Algorithmique Avancée

TD 2 : Structures de données avancées

1 Coût amorti

Dans l'analyse de *structure de données*, on s'intéresse à des *séquences* d'opérations. Étudier le coût pire-cas séparé de chacune des opérations est souvent trop pessimiste, car cela ne tient pas compte des répercussions que chaque opération a sur la structure de données. On s'intéresse donc au coût amorti d'une suite de n opérations, qui est *le coût moyen d'une opération dans le pire des cas*, et l'on se sert pour cela de deux méthodes :

- **la méthode par agrégat** consiste à *majorer* le coût de toute suite de n opérations effectuées sur la structure, et à diviser le coût total ainsi obtenu par le nombre n d'opérations ;
- **la méthode du potentiel** consiste à introduire une fonction $\phi(D_i)$ qui associe à l'état D_i de la structure de données après la i-ième opération un nombre réel ; le coût amorti $\hat{c}_i$ de la i-ième opération est alors le coût réel c_i, moins la différence de potentiel dûe à l'opération

$$\hat{c}_i = c_i + \phi\left(D_i\right) - \phi\left(D_{i-1}\right) ; \tag{1}$$

ainsi (puisque les $\phi(D_i)$ s'annulent deux à deux dans la somme), le coût amorti total est

$$\sum_{i=1}^{n} \hat{c}_i = \left(\sum_{i=1}^{n} c_i\right) + \phi(D_n) - \phi(D_0). \tag{2}$$

Exercice 1.1 ***: Coût amorti (piles avec multi-dépilement)***

On considère les deux opérations fondamentales sur les piles, `Empiler(S,x)` et `Dépiler(S)`, auxquelles on ajoute l'opération `MultiDépiler(S,k)` qui retire les k premiers objets du sommet de la pile si la pile contient au moins k objets et vide la pile sinon.

Question 1.1.1 Calculer le coût amorti d'une opération en utilisant la méthode par agrégat.

Question 1.1.2 Même question en utilisant la méthode du potentiel.

Exercice 1.2 ***: Coût amorti (piles avec multi-dépilement et multi-empilement)***

Aux trois opérations précédentes, `Empiler(S,x)`, `Dépiler(S)` et `MultiDépiler(S,k)`, définies sur les piles, on ajoute l'opération `MultiEmpiler(S,`x_1`,`x_1`,`x_2`,...,`x_k`)`, qui permet d'empiler les objets $x_1, x_2, \ldots, x_k$ sur la pile. On distingue deux implantations différentes de cette pile qui induisent chacune une restriction supplémentaire sur l'opération `MultiEmpiler` :

(a) le nombre d'objets que l'on peut empiler en une opération est borné par une constante c ;

(b) le nombre d'objets que l'on peut empiler à chaque fois est inférieur ou égal au nombre d'objets de la pile S, noté $|S|$.

Il s'agit de calculer le coût amorti d'une opération dans le cas (a), puis dans le cas (b).

Question 1.2.1 En utilisant la méthode par agrégat.

Question 1.2.2 En utilisant la méthode du potentiel.

Exercice* 1.3 *: Coût amorti (tableaux dynamiques)

Une séquence de n opérations est effectuée sur une structure de données. La i-ème opération coûte i si i est une puissance de 2 et 1 sinon.

Question 1.3.1 En utilisant la méthode par agrégat, montrer que le coût amorti par opération est en $O(1)$.

Question 1.3.2 Méthode du potentiel.

Idée : le potentiel croît pour chaque opération peu coûteuse (*i.e.* lorsque i n'est pas une puissance de 2) et retombe à 0 pour chaque opération coûteuse (*i.e.* lorsque i est une puissance de 2).

(a) *Essai 1.* Le potentiel croît de 1 pour chaque opération peu coûteuse. Autrement dit :

$$\phi(i) = \begin{cases} 0 & \text{si } i = 0 \\ \phi(i-1)+1 & \text{si } i \text{ n'est pas une puissance de 2} \\ 0 & \text{si } i \text{ est une puissance de 2} \end{cases}$$

Calculer le coût amorti en utilisant cette fonction de potentiel. Le résultat est-il satisfaisant ?

