USTHB le 03/01/2017

Examen Final

Exercice 1 : (6 pts)

Soit T un vecteur d'entiers de taille 100 et soit une suite de **n** nombres (n<=100) prenant leurs valeurs dans l'intervalle [1 , 100].

1. Ecrire une action paramétrée qui permet de ranger les **n** nombres dans les **n** premières cases du vecteur.
2. Ecrire une fonction qui recherche une valeur **Val** dans le vecteur et donner sa complexité en justifiant votre réponse.

A présent, on voudrait avoir une recherche avec une complexité constante (O(1)). Dans ce cas :

3. Proposer une méthode de rangement des **n** valeurs dans le vecteur (donner l'action de remplissage du vecteur).
4. Donner la nouvelle fonction de recherche et expliquer pourquoi elle a une complexité constante.

Exercice 2 : (9 pts)

Dans cet exercice, on se propose de vérifier l'existence d'un chemin entre un point de départ et un point d'arrivée dans un labyrinthe.

Le labyrinthe est représenté sous la forme d'une matrice carrée N×N, supposée construite. Une cellule est repérée par son numéro de ligne et son numéro de colonne. Dans chaque cellule, on indique s'il y a la présence d'un mur (valeur 1) ou rien (valeur 0). Un chemin entre deux cellules est une suite de cellules voisines ayant toutes la valeur 0. Chaque cellule a au plus quatre voisines : Haut, Bas, Gauche et Droite.

0	0	0	0	0	0	1	1
1	0	1	1	1	0	1	1
1	0	0	0	1	0	1	0
0	0	1	0	1	0	0	1
0	1	1	1	1	0	1	0
0	0	0	0	1	1	1	0
1	0	1	0	0	0	0	1
1	1	1	1	1	1	0	1

Exemple : il y a un chemin entre la cellule (1,1) et la cellule (8,7) par contre il n y a pas de chemin entre la cellule (1,1) et la cellule (6,8)

Soit une liste L d'éléments qui contiennent les coordonnées (ligne et colonne) des cellules du labyrinthe.

1. Donner la déclaration de cette liste.
2. Donner la fonction qui permet d'insérer une cellule dans une liste L.
3. Donner la définition d'une fonction **Appartient** qui vérifie l'appartenance d'une certaine cellule à une liste L.

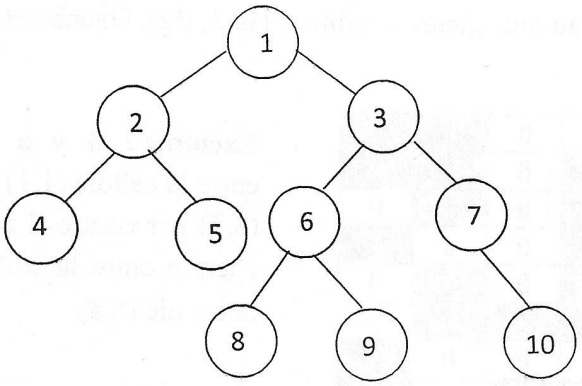
4. Etant données deux listes L_1 et L_2, écrire une fonction **SupListe** qui retourne une troisième liste contenant tous les éléments de L_1 qui n'appartiennent pas à L_2.
5. Donner la définition d'une fonction **Voisinage** qui retourne une liste de toutes les cellules voisines d'une certaine cellule. Exemple : les voisines de (1,1) sont (1,2) et les voisines de (6,2) sont (6,1) (7,2) (6,3).
6. Ecrire une action **Supprimer** qui supprime un élément de la liste des cellules. On doit récupérer le numéro de la ligne et le numéro de la colonne de la cellule supprimée.
7. En utilisant une pile et en supposant les primitives de manipulations des piles prédéfinies, écrire une fonction **Verifier** qui vérifie l'existence d'un chemin entre deux cellules données (la fonction doit avoir parmi ses paramètres, les coordonnées de la cellule de départ et celles de la cellule d'arrivée).

Exercice 3 : (5 pts)

Un arbre binaire est un arbre où chaque nœud possède au plus deux fils : un fils gauche et un fils droit.

1. Donner la déclaration d'un arbre binaire avec une représentation chainée en mémoire.

On voudrait réaliser un parcours en largeur de l'arbre. Ainsi, les nœuds de l'arbre ci-dessous doivent être visités dans l'ordre : 1, 2, 3, 4, 5, 6, 7, 8, 9, 10.



