Définition

Un arbre binaire, est une structure de données *hiérarchique* (les éléments, appelés noeuds sont rangés par niveau) qui peut se définir récursivement.
En effet, un arbre binaire est

Exemples

Définition

Un arbre binaire, est une structure de données *hiérarchique* (les éléments, appelés noeuds sont rangés par niveau) qui peut se définir récursivement.
En effet, un arbre binaire est

- soit vide, on le note alors $\varnothing$

Exemples

Définition

Un arbre binaire, est une structure de données *hiérarchique* (les éléments, appelés noeuds sont rangés par niveau) qui peut se définir récursivement.
En effet, un arbre binaire est

- soit vide, on le note alors $\varnothing$
- soit un noeud (sag, r, sad) appelé racine où r est l'étiquette de la racine et sag et sad sont deux arbres binaires (le sous arbre gauche, et le sous arbre droit)

Exemples

Définition

Un arbre binaire, est une structure de données *hiérarchique* (les éléments, appelés noeuds sont rangés par niveau) qui peut se définir récursivement.
En effet, un arbre binaire est

- soit vide, on le note alors $\varnothing$
- soit un noeud (sag, r, sad) appelé racine où r est l'étiquette de la racine et sag et sad sont deux arbres binaires (le sous arbre gauche, et le sous arbre droit)

Exemples

L'arbre $(\varnothing, a, (\varnothing, b, \varnothing))$

C12 Arbres binaires

1. Définition

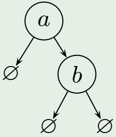
Définition

Un arbre binaire, est une structure de données *hiérarchique* (les éléments, appelés noeuds sont rangés par niveau) qui peut se définir récursivement.
En effet, un arbre binaire est

- soit vide, on le note alors $\varnothing$
- soit un noeud (sag, r, sad) appelé racine où r est l'étiquette de la racine et sag et sad sont deux arbres binaires (le sous arbre gauche, et le sous arbre droit)

Exemples

L'arbre $(\varnothing, a, (\varnothing, b, \varnothing))$



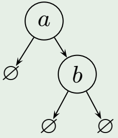
Définition

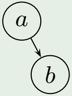
Un arbre binaire, est une structure de données *hiérarchique* (les éléments, appelés noeuds sont rangés par niveau) qui peut se définir récursivement.
En effet, un arbre binaire est

- soit vide, on le note alors $\varnothing$
- soit un noeud (sag, r, sad) appelé racine où r est l'étiquette de la racine et sag et sad sont deux arbres binaires (le sous arbre gauche, et le sous arbre droit)

Exemples

L'arbre $(\varnothing, a, (\varnothing, b, \varnothing))$





C12 Arbres binaires

1. Définition

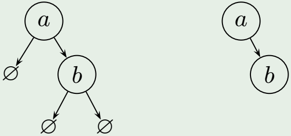
Définition

Un arbre binaire, est une structure de données *hiérarchique* (les éléments, appelés noeuds sont rangés par niveau) qui peut se définir récursivement.
En effet, un arbre binaire est

- soit vide, on le note alors $\varnothing$
- soit un noeud (sag, r, sad) appelé racine où r est l'étiquette de la racine et sag et sad sont deux arbres binaires (le sous arbre gauche, et le sous arbre droit)

Exemples

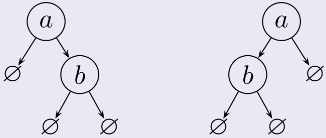
L'arbre $(\varnothing, a, (\varnothing, b, \varnothing))$



Représenté avec (à gauche) ou sans (à droite) les sous arbres vides.

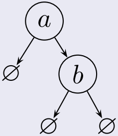
Remarques

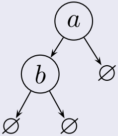
- ⚠ Les deux arbres ci-dessous sont différents !



Remarques

- ⚠ Les deux arbres ci-dessous sont différents !

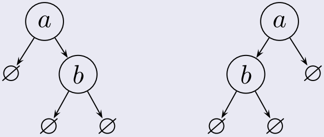




- On omet parfois de représenter les sous arbres vides, mais on doit garder à l'esprit qu'un noeud non vide est *toujours* un triplet. Et que donc les sous arbres gauche et droit même vide, sont toujours présents.

Remarques

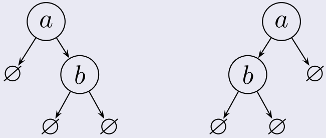
- Les deux arbres ci-dessous sont différents !



- On omet parfois de représenter les sous arbres vides, mais on doit garder à l'esprit qu'un noeud non vide est *toujours* un triplet. Et que donc les sous arbres gauche et droit même vide, sont toujours présents.
- Lorsque qu'un noeud a possède un sous arbre non vide dont la racine est b, on dit que a est le père de b et que b est le fils a.

Remarques

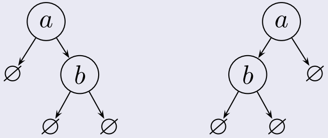
- ⚠ Les deux arbres ci-dessous sont différents !



- On omet parfois de représenter les sous arbres vides, mais on doit garder à l'esprit qu'un noeud non vide est *toujours* un triplet. Et que donc les sous arbres gauche et droit même vide, sont toujours présents.
- Lorsque qu'un noeud a possède un sous arbre non vide dont la racine est b, on dit que a est le père de b et que b est le fils a.
- Un noeud dont les deux sous arbres sont vides s'appelle une feuille.

Remarques

- ⚠ Les deux arbres ci-dessous sont différents !



- On omet parfois de représenter les sous arbres vides, mais on doit garder à l'esprit qu'un noeud non vide est *toujours* un triplet. Et que donc les sous arbres gauche et droit même vide, sont toujours présents.
- Lorsque qu'un noeud a possède un sous arbre non vide dont la racine est b, on dit que a est le père de b et que b est le fils a.
- Un noeud dont les deux sous arbres sont vides s'appelle une feuille.
- un noeud qui n'est pas une feuille s'appelle un noeud interne.

Définition récursive du nombre de noeuds et de la hauteur

- Le nombre de noeuds d'un arbre binaire A, noté $n(A)$, se définit récursivement par :

Définition récursive du nombre de noeuds et de la hauteur

- Le nombre de noeuds d'un arbre binaire A, noté $n(A)$, se définit récursivement par :

$$\left\{\begin{array}{lcll} n(A) & = & 0 & \text{si } A \text{ est vide} \\ n(A) & = & 1 + n(g) + n(d) & \text{si } A = (g, a, d) \end{array}\right.$$

Définition récursive du nombre de noeuds et de la hauteur

- Le nombre de noeuds d'un arbre binaire A, noté $n(A)$, se définit récursivement par :

$$\begin{cases} n(A) = 0 & \text{si } A \text{ est vide} \\ n(A) = 1 + n(g) + n(d) & \text{si } A = (g, a, d) \end{cases}$$

- La hauteur d'un arbre binaire A, noté $h(A)$, se définit récursivement par :

Définition récursive du nombre de noeuds et de la hauteur

- Le nombre de noeuds d'un arbre binaire A, noté $n(A)$, se définit récursivement par :

$$\begin{cases} n(A) & = & 0 & \text{si } A \text{ est vide} \\ n(A) & = & 1 + n(g) + n(d) & \text{si } A = (g, a, d) \end{cases}$$

- La hauteur d'un arbre binaire A, noté $h(A)$, se définit récursivement par :

$$\begin{cases} h(A) & = & -1 & \text{si } A \text{ est vide} \\ h(A) & = & 1 + \max(h(g), h(d)) & \text{si } A = (g, a, d) \end{cases}$$

Définition récursive du nombre de noeuds et de la hauteur

- Le nombre de noeuds d'un arbre binaire A, noté $n(A)$, se définit récursivement par :

$$\begin{cases} n(A) & = & 0 & \text{si } A \text{ est vide} \\ n(A) & = & 1 + n(g) + n(d) & \text{si } A = (g, a, d) \end{cases}$$

- La hauteur d'un arbre binaire A, noté $h(A)$, se définit récursivement par :

$$\begin{cases} h(A) & = & -1 & \text{si } A \text{ est vide} \\ h(A) & = & 1 + \max(h(g), h(d)) & \text{si } A = (g, a, d) \end{cases}$$

⚠ Certains auteurs prennent 0 comment hauteur de l'arbre vide.

