Projet — Algorithme de Dijkstra

Introduction *Le contexte.*

La recherche de plus courts chemins dans un graphe (orienté ou non) est un problème fondamental dans de nombreux domaines d'application. L'objectif du projet est d'implémenter un algorithme de recherche de plus courts chemins, appelé **algorithme de Dijkstra**. Cet algorithme utilise des arguments proches des algorithmes glouton ou de la programmation dynamique.

La classe `Graphe` *Un air de déjà-vu.*

Dans un premier temps, il est demandé de modifier légèrement la classe `Graphe` produite pour le TP5, et permettant de gérer des graphes (non-)orientés (non-)valués. Les méthodes créées sont suffisantes ; en revanche, il est nécessaire de permettre que les sommets (numérotés de 0 à $n-1$ dans tous les cas) puissent posséder des identifiants de n'importe quel type, et que les poids des arêtes puissent également être quelconques.

Un modèle de classe `Graphe`

$\mathcal{Q}_1$. Implémenter un modèle de classe `Graphe`, avec comme paramètres de modèle le type de sommets stockés dans le graphe, ainsi que le poids des arêtes.

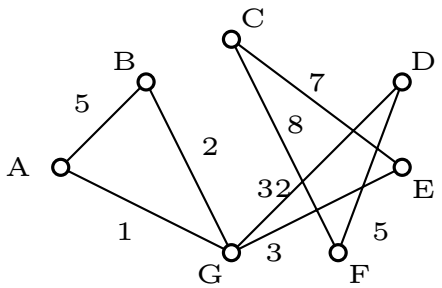
Algorithme de Dijkstra *[dɛikstra]*

L'algorithme de Dijkstra calcule les plus courts chemins d'un sommet v du graphe vers tous les autres sommets comme suit :

- Initialement, tous les sommets sont à distance $+\infty$ de v, sauf ses voisins u, où le poids de l'arête uv est utilisé ;
- Le sommet p le plus proche de v est choisi ;
- Pour tout sommet c non encore choisi par l'algorithme, si

$$d(v,p) + \omega(p,c) \leqslant d(v,c)$$

alors mettre à jour $d(v,c)$.
Les valeurs $d(u,v)$ et $\omega(u,v)$ représentent respectivement la distance de u à v stockée dans la table, et le poids de l'arête u,v.



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							

FIGURE 1 – Illustration de l'algorithm de Dijkstra en partant du sommet `B`. Seules les valeurs mises à jour sont affichées, et les cases grisées représentent le fait qu'un sommet choisi ne sera plus mis à jour.

Cet algorithme doit donc maintenir une table des distances, et sa complexité repose sur l'efficacité de recherche du minimum. Afin d'obtenir une bonne complexité, il est nécessaire d'utiliser une structure de données appropriée pour stocker les distances.

La classe `Tas`

A moment on the lips, forever on the heaps.

Pour être implémenté de manière efficace, l'algorithme de Dijkstra repose sur une structure de `tas minimum`, qui permet de retrouver le minimum en temps constant. Un tas est une structure de données qui stocke un ensemble d'éléments comparables et qui permet :

— d'extraire le minimum en temps constant ($O(1)$)
— d'ajouter un élément en temps logarithmique ($O(\log n)$)
— de rechercher un élément en temps linéaire ($O(n)$)

L'idée principale est de travailler avec un arbre binaire équilibré où chaque noeud contient une valeur plus grande que celle de son père. Les niveaux de l'arbre sont de plus complets (il n'y a aucun *trou*) sauf éventuellement le dernier, où les noeuds sont stockés de gauche à droite.

Représentation. Il n'est pas nécessaire d'utiliser un arbre pour représenter un tas : cette structure est effectivement particulièrement adaptée à une représentation par tableau (voir Figure 2).

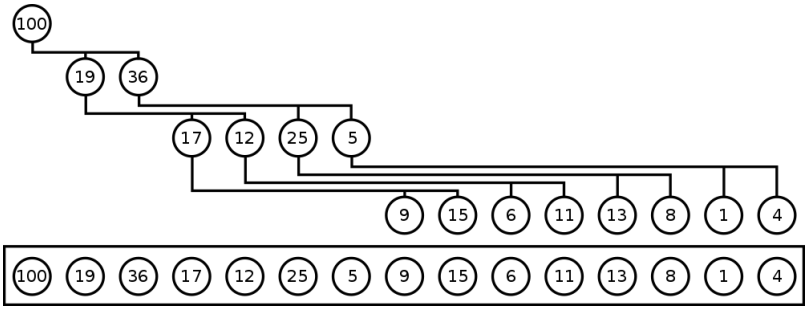


FIGURE 2 – Représentation d'un tax (maximum !) sous forme d'arbre et son tableau associé.

Un modèle de classe `Tas`

$\mathcal{Q}_2$ Impléter un modèle de classe (`template`) permettant la gestion d'un tas, en utilisant un paramètre de modèle pour le type de valeurs stockées dans le tas.

L'implémentation de l'algorithme de Dijkstra

[dɛikstra]

Afin d'obtenir une implémentation *efficace* de l'algorithme de Dijkstra, il est nécessaire de pouvoir intervenir sur n'importe quelle valeur contenue dans le tas, et demander au tas de la repositionner correctement (en la montant ou en la descendant). Les éléments contenus dans le tas ne seront donc plus des valeurs simples, mais une association valeur-identifiant. L'intérêt de l'identifiant est de permettre de *facilement* enregistrer la position d'une valeur dans le tas.

En particulier, les méthodes d'ajout et de suppression dans le tas doivent mettre à jour cette correspondance. La suppression d'un élément libèrera l'identifiant associé, et l'ajout d'un élément utilisera le premier identifiant libre.

Algorithme de Dijkstra

$\mathcal{Q}_3$. Implémenter un modèle de classe `Tas_id` permettant d'établir une correspondance entre les valeurs du tas et leur position dans ce dernier.

$\mathcal{Q}_4$. Implémenter l'algorithme de Dijkstra en utlisant la classe `Tas_id` précédente.