ГУАП

КАФЕДРА № 43

ОТЧЕТ   
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| старший преподаватель |  |  |  | С.А. Рогачев |
| 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. № | 4032 |  |  |  | М.В. Мухортов |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург 2021

**1. Цель работы**

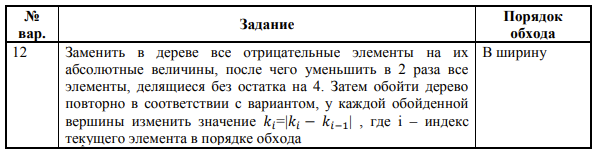
Целью работы является изучение деревьев поиска и получение практических навыков их использования

**2. Задание на лабораторную работу**

Разработать на языке программирования высокого уровня программу, которая должна выполнять следующие функции:   
 − добавлять элементы в сбалансированное дерево поиска;   
 − удалять элементы из сбалансированного дерева поиска;   
 − искать элементы в дереве поиска с выводом количества шагов, за которое осуществляется поиск;   
 − выводить дерево на экран (любым способом доступным для восприятия);   
 − выводить список, соответствующий обходу вершин, в соответствии с вариантом задания;   
 − осуществлять операцию, заданную в Таблице 1

**3. Вариант задания**

Вариант 12 приведён в Таблице 1.

