Module EA4 – Éléments d'Algorithmique II

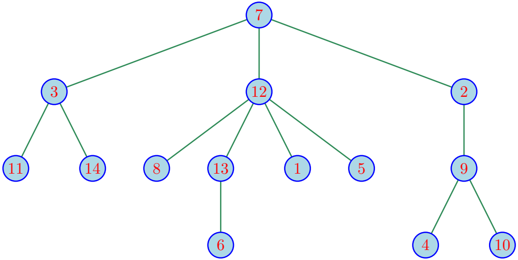
Outils pour l'analyse des algorithmes

Dominique Poulalhon
`dominique.poulalhon@irif.fr`

Université Paris Diderot
L2 Informatique & Math-Info
Année universitaire 2019-2020

Arbres Binaires de Recherche

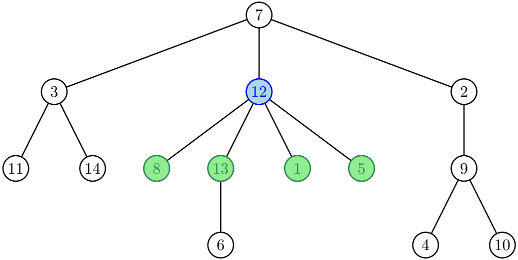
II. Généralités sur les arbres



sommets contenant des étiquettes reliés par des arêtes

Ici, 14 sommets, reliés par 13 arêtes, et étiquetés de 1 à 14

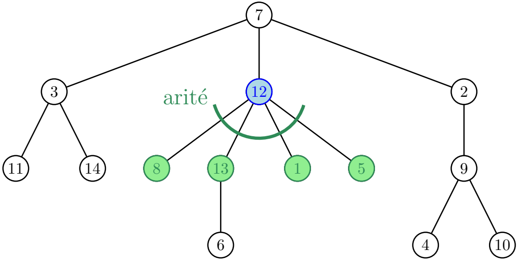
Rappels de terminologie



hiérarchie entre les sommets : père , fils

Le sommet d'étiquette 12 a 4 fils (d'étiquettes 8, 13, 1 et 5).

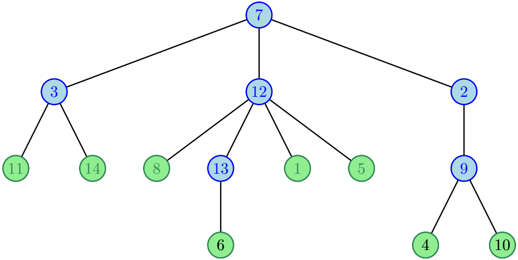
Rappels de terminologie



hiérarchie entre les sommets : père , fils

Le sommet d'étiquette 12 a 4 fils (d'étiquettes 8, 13, 1 et 5).
On dit qu'il est d'arité 4, ou que c'est un nœud quaternaire.

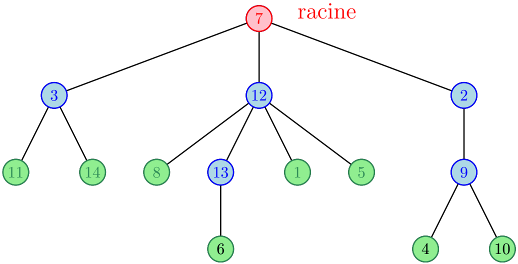
Rappels de terminologie



sommet = nœud ou feuille

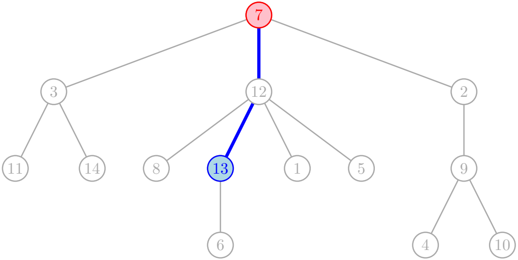
Les sommets d'arité 0 sont appelés *feuilles* – ici il y en a 8.
Les autres sont les *nœuds* – ici, 6.

Rappels de terminologie



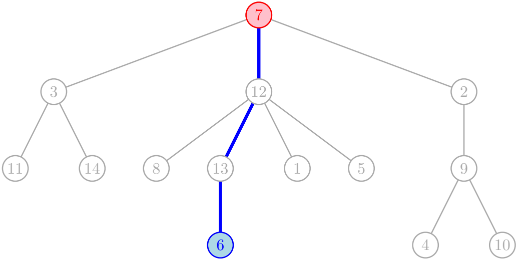
sommet = nœud ou feuille

Les sommets d'arité 0 sont appelés *feuilles* – ici il y en a 8.
Les autres sont les *nœuds* – ici, 6.
Le seul sommet sans père est la racine – c'est en général un nœud.



