Algorithmes et Structures de Données

Mardi 8 Novembre 2022

Durée 1H30 – Cours et TD NON autorisés

1. Motus - 10 pts

On souhaite écrire un programme qui joue au jeu MOTUS : retrouver un mot de 4 à 10 lettres en un minimum de coups !

Pour cela, on dispose d'un fichier texte comportant 100 mots permis, triés dans l'ordre alphabétique (un mot par ligne). L'utilisateur choisit un mot permis et le programme doit le deviner : la longueur du mot est donnée ainsi que la première lettre. A chaque étape, le programme propose un mot et l'utilisateur répond par un tableau de valeurs de la longueur du mot :

- 2 : la lettre correspondante est bien placée
- 1 : la lettre correspondante est présente mais mal placée
- 0 : la lettre correspondante n'est pas présente dans le mot

Exemple : mot à trouver = 'bonnet', mot proposé = 'bleues', réponse = 2 0 1 0 2 0

Le programme tient compte des réponses de l'utilisateur pour proposer de nouveaux mots candidats jusqu'à ce qu'il trouve le mot en question (il a gagné) ou qu'il atteigne le nombre maximal (par ex. 5) de propositions (il a perdu).

1.1. Ecrire en pseudo-langage une procédure `charger_tab` qui charge le fichier de mots (dont le nom est dans `nomf`) dans un tableau de mots t de taille effective `nbmots`.

```
Type tab_mots : tableau[1..100] de chaine

Procédure charger_tab (E nomf : chaine  S t : tab_mots, nbmots : entier)
var   f : FT
      c : chaine
Début
f←OuvreEnLecture(f,nomf)
nbmots←0
TantQue  ¬finFichier(f) Faire
 lireChaine(f,c)
  nbmots←nbmots+1
  t[nbmots]←c
FinTantQue
fermer(f)
fin
```

1.2. Ecrire en pseudo-langage une procédure `créer_mots_candidats` qui, à partir de `t`, renvoie un tableau de mots candidats `t_cand` (mots de même longueur et commençant par la même lettre que le mot cherché `mot`) de taille effective `nb`. L'algorithme doit être optimal.

```
Procédure créer_mots_candidats (E t : tab_mots, n : entier, mot : chaine,
                                S t_cand : tab_mots, nb : entier)
Var i : entier
Début
nb←0
i←1
```

```
TantQue (mot[1]<>t[i][1]) et (i<=n) Faire
  i←i+1
FinTantQue
TantQue (mot[1]=t[i][1]) et (i<=n) Faire
  Si (lg(mot)=lg(t[i]))
      Alors nb←nb+1
            t_cand[nb]←t[i]
  FinSi
  i←i+1
FinTantQue
Fin
```

1.3. Ecrire en pseudo-langage une fonction `correspondre` qui renvoie `vrai` ou `faux` selon si le mot `m1` correspond au mot `m2` (formé de lettres et de `'?'`), contient les lettres de `oui` (lettres présentes mais mal placées) et ne contient pas les lettres de `non` (lettres non présentes).

```
m1='bleues' ; m2='b???es' ; oui='l' ; non='rgt' ; m1 et m2 matchent
m1='bleues' ; m2='br??es' ; oui='l' ; non='rgt' ; m1 et m2 ne matchent pas
m1='brumes' ; m2='b???es' ; oui='u' ; non='m' ; m1 et m2 ne matchent pas

Fonction appartient (m1,m2,m3 : chaine) : booléen
var i : entier
    okoui, oknon : booléen
Début
okoui←vrai
i←1
TantQue okoui et (i<=lg(m1)) Faire
  okoui←pos(m1[i],m3)<>0
  i←i+1
FinTantQue
oknon←vrai
i←1
TantQue oknon et (i<=lg(m2)) Faire
  oknon←pos(m2[i],m3)=0
  i←i+1
FinTantQue
Retourner(okoui et oknon)
Fin

Fonction correspond (m1,m2,oui,non : chaine) : booléen
Var   ok : booléen;
      i : entier
Début
ok←vrai
i←1
Si appartient(oui, non, m1)
   Alors
      TantQue ok et (i<=lg(m1)) Faire
         ok:= (m1[i]=m2[i]) or (m2[i]='?')
         i←i+1
      FinTantQue
   Sinon ok:=false
FinSi
retourner(ok)
Fin
```

