AVL & Splay datastruktur implementering
ADA5 E14

Anders Launer Bæk
anbae12

Oktober 2014

1 Intro

Implementer et AVL og Splay datastruktur. Implementeringen skal indeholde:

- Insert(): Til at indsætte en ny værdi.
- Contains(): Til tjek om værdien allerede findes i træet.

Det er ikke nødvendigt med en remove-funktion, men det er i orden at lave den funktion alligevel. Strukturen skal testes på forskellige størrelser af datasæt, hvorved tiden undersøges i forhold til datastørrelsen. Forsøget skal foretages med tre forskellige typer af datasæt:

Inc: Værdi som monotomt øges.
Dec: Værdi som monotomt sænkes.
Ran: Værdi som vælges vilkårligt.

2 Implementering

Hele implementeringen er bestående af følgende moduler:

- main.cpp. Kode findes i appedix A.
- AvlTree.h og AvlTree.cpp. Kode findes i appedix B.
- SplayTree.h og SplayTree.cpp. Kode findes i appedix C.
- clock_timer.h og clock_timer.cpp. Er ikke vedlagt denne raport.

Til implementering af AVL og Splay datastruktur er henholdvis følgende links brugt til inspiration/implementering: `http://www.sanfoundry.com/cpp-program-implement-avl-trees/` og `http://www.sanfoundry.com/cpp-program-implement-splay-tree/`.
Hovedprogrammet er lavet således at før programmet kommer ind i main(), implementeres en funktion som generer de tre datalister med ønsket antal elementer, der senere skal bruges af de to implementeringer. Det er ligeledes den første function som der køres i main(). Herefter laves et objekt til hver implementeret klasse. Så udskrives information, der beskriver hvilke funktioner programmet indeholder. Hvorefter det er muligt at vælge hvilken funktionalitet der skal udføres. Funktionaliteten udføres og programmet slutter.

3 Data/plots for implementeringerne

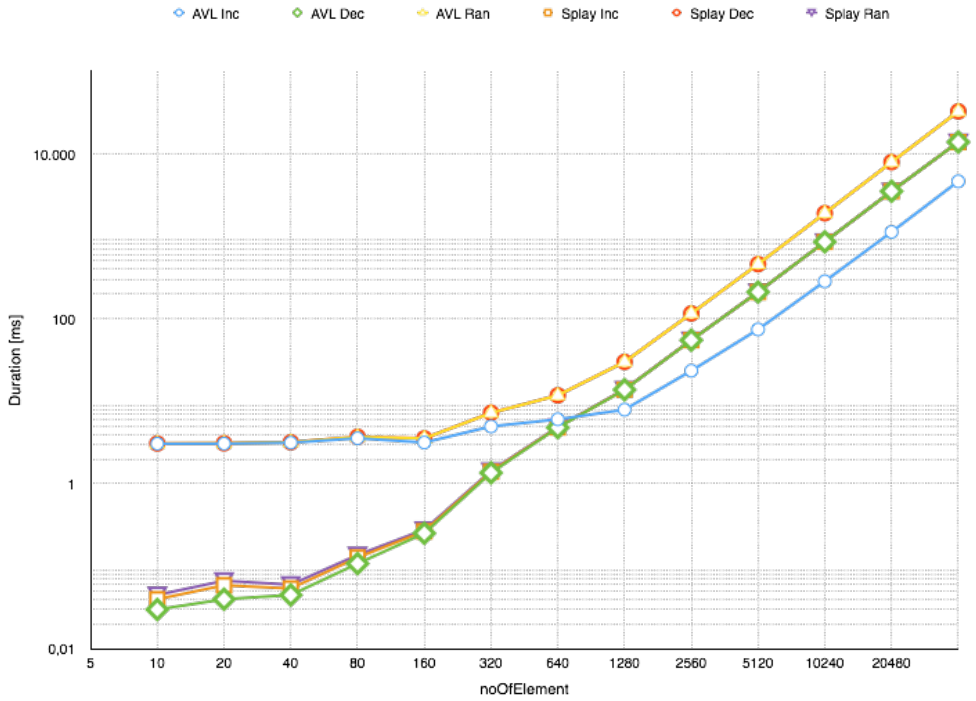
Som nævnt i ovenstående afsnit 2, er de to implementationer kørt for de tre forskellige datasæt Inc, Dec og Ran. Undersøgelserne er kun kørt én gang for hver datasæt for et givet N. Hvis forsøgene fortages flere gange for samme datastørrelse gives en mere korrekt billede af de to implementeringer.
Indsætning af dataværdier og tjek på forskellige dataværdier er beskrevet i de to næste afsnit, hhv. afsnit 3.1 og afsnit 3.2. Implementeringstiderne samt søgetiderne opsummeres i afsnit 3.3.

