Programmation impérative
Projet 2018 -2019

Algorithme de Dijkstra

DOUARE Juliette SAG Özgür

Pour réaliser ce projet, il nous a été demandé de créer trois modèles de classes : graphe, tas et tasid déclarées comme suit. Dans ce rapport nous expliciterons la structure de notre code et son utilisation, sans nous étendre sur la classe tas qui ne nous est pas utile pour l'implémentation de l'algorithme de Dijkstra.

```
template<class S, class P> class Graph {
    private:
        map<S,vector< pair<S , P> > > m;
    public:
        Graph();
        ~Graph();
        void ajouterSommet(S s);
        void ajouterArete(S s,const S a,const P p);
        void supprimerSommet(S s);
        void supprimerArete(S s, S a);
        void afficher();
        void dijkstra(pair<S,P> maPaire);
};
#endif
```

```
template<class T> class Tas {
    private:
        vector< T > v;
    public:
        Tas();
        Tas(vector<T> vec);
        ~Tas();
        T extraireMin();
        void ajouterElement(T ele);
        void reordonner();
        bool rechercherElement(T ele);
        void afficher();
};
#endif
```

```
template<class S, class P> class Tasid {
    private:
        map<int,pair <S,P> >m;
    public:
        Tasid();
        ~Tasid();
        pair<S,P> extraireMin();
        void ajouterElement(pair<S,P> ele); //Si ele dans le tas comparer le poids sinon l'ajouter
        void afficher();  // affichage  $
        int taille();
};
#endif
```

Classe Tas

Le modèle de classe « tas » comporte un constructeur, un destructeur et les fonctions nécessaires à son utilisation.

extraireMin — extrait le minimum en temps constant O(1)
→ Plutôt que d'appeler la méthode « reordonner » après extraction du minimum, nous avons fait le choix de l'appeler dans extraireMin dont la complexité passe à O(n).
ajouterElement — ajoute un élément en temps O(n log n).
rechercherElement — recherche un élément en temps linéaire O(n) qui renvoie vrai d'il est contenu dans le tas, sinon faux.
reordonner — permet de conserver la structure de tas.
afficher — affiche le tas sous forme de tableau.

Classe Tasid

Le modèle de classe « tasid » comporte un constructeur, un destructeur et les fonctions nécessaires à son utilisation. Tasid est un tas indicé, d'où sa signature map<int,pair <S,P> >m.

extraireMin — extrait le minimum en temps constant O(1)
ajouterElement — ajoute une paire<S,P> si le sommet de type S ne figure dans aucune paire du tas. Sinon, le poids P associé à S dans le tas est mis à jour s'il est inférieur.
afficher — affiche le tas sous forme de tableau.
taille — retourne la taille du tas.

Classe Graphe

Le modèle de classe « graphe » comporte un constructeur, un destructeur, les fonctions nécessaires à son utilisation et l'implémentation de l'algorithme de Dijkstra avec « tasid ».

ajouterSommet — ajoute un sommet de type S au graphe.
ajouterArete — ajoute l'arête **s – a** de poids p où s et a sont de type S. Si s et/ou a ne sont pas contenus dans le graphe, ils sont ajoutés.
afficher — affiche le graphe suivant l'exemple : [s]→[[s1,p1],[s2,p2]] où s est un sommet ayant deux voisins s1 et s2 dont les arêtes sont respectivement de poids p1 et p2.

taille — retourne la taille du tas.
lireFichier — créé un graphe à partir d'un fichier. Chaque ligne doit être de la forme A B P où A et B sont des sommets et P le poids de l'arête A — B. C'est dans le fichier qu'on détermine si le graphe est orienté (créer A — B et B — A s'il n'est pas orienté) et valué (mettre les poids à 1 s'il ne l'est pas).
→ CF test.txt qui reprend le graphe donné en exemple dans le sujet.
Dijkstra — retourne la liste des sommets parcourus dans l'ordre.

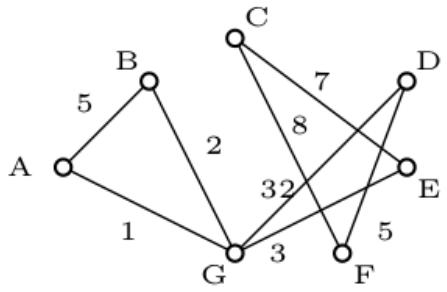
Utilisation du programme

Avant d'exécuter le programme, il faut s'assurer que le fichier contenant le graphe soit correctement écrit (ou simplement modifier le fichier test.txt) et qu'il se trouve dans le même dossier que le programme.
Pour l'utiliser, il faut taper la commande **./dijkstra nomFichier.txt sommetInitial** dans le terminal.
Par exemple, la commande **./dijkstra test.txt B** affiche :

```
juliette@juliette-G3-3779:~/Documents/C++/Projet.1.1$ ./dijkstra test.txt B

Votre graphe :
[A]->[[B,5],[G,1]]
[B]->[[A,5],[G,2]]
[C]->[[F,8],[E,7]]
[D]->[[G,32],[F,5]]
[E]->[[G,3],[C,7]]
[F]->[[C,8],[D,5]]
[G]->[[A,1],[B,2],[E,3],[D,32]]

Sommets choisis par l'algorithme de Dijkstra dans l'ordre:
B
G
A
E
C
F
D
juliette@juliette-G3-3779:~/Documents/C++/Projet.1.1$
```



Sommets choisis	A	B	C	D	E	F	G
B	5		+∞	+∞	+∞	+∞	2
B,G	3			34	5		
B,G,A							
B,G,A,E			12				
B,G,A,E,C						20	
B,G,A,E,C,F				25			
B,G,A,E,C,F,D							

test.txt reprend le graphe ci-dessus donné en exemple dans le sujet.