1.4. Ecrire en pseudo-langage une procédure `cherche_cand` qui cherche un mot `s` qui corresponde à `m`, `oui` et `non`, à partir du tableau de mots candidats `t_cand`.

```
Procédure chercher_cand(E t_cand : tab_mots, nb : entier, oui, non, m : chaine,
                        S s : chaine)
Var i : entier
Début
i:=1
TantQue ¬correspond(t_cand[i],m,oui,non) et (i<=nb) Faire
      i←i+1
FinTantQue
Si i<=nb Alors s←t_cand[i]
         Sinon s←''
FinSi
Fin
```

1.5. Ecrire en pseudo-langage le programme complet qui implémente l'interaction entre l'ordinateur et l'utilisateur jusqu'au succès ou l'échec du jeu.

```
Programme motus
Var t, t_cand : tab_mots
      val : array[1..10] d'entier
      mot, mot_cand, sol, oui, non : chaine
      i, n, nb, coup : entier
      fini : booléen
Début
charger_tab('fmots.txt', t, n)
écrire('Choisissez un mot possible :  ')
lire(mot)
charger_tab('fmots.txt', t, n)
mot_cand[1]←mot[1]
Pour i←2 à lg(mot) inc +1 Faire
      mot_cand[i]←'?'
FinPour
créer_mots_candidats(t, n, mot, t_cand, nb)
oui←mot[1]
non←''
fini←faux
coup←1
TantQue ¬fini et (coup<=5) Faire
      chercher_cand(t_cand, nb, oui, non, mot_cand, sol)
      écrire('je propose ',sol)
      Pour  i←1 à lg(sol) inc +1 Faire
            lire(val[i])
            Selon val[i]
              0 : Si pos(sol[i],non)=0
                        Alors concaténer(non,sol[i])
                      FinSi
              1 : Si pos(sol[i],oui)=0
                        Alors concaténer(oui,sol[i])
                      FinSi
              2 : mot_cand[i]←sol[i]
            FinSelon
      FinPour
      fini←sol=mot
      coup←coup+1
      Si fini
        Alors écrire('GAGNE !!')
```

```
         Sinon Si coup>5
                  Alors écrire('PERDU !!')
               FinSi
      FinSi
FinTantQue
Fin
```

2. Pile - 5 pts

```
Fonction Cantor(x,y : entier): entier
Si x=0 et y=0
      Alors res ← 0
      Sinon Si y=0
                  Alors res ← Cantor(0,x-1){@1} + 1
                  Sinon res ← Cantor(x+1,y-1){@2} + 1
            FinSi
FinSi
Retourner(res)
Fin
```

Simuler la pile pour l'appel `écrire(Cantor(3,2)){@0}`

~~@1, x=0 y=0~~ res=0
~~@2, x=1 y=0~~ res=1
~~@1, x=0 y=1~~ res=2
~~@2, x=2 y=0~~ res=3
~~@2, x=1 y=1~~ res=4
~~@1, x=0 y=2~~ res=5
~~@2, x=3 y=0~~ res=6
~~@2, x=2 y=1~~ res=7
~~@2, x=1 y=2~~ res=8
~~@1, x=0 y=3~~ res=9
~~@2, x=4 y=0~~ res=10
~~@2, x=3 y=1~~ res=11
~~@2, x=2 y=2~~ res=12
~~@2, x=1 y=3~~ res=13
~~@1, x=0 y=4~~ res=14
~~@2, x=5 y=0~~ res=15
~~@2, x=4 y=1~~ res=16
~~@0, x=3 y=2~~ res=17

3. Récursivité – 5 pts

On souhaite écrire à l'écran les chaines de caractères suivantes selon n :

Pour n=0 : `'0=0'`
n=1 : `'0=(0+0)'`
n=2 : `'0=((0+0)+(0+0))'`
n=3 : `'0=(((0+0)+(0+0))+((0+0)+(0+0)))'`
…

3.1. Expliquer le principe récursif.

La partie récursive de la chaine commence après le `'='`. Les 2 premiers caractères seront donc affichés dans le programme principal. La variable de récursivité est `n`. On doit écrire une `'('` puis appeler récursivement la fonction avec `n-1` puis écrire `'+'`, rappeler la fonction avec `n-1` et enfin écrire une `')'`. La condition d'arrêt est `n=0`, pour laquelle on doit écrire `0`.

3.2. Ecrire en pseudo-langage une procédure **récursive** qui effectue cet affichage.

```
Procédure teteToto(E n : entier)
Début
Si (n=0)
 Alors écrire('0')
 Sinon
      écrire('(' )
      teteToto(n-1)
      écrire('+')
      teteToto(n-1)
      écrire(')')
FinSi
Fin

Programme Toto
Var n : entier

Début
ecrire("Entrer un entier ? ")
lire(n)

écrire('0=')
teteToto(n)

Fin
```