Arbres binaires de recherche

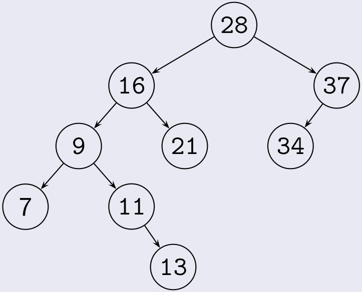
1. Rappeler les relations entre la hauteur h et la taille n d'un arbre binaire.

Arbres binaires de recherche

1. Rappeler les relations entre la hauteur h et la taille n d'un arbre binaire.
2. Rappeler la définition d'un arbre binaire de recherche (ABR).

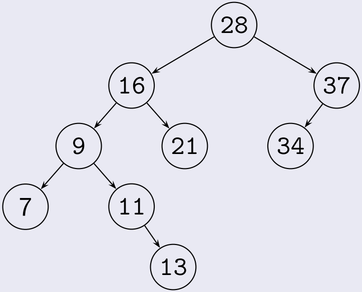
Arbres binaires de recherche

1. Rappeler les relations entre la hauteur h et la taille n d'un arbre binaire.
2. Rappeler la définition d'un arbre binaire de recherche (ABR).
3. L'arbre ci-dessous est-il un ABR ?



Arbres binaires de recherche

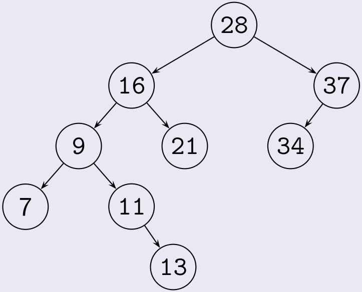
1. Rappeler les relations entre la hauteur h et la taille n d'un arbre binaire.
2. Rappeler la définition d'un arbre binaire de recherche (ABR).
3. L'arbre ci-dessous est-il un ABR ?



4. Quelle est le nombre maximal de comparaison lors de la recherche d'un élément dans cet arbre ?

Arbres binaires de recherche

1. Rappeler les relations entre la hauteur h et la taille n d'un arbre binaire.
2. Rappeler la définition d'un arbre binaire de recherche (ABR).
3. L'arbre ci-dessous est-il un ABR ?



4. Quelle est le nombre maximal de comparaison lors de la recherche d'un élément dans cet arbre ?
5. Construire un ABR contenant les valeurs $2, 9, 10, 17$ et 21 et de hauteur minimale. Même question avec la hauteur maximale.

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. On descend d'un niveau dans l'arbre à chaque comparaison 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. On descend d'un niveau dans l'arbre à chaque comparaison et la profondeur d'un noeud est inférieure à h.
Or on sait que $h + 1 \leqslant n \leqslant 2^{h+1} - 1$, et les deux bornes sont atteintes

Complexité

La complexité des opérations d'insertion et de recherche dans un ABR est majorée par la hauteur h de l'arbre. On descend d'un niveau dans l'arbre à chaque comparaison et la profondeur d'un noeud est inférieure à h.

Or on sait que $h+1 \leqslant n \leqslant 2^{h+1}-1$, et les deux bornes sont atteintes

- Dans le cas d'un peigne $(n = h+1)$ les opérations seront en $\mathcal{O}(n)$.

Complexité

La complexité des opérations d'insertion et de recherche dans un ABR est majorée par la hauteur h de l'arbre. On descend d'un niveau dans l'arbre à chaque comparaison et la profondeur d'un noeud est inférieure à h.

Or on sait que $h+1 \leqslant n \leqslant 2^{h+1}-1$, et les deux bornes sont atteintes

- Dans le cas d'un peigne ($n = h+1$) les opérations seront en $\mathcal{O}(n)$.
- Dans le cas d'un arbre complet ($n = 2^{h+1}-1$), les opérations seront en $\mathcal{O}(\log(n))$.

Complexité

La complexité des opérations d'insertion et de recherche dans un ABR est majorée par la hauteur h de l'arbre. On descend d'un niveau dans l'arbre à chaque comparaison et la profondeur d'un noeud est inférieure à h.
Or on sait que $h+1 \leqslant n \leqslant 2^{h+1}-1$, et les deux bornes sont atteintes

- Dans le cas d'un peigne ($n = h+1$) les opérations seront en $\mathcal{O}(n)$.
- Dans le cas d'un arbre complet ($n = 2^{h+1}-1$), les opérations seront en $\mathcal{O}(\log(n))$.