3.1 Indsætning af de tre datasæts

Tabel 3.1 viser de seks implementeringstider angivet i [ms] og implementeringstiderne er plottet i figur 3.1.

N/datasæt	5	10	20	40	80	160	320	640	1280	2560	5120	10240	20480	
AVL Inc	3,064	3,067	3,147	3,562	3,166	4,96	6,045	7,911	23,441	74,205	284,033	1123,63	4621,95	[ms]
AVL Dec	0,03	0,04	0,045	0,108	0,252	1,359	4,795	13,792	54,942	210,587	853,775	3501,78	13801,2	[ms]
AVL Ran	3,074	3,098	3,186	3,707	3,545	7,233	11,779	29,963	115,763	457,994	1894,21	7886,95	32435	[ms]
Splay Inc	0,04	0,059	0,054	0,128	0,266	1,403	4,835	13,851	55,071	210,811	854,414	3502,61	13802,7	[ms]
Splay Dec	3,08	3,097	3,191	3,714	3,55	7,247	11,789	29,975	115,787	458,051	1894,33	7887,11	32435,3	[ms]
Splay Ran	0,045	0,067	0,06	0,137	0,278	1,431	4,881	13,914	55,181	211,017	855,104	3503,45	13804,4	[ms]

Tabel 3.1: Implementeringstider for de seks datalister som funktion af datasætstørrelsen N.



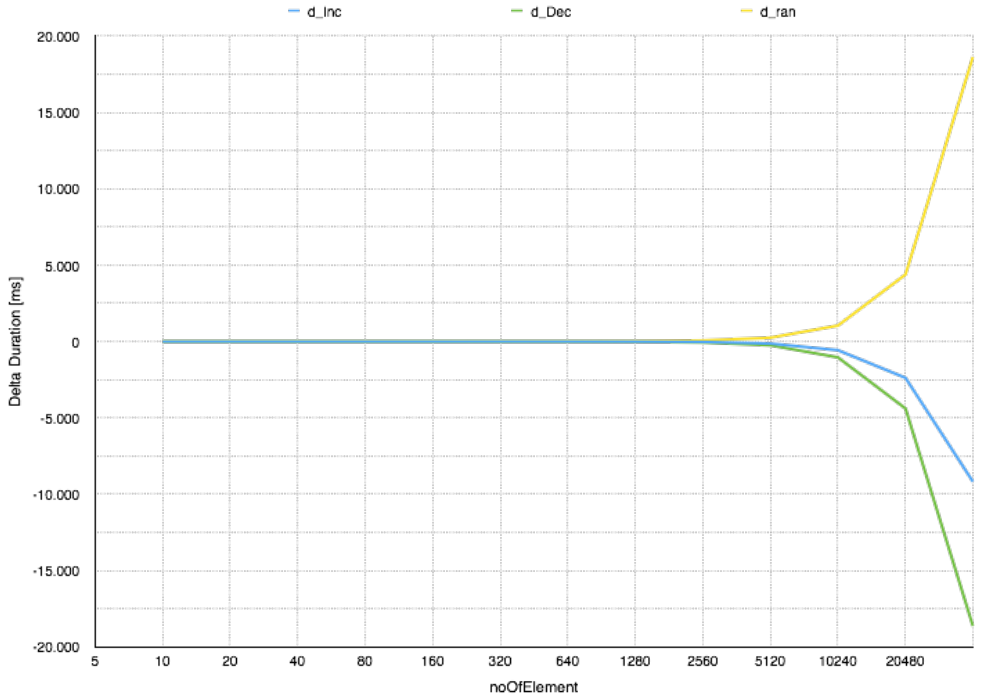
Figur 3.1: Logaritmisk plot af implementeringstider for datasætstørrelserne N.

