QROC A31 Structure de données

Février 2000

Documents distribués et notes personnelles autorisés

Il faut impérativement répondre sur la copie sujet

Faire d'abord au brouillon, aucun double de sujet ne pourra être distribué.

NOM : AUGERAUD

19,5

PRENOM : PASCAL

Récursivité (3 points)

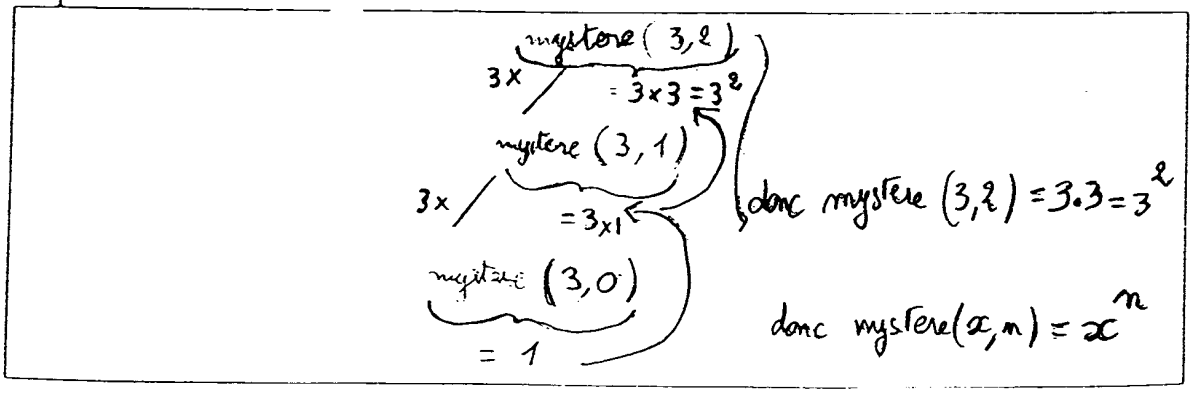
Soit la relation de récurrence mystere définie par :

```
double mystere(double x, int n)
{
if (n==0) return 1;
else return (x*mystere(x,n-1));
}
```

2,5

a) Donner une trace d'exécution de mystere(3,2) sous forme d'un arbre. En déduire ce que fait cette fonction et en déduire la complexité de celle-ci.

Réponse :



b) Calculer d'une manière formelle la complexité de la fonction mystere.

Réponse :

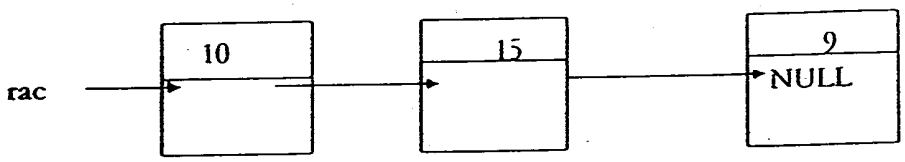
pour le calcul de x^n, la fonction mystere est appelée $(n+1)$ fois

donc $\lim_{n\to\infty}(n+1) = \lim_{n\to\infty}(n) \Rightarrow$ complexité de type $\Theta(n)$ (linéaire)

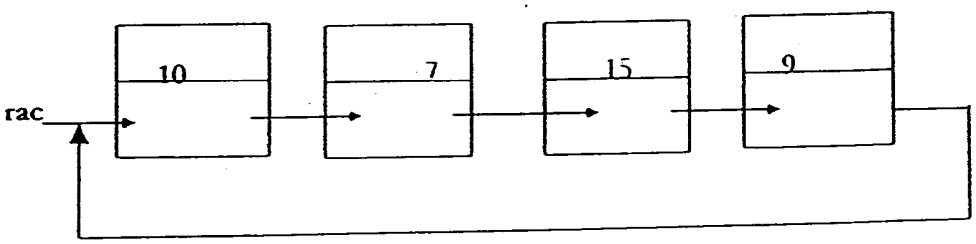
Listes chaînées

Exercice 1 : 4 points

On considère la structure chainée initiale suivante :