Définition

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

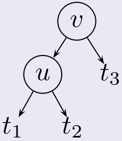
$$h(s) \leqslant C\log(n(s))$$

Rotation d'un ABR

On considère l'ABR suivant où u et v sont les étiquettes des noeuds représentés et t_1, t_2, t_3 des arbres binaires :

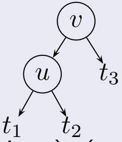
Rotation d'un ABR

On considère l'ABR suivant où u et v sont les étiquettes des noeuds représentés et t_1, t_2, t_3 des arbres binaires :



Rotation d'un ABR

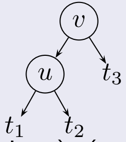
On considère l'ABR suivant où u et v sont les étiquettes des noeuds représentés et t_1, t_2, t_3 des arbres binaires :



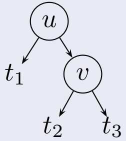
La rotation droite de cet arbre, consiste à réorganiser les noeuds *en conservant la propriété d'*ABR de la façon suivante :

Rotation d'un ABR

On considère l'ABR suivant où u et v sont les étiquettes des noeuds représentés et t_1, t_2, t_3 des arbres binaires :

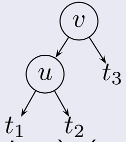


La rotation droite de cet arbre, consiste à réorganiser les noeuds *en conservant la propriété d'*ABR de la façon suivante :

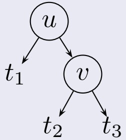


Rotation d'un ABR

On considère l'ABR suivant où u et v sont les étiquettes des noeuds représentés et t_1, t_2, t_3 des arbres binaires :



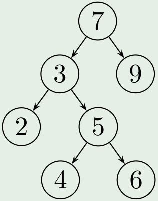
La rotation droite de cet arbre, consiste à réorganiser les noeuds *en conservant la propriété d'*ABR de la façon suivante :



De façon symétrique, la rotation gauche consiste en partant de cet arbre à revenir à l'arbre initial.

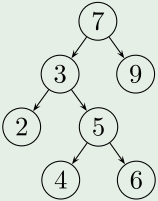
Exemple

On considère l'arbre binaire suivant :



Exemple

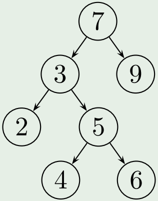
On considère l'arbre binaire suivant :



1. Vérifier qu'il s'agit d'un ABR

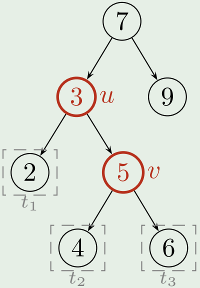
Exemple

On considère l'arbre binaire suivant :

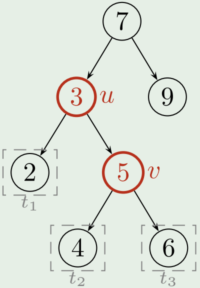


1. Vérifier qu'il s'agit d'un ABR
2. Montrer qu'un utilisant des rotations, on peut transformer cet arbre en un arbre binaire parfait.

