INF601 : Algorithme et Structure de données

Cours 2 : TDA Arbre Binaire

B. Jacob

IC2/LIUM

27 février 2010

Plan

1. Introduction
2. Primitives du TDA Arbin
3. Réalisations du TDA Arbin
   - par cellules chaînées
   - par cellules contiguës
   - par curseurs (faux pointeurs)
   - Réalisation des Arbres parfaits
4. Recherche d'un élément
5. Adjonction d'un élément
   - Adjonction aux feuilles
   - Adjonction à la racine
6. Suppression d'un élément
7. Conclusion sur les arbres

Plan

1 Introduction

Méthodes arborescentes

- Cours précédent :
  si ensemble d'éléments sont dans un TDA Liste triée
  $\longrightarrow$ recherche d'un élément en $log(n)$ comparaisons
- Problème : représentation contiguë (liste) mal adaptée lorsque l'ensemble évolue dynamiquement
  $\longrightarrow$ adjonction et suppression peuvent être en $O(n)$
- Solution : pour que les 3 opérations
  - recherche
  - adjonction
  - suppression

  soient efficaces $\longrightarrow$ structures arborescentes

Méthodes arborescentes

Elles reposent sur

1. une comparaison avec la valeur d'un noeud
2. "l'aiguillage" de la poursuite de la recherche dans un sous-arbre en fonction du résultat de la comparaison

La structure fondamentale des méthodes arborescentes
$\longrightarrow$ celle de l'arbre binaire de recherche

Définitions informelle

Définition informelle

un arbre = ensemble de sommets tel que :

- ∃ un sommet unique appelé racine r qui n'a pas de supérieur
- Tous les autres sommets sont atteints à partir de r par une branche unique

Définition récursive (et constructive)

un arbre =

- une racine r
- une liste d'arbres disjoints $A_1, \ldots, A_n$ (sous-arbres)

Un sommet de l'arbre est la racine d'un sous-arbre

Terminologie

- père d'un sommet : le prédécesseur d'un sommet
- fils d'un sommet : les successeurs d'un sommet
- frères : des sommets qui ont le même père
- noeud = sommet
- racine : noeud sans père
- feuille : noeud sans fils
- branche : chemin entre 2 noeuds

Mesures sur les arbres

- Niveau (profondeur) d'un noeud : la longueur de la branche depuis la racine
- Hauteur d'un noeud : la longueur de la plus longue branche de ce noeud jusqu'à une feuille
- Hauteur d'un arbre : la hauteur de la racine
- Taille d'un arbre : nombre de ses sommets

Arbres Binaires

Définition informelle

- Dans un arbre binaire tout noeud a au plus deux fils
- Un arbre binaire possède exactement deux sous-arbres (éventuellement vides)

Définition récursive (et constructive)

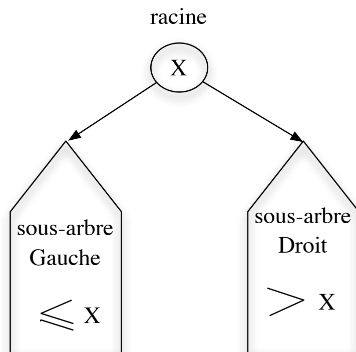
Un arbre binaire est

- Soit vide
- Soit composé
  - d'une racine r
  - de 2 sous arbres binaires ABG et ABD disjoints
    - ABG : sous Arbre Binaire Gauche
    - ABD : sous Arbre Binaire Droit

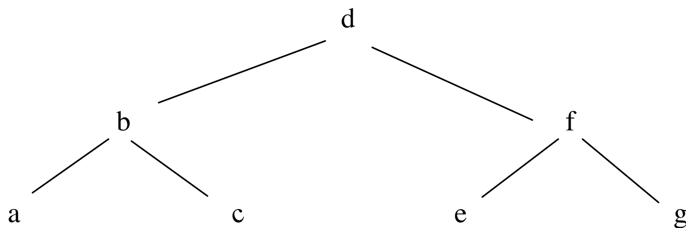
Eléments dans un arbre binaire

Généralement dans un arbre binaire

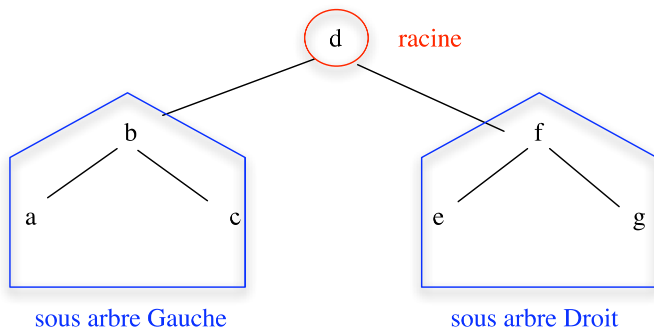
- noeud racine contient un élément X
- dans ABG $\rightarrow$ noeuds $\leq X$
- dans ABD $\rightarrow$ noeuds $> X$