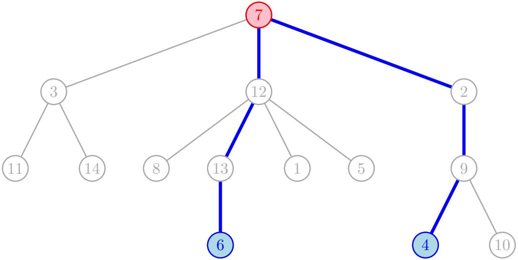
profondeur d'un sommet = distance à la racine

Le sommet d'étiquette 13 est à profondeur 2.



hauteur de l'arbre = profondeur maximale

La hauteur de cet arbre est 3.

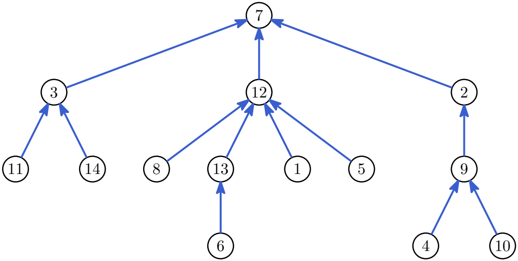


hauteur de l'arbre = profondeur maximale

La hauteur de cet arbre est 3.
Les sommets à profondeur 3 sont donc des feuilles
(et il peut très bien y en avoir plusieurs)

Représentation des arbres – solution I

en gardant pour chaque sommet la référence du père (dans le sommet, ou regroupées dans un tableau)



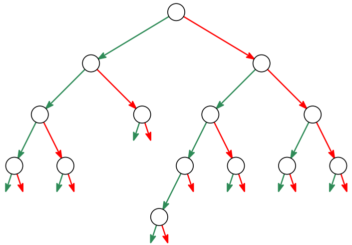
avantage : représentation extrêmement compacte

(gros) inconvénient : l'arbre ne peut être parcouru (efficacement) que de bas en haut... ce qui n'est pas ce qu'on souhaite faire en général

$\Longrightarrow$ essentiellement utilisée pour représenter une partition en sous-ensembles (structure *union-find*, *cf* cours d'algo de L3)

Représentation des arbres – solution II

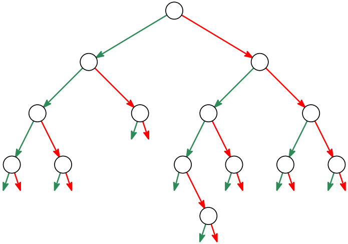
cas des arbres d'arité fixe, en particulier les arbres binaires : pour chaque nœud, stocker une référence de chaque fils



arbre binaire : au plus 2 fils par nœud, avec distinction entre fils gauche et droit. (échanger les deux fils donne un arbre différent)

Représentation des arbres – solution II

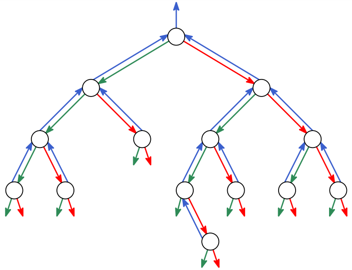
cas des arbres d'arité fixe, en particulier les arbres binaires : pour chaque nœud, stocker une référence de chaque fils



arbre binaire : au plus 2 fils par nœud, avec distinction entre fils gauche et droit. (échanger les deux fils donne un arbre différent)

Représentation des arbres – solution II

cas des arbres d'arité fixe, en particulier les arbres binaires : pour chaque nœud, stocker une référence de chaque fils (et éventuellement du père)



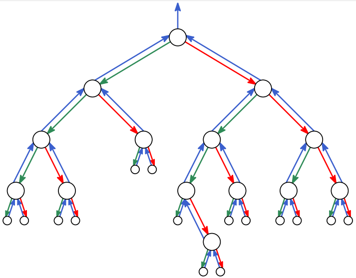
arbre binaire : au plus 2 fils par nœud, avec distinction entre fils gauche et droit. (échanger les deux fils donne un arbre différent)

Avec le père, chaque sommet stocke donc 3 références/pointeurs/champs. Ce 3e pointeur facilite énormément les manipulations, en particulier les parcours.

