L3-INFO Algorithmique

UFR IM²AG
UNIVERSITÉ Grenoble Alpes

Examen – 10 décembre 2019 – 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.

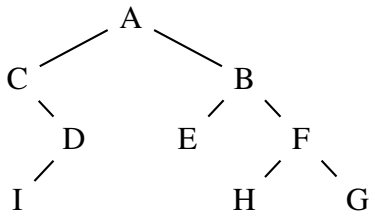
Comme d'habitude, vous êtes vivement encouragés à illustrer vos recherches et vos réponses par des schémas.

Rappels On manipule les arbres binaires à l'aide des primitives suivantes :

- L'appel de fonction $estArbreVide(A)$ renvoie vrai ou faux selon que l'arbre A est vide ou non.
- Pour un arbre non vide A, les appels de fonctions $G(A)$, $D(A)$ et $elem(A)$ renvoient respectivement le fils gauche, le fils droit et l'élément à la racine de A.
- On dispose d'une fonction $ArbreVide$ sans argument, permettant de créer un arbre vide.
- On dispose d'une fonction $Noeud$ prenant en argument deux arbres et un élément et permettant de créer un nouveau nœud (qui est donc la racine d'un nouvel arbre).
- En aucun cas il n'est possible de modifier un nœud « en place » ; en particulier, une instruction comme $G(A) := B$ n'a pas de sens.

1 Parcours en zig-zag d'un arbre binaire (4 points)

Soit un arbre binaire quelconque, comme par exemple :



Le *parcours en zig-zag* d'un tel arbre consiste à le parcourir par niveaux, mais le sens de parcours est alternativement de gauche à droite et de droite à gauche.

Ainsi, sur l'arbre ci-dessus, le parcours en zig-zag visite les nœuds dans l'ordre $ABCDEFGHI$.

Question 1 :

Écrire un algorithme de parcours en zig-zag d'un arbre binaire.
Vous êtes libres d'utiliser toutes les structures de données auxiliaires dont vous aurez besoin : piles, files, files à priorité, tableaux...

2 Gendarmes et voleurs (8 points)

Soit un tableau T de taille n tel que :

— Chaque cellule contient soit un gendarme, soit un voleur.

— Chaque gendarme peut attraper au maximum un voleur.

— Un gendarme ne peut attraper un voleur que si celui-ci est situé à une distance $\leqslant k$ de lui (où k est une donnée du problème).

L'objectif est de trouver le nombre maximal de voleurs que l'on peut attraper.

Exemples (où `G` représente un gendarme et `V` un voleur) :

— T = `['G', 'V', 'V', 'G', 'V']` et $k = 1$.
Le maximum de voleurs attrapé est de 2, par exemple si chaque gendarme attrape le voleur à sa droite.

— T = `['V', 'V', 'V', 'G', 'G', 'G']` et $k = 2$.
On ne peut attraper que les 2 voleurs les plus à droite, puisque le premier est hors de portée de tous les gendarmes.

— T = `['G', 'V', 'V', 'G', 'V', 'G']` et $k = 3$.
On peut clairement attraper les 3 voleurs.

On cherche à construire un algorithme glouton qui, étant donné un tableau de `G` et de `V`, renvoie le nombre maximal de voleurs pouvant être capturés.

2.1 Deux stratégies incorrectes

Question 2 : Première stratégie

On propose la stratégie gloutonne suivante : *En partant de la gauche, chaque gendarme attrape le voleur disponible le plus proche de lui.*
Donner un exemple d'instance sur laquelle cette stratégie échoue à trouver le nombre maximal de voleurs.

Question 3 : Deuxième stratégie

On propose une seconde stratégie gloutonne : *En partant de la gauche, chaque gendarme attrape le voleur disponible le plus loin de lui qui soit à sa portée.*
Donner un exemple d'instance sur laquelle cette stratégie échoue à trouver le nombre maximal de voleurs.

2.2 Une troisième stratégie

Cette fois on choisit le gendarme g et le voleur v les plus à gauche puis on répète le processus ci-dessous jusqu'à avoir épuisé tous les gendarmes ou voleurs disponibles :

1. Si la distance entre g et v est inférieure à k, ce gendarme attrape ce voleur et on passe au gendarme et au voleur suivants.
2. Sinon, identifier de g ou de v qui est le plus à gauche, et le remplacer respectivement par le gendarme ou le voleur suivant.

Question 4 :

Rédiger précisément en pseudo-code l'idée d'algorithme décrite ci-dessus.

Question 5 :

Déterminer la complexité au pire de l'algorithme écrit à la question 4.

Question 6 :

Démontrer la correction de l'algorithme écrit à la question 4. Comme toujours pour les algorithmes gloutons, on cherchera à prouver l'invariant suivant :

Il existe une solution optimale telle que la solution en construction soit incluse dans cette solution optimale.

3 Recherche des k plus grands éléments (8 points)

Dans cet exercice, on dispose de n éléments donnés dans un tableau (non trié), et on cherche à appliquer une certaine opération `Traiter` aux k plus grands d'entre eux, en effectuant aussi peu de comparaisons entre éléments que possible.

Question 7 : Sans structure de données supplémentaire

Écrire un algorithme qui détermine les k plus grands éléments d'un tableau non trié et leur applique la fonction `Traiter`.
Vous pourrez, par des échanges successifs, positionner les k plus grands éléments du tableau dans les k premières cellules du tableau, en vous inspirant de l'algorithme de tri par sélection.

Évaluer le coût de votre algorithme en nombre de comparaisons d'éléments.

Question 8 : Avec un arbre binaire de recherche

Écrire un second algorithme qui prend en entrée un tableau de n éléments et applique la fonction `Traiter` aux k plus grands d'entre eux. Vous pourrez cette fois-ci utiliser un arbre binaire de recherche, dont on suppose que les opérations d'insertion, de recherche d'un élement et de suppression sont déjà programmées.

Évaluer le coût de ce second algorithme en nombre de comparaisons d'éléments.

Question 9 : Avec un tas

Écrire un troisième algorithme qui prend en entrée un tableau de n éléments et applique la fonction `Traiter` aux k plus grands d'entre eux. Dans cette question vous utiliserez une structure de tas, dont les opérations d'insertion et d'extraction du maximum sont déjà programmées.

Évaluer le coût de ce troisième algorithme en nombre de comparaisons d'éléments.

Question 10 : Synthèse

Faire la synthèse des résultats obtenus dans les 3 situations : quelle est la méthode la plus efficace selon les valeurs respectives de n et de k ?
Préciser également l'espace mémoire utilisé par les structures de données auxiliaires utilisées.