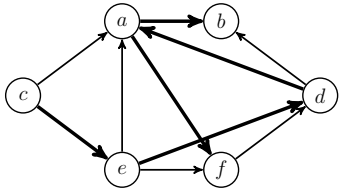
Chapitre 3

Algorithmes de plus court chemin

I Définitions

Définition 1 (racine, arborescence) *Un sommet x est* racine *d'un graphe G, ssi pour tout sommet $y \neq x$, il existe au moins un chemin dans G allant de x à y.*
Une arborescence *est un arbre qui admet une racine.*

Exemple 20 *Le graphe suivant admet une arborescence de racine 3.*



Définition 2 (poids d'un chemin, chemin ij-minimal) *Soient $G = (X = [\![1, n]\!], U)$ un graphe orienté pondéré, c'est-à-dire que chaque arc (i, j) est affecté d'un poids p_{ij}, le* poids *(ou "coût") d'un chemin c dans G est égal à la somme du poids de ses arcs sera noté $p(c)$).*
Un chemin ij-minimal *est un chemin de i à j de poids minimum parmi tous les chemins de i à j.*

Par abus, on dira souvent "plus court chemin" et on parlera de "longueur" du chemin au lieu de parler de chemin de poids minimum.
On suppose que x est racine de G, et on s'intéresse à déterminer pour tout sommet $y \neq x$, un plus court chemin de x à y, c'est à dire un chemin de poids minimum, ou encore chemin xy-minimal.

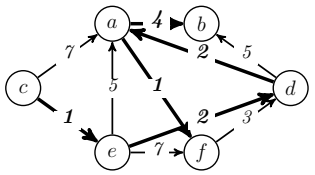
Propriété 1 *Si les $p_{i,j}$ sont positifs alors tout chemin ij-minimal est soit élémentaire (ne passant pas deux fois par le même sommet) soit contient un circuit de poids nul.*

II Graphe orienté pondéré positivement : algorithme de Dijkstra

Les chemins mis en évidence par les algorithmes proposés ci-dessous sont élémentaires.
L'algorithme de Dijkstra (1971) [6] (aussi appelé Moore-Dijkstra) permet d'obtenir une arborescence partielle $H = (X, V)$ de G de racine i_0 (c'est-à-dire que H est un graphe sans cycle de racine i_0 (et donc connexe) avec $V \subseteq U$) dans laquelle, pour tout sommet i de $X \setminus \{i_0\}$, l'unique chemin allant de i_0 à i est un chemin i_0i-minimal de G.

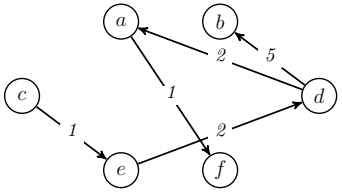
Algorithme de Dijkstra : cf. feuille algo.

Exemple 21 *Soit $G = (X = [\![1, 6]\!], U)$ le graphe orienté suivant de racine ③.*



Sommets \ *Sommets sélectionnés*	*a*	*b*	*c*	*d*	*e*	*f*
Initialisation	∞	∞	⓪	∞	∞	∞
c	7	∞	/	∞	①	∞
e	6	∞	/	③	/	8
d	⑤	⑧	/	/	/	8
a	/	8	/	/	/	⑥
f	/	8	/	/	/	/
b	/	/	/	/	/	/

Normalement, on doit noter les arcs v_j à chaque fois que l'on change le λ_j mais sur un tableau on peut s'en passer : Les nombres entourés dans la colonne d'un sommet j correspondent à la valeur définitive λ_j (distance du chemin minimal de i_0, ici ③, à j). On a entouré ce nombre sur la ligne où cette valeur apparaît pour la première fois pour ce sommet. Ainsi ce nombre se trouve sur la ligne du sommet i à partir duquel la marque λ_j a été ajustée à cette valeur. Cette convention permet de connaître l'arc dont l'extrémité terminale est j dans l'arborescence de chemin i_0j-minimaux.
Ainsi le sommet ① a obtenu sa valeur définitive à partir du sommet ④, ce qui signifie que l'arc (4,1) appartient à l'arborescence. On obtient finalement l'arborescence suivante :



