Московский авиационный институт (национальный исследовательский университет)

Институт информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №2 по курсу «Дискретный анализ»

Студент: Л. В. Ивенкова
Преподаватель: Н. С. Капралов
Группа: М8О-208Б-19

Дата:
Оценка:
Подпись:

Москва, 2021

Лабораторная работа №2

Задача: Необходимо создать программную библиотеку, реализующую указанную структуру данных, на основе которой разработать программу-словарь. В словаре каждому ключу, представляющему из себя регистронезависимую последовательность букв английского алфавита длиной не более 256 символов, поставлен в соответствие некоторый номер, от 0 до $2^{64}-1$. Разным словам может быть поставлен в соответствие один и тот же номер.

Программа должна обрабатывать строки входного файла до его окончания. Каждая строка может иметь следующий формат:

+ word 34 — добавить слово «word» с номером 34 в словарь. Программа должна вывести строку «OK», если операция прошла успешно, «Exist», если слово уже находится в словаре.

- word — удалить слово «word» из словаря. Программа должна вывести «OK», если слово существовало и было удалено, «NoSuchWord», если слово в словаре не было найдено.

word — найти в словаре слово «word». Программа должна вывести «OK: 34», если слово было найдено; число, которое следует за «OK:» — номер, присвоенный слову при добавлении. В случае, если слово в словаре не было обнаружено, нужно вывести строку «NoSuchWord».

! Save /path/to/file — сохранить словарь в бинарном компактном представлении на диск в файл, указанный парамером команды. В случае успеха, программа должна вывести «OK», в случае неудачи выполнения операции, программа должна вывести описание ошибки (см. ниже).

! Load /path/to/file — загрузить словарь из файла. Предполагается, что файл был ранее подготовлен при помощи команды Save. В случае успеха, программа должна вывести строку «OK», а загруженный словарь должен заменить текущий (с которым происходит работа); в случае неуспеха, должна быть выведена диагностика, а рабочий словарь должен остаться без изменений. Кроме системных ошибок, программа должна корректно обрабатывать случаи несовпадения формата указанного файла и представления данных словаря во внешнем файле.

Для всех операций, в случае возникновения системной ошибки (нехватка памяти, отсутсвие прав записи и т.п.), программа должна вывести строку, начинающуюся с «ERROR:» и описывающую на английском языке возникшую ошибку.

Вариант структуры данных: AVL-дерево.

1 Описание

Требуется написать реализацию АВЛ-дерева.

Справка вики ([3]): **АВЛ-дерево** — сбалансированное по высоте двоичное дерево поиска: для каждой его вершины высота её двух поддеревьев различается не более чем на 1.

Со сбалансированными деревьями можно выполнять следующие операции за $O(log(n))$ единицу времени даже в худшем случае[2]:

1. Вставить узел с данным ключом.
2. Удалить узел с данным ключом.
3. Найти узел с данным ключом.

Баланс поддерживается с помощью вращений; для его восстановления после добавления или удаления вершины может потребоваться $O(log(n))$ вращений (для дерева с n вершинами) [1].

2 Исходный код

В начале создаём структуру одного узла дерева **TNode**, в которой будут лежать пара «ключ - значение», указатели на правое и левое поддеревья и высота поддерева с корнем в данном узле.

Далее реализуем основные операции - вставки, поиска, удаления, балансировки, а также операции сохрания в файл и загрузки из файла.

В общих чертах процесс **включения** узла состоит из последовательности таких трёх этапов[2]:

1. Следовать по пути поиска, пока не окажется, что ключа нет в дереве.
2. Включить новый узел и определить новый показатель сбалансированности.
3. Пройти обратно по пути поиска и проверить показатель сбалансированности у каждого узла.

