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: А. К. Носов
Преподаватель: А. А. Кухтичев
Группа: М8О-206Б-22
Дата:
Оценка:
Подпись:

Москва, 2024

# Лабораторная работа №2

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.

**Вариант:** AVL дерево.

**Вариант ключа:** `Числа от 0 до` $2^{64}$ `- 1.`

**Вариант значения:** `Регистронезависимая последовательность букв английского алфавита длиной не более 256 символов.`

# 1 Описание

Требуется создать программную библиотеку, реализующую AVL-дерево.

1. AVL дерево (AVL tree): Свойство AVL-дерева заключается в том, что для любого узла разница высот его левого и правого поддеревьев (называемая баланс-фактором) не превышает 1. Это обеспечивает сбалансированность дерева, что, в свою очередь, гарантирует высокую эффективность основных операций.

2. Основная идея: сначала реализовать бинарное дерево поиска с основными операциями (вставка, удаление, поиск), а затем проапгрейдить его до AVL дерева путём модификации операций и структуры узла.

# 2 Исходный код

Создадим новую структуру $KV$, в которой хранятся ключ и значение. Далее создадим новую структуру $TNode$, которая содержит указателей на родителя, левого ребёнка и правого ребёнка, высоту узла и переменную типа $KV$. Затем мы объявляем класс $AVLTree$, в котором находятся реализации необходимых функций.

