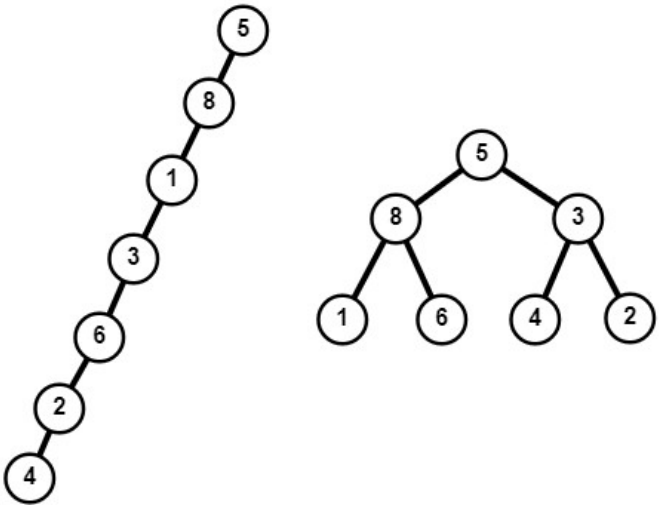
Auteur : Rehane Souhila

Université du Havre Normandie

Compte rendu du TP2 :
Sur les arbres binaires de recherche



Année Universitaire 2024/2025

Table des matières

Auteur : Rehane Souhila

I. Introduction :

L'objectif de ce TP est de mettre en pratique l'abstraction du type de données dictionnaire avec un arbre binaire de recherche. Puis, comparer l'efficacité de celui-ci avec une liste chainée.

Nous débuterons par une explication succincte de l'arbre binaire de recherche, mettant en lumière ses caractéristiques principales et son mode de fonctionnement. Ensuite, nous plongerons dans les détails des méthodes d'ajouter, de suppression, de chercher, et d'afficher ,dans un arbre binaire de recherche, afin de bien appréhender son utilisation pratique.

En n, nous analyserons les temps d'exécution des opérations d'insertion et de suppression dans une liste chainée, que nous comparerons ceux d'un arbre binaire de recherche. Cette analyse comparative nous permettra d'évaluer les forces et les faiblesses de chaque structure.

II. Description des classes et de leurs rôles :

II.1. Classe Element :

La classe `Element` représente un élément dans notre structure de données d'arbre binaire de recherche. Chaque élément contient une clé unique et une valeur associée. La clé est utilisée pour ordonner les éléments dans l'arbre binaire, tandis que la valeur représente l'information ou les données que nous voulons stocker.

II.2. Classe Nœud :

La classe `Nœud` représente un élément d'un arbre binaire, stockant une référence vers un élément (`Element`) ainsi que deux références vers d'autres nœuds. Ces nœuds, désignés par les attributs `gauche` et `droit`, représentent respectivement le sous-arbre gauche et le sous-arbre droit du nœud actuel. Chaque instance de la classe `Nœud` agit comme un maillon dans la structure de l'arbre binaire, reliant les éléments et les sous-arbres de manière hiérarchique.

II.3. Classe ArbreBinaire :

La classe ArbreBinaire représente une structure de données d'arbre binaire de recherche. Elle est caractérisée par : la racine de l'arbre, de type Nœud, qui représente le premier élément de l'arbre, et la hauteur de l'arbre, qui indique la profondeur maximale de l'arbre à partir de la racine. L'arbre binaire de recherche est une structure où chaque nœud a deux enfants maximum. Il respecte une propriété d'ordre : les valeurs du sous-arbre gauche sont inférieures à celle du nœud, et celles du sous-arbre droit sont supérieures. ArbreBinaire offre des méthodes pour l'insertion, la recherche et la suppression, tout en préservant cette propriété.

III. Les méthodes de la classe ArbreBinaire :

III.1. Ajouter :

Cas de racine vide :
Si le nœud courant `n`'est nul, cela signifie que l'arbre est vide. Dans ce cas, un nouveau nœud contenant l'élément à insérer est créé et affecté à la racine. Ainsi, un nouvel arbre est formé et l'insertion est considérée comme réussie.

Comparaison des clés :
Si l'élément à insérer possède une clé inférieure à celle du nœud courant, l'insertion doit se faire dans le sous-barbe gauche.
À l'inverse, si la clé est supérieure, l'élément doit être inséré dans le sous-arbre droit.

