Mastère IMSI, INSA de Lyon

Introduction à l'algorithmique : exercices

Voici une série d'exercices sur les notions vues en cours. Ces exercices sont de difficulté croissante, il est donc conseillé de les faire dans l'ordre.

1 Algorithmes sur les tableaux

1.1 Recherche itérative

Écrire une fonction qui recherche la valeur minimum d'un tableau d'entier et revoie son indice comme valeur de retour.

1.2 Renversement d'un tableau

Écrire une fonction qui renverse un tableau (la première valeur devient la dernière, la deuxième l'avant dernière, etc). Par exemple, si le tableau est trié en ordre croissant au début de la fonction il doit être trié en ordre décroissant par l'algorithme.

1.3 Palindromes

Un palindrome est un tableau de caractères qui se lit de la même façon dans les deux sens (ex : « elle », « radar », « laval »). La fonction doit renvoyer 1 si le tableau est un palindrome et 0 sinon.

1. Écrire une fonction qui dit si un tableau est un palindrome.
2. Écrire la même fonction mais cette fois-ci en ajoutant la possibilité d'avoir des espaces et des apostrophes dans le palindrome (ex : « tu l'as trop ecrase Cesar ce port salut »)

1.4 Recherche dichotomique

La recherche d'un élément dans un tableau non trié est linéaire (il faut parcourir en moyenne un nombre d'élément proportionnel au nombre d'élément du tableau et au pire les n éléments du tableau). Si le tableau est trié (on suppose en ordre croissant) il est possible de faire une recherche beaucoup plus rapidement en utilisant une méthode dichotomique.

La recherche dichotomique dans un ensemble ordonné suppose de connaître le nombre d'éléments dans le tableau. Il suffit de regarder au milieu du tableau, de comparer ce que l'on trouve avec l'élément que l'on recherche. Si l'élément est le bon alors le recherche est finie sinon il faut recommencer avec la partie gauche ou la partie droite du tableau selon le résultat de la comparaison.

1. Écrivez une fonction de recherche dichotimique dans un tableau. Cette fonction doit renvoyer l'indice de l'élément recherché s'il est présent dans le tableau ou -1 si l'élément n'a pas été trouvé.
2. Estimez le nombre de comparaisons que vous avez à faire pour arriver à trouver un élément (ou à dire qu'il n'est pas dans le tableau).
3. Est-ce que cette méthode peut être appliquée sur une structure de liste ?

2 Algorithme sur les listes chaînées

2.1 Recherche dans une liste

Écrire une fonction de recherche dans une liste (non triée).

2.2 Algorithmes sur les les piles

Une "pile" est une liste dans laquelle on se contraint à toujours faire des insertions et des suppréssions au début de la liste. Seul le premier élément est dit accessible directement (on ne s'autorise pas à aller regarder plus loin que le premier élément dans la liste à un instant donné). On garde donc un ordre particulier sur les éléments qui est "premier entré, dernier sorti".

1. Écrire une fonction qui ajoute un élément dans une pile.
2. Écrire une fonction qui renvoie la valeur de l'élément au sommet de la pile.
3. Écrire une fonction qui supprime le sommet de la pile.

2.3 Algorithmes sur les files

Une "file" est une liste dans laquelle on insère un élément en début de liste et dans laquelle on supprime un élément à la fin de la liste. De manière analogue aux piles on ne s'autorise à regarder que le dernier élément de la file pour avoir un ordre sur les éléments qui est "premier entré, premier sorti".

1. Écrire une fonction qui ajoute un élément dans une file.
2. Écrire une fonction qui renvoie la valeur de l'élément accessible de la file.
3. Écrire une fonction qui supprime le premier élément (l'élément accessible) de la file.

2.4 Versions récursives des algorithmes

Les listes et les structures dynamiques se prêtent généralement assez bien aux versions récursives des algorithmes de parcours et de recherche. Une version récursive des algorithmes sur les listes est obtenue en général en considérant la liste comme composée de deux entités : l'élément que l'on est en train de considérer et la liste des éléments suivants. Une version récursive des algorithmes considère donc les éléments un par un suivant le traitement que l'on veut faire dans la fonction et si le traitement n'a pas pu être fait (l'élément n'est pas le bon) se rappelle elle-même avec le reste de la liste comme argument.

1. Écrire une version récursive de l'algorithme de recherche dans une liste non triée.
2. Écrire une version récursive de l'algorithme d'insertion dans une liste triée.

3 Algorithmes sur les arbres

On se donne un arbre dont les cellules (nœuds) sont définis de la manière suivante :

```
structure noeud (
    entier val
    structure noeud* FilsGauche
    structure noeud* FilsDroit
)
```

Il est délicat de parcourir un arbre sans avoir recour à de la programmation récursive (on peut s'en passer mais il faut alors utiliser des structures de pile ou de file pour se rappeler des emplacements visités au cours du parcour de l'arbre).

Il existe trois grandes façons de parcourir un arbre : préfixée, infixée et postfixée.

```
prefix(structure noeud *racine)
{
    si racine != NULL alors
        traiter le noeud
        prefix((*racine).FilsGauche) (traiter le fils gauche)
        prefix((*racine).FilsDroit) (traiter le fils droit)
    fin si
}
```

```
infix(structure noeud *racine)
{
    si racine != NULL alors
        infix((*racine).FilsGauche) (traiter le fils gauche)
        traiter le noeud (avec le résultat du fils gauche)
        infix((*racine).FilsDroit) (traiter le fils droit)
    fin si
}

postfix(structure noeud *racine)
{
    si racine != NULL alors
        postfix((*racine).FilsGauche) (traiter le fils gauche)
        postfix((*racine).FilsDroit) (traiter le fils droit)
        traiter le noeud (avec le résultats des traitements des fils)
    fin si
}
```

3.1 Hauteur d'un arbre

La hauteur d'un arbre est la plus longue chaîne depuis la racine jusqu'aux feuilles. On compte +1 à chaque fois que l'on passe par un nœud de l'arbre. Écrire une fonction qui calcule la hauteur d'un arbre binaire (on pourra utiliser une fonction récursive de type postfixée). La fonction à écrire prend comme argument l'adresse du sommet de l'arbre et renvoi la hauteur de cet arbre.

3.2 Recherche dans un arbre trié

Un arbre trié est tel que les valeurs de tous les fils gauche d'un nœud n soient inférieures à la valeur de n et les valeurs de tous les fils droits soient supérieures à la valeur de n. Écrire une fonction de recherche qui indique si une valeur est présente dans un arbre binaire de recherche (arbre trié). Cette fonction prend en argument le sommet de l'arbre ainsi que la valeur à rechercher et renvoi 0 ou 1 suivant le résultat de la recherche (non trouvé, trouvé).