DS - Algorithmes et Structures de Données
Mardi 10 janvier 2023

Correction

1. Quand le récursif est plus simple que l'itératif – 5 pts

On dispose d'une liste de mots représentée par la structure de données suivante :

```
Type cellule : Enregistrement
                   mot : chaine
                   suiv : ^cellule
               FinEnregistrement
     liste : ^cellule
```

1.1. Ecrire en pseudo-langage une procédure **récursive** qui écrit les mots de la liste dans l'**ordre inverse**. Expliquer le principe récursif de cette procédure.

Il suffit d'appeler récursivement la procédure jusqu'à rencontrer la fin de liste (nil).

Il y aura empilement des différents appels et ensuite en dépilant et en revenant après l'appel récursif, on affiche.

```
Procedure écrire_liste_rec(E l : liste)
Début
Si l <> nil
      Alors écrire_liste_rec(l^.suiv)
            écrire(l^.mot)
FinSi
Fin
```

1.2. Ecrire en pseudo-langage une procédure **itérative** qui écrit les mots de la liste dans l'**ordre inverse**. Expliquer votre analyse.

Il faut utiliser une autre liste (ou une pile) pour mémoriser les mots dans le bon ordre.

```
Procedure écrire_liste_it(E l : liste)
Var lbis : liste
Début
lbis ← nil
TantQue l <> nil Faire        {on copie la liste à l'envers}
      insérer(lbis,1,l^.mot)  {on copie l dans lbis en tête}
      l ← l^.suiv
FinTantQue
TantQue lbis <> nil Faire
      écrire(lbis^.mot)
      lbis ← lbis^.suiv
FinTantQue
Fin
```

2. Chromosomes – 5 pts

Les chromosomes sont constitués d'ADN qui porte les gènes d'un être vivant. On peut schématiser un chromosome comme 2 brins chacun constitué d'information sous la forme d'une série de lettres (A, C, T et G). On s'intéresse ici à deux opérations qui peuvent s'effectuer sur un brin de chromosome :

- la mutation : une lettre au sein d'un brin de chromosome est substituée à une autre.
- l'enjambement : deux brins de 2 chromosomes différents s'échangent une partie de leur chaîne pour donner 2 nouveaux brins.

Exemple :

```
b1 =        TACTGCCTAGTCGGCGTTCGTTCGCCTTAA
b2 =        TGCGAATCCGTCACGTTGCATCAGGTCCAT
```

Mutation de b1 (en 12) : `TACTGCCTAGT`**`A`**`GGCGTTCGTTCGCCTTAA`

Enjambement de b1 et b2 (en 9) : donne 2 nouveaux brins `c1` et `c2`

```
c1 =  TACTGCCT CGTCACGTTGCATCAGGTCCAT
c2 =  TGCGAATC AGTCGGCGTTCGTTCGCCTTAA
```

On se propose de représenter un brin par les 2 structures de données suivantes :

- Représentation par tableau

  ```
  Type brin : tableau[1..30] de car
  ```

- Représentation par liste chainée

```
      Type code : Enregistrement
                        info : car
                        suiv : brin
                  FinEnregistrement
     brin : ^code
```

2.1. Pour chacune des structures de données, écrire en pseudo-langage une procédure `faire_mutation` qui réalise une mutation sur un brin : l'emplacement et la substitution de la mutation seront choisis au hasard (`random(n)` tire un entier au hasard entre 0 et n-1).

```
Type tab_code : tableau[0..3] de car
{initialisé avec 'A', 'C', 'T', 'G'}

Procédure faire_mutation_tab (E/S b : brin)
Var i, m : entier
      c : car
      t : tab_code
Début
i ← random(30)+1
Repeter
      m ← random(4)
Jusqu'à t[m] <> b[i]
b[i] ← c
Fin

Procédure faire_mutation_liste (E/S b : brin)
Var i, j, m : entier
      c : car
      t : tab_code
Début
i ← random(30)+1
l ← b
Pour j ← 1 à i-1 inc +1 Faire
      l ← l^.suiv
FinPour
Repeter
      m ← random(4)
Jusqu'à t[m] <> l^.info
l^.info ← c
Fin
```

