Algorithmes élémentaires

0.1 Boucle Pour

Traduisez la boucle **pour** suivante en utilisant une boucle **tant que**, puis une boucle **répéter** ($N \geq 0$).

pour

Pour i allant de 1 à N **faire**
    A
Fin pour

0.2 Boucle Répéter

Traduisez la boucle **répéter** suivante par une boucle **tant que**.

répéter

Répéter
    A
jusqu'à condition

0.3 Calcul

a - Calculer le quotient et le reste de la division de a par b en n'utilisant que les opérations + et -.

b - Calculer le PGCD à deux entiers en utilisant l'algorithme d'Euclide

0.4 Jeu

On doit deviner un nombre entre 1 et 1000 en 10 essais au plus. A chaque tentative, le programme indique si la réponse était trop petite ou trop grande.

Donner l'algorithme permettant de réaliser ce jeu.

0.5 Année

a - Ecrire un algorithme indiquant si une année est bissextile ou non.

a - Un jour de l'année est représenté par trois entiers j, m, a où :

- j représente le numéro du jour dans le mois (de 1 à 31),
- m le numéro du mois dans l'année (de 1 à 12),
- et a l'année.

Exemple : 1 10 1993

Ecrire un algorithme qui à partir d'un jour de l'année j, m, a calcule le jour suivant j', m', a'.

0.6 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.
- Quelle est la complexité de cet algorithme ?

0.7 Yam

[Epreuve 93-94]

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)

Les Vecteurs

0.1 Préliminaires

Ecrire les algorithmes permettant d'obtenir :

a - le nombre d'occurrences d'une valeur a dans un vecteur V indicé de 1 à N, ainsi que les indices de la première et de la dernière occurrence,

b - la plus petite et la plus grande valeur d'un vecteur,

c - la valeur moyenne d'un vecteur.

Ecrire un algorithme inversant les positions des valeurs d'un vecteur.

0.2 Vecteur trié

a - Ecrire un algorithme réalisant l'insertion d'une valeur a dans un vecteur trié.

b - Ecrire deux algorithmes de recherche d'une valeur a dans un vecteur trié, l'un linéaire, l'autre dichotomique. Etudier leur complexité.

0.3 Polynômes

On code les polynômes $P(x) = a_0 + a_1x + a_2x^2 + \ldots + a_{n-1}x^{n-1} + a_nx^n$ par des vecteurs V avec $\forall i \in [0, n],\ V[i] = a_i$

Donner les algorithmes calculant :

a - la dérivée d'un polynôme,

b - la somme de deux polynômes,

c - le produit de deux polynômes.

0.4 Les ensembles

Soit E un ensemble à n éléments indicés de 1 à n. On représente un sous-ensemble A de E par un vecteur de booléens $A[1..n]$ tel que :

- $A[i] = vrai$ si le i-*ième* élément de E appartient à A,
- $A[i] = faux$ sinon

A partir de cette représentation, écrire les algorithmes calculant :

a - l'union de deux sous-ensembles,

b - l'intersection de deux sous-ensembles,

c - la complémentation d'un sous-ensemble,

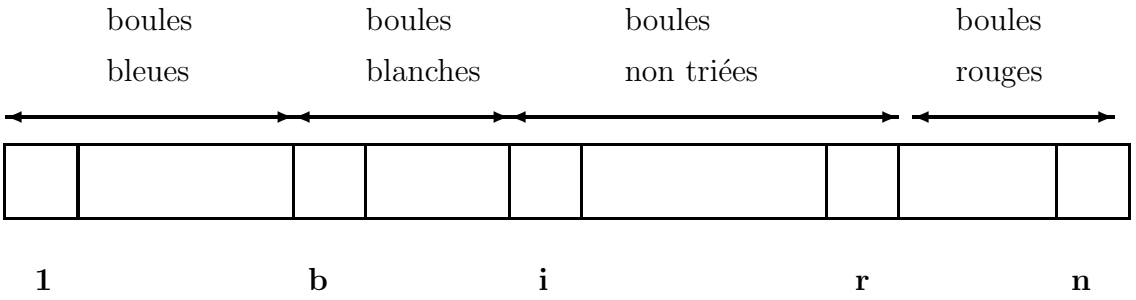
d - la différence symétrique de deux sous-ensembles,

e - le cardinal d'un ensemble.

