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 |
|  |
|  |

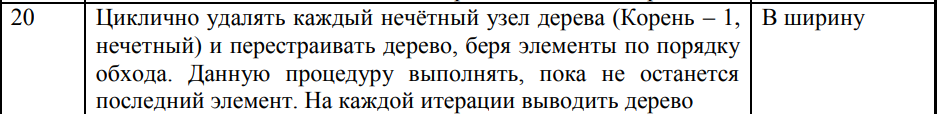
РАБОТУ ВЫПОЛНИЛ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТ ГР. № | 4131 |  |  |  | Д. А. Кузнецов |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург 2022

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

**Вариант:** 20



**Листинг программы:**

**Main.cpp**

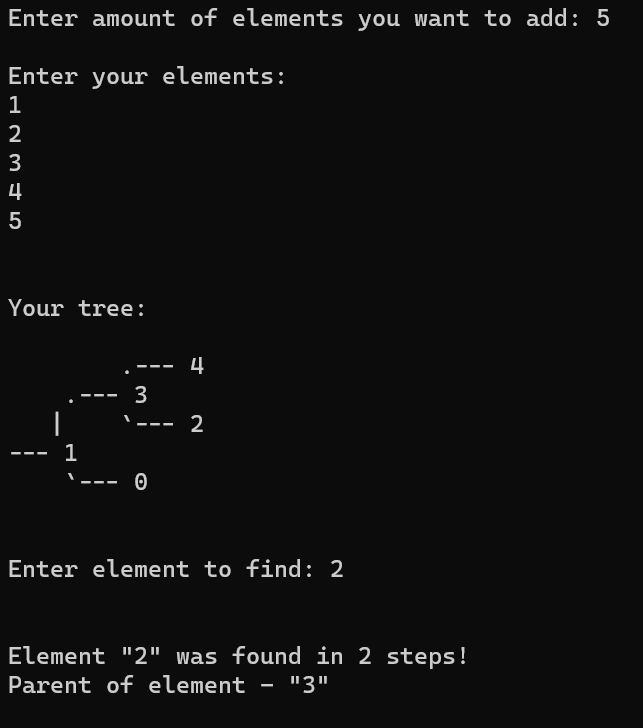
1. // Var 20
3. #include <iostream>
4. #include "Tree.h"
6. using namespace std;
8. int main() {
9. Tree tree;
11. int n, v;
12. cout << "Enter amount of elements you want to add: "; cin >> n;
13. cout << "\nEnter your elements:\n";
15. for (int i = 0; i < n; i++) {
16. cin >> v;
17. tree.add(i);
18. }
20. cout << "\n\nYour tree:\n\n";
21. tree.show();
23. cout << "\n\nEnter element to find: "; cin >> v;
24. tree.find(v);
26. cout << "\n\nEnter element to delete: "; cin >> v;
27. tree.remove(v);
29. cout << "\n\nResult\n\n";
30. tree.show();
32. cout << "\n\nBreadth first search\n\n";
33. tree.bfs();
35. cout << "\n\nFunction process\n\n";
36. tree.func();
37. }