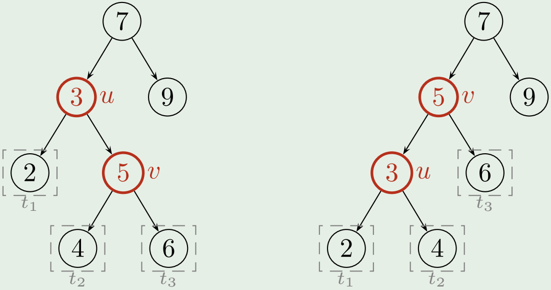
Correction



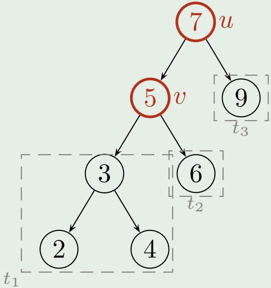
Correction



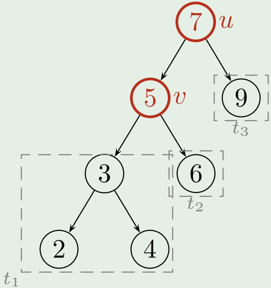
Correction

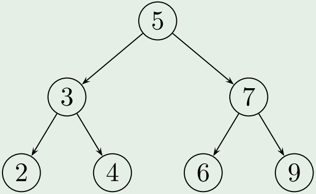


Correction

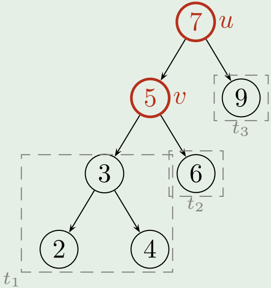


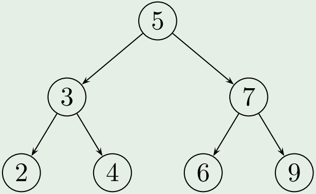
Correction





Correction





Equilibrage d'un abre binaire

Les rotations droite et gauche sont les opérations permettant de maintenir un certain équilibre dans un ABR. Et donc de garantir une complexité logarithmique des opérations usuelles. Parmi les nombreuses possibilités d'ABR équilibrés, nous allons détailler les arbres rouge-noir.

Définition des arbres rouge-noir

Un arbre rouge-noir t est un ABR (❶), dans lequel chaque noeud porte une information de couleur (rouge ou noir), et ayant les deux propriétés suivantes :

- le père d'un noeud rouge est noir (❷),

Définition des arbres rouge-noir

Un arbre rouge-noir t est un ABR (❶), dans lequel chaque noeud porte une information de couleur (rouge ou noir), et ayant les deux propriétés suivantes :

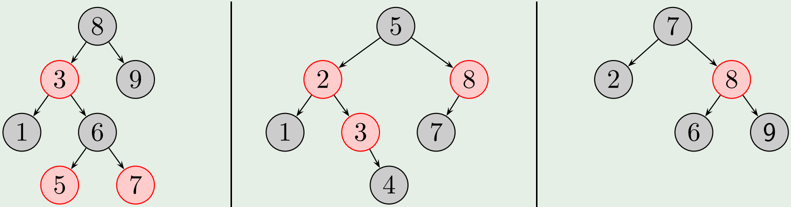
- le père d'un noeud rouge est noir (❷),
- le nombre de noeuds noirs le long d'un chemin de la racine à un sous arbre vide est toujours le même (❸), on appellera hauteur noire de t et on notera $b(t)$ cette quantité .

Définition des arbres rouge-noir

Un arbre rouge-noir t est un ABR (❶), dans lequel chaque noeud porte une information de couleur (rouge ou noir), et ayant les deux propriétés suivantes :

- le père d'un noeud rouge est noir (❷),
- le nombre de noeuds noirs le long d'un chemin de la racine à un sous arbre vide est toujours le même (❸), on appellera hauteur noire de t et on notera $b(t)$ cette quantité .

Exemples



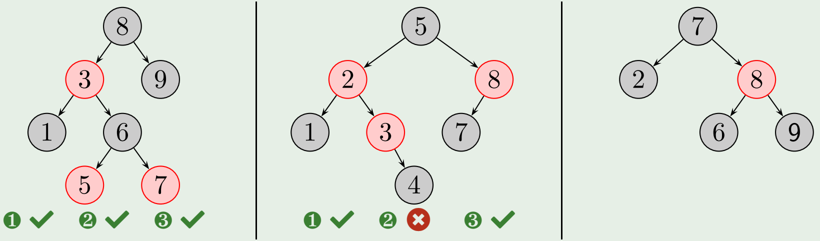
❶ ✓ ❷ ✓ ❸ ✓

Définition des arbres rouge-noir

Un arbre rouge-noir t est un ABR (❶), dans lequel chaque noeud porte une information de couleur (rouge ou noir), et ayant les deux propriétés suivantes :

- le père d'un noeud rouge est noir (❷),
- le nombre de noeuds noirs le long d'un chemin de la racine à un sous arbre vide est toujours le même (❸), on appellera hauteur noire de t et on notera $b(t)$ cette quantité .

Exemples

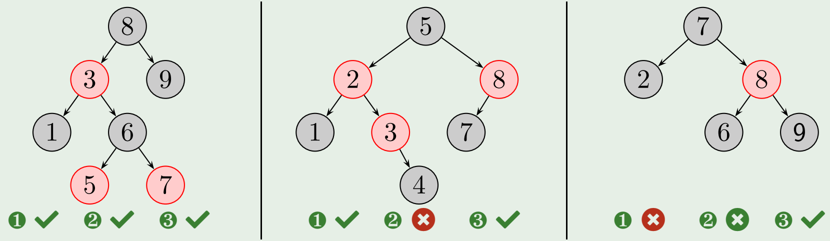


Définition des arbres rouge-noir

Un arbre rouge-noir t est un ABR (❶), dans lequel chaque noeud porte une information de couleur (rouge ou noir), et ayant les deux propriétés suivantes :

- le père d'un noeud rouge est noir (❷),
- le nombre de noeuds noirs le long d'un chemin de la racine à un sous arbre vide est toujours le même (❸), on appellera hauteur noire de t et on notera $b(t)$ cette quantité .

