Algorithmique Avancée

Michèle Soria

Michele.Soria@lip6.fr

Master Informatique M1-STL

http://www-master.ufr-info-p6.jussieu.fr/2009
http://www-master.ufr-info-p6.jussieu.fr/2009/algav

Année 2009-2010

Organisation

- **L'équipe pédagogique** :
  - **Cours** : Michèle Soria (me 10h45)
  - **TD-TME** : G1 (lu 13h30) : Maryse Pelletier
    G2 (ma 13h30) : Philippe Trébuchet
    G3 (me 13h30) : Philippe Aubry
- **L'organisation** :
  - 10 semaines (TDs décalés d'une semaine vs cours)
  - Session 1
    - ecrit réparti 1 (E1) : semaine du 9 Novembre
    - fin des enseignements le 5 décembre
    - ecrit réparti 2 (E2) : semaine du 14 décembre
    - note Session 1 = 0.2 E1 + 0.2 TP + 0.6 E2
  - Session 2
    - examen Session 2 (SS) : en janvier
    - note Session 2 = SS

Plan du cours

Objectifs : Complexité des algorithmes → Comparer, Optimiser

- **Structures de données avancées**
  - Files de priorités : Files Binomiales et Fibonacci
  - Recherche : Arbres équilibrés et méthodes de Hachage
  - coût amorti et coût moyen
- **Algorithmique géométrique**
  - Enveloppe convexe, statique et dynamique
  - Distances minimales
  - Analyse et implantation
- **Algorithmes de compression**
  - Compression statistique et compression par dictionnaires
  - Compression par dictionnaires
  - Applications

Bibliographie

- T. Cormen, C. Leiserson, R. Rivest, C. Stein
  Introduction à l'algorithmique,
- C. Froidevaux, M-C. Gaudel, M. Soria
  Types de données et algorithmes
- D. Beauquier, J. Berstel, P. Chrétienne
  Eléments d'algorithmique
- D. Salomon
  Data Compression : The Complete Reference

Plan Cours 1-2

CHAPITRE 0 : Introduction, Complexité des algorithmes

- Classification de problèmes
- Analyse d'algorithmes
- Cas pire, cas moyen, coût amorti

CHAPITRE 1 : Files de Priorités : binomiales et Fibonacci

- Opérations sur les files de priorités
- Arbres binomiaux : définition et propriétés
- Files binomiales : définition et propriétés
- Union de 2 files binomiales en temps logarithmique
- Autres opérations sur les files binomiales
- Analyse en Coût amorti

Complexité des algorithmes

- Théorie de la complexité et classification de problèmes
- Problèmes pôlynomiaux : tri, recherche, géométrie, arithmétique, ...
- Analyse des *algorithmes*
- Coût (temps, espace) fonction de la taille des données
- Cas pire, cas moyen, coût amorti
- Ordre de grandeur

Opérations sur les files de priorités

- **Ensemble d'éléments**
  - Chaque élément a une clé
  - Ordre total sur les clés
- **Opérations**
  - Ajouter un élément
  - Supprimer l'élément de clé minimale
  - Union de 2 files de priorités
  - *Construction*
  - *Modification d'une clé*

Représentations et Efficacité

Nombre de comparaisons dans le pire des cas

	Liste triée	Tas (Heap)	File Binomiale
Supp Min (n)	$O(1)$	$O(\log n)$	$O(\log n)$
Ajout (n)	$O(n)$	$O(\log n)$	$O(\log n)$
Construction (n)	$O(n^2)$	$O(n)$	$O(n)$
Union (n, m)	$O(n+m)$	$O(n+m)$	$O(\log(n+m))$

Applications des Files de priorité

- Tri heapsort
- Sur les graphes
  - plus court chemin à partir d'une source (Dijkstra)
  - plus court chemin entre tous les couples de sommets (Johnson)
  - arbre couvrant minimal (Prim)
- Interclassement de listes triées
- Code de Huffmann (compression)

Arbre binomial- Définition

Un arbre binomial pour chaque entier positif.