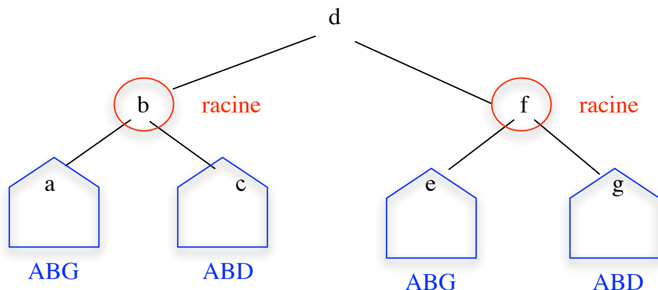
Organisation d'un arbre binaire



Organisation d'un arbre binaire



Organisation d'un arbre binaire

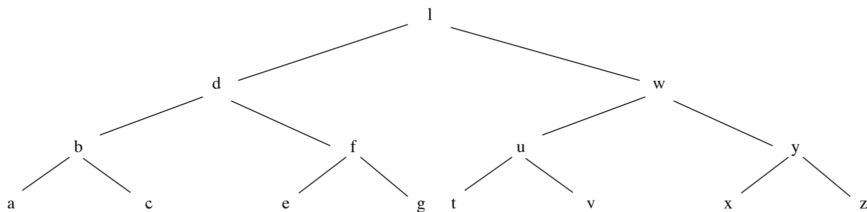


Arbres binaires particuliers

- Arbre binaire dégénéré, filiforme : Chaque noeud possède exactement un fils
  $\longrightarrow$ à éviter
- Arbre binaire complet (uniforme) : Chaque niveau est complètement rempli
  I.E. Tout sommet est soit une feuille au dernier niveau, soit possède exactement 2 fils
  $\longrightarrow$ situation idéale
- Arbre binaire parfait (presque complet) : Tous les niveaux sont complètement remplis sauf éventuellement le dernier et dans ce cas les feuilles sont le plus à gauche possible
- Arbre binaire équilibré : La différence de hauteur entre 2 frères ne peut dépasser 1

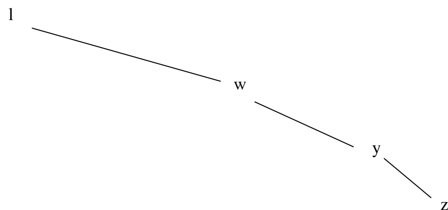
Arbres particuliers

Arbre Parfait → situation idéale



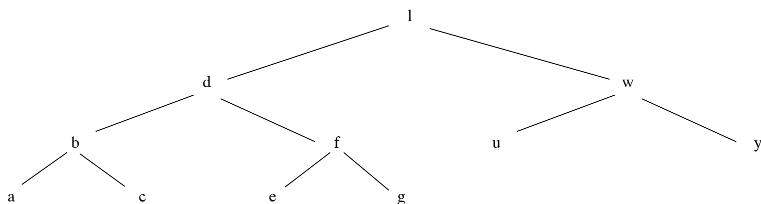
Arbres particuliers

Arbre dégénéré $\longrightarrow$ Pire des situations



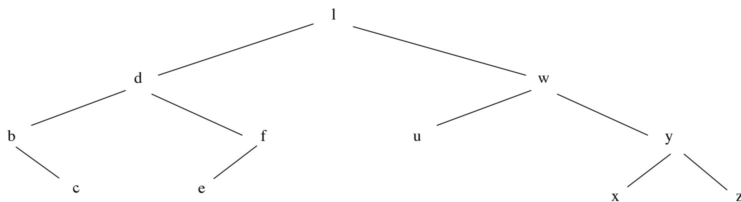
Arbres particuliers

Arbre presque parfait



Arbres particuliers

Arbre équilibré



Plan

2 Primitives du TDA Arbin

Définitions des primitives

Utilise le TDA ELEMENT

Primitives :

- Création et destruction
- Construction
- Primitives de modification d'un arbre non vide
- Accès aux caractéristiques des noeuds (aux éléments)

Plan

3. Réalisations du TDA Arbin
   - par cellules chaînées
   - par cellules contiguës
   - par curseurs (faux pointeurs)
   - Réalisation des Arbres parfaits

Plan

3. Réalisations du TDA Arbin
   - par cellules chaînées
   - par cellules contiguës
   - par curseurs (faux pointeurs)
   - Réalisation des Arbres parfaits

Pointeurs sur ABG et ABD

Un arbre binaire est soit

- un pointeur sur le noeud racine (arbre non vide)
- le pointeur NULL (arbre vide)

Un noeud est une structure à trois champs :

- Une étiquette (élément)
- le sous-arbre gauche
- le sous-arbre droit

Avantages :

- Définition récursive, simple à programmer,
- la plus utilisée

Inconvénients : consomme de la mémoire dynamique

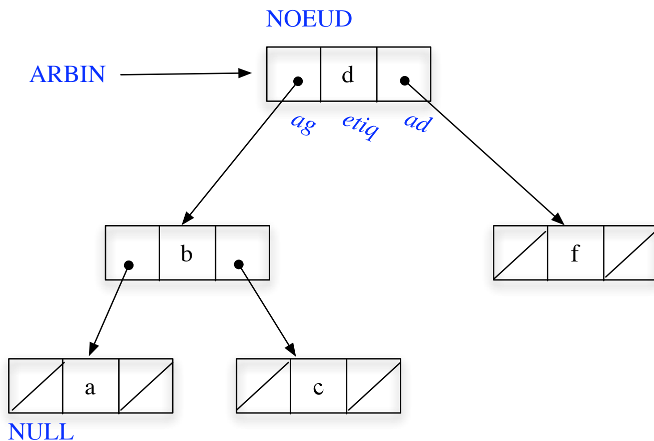
Traduction en C

```
/* Definition d'un noeud */
typedef struct noeud {
        ELEMENT etiq;             /* Etiquette */
        struct noeud * ag,        /* ABG */
                     * ad;        /* ABD */
} NOEUD;

/* Definition d'un arbre
 *  = un pointeur sur son Noeud racine */
typedef  NOEUD * ARBIN  ;

/* l'arbre vide est le pointeur null*/
#define ARBRE_VIDE NULL
```

Schéma réalisation par pointeurs



Plan

3 Réalisations du TDA Arbin

- par cellules chaînées
- par cellules contiguës
- par curseurs (faux pointeurs)
- Réalisation des Arbres parfaits

Par tableau

Un arbre est une structure à deux champs :

- Un tableau où sont mémorisés les noeuds
- Un entier qui donne l'indice de la racine dans le tableau

Un noeud est une structure à 3 champs :

- L'étiquette du noeud
- Les indices de ses fils gauche et droit (ou 0 si pas de fils)

Inconvénients :