For at kunne sammenligne de to datastrukturer er Δ-værdien mellem AVL og Splay beregnet. Hvis Δ-værdien er positiv er Splay strukturen hurtigst og omvendt. Δ-værdierne er angivet I tabel 3.2.
For datasættet Inc er Splay strukturen hurtigst indtil en datastørrelse på 640 elementer. Herefter overtager AVL strukturen.
For Dec datasættet ses der at AVL strukturen har den hurtigste implementeringstid for de afprøvede datasæt med N elementer.
For Ran datasættet er implementeringstiderne generelt hurtigere ved Splay strukturen.

N/datasæt	5	10	20	40	80	160	320	640	1280	2560	5120	10240	20480	
ΔInc	3,024	3,008	3,093	3,434	2,9	3,557	1,21	-5,94	-31,63	-136,606	-570,381	-2378,98	-9180,75	[ms]
ΔDec	-3,05	-3,057	-3,146	-3,606	-3,298	-5,888	-6,994	-16,183	-60,845	-247,464	-1040,555	-4385,33	-18634,1	[ms]
ΔRan	3,029	3,031	3,126	3,57	3,267	5,802	6,898	16,049	60,582	246,977	1039,106	4383,5	18630,6	[ms]

Tabel 3.2: Δ-værdien er angivet som differencen mellem implementeringstiden for AVL og Splay i [ms].

Figur 3.2 viser Δ-værdierne mellem AVL og Splay implementeringerne for en given datastørrelse. For en positiv difference er Splay implementeringen hurtigst og ligeledes med en negativ difference er AVL implementeringen hurtigst.



Figur 3.2: Plot af Δ-værdien for de tre datasæt for en given datastørrelse N.

3.2 Find vilkårlig værdi i datasæt

I tabel 3.3 ses søgetiden for en vilkårlig værdi. Værdien er forsøgt fundet i en AVL og Splay struktur med et vilkårlig generet datasæt med N antal elementer. Tabellen angiver også den søgte værdi og ligeledes en Δ-søgetid for de to strukturer. Figur 3.3 indeholder et plot af søgetiderne.

N/datasæt	5	10	20	40	80	160	320	640	1280	2560	5120	10240	20480	
AVL	3,464	3,158	3,014	4,114	3,061	3,467	4,43	7,839	22,422	79,205	339,533	1540,28	6989,17	[ms]
Splay	0,011	0,013	0,014	0,014	0,028	0,025	0,043	0,075	0,131	0,235	0,565	0,91	1,736	[ms]
Δ	3,45	3	3,145 3	4,1	3,033	3,442	4,387	7,764	22,291	78,97	338,968	1539,37	6987,434	[ms]
Ran number	4	2	18	20	62	137	29	364	449	2510	2400	3960	8303	

Tabel 3.3: Søgetider for AVL og Splay samt differencen i mellem angivet som Δ for en given datastørrelse. Og ligeledes vises det vilkårlige nummer for den given datastørrelse.

I tabel 3.3 er det tydeligt at se at Splay implementeringen er langt hurtigere til at finde den vilkårlige værdi i et givet data sæt. Dette underbygges af plottet i figur 3.3.

