Examen d'algorithmique

Mercredi 22 juin 2016 15h30–18h / Aucun document autorisé

Mode d'emploi : Le barème est donné à titre indicatif. **La qualité de la rédaction des algorithmes et des explications sera fortement prise en compte pour la note.** On peut toujours supposer une question résolue et passer à la suite.

Exercice 1 : Dérouler des algorithmes (3 points)

On considère l'algorithme `P` ci-dessous :

```
Def P(tableau d'entiers T) :
// les indices de T vont de 0 à |T|-1
Si |T|==0 Alors Retourner 0
Sinon :
  n = |T|
  s = 0
  i = 0
  tant que i < n && T[i]<10  faire:
     s = s+T[i]
     i=i+1
  Retourner s
```

1. Décrire ce que fait l'algorithme `P` appelé avec le tableau $\texttt{T} = [3, 4, 6, 1, 12, 2, 4]$.
2. Et avec le tableau $\texttt{T} = [3, 1, 7, 8]$?

Exercice 2 : Tri pour deux valeurs - 4 points

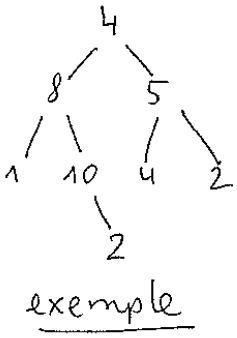
On veut définir un algorithme de tri pour des tableaux de taille n ne contenant que deux valeurs distinctes. L'objectif est de trier le tableau.

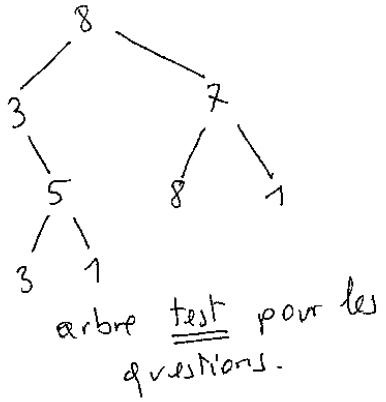
Par exemple pour le tableau suivant de taille 5 : `[2,4,4,2,2]`, on veut obtenir `[2,2,2,4,4]`

1. Ecrire un algorithme de tri basé sur une méthode de comptage. Ici on souhaite juste obtenir un tableau trié sans imposer l'ordre (croissant ou décroissant).
2. Ecrire un algorithme qui trie le tableau en ne faisant qu'un seul parcours du tableau.
3. Modifiez vos algorithmes pour obtenir un tri en ordre croissant.

Exercice 3 : Algorithmes sur les arbres - 6 points

On considère des arbre binaires contenant des valeurs entières dans les noeuds, comme ceux de la figure ci-dessous :





On suppose que ces arbres sont représentés par des structures chaînées (comme en cours). Un noeud de l'arbre (type `noeud`) sera représenté par une structure ayant les champs de valeurs suivants :

— un champ de nom `val` et de type `entier` contenant la valeur stockée ;
— un champ de nom `fg` et de type `arbre` contenant l'adresse du fils gauche ;
— un champ de nom `fd` et de type `arbre` contenant l'adresse du fils droit.

Et un `arbre` est un pointeur (adresse) vers un `noeud` (l'adresse 0 désigne un arbre vide). Lorsqu'un noeud n'a pas de fils gauche, son champ `fg` vaut 0 (et c'est pareil pour le fils droit avec `fd`).

