Université Bordeaux 1. Licence Sciences & Technologies, Informatique.
Examen UE INF 152, Algorithmique et Programmation.
Session 1 2008–2009. 18 mai 2009. Durée : 1h30.
Documents autorisés : une page manuscrite A4 recto-verso.
La notation tiendra compte du soin apporté à la rédaction.
Le barème est indicatif.

Exercice 1 (3 pts) Questions de cours. Répondre brièvement. On ne demande pas de justification.

1. Citer un algorithme de tri ayant une complexité asymptotique indépendante des entrées.
2. Pour rechercher un entier dans un tableau déjà trié de n éléments, peut-on faire mieux qu'un algorithme de complexité linéaire en n ? Si oui, par quel algorithme et avec quelle complexité ?
3. Construire l'arbre de Huffman obtenu sur le texte `ananas`, et le codage correspondant d'`ananas`.

Exercice 2 (9 pts) Dans cet exercice, `t` est un tableau d'entiers positifs ou nuls.

1. Étant donnés deux nombres entiers `a` et `b` (avec $0 \leqslant$ `a` $<$ `b`), on souhaite calculer le nombre d'éléments de `t` appartenant à l'intervalle]`a`, `b`] (c'est-à-dire entre `a` exclu et `b` inclus). Écrire pour cela une fonction `nb_elements(a, b, t)` et évaluer sa complexité.
   Exemple : pour le tableau `t=[1,2,2,2,5,1,3,5]`, `a` $= 2$ et `b` $= 5$ la fonction retournera 3.
2. Écrire une fonction `nb_occurrences(t, x)` qui retourne le nombre de fois où l'entier `x` apparaît dans le tableau `t`. Indiquer, en fonction du nombre d'éléments de `t`, le nombre de comparaisons entre `x` et un élément de `t` effectuées par votre fonction.
3. La fonction suivante utilise la fonction Python `max`, retournant la plus grande valeur d'un tableau.

```
def f(t):
  maximum = max(t)
  tab = [0]*(maximum + 1)
  for i in range(maximum + 1):
    tab[i] = nb_occurrences(t,i)
  return tab
```

   (*a*) Donner le résultat du programme suivant :

```
t = [1,2,2,2,5,1,3,5]
v = f(t)
print v
```

   Sur cet exemple, combien de fois la fonction `nb_occurrences` est-elle appelée lors de l'exécution du programme ?

   (*b*) De façon générale, que retourne la fonction `f` ?

   (*c*) Indiquer le nombre total de comparaisons effectuées par l'ensemble des appels de la fonction `nb_occurrences` durant une exécution de la fonction `f`.

   (*d*) Proposer une fonction `f_bis(t)` qui calcule le même résultat que la fonction `f(t)`, mais qui, après l'initialisation du tableau `tab`, ne parcourt qu'une seule fois le tableau `t`.

4. Le tableau retourné par la fonction `f` peut se révéler utile pour effectuer un tri du tableau `t`. Quel algorithme de tri peut l'utiliser à profit (il n'est pas demandé de le réécrire) ? Sous quelle condition l'algorithme de tri sera-t-il linéaire ?

5. Écrire une fonction `cumul(v)` de façon que, si `v = f(t)`, elle retourne un tableau dont le `i`-ème élément est égal au nombre d'éléments de `t` dont la valeur est inférieure ou égale à `i`.

   Par exemple, l'exécution du programme :

   ```
   t = [1,2,2,2,5,1,3,5]
   v = f(t)
   c = cumul(v)
   print c
   ```

   doit afficher le tableau $[0, 2, 5, 6, 6, 8]$. La fonction `cumul(v)` doit avoir une complexité linéaire par rapport au nombre d'éléments de `v`.

6. Écrire une fonction `nb_elements_bis(a, b, c)` qui calcule le même résultat que la fonction `nb_elements(a, b, t)` de la question 1, quand on utilise comme troisième paramètre `c` le tableau `cumul(f(t))`. Par exemple, après la suite d'instructions :

   ```
   t = [1,2,2,2,5,1,3,5]
   v = f(t)
   c = cumul(v)
   ```

   l'appel `nb_elements_bis(2,5,c)` retournera 3 (comme l'appel `nb_elements(2,5,t)`). Attention, il est utile de distinguer le cas où `b` $<$ `len(c)` du cas contraire. Évaluer la complexité de la fonction `nb_elements_bis`.

Exercice 3 (8 pts) Dans cet exercice, on demande de ne pas utiliser des expressions de la forme **`x in T`**.

1. Écrire une fonction `appartient(x, T)` qui retourne `True` si `x` est un élément du tableau `T` et `False` sinon.
2. Soit la fonction :

   ```
   def appartient_bis(x, T):
       return appartient_rec(x, T, 0, len(T)-1)
   ```

   qui retourne `True` si `x` est un élément du tableau `T` et `False` sinon.

   Écrire la fonction *récursive* `appartient_rec(x, L, i, j)`.

Soient `T1` et `T2` des tableaux d'entiers qui représentent chacun un ensemble. On suppose que chaque valeur présente dans `T1` n'y apparaît qu'une seule fois, et de même pour `T2`. Par exemple, `T1` peut être `[4,1,3]`, mais pas `[4,1,4,3]`, parce que 4 apparaît deux fois.

3. En utilisant la fonction `appartient`, écrire une fonction `card_intersection(T1,T2)` qui calcule et retourne le nombre d'éléments de l'intersection des deux tableaux passés en paramètre.

   Quelle est la complexité de la fonction `card_intersection` ? Justifier brièvement la réponse.

Par la suite on suppose que les éléments des tableaux sont triés dans l'ordre *décroissant*.

4. Écrire une fonction *efficace* `appartient_tab_trie(x, T)` qui retourne `True` si `x` est un élément de `T` et `False` sinon. Quelle est sa complexité ?
5. En s'inspirant de l'algorithme de fusion de tableaux triés, écrire une fonction `intersection(T1,T2)` qui, à partir de deux tableaux `T1` et `T2`, construit et retourne le tableau contenant l'intersection de `T1` et `T2`, trié lui aussi dans l'ordre *décroissant*.

   Quelle est la complexité de la fonction `intersection` ? Justifier brièvement la réponse.