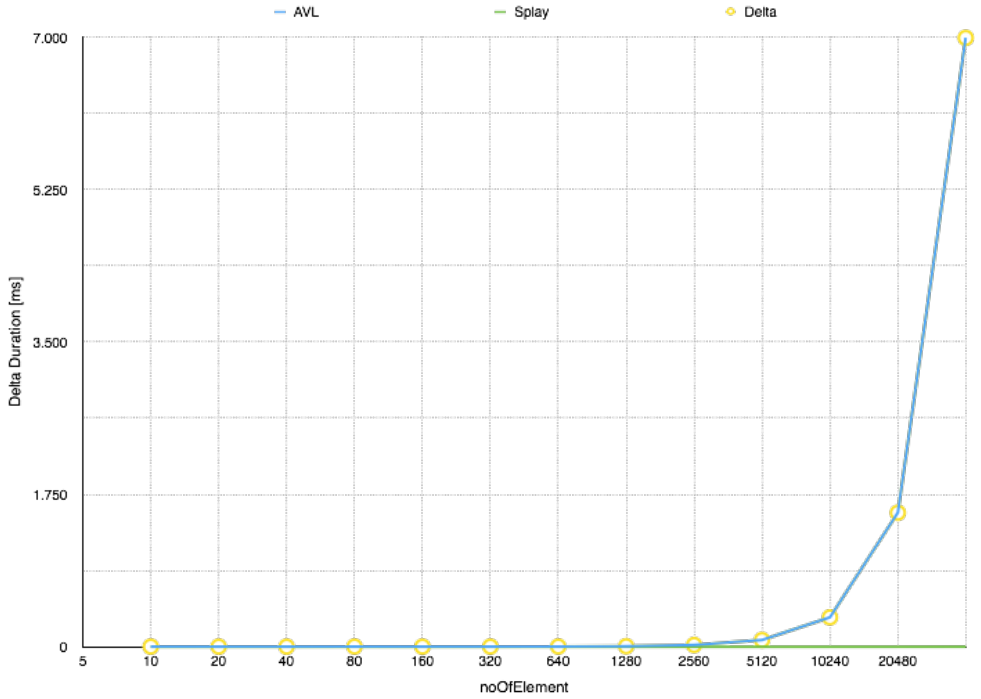
3.3 Opsummering

AVL strukturen:

- Er hurtigst for Dec datasættet.
- Er hurtigst for Inc datasættet efter dets datastørrelse er over 320 elementer.

Splay strukturen:

- Er hurtigst for Inc datasættet for en datastørrelse på 5-320 dataelementer.
- Er hurtigst for Ran datasættet.
- Er hurtigst til at finde en vilkårlig værdi for et Ran datasæt.

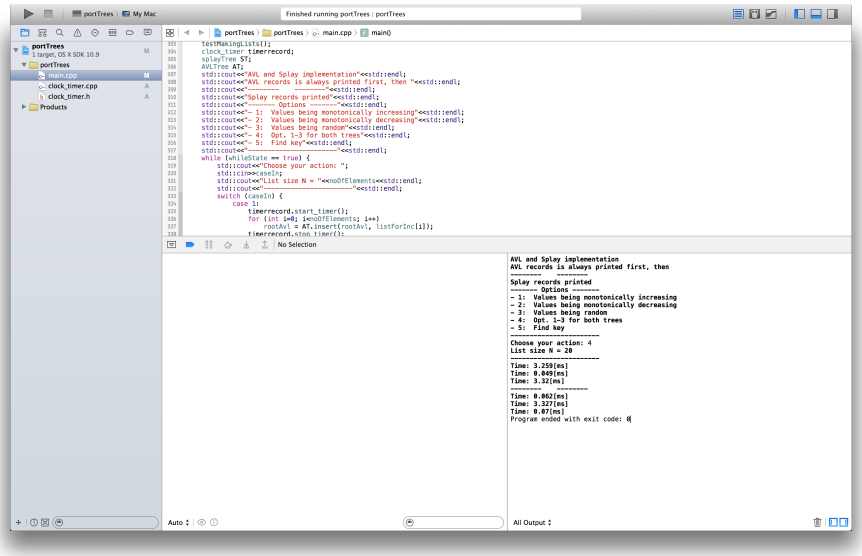


Figur 3.3: Plot af søgetiderne efter en vilkårlig værdi i datasæt for AVL, Splay samt differencen Δ for et givet datasæt med given datastørrelse N.

