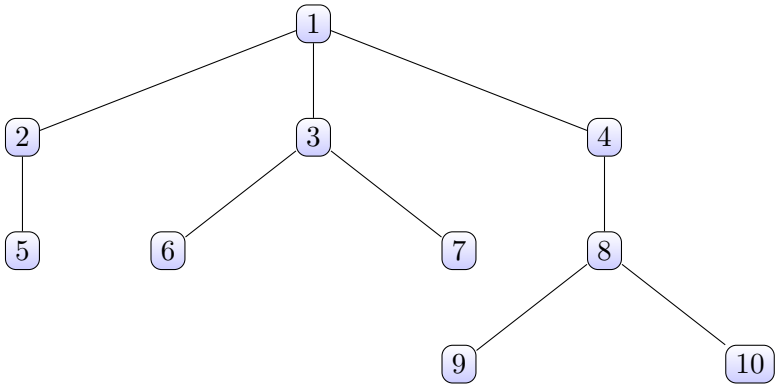
Université Paris Cité

L2 informatique - Année 2023–2024

TD d'Éléments d'Algorithmique n° 12

Exercice 1. *Arbres et Tas Binaires - CC3 21/22.*

1. Transformez l'arbre suivant en arbre binaire en utilisant la transformation vue en cours :



2. Sur l'arbre précédent, dans quel ordre voit-on les sommets pour un parcours Préfixe ? Et un parcours Postfixe ?
3. Dessinez le tas qui est produit quand on effectue les opérations suivantes (en précisant chaque tas intermédiaire) :

$$\texttt{insert}(3), \texttt{insert}(6), \texttt{insert}(2), \texttt{remove}, \texttt{insert}(8), \texttt{insert}(4), \texttt{remove}, \texttt{insert}(1).$$

Nous rappelons les fonctions suivantes sur les arbres binaires :

- `EstVide` (ABin $\mathfrak{t}$) : renvoie VRAI si $\mathfrak{t}$ est l'arbre vide.
- `SAG` (ABin $\mathfrak{t}$) : renvoie $\mathfrak{g}$ si $\mathfrak{t} = (\mathfrak{g}, \mathfrak{d})$, et n'est pas défini sinon.
- `SAD` (ABin $\mathfrak{t}$) : renvoie $\mathfrak{d}$ si $\mathfrak{t} = (\mathfrak{g}, \mathfrak{d})$, et n'est pas défini sinon.

4. Écrivez une fonction récursive qui renvoie le nombre de sommets dans un arbre binaire.
5. Montrez par récurrence sur la hauteur des arbres binaires que le nombre d'arêtes dans un arbre binaire est égal au nombre de sommets -1.

Exercice 2. *Files de Priorité.*

Une *file de priorité* est une structure de données abstraite qui a deux opérations :

- `insert(e, p)` qui insère l'élément e avec la priorité p ;
- `remove` qui retire et retourne un élément de priorité maximale (il peut en général y en avoir plusieurs).

Contrairement au TD précédent, il est possible que l'on tente de mettre à jour la priorité d'un élément.

1. Quel problème le fait de mettre à jour les priorités en utilisant `insert` va-t-il induire par rapport à votre solution du TD 11 ?
2. Écrivez un algorithme `trouveAB` qui cherche dans un tas binaire (implémenté sous forme d'arbre binaire) si un élément s'y trouve déjà, et renvoie le nœud où il se trouve si c'est le cas. Votre fonction renverra l'arbre vide sinon.

On suppose à présent qu'on utilise l'implémentation sous forme de tableau pour les tas binaires.

3. Écrivez un algorithme `trouve` qui cherche dans un tas binaire (implémenté sous forme de tableau) si un élément s'y trouve déjà, et renvoie son emplacement si c'est le cas. Votre fonction renverra -1 sinon.
4. Écrivez un algorithme `insertBis` qui vérifie d'abord si l'élément s'y trouve déjà. Quelle est la complexité en fonction de la taille n du tas ?

On considère à présent que la priorité ne peut qu'augmenter quand on la met à jour. On ne veut plus mettre à jour la priorité dans le tas, mais juste ajouter la nouvelle paire. normalement.

5. Écrivez un algorithme `removeBis` qui s'assure qu'on ne renvoie pas plusieurs fois le même élément. Quelle est la complexité en fonction du nombre de mises à jours m dans le tas et sa taille n ?

Exercice 3. *Solitaire simplifié - CC3 22/23.*

On simplifie le jeu du solitaire en supposant que les cartes sont numérotées de 1 à 52, sans couleur. Le but du jeu est de poser les cartes de manière décroissante de 52 à 1. Pour faire cela, on va pouvoir faire défiler les cartes du jeu mélangé les unes après les autres. On a du coup accès aux trois fonctions suivantes pour manipuler le jeu de cartes $\mathtt{j}$:

- `Val` (j) : renvoie la valeur de la carte visible au dessus du paquet de cartes.
- `Next` (j) : met la carte du dessus au fond du paquet. Par exemple, si les cartes sont de haut en bas dans l'ordre [2,52,13,42,8], après elles sont dans l'ordre [52,13,42,8,2].
- `Put` (j) : retire la carte du dessus du paquet et la pose sur la table.

Le but est donc de faire `Put` quand le 52 est au dessus, puis quand le 51 est au dessus... Pour avoir la bonne carte au dessus, il va falloir faire des `Next` encore et encore.