2. Quelle structure de données allez-vous utiliser pour un tel traitement.
3. Ecrire une action **ParcoursLargeur** qui affiche les valeurs des nœuds de l'arbre selon un parcours en largeur. On suppose que les primitives de manipulation des structures utilisées dans cet exercice sont prédéfinies.

Corrigé de l'Examen Final

Exercice 1: (6 pts)

1. Procédure Remplissage (E/S T : tableau [100] entier , E/n : entier)
```
variable i, val : entier ;
début
  pour i←1 à n
      faire
        lire(val) ; T[i]←val ;
      fait
fin
```

2. Fonction Recherche (E/ T : tableau[100] entier , E/n : entier, E/val : entier) :booléen
```
début
  pour i←1 à n
      faire
        si (T[i]= val) alors retourner Vrai ;
      fait
  retourner Faux ;
fin
```

Sa complexité en en O(n) car nous avons au plus (maximum) n comparaisons.

3. Procédure RemplissageBis (E/S T : tableau[100] entier , E/n : entier)
```
variable i, val : entier ;
début
  pour i←1 à 100  faire T[i]←0 ; fait      //Initialisation du tableau
  pour i←1 à n
      faire
        lire(val) ; T[val]←1 ;
      fait
 fin
```

4. Fonction RechercheBis (E/ T : tableau [100] entier , E/val : entier) : booléen
```
début
   si (T[val]= 1) alors retourner Vrai ;
                  sinon retourner Faux ;
fin
```

Sa complexité en en O(1) car chaque vérification nécessite une seule instruction de comparaison (accès direct à la case qui correspond à la valeur recherchée).

Exercice 2 : (9 pts)

1. Type Liste=^ Element ;
   Type Element = enregistrement
   inf : Cellule ;
   suivant : Liste ;
   finenreg ;

   Type Cellule = enregistrement
   lg : entier ;
   col : entier ;
   finenreg ;

2. Fonction Inserer (E/ L : Liste, E/ l : entier, E/ c : entier) : Liste
   variable nouv : Liste ;
   début
   nouv←allouer(TailleDe(Element)) ;
   ^nouv.inf.lg←l ; ^nouv.inf.col←c ;
   ^nouv.suivant←L ; L←nouv ;
   retourner L ;
   fin ;

3. Fonction Appartient (E/ L : Liste, E/ l : entier, E/ c : entier) : booléen
   debut
   tantque (L ≠ nil)
   faire
   si (^L.inf.lg=l) et (^L.inf.col=c) alors retourner vrai ;
   sinon L←^L.suivant ;
   fait
   retourner faux ;
   fin ;

4. Fonction SupListe (E/ L_1 : Liste, E/ L_2 : Liste) : Liste
   variable L_3 : Liste ;
   debut
   L_3 ← nil ;
   tantque (L_1 ≠ nil)
   faire
   si (Appartient (L_2 , ^L_1.inf.lg , ^L_1.inf.col) = faux)
   alors L_3←Inserer (L_3, ^L_1.inf.lg , ^L_1.inf.col)
   fsi
   L_1←^L_1.suivant ;
   fait
   retourner L3 ;
   fin