Idet AVL og Splay strukturerne har forskellige karakteristika er det essentielt at kende applikationen for at kunne benytte den mest optimale datastruktur.

4 Konsol output

Figur 4.1 viser et screenshot fra den bruge editor xCode. I programmet genereres de tre datasæt på 20 elementer hver og bruges i case option 4 til returnering af tiderne det tager for AVL og Splay strukturerne at indsætte de tre datasæt.



Figur 4.1: Konsol output med implementeringstid for de tre datasæt af 20 elementer: case action 4.

Appendices

A Main kildekode

```
1  //  main.cpp
2  //  portTrees
3  //  Created by Anders Launer Baek on 01/10/14.
4  #include <iostream>
5  #include <stdlib.h>
6  #include <time.h>
7  #include "clock_timer.h"
8  #include "AvlTree.h"
9  #include "SplayTree.h"
10 /////////////// Init ///////////////
11 int const noOfElements = 20480;
12 int listForInc[noOfElements];
13 int listForDec[noOfElements];
14 int listForRan[noOfElements];
15 void testMakingLists(){
16     for (int i=0; i<noOfElements; i++)
17         listForInc[i]=i;
18     for (int i=1; i<noOfElements+1; i++)
19         listForInc[i]=noOfElements-i;
20     srand((unsigned int) time(NULL)); // used for setting the rand() function random
21     for (int i=1; i<noOfElements+1; i++)
22         listForRan[i]=rand() % noOfElements;
23 }
24 bool whileState = true;
25 int caseIn;
26 int key;
27
28 /////////////// Main() ///////////////
29 int main(){
30     testMakingLists();
31     clock_timer timerrecord;
32     splayTree ST;
33     AVLTree AT;
34     std::cout<<"AVL and Splay implementation"<<std::endl;
35     std::cout<<"AVL records is always printed first, then "<<std::endl;
36     std::cout<<"--------    --------"<<std::endl;
37     std::cout<<"Splay records printed"<<std::endl;
38     std::cout<<"------- Options -------"<<std::endl;
39     std::cout<<"- 1:  Values being monotonically increasing"<<std::endl;
40     std::cout<<"- 2:  Values being monotonically decreasing"<<std::endl;
41     std::cout<<"- 3:  Values being random"<<std::endl;
42     std::cout<<"- 4:  Opt. 1-3 for both trees"<<std::endl;
43     std::cout<<"- 5:  Find key"<<std::endl;
44     std::cout<<"-----------------------"<<std::endl;
45     while (whileState == true) {
46         std::cout<<"Choose your action: ";
47         std::cin>>caseIn;
48         switch (caseIn) {
49             case 1:
50                 timerrecord.start_timer();
51                 for (int i=0; i<noOfElements; i++)
52                     rootAvl = AT.insert(rootAvl, listForInc[i]);
53                 timerrecord.stop_timer();
54                 std::cout<<"--------    --------"<<std::endl;
55                 timerrecord.start_timer();
56                 for (int i=0; i<noOfElements; i++)
57                     rootSplay = ST.insert(rootSplay, listForInc[i]);
58                 ST.balance(rootSplay);
59                 timerrecord.stop_timer();
60                 whileState = false;
61                 break;
62
63             case 2:
64                 timerrecord.start_timer();
65                 for (int i=0; i<noOfElements; i++)
66                     rootAvl = AT.insert(rootAvl, listForDec[i]);
```

```
67                  timerrecord.stop_timer();
68                  std::cout<<"--------    --------"<<std::endl;
69                  timerrecord.start_timer();
70                  for (int i=0; i<noOfElements; i++)
71                      rootSplay = ST.insert(rootSplay, listForDec[i]);
