

## Séance de TD 10

### 1 Exercice 1 : Recherche dans un TB

Donnez l'algorithme de recherche d'un minimum dans un tas binomial.

Rappel : un tas binomial est un ensemble d'arbres binaires.

La complexité de l'algorithme de recherche d'un minimum dans un tas binomial est de l'ordre de  $O(\log(n))$ , avec  $n$  le nombre de noeuds,  $\log(n)$  étant le nombre d'arbres composant le tas.

Le texte de l'algorithme est donné ci-après :

```
Minimum_T_Binomial(T)
  y <- Nil
  x <- tete(T)
  min <- infini
  Tant que x != Nil Faire
    Si Cle(x) < min Alors
      min <- cle(x)
      y <- x
    Finsi
  x <- frere(x)
  Fintantque
  retourner y
Fin
```

### 2 Exercice 2 : Union de 2 TB

#### 2.1 Introduction

**Principe :** Soit  $T_1$  et  $T_2$  2 TB à fusionner.

##### Etape 0

1. Si aucun  $T_1$  et  $T_2$  ne contient  $B_0$ , alors ne rien faire.
2. Si seul  $T_1$  ou  $T_2$  contient  $B_0$ , alors le mettre dans l'arbre  $T$  résultat.
3. Si  $T_1$  et  $T_2$  contiennent tous les deux  $B_0$ , alors
4. On les lie entre eux pour former  $B_1$ . Ensuite sauvegarder ce tas pour l'étape suivante.

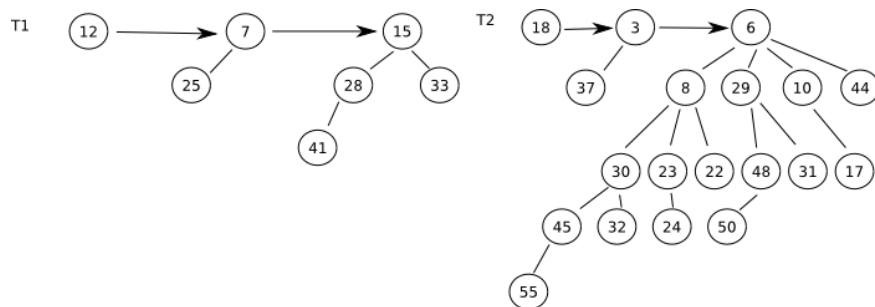
**Etape i, ( $i \in [1; \log(n)]$ )**

Il peut y avoir de 0 à 3 TB, un de  $T_1$ , un de  $T_2$ , et un autre de l'étape précédente.

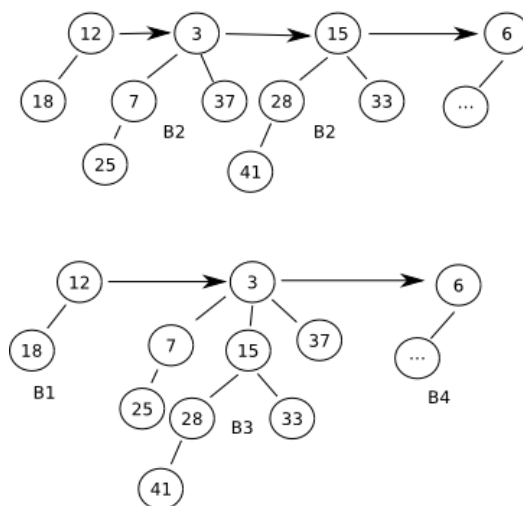
1. S'il n'y a pas  $B_1$ , ne rien faire
2. S'il n'y a qu'un seul  $B_1$ , le mettre dans le tas résultat.
3. Sinon, lier les 2  $B_i$  pour avoir  $B_{i+1}$  et le sauvegarder pour l'étape suivante.
4. S'il reste encore 1  $B_i$ , le mettre dans T résultat. S'il reste à la fin un  $B_i$  de sauvegarde de la dernière étape, le mettre dans T résultat.

## 2.2 Application

On va appliquer l'algorithme de fusion aux deux TB ci-dessous :



Ce qui donne :



Après fusion des  $B_k$ .

### 2.3 Code de l'algorithme de fusion

```
FusionnerTasBinomiaux(T1, T2)
  T <- créer TasBinomial()
  tete(T) <- FusionnerTasBinomiaux(T1, T2)
  Libérer les objets T1 et T2
  Si tete(T) = Nil Alors
    retourner T
  Finsi
  avant(x) <- Nil
  x <- tete(T)
  apres(x) <- frere(x)
  Tant que apres(x) != Nil Faire
    Si degre(x) != degre(apres(x)) ou frere(apres(x)) != Nil
      et degre(frere(apres(x))) = degre(x) Alors
        avant(x) <- x
        x <- apres(x)
    Sinon
      Si cle(x) <= cle(apres(x)) Alors
        frere(x) <- frere(apres(x))
        LienBinomial(apres(x), x)
      Sinon
        si avant(x) != Nil Alors
          tete(t) <- apres(x)
        Finsi
      LienBinomial(x, apres(x))
      x <- apres(x)
    Finsi
  Finsi
  apres(x) <- frere(x)
  Fintantque
retourner T
Fin
```

### 2.4 Coût de l'algorithme

La fusion est d'ordre  $O(1)$ , alors que l'organisation de T est d'ordre  $O(\log(n))$ .