Où rac désigne le pointeur de début de cette liste. On veut créer la liste suivante :



Parmi les cinq portions de codes proposées une seule réalise cette transformation (répondre dans le tableau réponse). Toutes ont en commun les déclarations suivantes :

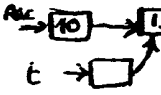
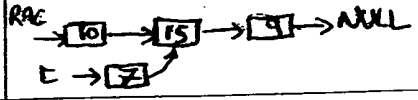
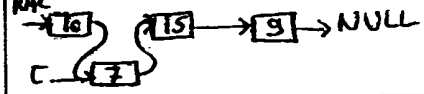
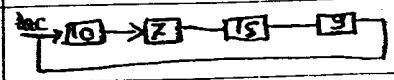
```
typedef struct noeud {
        int valeur;
        struct list_noeud *suiv;
        }list_noeud;
typedef list_noeud *LISTE;
LISTE rac;

/*--------------------Proposition 1---------------------
LISTE t;
t=(LISTE)malloc(sizeof(list_noeud));
rac->suiv=t;
t->suiv=rac->suiv->suiv->suiv;
t->valeur=7;
/*--------------------Proposition 2----------------------*/
LISTE t;
```

```
rac=rac->suiv->suiv;
t=(LISTE)malloc(sizeof(list_noeud));
t->suiv=rac;
t->valeur=7;
rac=t;
/**********************proposition 3**********************/
LISTE t;
t=(LISTE)malloc(sizeof(list_noeud));
t->suiv=rac;
t->valeur=7;
rac=t;
/***************proposition 4*************************/
LISTE t;
t=(LISTE)malloc(sizeof(list_noeud));
t->suiv=rac->suiv;
t->valeur=7;
rac=t;
/**********************proposition 5**********************/
LISTE t;
t=(LISTE)malloc(sizeof(list_noeud));
t->suiv=rac->suiv;
t->valeur=7;
rac->suiv=t;
rac->suiv->suiv->suiv->suiv=rac;
```

4

Réponse : proposition 5

Recopie des lignes	Schémas correspondants
LISTE t;	Crée le pointeur t
t=(LISTE)malloc(sizeof(list_noeud))	réserve une place mémoire de la taille de list_noeud et initialise t avec son adresse
t->suiv = rac->suiv	
t->valeur = 7	
rac->suiv = t	
rac->suiv->suiv->suiv->suiv = rac	

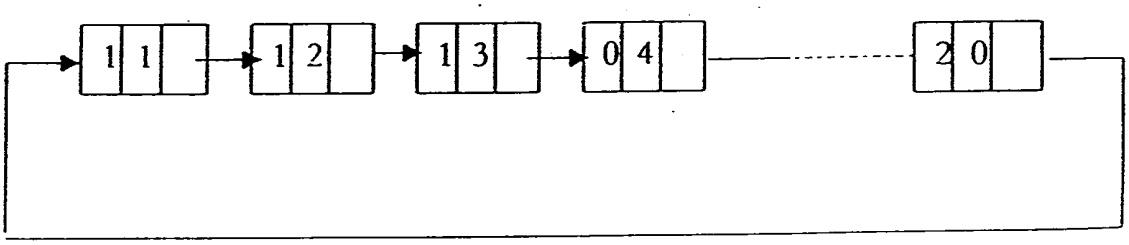
Exercice 2 : (6 points)

Le contour d'une image binaire joue un rôle très important dans beaucoup d'application telles que la compression d'images et la synthèse d'images. Il correspond à l'ensemble de points appartenant à la frontière d'un objet. Un contour est décrit par un ensemble de points P de coordonnées (X, Y). Il peut dont être représenté par une liste chainée de points P.

Voici un exemple de contour, le point de coordonnée (1,1) est un point du contour. Les 24 points en gras sont des points du contour.

