

## Exercices d'algorithmique (annales d'examens)

Jean-Marc.Vincent@imag.fr

### Préambule (à lire impérativement)

Ce document contient une compilation des exercices posés en examen d'algorithmique en licence d'informatique à l'UFR IMA de l'Université Joseph Fourier. Il est destiné aux étudiants de l'UE ALGO 5 pour s'entraîner à faire des exercices sur la fin de ce semestre. Comme le contenu pédagogique à évolué ces dernières années, certains exercices seront plus difficiles et ne correspondent pas au programme actuel. En particulier, l'écriture de programmes en langage C ne relève plus de cette UE.

Certains exercices comportent, malgré mes efforts, des coquilles qui avaient été corrigées le jour de l'épreuve mais n'ont pas été actualisées. Je remercie donc par avance les étudiants qui amélioreront ce document par leurs remarques.

Un grand nombre de ces exercices sont largement inspirés d'ouvrages et de sujets posés par des collègues enseignant l'algorithmique. Je tiens à les remercier pour cette mise à disposition.

Pour des raisons évidentes, ces sujets ne sont pas corrigés, une version avec des indications est en cours d'écriture. En cas de difficultés sérieuses, vous pouvez prendre contact avec vos enseignants de TD. Vous trouverez ci-dessous un accès aux différents exercices à partir de mots clés correspondant aux grandes parties de l'UE ALGO 5.

Bon courage

J.-M. Vincent

## Thèmes (par numéro d'exercice)

Calcul de coût [3,12,15,26,27,29, 33,35,36, 51, 55, 56, 57, 59]

Comptage [28,35,37, 44, 46, 49, 52, 56, 58]

Enrichissement de structure [1,8,35,39, 54]

Listes et ensembles [4,5,12,17,18,54,31,32, 33, 36, 40, 47, 49, 53, 57, 59, 60, 61,29]

Manipulation de structures [45, 51, 60,25]

Parcours [2,8,19,27,34,40, 50, 53, 54, 57, 62]

Recherche [16,37]

Structures d'arbres [1,2,7,8,19,13,14,19,28,34,35,37, 38, 43, 46, 50, 51, 52, 54, 62, 63, 30,,22,24]

Tables de hachage [4,5,12,26,31, 41, 40, 61]

Tri [1,3,10,12,18,21,27,32, 38, 39, 43, 44, 48, 55, 29, 59,23]

Chaînes de caractère [9,19,26]

Glouton [6]

## Table des matières

1	Médiane ( <i>Quick de décembre 2012</i> )	6
2	Nombre de Pivert ( <i>Quick de décembre 2012</i> )	6
3	Coût d'un algorithme ( <i>Quick d'octobre 2012</i> )	6
4	Étudiants assidus ( <i>Quick d'octobre 2012</i> )	7
5	Choix de représentants ( <i>Examen de décembre 2012</i> )	7
6	Planification ( <i>Examen de décembre 2012</i> )	8
7	Parcours d'arbres ( <i>Examen de décembre 2012</i> )	8
8	Feuille profonde ( <i>Examen de juin 2012</i> )	9
9	Séquence de lettres ( <i>Examen de juin 2012</i> )	10
10	ADE ( <i>Examen de décembre 2011</i> )	10
11	Arbres binaires complets ( <i>Examen de décembre 2011</i> )	11
12	Un tri linéaire ? ( <i>Examen de décembre 2011</i> )	12
13	Feuilles ( <i>Quick de novembre 2011</i> )	12
14	Tas ternaire ( <i>Quick de novembre 2011</i> )	13
15	C'est un truc... ( <i>Quick d'octobre 2011</i> )	13
16	Table ordonnée ( <i>Quick d'octobre 2011</i> )	14
17	Pile, File le retour ( <i>Quick d'octobre 2011</i> )	14
18	Tri par file ( <i>Quick d'octobre 2011</i> )	14
19	Arbres binaires complets ( <i>Examen de décembre 2011</i> )	15
20	ADE ( <i>Examen de décembre 2011</i> )	16
21	Un tri linéaire ? ( <i>Examen de décembre 2011</i> )	16
22	Recherche dans un tas ( <i>Examen de décembre 2010</i> )	17
23	Inversion ( <i>Examen de décembre 2010</i> )	17
24	Plus grand stable ( <i>Examen de décembre 2010</i> )	18

25	Algorithme de couverture ( <i>Examen de décembre 2010</i> )	18
26	Algorithme de Karp-Rabin ( <i>Examen de décembre 2009</i> )	19
27	Recherche des $k$ plus grands éléments ( <i>Examen de décembre 2009</i> )	21
28	Feuilles ( <i>Examen de décembre 2009</i> )	22
29	Un peu de couleur ( <i>Examen de juin 2009</i> )	22
30	Est-ce un tas ? ( <i>Examen de juin 2009</i> )	22
31	Anagrammes de phrase ( <i>Examen de juin 2008</i> )	23
32	Tableau trié ( <i>Examen de juin 2008</i> )	23
33	Extraction de couleurs ( <i>Examen de décembre 2007</i> )	24
34	Premier et second ( <i>Examen de décembre 2007</i> )	25
35	Arbres pondérés ( <i>Examen de décembre 2007</i> )	26
36	Matrice ( <i>Examen de juin 2007</i> )	27
37	B-arbre ( <i>Examen de décembre 2006 et juin 2007</i> )	28
38	Tous en rang ( <i>Examen de juin 2007</i> )	28
39	Tous en rang (autre version) ( <i>Examen de décembre 2006</i> )	29
40	Ensemble ( <i>Examen de décembre 2006</i> )	30
41	Hachage ( <i>Examen de décembre 2006</i> )	30
42	Mariages ( <i>Examen de décembre 2006</i> )	31
43	Tri par arbre ( <i>Examen de décembre 2006</i> )	32
44	Suites 2-3 triées ( <i>Examen de décembre 2005</i> )	33
45	Changement de racine ( <i>Examen de décembre 2005</i> )	33
46	Arbres binaires ( <i>Examen de juin 2005</i> )	34
47	File à double entrée ( <i>Examen de juin 2005</i> )	35
48	Deuxième maximum ( <i>Examen de juin 2005</i> )	35
49	Sortie de pile ( <i>Examen de décembre 2004</i> )	36

<b>50</b>	<b>Parcours</b> ( <i>Examen de décembre 2004</i> )	<b>36</b>
<b>51</b>	<b>Faire tourner les arbres</b> ( <i>Examen de décembre 2004</i> )	<b>37</b>
<b>52</b>	<b>Compter les tas</b> ( <i>Examen de décembre 2003</i> )	<b>38</b>
<b>53</b>	<b>Collection</b> ( <i>Examen de décembre 2003</i> )	<b>39</b>
<b>54</b>	<b>Arbres binaires de recherche</b> ( <i>Examen de décembre 2003</i> )	<b>41</b>
<b>55</b>	<b>Bulles</b> ( <i>Quick octobre 2007</i> )	<b>44</b>
<b>56</b>	<b>Fibonnacci</b> ( <i>Quick octobre 2007</i> )	<b>44</b>
<b>57</b>	<b>Fusion de liste</b> ( <i>Quick octobre 2007</i> )	<b>45</b>
<b>58</b>	<b>Relations antisymétriques</b> ( <i>Quick octobre 2005</i> )	<b>46</b>
<b>59</b>	<b>Tri à 3 couleurs</b> ( <i>Quick octobre 2005</i> )	<b>46</b>
<b>60</b>	<b>File en tableau</b> ( <i>Quick octobre 2005</i> )	<b>46</b>
<b>61</b>	<b>Éliminer les doublons</b> ( <i>Quick novembre 2005</i> )	<b>47</b>
<b>62</b>	<b>Réservation et conflit</b> ( <i>Quick novembre 2005</i> )	<b>48</b>
<b>63</b>	<b>Comprimons</b> ( <i>Quick novembre 2006</i> )	<b>49</b>

## 1 Médiane *(Quick de décembre 2012)*

La médiane d'un ensemble de  $n$  éléments distincts, comparables est l'élément de rang  $\lfloor \frac{n}{2} \rfloor$  dans l'ensemble des éléments triés.

On suppose donné un arbre binaire de recherche de  $n$  éléments,  $n$  est connu.

**Question 1.1 :** Algorithme naïf

Donner un algorithme qui retourne la valeur de la médiane, calculer sa complexité.

On enrichit la structure de donnée "arbre binaire de recherche" en marquant chaque nœud  $x$  de l'arbre par le nombre de nœuds du sous-arbre dont  $x$  est la racine.

**Question 1.2 :** Algorithme raffiné

Expliquer pourquoi cette information permet de trouver la médiane en  $\mathcal{O}(h)$  opérations et écrire l'algorithme correspondant.

## 2 Nombre de Pivert *(Quick de décembre 2012)*

Étant donné un arbre binaire  $T$  on appelle nombre de Pivert de  $T$ ,  $\pi(T)$  le nombre de nœuds de l'arbre n'ayant qu'un seul fils, soit le gauche soit le droit. On note  $n$  le nombre total de nœuds de l'arbre.

**Question 2.1 :**

1. Montrer que

$$\pi(T) \leq n - 1,$$

et que la borne est atteinte pour des arbres particuliers (préciser lesquels).

2. Donner une condition nécessaire et suffisante sur  $n$  pour qu'il existe un arbre de  $n$  nœuds satisfaisant  $\pi(T) = 0$ .
3. Écrire un algorithme récursif qui calcule le nombre de Pivert d'un arbre.
4. Écrire un algorithme itératif qui calcule le nombre de Pivert d'un arbre.
5. Calculer la complexité de ces deux algorithmes.

## 3 Coût d'un algorithme *(Quick d'octobre 2012)*

On se donne un tableau  $T$  de  $n$  éléments distincts comparables deux à deux. On souhaite analyser la complexité de l'algorithme suivant :

```
Truc(T)
  rpter
    XX=Vrai
    pour  $i = 1$  à  $n - 1$  faire
      si  $T[i] > T[i + 1]$  alors
        échanger( $i, i + 1$ )
    XX=Faux
```

**fin si**  
**fin pour**  
**jusqu' XX**

### Question 3.1 : Coût

Calculer le coût de l'algorithme **Truc** en nombre d'appels à la fonction **échange** (justifier votre réponse et donner les ordres de grandeur en utilisant la notation  $\mathcal{O}$ ) :

1. déterminer le coût au mieux et donner un exemple ;
2. déterminer le coût au pire et donner un exemple ;
3. **bonus** donner le coût moyen de l'algorithme.

### Question 3.2 : Synthèse

Déterminer ce que fait cet algorithme, écrire la preuve de l'algorithme et comparer cet algorithme aux autres algorithmes vus en cours et résolvant le même problème.

## 4 Étudiants assidus *(Quick d'octobre 2012)*

À chaque cours, l'enseignant établit une liste des étudiants présents. Le responsable de la formation collecte les listes et les fusionne pour avoir la liste des étudiants "actifs" qui ont fait acte de présence au moins une fois.

On note  $L_1, \dots, L_k$  les listes des étudiants de taille respective  $n_1, \dots, n_k$ . Ces listes sont implantées par chaînage simple et ne sont pas structurées de manière particulière (les étudiants ne sont pas classés par ordre alphabétique).

### Question 4.1 : Fusion

Proposer un algorithme réalisant la fusion des listes en  $\mathcal{O}(n_1 + \dots + n_k)$  opérations.

### Question 4.2 : Affinage

Les étudiants sérieux ont assisté à au moins 80% des cours. Modifier votre algorithme pour établir la liste des étudiants "sérieux", toujours en  $\mathcal{O}(n_1 + \dots + n_k)$ .

## 5 Choix de représentants *(Examen de décembre 2012)*

Une liste d'étudiants est gérée par chaînage simple et circulaire dans un tableau. Chaque étudiant fait partie d'un groupe et on souhaite construire une nouvelle liste (chaînage simple circulaire dans le même tableau) en ne gardant qu'un seul élément de chaque groupe dans la liste (évidemment tous les groupes de la liste initiale doivent être représentés).

### Question 5.1 : Algorithme d'extraction

Écrire un algorithme qui réalise la liste des représentants des groupes.

### Question 5.2 : Coût de l'algorithme

Calculer le coût (comparaison de groupes d'étudiants) au pire et au mieux de cet algorithme en ayant une liste de taille  $n$  et un nombre  $m$  de groupes (avec  $m \leq n$ ). On donnera des exemples "au mieux" et "au pire".

**Question 5.3 :** Amélioration de l'algorithme

En utilisant des structures auxiliaires peut-on améliorer la complexité de cet algorithme ?

## 6 Planification *(Examen de décembre 2012)*

Dans une agence de location de camion (il n'y en a qu'un seul),  $n$  clients souhaitent réserver le camion sur une période donnée. Le client  $i$  demande une réservation sur l'intervalle de temps  $[d_i, f_i]$ . Deux réservations sont incompatibles si leurs intervalles ont une intersection non-vide. L'objectif est d'allouer la ressource en maximisant le nombre clients satisfaits.

**Question 6.1 :** Tentative 1

On trie les réservations par durée croissantes puis on satisfait les clients dans cet ordre en respectant la contrainte de compatibilité.

Montrer par un contre-exemple que cet algorithme ne donne pas nécessairement une allocation optimale.