Définition par récurrence

- B_0 est l'arbre réduit à un seul nœud,
- Étant donnés 2 arbres binomiaux B_k, on obtient B_{k+1} en faisant de l'un des B_k le premier fils à la racine de l'autre B_k.

Exemples : dessiner B_0, B_1, B_2, B_3, B_4

Arbre binomial - Propriétés

Propriétés de B_k, ($k \geq 0$)

1. B_k a 2^k nœuds
2. B_k a $2^k - 1$ arêtes
3. B_k a hauteur k
4. Le degré à la racine est k
5. Le nombre de nœuds à profondeur i est $\binom{k}{i}$
6. La forêt à la racine de B_k est $< B_{k-1}, B_{k-2}, \ldots, B_1, B_0 >$

Arbre binomial - Preuves

1. $n_0 = 1$ et $n_k = 2n_{k-1}$
2. arbre : x nœuds $\Rightarrow x - 1$ arêtes
3. $h_0 = 0$ et $h_k = 1 + h_{k-1}$
4. $d_0 = 0$ et $d_k = 1 + d_{k-1}$
5. $n_{k,0} = 1$, $n_{k,l} = 0$ pour $l > k$, et $n_{k,i} = n_{k-1,i} + n_{k-1,i-1}$, pour $i = 1, \ldots, k$
6. propriété de décomposition, par récurrence sur k

File Binomiale

Tournoi Binomial

Un *tournoi binomial* est un arbre binomial étiqueté croissant (croissance sur tout chemin de la racine aux feuilles)

File Binomiale

Une *file binomiale* est une suite de tournois binomiaux de tailles strictement décroissantes

Exemples :

- $FB_{12} =< TB_3, TB_2 >$,
- $FB_7 =< TB_2, TB_1, TB_0 >$

Représentation d'une file de priorité

Représentation d'une file de priorité $\mathcal{P}$ de n éléments

- si $n = 2^k$, $\mathcal{P}$ tournoi binomial
- sinon $\mathcal{P}$ file binomiale, suite de tournois correspondants aux bits égaux à 1 dans la représentation binaire de n.

Représentation binaire de n

$$n = \sum_{i=0}^{\lfloor \log_2 n \rfloor} b_i 2^i, \qquad \text{avec} \qquad b_i \in \{0,1\},\ b_{\lfloor \log_2 n \rfloor} = 1$$

$\nu(n) = \sum_i b_i$: # bits à 1 dans représentation binaire de n.

File binomiale - Propriétes

Propriétés de FB_n

1. FB_n a n nœuds
2. FB_n a $n - \nu(n)$ arêtes
3. Le plus grand arbre de la file est $B_{\lfloor \log_2 n \rfloor}$ (hauteur $\lfloor \log_2 n \rfloor$ et nombre de nœuds $2^{\lfloor \log_2 n \rfloor}$)
4. Le nombre d'arbres de la file est $\nu(n)$ (avec $\nu(n) \leq 1 + \lfloor \log_2 n \rfloor$)
5. Le minimum de la file est à la racine de l'un des arbres

1. $n = \sum b_i 2^i$,
2. $n - \nu(n) = \sum b_i (2^i - 1)$,
3. $\nu(n) = \sum_i b_i$

Union de files binomiales

1. **Union de 2 tournois de tailles différentes** :
   $TB_{k1} \cup TB_{k2} \longrightarrow F_{2^{k1}+2^{k2}} =< TB_{k1}, TB_{k2} >$
   Exemple : $TB_1 \cup TB_2$
2. **Union de 2 tournois de même taille** :
   $TB_k \cup TB_{k'} \longrightarrow TB_{k+1}$,
   avec $rac(TB_{k+1}) = min(rac(TB_k), (rac(TB_{k'})))$
   Exemple : $TB_2 \cup TB'_2$
3. **Union de 2 files binomiales** $\equiv\equiv$ addition binaire
   Exemple : $FB_5 \cup FB_7$

Union de deux files

1. interclasser les 2 files en partant des tournois de degré minimum
2. lorsque 2 tournois de taille k on engendre un tournoi de taille $k+1$
3. à chaque étape au plus 3 tournois de même taille à fusionner (1 dans chacune des files + 1 retenue de la fusion à l'étape précédente)
4. lorsque 3 tournois de taille k on en retient 2 pour engendrer un tournoi de taille $k+1$, et l'on garde le troisième comme tournoi de taille k.