1. Proposez une structure représentant l'ensemble des cartes permettant ensuite d'implémenter les fonction du jeu.
2. Écrivez les fonctions `Val` et `Next` avec votre structure.
3. Proposez un algorithme qui résout une partie de solitaire en utilisant `Put` sur les cartes dans le bon ordre.
4. (Bonus) Quelle est la complexité de votre algorithme dans le meilleur et le pire des cas ?

Exercice 4. *Inclusion sur les arbres - CC3 22/23.*

Nous rappelons les fonctions suivantes sur les arbres binaires et généraux :

- `EstVide` ($\mathfrak{t}$) : renvoie VRAI si $\mathfrak{t}$ est l'arbre vide.
- `SAG` ($\mathfrak{t}$) : renvoie $\mathfrak{g}$ si $\mathfrak{t} = (\mathfrak{g}, \mathfrak{d})$, et n'est pas défini sinon.
- `SAD` ($\mathfrak{t}$) : renvoie $\mathfrak{d}$ si $\mathfrak{t} = (\mathfrak{g}, \mathfrak{d})$, et n'est pas défini sinon.
- `Val` ($\mathfrak{t}$) : renvoie la valeur de l'arbre $\mathfrak{t}$ si $\mathfrak{t}$ n'est pas l'arbre vide, et n'est pas défini sinon.
- `Enfants` ($\mathfrak{t}$) : renvoie la liste des enfants de $\mathfrak{t}$ si $\mathfrak{t}$ est un arbre général non vide, et n'est pas défini sinon.

On dit qu'un arbre $\mathfrak{t}_1$ est inclus dans $\mathfrak{t}_2$ si on peut compléter $\mathfrak{t}_1$ en ajoutant des arbres aux extrémités de $\mathfrak{t}_2$. Dans le cas des arbres binaires, cela revient à remplacer les emplacements vides de $\mathfrak{t}_1$ par de nouveaux arbres. Dans le cas des arbres généraux, cela revient à ajouter des arbres dans n'importes quelles listes d'enfants, et ce à n'importe quel emplacement.

1. Écrivez une fonction récursive `InclusAB` $(\mathfrak{t}_1, \mathfrak{t}_2)$ qui renvoie vrai si et seulement si l'arbre binaire $\mathfrak{t}_1$ est inclus dans l'arbre binaire $\mathfrak{t}_2$.
2. Écrivez une fonction récursive `InclusAG` $(\mathfrak{t}_1, \mathfrak{t}_2)$ qui renvoie vrai si et seulement si l'arbre général $\mathfrak{t}_1$ est inclus dans l'arbre général $\mathfrak{t}_2$. On supposera que dans chaque arbre, les valeurs dans les nœuds sont deux à deux différentes.

Exercice 5. *Construction d'un ABR de hauteur minimale - CC3 22/23.*

1. Si un arbre binaire de recherche contient n nœuds, quelle est la hauteur minimale possible pour cet arbre ?
2. Soit un noeud d'un ABR tel que toutes les feuilles depuis ce noeud sont à la même hauteur. Quelle est la différence entre le nombre de noeuds dans le sous arbre gauche et le nombre de noeuds dans le sous arbre droite.
3. Étant donné un tableau trié de n entiers, construire un arbre binaire de recherche ayant une hauteur minimal. Quelle est la complexité de votre algorithme en fonction de n ?
4. Étant donné un tableau (pas nécessairement trié) de n entiers, construire un arbre binaire de recherche ayant une hauteur minimale. Quelle est la complexité de votre algorithme ?

Exercice 6. *Tri d'un tableau presque trié - CC3 22/23.*

Donné est un tableau d'entiers qui est presque trié (il n'y a qu'un élément qui est hors place). Par exemple $T_1 = \{1, 3, 4, 5, 2, 7\}$ ou $T_2 = \{1, 3, 7, 4, 5, 6\}$. Le but de cet exercice est de trouver un algorithme efficace (linéaire) pour trier un tel tableau.

1. Exécuter à la main les fonctions triParInsertion(T) et triParSélection(T) sur les tableaux T_1 et T_2. Montrer l'état du tableau après chaque exécution de la boucle extérieure.
2. Pour un tel tableau (presque trié), analyser la complexité (# comparaisons et # affectations) de ces deux algorithmes de tri en fonction de la taille du tableau n dans le pire des cas. (Pour obtenir un score parfait, dans votre analyse il faut prendre en compte la particularité d'un tableau presque trié et donner la complexité qui correspond à ce type de tableau et non pas à un tableau arbitraire).
3. Donner une fonction efficace (linéaire) qui trie un tableau presque trié.

Exercice 7. *Fonctions mutuellement récursives - CC2 21/22.*

1. Écrivez deux fonctions booléennes, `estPair` et `estImpair`, qui déterminent si un entier positif passé en argument est pair ou impair. Ces deux fonctions devront être mutuellement récursives. C'est à dire que l'appel récursif de `estPair` sera un appel à `estImpair`. Réciproquement, l'appel récursif de `estImpair` devra être un appel à `estPair`.
2. Motivez brièvement pourquoi ces deux fonctions terminent.
3. Ces fonctions sont-elles récursives terminales ?
4. Représentez l'évolution de la pile lors du calcul de `estPair(5)`.