FACULTÉ DES SCIENCES ET TECHNIQUES
DE MOHAMMEDIA
UNIVERSITÉ HASSAN II DE CASABLANCA

Ingénierie Logicielle et Intégration
des Systèmes Informatiques

Compte Rendu

TP2

Algorithmes de Trie

Filière Ingénieur :

Ingénierie Logicielle et Intégration des Systèmes Informatiques

Réalisé par :

OUTGOUGA Jalal eddine

EZADDI Abdelmajid

Encadré par :

Prof. Abdelkrim BEKKHOUCHA

2022/2023

Table de matières

Tri à Bulles

Analyse :

Cette méthode consiste à parcourir les éléments en permutant chaque élément par son successeur s'il est plus grand jusqu'à la fin de la liste afin de placer le max en dernier rang et après on place avant ce dernier l'élément plus petit et plus grand que les autres éléments, et on continue ainsi jusqu'à l'obtention d'une liste complétement triée.

Exemple :

Nous prenons la liste suivante : 8 11 3 15

1ère itération :

(8 11 3 15) puisque 8<11 on ne fait rien.

(8 3 11 15) puisque 11>3 on les permute.

On ne fait rien puisque 11<15

2ème itération :

(8 11 3 15) puisque 8>3 les permute.

(3 8 11 15) puisque 18<11 on ne fait rien.

3ème itération :

(3 8 11 15) puisque 8>3 on ne fait rien.

Analyse fonctionnelle :

Cette fonction permet d'échanger 2 variables entières.

```
echange(a,b:entier)
Variables:
                aux entiers.
Début:
        aux=a;
        a=b;
        b=aux;
Fin.
```

```
triABullesTab(T:MaTable)
Variable:
        ind1,ind2:entier.
        Tri:boolean.
Début:
        tri← faux;
        tant que (non(tri)) alors:
        Début:
            tri← vrai;
            pour ind1 allant de T.nbElem à 1 par -1 faire:
                pour ind2 allant de 1 à ind2-1 faire:
                    si(T.tab[ind2]>T.tab[ind2+1])alors:
                    Début:
                      echange(T.tab[ind2],T.tab[ind2+1]);
                       tri← faux;
                    Finsi
        Fin tant que.
Fin.
```

Dossier de programmation :

a. Echange:

```
1     void echange(int *a,int *b)
2     {
3          int aux;
4
5          aux=*a;
6          *a=*b;
7          *b=aux;
}
```

b.mise en œuvre à l'aide du tableau:

```
8       void triABullesTab(MaTable *table)
9       {
10          int ind1,ind2,trie;
11
12          trie=0;
13          ///tant que le tableau n'est pas trié.
14          while(!trie)
15          {
16              trie=1;
17              ///la position où on s'arrête.
18              for(ind1=table->nbElem-1;ind1>=0;ind1--)
19                 ///on parcours la table.
20                  for(ind2=0;ind2<ind1;ind2++)
21                ///si l'element est sup à son successeur.
22                      if(table>tab[ind2]>table>tab[ind2+1])
23                    {
24                ///on      échange      et      on      met      tri      à      0.
echange(&table>tab[ind2],&table>tab[ind2+1]);
25                          trie=0;
26                    }
27          }
28      }
```

c.mise en œuvre à l'aide du pointeur :

```
Cellule*triABullesPoint(Cellule *liste)
1       {
2         Cellule *crt1,
3              *crt2,
4              *dernierElem;
5         int trie=0;
6
7         while(!trie)
8         {
9           trie=1;
10          dernierElem=dernier(liste);
11          ///la position où on s'arrête.
12          for(crt1=dernierElem;crt1->precedent;crt1=crt1->precedent)
13          {
14            ///on parcours la table.
15            for(crt2=liste;crt2!=crt1;crt2=crt2->suivant)
16              ///si l'element est sup à son successeur.
17              if(crt2->elem>crt2->suivant->elem)
18              {
19                ///on échange et on met tri à 0.
20                echange(&crt2->elem,&crt2->suivant->elem);
```

```
21                  trie=0;
22                }
23            }///fin for.
24
25          }///fin while(!trie).
26          return (Cellule*)liste;
27      }///fin triABullesPoint(Cellule *liste).
```