Exemples



Proriété d'équilibre des arbres rouge-noirs

Pour tout arbre rouge noir t :

Proriété d'équilibre des arbres rouge-noirs

Pour tout arbre rouge noir t :

- $h(t) \leqslant 2b(t)$

Proriété d'équilibre des arbres rouge-noirs

Pour tout arbre rouge noir t :

- $h(t) \leqslant 2b(t)$
- $2^{b(t)} \leqslant n(t) + 1$

Proriété d'équilibre des arbres rouge-noirs

Pour tout arbre rouge noir t :

- $h(t) \leqslant 2b(t)$
- $2^{b(t)} \leqslant n(t) + 1$

Conséquence : les arbres rouge-noir forment un ensemble d'arbres équilibrés.

Proriété d'équilibre des arbres rouge-noirs

Pour tout arbre rouge noir t :

- $h(t) \leqslant 2b(t)$
- $2^{b(t)} \leqslant n(t) + 1$

Conséquence : les arbres rouge-noir forment un ensemble d'arbres équilibrés.

Implémentation

Une implémentation en OCaml sera vue en TP, les opérations d'insertion et de suppression sont difficiles et reposent sur les rotations droite et gauche des ABR.

Définition

Un arbre binaire est strict si tous ses noeuds ont soit 0, soit 2 fils. Un noeud est donc soit une feuille (pas de fils), soit un noeud interne (deux fils).

Définition

Un arbre binaire est strict si tous ses noeuds ont soit 0, soit 2 fils. Un noeud est donc soit une feuille (pas de fils), soit un noeud interne (deux fils).

Exemples

Les arbres de codage de Huffman sont des arbres binaires stricts. Qu'on peut représenter en OCaml par le type :

```
type abs =
  | Feuille of int
  | Noeud of abs * abs
```

Ici les noeuds internes ne portent pas d'information.

Exercice

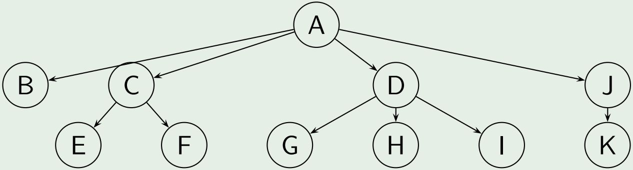
1. Donner une définition inductive des arbres binaires stricts.
2. Ecrire en OCaml la fonction renvoyant la taille d'un arbre binaire strict.
3. On note n le nombre de noeuds internes d'un arbres binaire strict et f son nombre de feuilles. Montrer que $f = n + 1$.
   On pourra raisonner par récurrence sur la taille de l'arbre ou dénombrer de deux façon différentes les noeuds ayant un père.

Définition

Un arbre est un ensemble de $n \geqslant 1$ noeuds structurés de la manière suivante :

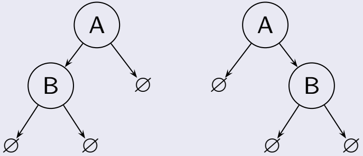
- un noeud particulier r est appelé la *racine* de l'arbre,
- les $n-1$ autres noeuds sont partitionnés en $k \geqslant 0$ sous ensembles disjoints qui forment autant d'arbres, appelés *sous-arbres* de r,
- la racine r est relié à la racine de chacun de ces sous-arbres par une arête.