Définition récursive du nombre de noeuds et de la hauteur

- Le nombre de noeuds d'un arbre binaire A, noté $n(A)$, se définit récursivement par :

$$\begin{cases} n(A) & = & 0 & \text{si } A \text{ est vide} \\ n(A) & = & 1 + n(g) + n(d) & \text{si } A = (g, a, d) \end{cases}$$

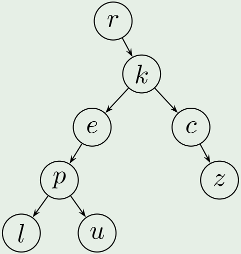
- La hauteur d'un arbre binaire A, noté $h(A)$, se définit récursivement par :

$$\begin{cases} h(A) & = & -1 & \text{si } A \text{ est vide} \\ h(A) & = & 1 + \max(h(g), h(d)) & \text{si } A = (g, a, d) \end{cases}$$

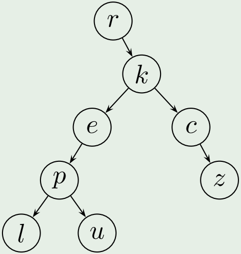
⚠ Certains auteurs prennent 0 comment hauteur de l'arbre vide.

- La profondeur d'un noeud est sa distance à la racine.

Exemple

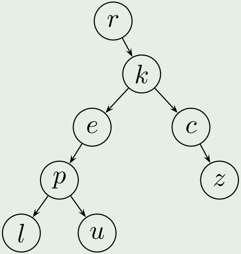


Exemple



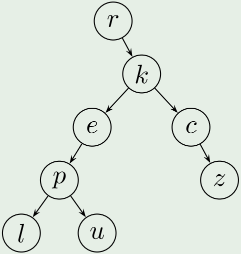
- Nommer les feuilles et les noeuds internes

Exemple



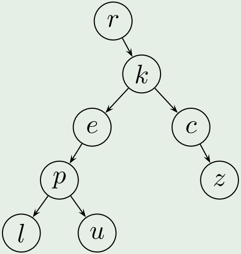
- Nommer les feuilles et les noeuds internes
- Donner le nombre de noeuds

Exemple



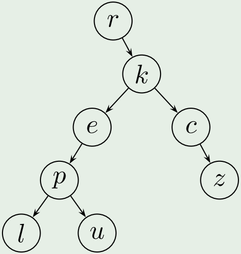
- Nommer les feuilles et les noeuds internes
- Donner le nombre de noeuds
- Donner la hauteur de cet arbre

Exemple



- Nommer les feuilles et les noeuds internes
- Donner le nombre de noeuds
- Donner la hauteur de cet arbre
- Donner un noeud de profondeur 2

Exemple



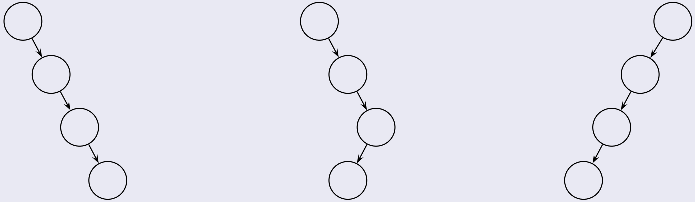
- Nommer les feuilles et les noeuds internes
- Donner le nombre de noeuds
- Donner la hauteur de cet arbre
- Donner un noeud de profondeur 2
- Donner l'écriture de cet arbre sous forme de triplet

Quelques cas particuliers

- Un arbre binaire est dit dégénéré lorsque tous les noeuds à l'exception des feuilles n'ont qu'un fils.

Quelques cas particuliers

- Un arbre binaire est dit dégénéré lorsque tous les noeuds à l'exception des feuilles n'ont qu'un fils.



Pour les arbres représentés à gauche et à droite on parle de *peigne*, à rapprocher de la liste chainée.

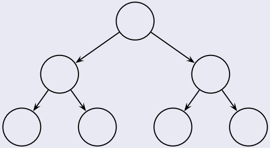
Quelques cas particuliers

Quelques cas particuliers

- Un arbre binaire est dit parfait lorsque tous les niveaux sont remplis :

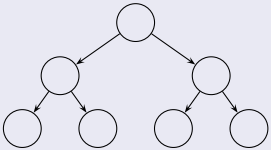
Quelques cas particuliers

- Un arbre binaire est dit parfait lorsque tous les niveaux sont remplis :



Quelques cas particuliers

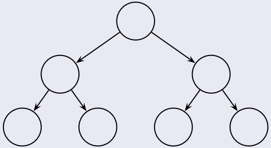
- Un arbre binaire est dit parfait lorsque tous les niveaux sont remplis :



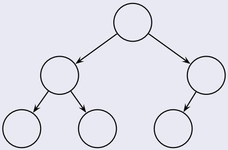
- Un arbre binaire est dit complet lorsque tous les niveaux à l'exception du dernier sont remplis et que le dernier niveau est rempli à parti de la gauche.

Quelques cas particuliers

- Un arbre binaire est dit parfait lorsque tous les niveaux sont remplis :



- Un arbre binaire est dit complet lorsque tous les niveaux à l'exception du dernier sont remplis et que le dernier niveau est rempli à parti de la gauche.



Nombre de sous abres vides

Le nombre de sous arbres vides d'un arbre binaire de taille n est $n + 1$.

Nombre de sous abres vides

Le nombre de sous arbres vides d'un arbre binaire de taille n est $n+1$.

Relation entre hauteur et taille

En notant n la taille et h la hauteur d'un arbre binaire, on a la relation suivante :

Nombre de sous abres vides

Le nombre de sous arbres vides d'un arbre binaire de taille n est $n+1$.

Relation entre hauteur et taille

En notant n la taille et h la hauteur d'un arbre binaire, on a la relation suivante :

$$h+1 \leqslant n \leqslant 2^{h+1}-1$$

Nombre de sous abres vides

Le nombre de sous arbres vides d'un arbre binaire de taille n est $n+1$.

Relation entre hauteur et taille

En notant n la taille et h la hauteur d'un arbre binaire, on a la relation suivante :

$$h + 1 \leqslant n \leqslant 2^{h+1} - 1$$

Remarque

On démontrera souvent une propriété $\mathcal{P}$, sur les arbres par récurrence (sur la hauteur ou la taille) ou par induction structurelle :

Nombre de sous abres vides

Le nombre de sous arbres vides d'un arbre binaire de taille n est $n + 1$.

Relation entre hauteur et taille

En notant n la taille et h la hauteur d'un arbre binaire, on a la relation suivante :

$$h + 1 \leqslant n \leqslant 2^{h+1} - 1$$

Remarque

On démontrera souvent une propriété $\mathcal{P}$, sur les arbres par récurrence (sur la hauteur ou la taille) ou par induction structurelle :

- $\mathcal{P}$ est vraie sur l'arbre vide,

Nombre de sous abres vides

Le nombre de sous arbres vides d'un arbre binaire de taille n est $n+1$.

Relation entre hauteur et taille

En notant n la taille et h la hauteur d'un arbre binaire, on a la relation suivante :

$$h+1 \leqslant n \leqslant 2^{h+1}-1$$

Remarque

On démontrera souvent une propriété $\mathcal{P}$, sur les arbres par récurrence (sur la hauteur ou la taille) ou par induction structurelle :

- $\mathcal{P}$ est vraie sur l'arbre vide,
- Si $\mathcal{P}$ est vraie pour g et d alors $\mathcal{P}$ est vraie pour un arbre dont les sous arbres sont g et d.

Exercices

1. Dessiner un arbres binaires de taille 4 dont l'un des noeuds au moins a un fils gauche vide.

Exercices

1. Dessiner un arbres binaires de taille 4 dont l'un des noeuds au moins a un fils gauche vide.
2. Dessiner un arbre binaire de hauteur taille 4 et de hauteur 3.