**Question 6.2 :** Tentative 2

On trie les réservations par date de début croissantes puis on satisfait les clients dans cet ordre en respectant la contrainte de compatibilité.

Montrer par un contre-exemple que cet algorithme ne donne pas nécessairement une allocation optimale.

**Question 6.3 :** Tentative 3

Enfin, on trie les réservations par date de fin croissantes puis on satisfait les clients dans cet ordre en respectant la contrainte de compatibilité.

Montrer sur les 2 contre-exemples précédents que cet algorithme donne une allocation optimale.

**Question 6.4 :** Preuve d'optimalité

Montrer que cet algorithme (glouton) construit une solution optimale.

## 7 Parcours d'arbres *(Examen de décembre 2012)*

On considère des arbres binaires de  $n$  nœuds étiquetés par les entiers de 1 à  $n$ .

**Question 7.5 :** Définition (cours)



Rappeler brièvement et en illustrant sur un exemple ce qu'est un parcours en profondeur d'abord **préfixé** explorant le sous-arbre gauche avant le sous-arbre droit.

**Question 7.6 :** Définition (cours)

Rappeler brièvement et en illustrant sur un exemple ce qu'est un parcours en profondeur d'abord **suffixé** explorant le sous-arbre gauche avant le sous-arbre droit.

**Question 7.7 :** Arbre défini par un parcours

Donner deux arbres ayant 8 nœuds dont le parcours préfixé soit 12345678.

**Question 7.8 :** Arbre défini par un parcours préfixé et un parcours suffixé

Existe-t-il un arbre à 8 sommets dont le parcours préfixé soit 12345678 et le parcours suffixé est 53247681 ?

Même question mais avec le parcours suffixé 43527861 ?

**Question 7.9 :** Existence d'un arbre solution

De façon générale, si on se donne deux permutations des nombres 1 à  $n$ , à quelle condition existe-t-il une arborescence de  $n$  sommets dont les permutations soient les parcours préfixé et suffixé ?

Lorsqu'elle existe, cette solution est-elle unique ?

**Question 7.10 :** Construction

Écrire un algorithme pour reconstruire un arbre à partir de son parcours préfixé et de son parcours suffixé.

## 8 Feuille profonde *(Examen de juin 2012)*

Considérons un arbre binaire  $\mathcal{T}_n$  de  $n$  nœuds étiquetées.

**Question 8.1 :**

Écrire un algorithme qui retourne l'étiquette d'une feuille de plus grande profondeur ainsi que sa profondeur. Donner un exemple d'exécution et une preuve de votre algorithme.

**Question 8.2 :**

Évaluer le coût de l'algorithme au mieux et au pire en nombre d'examen de pour un arbre de  $n$  nœuds. Donner un exemple au mieux et au pire.

**Question 8.3 :**

Comment enrichir la structure de donnée d'arbre binaire de manière à retourner une feuille plus profonde en  $\mathcal{O}(\log n)$ .

**Question 8.4 :**

Supposons que l'arbre  $\mathcal{T}_n$  soit un arbre binaire de recherche (les étiquettes sont dans un ensemble totalement ordonné). Écrire l'algorithme qui retourne la plus petite feuille la plus profonde.

**Question 8.5 :**

Expliquer comment les opérations d'insertion et de suppression sont modifiées pour maintenir la structure enrichie.

## 9 Séquence de lettres *(Examen de juin 2012)*

On considère une séquence  $\mathcal{S}$  de  $n$  bits. Un motif  $M$  est constitué d'un entête 000 suivi de  $k$  bits arbitraires suivi d'une terminaison 111.

**Question 9.1 :** Algorithme de reconnaissance

Écrire un algorithme simple qui calcule dans une séquence  $\mathcal{S}$  les positions éventuelles du motif  $M$ . Donner sa preuve et sa complexité.

**Question 9.2 :** Tables de Hachage

En vous inspirant de l'algorithme de Karp-Rabin vu en cours, proposer une utilisation de table de hachage pour obtenir un algorithme en  $\mathcal{O}(n)$ .

## 10 ADE *(Examen de décembre 2011)*

On suppose que  $n$  cours  $c_1, \dots, c_n$  sont programmés sur des plages horaires données par les fonctions  $debut(c_i)$  et  $fin(c_i)$ .

**Question 10.1 :** Algorithme d'affectation

Écrire un algorithme attribuant une salle à chaque cours, minimisant le nombre total de salles occupées, sachant que 2 cours ne peuvent pas avoir lieu dans la même salle. *Indication : on pourra faire un prétraitement sur les cours avant de réaliser l'allocation.*

**Question 10.2 :** Preuve

Rédiger avec soin la preuve de l'optimalité de votre algorithme et calculer son coût.

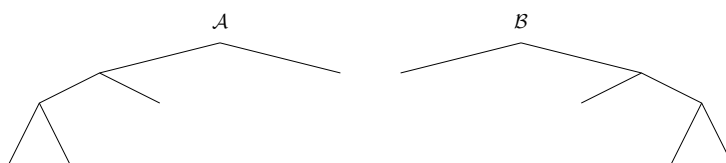
## 11 Arbres binaires complets *(Examen de décembre 2011)*

On considère des arbres binaires complets, c'est à dire que tout nœud est soit une feuille soit un nœud interne avec 2 fils. À un arbre  $\mathcal{A}$  on associe une séquence de bits  $S(\mathcal{A})$  définie de la manière suivante

$$S(\mathcal{A}) = \begin{cases} 1 & \text{si } \mathcal{A} \text{ est réduit à une feuille ;} \\ 0.S(\mathcal{A}_1)S(\mathcal{A}_2) & \text{si } \mathcal{A}_1 \text{ (resp } \mathcal{A}_2) \text{ est le sous-arbre gauche (resp droit) de } \mathcal{A}. \end{cases}$$

### Question 11.1 : Exemples de codage d'arbre

Donner la valeur de  $S(\mathcal{A})$  et  $S(\mathcal{B})$  pour les 2 arbres suivants :



### Question 11.2 : Exemples de décodages d'arbres

Dessiner les arbres correspondant aux mots  $m_1 = 0001111$  et  $m_2 = 0101011$ .

### Question 11.3 : Propriétés

Pour un arbre  $\mathcal{A}$  de taille  $n$ , interpréter le nombre de 0 et de 1 dans  $S(\mathcal{A})$ .

### Question 11.4 :

Toute séquence de bits représente-t'elle un arbre binaire complet ?

### Question 11.5 : Unicité

Montrer que si deux arbres  $\mathcal{A}$  et  $\mathcal{B}$  vérifient  $S(\mathcal{A}) = S(\mathcal{B})$  alors  $\mathcal{A}$  et  $\mathcal{B}$  ont la même structure.

### Question 11.6 : Algorithme de codage

Écrire l'algorithme de codage d'un arbre  $\mathcal{A}$ .

### Question 11.7 : Algorithme de décodage

Écrire l'algorithme de décodage qui à partir d'un code d'arbre reconstruit l'arbre  $\mathcal{A}$  correspondant.

## 12 Un tri linéaire ? (Examen de décembre 2011)

On suppose que  $n$  objets sont à trier selon une clé dont la valeur est un nombre réel compris entre 0 et 1. Pour réaliser le tri on construit une nouvelle structure de donnée constituée d'un tableau  $L$  de  $n$  listes d'objets. Pour  $1 \leq i \leq n$ , les objets de la liste  $L[i]$  ont une clé dans l'intervalle  $[\frac{i-1}{n}, \frac{i}{n}[$  et sont rangés par ordre croissant dans la liste.

L'algorithme de tri consiste à insérer tous les objets dans la structure  $L$ , puis de parcourir la structure pour obtenir la liste des objets triés.

### Question 12.1 : Exemple

Pour  $n = 10$  et le tableau objet/clé suivant, donner la valeur de la structure  $L$  après l'insertion des  $n$  objets.

Objet	A	B	C	D	E	F	G	H	I	J
Clé	0.32	0.05	0.99	0.21	0.82	0.79	0.06	0.66	0.75	0.46

### Question 12.2 : Insertion

Écrire l'algorithme d'insertion d'un objet dans une liste ordonnée par clés croissantes. Calculer le coût de l'algorithme en nombre de comparaisons de clés (au mieux et au pire).

### Question 12.3 : Parcours

Écrire l'algorithme qui parcourt la structure  $L$  et renvoie la liste des objets triés par clé croissante. Donner le coût de ce parcours.

### Question 12.4 : Coût de l'algorithme

Quel est le coût au pire et au mieux de cet algorithme en nombre de comparaisons de clés ?

On suppose maintenant que les valeurs des clés sont réparties uniformément sur  $[0, 1[$ . C'est à dire que tout se passe comme si les clés étaient générées aléatoirement de manière indépendante et de loi uniforme sur  $[0, 1[$ . On note  $N_i$  le nombre d'objets dans la liste  $L[i]$  à la fin de l'insertion des  $n$  objets.

### Question 12.5 : Loi de $N_i$

Quelle est la loi de probabilité de la variable aléatoire  $N_i$  ? Calculer sa moyenne  $\mathbb{E}N_i$  et montrer que  $\mathbb{E}N_i^2 = 2 - \frac{1}{n}$ .

### Question 12.6 : Complexité de l'algorithme

Donner la complexité en moyenne de l'algorithme et commenter ce résultat.

## 13 Feuilles (Quick de novembre 2011)

### Question 13.1 : Arbres et feuilles

Dans un arbre binaire ayant  $n$  nœuds, calculer le nombre minimal de feuilles  $m_n$  et le nombre maximal de feuilles  $M_n$ , le calcul devra être rédigé avec soin.

## 14 Tas ternaire (*Quick de novembre 2011*)

On appelle arbre ternaire un arbre tel que tout nœud possède de 0 à 3 fils, nommés  $fil_s_a$ ,  $fil_s_b$ ,  $fil_s_c$ .

**Question 14.1 :**

1. Calculer la hauteur d'un arbre ternaire ayant  $n$  nœuds.
2. Rappeler la définition d'un arbre ordonné.
3. Proposer une implantation d'un tas ternaire (arbre ordonné tassé à gauche) dans un tableau.
4. Pour  $x$  nœud de l'arbre, écrire les primitives  $Père(x)$ ,  $Fils_a(x)$ ,  $Fils_b(x)$  et  $Fils_c(x)$ .
5. Pour  $T$  tas ternaire et  $x$  nœud, écrire la primitive  $Insérer(x, T)$  qui insère un nouveau nœud dans le tas.
6. Pour  $T$  tas ternaire, écrire la primitive  $Extraire(T)$  qui retourne un nœud d'étiquette maximale et le supprime du tas  $T$ .
7. Calculer le coût des primitives  $Insérer(x, T)$   $Extraire(T)$  en nombre de comparaisons d'étiquettes de nœuds.
8. Faire la trace d'exécution du tri par tas ternaire du tableau [C,A,R,I,B,O,U].
9. Calculer la complexité de l'algorithme de tri par tas ternaire. Comparer cette complexité avec celle du coût du tri par tas (binaire). Commenter votre résultat, peut-on le généraliser ?

## 15 C'est un truc... (*Quick d'octobre 2011*)

**Truc**( $n$ )

pour  $i = 1$  à  $n$  faire

$j = 1$

  tant que  $j * j \leq i$  faire

**Traiter**( $j$ )

$j = j + 1$

  fin tant que

fin pour

**Question 15.1 :** Coût d'un algorithme

Calculer le coût de l'algorithme **Truc** en nombre d'appels à la fonction **Traiter** (on pourra laisser le coût sous la forme  $\sum \dots$ ). Donner l'ordre de grandeur de ce coût (indication : encadrer la somme).

cpt-quest0

## 16 Table ordonnée (*Quick d'octobre 2011*)

On considère un tableau ordonné  $T$  de  $n$  entiers distincts (ces entiers peuvent être négatifs), le tableau  $T$  est indicé de 1 à  $n$ . L'objectif est de construire un algorithme qui renvoie un indice  $i$  tel que  $T[i] = i$  ou 0 s'il n'en existe pas.

**Question 16.2 :** Recherche en table

1. Écrire un algorithme "force brute", qui renvoie un indice  $i$  tel que  $T[i] = i$  ou 0 s'il n'en existe pas. Calculer son coût.
2. Montrer que si  $T[j] > j$  alors il en est de même pour tout  $k \geq j$ .
3. En déduire un algorithme en  $\mathcal{O}(\log n)$ , prouver que le coût est en  $\mathcal{O}(\log n)$ .

cpt-quest0

## 17 Pile, File le retour (*Quick d'octobre 2011*)

**Question 17.3 :** P/F-iles

On dispose des types abstraits classiques **pile** et **file** avec les opérateurs de construction et de transformation classiques.

Écrire un algorithme qui prend en argument une pile  $P$  et renvoie une nouvelle pile  $P'$  contenant les mêmes éléments que  $P$  mais rangés dans l'ordre inverse. (Le sommet de  $P'$  est l'élément au fond de  $P$ ,  $\dots$ , et le sommet de  $P$  se retrouve au fond de  $P'$ ).

Écrire un algorithme qui prend en argument une file  $F$  et renvoie une nouvelle file  $F'$  contenant les mêmes éléments que  $F$  mais rangés dans l'ordre inverse. (Le premier élément de  $F'$  est le dernier élément de  $F$ ,  $\dots$ , et le premier de  $F$  se retrouve le dernier de  $F'$ ).