72                  ST.balance(rootSplay);
73                  timerrecord.stop_timer();
74                  whileState = false;
75                  break;
76
77              case 3:
78                  timerrecord.start_timer();
79                  for (int i=0; i<noOfElements; i++)
80                      rootAvl = AT.insert(rootAvl, listForRan[i]);
81                  timerrecord.stop_timer();
82                  std::cout<<"--------    --------"<<std::endl;
83                  timerrecord.start_timer();
84                  for (int i=0; i<noOfElements; i++)
85                      rootSplay = ST.insert(rootSplay, listForRan[i]);
86                  ST.balance(rootSplay);
87                  timerrecord.stop_timer();
88                  whileState = false;
89                  break;
90
91              case 4:
92                  timerrecord.start_timer();
93                  for (int i=0; i<noOfElements; i++)
94                      rootAvl = AT.insert(rootAvl, listForInc[i]);
95                  timerrecord.stop_timer();
96                  timerrecord.start_timer();
97                  for (int i=0; i<noOfElements; i++)
98                      rootAvl = AT.insert(rootAvl, listForDec[i]);
99                  timerrecord.stop_timer();
100                 timerrecord.start_timer();
101                 for (int i=0; i<noOfElements; i++)
102                     rootAvl = AT.insert(rootAvl, listForRan[i]);
103                 timerrecord.stop_timer();
104                 std::cout<<"--------    --------"<<std::endl;
105                 timerrecord.start_timer();
106                 for (int i=0; i<noOfElements; i++)
107                     rootSplay = ST.insert(rootSplay, listForInc[i]);
108                 ST.balance(rootSplay);
109                 timerrecord.stop_timer();
110                 timerrecord.start_timer();
111                 for (int i=0; i<noOfElements; i++)
112                     rootSplay = ST.insert(rootSplay, listForDec[i]);
113                 ST.balance(rootSplay);
114                 timerrecord.stop_timer();
115                 timerrecord.start_timer();
116                 for (int i=0; i<noOfElements; i++)
117                     rootSplay = ST.insert(rootSplay, listForRan[i]);
118                 ST.balance(rootSplay);
119                 timerrecord.stop_timer();
120                 whileState = false;
121                 break;
122
123             case 5:
124                 srand((unsigned int) time(NULL)); // used for setting the
125                 key=rand() % noOfElements;
126                 std::cout<<"Rand Number: "<<key<<std::endl;
127                 for (int i=0; i<noOfElements; i++)
128                     rootAvl = AT.insert(rootAvl, listForRan[i]);
129                 timerrecord.start_timer();
130                 AT.findKey(rootAvl, key);
131                 timerrecord.stop_timer();
132                 std::cout<<"--------    --------"<<std::endl;
133                 for (int i=0; i<noOfElements; i++)
134                     rootSplay = ST.insert(rootSplay, listForRan[i]);
135                 ST.balance(rootSplay);
136                 timerrecord.start_timer();
137                 ST.findKey(rootSplay, key);
138                 timerrecord.stop_timer();
139                 whileState = false;
```

```
                break;

            default:
                std::cout<<"Wrong choice... please type action\n"<<std::endl;
                break;
        }
    }
    return 0;
}
```