0.5 Drapeau français

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 ?



0.6 Anagramme

Ecrire une fonction qui dcide si une chane de caractres C est un anagramme d'une chane de caractres D, le blanc tant considr comme caractre.

Mme question en ignorant les blancs.

0.7 Plateau

Etant donne une suite $A[1..n]$ de n lments, on appelle plateau de la suite A toute sous-suite $A[i..j]$ qui contient des lments conscutifs et gaux.

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

Récursivité

0.1 Préliminaires

Ecrire sous forme récursive les fonctions qui calculent :

a - $n!$ (fonction factorielle) pour un entier positif n quelconque,

b - le pgcd de deux nombres entiers strictement positifs,

c - la puissance entière d'un réel.

0.2 C_n^p

[Epreuve 93-94]

C_n^p représente le nombre de combinaisons de p éléments parmi un ensemble de n éléments ($n \geq p \geq 0$). Ce nombre peut se calculer de la manière suivante :

C_n^p

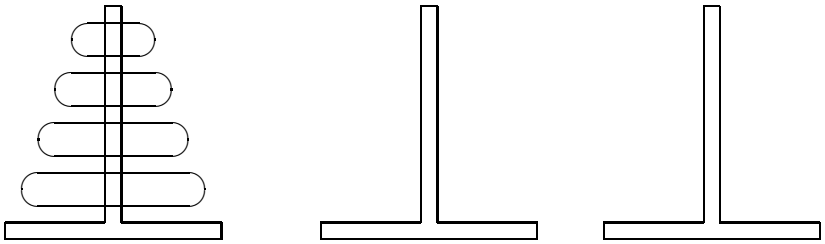
$C_n^0 = C_n^n = 1$
$\forall n \geq 1,\ p \geq 1,\ C_n^p = C_{n-1}^p + C_{n-1}^{p-1}$

- Ecrire une fonction qui calcule C_n^p.
- Calculer sa complexité C(n) (On posera : K(n)=1+C(n))

0.3 Tour de Hanoi

Il s'agit d'un jeu et d'une légende, rapportée d'extrême-orient. Des moines bouddhistes jouent avec des tours et des disques, et la fin de la partie est censée symboliser et annoncer la fin du monde.

On dispose de trois tours susceptibles de recevoir des disques empilés suivant des diamètres décroissants. La situation de départ peut être représentée de la manière suivante :



Départ *Inter* *Arrivée*

Le jeu consiste à faire passer les disques de la tour *Départ* sur la tour *Arrivée*, en ne déplaçant qu'un seul disque à la fois. Aucun disque ne doit être empilé sur un disque de plus petite dimension. Il n'est pas nécessaire de passer par la tour *Inter* à chaque fois.

Indication : si l'on sait résoudre le problème avec $n-1$ disques, la solution pour n disques est triviale (trouvez-la!). Une procédure récursive semble donc judicieuse.

0.4 Fonction Machin

[Epreuve 95-96]

Soient m un entier naturel, $N = 2^m$ et T un tableau [1..N] d'entiers. On considère la fonction suivante :

```
fonction Machin (T : tableau; I, J : entier) : entier
var A, B, D, S : entier
Début
    Si (J = 1) Alors
        retour (T[I])
    Sinon Début
        S ← J div 2
        D ← I + S
        A ← Machin (T, I, S)
        B ← Machin (T, D, S)
        Si (A > B) Alors
            retour (A)
        Sinon
            retour (B)
        Fin Si
    Fin Si
Fin
```

1) Que retourne la fonction Machin(T,1,N) ?
   Essayer, par exemple, avec le tableau : T = [3,7,1,4,5,9,2,6].

2) On note $C(N)$ le nombre d'affectations (symbole ←) dû à l'appel de Machin(T,1,N).
   Pour N pair quelconque, établir une relation de récurrence (R_N) entre $C(N)$ et $C(N/2)$, c'est à dire exprimer $C(N)$ en fonction de $C(N/2)$.