Complexité de l'algorithme $o(n^2)$. Chaque étape de l'algorithme de Dijkstra nous fournit la valeur définitive d'un sommet, et on le sait. Si on s'intéresse aux chemins de valeur minimale entre le sommet de départ A et un sommet particulier B, on peut arrêter le calcul dès que B est calculé, même si tous les sommets ne sont pas calculés. La stratégie est donc **gloutonne**.

Remarque *Dans le cas où i_0 n'est pas racine de G l'algorithme se termine avec des sommets dont la marque reste infinie, ces sommets ne sont pas des descendants de i_0. Dans le cas d'un graphe non orienté, l'algorithme fonctionne en considérant qu'une arête équivaut à avoir un arc dans chaque sens. On obtient donc des* chaînes *$i_0 j$-minimales.*

III Matrice de routage

Définition 3 (matrice des poids des chemins ij-minimaux) *La matrice des poids des chemins ij-minimaux associée à un graphe orienté valué d'ordre n est une matrice de dimension $n \times n$ telle que*

$$\lambda_{ij} = \textit{longueur du chemin } ij\textit{-minimal}$$

Dans le cas non-orienté on considère les chaînes ij-minimales.

Définition 4 (matrice de routage) *La matrice de routage associée à un graphe orienté valué d'ordre n est une matrice de dimension $n \times n$ telle que*

$$m_{ij} = \textit{sommet adjacent à } i \textit{ sur le chemin } ij\textit{-minimal}$$

Dans le cas non-orienté on considère les chaînes ij-minimales.

Exemple 22 *L'arborescence est une arborescence de chemin 3i-minimaux dans le graphe, elle permet de remplir les matrices des longueurs et de routage de la façon suivante :*

longueurs

$$\begin{pmatrix} \times & ? & ? & ? & ? & 1 \\ ? & \times & ? & ? & ? & ? \\ 5 & 8 & \times & 3 & 1 & 6 \\ 2 & 5 & ? & \times & ? & 3 \\ 4 & 7 & ? & 2 & \times & 5 \\ ? & ? & ? & ? & ? & \times \end{pmatrix}$$

routage

$$\begin{pmatrix} \times & ? & ? & ? & ? & 6 \\ ? & \times & ? & ? & ? & ? \\ 5 & 5 & \times & 5 & 5 & 5 \\ 1 & 2 & ? & \times & ? & 1 \\ 4 & 4 & ? & 4 & \times & 4 \\ ? & ? & ? & ? & ? & \times \end{pmatrix}$$

longueurs cas non-orienté

$$\begin{pmatrix} \times & ? & 5 & 2 & 4 & 1 \\ ? & \times & 8 & 5 & 7 & ? \\ 5 & 8 & \times & 3 & 1 & 6 \\ 2 & 5 & 3 & \times & 2 & 3 \\ 4 & 7 & 1 & 2 & \times & 5 \\ 1 & ? & 7 & 3 & 5 & \times \end{pmatrix}$$

routage cas non-orienté

$$\begin{pmatrix} \times & ? & 4 & 4 & 4 & 6 \\ ? & \times & 4 & 4 & 4 & ? \\ 5 & 5 & \times & 5 & 5 & 5 \\ 1 & 2 & 5 & \times & 5 & 1 \\ 4 & 4 & 3 & 4 & \times & 4 \\ 1 & ? & 1 & 1 & 1 & \times \end{pmatrix}$$

Dans le cas non-orienté, la matrice des longueurs est symétrique mais pas la matrice de routage.