 Таблица 1

**4. Текст программы**

**main.cpp**

1. #include <iostream>
2. #include "tree.h"
3. using namespace std;
4. int capacity = 0;
5. int main() {
6. setlocale(LC\_ALL, "rus");
7. tree\* root = nullptr;
8. int key = -1;
9. while (key != 0) {
10. cout << "Выберете действие:" << endl
11. << "[1] Вывод дерева на экран " << endl
12. << "[2] Добавление элементов" << endl
13. << "[3] Удаление элемента по его значению" << endl
14. << "[4] Поиск элемента по его значению" << endl
15. << "[5] Выполнить задание варианта 6" << endl
16. << "[0] Завершить выполнение программы" << endl << ">>> ";
17. cin >> key;
18. if (cin.get() != '\n') {
19. cin.clear();
20. while (cin.get() != '\n');
21. key = -1;
22. system("cls");
23. cout << "Ошибка ввода. Введите число повторно." << endl;
24. continue;
25. }
26. switch (key) {
27. case 1: {
28. system("cls");
29. key = -1;
30. if (capacity == 0) {
31. cout << "Дерево пустое" << endl << endl;
32. break;
33. }
34. print(root, 0);
35. cout << endl;
36. break;
37. }
38. case 2: {
39. system("cls");
40. key = -1;
41. while (key == -1) {
42. cout << "Введите количество вводимых элементов, либо [0] для выхода в меню:" << endl << endl << ">>> ";
43. cin >> key;
44. if (cin.get() != '\n') {
45. cin.clear();
46. while (cin.get() != '\n');
47. key = -1;
48. system("cls");
49. cout << "Ошибка ввода. Введите число повторно." << endl;
50. continue;
51. }
52. if (key < 0) {
53. key = -1;
54. system("cls");
55. cout << "Ошибка ввода. Введите положительное число повторно." << endl;
56. continue;
57. }
58. if (key == 0) {
59. system("cls");
60. key = -1;
61. break;
62. }
63. system("cls");
64. int buffer;
65. cout << "Введите значения элементов" << endl << endl;
66. for (int i = 0; i < key; i++) {
67. cout << ">>> ";
68. cin >> buffer;
69. if (cin.get() != '\n') {
70. cin.clear();
71. while (cin.get() != '\n');
72. cout << "Ошибка ввода. Введите число повторно." << endl;
73. i--;
74. continue;
75. }
76. if (search(root, buffer, -1) != nullptr) {
77. cout << "Ошибка! Элементы АВЛ-дерева не могут повторяться" << endl;
78. i--;
79. continue;
80. }
81. root = push(root, buffer);
82. buffer = 0;
83. capacity++;
84. }
85. system("cls");
86. cout << "Элементы были добавлено в дерево" << endl << endl;
87. key = -2;
88. }
89. break;
90. }
91. case 3: {
92. system("cls");
93. key = -1;
94. if (capacity == 0) {
95. cout << "Дерево пустое" << endl << endl;
96. break;
97. }
98. while (key == -1) {
99. cout << "Введите значение элемента:" << endl << endl << ">>> ";
100. cin >> key;
101. if (cin.get() != '\n') {
102. cin.clear();
103. while (cin.get() != '\n');
104. key = -1;
105. system("cls");
106. cout << "Ошибка ввода. Введите число повторно." << endl;
107. continue;
108. }
109. system("cls");
110. if (search(root, key, -1) == nullptr)
111. cout << "Этого элемента в дереве нет" << endl << endl;
112. else {
113. cout << "Элемент " << key << " удалён" << endl << endl;
114. root = remove(root, key);
115. capacity--;
116. }
117. key = -2;
118. break;
119. }
120. break;
121. }
122. case 4: {
123. system("cls");
124. key = -1;
125. if (capacity == 0) {
126. cout << "Дерево пустое" << endl << endl;
