Algorithmique TD2 (Corrigé): Tableaux: recherche et modification d'éléments

22 octobre 2008

1. Ecrire une procédure qui inverse les éléments d'un tableau :
   - le premier devient le dernier
   - le deuxième devient l'avant dernier
   - etc.

```
procedure inverse(ES tableau t(N):entier,E taille:entier)
variables
  i:entier
  j:entier
  i←1
  j←taille
  tant que i<j faire
    tmp ← t(i)
    t(i) ← t(j)
    t(j) ← temp
    i ← i+1
    j ← j-1
  ftq
finprocedure
```

2. Ecrire une procédure qui supprime d'un tableau d'entiers la valeur min et la valeur max.

```
procedure  supprimeElement(ES tableau t(N):entier,ES taille:
    entier, E i:entier)
  pour j←1 a taille pas 1
    si t(i)=t(j) alors
      t(i) ← t(taille)
      taille ← taille-1
    fsi
  fpour
finprocedure

procedure supprimerMinMax(ES tableau t(N):entier,ES taille:
    entier)
variables
  i:entier
  imin:entier
  imax:entier
  imin ← 1
```

```
  imax ← 1
  pour i←2 a taille pas 1
    si t(i) < t(imin) alors
      imin ← i
    fsi
    si t(i) > t(imax) alors
      imax ← i
    fsi
  fpour
  supprimeElement(t,taille,imax)
  supprimeElement(t,taille,imin)
finprocedure
```

3. Ecrire une fonction qui à partir de deux tableaux de même taille passés en paramètres, construit un nouveau tableau contenant la somme des éléments de mêmes indices.

```
fonction sommeTab(E tableau t1(N):entier,E tableau t2(N):
    entier,E
taille:entier):tableau(N):entier
variables
  i:entier
  tableau res(N):entier
  pour i←1 a taille pas 1
    res(i) ← t1(i)+t2(i)
  fpour
  retourner res
```

4. Fusion et suppresiion des doublons :
   Soient deux tableaux d'entiers triés par ordre croissant. Ecrire une fonction qui fusionne les deux tableaux en supprimant les doublons. Le tableau final doit bien sur être également trié.

```
fonction fusionTab(ES tableau t1(N):entier,ES tableau t2(M):
    entier, E
nb1:entier, E nb2:entier S taille:entier):tableau(N+M):entier
variables
  i:entier
  j:entier
  tableau res(N+M):entier

  //Ajout des sentinelles
  t1(nb1+1) ← MAX_INT
  t2(nb2+1) ← MAX_INT

  i←1
  j←1
  taille←1
  tant que i ≤ nb1 ou j≤nb2 faire
    si t1(i)=t2(j) alors
      res(taille)←t1(i)
      i← i+1
      j← j+1
    sinonsi t1(i)<t2(j) alors
      res(taille)←t1(i)
```

```
      i←i+1
    sinon
      res(taille)←t2(j)
      j←j+1
    fsi
    taille ← taille+1
  ftq
  retourner res
finfonction
```

5. Nombres premiers :
   Ecrire une fonction qui retourne un tableau contenant l'ensemble des nombres premiers inférieurs à 100. On utilisera le crible d'érastostène dont le principe est le suivant :
   – on initialise la première case du tableau avec le nombre premier 2.
   – on cherche ensuite un nombre premier supérieur à 2 qui possède la propriété de n'avoir que 2 diviseurs distincts (1 et lui-même) et on l'insère dans le tableau
   – on réitère le traitement pour les 100 premiers nombres

```
fonction premier100(S taille:entier):tableau(100):entier
variables
  i:entier
  premier:booleen
  tableau res(100):entier

  taille ←1
  res(taille) ← 2
  pour i ← 3 a 100 pas 2
    j ← 1
    premier ← vrai
    tant que premier et j ≤ taille  faire
      premier ← (mod(i,res(j)) ≠ 0)
      j ← j+1
    ftq
    si premier alors
      taille ← taille + 1
      res(taille) ← i
    fsi
  fpour
  retourner res
finfonction
```