La complexité de l'algorithme est donc en  $O(\log(n))$ .

### 3 Exercice 3 : Tas de Fibonacci

1. Donnez l'algorithme permettant de diminuer une clé dans un tas de Fibonacci.
2. Donnez le coût de l'algorithme.

Principe de l'algorithme de T.F. pour diminuer une clé. Plusieurs cas se présentent

**Cas 1 :** Pas de violation de la propriété d'infériorité des pères. Il faut modifier la clé et mettre à jour le minimum.

**Cas 2 :** Violation de la propriété, avec un père non marqué.

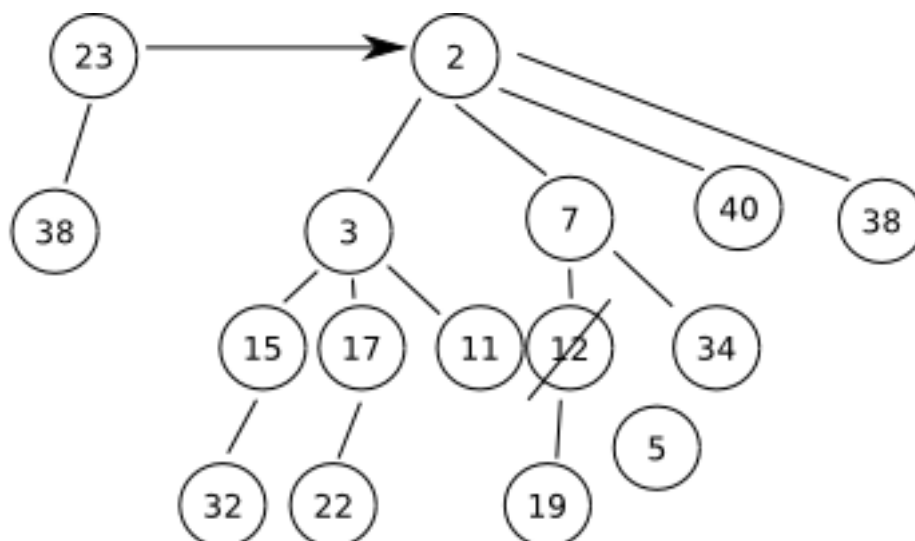
- Modifier la clé
- Marquer le père
- Insérer le sous-arbre dans la liste des racines
- Mettre à jour le minimum

**Cas 3 :** Violation de la propriété, avec un père marqué.

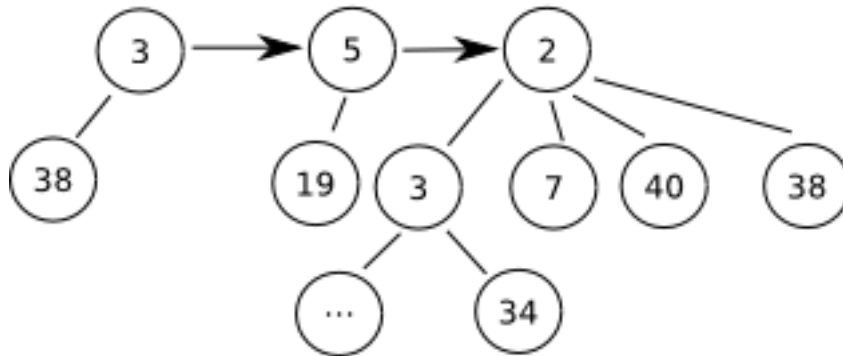
- Modifier la clé
- Insérer le sous-arbre dans la liste des racines
- Mettre à jour le min.
- Tant que le père est marqué, insérer le sous-arbre dans la liste des racines.
- Marquer le premier ascendant non-marqué

#### 3.1 Application

Diminuons la clé 12 à 5 dans le graphe suivant :



Cela donne :



### 3.2 Algorithme

DiminuerCleTF(T,a,k)

Si  $k > \text{cle}(x)$  Alors

    erreur << nouvelle clé plus grande que la clé courante >>

Finsi

$\text{cle}(x) \leftarrow k$

$y \leftarrow p(x)$

Si  $(y \neq \text{Nil})$  et  $(\text{cle}(x) < \text{cle}(y))$  Alors

    couper(T,x,y)

    couperEnCascade(T,y)

Finsi

Si  $\text{cle}(x) < \text{cle}(\min(T))$  Alors

$\min(T) \leftarrow x$

Finsi

Fin

Détails de la fonction  $\text{couper}(T, x, y)$  :

1. Supprimer  $x$  de la liste des enfants de  $y$  en décrémentant  $\text{degré}(y)$
2. Ajouter  $x$  à la liste des racines de  $T$
3.  $P(x) \leftarrow \text{nil}$
4.  $\text{Marque}(x) \leftarrow \text{faux}$

Code de la fonction  $\text{CouperEnCascade}(T, y)$  :

$z \leftarrow P(y)$

```
Si z != Nil Alors
  Si marque(z) = Faux Alors
    marque(y) <- vrai
  Sinon
    couper(T,y,z)
    CouperEnCascade(T,z)
  Finsi
Finsi
```

### 3.3 Coût

La modification de la clé est en  $O(1)$ , puis le coût est d'ordre  $O(1)$  pour chaque ascendant marqué.

Le coût global est donc  $O(1)$ .