

TRAVAUX DIRIGES (Listes)

Développer les algorithmes suivants sur les listes linéaires chaînées :

1. Construire une Llc à partir de n données lues.
2. Longueur d'une Llc.
3. Rechercher dans une Llc l'élément qui a le plus grand nombre d'occurrences.
4. Accès par valeur dans une Llc.
5. Accès par position dans une Llc.
6. Suppression par valeur dans une Llc.
7. Suppression par position dans une Llc.
8. Insertion par position dans une Llc.
9. Interclassement de deux listes ordonnées.
10. Eclater une liste en 2 Llcs selon un critère donné.
11. Trier une Llc par la méthode des bulles.
12. Implémenter le modèle de Llc en utilisant la représentation contigue.
13. Un polynôme peut être représenté par une Llc. Dire comment. Ecrire les algorithmes suivants :
 - calcul du polynôme en un point x donné.
 - dérivé d'un polynôme.
 - somme de deux polynômes.
 - produit de deux polynômes.
14. Etudier les algorithmes de recherche, insertion et suppression d'un élément dans un vecteur. Les comparer avec ceux correspondant sur les Llcs.
15. Construire une liste bidirectionnelle à partir de n données.
16. Insérer un élément dans une liste bidirectionnelle.
17. Supprimer un élément dans une liste bidirectionnelle.

TRAVAUX DIRIGÉS (Files)

1. Implémenter une file d'attente d'entiers en utilisant un tableau $\text{File}(-1 : 100)$, où $\text{File}(-1)$ est utilisée pour indiquer la tête, $\text{File}(0)$ pour indiquer la queue et $\text{File}(1:100)$ contient les éléments de la file d'attente. Comment initialiser la file vide ? Traduire les opérations du modèle.
2. Comment implémenter une file d'attente où chaque élément contient un nombre variable d'entiers.
3. Une *file d'attente avec priorité* est une file d'attente dans laquelle l'opération de défilement récupère l'élément le plus prioritaire. Définir le modèle et l'implémenter.

TRAVAUX DIRIGES (Piles)

1. Représenter les expressions suivantes sous forme polonaise postfixée :

$a+b$, $(a+b)/d$, $((c+d) +(d-e))+5$, $-(a+b)+(5+b)c$, $-(((a+b)+(c-d))/5)+a^5$

Essayer de donner l'algorithme d'évaluation sans utiliser la pile. Quels sont les problèmes rencontrés? Donner l'algorithme d'évaluation avec l'utilisation d'une pile.

2. Soit à analyser des expressions mathématiques qui utilisent les symboles (, { et [pour l'ouverture des sous expressions et les symboles), } et] pour leur fermeture respective. Chaque symbole de fermeture doit être associé à son symbole d'ouverture. Utiliser le modèle de pile pour écrire l'algorithme qui effectue une telle vérification.

3. Les éditeurs de texte utilisent, en général, deux caractères particuliers pour traiter une ligne d'un texte:

- le caractère d'effacement, noté #, permet d'effacer le caractère précédent,
- le caractère d'annulation, noté @, permet d'effacer toute la ligne courante.

Utiliser le modèle de pile pour écrire l'algorithme qui édite une ligne d'un texte.

4. Donner une implémentation possible(schéma+description) pour :

- une file de files,
- une file de piles,
- une pile de files,
- une pile de piles.