IV L'algorithme de Bellman

Théorème 2 (Principe d'optimalité de Bellman) *Si le chemin minimum de A à B passe par le sommet C alors la portion de A à C est elle-même minimale.*

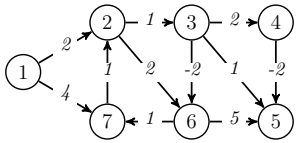
L'algorithme de Bellman-Kalaba 1958 calcule la longueur d'un plus court chemin à partir d'un sommet i_0 d'un graphe valué quelconque.
Le principe de l'algorithme est de considérer d'abord les chemins de 1 arc (longueur λ_1) entre i_0 et tous les autres sommets puis 2 arcs (longueur λ_2) et ainsi de suite.
Pour ce faire,

- il se base sur le principe d'optimalité (donc pour trouver un plus court chemin de k arcs, de i_0 vers i on compare tous les chemins de $k-1$ arcs de i_0 vers un prédécesseur de i + longueur de l'arc (prédé,i) et on prend le meilleur.
- il utilise aussi l'astuce d'ajouter un arc fictif (i_0, i_0) de longueur 0 afin que l'ensemble des chemins de i_0 vers i ayant $k-1$ arcs dans le graphe avec l'arc fictif, correspondent dans le vrai graphe à tous les chemins de i_0 vers i possédant entre 0 et k-1 arcs. (Ce qui permet à l'item précédent d'être sûr d'avoir pu comparer tous les meilleurs chemins possédant entre 0 et k-1 arcs (donc choisir le meilleur parmi (les chemins entre i_0 et i qui passe par un autre sommet et le chemin direct (arc) entre i_0 et i)).

Si le graphe ne contient pas de circuit de longueur négative il existe un plus court chemin qui est élémentaire qui a donc au plus n-1 arcs. Si G n'a pas de circuit de longueur négative $\lambda_{n-1}(j)$ est la longueur du plus court chemin de i_0 à j.

Algorithme de Bellman-Kalaba : cf. feuille algo.

Exemple 23 *Soit $G = (X = [\![1, 7]\!], U)$ le graphe orienté suivant, on désire connaître les plus courts chemins d'origine ① :*



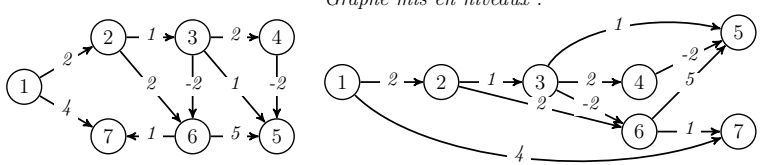
k	*Sommets*	*1*	*2*	*3*	*4*	*5*	*6*	*7*
0	λ_0	*0*	∞	∞	∞	∞	∞	∞
1	λ_1	*0*	*2*	∞	∞	∞	∞	*4*
2	λ_2	*0*	*2*	*3*	∞	∞	*4*	*4*
3	λ_3	*0*	*2*	*3*	*5*	*4*	*1*	*4*
4	λ_4	*0*	*2*	*3*	*5*	*3*	*1*	*2*
5	λ_5	*0*	*2*	*3*	*5*	*3*	*1*	*2*
STOP								

La complexité de l'algorithme de Bellman-Kalaba est de l'ordre de n^3 puisqu'elle est en $o(nm)$: au pire, on regarde tous les prédecesseurs de tous les sommets (m arcs), et ceci n fois. Lorsque le graphe est sans circuit, on peut diminuer grandement sa complexité. L'algorithme de Bellman pour les graphes *sans circuit* se sert des niveaux pour calculer les *plus courts chemins* en une seule passe sa complexité est en $o(m)$.

Algorithme de Bellman pour les graphes sans circuit : cf. feuille algo.

