

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, 2021

Лабораторная работа №2

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

Вариант дерева: Красночерное 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 вариант. Далее, по необходимости, мы считываем дополнительное слово или число. В случае вызова switch default, мы принимаем первое слово с командой как ключ для поиска.

```
1 class TDict
2 {
3     rb::rb_tree<NPair::TPair> Tree;
4     public:
5     void parse_comand(NString::TString &cmd)
6     {
7         NString::TString word;
8         NString::TString path;
9         NPair::TPair Data;
10        bool res;
11        switch(cmd[0])
12        {
13            case '+':
14                std::cin >> Data.Str >> Data.Num;
15                Data.Str.lower();
16
17                Tree.Search(Data,res);
18                if(res == false)
19                {
20                    Tree.insert_data(Data);
21                    std::cout << "OK" << std::endl;
22                }
23            else
24                std::cout << "Exist" << std::endl;
25            break;
26            case '-':
27                std::cin >> Data.Str;
28                Data.Str.lower();
29                Tree.Search(Data,res);
30                if(res == true)
31                {
32                    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 функция `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"у узла z - красный. Так как z и z.p красный мы красим родителя z и y в черный, а для сохранения количества черных узлов мы красим z.p.p в красный, z.p.p становится новым узлом z.
- "Дядя"у 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"у 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 элемент в дерево.
void insert_data(T data)	Создает элемент 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:

```
1  #pragma once
2  #include <iostream>
3  #include <stdio.h>
4  #include <fstream>
5
6  namespace rb
7  {
8      template <typename T>
9      class rb_tree_elem
10     {
11     public:
12         bool Color;
13         T Key;
14         rb_tree_elem *Left;
15         rb_tree_elem *Right;
16         rb_tree_elem *Par;
17
18         rb_tree_elem()
19     };
20
21     template <typename T>
22     class rb_tree
23     {
24     public:
25         rb_tree_elem<T> *Root;
26         rb_tree_elem<T> *Nil;
27
28         rb_tree()
29
30         ~rb_tree()
31
32         void left_rotate(rb_tree &Tree, rb_tree_elem<T> *x)
33
34         void right_rotate(rb_tree &Tree, rb_tree_elem<T> *y)
35
36         void ins_fix(rb_tree_elem<T> *z)
37
38         void insert(rb_tree_elem<T> *z)
39
40         void transplant(rb_tree_elem<T> *u, rb_tree_elem<T> *v)
41
42         void insert_data(T data)
43
44         void print(rb_tree_elem<T> *Tree, int lvl)
45
46         void clear(rb_tree_elem<T> *Tree)
47
48         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 isEmpty(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 производиться со стандартным красно-черным деревом, представленным в стандартной библиотеке шаблонов C++ контейнером `map`. Сравним скорость добавления, удаления и нахождения ключа в дерево.

	Количество вставок	Количество удалений	Количество по- исков
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 красное-черное, то время получилось примерно одинаковым. Сложность операций добавления и удаления - $O(\lg(n))$

5 Выводы

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

Сложность вставки и удаления $O(\lg(n))$, а максимальная высота корня также $O(\lg(n))$, 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 быстрое 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.))