## 18 Tri par file (*Quick d'octobre 2011*)

Pour trier  $n$  objets distincts on dispose de l'opérateur **fusion** qui prend en argument deux listes triées et renvoie la fusion triée de ces deux listes. On utilise une file de listes que l'on initialise avec  $n$  listes, chacune contenant un des objets à trier.

L'algorithme opère en prenant les deux premières listes de la file, en opérant leur fusion et en enfilant le résultat dans la file. L'algorithme se termine lorsque la file ne contient plus qu'une seule liste, résultat du tri.

**Question 18.1 :** Tri par file

1. Donner les étapes de l'algorithme sur l'exemple  $C, A, R, I, B, O, U, S$ .
2. Écrire l'algorithme de tri par file/fusion.
3. Donner une preuve de votre algorithme.

4. Calculer le coût de votre algorithme.
5. Quel algorithme obtient-on en remplaçant la file par une pile ?
6. Question subsidiaire : Écrire l'algorithme de fusion de listes triées implémentées par des listes circulaires doublement chaînées et calculer son coût.

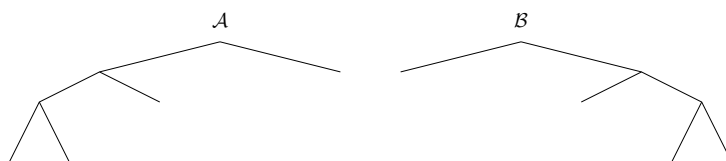
## 19 Arbres binaires complets (*Examen de décembre 2011*)

On considère des arbres binaires complets, c'est à dire que tout nœud est soit une feuille soit un nœud interne avec 2 fils. À un arbre  $\mathcal{A}$  on associe une séquence de bits  $S(\mathcal{A})$  définie de la manière suivante

$$S(\mathcal{A}) = \begin{cases} 1 & \text{si } \mathcal{A} \text{ est réduit à une feuille ;} \\ 0.S(\mathcal{A}_1)S(\mathcal{A}_2) & \text{si } \mathcal{A}_1 \text{ (resp } \mathcal{A}_2) \text{ est le sous-arbre gauche (resp droit) de } \mathcal{A}. \end{cases}$$

### Question 19.1 : Exemples de codage d'arbre

Donner la valeur de  $S(\mathcal{A})$  et  $S(\mathcal{B})$  pour les 2 arbres suivants :



### Question 19.2 : Exemples de décodages d'arbres

Dessiner les arbres correspondant aux mots  $m_1 = 0001111$  et  $m_2 = 0101011$ .

### Question 19.3 : Propriétés

Pour un arbre  $\mathcal{A}$  de taille  $n$ , interpréter le nombre de 0 et de 1 dans  $S(\mathcal{A})$ .

### Question 19.4 :

Toute séquence de bits représente-t'elle un arbre binaire complet ?

### Question 19.5 : Unicité

Montrer que si deux arbres  $\mathcal{A}$  et  $\mathcal{B}$  vérifient  $S(\mathcal{A}) = S(\mathcal{B})$  alors  $\mathcal{A}$  et  $\mathcal{B}$  ont la même structure.

### Question 19.6 : Algorithme de codage

Écrire l'algorithme de codage d'un arbre  $\mathcal{A}$ .

### Question 19.7 : Algorithme de décodage

Écrire l'algorithme de décodage qui à partir d'un code d'arbre reconstruit l'arbre  $\mathcal{A}$  correspondant.

## 20 ADE (Examen de décembre 2011)

On suppose que  $n$  cours  $c_1, \dots, c_n$  sont programmés sur des plages horaires données par les fonctions  $debut(c_i)$  et  $fin(c_i)$ .

### Question 20.1 : Algorithme d'affectation

Écrire un algorithme attribuant une salle à chaque cours, minimisant le nombre total de salles occupées, sachant que 2 cours ne peuvent pas avoir lieu dans la même salle. *Indication : on pourra faire un prétraitement sur les cours avant de réaliser l'allocation.*

### Question 20.2 : Preuve

Rédiger avec soin la preuve de l'optimalité de votre algorithme et calculer son coût.

## 21 Un tri linéaire ? (Examen de décembre 2011)

On suppose que  $n$  objets sont à trier selon une clé dont la valeur est un nombre réel compris entre 0 et 1. Pour réaliser le tri on construit une nouvelle structure de donnée constituée d'un tableau  $L$  de  $n$  listes d'objets. Pour  $1 \leq i \leq n$ , les objets de la liste  $L[i]$  ont une clé dans l'intervalle  $[\frac{i-1}{n}, \frac{i}{n}]$  et sont rangés par ordre croissant dans la liste.

L'algorithme de tri consiste à insérer tous les objets dans la structure  $L$ , puis de parcourir la structure pour obtenir la liste des objets triés.

### Question 21.1 : Exemple

Pour  $n = 10$  et le tableau objet/clé suivant, donner la valeur de la structure  $L$  après l'insertion des  $n$  objets.

Objet	A	B	C	D	E	F	G	H	I	J
Clé	0.32	0.05	0.99	0.21	0.82	0.79	0.06	0.66	0.75	0.46

### Question 21.2 : Insertion

Écrire l'algorithme d'insertion d'un objet dans une liste ordonnée par clés croissantes. Calculer le coût de l'algorithme en nombre de comparaisons de clés (au mieux et au pire).

### Question 21.3 : Parcours

Écrire l'algorithme qui parcourt la structure  $L$  et renvoie la liste des objets triés par clé croissante. Donner le coût de ce parcours.

### Question 21.4 : Coût de l'algorithme

Quel est le coût au pire et au mieux de cet algorithme en nombre de comparaisons de clés ?



On suppose maintenant que les valeurs des clés sont réparties uniformément sur  $[0, 1[$ . C'est à dire que tout se passe comme si les clés étaient générées aléatoirement de manière indépendante et de loi uniforme sur  $[0, 1[$ . On note  $N_i$  le nombre d'objets dans la liste  $L[i]$  à la fin de l'insertion des  $n$  objets.

**Question 21.5 :** Loi de  $N_i$

Quelle est la loi de probabilité de la variable aléatoire  $N_i$ ? Calculer sa moyenne  $\mathbb{E}N_i$  et montrer que  $\mathbb{E}N_i^2 = 2 - \frac{1}{n}$ .

**Question 21.6 :** Complexité de l'algorithme

Donner la complexité en moyenne de l'algorithme et commenter ce résultat.

## 22 Recherche dans un tas (*Examen de décembre 2010*)

**Question 22.1 :** Tas (définition)

Rappeler brièvement la définition de la structure de donnée *Tas* implantée dans un tableau, on supposera que les éléments du tas sont comparables au sens d'une relation d'ordre  $\prec$  et que le *Tas* est construit pour cette relation d'ordre (élément maximum au sommet du tas).

Écrire un algorithme qui vérifie qu'un tableau est un tas. Calculer le nombre de comparaisons entre éléments du tableau effectuées par votre algorithme.

**Question 22.2 :** Position des éléments les plus grands

Dans un tas, où se trouve la valeur la plus grande, la 2<sup>ième</sup> valeur la plus grande, la troisième ?

**Question 22.3 :** Valeur minimale

Écrire un algorithme qui cherche la valeur minimale dans un tas de taille  $n$ . Calculer son coût au pire. Pourquoi cette recherche est, en ordre de grandeur, équivalente à parcourir tout le tas ?

## 23 Inversion (*Examen de décembre 2010*)

Une inversion dans un tableau  $T$  de taille  $n$  est un couple d'indices  $(i, j)$  vérifiant  $i < j$  et  $T[i] > T[j]$ .

**Question 23.1 :** Nombre minimal d'inversions

Donner pour un tableau de taille  $n$  le nombre minimum d'inversions et fournir un exemple de tableau donnant cette valeur minimale.

**Question 23.2 :** Nombre maximal d'inversions

Montrer que le nombre maximal d'inversions est  $\frac{n(n-1)}{2}$ .

**Question 23.3 :** Algorithme bulldozer

Écrire un algorithme qui calcule le nombre d'inversions en  $\mathcal{O}(n^2)$ .

**Question 23.4 :** Raffinement

Écrire un algorithme qui calcule le nombre d'inversions en  $\mathcal{O}(n \log n)$ .

## 24 Plus grand stable (*Examen de décembre 2010*)

On appelle partie **stable** d'un arbre  $A$  un ensemble  $S$  de nœuds de l'arbre qui ne sont pas voisins deux à deux. On suppose que  $A$  est un arbre binaire et qu'à chaque nœud  $x$  on associe un poids  $p(x) \geq 0$ . L'objectif est de trouver un algorithme récursif qui calcule un sous-ensemble stable de poids maximal.

Pour tout nœud  $x$  de l'arbre on note :

- $S(x)$  le stable maximal du sous-arbre de racine  $x$ , on note  $P(x)$  son poids ( $x$  n'est pas forcément dans  $S(x)$ );
- $S'(x)$  le stable maximal du sous-arbre de racine  $x$  et ne contenant pas  $x$ , on note  $P'(x)$  son poids.

**Question 24.1 :** Relation de récurrence

Donner une formule de récurrence entre  $S(x)$ ,  $S'(x)$  et  $S(\text{gauche}(x))$ ,  $S'(\text{gauche}(x))$ ,  $S(\text{droit}(x))$ ,  $S'(\text{droit}(x))$  et de leurs poids respectifs et de  $p(x)$ .

**Question 24.2 :** Algorithme récursif

En déduire un algorithme calculant un ensemble stable maximal. Calculer le coût de votre algorithme.

## 25 Algorithme de couverture (*Examen de décembre 2010*)

On se donne  $n$  points sur l'axe réel  $x_1 < x_2 < \dots < x_n$ . On souhaite trouver le nombre minimal  $K$  tel que  $K$  intervalles  $I_1, \dots, I_K$  de longueur 1 recouvrent tous les points, c'est à dire que tout point appartient, au moins, à l'un de ces intervalles.

**Question 25.1 :** Exemple

Donner un exemple avec  $n$  petit montrant qu'il peut y avoir plusieurs recouvrements minimaux. Donner également un exemple où ce recouvrement est unique.

**Question 25.2 :** Algorithme de recouvrement

Écrire un algorithme glouton qui construit les  $K$  intervalles.

**Question 25.3 :** Preuve

Écrire la preuve de votre algorithme.

Maintenant on se donne un ensemble de  $N$  intervalles  $J_1, \dots, J_N$ , les longueurs de ces intervalles peuvent être différentes et leur union recouvre l'intervalle  $[x_1, x_n]$ .

**Question 25.4 :** Intervalles donnés

Écrire un algorithme qui calcule le nombre minimal d'intervalles nécessaires pour recouvrir  $x_1, x_2, \dots, x_n$ , bien préciser les structures de donnée utilisées. Donner une preuve de votre algorithme ainsi que sa complexité.

## 26 Algorithme de Karp-Rabin (*Examen de décembre 2009*)

On veut rechercher un motif  $M$  de longueur  $m$  dans un texte  $T$  de longueur  $n$ . Les textes sont exprimés dans un alphabet de  $K$  lettres. Le texte et le motif seront donnés dans deux tableaux, ces tableaux seront indicés à partir de 1.

**Question 26.1 :** Test d'égalité de mots

Écrire un algorithme qui teste l'égalité de deux mots. Calculer le coût de votre algorithme en nombre de comparaisons de caractères.

Afin de rechercher plus efficacement un motif, on dispose d'une fonction de hachage  $H$  sur les mots. Si  $A$  est un tableau de lettres  $h(A, i, j)$  représente la valeur de hachage du sous-mot du tableau  $A$  défini de l'indice  $i$  à  $j$  inclus. On suppose que la fonction  $h$  prend des valeurs entières avec  $0 \leq h(A, i, j) \leq H - 1$ .

**Question 26.2 :** Commentaires

Écrire les commentaires 1, 2 et 3 et calculer le coût maximal de cet algorithme en nombre de comparaisons de caractères.

---

**Algorithme 1** Algorithme de Karp-Rabin

---

**Require:**

T texte dans lequel le motif est recherché (tableau de caractères)  
M motif recherché (tableau de caractères)

**Ensure:** Ecrire la liste des indices où démarrent le motif M dans le texte T

```
1: n=taille(T)
2: m=taille(M)
3: hmotif=h(M[1..m])
4: hcourant = h(T[1..m])
5: pour i=1 à n-m+1 faire
6:   si hcourant=hmotif alors
7:     {Commentaire 1 :.....}
8:     si egalite(T[i..i+m-1],M[1..m]) alors
9:       {Commentaire 2 :.....}
10:      Ecrire i
11:    fin si
12:  fin si
13:  hcourant= h(T[i+1..i+m])
14:  {Commentaire 3 :.....}
15: fin pour
```

---

**Question 26.3 :** Propriétés de  $h$ 

Quelles sont les propriétés que  $h$  doit vérifier pour que cet algorithme soit efficace en moyenne ? Calculer dans ce cas le coût moyen de cet algorithme.

