TP 3
Plus Court Chemin, Tas binaire et Arbre couvrant de poids min

Gilles Simonin, Sulian Le Bozec-Chiffoleau

Résumé

Votre travail lors de ce TP porte sur deux familles de problèmes essentiels en graphe. Dans un premier temps, la recherche des plus courts chemins depuis un sommet, en implémentant une version de l'algorithme de Dijkstra. Dans un second temps, les concepts d'arbre et de couverture. Une première partie portera sur la représentation efficace en tas binaire via un ensemble d'éléments ordonnés, puis nous vous demanderons d'étudier l'utilisation de cette représentation dans l'exécution de l'algorithme de PRIM.

1 Plus Court Chemin depuis un sommet

L'algorithme de Dijkstra est un des plus utilisé en recherche opérationnelle et dans le domaine du transport. Vous allez développer une version de cet algorithme pour la représentation de votre choix (matrice d'adjacence ou liste d'adjacence).

Question 1 *Placez vous dans la classe de votre choix et définissez une méthode Dijkstra prenant un sommet initial en entrée. L'algorithme est le suivant :*

Algorithm 1 Pseudocode de l'algorithme de Dijkstra

Require: Un graphe $G = (V, E)$, d'ordre n, un sommet $s \in V$, une fonction de poids $cout(x, y)$ pour tout arc/arête (x, y).
Ensure: Affichage final des distances et des prédécesseurs pour chaque sommet.

```
1: boolean mark[] = new boolean[n] ; int val[] = new int[n] ; Sommet pred[] = new Sommet[n] ;
2: for all v ∈ V do
3:    mark[v] ← faux ;  val[v] ← MaxInt/2 ; pred[v] ← null ;
4: end for
5: val[s] ← 0 ; pred[s] ← s ;
6: while il reste un sommet non marqué do
7:    int x ← 0 ;
8:    // Recherche du sommet x non marqué de val minimum :
9:    int min ← MaxInt/2 ;
10:   for all y allant de 0 à n − 1 do
11:      if mark[y] == faux et val[y] < min then
12:         x ← y ; min ← val[y] ;
13:      end if
14:   end for
15:   // Mise à jour des successeurs non fixés de x :
16:   if min < MaxInt/2 then
17:      mark[x] ← vrai ;
18:      for all y ∈ N+(x) do
19:         if mark[y] == faux et val[x] + cout(x, y) < val[y] then
20:            val[y] ← val[x] + cout(x,y) ; pred[y] ← x ;
21:         end if
22:      end for
23:   end if
24: end while
```

2 Arbre binaire et structures de données

Le *tas binaire* est une des structures de données les plus classique en informatique permettant de trier et stocker un ensemble d'éléments de manière efficace. Cette structure repose sur la classification des éléments à ordonner dans un *arbre binaire* avec la propriété que le dit arbre soit *complet* et que l'élément stocké à la racine de l'arbre soit plus petit ou égal (dans le cas tas binaire minimum) à tous les éléments stockés dans les sommets fils de la racine.

Nous vous proposons dans cette deuxième partie de TP d'implémenter cette structure. Vous ferez une première version naïve du tas binaire avec des valeurs entières aux noeuds avec la classe *BinaryHeap*, en veillant à bien décomposer votre code afin de pouvoir modifier aisément chaque étape. Vous développerez ensuite une version améliorée pour les graphes dans la classe *BinaryHeapEdge*, où chaque noeud contiendra une arête (utiliser la classe Edge pour représenter les arêtes).

Question 2 *Identifiez bien dans la classe "BinaryHeap" le tableau d'entiers "nodes"représentant la structure d'arbre binaire introduite dans le cours. Dans cette classe les sommets du tas binaire sont des entiers, le but est de maintenir ce tas binaire bien défini tout en ajoutant ou enlevant des sommets.*

Question 3 *Proposez une méthode* ***insert*** *d'un nouvel élément dans le tas binaire assurant que celui-ci soit correctement positionné. Pour cela, vous opérerez de la manière suivante :*

- *Ajouter l'élément dans la première feuille disponible de l'arbre ;*
- *Remonter cet élément jusqu'à rencontrer un parent qui soit plus petit ou égal. Pensez à utiliser la méthode "swap".*
- *Retourner* True *si l'insertion a bien été faite.*

Question 4 *Faites les insertions suivantes en faisant des tests dans la méthode "main" :* $4, 10, 8, 6, 3$. *Discutez de la complexité de l'opération d'insertion.*

Question 5 *Avant de travailler sur la méthode* remove*, proposez une implémentation de la fonction bien utile* ***getBestChildPos*** *qui retourne l'indice du fils ayant la plus petite valeur. Pour cela, vous opérerez de la manière suivante :*

- *Une fois passé le test d'un noeud feuille, si le noeud à l'index* src *a un seul fils, retourner l'index* $2 \times src + 1$ *;*
- *Sinon retourner l'index du fils ayant la plus petite valeur. Cela sera* $2 \times src + 1$ *ou* $2 \times src + 2$.

Question 6 *Proposez une méthode* remove *du plus petit élément dans le tas binaire. Pour cela, vous opérerez de la manière suivante :*

- *Permuter la racine de l'arbre avec la dernière feuille utilisée de l'arbre ;*
- *Descendre l'élément contenu dans la racine jusqu'à ce que tous les éléments contenus dans les fils du sommet courant soient plus grand ou égaux.*
- *Retourner l'élément retiré (penser à l'enlever du tas).*

Question 7 *Faites deux suppressions consécutives en faisant ces tests. Discutez de la complexité de l'opération d'insertion.*

Question 8 *Vous pouvez recopier ce que vous venez de faire dans la classe BinaryHeapEdge en utilisant des objets "Edge" au lieu d'entiers. Il vous faudra un peu adapter votre code mais les algorithmes restent les mêmes. On compare les arêtes par leur poids.*

3 Algorithme de Prim via la structure de tas binaire

Nous vous proposons ici de mettre en œuvre l'algorithme de **PRIM** introduit rapidement en cours.

Question 9 *Comment utiliseriez-vous la structure de tas binaire dans votre mise en œuvre de l'algorithme PRIM ?*

Question 10 *Vous allez créer une méthode* ***Prim*** *dans la classe de votre choix.*

Implémentez l'algorithme de Prim en utilisant la structure de tas binaire développé précédemment. Pour cela, vous opérerez de la manière suivante :

- *Ajouter un sommet initial dans la liste des noeuds visités ;*
- *Insérer dans le tas binaire chaque arête adjacente au sommet initial ;*
- *Retirer de manière itérative l'élément du tas binaire ayant la plus petite valeur pour l'ajouter dans la solution, ainsi que le sommet visité via l'arête sélectionnée. Avant de réitérer cette action, insérer dans le tas binaire les nouvelles arêtes adjacentes (ne créant pas de cycle) au sommet ajouté ;*
- *Retourner la solution finale.*

Question 11 *Pensez à tester votre algorithme dans la méthode main de votre classe.*