Si `a` est un arbre non vide, `a->val` désigne la valeur stockée à sa racine (le premier noeud de l'arbre), `a->fg` désigne l'adresse du fils gauche (donc un arbre), et `a->fg` désigne l'adresse du fils droit, *etc.*

1. Dessiner la structure chaînée représentant l'arbre `test`.
2. Écrire un algorithme `Somme` qui étant donné un arbre `a` retourne la somme de toutes les valeurs stockées dans les noeuds de cet arbre (et 0 si l'arbre est vide). Sur l'exemple, on doit renvoyer 36.
   Profil suggéré : `entier Somme(arbre a)`
   Appliquer votre algorithme sur l'arbre `test` et décrire son comportement.
3. Écrire un algorithme de parcours d'arbre (au choix parmi tous ceux possibles) qui affichera les valeurs stockées dans les noeuds.
   Profil suggéré : `void Parcours(arbre a)`
   Appliquer votre algorithme sur l'arbre `test` et décrire son comportement.
4. Écrire un algorithme `CptNoeudBin` qui étant donné un arbre `a` retourne le nombre de noeuds binaires (c'est-à-dire avec deux sous-arbres non vides) de l'arbre `a`. Sur l'exemple, on doit renvoyer 3.
   Profil : `entier CptNoeudBin(arbre a)`
   Quelle valeur est retournée votre algorithme sur l'arbre `test` ?

5. Ecrire un algorithme `ProfOcc` qui étant donné un arbre `a` et un entier `x` affiche la profondeur de chaque occurrence de la valeur x (dans les champs `val`). On rappelle que la racine est à la profondeur 0, ses fils à la profondeur 1, les fils de ses fils à la profondeur 2, *etc.* Sur l'exemple et avec $x = 4$, on doit donc renvoyer 0 2.
   Profil suggéré : `void ProfOcc(arbre a, entier x)`. On pourra utiliser une fonction auxiliaire de profil `void ProfOccAux(arbre a, entier x, entier p)`.
   Que retourne votre algorithme sur l'arbre `test` avec $x = 3$?

Exercice 4 : Backtracking - 6 points

Dans cet exercice, on dispose d'un ensemble $\mathcal{E} = \{v_0, \ldots, v_{n-1}\}$ de n entiers distincts et d'une valeur entière V, et on cherche à savoir si il existe un (ou plusieurs) sous-ensemble(s) de $\mathcal{E}$ dont la somme vaut V. Par exemple, si $\mathcal{E} = \{4, 3, 2, 5\}$ et si $V = 9$, alors deux sous-ensembles sont acceptables : $\{4, 3, 2\}$ et $\{4, 5\}$.

Pour cela, on veut utiliser un algorithme de backtracking pour générer tous les sous-ensembles de $\mathcal{E}$ et tester pour chacun si la somme de ses éléments est égale à V.

1. Etant donné un ensemble $\mathcal{E}$ représenté par un tableau `T` de taille n (`T[i]` vaut v_i), **écrire un algorithme qui affiche tous les sous-ensembles de $\mathcal{E}$.**
   NB : Avec l'ensemble $\mathcal{E} = \{2, 4, 6\}$, l'algorithme devra afficher (dans un certain ordre) les sous-ensembles : {}, {2},{4},{6},{2, 4},{4, 6},{2, 6},{2, 4, 6}.
   Appliquer votre algorithme à `T` $= [2, 3]$. On décrira avec précision le déroulé de l'algorithme.
   Profil suggéré (mais non imposé) si algorithme récursif : `void GenererEns(T,S,k)` où `S` est le sous-ensemble en cours de construction représenté par un tableau de booléen (`S`[`i`] est vrai ssi `T`[`i`] est dans le sous-ensemble) et `k` est un entier (entre 0 et $n - 1$) indiquant que le prochain choix à faire portera sur l'élément k (et donc qu'un choix a déjà été fait pour $v_0, \ldots, v_{k-1}$).
2. Etant donné un ensemble $\mathcal{E}$ représenté par un tableau `T` de taille n et un entier V, **écrire un algorithme qui affiche tous les sous-ensembles de $\mathcal{E}$ dont la somme des éléments vaut V.**.
   NB : Avec l'ensemble $\mathcal{E} = \{2, 4, 6\}$ et $V = 6$, l'algorithme devra afficher les sous-ensembles : $\{2, 4\}$ et $\{6\}$.

   Profil suggéré si algorithme récursif : `void GenererEns2(T,V,S,k)`.
3. Modifier l'algorithme précédent pour compter le nombre de solutions.