# Московский авиационный институт (национальный исследовательский университет)

## Факультет информационных технологий и прикладной математики

## Кафедра вычислительной математики и программирования


**Лабораторная работа №2 по курсу «Дискретный анализ»**


Студент: П. А. Мохляков
Преподаватель: Н. С. Капралов
Группа: М8О-208Б

Дата:
Оценка:
Подпись:

**Москва, 2020**

# Лабораторная работа №2

**Задача:** Требуется разработать программу словарь, построенною на основе дерева, осуществляющую добавление и удаление пар «ключ-значение», поиск по словарю и сохранение с загрузкой словая из бинарного компактного файла.

**Вариант дерева:** Крсаночерное дерево.

**Вариант ключа:** `Последовательность букв английского алфавита длиной не более 256 символов`

**Вариант значения:** `от 0 до` $2^{64} - 1$

# 1 Описание

Как сказано в [1]: «Красно-черное дерево представляет собой бинарное дерево поиска с одним дополнительным битом цвета в каждом узле». Цвет узла либо черный, либо красный. Такое дерево должно удовлетворять следующим свойствам:

- Каждый узел является либо красным, либо черным.
- Корень дерева является черным узлом.
- Каждый лист дерева (Nil) является черным узлом.
- Если узел красный, то оба его дочерних узла черные.

Вставка и удаление в дереве будет описана далее на примере исходного кода.

# 2 Исходный код

Данную программу можно разбить на несколько основных частей:

- Парсинг введенной команды.
- Вставка, удаление и поиск в дереве.
- Сохранение в файл и загрузка дерева из файла.

Парсинг происходит в два этапа. Сначала мы считываем первое слово, из которого можно определить тип команды. Далее мы смотрим на первый символ этой строки и при помощи switch мы выбираем нужный вариан. Далее, по необходимости, мы считываем дополнительное слово или число. В случае вызова switch default, мы принимаем первое слово с командой как ключ для поиска.

```
class TDict
{
  rb::rb_tree<NPair::TPair> Tree;
  public:
  void parse_comand(NString::TString &cmd)
  {
    NString::TString word;
    NString::TString path;
    NPair::TPair Data;
    bool res;
    switch(cmd[0])
    {
      case '+':
      std::cin >> Data.Str >> Data.Num;
      Data.Str.lower();

      Tree.Search(Data,res);
      if(res == false)
      {
        Tree.insert_data(Data);
        std::cout << "OK" << std::endl;
      }
      else
        std::cout << "Exist" << std::endl;
      break;
      case '-':
      std::cin >> Data.Str;
      Data.Str.lower();
      Tree.Search(Data,res);
      if(res == true)
      {
        Tree.Delete(Data);
```

```
33        std::cout << "OK" << std::endl;
34      }
35      else
36        std::cout << "NoSuchWord" << std::endl;
37
38      break;
39      case '!':
40      std::cin >> word >> path;
41      word.lower();
42      if(word == "load")
43      {
44        if(Tree.load(path.str))
45          std::cout << "OK" <<std::endl;
46        else
47          std::cout << "ERROR: can't open file" <<std::endl;
48      }
49      else
50      {
51        if(Tree.save(path.str))
52          std::cout << "OK" <<std::endl;
53        else
54          std::cout << "ERROR: can't open file" <<std::endl;
55      }
56      break;
57      default:
58      Data.Str = cmd;
59      Data.Str.lower();
60      rb::rb_tree_elem<NPair::TPair> *elem = Tree.Search(Data,res);
61
62      if(res)
63      {
64        std::cout<< "OK: " << elem->Key.Num << std::endl;
65      }
66      else
67        std::cout << "NoSuchWord" << std::endl;
68      break;
69    }
70  }
71
72 };
```