On suppose que chaque caractère est codé par un entier et que la fonction  $h$  s'écrit pour un mot  $A = (a_1, \dots, a_m)$

$$h(A[1..m]) = (a_1 + a_2 + \dots + a_m) \text{ modulo } p,$$

avec  $p$  un entier donné.

**Question 26.4 :**  $h$  additive

Expliquer le rôle de l'entier  $p$ . Comment calculer  $h(T[i..i+m-1])$  en fonction de

$h(T[i-1..i+m-2])$ , en déduire que la ligne 13 de l'algorithme ne nécessite que deux consultations du tableau  $T$ .

On se donne l'alphabet  $\{A, B, C, D\}$  avec le codage standard  $\{0, 1, 2, 3\}$ . Soit

$$T = ABCADBACABAC, M = CAB.$$

**Question 26.5 :** Exemple

Dérouler l'algorithme pour  $p = 9$  et évaluer le nombre de comparaisons de caractères effectuées. Quel inconvénient voyez-vous à cette fonction  $h$  ?

On considère maintenant une nouvelle fonction  $h_1$  construite sur le mot  $A$  par

$$h_1(A[1..m]) = (a_1.d^{m-1} + a_2.d^{m-2} + \dots + a_{m-1}.d^1 + a_m.d^0) \text{ modulo } p,$$

avec  $d$  un entier donné (base).

**Question 26.6 :**  $h$  polynomiale

Expliquer le rôle de l'entier  $d$ . Comment calculer  $h(T[i..i + m - 1])$  en fonction de

$h(T[i - 1..i + m - 2])$ , en déduire que la ligne 13 de l'algorithme ne nécessite que deux consultations du tableau  $T$ .

**Question 26.7 :** Exemple

Dérouler l'algorithme sur le même exemple ci-dessus pour  $p = 9$  et  $d = 4$ . Évaluer le nombre de comparaisons de caractères effectuées. Commenter votre résultat.

## 27 Recherche des $k$ plus grands éléments (*Examen de décembre 2009*)

**Question 27.1 :** Tableau non trié (méthode 1)

En vous inspirant du tri par sélection écrire un algorithme qui par des échanges successifs positionne les  $k$  plus grands éléments dans les  $k$  premières cases du tableau. Évaluer le coût de votre algorithme en nombre de comparaisons d'éléments.

**Question 27.2 :** Arbre binaire de recherche (méthode 2)

Rappeler brièvement la définition d'un arbre binaire de recherche. En supposant les  $n$  éléments dans un ABR, écrire l'algorithme de recherche des  $k$  plus grands éléments. Évaluer le coût de votre algorithme en nombre de comparaisons d'éléments.

**Question 27.3 :** Tas (méthode 3)

Rappeler brièvement la définition de la structure de donnée *Tas*. En supposant les  $n$  éléments dans un tas, écrire l'algorithme de recherche des  $k$  plus grands éléments. Évaluer le coût de votre algorithme en nombre de comparaisons d'éléments.

**Question 27.4 :** Synthèse

Faire la synthèse des résultats obtenus pour les 3 structures de donnée, on précisera en particulier le coût en nombre d'opérations et en espace mémoire utilisé (si on utilise des structures de données auxiliaires).

## 28 Feuilles (*Examen de décembre 2009*)

On définit un arbre binaire de manière récursive par : *un arbre est soit un arbre vide noté  $\varepsilon$ , soit un couple d'arbres  $(a_1, a_2)$* . La taille d'un arbre binaire se définit également de manière récursive par la taille d'un arbre vide est 0 et la taille de  $(a_1, a_2) = 1 + \text{taille}(a_1) + \text{taille}(a_2)$

### Question 28.1 : Sous-arbres vides

Montrer que le nombre de sous-arbres vides d'un arbre de taille  $n$  est égal à  $n + 1$ .

### Question 28.2 :

Donner la définition d'une feuille dans un arbre binaire. Montrer que le nombre de feuilles est inférieur ou égal à  $\frac{n+1}{2}$  et qu'il y a égalité si et seulement si chaque sous-arbre de l'arbre est soit une feuille, soit a deux fils.

### Question 28.3 : Comptage de feuilles

Montrer que le nombre de feuilles d'un arbre est égal au nombre de sous-arbres de ayant 2 sous-arbres non-vides, plus un.

## 29 Un peu de couleur (*Examen de juin 2009*)

On considère un tableau  $T$  de  $n$  objets coloriés et on veut connaître le nombre de couleurs différentes des objets du tableau, puis de sélectionner un objet de chaque couleur. Pour cela on dispose d'une fonction  $\text{couleur}(o)$  qui renvoie la couleur de l'objet  $o$  et d'un opérateur booléen  $==$  permettant de tester l'égalité de deux couleurs.

### Question 29.1 : Nombre de couleurs

Écrire un algorithme qui prend en argument un tableau  $T$  d'objets coloriés et renvoie le nombre de couleurs différentes. On justifiera les différentes structures de donnée auxiliaires dont on aura besoin et on calculera le coût de cet algorithme (en précisant les opérateurs considérés).

### Question 29.2 : Liste d'objets de couleur différentes

Ecrire un algorithme qui prend en argument un tableau  $T$  d'objets coloriés et renvoie une liste  $L$  d'objets du tableau  $T$  telle que toute couleur du tableau  $T$  est portée par un et un seul objet de  $L$ . On précisera les structures de donnée utilisées et on analysera le coût de cet algorithme.

## 30 Est-ce un tas ? (*Examen de juin 2009*)

### Question 30.1 : Qu'est ce qu'un tas ?

Donner la définition de la structure de donnée *Tas*. (question de cours)

**Question 30.2 : Vérification**

Écrire un algorithme de vérification qui prend en argument un tableau de taille  $n$  et vérifie si ce tableau est un tas

**Question 30.3 : Exemple d'exécution**

Donner deux exemples d'exécution sur des tas judicieusement choisis pour illustrer votre algorithme.

**Question 30.4 : Coût de l'algorithme**

Calculer le coût de votre algorithme et expliquer pourquoi il ne peut être inférieur à  $n - 1$  comparaisons pour vérifier un tas de taille  $n$ .

### 31 Anagrammes de phrase (*Examen de juin 2008*)

Une phrase est donnée par une liste de mots simplement chaînée. On souhaite déterminer si deux phrases sont des anagrammes, c'est à dire sont composées exactement des mêmes mots.

Par exemple les phrases

*Le voleur porte la montre au bijoutier* et *Le bijoutier montre la porte au voleur* sont des phrases anagramme.

**Question 31.1 : Algorithme**

Ecrire un algorithme qui réalise le test d'anagramme.

**Question 31.2 : Coût de l'algorithme**

Calculer le coût (comparaison de couleurs) au pire et au mieux de votre algorithme en ayant deux phrase de taille  $n$ .

**Question 31.3 : Amélioration de l'algorithme**

Construire une structure auxiliaire et l'algorithme pour obtenir une complexité en moyenne de  $\mathcal{O}(n)$ .

### 32 Tableau trié (*Examen de juin 2008*)

Un tableau  $m \times n$  est dit de *Young* si tous les éléments d'une même ligne sont triés de gauche à droite et tous ceux d'une même colonne de haut en bas. Les cases ne contenant pas d'élément sont supposées contenir  $+\infty$ . Le tableau  $Y$  suivant donne un tableau de *Young*  $5 \times 4$  contenant les éléments  $\{9, 16, 3, 2, 4, 8, 5, 14, 12\}$ .

$Y =$

2	8	12	14
3	9	16	$+\infty$
4	$+\infty$	$+\infty$	$+\infty$
5	$+\infty$	$+\infty$	$+\infty$
$+\infty$	$+\infty$	$+\infty$	$+\infty$

**Question 32.1 : Propriétés élémentaires**

Montrer que si  $Y_{m,n} < +\infty$  alors le tableau est plein. Où se trouve le plus petit élément d'un tableau non vide ? Montrer que si  $Y_{1,1} = +\infty$  alors le tableau ne contient pas d'élément.

**Question 32.2 : Exemple d'insertion**

On insère le nombre 6 dans la case  $(2, 4)$  du tableau de l'exemple. Le "remonter" à sa place en ne faisant que des échanges avec des cases voisines

**Question 32.3 : Modification (1)**

On suppose que l'on change la valeur de la case  $(i, j)$  avec une valeur  $v < Y_{i,j}$ . Ecrire une fonction `Remonte(i, j, v)` qui "remonte" l'élément actuellement dans la case  $(i, j)$  pour le mettre à sa place ; le coût de cette fonction doit être  $O(n + m)$ .

**Question 32.4 : Modification (2)**

On suppose maintenant que l'on change la valeur de la case  $(i, j)$  avec une valeur  $v > Y_{i,j}$ . Sans écrire l'algorithme, donner l'idée d'une fonction `Descente(i, j, v)` qui "descend" l'élément actuellement dans la case  $(i, j)$  pour le mettre à sa place ; quel est le coût de cette fonction ?

**Question 32.5 : Insertion**

Donner un algorithme qui insère un nouvel élément dans un tableau de Young  $m \times n$  non plein. Quel est son coût en pire cas ?

**Question 32.6 : Suppression**

Donner un algorithme qui supprime le plus petit élément du tableau et qui utilise une des fonctions de la question précédente.  
Quel est le coût en pire cas de cette suppression du minimum ?

**Question 32.7 : Tri**

En utilisant un tableau de Young  $n \times n$  montrer comment on peut trier  $n^2$  éléments en  $O(n^3)$ . On peut répondre à cette question sans avoir répondu à aucune autre question de ce problème.

### 33 Extraction de couleurs (*Examen de décembre 2007*)

Une liste d'éléments colorés est gérée par chaînage simple et circulaire dans un tableau. On souhaite construire une nouvelle liste (chaînage simple circulaire dans le même tableau) en ne



gardant qu'un seul élément de chaque couleur dans la liste (évidemment on garde toutes les couleurs de la liste initiale).

**Question 33.1 :** Algorithme d'extraction

Ecrire un algorithme qui réalise l'extraction des couleurs.

**Question 33.2 :** Coût de l'algorithme

Calculer le coût (comparaison de couleurs) au pire et au mieux de cet algorithme en ayant une liste de taille  $n$  et un nombre  $m \leq n$  de couleurs. On donnera des exemples "au mieux" et "au pire".

**Question 33.3 :** Amélioration de l'algorithme

En utilisant des structures auxiliaires peut-on améliorer la complexité de cet algorithme ?

### 34 Premier et second (*Examen de décembre 2007*)

On dispose de  $n$  éléments distincts d'un espace ordonné placés dans un tableau  $T$  indicé de 0 à  $n - 1$ .

**Question 34.1 :** Algorithme premier et second

Ecrire un algorithme qui calcule simultanément l'indice du plus grand élément et du deuxième plus grand élément du tableau  $T$ .

**Question 34.2 :** Coût de l'algorithme

Calculer le coût au pire de cet algorithme pour une taille  $n$  du tableau.

On suppose que  $n = 2^k$  et on utilise la technique des tournois pour déterminer le champion. C'est à dire que l'on compare les éléments en position  $T[2i]$  et  $T[2i + 1]$ . On obtient un maximum qui est placé dans un deuxième tableau  $T'$  de taille  $\frac{n}{2} = 2^{k-1}$ .

$$T'[i] = \max\{T[2i], T[2i + 1]\}.$$

On recommence alors la même procédure avec ce nouveau tableau. Le maximum est obtenu pour  $k = 0$

**Question 34.3 :** Algorithme du tournoi

Exécuter cet algorithme sur le tableau initial  $T$

$$T = \begin{array}{|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|} \hline 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \\ \hline f & b & g & e & l & n & i & m & k & h & j & a & p & c & o & d \\ \hline \end{array}$$

**Question 34.4 :** Où est l'arbre ?

Expliquer la structure d'arbre sous jacente à cet algorithme. Comment l'implanter dans un tableau de taille  $2n - 1$  ?

**Question 34.5 :** Le second

Calculer le coût nécessaire à la construction de l'arbre. Donner l'algorithme de calcul du second et montrer que le coût de ce calcul est en  $\mathcal{O}(\log n)$ .

## 35 Arbres pondérés (*Examen de décembre 2007*)

L'objectif de ce problème est de construire un arbre binaire de recherche (ABR) optimal sur  $n$  étiquettes, que par simplicité nous fixerons à  $\{1, 2, 3, \dots, n\}$ .

**Question 35.1 :** Enumération

Pour  $n = 1, n = 2, n = 3$  énumérer tous les arbres binaires de recherche possibles.

Soit  $c_n$  le nombre d'arbres binaires de recherche sur les étiquettes  $\{1, 2, 3, \dots, n\}$ .

**Question 35.2 :** Equation de récurrence

Montrer que les  $c_n$  vérifient l'équation de récurrence

$$c_0 = c_1 = 1;$$
$$c_n = \sum_{k=0}^{n-1} c_k c_{n-k}.$$

En déduire la valeur de  $c_4$ .

On suppose l'ABR construit et on effectue uniquement des recherches d'éléments. On note  $f_i$  le nombre de fois que l'élément recherché portant l'étiquette  $i$  est recherché. Le coût  $\bar{C}$  de l'algorithme correspond à la profondeur des étiquettes pondérées par les  $f_i$

$$\bar{C}(n) = \sum_{k=1}^n f_k \cdot \text{profondeur}(k).$$

L'objectif est alors de trouver un ABR optimal qui minimise  $\bar{C}$ .