3) Pour $N = 2^k$, en écrivant les relations R_{2^k}, $R_{2^{k-1}}$, ..., déduire la valeur de $C(N)$.

4) Ecrire l'algorithme itératif classique calculant le même résultat.
   Quelle est sa complexité au pire ?
   Conclure.

Tris

0.1 Tri linaire

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

0.2 Tri par insertion

Donner un algorithme de tri par insertion dont le principe est le suivant. Supposons tries les i premires valeurs du vecteur V, on insre la $(i+1)$ i-me valeur sa place parmi les i premires.

0.3 Tri par bulle

Donner un algorithme de tri par bulles dont le principe est le suivant. Supposons les i dernires valeurs du vecteur V sont telles que $V[n-i+1] \leq\leq V[n]$ et que $V[j] \leq V[i]$ pour tout $j < i$. On place alors la $(i+1)$-ime valeur de V en partant de $V[1]$ et en permutant 2 valeurs conscutives de V quand elles sont pas dans le bon ordre.
Calculer la complexit de l'algorithme.

0.4 Tri par slection-permutation

Donner un algorithme de tri par slection-permutation dont le principe est le suivant. Supposons les i plus petites valeurs du vecteur V ranges dans les i premires cases de V, on cherche la plus petite valeur parmi $V[i+1] \cdots V[n]$, puis on l'change avec la $(i+1)$-ime premire valeur de V.
Calculer la complexit de l'algorithme.

0.5 Tri par numration

Soit trier une liste L de taille n dont les lments sont 2 2 diffrents. On compare chaque lment l_i tous les autres et on compte le nombre n_i d'lments infrieurs l_i.

- Que reprsente n_i?
- Ecrire un algorithme qui construit le tableau des n_i, puis ordonne sur place la liste L.
- Quelle modification faut-il apporter si on n'impose pas aux lments d'tres 2 2 diffrents.

0.6 Le tri rapide ou quicksort

On se propose d'étudier un algorithme récursif qui trie un vecteur. Il fonctionne en plusieurs étapes. L'une de ces étapes est l'algorithme de segmentation.

La segmentation

L'algorithme de segmentation procède sur un vecteur V de la manière suivante: soit x un élément quelconque du vecteur,

1) à partir de la gauche, on balaye le vecteur pour trouver le premier élément v_i tel que $v_i > x$,

2) à partir de la droite, on balaye le vecteur pour trouver le premier élément v_j tel que $v_j \leq x$,

3) on permute alors les deux éléments v_i et v_j.

4) on répète les trois étapes précédentes sur le vecteur réduit aux éléments compris entre les indices i et j, et cela jusqu'à ce que les deux balayages se rencontrent quelque part au centre du vecteur.

a - Segmenter le vecteur $V = [44, 55, 12, 42, 94, 6, 16, 67]$. Quelle est la nature de la modification effectuée par la segmentation ?
b - Ecrire la procédure de segmentation.

L'algorithme quicksort

c - Ecrire une procédure (récursive) de tri utilisant l'algorithme de segmentation.
d - Quelle est l'influence du choix de x sur l'algorithme ?
e - Evaluer sa complexité (nombre de comparaisons et de permutations) dans le pire et et dans le meilleur des cas.

Programmation dynamique

0.1 Coefficients binomiaux

C_n^p représente le nombre de combinaisons de p éléments parmi un ensemble de n éléments ($n \geq p \geq 0$). Ce nombre peut se calculer de la manière suivante :

C_n^p

$C_n^0 = C_n^n = 1$
$\forall n \geq 1,\ p \geq 1,\ C_n^p = C_{n-1}^p + C_{n-1}^{p-1}$

- En utilisant un tableau á deux dimensions, ecrire une fonction itrative qui calcule C_n^p.
- Calculer sa complexité en temps et en espace.

0.2 Fibonacci

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

F_n

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

- Quelle est sa complexité pour calculer le n^{ime} terme de la suite ($n \geq 0$).
- Modifier cette fonction pour que les valeurs déjá calculés soient stockes.
- Calculer sa complexité en temps et en espace.

