Chapitre 1 : Les arbres

Option Informatique – MP

Lycée Chateaubriand

Table des matières

1 Les arbres
- Rappels
- Définition

2 Implémentation
- Définition du type
- Induction structurelle
- Quelques fonctions
- Arbre de calcul

3 Arbres binaires de recherche

4 Tas et liste de priorité

Plan

1. Les arbres
2. Implémentation
3. Arbres binaires de recherche
4. Tas et liste de priorité

1. Les arbres
   - Rappels
   - Définition
2. Implémentation
3. Arbres binaires de recherche
4. Tas et liste de priorité

1 Les arbres
Rappels
2 Implémentation
3 Arbres binaires de recherche
4 Tas et liste de priorité

Introduction

Cette année nous reprenons la notion d'arbre qui a été développée l'année passée. On va étudier la manière dont les arbres permettent de réaliser certaines structures de données.

Introduction

Cette année nous reprenons la notion d'arbre qui a été développée l'année passée. On va étudier la manière dont les arbres permettent de réaliser certaines structures de données.

Commençons par quelques rappels terminologiques sur les structures de données qui ont été vue en première année.

Structure de données abstraite

Définition 1 (Structure de données abstraite)

1. *On appelle structure de données abstraite un type de données muni d'opérations.*
2. *Si on peut modifier les données sans devoir toutes les réécrire, on dit que la structure est :*

Structure de données abstraite

Définition 1 (Structure de données abstraite)

1. *On appelle structure de données abstraite un type de données muni d'opérations.*
2. *Si on peut modifier les données sans devoir toutes les réécrire, on dit que la structure est :* ***impérative ou modifiable ou mutable****.*

Structure de données abstraite

Définition 1 (Structure de données abstraite)

1. *On appelle structure de données abstraite un type de données muni d'opérations.*
2. *Si on peut modifier les données sans devoir toutes les réécrire, on dit que la structure est :* ***impérative ou modifiable ou mutable****.*
3. *Si une modification de la structure nécessite de construire un nouvel objet on parle de structure de données :*

Structure de données abstraite

Définition 1 (Structure de données abstraite)

1. *On appelle structure de données abstraite un type de données muni d'opérations.*
2. *Si on peut modifier les données sans devoir toutes les réécrire, on dit que la structure est :* ***impérative ou modifiable ou mutable****.*
3. *Si une modification de la structure nécessite de construire un nouvel objet on parle de structure de données :* ***persistante ou immuable***

© Lycée Chateaubriand

Exemples

- En Caml, les listes sont des structures de données :

Exemples

- En Caml, les listes sont des structures de données : **persistante ou immuable**
- Les tableaux sont des structures de données :

Exemples

- En Caml, les listes sont des structures de données : **persistante ou immuable**
- Les tableaux sont des structures de données : **impérative ou modifiable ou mutable**.

Structure de données abstraite

Pour définir une structure de données, on précise les opérations que l'on veut pouvoir réaliser.

Par exemple, les files vues en première année qui sont une structure de données de type FIFO (*first in first out*), on désire les opérations suivantes :

Structure de données abstraite

Pour définir une structure de données, on précise les opérations que l'on veut pouvoir réaliser.

Par exemple, les files vues en première année qui sont une structure de données de type FIFO (*first in first out*), on désire les opérations suivantes :

- `creationVide` : qui créé une file vide.

Structure de données abstraite

Pour définir une structure de données, on précise les opérations que l'on veut pouvoir réaliser.

Par exemple, les files vues en première année qui sont une structure de données de type FIFO (*first in first out*), on désire les opérations suivantes :

- `creationVide` : qui créé une file vide.
- `estVide` : qui teste si une file est vide (et renvoie un booléen)

Structure de données abstraite

Pour définir une structure de données, on précise les opérations que l'on veut pouvoir réaliser.

Par exemple, les files vues en première année qui sont une structure de données de type FIFO (*first in first out*), on désire les opérations suivantes :

- `creationVide` : qui créé une file vide.
- `estVide` : qui teste si une file est vide (et renvoie un booléen)
- `ajouteFin` : qui ajoute un élément en fin de la file.

Structure de données abstraite

Pour définir une structure de données, on précise les opérations que l'on veut pouvoir réaliser.