Exercices

1. Dessiner un arbres binaires de taille 4 dont l'un des noeuds au moins a un fils gauche vide.
2. Dessiner un arbre binaire de hauteur taille 4 et de hauteur 3.
3. Dessiner un arbre binaire de hauteur 2 et de taille $2^3 - 1$.

Exercices

1. Dessiner un arbres binaires de taille 4 dont l'un des noeuds au moins a un fils gauche vide.
2. Dessiner un arbre binaire de hauteur taille 4 et de hauteur 3.
3. Dessiner un arbre binaire de hauteur 2 et de taille $2^3 - 1$.
4. Montrer que le nombre d'arêtes (trait reliant un noeud à un fils *non vide*) d'un arbre binaire à n noeuds ($n \geqslant 1$) est $n - 1$

Type structuré en C

En C, on représente un arbre binaire par un pointeur vers un type structuré contenant trois champs : l'étiquette (un `int`) de la racine, et les pointeurs vers les deux sous arbres (gauche et droit).

Type structuré en C

En C, on représente un arbre binaire par un pointeur vers un type structuré contenant trois champs : l'étiquette (un `int`) de la racine, et les pointeurs vers les deux sous arbres (gauche et droit).
L'arbre vide est alors représenté par le pointeur NULL.

Type structuré en C

En C, on représente un arbre binaire par un pointeur vers un type structuré contenant trois champs : l'étiquette (un `int`) de la racine, et les pointeurs vers les deux sous arbres (gauche et droit).
L'arbre vide est alors représenté par le pointeur NULL.

```
struct noeud
{
    struct noeud *sag;
    int valeur;
    struct noeud *sad;
};
typedef struct noeud noeud;
typedef noeud *ab;
```

Création d'un arbre

Création d'un arbre

On écrit alors une fonction `ab cree_arbre(ab g, int v, ab d)` qui renvoie un arbre donne on donne l'étiquette de la racine et les deux sous arbres :

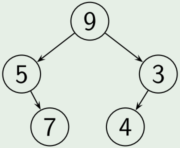
Création d'un arbre

On écrit alors une fonction `ab cree_arbre(ab g, int v, ab d)` qui renvoie un arbre donne on donne l'étiquette de la racine et les deux sous arbres :

```
ab cree_arbre(ab g, int v, ab d)
{
    noeud *n = malloc(sizeof(noeud));
    {
        n->sag = g;
        n->valeur = v;
        n->sad = d;
        return n;
    }
}
```

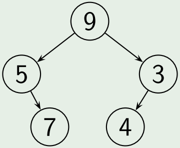
Exemple

1. En utilisant cette représentation, créer l'arbre binaire suivant :



Exemple

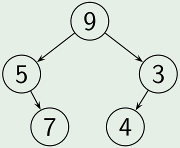
1. En utilisant cette représentation, créer l'arbre binaire suivant :



2. Ecrire la fonction qui calcule le nombre de noeuds d'un arbre binaire.

Exemple

1. En utilisant cette représentation, créer l'arbre binaire suivant :



2. Ecrire la fonction qui calcule le nombre de noeuds d'un arbre binaire.

```c
int taille(ab a)
{
    if (a == NULL)
    {
        return 0;
    }
    return 1 + taille(a->sag) + taille(a->sad);
}
```

Type structuré en OCaml

Un arbre binaire étant soit vide soit constituée d'une étiquette (qu'on suppose de type `int`), on le définit en OCaml en envisageant les 2 cas :

Type structuré en OCaml

Un arbre binaire étant soit vide soit constituée d'une étiquette (qu'on suppose de type `int`), on le définit en OCaml en envisageant les 2 cas :

```
type ab =
    |Vide
    | Noeud of ab * int * ab ;;
```

Type structuré en OCaml

Un arbre binaire étant soit vide soit constituée d'une étiquette (qu'on suppose de type `int`), on le définit en OCaml en envisageant les 2 cas :

```
type ab =
   |Vide
   | Noeud of ab * int * ab ;;
```

Pour un type quelconque d'étiquette, on utilise un type paramétré `'a` :

Type structuré en OCaml

Un arbre binaire étant soit vide soit constituée d'une étiquette (qu'on suppose de type `int`), on le définit en OCaml en envisageant les 2 cas :

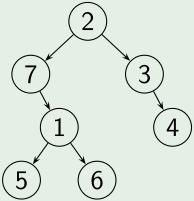
```
type ab =
   |Vide
   | Noeud of ab * int * ab ;;
```

Pour un type quelconque d'étiquette, on utilise un type paramétré `'a` :

```
type 'a ab =
   |Vide
   |Noeud of 'a ab *  'a * 'a ab;;
```

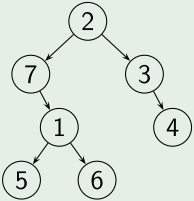
Exemple

1. En utilisant cette représentation, créer l'arbre binaire suivant :



Exemple

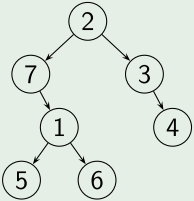
1. En utilisant cette représentation, créer l'arbre binaire suivant :



2. Ecrire la fonction permettant de calculer le nombre de noeuds d'un arbre binaire et donner sa complexité.

Exemple

1. En utilisant cette représentation, créer l'arbre binaire suivant :



2. Ecrire la fonction permettant de calculer le nombre de noeuds d'un arbre binaire et donner sa complexité.

```
let rec taille ab =
  match ab with
   | Vide -> 0
   | Noeud (sag, _, sad) -> 1 + taille sag + taille sad;;
```

Cas particulier d'un arbre binaire complet

Cas particulier d'un arbre binaire complet

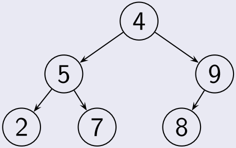
Un arbre binaire complet de taille n peut être représenté de façon compacte à l'aide d'un tableau de taille n.

Cas particulier d'un arbre binaire complet

Un arbre binaire complet de taille n peut être représenté de façon compacte à l'aide d'un tableau de taille n. On numérote les noeuds depuis la racine, de gauche à droite et de haut en bas, le noeud numeroté i est placé dans la case d'indice i.

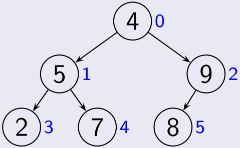
Cas particulier d'un arbre binaire complet

Un arbre binaire complet de taille n peut être représenté de façon compacte à l'aide d'un tableau de taille n. On numérote les noeuds depuis la racine, de gauche à droite et de haut en bas, le noeud numeroté i est placé dans la case d'indice i.
Par exemple :



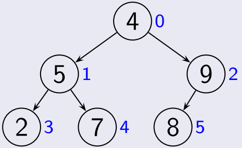
Cas particulier d'un arbre binaire complet

Un arbre binaire complet de taille n peut être représenté de façon compacte à l'aide d'un tableau de taille n. On numérote les noeuds depuis la racine, de gauche à droite et de haut en bas, le noeud numeroté i est placé dans la case d'indice i.
Par exemple :



Cas particulier d'un arbre binaire complet

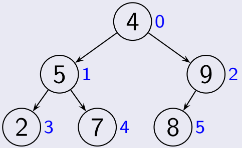
Un arbre binaire complet de taille n peut être représenté de façon compacte à l'aide d'un tableau de taille n. On numérote les noeuds depuis la racine, de gauche à droite et de haut en bas, le noeud numeroté i est placé dans la case d'indice i.
Par exemple :



sera représenté par le tableau :

Cas particulier d'un arbre binaire complet

Un arbre binaire complet de taille n peut être représenté de façon compacte à l'aide d'un tableau de taille n. On numérote les noeuds depuis la racine, de gauche à droite et de haut en bas, le noeud numeroté i est placé dans la case d'indice i.
Par exemple :

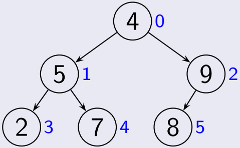


sera représenté par le tableau :

4	5	9	2	7	8
0	1	2	3	4	5

Cas particulier d'un arbre binaire complet

Un arbre binaire complet de taille n peut être représenté de façon compacte à l'aide d'un tableau de taille n. On numérote les noeuds depuis la racine, de gauche à droite et de haut en bas, le noeud numeroté i est placé dans la case d'indice i.
Par exemple :



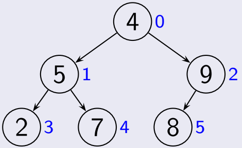
sera représenté par le tableau :

4	5	9	2	7	8
0	1	2	3	4	5

- Le fils gauche (resp. droit) du noeud i se trouve à l'indice $2i+1$ (resp.) $2i+2$.