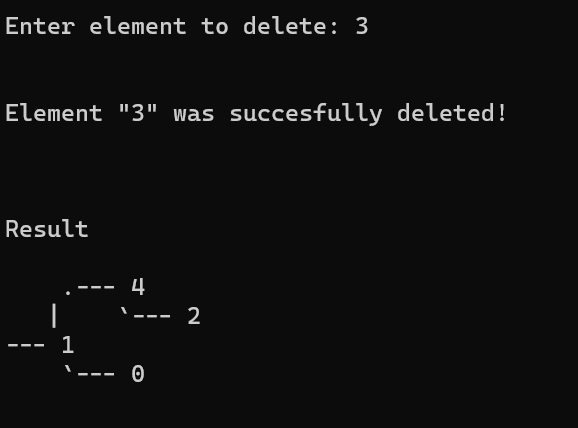
**Tree.cpp**

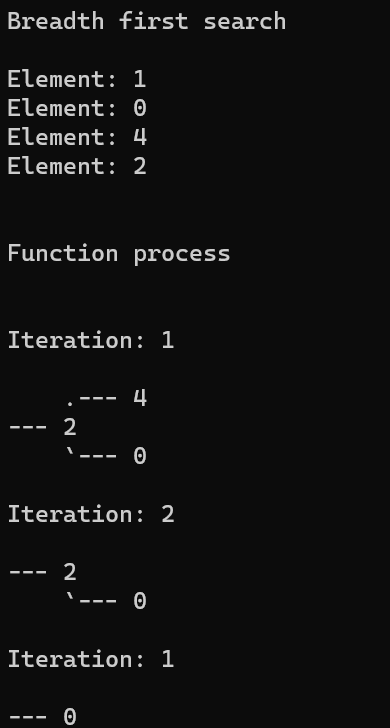
1. #include <iostream>
2. #include <queue>
3. #include "Tree.h"
5. using namespace std;
7. // Helper function to print branches of the binary tree
8. void Tree::showTrunks(Trunk\* p)
9. {
10. if (p == nullptr) {
11. return;
12. }
14. showTrunks(p->prev);
15. cout << p->str;
16. }
18. void Tree::printTree(Struct\* root, Trunk\* prev, bool isLeft) {
20. if (root == nullptr) {
21. return;
22. }
24. string prev\_str = " ";
25. Trunk\* trunk = new Trunk(prev, prev\_str);
27. Tree::printTree(root->right, trunk, true);
29. if (!prev) {
30. trunk->str = "---";
31. }
32. else if (isLeft)
33. {
34. trunk->str = ".---";
35. prev\_str = " |";
36. }
37. else {
38. trunk->str = "`---";
39. prev->str = prev\_str;
40. }
42. showTrunks(trunk);
43. cout << " " << root->value << endl;
45. if (prev) {
46. prev->str = prev\_str;
47. }
48. trunk->str = " |";
50. Tree::printTree(root->left, trunk, false);
51. }
53. void Tree::show() {
54. printTree(head, nullptr, false);
55. };
57. Struct\* Tree::insert(Struct\* node, Struct\* parent, int val) {
58. if (!node) {
59. Struct\* new\_data = new Struct();
60. new\_data->value = val;
61. new\_data->parent = parent;
62. return new\_data;
63. }
65. if (node->value == val) {
66. cout << "\n\nElement (" << val << ") is already exist!\n\n";
67. return node;
68. }
70. if (node->value > val) {
71. node->left = insert(node->left, node, val);
72. }
73. else {
74. node->right = insert(node->right, node, val);
75. }
77. return balance(node, parent);
78. }
80. void Tree::add(int val) {
81. head = insert(head, nullptr, val);
82. }
84. int Tree::getHeight(Struct\* node) {
85. return node ? node->height: 0;
86. }
88. int Tree::bfactor(Struct\* node) {
89. return getHeight(node->right) - getHeight(node->left);
90. }
92. void Tree::fixHeight(Struct\* node) {
93. int hl = getHeight(node->left);
94. int hr = getHeight(node->right);
96. node->height = (hl > hr ? hl : hr) + 1;
97. }
99. Struct\* Tree::getHead() {
100. return head;
101. }
103. Struct\* Tree::rotateRight(Struct\* node, Struct\* parent) {
104. Struct\* q = node->left;
105. Struct\* qr = q->right;
107. q->right = node; node->parent = q;
108. node->left = qr;
109. if (qr)
110. qr->parent = node;
111. q->parent = parent;
113. fixHeight(node);
114. fixHeight(q);
116. return q;
117. }
119. Struct\* Tree::rotateLeft(Struct\* node, Struct\* parent) {
120. Struct\* q = node->right;
121. Struct\* ql = q->left;
123. q->left = node; node->parent = q;
124. node->right = ql;
125. if (ql)
126. ql->parent = node;
127. q->parent = parent;
129. fixHeight(node);
130. fixHeight(q);
132. return q;
133. }
135. Struct\* Tree::balance(Struct\* node, Struct\* parent) {
136. fixHeight(node);
138. if (bfactor(node) == 2) {
139. if (bfactor(node->right) < 0) {
140. node->right = rotateRight(node->right, node);
141. }
142. return rotateLeft(node, parent);
143. }
145. if (bfactor(node) == -2) {