**Question 35.3 :** Exemple d'arbre optimal

Pour  $n = 4$  et  $f_1 = 1, f_2 = 9, f_3 = 6, f_4 = 4$ , donner un ABR optimal.

Pour  $i \leq j$  on note  $t[i, j]$  le coût de l'arbre optimal pour les étiquettes  $i, \dots, j$  affectées de la pondération  $f_i, \dots, f_j$ .

**Question 35.4 :** Equation d'optimalité

Montrer que pour  $i \leq j$

$$t[i, j] = \max_{i \leq k \leq j} \{t[i, k-1] + f_k + t[k+1, j] + \sum_{l=i}^{k-1} f_l + \sum_{l=k+1}^j f_l\},$$

avec comme convention  $t[i, i-1] = 0$ .

**Question 35.5 :** Algorithme de calcul

Ecrire l'algorithme de calcul des  $t[i, j]$ . Calculer l'ordre de grandeur du coût de cet algorithme.

**Question 35.6 : Arbre optimal**

A partir de cette matrice écrire l'algorithme de construction d'un arbre binaire de recherche optimal.

**36 Matrice (Examen de juin 2007)**

On rappelle qu'une matrice d'éléments de taille  $n \times n$  est un ensemble de  $n^2$  éléments associés à une position dans un tableau à 2 dimensions d'éléments. Pour simplifier l'écriture dans l'exercice on supposera que les éléments sont des entiers. On se donne également une opération binaire sur les éléments notée  $+$ .

**Question 36.1 : Type matrice**

Proposer, en C, une implémentation du type matrice à l'aide d'un tableau d'éléments, on précisera la manière d'accéder à un élément en position (ligne  $i$ , colonne  $j$ ). On supposera que les indices de ligne et de colonne vont de 0 à  $n - 1$  et que  $n$  est fixé au préalable.

**Question 36.2 : Somme de matrices**

Ecrire l'algorithme qui réalise la somme de 2 matrices (éléments par élément). Calculer, en fonction de  $n$ , le coût de cet algorithme en nombre de sommes d'éléments.

Dans le cas où la matrice est creuse, c'est à dire contient un grand nombre d'éléments nuls, on stocke la matrice sous la forme d'un ensemble de triplets (*valeur, ligne, colonne*). Par exemple la matrice

$$A = \begin{pmatrix} 0 & 1 & 2 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{pmatrix},$$

est représentée par  $\{(1, 0, 1), (2, 0, 2), (1, 1, 0), (2, 2, 4), (1, 3, 1), (1, 3, 2), (1, 4, 1)\}$ .

**Question 36.3 : Changement de représentation**

Ecrire le type correspondant à la nouvelle représentation d'une matrice. Quelles sont les contraintes vérifiées par ce type ? Ecrire l'algorithme qui permet le passage d'une représentation de matrice pleine à sa représentation creuse.

**Question 36.4 : Somme de matrices creuses**

Etant données 2 matrices  $A$  et  $B$  de même dimension  $n$  et de nombre d'éléments non nuls respectifs  $m$  et  $m'$ , écrire l'algorithme qui réalise la somme de ces deux matrices au format creux.

**Question 36.5 : Coût de la somme**

Calculer le coût au mieux et au pire de cet algorithme et comparer avec le résultat obtenu en question 1.2).

## 37 B-arbre (*Examen de décembre 2006 et juin 2007*)

On rappelle qu'un B-arbre est un arbre enraciné vérifiant les propriétés suivantes :

1. Chaque nœud  $x$  contient un nombre  $n[x]$  de clés portées par le nœud  $x$  et une liste ordonnée  $n[x]$  de clés  $cle_1(x) \leq \dots \leq cle_{n(x)}(x)$  ;
2. Si le nœud interne  $x$  contient  $n[x]$  clés, il contient également  $n[x] + 1$  pointeurs vers des B-arbres,  $B_0(x), \dots, B_{n(x)}(x)$ , toute clé  $c$  du sous-arbre  $B_i(x)$  vérifie  $cle_{i-1}(x) \leq c \leq cle_i(x)$  ;
3. Toutes les feuilles sont à la même profondeur, qui est égale à  $h$  hauteur de l'arbre ;
4. Pour un degré  $d \geq 2$  (degré minimum) donné du B-arbre, tout nœud autre que la racine contient au moins  $d - 1$  clés. Tout nœud interne contient donc au moins  $d$  fils. Si l'arbre n'est pas vide, la racine contient au moins une clé.
5. Tout nœud peut contenir au plus  $2d - 1$  clés. Donc tout nœud interne contient au plus  $2d$  fils.

**Question 37.1 :** Nombre d'étiquettes : hauteur  $h = 0, 1$  ou  $2$

Dans cette question le degré minimal du B-arbre est fixé à  $d = 2$ . Donner le nombre minimal et maximal d'étiquettes portées par un B-arbre de hauteur  $0, 1$  ou  $2$ . On donnera un exemple de chaque arbre pour  $h = 0$  et  $h = 1$ .

**Question 37.2 :** Nombre maximal d'étiquettes

Calculer le nombre maximal d'étiquettes portées par un B-arbre de degré  $d$  et de hauteur  $h$ .

**Question 37.3 :** Nombre minimal d'étiquettes

Calculer le nombre minimal d'étiquettes portées par un B-arbre de degré  $d$  et de hauteur  $h$ .

**Question 37.4 :** Recherche

En déduire que la recherche d'une clé dans un B-arbre de degré  $d$  portant  $n$  étiquettes se fait en au pire  $\mathcal{O}(\log n)$  comparaisons de clés.

## 38 Tous en rang (*Examen de juin 2007*)

On considère un tableau de  $n$  objets d'objets. Chaque objet dispose d'une clé (unique) et l'espace des clés est totalement ordonné et on veut implémenter une primitive *rang* qui prend en argument un tableau  $T$  de  $n$  objets, un indice  $i \in \{0 \dots n - 1\}$  et qui renvoie l'objet dont la clé est la  $i$ -ème dans l'ordre des clés des objets du tableau.

**Question 38.1 :** Tri du tableau

Ecrire l'algorithme de tri par insertion du tableau d'objets. Quel est le coût de cet algorithme en nombre d'échanges d'objets ? Quel est le coût de la primitive de recherche de l'élément de rang  $i$  ?

Quels avantages et désavantages voyez-vous à cette implémentation ?

**Question 38.2 :** Tri par segmentation

Ecrire l'algorithme de tri par segmentation (Quicksort). Quel est le coût au pire de cet algorithme ? Quel est son coût moyen ? (énoncer le résultat, ne pas le redémontrer)

**Question 38.3 :** Recherche du rang  $i$

Modifier l'algorithme de tri par segmentation pour obtenir directement l'élément de rang  $i$ .

**Question 38.4 :** Coût de la recherche

Quel est le coût moyen de cet algorithme ? (on pourra dessiner l'arbre des appels associés)

Quels avantages et désavantages voyez-vous à cette implémentation ?

## 39 Tous en rang (autre version) (*Examen de décembre 2006*)

On souhaite construire une structure de donnée sur un ensemble dynamique d'objets. Chaque objet dispose d'une clé (unique) et l'espace des clés est totalement ordonné. En plus des primitives classiques de création de la structure vide, de l'insertion, de la suppression et du test à vide, on veut implémenter une primitive qui recherche le  $i^{\text{ème}}$  objet dans l'ordre des clés.

**Question 39.1 :** Tableau trié

On utilise un tableau trié dans l'ordre des clés et tassé à gauche (sans trous). Calculer le coût maximum en mouvements d'éléments du tableau associé aux opérations d'insertion et de suppression. Quel est le coût de la primitive de recherche de l'élément de rang  $i$  ?

**Question 39.2 :** Arbre binaire de recherche

On utilise un arbre binaire de recherche. On suppose que les objets ont été insérés dans l'arbre selon un ordre arbitraire (aléatoire et uniforme). Donner sans le démontrer, le coût moyen en nombre de comparaisons de clés associé aux opérations d'insertion et de suppression.

Proposer un algorithme basé sur le parcours infixé permettant d'obtenir l'élément de rang  $i$ . On présentera d'abord cet algorithme sous forme itérative. On pourra utiliser une structure de donnée auxiliaire en précisant sa spécification. Donner ensuite cet algorithme sous forme récursive.

**Question 39.3 :** Arbre binaire de recherche enrichi

On enrichit la structure de donnée de l'arbre binaire de recherche en ajoutant à chaque nœud  $x$  la *taille* du sous-arbre défini par ce nœud. La fonction *taille* se calcule donc récursivement par

$$taille(x) = 1 + taille(gauche(x)) + taille(droit(x)),$$

avec la *taille* d'un arbre vide égale à 0.

Proposer un nouvel algorithme de recherche de l'élément de rang  $i$  et calculer son coût moyen.

**Question 39.4 :** Arbre binaire de recherche enrichi (suite)

Modifier l'opération d'insertion dans l'arbre binaire de recherche pour maintenir cohérentes les valeurs de *taille* associées à chaque nœud. Quel est le surcoût associé au maintien des valeurs de *taille* ?

Dire rapidement comment est modifiée l'opération de suppression.

Faire un commentaire sur l'apport de l'enrichissement de la structure sur les coûts associés à la structure de donnée.

## 40 Ensemble (*Examen de décembre 2006*)

On considère un ensemble  $\mathcal{E}$  d'objets distincts. Chaque objet porte un identificateur unique (chaîne de caractères).

**Question 40.1 :** Structure de donnée

Proposer une structure de donnée permettant de représenter un sous-ensemble  $\mathcal{F}$  de  $\mathcal{E}$ .

**Question 40.2 :** Test d'égalité

Etant donné  $\mathcal{F}$  et  $\mathcal{G}$  deux sous-ensembles de  $\mathcal{E}$ . Proposer un algorithme efficace qui teste l'égalité de ces sous-ensembles. On justifiera l'efficacité de l'algorithme.

**Question 40.3 :** Analyse

Commenter votre choix de structure de donnée en prenant en compte le coût des opérations d'ajout ou de suppression d'un objet d'un sous-ensemble.

## 41 Hachage (*Examen de décembre 2006*)

**Question 41.1 :**

Montrer comment on réalise l'insertion des clés (entiers) 5, 28, 19, 15, 20, 33, 12, 17, 10 dans une table de hachage où les collisions sont résolues par liste chaînée et l'insertion se fait en fin de liste. On suppose que la table est de taille 9 et que la fonction de hachage est  $h(k) = k \bmod 9$ . Quel est coût associé à une insertion ?

**Question 41.2 :**

On souhaite maintenant que les éléments chaînés soient chaînés dans l'ordre croissant de leur clé. Modifier les algorithmes d'insertion et de suppression dans la table.

**Question 41.3 :**

Comment les coûts moyens de recherche, d'insertion et suppression sont-ils modifiés ? Commenter votre résultat.

## 42 Mariages (*Examen de décembre 2006*)

On considère deux listes  $L_1$  et  $L_2$  d'éléments colorés. On veut construire une liste de couples d'éléments le premier dans  $L_1$  le deuxième dans  $L_2$  et possédant la même couleur (un élément d'une liste se retrouvera dans au plus un couple). Par exemple, pour les listes :

$$L_1 = [(a, \text{rouge}), (b, \text{rouge}), (c, \text{vert}), (d, \text{bleu}), (e, \text{vert}), (f, \text{rouge}), (g, \text{vert}), (h, \text{vert}), (i, \text{jaune}), (j, \text{jaune})]$$

$$L_2 = [(A, \text{vert}), (B, \text{rouge}), (C, \text{jaune}), (D, \text{jaune}), (E, \text{rose}), (F, \text{rouge}), (G, \text{bleu}), (H, \text{vert})]$$

Une solution maximale possible est

$$L_3 = [((f, B), \text{rouge}), ((i, D), \text{jaune}), ((c, H), \text{vert}), ((e, A), \text{vert}), ((a, F), \text{rouge}), ((d, G), \text{bleu})]$$

Les listes de célibataires restants sont

$$L'_1 = [(b, \text{rouge}), (h, \text{vert}), (j, \text{bleu})] \text{ et } L'_2 = [(C, \text{jaune}), (E, \text{rose})]$$

La liste  $L_3$  est maximale au sens qu'il n'existe plus de mariages possibles. On admet les propriétés suivantes :

- il peut exister plusieurs solutions maximales ;
- toute solution maximale a le même nombre d'éléments.

**Question 42.1 :**

Ecrire un algorithme qui réalise un mariage des listes (un parmi tous ceux qui sont possibles).

On note  $n_1$  et  $n_2$  la taille des listes  $L_1$  et  $L_2$ .  $n_c$  est le nombre de couleurs, à priori inconnu.

**Question 42.2 :**

Calculer le coût de votre algorithme et proposer un cas "pire".