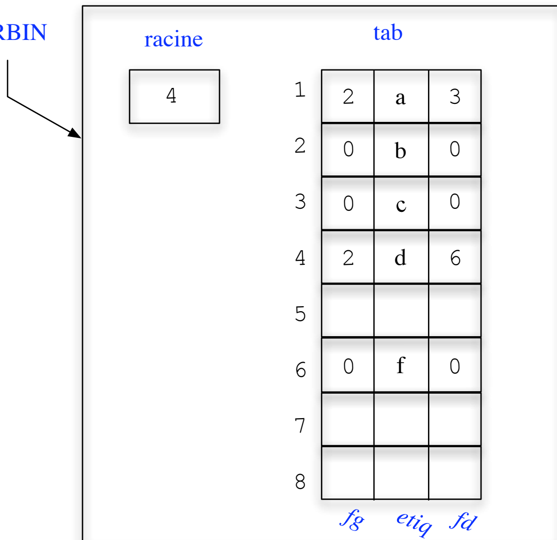
- Définition non récursive (arbre $\neq$ sous-arbre)
- Utilisée uniquement si on traite un arbre unique

Traduction en C

```
/* Definition d'un noeud */
typedef struct noeud {
    ELEMENT etiq; /* Etiquette */
    int fg,       /* ABG */
        fd;       /* ABD */
} NOEUD;

/* Definition d'un arbre */
typedef struct {
    NOEUD tab[TAILLE_MAX];
    int racine ;
} rep;
typedef  rep * ARBIN ;
```

Schéma réalisation par tableau



Plan

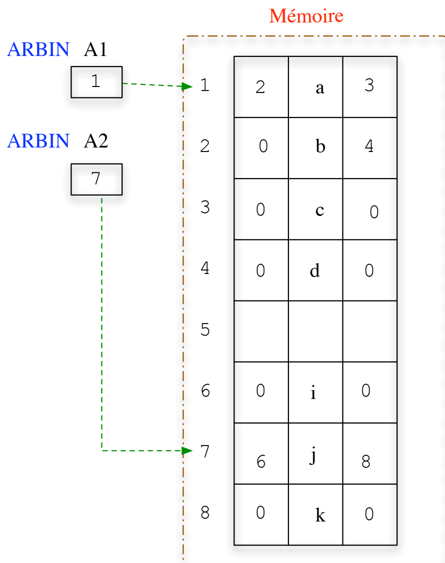
3. Réalisations du TDA Arbin
   - par cellules chaînées
   - par cellules contiguës
   - par curseurs (faux pointeurs)
   - Réalisation des Arbres parfaits

Simulation pointeurs sur ABG et ABD

(même principe que les faux pointeurs dans le TDA Liste)

- Simulation de la mémoire : les noeuds sont dans un tableau en variable globale
- Un arbre = indice de la racine (0 si vide)
- Un noeud est une structure à 3 champs
  - l'étiquette du noeud
  - indice fils ABG (0 si vide)
  - indice fils ABD (0 si vide)
- 2 stratégies de gestion des cellules disponibles
  1. Chaîner entre elles les cellules disponibles
  2. Marquer les cellules libres et parcourir le tableau pour trouver la 1re place libre

Schéma réalisation par faux pointeurs



Plan

3. Réalisations du TDA Arbin
   - par cellules chaînées
   - par cellules contiguës
   - par curseurs (faux pointeurs)
   - Réalisation des Arbres parfaits

Cas particuliers des arbre parfaits

Solution efficace (accès + place mémoire) de réalisation d'un arbre parfait A de N étiquettes

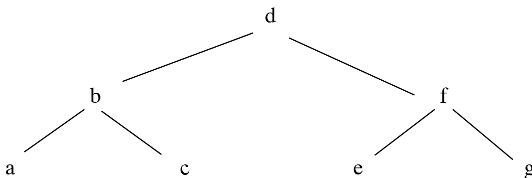
$\rightarrow$ un tableau avec :

- 1 : indice de la racine de A
- T[i] étiquette du père
- T[2i] étiquette du fils gauche
- T[2i+1] étiquette du fils droit

Attention : inefficace si N "loin de" n^2 (si A n'est pas parfait)

Schéma Arbre parfait avec tableau

Si A est l'abre parfait suivant :



Alors la réalisation de A est le tableau :

	d	b	f	c	a	e	f
0	1	2	3	4	5	6	7

Plan

4 Recherche d'un élément

Parcours d'arbres binaires

Recherche d'un élément $\longrightarrow$ parcours d'un arbre

Plusieurs types de parcours (selon application)

- Parcours en profondeur à main gauche (le + utilisé)
- Parcours récursif général
- Parcours particuliers (parcours préfixe, infixe, postfixe)
- Parcours en largeur (par niveaux)

Recherche en profondeur à main gauche

- chemin qui descend toujours le plus à gauche possible
- dans ce parcours chaque noeud est rencontré 3 fois
  - 1^{iere} fois : à la descente à gauche dans ABG
    $\longrightarrow$ on applique au noeud le traitement 1
  - 2^{ieme} fois : quand ABG a été parcouru, remontée par la gauche descente à gauche dans ABD
    $\longrightarrow$ on applique au noeud le traitement 2
  - 3^{ieme} et dernière fois : quand ABG et ABD ont été parcourus, en remontant à droite
    $\longrightarrow$ on applique au noeud le traitement 3
- si l'arbre est vide on lui applique un traitement appelé terminaison

Récursif à main gauche

```
Parcours(ARBIN A)
Début
    SI A est vide ALORS
        terminaison(A)
    SINON
        traitement1 (A)
        Parcours(sous-arbreGauche(A))
        traitement2 (A)
        Parcours(sous-arbreDroit(A))
        traitement3 (A)
    FSI
Fin
```

Cas particuliers

Lorsque le noeud est traité une seule fois

- Parcours préfixe (Racine Gauche Droit)
  $\longrightarrow$ le noeud racine est traité au premier passage
  avant le parcours des sous-arbres
- Parcours infixe ou symétrique (Gauche Racine Droit)
  $\longrightarrow$ le noeud racine est traité au second passage
  après le parcours du sous-arbre gauche
  et avant le parcours du sous-arbre droit
- Parcours postfixe (Gauche Droit Racine)
  $\longrightarrow$ le noeud racine est traité au dernier passage
  après le parcours des sous-arbres