Вставка в дерево это немного модифицированная вставка в обычное бинарное дерево поиска. Для того чтобы вставка сохраняла красно-черные свойства дерево используется фенкция ins_fix(), которая перекрашивает узлы и выполняет повороты.

```
1 void insert(rb_tree_elem<T> *z)
2 {
3   rb_tree_elem<T> *y = Nil;
4   rb_tree_elem<T> *x = Root;
5   while(x != Nil)
```

```
6     {
7       y = x;
8       if(z->Key < x->Key)
9       {
10        x = x->Left;
11      }
12      else
13      {
14        x = x->Right;
15      }
16    }
17    z->Par = y;
18    if(y == Nil)
19    {
20      Root = z;
21    }
22    else if(z->Key < y->Key)
23    {
24      y->Left = z;
25    }
26    else
27    {
28      y->Right = z;
29    }
30    z->Left = Nil;
31    z->Right = Nil;
32    z->Color = 1;
33
34    ins_fix(z);
35  }
```

Исправление вставки можно разделить на 3 случая:

- "Дядя"y узла z - красный. Так как z и z.p красный мы красим родителя z и y в черный, а для сохраннения колличества черных узлов мы красим z.p.p в красный, z.p.p становится новым узлом z.
- "Дядя"y узла z черный, и z - правый потомок. В случае если мы правый потомок мы поднимаемся на уровень выше и делаем левый поворот. Далее делаем те же действия, что и в случае 3.
- "Дядя"y узла z черный, и z - левый потомок.

Определение функций:

| dict.hpp | |
|---|---|
| void parse_comand(NString::TString &cmd | Парсинг запросов |
| **rb.hpp** | |
| void left_rotate(rb_tree &Tree,rb_tree_elem<T> *x) | Левый поворот дерева. |
| void right_rotate(rb_tree &Tree,rb_tree_elem<T> *y) | Правый поворот дерева. |
| void ins_fix(rb_tree_elem<T> *z) | Исправление вставки в дерево |
| void insert(rb_tree_elem<T> *z) | Вставка в дерево. |
| void transplant(rb_tree_elem<T> *u,rb_tree_elem<T> *v) | Переносит элимент в дереве. |
| void insert_data(T data) | Создает элимент дерева для вызова вставки. |
| void print(rb_tree_elem<T> *Tree,int lvl) | Вывод дерева. |
| void clear(rb_tree_elem<T> *Tree) | Очистка дерева. |
| void save_tree(rb_tree_elem<T> *Tree,std::ofstream& wf) | Сохраняет дерево в файл. |
| bool save(char *ch) | Проверяет возможность создания и открывает файл на запись. |
| void load_tree(rb_tree_elem<T> *Tree,std::ifstream& rf) | Загрузка дерева из файла. |
| bool load(char *ch) | Проверяет наличие файла и возможность открыть его на чтение. |
| rb_tree_elem<T>* Search(T& sample,bool& success) | Поиск элемента в дереве. |
| rb_tree_elem<T>* Tree_min(rb_tree_elem<T>* elem) | Поиск минимального элемента в дереве |
| void rb_delete(rb_tree_elem<T>* &elem) | Удаление элемента из дерева. |
| void delete_fix(rb_tree_elem<T>* x) | Исправление дерева после удаления элемента. |
| void Delete(T& sample) | Проверяет дерево на возможность удаления и вызывает удаление. |
| bool isFileEmpty(const char* filename) | Проверяет закончился ли файл. |