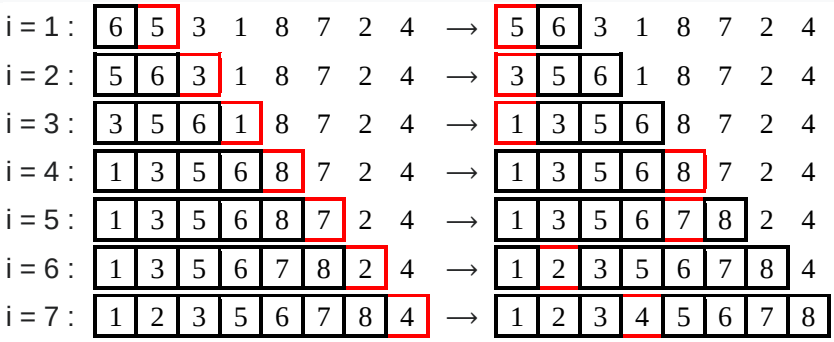
Trie par insertion

Analyse :

Le tri par insertion considère chaque élément du tableau et l'insère à la bonne place parmi les éléments déjà triés. Ainsi, au moment où on considère un élément, les éléments qui le précèdent sont déjà triés, tandis que les éléments qui le suivent ne sont pas encore triés.

Pour trouver la place où insérer un élément parmi les précédents, il faut le comparer à ces derniers, et les décaler afin de libérer une place où effectuer l'insertion. Le décalage occupe la place laissée libre par l'élément considéré. En pratique, ces deux actions s'effectuent en une passe, qui consiste à faire « remonter » l'élément au fur et à mesure jusqu'à rencontrer un élément plus petit.

Voici les étapes de l'exécution du tri par insertion sur le tableau [6, 5, 3, 1, 8, 7, 2, 4]. Le tableau est représenté au début et à la fin de chaque itération.



Analyse fonctionnelle :

triInsert(T:MaTable)

Variables:

i,j:entiers.

Début:

pour i allant de 1 à T.nbElem faire:

j← i;

tant que((T.tab[j-1]>T.tab[j])et j>0) faire:

echange(T.tab[j],T.tab[j-1]);

j← j-1;

Fin tant que.

Fin pour.

Fin

Dossier de programmation :

a.mise en oeuvre à l'aide du tableau:

```
1     .int triInsert(MaTable *table)
2     {
3         int ind1,ind2;
4         ///si la table n'existe pas.
5         if(!table)  return (int)0;
6
7         ///sinon on commence le tri.
8         for(ind1=1;ind1<table->nbElem;ind1++)
9         {
10            ind2=ind1;
11            ///tant que l'élément est inf à son précedeur.
12            while((table->tab[ind2]<table->tab[ind2-1])&&(ind2>0))
13            {   ///on les échange.
14                echange(table->tab+ind2,table->tab+ind2-1);
15                ind2--;
16            }
17        }
18        return (int)1;
19    }
```

b.mise en œuvre à l'aide du pointeur:

```
20    Cellule *triInsertPoint(Cellule *liste)
21    {
22        Cellule *crt1,
23                *crt2;
24        ///on commence le tri.
25        for(crt1=liste->suivant;crt1;crt1=crt1->suivant)
26        {
27            crt2=crt1;
28            ///tant que l'élément est inf à son précedeur.
29            while(crt2->elem<crt2->precedent->elem)
30            {
31                ///on les échange.
32                echange(&crt2->elem,&crt2->precedent->elem);
33                crt2=crt2->precedent;
34                if(!crt2->precedent)    break;
35            }///fin while(crt2->elem<crt2->precedent->elem).
36        }///fin for(crt1=liste->suivant;crt1;crt1=crt1->suivant).
37        return (Cellule*)liste;
38    }
```

Trie par sélection

Analyse :

Dans un tableau on effectue les étapes suivantes jusqu'à on obtient un tableaux trie

- Rechercher le plus petit élément du tableau, et l'échanger avec l'élément d'indice 0 ;
- Rechercher le second plus petit élément du tableau, et l'échanger avec l'élément d'indice 1 ;
- Continuer de cette façon jusqu'à ce que le tableau soit entièrement trié.

Voici les étapes de l'exécution du tri par insertion sur le tableau [7 ,2, 8, 1, 4]. Le tableau est représenté au début et à la fin de chaque itération.