0.3 Gain sur un damier

On dispose d'un damier $n \times n$ et d'un jeton.
On part de la ligne inférieure en se déplacant uniquement vers le haut.
A chaque passage d'une case x á une case y, on obtient un gain $g(x, y)$ (pas necessairement positif)

Donner un algorithm qui détermine le trajet entre le bord inférieur et le bord supérieur réalisant le gain maximal. Calculer sa complexité.

0.4 Organisation d'une fete

Une entreprise veut organiser une fete 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 etre invité en meme temps que son supèrieur direct.
- Chaque membre ayant une note de convivialité, le total de ces notes doit etre maximal.

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

0.5 Le plein d'essence

Vous devez vous rendre de "Amsterdam" á *Lisbonne* par l'autoroute. Votre plein vous permet de parcourir n kilométres et votre carte vous donne les distances entre les stations-services.
Donner une méthode pour déterminer les stations ou vous devez vous arrer pour faire le moins d'arret possible et prouver la validité de la méthode.

0.6 Rendu de monnaie

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

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

Algorithmes gloutons

0.1 Le problème du choix d'activités

Ce problème consiste à répartir une ressource parmi plusieurs activités en compétition. Soit A un ensemble de n activités concurrentes ($A[i]$ d'ésigne l'activité numéro i), ces activités souhaitent utiliser une ressource, comme une salle de cours, qui ne peut être utilisée que pour une activité à la fois. Chaque activité i posséde un horaire de début d_i et un horaire de fin f_i, avec $d_i < f_i$. Si elle est choisie, l'activité i a lieu dans l'intervalle de temps ouvert $[d_i, f_i[$. Les activités i et j sont compatibles si les intervalles $[d_i, f_i[$ et $[d_j, f_j[$ ne se superposent pas (i.e. $d_i \geq f_j$ ou $d_j \geq f_i$). Le problème du choix d'activités est de choisir un ensemble contennant le plus grand nombre possible d'activitś compatibles entre elles.

0.2 Le problème du sac à dos

Le problème du sac à dos est le suivant. On dispose de n objets pour remplir un sac à dos. Chaque objet i à un poids w_i et une valeur v_i et le sac à dos a une capacité maximale de W kilos. Il s'agit de remplir le sac à dos avec des objets avec une valeur totale maximale mais sans pour autant dépasser la capacité du sac à dos. On propose d'étudier deux variantes. On considère un exemple composé de 05 objets représentés dans la liste suivante $(poids, valeur)_i$: $\{(9,10),(3,5),(2,4),(2,3),(1,2)\}$ et un sac de capacité 10.

- Dans la variante *tout ou rien*, les objets sont indivisibles. Un objet est mis entièrement dans le sac à dos ou il est laissé complètement en dehors du sac à dos. Donner un algorithme glouton pour cette version du sac à dos. Comparez la solution optimale et la solution calculée par votre algorithme pour l'exemple ci-dessus.
- Dans la variante *fractionnaire*, il est possible de mettre tout un objet ou seulement une fraction $0 < p_i \leq 1$ d'un objet i. les contributions de l'objet sont alors de $p_i v_i$ en valeur et de $p_i w_i$ en poids. Donner un algorithme glouton pour la variante *fractionnaire* du sac à dos. Comparez la solution optimale et la solution calculée par votre algorithme pour l'exemple ci-dessus.

0.3 Le problème du voyageur de commerce

Un voyageur de commerce doit visiter n villes. Il veut le faire en parcourant le moins de distance possible et sans repasser deux fois par la même ville, sauf la première (il aimerait bien rentrer chez lui à la fin!). Il faut donc trouver le bon ordre de visite de ces villes. L'idée est de choisir pour le chemin les arêtes (distance entre deux villes) de longueur minimal.

Listes, files et piles

0.1 Applications

a - Ecrire une fonction *longueur* qui calcule le nombre d'éléments d'une liste.

b - Ecrire une fonction *égale* testant l'égalité de deux liste.

c - Ecrire une procédure supprimant la première occurrence d'un élément dans une liste.

0.2 Fusion de deux listes triées

Ecrire une procédure de fusion de deux listes triées en une nouvelle liste triée.