Листинг:

```
#pragma once
#include <iostream>
#include <stdio.h>
#include <fstream>

namespace rb
{
  template <typename T>
  class rb_tree_elem
  {
    public:
    bool Color;
    T Key;
    rb_tree_elem *Left;
    rb_tree_elem *Right;
    rb_tree_elem *Par;

    rb_tree_elem()
  };

  template <typename T>
  class rb_tree
  {
    public:
    rb_tree_elem<T> *Root;
    rb_tree_elem<T> *Nil;

    rb_tree()

    ~rb_tree()

    void left_rotate(rb_tree &Tree,rb_tree_elem<T> *x)

    void right_rotate(rb_tree &Tree,rb_tree_elem<T> *y)

    void ins_fix(rb_tree_elem<T> *z)

    void insert(rb_tree_elem<T> *z)

    void transplant(rb_tree_elem<T> *u,rb_tree_elem<T> *v)

    void insert_data(T data)

    void print(rb_tree_elem<T> *Tree,int lvl)

    void clear(rb_tree_elem<T> *Tree)

    void save_tree(rb_tree_elem<T> *Tree,std::ofstream& wf)
```

```
49
50      bool save(char *ch)
51
52      void load_tree(rb_tree_elem<T> *Tree,std::ifstream& rf)
53
54      bool load(char *ch)
55
56      rb_tree_elem<T>* Search(T& sample,bool& success)
57
58      rb_tree_elem<T>* Tree_min(rb_tree_elem<T>* elem)
59
60      void rb_delete(rb_tree_elem<T>* &elem)
61
62      void delete_fix(rb_tree_elem<T>* x)
63
64      void Delete(T& sample)
65
66      bool isFileEmpty(const char* filename)
67    };
68
69  }//namespace rb
```

# 3 Консоль

```
pavel@DESKTOP-VBSMFB3:~/Projects/mai/2_course/DA/LB2/tosol/solution$ cat test0
+ a 1
+ A 2
+ aa 18446744073709551615
aa
A
-A
a
pavel@DESKTOP-VBSMFB3:~/Projects/mai/2_course/DA/LB2/tosol/solution$ cat test0
|./solution
OK
Exist
OK
OK: 18446744073709551615
OK: 1
OK
NoSuchWord
```

# 4 Тест производительности

Сравнение производительности будет производится со стандартным красно-черным деревом, представленным в стандартной библиотеке шаблонов C++ контейнером map. Сравниватьбудем скорость добавления, удаления и нахождения ключа в дерево.

| | Количество вставок | Количество удалений | Количество поисков |
|---|---|---|---|
| test1.txt | 3219 | 3195 | 3238 |
| test2.txt | 32170 | 32001 | 32363 |
| test3.txt | 321667 | 320960 | 322447 |

```
pavel@DESKTOP-VBSMFB3:~/solution$ make
pavel@DESKTOP-VBSMFB3:~/solution$ ./solution <test1.txt
Insert:
Map: 8 ms
My_RB: 9 ms
Delete:
Map: 8 ms
My_RB: 9 ms
Find:
Map: 8 ms
My_RB: 8 ms
pavel@DESKTOP-VBSMFB3:~/solution$ ./solution <test2.txt
Insert:
Map: 73 ms
My_RB: 75 ms
Delete:
Map: 70 ms
My_RB: 77 ms
Find:
Map: 72 ms
My_RB: 73 ms
pavel@DESKTOP-VBSMFB3:~/solution$ ./solution <test3.txt
Insert:
Map: 490 ms
My_RB: 506 ms
Delete:
Map: 485 ms
My_RB: 493 ms
Find:
```

```
Map: 480 ms
My_RB: 483 ms
```

Так как и мое дерево, и дерево из стандартной библиотеки шаблонов красночерное, то время получилось примерно одинаковым. Сложноть операций добавления и удаления - $O(log(n))$

# 5 Выводы

Выполнив вторую лабораторную работу по курсу «Дискретный анализ», я реализовал Красно-черное дерево.

Сложность вставки и удаления $O(log(n))$, а максимальная высота корня также $O(log(n))$,что намного лучше, чем у обычного двоичного дерева.

Главный недостаток в этой структуре это перебалансировка, что приводит к увеличению сложности алгоритма и времени затраченного на вставку. Тем не менее как раз за счет балансировки мы и получаем бысрое удаление, вставку и поиск.

# Список литературы

[1] Томас Х. Кормен, Чарльз И. Лейзерсон, Рональд Л. Ривест, Клиффорд Штайн. *Алгоритмы: построение и анализ, 2-е издание.* — Издательский дом «Вильямс», 2007. Перевод с английского: И. В. Красиков, Н. А. Орехова, В. Н. Романов. — 1296 с. (ISBN 5-8459-0857-4 (рус.))