1

Premiers algorithmes - Complexité

1.1 Yam

On dispose du résultat du lancer de 5 dés sous forme d'un tableau à 5 éléments. Ecrire un algorithme permettant d'afficher le résultat (rien, paire, double paire, brelan, full, carré, suite ou yam)

1.2 Nombres parfaits

Un nombre entier n est dit *parfait* s'il est égal à la somme de ses diviseurs, 1 compris mais n exclus.

Exemple: $6 = 1 + 2 + 3$.

Ecrire une fonction qui décide si un entier donné est ou non parfait.

1.3 Nombres premiers

Un nombre premier est un nombre divisible uniquement par lui-même et par 1 (1 n'est pas premier). On désire calculer tous les nombres premiers inférieurs ou égaux à N.

a - L'une des méthodes, connue sous le nom de crible d'Eratosthène, opère sur un vecteur de N booléens de la manière suivante:

1. le vecteur V est initialisé à *vrai*; $V[1]$ est initialisé à faux (1 n'est pas premier).
2. 2 est premier ($V[2]$ vaut vrai). On marque à faux tous les éléments $V[2k]_{k>1}$ du vecteur.
3. 3 est premier ($V[3]$ vaut vrai). On marque à faux tous les éléments $V[3k]_{k>1}$ du vecteur.
4. 4 n'est pas premier ($V[4]$ vaut faux).
5. etc ...

Ecrire l'algorithme correspondant.

b - Un nombre est premier s'il n'est divisible par aucun des nombres premiers qui lui sont inférieurs.

Ecrire un second algorithme utilisant cette propriété (les nombres premiers trouvés seront conservés dans un vecteur d'entiers P: $P[1] = 2$, $P[2] = 3$, ...)

1.4 Plateau

Etant donnée une suite $A[1..n]$ de n éléments, on appelle *plateau de la suite* A toute sous-suite $A[i..j]$ continuée d'éléments successifs égaux.

Ecrire une fonction qui retourne la longueur du plus long plateau de A.

1.5 Anagramme

Ecrire une fonction qui décide si une chaîne de caractères C est un anagramme d'une chaîne de caractères D, le blanc étant considéré comme un caractère.

Même question en ignorant les blancs.

1.6 Palindrome

Reprendre les questions précédentes pour un palindrome

1.7 Trouver le total

1. Ecrire une fonction *cherche_total* qui, étant donnés un vecteur V de n nombres positifs et un nombre x retourne $VRAI$ si x peut être obtenu comme somme de certains éléments de V et $FAUX$ sinon.
2. Modifier la fonction *cherche_total* pour qu'elle retourne une liste des éléments de V dont la somme est x (quand il en existe)

Epreuve 92-93

2

Piles, files, listes

2.1 Piles et files

Ecrire les fonctions d'accès des types *pile* et *file* en utilisant celles de*liste*

2.2 Implémentations

Ecrire les fonctions de consultation, ajout et suppression d'un élément (donné par sa position), ajout d'un élément x en position k, concaténation de deux listes pour les représentations de listes suivantes:

a - représentation contiguë dans un vecteur,

b - représentation par chaînage dans un tableau,

c - représentation par pointeur avec chaînage *avant*,

d - représentation par pointeur avec double chaînage *avant* et *arrière*,

e - représentation par pointeur avec chaînage *avant* et chaînage du dernier élément au premier (listes *circulaires*).

f - Quelles représentations paraîssent adaptées pour les piles ? pour les files ?

2.3 Le solitaire bulgare

On veut illustrer l'expérience suivante :

1. On prend un paquet de N cartes (N est un nombre triangulaire, c'est à dire de la forme $p(p+1)/2$) que l'on partage en un nombre quelconque de tas.
2. On prend une carte sur chaque tas et on forme un nouveau tas.
3. On recommence l'étape 2 jusqu'à ce que les nombres de cartes des tas soient décroissants de p à 1, ce qui constitue une suite stable. (cela se produit en un nombre fini de manipulations(*)).