Parcours particuliers

Parcours préfixé R G D

→ le noeud racine est traité au premier passage avant le parcours des sous-arbres

```
Parcours(ARBIN A)
Début
    SI A est vide ALORS
        terminaison(A)
    SINON

        traitement (A)

        Parcours(ABG de A)

        Parcours(ABD de A)

    FSI
Fin
```

Parcours particuliers

Parcours symétrique G R D

→ le noeud racine est traité au second passage après le parcours du sous-arbre gauche et avant le parcours du sous-arbre droit

```
Parcours(ARBIN A)
Début
    SI A est vide ALORS
        terminaison(A)
    SINON

        Parcours(ABG de A)

        traitement (A)

        Parcours(ABD de A)

    FSI
Fin
```

Parcours particuliers

Parcours postfixé G D R

$\longrightarrow$ le noeud racine est traité au troisième passage après le parcours des sous-arbres gauche et droit

```
Parcours(ARBIN A)
Début
    SI A est vide ALORS
        terminaison(A)
    SINON

        Parcours(ABG de A)

        Parcours(ABD de A)

        traitement (A)

    FSI
Fin
```

Parcours en Largeur

```
ParcoursEnLargeur(ARBIN A)
Début
    créer une file vide F
    SI A est non vide ALORS
        Enfiler A dans F
        TQ F non vide FRE
            A <-- Défiler(F)
            traitement(A)
            SI ABG de A non vide ALORS
                Enfiler ABG de A dans F
            FSI
            SI  ABD de A non vide ALORS
                Enfiler ABD de A dans F
            FSI
        FTQ
    FSI
    détruire F
Fin
```

Plan

5 Adjonction d'un élément
- Adjonction aux feuilles
- Adjonction à la racine

Construction d'un arbre

Un arbre A se construit par adjonction successives d'éléments x
2 méthodes principales :

- Adjonction aux feuilles :
  $\longrightarrow$ ajout de chaque x à une feuille de A
  $\longrightarrow$ construction "sur place"
- Adjonction à la racine :
  $\longrightarrow$ x devient la nouvelle racine de A
  $\longrightarrow$ construction "à côté"

Préparation

On transforme ELEMENT x en noeud de l'Arbre X
Soit ARBIN X :

- valeur de X $\leftarrow$ x
- sous arbre gauche de X $\leftarrow$ vide
- sous arbre droit de X $\leftarrow$ vide

Insertion à une feuille

```
ARBIN ArbinAjouterFeuille( ARBIN A ,ARBIN X)
Début
    SI A est vide ALORS
        A <— X
        renvoyer A
    SINON
        SI X <= A ALORS
            renvoyer ArbinAjouterFeuille( ABG(A) ,X)
        SINON
            renvoyer ArbinAjouterFeuille( ABD(A) ,X)
        FSI
    FSI
Fin
```

Insertion à la racine

Ajout à la racine

$\longrightarrow$ ajout à n'importe quel niveau de A

$\longrightarrow$ à rapprocher des méthodes auto-adaptatives des listes

Méthode pour ajouter X dans A :

1. Couper A en 2 ARBIN G et D

   G contient tous les éléments de $A \leq X$
   D contient tous les éléments de $A > X$

2. Former un nouvel ARBIN A' dont la racine avec :
   - étiquette de A' $\leftarrow X$
   - ABG(A') $\leftarrow G$
   - ABD(A') $\leftarrow D$

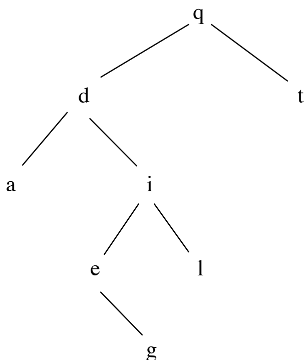
Méthode de coupure

Coupure d'un ELEMENT X dans un ARBIN A en 2 ARBIN G et D

- Il n'est pas nécessaire de parcourir TOUS les noeuds de A
  $\longrightarrow$ seulement les noeuds N situés sur le chemin de recherche de X dans A
- si noeud $N \leq X$: $G \leftarrow N + ABG(N)$
  sur le bord droit de G
- si noeud $N > X$: $D \leftarrow N + ABD(N)$
  sur le bord gauche de D