Теперь рассмотрим алгоритм **удаления** узла[4]:

1. Находим узел p с заданным ключом k (если не находим, то делать ничего не надо).
2. Если у найденный узел p не имеет правого поддерева, то по свойству АВЛ-дерева слева у этого узла может быть либо только один единственный дочерний узел (дерево высоты 1), либо узел p вообще чист. В обоих этих случаях надо просто удалить узел p и вернуть в качестве результата указатель на левый дочерний узел узла p.
3. Если же правое поддерево у p есть, то находим в нём узел min с наименьшим ключом и заменяем удаляемый узел p на найденный узел min.

Алгоритм же **поиска** в АВЛ-деревьях точно такой же, как в обычных двоичных деревьях.

Для операций **сохранения** и **загрузки** будем использовать классы std::ofstream и std::ifstream.

Сам код:

AVL-tree.hpp

```
#pragma once
#include <iostream>
#include <cstring>
#include <fstream>

const int SIZE_KEY = 256;
using TKey = char;
using TValue = unsigned long long;

class TAVL {
public:
    struct TNode {
        TNode() {
            Left = Right = nullptr;
            Height = 1;
        }
        TNode(TKey* key, TValue value) {
            Key = Substitution(key);
            Value = value;
            Left = Right = nullptr;
            Height = 1;
        }
        ~TNode() {
            delete[] Key;
        }
        TKey* Substitution(TKey* key) {
            if (key == nullptr) {
                return nullptr;
            }
            char* sub = new TKey[SIZE_KEY + 1];
            memcpy(sub, key, strlen(key));
            memset(sub + strlen(key), '\0', SIZE_KEY + 1 - strlen(key));
            return sub;
        }
        TKey* Key = nullptr;
        TValue Value = 0;
        unsigned char Height;
        TNode* Left = nullptr;
        TNode* Right = nullptr;
    };

    TNode* Root;

    TAVL() {
        Root = nullptr;
    }