Par exemple, les files vues en première année qui sont une structure de données de type FIFO (*first in first out*), on désire les opérations suivantes :

- `creationVide` : qui créé une file vide.
- `estVide` : qui teste si une file est vide (et renvoie un booléen)
- `ajouteFin` : qui ajoute un élément en fin de la file.
- `retireTete` : qui renvoie le premièr élément d'une file non vide. Cela enlève cet élément.

ATTENTION

Attention

La notion de structure de données abstraite ne doit pas être confondue avec l'implémentation d'une structure de données.
On peut imaginer que Alice code une structure de données et elle précise juste dans la documentation, les fonctions que l'on peut utiliser (ainsi que leur complexité la plupart du temps). Si Bob veut utiliser cette structure de données dans son programme, il ne va pas chercher à comprendre comment Alice a codé ses fonctions. Il va se contenter d'utiliser les fonctions définies. Il l'utilise comme une « boite noire ».

1 Les arbres
- Définition

2 Implémentation

3 Arbres binaires de recherche

4 Tas et liste de priorité

Défintion

Définition 2

Un arbre (enraciné) est un ensemble fini non vide A avec un élément r que l'on appelle la racine.
Il est muni d'une application $p : A \setminus \{r\} \to A$ telle que pour tout $x \in A \setminus \{r\}$, il existe $k \in \mathbb{N}$ tel que $r = p^k(x)$.

La plupart du temps, on se donne de plus un ensemble B (ensemble des étiquettes) et une application de A dans B.

Terminologie

- L'élément r s'appelle la racine de l'arbre.

Terminologie

- L'élément r s'appelle la racine de l'arbre.
- Pour tout élément $x \neq r$, l'élément $p(x)$ s'appelle le père de x. Il est unique. Il arrive que, par convention, on note aussi $p(r) = r$.

Terminologie

- L'élément r s'appelle la racine de l'arbre.
- Pour tout élément $x \neq r$, l'élément $p(x)$ s'appelle le père de x. Il est unique. Il arrive que, par convention, on note aussi $p(r) = r$.
- Soit x un élément de A, les antécédents de x par p, $p^{-1}(\{x\}) = \{y \in A \mid p(y) = x\}$ s'appellent les fils de x.

Terminologie

- L'élément r s'appelle la racine de l'arbre.
- Pour tout élément $x \neq r$, l'élément $p(x)$ s'appelle le père de x. Il est unique. Il arrive que, par convention, on note aussi $p(r) = r$.
- Soit x un élément de A, les antécédents de x par p, $p^{-1}(\{x\}) = \{y \in A \mid p(y) = x\}$ s'appellent les fils de x.
- On appelle nœuds les éléments de A.

Terminologie

- L'élément r s'appelle la racine de l'arbre.
- Pour tout élément $x \neq r$, l'élément $p(x)$ s'appelle le père de x. Il est unique. Il arrive que, par convention, on note aussi $p(r) = r$.
- Soit x un élément de A, les antécédents de x par p, $p^{-1}(\{x\}) = \{y \in A \mid p(y) = x\}$ s'appellent les fils de x.
- On appelle nœuds les éléments de A.
- On appelle feuille les éléments de A qui n'ont pas de fils et on appelle nœuds internes les nœuds qui ne sont pas des feuilles.

Terminologie

- L'élément r s'appelle la racine de l'arbre.
- Pour tout élément $x \neq r$, l'élément $p(x)$ s'appelle le père de x. Il est unique. Il arrive que, par convention, on note aussi $p(r) = r$.
- Soit x un élément de A, les antécédents de x par p, $p^{-1}(\{x\}) = \{y \in A \mid p(y) = x\}$ s'appellent les fils de x.
- On appelle nœuds les éléments de A.
- On appelle feuille les éléments de A qui n'ont pas de fils et on appelle nœuds internes les nœuds qui ne sont pas des feuilles.
- Soit x un élément de A. L'arité de x (ou le degré de x) est le nombre de ses fils. Les feuilles d'un arbre sont d'arité 0.