	0	1	2	3	4	5	5	7	8	9
0	0	0	0	0	**1**	**1**	0	0	0	0
1	0	**1**	**1**	**1**	0	0	**1**	0	0	0
2	**1**	0	0	0	0	0	0	**1**	**1**	0
3	**1**	0	0	0	0	0	0	0	0	**1**
4	**1**	0	0	0	0	0	0	0	0	**1**
5	0	**1**	0	0	0	0	0	0	0	**1**
6	0	0	**1**	0	0	0	0	0	**1**	0
7	0	0	0	**1**	0	0	0	0	0	**1**
8	0	0	0	0	**1**	0	0	**1**	**1**	0
9	0	0	0	0	0	**1**	**1**	0	0	0

Ce contour peut être représenté sous la forme d'une liste circulaire chainée de la façon suivante. Remarquez qu'en raison de la circularité de la liste dans aucun nœud le champs suivant ne vaut NULL.



Parmi les propositions suivantes laquelle représente un contour :

1)

```
typedef struct{
          int x;
          int y;
          } POINT;
typedef struct Noeud {
          POINT element;
          struct Nœud suivant;
          }list_noeud;
typedef  list_noeud *CONTOUR;
```

2)

```
typedef struct{
          int x;
          int y;
          } POINT;
typedef struct Noeud {
```

```
            POINT element;
            struct Nœud suivant,precedent;
            }list_noeud;
        typedef  list_noeud *CONTOUR;
```

3)

```
        typedef struct{
            int y;
            } POINT;
        typedef struct Noeud {
            POINT element;
            struct Nœud *suivant;
            }list_noeud;
        typedef  list_noeud *CONTOUR;
```

4)

```
        typedef struct{
            int x;
            int y;
            } POINT;
        typedef struct Noeud {
            POINT element;
            struct Nœud *suivant;
            }list_noeud;
        typedef  list_noeud *CONTOUR;
```

OUI

Réponse :

Réponse 4) car c'est la seule structure à posséder une structure POINT avec x et y et une structure list_noeud avec un pointeur : *suivant

1) Ecrire les fonctions suivantes

1.1 Recherche d'un point dans un contour dans une liste (retour 1 si le point appartient au contour, 0 sinon) : int Recherche (CONTOUR MyContour, POINT MyPoint).

```
int Recherche (CONTOUR MyContour , POINT MyPoint)
{  CONTOUR Temp = MyContour ;
   do
   {
      if ((( temp -> element.x == MyPoint.x) && ( temp -> element.y == MyPoint.y)
         return (1);
      else  temp= temp -> suiv ;
   } while (Temp != MyContour );   return (0);
}
```

2

1.2 Ajout d'un point au contour : **CONTOUR Ajout(CONTOUR MyContour, POINT MyPoint).**

```
CONTOUR Ajout (CONTOUR MyContour, POINT MyPoint)
{
    CONTOUR Temp ;
    Temp = (CONTOUR) malloc (sizeof (Lst_noeud));
    Temp → suivant = MyContour → suivant ;
    MyContour → suivant = Temp ;
    Temp → élément = MyPoint ;
    return (MyContour) ;
}
```