(b) *Essai 2.* Le potentiel croît de 2 pour chaque opération peu coûteuse. Autrement dit :

$$\phi(i) = \begin{cases} 0 & \text{si } i = 0 \\ \phi(i-1)+2 & \text{si } i \text{ n'est pas une puissance de 2} \\ 0 & \text{si } i \text{ est une puissance de 2} \end{cases}$$

Calculer le coût amorti en utilisant cette fonction de potentiel. Le résultat est-il satisfaisant ?

Question 1.3.3 Donner un exemple d'une suite d'opérations ayant comme coût le coût décrit dans cet exercice (préciser la structure de données).

Chercher éventuellement un exemple de coût moyen facile a calculer puis faire un coût amorti qu'on espère différent.

Exercice* 1.4 *: Coût amorti (incrémentation d'un compteur)

Etant donnés des entiers positifs n et k tels que $n < 2^k$, on dispose d'un tableau $A[0..k-1]$ ne contenant que des 0 et des 1 pour représenter l'écriture en binaire des entiers positifs inférieurs à 2^k (le bit de poids le plus faible est stocké dans $A[0]$). On définit l'opération `incrementer(A)` comme suit :

```
Incrementer(A)
  i <- 0 ;
  tantque i < k et A[i] = 1 faire
    A[i] <- 0
    i <- i+1
  fintantque
  si i < k alors
    A[i] <- 1
  finsi
fin
```

Question 1.4.1 Quel est l'effet de `Incrementer` sur le tableau A ?

Question 1.4.2 Le coût réel de l'opération `Incrementer` est le nombre de bits qui changent. On considère une suite de n incrémentations à partir de 0. Calculer le coût amorti de l'opération `Incrementer` :

(a) en utilisant la méthode par agrégat ;

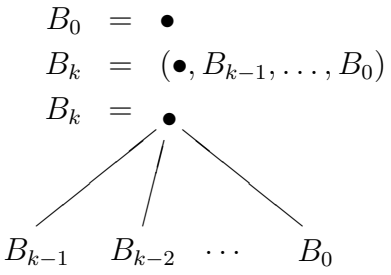
(b) en utilisant la méthode du potentiel (une fois la fonction de potentiel choisie, il est conseillé de faire un tableau " *coût réel/variation de potentiel/coût amorti* " pour les premières incrémentations d'un compteur à 4 bits).

2 Files binomiales

Exercice 2.1 ***: Définitions et propriétés des arbres binomiaux***
On rappelle deux définitions équivalentes des *arbres binomiaux*, introduits dans le premier cours.

Définition 1.
- B_0 est l'arbre réduit à un seul nœud ;
- étant donnés deux arbres binomiaux B_{k-1}, on obtient B_k en faisant de l'un des B_{k-1} le premier enfant à la racine de l'autre B_{k-1}.

Définition 2.
- B_0 est l'arbre réduit à un seul nœud ;
- B_k est l'arbre dont la racine à k enfants : B_{k-1}, ..., B_0.



Question 2.1.1 Prouver d'abord que ces deux définitions sont équivalentes. (noter $\mathcal{D}_i$ l'ensemble défini inductivement par la définition i et montrer que $\mathcal{D}_1 \subset \mathcal{D}_2$ et $\mathcal{D}_2 \subset \mathcal{D}_1$).

Question 2.1.2 Montrer ensuite chacune de ces propriétés portant sur les arbres binomiaux en choisissant à chaque fois la définition qui semble la plus pratique :

(a) B_k a 2^k nœuds ;

(b) la hauteur de B_k est k ;

(c) il y a $\binom{k}{i}$ nœuds à la profondeur i $(i = 0, \ldots, k)$;

(d) la racine de B_k est de degré k, supérieur au degré de n'importe quel autre nœud de B_k.

Conclusion. Comme la propriété (d) remarque que le degré (à la racine) de B_k est k, la propriété (a) permet de conclure que les arbres binomiaux ont *une taille exponentielle en leur degré* ; les propriétés (a) et (b) ensemble montrent que les arbres binomiaux ont *une hauteur logarithmique en leur taille.* Ces deux constatations expriment le caractère "bien équilibré" de ces arbres.

