RN40 : Rapport de projet tri seau

Chausson Thibault

16 décembre 2021

Table des matières

Liste des algorithmes

Liste des codes sources

1 Mise en place du programme

Dans cette partie, nous verrons les structures utilisées et par la même occasion les algorithmes utilisés.

1.1 Structures

Avant de commencer à parler d'algorithmique, accordons nous sur les structures de données, que seront utilisées dans la suite de ce projet. Ainsi, j'ai décidé de créer une structure « élément d'un seau », qui me permet de stocker le nombre saisis par l'utilisateur sous forme de caractères.

```
1 typedef struct elem {
2     char nombre[NbChiffre];
3     struct elem *suivant;} element;
```

Par la suite, j'ai utilisé cette structure pour créer une liste chaînée de ces éléments afin de créer le type seau.

```
1 typedef element *seau;
```

Une fois cette structure définie, nous pouvons commencer la partie algorithmique.

1.2 Algorithmique

1.2.1 Fonction : est_vide

Cette fonction a pour but de déterminer si un seau (liste chainée de chaines de caractères) est vide ou non. Par la suite, nous pourrons utiliser cette fonction dans « afficher », ce qui nous permettra de déterminer rapidement si un seau est vide ou non et de l'afficher à l'utilisateur si besoin.

Algorithme 1 *est_vide*(*seau s*) → *booléen*

Donnée(s) : un seau (s)
Résultat(s) : un booléen (res)
1: $res \leftarrow faux$
2: **si** $s = indéfinie$ **alors**
3: $res \leftarrow vrai$
4: **fin si**

En résumé, cette fonction retourne un booléen « vrai » si la liste est vide, sinon « faux ».

1.2.2 Fonction : ajout_queue

Cette fonction permet d'ajouter un élément en queue d'un seau (un nombre en chaine de caractères au bout d'une liste chainée de type seau). Cette procédure va nous être utilie, par exemple dans le main, nous l'utiliserons pour remplir le seau de nombres à trier. De plus, dans la fonction *tri_seau* elle nous permettra d'ajouter le nombre que l'on trie dans le seau qui lui correspond. Nous allons en fin de la liste grâce au « tant que », et arrivée au dernier élément nous le relions avec le nouvel élément à ajouter.

Algorithme 2 $ajout\_queue(seau\ s,\ chaine\ de\ caractères\ a) \rightarrow seau$

Donnée(s) : un seau (s), une chaine de caractères (a) de taille 10
Résultat(s) : un seau (s)

1: $nouvel\_élément = créer\_seau()$
2: $nombre(nouvel\_élément) \leftarrow a$
3: $suivant(nouvel\_élément) \leftarrow indéfinie$
4: **si** $s = indéfinie$ **alors**
5: $\quad s \leftarrow nouvel\_élément$
6: **sinon**
7: $\quad t \leftarrow tête(s)$
8: $\quad$ **tant que** $suivant(t) \neq indéfinie$ **faire**
9: $\quad\quad t \leftarrow suivant(t)$
10: $\quad$ **fin tant que**
11: $\quad suivant(t) \leftarrow nouvel\_élément$
12: **fin si**

Ainsi, nous retournons une liste chainée avec le nombre ajouté à la queue de ce seau.

1.2.3 Fonction : taille

Nous allons écrire une fonction qui permet de déterminer la taille maximale des chaines de caractères d'un seau. En connaissant la taille maximale en quantité de caractères d'un nombre, nous allons pouvoir ajouter la bonne quantité de zéro en début de celui-ci. Cette opération sera réalisée par la fonction « ajout_manquant ». Nous définissons que si un seau est vide alors la taille de sa plus grande chaine de caractères est « 0 ». De plus, nous lisons

tout le seau seau grâce au « tant que », et on regarde si la taille du nombre est plus grandes que les précédents (en nombre de caractères).

Algorithme 3 $taille(seau\ s) \rightarrow entier$

Donnée(s) : un seau (s)
Résultat(s) : un entier (t) qui contient la taille de la plus grande chaine de caractères d'un seau

```
1: t ← 0
2: si est_vide(s) = vrai alors
3:     t ← 0
4: sinon
5:     p ← créer_seau()
6:     p ← tête(s)
7:     tant que p ≠ indéfinie faire
8:         si longueur(nombre(p)) > t alors
9:             t ← longueur(nombre(p))
10:        fin si
11:        p ← suivant(p)
12:    fin tant que
13: fin si
```

1.2.4 Fonction : ajout_manquant

Nous implémentons une fonction qui permet d'ajouter des « 0 » au début de chaque nombre (comme dans l'exemple d'exécution du sujet). Pour ce faire, si le seau est non vide nous regardons chaque nombre, et si nous voyons que sa taille en quantité de caractères est inférieure à la taille maximale du nombre de caractères, nous ajoutons le nombre de « 0 » nécessaire.