2

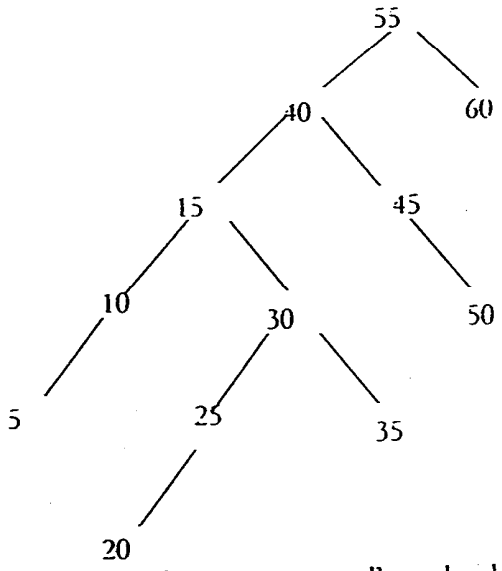
1.3 Suppression d'un point du contour : **CONTOUR Supprimer(CONTOUR MyContour, POINT MyPoint).**

```
CONTOUR Supprimer (CONTOUR MyContour, POINT MyPoint)
{
    CONTOUR Temp ;
    if (recherche (MyContour, MyPoint) == 1)
    {
        if (MyContour → suivant → element == MyPoint)
        {
            Temp = MyContour → suivant ;
            MyContour → suivant = MyContour → suivant → suivant ;
            free (Temp) ;
            return (MyContour) ;
        }
        else
            return (Supprimer (MyContour → suivant, MyPoint)) ;
    }
    return (NULL)
}
```

1,5

Arbres (7 points)

Soit l'arbre de recherche binaire défini par :



On définit la longueur de cheminement externe d'un arbre binaire A (LCE(A)) comme la somme des longueurs de toutes les branches issues de la racine.
Dans l'exemple précédent LCE(A) = 17
On peut définir LCE(A) d'une manière récursive de la façon suivante :

LCE(A) = 0 Si A = ∅
Sinon soit A = <O, A1, A2>
LCE(A1) + LCE (A2) + nf(A1) +nf (A2).

nf(A) est le nombre de feuilles de l'arbre binaire A (vu en cours et en TD).

Ecrire la fonction récursive en C qui code LCE(A).

```
int LCE (Arbre A)
{
  if (A == NULL) return(0);
  else return ( LCE(A->fils_gauche) + LCE(A->fils_droit)
              + nf(A->fils_gauche) + nf(A->fils_droit) );
}
int nf (Arbre A)
{ if (A == NULL ) return(0);
  else return (max (1, nf(A->fils_gauche) + nf(A->fils_droit) );
}
```

Montrer par récurrence que : $\forall A \neq \emptyset$, $LCE(A) \geq nf(A) * \log_2(nf(A))$
Pour vous aider dans cette démonstration, vous pouvez supposer les relations suivantes :
nf(A) = nf(A1) + nf(A2)

$\text{Log}(ab) = \log(a) + \log(b)$

$\forall a \text{ et } b > 0 \quad a\log_2 a + b\log_2 b \geq (a+b)\log_2(a+b)/2.$

cas de base initiale : si A est une feuille : $LCE(A) = 0 \geq nf(A) \cdot \log_2(nf(A)) = 1 \cdot 0$

hypothèse : vrai au rang $(n-1)$:

donc $LCE(A \to fg) \geq nf(A \to fg) \cdot \log(nf(A \to fg))$

et $LCE(A \to fd) \geq nf(A \to fd) \cdot \log(nf(A \to fd))$

or $LCE(A) = LCE(A \to fg) + LCE(A \to fd) + nf(A \to fg) + nf(A \to fd)$

$LCE(A) \geq (nf(A \to fg) + nf(A \to fd)) \cdot \log_2\left(\frac{nf(A \to fg) + nf(A \to fd)}{2}\right) + nf(A \to fg) + \ldots$

$\geq \left[nf(A) \cdot \log_2\left(\frac{nf(A)}{2}\right) + nf(A)\right] = nf(A)\left(\underbrace{-\log(2)}_{=-1} + \log nf(A) + 1\right)$

$\geq nf(A) \cdot \log_2(nf(A)) \Rightarrow$ Vrai au rang n.

A votre à vis que devient la relation précédente quand l'arbre est complet.

