TP 3 : Calculs de plus courts chemins dans un graphe

lionel.rieg@ens-lyon.fr

Dans ce TP, nous allons voir deux algorithmes de calcul de plus courts chemins à partir d'un point dans un graphe, qu'on appellera la *source*. Ces algorithmes (ou leurs variantes) sont souvent utilisés pour obtenir des trajet entre deux points (par exemple avec un GPS).

1 Calcul des plus courts chemins dans un graphe non pondéré

Pour un graphe non pondéré, les distances par rapport à la source sont données par un parcours en largeur du graphe depuis la source.

1. Donner le pseudo-code du parcours en largeur. Quelles en sont les entrées/sorties ?
2. Dans cet algorithme, on a souvent besoin de trouver tous les voisins d'un sommet. La représentation du graphe vue aux TP précédents n'est pas la plus efficace pour cela. Quelle autre représentation choisir ?
3. Écrire une fonction de conversion entre ces deux représentations. On pensera à utiliser le type `vector<int>` de C++.
4. Implémenter le parcours en largeur. Pour représenter la file, on prendra le type `queue<int>` de C++ , qui autorise les opérations suivantes :
   - `push(e)` enfile l'élément e à la fin de la file,
   - `pop()` retire le premier élément de la file,
   - `front()` renvoie la valeur du premier élément de la file,
   - `empty()` renvoie le booléen `true` si la file est vide et `false` sinon.

2 Calcul des plus courts chemins dans un graphe pondéré

Dans cette partie, nous allons calculer les distances à la source de tous les points d'un graphe pondéré par une fonction `poids` de l'ensemble des arêtes dans $\mathbb{R}^+$. On utilise l'*algorithme de Dijkstra* dont voici le pseudo-code :

```
Initialiser tous les sommets comme étant "non marqué".
Affecter la valeur +∞ à tous les distances D
D(s) := 0
Tant Qu'il existe un sommet non marqué
    Choisir le sommet a non marqué de plus petite distance D
    Marquer a
    Pour chaque sommet b non marqué voisin de a
       D(b) := min(D(b), D(a) + poids(a, b))
    Fin Pour
Fin Tant Que
```

1. Pourquoi faut-il que la valuation soit positive ? Donner un exemple de problème avec des poids négatifs.
2. Y a-t-il besoin de considérer les sommets de distance $+\infty$ dans la boucle ?
3. Quelles sont les opérations que l'on a besoin de faire sur les sommets ?
4. La structure de données qui permet d'effectuer efficacement ces opérations s'appelle un tas. En C++, on peut utiliser le type `priority_queue` pour cela. Attention cependant, il permet d'extraire le maximum et non le minimum. Comment corriger cela ?
5. Quelle est la complexité de cet algorithme ?

6. L'implémenter.
7. Dans le cas, où l'on ne s'intéresse qu'à la distance entre deux sommets précis (la source et le cible) et non pas à toutes les distances depuis la source, comment peut-on améliorer l'algorithme ?

3 Calcul de toutes les paires de distances

Lorsque que l'on veut obtenir les distances entre tous les points, plutôt que de réutiliser l'algorithme de Dijkstra pour chaque point du graphe (ce qui donne une complexité en $O(nm \log n)$), on préfère utiliser un algorithme de programmation dynamique qui a en outre l'avantage de ne pas être sensible aux poids négatifs.

Pour obtenir le chemin optimal du sommet $i$au sommet j, l'idée est de limiter les sommets intermédiaires par lesquels on peut passer : on définit $d_{ij}^{(k)}$, le coût minimum d'un chemin de i à j dont les sommets intermédiaires sont dans l'intervalle $\{1, \ldots, k\}$. Il vérifie l'équation de récurrence suivante :

$$\begin{cases} d_{ij}^{(0)} = \text{poids}(i,j) \\ d_{ij}^{(k)} = \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}) \quad \forall k \geq 1 \end{cases}$$

1. Quelle est la complexité en temps de cet algorithme ? Et en espace ?
2. L'implémenter.