Structures de données hiérarchiques

Cours de spécialité NSI de Terminale

D Pihoué

Lycée Camille Jullian Bordeaux

11 janvier 2024

Capacités attendues

1. Identifier des situations nécessitant une structure de données arborescente.
2. Évaluer quelques mesures des arbres binaires (taille, encadrement de la hauteur, etc.).
3. Calculer la taille et la hauteur d'un arbre.
4. Parcourir un arbre de différentes façons (ordres infixe, préfixe ou suffixe ; ordre en largeur d'abord).
5. Rechercher une clé dans un arbre de recherche, insérer une clef.

Définition

Un arbre est constitué de nœuds connectés entre eux par des arêtes depuis le **nœud racine** jusqu'aux **nœuds feuilles** en passant par des **nœuds internes**.

Définition

Un arbre est constitué de nœuds connectés entre eux par des arêtes depuis le **nœud racine** jusqu'aux **nœuds feuilles** en passant par des **nœuds internes**.

Les connexions entre les nœuds sont orientées du **nœud parent** vers ses **nœuds enfants**.

Définition

Un arbre est constitué de nœuds connectés entre eux par des arêtes depuis le **nœud racine** jusqu'aux **nœuds feuilles** en passant par des **nœuds internes**.

Les connexions entre les nœuds sont orientées du **nœud parent** vers ses **nœuds enfants**.

Généralement, un nœud transporte au moins une information.

Définition

Un arbre est constitué de nœuds connectés entre eux par des arêtes depuis le **nœud racine** jusqu'aux **nœuds feuilles** en passant par des **nœuds internes**.

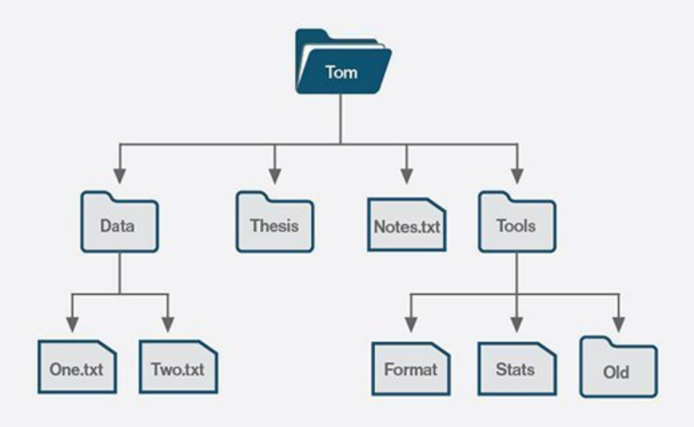
Les connexions entre les nœuds sont orientées du **nœud parent** vers ses **nœuds enfants**.

Généralement, un nœud transporte au moins une information.

Un arbre peut être implémenté par une structure récursive : chaque nœud contient la suite éventuellement vide de ses enfants qui sont eux-mêmes des nœuds.

Exemple

Identifiez sur cet exemple la nature des nœuds.



Vocabulaire

- La **profondeur** d'un nœud est le nombre d'arêtes du chemin le plus court jusqu'à la racine de l'arbre. Celle de la racine est donc 0.

Vocabulaire

- La **profondeur** d'un nœud est le nombre d'arêtes du chemin le plus court jusqu'à la racine de l'arbre. Celle de la racine est donc 0.
- La **hauteur** d'un arbre est la plus grande profondeur de cet arbre. Celle d'un arbre vide est 0.

Vocabulaire

- La **profondeur** d'un nœud est le nombre d'arêtes du chemin le plus court jusqu'à la racine de l'arbre. Celle de la racine est donc 0.
- La **hauteur** d'un arbre est la plus grande profondeur de cet arbre. Celle d'un arbre vide est 0.
- Le **degré** d'un nœud est le nombre de ses enfants.

Vocabulaire

- La **profondeur** d'un nœud est le nombre d'arêtes du chemin le plus court jusqu'à la racine de l'arbre. Celle de la racine est donc 0.
- La **hauteur** d'un arbre est la plus grande profondeur de cet arbre. Celle d'un arbre vide est 0.
- Le **degré** d'un nœud est le nombre de ses enfants.
- Le **degré** de l'arbre est le maximum des degrés de ses nœuds.