**Question 42.3 :**

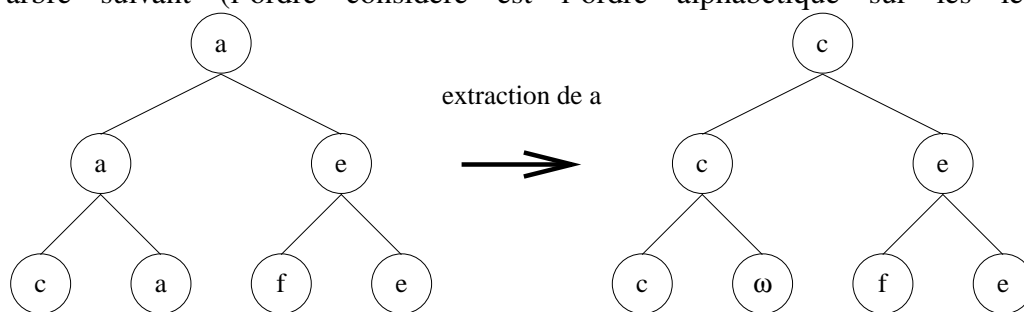
En supposant que le nombre de couleurs  $n_c$  est égal à 3 et en utilisant une structure de tableau, proposer un algorithme qui résolve ce problème en temps linéaire (nombre de comparaisons), c'est à dire

$$\text{Coût} \leq \alpha(n_1 + n_2),$$

où  $\alpha$  est un coefficient que l'on précisera.

### 43 Tri par arbre (*Examen de décembre 2006*)

Le tri par sélection consiste à extraire un élément minimum de l'ensemble des éléments à trier puis à recommencer la procédure avec les éléments restants. On considère une variante de ce tri qui utilise une structure auxiliaire d'arbre binaire étiqueté complet. Par exemple le tableau [c a f e] sera représenté par l'arbre suivant (l'ordre considéré est l'ordre alphabétique sur les lettres) :



Les feuilles sont étiquetées par les éléments du tableau, l'étiquette d'un nœud interne de l'arbre est égale au minimum de l'étiquette de ses deux fils. Après extraction du minimum on remplace dans l'arbre l'étiquette de la feuille du minimum par un symbole, ici  $\omega$ , qui est plus grand que toutes les éléments à trier et on actualise l'arbre. Pour simplifier on suppose que le nombre d'éléments à trier est  $n = 2^k$ .

#### Question 43.1 :

Donner la suite des arbres obtenus lors de l'exécution de cet algorithme sur le tableau [ l a n f e u s t ].

On utilise la structure de donnée "tas" pour représenter cet arbre. On rappelle que dans un tas, l'arbre est représenté par un tableau, la racine est en première position du tableau et les fils du nœud  $i$  sont en position  $2i$  et  $2i + 1$ .

#### Question 43.2 :

Donner le tas initial pour l'exemple de la question précédente.

#### Question 43.3 :

Ecrire l'algorithme d'initialisation du tas pour un tableau donné de taille  $n = 2^k$  et calculer son coût en nombre d'affectations dans le tableau. Ecrire l'algorithme d'actualisation du tas lors de l'extraction du minimum et calculer son coût en nombre d'affectations dans le tableau.



**Question 43.4 :**

Combien de fois au total compare-t-on  $\omega$  et  $\omega$  en triant tous les éléments ? Calculer le coût de cet algorithme de tri et commenter ce résultat. Comment modifier cet algorithme pour traiter les cas où  $n$  n'est pas une puissance de 2.

**44 Suites 2-3 triées (Examen de décembre 2005)**

Au cours de l'exécution d'algorithmes de tri tels que le *shell-sort*, on génère des tableaux partiellement triés. On dit qu'un tableau est 2-3-trié s'il contient tous les éléments initiaux et si chaque élément est à sa place ou dans une place voisine. Le  $i$ -ième élément dans un tableau  $T$  de taille  $n$  est soit dans la case  $T[i - 1]$ ,  $T[i]$  ou  $T[i + 1]$  excepté aux bords (le plus petit élément est dans  $T[1]$  ou  $T[2]$  le plus grand dans  $T[n - 1]$  ou  $T[n]$ ). Par exemple, la suite suivante est 2-3 triée :

Position	1	2	3	4	5	6	7	8	9
Elément	2	1	4	3	5	6	8	7	9

**Question 44.1 :** Enumération de tableaux 2-3-triés

Dans les cas  $n = 1$ ,  $n = 2$ ,  $n = 3$ ,  $n = 4$  et  $n = 5$ , donner tous les tableaux 2-3 triés possibles.

**Question 44.2 :** Comptage

Donner, en la justifiant, l'équation de récurrence vérifiée par  $c(n)$  nombre de tableaux 2-3-triés de taille  $n$ . (Bonus : calculer  $c(n)$  et commenter votre résultat.)

**Question 44.3 :** Algorithme de réarrangement

Ecrire un algorithme qui prend en argument un tableau 2-3-trié de taille  $n$  et réarrange tous ses éléments par ordre croissant. Prouver votre algorithme et calculer son coût.

**45 Changement de racine (Examen de décembre 2005)**

On considère un arbre binaire de recherche de racine  $r$  dont toutes les étiquettes sont différentes. On notera, pour un nœud  $x$ ,  $G(x)$  (respectivement  $D(x)$ ) le sous-arbre gauche (respectivement droit) de  $x$ .

On souhaite transformer l'ABR de manière à ce qu'un nœud  $a$  donné soit racine du nouvel ABR. Cela revient à construire deux arbres binaires de recherche  $G_a$  et  $D_a$ . Les nœuds de  $G_a$  ont une étiquette inférieure à l'étiquette de  $a$  et ceux de  $D_a$  une étiquette supérieure.

**Question 45.1 :**

Proposer une solution lorsque  $a$  est la racine.

On suppose que l'étiquette du nœud  $a$  est inférieure à l'étiquette de la racine  $r$ .

**Question 45.2 :**

Montrer que les nœuds de  $G_a$  sont des nœuds contenus dans  $G(r)$ .

Montrer que les nœuds de  $D(r)$  et  $r$  sont des nœuds contenus dans  $D_a$ .

En déduire une construction récursive de l'ensemble  $D_a$ .

**Question 45.3 :**

Si l'étiquette de  $r$  est plus petite que l'étiquette de  $a$  proposer une construction analogue de l'arbre  $G_a$ .

**Question 45.4 :**

Ecrire l'algorithme qui, à partir d'un arbre binaire de recherche de racine  $r$  et d'un nœud  $a$  renvoie un arbre binaire de recherche de racine  $a$  contenant tous les nœuds de l'arbre initial.

**Question 45.5 :**

Calculer le coût de votre algorithme et donner un exemple de cas "pire".

**Question 45.6 :**

Donner un exemple d'arbre de binaire de recherche et donner la trace d'exécution de votre algorithme sur celui-ci.

## 46 Arbres binaires (*Examen de juin 2005*)

On note  $n$  le nombre de nœuds d'un arbre binaire,  $f$  son nombre de feuilles et  $h$  sa hauteur. On rappelle que la hauteur d'un arbre réduit à 1 nœud est 0 et que les feuilles sont les nœuds de l'arbre dont le fils gauche et le fils droit sont vides.

**Question 46.1 :**

Quelle est la hauteur maximale d'un arbre de  $n$  nœuds ?

**Question 46.2 :**

Quel est le nombre maximum de feuilles d'un arbre de hauteur  $h$  ?

**Question 46.3 :**

Quel est le nombre maximum de nœuds d'un arbre de hauteur  $h$  ?

**Question 46.4 :**

Quelle est la hauteur minimale d'un arbre de  $n$  nœuds ?

**Question 46.5 :**

Montrer que le nombre de branches vides (nombre de fils gauches et de fils droits vides) d'un arbre à  $n$  nœuds est égal à  $n + 1$ .

**Question 46.6 :**

Montrer que le nombre de feuilles d'un arbre à  $n$  nœuds est inférieur ou égal à  $\frac{n+1}{2}$  et que l'on a égalité si et seulement si chaque nœud de l'arbre est une feuille ou bien a 2 fils.

## 47 File à double entrée (*Examen de juin 2005*)

On veut construire une structure de donnée permettant la gestion d'une file à double entrée. Les opérations d'insertion et de retrait peuvent se faire en tête ou en queue de la file.

**Question 47.1 :**

Proposer une structure de donnée FDE (écrite en C) implémentant ce type de file dans un tableau de taille fixe  $N$ .

**Question 47.2 :**

Ecrire une implémentation de la fonction `Test_File_Vide(f)`.

**Question 47.3 :**

Ecrire une implémentation des procédures `Inserer_en_tete(e, f)`, `Inserer_en_queue(e, f)`, `Retirer_en_tete(f)` et `Retirer_en_queue(f)`.

## 48 Deuxième maximum (*Examen de juin 2005*)

On suppose donné un tableau d'éléments comparables par une relation d'ordre  $\leq$  et tous distincts. On note  $n$  la taille de ce tableau et on dispose uniquement d'un opérateur de comparaison entre 2 éléments.

**Question 48.1 :**

Ecrire un algorithme de recherche de l'indice dans le tableau de l'élément maximal pour la relation  $\leq$ . Calculer son coût (en nombre de comparaisons d'éléments).

**Question 48.2 :**

Ecrire un algorithme de recherche de l'indice dans le tableau du deuxième élément maximal (plus grand élément plus petit que l'élément maximal) pour la relation  $\leq$ . Calculer son coût (en nombre de comparaisons d'éléments).

**Question 48.3 :**

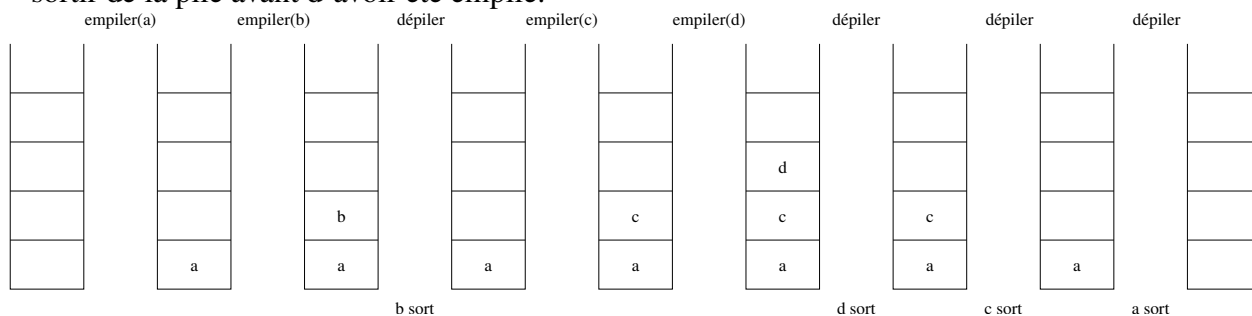
En utilisant le principe des tournois (coupe de France, Roland Garros,...) proposer un algorithme de recherche de l'élément maximal. Il n'est pas nécessaire de formaliser l'algorithme, une explication appuyée sur un schéma clair est suffisante.

**Question 48.4 :**

Proposer un deuxième algorithme de recherche du deuxième plus grand élément et calculer son coût en nombre de comparaisons.

## 49 Sortie de pile (*Examen de décembre 2004*)

On s'intéresse à l'ordre de sortie d'éléments d'une pile en fonction de leur ordre d'insertion. Par exemple, si l'on insère  $n = 4$  éléments  $a, b, c, d$  dans cet ordre dans une pile, ils peuvent en ressortir dans l'ordre  $b, d, c, a$ . En effet, on a effectué successivement empiler(a), empiler(b), dépiler, empiler(c), empiler(d), dépiler, dépiler, dépiler. La contrainte est qu'un élément ne peut sortir de la pile avant d'avoir été empilé.



On note  $c(n)$  le nombre d'ordres de sortie de la pile possibles, l'ordre d'entrée étant fixé.

**Question 49.1 :**

Dans les cas  $n = 1$ ,  $n = 2$  et  $n = 3$  donner tous les ordres de sortie possibles.

**Question 49.2 : bonus**

Donner en la justifiant l'équation de récurrence vérifiée par les  $c(n)$ . Commenter votre résultat.

## 50 Parcours (*Examen de décembre 2004*)

On rappelle que dans un arbre la racine est au niveau 0, les fils de la racine sont au niveau 1, etc...

**Question 50.1 :**

Dans un arbre binaire, combien de feuilles peut-on avoir au maximum au niveau  $i$ .

**Question 50.2 :**

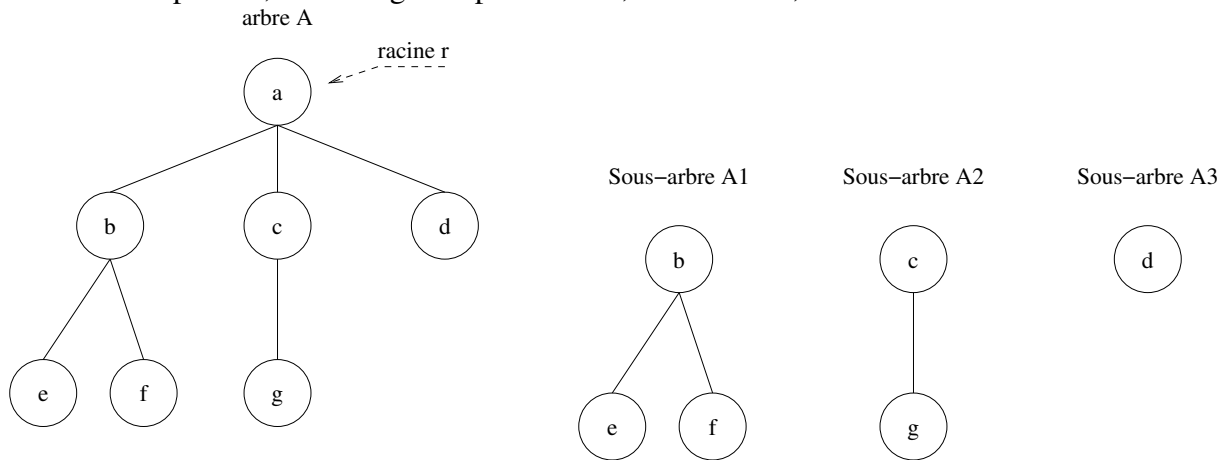
Proposer un algorithme qui prend en argument un arbre binaire  $A$  et un niveau  $i$  et renvoie le nombre de feuilles au niveau  $i$ .