Exemple 24 *Soit $G = (X = [\![1, 7]\!], U)$ le graphe orienté sans circuit suivant, on désire connaître les plus courts chemins d'origine* ①

Graphe mis en niveaux :



On obtient donc successivement, $\lambda(1) = 0$, $\lambda(2) = 2$, $\lambda(3) = 3$, $\lambda(4) = 5$, $\lambda(6) = min(\lambda(2) + p_{26}, \lambda(3) + p_{36}) = 1$, $\lambda(5) = 3$, $\lambda(7) = 2$.

Ici, grace à la mise à niveau, et comme pour l'algorithme de Dijkstra, chaque étape nous fournit la valeur définitive d'un sommet, et on le sait. Si on s'intéresse aux chemins de valeur minimale entre le sommet de départ A et un sommet particulier B, on peut arrêter le calcul dès que B est calculé, même si tous les sommets ne sont pas calculés. On dit que la stratégie est **gloutonne**, ce qui signifie que certains résultats ne seront plus remis en question (un glouton est quelqu'un qui avale et ne peut pas revenir en arrière, c'est-à-dire ne peut pas remettre en cause ce qu'il vient de faire).

Remarque *L'algorithme de Bellman-Kalaba permet aussi de calculer les plus longs chemins, il admet les valuations négatives et les circuits.*

Il en existe une variante un peu plus efficace appelée Bellman-Ford ou Bellman ou Ford (cf. feuille d'algo).
Les graphes de jeux sont énormes puisque les sommets sont les différents états du jeux et les arcs les passage possibles entre deux états en jouant un coup. La recherche de plus courts chemins d'un sommet vers un sommet intéressant revient à rechercher une stratégie gagnante mais vu l'énormité des graphes, on ne génère pas forcément tout le graphe. C'est pourquoi on utilise plutot des algorithmes de type A^* [9] qui est une variante de Dijkstra utilisant une fonction appelée heuristique qui estime la valeur d'un plus court chemin d'un sommet donné à la solution et une fonction qui donne les successeurs d'un sommet.
En résumé les principaux algorithmes de recherche de plus court chemin sont

- Bellman-Kalaba pour les graphes pondérés de poids quelconque (cet algorithme est aussi utilisable pour trouver les chemins maximaux contrairement à Dijkstra).
- et Bellman pour les graphes sans-circuit.
- Dijkstra qui est plus efficace que Bellman-Kalaba mais ne s'applique qu'à des graphes pondérés positivement,

- l'algorithme de Moore est une variante non-gloutonne de Dijkstra permettant de traiter des graphes pondérés quelconques.
- Si l'on désire connaître tous les plus courts chemins possibles à partir de tous les sommets du graphe vers tous les autres sommets, il existe l'algorithme de Roy-Floyd-Warshall (1959) qui travaille sur une matrice d'adjacence. Sa complexité étant en $o(n^3)$ il est en général moins couteux en mémoire d'utiliser Dijkstra (ou ses variantes utilisant une structure spéciale pour stocker les sommets non-encore explorés en $o(n.log(n))$) en considérant chaque sommet successivement comme source du graphe.
- Si la pondération des arcs est uniforme alors le plus court chemin est le plus court en nombre d'arcs et un algorithme d'exploration en largeur d'abord est le plus efficace $o(m)$.

Remarque *Les algorithmes de plus courts/longs chemins peuvent s'utiliser sur des graphes non-orientés en considérant qu'une arête représente deux arcs de sens opposés avec le même coût sur les deux arcs que sur l'arête.*

V Problèmes d'ordonnancement

Résoudre un problème d'ordonnancement consiste à déterminer un calendrier d'exécution des différentes tâches d'un projet (ou *ordonnancement*) qui minimise la durée de ce projet.