TRAVAUX DIRIGES (Récursivité) 1/2

1. Ecrire l'algorithme récursif qui calcule la somme des n premiers entiers naturels.
 2. Ecrire les algorithmes récursifs correspondant au:
 - (a) quotient de a par b ,
 - (b) reste de a et b ,
 - (c) pgcd (plus grand diviseur commun) de deux entiers non négatifs.
 3. Donner une définition récursive de $a + b$, où a et b sont des entiers non négatifs. Ecrire l'algorithme récursif correspondant.
 4. Soit A un tableau d'entiers. Présenter des algorithmes récursifs pour
 - (a) le maximum de A ,
 - (b) le minimum de A ,
 - (c) la somme des éléments de A ,
 - (d) le produit des éléments de A ,
 - (e) la moyenne des éléments de A .
 5. Développer les algorithmes itératifs et récursifs pour:
 - (a) la factorielle,
 - (b) le produit $a*b$,
 - (c) la séquence de Fibonacci.
- Evaluer (trace) les algorithmes pour les cas suivants : $6!$; $9!$; $100*3$; $6*4$; $\text{fib}(10)$; $\text{fib}(11)$.
Comparer les algorithmes itératifs et récursifs
6. Algorithmes récursifs sur les listes linéaires chaînées (llc):
 - (a) inverser une llc,
 - (b) rechercher un élément donné dans une llc.
 7. Soit la définition récursive suivante :

```
FONCTION Fib (N:ENTIER):ENTIER
VAR X,Y : ENTIER
SI N <= 1
  Fib := N
SINON
  X := Fib(N-1)
  Y := Fib(N-2)
  Fib := X + Y
FSI
```

TRAVAUX DIRIGES (Récursivité) 2/2

- (a) Donner une expression complètement parenthésée de $\text{fib}(6)$, puis l'évaluer.
- (b) Définir les points de retour de la procédure.
- (c) Que contient la zone de données ? Définir la structure.
- (d) Transformer l'algorithme récursif en utilisant la pile. L'algorithme obtenu est donc informel.
- (e) Faire une trace.
- (f) Peut-on réduire la zone de données ? Redéfinir alors la zone de données.
- (g) Essayer de rendre l'algorithme formel. Cela revient à supprimer les "ALLERA". Peut-on se passer complètement de la pile ?

8. Refaire la même chose pour la fonction suivante :

```
FONCTION Comb (N, M : ENTIER) : ENTIER
{ Calcul DE Cnm POUR 0 <= M<= N ET N >=1 ) }
SI (N=1) OU (M=0) OU (M=N)
    Comb <-- 1
SINON
    Comb <-- Comb (N-1, M) + Comb(N-1, M-1)
FSI
```

9. Transformer l'algorithme récursif correspondant à la tour de Hanoi.

10. Reprendre l'algorithme récursif qui inverse une liste linéaire chaînée de l'exercice 6. , puis utiliser une pile pour éliminer la récursivité.

TRAVAUX DIRIGES (Arbres) 1/3

1. Ecrire des algorithmes récursifs et non récursifs pour déterminer dans un arbre binaire:

- (a) le nombre de noeuds,
- (b) le nombre de feuilles,
- (c) la somme des contenus de tous les noeuds,
- (d) la profondeur.

2. Ecrire un algorithme qui détermine si un arbre binaire est :

- (a) strictement binaire,
- (b) complet.

3. Développer un algorithme pour trouver les doubles dans une suite de n nombres.

- (a) avec une liste linéaire chaînée,
- (b) avec un arbre de recherche binaire.

Dénombrer les comparaisons dans chaque cas.

4. Trouver les algorithmes récursifs de parcours d'un arbre binaire.

5. Trouver les algorithmes de recherche, d'insertion et de suppression dans un arbre de recherche binaire.

6. Etude du parcours Inordre

6.1 Utiliser le modèle défini en cours pour écrire l'algorithme récursif de parcours *inordre* dans un arbre binaire.

6.2 Trouver l'algorithme équivalent non récursif.

6.3 Un arbre binaire enfilé est défini de telle sorte qu'un noeud, au lieu de contenir un pointeur NIL dans son champ droit, il contient un pointeur vers son successeur inordre (arbre binaire enfilé à droite). Pour implémenter ces sortes d'arbres, il suffit de rajouter un booléen au niveau du noeud pour dire si celui-ci est enfilé ou non.

