Université de Reims Champagne-Ardenne
Département de Mathématiques, Mécanique et Informatique

Pierre Delisle

Travaux pratiques 3

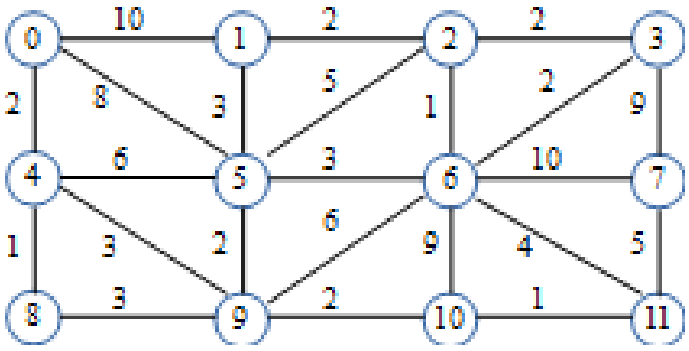
Tas – Algorithmes de tri

Les objectifs de ce TP sont :

- D'implémenter une structure de tas qui sera utilisée dans de futurs TPs ;
- De programmer et de comparer deux algorithmes de tri différents : le tri par insertion et le tri par tas ;
- De programmer des parties de code permettant de traiter les graphes valués de futurs TPs.

Exercice 1 – Importation des données et tri par insertion

Dans de futurs TPs, nous utiliserons ce graphe d'exemple :



Ce graphe correspond au fichier **graphe5.txt** que vous pouvez télécharger sur la page Moodle du cours. Ce fichier a la même structure que celui des TP 1 et 2 (**graphe2.txt**), à la différence que le graphe est maintenant valué et qu'une troisième valeur a ainsi été ajoutée à la définition de chaque arête : son poids.

1) Modifiez vos codes d'importation de graphes du TP 1 pour intégrer les poids des arêtes dans vos listes d'adjacences et votre matrice d'adjacences.

Dans de futurs TP, il vous sera demandé de manipuler les arêtes d'un graphe, chacune d'elles étant représentée par un sommet d'origine, un sommet d'extrémité et un poids.

2) Définissez une structure de données appropriée pour représenter une arête.

3) Écrivez un code permettant de récupérer toutes les arêtes du graphe à partir de ses listes d'adjacences ou de sa matrice d'adjacences, et de les stocker dans un tableau d'arêtes.

On veut maintenant ordonner les arêtes du graphe par ordre croissant de poids en triant le tableau d'arêtes. Voici un pseudo-code de l'algorithme de tri par insertion qui prend un tableau *t*[1..*t.longueur*] et le trie en $O(n^2)$:

```
TRI-INSERTION(t)
    pour j = 2 à t.longueur
        clé = t[j]
        i = j – 1
        tant que i > 0 et t[i] > clé
            t[i+1] = t[i]
            i = i – 1
        t[i+1] = clé
```

4) Écrivez une fonction **tri_insertion** (par exemple en définissant les fichiers **tri.h** et **tri.c**) qui trie le tableau d'arêtes créé précédemment à l'aide de l'algorithme de tri par insertion. Affichez le contenu du tableau avant et après le tri pour vérifier que les arêtes ont effectivement été triées par la fonction.

Exercice 2 – Tas et tri par tas

Comme nous l'avons vu en cours et en TD, le tri par tas peut trier un tableau en O (n lg n).

1) Créez les fichiers **tas.h** et **tas.c** et implémentez-y une structure de tas d'arêtes ainsi que les procédures de tas suivantes :

- **parent**, **gauche** et **droite** : permettent de connaitre l'indice de l'élément dans le tas correspondant respectivement au parent, au fils gauche et au fils droit d'un élément d'indice spécifique ;
- **entasser_max** : entasse l'élément d'indice *i* afin de rétablir la propriété de tas ;
- **initialiser_tas/construire_tas_max** : initialise un tas max à partir d'un tableau de sorte à ce que ses éléments soient structurés sous forme de tas max ;
- **detruire_tas** : détruit un tas en libérant ses ressources mémoire.

2) Écrivez une fonction **tri_par_tas** (par exemple en l'ajoutant à vos fichiers **tri.h** et **tri.c**) qui trie le tableau d'arêtes créé précédemment à l'aide de l'algorithme de tri par tas. Affichez le contenu du tableau avant et après le tri pour vérifier que les arêtes ont effectivement été triées par la fonction.

3) Comparez le temps d'exécution de vos deux algorithmes de tri :

a) Écrivez un code permettant de générer un tableau de n arêtes aléatoires et faites une copie de ce tableau dans un deuxième tableau ;
b) Estimez le temps d'exécution des deux algorithmes de tri avec n = {10, 100, 1000, 10 000, 100 000, 1 000 000, …} ;
c) Écrivez un code permettant de calculer, en secondes, le temps d'exécution de chaque algorithme de tri ;
d) Exécutez chaque algorithme de tri sur un tableau distinct et comparez les temps d'exécution avec chaque valeur de n. Observe-t-on des différences de temps significatives ? Dans l'affirmative, à partir de quelle valeur de n la différence devient-elle observable ?