```

```
48      ~TAVL() {
49          DeleteTree(Root);
50      }
51
52      void DeleteTree(TNode* root) {
53          if (root != nullptr) {
54              DeleteTree(root->Left);
55              DeleteTree(root->Right);
56              delete root;
57          }
58      }
59
60      int CompareKeys(TKey* lhs, TKey* rhs) {
61          return strcmp(lhs, rhs);
62      }
63
64      unsigned char GetHeight(TNode* p) {
65          return p ? p->Height : 0;
66      }
67
68      int BFactor(TNode* p) {
69          return GetHeight(p->Right) - GetHeight(p->Left);
70      }
71
72      void FixHeight(TNode* p) {
73          unsigned char hl = GetHeight(p->Left);
74          unsigned char hr = GetHeight(p->Right);
75          p->Height = (hl > hr ? hl : hr) + 1;
76      }
77
78      TNode* RotateRight(TNode* p) {
79          TNode* q = p->Left;
80          p->Left = q->Right;
81          q->Right = p;
82          FixHeight(p);
83          FixHeight(q);
84          return q;
85      }
86
87      TNode* RotateLeft(TNode* q) {
88          TNode* p = q->Right;
89          q->Right = p->Left;
90          p->Left = q;
91          FixHeight(q);
92          FixHeight(p);
93          return p;
94      }
95
96      TNode* Balance(TNode* p) {
```

```
97          FixHeight(p);
98          if (BFactor(p) == 2)
99          {
100             if (BFactor(p->Right) < 0)
101                 p->Right = RotateRight(p->Right);
102             return RotateLeft(p);
103         }
104         if (BFactor(p) == -2)
105         {
106             if (BFactor(p->Left) > 0)
107                 p->Left = RotateLeft(p->Left);
108             return RotateRight(p);
109         }
110         return p;
111     }
112
113     TNode* Insert(TNode* p, TKey* key, TValue value) {
114         if (p == nullptr) {
115             p = new TNode(key, value);
116             std::cout << "OK\n";
117             return p;
118         }
119         if (CompareKeys(key, p->Key) < 0) {
120             p->Left = Insert(p->Left, key, value);
121         }
122         else if (CompareKeys(key, p->Key) > 0) {
123             p->Right = Insert(p->Right, key, value);
124         }
125         else {
126             std::cout << "Exist\n";
127         }
128         return Balance(p);
129     }
130
131     TNode* FindMin(TNode* p) {
132         return p->Left ? FindMin(p->Left) : p;
133     }
134
135     TNode* RemoveMin(TNode* p) {
136         if (p->Left == 0)
137             return p->Right;
138         p->Left = RemoveMin(p->Left);
139         return Balance(p);
140     }
141
142     TNode* Remove(TNode* p, TKey* key) {
143         if (p == nullptr) {
144             std::cout << "NoSuchWord\n";
145             return nullptr;
```

```
146         }
147         if (CompareKeys(key, p->Key) < 0)
148             p->Left = Remove(p->Left, key);
149         else if (CompareKeys(key, p->Key) > 0)
150             p->Right = Remove(p->Right, key);
151         else // k == p->Key
152         {
153             TNode* q = p->Left;
154             TNode* r = p->Right;
155             delete p;
156             std::cout << "OK\n";
157             if (r == nullptr) return q;
158             TNode* min = FindMin(r);
159             min->Right = RemoveMin(r);
160             min->Left = q;
161             return Balance(min);
162         }
163         return Balance(p);
164     }
165 
166     TNode* Find(TNode* node, TKey* key) {
167         if (node == nullptr) {
168             return nullptr;
169         }
170         if (CompareKeys(node->Key, key) > 0) {
171             return Find(node->Left,key);
172         }
173         else if (CompareKeys(node->Key, key) < 0) {
174             return Find(node->Right,key);
175         }
176         else {
177             return node;
178         }
179         return nullptr;
180     }
181 
182     void PrintFind(TKey* key) {
183         TNode* res = Find(Root, key);
184         if (res != nullptr) {
185             std::cout << "OK: " << res->Value << std::endl;
186         }
187         else {
188             std::cout << "NoSuchWord\n";
189         }
190     }
191 
192     void Save(std::ostream& file, TNode* node) {
193         if (node == nullptr) {
194             return;
```

```
195         }
196         int keySize = strlen(node->Key);
197         file.write((char*)&keySize, sizeof(int));
198         file.write(node->Key, keySize);
199         file.write((char*)&(node->Value), sizeof(TValue));
200 
201         bool hasLeft = (node->Left != nullptr);
202         bool hasRight = (node->Right != nullptr);
203 
204         file.write((char*)&hasLeft, sizeof(bool));
205         file.write((char*)&hasRight, sizeof(bool));
206 
207         if (hasLeft) {
208             Save(file, node->Left);
209         }
210         if (hasRight) {
211             Save(file, node->Right);
212         }
213     }
214 
215     TNode* Load(std::istream& file, TNode* node) {
216         TNode* root = nullptr;
217 
218         int keysize;
219         file.read((char*)&keysize, sizeof(int));
220 
221         if (file.gcount() == 0) {
222             return root;
223         }
224 
225         TKey* key = new char[keysize + 1];
226         key[keysize] = '\0';
227         file.read(key, keysize);
228 
229         TValue value;
230         file.read((char*)&value, sizeof(TValue));
231 
232         bool hasLeft = false;
233         bool hasRight = false;
234         file.read((char*)&hasLeft, sizeof(bool));
235         file.read((char*)&hasRight, sizeof(bool));
236 
237         root = new TNode(key,value);
238         if (hasLeft) {
239             root->Left = Load(file, root);
240         }
241         else {
242             root->Left = nullptr;
243         }
```

```
244
245         if (hasRight) {
246             root->Right = Load(file, root);
247         }
248         else {
249             root->Right = nullptr;
250         }
251         delete[] key;
252         return root;
253     }
254 };
```