Mais certains ne pointent sur « rien » : le père de la racine, les fils absents. Cela peut en fait compliquer les choses : ces « rien » ne savent rien... impossible de distinguer le « non-fils-gauche » d'un sommet du « non-fils-droit » d'un autre.

Représentation des arbres – solution II

cas des arbres d'arité fixe, en particulier les arbres binaires : pour chaque nœud, stocker une référence de chaque fils (et éventuellement du père)



Il est plus commode de *compléter* un arbre binaire en rajoutant une feuille vide à la place de chaque pointeur vers un sommet absent. C'est plus homogène, et cela facilite grandement la programmation des modifications.

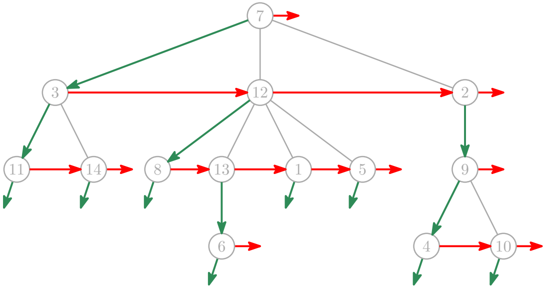
on manipule donc plutôt des arbres binaires *complets* : exactement 2 fils par nœud

la « complétion » met exactement en correspondance les arbres binaires (quelconques) à n sommets et les arbres binaires complets à n nœuds binaires (et $n+1$ feuilles vides).

Représentation des arbres – solution III

(Pour la culture générale, mais pas indispensable pour le cours sur les ABR)

cas général : références du fils aîné et du frère cadet



Le fils aîné sert de référence à la liste chaînée des fils.

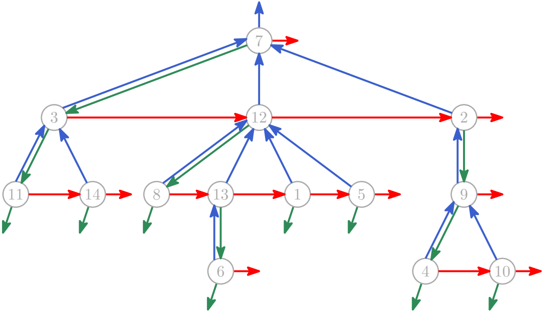
Autre manière de voir les choses, on représente les arbres généraux par des arbres binaires : le fils aîné d'un sommet devient son fils gauche, et son frère cadet devient son fils droit. On obtient un arbre binaire dont la racine n'a pas de fils droit.

Représentation des arbres – solution III

(Pour la culture générale, mais pas indispensable pour le cours sur les ABR)

cas général : références du fils aîné et du frère cadet

(et éventuellement du père)



Le fils aîné sert de référence à la liste chaînée des fils.

Autre manière de voir les choses, on représente les arbres généraux par des arbres binaires : le fils aîné d'un sommet devient son fils gauche, et son frère cadet devient son fils droit. On obtient un arbre binaire dont la racine n'a pas de fils droit.

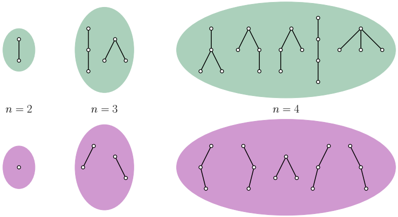
Il est plus commode que le père pointe réellement sur le père (dans l'arbre de départ), pas sur le « père » dans l'arbre binaire correspondant.

Théorème

Les arbres binaires à n sommets sont en bijection avec les arbres binaires complets à n nœuds (binaires) et $n+1$ feuilles.

Théorème

Les arbres à n sommets sont en bijection avec les arbres binaires à $n-1$ sommets (et donc avec les arbres binaires complets à $n-1$ nœuds)



Théorème (*admis*)

Le nombre d'arbres à n sommets est $\dfrac{1}{n+1}\dbinom{2n}{n} \in \Theta\left(\dfrac{4^n}{n\sqrt{n}}\right)$

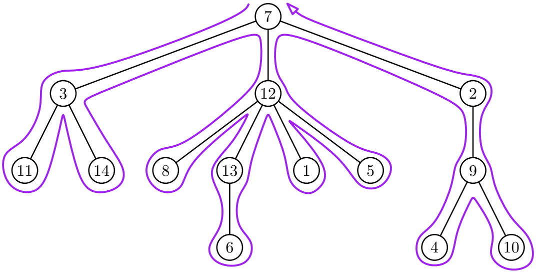
Représentation des arbres

À partir de maintenant, on suppose qu'on dispose des fonctions suivantes, dont le code dépend de la représentation choisie :