I1=[7 ,2, 8, 1, 4]

I2=[1 ,2, 8, 7, 4]

I2=[1 ,2, 8, 7, 4]

I2=[1 ,2, 4, 7, 8]

I2=[1 ,2, 4, 7, 8]

Analyse fonctionnelle :

```
Tri_par_selection (table:MaTable)
variables:
ind,tmp,ind_min:entiers
Début:
    pour (ind allant de 1 à (NbElem-1)) faire
    début
        ind_min← emp_min(table,ind);
        si(ind_min<>ind)
                tmp← table->tab[ind];
                table->tab[ind]<- table->tab[ind_min];
                table->tab[ind_min]<- tmp;
        finSi
    fin
Fin

emp_min(table:MaTable,indice:entier)
varieble:
   crt,ind_min:entiers
début
    ind_min← indice;
    pour (crt allant de (indice+1)à  NbElem) faire
    début
        si(table→ tab[crt]<table→ tab[ind_min]) alors
       ind_min←crt;
      fin
retourner(ind_min);
fin
```

Dossier de programmation :

29 a.mise en oeuvre à l'aide du tableau:

```
30  int emp_min (MaTable t,int pos)
31  {
32      int crt,,//pour le parcour
33      ind_min;//pour garder l'indice du min
34      ind_min=pos;//initialisation
35      //recherche du min
36      for(crt=(pos+1);crt<t->nbElem;crt++)
37      {
38          if(t->tab[crt]<t->tab[ind_min]) ind_min=crt;
39      }///fin for(crt=(pos+1);crt<t->nbElem;crt++)
40      return (int)ind_min;
41  }///fin int emp_min (MaTable t,int pos)
```

42

```
43  void tri_par_selection(MaTable *t)
44  {
45      int crt,//pour le parcour
46      tmp,//pour l'echange
47      ind_min;//pour garder l'indice du min
48      for(crt=0;crt<(t->nbElem-1);crt++)
49      {
50         //recherche du min
51         ind_min=emp_min(*t,crt);
52         if(ind_min=!crt)
53         {
54             tmp=t->tab[crt];
55             t->tab[crt]=t->tab[ind_min];
56             t->tab[ind_min]=tmp;
57         }///fin if(ind_min=!crt)
58      }/// fin for(crt=0;crt<(t->nbElem-1);crt++)
59  }///fin void tri_par_selection(MaTable *t)
```

b.mise en oeuvre à l'aide du pointeur:

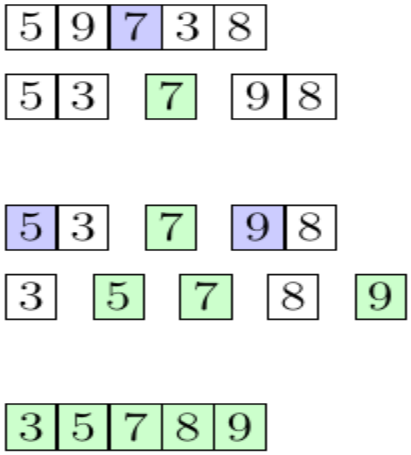
```
39    Cellule *emp_min_point(Cellule *lst)
40    {
41        Cellule *crt,//pour le parcour
42        *min;//pour garder le min
43        min=lst;//initialisation
44        //recherche du min
45        for(crt=lst->suivant;crt;crt=crt->suivant)
46        {
47            if((crt->elem)<(min->elem)) min=crt;
48        }///fin for(crt=lst->suivant;crt;crt=crt->suivant)
49        return (Cellule*)min;
50
51    }
```

```
52
53    Cellule *tri_par_selection_point(Cellule *liste)
54    {
55        Cellule *crt,//pour le parcour
56        *min;//pour garder le min
57        int tmp;//pour l'echange
58        for(crt=liste;crt;crt=crt->suivant)
59        {
60            min=emp_min(crt);
61            if(min!=crt)
62            {
63                tmp=min->elem;
64                min->elem=crt->elem;
65                crt->elem=tmp;
66            }
67        }
68        return (Cellule*)liste;
69    }
```

Trie rapide

Analyse :

La méthode consiste à placer un élément du tableau (appelé pivot) à sa place définitive, en permutant tous les éléments de telle sorte que tous ceux qui sont inférieurs au pivot soient à sa gauche et que tous ceux qui sont supérieurs au pivot soient à sa droite.



