Université de Lille 1 SCIENCES ET TECHNOLOGIES

ASD

2015-2016
Licence SESI - Semestre 4

DS2 - documents de cours, TD, TP autorisés - durée 2h

Note : lorsque, dans une question, est demandée la complexité, c'est une fonction de la taille de la donnée qui est attendue. Lorsque c'est le comportement asymptotique, la réponse attendue est soit une notation $\mathcal{O}$, soit une notation Θ soit une notation Ω (le choix vous incombe).

Vous pourrez utiliser toute fonction vue en cours, TD, TP à la condition expresse de spécifier correctement les entrées et les sorties.

Le barême et le temps sont donnés à titre indicatif.

Exercice 1 : Parcours d'arbres

Compétence évaluée : manipuler des arbres binaires.

On suppose manipuler des arbres binaires pour lesquels on dispose des primitives vues en TP.

Q 1.1 Dessiner un arbre qui contient plus de nœuds à la profondeur 3 qu'à la profondeur 2.

Q 1.2 Donner (en Python, C ou pseudocode) le code d'une fonction qui étant donné un arbre binaire retourne un entier correspondant à une des profondeurs qui contient le plus de nœuds (on ne cherche pas à obtenir une méthode particulièrement efficace).

Q 1.3 Quelle est la complexité de votre algorithme dans le pire des cas ? Justifier.

Exercice 2 : Comparaison de listes

Compétence évaluée : choisir des structures de données et des algorithmes pour répondre à un problème.

Etant données deux listes d'entiers, on souhaite écrire un prédicat qui teste si les deux listes de longueur respective n et m, $n > m$ sont *équivalentes*. On dira que deux listes sont équivalentes si elles contiennent les mêmes éléments, peu importe l'ordre et le nombre d'occurrences. Par exemple :

`[1; 9; 6; 1; 1; 5]` et `[ 5; 6; 1; 9; 5]` sont équivalentes,
`[ 5; 6; 1; 9; 5]` et `[ 6; 1; 9; 1]` ne sont pas équivalentes (la valeur 5 n'apparaît pas).

On ne présuppose rien sur les éléments contenus dans les listes si ce n'est que ce sont des entiers.

Q 2.1 Proposer un algorithme en français ou en pseudocode pour réaliser cette tâche dont la complexité asymptotique en temps sera dans le pire des cas $\Theta(n \log n)$ (on s'intéresse à la comparaison d'élements).

Q 2.2 Justifier le comportement asymptotique de l'algorithme proposé.

On suppose maintenant que les valeurs dans les listes sont comprises entre 0 et `MAX`, une constante.

Q 2.3 Proposer un algorithme en français ou en pseudocode pour réaliser cette tâche dont la complexité asymptotique en temps sera $\Theta(n)$ (on s'intéresse à la comparaison d'élements) et en $\Theta(1)$ en espace.

Q 2.4 Justifier les comportements asymptotiques de l'algorithme proposé.

Q 2.5 Que penser du second algorithme dans la pratique ? Argumenter.

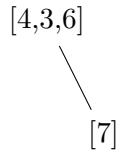
Exercice 3 : Arbres binaires avec liste de valeurs

Compétence évaluée : comprendre un algorithme, une nouvelle structure de données, évaluer une complexité en fonction des structures de données choisies.

On manipule ici des arbres binaires dont l'étiquette est (dans un premier temps) une liste de valeurs de longueur maximale `MAX` fixée. On considérera que `MAX` est une variable globale. `MAX` peut être arbitrairement grand.

Les valeurs dans tout l'arbre seront toujours toutes différentes deux à deux.

Un exemple d'un tel arbre (avec `MAX=3`) est donné ci-dessous :



L'implantation de cette structure de données pourra être réalisée ainsi (l'arbre vide étant représenté par `None`) :

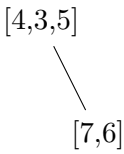
```
    def create_node (left, right):
        """
        :param left: The left son
        :type left: dict or None
        :param right: The right son
        :type right: dict or None
        :return: a tree
        :rtype: dict
        """
        return { "values" : [], "left" : left, "right" : right }
```

L'arbre en exemple ci-dessous sera alors représenté par le dictionnaire :

```
{ 'values' : [4,3,6],
  'left'   : None,
  'right'  : { 'values' : [ 7 ], 'left' : None, 'right' : None } }
```

Pour ajouter de nouvelles étiquettes dans l'arbre, on suit le schéma récursif suivant (implanté dans une fonction `add`). S'il existe un fils gauche (respectivement un fils droit) et que la nouvelle étiquette est inférieure au minimum des étiquettes de la racine (respectivement supérieure au maximum des étiquettes de la racine) alors on insère dans le sous-arbre gauche (respectivement le sous-arbre droit). Dans le cas contraire, si la racine n'est pas pleine, on ajoute la nouvelle étiquette à celles de la racine. Dans le dernier cas on extrait le maximum de la racine puis on ajoute la nouvelle valeur à la racine et on insère le maximum extrait dans le sous-arbre droit (remarquez que l'ordre des étiquettes dans un nœud peut varier au fur et à mesure des insertions).

Par exemple, l'insertion de 5 dans l'arbre ci-dessus aboutira à :



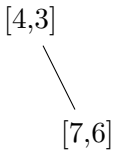
Q 3.1 Dessiner les arbres successifs obtenus avec l'algorithme de construction (avec `MAX=3`) avec la suite d'instructions suivante :

```
a = None
a = add(a,1)
a = add(a,7)
a = add(a,4)
```

```
a = add(a,2)
a = add(a,6)
a = add(a,5)
a = add(a,3)
a = add(a,8)
```

Q 3.2 Quelle propriété remarquable a-t-on entre les étiquettes de la racine et celles des sous-arbres gauche et droit ?

Q 3.3 Proposer en Python une fonction `extractMax` qui étant donné un arbre retourne la valeur max de la racine et la supprime de la liste des étiquettes. Après l'exécution de `extractMax` sur l'arbre ci-dessus, on récupère 5 et l'arbre est dans l'état suivant :



Dans la suite on supposera l'existence d'une fonction `extractMin`.

Q 3.4 Proposer en Python une implémentation de la fonction `add` qui étant donné un arbre et une étiquette ajoute celle-ci dans l'arbre et retourne l'arbre modifié.

Q 3.5 Proposer en Python un prédicat `hasLabel` qui étant donné un arbre et une étiquette retourne vrai si l'arbre contient l'étiquette et faux sinon.

Q 3.6 Donner un meilleur des cas pour le prédicat `hasLabel`. Donner la complexité dans ce cas. Justifier.

Q 3.7 Donner un pire des cas pour le prédicat `hasLabel`. Donner la complexité dans ce cas. Justifier.

Q 3.8 Pour améliorer la structure de données, on propose d'utiliser des listes qu'on maintient triées. Que pensez-vous de cette proposition vis-à-vis des complexités des fonctions `add` et `hasLabel` ? Argumenter.

Q 3.9 Que proposer comme autre structure de données pour remplacer les listes ?

Q 3.10 En quoi celle-ci améliorera ou non le comportement de l'ajout ou de la recherche ? Argumenter.