Exemple

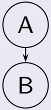


Remarques

- La séquence des sous arbres d'un noeud est appelée forêt.
- Un arbre réduit à un seul noeud est appelé feuille.
- Un arbre binaire n'est *pas* un arbre. En effet :
  - un arbre binaire peut être vide (et pas un arbre) ;
  - dans un arbre binaire on distingue le fils gauche du fils droit, c'est à dire que les deux arbres binaires ci-dessous sont différents :



Alors que le seul *arbre* ayant deux noeuds est :



Représentation en OCaml

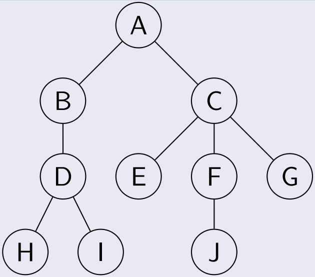
```ocaml
type 'a tree =
  | Node of 'a * 'a tree list
```

Exercice

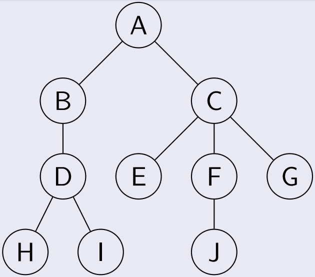
Ecrire une fonction `size : 'a tree -> int` qui renvoie le nombre de noeuds d'un arbre.

On pourra utiliser deux fonctions *mutuellement récursive* qui renvoient respectivement la taille d'un arbre et celle d'une forêt.

Représentation en C



Représentation en C

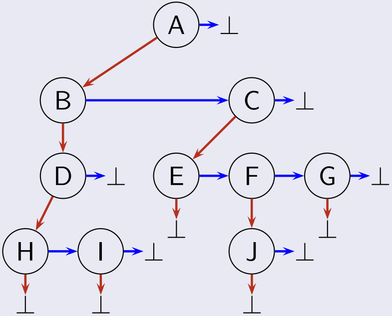


Principe : utiliser un pointeur vers le premier fils et un pointeur vers le frère suivant. En anglais, LCRS : *left child right sibling*.

Représentation en C

En notant en :

- $\longrightarrow$ le pointeur vers le fils gauche,
- $\longrightarrow$ le pointeur vers le frère suivant,
- $\perp$ le pointeur NULL.



Représentation en C

```c
struct node_s
{
    int value;
    struct node_s *leftchild;
    struct node_s *rightsibling;
};
typedef struct node_s node;
typedef node *tree;
```

Représentation en C

```c
struct node_s
{
    int value;
    struct node_s *leftchild;
    struct node_s *rightsibling;
};
typedef struct node_s node;
typedef node *tree;
```

Exercice

1. Ecrire une fonction de signature `tree create_tree(int value)` qui renvoie un arbre réduit à un noeud de valeur `value`.
2. Ecrire une fonction de signature `int size(tree t)` qui renvoie le nombre de noeuds d'un arbre.

Remarque

Arbres

```c
struct node_s
{
    int value;
    struct node_s *leftchild;
    struct node_s *rightsibling;
};
typedef struct node_s node;
typedef node *tree;
```

Arbres binaires

```c
struct noeud_s
{
    int valeur;
    struct noeud_s *sag;
    struct noeud_s *sad;
};
typedef struct noeud_s noeud;
typedef noeud *arbrebin;
```

Remarque

Arbres

```c
struct node_s
{
    int value;
    struct node_s *leftchild;
    struct node_s *rightsibling;
};
typedef struct node_s node;
typedef node *tree;
```

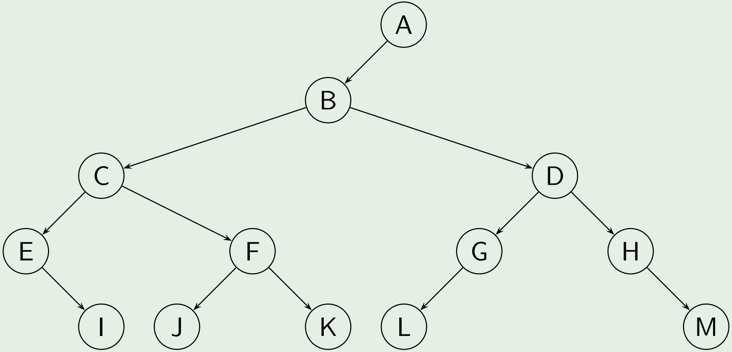
Arbres binaires

```c
struct noeud_s
{
    int valeur;
    struct noeud_s *sag;
    struct noeud_s *sad;
};
typedef struct noeud_s noeud;
typedef noeud *arbrebin;
```

On constate que les types représentant les arbres et les arbres binaires sont identiques, il y a un *isomorphisme naturel* entre les arbres et les arbres binaires. On peut convertir un arbre en arbre binaire et inversement