Exercice 2.2 ***: Files binomiales relâchées***

Par définition, une file binomiale est composée de tournois binomiaux de tailles toutes différentes. À cause de cette contrainte sur les tailles, l'union de deux files binomiales de tailles respectives n et m a une complexité en $O(\log(n+m))$ et l'insertion d'un nouvel élément dans une file binomiale de taille n a une complexité en $O(\log(n))$. On souhaite maintenant avoir une structure plus souple dans laquelle on pourra réaliser l'union et l'insertion en $O(1)$, en reportant les opérations coûteuses sur la suppression du minimum. Cela est possible en supprimant la contrainte sur les tailles.

Question 2.2.1 Rappeler les tailles des arbres binomiaux qui forment une file binomiale de taille n, et les degrés des différentes racines. Quelle est la composition d'une file binomiale de taille 143 ?

Question 2.2.2 On pose $D(n) = \lfloor \log_2(n) \rfloor$.
Montrer que le degré maximum d'un nœud d'une file binomiale de taille n est $D(n)$.

On appelle *file binomiale relâchée* une suite de tournois binomiaux de tailles quelconques.
Un exemple de file binomiale relâchée : $(B_3, B_1, B_2, B_1, B_1, B_2, B_4, B_0, B_0, B_0)$.

Question 2.2.3 Montrer que le degré d'un nœud d'une file binomiale relâchée est au plus $D(n)$.

Étant donnés une file binomiale relâchée H et un nœud x quelconque de H :

- $\min(H)$ désigne la racine de l'arbre qui contient la plus petite clé (si H est vide alors $\min(H) = nil$)
- `taille` (H) désigne le nombre de nœuds dans la file H
- `clef` (x) est la valeur de la clé x
- `degre` (x) est le nombre d'enfants de x
- (x) est l'un des enfants de x (si x est une feuille alors $(x) = nil$)
- `pred` (x) est le l'enfant à gauche de x du parent de x et `succ` (x) est l'enfant à droite de x du parent de x (si x est une racine alors `pred` (x) et `succ` (x) sont des racines, si x est enfant unique alors `pred` $(x) =$ `succ` $(x) = x$)
- (x) est le parent de x, mais uniquement lorsque $x = ((x))$ (si x est une racine ou s'il n'y a pas de pointeur enfant vers x alors $(x) = nil$)

On peut s'appuyer sur une structure de données qui permet de lire et modifier ces valeurs en temps constants. De plus on dispose de la procédure `CréerFBRVide` de création d'une file binomiale relâchée vide.

Question 2.2.4 Écrire les procédures

(a) `ConcatenerFBR` d'union de deux files binomiales relâchées (en une file binomiale relâchée). Quelle est la complexité ?

(b) `InsererFBR` d'insertion d'une nouvelle valeur v dans une file binomiale relâchée. Quelle est la complexité ?

Question 2.2.5

(a) Écrire une procédure `Consolider` qui transforme une file binomiale relâchée en une file binomiale (avec contrainte sur les tailles).

(b) On note $a(H)$ le nombre d'arbres d'une file binomiale relâchée H.
Montrer que le coût de la procédure `Consolider` appliquée à une file H de taille n est inférieur ou égal à $\alpha(a(H) + D(n))$ où α est une constante (qui ne dépend pas de H).

Question 2.2.6 Écrire une procédure `ExtraireMinFBR` qui extrait le minimum d'une file binomiale relâchée non vide tout en la consolidant. La file obtenue après extraction du minimum est donc une file binomiale (avec contrainte sur les tailles).
S'assurer que $\min(H)$ est bien positionné sur la racine de plus petite clé de l'arbre obtenu après application de la procédure. Si nécessaire on pourra utiliser une procédure `MettreAJourMin` qui remet à jour $\min(H)$.

Question 2.2.7 En utilisant la méthode du potentiel, évaluer le coût amorti de la procédure `ExtraireMinFBR` d'extraction du minimum d'une file binomiale relâchée.
Indication : prendre une fonction de potentiel proportionnelle au nombre d'arbres de la file initiale.