COMPLEX - TME Projet
2024-2025

Énoncé

Les arbres cartésiens sont une structure de données qui combine les propriétés des arbres binaires de recherche et des tas. Ils ont été proposés par Jean Vuillemin en 1980. Voici un aperçu de leur structure et de leur fonctionnement :

Structure

- **Nœuds :** chaque nœud d'un arbre cartésien contient une clé (qui respecte l'ordre d'un arbre binaire de recherche) et une priorité (qui respecte l'ordre d'un tas).
- **Arbre binaire de recherche :** les nœuds sont organisés de manière à ce que, pour tout nœud, les clés de son sous-arbre gauche soient inférieures à sa clé, et celles de son sous-arbre droit soient supérieures.
- **Tas :** les nœuds sont également organisés selon la priorité, de sorte qu'un parent ait toujours une priorité inférieure à celle de ses enfants.

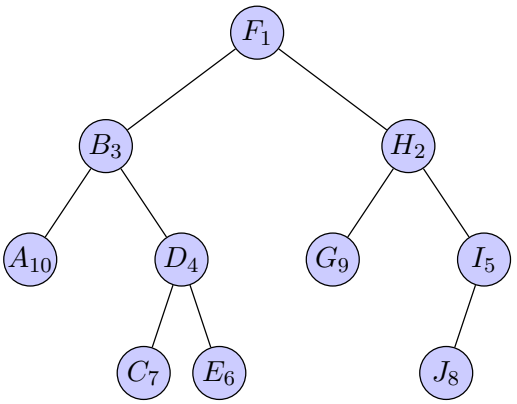


FIGURE 1 – Exemple d'arbre cartésien contenant les clés de $\{A, B, \ldots, J\}$ avec les valeurs des priorités dans $\{1, \ldots, 10\}$ données en indice

Propriétés

- **Équilibre probabiliste :** Les arbres cartésiens maintiennent une structure équilibrée de manière **probabiliste**. Cela signifie que les opérations d'insertion, de suppression et de recherche ont une complexité en temps moyenne de $O(\log n)$ même si dans le pire des cas, cela peut atteindre $O(n)$ (où n désigne le nombre de nœuds de l'arbre cartésien).
- **Insertion et suppression :** Lors de l'insertion, un nouveau nœud est placé comme un nœud feuille dans l'arbre binaire de recherche. Puis, si sa priorité est inférieure à celle de son parent, il est "élevé" à travers des rotations jusqu'à ce que l'ordre du tas soit respecté (*cf.* Exercice 3).
- **Recherche :** La recherche d'une clé suit les règles d'un arbre binaire de recherche, ce qui la rend efficace.

Les arbres cartésiens sont utilisés dans diverses applications où une structure de données dynamique et équilibrée est nécessaire, comme la gestion de fichiers, les bases de données et les algorithmes de traitement d'événements. Le but de ce projet est d'implanter et d'analyser cette structure de données et d'effectuer des tests d'efficacités.

Organisation du travail

- Le choix du langage de programmation pour implanter les différents algorithmes est libre mais vous devez pouvoir gérer les pointeurs et la mémoire.
- Le travail est à effectuer en binôme.
- La date de rendu des projets est le :
  - lundi 25 novembre 2024 pour le groupe du lundi ;
  - mercredi 27 novembre 2024 pour le groupe du mercredi et du jeudi ;

  cependant, comme il est toujours possible que vous ayez un imprévu, nous tolérons pour tous un délai de deux jours de retard et pouvez rendre le projet au plus tard le :
  - mercredi 27 novembre 2024 pour le groupe du lundi ;
  - vendredi 29 novembre 2024 pour le groupe du mercredi et du jeudi ;

  mais aucun délai supplémentaire ne sera accordé. Ils devront être envoyés par mail à votre chargé de TD ; le sujet de l'email doit être de la forme,

  `[CPLX] Projet, NOMbinôme1-NOMbinôme2`

  Votre livraison sera constituée d'une archive tar.gz qui doit comporter un rapport décrivant vos choix d'implémentation et votre code ainsi que la description des tests de validation et les réponses aux questions.
- Pour la notation de ce projet, nous pourrons utiliser des outils anti-plagiat ainsi que des outils de détection d'écriture générée par des modèles de langage, tels que ChatGPT.
- La maîtrise de tous les aspects de votre projet sera vérifiée lors d'une soutenance d'environ 15 minutes. Elle est prévue lors des deux séances de TME (de chaque groupe) de la semaine du 2 décembre 2024.