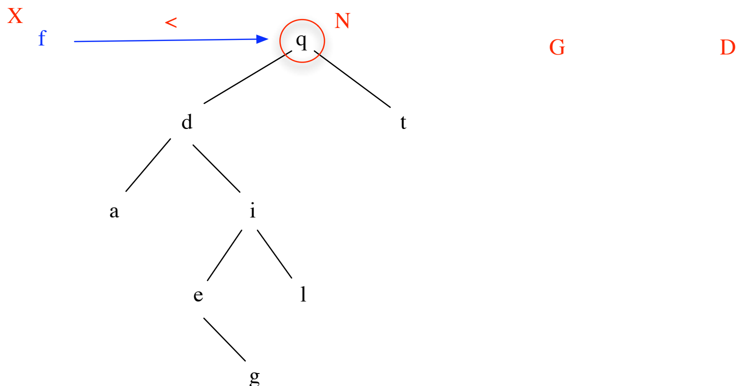
Exemple de coupure

f

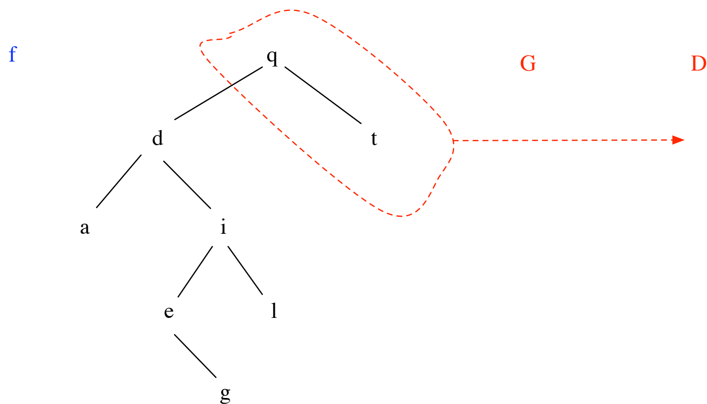


G D

Exemple de coupure

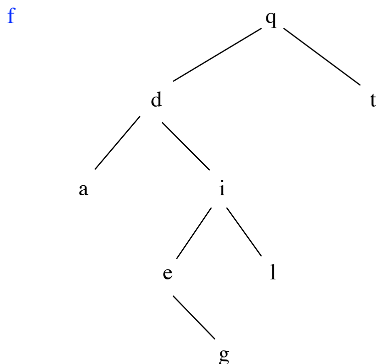


Exemple de coupure



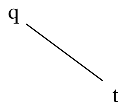
Exemple de coupure

f

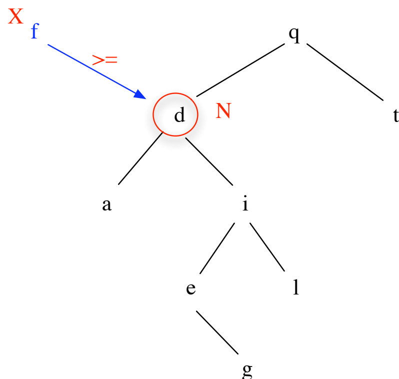


G

D

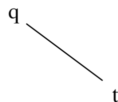


Exemple de coupure

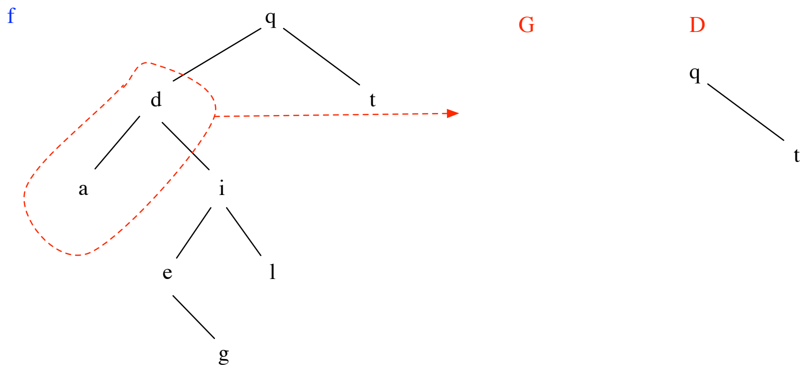


G

D

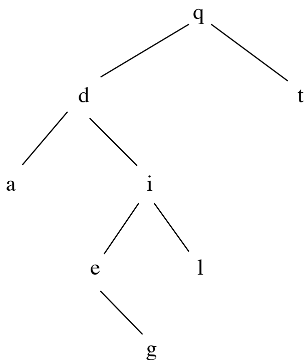


Exemple de coupure

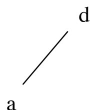


Exemple de coupure

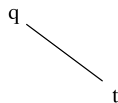
f



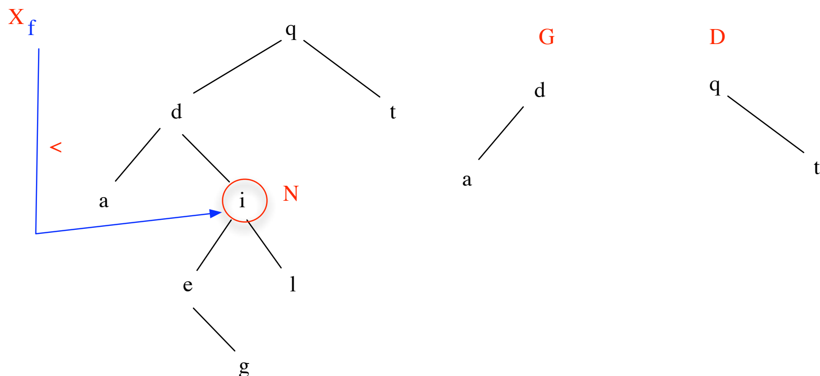
G



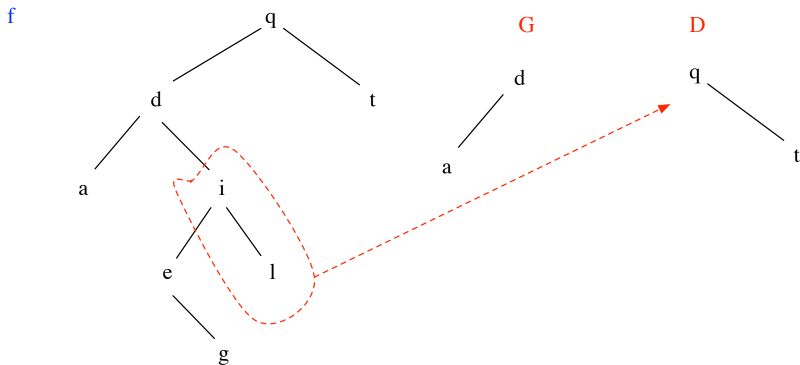
D



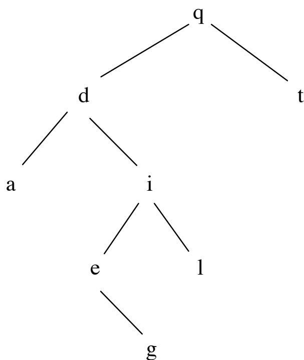
Exemple de coupure

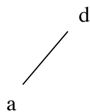


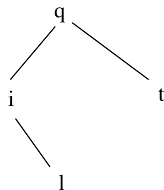
Exemple de coupure



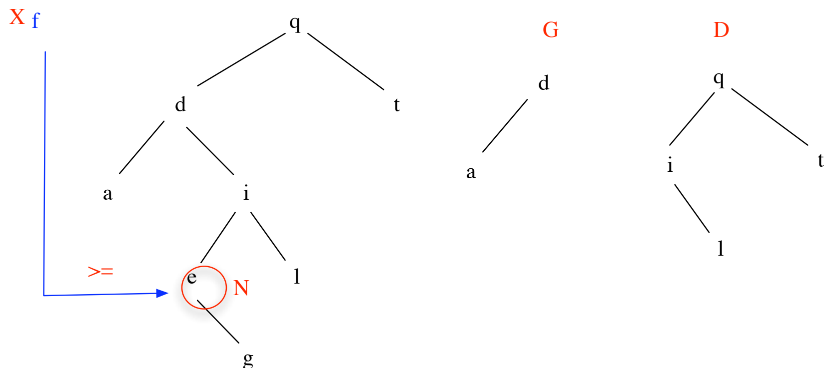
Exemple de coupure



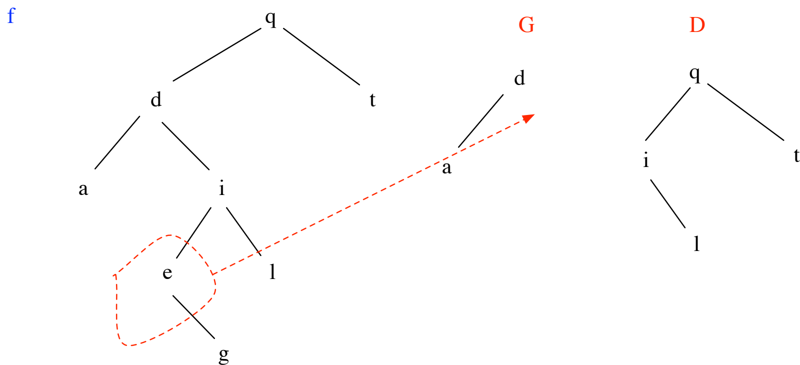




Exemple de coupure