5. Fonction Voisinage (E/ Lab : tableau [100][100] entier, E/ n : entier, E/ i : entier, E/ j : entier) : Liste

```
Variable L : Liste ;  //Lab est la matrice qui représente le labyrinthe
debut
 L←nil ;
 si (i=1 et j=1)
        alors si (Lab[i][j+1]=0) alors L ←Inserer (L, i , j+1) ; fsi ;
              si (Lab[i+1][j]=0) alors L ←Inserer (L, i+1 , j) ; fsi ;fsi
 si (i=1 et j=n)
        alors si (Lab[i][j-1]=0) alors L ←Inserer (L, i , j-1) ; fsi ;
              si (Lab[i+1][j]=0) alors L ←Inserer (L, i+1 , j) ; fsi ;fsi
 si (i=n et j=1)
        alors si (Lab[i-1][j]=0) alors L ←Inserer (L, i-1 , j) ; fsi ;
              si (Lab[i][j+1]=0) alors L ←Inserer (L, i , j+1) ; fsi ;fsi
 si (i=n et j=n)
        alors si (Lab[i-1][j]=0) alors L ←Inserer (L, i-1 , j) ; fsi ;
              si (Lab[i][j-1]=0) alors L ←Inserer (L, i , j-1) ; fsi ;fsi
 si (i=1 et 1<j  et j<n )
        alors  si (Lab[i][j-1]=0) alors L ←Inserer (L, i , j-1) ; fsi ;
               si (Lab[i][j+1]=0) alors L ←Inserer (L, i , j+1) ; fsi ;
               si (Lab[i+1][j]=0) alors L ←Inserer (L, i+1 , j) ; fsi ;fsi
 si (i=n et 1<j  et j<n )
        alors  si (Lab[i][j-1]=0) alors L ←Inserer (L, i , j-1) ; fsi ;
               si (Lab[i][j+1]=0) alors L ←Inserer (L, i , j+1) ; fsi ;
               si (Lab[i-1][j]=0) alors L ←Inserer (L, i-1 , j) ; fsi ;fsi
 si (j=1 et 1<i  et i<n )
        alors  si (Lab[i-1][j]=0) alors L ←Inserer (L, i-1 , j) ; fsi ;
               si (Lab[i+1][j]=0) alors L ←Inserer (L, i+1 , j) ; fsi ;
               si (Lab[i][j+1]=0) alors L ←Inserer (L, i , j+1) ; fsi ;fsi
 si (j=n et 1<i  et i<n )
        alors  si (Lab[i-1][j]=0) alors L ←Inserer (L, i-1 , j) ; fsi ;
               si (Lab[i+1][j]=0) alors L ←Inserer (L, i+1 , j) ; fsi ;
               si (Lab[i][j-1]=0) alors L ←Inserer (L, i , j-1) ; fsi ;fsi
 si ( 1<i  et i<n et  1<j  et j<n)
        alors  si (Lab[i-1][j]=0) alors L ←Inserer (L, i-1 , j) ; fsi ;
               si (Lab[i+1][j]=0) alors L ←Inserer (L, i+1 , j) ; fsi ;
               si (Lab[i][j+1]=0) alors L ←Inserer (L, i , j+1) ; fsi ;
               si (Lab[i][j-1]=0) alors L ←Inserer (L, i , j-1) ; fsi ; fsi
 retourner L ;
fin
```

6. Procédure Supprimer (E/S L : Liste , E/S l : entier , E/S c : entier)

```
variable nouv : Liste ;
début
l=^L.inf.lg ; c=^L.inf.col ;
nouv←L ; L←^L.suivant ; liberer(nouv) ; fin
```

7. Fonction Vérifier (E/ Lab : tableau [100][100] entier, E/ n : entier,
   E/ i_1 : entier, E/ j_1 : entier , E/ i_2 : entier, E/ j_2 : entier) : booléen

```
variable i, j : entier ; trouv : booleen ; P : Pile ;
         CellulesVisites, Voisins, VoisinsNonVisites : Liste ;
debut
 i←i1 ; j←j1 ; trouv=faux ;
 P← InitPile() ;
 tantque (trouv = faux )                                          3,0
  faire
     CellulesVisites←Inserer(CellulesVisites,i,j) ;
     Voisins ← Voisinage(Lab,n,i,j) ;
     VoisinsNonVisites←SupListe(Voisins,CellulesVisites) ;
     si (VoisinsNonVisites≠ nil)
        alors si (Appartient (VoisinsNonVisites, i2 , j2 )=vrai)
                 alors retourner vrai ;
                 sinon Supprimer(VoisinsNonVisites,i,j) ; //avancer dans le chemin
                       si(VoisinsNonVisites≠ nil alors Empiler(P, VoisinsNonVisites)
        sinon si (PileVide(P))
                 alors retourner faux ; // il n y a plus de choix a explorer
                 sinon Desempiler (P, VoisinsNonVisites) ;
                       Supprimer(VoisinsNonVisites,i,j) ;//avancer dans autre chemi
                       si(VoisinsNonVisites≠ nil alors Empiler(P, VoisinsNonVisites)
  fait
fin
```

Exercice 3: (5 pts)

1.
```
Type arbre=^nœud ;
Type nœud = enregistrement
                info : entier ;
                FilsGauche : arbre ;        (1)
                FilsDroit : arbre ;
              finenreg ;
```

2. La structure à utiliser est la File. (1)
3.
```
Procédure ParcoursLargeur (E/ A : arbre)
variable F : File ;
debut                                                    (3)
 F← InitFile() ;
 si (non ArbreVide (A)) alors Enfiler(F,A) ;
 tantque (non FileVide(F))
   faire Defiler (F,A) ;
         écrire (^A.info) ;
         si (non ArbreVide(FilsGauche(A))) alors Enfiler(F, FilsGauche(A)) ;
         si (non ArbreVide(FilsDroit(A))) alors Enfiler(F, FilsDroit(A)) ;
   fait
fin
```

U.S.T.H.B - FEI
Département Informatique
N. Bensaou

ALGO 2ème année
Licence Informatique
Section : ISIL B

Examen Final du 3 janvier 2016

Exercice 1.- (10pts) Soient L1 et L2 deux listes linéraire d'entiers, simplement chaînées. L1 est dite *préfixe* de L2 si L2 commence par L1.