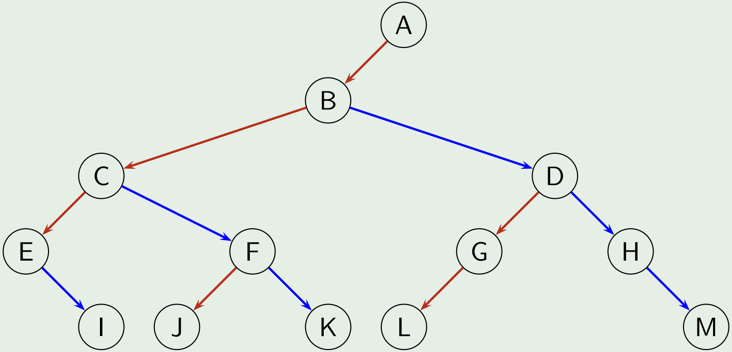
Exercice

Donner la représentation LCRS de l'arbre binaire suivant (on n'a pas fait figuré les sous arbres vides) puis sa représentation sous la forme d'arbre généralisé :



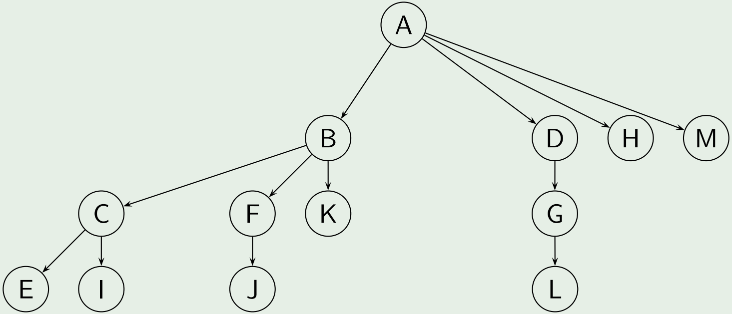
Exercice

Représentation LCRS :



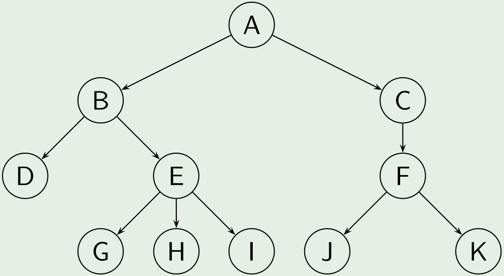
Exercice

Conversion en arbre généralisé :



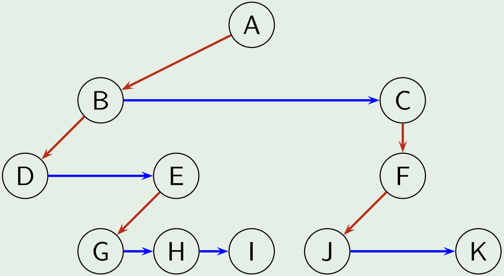
Exercice

Donner la représentation LCRS de l'arbre suivant puis sa représentation sous la forme d'arbre binaire :



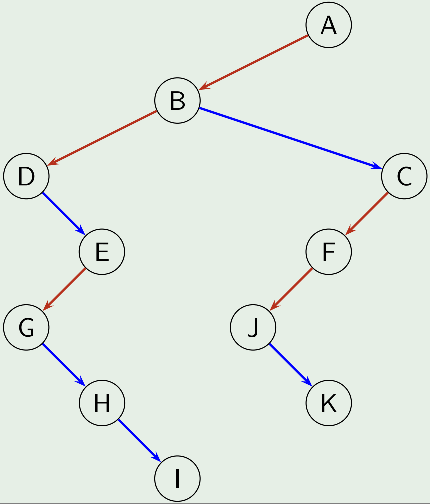
Exercice

Représentation LCRS :



Exercice

Conversion en arbre binaire :

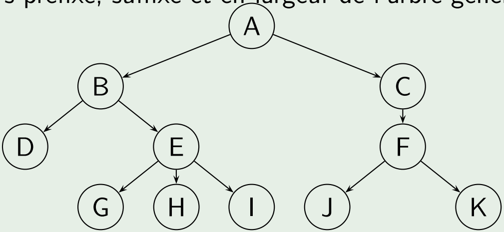


Parcours

La notion de parcours *infixe* n'a plus de sens cependant, pour un arbre généralisé, on peut définir les parcours suivants :

Exemples

Donner les parcours prefixe, suffixe et en largeur de l'arbre généralisé



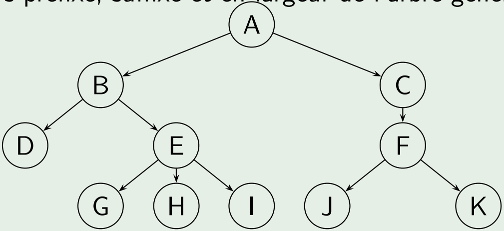
Parcours

La notion de parcours *infixe* n'a plus de sens cependant, pour un arbre généralisé, on peut définir les parcours suivants :

- prefixe : on visite le noeud avant ses fils,
- suffixe : on visite le noeud après ses fils,
- en largeur : on visite les noeuds par niveau.