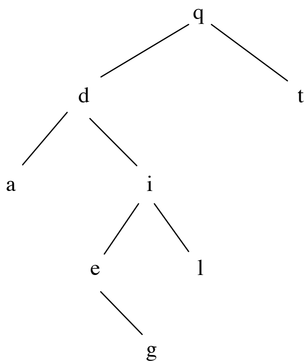


Exemple de coupure

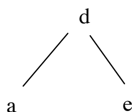


Exemple de coupure

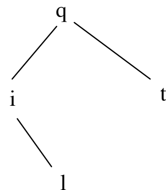
f



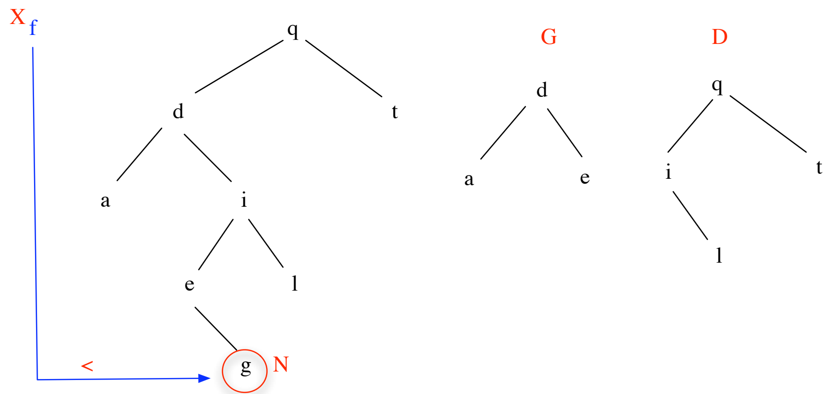
G



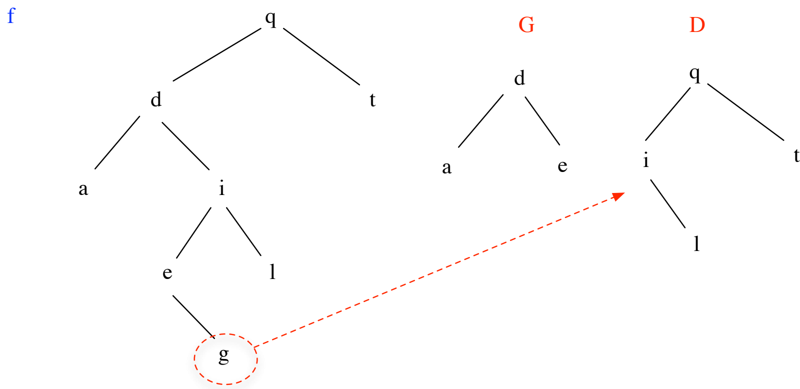
D



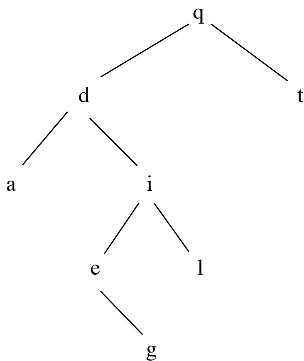
Exemple de coupure

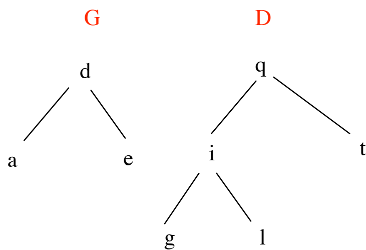


Exemple de coupure



Exemple de coupure





Procédure de coupure

Coupure de A en 2 ARBIN G et D

```
Coupure(ELEMENT X, ARBIN A, G , D )
Début
   SI A est vide ALORS
      G <-- vide
      D <-- vide
   SINON
      SI X <= A ALORS
         D <-- A
         Coupure( X, ABG(A), G, ABG(D))
      SINON
         G <-- A
         Coupure( X, ABD(A), ABD(G), D)
      FSI
   FSI
Fin
```

Procédure d'ajout à la racine

Ajout d'un élément X à la racine de l'arbre A

```
ArbinAjouterRacine(ELEMENT X, ARBIN A)
ARBIN R
Début
    étiquette de R <-- X
    ABG(R) <-- vide
    ABD(R) <-- vide
    Coupure(X, A, ABG(R), ABD(R))
    A <-- R
Fin
```