127. break;
128. }
129. while (key == -1) {
130. cout << "Введите значение элемента:" << endl << endl << ">>> ";
131. cin >> key;
132. if (cin.get() != '\n') {
133. cin.clear();
134. while (cin.get() != '\n');
135. key = -1;
136. system("cls");
137. cout << "Ошибка ввода. Введите число повторно." << endl;
138. continue;
139. }
140. system("cls");
141. if (search(root, key, 0) == nullptr)
142. cout << "Этого элемента в дереве нет" << endl << endl;
143. else
144. cout << "Элемент " << key << " найден в АВЛ-дереве" << endl << endl;
145. key = -2;
146. break;
147. }
148. break;
149. }
150. case 5: {
151. system("cls");
152. key = -1;
153. if (capacity == 0) {
154. cout << "Дерево пустое" << endl << endl;
155. break;
156. }
157. key = -1;
159. std::queue<tree\*> q = bfs(root);
160. std::queue<int> new\_q;
161. int tmp, prev, buf;
162. cout << "Исходное АВЛ-дерево:" << endl;
163. print(root, 0);
164. cout << endl << "Порядок обхода:" << endl;
165. while (!q.empty()) {
166. tmp = q.front()->key;
167. root = remove(root, tmp);
168. capacity--;
169. cout << tmp << " ";
170. if (tmp < 0)
171. tmp = abs(tmp);
172. new\_q.push(tmp);
173. q.pop();
174. }
175. while (!new\_q.empty()) {
176. if (search(root, new\_q.front(), -1) == nullptr) {
177. root = push(root, new\_q.front());
178. capacity++;
179. }
180. new\_q.pop();
181. }
182. cout << endl << endl << "Все отрицательные элементы заменены на их абсолютные величины. Повторения исключены. АВЛ-дерево:" << endl;
183. print(root, 0);
184. q = bfs(root);
185. cout << endl << "Порядок обхода:" << endl;
186. while (!q.empty()) {
187. tmp = q.front()->key;
188. root = remove(root, tmp);
189. capacity--;
190. cout << tmp << " ";
191. if ((tmp % 4) == 0)
192. tmp = tmp / 2;
193. new\_q.push(tmp);
194. q.pop();
195. }
196. while (!new\_q.empty()) {
197. if (search(root, new\_q.front(), -1) == nullptr) {
198. root = push(root, new\_q.front());
199. capacity++;
200. }
201. new\_q.pop();
202. }
203. cout << endl << endl << "Все элементы, делящиеся без остатка на 4 уменьшены в 2 раза. Повторения исключены. АВЛ-дерево:" << endl;
204. print(root, 0);
206. q = bfs(root);
207. prev = q.back()->key;
208. cout << endl << "Порядок обхода:" << endl;
209. while (!q.empty()) {
210. tmp = q.front()->key;
211. buf = tmp;
212. root = remove(root, tmp);
213. capacity--;
214. cout << tmp << " ";
215. tmp = abs(tmp - prev);
216. prev = buf;
217. new\_q.push(tmp);
218. q.pop();
219. }
220. while (!new\_q.empty()) {
221. if (search(root, new\_q.front(), -1) == nullptr) {
222. root = push(root, new\_q.front());
223. capacity++;
224. }
225. new\_q.pop();
226. }
227. cout << endl << endl << "Все элементы заменены на модуль разницы с предыдущим элементом. Повторения исключены." << endl <<
228. "АВЛ-дерево:" << endl;
229. print(root, 0);
230. cout << endl;
231. break;
232. }
233. case 0: {
234. system("cls");
235. cout << "Завершение работы программы" << endl;
236. key = 0;
237. delete\_tree(root);
238. break;
239. }
240. default:
241. system("cls");
242. key = -1;
243. cout << "Ошибка ввода. Введите число повторно." << endl;
244. break;
245. }
246. }