Primitives sur les tournois binomiaux

EstVide : TournoiB → booleen
renvoie *vrai ssi le tournoi est vide*

Degre : TournoiB → entier
renvoie *le dégré du tournoi*

UTid : TournoiB * TournoiB → TournoiB
renvoie *l'union de 2 tournois de même taille* $T_k * T_k \mapsto T_{k+1}$

Decapite : TournoiB → FileB
renvoie *la file binomiale obtenue en enlèvant la racine du tournoi* $T_k \mapsto < T_{k-1}, T_{k-2}, ..., T_1, T_0 >$

Primitives sur les files binomiales

EstVide : FileB → booleen
renvoie *vrai ssi la file est vide*

MinDeg : FileB → TournoiB
renvoie *le tournoi de degré minimal de la file*

Reste : FileB → FileB
renvoie *la file privée de son tournoi de degré minimal*

AjoutMin : TournoiB * FileB → FileB
hypothèse *: le tournoi est de degré inférieur au MinDeg de la file*
renvoie *la file obtenue en ajoutant le tournoi comme tournoi de degré minimal de la file initiale*

Algorithme d'Union

*UnionFile : FileB * FileB ⟶ FileB*
renvoie *la file binomiale union des deux files F1 et F2*
Fonction UnionFile(F1, F2)
Retourne UFret(F1, F2, Ø)
FinFonction *UnionFile*

*UFret : FileB * FileB * TournoiB ⟶ FileB*
renvoie *la file binomiale union de deux files et d'un tournoi*
Fonction UFret(F1, F2, T)

```
Fonction UFret(F1, F2, T)
  Si EstVide(T) ; pas de tournoi en retenue
    Si EstVide(F1) Retourne F2
    Si EstVide(F2) Retourne F1
    Soient T1=MinDeg(F1) et T2=MinDeg(F2) ;tourn deg min
      Si Degre(T1)<Degre(T2)
        Retourne AjoutMin(T1,UnionFile(reste(F1),F2))
      Si Degre(T2)<Degre(T1)
        Retourne AjoutMin(T2,UnionFile(F1,reste(F2)))
      Si Degre(T2)=Degre(T1)
        Retourne UFret(reste(F1),reste(F2),UTid(T1,T2))
```

⟶


```
  Sinon ; un tournoi en retenue
    Si EstVide(F1) Retourne UnionFile(File(T), F2)
    Si EstVide(F2) Retourne UnionFile(File(T), F1)
    Soient T1=MinDeg(F1) Et T2=MinDeg(F2)
      Si Degre(T)<Degre(T1) Et Degre(T)<Degre(T2)
        Retourne AjoutMin (T,UnionFile(F1,F2)))
      Si Degre(T)=Degre(T1)=Degre(T2)
        Retourne
          AjoutMin(T,UFret(reste(F1), reste(F2), UTid(T1,T2)))
      Si Degre(T)=Degre(T1) ; et < Degre(T2)
        Retourne UFret(reste(F1),F2,UTid(T1,T))
      Si Degre(T)=Degre(T2) ; et < Degre(T1)
        Retourne UFret(F1,reste(F2),UTid(T2,T))
FinFonction UFret
```


Analyse de complexité

Union de 2 files binomiales FB_n **et** FB_m **en** $O(\log_2(n+m))$

- Hypothèse :
  toutes les primitives ont une complexité en $O(1)$
- Critère de complexité : *nombre de comparaisons entre clés*
- Complexité dans le *pire des cas*
- idée
  1 union de 2 tournois de même taille ⟶ 1 comparaison entre clés et ajoute une arête dans la file résultat.
- Conséquence : nombre de comparaisons pour faire l'union de 2 files égale nombre d'arêtes de la file union diminué du nombre d'arêtes des files de départ