Plan

6 Suppression d’un élément

Méthodes de suppression

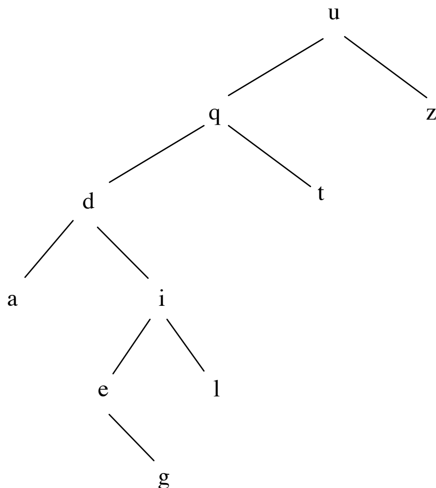
Pour supprimer un élément X dans A il faut

1. déterminer la place de X dans $A \longrightarrow$ noeud N
2. supprimer X avec réorganisation des éléments de A.
   3 cas :

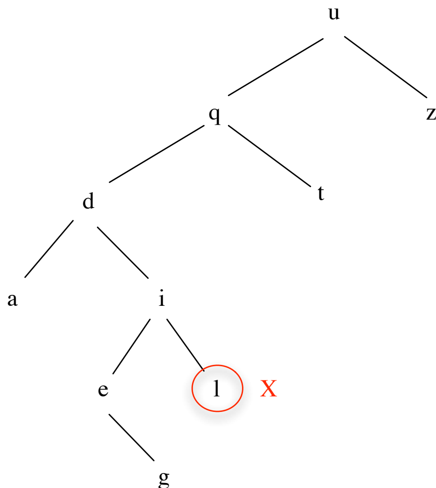
   N à 0 fils : suppression immédiate
   N à 1 fils : on remplace X par ce fils
   N à 2 fils : 2 solutions

   - remplacer X par l'élément qui lui est immédiatement inférieur
     $\longrightarrow$ Le MAX dans ABG(N)
   - remplacer X par l'élément qui lui est immédiatement supérieur
     $\longrightarrow$ Le MIN dans ABD(N)

