UNIVERSITÉ DE REIMS CHAMPAGNE-ARDENNE

LICENCE Informatique
Info0501 – 2018/2019

Travaux dirigés n° 1

Structures de données élémentaires : tas, piles, files et listes chaînées

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

1°) 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. Quand la procédure est appelée, elle suppose que les arbres binaires enracinés en `gauche(`i`)` et `droite(`i`)` sont des tas-max, mais que $t[i]$ pourrait être plus petit que ses enfants, violant ainsi la propriété de tas-max. La procédure fait descendre la valeur de $t[i]$ dans le tas-max, de manière que le sous-arbre enraciné en i respecte la propriété de tas-max ;

b) `construire-tas-max(`t`)` : prend en paramètre un tableau t et le convertit en tas-max. Elle part du principe que les éléments du sous-tableau $t[(\lfloor n/2 \rfloor + 1)..n]$ sont tous des feuilles de l'arbre, donc initialement des tas à 1 élément. Elle parcourt donc les autres noeuds de l'arbre et les entasse dans leur sous-arbre.

2°) 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. À titre d'indication, si on transforme le tableau en tas, on sait que l'élément maximal est à la première case donc on peut le placer dans sa position finale correcte par un échange et l'enlever du tas en décrémentant t.taille. Les enfants de la racine restent alors des tas-max, mais la nouvelle racine risque d'enfreindre la propriété de tas-max. Pour restaurer cette dernière, il faut entasser cette nouvelle racine. En répétant le processus jusqu'à arriver à un tas de taille 2, on obtient un tableau trié.

Exercice 3 (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 (Piles)
1°) Rappelez le fonctionnement d'une pile d'au plus n éléments avec un tableau $p[1..n]$. Le tableau possède un attribut $p.sommet$ qui indexe l'élément le plus récemment inséré. La pile est constituée des éléments $p[1..p.sommet]$ où $p[1]$ est l'élément situé à la base de la pile et $p[p.sommet]$ l'élément l'élément situé au sommet. Donnez les trois états successifs suivants d'une pile d'au plus 7 éléments :

1. la pile après empilage successif des valeurs 15, 6, 2 et 9 ;
2. la pile après empilage successif des valeurs 17 et 3 ;
3. la pile après un dépilage.

2°) Définissez les procédures de pile suivantes :

a) `pile-vide` : teste si la pile est vide ;

b) `empiler` : empile une valeur dans la pile ;

c) `dépiler` : dépile un élément de la pile.

Exercice 5 (Files)
1°) Rappelez le fonctionnement d'une file d'au plus $n-1$ éléments à l'aide d'un tableau $f[1..n]$. La file comporte un attribut $f.tete$ qui indexe vers sa tête où sera retiré le prochain élément. L'attribut $f.queue$ indexe le prochain emplacement où sera inséré un élément nouveau. Les éléments de la file se trouvent aux emplacements $f.tete$, $f.tete+1$, ... $f.queue-1$, après quoi l'on boucle : l'emplacement 1 suit immédiatement l'emplacement n dans un ordre circulaire. Quand $f.tete = f.queue$, la file est vide. Au départ, on a $f.tete = f.queue = 1$. Quand $f.tete = f.queue + 1$, la file est pleine. Donnez les trois états successifs suivants d'une file d'au plus 11 éléments :

1. la file où $f.tete = 7$, $f.queue = 12$ et où les valeurs, de la tête à la queue, sont 15, 6, 9, 8 et 4 ;
2. la file après l'enfilage successif des valeurs 17, 3 et 5 ;
3. la file après un défilage.

2°) Définissez les procédures de file suivantes :

a) `enfiler` : insère un élément dans la file ;

b) `défiler` : retire un élément de la file.

Exercice 6 (Listes chaînées)
1°) Rappelez le fonctionnement d'une liste doublement chaînée l où chaque élément est un objet comportant un attribut *cle* et deux autres attributs pointeurs : *succ* et *pred*. Étant donné un élément x de la liste, $x.succ$ pointe sur son successeur dans la liste et $x.pred$ pointe sur son prédécesseur. Si $x.pred = \text{NIL}$, l'élément x n'a pas de prédécesseur et est donc le premier élément, aussi appelé tête de liste. Si $x.succ = \text{NIL}$, l'élément x n'a pas de successeur et est donc le dernier élément de la liste, aussi appelé queue de liste. Un attribut $l.tete$ pointe sur le premier élément de la liste. Si $l.tete = \text{NIL}$, la liste est vide. Donnez les trois états successifs suivants :

1. la liste représentant l'ensemble 9, 16, 4, 1 ;
2. la liste après insertion, en tête de liste, d'un élément dont la clé vaut 25 ;
3. la liste après suppression de l'élément dont la clé vaut 4.

2°) Définissez les procédures de liste suivantes :

a) `rechercher-liste` : trouve le premier élément dont la valeur de la clé est spécifié et retourne un pointeur sur cet élément. Si aucune élément ne possède cette valeur, la procédure retourne NIL ;

b) `insérer-liste` : étant donné un élément dont l'attribut *cle* a deja été initialisé, greffe l'élément en tête de liste.

c) `supprimer-liste` : élimine un élément de la liste chaînée dont un pointeur lui est fourni.

Références

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