Travail à réaliser

Exercice 1 : Arbres cartésiens - Premières propriétés

1.a] Construire un arbre cartésien dont les nœuds sont donnés par la liste suivante (la lettre représentant la clé du nœud et l'entier sa priorité) :

$$(A : 5), (B : 3), (C : 8), (D : 2), (E : 6), (F : 7), (G : 9), (H : 1), (I : 10), (J : 12)$$

Existe-t'il plusieurs solutions ? Qu'en est-il pour un arbre cartésien dont toutes les priorités sont différentes (ce que nous ferons dans toute la suite du projet). Justifier votre réponse.

1.b] Considérer l'arbre binaire de recherche construit en insérant dans l'ordre les nœud dont les clés sont les suivants : H, D, B, A, E, F, C, G, I et J et comparer à l'arbre cartésien de la question précédente. Généraliser et démontrer ce résultat pour un arbre cartésien général (dont toutes les priorités sont différentes).

1.c] Programmer la structure de données d'un nœud (contenant la clé et la priorité, le pointeur vers le fils gauche, le pointeur vers le fils droit) avec les constructeurs et les fonctions associés.

1.d] Programmer la structure de données d'un arbre cartésien (avec au minimum un constructeur pour l'arbre cartésien vide, une fonction pour tester si un arbre cartésien est vide, pour accéder au fils droit/gauche d'un nœud donné).

1.e] Avec votre implantation, construire « manuellement » l'arbre cartésien de la question **1.a**.

Exercice 2 : Recherche dans un arbre cartésien

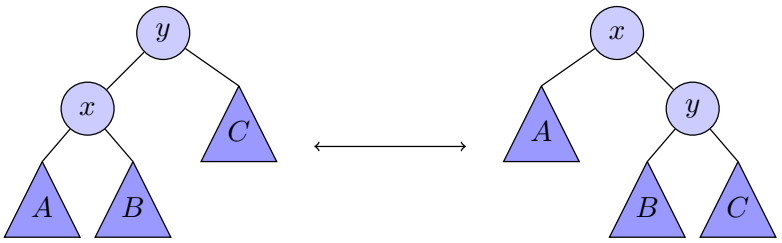
2.a] Implanter l'algorithme de recherche d'un nœud contenant une clé donnée dans un arbre cartésien. L'algorithme est identique à celui de recherche dans un arbre binaire de recherche.

2.b] Donner la complexité de cet algorithme en fonction de la profondeur du nœud recherché (en cas de recherche fructueuse) ou de la profondeur de son successeur ou prédécesseur du nœud recherché (en cas de recherche infructueuse).

Exercice 3 : Insertion dans un arbre cartésien

3.a] Montrer, sur l'exemple de la question **1.a**. que l'insertion d'un nœud dans un arbre cartésien en suivant la méthode d'insertion dans un arbre binaire de recherche, peut résulter en un arbre qui ne vérifie plus la propriété de tas.

Pour réparer cela, l'insertion dans un arbre cartésien va se faire initialement comme dans un arbre binaire de recherche mais devra effectuer des *rotations* d'arbre (voir le diagramme suivant pour une rotation au nœud x) pour rétablir la propriété de tas.



Une telle rotation préserve la propriété d'arbre binaire mais permet de rétablir la propriété de tas. Désormais pour insérer un nœud z dans un arbre cartésien, il sera inséré comme dans un arbre binaire de recherche mais une fois cela effectué, tant que sa priorité est inférieure à celle de son nœud parent, effectuer une rotation au nœud z. Chaque rotation diminue la profondeur du nœud z de 1 et augmente celle de son nom parent de 1. Les rotations peuvent être effectuées en temps constant puisqu'il s'agit simplement de manipulation de pointeurs.