© Lycée Chateaubriand

Remarque

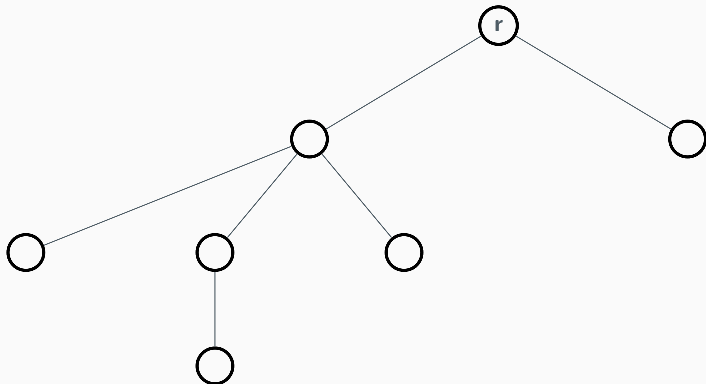
Il existe de nombreuses définitions équivalentes pour les arbres. Nous en verrons une autre dans le cours sur les graphes.

Remarque

Il existe de nombreuses définitions équivalentes pour les arbres. Nous en verrons une autre dans le cours sur les graphes. Dans ce chapitre nous parlerons d'arbre qui ne sont pas enracinés (les arbres libres).

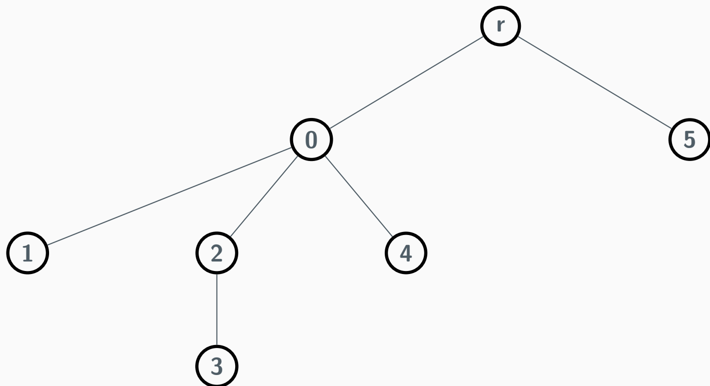
Exemple

Les arbres se représentent de la manière suivante.



Exemple

Le même arbre mais étiqueté



Terminologie

Définition 3

Soit A un arbre.

1. *La taille de l'arbre est le nombre de ses nœuds. On la note $|A|$.*

Définition 3

Soit A un arbre.

1. *La taille de l'arbre est le nombre de ses nœuds. On la note $|A|$.*
2. *Soit x un élément de A. La profondeur de x, notée $prof(x)$, vaut 0 si $x = r$ et vaut 1 de plus que celle de son père si $x \neq r$. On a donc*

$$prof(x) = \begin{cases} 0 & \text{si } x = r \\ 1 + prof(p(x)) & \text{sinon.} \end{cases}$$

© Lycée Chateaubriand

Terminologie

Définition 3

Soit A un arbre.

1. *La taille de l'arbre est le nombre de ses nœuds. On la note* $|A|$.
2. *Soit* x *un élément de* A. *La profondeur de* x, *notée* $prof(x)$, *vaut* 0 *si* $x = r$ *et vaut* 1 *de plus que celle de son père si* $x \neq r$. *On a donc*

$$prof(x) = \begin{cases} 0 & \text{si } x = r \\ 1 + prof(p(x)) & \text{sinon.} \end{cases}$$

3. *La hauteur d'un arbre est la profondeur maximale d'un de ses nœuds*

Remarques

- La profondeur d'un nœud $x \neq r$ est le plus petit entier k tel que $p^k(x) = r$. C'est donc le nombre d'arêtes traversées pour aller de la racine r au nœud x.
- En pratique on parle indifféremment de hauteur et de profondeur.
- Par convention l'arbre vide est de hauteur -1.

Remarques

- La profondeur d'un nœud $x \neq r$ est le plus petit entier k tel que $p^k(x) = r$. C'est donc le nombre d'arêtes traversées pour aller de la racine r au nœud x.
- En pratique on parle indifféremment de hauteur et de profondeur.
- Par convention l'arbre vide est de hauteur -1.

Attention

Il arrive de trouver une convention différente en prenant la racine de hauteur 1. Pensez à bien lire l'énoncé !