249. return 0;
250. }

**tree.h**

1. #pragma once
2. #include <queue>
3. #include <stack>
4. struct tree {
5. int key; // содержание узла
6. int h = 1; // высота узла
7. bool visited = false;
8. tree\* left = nullptr; // указатель на младшего потомка
9. tree\* right = nullptr; // указатель на старшего потомка
10. };
11. void print(tree\*, int); // печать дерева
12. int height(tree\*); // возвращает h высоту узла
13. int bfactor(tree\*); // вычисляет фактор баланса узла
14. void fixheight(tree\*); // вычисляет высоту узла (calc)
15. tree\* rotate\_r(tree\*); // малое правое вращение
16. tree\* rotate\_l(tree\*); // малое левое вращение
17. tree\* balance(tree\*); // балансировка узла
18. tree\* push(tree\*, int); // добавление нового узла в дерево (принимает корень)
19. tree\* search(tree\*&, int, int); // поиск по значению (принимает корень, ключ, счётчик)
20. tree\* remove(tree\*, int); // удаление по значению (принимает корень, ключ)
21. std::queue <tree\*> bfs(tree\*); // возвращает очередь обхода дерева в ширину
22. void delete\_tree(tree\*); // удаление всего дерева с освобождением памяти

**tree.cpp**

1. #include "tree.h"
2. #include <iostream>
3. using namespace std;
4. int height(tree\* R) {
5. if (R != nullptr) return R->h;
6. else return 0;
7. }
8. int bfactor(tree\* R) {
9. return height(R->right) - height(R->left);
10. }
11. void fixheight(tree\* R) {
12. int hl = height(R->left);
13. int hr = height(R->right);
14. if (hl > hr) R->h = hl + 1;
15. else R->h = hr + 1;
16. }
17. tree\* rotate\_r(tree\* p) {
18. tree\* q = p->left;
19. p->left = q->right;
20. q->right = p;
21. fixheight(p);
22. fixheight(q);
23. return q;
24. }
25. tree\* rotate\_l(tree\* q) {
26. tree\* p = q->right;
27. q->right = p->left;
28. p->left = q;
29. fixheight(q);
30. fixheight(p);
31. return p;
32. }
33. tree\* balance(tree\* R) {
34. fixheight(R);
35. if (bfactor(R) == 2) {
36. if (bfactor(R->right) < 0)
37. R->right = rotate\_r(R->right);
38. return rotate\_l(R);
39. }
40. if (bfactor(R) == -2) {
41. if (bfactor(R->left) > 0)
42. R->left = rotate\_l(R->left);
43. return rotate\_r(R);
44. }
45. return R;
46. }
47. tree\* push(tree\* R, int val) {
48. if (R == nullptr) {
49. tree\* root = new tree;
50. root->key = val;
51. root->right = nullptr;
52. root->left = nullptr;
53. R = root;
54. return R;
55. }
56. if (val < R->key)
57. R->left = push(R->left, val);
58. else
59. R->right = push(R->right, val);
60. return balance(R);
61. }
62. tree\* search(tree\*& R, int val, int k) {
63. if (R == nullptr) return nullptr;
64. if (val == R->key) {
65. if (k>-1)
66. cout << "Количествово шагов: " << k++ << endl;
67. return R;
68. }
69. if (k>-1) k++;
70. if (val < R->key) {
71. if (R->left == nullptr)
72. return nullptr;
73. search(R->left, val, k);
74. }
75. else {
76. if (R->right == nullptr)
77. return nullptr;
78. search(R->right, val, k);
79. }
80. }
81. tree\* find\_min(tree\* R) {
82. if (R->left == nullptr) return R;
83. else find\_min(R->left);
84. }
85. tree\* remove\_min(tree\* R) {
86. if (R->left == NULL) return R->right;
87. R->left = remove\_min(R->left);
88. return balance(R);
89. }
90. tree\* remove(tree\* R, int val) {
91. if (R == nullptr) return 0;
92. if (val < R->key)
93. R->left = remove(R->left, val);
94. else if (val > R->key)
95. R->right = remove(R->right, val);
96. else {
97. tree\* l = R->left;
98. tree\* r = R->right;
99. delete R;
100. if (r == nullptr) return l;
101. tree\* min = find\_min(r);
102. min->right = remove\_min(r);
103. min->left = l;
104. return balance(min);
105. }
106. return balance(R);
107. }
108. void print(tree\* R, int l) {
109. if (R != nullptr) {
110. print(R->right, l + 1);
111. for (int i = 1; i <= l; i++) cout << " ";
112. cout << R->key << endl;
113. print(R->left, l + 1);
114. }
115. }
116. std::queue <tree\*> bfs(tree\* R) {
117. std::queue <tree\*> q, res;
118. q.push(R);
119. tree\* tmp;
120. while (!q.empty()) {
121. res.push(q.front());
122. q.pop();
123. if (res.back()->left) {
124. q.push(res.back()->left);
125. }
126. if (res.back()->right) {
127. q.push(res.back()->right);
128. }
129. }
130. return res;
131. }
132. void delete\_tree(tree\* R) {
133. queue<tree\*> q = bfs(R);
134. stack<tree\*> s;
135. while (!q.empty()) {
136. s.push(q.front());
137. q.pop();
138. }
139. while (!s.empty()) {
140. R = remove(R, s.top()->key);
141. s.pop();
142. }
143. }

**5. Результат выполнения программы**

Результат выполнения программы показан на Рисунках 1-3.

