Algorithmes et Structures de Données

Mardi 24 octobre 2023

Durée 1H30 – Cours et TD NON autorisés

1. Fonction d'Ackermann - 6 pts

```
Fonction Ack (m, n : entier) : entier
Var r : entier
Début
    Si m=0
        Alors r ← n+1
        Sinon Si n=0
                  Alors r ← Ack(m-1,1)
                  Sinon r ← Ack(m-1, Ack(m,n-1))
              FinSi
    FinSi
Retourner(r)
Fin
```

Simuler les **états successifs de la pile** pour l'appel de l'instruction `écrire(Ack(2,1))`.

~~@3, m=0, n=4 r=5~~
~~@3, m=0, n=3 r=4~~
~~@3, m=0, n=2 r=3~~
~~@1, m=0, n=1 r=2~~
~~@2, m=1, n=0 r=2~~
~~@2, m=1, n=1 r=3~~
~~@2, m=1, n=2 r=4~~
~~@3, m=1, n=3 r=5~~
~~@3, m=0, n=2 r=3~~
~~@1, m=0, n=1 r=2~~
~~@2, m=1, n=0 r=2~~
~~@1, m=1, n=1 r=3~~
~~@2, m=2, n=0 r=3~~
~~@0, m=2, n=1 r=5~~

14 appels et écrit 5

2. Calcul d'une fraction continue - 6 pts

On suppose donnés `n+1` nombres entiers strictement positifs $a_0, a_1, \ldots, a_n$ dans un tableau `a` (`a[i]` = a_i, `i = 0...n`). `Type a = tableau[0..100] d'entier`

On souhaite calculer la valeur du nombre :

$$x = a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{\ldots + \cfrac{1}{a_n}}}}$$

2.1 - Ecrire une **fonction itérative** qui calcule la valeur de ce nombre.

```
Fonction frac_it (a : tab, n : entier) : reel
Var i : entier
    x : reel
Début
x ← a[n]
Pour i ← n-1 à 0 inc -1 Faire
    x ← a[i] + 1/x
FinPour
Retourner(x)
Fin
```

2.2 - Ecrire une **fonction récursive** qui calcule la valeur de ce nombre. Expliquer le principe récursif.

```
Fonction frac_rec (a : tab, n, i : entier) : reel
Var x : reel
Début
Si i=n
      Alors x ← a[i]
      Sinon x ← a[i] + 1/frac_rec(a,n,i+1)
FinSi
retourner(x)
Fin
```

Au départ appel de `frac_rec(a,n,0)`

3. Recherche de facteurs - 8 pts

On dispose de `n` chaînes de caractères (toutes de même longueur `lg`) qui résultent d'un « découpage » (avec chevauchement) d'une chaîne initiale. On suppose que le chevauchement est de longueur fixe `k`. On souhaite reconstituer la chaîne initiale à partir des `n` chaînes (version très simplifiée d'un des problèmes de séquencement de l'ADN).
On fait l'hypothèse qu'il existe une unique solution au problème, c'est-à-dire qu'il n'y a qu'un moyen d'agencer les `n` chaînes avec un chevauchement `k`.

Exemple : `lg=9, n=4, k=3`

Soient 4 chaines :

```
C A T T G A C T A       (num=3)
C T A C C T A T G       (num=4)
A C G T G A C A T       (num=2)
A G G C T A A C G       (num=1)
```

Reconstitution :

A G G C T A **A C G** T G A **C A T** T G A **C T A** C C T A T G
les chevauchements sont mis en gras pour mieux comprendre)

3.1. Ecrire en pseudo-langage une procédure qui charge le fichier de nom `nchaines.txt` contenant les `n` chaines (une par ligne) dans un tableau `t` de type `tab (les champs val contiennent les chaines et les champs num sont mis à 0)`

```
Type case = Enregistrement
              val : chaine
              num : entier
            FinEnregistrement
      tab = tableau[1..max] de case  {max ≥ n}
```

```
Procedure Charger (E nomf : chaine, S t : tab, n : entier)
Var f : FT
Début
f ← OuvrirEnLecture(nomf)
n ← 0
TantQue ¬finFichier(f) Faire
      n ← n+1
      t[n].val ← lireChaine(f)
      t[n].num ← 0
FintantQue
Fermer(f)
Fin
```