Algorithme 4 $ajout\_manquant(seau\ s) \rightarrow seau$

Donnée(s) : un seau (s)
Résultat(s) : un seau avec des nombres ayant tous autant de chiffres (mais sans en changer la valeur)

```
1: t ← taille(s)
2: si est_vide(s) = faux alors
3:     p ← créer_seau()
4:     p ← tête(s)
```

```
    tant que p ≠ indéfinie faire
      tBis ← longueur(nombre(p))
      si tBis ≠ t alors
        pour k allant de 0 à (t − tBis) faire
          aux ← "0"
          nombre(p) ← concaténation(aux, nombre(p))
        fin pour
      fin si
      p ← suivant(p)
    fin tant que
fin si
```

1.2.5 Fonction : correspondance

Nous allons écrire une fonction qui permet d'associer à un caractère un nombre lui correspondant. En hexadécimal le « a » représente la valeur « 10 », le « b » la valeur « 11 », ... Ce programme est aussi faisable avec des « switch » et des « case ».

Algorithme 5 *correspondance*(*char a*) → *entier*

Donnée(s) : caractère (a)
Résultat(s) : un entier (b)

```
b ← −1
si a =′ 0′ alors
  b ← 0
sinon si a =′ 1′ alors
  b ← 10
fin si
etc...
```

Ainsi de suite pour la fonction correspondance.

1.2.6 Fonction : tri_seau

Elle a pour but de trier la liste des nombres donnée par l'utilisateur. Pour en résumer rapidement le fonctionnement, on crée un tableau de b seau (où b correspond à notre base de travail), nous lisions le dernier caractère du nombre que nous étudions (notons le c), et on met ce nombre dans la $c^{ème}$ case du tableau. Ensuite nous passons à l'avant dernier caractère (notons le c), et nous le mettons dans la $c^{ème}$ case du tableau. Et ainsi de suite pour les caractères.

Algorithme 6 *tri_seau*(*seau s, entier base*) → *seau*

Donnée(s) : un seau (s) et un entier (base)
Résultat(s) : un seau trié

```
1: t ← taille(s)
2: T ← créer_tableau_seau[base]()
3: si est_vide(s) = faux alors
4:    p ← tête(s)
5:    tant que p ≠ indéfinie faire
6:       i ← correspondance(nombre(p)[t])
7:       T[i] ← ajout_queue(T[i], nombre(p))
8:       p ← suivant(p)
9:    fin tant que
10:   pour k allant de 0 à (t − 1) faire
11:      T_aux ← créer_tableau_seau[base]()
12:      pour j allant de 0 à base faire
13:         si est_vide(T[j]) = faux alors
14:            q ← tête(T[j])
15:            tant que q ≠ indéfinie faire
16:               i ← correspondance(nombre(p)[t − k − 2])
17:               T_aux[i] ← ajout_queue(T[i], nombre(q))
18:               q ← suivant(q)
19:            fin tant que
20:         fin si
21:      fin pour
22:      pour x allant de 0 à base faire
23:         T[x] ← T_aux[x]
24:      fin pour
25:   fin pour
26: fin si
```

Donc, cet algorithme trie la liste des seaux comme demandé.

1.2.7 Fonction : afficher

Cette fonction imprime à l'écran les éléments contenu dans un seau.

Algorithme 7 $affiche r(seau\ s) \rightarrow impression$

Donnée(s) : un seau (s)
Résultat(s) : aucun (sauf une « impression » du seau à l'écran)

si $est\_vide(s) = vrai$ **alors**
  $imprimer(”Le\ seau\ est\ vide”)$
sinon
  $p = créer\_seau()$
  $p \leftarrow tête(s)$
  $imprimer(”[”)$
  tant que $suivant(p) \neq indéfinie$ **faire**
    $imprimer(nombre(p)\ ;)$
    $p \leftarrow suivant(p)$
  fin tant que
  $imprimer(nombre(p)\ )$
  $imprimer(”]”)$
fin si

Ainsi, le seau est affiché à l'écran.