Exemples

Donner les parcours prefixe, suffixe et en largeur de l'arbre généralisé



Remarques

Un arbre généralisé est un graphe (S, A) non orienté, connexe sans cycle.

- Acyclique : il n'existe pas de cycle dans l'arbre, c'est à dire qu'il n'existe pas de suite de noeuds $x_1, \dots, x_n$ tels que $\{x_1, x_2\}, \{x_2, x_3\}, \dots, \{x_{n-1}, x_n\}, \{x_n, x_1\} \in A$.

⚠ On peut choisir n'importe quel noeud comme racine.

Remarques

Un arbre généralisé est un graphe (S, A) non orienté, connexe sans cycle.

- S est l'ensemble des noeuds de l'arbre,
- Acyclique : il n'existe pas de cycle dans l'arbre, c'est à dire qu'il n'existe pas de suite de noeuds $x_1, \dots, x_n$ tels que $\{x_1, x_2\}, \{x_2, x_3\}, \dots, \{x_{n-1}, x_n\}, \{x_n, x_1\} \in A$.

⚠ On peut choisir n'importe quel noeud comme racine.

Remarques

Un arbre généralisé est un graphe (S, A) non orienté, connexe sans cycle.

- S est l'ensemble des noeuds de l'arbre,
- A est l'ensemble des arêtes de l'arbre, chaque arête est de la forme (x, y) où x est le père de y.
- Acyclique : il n'existe pas de cycle dans l'arbre, c'est à dire qu'il n'existe pas de suite de noeuds $x_1, \dots, x_n$ tels que $\{x_1, x_2\}, \{x_2, x_3\}, \dots, \{x_{n-1}, x_n\}, \{x_n, x_1\} \in A$.

⚠ On peut choisir n'importe quel noeud comme racine.

Remarques

Un arbre généralisé est un graphe (S, A) non orienté, connexe sans cycle.

- S est l'ensemble des noeuds de l'arbre,
- A est l'ensemble des arêtes de l'arbre, chaque arête est de la forme (x, y) où x est le père de y.
- Connexité : il existe un chemin entre deux noeuds quelconques de l'arbre., $\forall (x, y) \in S^2, x \neq y$, il existe $x_1, \dots x_n$ tels que $\{x, x_1\}, \{x_1, x_2\}, \dots, \{x_{n-1}, y\} \in A$.
- Acyclique : il n'existe pas de cycle dans l'arbre, c'est à dire qu'il n'existe pas de suite de noeuds $x_1, \dots, x_n$ tels que $\{x_1, x_2\}, \{x_2, x_3\}, \dots, \{x_{n-1}, x_n\}, \{x_n, x_1\} \in A$.

⚠ On peut choisir n'importe quel noeud comme racine.

Définition

La sérialisation d'un arbre est une représentation de l'arbre sous forme de chaîne de caractères (à des fins de sauvegarde, transmission, reconstruction ...).

Définition

La sérialisation d'un arbre est une représentation de l'arbre sous forme de chaîne de caractères (à des fins de sauvegarde, transmission, reconstruction ...).
Un unique parcours (prefixe, infixe, suffixe ou en largeur) *ne permet pas* de sérialiser un arbre, en effet des arbres différents peuvent avoir le même parcours.

