

TP Informatique 04

Exercice 1

L'algorithme de Dijkstra réalise, par minimisations et actualisations successives, la recherche d'un plus court chemin dans un graphe orienté ou non orienté valué positivement. On note s_0 le sommet source et $d[u]$ la longueur du plus court chemin de s_0 à un sommet u découvert à un certain stade de l'algorithme. Le principe de l'algorithme est le suivant :

- on initialise $d[s_0] = 0$ et $d[u] = \infty$ pour tout $u \in S \setminus \{s_0\}$;
- tant qu'il y a des sommets « non visités » :
 - on détermine un sommet u non visité tel que $d[u]$ est minimal ;
 - le sommet u est considéré comme « visité » ;
 - pour chaque sommet s non visité et successeur de u , on effectue l'opération de *relâchement* :

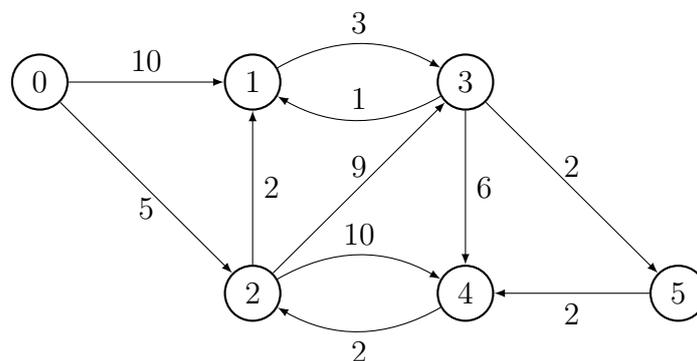
$$\text{si } d[s] > d[u] + \nu(u, s), \text{ alors } d[s] \leftarrow d[u] + \nu(u, s)$$

autrement dit : si pour aller à s depuis s_0 , il est plus court de passer par u , alors on le fait.

On utilisera les dictionnaires suivants :

- **cout** : pour s un sommet, **cout**[s] est la valeur de $d[s]$;
- **predec** : pour s un sommet, **predec**[s] est son prédécesseur dans le plus court chemin depuis s_0 ;
- **djvu** : pour s un sommet, **djvu**[s] est un booléen qui dit si le sommet a été déjà vu ou pas.

1. Écrire une fonction `dijkstra(S,A,s0)` d'arguments S la liste des sommets, A la liste des arêtes valuées, s_0 un sommet et qui renvoie les dictionnaires **cout**, **predec** des plus courts chemins depuis s_0 vers chaque sommet.
2. Tester la fonction `dijkstra` sur le graphe fourni en exemple dans le fichier `TP04_EX01_std` depuis le sommet 0.



Exercice 2

L'algorithme $A\star$ est un algorithme de recherche d'un plus court chemin dans un graphe $G = (S, A)$ orienté valué positivement, variante de l'algorithme de Dijkstra utilisant une *heuristique*.

On s'intéresse à la recherche d'un plus court chemin d'un sommet *deb* à un sommet *fin*, sommets qu'on suppose fortement connectés. Si les sommets sont repérés par leurs coordonnées dans le plan euclidien \mathbb{R}^2 , il est raisonnable de privilégier la recherche de chemin en s'efforçant d'aller dans la direction de *fin* et donc de réduire la distance euclidienne à *fin*. C'est précisément le rôle joué par la fonction heuristique h qu'on choisit, à savoir $h : s \in S \mapsto d_2(s, fin)$ où d_2 est la fonction distance dans \mathbb{R}^2 .

Le principe de l'algorithme $A\star$ consiste à considérer, dans l'algorithme de Dijkstra, pour $u \in S$, non plus la quantité $d[u]$ mais $d[u] + h(u)$, quantité qu'on note $c[u]$ comme *coût* depuis u . On effectue :

- on initialise $d[deb] \leftarrow 0$, $c[deb] \leftarrow h(deb)$, $d[u] \leftarrow \infty$ et $c[u] \leftarrow \infty$ pour tout $u \in S \setminus \{deb\}$;
- tant qu'on n'a pas visité le sommet *fin* :
 - on détermine un sommet u non visité tel que $c[u]$ est minimal ;
 - le sommet u est considéré comme « visité » ;
 - pour chaque sommet s non visité et successeur de u , on effectue l'opération de *relâchement* :
 - si $d[s] > d[u] + \nu(u, s)$, alors $d[s] \leftarrow d[u] + \nu(u, s)$ et $c[s] \leftarrow d[s] + h(s)$
 - autrement dit : si pour aller à s depuis *deb*, il coûte « moins cher » de passer par u , alors on le fait.

On utilisera les dictionnaires suivants :

- **dist** : pour s un sommet, **dist**[s] est la valeur de $d[s]$;
- **cout** : pour s un sommet, **cout**[s] est la valeur de $d[s] + h(s)$;
- **predec** : pour s un sommet, **predec**[s] est son prédécesseur dans le plus court chemin depuis s_0 ;
- **djvu** : pour s un sommet, **sommet**[s] est un booléen qui dit si le sommet a été déjà vu ou pas.