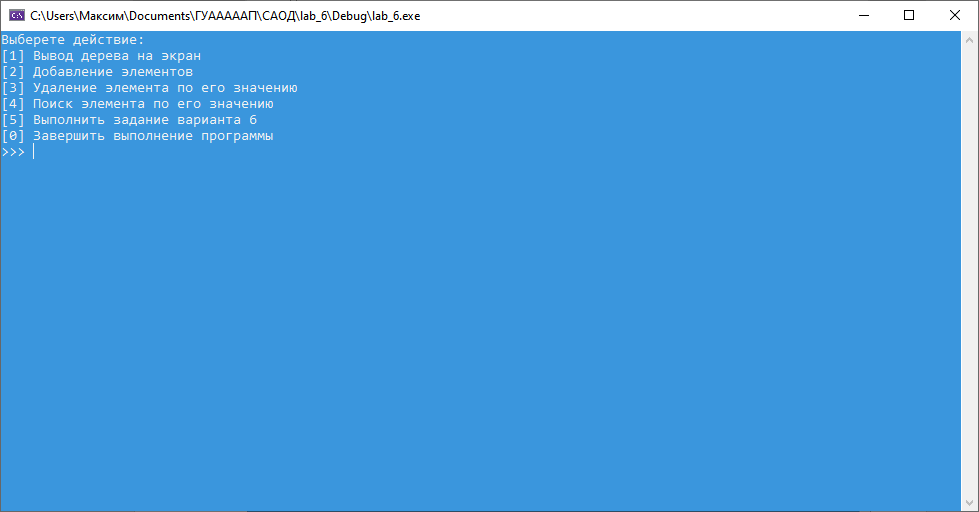


Рисунок 1. Пример выполнения программы, главное меню

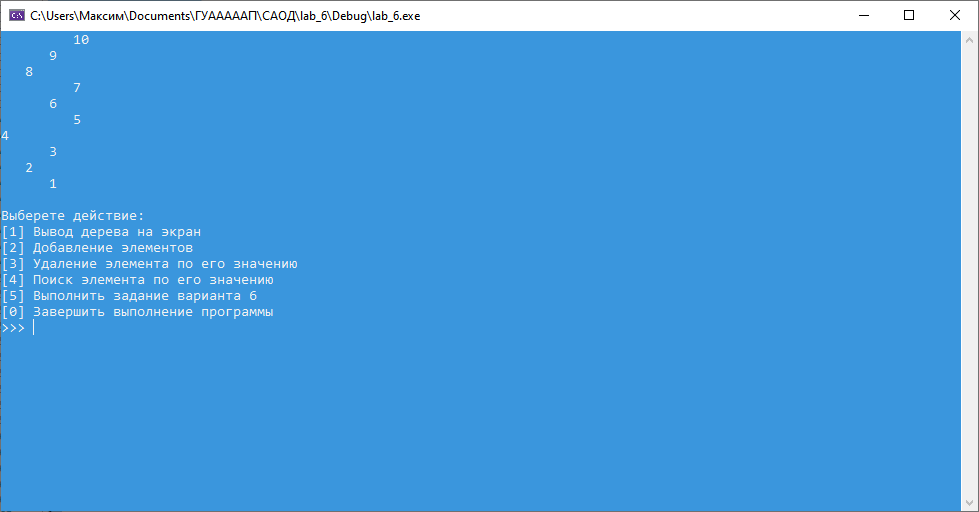


Рисунок 2. Пример выполнения программы, вывод дерева на экран

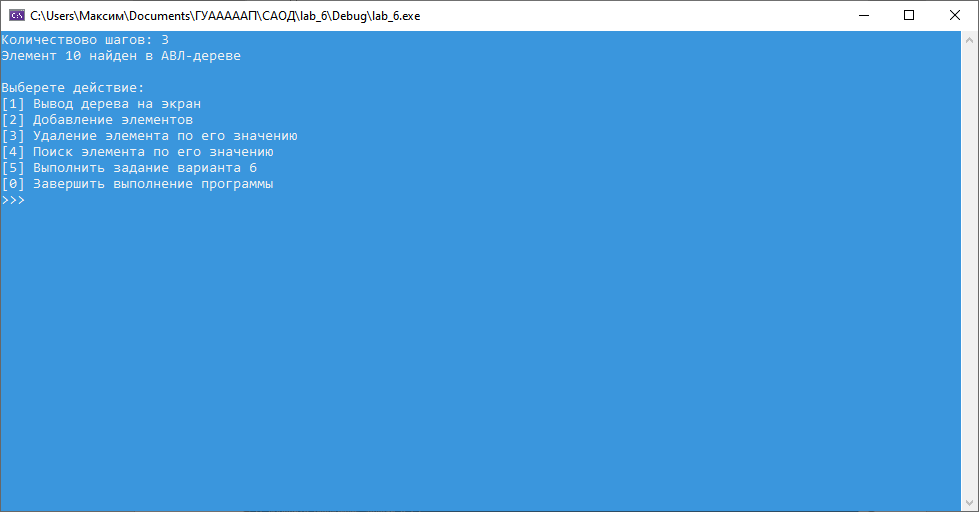


Рисунок 3. Пример выполнения программы, поиск по значению

**6. Выводы по работе**

В результате выполнения лабораторной работы:

* Были изучены и реализованы на практике такая структура данных, как АВЛ-дерево.
* Был реализован пользовательский интерфейс в виде консоли.
* Были учтены такие случаи, как некорректный пользовательский ввод, и ввод повторяющихся элементов дерева.
* Были реализованы функции левого и правого поворота дерева.
* Была реализована функция, возвращающая контейнер queue, содержащий указатели на листья дерева в порядке обхода в ширину.