Montrer qu'avec ces sortes d'arbres, on peut éliminer la pile dans l'algorithme non récursif de parcours inordre. Il s'agit donc d'écrire l'algorithme.

6.4 En considérant en plus l'opération Père(p) dans le modèle (qui délivre le père du noeud référencé par p), on peut développer l'algorithme de parcours inordre sans utiliser la pile. Ecrire l'algorithme correspondant.

7. Représentation d'une liste linéaire chaînée par un arbre

On peut représenter une liste linéaire chaînée par un arbre binaire de la façon suivante : Les éléments de la liste sont au niveau des feuilles. Chaque noeud qui n'est pas une feuille contient le nombre de feuilles de son sous arbre gauche.

7.1 Trouver le K-ième élément de la liste représentée ainsi.

7.2 Etudier et développer l'algorithme de construction.

7.3 Etudier et développer l'algorithme de suppression.

TRAVAUX DIRIGES (Arbres) 2/3

8 Implémentation des arbres

8.1 Implémenter le modèle défini sur les arbres au moyen des représentations définies en cours : (a) représentation contigue (b) représentation contigue séquentielle

8.2 Un arbre binaire peut être représenté comme suit:

Si A est un tableau à n éléments : $A(i)=j$ si j est le parent du noeud i. $A(i)=0$ si i est la racine. C'est ce qu'on appelle la représentation des parents. Traduire le modèle.

8.3 Représentation des arbres au moyen des listes de fils.

On a un tableau T de listes, indexé par des noeuds que nous supposons numérotés de 1 à n. Chaque liste pointée par Tab (i) contient les fils du noeud i.

La description peut être la suivante :

```
TYPE S = STRUCTURE                { Maillon de la liste }
  Val : ENTIER
  ADR : POINTEUR(S)
FIN
TYPE List = POINTEUR(S)           { Ensemble des listes }
TYPE Typearbre = STRUCTURE        { Ensemble des arbres }
  Tete : TABLEAU(1..Max) DE List
  Racine: ENTIER
FIN
VAR Arbre : Typearbre { Un arbre }
```

Implémenter le modèle. (Structures de données et algorithmes)

9. Algorithme de Huffman

Supposons que nous avons des messages composés de séquences de caractères. Dans chaque message, les caractères sont indépendants et apparaissent avec des probabilités connues . Nous voulons coder chaque caractère en une séquence de 0 et de 1 de telle sorte qu'aucun code n'est le préfixe d'un autre code. Grâce à cette propriété dite du préfixe on peut décoder une chaîne de 0 et de 1. Nous voulons écrire l'algorithme de Huffman permettant de réaliser ce code. Pour le faire on utilise les structures de données suivantes :

a) un tableau ARBRE où chaque élément est du type T1 défini comme suit :

```
TYPE T1 = Structure
  Fg, Fd, Pere : ENTIER
FIN
```

pour représenter les arbres binaires.

Le champ Père permet de retrouver les chemins des feuilles vers la racine, aussi nous pouvons trouver le code pour chaque caractère.

b) un tableau ALPHABET d'éléments du type T2 défini comme suit :

```
TYPE T2 = STRUCTURE
  Symbole : CAR
  Probabilité : REEL
  Feuille : ENTIER          { Indice vers l'arbre }
FIN
```

TRAVAUX DIRIGES (Arbres) 3/3

pour associer à chaque symbole de l'alphabet à coder avec sa feuille correspondante. Ce tableau enregistre aussi la probabilité pour chaque caractère.

c) un tableau FORET d'éléments qui représente les arbres. Chaque élément est alors du type T3 défini comme suit:

```
TYPE T3 = STRUCTURE
  Poids : REEL
  Racine : ENTIER
FIN
```

9.1 Comment initialiser les tableaux FORET, ALPHABET et ARBRE.

9.2 Donner le pseudo algorithme en faisant ressortir les modules nécessaires.

9.3 Développer ces modules.

9.4 Donner l'algorithme de Huffman.

9.5 Application : Ecrire l'algorithme qui permet d'archiver un programme écrit en PASCAL, puis l'algorithme qui permet de le désarchiver en utilisant le code de Huffman.