0.3 AM-STRAM-GRAM

Pour jouer à AM-STRAM-GRAM, n enfants forment une ronde, choisissent l'un d'eux comme premier, commencent à compter à partir de ce premier et sortent le $k^{iéme}$ de la ronde. L'enfant suivant devient le premier et on recommence jusqu'à ce qu'il n'y ait plus d'enfant.

En représentant la ronde par une liste circulaire, écrire une procédure qui donne la suite des enfants dans l'ordre où ils sont sortis de la ronde.

0.4 Le solitaire bulgare

On veut illustrer l'expérience suivante :

1) On prend un paquet de N cartes (N= 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 consécutifs (on admettra que 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 récursives (ainsi que leurs opérations associées : cons, tête, fin et vide) pour représenter les nombres de cartes des tas.

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 *suite* ayant pour argument une liste et qui retourne vrai si les éléments forment une suite arithmétique de raison -1 et faux sinon.

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

0.5 Une file de points à colorier

On dispose d'un écran constitué de n×n points. Chacun de ces points peut prendre soit la couleur noire, soit la couleur blanche, soit la couleur rouge.

Au départ, une région fermée de l'écran est blanche, tandis que le reste de l'écran est noir. On souhaite changer les points blancs en points rouges, et ceci sans parcourir tout l'écran. Pour cela, on suppose que l'on connaît les coordonnées (x,y) d'un point de la région blanche.

a - Ecrire l'algorithme réalisant le coloriage. On utilisera une file.

b - Y a-t-il un algorithme plus efficace ?

0.6 Implementation de file

[Epreuve 95-96]

On désire représenter des **files** en utilisant des listes **circulaires** implémentées avec des **pointeurs**.

1) Quelle est la structure de données utilisée.
2) Sur quel élément le pointeur de file doit-il pointer pour que les opérations enfiler et défiler soient simples à réaliser ?
3) Ecrire les fonctions associées au type abstrait de données **file** (vide, premier, enfiler, défiler) en utilisant cette représentation.

0.7 Transformation d'expressions arithmétiques

Ecrire une procédure qui effectue la transformation d'une expression complètement parenthèsée en l'expression postfixée correspondante.

Exemple : $(a \ - \ ((b \ + \ c) \ * \ d)) \ \rightarrow \ abc + d * -$

Cette procédure utilisera une pile permettant de stocker les opérateurs.

Arbres

0.1 Une représentation par pointeurs

a- Donner une représentation en C des arbres binaires et écrire les primitives de manipulation.

b- Définir les différents parcours sur les arbres binaires : en profondeur (préordre, infixé et postordre). Ecrire les procédures correspondantes.

0.2 Applications

a- Ecrire les fonctions retournant la hauteur, le nombre de noeuds et le nombre de feuilles d'un arbre binaire.

b- La longueur de cheminement d'un arbre est définie par :

$$longueur\ de\ cheminement(B) = \sum_{x\ noeud\ de\ B} profondeur(x)$$

Ecrire une fonction qui calcule la longueur de cheminement d'un arbre. (La profondeur d'un sommet étant définie comme sa distance depuis la racine de l'arbre).

0.3 Codage

[Epreuve 94-95]

1 - Ecrire un algorithme récursif qui affiche les éléments d'une pile dans l'ordre où ils ont été empilés.

2 - On code les nœuds d'un arbre binaire de la manière suivante:

- la racine est codée par le mot vide ε,
- si m est le mot codant un nœud a, alors le code du fils gauche (respectivement droit) de a est obtenu en concaténant 0 (resp. 1) à la fin de m.

Ecrire un algorithme qui affiche la liste des mots codant les feuilles d'un arbre.

0.4 Arbres binaires et expressions

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

0.5 Nouvelle implémentation

a- On représente les arbres quelconques avec 2 pointeurs : le premier pointe vers le premier fils, le deuxième pointe vers le frère droit.

Implémenter cette représentation et l'utiliser pour donner des algorithmes récursif de parcours.

b- On représente les arbres quelconques avec 3 pointeurs : le premier pointe vers le premier fils, le deuxième pointe vers le frère droit, le troisième pointe sur le père.

Implémenter cette représentation et l'utiliser pour donner un algorithme *itératif* de parcours *sans utiliser de piles*.