Calcul

Nombre de comparaisons pour faire l'union d'une file binomiale de n éléments et d'une file binomiale de m éléments.

$$\begin{aligned} \#cp(FB_n \cup FB_m) &= n+m-\nu(n+m)-(n-\nu(n))-(m-\nu(m)) \\ &= \nu(n)+\nu(m)-\nu(n+m) \\ &< \lfloor \log_2 n \rfloor + 1 + \lfloor \log_2 m \rfloor + 1 \\ &= O(\log_2(n+m)) \end{aligned}$$

Exemples :

- $FB_{21} \cup FB_{11}$
- $FB_{21} \cup FB_{10}$

Ajout d'un élément x à une file FB_n

Algorithme :
Créer une file binomiale FB_1 contenant uniquement x.
Puis faire l'union de FB_1 et FB_n

Complexité : $\nu(n) + 1 - \nu(n+1) \longrightarrow$ entre 0 et $\nu(n)$

Exemples :

- $FB_1 \cup FB_8$
- $FB_1 \cup FB_7$

Construction

Complexité de la construction d'une file binomiale par **adjonctions successives** de ses n éléments.

$$\begin{aligned} \#cp(FB_n) &= \nu(n-1) + 1 - \nu(n) \\ &+ \nu(n-2) + 1 - \nu(n-1) \\ &+ \dots \\ &+ \nu(1) + 1 - \nu(2) \\ &= n - \nu(n) \end{aligned}$$

Donc le nombre moyen de comparaisons pour 1 ajout est $1 - \nu(n)/n < 1$.
Coût amorti d'une opération dans une série d'opérations : $couttotal/nbreop$

Suppression du minimum de FB_n

Recherche du minimum
Le minimum de la file est à la racine d'un des tournois la composant
$\longrightarrow \nu(n) - 1$ comparaisons = $O(\log n)$

Suppression du minimum

- Déterminer l'arbre B_k de racine minimale
- Supprimer la racine de $B_k \longrightarrow$ File $< B_{k-1}, \dots, B_0 >$
- Faire l'union des files $FB_n - B_k$ et $< B_{k-1}, \dots, B_0 >$

Complexité : $O(\log n)$

Diminuer une clé

N.B. **Accès direct au nœud dont il faut diminuer la clé**

- modifier la clé
- échanger le nœud avec son père jusqu'à vérifier l'hypothèse de croissance ($\equiv$ tas)

Le nombre maximum de comparaisons est la hauteur de l'arbre ($O(\log n)$)

Coût amorti

Files Binomiales :

- ajout d'un élément et recherche du minimum en $O(1)$
- suppression du minimum et union de 2 files en $O(\log n)$

Files de Fibonacci :

- ajout d'un élément et union de 2 files en $O(1)$
- suppression du minimum en $O(\log n)$

Remarque : on ne peut pas espérer avoir $O(1)$ pour ajout et suppression du minimum, car alors on serait en contradiction avec les résultats de borne inférieure en $O(n \log n)$ pour le tri par comparaisons.

Coût amorti

Définition

- *Coût amorti* d'une opération dans une suite d'opérations = coût moyen d'une opération dans le pire cas.
- coût amorti = coût total / nombre d'opérations

Méthodes

- méthode par agrégat
- méthode du potentiel
- autres...

Coût amorti : méthode par agrégat

- **Principe :** majorer le coût total d'une suite de n opérations et diviser par n.
- **Exemple :** opérations sur les piles
  - `empiler(S,x)`
  - `dépiler(S)`
  - `multidépiler(S,k)`

  Coût amorti de chaque opération en $O(1)$.

Coût amorti : méthode du potentiel

- **Principe :**
  - structure de données D_i,
  - fonction *potentiel* Φ vérifiant $\Phi(D_i) \geq \Phi(D_0)$
  - *coût amorti* de la i-ème opération : $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$ (c_i coût réel de la i-ème opération)
  - coût amorti = $\sum_{i=1}^{n} \hat{c}_i / n$
- **Exemple :** opérations sur les piles
  - $\Phi(D_i) =$ nombre d'objets de D_i
  - coût amorti de chaque opération en $O(1)$