Cas particulier d'un arbre binaire complet

Un arbre binaire complet de taille n peut être représenté de façon compacte à l'aide d'un tableau de taille n. On numérote les noeuds depuis la racine, de gauche à droite et de haut en bas, le noeud numeroté i est placé dans la case d'indice i.
Par exemple :



sera représenté par le tableau :

4	5	9	2	7	8
0	1	2	3	4	5

- Le fils gauche (resp. droit) du noeud i se trouve à l'indice $2i+1$ (resp.) $2i+2$.
- Le père du noeud d'indice i se trouve à l'indice $\left\lfloor \dfrac{i-1}{2} \right\rfloor$.

Parcours récursifs

On appelle *parcours d'un arbre binaire* un algorithme permettant de visiter chaque noeud de cet arbre une et une seule fois afin d'y effectuer un traitement (tester la présence d'une valeur, chercher la plus petite valeur, ...).

Parcours récursifs

On appelle *parcours d'un arbre binaire* un algorithme permettant de visiter chaque noeud de cet arbre une et une seule fois afin d'y effectuer un traitement (tester la présence d'une valeur, chercher la plus petite valeur, ...). Compte tenu de la structure récursive des arbres binaires, trois parcours récursifs émergent suivant le choix du moment où on traite la racine du noeud (g, r, d) :

Parcours récursifs

On appelle *parcours d'un arbre binaire* un algorithme permettant de visiter chaque noeud de cet arbre une et une seule fois afin d'y effectuer un traitement (tester la présence d'une valeur, chercher la plus petite valeur, ...). Compte tenu de la structure récursive des arbres binaires, trois parcours récursifs émergent suivant le choix du moment où on traite la racine du noeud (g, r, d) :

- Dans le parcours préfixe, la racine est traitée avant de relancer le parcours sur le sous arbre gauche g et le sous arbre droit d.

Parcours récursifs

On appelle *parcours d'un arbre binaire* un algorithme permettant de visiter chaque noeud de cet arbre une et une seule fois afin d'y effectuer un traitement (tester la présence d'une valeur, chercher la plus petite valeur, ...). Compte tenu de la structure récursive des arbres binaires, trois parcours récursifs émergent suivant le choix du moment où on traite la racine du noeud (g, r, d) :

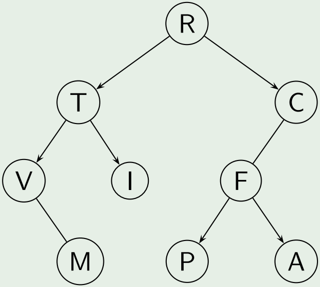
- Dans le parcours préfixe, la racine est traitée avant de relancer le parcours sur le sous arbre gauche g et le sous arbre droit d.
- Dans le parcours infixe, la racine est traitée après le parcours du sous arbre gauche g mais avant celui du sous arbre droit d.

Parcours récursifs

On appelle *parcours d'un arbre binaire* un algorithme permettant de visiter chaque noeud de cet arbre une et une seule fois afin d'y effectuer un traitement (tester la présence d'une valeur, chercher la plus petite valeur, ...). Compte tenu de la structure récursive des arbres binaires, trois parcours récursifs émergent suivant le choix du moment où on traite la racine du noeud (g, r, d) :

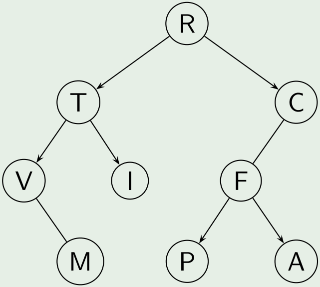
- Dans le parcours préfixe, la racine est traitée avant de relancer le parcours sur le sous arbre gauche g et le sous arbre droit d.
- Dans le parcours infixe, la racine est traitée après le parcours du sous arbre gauche g mais avant celui du sous arbre droit d.
- Dans le parcours suffixe, la racine est traitée après le parcours du sous arbre gauche g et du sous arbre droit d.

Exemple



Donner l'ordre des noeuds lorsqu'on parcourt l'arbre ci-dessus :

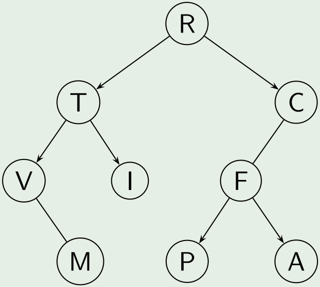
Exemple



Donner l'ordre des noeuds lorsqu'on parcourt l'arbre ci-dessus :

- En profondeur préfixé :

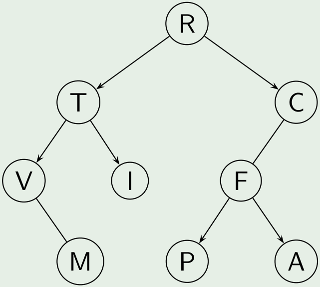
Exemple



Donner l'ordre des noeuds lorsqu'on parcourt l'arbre ci-dessus :

- En profondeur préfixé : R, T, V, M, I, C, F, P, A

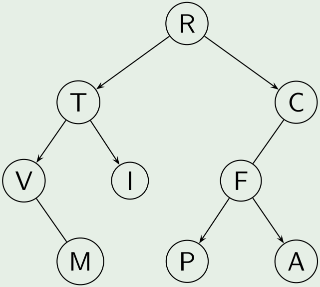
Exemple



Donner l'ordre des noeuds lorsqu'on parcourt l'arbre ci-dessus :

- En profondeur préfixé : R, T, V, M, I, C, F, P, A
- En profondeur infixé :

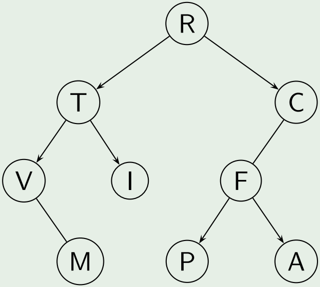
Exemple



Donner l'ordre des noeuds lorsqu'on parcourt l'arbre ci-dessus :

- En profondeur préfixé : R, T, V, M, I, C, F, P, A
- En profondeur infixé : V, M, T, I, R, P, F, A, C

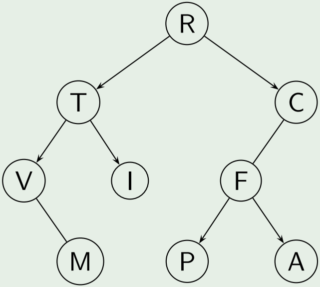
Exemple



Donner l'ordre des noeuds lorsqu'on parcourt l'arbre ci-dessus :

- En profondeur préfixé : R, T, V, M, I, C, F, P, A
- En profondeur infixé : V, M, T, I, R, P, F, A, C
- En profondeur suffixé :

Exemple



Donner l'ordre des noeuds lorsqu'on parcourt l'arbre ci-dessus :

- En profondeur préfixé : R, T, V, M, I, C, F, P, A
- En profondeur infixé : V, M, T, I, R, P, F, A, C
- En profondeur suffixé : M, V, I, T, P, A, F, C, R

Parcours en largeur

La parcours en largeur revient à lister les noeuds par ordre croissant de profondeur et de gauche à droite

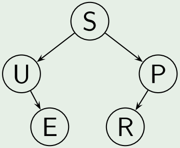
Parcours en largeur

La parcours en largeur revient à lister les noeuds par ordre croissant de profondeur et de gauche à droite
L'implémentation de ce parcours peut se faire à l'aide d'une file dans laquelle on stocke les noeuds restants à parcourir. A chaque fois qu'on traite un noeud, on le defile et on enfile ses fils.