main.cpp

```
1   #include <iostream>
2   #include <cstdio>
3   #include <cstring>
4   #include <fstream>
5
6   #include "AVL-tree.hpp"
7
8   void DoLower(TKey* key) {
9       int lengh = strlen(key);
10      for (int i = 0; i < lengh; i++) {
11          key[i] = tolower(key[i]);
12      }
13  }
14
15  int main() {
16
17      std::ios::sync_with_stdio(false);
18      std::cin.tie(nullptr);
19      std::cout.tie(nullptr);
20
21      TAVL tree;
22      TKey key[SIZE_KEY + 1];
23      TValue value;
24
25      std::ofstream ofstr;
26      std::ifstream ifstr;
27
28      while (std::cin >> key) {
29          if (key[0] == '+') {
30              std::cin >> key >> value;
31              DoLower(key);
32              tree.Root = tree.Insert(tree.Root, key, value);
33          }
34          else if (key[0] == '-') {
```

```
35              std::cin >> key;
36              DoLower(key);
37              tree.Root = tree.Remove(tree.Root, key);
38          }
39          else if (key[0] == '!' && strlen(key) == 1) {
40              std::cin >> key;
41              if (key[0] == 'S') {
42                  std::cin >> key;
43                  ofstr.open(key, std::ios::out | std::ios::binary);
44                  if (ofstr) {
45                      tree.Save(ofstr, tree.Root);
46                      std::cout << "OK\n";
47                  }
48                  else {
49                      std::cout << "ERROR: can't open file\n";
50                  }
51                  ofstr.close();
52              }
53              else if (key[0] == 'L') {
54                  std::cin >> key;
55                  ifstr.open(key, std::ios::in | std::ios::binary);
56                  if (ifstr) {
57                      tree.DeleteTree(tree.Root);
58                      tree.Root = tree.Load(ifstr, nullptr);
59                      std::cout << "OK\n";
60                      ifstr.close();
61                  }
62                  else {
63                      std::cout << "ERROR: can't open file\n";
64                  }
65              }
66          }
67          else {
68              DoLower(key);
69              tree.PrintFind(key);
70          }
71      }
72      return 0;
73  }
```

3 Консоль

```
parsifal@DESKTOP-3G70RV4:~/DA/Lab2$ g++ main.cpp
parsifal@DESKTOP-3G70RV4:~/DA/Lab2$ cat test.t
+ a 1
+ A 2
+ aa 18446744073709551615
aa
A
-A
a
! Save test2.t
+ Yatty 67
! Load test2.t
Yatty
parsifal@DESKTOP-3G70RV4:~/DA/Lab2$ ./a.out <test.t
OK
Exist
OK
OK: 18446744073709551615
OK: 1
OK
NoSuchWord
OK
OK
OK
NoSuchWord
```

4 Тест производительности

Тест производительности представляет из себя следующее: мы обрабатываем начальный набор данных и сохраняем пары «ключ - значение» двумя способами: нашим АВЛ-деревом и с помощью std::map.

Для этого создаём программу для генерации тестов.

```
import sys
import random
import string

def get_random_key():
    return random.choice(string.ascii_letters)

actions = ["+", "-", ""]
keys = dict()
test_file_name = "test{:02d}".format(1)
with open("{0}.t".format(test_file_name), 'w') as output_file, \
     open("{0}.a".format(test_file_name), "w") as answer_file:

    for _ in range(random.randint(1000000, 1000000)):
        action = random.choice(actions)
        if action == "+":
            key = get_random_key()
            value = random.randint(1, 2 ** 64 - 1)
            output_file.write("+ {0} {1}\n".format(key, value))
            key = key.lower()
            answer = "Exists"
            if key not in keys:
                answer = "OK"
                keys[key] = value
            answer_file.write("{0}\n".format(answer))

        if action == "-":
            key = get_random_key()
            value = random.randint(1, 2 ** 64 - 1)
            output_file.write("- {0}\n".format(key))
            key = key.lower()
            answer = "NoSuchWord"
            if key in keys:
                answer = "OK"
                del keys[key]
            answer_file.write("{0}\n".format(answer))

        elif action == "":
            search_exist_element = random.choice([True, False])
```

```
40              key = random.choice([key for key in keys.keys()]) if search_exist_element
                    and len(keys.keys()) > 0 else get_random_key()
41              output_file.write("{0}\n".format(key))
42              key = key.lower()
43              if key in keys:
44                  answer = "OK: {0}".format(keys[key])
45              else:
46                  answer = "NoSuchWord"
47              answer_file.write("{0}\n".format(answer))
```