Vocabulaire

- La **profondeur** d'un nœud est le nombre d'arêtes du chemin le plus court jusqu'à la racine de l'arbre. Celle de la racine est donc 0.
- La **hauteur** d'un arbre est la plus grande profondeur de cet arbre. Celle d'un arbre vide est 0.
- Le **degré** d'un nœud est le nombre de ses enfants.
- Le **degré** de l'arbre est le maximum des degrés de ses nœuds.
- La **taille** d'un arbre est le nombre des nœuds qui le composent.

Vocabulaire

- La **profondeur** d'un nœud est le nombre d'arêtes du chemin le plus court jusqu'à la racine de l'arbre. Celle de la racine est donc 0.
- La **hauteur** d'un arbre est la plus grande profondeur de cet arbre. Celle d'un arbre vide est 0.
- Le **degré** d'un nœud est le nombre de ses enfants.
- Le **degré** de l'arbre est le maximum des degrés de ses nœuds.
- La **taille** d'un arbre est le nombre des nœuds qui le composent.

On identifie aussi les **ancêtres** et les **descendants** d'un nœud.

Définition

Un arbre est un arbre binaire s'il est

Définition

Un arbre est un arbre binaire s'il est

- soit vide, c'est-à-dire qu'il ne contient aucun nœud,

Définition

Un arbre est un arbre binaire s'il est

- soit vide, c'est-à-dire qu'il ne contient aucun nœud,
- soit formé d'un nœud **racine** et de deux **sous-arbres binaires gauche et droit**.

Définition

Un arbre est un arbre binaire s'il est

- soit vide, c'est-à-dire qu'il ne contient aucun nœud,
- soit formé d'un nœud **racine** et de deux **sous-arbres binaires gauche et droit**.

Il s'agit d'une définition récursive.

Définition

Un arbre est un arbre binaire s'il est

- soit vide, c'est-à-dire qu'il ne contient aucun nœud,
- soit formé d'un nœud **racine** et de deux **sous-arbres binaires gauche et droit**.

Il s'agit d'une définition récursive.

Un arbre binaire est donc un arbre de degré 2.

Définition

Un arbre est un arbre binaire s'il est

- soit vide, c'est-à-dire qu'il ne contient aucun nœud,
- soit formé d'un nœud **racine** et de deux **sous-arbres binaires gauche et droit**.

Il s'agit d'une définition récursive.

Un arbre binaire est donc un arbre de degré 2.

La racine d'un arbre binaire est reliée à la racine de chacun de ses deux sous-arbres.

Théorème

On désigne par n la taille d'un arbre binaire et par h sa hauteur.

La double inégalité $h+1 \leqslant n \leqslant 2^{h+1}-1$ est vraie.

Théorème

On désigne par n la taille d'un arbre binaire et par h sa hauteur.

La double inégalité $h+1 \leqslant n \leqslant 2^{h+1}-1$ *est vraie.*

Identifiez un exemple d'arbre binaire pour chacun des deux cas extrêmes.

Définition

On appelle algorithme de **parcours** l'organisation d'une visite de chaque nœud de l'arbre une seule fois en lui appliquant un traitement.

Définition

On appelle algorithme de **parcours** l'organisation d'une visite de chaque nœud de l'arbre une seule fois en lui appliquant un traitement.

On distingue deux approches :

Définition

On appelle algorithme de **parcours** l'organisation d'une visite de chaque nœud de l'arbre une seule fois en lui appliquant un traitement.

On distingue deux approches :

- le parcours en largeur d'abord, BFS pour *Breadth First Search*, qui explore l'arbre sur chaque niveau avant de descendre d'un niveau ;

Définition

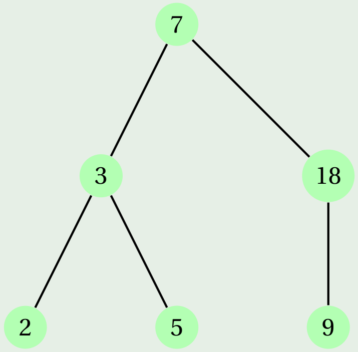
On appelle algorithme de **parcours** l'organisation d'une visite de chaque nœud de l'arbre une seule fois en lui appliquant un traitement.

On distingue deux approches :

- le parcours en largeur d'abord, BFS pour *Breadth First Search*, qui explore l'arbre sur chaque niveau avant de descendre d'un niveau ;
- le parcours en profondeur d'abord, DFS pour *Deep First Search*, qui explore chaque sous-arbre jusqu'à parvenir à une feuille avant de remonter pour parcourir les autres descendants qui n'ont pas encore été vus.