2.2. Pour chacune des structures de données, écrire en pseudo-langage une procédure `faire_enjambement` qui réalise l'échange des parties de brin : le point où se fait l'échange sera tiré au hasard.

```
Procédure faire_enjambement_tab (E b1, b2 : brin, S c1, c2 : brin)
Var i, j: entier
Début
i ← random(30)+1
Pour j ← 1 à i-1 inc +1 Faire {ne rentre pas dans la boucle si i=1}
      c1[j] ← b1[j]
      c2[j] ← b2[j]
FinPour
Pour j ← i à 30 inc +1 Faire
      c1[j] ← b2[j]
      c2[j] ← b1[j]
FinPour
Fin

Procédure faire_enjambement_liste (E b1, b2 : brin, S c1, c2 : brin)
Var i, j: entier
      t, q1, q2 : brin
Début
i ← random(30)+1
c1 ← nil
c2 ← nil
Pour j ← 1 à i-1 inc +1 Faire {ne rentre pas dans la boucle si i=1}
      t ← allouer(code)
```

```
      t^.info ← b1^.info
      Si j=1
            Alors c1 ← t
                  q1 ← t
            Sinon q1^.suiv ← t
                  q1 ← q1^.suiv
      Finsi
      t ← allouer(code)
      t^.info ← b2^.info
      Si j=1
            Alors c2 ← t
                  q2 ← t
            Sinon q2^.suiv ← t
                  q2 ← q2^.suiv
      Finsi
      b1 ← b1^.suiv
      b2 ← b2^.suiv
FinPour
Pour j ← i à 30 inc +1 Faire
      t ← allouer(code)
      t^.info ← b2^.info
      Si j=1
            Alors c1 ← t
                  q1 ← t
            Sinon q1^.suiv ← t
                  q1 ← q1^.suiv
      Finsi
      t ← allouer(code)
      t^.info ← b1^.info
      Si j=1
            Alors c2 ← t
                  q2 ← t
            Sinon q2^.suiv ← t
                  q2 ← q2^.suiv
      Finsi
      b1 ← b1^.suiv
      b2 ← b2^.suiv
FinPour
q1^.suiv ← nil
q2^.suiv ← nil
Fin
```

3. Mots croisés – 10 pts

E	X	A	M	E	N
P	■	N	E	■	I
R	E	G	L	E	■
E	C	L	A	T	E
U	R	E	■	U	T
V	A	■	O	D	E
E	N	T	R	E	■

```
Const m = 7
      n = 6
Type grille : tableau[1..m, 1..n] de car
```

Les cases noires seront représentées par le caractère espace `' '`.

3.1. Ecrire en pseudo-langage une fonction `noirsuiv` qui cherche la case noire suivante dans la grille `g`, en partant de la case de coordonnées `(i,j)`, dans la direction spécifiée par le booléen `horiz` (si `horiz=vrai`, on cherche vers la droite, sinon on cherche vers le bas). La réponse est un numéro de

colonne si `horiz=vrai`, un numéro de ligne si `horiz=faux` et 0 s'il n'y a pas de case noire dans la direction spécifiée.

```
Fonction noirsuiv (g : grille, i,j : entier, horiz : booléen) : entier
Début
TantQue (i<=m) et (j<=n) et (g[i,j]≠' ') Faire
      Si horiz=V
            Alors j←j+1
            Sinon i←i+1
      Finsi
FintantQue
Si g[i,j]≠' '
      Alors retourner(0)
      Sinon Si horiz=V
                  Alors retourner(j)
                  Sinon retourner(i)
            FinSi
FinSi
Fin
```

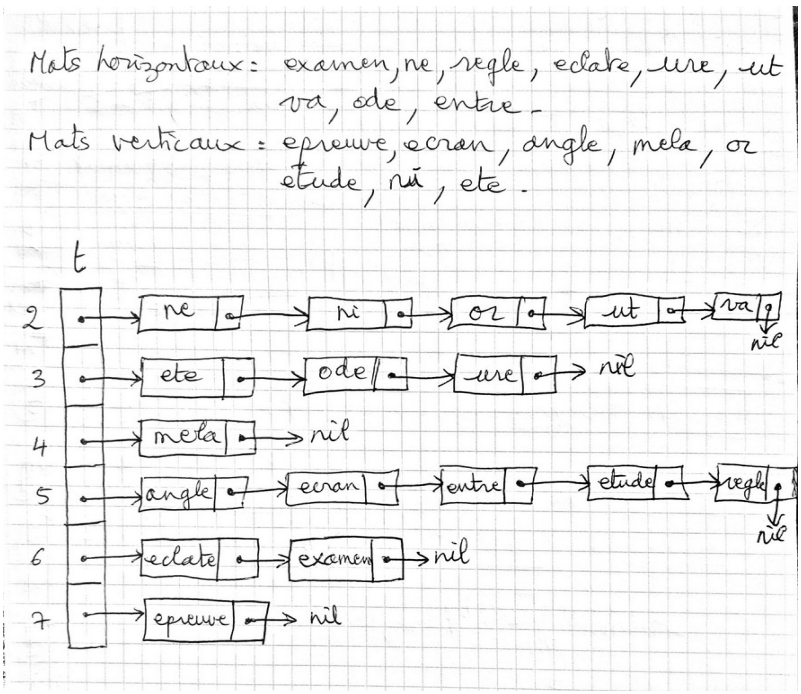
3.2. Seules sont considérées comme mots les suites de 2 lettres ou plus, horizontales ou verticales, (dans l'exemple, les mots sont : examen, ne, règle, éclate, ure, ut, va, ode, entre, épreuve, écran, angle, méla, or, étude, ni, été). Ecrire en pseudo-langage une fonction qui renvoie le nombre de mots de la grille.

```
Fonction comptemot (g : grille) : entier
Var nb, i, j, k : entier
Debut
nb ← 0
i ← 1
j ← 1
{On compte les mots horizontaux}
TantQue i<=m Faire
      TantQue j<=n Faire
            k ← noirsuiv(g,i,j,V)
            Si k=0
                  Alors k ← n+1
            FinSi
            Si(k-j>=2)
                  Alors nb←nb+1
            Finsi
            Si k=n+1
                  Alors j ← n+1
                  Sinon j ← k+1
            FinSi
      FinTantQue
      i←i+1
FinTantQue
{On compte les mots verticaux}
i ← 1
j ← 1
TantQue j<=n Faire
      TantQue i<=m Faire
            k ← noirsuiv(g,i,j,F)
            Si k=0
                  Alors k ← m+1
            FinSi
            Si(k-i>=2)
                  Alors nb ← nb+1
            Finsi
            Si k=n+1
                  Alors i ← m+1
                  Sinon i ← k+1
            FinSi
      FinTantQue
      j←j+1
FinTantQue
Retourner(nb)
Fin
```