Exemple : avec 10 cartes , on fait 2 tas de 6 et 4 cartes .
Les situations successives sont (6 4) , (2 5 3) , (3 1 4 2) , (4 2 3 1) , (4 3 1 2) , (4 3 2 1).

Pour simuler cette expérience, on utilisera des listes pour représenter les nombres de cartes des tas. Toutes les fonctions devront être récursives.

a - Ecrire une fonction *longueur* ayant pour argument une liste et retournant le nombre d'éléments de cette liste.

b - Ecrire une fonction *décrémenter* ayant pour argument une liste et retournant la liste de ses éléments diminués de 1 .

c - Ecrire une fonction *compresser* ayant pour argument une liste et retournant la liste privée de ses 0.

d - Ecrire une procédure *renverser* ayant pour argument une liste et qui affiche cette liste dans l'ordre inverse.

e - Ecrire une fonction *stable* ayant pour argument une liste et qui retourne vrai si les éléments forment une suite stable et faux sinon.

f - Ecrire une procédure *solitaire* ayant pour argument une liste et affichant toutes les situations intermédiaires du solitaire bulgare.

Exemple : solitaire(4,6) affiche :

4 6 , 3 5 2 , 2 4 1 3 , 1 3 2 4 , 2 1 3 4 , 1 2 3 4

Problème étudié par Martin Gardner - Scientific American - 1983

2.4 Fusion

Ecrire deux fonctions de fusion de deux listes ordonnées (sans répétition) en une nouvelle liste ordonnée. Dans la première procédure les éléments apparaissant à la fois dans les deux listes sont répétés dans la liste fusion. Dans la deuxième, il ne le sont pas.

2.5 Tri récursif de liste

1. Ecrire une fonction *range* ayant pour arguments un élément x et une liste L supposée triée et retournant une liste triée contenant x et les éléments de L.

2. En utilisant *range*, ecrire une fonction *tri* ayant pour argument une liste L et retournant cette liste triée.

2.6 Suppression des répétitions

Ecrire une fonction (récursive) *unique* ayant pour argument une liste L et retournant une liste contenant à un seul exemplaire les éléments de L.

Exemple:

unique([a,d,b,a,c,a,a,d,c]) vaut [b,a,d,c]

unique([b,a,c]) vaut [b,a,c]

2.7 Transformation d'expressions arithmétiques

1. Ecrire un algorithme de la transformation d'une expression complètement parenthèsée en l'expression postfixée correspondante.

   Exemple : $(a - ((b + c) * d)) \rightarrow abc + d * -$
   (on utilisera une pile permettant de stocker les opérateurs.)

2. Ecrire un algorithme d'évaluation tion d'une expression postfixée

3. Ecrire un algorithme de la transformation d'une expression infixée en l'expression postfixée correspondante.

2.8 Ensembles

On veut réaliser sur les ensembles les opérations **union, intersection** et **différence**. Les ensembles seront représentés par des **listes récursives** ne contenant pas de répétition.

1. Ecrire une fonction *appartient* ayant pour argument un élément x et une liste L retournant *vrai* si x est dans L et *faux* sinon, puis une fonction *est_ensemble* ayant pour argument une liste L retournant *vrai* si L est sans répétition et *faux* sinon.

2. Ecrire les fonctions *union, intersection* et *différence*.

Partiel 97-98

3

Diviser pour régner - Arbres

3.1 Fonctions symétriques des éléments d'un liste

Etant donné une liste de nombres $L = [x_1, x_2, \ldots, x_n]$, on note $S_{n,k}$ la somme de tous les produits possibles de k éléments distincts de L; soit :

