ISM MPSI/PCSI, Informatique
Année 2024/2025

TP Informatique 33

On complètera les fichiers `TP33_EX01_std.py` et `TP33_EX02_std.py` en ligne sur le site de la classe.

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 $\mathrm{d}[u]$ 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 $\mathrm{d}[s_0] = 0$ et $\mathrm{d}[u] = \infty$ pour tout $u \in \mathrm{S} \setminus \{s_0\}$;
- tant qu'il y a des sommets « non visités » :
  - on détermine un sommet u non visité tel que $\mathrm{d}[u]$ 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* :

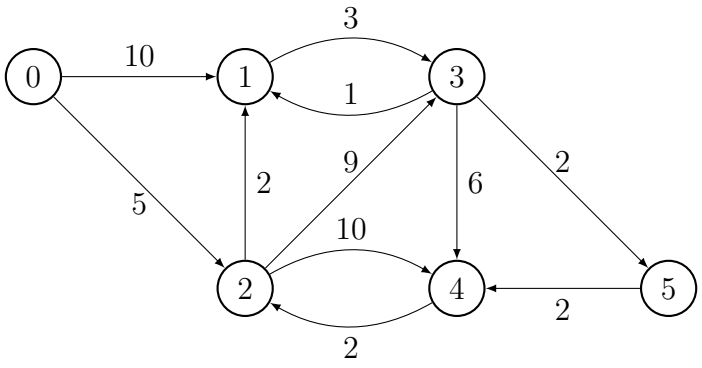
$$\text{si } \mathrm{d}[s] > \mathrm{d}[u] + \nu(u,s), \text{ alors } \mathrm{d}[s] \leftarrow \mathrm{d}[u] + \nu(u,s)$$

  autrement dit : si pour aller à `s` depuis `s0`, 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 $\mathrm{d}[s]$;
- `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, `s0` un sommet et qui renvoie les dictionnaires `cout`, `predec` des plus courts chemins depuis `s0` vers chaque sommet atteignable.
2. Tester la fonction `dijkstra` sur le graphe fourni en exemple dans le fichier `TP33_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 $\mathrm{G} = (\mathrm{S}, \mathrm{A})$ 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 $h : s \in \mathrm{S} \mapsto \mathrm{d}_2(s, fin)$ où d_2 est la fonction distance euclidienne dans $\mathbb{R}^2$.

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

- on initialise $\mathrm{d}[deb] \leftarrow 0$, $c[deb] \leftarrow h(deb)$, $\mathrm{d}[u] \leftarrow \infty$ et $c[u] \leftarrow \infty$ pour tout $u \in \mathrm{S} \setminus \{deb\}$;
- 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 $\mathrm{d}[s] > \mathrm{d}[u] + \nu(u, s)$, alors $\mathrm{d}[s] \leftarrow \mathrm{d}[u] + \nu(u, s)$ et $c[s] \leftarrow \mathrm{d}[s] + h(s)$

    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 $\mathrm{d}[s]$;
- `cout` : pour `s` un sommet, `cout[s]` est la valeur de $\mathrm{d}[s] + h(s)$;
- `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 `TP33_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.