Définition

La sérialisation d'un arbre est une représentation de l'arbre sous forme de chaîne de caractères (à des fins de sauvegarde, transmission, reconstruction . . .).
Un unique parcours (prefixe, infixe, suffixe ou en largeur) *ne permet pas* de sérialiser un arbre, en effet des arbres différents peuvent avoir le même parcours.
Les solutions suivantes sont envisageables dans le cas d'un un arbre binaire :

Définition

La sérialisation d'un arbre est une représentation de l'arbre sous forme de chaîne de caractères (à des fins de sauvegarde, transmission, reconstruction ...).
Un unique parcours (prefixe, infixe, suffixe ou en largeur) *ne permet pas* de sérialiser un arbre, en effet des arbres différents peuvent avoir le même parcours.
Les solutions suivantes sont envisageables dans le cas d'un un arbre binaire :

- stocker les parcours infixe et préfixe (mais cela impose de stocker les étiquettes en double) ;

Définition

La sérialisation d'un arbre est une représentation de l'arbre sous forme de chaîne de caractères (à des fins de sauvegarde, transmission, reconstruction ...).
Un unique parcours (prefixe, infixe, suffixe ou en largeur) *ne permet pas* de sérialiser un arbre, en effet des arbres différents peuvent avoir le même parcours.
Les solutions suivantes sont envisageables dans le cas d'un un arbre binaire :

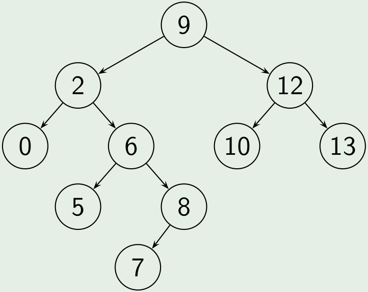
- stocker les parcours infixe et préfixe (mais cela impose de stocker les étiquettes en double) ;
- utiliser un parcours en largeur avec un marqueur spécial indiquant un fils absent ;

Définition

La sérialisation d'un arbre est une représentation de l'arbre sous forme de chaîne de caractères (à des fins de sauvegarde, transmission, reconstruction ...).
Un unique parcours (prefixe, infixe, suffixe ou en largeur) *ne permet pas* de sérialiser un arbre, en effet des arbres différents peuvent avoir le même parcours.
Les solutions suivantes sont envisageables dans le cas d'un un arbre binaire :

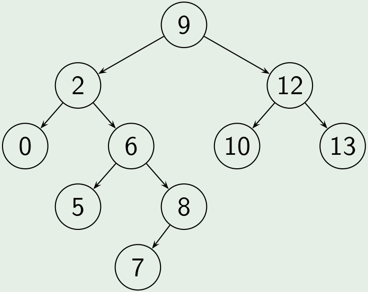
- stocker les parcours infixe et préfixe (mais cela impose de stocker les étiquettes en double) ;
- utiliser un parcours en largeur avec un marqueur spécial indiquant un fils absent ;
- utiliser un parcours prefixe avec un marquer pour les fils absents

Exemple



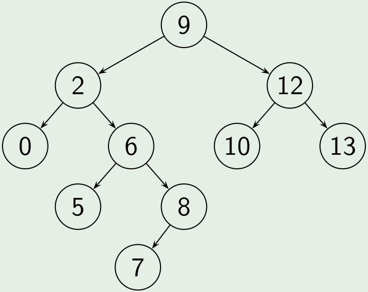
- Parcours préfixe et infixe :
- Parcours en largeur avec le marqueur # pour fils absent :
- Parcours préfixe avec le marqueur # pour fils absent :

Exemple



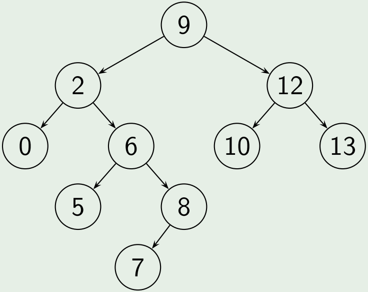
- Parcours préfixe et infixe :
  9 2 0 6 5 8 7 12 10 13 et 0 2 5 6 7 8 9 10 12 13
- Parcours en largeur avec le marqueur # pour fils absent :
- Parcours préfixe avec le marqueur # pour fils absent :

Exemple



- Parcours préfixe et infixe :
  9 2 0 6 5 8 7 12 10 13 et 0 2 5 6 7 8 9 10 12 13
- Parcours en largeur avec le marqueur # pour fils absent :
  9 2 12 0 6 10 13 # # 5 8 # # # # # # 7
- Parcours préfixe avec le marqueur # pour fils absent :

Exemple



- Parcours préfixe et infixe :
  9 2 0 6 5 8 7 12 10 13 et 0 2 5 6 7 8 9 10 12 13
- Parcours en largeur avec le marqueur # pour fils absent :
  9 2 12 0 6 10 13 # # 5 8 # # # # # # 7
- Parcours préfixe avec le marqueur # pour fils absent :
  9 2 0 # # 6 5 # # 8 7 # 12 10 # # 13 # #