$$S_{n,1} = \sum_{k=1}^{n} x_k$$

$$S_{n,k} = \sum_{i_1 < i_2 < \ldots < i_k} x_{i_1}.x_{i_2} \ldots x_{i_k}$$

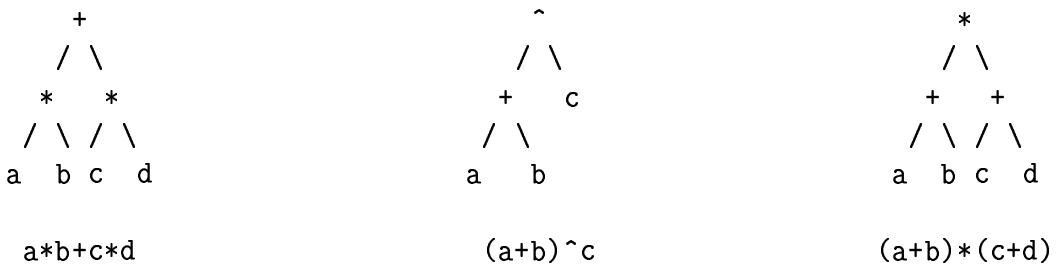
$$S_{n,n} = x_1.x_2 \ldots x_n$$

1. Montrer que $S_{n,k} = x_n.S_{n-1,k-1} + S_{n-1,k}$
2. En déduire une fonction *Somme(L,k,n)* calculant $S_{n,k}$
3. On note $C(n,k)$ le nombre d'additions et de multiplications nécessaires au calcul de *Somme(L,k,n)*. Montrer que : $C(n,k) = O(C_{n+1}^k)$

3.2 Arbres arithmétiques

On utilise des arbres binaires pour représenter des expressions arithmétiques.

Exemples de représentation :



1. Donner un algorithme d'évaluation d'une expression représentée par un arbre.
2. On veut convertir l'arbre en une expression linéaire. Choisir un type de données approprié pour cette nouvelle représentation et écrire une fonction *convertir* effectuant cette conversion ; on devra tenir compte des priorités dans l'ordre suivant : ^, puis * et /, puis + et -.

[Partiel 99-00]

3.3 Conversion d'une expression en arbre

a - Ecrire une fonction qui transforme une expression complètement parenthèsée en un arbre binaire.

b - Ecrire une fonction récursive qui construit un arbre binaire à partir de la lecture linéaire d'une expression algébrique en ordre préfixe.

3.4 Produit de deux entiers de n bits

a - La méthode que l'on enseigne habituellement dans les classes primaires pour multiplier deux entiers de n chiffres consiste à fabriquer n produits par-

tiels de taille n et ajouter ces n produits partiels.

En considérant comme opérations élémentaires les multiplications et des additions entre chiffres, quelle est la complexité de cette méthode?

b - Une seconde méthode consiste à scinder les deux nombres X et Y en deux entiers de n/2 bits:

$$X = A2^{n/2} + B$$

$$Y = C2^{n/2} + D$$

On a alors:

$$XY = AC2^n + (AD + BC)2^{n/2} + BD$$

Le calcul de XY nécessite ainsi quatre multiplications de $\frac{n}{2}$ chiffres, trois additions de chiffres ayant au maximum $2n$ chiffres et deux décalages (multiplications par 2^n et $2^{\frac{n}{2}}$)

Ecrire la fonction récursive calculant le produit de X par Y selon cette méthode.

Donner la complexité dans le pire des cas $T(n)$ en fonction de $T(\frac{n}{2})$ et préciser à quelle classe de fonction elle appartient.

c - Pour obtenir un algorithme plus efficace, on diminue le nombre le sous-problèmes:

$$XY = AC2^n + [(A - B)(D - C) + AC + BD]2^{\frac{n}{2}} + BD$$

Le calcul de XY ne nécessite plus maintenant que trois multiplications de $\frac{n}{2}$ chiffres, six additions ou soustractions et deux décalages

Calculer la nouvelle complexité

4

Programmation dynamique - Algorithmes gloutons

4.1 Coefficients binômiaux

C_n^p représente le nombre de parties à p éléments d'un ensemble à n éléments ($n \geq p \geq 0$).

a - On peut exprimer C_n^p à l'aide de factorielles.

Pourquoi cela ne donne-t-il pas un algorithme de calcul "utilisable" ?

b - Ecrire une fonction *récursive* qui calcule C_n^p en utilisant la formule :

$$\forall n > 1, \forall p \in \{1, \ldots, n-1\}, C_n^p = C_{n-1}^p + C_{n-1}^{p-1}$$

Calculer la complexité en temps de cette fonction

c - En utilisant un tableau à deux dimensions, donner une fonction itérative qui calcule C_n^p.