Exemple

Citez un ordre de visite des nœuds de l'arbre binaire ci-dessous correspondant à un parcours en largeur d'abord puis à un parcours en profondeur d'abord.



Un algorithme itératif pour le parcours en largeur d'abord

```
créer F une file vide
enfiler la racine de l'arbre
Tant que F n'est pas vide Faire
  n ←— défiler F
  traiter n
  enfiler tous les enfants de n
Fin Tant que
```

Trois algorithmes **récursifs** sont possibles pour le parcours en profondeur d'abord.

Trois algorithmes **récursifs** sont possibles pour le parcours en profondeur d'abord.

parcours **préfixe**

```
fonction parcours(arbre)
  Si arbre vide Alors
    Renvoyer rien
  Sinon
    Traiter la racine
    parcours(enfant gauche)
    parcours(enfant droit)
  Fin Si
```

parcours **infixe**

```
fonction parcours(arbre)
  Si arbre vide Alors
    Renvoyer rien
  Sinon
    parcours(enfant gauche)
    Traiter la racine
    parcours(enfant droit)
  Fin Si
```

parcours **suffixe**

```
fonction parcours(arbre)
  Si arbre vide Alors
    Renvoyer rien
  Sinon
    parcours(enfant gauche)
    parcours(enfant droit)
    Traiter la racine
  Fin Si
```

À retenir

On retiendra pour les parcours en profondeur d'abord.

À retenir

On retiendra pour les parcours en profondeur d'abord.

- Avec l'**ordre préfixe** un nœud est traité **avant** ses enfants.

À retenir

On retiendra pour les parcours en profondeur d'abord.

- Avec l'**ordre préfixe** un nœud est traité **avant** ses enfants.
- Avec l'**ordre infixe** un nœud est traité **entre** ses enfants.

À retenir

On retiendra pour les parcours en profondeur d'abord.

- Avec l'**ordre préfixe** un nœud est traité **avant** ses enfants.
- Avec l'**ordre infixe** un nœud est traité **entre** ses enfants.
- Avec l'**ordre suffixe** un nœud est traité **après** ses enfants.

À retenir

On retiendra pour les parcours en profondeur d'abord.

- Avec l'**ordre préfixe** un nœud est traité **avant** ses enfants.
- Avec l'**ordre infixe** un nœud est traité **entre** ses enfants.
- Avec l'**ordre suffixe** un nœud est traité **après** ses enfants.

Le choix du type de parcours dépend du problème à résoudre et beaucoup de problèmes peuvent être résolus indifféremment par un parcours en largeur d'abord ou par un parcours en profondeur d'abord.

On va considérer que chaque nœud d'un arbre binaire contient un attribut particulier que l'on va appeler la **clé** du nœud.

On va considérer que chaque nœud d'un arbre binaire contient un attribut particulier que l'on va appeler la **clé** du nœud.

Les valeurs possibles pour cette clé doivent être comparables.

On va considérer que chaque nœud d'un arbre binaire contient un attribut particulier que l'on va appeler la **clé** du nœud.

Les valeurs possibles pour cette clé doivent être comparables.

Définition

Un arbre binaire est un **arbre binaire de recherche** ou ABR, ou aussi BST pour *Binary Search Tree* si, pour tout nœud de l'arbre :

On va considérer que chaque nœud d'un arbre binaire contient un attribut particulier que l'on va appeler la **clé** du nœud.

Les valeurs possibles pour cette clé doivent être comparables.

Définition

Un arbre binaire est un **arbre binaire de recherche** ou ABR, ou aussi BST pour *Binary Search Tree* si, pour tout nœud de l'arbre :

1. sa clé est supérieure ou égale à toutes les clés des nœuds du sous-arbre gauche,

On va considérer que chaque nœud d'un arbre binaire contient un attribut particulier que l'on va appeler la **clé** du nœud.

Les valeurs possibles pour cette clé doivent être comparables.

Définition

Un arbre binaire est un **arbre binaire de recherche** ou ABR, ou aussi BST pour *Binary Search Tree* si, pour tout nœud de l'arbre :

1. sa clé est supérieure ou égale à toutes les clés des nœuds du sous-arbre gauche,
2. sa clé est inférieure à toutes les clés des nœuds du sous-arbre droit.

