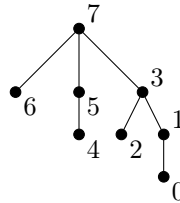


Exercice 1 (Tas binomiaux (Vuillemin 78)) Le tas binomial B_k est défini inductivement comme suit : B_k consiste en une racine ayant k fils B_0, B_1, \dots, B_{k-1} . On pourra supposer les clés, portées par les sommets, 2 à 2 distinctes. La propriété de tas est respectée : la clé d'un sommet est inférieure à la clé de n'importe lequel de ses descendants.

Une forêt binomiale est une collection de tas binomiaux à clés 2 à 2 distinctes, avec un pointeur sur le sommet portant la plus petite clé. On appelle rang d'un sommet son nombre de fils, et rang d'un tas le rang de sa racine. Une forêt est *propre* si elle contient au plus un tas de rang k , pour tout k .

1. Montrer que toutes les forêts binomiales propres à n sommets ne diffèrent que par les clés attribuées aux sommets.
2. Montrer que B_k est de rang k et a 2^{k-i-1} sommets de rang i , pour $0 \leq i < k$.
3. On numérote les sommets de B_k dans l'ordre postfixe de droite à gauche, de 0 à $2^k - 1$. On obtient par exemple la numérotation suivante sur B_3 .



Montrer qu'un sommet est de profondeur i si et seulement l'écriture en base 2 de son numéro a $k - i$ chiffres "1". (Il en résulte que B_k a $\binom{k}{i}$ sommets de profondeur i , ce qui explique le nom de « tas binomial »).

4. Montrer que si on a deux tas B_k , on obtient un tas B_{k+1} en faisant du tas B_k dont la racine a la clé minimale un fils de la racine de l'autre tas B_k . Cette opération s'appelle la *liaison*.
5. Montrer qu'on peut effectuer les opérations suivantes avec les temps amortis donnés, pour des forêts propres, où n est le nombre de sommets de la forêt produite. Avec la méthode des crédits, on pourra maintenir l'invariant : *chaque racine porte un crédit*.

Nom	Description	Coût amorti
CREERTAS(i)	Crée une forêt ayant un unique sommet i .	$O(1)$
MIN(f)	Retourne un pointeur vers le sommet de clé minimum dans la forêt f .	$O(1)$
INSÉRER(f, i)	Insère un sommet de clé i dans la forêt f .	$O(1)$
UNION(f_1, f_2)	Construit une forêt contenant l'union des clés (2 à 2 distinctes) de f_1 et f_2 .	$O(\log n)$
SUPPRIMERMIN(f)	Supprime la clé minimale de la forêt f .	$O(\log n)$

Combien de liaisons doit-on faire, au pire, pour une union ou une suppression ?

6. Si on utilise des forêts non nécessairement propres, les 4 premiers types d'opérations peuvent se faire en coût réel $O(1)$, en représentant une forêt par une liste doublement chaînée des tas qui la composent : l'union (dite alors *paresseuse*) se fait simplement en concaténant les deux listes. Montrer qu'avec cette approche, on peut maintenir le coût amorti de SUPPRIMERMIN à $O(\log n)$. L'opération SUPPRIMERMIN rétablira la propriété que la forêt est propre, et sera ainsi la seule à utiliser des liaisons.

Exercice 2 (Tas de Fibonacci (Fredman et Tarjan 84)) En utilisant encore l'union paresseuse, on veut maintenant pouvoir réaliser les opérations suivantes sur les forêts (en conservant les coûts précédents).

Nom	Description	Coût amorti
SUPPRIMER(f, x)	Supprime le sommet x de la forêt f .	$O(\log n)$
DÉCRÉMENTER(f, x, δ)	Décrémente la clé de x de $\delta > 0$.	$O(1)$

On suppose ici qu'on n'a pas besoin de chercher x au préalable. Pour les réaliser, on peut

- pour SUPPRIMER, couper le sous-arbre enraciné en x (c'est-à-dire couper le lien entre x et son père, s'il existe), supprimer la racine x de l'arbre coupé et faire l'union des tas ainsi obtenus.
- pour DÉCRÉMENTER, décrémente effectivement la clé de x , et si la propriété de tas est violée, couper le sous-arbre enraciné en x et faire l'union des tas ainsi obtenus.

Les tas obtenus ne seront donc plus binomiaux. Si on ne contrôle pas les coupes, ils n'ont plus de raison d'être suffisamment équilibrés pour pouvoir assurer les coûts amortis voulus.

Pour assurer aux tas une taille exponentielle par rapport à leur rang, on va limiter à 2 le nombre de coupes de fils d'un sommet, avant que celui-ci ne soit lui-même coupé. On marque les sommets par des crédits : la première fois qu'on coupe un fils d'un sommet y , on marque y en lui attribuant un crédit de 2. Ce crédit sera utilisé d'une part pour détecter que y a déjà un fils qui lui a été coupé, d'autre part pour payer une opération UNION ultérieure. Dès qu'on coupe un second fils à y , on coupe y lui-même de son père z et on l'ajoute dans la forêt par l'opération UNION en utilisant pour cette union l'un des 2 crédits de y . Le sommet y est donc devenu une racine de crédit 1.

Si z avait lui-même un crédit de 2 avant la coupe de y , on répète l'opération sur z , qu'on coupe de son père éventuel, etc. Il en résulte une *cascade de coupes* qui se propage jusqu'à un sommet n'ayant pas un crédit de 2, ou jusqu'à la racine.

