Algorithmique et Programmation

Projet : plus courts chemins à origine unique par l'algorithme de Dijkstra

Ecole normale supérieure
Département d'informatique
td-algo@di.ens.fr

2014-2015

Pour ce projet, nous cherchons à trouver le chemin le plus court entre deux sommets d'un graphe oriénté et pondéré dont lse poids représentent les longueurs d'arc.

L'algorithme de Dijkstra nous permet de calculer le plus chemin entre une source s et tous les autres sommets du graphe. Cet algorithme emploie à la base une *file de priorité* comme structure de donnée. Cette structure permet d'inserer des éléments pondérés, de retrouver l'élément de poids minimum et d'éffacer ce élément de poids minimum de la file.

La file de priorité peut être implémentée de différentes façons. Pour cet exercise vous utiliserez des *Tas de Fibonacci*.

1 Algorithme de Dijkstra

L'algorithme de Dijkstra recherche la distance à partir de la source s de manière gloutonne. Il trouve les sommets du graphe orienté $G = (S, A, w)$ donné en entré (ainsi que le chemin le plus court vers ces sommets) en order croissant de leur distance de s. Notons pas $d_s(t)$ la distance (du plus court chemin) de s à t.

Au départ, seulement la distance de s vers s est connue (cette distance $d_s(s)$ est 0).

À chaque étape, l'algorithme choisi un arc (u, v) d'un sommet u dont la distance est connue à un sommet v dont la distance (de s) est inconnue. Cet arc est choisi à minimiser la distance totale de s à v si (u, v) est le dernier arc utilisé. C'est-à-dire l'arc (u, v) minimisant $d_s(u) + w(u, v)$ est choisi et ce chemin utilisant le plus court chemin de s à u et l'arc (u, v) est déclaré comme (un) plus court de s à v. Par conséquent, on inscrit $d_s(u) + w(u, v)$ pour $d_s(v)$.

Et on continue choissisant un nouvel arc pour la prochaine étape et ainsi de suite.

Quelques astuces sont utile pour trouver rapidement l'arc (u, v) minimisant $d_s(u) + w(u, v)$ à chaque étape. Nous maintenons une liste d'arrêtes non-vus provenant d'un sommet dont la distance est connue. Si une arrête (minimisante) entre deux sommets à distances connue est choisie, nous marquons cette arrête comme vue et passont simplement à l'étape suivante (nous pouvons prouver qu'elle nous servira pas dans un chemin le plus court). Finalement, plutôt que de maintenir une liste d'arcs, nous utilisons une file de priorité d'arcs non-vus pour pouvoir rapidement trouve l'élément minimisant la distance totale.

2 File de priorité

Une file de priorité est simplement un ensemble d'objets pondérés (dans notre cas les objets seront des arcs du graphe dont le poids est la distance totale de s si on utilise cet arc comme dernier arc du chemin) qui permet les opérations suivantes:

- INSÉRER(objet,poid) qui ajoute un objet avec poid.
- EXTRAIRE-MIN() qui enlève l'objet de poids minimum de l'ensemble et retourne cet objet avec son poid.

3 Pseudo-code

3.1 Algorithme de Dijkstra

Voir [1, §25.2]

Entrée:

1. Un graphe orienté $G = (S, A)$
2. Poids w sur les arcs A (i.e., $w : A \to \mathbb{R}^+$.
3. Un sommet source $s \in S$.

Sortie: Un tableau d_s indexé par S tel que $d_s[t]$ est la longueur du plus court chemin de s à t dans G.

```
1: function DIJKSTRA(G, w, s)
2:     Soit P une file de priorité vide.
3:     Soit d_s un tableau de taille |S| initialisé à «non définie»
4:     Soit Vus un ensemble vide de sommets.
5:     d_s[s] ← 0
6:     Ajouter v à Vus
7:     for all arc a sortant de s do
8:         longueur ← d_s[s] + w_a
9:         INSÉRER(P, a, longueur)
10:    end for
11:    while P est non-vide. do
12:        (u, v), poid ← EXTRAIRE-MIN(P)
13:        if v ∉ Vus then
14:            d_s(v) ← poid
15:            Ajouter v à Vus
16:            for all arc a sortant de v do
17:                longueur ← d_s[u] + w_a
18:                INSÉRER(P, a, longueur)
19:            end for
20:        end if
21:    end while
22: end function
```