**Question 50.3 :**

Calculer le coût de votre algorithme en nombre de nœuds examinés. Commenter votre résultat.

**51 Faire tourner les arbres (Examen de décembre 2004)**

On considère un arbre étiqueté défini récursivement : un arbre non vide est constitué d'une racine  $r$  et d'une liste d'arbres (les sous-arbres de la racine  $r$ ). L'ordre des sous-arbres fils de la racine est important, on distingue le premier fils, le deuxième, etc.



Dans l'exemple la racine  $a$  possède 3 fils où sont attachés les sous-arbres A1, A2 et A3.

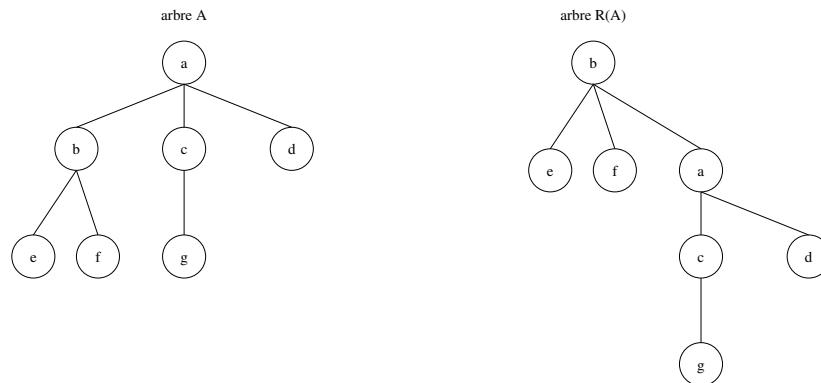
**Question 51.1 :**

Proposer une structure de donnée (en C) pour ce type d'arbres.

La transformation  $R$  de l'arbre  $\mathcal{A}$  opère de la manière suivante :

- soit  $\mathcal{A}$  est réduit à un seul nœud, alors  $R(\mathcal{A}) = \mathcal{A}$  ;
- soit  $\mathcal{A}$  n'est pas réduit à un nœud, dans ce cas  $\mathcal{A}$  est de la forme  $\mathcal{A} = (r, \mathcal{A}_1, \mathcal{A}_2, \dots, \mathcal{A}_k)$  avec  $\mathcal{A}_1$  non vide de la forme  $\mathcal{A}_1 = (s, \mathcal{B}_1, \mathcal{B}_2, \dots, \mathcal{B}_k)$ . Alors  $R(\mathcal{A}) = (s, \mathcal{B}_1, \mathcal{B}_2, \dots, \mathcal{B}_k, (r, \mathcal{A}_2, \mathcal{A}_3, \dots, \mathcal{A}_k))$ .

Pour l'exemple ci-dessus on obtient :



**Question 51.2 :**

Pour l'exemple donné appliquer la transformation  $R$  successivement jusqu'à obtenir l'arbre initial. Combien a-t-il fallu d'itérations ? Donner la suite des racines des arbres obtenue successivement.

**Question 51.3 :**

Ecrire en C la procédure associée à la transformation  $R$ .

On note  $\mathcal{A}^1 = R(\mathcal{A})$  et de manière générale  $\mathcal{A}^i = R(\mathcal{A}^{i-1})$ . Soit  $m(\mathcal{A})$  le nombre minimal d'itérations nécessaires pour retomber sur  $\mathcal{A}$

$$m(\mathcal{A}) = \min\{i, i \geq 1 \text{ et } \mathcal{A}^i = \mathcal{A}\}.$$

**Question 51.4 :**

Montrer que tous les nœuds de l'arbre  $\mathcal{A}$  sont racine de l'un des arbres  $\mathcal{A}^i$ . Donner la valeur de  $m(\mathcal{A})$  si l'arbre  $\mathcal{A}$  possède  $n$  nœuds, on pourra raisonner d'abord sur le nombre d'arêtes.

**Question 51.5 : bonus**

Donner un critère d'arrêt de l'itération lorsque le nombre de nœuds de  $\mathcal{A}$  et  $m(\mathcal{A})$  ne sont pas connus. Ecrire en C la procédure qui liste les racines de arbres  $\mathcal{A}^i$ .

**Question 51.6 :**

Comparer cet algorithme de parcours d'arbre avec les algorithmes de parcours vus en TD (parcours en profondeur d'abord et parcours en largeur d'abord).

## 52 Compter les tas (*Examen de décembre 2003*)

Dans cette exercice on va étudier des tas construits par la procédure d'insertion d'éléments (procédure vue en TD). On note  $n$  la taille du tas. Sans perte de généralité, on supposera que les clés des  $n$  éléments du tas sont  $\{1, \dots, n\}$ . On notera  $t_n$  le nombre de tas différents de taille  $n$  avec les clés  $\{1, \dots, n\}$ .

**Question 52.1 :**

Dans le cas  $n = 3$ , on a  $t_3 = 2$ . Quels sont les 2 tas  $T_1$  et  $T_2$  possibles ? En supposant équiprobable l'ordre d'insertion des éléments dans le tas, calculer la proportion de tas  $T_1$  construits.

**Question 52.2 :**

Quels sont les tas possibles dans le cas  $n = 5$  ? Donner pour chaque tas un ordre d'insertion particulier des éléments qui produise ce tas.

**Question 52.3 :**

Pour un tas de taille  $n$ , calculer sa hauteur puis calculer la taille du sous-arbre gauche et du sous-arbre droit. *Indication : regarder les cas  $n = 2^k - 1$ ,  $n = 2^k$ .*

**Question 52.4 : bonus**

Donner une équation de récurrence sur les nombres  $t_n$  pour  $n = 2^k - 1$ .

### **53 Collection (*Examen de décembre 2003*)**

On considère un ensemble  $\mathcal{O} = \{o_1, \dots, o_n\}$  d'objets. On appelle collection  $\mathcal{C}$  un groupe d'objets dans lequel un objet peut être représenté plusieurs fois, par exemple : des timbres, des pièces de monnaie, les mots d'un texte, les résultats de tirages au loto... Une spécification de la sorte collection est présentée figure 1.

```

#ifndef __COLLECTION__
#define __COLLECTION__
ANNEXE : SPECIFICATION DE LA SORTE COLLECTION

typedef char element;

struct noeud_collection { ..... QUESTION 2.1 .....
};
typedef struct noeud_collection *collection;

/*
collection_vide
description : construit une collection vide
parametres : aucun
valeur de retour : une collection vide
effets de bord : aucun
*/
collection collection_vide();

/*
inserer_dans_collection
description : ajoute une donnee a la collection
parametres : donnee a ajouter, collection a laquelle on ajoute
valeur de retour : la collection avec la donnee en plus
effets de bord : la collection passee en parametre n'est plus utilisable
(sinon perte de coherence du chainage)
*/
collection inserer_dans_collection(element donnee, collection c);

/*
appartient_a_collection
description : retourne 1 si l'element fournit appartient a la collection
et 0 sinon
parametres : element, collection
valeur de retour : 0 ou 1
effets de bord : aucun
*/
int appartient_a_collection(element donnee, collection c);

/*
supprimer_de_collection
description : supprime un element d'une collection
parametres : element, collection
valeur de retour : ..... QUESTION 2.2
effets de bord : ..... QUESTION 2.2
*/
collection supprimer_de_collection(element donnee, collection c);

/*
est_collection_vide
description : retourne 1 si la collection passee en parametre est vide
parametres : collection
valeur de retour : 1 si la collection passee en parametre est vide, 0 sinon
effets de bord : aucun
*/
int est_collection_vide(collection c);

/*
est_inclus_dans_collection
description : retourne 1 si la collection c1 est incluse dans c2
parametres : 2 collections
valeur de retour : 1 si c1 est incluse dans c2, 0 sinon
effets de bord : aucun
*/
int est_inclus_dans_collection(collection c1, collection c2);

#endif

```

FIGURE 1 – Spécification de “Collection”

On propose deux structures de données pour représenter une telle collection (figure 2) :

**Structure 1** Une liste simplement chaînée des objets ;

**Structure 2** Une liste de couples (*objet, entier strictement positif*) où un objet ne sera cité qu’une seule fois dans la liste et l’entier associé sera le nombre d’occurrences de l’objet dans la collection.

Une spécification de la sorte collection est donnée en annexe.

**Question 53.1 :**

Écrire une implémentation des deux structures de données.



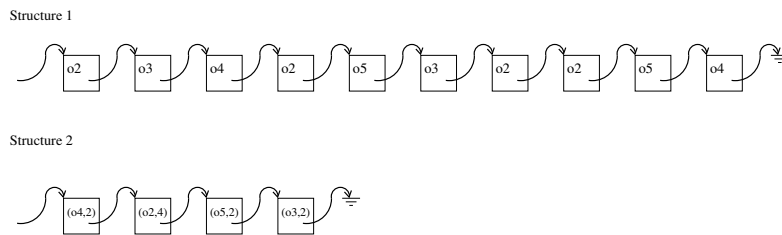


FIGURE 2 – Représentations de la collection  $C = \{o_2, o_3, o_4, o_2, o_5, o_3, o_2, o_2, o_5, o_4\}$

**Question 53.2 :**

Écrire la spécification de l'opérateur `supprimer_de_collection`. Puis, pour chacune des deux structures de données, proposer un algorithme et son implémentation.

**Question 53.3 :**

Pour les deux implémentations proposées ci-dessus, évaluer le coût en nombre de comparaisons entre des éléments. Donner des exemples réalisant le coût minimum et des exemples pour le coût maximum.

**Question 53.4 :**

Pour la structure de données 2, proposer un algorithme et son implémentation de l'opérateur `est_inclus_dans_collection`.

**Question 53.5 :**

Proposer des principes de solutions pour implémenter `est_inclus_dans_collection` avec la structure 1, premièrement sans modifier la structure, deuxièmement en enrichissant la structure.

## 54 Arbres binaires de recherche (*Examen de décembre 2003*)

Dans cet exercice on considérera des arbres binaires de recherche dont les étiquettes portées par les nœuds sont toutes différentes. On cherchera à lister  $\mathcal{A}$  ensemble des nœuds dont l'étiquette est supérieure à une valeur  $a$ . On notera par  $N_a$  le nombre de nœuds dont l'étiquette est supérieure ou égale à  $a$  et  $h$  la hauteur de l'arbre.

**Question 54.1 :**

Modifier le parcours infixé vu en cours de manière à lister tous les nœuds de  $\mathcal{A}$  avec un nombre de nœuds examinés majoré par  $N_a + (h + 1)$ . Justifier votre réponse.

**Question 54.2 :**

Sur la figure 3, marquer les sommets examinés par l'algorithme pour les valeurs  $a = 24$ ,  $a = 65$ ,  $a = 90$  et  $a = 105$ .

FEUILLE A ANNOTER ET A METTRE DANS LA COPIE No : \_\_\_\_\_

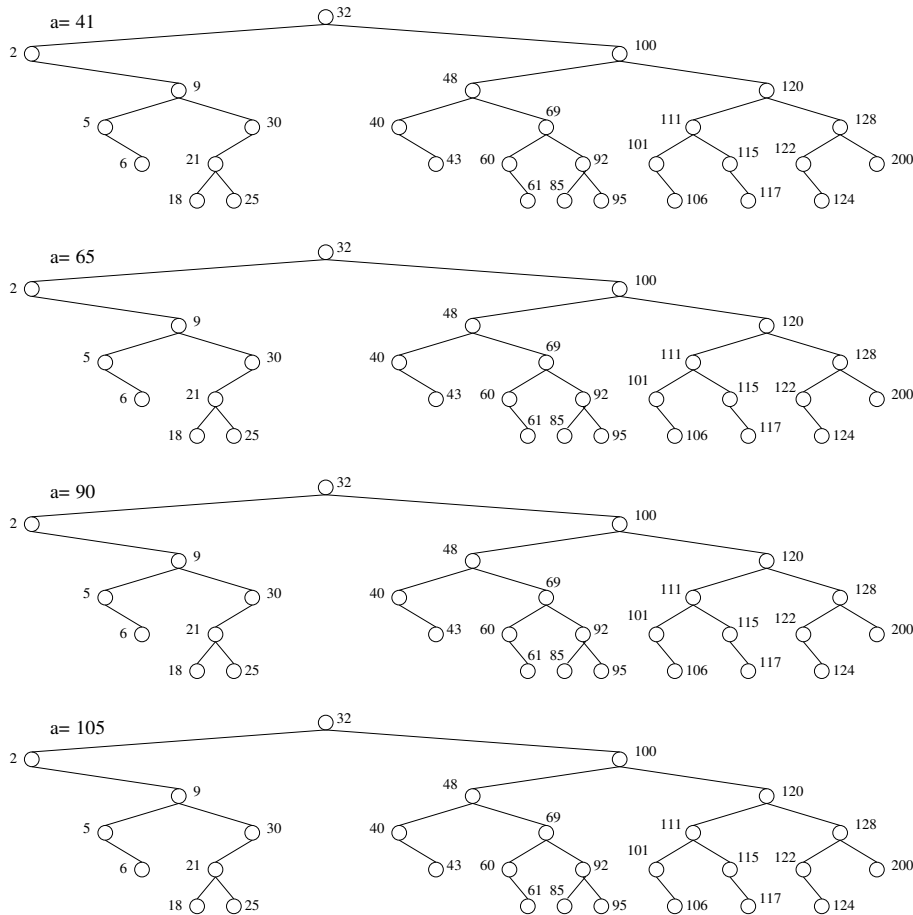


FIGURE 3 – Arbre binaire de recherche

**Question 54.3 :**

Modifier l'algorithme de manière à fournir la liste des nœuds dont l'étiquette est supérieure ou égale à  $a$  et inférieure strictement à  $b$ . Majorer la complexité de cet algorithme.

