UNIVERSITÉ DE REIMS CHAMPAGNE-ARDENNE

LICENCE Informatique
Info0501 – 2020/2021

Travaux dirigés n° 3

Arbres couvrants de poids minimal

Exercice 1 (Préliminaire - Structure de tas)
1°) Montrez comment on construit un tas à partir du tableau $t = \{4, 1, 3, 2, 16, 9, 10, 14, 8, 7\}$.

2°) À partir de la représentation en tableau, définissez les fonctions suivantes :

a) `parent(`i`)`, qui retourne l'indice du parent d'un noeud, i étant l'indice du noeud dans le tableau ;

b) `gauche(`i`)`, qui retourne l'indice de l'enfant de gauche, i étant l'indice du noeud dans le tableau ;

c) `droite(`i`)`, qui retourne l'indice de l'enfant de droite, i étant l'indice du noeud dans le tableau.

3°) Donnez les propriétés d'un tas-max.

Exercice 2 (Préliminaire - Tri par tas)
1°) Rappelez le fonctionnement du tri par tas à partir du tableau $t = \{4, 1, 3, 2, 16, 9, 10, 14, 8, 7\}$.

2°) Définissez les procédures suivantes applicables à un tas-max :

a) `entasser-max(`t, i`)` : prend en paramètre un tableau t et un indice i pointant sur un élément du tableau.

b) `construire-tas-max(`t`)` : prend en paramètre un tableau t et le convertit en tas-max.

3°) Définissez la procédure `tri-par-tas(`t`)`, qui prend en paramètre un tableau $t[1..n]$ où $n = t.longueur$ et le trie en utilisant la propriété de tas-max.

Exercice 3 (Préliminaire - Files de priorités)
Une file de priorités est une structure de données qui permet de gérer un ensemble d'éléments dont chacun a une valeur associée baptisée *cle*. Elle permet de gérer des tâches en attente avec leurs priorités relatives, d'extraire à tout moment la tâche de plus forte priorité et d'insérer de nouvelles tâches à la file. L'une des applications les plus répandues du tas est la gestion d'une file de priorités efficace.

1°) Définissez les procédures suivantes applicables à une file de priorités max utilisant la structure de tas :

a) `maximum-tas(`t`)` : retourne l'élément du tas max t ayant la clé maximale ;

b) `extraire-max-tas(`t`)` : supprime et retourne l'élément de t qui a la clé maximale tout en s'assurant de conserver la propriété de tas. Elle affiche un message d'erreur si le tas est vide ;

c) `augmenter-clé-tas(`t, i, cle`)` : accroît la valeur de l'élément d'indice i pour lui donner la nouvelle valeur *cle* tout en s'assurant de conserver la propriété de tas max ;

d) `insérer-tas-max(`t, cle`)` : prend en entrée la valeur *cle* à insérer dans le tas max t et insère un nouveau noeud associé à cette valeur tout en s'assurant de conserver la propriété de tas.

Exercice 4 (Préliminaire - Gestion des composantes connexes d'un graphe)
On suppose disposer des opérations suivantes :

- `creer-ensemble(x)` : créé un nouvel ensemble dont le seul membre (et donc le représentant) est x. Comme les ensembles sont disjoints, il faut que x ne soit pas déjà membre d'un autre ensemble.
- `union(x,y)` : réunit les ensembles qui contiennent x et y, disons S_x et S_y, dans un nouvel ensemble qui est l'union de ces deux ensembles. Les deux ensembles sont supposés être disjoints avant l'opération. Le représentant de l'ensemble résultant est un membre quelconque de $S_x \cup S_y$. Après l'union, S_x et S_y n'existent plus dans la collection S.

— `trouver-ensemble(x)` : retourne un pointeur vers le représentant de l'ensemble (unique) contenant x.

L'une des nombreuses applications des structures de données d'ensembles disjoints apparaît lorsqu'il s'agit de déterminer les composantes connexes d'un graphe non orienté. Pour ce faire, une façon de procéder est de placer initialement chaque sommet v dans son propre ensemble. Ensuite, Pour chaque arête (u, v), on réunit les ensembles contenant u et v. Quand toutes les arêtes ont été traitées, deux sommets se trouvent dans la même composante connexe si et seulement si les objets correspondants se trouvent dans le même ensemble.

1°) Définissez les procédures suivantes :

a) `composantes-connexes` : utilise les opérations d'ensembles disjoints pour calculer les composantes connexes d'un graphe, tel que défini précédemment ;

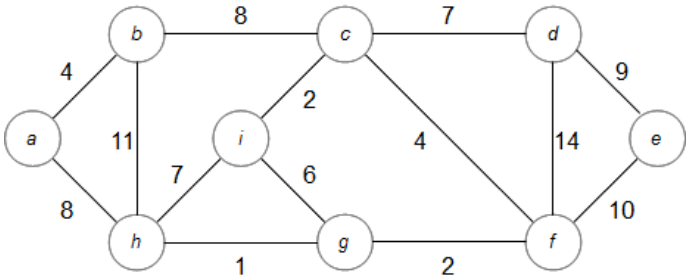
b) `meme-composante` : indique, après un prétraitement du graphe par `composantes-connexes`, si deux sommets se trouvent dans la même composante connexe.

Une façon simple de mettre en oeuvre une structure de données d'ensemble disjoints est de représenter chaque ensemble par sa propre liste chaînée. L'objet associé à chaque ensemble a les attributs *tete*, pointant vers le premier objet de la liste, et *queue*, pointant vers le dernier objet. Chaque objet de la liste contient un membre de l'ensemble, un pointeur vers l'objet suivant dans la liste et un pointeur vers l'objet ensemble. Dans chaque liste chaînée, les objets peuvent figurer dans n'importe quel ordre. Le représentant est le membre de l'ensemble contenu dans le premier objet de la liste.

2°) Illustrez cette représentation avec les ensembles $S_1 = \{d, f, g\}$ avec f comme représentant, et $S_2 = \{b, c, e, h\}$ avec c comme représentant. Illustrez ensuite le résultat de `union(g,e)`.

3°) Définissez les procédures `creer-ensemble`, `union` et `trouver-ensemble` utilisant la représentation par listes chaînées. Donnez aussi leur temps d'exécution.

Exercice 5 (Algorithme de Kruskal)
Soit le graphe non orienté, connexe et pondéré suivant :



1°) Illustrez l'exécution de l'algorithme de Kruskal et donnez l'arbre couvrant de poids minimal obtenu.

2°) Définissez la procédure `acpm-kruskal-tableau` qui implémente l'algorithme de Kruskal avec gestion des composantes connexes par tableau, et donnez son temps d'exécution.

3°) Définissez la procédure `acpm-kruskal-ensembles` qui implémente l'algorithme de Kruskal avec gestion des composantes connexes par ensembles disjoints, et donnez son temps d'exécution.

Exercice 6 (Algorithme de Prim)
1°) Illustrez l'exécution de l'algorithme de Prim sur le graphe de l'exercice précédent et donnez l'arbre couvrant de poids minimal obtenu.

2°) Définissez la procédure `acpm-prim-tableau` qui implémente l'algorithme de Prim avec gestion des sommets non couverts par tableau, et donnez son temps d'exécution.

3°) Définissez la procédure `acpm-prim-file-priorites` qui implémente l'algorithme de Prim avec gestion des sommets non couverts par file de priorités, et donnez son temps d'exécution.

Références

Cormen T. H., Leiserson C. E., Rivest R. L. et Stein C., "Algorithmique" 3e édition, Dunod, 2010.