10. les arbres m-aires

10.1 Rappeler le modèle défini sur les arbres m-aires. Ecrire les algorithmes de parcours.

10.2 Implémenter le modèle des arbres m-aires avec les représentations définies en cours. Peut on utiliser les représentations définies dans les arbres binaires ? Traduire le modèle dans chaque représentation possible.

10.3 Transformer des arbres m-aires en arbres binaires.

10.4 Transformer des forêts en arbres binaires.

TRAVAUX DIRIGÉS (Hachage)

1. Reprendre la technique de résolution des collisions par essai linéaire. Donner une expression formelle de l'algorithme d'insertion. Etudier et développer l'algorithme de suppression.
2. Dans la technique dite d'essai linéaire, la séquence de test est donnée par la séquence cyclique $h(K), h(K)-1, \dots, 0, M-1, \dots$. Une autre technique de résolution des collisions consiste à construire la séquence de tests par une autre fonction de hachage. C'est ce qu'on appelle adressage ouvert avec double hachage ou hachage uniforme. Ecrire les algorithmes d'insertion et de suppression.
3. Une autre méthode de résolution de collision consiste à chaîner les synonymes dans la même table ("Chaînage dispersée dans une table"). chaque élément de la table contient alors un champ additionnel pour le lien vers le prochain synonyme éventuel. Etudier et développer les algorithmes d'insertion et de suppression.
4. Reprendre les algorithmes sur l'essai linéaire dans le cas où le nombre d'éléments par case est supérieur à 1.
5. Reprendre les algorithmes sur le chaînage interne dans le cas où le nombre d'éléments par case est supérieur à 1.
6. Déterminer le nombre d'articles en débordement dans une table de 1000 éléments, quand la densité est 40, 50, 60, 70, 80 et 90%.

TRAVAUX DIRIGES (Fichiers-struct. simple)

1. Considérer les douze cas de la figure présentée en cours donnant les différentes méthodes d'accès. Pour chaque cas, développer les algorithmes de
 - recherche
 - insertion
 - suppression
 - requête à intervalle

Dans le cas où le fichier est un tableau ordonné, développer en plus l'opération de chargement initial qui consiste à remplir le fichier à raison de $\mu\%$ par bloc par des articles qui sont lus en ordre croissant selon leur clé.

La requête à intervalle consiste à récupérer tous les articles dont les clés sont comprises entre deux clés a et b données.

TRAVAUX DIRIGES (Fichiers-index primaire)

1. Fichier avec un seul index (accès par clé primaire).

1.a) Le fichier est un ensemble de blocs contigus 1, 2,N. A chaque article est associée une clé qui l'identifie de façon unique. Les articles sont de longueur fixe. Le fichier peut être le suivant :

Le fichier d'index est ordonné. Le fichier de données ne l'est pas. L'index est supposé en mémoire.

Concevoir les algorithmes de :

- définition et de création des fichiers d'index et de données.
- chargement du fichier d'index en mémoire avant son utilisation. Nous supposons que le fichier d'index est assez petit pour qu'il puisse entrer et rester en mémoire. Cela revient donc à ranger l'index dans un tableau en mémoire.
- réécriture du fichier d'index de la mémoire après son utilisation. Si l'index a été modifié, le réécrire comme un nouveau fichier.
- recherche d'un article de clé donnée. Trouver la position de la clé au niveau de l'index (recherche binaire), puis faire un accès direct pour retrouver la donnée.
- insertion des articles au fichier de données et d'index. On l'ajoute en fin du fichier de données. Sur l'index, les décalages sont obligatoires, ce qui n'est pas très grave car le traitement se fait entièrement en mémoire (aucun accès disque).
- suppression des articles du fichier . Considérer le cas où elle est logique (simple) et le cas où elle est physique (plus complexe)
- modification des articles. 2 sortes de modifications selon que la clé est modifiée ou non.

