Université Bordeaux 1 LST Mention Informatique Semestre 3 2013-2014

Algorithmique 1 : Devoir Surveillé 1

Complexité. Listes

Durée : 30mn
Sans documents

Exercice 1.1 *Complexité*
Soit la fonction `f`.

```
fonction f(val n: entier, val p: entier): entier;
var t: tableau[1..n+1] d'entier;
var i: entier;
debut
  si p > (n - p) alors
    p= n - p;
  finsi
  pour i=1 a n+1 faire
    t[i]=1;
    j=i-1
    tant que j>1 faire
      t[j]= t[j] + t[j-1];
      j= j-1;
    fintq
  fin pour
  retourner t[p+1]
fin
```

1. On fait appel à la fonction `f` avec `n=6` et `p=4`. Quelle est la valeur retournée par cet appel ?
2. Quelles sont les complexités en temps et en mémoire de cet algorithme ?

Exercice 1.2 *Listes chaînées : primitives*

On considère l'implémentation des listes chaînées par allocation dynamique. Ecrire les primitives :

```
fonction supprimerApres(ref L:listeSC) : vide;
fonction supprimerApres(ref L:listeDC) : vide;
```

Exercice 1.3 *Listes chaînées : fonctions*
On considère un tableau de N entiers contenant des nombres pairs. Par exemple, pour $N = 8$, un tableau vérifant ces contraintes est `T={4,8,16,6,10,24,10,26}`.

1. Ecrire une fonction `tableauToListe` qui transforme un tableau en une liste simplement chaînée en conservant l'ordre des entiers dans le tableau et renvoie `vrai` si les éléments du tableau sont conformes à l'énoncé (des entiers tous pairs) et `faux` sinon.

2. Ecrire une fonction `supMoyenneListeSC` qui à partir de la liste ainsi constituée calcule la moyenne `M` et fournit la liste des valeurs $T[j]$ telles que $T[j-1] < M$ et $M < T[j]$, $2 \leq j < N$. Sur l'exemple, la moyenne $M = 13$ et la liste produite est `{16, 24, 26}`.
3. Si on choisit d'utiliser des listes doublement chaînées, écrire la fonction `supMoyenneListeDC`.
4. Donner les avantages et les inconvénients des deux implémentations.

Annexe A **Liste simplement chaînée listeSC**

```
listeSC= liste d'objets;
fonction valeur(val L:listeSC) : objet;
fonction debutListe(ref L:listeSC) :vide;
fonction suivant(ref L:listeSC) : vide;
fonction listeVide(val L:listeSC) : booleen;
fonction getCleListe(val L: listeSC) : curseur;
fonction creerListe(ref L:listeSC) : vide;
fonction insererApres(ref L:listeSC, val x:objet;) : vide;
fonction insererEnTete(ref L:listeSC, val x:objet) : vide;
fonction supprimerApres(ref L:listeSC) : vide;
fonction supprimerEnTete(ref L:listeSC) : vide;
fonction setCleListe(ref L: listeSC, val c:curseur) : vide;
fonction detruireListe(ref L:listeSC) : vide;
```

Annexe B **Liste doublement chaînée listeDC**

```
listeDC= liste d'objets;
fonction valeur(val L:listeDC) : objet;
fonction debutListe(ref L:listeDC) : vide;
fonction finListe(ref L:listeDC): vide;
fonction suivant(ref L:listeDC) : vide;
fonction precedent(ref L:listeDC) : vide;
fonction listeVide(val L:listeDC) : booleen;
fonction getCleListe(val L:listeDC) : curseur;
fonction creerListe(ref L:listeDC) : vide;
fonction insererApres(ref L:listeDC, val L:objet;) : vide;
fonction insererEnTete(ref L:listeDC, val X:objet) : vide;
fonction supprimerApres(ref L:listeDC) : vide;
fonction supprimerEnTete(ref L:listeDC) : vide;
fonction detruireListe(ref L:listeDC) : vide;
fonction setCleListe(ref L:listeDC, curseur c) : vide;
```