B AVL tree kildekode

```
//  AvlTree.h
//  portTrees
//  Created by Anders Launer Baek on 02/10/14.
#ifndef __portTrees__AvlTree__
#define __portTrees__AvlTree__
struct avlNode{
    int data;
    struct avlNode *left;
    struct avlNode *right;
}*rootAvl;
class AVLTree{
public:
    int height( avlNode *);
    int diff( avlNode *);

    avlNode* balance(avlNode*);
    avlNode* insert(avlNode*,int);
    avlNode *rightRightRot(avlNode *);
    avlNode *rightLeftRot(avlNode *);
    avlNode *leftLeftRot(avlNode *);
    avlNode *leftRightRot(avlNode *);
    avlNode *findKey(avlNode *, int);
    AVLTree(){rootAvl = NULL;}
    void printTree(avlNode *, int, bool);
};
#endif /* defined(__portTrees__AvlTree__) */
```

```
//  AvlTree.cpp
//  portTrees
//  Created by Anders Launer Baek on 02/10/14.
#include "AvlTree.h"
#include <iostream>
int AVLTree::height(avlNode *temp){
    int h = 0;
    if (temp != NULL)
    {
        int leftHeight = height(temp -> left);
        int rightHeight = height(temp -> right);
        int maxHeight = std::max (leftHeight, rightHeight);
        h = maxHeight + 1;
    }
    return h;
}
int AVLTree::diff(avlNode *temp){
    int leftHeight = height(temp -> left);
    int rightHeight = height(temp -> right);
    int bFact = leftHeight - rightHeight;
    return bFact;

}
avlNode *AVLTree::rightRightRot(avlNode *parent){
    avlNode *temp;
    temp = parent -> right;
    parent -> right =temp -> left;
    temp -> left = parent;
    return temp;
}
avlNode *AVLTree::rightLeftRot(avlNode *parent){
    avlNode *temp;
    temp = parent -> right;
    parent -> right = leftLeftRot(temp);
```

```
35      return rightRightRot(parent);
36  }
37  avlNode *AVLTree::leftLeftRot(avlNode *parent){
38      avlNode *temp;
39      temp = parent -> left;
40      parent -> left = temp -> right;
41      temp -> right = parent;
42      return temp;
43  }
44  avlNode *AVLTree::leftRightRot(avlNode *parent){
45      avlNode *temp;
46      temp = parent -> left;
47      parent -> left = rightRightRot(temp);
48      return leftLeftRot(parent);
49  }
50  avlNode *AVLTree::balance(avlNode *temp){
51      int bFact = diff(temp);
52      if (bFact > 1) {
53          if (diff(temp -> left) > 0)
54              temp = leftLeftRot(temp);
55          else
56              temp = leftRightRot(temp);
57      }
58      else if (bFact < -1){
59          if (diff(temp -> right) > 0)
60              temp = rightLeftRot(temp);
61          else
62              temp = rightRightRot(temp);
63      }
64      return temp;
65  }
66  avlNode *AVLTree::insert(avlNode *rootAvl, int value){
67      if (rootAvl == NULL) {
68          rootAvl = new avlNode;
69          rootAvl -> data = value;
70          rootAvl -> left = NULL;
71          rootAvl -> right = NULL;
72          return rootAvl;
73      }
74      else if (value < rootAvl -> data){
75          rootAvl -> left = insert(rootAvl -> left, value);
76          rootAvl = balance(rootAvl);
77      }
78      else if (value >= rootAvl -> data){
79          rootAvl -> right = insert(rootAvl -> right, value);
80          rootAvl = balance(rootAvl);
81      }
82      return rootAvl;
83  }
84  void AVLTree::printTree(avlNode *ptr, int level, bool print){
85      if (print == true) {
86          int i;
87          if (ptr != NULL) {
88              printTree(ptr -> right, level+1,print);
89              std::cout<<""<<std::endl;
90              if (ptr == rootAvl)
91                  std::cout<<"rootAvl -> ";
92              for (i=0; i < level && ptr != rootAvl; i++)
93                  std::cout<<"       ";
94              std::cout<<ptr -> data;
95              printTree(ptr -> left, level+1,print);
96          }
97      }
98      std::cout<<""<<std::endl;
99  }
100 int compare(int k1, int k2){
101     if (k1 < k2)
102         return -1;
103     else if (k1 == k2)
104         return 0;
105     else if (k1 > k2)
106         return 1;
107     return 0;
```

```
}
avlNode *AVLTree::findKey(avlNode *rootAvl, int key){
    if (rootAvl == NULL)
        return NULL;
    else if (key < rootAvl -> data)
        return findKey(rootAvl -> left, key);
    else if (key > rootAvl -> data)
        return findKey(rootAvl -> right, key);
    return 0;
}
```

