
```

## 4 Plus court chemin

Pour un graphe orienté  $G = (V, E)$ , on dit que  $w$  est une **pondération des arêtes** ssi  $w \in \mathcal{F}(E, \mathbb{R})$ .

On dit alors que  $G$  est un **graphe pondéré**.

Soit  $H = (S, A, w)$  un graphe orienté. Si  $\gamma = (\gamma_i)_{i \in \llbracket 0, n \rrbracket}$  est une chaîne ou un chemin de  $G$ , sa longueur est  $\text{long}(\gamma) = \sum_{i=0}^{n+1} w(\gamma_i, \gamma_{i+1})$ .

On définit alors la distance entre deux sommets de  $S$  par

$$d(a, b) = \min \{ \text{long}(\gamma) \mid \gamma \text{ est un chemin reliant } a \text{ et } b \}$$

**Remarque :** Ces définitions se généralisent naturellement aux graphes non-orientés.

### 4.1 Algorithme de Dijkstra

Considérons  $G = (S, A, w)$  orienté, pondéré **positivement**, où  $S = \llbracket 1, n \rrbracket$ ,  $s \in S$  un sommet de départ.

•

```

1  n <- nombre de sommets de G
2  D <- tableau indicé par  $\llbracket 1, n \rrbracket$ , initialisé à  $+\infty$ 
3  P <- tableau indicé par  $\llbracket 1, n \rrbracket$ , initialisé à -1.
4  D[s] = 0
```

```

5 | P[S] = s
6 |
7 | O <- {0}
8 | F <- ∅
9 | Tant que O est non vide :
10 |     u <- extraire de O l'élément minimisant D // (u = argmin {D[i] | i ∈ O})
11 |     Pour v ∈ succ(u):
12 |         Si D[v] = +∞:
13 |             O.ajouter(v)
14 |         Si D[u] + w(u, v) < D[v]:
15 |             D[v] <- D[u] + w(u, v)
16 |             P[v] <- u
17 |     F.ajouter(u)
18 | renvoyer D

```

**Propriété :** Soit  $G = (S, A, w)$  un graphe orienté pondéré positivement. Soit  $s \in S$ . Le tableau  $D$  retourné par l'algorithme ci-dessus est tel que  $\forall u \in S, D[u] = d(s, u)$ .

## 4.2 Démonstration

Pour  $X \subset S$ , et  $(u, v) \in S^2$ , on note  $C_X(u, v)$  l'ensemble des chemins élémentaires de  $u$  à  $v$  dans  $G$ , dont tous les sommets sauf éventuellement  $v$  sont dans  $X$  et  $d_X(u, v)$  la longueur minimale d'un chemin dans  $C_X(u, v)$ .

**Invariant :** Soit  $(G, S)$  une entrée de l'algorithme. On admet que l'appel termine. Soit  $K$  le nombre de tours de boucle tant que effectués. Pour  $k \in \llbracket 0, K \rrbracket$ , on note  $O_k, F_k, D_k$  l'état des ensembles  $O$  et  $F$  et le tableau  $D$  à la fin du  $k$ -ième tour de boucle.

- (a) Pour tout  $u \in S$ , pour tout  $k \in \llbracket 0, K \rrbracket, D_k[u] = d_{D_k} s, u$ ;
- (b)  $\forall u \in F_k, D_k[u] = d(s, u)$ ;
- (c)  $O_k \cap F_k \neq \emptyset$  et  $O_k \cup F_k = \{i \in \llbracket 1, n \rrbracket \mid D[i] < +\infty\}$ .

▷ On procède par récurrence finie :

▷ Au rang  $k = 0$ , on a :

- $F_0 \neq \emptyset$  donc (b) est vraie ;
- On a aussi  $O_k \cap F_k = \emptyset$ . De plus  $O_0 = \{s\}$  donc  $O_0 \cup F_0 = \{s\}$ , or  $s$  est le seul sommet  $i$  pour lequel  $D_0[i] \neq +\infty$ , donc on a bien (c).
- $\forall u \in S, C_{F_0}(s, u) = \{(s)\}$  si  $u = s$ ,  $\emptyset$  sinon. donc  $d_{F_0}(s, u) = 0$  si  $u = s$ ,  $+\infty$  sinon. On a donc bien (a)

▷ Soit  $k \in \llbracket 0, K \rrbracket$ . On suppose  $P_k$ . On veut montrer  $P_{k+1}$ . Soit  $x \in O_k$  le sommet extrait de  $O$  à l'étape  $k + 1$ . On a alors  $F_{k+1} = F_k \cup x$ . De plus,  $D_k[x] = \min \{D_k[i] \mid i \in O_k\}$ .  $D_{k+1}[u] = D_k[u]$  si  $u \notin \text{succ}(x)$ ,  $\min \{D_k x + w(x, u), D_k[u]\}$  sinon.  $F_k \subset F_{k+1}$ , donc  $\forall u \in S, D_{F_k}(s, u) \geq d_{F_{k+1}}(s, u)$ .  $O_{k+1} = (O_k \cup \{u \in \text{succ}(x) \mid D_k[u] = +\infty\}) \setminus \{x\}$

- On remarque que nécessairement  $D_k[s] = 0, s \in F_k$  (car  $F_1 = \{s\}$  et  $(F_i)$  croissante), donc  $d_{F_k}(s, s) \leq \text{long}(s) = 0$ , soit  $d_{F_{k+1}}(s, s) = 0 = D_{k+1}[s]$ .
- Soit  $u \in S \setminus s$ . Soit  $c = (c_i)_{i \in \llbracket 0, l \rrbracket} \in C_{F_{k+1}}(s, u)$  (nécessairement  $c_0 = s$  et  $c_l = u$ ). On note alors  $p = c_{l-1}$ . Aini  $c = \overset{\tilde{c}}{s..p} u$ .

**Cas 1 :**  $p \neq x$  : Alors  $p \in F_{k+1} \setminus \{x\} = F_k$ . [On remarque que  $\forall y \in F_k, d_{F_k}(s, y) = D_k[y] = d(s, y)$ . Donc  $\forall y \in F_k, \exists c \in C_{F_k}(s, y)$  de longueur  $(d, x, y)$ .] ( $\theta$ ) Ici en particulier, il existe  $\tilde{b} \in C_{F_k}(s, p)$  de longueur  $d(s, p)$ . Ainsi, le chemin  $d = \tilde{b}.u \in C_{F_k}(s, u)$ , et  $\text{long}(b) =$

$\text{long}(\tilde{b}) + w(p, u) = d(s, p) + w(p, u) \leq \underbrace{\text{long}(\tilde{c}) + w(p, u)}_{\text{long}(c)}$ . donc  $d_{F_k}(s, u) \leq \text{long}(b) \leq \text{long}(c)$  d'où

$\text{long}(c) \geq D_k[u]$ .

**Cas 2 :**  $p = x$  : ...  $\text{long}(c) \geq D_k[x] + w(x, u)$ .