Insertion dans le sous-arbre correspondant :

- Si le sous-arbre ciblé (gauche ou droit) est vide, un nouveau nœud est créé avec l'élément à insérer, puis rattaché comme fils du nœud courant.
- Si le sous-arbre n'est pas vide, la méthode appelle récursivement la fonction `ajouter` pour continuer la recherche de l'emplacement d'insertion approprié.

Gestion des doublons :
Si la clé de l'élément à insérer est identique à celle du nœud courant, cela signifie que l'élément est déjà présent dans l'arbre. Aucune insertion n'est effectuée et la méthode retourne `false` pour signaler l'échec à cause d'un doublon.

III.2. Chercher :

1. Vérification du nœud vide :
La méthode commence par vérifier si le nœud courant `nœud` n'est pas nul, et si l'élément qu'il contient n'est pas nul non plus. Cela permet de s'assurer que la recherche s'effectue uniquement sur des nœuds valides de l'arbre.

2. Comparaison de la clé de l'élément :
Si la clé de l'élément contenu dans le nœud courant est égale à la clé de l'élément cherché `clé`, cela signifie que l'élément a été trouvé à cet emplacement. Dans ce cas, l'élément est retourné.

3. Navigation dans l'arbre :

- Si la clé de l'élément cherché est supérieure à celle du nœud courant, cela indique que l'élément se trouve potentiellement dans le sous-arbre droit. La méthode appelle alors récursivement `chercher` avec le sous-arbre droit comme nouveau nœud courant.
- Si la clé cherchée est inférieure à celle du nœud courant, cela signifie que l'élément peut se trouver dans le sous-arbre gauche. La méthode appelle donc récursivement `chercher` avec le sous-arbre gauche.

4. Gestion de l'élément non trouvé :
Si aucun nœud valide correspondant à la clé cherchée n'est trouvé, la méthode retourne `null` pour indiquer que l'élément n'existe pas dans l'arbre.

III.3. Supprimer :

1. Vérification de l'arbre vide :
Si le nœud courant `n` est `null`, cela signifie que l'arbre est vide. Dans ce cas, la méthode retourne `null` pour indiquer que l'élément n'a pas été trouvé dans l'arbre.

2. Navigation dans l'arbre :
La méthode parcourt l'arbre en comparant la clé de l'élément à supprimer avec celle du nœud courant.

- Si la clé de l'élément à supprimer est inférieure à celle du nœud courant, la recherche continue dans le sous-arbre gauche.
- Si la clé de l'élément à supprimer est supérieure, la recherche continue dans le sous-arbre droit.

3. Suppression du nœud contenant l'élément :
Lorsqu'un nœud contenant l'élément à supprimer est trouvé, plusieurs scénarios peuvent se produire :

- Si le nœud est une feuille ou possède au maximum un successeur, il est simplement retiré de l'arbre.
- Si le nœud a deux successeurs, il est remplacé par le nœud ayant la plus petite valeur dans son sous-arbre droit, ce qui permet de maintenir l'ordre de l'arbre binaire de recherche.
  Une fois l'élément supprimé, la méthode marque la suppression du nœud avec succès.

4. Recherche du nœud le plus à gauche dans le sous-arbre droit :
La méthode `minimumValeur(Noeud n)` est utilisée pour localiser le nœud le plus à gauche dans le sous-arbre droit, représentant ainsi la valeur minimale de cet arbre.

III.4. La méthode d'affichage :

La méthode d'affichage effectue un parcours récursif de l'arbre binaire de recherche en suivant un parcours en ordre (infixe).

1. Affichage récursif :
La méthode commence par afficher tous les éléments du sous-arbre gauche.
Ensuite, elle affiche l'élément de la racine.
Enfin, elle affiche tous les éléments du sous-arbre droit.

2. Parcours en ordre :
Ce type de parcours traverse l'arbre de manière séquentielle, en affichant les éléments dans l'ordre croissant de leur valeur.
Cela permet de visualiser la structure de l'arbre tout en respectant l'organisation des éléments d'un arbre binaire de recherche.