4 Temps de calcul

Si la file de priorité est implantée à l'aide de la structure de données des tas de Fibonacci (voir [1, §21]), les fonctions `Extraire-Min` et $P(v) = \delta_s(v)$ (qui ne peut que diminuer la priorité d'un nœud) se font respectivement en $\mathcal{O}(\log |S|)$ et $\mathcal{O}(1)$. On obtient un coût total $\mathcal{O}(|A| + |S| \log |S|)$ pour l'algorithme de Dijkstra.

5 Travail demandé

Vous écrirez une fonction C qui calcule le plus court chemin à origine unique par l'algorithme de Dijsktra. L'argument de la fonction sera un graphe de taille arbitraire sous forme de listes d'adjacence (efficace pour représenter les graphes peu denses) ainsi qu'un nœud s. Les poids seront de type `int`. La sortie sur écran sera, pour chaque $t \in S$, la liste ordonnée des sommets formant un chemin de poids minimum entre s et t ainsi que son poids, ou $+\infty$ si t n'est pas accessible depuis s.

1. Implémenter une fonction qui test si un graphe est connexe.
2. Implémentation l'algorithme que Dijkstra avec une file de priorité naïve représenté par un tableau où le EXTRAIRE-MIN regarde tous les poids pour trouver l'élément min (et ensuite déplace tous les élément après le min vers l'avant).
3. Implémenter une file de priorité utilisant les tas de Fibonacci.
4. Implémentation l'algorithme que Dijkstra avec une file de priorité représenté par des tas de Fibonacci.
5. Comparer les différentes implémentation pour des valeurs variés de $|A|$ et $|S|$ pour vérifier expérimentalement le coût théorique (attention au coût de la génération du graphe et de la sortie sur écran).

6 Bonus

Téléchargez les donnés géographique de OpenStreetMap (`www.openstreetmap.org`) pour la région Parisienne et trouver les chemins les plus courts de l'ENS. Quel est la distance de l'ENS vers l'intersection du boulevard Brune et l'avenue Jean Moulin? Vous pouvez supposez que la Terre est localement plate.

Téléchargez l'emplacement des stations de bus, métros et trains de la rubique Open Data du site de la RATP: `http://data.ratp.fr/fr/les-donnees.html`. Quel est l'entrée de métro la plus proche de l'ENS pour chaque ligne? La station de train? De bus?

Donnez aussi le chemin le plus court vers ces stations. Comparez vos résultats avec ceux des programmes de navigation (par exemple, Google maps).

6.1 Format des données

Une des difficultés du bonus est d'extraîre le graphe pondéré des données réels. Alors que nous avons qu'il doit y avoir un graphe quelque part, ces données servent à des buts diverses et contiennent donc beaucoup plus d'informaton. Pour commencer, regardé la description donné pour OpenStreetMap

`http://wiki.openstreetmap.org/wiki/OSM_XML`

et pour la RATP

`http://data.ratp.fr/?eID=ics_od_datastoredownload&file=65`

Bien nous indiquer quel interpretation vous avez pris des données (quels sont les noeuds et arêtes du graphe).

6.2 Liens directs

Pour le plan de Paris, peut être téléchargé directement de cette page en choisissant une région.

`http://www.openstreetmap.org/export#map=12/48.8588/2.3468`

ou bien vous pouvez obtenir la ville entière sur un site alternatif:

`http://download.geofabrik.de/europe/france/ile-de-france.html`

avec lien direct pour Paris

`http://download.geofabrik.de/europe/france/ile-de-france-latest.osm.bz2`

Pour la RATP, les données sont à

`http://data.ratp.fr/fr/les-donnees.html`

Voici un lien direct vers le ficher ZIP des données pour les arrêts

`http://data.ratp.fr/?eID=ics_od_datastoredownload&file=64`

et la note explicatif des données en PDF

`http://data.ratp.fr/?eID=ics_od_datastoredownload&file=65`

References

[1] Charles E Leiserson, Ronald L Rivest, Clifford Stein, and Thomas H Cormen. *Introduction to algorithms*. The MIT press, 2001.