Arbre binaire

Définition 4

Si tous les nœuds ***internes*** *sont d'arité au plus deux (resp. exactement deux) on dit que l'arbre est un arbre binaire (resp. un arbre binaire entier).*

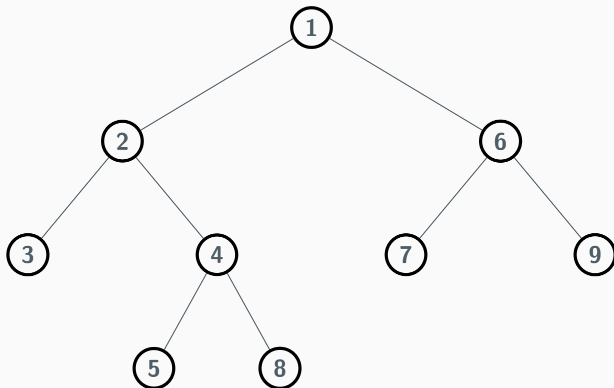
Définition 4

*Si tous les nœuds **internes** sont d'arité au plus deux (resp. exactement deux) on dit que l'arbre est un arbre binaire (resp. un arbre binaire entier).*

Remarque : Avec les notations précédentes, les arbres binaires sont des sous-cas des arbres. De ce fait on n'ordonne pas les deux fils. En pratique (voir les implémentations) on ordonne les deux fils en précisant le fils de gauche et le fils de droite. La structure obtenue n'est plus un sous-cas des arbres généraux mais un sous-cas des arbres planaires où, pour chaque noeud, la liste de ses fils est ordonnée.

Exemple

Considérons l'arbre suivant



Exemple

Dans cet exemple,

- La racine est 1
- 4 est le père 5
- 5 et 8 sont les fils de 4.
- 3 est une feuille
- 4 est de profondeur 2
- L'arbre est de hauteur 3 et de taille 9
- C'est un arbre binaire entier

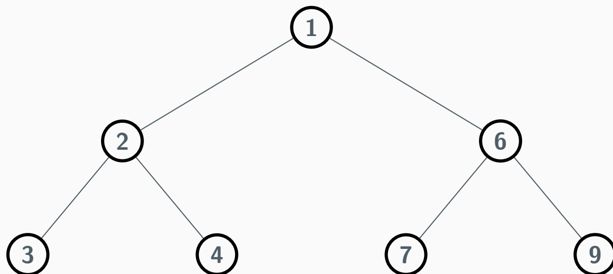
© Lycée Chateaubriand

Définition 5

On appelle arbre binaire complet (ou parfait) un arbre binaire entier tel que toutes les feuilles aient la même profondeur.

Définition 5

On appelle arbre binaire complet (ou parfait) un arbre binaire entier tel que toutes les feuilles aient la même profondeur.



© Lycée Chateaubriand

Exercices

- Quel est la taille d'un arbre binaire complet de hauteur h ? Combien a-t-il de feuilles ?

Exercices

- Quel est la taille d'un arbre binaire complet de hauteur h ? Combien a-t-il de feuilles ?

  La taille est
  $$1 + 2 + 4 + \cdots + 2^h = 2^{h+1} - 1.$$

Exercices

- Quel est la taille d'un arbre binaire complet de hauteur h ? Combien a-t-il de feuilles ?

La taille est

$$1 + 2 + 4 + \cdots + 2^h = 2^{h+1} - 1.$$

Le nombre de feuilles est 2^h.

- Justifier que si A est un arbre binaire de taille n et de hauteur h alors :

$$h + 1 \leq n \leq 2^{h+1} - 1.$$

  En déduire un encadrement de la hauteur en fonction de la taille. *On veut juste l'encadrement, nous verrons plus loin comment rédiger une preuve précise.*

Exercices

- Justifier que si A est un arbre binaire de taille n et de hauteur h alors :

$$h + 1 \leq n \leq 2^{h+1} - 1.$$

  En déduire un encadrement de la hauteur en fonction de la taille. *On veut juste l'encadrement, nous verrons plus loin comment rédiger une preuve précise.*

  Le plus petit arbre de hauteur h est un arbre « peigne » de taille $n = h + 1$. Le plus grand est l'arbre complet où $n = 2^{h+1} - 1$.

- Justifier que si A est un arbre binaire de taille n et de hauteur h alors :

$$h + 1 \leq n \leq 2^{h+1} - 1.$$

En déduire un encadrement de la hauteur en fonction de la taille. *On veut juste l'encadrement, nous verrons plus loin comment rédiger une preuve précise.*

Le plus petit arbre de hauteur h est un arbre « peigne » de taille $n = h + 1$. Le plus grand est l'arbre complet où $n = 2^{h+1} - 1$.

On a $\log_2(n + 1) \leq h + 1 \leq n$.