И проверяем время работы структур данных (с помощью библиотеки chrono - она позволяет замерить время от запуска сортировки до её завершения). Причём проверям на одинаковых данных, но разных рамеров.

```
parsifal@DESKTOP-3G70RV4:~/DA/Lab2$ python3 generator_tests.py
Count of lines is 1000000
parsifal@DESKTOP-3G70RV4:~/DA/Lab2$ g++ benchmark.cpp
parsifal@DESKTOP-3G70RV4:~/DA/Lab2$ ./a.out <test01.t
STL std::map time: 24455
AVL-tree time: 142679
```

Количество строк в тесте	Время работы структур данных (мкс)
100	STL std::map time: 19 AVL-tree time: 32
1000	STL std::map time:: 47 AVL-tree time: 360
5000	STL std::map time: 179 AVL-tree time: 1670
10.000	STL std::map time: 593 AVL-tree time: 4109
100.000	STL std::map time: 2628 AVL-tree time: 21354
1.000.000	STL std::map time: 20607 AVL-tree time: 128155

Сложности вставки, удаления и поиска в std::map и в АВл-деревьях одинаковые - $O(log(n))$ (так как std::map основана на красно-чёрных деревьях). Однако у меня вышло, что время выполнения АВЛ-дерева в несколько раз выше, чем выполнениеstd::map.

Я думаю это может быть по двум причинам.

Первая - это то, что при работе АВЛ-дерева очень много времени занимает балансировка и перебалансировка узлов. Ведь у АВЛ-дерева должна сохраняться разница между высотами поддеревьев <=1, тогда как в красно-черных дереьях (на которых основана std::map) левое и правое поддеревья могут отличаться длиной почти в 2 раза. Соответственно, красно-черные деревья тратят меньше ресурсов на поддежание сбалансированности.

У Н.Вирта[2] говорится: «Из-за сложности операций балансировки считается, что АВЛ-деревья следует использовать лишь в том случае, когда поиск информации происходит значительно чаще, чем включение.»

Действительно, если отрегулировать количество генерируемых тестов поиска и включений соответствующим образом можно сразу же увидеть, как время работы АВл-дерева уменьшается в несколько раз:

```
parsifal@DESKTOP-3G70RV4:~/DA/Lab2$ python3 generator_tests.py
38339 +, 38765 -, 922896 ?
parsifal@DESKTOP-3G70RV4:~/DA/Lab2$ g++ benchmark.cpp
parsifal@DESKTOP-3G70RV4:~/DA/Lab2$ ./a.out <test01.t
STL std::map time: 18801
AVL-tree time: 24902
```

Однако время работы этих структур всё еще отличается в пару раз. Это так же может быть из-за того, что при удалении элемента я дважды прохожусь по правому поддереву.

5 Выводы

Выполнив вторую лабораторную работу по курсу «Дискретный анализ», я изучила сбалансированные деревья поиска и научилась их реализовывать, создавать на их основе словари.

Научилась генерировать тесты для этих деревьев, а также оценивать сложность и эффективность их работы.

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.))

[2] Н. Вирт. *Алгоритмы + структуры данных = программы.*

[3] *АВЛ-дерево — Википедия.*
URL: `https://ru.wikipedia.org/wiki/АВЛ-дерево` (дата обращения: 2.01.2021).

[4] *АВЛ-деревья — habr.*
URL: `https://habr.com/ru/post/150732/` (дата обращения: 2.01.2021).