1.b) Même chose dans le cas où les articles sont de longueur variable.

2. Fichier avec deux niveaux d'index (accès par clé primaire).

Considérer le cas où les articles sont de longueur fixe et refaire la même chose qu'en 1).

TRAVAUX DIRIGES (Fichiers-index secondaire)

1. Le fichier est un ensemble de blocs contigus 1, 2,N. A chaque article est associée une clé primaire qui l'identifie de façon unique. Les articles sont de longueur fixe. On convient de mettre B articles par bloc.

On considère deux index secondaires pour l'attribut X et Y de l'article. Ces index sont organisés sous forme de "listes inversées". Les fichiers d'index sont ordonnés. Le fichier de données ne l'est pas.

Concevoir les algorithmes de :

- chargement des fichiers d'index secondaire et primaire.
- sauvegarde des fichiers d'index secondaire et primaire.
- listage des articles de clé secondaire, selon l'attribut X , donnée.
- listage des articles ayant une clé secondaire, selon l'attribut X, donnée et une clé secondaire, selon l'attribut Y, donnée.
- insertion d'un article ayant A comme clé primaire, Ax et Ay comme clés secondaires selon les attributs X et Y respectivement.
- suppression d'un article de clé primaire donnée.

TRAVAUX DIRIGES (Fichiers-arbre)

1. Retrouver les algorithmes de recherche, insertion, suppression, requête à intervalle et de parcours en ordre croissant dans un fichier structuré en arbre de recherche m-aire. On considère le cas où les articles sont rangés : (i) avec les clés , (ii) séparément

On examine le cas où les articles sont de longueur : fixe ou variable

NB. Pour l'algorithme de recherche utiliser la fonction $\text{Rechnoeud}(P, \text{Clef})$ qui retourne soit le plus petit indice J tel que $\text{Clef} \leq \text{Clé}(P, J)$ ou $\text{Nbr}(P)$ si clé est supérieur à toutes les clés dans $\text{Nœud}(P)$.

Pour l'algorithme d'insertion on peut utiliser :

** La procédure Trouver qui a comme paramètre

En entrée : Arbre et Clef

En sortie

Si la clé est trouvée

Rech : pointe le nœud contenant la clé trouvée

Position : contient la position de la clé à l'intérieur du nœud.

Si la clé n'est pas trouvée

Rech : pointe la feuille qui pourrait contenir la clé.

Position : contient alors l'index de la plus petite clé dans le nœud pointé par Rech qui est supérieure à la clé recherchée Clef. Si toutes les clés dans le nœud pointé par Rech sont inférieures à Clef, Position est mis $\text{Nbr}(\text{Rech})$.

Une variable Trouv est mise à VRAI ou à FAUX selon l'appartenance ou non de la clé.

** La procédure $\text{Insérercle}(\text{Rech}, \text{Clef}, \text{Position})$ qui insère la clé Clef à la position Position dans le nœu Rech.

2. ISAM : Retrouver les algorithmes de recherche, insertion, suppression et de requête à intervalle pour la méthode d'accès ISAM présentée en cours.

3. Etudier et développer les algorithmes de recherche, insertion et de suppression dans un arbre AVL.

4. Retrouver les algorithmes de recherche, insertion et de suppression dans un fichier structuré en B-arbre.

5. SIS : Retrouver les algorithmes de recherche, insertion, suppression et de requête à intervalle pour la méthode d'accès SIS présentée en cours.

TRAVAUX DIRIGES (Fichiers-hachage)

1. Développer les algorithmes de recherche, insertion et suppression pour les méthodes de hachage suivantes appliquées aux fichiers :

- essai linéaire
- double hachage
- chaînage interne
- chaînage séparé

On écrira également le module d'initialisation.