Exercice

Reprendre l'exercice précédent avec un arbre dont les nœuds sont d'arité maximale a.

1 Les arbres

2 **Implémentation**
- Définition du type
- Induction structurelle
- Quelques fonctions
- Arbre de calcul

3 Arbres binaires de recherche

4 Tas et liste de priorité

© Lycée Chateaubriand

Définition du type

1 Les arbres

2 Implémentation

Définition du type

3 Arbres binaires de recherche

4 Tas et liste de priorité

Définiton

Il y a de nombreuses manières d'implémenter des arbres binaires en CAML.

Pour la définition mathématique des arbres on utilise essentiellement la notion de père mais pour l'implémentation on utilise essentiellement la notion de fils.

Nous utiliserons des types récursifs.

Définiton

- Un arbre binaire étiqueté peut être implémenté par

```
type 'a arbreBin =
  |Vide
  |N of 'a arbreBin*'a*'a arbreBin;;
```

Ici une feuille est de la forme :

Définiton

- Un arbre binaire étiqueté peut être implémenté par

```
type 'a arbreBin =
  |Vide
  |N of 'a arbreBin*'a*'a arbreBin;;
```

Ici une feuille est de la forme : `N(Vide,x,Vide)`

Définiton

- Un arbre binaire étiqueté peut être implémenté par

```
type 'a arbreBin =
  |Vide
  |N of 'a arbreBin*'a*'a arbreBin;;
```

Ici une feuille est de la forme : `N(Vide,x,Vide)`

- Si l'arbre n'est plus binaire

```
type 'a noeud = Noeud of 'a * ('a noeud list);;
type 'a arbre = Vide | Racine of 'a noeud;;
```

Les feuilles sont les noeuds de la forme :

Définiton

- Un arbre binaire étiqueté peut être implémenté par

```
type 'a arbreBin =
  |Vide
  |N of 'a arbreBin*'a*'a arbreBin;;
```

  Ici une feuille est de la forme : `N(Vide,x,Vide)`

- Si l'arbre n'est plus binaire

```
type 'a noeud = Noeud of 'a * ('a noeud list);;
type 'a arbre = Vide | Racine of 'a noeud;;
```

  Les feuilles sont les noeuds de la forme : `Noeud(x,[])`

ATTENTION

Il existe plein d'autres implémentations récursives du même genre (voir plus loin et en exercice).

Il peut aussi être intéressant d'implémenter un arbre binaire par un tableau, nous le ferrons plus loin.

1 Les arbres

2 Implémentation

- Induction structurelle

3 Arbres binaires de recherche

4 Tas et liste de priorité

Induction structurelle

On voit que les arbres sont donc définis de manière récursive par :

- un cas de base qui est l'arbre vide.
- un constructeur qui correspond à un noeud dont les fils sont des arbres.

La plupart des preuves vont donc se faire par induction structurelle[1] qui est l'analogue de la récurrence.

1. Nous reviendrons plus tard sur une définition précise d'une induction structurelle.

Induction structurelle

Pour montrer qu'un prédicat $\mathscr{P}$ est vrai pour tous les arbres il faut :

- Montrer que le prédicat est vrai pour le cas de base.
- Montrer que le prédicat est vrai pour un arbre obtenu par application du constructeur en supposant qu'il l'était pour les données auxquelles on a appliquées le constructeur.

Exemple

Remontrons que si a est un arbre binaire de taille n et de hauteur h alors :

$$h + 1 \leq n \leq 2^{h+1} - 1.$$

- **Cas de base :**

Exemple

Remontrons que si a est un arbre binaire de taille n et de hauteur h alors :

$$h + 1 \leq n \leq 2^{h+1} - 1.$$

- **Cas de base :**
  Pour l'arbre vide, $n = 0$ et $h = -1$. On a bien

$$0 \leq 0 \leq 2^0 - 1$$

Exemple

Remontrons que si a est un arbre binaire de taille n et de hauteur h alors :

$$h + 1 \leq n \leq 2^{h+1} - 1.$$

- **Cas de base :**
  Pour l'arbre vide, $n = 0$ et $h = -1$. On a bien

  $$0 \leq 0 \leq 2^0 - 1$$

  Comme le cas de l'arbre vide est un peu pathologique on peut tester un arbre qui n'a qu'un noeud : $n = 1$ et $h = 0$.