## 55 Bulles (*Quick octobre 2007*)

On se donne un tableau  $A$  de taille  $n$  d'éléments. L'ensemble des éléments est muni d'un ordre total et on suppose que tous les éléments du tableau sont distincts. Soit l'algorithme suivant :

```
pour i=1 à n faire  
  pour j=n à i+1 pas -1 faire  
    si A[j]<A[j-1] alors  
      Permute(A[j-1],A[j])  
    fin si  
  fin pour  
fin pour
```

**Question 55.1 :** Coût de l'algorithme

Calculer les coûts, au pire et au mieux, de cet algorithme en nombre d'opérations de permutations d'éléments dans le tableau. On donnera également les ordres grandeur.

**Question 55.2 :** Coût moyen de l'algorithme (question bonus)

Calculer le coût moyen de cet algorithme en supposant que les entrées sont modélisées par une permutation de  $\{1, \dots, n\}$  et que toutes les permutations ont la même probabilité.

## 56 Fibonacci (*Quick octobre 2007*)

Les nombres de Fibonacci  $\{F_n\}_{n \in \mathbb{N}}$  sont définis par la relation de récurrence suivante :

$$F_0 = F_1 = 1;$$

$$F_n = F_{n-1} + F_{n-2} \text{ pour } n \geq 2.$$

On souhaite calculer le nombre de Fibonacci d'ordre  $n$ . Pour cela on utilise le programme récursif suivant :

```
Fibo_recuratif(entier n)  
si n=0 ou n=1 alors  
  retourner 1  
sinon  
  retourner Fibo_recuratif(n-1) + Fibo_recuratif(n-2)  
fin si
```

On note  $C(n)$  le coût de l'algorithme en nombre d'additions.

**Question 56.1 :** Encadrement de  $C(n)$

Quelle est l'équation de récurrence vérifiée par  $C(n)$  ? Montrer que  $C(n)$  est croissante et en déduire que

$$2^{\frac{n}{2}} \leq C(n) \leq 2^n.$$

**Question 56.2 :** Algorithme itératif

Ecrire un algorithme itératif qui calcule  $F(n)$  et donner son coût en nombre d'additions.

**Question 56.3 :** Diviser pour régner

En remarquant que, pour  $n \geq 2$ ,

$$\begin{pmatrix} F_n \\ F_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} F_{n-1} \\ F_{n-2} \end{pmatrix},$$

en déduire que  $F_n$  peut être calculé en un temps  $\mathcal{O}(\log n)$ .

## 57 Fusion de liste (*Quick octobre 2007*)

On considère une implémentation de plusieurs listes au sein d'un même tableau par chaînage et on veut implémenter l'opérateur de fusion de liste dont le prototype est :

```
/*
```

```
fusion
```

```
description : retourne une liste qui est la fusion de 2 listes
```

```
parametres : deux listes gérées par chaînage dans un même tableau
```

```
valeur de retour : une liste chaînée dans le même tableau
```

```
effets de bord : les deux listes données en paramètre ne sont plus utili
```

```
*/
```

```
list fusion(liste l1, liste l2);
```

**Question 57.1 :**

On suppose que la représentation des listes se fait par chaînage **simple** dans le tableau. Ecrire l'algorithme de fusion et calculer son coût.

**Question 57.2 :**

On suppose que la représentation des listes se fait par chaînage **double** dans le tableau. Ecrire l'algorithme de fusion et calculer son coût.

**Question 57.3 :**

En quoi un chaînage circulaire (simple ou double) permettrait d'améliorer le coût de ces algorithmes

## 58 Relations antisymétriques (*Quick octobre 2005*)

### Question 58.1 :

Soit  $X$  un ensemble de  $n$  éléments. Donner, en le justifiant, le nombre de relations antisymétriques possibles sur  $X$ .

## 59 Tri à 3 couleurs (*Quick octobre 2005*)

### Question 59.1 : Réorganisation d'éléments

On considère un tableau de  $n$  éléments. Chaque élément est de couleur rouge, jaune ou bleue. On souhaite réorganiser le tableau de manière à ce que les éléments de couleur rouge soient en tête du tableau suivis par les éléments de couleur jaune et enfin les éléments bleus. On dispose uniquement de l'opérateur  $permute(i,j)$  qui échange le contenu des cases  $i$  et  $j$  du tableau.

En vous inspirant de l'algorithme de tri par insertion, écrire un algorithme qui réalise cette réorganisation. Justifier à l'aide de dessins cet algorithme et évaluer son coût en nombre d'appels à l'opérateur  $permute$ , donner son ordre de grandeur.

## 60 File en tableau (*Quick octobre 2005*)

On implémente une file par un tableau circulaire d'éléments avec la structure de donnée et du constructeur suivants :

```
typedef struct {
    element *donnee;
    int taille;      /* taille de la file */
    int tete;       /* indice de la tête de
                    la file dans le tableau*/
    int longueur;
    /* nb d'éléments dans la file */
} file;
/*
    est_file_invalide
    description : retourne 1 si la
                    file est invalide
    parametres : la file
    valeur de retour : 1 (invalide) ou 0
    effets de bord : libere de la memoire
*/
int est_file_invalide(file f);

/*
```

```

    allouer_file
    description : construit une file vide
    parametres : taille maximale de la file
    valeur de retour : une file vide
    effets de bord : aucun
*/
file file_vide(int taille);
/*
    liberer_file
    description : libere une file precedemment
                  allouee par alloue_vecteur
    parametres : la file
    valeur de retour : aucune
    effets de bord : libere de la memoire
*/
void liberer_file(file v);

```

### Question 60.1 :

Ecrire les spécifications, c'est à dire la description, les paramètres, la valeur de retour et les effets de bord, des opérations `enfiler (file f, element e)` et `defiler (file f)`.

### Question 60.2 :

Ecrire les procédures correspondant à votre spécification.

## 61 Eliminer les doublons (*Quick novembre 2005*)

Un objet du type Liste est appelé *liste*. Il représente un ensemble d'*éléments*. Chaque *élément* a une *position* dans la liste.

Ce type utilise un ensemble E pour ses *éléments*, l'ensemble N des entiers (pour les *positions*) et l'ensemble B des booléens.

La liste peut être manipulée à l'aide des opérateurs suivants :

liste_vide :	$\emptyset$	→	Liste
insérer	Liste $x \in E$ $x \in N$	→	Liste
premier	Liste	→	E
suisant	Liste	→	Liste
est_vide	Liste	→	B

**liste\_vide()** est une liste de zéro élément.

**insérer(l,e,i)** insère l'élément e en position i.

**premier(l)** est l'élément de la liste l qui est en première position.

**suisant(l)** est la liste l privée de l'élément en première position.

**est\_vide(l)** si et seulement si  $l$  a zéro élément.

On désire écrire la primitive Enlever\_doublon

**Question 61.1 :**

A l'aide des opérateurs de base écrire la primitive Enlever\_Doublons.

**Question 61.2 :**

Calculer, pour la primitive Enlever\_Doublons, le coût en nombre d'appels des primitives de base et commenter le résultat obtenu.

**Question 61.3 :**

En quoi une table de hachage pourrait-elle améliorer le coût de cette primitive ?

## 62 Réserve et conflit (*Quick novembre 2005*)

On considère un mécanisme de gestion de mémoire, celui-ci alloue des plages mémoires aux utilisateurs. Une requête mémoire est donc une plage d'adresses contigües, c'est à dire un intervalle de la forme  $r = [a_1, a_2]$ . Lors de la réception d'une requête  $r$  l'allocateur vérifie d'abord que cette demande n'entre pas en conflit avec les allocations précédemment effectuées, c'est à dire que l'intervalle  $[a_1, a_2]$  ne recouvre pas une plage déjà allouée. S'il n'y a pas de conflit, il alloue la mémoire à l'utilisateur et retourne un identificateur  $id$  de la plage (ou 0 s'il y a eu conflit). L'utilisateur libère la plage par une requête de type  $Libere(id)$  à l'allocateur.

L'allocateur de ressource utilise donc une structure de donnée notée  $\mathcal{EI}$  qui contient un ensemble d'intervalles. Il faut remarquer que les intervalles dans cette structure ont deux à deux des intersections vides. Les opérateurs sur cette structure de donnée sont

$Conflit(r)$  qui retourne vrai si l'intervalle  $r$  intersecte au moins un intervalle de la structure ;

$Verrouille(r)$  insère l'élément  $r$  dans la structure et renvoie un identifiant  $id$  ;

$Libere(id)$  supprime la plage d'identifiant  $id$  de la structure.

**Question 62.1 :** Liste d'intervalles

Lorsque la structure  $\mathcal{EI}$  est implémentée par une liste d'intervalles donner un algorithme pour la primitive  $Conflit(r)$ . Calculer le coût de cette primitive en nombre de comparaisons d'intervalles.

On décide pour implémenter  $\mathcal{EI}$  d'utiliser une structure d'arbre binaire de recherche dans lequel les nœuds sont étiquetés par des intervalles.

**Question 62.2 :**

Préciser la propriété vérifiée par les étiquettes dans cette structure de donnée.

**Question 62.3 :** Exemple



Construire l'arbre obtenu après le traitement successif des requêtes (initialement l'arbre est vide) :

[21, 25], [40, 41], [63, 71], [24, 30], [45, 60], [6, 10], [70, 72], [1, 3], [32, 36]

On précisera les plages qui n'auront pas été insérées.

Donner, en expliquant la méthode utilisée, l'arbre binaire de recherche obtenu si on supprime l'intervalle [40, 41] dans l'arbre précédent ?

#### Question 62.4 : Test de conflit

Ecrire l'algorithme de l'opérateur *Conflit*.

Quel est son coût en nombre de comparaisons d'intervalles ?

### 63 Comprimons (*Quick novembre 2006*)

On dispose d'un texte codé avec un alphabet  $\mathcal{A} = \{A, B, C, D, E, F, G, H\}$ . Les proportions de ces symboles dans le texte sont données en pourcentage dans le tableau suivant :

Symbole	A	B	C	D	E	F	G	H
Proportion (%)	2	10	4	18	12	16	32	6

#### Question 63.1 : Code de longueur fixe

Donner la taille du code de longueur fixe nécessaire pour coder cet alphabet. Calculer l'entropie associée à cet alphabet et commenter en une phrase cette valeur.

#### Question 63.2 : Code de longueur variable

Construire avec l'algorithme de Huffman statique un codage de longueur variable. Donner l'arbre de codage correspondant et le tableau des codes. Calculer la longueur moyenne du code et commenter le résultat en une phrase.

#### Question 63.3 : Algorithme de décodage (cas général)

Etant donné un arbre de codage binaire, écrire un algorithme de décodage qui prend en argument un texte codé et renvoie le texte décodé. (Les opérateurs de base sur du type arbre de codage sont donnés au verso de cette feuille).

#### Question 63.4 : Tableau des codes (cas général)

Etant donné un arbre de codage binaire, écrire un algorithme qui construit la table qui associe les codes aux symboles.

## Opérateurs sur le type Arbre de codage

EstFeuille

paramètres : un Arbre de codage A (donnée)

valeur de retour : un booléen

description : vaut vrai ssi A est réduit à une feuille

effets de bord : aucun

Symbole

paramètres : un Arbre de codage A (donnée)

valeur de retour : un symbole de l'alphabet de départ

description : A doit être réduit à une feuille,  
renvoie le symbole associé à la feuille

effets de bord : aucun

FDroit

paramètres : un Arbre de codage A (donnée)

valeur de retour : un Arbre de codage

description : A est un arbre non réduit à une feuille,  
renvoie le fils droit associé à la racine de A

effets de bord : aucun

FGauche

paramètres : un Arbre de codage A (donnée)

valeur de retour : un Arbre de codage

description : A est un arbre non réduit à une feuille,  
renvoie le fils gauche associé à la racine de A

effets de bord : aucun