2 Codes sources

2.1 Le code du header : « structures » :

```
1  //
2  // Created by Thibault on 01/11/2021.
3  //
4
5  #ifndef TEST_STRUCTURE_H
6  #define TEST_STRUCTURE_H
7
8  //Quantité de chiffres maximale  par nombre
9  #define NbChiffre 11 //Ici 11 car, il met un "\0" à la fin
10
11 #define TRUE 1
12 #define FALSE 0
13
```

```
14  typedef int BOOL;
15
16  //Déclaration de la structure seau
17
18  typedef struct elem {
19      char nombre[NbChiffre];
20      struct elem *suivant;} element;
21
22  typedef element *seau;
23
24  #endif //TEST_STRUCTURE_H
```

Code source 1 – structures.h

2.2 Le code du header : « fonctions » :

```
1   //
2   // Created by Thibault on 01/11/2021.
3   //
4
5   #ifndef TEST_FONCTION_H
6   #define TEST_FONCTION_H
7   #include<stdio.h>
8   #include <string.h>
9   #include "structure.h"
10
11  BOOL est_vide (seau s);
12
13  seau ajout_queue (seau s, char a[NbChiffre]);
14
15  int taille(seau s);
16
17  seau ajoute_manquant(seau s);
18
19  int correspondence (char a);
20
21  void tri_seau(seau s, int B, seau T[]);
22
23  void afficher (seau s);
24
25  #endif //TEST_FONCTION_H
```

Code source 2 – fonctions.h

2.3 Le code du fichier .c : « fonctions » :

```c
//
// Created by Thibault on 31/10/2021.
//

#include "fonction.h"
#include "structure.h"
#include <stdlib.h>

BOOL est_vide(seau s){
    if (s==NULL) /*Si c'est vide nous le mettons à vrai*/ {
        return TRUE;
    }
    else {
        return FALSE;
    }
}

seau ajout_queue (seau s, char a[NbChiffre]){
    seau nouvel_element;
    seau t;
    nouvel_element=(element *) malloc(sizeof(element));
    snprintf(nouvel_element->nombre, sizeof a, "%s", a); /*
    Nous mettons la chaine de caractère "a" dans la partie
    nombre du nouvel_element*/
    nouvel_element->suivant=NULL; /*Comme c'est le dernier
    nous le mettons à NULL*/
    if(s == NULL) {
        s=nouvel_element;
    }
    else {
        t = s;
        while (t->suivant !=NULL) /*Nous allons jusqu'à la fin
    */ {
            t=t->suivant;
        }
        t->suivant=nouvel_element; /*Nous le mettons à la fin
    */
    }
    return (s);
}


int taille (seau s){
    int t=0;
    if (est_vide(s)==TRUE)/*Si c'est vide la taille est nulle
    */{
```

```
41          t=0;
42      }
43      else {
44          seau p = s;
45          while (p != NULL) {
46              if (strlen(p->nombre) > t) /*Nous regardons la
    taille si elle est plus grande nous la mettons dans t*/ {
47                  t=strlen(p->nombre);
48              }
49              p = p->suivant;
50          }
51      }
52      return(t);
53  }
54
55  seau ajoute_manquant(seau s){
56      int t=taille(s);
57      int k;
58      int tBis;
59
60      if (est_vide(s)!=TRUE){
61          seau p = s;
62          while (p!= NULL) {
63              tBis=strlen(p->nombre);
64              if (tBis!=t) /*Si nous ne sommes pas à la taille
    maximale nous ajoutons des 0*/ {
65                  for (k=0;k<(t-tBis);k++)/*Nous ajoutons le bon
     nombre de 0*/{
66                      char aux[NbChiffre]="0";
67                      strcat(aux,p->nombre);
68                      snprintf(p->nombre, sizeof p->nombre, "%s"
    , aux);
69                  }
70              }
71              p=p->suivant;
72          }
73      }
74      return(s);
75  }
76
77
78  int correspondence (char a)/*Nous faisons correspondre un
    caractère à un nombre de 0 à 15*/{
79      int b;
80      if (a=='0'){
81          b=0;
82      }
83      else if (a=='1'){
```

```
84              b=1;
85          }
86          else if (a=='2'){
87              b=2;
88          }
89          else if (a=='3'){
90              b=3;
91          }
92          else if (a=='4'){
93              b=4;
94          }
95          else if (a=='5'){
96              b=5;
97          }
98          else if (a=='6'){
99              b=6;
100         }
101         else if (a=='7'){
102             b=7;
103         }
104         else if (a=='8'){
105             b=8;
106         }
107         else if (a=='9'){
108             b=9;
109         }
110         else if (a=='a' || a=='A'){
111             b=10;
112         }
113         else if (a=='b'|| a=='B'){
114             b=11;
115         }
116         else if (a=='c'|| a=='C'){
117             b=12;
118         }
119         else if (a=='d'|| a=='D'){
120             b=13;
121         }
122         else if (a=='e'|| a=='E'){
123             b=14;
124         }
125         else if (a=='f'|| a=='F'){
126             b=15;
127         }
128         return(b);
129     }
130
131
```

```
132 void tri_seau(seau s, int B, seau T[]){
133
134     int t=taille(s);
135     int k;
136     int i;
137     int j;
138     int y;
139
140     //Initialisation
141     for (y=0;y<B;y++) /*Nous initialisons le tableau de seau*/
        {
142         T[y]=(element *) malloc(sizeof(element));
143         T[y]=NULL;
144     }
145
146     if (est_vide(s)==FALSE){
147         seau p=s;
148         while (p!=NULL)/*Nous ajoutons les éléments du seau à
        trier dans le tableau*/{
149             i=correspondence((p->nombre)[t-1]);/*Nous
        regardons dans quelle case du tableau nous devons reporter
         le nombre*/
150             T[i]=ajout_queue(T[i],p->nombre);/*Nous reportons
        le nombre*/
151             p=p->suivant;
152         }
153
154         /*Nous trions de la même manière que l'initialisation,
         mais en avançant vers les chiffres de gauche*/
155         for (k=0;k<t-1;k++){
156             seau T_aux[B];
157             for (y=0;y<B;y++)/*Initialisation du tableau*/{
158                 T_aux[y]=(element *) malloc(sizeof(element));
159                 T_aux[y]=NULL;
160             }
161             for (j=0;j<B;j++){
162                 if (est_vide(T[j])==FALSE){
163                     seau q=T[j];
164                     while (q!=NULL){
165                         i=correspondence((q->nombre)[t-k-2]);
166                         T_aux[i]=ajout_queue(T_aux[i],q->
        nombre);
167                         q=q->suivant;
168                     }
169                 }
170             }
171             int x;
172             for (x=0;x<B;x++){
```

```
173                     T[x]=T_aux[x];
174                 }
175             }
176         }
177     }
178
179     void afficher (seau s) {
180         if (est_vide(s)==TRUE){
181             printf("Est vide seau\n"); /*Si le seau est vide*/
182         }
183         else {
184             seau p = s;
185             printf("[ ");
186             while (p->suivant != NULL) /*Nous avançons et nous
        affichons*/ {
187                 printf("%s\t; ", p->nombre);
188                 p = p->suivant;
189             }
190             printf(" %s ]\n", p->nombre); /*Nous affichons le
        dernier*/
191         }
192     }
```