Calculer les complexités en temps et en espace de cette fonction.

d - Modifier cette fonction pour qu'elle ait un complexité linéaire en espace.

4.2 Fibonacci

La suite de Fibonacci $(F_n)_{n\geq 0}$ est définie par :

$$F_0 = F_1 = 1 \qquad ; \qquad \forall n > 1,\ F_n = F_{n-2} + F_{n-1}$$

1. une fonction récursive qui permet de calculer le $n^{ième}$ terme de la suite $(n \geq 0)$.
   Quelle est sa complexité ?
2. Modifier cette fonction pour que les valeurs déjà calculées soient stockées.
   Quelle est alors la complexité ?

4.3 Gains sur un damier

On dispose d'un damier $n \times n$ et d'un jeton.

> On part de la ligne inférieure en se déplaçant uniquement vers le haut.
>
> A chaque passage d'une case x à une case y, on obtient un gain $g(x, y)$ (pas nécessairement positif)

Donner un algorithme qui détermine le trajet entre le bord inférieur et le bord supérieur réalisant le gain maximal.
Quelle est sa complexité ?

4.4 Organisation d'une fête

Une entreprise veut organiser une fête en respectant les contraintes suivantes:

- L'entreprise a une structure hiérarchique dont la racine est le P.D.G.
- Un membre de l'entreprise ne doit pas être invité en même temps que son supérieur direct.
- Chaque membre ayant une note de convivialité, le total de ces notes doit être maximal.

Donner un algorithme établissant la liste des invités.
Calculer sa complexité.

4.5 Le plein d'essence

Vous devez vous rendre de d'Amsterdam à Lisbonne par l'autoroute. Votre plein vous permet de parcourir n kilomètres et votre carte vous donne les distances entre les stations-service.
Donner une méthode pour déterminer les stations où vous devez vous arrêter pour faire le moins d'arrêts possible et prouver la validité de la méthode.

4.6 Rendu de monnaie

On dispose de pièces de monnaie de k valeurs (entières) différentes pour rendre une somme de n cents.

- Donner un algorithme permettant de rendre la monnaie en utilisant le moins de pièces possible.
- Donner un ensemble de valeurs pour lesquelles l'algorithme glouton ne donne pas de solution optimale. (L'ensemble doit inclure une pièce de 1 cent pour qu'il y ait une solution pour chaque valeur de n.

4.7 Codage d'Huffman

4.7.1 But

Il s'agit de coder les éléments de l'alphabet par un codage binaire tel que la longueur moyenne de chaque mot soit minimale. Pour obtenir ce résultat, on affecte les codes les plus courts aux lettres les plus fréquentes.
Ce code est obtenu à l'aide d'un arbre binaire.

4.7.2 Construction de l'arbre binaire

On construit un ensemble d'arbres à un noeud dont la racine contient un caractère muni de sa fréquence. A chaque étape, on *enlève* les éléments dont la fréquence est la plus faible. Ces deux éléments deviennent les fils d'un nouveau noeud dont la fréquence est la somme des fréquences des deux fils. Le nouvel arbre est alors *ajouté* à l'ensemble. On continue ainsi jusqu'à ce qu'il ne reste qu'un seul arbre.

Remarque: les noeuds internes ne contiennent pas les caractères.

	0.16460	a	0.06737	b	0.00680	c	0.02900	d	0.03037	e	0.14326
f	0.00921	g	0.01058	h	0.00568	i	0.06686	j	0.00103	k	0.00034
l	0.04483	m	0.03123	n	0.05817	o	0.04414	p	0.02736	q	0.00843
r	0.06582	s	0.06075	t	0.06410	u	0.04655	v	0.00826	w	0.00001
x	0.00243	y	0.00181	z	0.00101						

Figure 4.1: Exemple de table des fréquences relevées dans un texte

4.7.3 Codage des caractères

On a obtenu un arbre A dont toutes les feuilles contiennent des caractères. Pour obtenir le codage d'un caractère il suffit de parcourir la branche qui le contient en ajoutant un 0 chaque fois qu'on passe par le fils gauche et un 1 chaque fois qu'on passe par le fils droit.