3.3. On veut installer les mots dans un tableau de listes : chaque liste contient des mots de même longueur et est **triée dans l'ordre alphabétique**. Pour cela, on définit le type `tabliste` de la façon suivante :

```
Type cellule = Enregistrement
                  mot : chaine
                  suiv : liste
               FinEnregistrement
      liste = ^cellule
      tabliste = tableau[2..m] de liste
```

Faire le dessin correspondant à la grille donnée en exemple.



3.4. Ecrire en pseudo-langage une fonction qui remplit cette structure de données à partir de la grille de mots croisés.

```
Procedure insere (E m : chaine, E/S t : tabliste)
Var p : ^cellule
Debut
q ← allouer(cellule)
q^.mot ← m
Si t[lg(m)]=nil ou m=<t[lg(m)]^.mot
      {insertion en tête}
      Alors q^.suiv ← t[lg(m)]
            t[lg(m)] ← q

      {insertion entre 2 cellules ou en queue}
      Sinon p ← t[lg(m)]
            {On cherche la bonne place}
            TantQue p^.suiv≠nil et p^.suiv^.mot<m Faire
                  p ← p^.suiv
            FinTantQue
            q^.suiv ← p^.suiv
            p^.suiv ← q
FinSi
Fin


Procedure creerTab (E g : grille, S t : tabliste)
Var nb, i, j, k : entier
    mot : chaine
Debut
Pour i←2 à m inc +1 Faire
      t[i]← nil
FinPour
nb ← 0
i ← 1
j ← 1
```

```
{On compte les mots horizontaux}
TantQue i<=m Faire
      TantQue j<=n Faire
            k ← noirsuiv(g,i,j,V)
            Si k=0
                  Alors k ← n+1
            FinSi
            Si(k-j>=2)
                  Alors mot ← ''
                        Pour l ← j à k-1 inc +1 Faire
                              mot ← concat(mot,g[i,l])
                        FinPour
                        insere(mot,t)
            Finsi
            Si k=n+1
                  Alors j ← n+1
                  Sinon j ← k+1
            FinSi
      FinTantQue
      i←i+1
FinTantQue
{On compte les mots verticaux}
i ← 1
j ← 1
TantQue j<=n Faire
      TantQue i<=m Faire
            k ← noirsuiv(g,i,j,F)
            Si k=0
                  Alors k ← m+1
            FinSi
            Si(k-i>=2)
                  Alors mot ← ''
                        Pour l ← i à k-1 inc +1 Faire
                              mot ← concat(mot,g[l,j])
                        FinPour
                        insere(mot,t)
            Finsi
            Si k=n+1
                  Alors i ← m+1
                  Sinon i ← k+1
            FinSi
      FinTantQue
      j←j+1
FinTantQue
Fin
```

3.5. On veut savoir si les mots sont tous différents. Ecrire en pseudo-langage une fonction qui renvoie un booléen selon si les mots de la grille sont tous distincts ou non. Cette fonction utilise la structure de données remplie au 3.4 et **sera optimisée**.

```
{Il faut vérifier que 2 mots consécutifs sont différents dans toutes les listes}
Fonction motsTousDiff (t : tabliste) : booleen
Var   i : entier
      diff : booleen
Debut
i ← 2
diff ← V
TantQue i<=m et diff Faire
      p ← t[i]
      TantQue p≠nil et p^.suiv≠nil et diff Faire
            diff ← (p^.mot≠p^.suiv^.mot)
            p ← p^.suiv
      FinTantQue
      i ← i+1
FinTantQue
retourner(diff)
Fin
```