6. Cryptographie :
   (a) Ecrire une fonction permettant de crypter un message selon le code de César dont le principe consiste à décaler chaque lettre du message d'une valeur passée en paramètre. Par exemple, la lettre 'a' décalée de 3 se transforme en la lettre 'd' tout comme la lettre 'w' décalée de 7. On utilisera un tableau pour stocker l'ensemble des lettres de l'alphabet. Pour simplifier le traitement, on supposera que les messages sont des tableaux de caractères.

```
fonction valmod(E tableau t(26):caractere,E c:caractere, E
     d:entier):caractere
variables
  i:entier

  i←1
  tant que c≠t(i) faire
    i ← i+1
  ftq
  retourner t(mod(i+d,26))
finfonction

fonction cesar(E tableau m(N):caractere, E tableau t(26):
    caractere, E
taille:entier, E dec:entier):tableau(N):caractere
variables
  i:entier
  tableau res(N):caractere

  pour i ←1 a taille pas 1
    res(i) ← valmod(t,m(i),dec)
  fpour
  retourner res
finfonction
```

(b) Ecrire une nouvelle fonction de cryptage selon le principe suivant :
- additionner chaque lettre du message avec la lettre suivante de l'alphabet
- normaliser la valeur obtenue pour qu'elle corresponde à une lettre de l'alphabet
- utiliser le code de César sur la valeur obtenue

```
fonction valsuiv(E tableau t(26):caractere,E c:caractere,
    E d:entier):caractere
variables
  i:entier

  i←1
  tant que c≠t(i) faire
    i ← i+1
  ftq
  retourner t(mod(2*i+1+dec,26))
finfonction

fonction crypto(E tableau m(N):caractere, E tableau t(26):
    caractere, E
taille:entier, E dec:entier):tableau(N):caractere
variables
  i:entier
  temp:entier
  tableau res(N):caractere

  pour i←1 a taille pas 1
    res(i) ← valsuiv(t,m(i),dec)
  fpour
  retourner res
```

```
finfonction
```

7. Soit la fonction suivante vue en cours :

```
fonction recherche(tableau tab(15):entier, taille:entier,elem:
    entier):booleen
variables
  debt:entier
  fint:entier
  mil:entier
  debt ← 1                    //1 instructions
  fint ← taille               //1 instruction
  mil ← (fint+debt)/2         //2 instructions
  tant que debt<fint et tab(mil)≠elem faire//2 instructions
  {(tab(debt) ≤ elem ≤ tab(fint)) ou (tab(debt) > elem) ou (
      tab(fint) < elem)
   et (mil=(debt+fint)/2)}
    si tab(mil) < elem alors                    //1 instruction
      debt ← mil+1                               //2 instructions
    sinon
      fint ← mil-1                               //2 instructions
    fsi
    mil ← (fint+debt)/2                          //4 instructions
  ftq                                           //log2(taille) boucles
  retourner tab(mil)=elem                       //1 instruction
finfonction                                     //Coût: 5+9log2(taille)
```

(a) Trouver un invariant pour cette boucle
Preuve de l'invariant :
- vrai en entrée de boucle (propriété des entiers car tableau trié)
- si $tab(mil) < elem$ alors $tab(debt') = tab(mil + 1) \leq elem$
- si $tab(mil) > elem$ alors $tab(fint') = tab(mil - 1) \geq elem$
- $mil' = (debt' + fint')/2$
- donc vrai en sortie de boucle

Valeur de sortie :
- si $debt = fint$ alors $elem = tab(debt) = tab(fint) = tab(mil)$: élément trouvé
- si $debt > fint$ alors $elem < tab(debt)$ ou $elem > tab(fint)$ donc $elem \neq tab(mil)$: élément non trouvé
- si $tab(mil) = elem$: élément trouvé

(b) Quelle est le coût de cette fonction :
- Il faut évaluer le nombre de boucles
- On coupe en deux la taille du tableau à chaque boucle
- On peut donc effectuer un changement de variable en posant (approximation) $taille = 2^n$.
- On en déduit qu'on effectue donc n boucles dans le pire des cas
- Le nombre de boucles en fonction de la taille (paramètre d'entré de la fonction) est donc : $log_2(taille)$
- On en déduit donc un coût majoré par $5+9log_2(taille)$ instructions