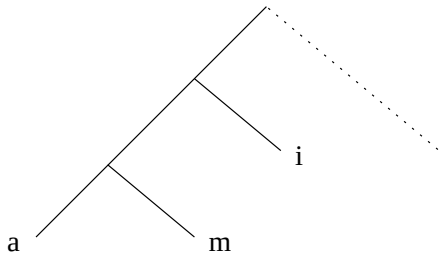


Figure 4.2: Codage de **m** est 001, celui de **i** est 01 et celui de **a** est 000

Ecrire les algorithmes permettant :

- La construction de l'arbre
- La construction d'une table de codage
- Le codage d'une chaîne de caractères et décodage d'une suite de 0 et de 1.

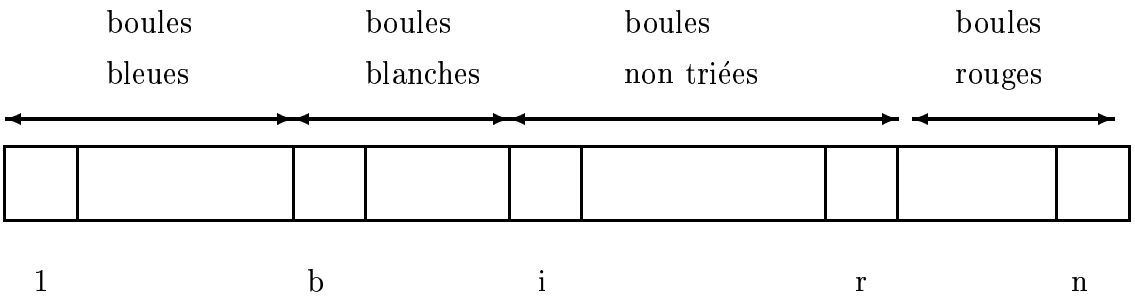
5

Tris

5.1 Drapeau hollandais

Un vecteur contient N boules qui peuvent être bleues, blanches ou rouges. On demande de trier le vecteur de telle sorte qu'il contienne d'abord les boules bleues, ensuite les boules blanches et enfin les boules rouges. On impose de plus que la couleur de chaque boule ne soit testée qu'une seule fois.

Principe : supposons que quelques-unes des premières boules aient été déjà traitées. Cette situation est représentée sur la figure ci-dessous. Quelle est l'étape suivante ?



- Ecrire l'algorithme.

- Quelles modifications faut-il apporter pour réaliser la segmentation dans le tri rapide ?

5.2 Tri linéaire

On a un "grand" vecteur de valeurs prises dans un "petit" ensemble V. Donner un algorithme *linéaire* de tri de ce vecteur.

5.3 Tri par insertion

Donner un algorithme de *tri par insertion* dont le principe est le suivant. Supposons triées les i premières valeurs du vecteur V, on insère la $(i+1)$-ième valeur à sa place parmi les i premières.

Calculer la complexité (comparaisons et échanges) de l'algorithme.

5.4 Tri par sélection-permutation

Donner un algorithme de *tri par sélection-permutation* dont le principe est le suivant. Supposons les i plus petites valeurs du vecteur V rangées dans les i premières cases de V, on cherche la plus petite valeur parmi $V[i+1], \ldots, V[n]$, puis on l'échange avec la $(i+1)$-ième première valeur de V.

Calculer la complexité (comparaisons et échanges) de l'algorithme.

5.5 Tri par bulle

Donner un algorithme de *tri par bulles* dont le principe est le suivant. Supposons les i dernières valeurs du vecteur V sont telles que $V[n-i+1] \leq \ldots \leq$

$V[n]$ et que $V[j] \leq V[i]$ pour tout $j < i$. On place alors la $(i+1)$-ième valeur de V en partant de $V[1]$ et en permutant 2 valeurs consécutives de V quand elles ne sont pas dans le bon ordre.

Calculer la complexité (comparaisons et échanges) de l'algorithme.

