Deug Mias-2
Informatique
Institut Galilée
partiel de novembre 2000

-I- Ensembles et Listes Triées (2+2+2+2)

Nous utiliserons pour représenter une liste un enregistrement contenant un champ "dernier" indiquant la position du dernier élément de la liste dans un tableau, et le champ "contenu" qui représente ce tableau. Le premier élément est toujours en position 1 . Un ensemble vide est représenté par une liste vide, c'est à dire que le champ "dernier" vaut 0. On ne peut ranger plus de NMAX élément dans le tableau et donc les listes sont de taille au plus égale à NMAX (dans la suite on ne testera pas le dépassement)

1) Ecrire une procédure itérative *append(Ed,k,Er)*qui ajoute les éléments entre le rang *k* et le rang *Ed.dernier* de la liste *Ed* à la liste *Er*.
Exemple : *Ed={7, 8, 9} , Er = {2, 4, 6},* après *append(Ed,2,Er) on a Er= {2, 4, 6,8,9}*

2) Ecrire une version récursive de *append* et dire en une phrase le principe du raisonnement récursif.

3) Ecrire une fonction itérative *Palindrome(Ed)* qui renvoie *Vrai* si *Ed* est une liste "palindromique".
Exemple: {1,3 5, 5, 3,1} est palindromique.

4) Ecrire une version récursive de *Palindrome* et dire en une phrase le principe du raisonnement récursif.
On utilisera une fonction récursive *PalindromeRec(...)* qui sera appellée une première fois par *Palindrome*.

-II- Nombre de parties de k éléments d'un ensemble (3+3)

Nous allons écrire une fonction *Parties(k, n)* qui renvoie le nombre de parties de *k* éléments d'un ensemble de *n* éléments. Pour cela nous allons utiliser les formules de récurrences suivante:
Parties(k,n) = Parties(k, n-1) + Parties (k-1, n-1)
Parties (n,n) = Parties(0,n) =1
Parties(1,n) = n

NB: Elles proviennent de la remarque suivante : si on fixe un élément *e* parmi les *n*, les parties de *k* éléments se partagent en deux catégories : celles qui contiennent *e* (il reste alors *k-1* éléments à fixer parmi *n-1*) et celles qui ne contiennent pas *e* (il reste *k* éléments à fixer parmi *n-1*)

1) Ecrire la fonction récursive *PartiesRec(k,n)* et décrire l'arbre des appels de *PartiesRec(3,5)* en numérotant les appels selon l'ordre chronologique.

2) Ecrire une version (très) économique *Parties (k,n)* qui appelle une fonction récursive *PartiesRecM* (à écrire également). *Parties* a un tableau local *M* bidimensionnel où sont rangés les résultats des appels de *PartiesRecM*, ce qui évite de refaire plusieurs fois le même appel. Décrire également ici l'arbre des appels déclenchés par *Parties (3,5)*

-III-Mystère (2+4)

Soit un tableau de *N* réels *T*. Le système ci-dessous défini le tableau *TL* à partir du tableau *T*:

--

$TL[1] = (T[1]+T[2]+...+T[k])/k$

$TL[p] = TL[p-1] + (-T[p-1] + T[p+k-1])/k$

pour $p>1$ et $p \leq n-k+1$

$TL[p] = (TL[p-1] - (T[p-1] / k-m+1)) * (k-m+1/ k-m)$

pour $p = n-k+1+m$ et m variant de 1 à $k-1$

--

1) Que représente le tableau *TL* ? pourquoi y a t'il deux sortes d'équations de récurrence ?

2) Ecrire une procédure récursive *ConstruitL(T,TL,k)* et simulez l'appel de *ConstruitL(Tor,Tres,2)* sur le tableau de3 éléments suivant :

Tor = { 1, 0 ,1 }

En réalité on construira un procédure directement récursive *ConstruitLRec(T,TL, k , p)* qui construit *TL* entre les indices *1* et *p*, puis la procédure *ConstruitL(T,TL,k)* qui fait un appel à *ConstruitLRec* pour construire le tableau TL en entier.