Exemple

1. Représentez plusieurs arbres binaires de recherche ayant 4 nœuds avec les clefs 1, 2, 3 et 4.
2. Vérifiez que le parcours infixe renvoie les valeurs des clés dans l'ordre croissant.

Exemple

1. Représentez plusieurs arbres binaires de recherche ayant 4 nœuds avec les clefs 1, 2, 3 et 4.
2. Vérifiez que le parcours infixe renvoie les valeurs des clés dans l'ordre croissant.

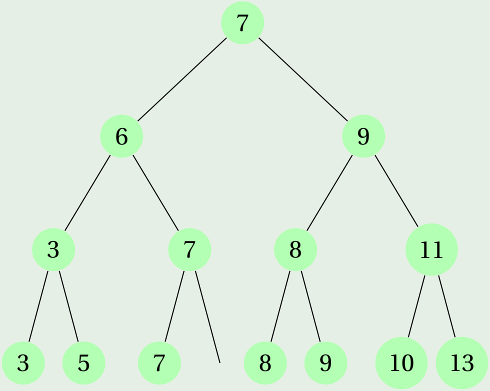
L'intérêt du choix d'organisation des clés dans un ABR est de permettre une recherche très efficace d'une clé.

Algorithme dichotomique de recherche d'une clé

```
fonction appartient(clef, arbre)
  Si arbre vide Alors
    Renvoyer Faux
  Sinon Si clef < racine Alors
    Renvoyer appartient(clef, sous arbre ..
   ..gauche)
  Sinon Si clef > racine Alors
    Renvoyer appartient(clef, sous arbre ..
   ..droit)
  Sinon
    Renvoyer Vrai
  Fin Si
```

Exemple

On donne cet ABR



Réaliser l'algorithme pour rechercher 7 puis 10 puis 5 puis 4.

Complexité en temps

- À chaque étape de l'algorithme, si l'arbre n'est pas vide et que la valeur de `clef` ne figure pas dans l'ABR alors on n'explore qu'un seul des sous-arbres gauche ou droit.

Complexité en temps

- À chaque étape de l'algorithme, si l'arbre n'est pas vide et que la valeur de `clef` ne figure pas dans l'ABR alors on n'explore qu'un seul des sous-arbres gauche ou droit.
- La complexité en temps dans le pire des cas est donc une **fonction linéaire de la hauteur** h de l'arbre.

Complexité en temps

- À chaque étape de l'algorithme, si l'arbre n'est pas vide et que la valeur de `clef` ne figure pas dans l'ABR alors on n'explore qu'un seul des sous-arbres gauche ou droit.
- La complexité en temps dans le pire des cas est donc une **fonction linéaire de la hauteur** h de l'arbre.
- Ainsi la complexité en temps de cet algorithme est, dans le pire des cas, une **fonction logarithmique de la taille** n de l'arbre.

Complexité en temps

- À chaque étape de l'algorithme, si l'arbre n'est pas vide et que la valeur de `clef` ne figure pas dans l'ABR alors on n'explore qu'un seul des sous-arbres gauche ou droit.
- La complexité en temps dans le pire des cas est donc une **fonction linéaire de la hauteur** h de l'arbre.
- Ainsi la complexité en temps de cet algorithme est, dans le pire des cas, une **fonction logarithmique de la taille** n de l'arbre.

En effet, on a vu qu'au maximum $n = 2^{h+1} - 1 \Leftrightarrow 2^{h+1} = n + 1$ et donc $h = \text{elog}(n+1) - 1$.

L'insertion d'un nœud par la connaissance de la valeur de sa clé est réalisée au fond de l'arbre, au niveau d'une feuille.

L'insertion d'un nœud par la connaissance de la valeur de sa clé est réalisée au fond de l'arbre, au niveau d'une feuille.

Algorithme dichotomique d'insertion

```
fonction inserer(clef, arbre)
  Si arbre vide Alors
    Affecter la clef à la racine
  Sinon Si clef <= racine Alors
    inserer(clef, sous-arbre gauche)
  Sinon
    inserer(clef, sous-arbre droit)
  Fin Si
```

Exemple

1. Réalisez cet algorithme pour insérer successivement dans un arbre initialement vide les clefs de valeurs 3, 1, 5, 7, 2, 4, 5, 3.
2. Réalisez cet algorithme pour insérer successivement dans un arbre initialement vide les clefs de valeurs 7, 3, 5, 18, 2 et 9.
3. Même consigne mais dans l'ordre 3, 5, 18, 7, 2 et 9.