1. Écrire une fonction **d2(A,B)** d'arguments A et B des couples de nombres et qui renvoie la distance euclidienne entre les points A et B .
2. Écrire une fonction **Astar_deb_fin(S,A,deb,fin)** implémentant l'algorithme $A\star$ d'arguments S la liste des sommets, A le dictionnaire des arêtes valuées, *deb* le sommet de départ, *fin* le sommet de fin et qui renvoie les dictionnaires **cout**, **predec** et le nombre **nb**, les dictionnaires étant ceux associés à un plus court chemin entre *deb* et *fin* et **nb** le nombre de « minimisation » effectuées par le code.
3. Écrire une fonction **dijkstra_deb_fin(S,A,deb,fin)** implémentant l'algorithme de Dijkstra d'arguments S la liste des sommets, A le dictionnaire des arêtes valuées, *deb* le sommet de départ, *fin* le sommet de fin et qui renvoie les dictionnaires **cout**, **predec** et le nombre **nb**, les dictionnaires étant ceux associés à un plus court chemin entre *deb* et *fin* et **nb** le nombre de « minimisation » effectuées par le code.
4. Comparer les deux fonctions sur le scénario proposé dans le fichier **TP04_EX02_std**. On pourra tester les fonctions en faisant varier le point de départ. Les cases de la pièce considérée sont les sommets du graphe et les arêtes, toutes de valuation égale à 1, sont les transitions d'une case vers celle du haut, du bas, de la gauche, de la droite en tenant compte des bords et des murs.

Exercice 3

L'algorithme de Bellman-Ford réalise, lui-aussi, par relâchements successifs, la recherche d'un plus court chemin dans un graphe orienté, pas forcément valué positivement mais sans circuit absorbant depuis un sommet source s_0 . Contrairement à ce qui était fait dans l'algorithme de Dijkstra, chaque arc va être relâché plusieurs fois. On relâche une première fois tous les arcs ce qui permet de déterminer tous les plus courts chemins de longueur 1 depuis s_0 . On relâche ensuite une deuxième fois ce qui permet de déterminer tous les plus courts chemins de longueur 2 depuis s_0 . On répète le processus $n - 1$ fois avec n le nombre de sommets du graphe. Pour tout sommet, il existe un plus court chemin depuis s_0 de longueur $n - 1$; en effet, considérant un plus court chemin depuis s_0 vers un sommet, s'il n'est pas élémentaire, alors on peut en extraire un chemin élémentaire de même coût puisque le circuit éliminé sera de coût positif et donc nul, du fait du caractère « le plus court ». On conserve les notations introduites pour l'algorithme de Dijkstra. Le principe est le suivant :

- on initialise $d[s_0] = 0$ et $d[u] = \infty$ pour tout $u \in S \setminus \{s_0\}$;
- Pour tout k de 1 à $n - 1$, on répète :
 - Pour tout arc $(x, y) \in A$, on effectue l'opération de *relâchement* :
si $d[y] > d[x] + \nu(x, y)$, alors $d[y] \leftarrow d[x] + \nu(x, y)$

On utilisera les dictionnaires `cout` et `predec`.

1. Écrire une fonction `BF(S, A, s0)` d'arguments `S` la liste des sommets, `A` la liste des arêtes valuées, `s0` un sommet et qui renvoie les dictionnaires `cout`, `predec` des plus courts chemins depuis `s0` vers chaque sommet.
2. Tester la fonction `BF` sur les graphes orientés fourni en exemple dans le fichier `TP04_EX03_std` depuis le sommet 0 et comparer avec les solutions fournis par `dijkstra`.
3. Écrire une version améliorée de `BF` qui détecte l'absence de relâchement de tous les arcs lors d'un parcours puis tester cette version. On rappelle que l'instruction `break` permet de casser une boucle `for`.
4. Modifier le graphe 2 (par exemple en changeant l'arc de poids 9 en -7) puis écrire une nouvelle version de `BF` qui détecte la présence d'un circuit absorbant et la tester sur le graphe modifié.

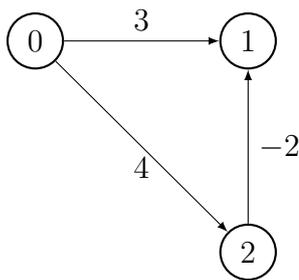


FIGURE 1 – Graphe 1

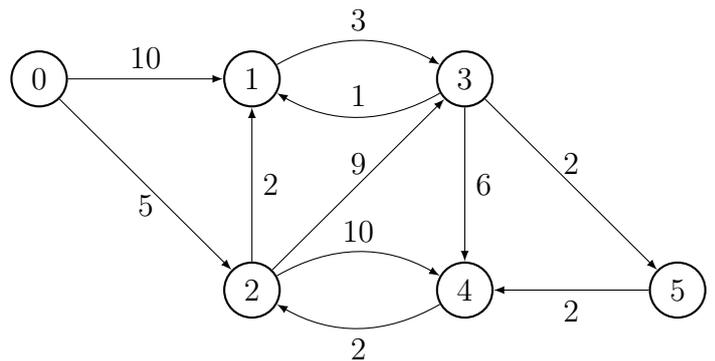


FIGURE 2 – Graphe 2