5.6 Tri par énumération

Soit à trier une liste L de taille n dont les éléments sont 2 à 2 différents. On compare chaque élément l_i à tous les autres et on compte le nombre n_i d'éléments inférieurs à l_i.

a - Que représente n_i ?

b - Ecrire un algorithme qui construit le tableau des n_i, puis ordonne sur place la liste L.

c - Quelle modification faut-il apporter si on n'impose pas aux éléments d'être 2 à 2 différents.

5.7 Complexité du tri rapide

On note $M(n)$ le nombre moyen de de comparaisons effectuées par la procédure *TriRapide* sur n éléments.

1. Montrer que : $M(n) \leq n + 1 + \sum_{k=1}^{n} M(k-1) + M(n-k)$
2. On pose : $\begin{cases} A(n) = n - 1 + \sum_{k=1}^{n} A(k-1) + A(n-k) \\ B(n) = n + 1 + \sum_{k=1}^{n} B(k-1) + B(n-k) \end{cases}$
   Calculer $A(n)$ et $B(n)$
3. Montrer que $A(n) \leq M(n) \leq B(n)$. Que peut-on en déduire pour $M(n)\tilde{?}$

5.8 Tri rapide itératif

Ecrire un algorithme itératif de **tri rapide** utilisant une pile. (On suppose que ls fonction de "segmentation" conduisant au placement correct du pivot est donnée.)

5.9 Tri partiel

Etant donné un tableau de n elements , on veut trier les k plus grands ($n > k$).

Calculer en fonction de n et k la complexite de ce tri pour les tris suivants : tri par insertion, tri par selection, tri rapide, tri par tas.

Quel est le meilleur tri si k est petit par rapport à n ?

5.10 Tri par tournoi

On va trier une liste de n valeurs de la manière suivante. On organise un tournoi des n éléments à trier, ce qui donne un arbre binaire tel que chaque noeud interne est égal au plus petit de ses 2 fils. On supprime ensuite la racine, en le remplacant partout dans l'arbre par un élément supérieur à tous ceux de la liste à trier, puis on reconstruit le tournoi à partir des feuilles. Après n suppressions de ce type, on a trié les n éléments.

a - Faire tourner l'algorithme sur $4, 7, 10, 3, 15, 12, 1, 8, 2, 6$.

b - Implémenter cet algorithme.

c - Quelle est sa complexité au pire en nombre de comparaisons?

5.11 Tri externe

On considère le problème du tri d'un fichier F à n éléments sur un ordinateur dont la mémoire peut contenir au plus p éléments. $(p < n)$

On va donc trier F en sous-séquences de taille p que l'on écrit dans des fichiers (triés) $f_1, f_2 \dots f_k$
$(k = \lceil \frac{n}{p} \rceil)$ que l'on fusionnera ensuite.

On suppose connues les opérations *lire, est-vide* et *ecrire* pour des fichiers à accés séquentiel telles que :

- lire(x,f) place dans x le 1er élément de f et le retire de f.
- est-vide(f) retourne vrai si f est vide.
- ecrire(x,f) écrit x a la fin de f.

1ère méthode

on fusionne f_1 et f_2 dans un nouveau fichier f_{k+1}, f_3 et f_4 en f_{k+2} et on recommence jusqu'à obtention d'un unique fichier.

1. Décrire une structure de données permettant d'organiser ces fusions successives.
2. Ecrire un algorithme qui génère un fichier trié a partir du fichier F (on suppose donnée une procédure de tri en mémoire de complexité au pire $p \ln p$)
3. Déterminer la complexité au pire de cet algorithme.

2ème méthode

On fusionne simultanément les k fichiers, en utilisant un ensemble S de taille au plus k et contenant le plus petit élément courant de chaque fichier lors de la fusion.

1. Comment organiser S pour qu'à chaque étape le plus petit élément de S soit accessible en temps constant ?
   De quelles primitives a-t-on besoin pour assurer cette complexité ?

2. Utiliser cette structure et les primitives associées pour écrire un algorithme de k fusions simultanées .

3. Determiner la complexité au pire de cet algorithme.

Epreuve 98-99