3.b] Donner la complexité de cet algorithme en fonction de la profondeur de son prédécesseur.

3.c] Implanter cet algorithme.

3.d] Appliquer votre algorithme pour créer les arbres cartésiens obtenus en insérant les nœuds suivants dans l'ordre donné :

1. $(A:5),(B:3),(C:8),(D:2),(E:6),(F:7),(G:9),(H:1),(I:10),(J:12)$
2. $(H:1),(G:9),(A:5),(B:3),(D:2),(F:7),(C:8),(J:12),(I:10),(E:6)$
3. $(E:6),(H:1),(B:3),(D:2),(C:8),(F:7),(G:9),(J:12),(A:5),(I:10)$

et vérifier que vous obtenez bien à chaque fois le même arbre cartésien que dans la question **1.a**.

Exercice 4 : Suppression dans un arbre cartésien

4.a] Expliquer pourquoi pour supprimer un nœud z dans un arbre cartésien, il est possible de faire des rotations dans l'autre sens avec son fils qui a la priorité la plus faible jusqu'à l'amener à une feuille de l'arbre où il peut être supprimer. Chaque rotation augmente alors la profondeur du nœud z de 1 et diminue celle de son nom fils qui a la priorité la plus faible de 1.

4.b] Donner la complexité de l'algorithme obtenu.

4.c] Implanter cet algorithme.

4.d] Appliquer votre algorithme pour créer les arbres cartésiens obtenus en supprimant successivement les nœuds suivants dans l'arbre cartésien de la question **1.a** :

1. $(A : 5)$
2. $(J : 12)$,
3. $(H : 1)$,

Exercice 5 : Priorités aléatoires – Aspect expérimental

Un arbre cartésien *aléatoire* est un arbre dans lequel les priorités sont tirés uniformément aléatoirement dans l'intervalle $[0, 1]$. Il est possible d'utiliser des entiers dans un intervalle suffisamment grand pour espérer que tous les nœuds auront des clés différentes avec une très forte probabilité.

Pour insérer une nouvelle clé dans un arbre cartésien aléatoire, une priorité est tirée uniformément aléatoirement et c'est le nœud formé de cette clé et de cette priorité qui est inséré dans l'arbre.

5.a] Effectuer une analyse très détaillée d'efficacité expérimentale de votre structure de données pour un tel arbre cartésien aléatoire. En particulier,

- expliquer quelles métriques sont pertinentes pour évaluer l'équilibre et l'efficacité d'un arbre cartésien par rapport à d'autres structures de données ;
- faire une analyse de ces métriques sur des arbres cartésiens avec un nombre de nœuds le plus grand possible (pour votre implantation) ;
- donner un exemple d'une séquence d'insertion qui pourrait déséquilibrer un arbre binaire de recherche classique et expliquez comment un arbre cartésien se défend contre cela grâce à sa structure aléatoire ;
- Discuter, le cas échéant, des implications de la collision des priorités sur la structure de l'arbre et sur les performances des opérations.

Exercice 6 : Priorités aléatoires – Aspect théorique

L'objectif de cet exercice est de démontrer que la profondeur moyenne d'un nœud d'un arbre cartésien à n nœuds est en $O(\log n)$. Considérons un tel arbre et x_k le nœud dont la clé est la k-ième plus petite de l'arbre. Notons X_{ij} la variable aléatoire indiquant que x_i est un ancêtre propre de x_j (pour le tirage aléatoire des priorités).

6.a] Montrer que la profondeur moyenne de x_k est égale à $\sum_{i=1}^{n} \mathbb{E}(X_{ik})$.

6.b] Notons $X(i, k)$ l'ensemble des nœuds $\{x_i, x_{i+1}, \ldots, x_k\}$ ou $\{x_k, x_{k+1}, \ldots, x_i\}$ suivant selon $i < k$ ou $i > k$. Montrer que $X_{ik} = 1$ si et seulement si x_i est le nœud qui a la plus petite priorité dans $X(i, k)$.

6.c] Conclure (en vous inspirant de la preuve de la complexité en moyenne du tri rapide vue en cours).