C Splay tree kildekode

```
//  SplayTree.h
//  portTrees
//  Created by Anders Launer Baek on 02/10/14.
#ifndef __portTrees__SplayTree__
#define __portTrees__SplayTree__
struct splayNode{
    int data;
    struct splayNode *left;
    struct splayNode *right;
}*rootSplay;
class splayTree{
public:
    splayNode* rightRot(splayNode *);
    splayNode* leftRot(splayNode *);
    splayNode* splay(int, splayNode*);
    splayNode* newNode(int);
    splayNode* findKey(splayNode*, int);
    splayNode* insert(splayNode* rootSplay, int);
    void balance(splayNode*);
    splayTree(){rootSplay = NULL;}
};
#endif /* defined(__portTrees__SplayTree__) */
```

```
//  SplayTree.cpp
//  portTrees
//  Created by Anders Launer Baek on 02/10/14.
#include "SplayTree.h"
#include <iostream>

splayNode* splayTree::rightRot(splayNode* temp2){
    splayNode* temp1 = temp2 -> left;
    temp2 -> left = temp1 -> right;
    temp1 -> right = temp2;
    return temp1;
}
splayNode* splayTree::leftRot(splayNode* temp2){
    splayNode* temp1 = temp2 -> right;
    temp2 -> right = temp1 -> left;
    temp1 -> left = temp2;
    return temp1;
}
splayNode* splayTree::splay(int data, splayNode* rootSplay){
    if (!rootSplay)
        return NULL;
    splayNode top;
    top.left = top.right = NULL;
    splayNode* leftTreeMax = &top;
    splayNode* rightTreeMin = &top;
    while (1) {
        if (data < rootSplay -> data) {
            if (!rootSplay -> left)
                break;
            if (data < rootSplay ->left->data) {
                rootSplay = rightRot(rootSplay);
                if (!rootSplay -> left)
                    break;
            }
            rightTreeMin -> left = rootSplay;
            rightTreeMin = rightTreeMin -> left;
            rootSplay = rootSplay -> left;
```

```
38              rightTreeMin -> left = NULL;
39          }
40          else if (data > rootSplay -> data){
41              if (!rootSplay -> right)
42                  break;
43              if (data > rootSplay -> right -> data) {
44                  rootSplay = leftRot(rootSplay);
45                  if (!rootSplay -> right)
46                      break;
47              }
48              leftTreeMax -> right = rootSplay;
49              leftTreeMax = leftTreeMax -> right;
50              rootSplay = rootSplay -> right;
51              leftTreeMax -> right = NULL;
52          }
53          else
54              break;
55      }
56      leftTreeMax -> right = rootSplay -> left;
57      leftTreeMax -> left = rootSplay -> right;
58      rootSplay -> left = top.right;
59      rootSplay -> right = top.left;
60      return rootSplay;
61  }
62  splayNode* splayTree::newNode(int data){
63      splayNode* tempNode = new splayNode;
64      if (!tempNode) {
65          std::cout<<"Out of memory..."<<std::endl;
66          exit(1);
67      }
68      tempNode -> data = data;
69      tempNode -> left = tempNode -> right = NULL;
70      return tempNode;
71  }
72  splayNode* splayTree::findKey( splayNode* rootSplay, int data){
73      return splay(data,rootSplay);
74  }
75  splayNode* splayTree::insert(splayNode* rootSplay, int data){
76      static splayNode* tempNode = NULL;
77      if (!tempNode)
78          tempNode = newNode(data);
79      else
80          tempNode -> data = data;
81      if (!rootSplay) {
82          rootSplay = tempNode;
83          tempNode = NULL;
84          return rootSplay;
85      }
86      rootSplay = splay(data, rootSplay);
87      if (data < rootSplay -> data) {
88          tempNode -> left = rootSplay -> left;
89          tempNode -> right = rootSplay;
90          rootSplay -> right = NULL;
91          rootSplay = tempNode;
92      } else if (data > rootSplay -> data){
93          tempNode -> right = rootSplay -> right;
94          tempNode -> left = rootSplay;
95          rootSplay -> right = NULL;
96          rootSplay = tempNode;
97      } else
98          return rootSplay;
99      tempNode = NULL;
100     return rootSplay;
101 }
102 void splayTree::balance(splayNode* rootSplay){
103     if (rootSplay) {
104         balance(rootSplay -> left);
105         balance(rootSplay -> right);
106     }
107 }
```