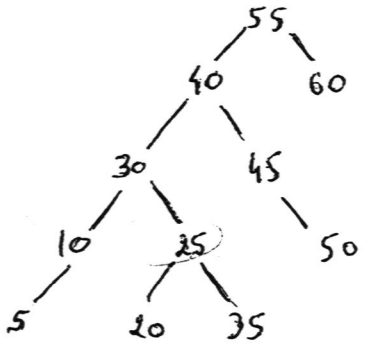
si l'arbre est complet : $nf(A) = 2 \cdot nf(A \to fg) = 2 \cdot nf(A \to fd)$

et $LCE(A \to fg) = LCE(A \to fd)$

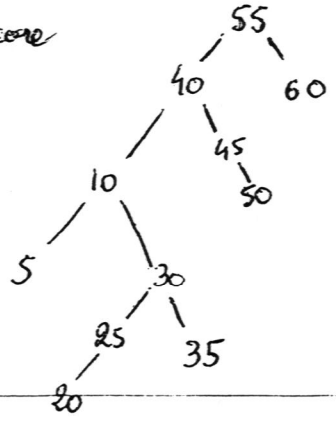
donc $LCE(A) = 2\left(LCE(A \to fg) + nf(A \to fg)\right) = nf(A) \cdot \log_2(nf(A))$