Exemple : L1 $= \{3, 7\}$ est préfixe de L2 $= \{3, 7, 1, 4, 0\}$.

a.) Écrire la fonction : `Fonction estPrefixe(E L1, L2:LISTE): booleen` qui retourne vrai si L1 est préfixe de L2 et faux sinon. Donnez sa complexité.

b.) L1 est dite *suffixe* de L2 si L2 se termine par L1. Exemple : L1 $= \{3, 7\}$ est suffixe de L2 $= \{1, 4, 0, 3, 7\}$. Expliquez le principe d'une fonction `estSuffixe(L1, L2)` et donnez sa complexité.

c.) Écrire la fonction `Fonction estSuffixe(E L1, L2: LISTE): booleen` lorsque L1 et L2 sont des listes doublement chaînées.

Soit L une liste circulaire simplement chaînée dont le dernier nœud est pointé par FIN.

d.) Écrire la procédure : `Procedure decalage(E/S L, FIN:LISTE)` ; qui décale les éléments de la liste d'une position vers la gauche, c'est-à-dire que le deuxième devient premier, le troisième devient deuxième, etc. et le premier devient dernier.

Par exemple si $L = \{1, 2, 3, 4, 5, 6, 7\}$ alors après décalage on aura $L = \{2, 3, 4, 5, 6, 7, 1\}$.

La complexité dépend t-elle de la longueur de la liste ? Donnez cette complexité en justifiant votre réponse.

e.) Écrire une fonction récursive qui retourne 1 si une liste L1 est plus courte qu'une liste L2 et 0 sinon.

Exercice 2.- (10 pts)

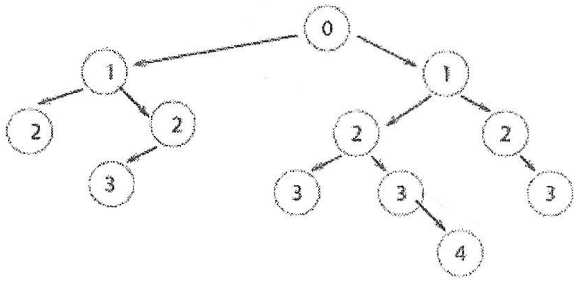
I.) **Arbre binaire** :

On considère un arbre binaire de racine `r` donnée.

a.) Écrire une fonction `estFeuille` qui retourne vrai si un nœud `x` est une feuille et faux sinon.

b.) Écrire une fonction `Fonction toutesLesFeuilles (E r:arbre): Pile;` qui retourne une pile contenant toutes les feuilles de l'arbre de racine `r` donnée.

c.) Déclarer un arbre binaire dont les nœuds ont la référence au père et la profondeur (entier) et écrire une fonction récursive qui affecte à chaque nœud d'un arbre binaire donné la profondeur de ce nœud. La profondeur de la racine vaut 0, la profondeur d'un nœud fils de la racine vaut 1, la profondeur d'un nœud fils du fils de la racine vaut 2, ainsi de suite.



d.) Un arbre binaire est dit *complet* si toutes ses feuilles sont à la même *profondeur*. Écrire une fonction `Fonction estComplet(E r: arbre) : booleen;` qui retourne 1 si un arbre binaire donné est complet et 0 sinon.

II.) **Arbre binaire de recherche :**

Soit $\{5, 19, 2, 14, 3, 4, 1, 17, 24, 16\}$ une suite de clés données dans cet ordre.

a.) Dessinez l'arbre binaire de recherche de cette suite et donnez la liste des clés selon le parcours postfixé et le parcours préfixé.

b.) Affichez la liste des clés visités lors de la recherche de la clé 18. Insérez cette clé, puis insérez la clé 15.

c.) Dessinez l'arbre après la suppression de la racine.

Bon travail