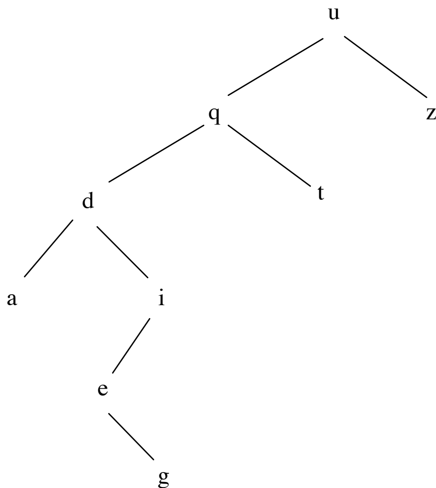
Suppression d'un noeud à 0 fils



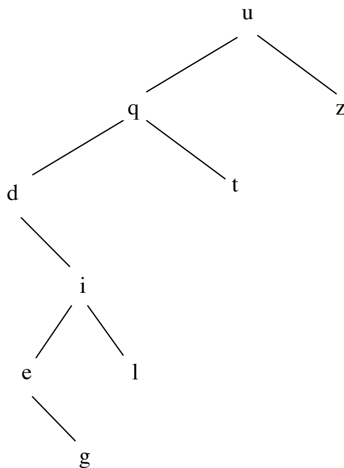
Suppression d'un noeud à 0 fils



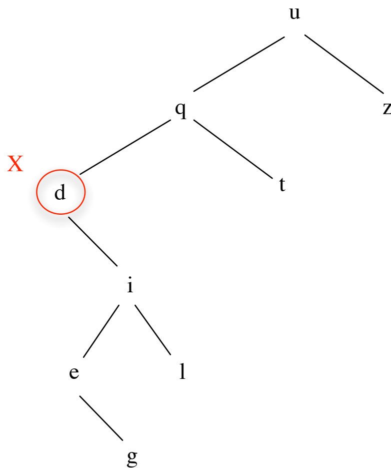
Suppression d'un noeud à 0 fils



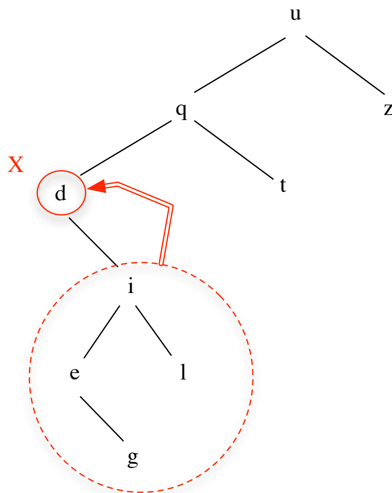
Suppression d'un noeud à 1 fils



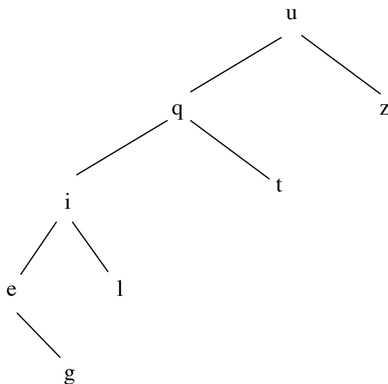
Suppression d'un noeud à 1 fils



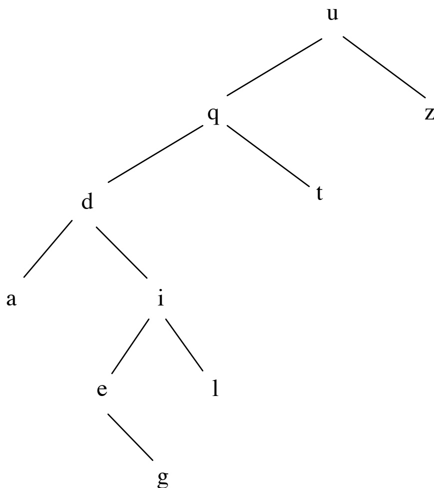
Suppression d'un noeud à 1 fils



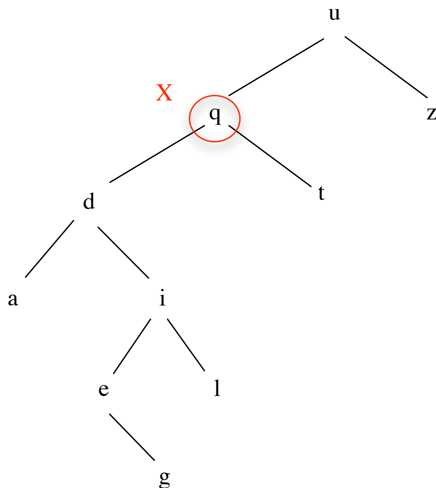
Suppression d'un noeud à 1 fils



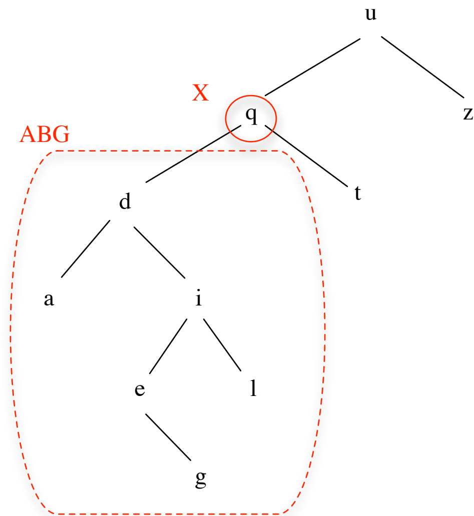
Suppression d'un noeud à 2 fils



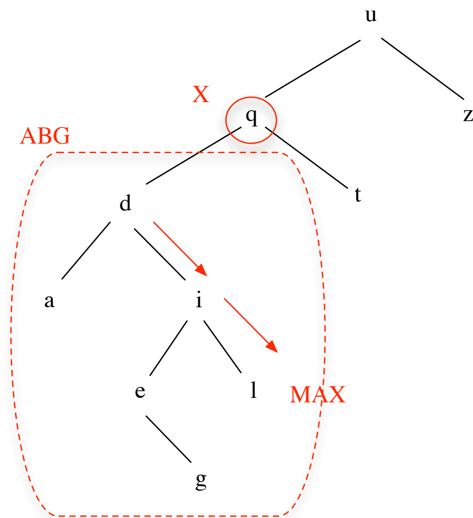
Suppression d'un noeud à 2 fils



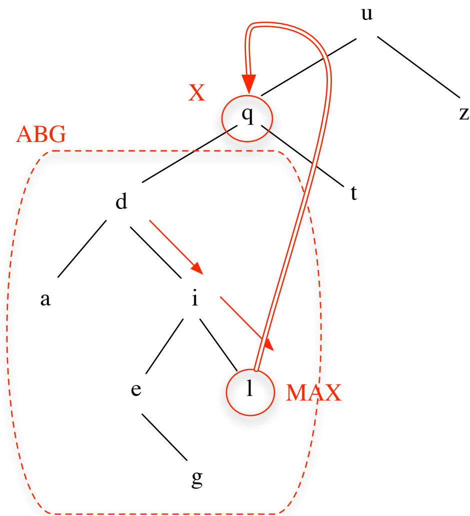
Suppression d'un noeud à 2 fils



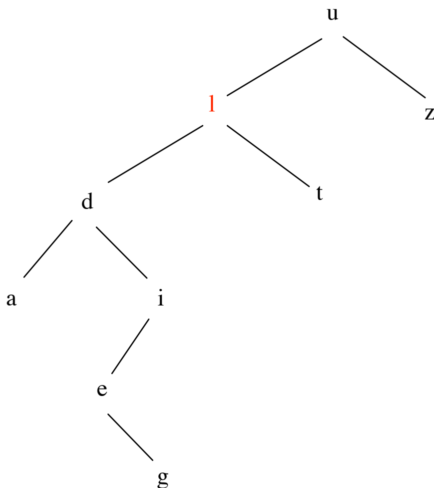
Suppression d'un noeud à 2 fils



Suppression d'un noeud à 2 fils



Suppression d'un noeud à 2 fils



Suppression du MAX

(Le Max dans un ARBIN est l'élément le plus à droite)

Retourne l'élément le plus grand dans MAX et ampute cet élément de A

```
SupprimerMax(ELEMENT MAX, ARBIN A)
Début
    SI ABD(A) est vide ALORS
        MAX <—— étiquette de A
        A <—— ABG(A)
    SINON
        SupprimerMax( MAX, ABD(A) )
    FSI
Fin
```

Suppression d'un élément

Supprime l'élément X dans A. Retour : A si $X \notin A$ sinon $A - X$

```
ArbinSupprimer(ELEMENT X, ARBIN A)
Début
  SI A n'est pas vide ALORS
    SI X < étiquette de A ALORS ArbinSupprimer(X, ABG(A))
    SINON SI X > étiquette de A ALORS ArbinSupprimer(X, ABD(A))
        SINON /* On a trouvé l'élément */
          SI ABG(A) est vide ALORS /* 0 ou 1 fils */
            A <-- ABD(A)
          SINON
            SI ABD(A) est vide ALORS /* 1 fils */
              A <-- ABG(A)
            SINON /* 2 fils */
              SupprimerMax( MAX, ABG(A) )
              étiquette de A <-- MAX
            FSI
          FSI
        FSI
    FSI
  FSI
Fin
```

Plan

7 Conclusion sur les arbres

Propriétés des arbres binaires

Propiétés :

- Un arbre binaire ayant n sommets a une hauteur h qui vérifie : $[log_2(n+1)] - 1 \leq h(a) \leq n - 1$
- Un arbre binaire de hauteur h a un nombre de sommets n qui vérifie : $h + 1 \leq n \leq 2^{h+1} - 1$

Utilisation : mesures de complexité

- Parcours de chaque noeud de l'arbre en $O(n)$
- Parcours d'une branche : complexité dans le pire des cas en $O(h) = O(log(n))$

Objectif :
$\rightarrow$ avoir h minimum (arbre complet ou presque ou équilibré)

The End

That all folks. . .