Analyse fonctionnelle :

```
triRapide(T:MaTable,i:entier,j:entier)
Variables:
        s:entier.
Début:
        si(i<j) alors:
        Début:
                s=partition(T,i,j);
                triRapide(T,i,s);
```

```
        triRapide(T,s+1,j);
    Fin si.
Fin.
```

```
partition(T:MaTable,gche:entier,dt:entier):entier
Variables:
    pivot:entier.
Début:
    pivot← T.tab[gche];
    tant que (gche<dt) faire:
    Début
        echange(T.tab[gche],T.tab[dt]);
        tant que(T.tab[gche]<=pivot) faire
            gche← gche+1;
        tant que(T.tab[dt]>pivot) faire
            dt← dt-1;
    fin tant que.
    retourner (dt).
Fin
```

Dossier de programmation :

```
1     int partition(MaTable *T,int gche,int dt)
2     {
3         int pivot;
4
5         pivot=T->tab[gche];
6         while(gche<dt)
7         {
```

```
8
9               ///on echange le gauche avec le droit.
10              echange(&T->tab[gche],&T->tab[dt]);
11              ///si le gauche est inferieur au pivot.
12              while(T->tab[gche]<=pivot)    gche++;
13              ///si le droit est supérieur au pivot.
14              while(T->tab[dt]>pivot)    dt--;
15          }///fin while(i<j).
16
17          return (int)dt;
18      }///fin partition(MaTable *T,int i,int j).
19
```

```
20      void triRapide(MaTable *T,int i,int j)
21      {
22          int s;
23
24          if(i<j)
25          {
26              ///la partition.
27              s=partition(T,i,j);
28              ///le tableau gauche.
29              triRapide(T,i,s);
30              ///le tableau droit.
31              triRapide(T,s+1,j);
32          }///fin if(i>j).
1       }///fin triRapide(MaTable *T,int i,int j).
```

Trie par extraction

Analyse :

Après la construction de l'arbre avec les éléments de la liste on cherche chaque fois le plus petit élément at on l'insère à la racine puis on l'échange avec l'élément le plus loin en diminuant le nombre d'éléments et on reprend cette étape jusqu'à ordonner tous les éléments

Analyse fonctionnelle :

```
heapSort(T:MaTable)
Variables:
        i:entier.
Début:
        pour i allant de T.nbElem/2 à 1 par par -1 faire:
                construireHeap(T,T.nbElem,i);
        pour i allant de T.nbElem à 1 par -1 faire:
        Début:
                echange(T.tab[1],T.tab[i]);
                construireHeap(T,i,1);
        Fin pour.
Fin.



construireHeap(T:MaTable,n:entier,i:entier)
Variables:
        gche,dt,indMax:entier.
Début:
        indMax← i;
        gche← 2*i;
        dt← 2*i+1;
```

si(gche<n et T.tab[gche]>T.tab[indMax]) alors:

indMax← gche;

si(dt<n et T.tab[dt]>T.tab[indMax]) alors:

indMax← dt;

si(indMax<>i) alors:

Début:

echange(T.tab[i],T.tab[indMax]);

construireHeap(T,n,indMax);

Fin si.

Fin.

Dossier de programmation :

```
1
2     Cellule*RechercheFils(Cellule*pere, int pospere)
3     {
4          int ind;
5          Cellule*fils = pere;
6          for (ind = pospere; ind < 2 * pospere; ind++)
7          {
8               if (fils == NULL)break;
9               fils = fils->suivant;
10         }
11         return((Cellule*)fils);
12    }
```

```
1
2     void Descent(Cellule*Debut,int posDebut)
3     {
4          Cellule*Racine = Debut,*fils=RechercheFils(Debut,posDebut);
5          int desent = 1;
6          while ((fils) && desent)
```

```
7         {
8             if (fils->suivant)
9                 if (fils->elem < fils->suivant->elem)fils = fils-
>suivant;
10            if (Racine->elem < fils->elem)
11            {
12                echange(&fils->elem, &Racine->elem);
13                Racine = fils;
14                posDebut *= 2;
15                fils = RechercheFils(fils, posDebut);
16            }//fin du if (Racine->elem < fils->elem)
17            else desent = 0;
18        }//fin du while ((fils) && desent)
19    }
```

```
20    void create_Heap(Cellule*fin,int pos)
21    {
22        Cellule*deb=fin;
23        while (deb)
24        {
25            Descent(deb,  pos);
26            pos--;
27            deb = deb->precedent;
28        }//fin du while (deb)
29    }
30
```

```
31
32    void tri_extraction(Cellule*debut, Cellule*fin,int posfin)
33    {
34        Cellule*crt = fin;
35        while (crt)
36        {
37            create_Heap(crt, posfin);
38            echange(&crt->elem, &debut->elem);
39            crt = crt->precedent;
40            posfin--;
41        }//fin du while (crt)
42    }
```