Exemple

Remontrons que si a est un arbre binaire de taille n et de hauteur h alors :

$$h + 1 \leq n \leq 2^{h+1} - 1.$$

- **Cas de base :**
  Pour l'arbre vide, $n = 0$ et $h = -1$. On a bien

  $$0 \leq 0 \leq 2^0 - 1$$

  Comme le cas de l'arbre vide est un peu pathologique on peut tester un arbre qui n'a qu'un noeud : $n = 1$ et $h = 0$. On a bien :

  $$1 \leq 1 \leq 2 - 1.$$

Exemple

- **Induction :**
  Soit a un arbre non vide de la forme `N(g,x,d)`.On note n_g, n_d, h_g et h_d les tailles et hauteurs de d et g. Par hypothèses :

$$h_g + 1 \leq n_g \leq 2^{h_g+1} - 1 \text{ et } h_d + 1 \leq n_d \leq 2^{h_d+1} - 1.$$

© Lycée Chateaubriand

Exemple

- **Induction :**
  Soit a un arbre non vide de la forme `N(g,x,d)`.On note n_g, n_d, h_g et h_d les tailles et hauteurs de d et g. Par hypothèses :

$$h_g + 1 \leq n_g \leq 2^{h_g+1} - 1 \text{ et } h_d + 1 \leq n_d \leq 2^{h_d+1} - 1.$$

  En faisant la somme (et en ajoutant 1) :

$$h_d + h_g + 3 \leq n = n_d + n_g + 1 \leq 2^{h_g+1} + 2^{h_d+1} - 1.$$

© Lycée Chateaubriand

Exemple

- **Induction :**
  Soit a un arbre non vide de la forme `N(g,x,d)`.On note n_g, n_d, h_g et h_d les tailles et hauteurs de d et g. Par hypothèses :
  $$h_g + 1 \leq n_g \leq 2^{h_g+1} - 1 \text{ et } h_d + 1 \leq n_d \leq 2^{h_d+1} - 1.$$
  En faisant la somme (et en ajoutant 1) :
  $$h_d + h_g + 3 \leq n = n_d + n_g + 1 \leq 2^{h_g+1} + 2^{h_d+1} - 1.$$
  Maintenant $h = 1 + max(h_d, h_g)$ et $h_d + h_g \geq \max(h_g, h_d) - 1$ car h_g et h_d sont supérieurs ou égaux à -1.
  Cela donne que
  $$h_d + h_g + 3 \geq h + 1$$

Exemple

- **Induction :**
  Soit a un arbre non vide de la forme `N(g,x,d)`.On note n_g, n_d, h_g et h_d les tailles et hauteurs de d et g. Par hypothèses :

  $$h_g + 1 \leq n_g \leq 2^{h_g+1} - 1 \text{ et } h_d + 1 \leq n_d \leq 2^{h_d+1} - 1.$$

  En faisant la somme (et en ajoutant 1) :

  $$h_d + h_g + 3 \leq n = n_d + n_g + 1 \leq 2^{h_g+1} + 2^{h_d+1} - 1.$$

  Maintenant $h = 1 + max(h_d, h_g)$ et $h_d + h_g \geq \max(h_g, h_d) - 1$ car h_g et h_d sont supérieurs ou égaux à -1.
  Cela donne que

  $$h_d + h_g + 3 \geq h + 1$$

  De même,

  $$2^{h_d+1} \leq 2^h \quad \text{et} \quad 2^{h_g+1} \leq 2^h$$

  donc

  $$2^{h_g+1} + 2^{h_d+1} - 1 \leq 2^h + 2^h - 1 = 2^{h+1} - 1$$

Exemple

- **Induction :**
  Soit a un arbre non vide de la forme `N(g,x,d)`.On note n_g, n_d, h_g et h_d les tailles et hauteurs de d et g. Par hypothèses :

$$h_g + 1 \leq n_g \leq 2^{h_g+1} - 1 \text{ et } h_d + 1 \leq n_d \leq 2^{h_d+1} - 1.$$

  En faisant la somme (et en ajoutant 1) :

$$h_d + h_g + 3 \leq n = n_d + n_g + 1 \leq 2^{h_g+1} + 2^{h_d+1} - 1.$$

  Maintenant $h = 1 + max(h_d, h_g)$ et $h_d + h_g \geq \max(h_g, h_d) - 1$ car h_g et h_d sont supérieurs ou égaux à -1.
  Cela donne que

$$h_d + h_g + 3 \geq h + 1$$

  De même,

$$2^{h_d+1} \leq 2^h \quad \text{et} \quad 2^{h_g+1} \leq 2^h$$

  donc

$$2^{h_g+1} + 2^{h_d+1} - 1 \leq 2^h + 2^h - 1 = 2^{h+1} - 1$$

  Finalement

$$h + 1 \leq n \leq 2^{h+1} - 1.$$

Remarque

C'est ce même principe d'induction structurelle qui permet de d'assurer que les fonctions comme la taille ou la hauteur sont bien définies.