Code source 3 – fonctions.c

2.4 Le code du fichier : « main.c » :

```
1   #include "structure.h"
2   #include "fonction.h"
3
4   int main(void)
5   {
6       int B;
7       char nb[NbChiffre];
8       seau Tab_aux=NULL;
9       int total;
10      int k=0;
11      BOOL base=TRUE;
12      int longueur;
13      int lo=0;
14      char continu='o';
15
16      printf("Dans quelle base travaille-t-on (entre 2 et 16) ?
        \n");
17      scanf("%d", &B);
18
```

```
19      while(B>16 || B<2)/*Si la base saisie par l'utilisateur
        est incorrecte*/{
20          printf("Dans quelle base travaille-t-on (entre 2 et
        16) ? \n");
21          scanf("%d", &B);
22      }
23
24      while (continu=='o'){
25          printf("Quelle est la valeur ? \n");
26          scanf("%s", nb);
27          longueur= strlen(nb);
28          while( base && lo<longueur) /*Nous vérifions que le
        nombre donné par l'utilisateur soit dans la bonne base et
        qu'il ne dépasse pas les 10 caratères*/{
29              if (correspondence(nb[lo])>B-1)/*Si un des caratè
        res n'est pas un caractère conforme à la base*/{
30                  base=FALSE;
31              }
32              lo=lo+1;
33          }
34          if (base) /*S'il répond à toutes les conditions nous l
        'ajoutons dans la liste des nombres à trier*/ {
35              Tab_aux = ajout_queue(Tab_aux, nb);
36              printf("Avez-vous d'autre nombre à trier ? (o/n)\n
        ");
37              scanf("%s",&continu);
38          }
39          else /*Nous informons l'utilisateur que le nombre
        saisi ne convient pas*/ {
40              base=TRUE;
41              printf("Le nombre saisi n'est pas de la bonne base
         !! \n");
42              printf("Avez-vous d'autre nombre à trier ? (o/n)\n
        ");
43              scanf("%s",&continu);
44          }
45      }
46
47      Tab_aux=ajoute_manquant(Tab_aux); /*Nous créons un seau ou
        tous les nombres ont la même quantité de caractères*/
48
49      seau p[B];
50      int i;
51      tri_seau(Tab_aux,B, p); /*Nous trions le seau*/
52
53      for ( i = 0; i < B; i++ ) /*Nous affichons les nombres tri
        és*/ {
54          afficher (p[i] );
```

```
55      }
56      return 0;
57  }
```

Code source 4 – main.c