3.2. Ecrire en pseudo-langage les fonctions `préfixe(c,k)` et `suffixe(c,k)`, qui renvoie la chaine de caractères correspondant au préfixe (ou suffixe) de longueur `k` de la `c`.

```
Fonction préfixe (c : chaine, k : entier) : chaine
Début
retourner(copier(c,1,k))
Fin

Fonction suffixe (c : chaine, k : entier) : chaine
Début
retourner(copier(c,lg(c)-k+1,k))
Fin
```

3.3. Ecrire en pseudo-langage une procédure qui, étant données `n` chaînes de longueur `lg` et le chevauchement `k`, donne en sortie la chaine reconstituée `c`, ainsi que `t` avec les champs `num` remplis avec le numéro d'apparition de la chaine dans `c` (respectivement 3 4, 2 et 1 pour l'exemple). Expliquer la méthode en français.

On commence par chercher la première chaine : celle dont il ne correspond pas une autre chaine ayant comme suffixe le préfixe de celle-ci. Ensuite, on cherche la chaine suivante : celle qui a pour préfixe le suffixe de la précédente. Et on recolle les morceaux.

```
Fonction TrouverPrems (E t : tab, n,k : entier) : entier
Var prems,trouve : booléen
    i, j : entier
Début
i←1
prems ← faux
TantQue i<=n et ¬prems Faire
      j←1
      trouve ← faux
      TantQue j<=n et ¬trouve Faire
            Si i<>j et préfixe(t[i].val,k) = suffixe(t[j].val,k)
                  Alors trouve ← vrai
                  Sinon j ← j+1
            Finsi
      FintantQue
      Si ¬trouve
            Alors prems ← vrai
            Sinon i ← i+1
      Finsi
FintantQue
retourner(i)
Fin
```

```
Fonction TrouverChaineSuiv (t : tab, n, k : entier, c : chaine) : entier
Var trouve : booléen
    i : entier
Début
trouve ← faux
i ← 1
Tantque ¬trouve et i<=n Faire
      Si (t[i].num=0) et (c≠t[i].val) et (suffixe(c,k)=préfixe(t[i].val),k)
            Alors trouve ← vrai
            Sinon i ← i+1
      FinSi
FintantQue
retourner(i)      {le mot suivant existe forcément !}
Fin

Procedure Reconstituer (E n,k : entier, E/S t : tab, S c : chaine)
Var i,j : entier
Début
i ← trouverPrems(t,n,k)
t[i].num ← 1
c ← t[i].val
Pour j←2 à n inc +1 Faire
      i ← TrouverChaineSuiv(t,n,k,t[i].val)
      c ← concatener(c, effacer(t[i].val,1,k))
      t[i].num ← j
FinPour
Fin
```

3.4. Ecrire en pseudo-langage une procédure qui enregistre les résultats dans un nouveau fichier (formaté comme ci-dessous) contenant les `n` chaines (une par ligne) avec en fin de ligne la valeur de `num` correspondante et en dernière ligne du fichier la chaine reconstituée.

```
C A T T G A C T A ; 3
C T A C C T A T G ; 4
A C G T G A C A T ; 2
A G G C T A A C G ; 1
A G G C T A A C G T G A C A T T G A C T A C C T A T G
```

```
Procedure Enregistrer (E t : tab, n : entier, nomf, c : chaine)
Var f : FT
    temp : chaine
    i : entier
Début
f ← CréerFichier(nomf)
Pour i←1 à n inc +1 Faire
      temp ← concaténer(t[n].val,' ; ', entier2chaine(t[n].num))
      écrireChaine(f,temp)
FinPour
écrireChaine(f,c)
Fermer(f)
Fin
```