ГУАП

КАФЕДРА № 43

ОТЧЕТ ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

ассистент		К.А. Кочин
должность, уч. степень, звание	подпись, дата	инициалы, фамилия

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ

АВЛ - ДЕРЕВЬЯ ПОИСКА

по курсу: Структуры и алгоритмы обработки данных

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР. № 4136		Бобрович Н. С.
	подпись, дата	инициалы, фамилия

Санкт-Петербург 2022

Цель работы

Целью работы является изучение деревьев поиска и получение практических навыков их использования..

Задание

Вариант №1.

Вывести глубину самого верхнего листа дерева (maxh) и самого нижнего листа (ов) дерева (minh), а так же их значения. Удалить элементы и сбалансировать дерево. Процедуру повторять до тех пор, пока не выполнится условие maxh = minh.

Прямой порядок обхода.

Листинг программы

```cpp
#include <iostream>
#include <vector>
#include <cmath>

using namespace std;

struct nodeptr
{
    int key;
    char height;
    nodeptr* left;
    nodeptr* right;
    nodeptr(int k) {
        key = k;
        left = right = nullptr;
        height = 1;
    }
};

char height(nodeptr* p) {
    return p ? p->height : 0;
}

int balancefact(nodeptr* p) {
    return height(p->right) - height(p->left);
}

void balanceheight(nodeptr* p) {
    char hr = height(p->right);
    char hl = height(p->left);
    p->height = (hr > hl ? hr : hl) + 1;
}

// правый поворот вокруг p
nodeptr* rotateright(nodeptr* p)
{
    nodeptr* q = p->left;
    p->left = q->right;
    q->right = p;
    balanceheight(p);
    balanceheight(q);
    return q;
}

// левый поворот вокруг q
nodeptr* rotateleft(nodeptr* q)
{
    nodeptr* p = q->right;
```

```
49      q->right = p->left;
50      p->left = q;
51      balanceheight(q);
52      balanceheight(p);
53      return p;
54  }
55
56  // балансировка узла p
57  nodeptr* balance(nodeptr* p)
58  {
59      balanceheight(p);
60      if (balancefact(p) == -2)
61      {
62          if (balancefact(p->left) > 0) {
63              p->left = rotateleft(p->left);
64              cout << "Сделан большой правый поворот" << endl;
65          }
66          return rotateright(p);
67      }
68      if (balancefact(p) == 2)
69      {
70          if (balancefact(p->right) < 0) {
71              p->right = rotateright(p->right);
72              cout << "Сделан большой левый поворот" << endl;
73          }
74          return rotateleft(p);
75      }
76      return p;
77  }
78
79  nodeptr* insert(nodeptr* p, int k)
80  {
81      if (!p) {
82          return new nodeptr(k);
83      }
84      if (k < p->key)
85          p->left = insert(p->left, k);
86      else if (k > p->key)
87          p->right = insert(p->right, k);
88      return balance(p);
89  }
90
91  void find_elem(nodeptr* p, int key, int go = 0) {
92      if (!p) {
93          cout << "Дерево пустое или такого элемента нет" << endl;
94          return;
95      }
96      else if (p->key == key) {
```

```