La liaison de deux tas de même rang est définie comme dans l'exercice 1, et est utilisée seulement par les opérations SUPPRIMER et SUPPRIMERMIN pour rétablir une forêt propre. Les tas des forêts ainsi obtenues s'appellent *tas de Fibonacci*.

1. Montrer que le coût amorti de DÉCRÉMENTER est alors $O(1)$.
2. Soit x_1, \dots, x_k les fils d'un même sommet y d'un tas de Fibonacci. Ces fils ont été obtenus par liaisons lors d'opérations SUPPRIMER ou SUPPRIMERMIN. On suppose que x_i a été lié avant x_{i+1} pour tout $i < k$. Montrer que pour tout i , le rang de x_{i+2} est au moins i .
3. En déduire que la taille d'un arbre de Fibonacci est exponentiel en son rang.
4. En déduire qu'on peut assurer les coûts amortis annoncés.
5. On marque les sommets de la façon suivante : un sommet est marqué s'il a déjà eu un fils supprimé (la prochaine suppression entraînera donc une cascade de coupes, à la suite de quoi le sommet sera à nouveau non marqué). Trouver une fonction de potentiel permettant de retrouver les résultats précédents.

Exercice 3 (Minimisation d'automates acycliques (Revuz 92)) Dans cet exercice, on considère uniquement des automates déterministes accessibles et co-accessibles. On les suppose de plus *acycliques* : il n'existe aucun chemin de longueur 1 ou plus allant d'un état à lui-même. En particulier, un tel automate $\mathcal{A} = (A, Q, i, F, \delta)$ n'est pas complet, si $|A| \geq 1$. Pour un tel automate, on note $\ell = |A|$ le nombre de lettres, $m = |Q|$ son nombre d'états, $d = |\delta|$ son nombre de transitions, et $n = \ell + d$.

On suppose par ailleurs que A est représenté par l'ensemble d'entiers $[0, \ell - 1] \cap \mathbb{N}$ et que Q est représenté par l'ensemble d'entiers $[0, m - 1] \cap \mathbb{N}$.

1. Caractériser les langages reconnus par les automates acycliques.

On veut minimiser les automates acycliques en temps $O(n)$. Au lieu de partir de la partition $\{F, Q \setminus F\}$ de Q (comme dans les algorithmes de Hopcroft ou Moore) et de procéder par raffinements successifs, on part de la partition $\{\{q\} \mid q \in Q\}$ correspondant à l'égalité, et on fusionne itérativement des classes, rendant ainsi la partition plus grossière, jusqu'à obtenir l'équivalence de Nérode.

2. Pour un état $q \in Q$, montrer que

$$h(q) = \max\{|u| \mid u \in A^* \text{ et } q \cdot u \in F\}$$

est bien défini.

3. Montrer qu'on peut calculer en temps $O(n)$ tous les $h(q)$, pour $q \in Q$.
4. Quelle relation y a-t-il entre l'équivalence de Nérode et l'équivalence définie par $p \sim_h q$ si et seulement si $h(p) = h(q)$?
5. Le tri par casiers (ou baquets) *bucket sort* permet de trier en temps $O(\ell + k)$ une suite de k lettres b_1, \dots, b_k de A :
 - on alloue un tableau de ℓ cases indexé par A . Chaque case du tableau contient une liste doublement chaînée, initialement vide, d'éléments de A . Cette étape demande un temps $O(\ell)$;
 - on parcourt la suite $(b_i)_{1 \leq i \leq k}$ en ajoutant chaque élément b_i à la liste se trouvant en case b_i du tableau. Cette étape se fait en temps $O(k)$;
 - finalement, on parcourt le tableau, en concaténant les listes non vides d'éléments, ce qui se fait en temps $O(\ell)$.

Proposer un algorithme utilisant un tableau de « casiers » pour grouper une suite de mots de A^* en classes d'égalité en temps $O(K + \ell)$, où K est la somme des longueurs des mots à grouper: partant d'une suite de mots $(u_i)_i = u_1, \dots, u_p$, l'algorithme doit calculer les sous-suites $(v_i^{(1)})_i, \dots, (v_i^{(\ell)})_i$ de u telles qu'un élément de $v^{(j)}$ est égal à un élément de $v^{(k)}$ si et seulement si $j = k$.

6. On veut maintenant trier une suite de mots de A^* dans l'ordre hiérarchique. Dans cet ordre, $u \prec v$ si et seulement si $|u| < |v|$ ou $|u| = |v|$ et u est inférieur à v dans l'ordre lexicographique. Proposer un algorithme utilisant des tris par casiers pour effectuer ce tri en temps $O(K + \ell)$.
7. On suppose qu'on part d'une représentation des transitions de l'automate sous la forme suivante: pour chaque état q , on a une liste de transitions $(a_1, q_1), \dots, (a_k, q_k)$ issues de q . Montrer qu'on peut trier les transitions de l'automate, c'est-à-dire ordonner toutes ces listes de telle sorte que les suites a_i des étiquettes de transitions issues d'un même état apparaissent dans l'ordre croissant, en temps $O(n)$.
8. En utilisant les questions précédentes, décrire un algorithme pour minimiser un automate acyclique en temps $O(n)$. Décrire les structures de données proposées et la gestion des partitions.