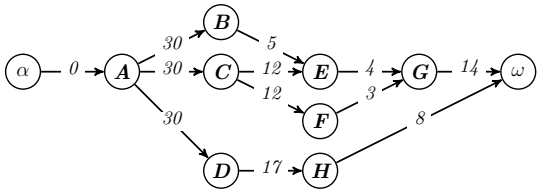
Exemple 25 *La réalisation d'un projet nécessite l'exécution de huit tâches A, B, C, D, E, F, G, H. Les durées de ces tâches et le contraintes de postériorité auxquelles elles sont soumises sont données dans le tableau suivant :*

Tâches	*A*	*B*	*C*	*D*	*E*	*F*	*G*	*H*
Durées	*30*	*5*	*12*	*17*	*4*	*3*	*14*	*8*
Tâches pré-requises		*A*	*A*	*A*	*B,C*	*C*	*E,F*	*D*

Définition 5 (début au plus tôt, au plus tard, tâche critique, marge totale) *La* date de début au plus tôt d'une tâche, *c'est la date minimum à laquelle peut commencer une tâche.*
La date de début au plus tard *est la date maximale à laquelle la tâche peut commencer si l'on veut que la durée minimale du projet ne soit pas modifiée.*
Une tâche est critique *ssi tout retard dans l'exécution de cette tâche allonge d'autant la durée minimale du projet.*
La marge totale *MT est le temps maximal dont cette tache peut être différée ou allongée sans retarder la fin des travaux.*
On appelle marge libre *ML d'une tâche le délai pouvant être accordé au commencement d'une tâche sans modifier les marges totales des tâches suivantes.*

Il existe deux méthodes de modélisation de ce problème : la méthode potentiels-tâches et la méthode potentiels-étapes (PERT [1]). La modélisation par un graphe potentiels-tâches (tâches = sommets) est assez intuitive et n'admet qu'un seul graphe pour modéliser un problème donné. Cette modélisation potentiels-tâches était moins utilisée que la modélisation PERT. En effet, la modélisation PERT (tâches= arcs) était préférée par les praticiens car elle utilise des graphes comportant moins d'arcs. Cependant la modélisation PERT ne donne pas nécessairement un graphe unique.
Le calcul des débuts au plus tôt t_v d'une tâche v revient à appliquer Bellman sans-circuit pour calculer les plus longs chemins : $\boxed{t_v = \max_{x\in\Gamma^-(v)}(t_x + \text{duree}(x))}$.
Le calcul des débuts au plus tard t^*_v se ramène aussi à un Bellman sans-circuit sur le graphe transposé (en partant de la fin du projet) : $\boxed{t^*_v = \min_{x\in\Gamma^+(v)} t^*_x - \text{duree}(v)}$.
La marge totale MT se calcule en faisant $\boxed{MT_v = t^*_v - t_v}$.
La marge libre $\boxed{ml_v = \min_{x\in\Gamma^+(v)} t_x - f_v}$ où f_v est la fin au plus tôt de v c'est-à-dire $f_v = t_v + \text{duree}(v)$.

Exemple 26



Résultats numériques :

Tâches v	α	*A*	*B*	*C*	*D*	*E*	*F*	*G*	*H*	ω
début au plus tôt t_v	*0*	*0*	*30*	*30*	*30*	*42*	*42*	*46*	*47*	*60*
début au plus tard t^*_v	*0*	*0*	*37*	*30*	*35*	*42*	*43*	*46*	*52*	*60*
fin au plus tot f_v	*0*	*30*	*35*	*42*	*47*	*46*	*45*	*60*	*55*	*60*
marges totales MT_v	*0*	*0*	*7*	*0*	*5*	*0*	*1*	*0*	*5*	*0*
marges libres ml_v	*0*	*0*	*7*	*0*	*0*	*0*	*1*	*0*	*5*	*0*

Les tâches critiques A, C, E, G sont caractérisées par l'égalité entre leur date de début au plus tôt et leur date de début au plus tard ($t^*_v - t_v = 0$).
D a une marge libre de 0 alors qu'il a une marge total de 5, car un retard dans l'exécution de D entrainerait une diminution dans la marge de H.

1. 1956. Program evaluation and review technik (technique d'évaluation et de remise à jour de programmes/projets)