IV. Complexité et interprétation des résultats :

IV.1. Meilleur des cas O(1) :

Ce cas se présente lorsque l'arbre est vide, ce qui permet une insertion, une suppression, une cherche ou un affichage direct de l'élément à la racine de l'arbre. Dans cette situation, la complexité temporelle est constante, garantissant des opérations rapides quel que soit le nombre d'éléments dans l'arbre.

IV.2. Pire des cas O(h) :

Dans le scénario le plus défavorable, la hauteur de l'arbre (h) atteint son maximum, ce qui se produit lorsque l'arbre est complètement déséquilibré et prend la forme d'une liste chaînée. Dans cette configuration, chaque nœud a uniquement un seul enfant, soit à gauche, soit à droite, conduisant à une hauteur égale au nombre total de nœuds dans l'arbre. Par conséquent, dans ce pire des cas, la complexité peut atteindre O(n), où n représente le nombre d'éléments dans l'arbre.

Pour **l'ajout** :Cette situation se produit lorsque les éléments sont insérés dans un ordre particulier, conduisant à un déséquilibre significatif de l'arbre.

Pour **la suppression** : se produit dans le cas d'un ordre total où chaque nœud dans l'arbre est relié au suivant de manière linéaire, formant ainsi une structure similaire à une liste chaînée.

Pour **la cherche** : se produit dans le cas où chaque nœud dans l'arbre doit être parcouru de manière linéaire pour atteindre le nœud recherché, formant ainsi une structure similaire à une liste chaînée.

Pour **l'affichage** : la complexité de l'affichage préfixé sera également O(n), car tous les nœuds doivent être visités une fois.

IV.3. Moyen des cas O(logn) :

Dans le cas moyen, en supposant un arbre équilibré, la complexité temporelle des opérations sur un arbre binaire de recherche est O(logn) .

Pour **l'insertion** : lorsque l'arbre est équilibré, la hauteur de l'arbre est logarithmique par rapport au nombre d'éléments (O(log n)), où chaque nouvel élément est inséré à une profondeur relativement faible dans l'arbre.

Pour **la suppression** : Bien que la méthode de de suppression puisse nécessiter l'utilisation de la méthode « minimum », ajoutant ainsi un coût supplémentaire, ce dernier peut être considéré comme

négligeable étant donné que la complexité globale de la méthode de « suppressionRec » dépend principalement de la hauteur équilibrée de l'arbre.

Pour **la cherche** : complexité est obtenue grâce à la nature de l'arbre binaire de recherche équilibré, où chaque comparaison de clé permet de diviser l'espace de recherche par deux, réduisant ainsi le nombre de nœuds à parcourir à chaque étape. En conséquence, la recherche progresse de manière efficace à travers l'arbre, suivant le chemin optimal pour trouver le nœud recherché.

IV.4. Interprétations des données :

	Méthode insertion (ms)	Méthode suppression (ms)
Liste chainée	0	54
Arbre Binaire de Recherche	1	2

Table 1 : Comparaison de la durée moyenne d'exécution des méthodes insertion et suppression dans des listes chainées et des arbres binaires de recherche.

Commentaire :

Tout d'abord, en ce qui concerne l'opération d'insertion, il est à noter que les deux structures de données offrent des performances rapides. Cependant, la liste chaînée est plus efficace. Cela est dû à la nature séquentielle de la liste chaînée (on insère directement à la tête de la liste à chaque fois), tandis que l'arbre binaire nécessite des opérations de recherche et d'ajustement pour maintenir son organisation hiérarchique.

En revanche, concernant l'opération de suppression, l'arbre binaire de recherche surpasse nettement la liste chaînée en termes de performances. Alors que l'arbre binaire prend en moyenne 2 millisecondes pour effectuer une suppression, la liste chaînée nécessite en moyenne 54 millisecondes. Cette différence est attribuée à la structure des deux types de données. L'arbre binaire de recherche offre une suppression efficace grâce à ses opérations de réorganisation d'arbre, tandis que la suppression dans une liste chaînée nécessite souvent un parcours séquentiel pour localiser et retirer l'élément, ce qui entraîne un coût temporel plus élevé, surtout lorsque la liste est longue.

En conclusion, bien que les performances des méthodes d'insertion dans les deux structures soient comparables, l'efficacité de la suppression est nettement en faveur de l'arbre binaire de recherche.