0,5

Que devient l'arbre après la suppression du nœud 15

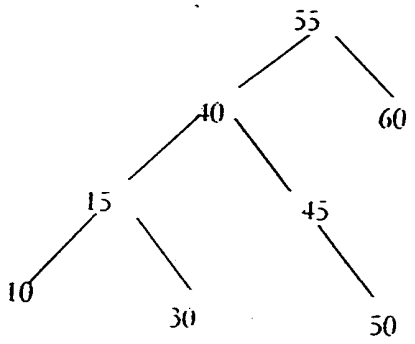


ou encore

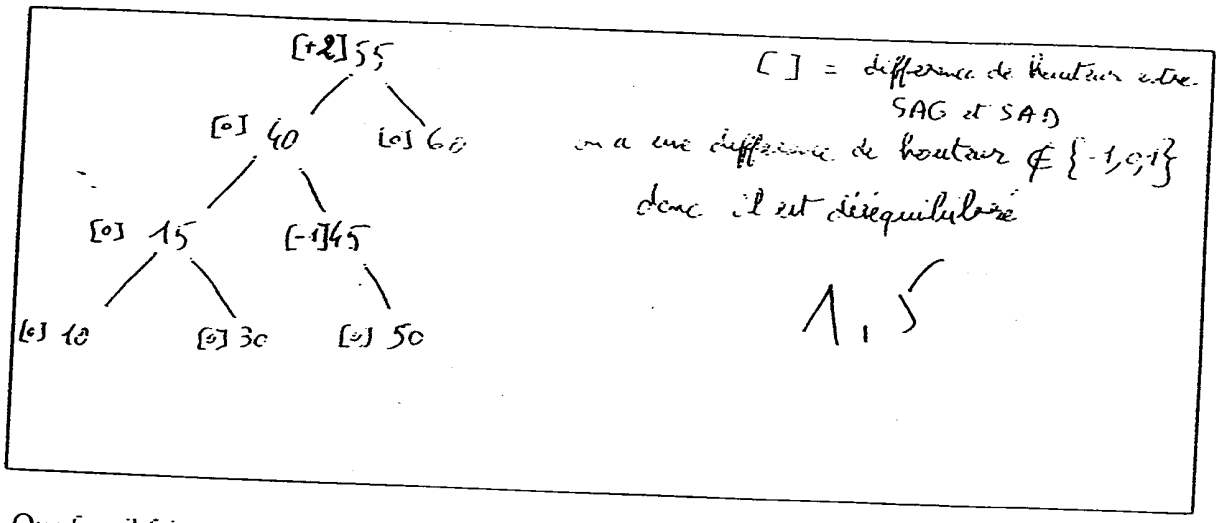


1

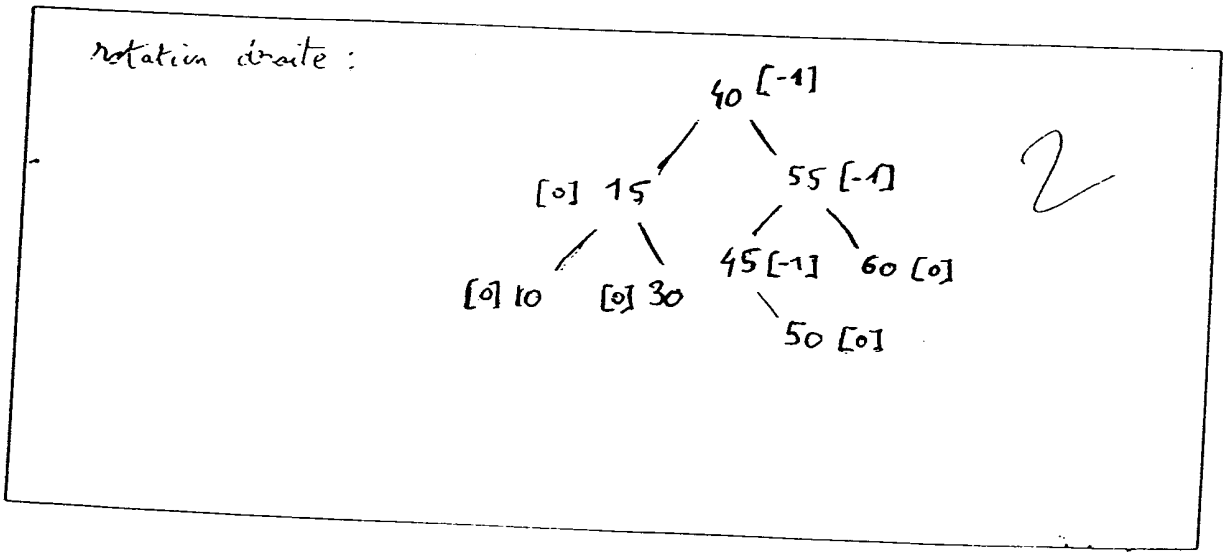
Soit l'arbre suivant :



Pourquoi cet arbre binaire n'est-il pas équilibré ?



Que faut-il faire pour le rendre équilibré ?



- Montrons par récurrence que $\forall A \neq \emptyset$ $\boxed{LCE(A) \geq nf(A) \times \log_2(nf(A))}$

sait $nf(A) = nf(A_1) + nf(A_2)$

$\log a \times b = \log a + \log b$

$\forall a$ et $b > 0$ $a \log_2 a + b \log_2 b \geq (a+b) \frac{\log_2(a+b)}{2}$

1 + (n-1)

n

1° cas 1 feuille $n = 1$ feuilles

2° cas à N-1

$LCE A = LCF(A_1) + LCE(A_2) + nf(A_1) + Nf(A_2)$

$= LCE(A \to fg) + LCE(A \to fd) + nf(A \to fg) + nf(A \to fd)$

$\geq$ $= nf(A \to fg) \times \log_2 nf(A \to fg) + nf(A \to fd) \times \log_2 nf(A \to fd) + nf(A \to fg) + nf(A \to fd)$

$\geq \left[nf(A \to fg) + nf(A \to fd) \right] \log_2 \frac{(nf(A \to fg) + nf(A \to fd))}{2} + \ldots$

$\geq nf(A) \cdot \log_2 \frac{nf(A)}{2} + nf(A)$

$\geq nf(A) \left[\log_2 nf(A) - \log_2 2 + 1 \right]$

$\geq nf(A) \left[\log_2 nf(A) - 1 + 1 \right]$

$\boxed{LCF A \geq nf(A) \log_2 nf(A)}$

1° condition initiale

si A est une feuille $n = 1$ $LCE(A) \geq nf(A) \cdot \log_2(nf(A))$

$0 \geq 1 \cdot 0$

vraie

1° est vrai 2° est vrai (n-1) donc à n est vrai