Parcours en largeur

La parcours en largeur revient à lister les noeuds par ordre croissant de profondeur et de gauche à droite
L'implémentation de ce parcours peut se faire à l'aide d'une file dans laquelle on stocke les noeuds restants à parcourir. A chaque fois qu'on traite un noeud, on le defile et on enfile ses fils.

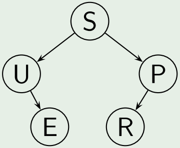
Exemple



Parcours en largeur

La parcours en largeur revient à lister les noeuds par ordre croissant de profondeur et de gauche à droite
L'implémentation de ce parcours peut se faire à l'aide d'une file dans laquelle on stocke les noeuds restants à parcourir. A chaque fois qu'on traite un noeud, on le defile et on enfile ses fils.

Exemple



Le parcours en largeur donne : S, U, P, E, R.

Exemples d'implémentation

- Parcours préfixe en C (affichage des étiquettes)

Exemples d'implémentation

- Parcours préfixe en C (affichage des étiquettes)

```
void prefixe(ab a)
{
    if (a != NULL)
    {
        printf("%d ", a->valeur);
        prefixe(a->sag);
        prefixe(a->sad);
    }
}
```

Exemples d'implémentation

- Parcours préfixe en C (affichage des étiquettes)

```c
void prefixe(ab a)
{
    if (a != NULL)
    {
        printf("%d ", a->valeur);
        prefixe(a->sag);
        prefixe(a->sad);
    }
}
```

- Parcours infixe en OCaml (affichage des étiquettes)

Exemples d'implémentation

- Parcours préfixe en C (affichage des étiquettes)

```c
void prefixe(ab a)
{
    if (a != NULL)
    {
        printf("%d ", a->valeur);
        prefixe(a->sag);
        prefixe(a->sad);
    }
}
```

- Parcours infixe en OCaml (affichage des étiquettes)

```ocaml
let rec infixe a =
  match a with
  | Vide -> ()
  | Noeud (g,e,d) ->  infixe g; Printf.printf "%c " e; infixe d;;
```

Arbre binaire de recherche

Un arbre binaire de recherche (noté ABR), est un arbre binaire tel que :

Arbre binaire de recherche

Un arbre binaire de recherche (noté ABR), est un arbre binaire tel que :

- Les étiquettes des noeuds, appelées clés sont toutes comparables entre elles.

Arbre binaire de recherche

Un arbre binaire de recherche (noté ABR), est un arbre binaire tel que :

- Les étiquettes des noeuds, appelées clés sont toutes comparables entre elles.

Arbre binaire de recherche

Un arbre binaire de recherche (noté ABR), est un arbre binaire tel que :

- Les étiquettes des noeuds, appelées clés sont toutes comparables entre elles.
  Par exemple, les étiquettes sont toutes des nombres ou encore des chaines de caractères (comparées par ordre alphabétique).

Arbre binaire de recherche

Un arbre binaire de recherche (noté ABR), est un arbre binaire tel que :

- Les étiquettes des noeuds, appelées clés sont toutes comparables entre elles. Par exemple, les étiquettes sont toutes des nombres ou encore des chaines de caractères (comparées par ordre alphabétique).
- Pour tous les noeuds $N(g, v, d)$ l'ensemble des clés présentes dans le sous arbre gauche g (resp. droit d) sont strictement inférieures (resp. supérieures) à v.

Arbre binaire de recherche

Un arbre binaire de recherche (noté ABR), est un arbre binaire tel que :

- Les étiquettes des noeuds, appelées clés sont toutes comparables entre elles. Par exemple, les étiquettes sont toutes des nombres ou encore des chaines de caractères (comparées par ordre alphabétique).
- Pour tous les noeuds $N(g, v, d)$ l'ensemble des clés présentes dans le sous arbre gauche g (resp. droit d) sont strictement inférieures (resp. supérieures) à v.
- Les clés sont uniques.

Arbre binaire de recherche

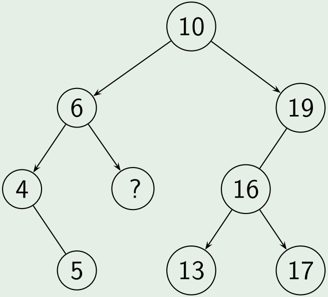
Un arbre binaire de recherche (noté ABR), est un arbre binaire tel que :

- Les étiquettes des noeuds, appelées clés sont toutes comparables entre elles. Par exemple, les étiquettes sont toutes des nombres ou encore des chaines de caractères (comparées par ordre alphabétique).
- Pour tous les noeuds $N(g, v, d)$ l'ensemble des clés présentes dans le sous arbre gauche g (resp. droit d) sont strictement inférieures (resp. supérieures) à v.
- Les clés sont uniques.

Caractérisation par le parcours infixe

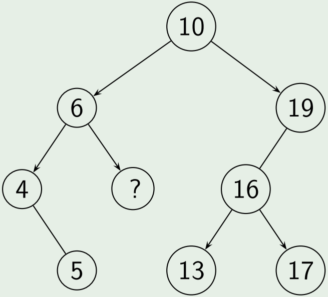
Un arbre binaire est un ABR si et seulement si le parcours infixe fournit les clés dans l'ordre croissant.

Exemple



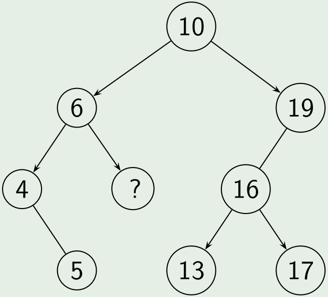
- Cet arbre est-il un ABR si la clé manquante est 2 ? 9 ? 12 ?

Exemple



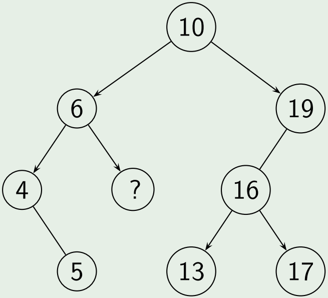
- Cet arbre est-il un ABR si la clé manquante est 2 ? 9 ? 12 ?
- On suppose que la clé manquante est 9. Proposer une nouvelle valeur pour le noeud de clé 16 de façon à ce que cet arbre reste un ABR.

Exemple



- Cet arbre est-il un ABR si la clé manquante est 2 ? 9 ? 12 ?
- On suppose que la clé manquante est 9. Proposer une nouvelle valeur pour le noeud de clé 16 de façon à ce que cet arbre reste un ABR.
- Proposer une valeur pour le noeud de clé 16 de façon à ce que cet arbre ne soit pas un ABR.

Exemple



- Cet arbre est-il un ABR si la clé manquante est 2 ? 9 ? 12 ?
- On suppose que la clé manquante est 9. Proposer une nouvelle valeur pour le noeud de clé 16 de façon à ce que cet arbre reste un ABR.
- Proposer une valeur pour le noeud de clé 16 de façon à ce que cet arbre ne soit pas un ABR.
- Donner l'ordre des clés lors d'un parcours infixe.

Insertion dans un ABR

Pour insérer une nouvelle valeur u dans un ABR A :

Insertion dans un ABR

Pour insérer une nouvelle valeur u dans un ABR A :

- Si A est vide on renvoie $(\varnothing, u, \varnothing)$.

Insertion dans un ABR

Pour insérer une nouvelle valeur u dans un ABR A :

- Si A est vide on renvoie $(\varnothing, u, \varnothing)$.
- Sinon $A = (g, v, d)$ et on insère dans g si $u < v$ et dans d sinon.

Insertion dans un ABR

Pour insérer une nouvelle valeur u dans un ABR A :

- Si A est vide on renvoie $(\varnothing, u, \varnothing)$.
- Sinon $A = (g, v, d)$ et on insère dans g si $u < v$ et dans d sinon.