Exemple

1. Réalisez cet algorithme pour insérer successivement dans un arbre initialement vide les clefs de valeurs 3, 1, 5, 7, 2, 4, 5, 3.
2. Réalisez cet algorithme pour insérer successivement dans un arbre initialement vide les clefs de valeurs 7, 3, 5, 18, 2 et 9.
3. Même consigne mais dans l'ordre 3, 5, 18, 7, 2 et 9.

Il existe d'autres algorithmes d'insertion, notamment pour maintenir la hauteur de l'arbre assez compacte afin d'optimiser le temps de recherche d'une clef.

En appliquant la même stratégie algorithmique le cas de base consiste à supprimer la racine d'un arbre mais il faut alors résoudre le problème du recollage des deux sous-arbres devenus orphelins en respectant la définition d'un ABR afin d'en maintenir la structure.

En appliquant la même stratégie algorithmique le cas de base consiste à supprimer la racine d'un arbre mais il faut alors résoudre le problème du recollage des deux sous-arbres devenus orphelins en respectant la définition d'un ABR afin d'en maintenir la structure.

Choix de la nouvelle racine

Pour cela, une clef possible pour se substituer à la racine supprimée est

- soit la plus petite clef du sous-arbre droit
- soit la plus grande clef du sous-arbre gauche.

En appliquant la même stratégie algorithmique le cas de base consiste à supprimer la racine d'un arbre mais il faut alors résoudre le problème du recollage des deux sous-arbres devenus orphelins en respectant la définition d'un ABR afin d'en maintenir la structure.

Choix de la nouvelle racine

Pour cela, une clef possible pour se substituer à la racine supprimée est

- soit la plus petite clef du sous-arbre droit
- soit la plus grande clef du sous-arbre gauche.

Déterminez mentalement lequel de ces choix va conserver la structure d'ABR choisie ici.

En retenant donc la deuxième solution, la suppression d'un nœud se décompose en quatre étapes.

En retenant donc la deuxième solution, la suppression d'un nœud se décompose en quatre étapes.

Algorithme général de suppression

1. Trouver le premier nœud dans l'ABR ayant la clef à supprimer, c'est-à-dire un arbre de racine ce nœud,

En retenant donc la deuxième solution, la suppression d'un nœud se décompose en quatre étapes.

Algorithme général de suppression

1. Trouver le premier nœud dans l'ABR ayant la clef à supprimer, c'est-à-dire un arbre de racine ce nœud,
2. déterminer le maximum du sous-arbre gauche de cet arbre,

En retenant donc la deuxième solution, la suppression d'un nœud se décompose en quatre étapes.

Algorithme général de suppression

1. Trouver le premier nœud dans l'ABR ayant la clef à supprimer, c'est-à-dire un arbre de racine ce nœud,
2. déterminer le maximum du sous-arbre gauche de cet arbre,
3. faire de la clef de ce maximum celle de sa racine,

En retenant donc la deuxième solution, la suppression d'un nœud se décompose en quatre étapes.

Algorithme général de suppression

1. Trouver le premier nœud dans l'ABR ayant la clef à supprimer, c'est-à-dire un arbre de racine ce nœud,
2. déterminer le maximum du sous-arbre gauche de cet arbre,
3. faire de la clef de ce maximum celle de sa racine,
4. supprimer le nœud du maximum du sous-arbre gauche en y plaçant son sous-arbre gauche.

En retenant donc la deuxième solution, la suppression d'un nœud se décompose en quatre étapes.

Algorithme général de suppression

1. Trouver le premier nœud dans l'ABR ayant la clef à supprimer, c'est-à-dire un arbre de racine ce nœud,
2. déterminer le maximum du sous-arbre gauche de cet arbre,
3. faire de la clef de ce maximum celle de sa racine,
4. supprimer le nœud du maximum du sous-arbre gauche en y plaçant son sous-arbre gauche.

Il faut cependant s'assurer que le sous-arbre gauche n'est pas vide pour réaliser les étapes ② à ④, mais dans ce cas c'est le sous-arbre droit qui constituera le nouvel arbre … même s'il est vide aussi il suffira donc d'y placer celui-ci.