UNIVERSITÉ DE MONTPELLIER

UE : HLIN301

Session : 1 — Durée de l'épreuve : 2 heures

Date : Janvier 2018 — Documents autorisés : 1 feuille A4 recto verso

Licence $2^{ème}$ année, parcours Informatique et Math-Informatique — Matériel utilisé : aucun

1 Tableaux

Question 1. (**3,5 points**) Un tableau d'entiers de taille n est une permutation de l'intervalle $[0, n-1]$, s'il contient une et une seule fois chaque entier de l'intervalle $[0, n-1]$.

Exemples :

le tableau | 1 | 4 | 2 | 5 | 0 | 3 | est une permutation de $[0, 5]$

les tableaux | 1 | 4 | 6 | 5 | 8 | 0 | et | 1 | 4 | 2 | 5 | 2 | 0 | ne sont pas des permutations de $[0, 5]$.

Écrivez deux algorithmes qui vérifient si un tableau de taille n est une permutation de l'intervalle $[0, n-1]$:

1. Dans le premier cas on suppose que la donnée est un tableau d'entiers triés par ordre croissant. Quelle est sa complexité ?
2. Dans le second cas la donnée est un tableau d'entiers, pas nécessairement trié. Votre algorithme ne doit pas trier le tableau. Quelle est sa complexité ? Celle-ci doit être la plus basse possible.

Question 2. (**2,5 points**) Un tableau T d'entiers de taille n est une permutation d'intervalle si il existe un entier positif k tel que T est une permutation de l'intervalle $[k, k+n-1]$, c'est à dire s'il contient une et une seule fois chaque entier de l'intervalle $[k, k+n-1]$.

Exemples :

le tableau | 12 | 11 | 8 | 10 | 9 | 13 | est une permutation d'intervalle. Il est une permutation de l'intervalle de $[8, 13]$

| 15 | 20 | 16 | 17 | 19 | 18 | est une permutation d'intervalle (de l'intervalle de $[15, 20]$)

les tableaux | 2 | 4 | 9 | 5 | 8 | et | 2 | 1 | 3 | 2 | 1 | ne sont pas des permutations d'intervalle.

Écrivez un algorithme qui teste si un tableau d'entiers est une permutation d'intervalle. On souhaite un algorithme de complexité optimale. Si vous le voulez, vous pouvez trier le tableau. Dans ce cas vous indiquez quel tri vous utiliser, sans redonner l'algorithme. Quelle est la complexité de votre algorithme ?

Question 3. (3 points) n joueurs participent à une compétition composée de deux épreuves. Les règles sont les suivantes :

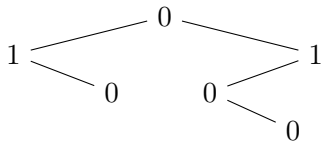
- chaque joueur est numéroté de 1 à n
- le résultat d'une épreuve est un tableau de taille n, permutation de l'intervalle $[1, n]$. Il représente le classement à l'épreuve : le vainqueur est à l'indice 0, le second à l'indice 1, ...
- le score obtenu par un joueur à une épreuve est son indice dans le tableau résultat. Le score final d'un joueur est la somme de ses scores aux deux épreuves.
- un joueur `j1` est classé avant un joueur `j2` si son score final est inférieur à celui de `j2`, ou bien si les deux scores sont égaux et `j1` est mieux classé que `j2` à l'épreuve 1.

Par exemple si | 2 | 5 | 7 | 4 | 1 | 6 | 3 | et | 6 | 4 | 7 | 5 | 3 | 1 | 2 | sont les résultats aux deux épreuves, le score final du joueur 1 est 4+5=9. Le meilleur score est obtenu par les joueurs 4, 5 et 7. Ils ont tous trois le score 4. Le vainqueur est le joueur 5 car il est le mieux classé des trois joueurs à la première épreuve.

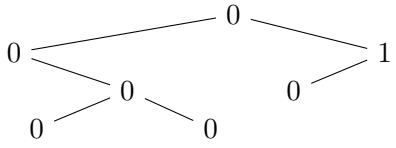
Écrivez un algorithme qui étant les classements des n joueurs aux deux épreuves calcule le vainqueur de la compétition. Quelle est sa complexité ?

2 Arbres Binaires

Question 4. (**1,5 points**) Écrivez un algorithme `nbFeuille0(A)` qui compte le nombre de feuilles de valeur 0 dans l'arbre binaire `A`. Des exemples illustrant cette opération sont donnés ci–dessous :

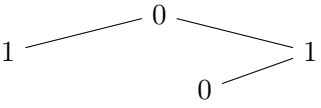


Arbre `A1` ; `nbFeuille0(A1) = 2`

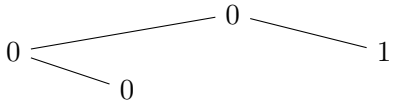


Arbre `A2` ; `nbFeuille0(A2) = 3`

Question 5. (**1,5 points**) Écrivez un algorithme `oterFeuille0(A)` qui supprime de l'arbre `A` toutes les feuilles de valeurs 0. Des exemples illustrant cette opération sont donnés ci–dessous :

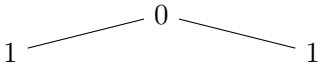


Résultat de `oterFeuille0(A1)`



Résultat de `oterFeuille0(A2)`

Question 6. (**2 points**) Écrivez un algorithme `oterSsArbre0(A)` qui supprime de l'arbre `A` tous les sous-arbres composés uniquement de noeuds de valeur 0. Des exemples illustrant cette opération sont donnés ci–dessous :



Résultat de `oterSsArbre0(A1)`

Résultat de `oterSsArbre0(A2)`

3 Listes chaînées

Question 7. (3 points) L'algorithme `supRepet(L)` supprime les répétitions de valeurs d'une liste triée d'entiers `L`. L'algorithme modifie le paramètre `L` sans créer de nouvelle liste. Par exemple si la valeur de L est `(3, 3, 7, 8, 8, 8)`, après l'exécution de `supRepet(L)` la valeur de `L` est `(3, 7, 8)`. Écrivez deux versions d'algorithme, toutes les deux de complexité linéaire dans la taille de la liste :

1. une version récursive
2. une version utilisant une itération.

Question 8. (**4 points**) `L` étant une liste non nulle dont la queue (la liste `L` sans son premier élément) est triée par valeur croissante, `insérerTête(L)` déplace le premier élément de la liste `L` de sorte que `L` soit entièrement triée. Par exemple si la valeur de `L` est la liste `(8, 3, 5, 6, 9)`, après l'exécution de `insérerTête(L)` la valeur de `L` est `(3, 5, 6, 8, 9)`. Aucun élément n'est dupliqué, seul le chaînage de la liste peut être modifié.

1. Écrivez un algorithme pour l'opération `insérerTête(L)`. Quelle est sa complexité ?
2. Utilisez `insérerTête` pour écrire un algorithme triant une liste par insertion successive. Quelle est sa complexité ?