97              cout << "Элемент " << p->key << " найден за " << go << " шагов"
98          }
99          go++;
100         if (p->key < key) {
101             find_elem(p->right, key, go);
102         }
103         else {
104             find_elem(p->left, key, go);
105         }
106     }
107
108 int calc_height(nodeptr* p, int key, int go = 0) {
109     if (!p) {
110         return 0;
111     }
112     else if (p->key == key) {
113         return go;
114     }
115     go++;
116     if (p->key < key) {
117         calc_height(p->right, key, go);
118     }
119     else {
120         calc_height(p->left, key, go);
121     }
122 }
123
124 nodeptr* findMin(nodeptr* p)
125 {
126     if (!p)
127         return NULL;
128     else if (p->left == NULL)
129         return p;
130     else
131         return findMin(p->left);
132 }
133
134 nodeptr* removeMin(nodeptr* p)
135 {
136     if (p->left == nullptr)
137         return p->right;
138     p->left = removeMin(p->left);
139     return balance(p);
140 }
141
142 nodeptr* removeMax(nodeptr* p)
143 {
144     if (p->right == nullptr)
```

```
        return p->left;
    p->right = removeMax(p->right);
    return balance(p);
}

nodeptr* remove(nodeptr* p, int k)
{
    if (!p) return 0;
    if (k < p->key)
        p->left = remove(p->left, k);
    else if (k > p->key)
        p->right = remove(p->right, k);
    else
    {
        nodeptr* q = p->left;
        nodeptr* r = p->right;
        delete p;
        if (!r)
            return q;
        nodeptr* min = findMin(r);
        min->right = removeMin(r);
        min->left = q;
        return balance(min);
    }
    return balance(p);
}

//Прямой обход дерева
void PreOpder(nodeptr* p, vector <int>& pr) {
    if (!p) {
        return;
    }
    pr.push_back(p->key);
    PreOpder(p->left, pr);
    PreOpder(p->right, pr);
}

struct Trunk
{
    Trunk* prev;
    string str;

    Trunk(Trunk* prev, string str)
    {
        this->prev = prev;
        this->str = str;
    }
};
```

```
193
194 void showTrunks(Trunk* p)
195 {
196     if (p == nullptr) {
197         return;
198     }
199
200     showTrunks(p->prev);
201     cout << p->str;
202 }
203
204 void printTree(nodeptr* root, Trunk* prev, bool isLeft)
205 {
206     if (root == nullptr) {
207         return;
208     }
209
210     string prev_str = "    ";
211     Trunk* trunk = new Trunk(prev, prev_str);
212
213     printTree(root->right, trunk, true);
214
215     if (!prev) {
216         trunk->str = "———";
217     }
218     else if (isLeft)
219     {
220         trunk->str = ".———";
221         prev_str = "   |";
222     }
223     else {
224         trunk->str = "`———";
225         prev->str = prev_str;
226     }
227
228     showTrunks(trunk);
229     cout << " " << root->key << endl;
230
231     if (prev) {
232         prev->str = prev_str;
233     }
234     trunk->str = "   |";
235
236     printTree(root->left, trunk, false);
237 }
238
239 void count_list(nodeptr* p, int& count_lists_p) {
240     if (!p) {
```