Implémentation en OCaml

```ocaml
let rec insere abr nv =
  match abr with
  | Vide -> Noeud(Vide,nv,Vide)
  | Noeud(g,v,d) -> if nv<v then Noeud(insere g nv, v, d)
   ↪  else Noeud(g, v, insere d nv);;
```

Exercice

- Dessiner l'arbre obtenu en partant de l'arbre vide puis en insérant dans cet ordre les valeurs :

Exercice

- Dessiner l'arbre obtenu en partant de l'arbre vide puis en insérant dans cet ordre les valeurs :
  - 2, 5, 7, 9 et 11

Exercice

- Dessiner l'arbre obtenu en partant de l'arbre vide puis en insérant dans cet ordre les valeurs :
  - 2, 5, 7, 9 et 11
  - 9, 5, 11, 2, 7

Exercice

- Dessiner l'arbre obtenu en partant de l'arbre vide puis en insérant dans cet ordre les valeurs :
  - 2, 5, 7, 9 et 11
  - 9, 5, 11, 2, 7
  - 7, 5, 9, 2, 11

Exercice

- Dessiner l'arbre obtenu en partant de l'arbre vide puis en insérant dans cet ordre les valeurs :
  - 2, 5, 7, 9 et 11
  - 9, 5, 11, 2, 7
  - 7, 5, 9, 2, 11
- En vous inspirant de la fonction d'insertion, écrire une fonction `in_abr` (`abr -> int -> bool`) qui renvoie un booléen indiquant si la valeur passé en argument se trouve ou non dans l'ABR.

Exercice

- Dessiner l'arbre obtenu en partant de l'arbre vide puis en insérant dans cet ordre les valeurs :
    - 2, 5, 7, 9 et 11
    - 9, 5, 11, 2, 7
    - 7, 5, 9, 2, 11
- En vous inspirant de la fonction d'insertion, écrire une fonction `in_abr` (`abr -> int -> bool`) qui renvoie un booléen indiquant si la valeur passé en argument se trouve ou non dans l'ABR.

```
let rec in_abr abr u =
  match abr with
   | Vide -> false
   | Noeud(g,v,d) -> if u=v then true else
     if u<v then in_abr g u else in_abr d u;;
```

Complexité

La complexité des opérations d'insertion et de recherche dans un ABR est majorée par la hauteur h de l'arbre.

Complexité

La complexité des opérations d'insertion et de recherche dans un ABR est majorée par la hauteur h de l'arbre.

En effet, on descend d'un niveau dans l'arbre à chaque appel récursif et la profondeur d'un noeud est inférieure à h.

Complexité

La complexité des opérations d'insertion et de recherche dans un ABR est majorée par la hauteur h de l'arbre.

En effet, on descend d'un niveau dans l'arbre à chaque appel récursif et la profondeur d'un noeud est inférieure à h.

Cas des arbres binaires équilibrés

Soit S, un ensemble d'arbres binaires. On dit que les arbres de S sont équilibrés s'il existe une constante C telle que, pour tout arbre $s \in S$:

$$h(s) \leqslant C \log(n(s))$$

Conséquence : pour tout $s \in S$, la complexité des opérations est logarithmique en fonction de la taille n de s.

Implémentation des tableaux associatifs

- Si l'ensemble des clés est ordonné, on peut implémenter les tableaux associatifs en utilisant une famille d'arbres binaires équilibrés. Chaque noeud de l'arbre contient alors une information supplémentaire (sa valeur) en plus de la clé associée.

Implémentation des tableaux associatifs

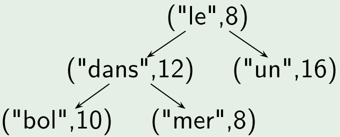
- Si l'ensemble des clés est ordonné, on peut implémenter les tableaux associatifs en utilisant une famille d'arbres binaires équilibrés. Chaque noeud de l'arbre contient alors une information supplémentaire (sa valeur) en plus de la clé associée.
- Les opérations usuelles (insertion, test d'appartenance, ...) ont alors une complexité logarithmique.

Implémentation des tableaux associatifs

- Si l'ensemble des clés est ordonné, on peut implémenter les tableaux associatifs en utilisant une famille d'arbres binaires équilibrés. Chaque noeud de l'arbre contient alors une information supplémentaire (sa valeur) en plus de la clé associée.
- Les opérations usuelles (insertion, test d'appartenance, ...) ont alors une complexité logarithmique.

Exemple

`{ "dans":12 , "le" : 8, "un" : 16, "bol" : 10, "mer" : 8}` est représenté par :



Definition

Une file de priorité est une structure de données, dans laquelle :

Definition

Une file de priorité est une structure de données, dans laquelle :

- chaque élément enfilé possède une priorité.

Definition

Une file de priorité est une structure de données, dans laquelle :

- chaque élément enfilé possède une priorité.
- lorsqu'on défile un élément, on enlève l'élément le plus prioritaire (et donc pas forcément le premier enfilé comme dans une file classique).

Definition

Une file de priorité est une structure de données, dans laquelle :

- chaque élément enfilé possède une priorité.
- lorsqu'on défile un élément, on enlève l'élément le plus prioritaire (et donc pas forcément le premier enfilé comme dans une file classique).

Les applications de cette structure de données sont nombreuses en informatique, par exemple l'ordonnanceur (*scheduler*) d'un système d'exploitation ou encore comme ingrédient d'autres algorithmes.

Definition

Une file de priorité est une structure de données, dans laquelle :

- chaque élément enfilé possède une priorité.
- lorsqu'on défile un élément, on enlève l'élément le plus prioritaire (et donc pas forcément le premier enfilé comme dans une file classique).

Les applications de cette structure de données sont nombreuses en informatique, par exemple l'ordonnanceur (*scheduler*) d'un système d'exploitation ou encore comme ingrédient d'autres algorithmes.

Exercice

Déterminer les complexités des opérations enfiler et défiler si on implémente une file de priorité à l'aide d'une liste chainée.

Structure de tas

Soit t un arbre binaire dont les noeuds sont étiquetés par des éléments d'un ensemble ordonné.

Structure de tas

Soit t un arbre binaire dont les noeuds sont étiquetés par des éléments d'un ensemble ordonné.

- si l'étiquette d'un noeud est toujours inférieure à celle de ses enfants (lorsqu'ils existent), on dit que t possède la propriété de tas-min.

Structure de tas

Soit t un arbre binaire dont les noeuds sont étiquetés par des éléments d'un ensemble ordonné.

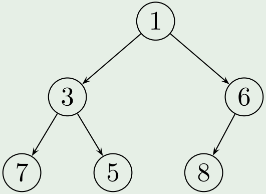
- si l'étiquette d'un noeud est toujours inférieure à celle de ses enfants (lorsqu'ils existent), on dit que t possède la propriété de tas-min.
- si de plus t est complet, on dit que t est un tas-min binaire.

Structure de tas

Soit t un arbre binaire dont les noeuds sont étiquetés par des éléments d'un ensemble ordonné.

- si l'étiquette d'un noeud est toujours inférieure à celle de ses enfants (lorsqu'ils existent), on dit que t possède la propriété de tas-min.
- si de plus t est complet, on dit que t est un tas-min binaire.

Exemple



Remarques

- On parle de tas-max lorsque l'étiquette du noeud est toujours supérieure ou égale à celle de ses enfants.

Remarques

- On parle de tas-max lorsque l'étiquette du noeud est toujours supérieure ou égale à celle de ses enfants.
- Le terme anglais est *(binary) heap* (mais n'a rien à voir avec la zone mémoire dans laquelle on alloue à l'aide de `malloc`)

Remarques

- On parle de tas-max lorsque l'étiquette du noeud est toujours supérieure ou égale à celle de ses enfants.
- Le terme anglais est *(binary) heap* (mais n'a rien à voir avec la zone mémoire dans laquelle on alloue à l'aide de `malloc`)

Propriété

- La racine contient un élément de clé minimale

Remarques

- On parle de tas-max lorsque l'étiquette du noeud est toujours supérieure ou égale à celle de ses enfants.
- Le terme anglais est *(binary) heap* (mais n'a rien à voir avec la zone mémoire dans laquelle on alloue à l'aide de `malloc`)

Propriété

- La racine contient un élément de clé minimale
- Si a est un ancêtre de b alors l'étiquette de a est inférieure à celle de b.

