1 Manipulations simples des files de priorité

Les questions qui suivent montrent que les files de priorité généralisent les files et les piles simples et qu'elles permettent de réaliser facilement un algorithme de tri. Donc, dans toutes ces questions, seule la fonction de priorité doit être changée.

Question 1. Télécharger directement sur le site `www.lifl.fr/~boulier` ou par la commande suivante le module de file de priorité suivant :

```
wget --no-cache http://www.lifl.fr/~boulier/polycopies/SD/priorite.tgz
tar xzf priorite.tgz
```

Question 2. Compiler le programme et l'exécuter. Vérifier qu'il trie une suite de `double` par ordre croissant.

Question 3. Modifier le programme pour qu'il trie la suite de `double` par ordre décroissant.

Question 4. Modifier le programme (et éventuellement le type `struct element`) pour que la file de priorité se comporte comme une file ordinaire.

Question 5. Modifier le programme (et éventuellement le type `struct element`) pour que la file de priorité se comporte comme une pile ordinaire.

2 L'algorithme de Dijkstra

2.1 Principe

Il s'applique à des graphes orientés, valués positivement. Il calcule la longueur minimale des chemins partant de la racine A vers tous les autres sommets du graphe, accessibles à partir de A.

Chaque sommet est muni d'un *potentiel*, qui vaut initialement $+\infty$, à l'exception du potentiel de A, qui est initialisé à 0. À la fin de l'exécution de l'algorithme, ce potentiel est égal à la longueur minimale recherchée. L'algorithme de Dijkstra s'implante efficacement avec une file de priorité. Plus le potentiel d'un sommet est petit, plus il est prioritaire. L'algorithme en pseudo-code est donné Figure 1.

```
begin
  Mettre le potentiel de A à 0, tous les autres à +∞
  Enfiler tous les sommets de T dans la file de priorité F
  while F n'est pas vide do
    X = defiler un sommet de F
    imprimer "dist. min. de A à X = le potentiel de X"
    Soit ℓ l'indice de X dans T
    for c variant de 0 à N − 1 do
      if B(ℓ, c) ≠ 0 then
        if potentiel(T[c]) > potentiel(X) + B(ℓ, c) then
          potentiel(T[c]) = potentiel(X) + B(ℓ, c)
          reporter un changement de priorité pour T[c] dans F (utiliser pour cela la
                    fonction changement_priorite et le champ indice de T[c])
        fi
      fi
    od
  od
end
```

FIGURE 1 – L'algorithme de Dijkstra en pseudo-code. Le graphe est implanté ainsi : il y a N sommets indicés de 0 à $N-1$, un tableau T contenant tous les sommets, une fonction $B(\ell, c)$ qui retourne la valeur de l'arc de $T[\ell]$ vers $T[c]$, ou 0 si cet arc n'existe pas.

2.2 Implantation des graphes

L'archive `graphe.tgz` (disponible sur le site du cours) contient les types nécessaires à une implantation minimaliste des graphes. Le fichier `sommet.h` contient la déclaration du type ci-dessous, inspiré du type `struct element`. Il permet de représenter un sommet d'un graphe :

```
struct sommet {
    char nom;
    int potentiel;    /* Pour Dijkstra */
    int indice;       /* Ce champ est utilisé par les files de priorité */
};
```

Les fichiers `graphe.h` et `graphe.c` contiennent un codage du graphe de la Figure 2. Voici les prototypes déclarés dans `graphe.h` :

```
#include "sommet.h"
/*
 * L'entier N est égal au nombre de sommets du graphe
 * Les sommets sont dans T, (indices de 0 à N-1).
 * La fonction B(l,c) implante la matrice d'adjacence du graphe.
 * Elle retourne le « poids » de l'arc  T[l] -> T[c], 0 si l'arc n'existe pas.
 * La fonction dessin dessine le graphe dans une fenêtre graphique.
```

```
 */
extern int N;
extern struct sommet T [];
extern int indice_dans_T (struct sommet*);
extern int B (int, int);
extern void dessin (void);
```

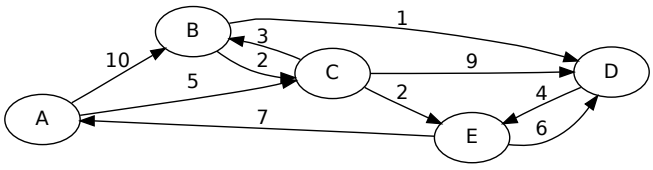

FIGURE 2 – Le graphe donné dans l'archive.

Question 6. Adapter le type `struct file_priorite_element` pour obtenir une file avec priorité de sommets, et donc un type `struct file_priorite_sommet`.

Question 7. Il manque une fonction dans le module `file_priorite_sommet`, pour implanter l'algorihtme de Dijkstra. Rajouter cette fonction.

Question 8. Programmer l'algorithme de Dijkstra dans `main.c`. À l'exécution, sur le graphe de la Figure 2 on doit avoir :

```
$ ./main
distance minimale de A -> A = 0
distance minimale de A -> C = 5
distance minimale de A -> E = 7
distance minimale de A -> B = 8
distance minimale de A -> D = 9
```

Question 9. Modifier `graphe.c` pour représenter le graphe de la Figure 3. Tester votre programme sur cet exemple.

2.3 Développements

Question 10. Plutôt que d'enfiler initialement tous les sommets, il est possible de n'enfiler que les sommets dont les potentiels sont différents de $+\infty$. Modifier l'algorithme en conséquence.

Question 11. Implanter la file de priorité dans un tableau redimensionnable.

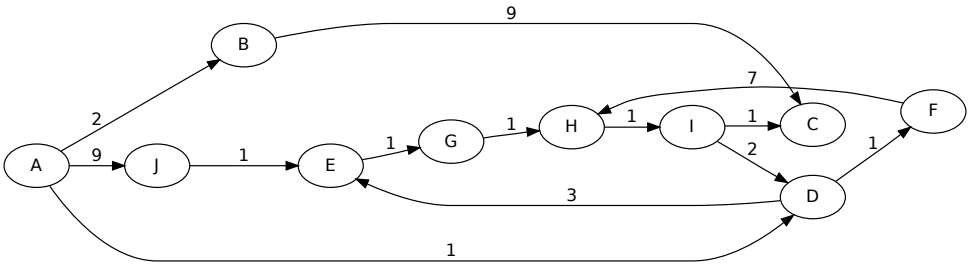

FIGURE 3 – Un autre graphe.

Question 12. On souhaite implanter la matrice d'adjacence en utilisant des listes chaînées. Proposer des types.

Question 13. Modifier l'implantation des files de priorités pour que les fonctions retournent le nombre de comparaisons de potentiels effectuées. Compter le nombre total de comparaisons de potentiels effectuées par l'algorithme de Dijkstra. Comparer avec une implantation naïve où, à chaque itération, le sommet de potentiel minimal est déterminé par un parcours complet du tableau T.

Mutatis mutandis, appliquer les deux méthodes sur des graphes aléatoires de plus en plus grands, fabriquer des fichiers de mesures et analyser les algorithmes à la façon du TP précédent.