IFT339 - Aut 2014 travail 2 (partie 2)

Appliquer l'algorithme de Dijkstra à de gros fichiers…

Dans ce travail, nous allons utiliser les données d'un graphe selon le format décrit dans la partie 1, soit un fichier binaire où on peut aller lire directement l'information concernant un nœud sans avoir à lire les nœuds précédents.

Votre programme devra faire les opérations suivantes:

1. lire sur cin (normalement le clavier) un nom de fichier contenant un graphe
2. répéter les trois opérations suivantes jusqu'à la "fin" de cin
    a. lire deux numéros de nœuds
    b. déterminer le chemin optimal du premier au second de ces nœuds
    c. afficher le chemin et son poids

Par exemple, avec les données suivantes, le programme devrait lire dans le fichier exemple.ibin et déterminer le chemin optimal du nœud numéro 9 au nœud numéro 0, puis le chemin de 2 à 5:

```
exemple.ibin
9 0
2 5
```

Remise du travail

Dans le turnin pour ce travail, remettez trois fichiers:

- tp2.cpp est votre programme principal qui lit au clavier le nom du fichier et détermine les chemins optimaux dans le graphe correspondant.
- graphe.h et graphe.cpp contiennent toutes les classes dont vous avez besoin

```
turnin –c ift339 –p tp2 tp2.cpp graphe.h graphe.cpp
```

N'utilisez pas d'autres noms, et ne mettez pas de majuscules!

Au verso une brève description de l'algorithme de Dijkstra. Attention, il ne faut pas l'utiliser tel quel. Il est décrit de façon un peu théorique. Il ne faut avoir AUCUNE opération qui prend un temps O(n). Il faut donc repenser l'algorithme en conséquence. On en parlera en classe.

D'après le plan de cours, ce travail devrait être récupéré automatiquement le 3 novembre. C'est sans doute un peu trop tôt. On discutera en classe du meilleur moment pour cette remise, compte tenu du troisième travail à faire, ainsi que de vos autres travaux.

Jean Goulet

Le plus court chemin dans un graphe

Après des études de physique théorique, **Edsger W. Dijkstra** s'engage dès 1955 dans le domaine de l'informatique alors naissante, dont il est l'un des pionniers les plus éclairés. Parmi ses contributions se trouve un algorithme de calcul du plus court chemin dans les graphes, connu sous le nom d'algorithme de Dijkstra. C'est un algorithme d'une grande simplicité, mais très efficace. Pour plus de détails, voir :

http://fr.wikipedia.org/wiki/Edsger_Dijkstra

Description succincte de l'algorithme de Dijkstra

On dispose d'un graphe orienté valué. On dispose de l'ensemble T des noeuds du graphe, et des deux noeuds particuliers DEP et ARR. On peut connaître le poids d'un arc entre les sommets s1 et s2 avec l'énoncé lesPoids[s1][s2]. « s » étant un sommet quelconque, on construit un conteneur PRED[s] qui nous donne le prédécesseur d'un noeud donné dans le chemin optimal allant de DEP à ARR et un conteneur TOTAL[s] qui nous donne le poids total du chemin optimal allant de DEP à s. Le conteneur lesPoids est une structure de données de type "dictionnaire", ou une mise en correspondance entre deux types d'objets (un map de la STL).

Au départ de l'algorithme, PRED est vide. RESULTAT est un chemin (une liste de noeuds) vide. TOTAL[DEP]=0 et TOTAL[s] vaut $+\infty$ pour tout autre s. L'algorithme se déroule en deux phases : détermination des prédécesseurs optimaux, puis construction du chemin optimal de DEP à ARR.

Phase 1 (détermination des prédecesseurs) :

```
Tant que T n'est pas vide
    choisir un sommet us de T pour lequel TOTAL[us] est minimal
    si us==ARR c'est fini! (sortir de la boucle)
    Retirer us de T
    Pour chaque successeur s de us,
        calculer D=TOTAL[us]+lesPoids[us][s]
        si D<TOTAL[s] alors PRED[s]=us et TOTAL[s]=D
```

Phase 2 (construction du chemin) :

```
Si us!=ARR, il n'y a pas de chemin de DEP à ARR
Sinon répéter
    empiler us dans RESULTAT (insérer au début)
    us=PRED[us]
tant que us!=""
```

RESULTAT contient le chemin de poids minimal de DEP à ARR.