Implémentation

Un tas binaire étant un arbre binaire complet, on peut utiliser la représentation des arbres binaires sour forme de tableau. On rappelle qu'alors :

Implémentation

Un tas binaire étant un arbre binaire complet, on peut utiliser la représentation des arbres binaires sour forme de tableau. On rappelle qu'alors :

- Le fils gauche (resp. droit) du noeud i se trouve à l'indice $2i+1$ (resp. $2i+2$)

Implémentation

Un tas binaire étant un arbre binaire complet, on peut utiliser la représentation des arbres binaires sour forme de tableau. On rappelle qu'alors :

- Le fils gauche (resp. droit) du noeud i se trouve à l'indice $2i+1$ (resp. $2i+2$)
- Le père du noeud d'indice i se trouve à l'indice $\left\lfloor \dfrac{i-1}{2} \right\rfloor$.

Implémentation

Un tas binaire étant un arbre binaire complet, on peut utiliser la représentation des arbres binaires sour forme de tableau. On rappelle qu'alors :

- Le fils gauche (resp. droit) du noeud i se trouve à l'indice $2i+1$ (resp. $2i+2$)
- Le père du noeud d'indice i se trouve à l'indice $\left\lfloor \dfrac{i-1}{2} \right\rfloor$.

Exemple

Donner la représentation sous forme de tableau du tas binaire donné en exemple

Implémentation en C

Les opérations intéressantes faisant varier la taille du tas (insertion, extraction du minimum) on utilise un tableau de capacité fixée mais de taille variable :

```c
struct heap_s
{
    int size;
    int capacity;
    int *tab;
};
typedef struct heap_s heap;
```

Implémentation en OCaml

En OCaml, la capacité du tableau étant accessible (avec `Array.length`) on se contente de garder la taille effective utilisée dans un champ mutable

Implémentation en OCaml

En OCaml, la capacité du tableau étant accessible (avec `Array.length`) on se contente de garder la taille effective utilisée dans un champ mutable

```ocaml
type 'a heap = {mutable size : int; data : 'a array};;
```

Implémentation en OCaml

En OCaml, la capacité du tableau étant accessible (avec `Array.length`) on se contente de garder la taille effective utilisée dans un champ mutable

```
type 'a heap = {mutable size : int; data : 'a array};;
```

On remarque l'utilisation d'un type paramétré `'a` pour le type des éléments du tas. Ce type sera instancié dès la création du tas, par exemple :

Implémentation en OCaml

En OCaml, la capacité du tableau étant accessible (avec `Array.length`) on se contente de garder la taille effective utilisée dans un champ mutable

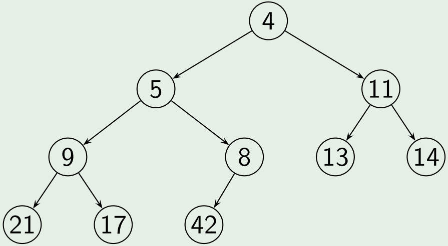
```
type 'a heap = {mutable size : int; data : 'a array};;
```

On remarque l'utilisation d'un type paramétré `'a` pour le type des éléments du tas. Ce type sera instancié dès la création du tas, par exemple :

```
let heap_of_int = {size = 0; data = Array.make 1000 0}
let heap_of_char = {size = 0; data = Array.make 1000 ' '}
```

Exemple

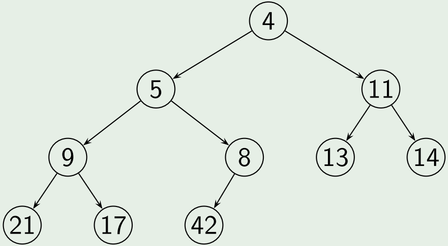
On considère l'arbre binaire suivant :



1. Vérifier qu'il s'agit bien d'un tas binaire (min)

Exemple

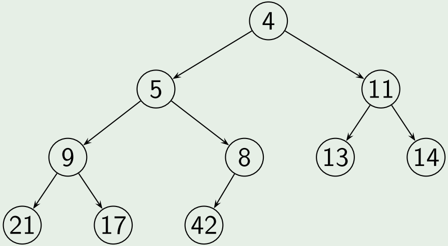
On considère l'arbre binaire suivant :



1. Vérifier qu'il s'agit bien d'un tas binaire (min)
2. Donner par deux méthodes différentes (un parcours en largeur et les formules donnant les indices des fils), le tableau représentant ce tas binaire,

Exemple

On considère l'arbre binaire suivant :



1. Vérifier qu'il s'agit bien d'un tas binaire (min)
2. Donner par deux méthodes différentes (un parcours en largeur et les formules donnant les indices des fils), le tableau représentant ce tas binaire,
3. Dessiner l'arbre correspondant au tableau `[5; 7; 12; 9; 11: 10; 22; 31]`, est-ce un tas binaire ?

Insertion dans un tas binaire

Pour insérer une nouvelle valeur x dans un tas binaire, on écrit cette valeur dans la première case libre du tableau et on incrémente la taille du tas. Puis, on effectue une percolation vers le haut (*sift-up* en anglais) pour rétablir la propriété de tas si elle a été perdue.

Insertion dans un tas binaire

Pour insérer une nouvelle valeur x dans un tas binaire, on écrit cette valeur dans la première case libre du tableau et on incrémente la taille du tas. Puis, on effectue une percolation vers le haut (*sift-up* en anglais) pour rétablir la propriété de tas si elle a été perdue.
La percolation vers le haut, consiste à échanger l'élément inséré avec son père tant que ce dernier lui est supérieur (et qu'il existe).

Insertion dans un tas binaire

Pour insérer une nouvelle valeur x dans un tas binaire, on écrit cette valeur dans la première case libre du tableau et on incrémente la taille du tas. Puis, on effectue une percolation vers le haut (*sift-up* en anglais) pour rétablir la propriété de tas si elle a été perdue.
La percolation vers le haut, consiste à échanger l'élément inséré avec son père tant que ce dernier lui est supérieur (et qu'il existe).
Cet algorithme :

Insertion dans un tas binaire

Pour insérer une nouvelle valeur x dans un tas binaire, on écrit cette valeur dans la première case libre du tableau et on incrémente la taille du tas. Puis, on effectue une percolation vers le haut (*sift-up* en anglais) pour rétablir la propriété de tas si elle a été perdue.
La percolation vers le haut, consiste à échanger l'élément inséré avec son père tant que ce dernier lui est supérieur (et qu'il existe).
Cet algorithme :

- termine (x remonte d'un niveau à chaque étape).

Insertion dans un tas binaire

Pour insérer une nouvelle valeur x dans un tas binaire, on écrit cette valeur dans la première case libre du tableau et on incrémente la taille du tas. Puis, on effectue une percolation vers le haut (*sift-up* en anglais) pour rétablir la propriété de tas si elle a été perdue.
La percolation vers le haut, consiste à échanger l'élément inséré avec son père tant que ce dernier lui est supérieur (et qu'il existe).
Cet algorithme :

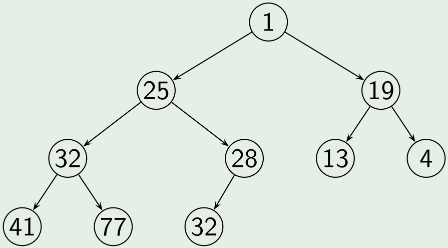
- termine (x remonte d'un niveau à chaque étape).
- est correct car l'invariant suivant est préservé : *"à tout moment, les deux seuls éléments pouvant entrer en conflit avec la propriété de tas sont x et son père"*.