1 Les arbres

2 Implémentation

Quelques fonctions

3 Arbres binaires de recherche

4 Tas et liste de priorité

Fonctions

Nous allons donner quelques exemples de fonctions. Nous travaillerons avec le type d'arbre binaire `'a arbreBin`défini ci-dessus.

- Commençons par une fonction donnant la hauteur d'un arbre :

```
let rec hauteur a = match a with
  |Vide -> -1
  |N(g,_,d) -> 1 + max (hauteur d) (hauteur g);;
```

Fonctions

- Écrivons maintenant une fonction pour la taille d'un arbre.

- Écrivons maintenant une fonction pour la taille d'un arbre.

```
let rec taille a = match a with
  |Vide -> 0
  |N(g,_,d) -> 1 + (taille d) + (taille g);;
```

Exercices

Écrire une fonction qui calcule le nombre de nœuds internes et une qui calcule le nombre de feuilles.

Exercices

Écrire une fonction qui calcule le nombre de nœuds internes et une qui calcule le nombre de feuilles.

```
let rec feuille a = match a with
^^I| Vide -> 0
^^I| N(Vide,_,Vide) -> 1
^^I| N(g,_,d) -> (feuille g)+(feuille d);;
^^I
```

Écrire une fonction qui calcule le nombre de nœuds internes et une qui calcule le nombre de feuilles.

```
let rec feuille a = match a with
^^I| Vide -> 0
^^I| N(Vide,_,Vide) -> 1
^^I| N(g,_,d) -> (feuille g)+(feuille d);;
^^I
```

```
let rec noeudint a = match a with
^^I| Vide -> 0
^^I| N(Vide,_,Vide) -> 0
^^I| N(g,_,d) -> 1+ (noeudint g)+(noeudint d);;
```

1 Les arbres

2 Implémentation

Arbre de calcul

3 Arbres binaires de recherche

4 Tas et liste de priorité

Arbre de calcul

Les calculs algébriques peuvent se représenter dans des arbres. Par exemple

$$(2 + x) * (x - (3 - y))$$

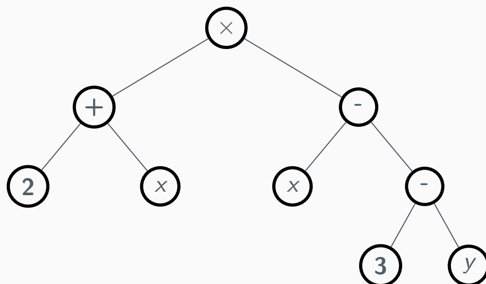
se représente par

Arbre de calcul

Les calculs algébriques peuvent se représenter dans des arbres. Par exemple

$$(2 + x) * (x - (3 - y))$$

se représente par



Arbre de calcul

La différence par rapport aux arbres étudiées pour le moment c'est que les étiquettes des feuilles et celles des nœuds internes n'ont pas le même type.

On utilise donc un type plus compliqué qui traite les feuilles différemment des noeuds internes.

```
type ('a,'b) arb =
^^I|F of 'a
^^I|N of ('a,'b) arb * 'b * ('a,'b) arb;;
^^I
```

Remarque

Il n'est plus nécessaire d'avoir l'arbre vide car :

-

Remarque

Il n'est plus nécessaire d'avoir l'arbre vide car :

- les feuilles sont déterminées par le constructeur `F`.
-

Remarque

Il n'est plus nécessaire d'avoir l'arbre vide car :

- les feuilles sont déterminées par le constructeur `F`.

- Une opération ne peut pas avoir un seul fils.

1 Les arbres

2 Implémentation

3 **Arbres binaires de recherche**

4 Tas et liste de priorité

1 Les arbres
2 Implémentation
3 Arbres binaires de recherche
4 Tas et liste de priorité