- `pere(noeud)`
- `liste_des_fils(noeud)`
- `etiquette(noeud)`
- (pour les arbres binaires) `gauche(noeud)` et `droite(noeud)`

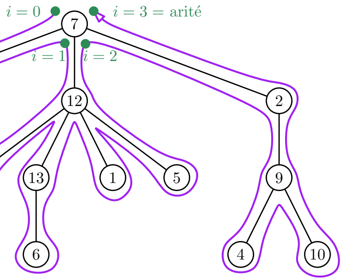
Parcours en profondeur générique

Effectuer un parcours en profondeur d'un arbre consiste à en faire le tour complet, en partant de la racine et en le tenant fermement de la main gauche :



Parcours en profondeur générique

Effectuer un parcours en profondeur d'un arbre consiste à en faire le tour complet, en partant de la racine et en le tenant fermement de la main gauche :



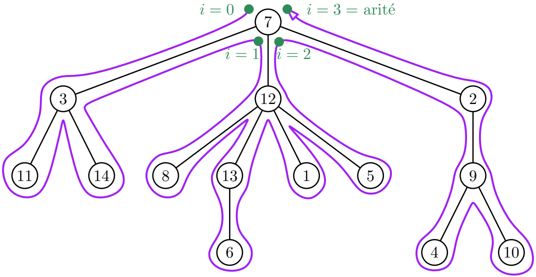
Ce tour passe plusieurs fois en chaque sommet : en arrivant depuis le père et en allant vers le 1$^{\text{er}}$ fils, en revenant du 1$^{\text{er}}$ fils et en allant vers le 2$^{\text{e}}$, ..., et finalement après avoir visité le dernier fils et en remontant vers le père.

Chaque passage en un sommet est l'occasion d'effectuer un traitement : un affichage, un stockage, un décompte... ou rien du tout.

Un tel parcours se décrit particulièrement simplement de façon récursive.

Parcours en profondeur générique

```
def parcours_generique(racine) :
  L = liste_des_fils(racine) # éventuellement vide
  for i, noeud in enumerate(L) :
    traitement(i, racine)
    # = traitement après avoir visité i sous-arbres
    parcours_generique(noeud)
  traitement(len(L), racine)
```



Parcours en profondeur générique

```
def parcours_generique(racine) :
  L = liste_des_fils(racine) # éventuellement vide
  for i, noeud in enumerate(L) :
    traitement(i, racine)
    # = traitement après avoir visité i sous-arbres
    parcours_generique(noeud)
  traitement(len(L), racine)
```

Théorème

parcours_generique(racine) visite tous les sommets de l'arbre enraciné en racine, en temps $\Theta(n)$ si chaque traitement est en $\Theta(1)$

Correction : par récurrence (forte) sur n :

- si $n = 0$ ou $n = 1$, c'est manifestement vrai
- soit $n > 1$; si parcours_generique() est correct pour tous les arbres ayant au plus $n - 1$ sommets, alors chacun des appels récursifs visite tout le sous-arbre dont la racine est le fils correspondant de racine puisque ces sous-arbres ont (au moins) un sommet de l'arbre entier ; parcours_generique(racine) visite donc tous les sommets de l'arbre.

Complexité : exactement un appel récursif par sommet !

- si i = 0 : on parle de pré-traitement
- si i = len(L) : on parle de post-traitement

Parcours en profondeur spécifiques

Trois cas particuliers (et particulièrement importants) :

Parcours préfixe : `traitement(i, racine)` vide sauf si `i = 0`

```
def parcours_prefixe(racine) :
  pre_traitement(racine)
  for noeud in liste_des_fils(racine) : parcours(noeud)
```

Parcours postfixe : `traitement(i, racine)` vide sauf si `i = len(L)`

```
def parcours_postfixe(racine) :
  for noeud in liste_des_fils(racine) : parcours(noeud)
  post_traitement(racine)
```

Parcours infixe : cas binaire avec seulement un traitement intermédiaire

```
def parcours_infixe(racine) :
  if racine != None : # vérification nécessaire car les appels récursifs ne la font pas
    parcours(gauche(racine))
    traitement(racine)
    parcours(droite(racine))
```