Insertion dans un tas binaire

Pour insérer une nouvelle valeur x dans un tas binaire, on écrit cette valeur dans la première case libre du tableau et on incrémente la taille du tas. Puis, on effectue une percolation vers le haut (*sift-up* en anglais) pour rétablir la propriété de tas si elle a été perdue.
La percolation vers le haut, consiste à échanger l'élément inséré avec son père tant que ce dernier lui est supérieur (et qu'il existe).
Cet algorithme :

- termine (x remonte d'un niveau à chaque étape).
- est correct car l'invariant suivant est préservé : *"à tout moment, les deux seuls éléments pouvant entrer en conflit avec la propriété de tas sont x et son père"*.
- est de complexité $O(\log n)$.

Exemple



Donner les étapes successives de l'insertion de 11 dans le tas ci-dessus

Extraction du minimum

Pour extraire le minimum d'un tas binaire :

Extraction du minimum

Pour extraire le minimum d'un tas binaire :

- On extrait l'élément d'indice 0 du tableau (donc la racine du tas)

Extraction du minimum

Pour extraire le minimum d'un tas binaire :

- On extrait l'élément d'indice 0 du tableau (donc la racine du tas)
- On le remplace par le dernier élément du tableau et on diminue la taille du tas

Extraction du minimum

Pour extraire le minimum d'un tas binaire :

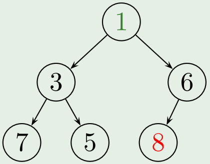
- On extrait l'élément d'indice 0 du tableau (donc la racine du tas)
- On le remplace par le dernier élément du tableau et on diminue la taille du tas
- On effectue une percolation vers le bas de cet élément pour cela, on l'échange avec le plus petit de ses deux fils tant qu'il est supérieur à l'un des deux (et que les fils existent).

Extraction du minimum

Pour extraire le minimum d'un tas binaire :

- On extrait l'élément d'indice 0 du tableau (donc la racine du tas)
- On le remplace par le dernier élément du tableau et on diminue la taille du tas
- On effectue une percolation vers le bas de cet élément pour cela, on l'échange avec le plus petit de ses deux fils tant qu'il est supérieur à l'un des deux (et que les fils existent).

Exemple

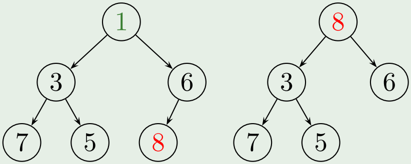


Extraction du minimum

Pour extraire le minimum d'un tas binaire :

- On extrait l'élément d'indice 0 du tableau (donc la racine du tas)
- On le remplace par le dernier élément du tableau et on diminue la taille du tas
- On effectue une percolation vers le bas de cet élément pour cela, on l'échange avec le plus petit de ses deux fils tant qu'il est supérieur à l'un des deux (et que les fils existent).

Exemple

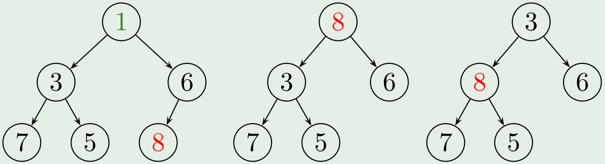


Extraction du minimum

Pour extraire le minimum d'un tas binaire :

- On extrait l'élément d'indice 0 du tableau (donc la racine du tas)
- On le remplace par le dernier élément du tableau et on diminue la taille du tas
- On effectue une percolation vers le bas de cet élément pour cela, on l'échange avec le plus petit de ses deux fils tant qu'il est supérieur à l'un des deux (et que les fils existent).

Exemple

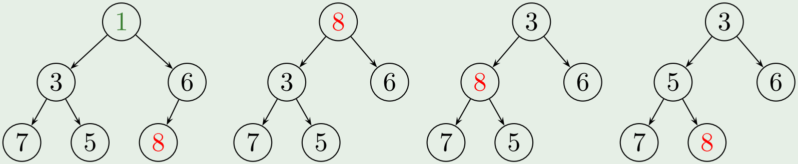


Extraction du minimum

Pour extraire le minimum d'un tas binaire :

- On extrait l'élément d'indice 0 du tableau (donc la racine du tas)
- On le remplace par le dernier élément du tableau et on diminue la taille du tas
- On effectue une percolation vers le bas de cet élément pour cela, on l'échange avec le plus petit de ses deux fils tant qu'il est supérieur à l'un des deux (et que les fils existent).

Exemple



Implémentation : percolation vers le haut en C

```c
bool insert_heap(int nv, heap *mh)
{
    int cp = mh->size;
    if (mh->size == mh->capacity)
    {
        return false;
    }
    else
    {
        mh->tab[cp] = nv;
        while (father(cp) != -1 && mh->tab[cp] < mh->tab[father(cp)])
        {
            swap(mh->tab, cp, father(cp));
            cp = father(cp);
        }
        mh->size = mh->size + 1;
    }
}
```

Implémentation : percolation vers le bas en OCaml

```
let extract_min theap =
  let minv = theap.data.(0) in
  theap.data.(0) <- theap.data.(theap.size-1);
  let cp = ref 0 in
  let lc = ref (leftchild !cp theap.size) in
  while ( !lc <> (-1) && (theap.data.(!lc) < theap.data.(!cp) || (!lc+1 <
  ↪  theap.size && theap.data.(!lc+1) < theap.data.(!cp)))) do
    if (!lc+1 >= theap.size || theap.data.(!lc)<theap.data.(!lc+1)) then
     (swap theap.data !cp !lc;
      cp := !lc)
    else
      (swap theap.data !cp (!lc+1);
      cp := !lc+1);
    lc := leftchild !cp theap.size;
  done;
  theap.size <- theap.size -1;
  minv;;
```

Tri par tas

A partir de la structure de tas, on obtient un algorithme de tri :

Tri par tas

A partir de la structure de tas, on obtient un algorithme de tri :

- on insère les n éléments à trier dans un tas initialement vide,

Tri par tas

A partir de la structure de tas, on obtient un algorithme de tri :

- on insère les n éléments à trier dans un tas initialement vide,
- on réalise ensuite n extractions du minimum.

Tri par tas

A partir de la structure de tas, on obtient un algorithme de tri :

- on insère les n éléments à trier dans un tas initialement vide,
- on réalise ensuite n extractions du minimum.

Exercice

1. Déterminer la complexité de cet algorithme de tri

Tri par tas

A partir de la structure de tas, on obtient un algorithme de tri :

- on insère les n éléments à trier dans un tas initialement vide,
- on réalise ensuite n extractions du minimum.

Exercice

1. Déterminer la complexité de cet algorithme de tri
2. En proposer une implémentation en C, en supposant déjà écrite les fonctions :

Tri par tas

A partir de la structure de tas, on obtient un algorithme de tri :

- on insère les n éléments à trier dans un tas initialement vide,
- on réalise ensuite n extractions du minimum.

Exercice

1. Déterminer la complexité de cet algorithme de tri
2. En proposer une implémentation en C, en supposant déjà écrite les fonctions :
   - `heap make_heap(int c)` qui renvoie un tas vide de capacité c

Tri par tas

A partir de la structure de tas, on obtient un algorithme de tri :

- on insère les n éléments à trier dans un tas initialement vide,
- on réalise ensuite n extractions du minimum.

Exercice

1. Déterminer la complexité de cet algorithme de tri
2. En proposer une implémentation en C, en supposant déjà écrite les fonctions :
   - `heap make_heap(int c)` qui renvoie un tas vide de capacité `c`
   - `bool insert_heap(int nv, heap* mh)` qui insère `nv` dans le tas `mh`

Tri par tas

A partir de la structure de tas, on obtient un algorithme de tri :

- on insère les n éléments à trier dans un tas initialement vide,
- on réalise ensuite n extractions du minimum.

Exercice

1. Déterminer la complexité de cet algorithme de tri
2. En proposer une implémentation en C, en supposant déjà écrite les fonctions :
    - `heap make_heap(int c)` qui renvoie un tas vide de capacité `c`
    - `bool insert_heap(int nv, heap* mh)` qui insère `nv` dans le tas `mh`
    - `int extract_min(heap * mh)` qui renvoie le minimum du tas `*mh` en le supprimant de `*mh`