L3-INFO Algorithmique

Examen – 15 décembre 2020 – durée 2 h 30

Les documents et les téléphones (incluant smartphone, tablettes,... tout ce qui contient une interface réseau) interdits.
Seuls les dictionnaires pour les personnes de langue étrangère sont autorisés.
Le barème est indicatif.

Comme d'habitude, vous êtes vivement encouragés à illustrer vos recherches et vos réponses par des schémas.

1 Est-ce un tas ? *(8 points)*

Question 1 : Qu'est ce qu'un tas ?

Donner la définition de la structure de données Tas (question de cours).
Rappeler également comment on représente une telle structure sous forme d'un tableau.

Question 2 : Vérification

Écrire un algorithme de vérification qui prend en argument un tableau quelconque de taille n et vérifie si ce tableau est un tas.

Question 3 : Exemple d'exécution

Donner deux exemples d'exécution sur des tableaux judicieusement choisis pour illustrer votre algorithme :

- l'un devrait représenter un tas correct
- l'autre devrait être incorrect et votre algorithme devrait le détecter dès que possible

Question 4 : Coût de l'algorithme

Calculer le coût de votre algorithme.

Expliquer pourquoi il n'est pas possible d'écrire un algorithme qui vérifie un tas de taille n en moins de $n - 1$ comparaisons.

Question 5 : Tas ternaire

On considère maintenant un *tas ternaire*, dans lequel tout nœud interne (sauf éventuellement le dernier) possède 3 fils.
Expliquer à l'aide d'un schéma comment on peut stocker un tas ternaire dans un tableau.
Réécrire votre algorithme pour vérifier si un tableau donné est un tas ternaire.

2 Plateau *(5 points)*

Dans un tableau T indexé de 1 à n, on cherche un *plateau*, c'est-à-dire deux indices i et j tels que toutes les cellules de T d'indices compris entre i et j (inclus) contiennent une seule et même valeur.

Question 6 : Plateau maximal

On cherche le plateau maximal, c'est-à-dire qui maximise la différence $j - i$.
Proposer un algorithme qui résoud ce problème en un minimum de comparaisons.

Question 7 : Complexité

Déterminer la complexité au pire de votre algorithme.
Justifier qu'il n'est pas possible de trouver un algorithme plus efficace que le votre.

On note « $T[i..j]$ est un plateau » la propriété suivante : $\forall k,\ i \leqslant k \leqslant j \Rightarrow T[k] = T[i]$ autrement dit « toutes les cellules d'indices compris entre i et j contiennent la même valeur ».

Question 8 : Postcondition

Soient $imax$ et $jmax$ les indices renvoyés par votre algorithme. Quelles sont les **deux** propriétés que doivent vérifier T, $imax$ et $jmax$ pour justifier que l'algorithme est correct ?

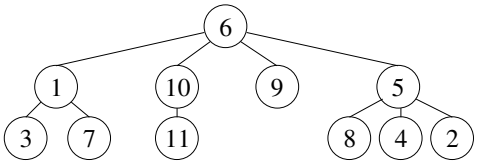
Question 9 : Invariant

Exprimer les deux invariants de la boucle principale de votre algorithme qui permettraient de démontrer chacune des deux propriétés énoncées à la question précédente.
Vous êtes encouragés à vous aider de schémas pour exprimer ces invariants.

3 Représentation d'un arbre quelconque par les liens père *(8 points)*

On cherche dans cet exercice à représenter un arbre enraciné dont les nœuds sont d'arité quelconque.

De façon similaire à la structure de *Union-Find*, on choisit de numéroter les nœuds de 1 à n et d'utiliser un tableau P de taille n : pour chaque nœud i de l'arbre, $P[i]$ contient le numéro de son père (-1 s'il s'agit de la racine de l'arbre).



Par exemple, l'arbre ci-dessus est représenté par le tableau :

i	1	2	3	4	5	6	7	8	9	10	11
$P[i]$	6	5	1	5	6	-1	1	5	6	6	10

3.1 Opérations simples

Il vous est demandé d'écrire quatre algorithmes réalisant les opérations ci-dessous sur des arbres représentés de cette façon.

Vous donnerez la complexité de chaque algorithme écrit ; il n'est pas demandé de justifier cette complexité, mais les algorithmes les moins coûteux seront mieux valorisés.

Vous pourrez au besoin vous servir d'une structure de données auxiliaire, utiliser la récursivité, etc.

Question 10 :

Déterminer l'indice de la racine de l'arbre.

Question 11 :

Construire un tableau de booléens F indexé de 1 à n et tel que $F[i]$ vaut $vrai$ si et seulement si i est une feuille de l'arbre.

Question 12 :

Déterminer le nombre de fils d'un nœud dont l'indice i est donné.

Question 13 :

Calculer la hauteur de l'arbre.

3.2 Est-ce un arbre ?

On considère maintenant un type abstrait *Graphe non orienté* dont on rappelle les opérations :

- GrapheVide : $() \to Graphe$
- ajouterSommet : $Etiquette \times Graphe \to Graphe$
- etiquette : $Sommet \to Etiquette$
- ajouterArete : $Sommet \times Sommet \times Graphe \to Graphe$
- nbSommets : $Graphe \to int$
- ensSommets : $Graphe \to Liste(Sommet)$
- sontVoisins : $Sommet \times Sommet \times Graphe \to bool$
- ensVoisins : $Sommet \times Graphe \to Liste(Sommet)$
- retirerSommet : $Sommet \times Graphe \to Graphe$
- retirerArete : $Sommet \times Sommet \times Graphe \to Graphe$

Une $Liste(Sommet)$ L peut être parcourue au moyen d'une boucle « Pour x dans L... ».

Question 14 :

Écrire un algorithme qui prend en entrée un graphe G et :

- si G n'est pas un arbre l'algorithme renvoie une valeur spéciale $\perp$;
- si G est un arbre, l'algorithme construit un tableau P correct pour cet arbre. Aucune contrainte n'est imposée sur le sommet qui sera choisi pour être la racine.

On supposera que le graphe G est **connexe** et que ses sommets sont déjà étiquetés de 1 à n.