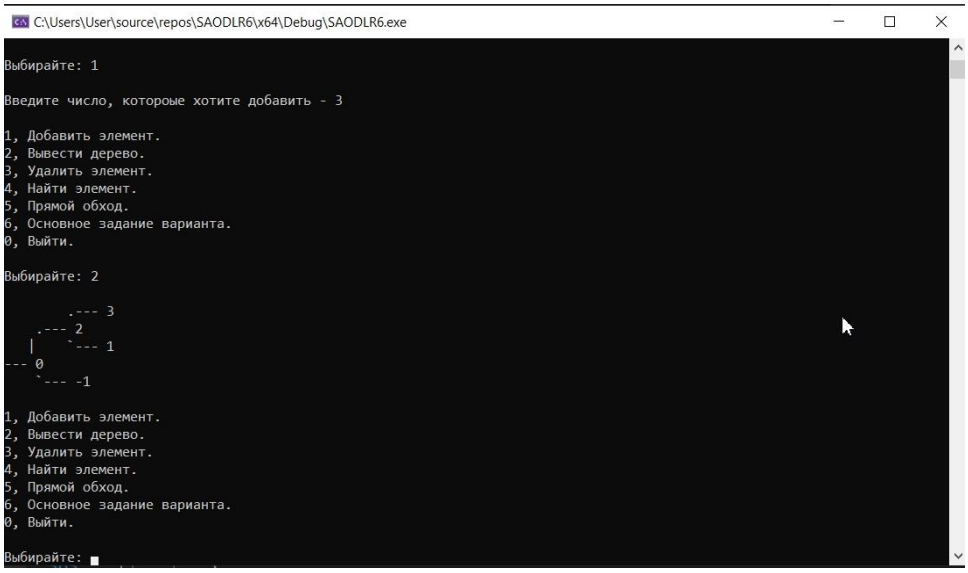
```
241             return;
242         }
243         if (p->left == nullptr && p->right == nullptr) {
244             count_lists_p += 1;
245         }
246         count_list(p->left, count_lists_p);
247         count_list(p->right, count_lists_p);
248     }
249
250     void fortask1(nodeptr* p, int& mx, nodeptr* p_d) {
251         if (p->left || p->right) {
252             if (p->left)
253                 fortask1(p->left, mx, p_d);
254             if (p->right)
255                 fortask1(p->right, mx, p_d);
256         }
257         else {
258             if (mx != calc_height(p_d, p->key)) {
259                 mx = max(calc_height(p_d, p->key), mx);
260             }
261         }
262     }
263
265         if (p->left || p->right) {
266             if (p->left)
267                 if (!p->left && !p->right) {
268                     mx = calc_height(p_d, p->key);
269                 }
270                 else {
271                     fortask1(p->left, mx, p_d);
272                 }
273             if (p->right)
274                 if (!p->left && !p->right) {
275                     mx = calc_height(p_d, p->key);
276                 }
277                 else {
278                     fortask1(p->left, mx, p_d);
279                 }
280         }
281     }
282
283     void task(nodeptr* p) {
284         int max_height = -1, count_lists = 0, min_height = -1;
285         nodeptr* p_d = p;
286         fortask1(p, max_height, p_d);
287         fortask2(p, min_height, p_d);
288         int tmp1 = max_height;
```

```
289         int tmp2 = min_height;
290         cout << "Высота самого верхнего листа = " << max_height << endl;
291         cout << "Высота самого нижнего листа = " << min_height << endl;
292         while (tmp1 != tmp2) {
293             removeMax(p_d);
294             max_height = -1;
295             fortask1(p, max_height, p_d);
296             if (tmp1 != max_height) {
297                 tmp1--;
298             }
299             printTree(p, nullptr, false);
300             cout << endl;
301         }
302     }
303
304     int main()
305     {
306         setlocale(0, "");
307         nodeptr* p = nullptr;
308         for (;;) {
309             cout << endl;
310             cout << "1, Добавить элемент.\n";
311             cout << "2, Вывести дерево.\n";
312             cout << "3, Удалить элемент.\n";
313             cout << "4, Найти элемент.\n";
314             cout << "5, Прямой обход.\n";
315             cout << "6, Основное задание варианта.\n";
316             cout << "0, Выйти.\n";
317             cout << endl;
318             cout << "Выбирайте: ";
319             int choise;
320             cin >> choise;
321             cout << endl;
322             switch (choise) {
323             case 1: {
324                 cout << "Введите число, которое хотите добавить - ";
325                 int k;
326                 cin >> k;
327                 p = insert(p, k);
328                 break;
329             }
330             case 2: {
331                 printTree(p, nullptr, false);
332                 break;
333             }
334             case 3: {
335                 cout << "Введите элемент, который хотите удалить - ";
336                 int k;
```

```
337             cin >> k;
338             p = remove(p, k);
339             break;
340         }
341         case 4: {
342             int k;
343             cout << "Введите элемент, который хотите найти - ";
344             cin >> k;
345             find_elem(p, k, 0);
346             break;
347         }
348         case 5: {
349             vector <int> pr;
350             PreOpder(p, pr);
351             for (int i = 0; i < pr.size(); i++) {
352                 cout << pr[i] << "\t";
353             }
354             break;
355         }
356         case 6: {
357             task(p);
358             printTree(p, nullptr, false);
359             break;
360         }
```

```
361         case 0: {
362             return 0;
363         }
364         }
365
366     }
367
368 }
```

Результаты:



Вывод

Изучил деревья поиска.