lab2.cpp

```
    #include <iostream>
#include <fstream>
#include <algorithm>


using namespace std;

struct KV {
    string key;
    unsigned long long int val;

};

struct TNode {
    KV data;
    int height;
    TNode* left;
    TNode* right;
    TNode(KV d)
        : data(d), height(1), left(nullptr), right(nullptr) {}
};

class TAVLTree {

public:

    bool IsLoading = false;

    TNode* root;

    TAVLTree() : root(nullptr) {}

    ~TAVLTree() {
        Clear(root);
    }

    void Save(ofstream& file, TNode* node) {
        if (node) {
            file << node->data.key << " " << node->data.val << endl;
            Save(file, node->left);
```

```
41            Save(file, node->right);
42        }
43    }
44
45    TNode* Load(ifstream& file, TNode* node) {
46        Clear(node);
47        node = nullptr;
48        KV kv;
49        IsLoading = true;
50        while (file >> kv.key >> kv.val) {
51            node = Insert(node, kv);
52        }
53        IsLoading = false;
54        return node;
55    }
56
57
58    TNode* Insert(TNode* node, KV d) {
59        if (!node) {
60            if (!IsLoading){
61                cout << "OK" << endl;
62            }
63            return new TNode(d);
64        }
65        if (d.key == node->data.key) {
66            cout << "Exist" << endl;
67            return node;
68        }
69        if (d.key < node->data.key){
70            node->left = Insert(node->left, d);
71        } else {
72            node->right = Insert(node->right, d);
73        }
74        return Balance(node);
75    }
76
77    TNode* Remove(TNode* node, const string& key) {
78        if (node == nullptr){
79            cout << "NoSuchWord" << endl;
80            return nullptr;
81        }
82        if (key == node->data.key) {
83            TNode* l = node->left;
84            TNode* r = node->right;
85            delete node;
86            cout << "OK" << endl;
87            if (r == nullptr){
88                return l;
89            }
```

```
90              TNode* min = FindMin(r);
91              min->right = RemoveMin(r);
92              min->left = l;
93              return Balance(min);
94          }
95          if (key < node->data.key){
96              node->left = Remove(node->left, key);
97          } else {
98              node->right = Remove(node->right, key);
99          }
100         return Balance(node);
101     }
102
103     TNode* Find(TNode* node, const string& key) {
104         if (!node) {
105             return nullptr;
106         }
107         if (node->data.key > key) {
108             return Find(node->left, key);
109         }
110         else if (node->data.key < key) {
111             return Find(node->right, key);
112         }
113         else {
114             return node;
115         }
116         return nullptr;
117     }
118
119
120 private:
121
122     void Clear(TNode* node) {
123         if (node) {
124             Clear(node->left);
125             Clear(node->right);
126             delete node;
127         }
128     }
129
130     int Height(TNode* node) {
131         return node ? node->height : 0;
132     }
133
134     int BalanceF(TNode* node) {
135         return Height(node->right) - Height(node->left);
136     }
137
138     void FixHeight(TNode* node) {
```

```
139         int hl = Height(node->left);
140         int hr = Height(node->right);
141         node->height = max(hl, hr) + 1;
142     }
143 
144     TNode* RotateLeft(TNode* x) {
145         TNode* y = x->right;
146         x->right = y->left;
147         y->left = x;
148         FixHeight(x);
149         FixHeight(y);
150         return y;
151     }
152 
153     TNode* RotateRight(TNode* y) {
154         TNode* x = y->left;
155         y->left = x->right;
156         x->right = y;
157         FixHeight(y);
158         FixHeight(x);
159         return x;
160     }
161 
162     TNode* Balance(TNode* node) {
163         FixHeight(node);
164         if (BalanceF(node) == 2) {
165             if (BalanceF(node->right) < 0)
166                 node->right = RotateRight(node->right);
167             return RotateLeft(node);
168         }
169         if (BalanceF(node) == -2) {
170             if (BalanceF(node->left) > 0)
171                 node->left = RotateLeft(node->left);
172             return RotateRight(node);
173         }
174         return node;
175     }
176 
177     TNode* FindMin(TNode* node) {
178         return node->left ? FindMin(node->left) : node;
179     }
180 
181     TNode* RemoveMin(TNode* node) {
182         if (node->left == nullptr)
183             return node->right;
184         node->left = RemoveMin(node->left);
185         return Balance(node);
186     }
187 
```

```
188     string DoLower(string str){
189         for (int i = 0; i < str.size(); ++i){
190             str[i] = tolower(str[i]);
191         }
192         return str;
193     }
194 };
195 
196 
197 int main()
198 {
199     TAVLTree tree;
200     string req;
201     while (cin >> req) {
202         if (req == "+") {
203             KV el;
204             cin >> el.key >> el.val;
205             transform(el.key.begin(), el.key.end(), el.key.begin(), ::tolower);
206             tree.root = tree.Insert(tree.root, el);
207         } else if (req[0] == '-') {
208             string key;
209             cin >> key;
210             transform(key.begin(), key.end(), key.begin(), ::tolower);
211             tree.root = tree.Remove(tree.root, key);
212 
213         } else if (req == "!") {
214             string comand;
215             string path;
216             cin >> comand >> path;
217             if (comand == "Save"){
218                 ofstream file;
219                 file.open(path);
220                 tree.Save(file, tree.root);
221                 cout << "OK" << endl;
222                 file.close();
223             } else if (comand == "Load"){
224                 ifstream file;
225                 file.open(path);
226                 tree.root = tree.Load(file, tree.root);
227                 cout << "OK" << endl;
228                 file.close();
229             }
230         } else {
231             transform(req.begin(), req.end(), req.begin(), ::tolower);
232             TNode* fn = tree.Find(tree.root, req);
233             if (fn == nullptr){
234                 std::cout << "NoSuchWord\n";
235             } else {
236                 std::cout << "OK: " << fn->data.val << endl;
```

```
237             }
238         }
239     }
240     return 0;
241 }
```

# 3 Тест: консоль

```
+ Semyon 19
OK
+ Anna 18
OK
+ Pencil 1000
OK
+ Book 25
OK
+ Behemoth 1000000
OK
+ Carrot 504
OK
+ Umbrella 1
OK
-anna
OK
anna
NoSuchWord
+ anna 19
OK
! Save asd
OK
-anna
OK
-book
OK
anna
NoSuchWord
book
NoSuchWord
! Load asd
OK
anna
OK: 19
book
OK: 25
```

# 4 Выводы

Выполняя 2 лабораторную работу по курсу «Дискретный анализ», я узнал об устройстве AVL-деревьев и основных операциях с ними.

# 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] Список использованных источников оформлять нужно по ГОСТ Р 7.05-2008