146. if (bfactor(node->left) > 0) {
147. node->left = rotateLeft(node->left, node);
148. }
149. return rotateRight(node, parent);
150. }
152. return node;
153. }
155. Struct\* Tree::findmin(Struct\* node) {
156. return (node->left ? findmin(node->left) : node);
157. }
159. Struct\* Tree::removemin(Struct\* node, Struct\* parent) {
160. if (node->left == nullptr) {
161. return node->right;
162. }
163. node->left = removemin(node->left, node);
164. return balance(node, parent);
165. }
167. Struct\* Tree::del(Struct\* node, Struct\* parent, int val) {
168. if (!node) {
169. cout << "\n\nElement \"" << val << "\" was not found!\n";
170. return node;
171. }
173. if (val < node->value) {
174. node->left = del(node->left, node, val);
175. }
176. else if (val > node->value) {
177. node->right = del(node->right, node, val);
178. }
179. else {
180. Struct\* l = node->left;
181. Struct\* r = node->right;
182. delete node;
184. if (!r) return l;
186. Struct\* min = findmin(r);
187. min->right = removemin(r, min);
188. min->left = l;
190. cout << "\n\nElement \"" << val << "\" was succesfully deleted!\n\n";
191. return balance(min, parent);
192. }
193. return balance(node, parent);
194. }
196. Struct\* Tree::del(Struct\* node, Struct\* parent) {
197. Struct\* l = node->left;
198. Struct\* r = node->right;
199. delete node;
201. if (!r) return l;
203. Struct\* min = findmin(r);
204. min->right = removemin(r, min);
205. min->left = l;
207. return balance(min, parent);
208. }
210. void Tree::remove(int value) {
211. head = del(head, head->parent, value);
212. }
214. Struct\* Tree::get(Struct\* node, int val, int step) {
216. if (node->left == nullptr && node->right == nullptr && node->value != val) {
217. cout << "\n\nElement \"" << val << "\" was not found!\n";
218. return nullptr;
219. }
220. else if (node->value == val) {
221. cout << "\n\nElement \"" << val << "\" was found in "<< step <<" steps!\n";
222. if (node->parent)
223. cout << "Parent of element - \"" << node->parent->value << "\"\n";
224. else
225. cout << "Parent of element - \"nullptr\"\n";
226. return node;
227. }
228. else if (node->value < val) {
229. node = get(node->right, val, step + 1);
230. }
231. else if (node->value > val) {
232. node = get(node->left, val, step + 1);
233. }
234. }
236. Struct\* Tree::find(int val) {
237. return get(head, val, 0);
238. }
240. void Tree::bfs\_func() {
241. queue<Struct\*\*> q;
242. bool parity = false;
243. int i = 0;
245. q.push(&head);
247. if (!head)
248. return;
250. while (!q.empty()) {
251. Struct\*\* cur = q.front(); q.pop();
253. if (cur) {
254. if (!parity) {
255. Struct\* tmp = \*cur;
256. \*cur = del(\*cur, tmp->parent);
258. cout << "\nIteration: " << ++i << "\n\n";
259. show();
260. }
262. parity = (parity ? false : true);
263. }
265. Struct\* tmp = \*cur;
266. if (tmp) {
267. if (tmp->left)
268. q.push(&tmp->left);
269. if (tmp->right)
270. q.push(&tmp->right);
271. }
273. }
274. }
276. void Tree::func() {
277. while (head->height > 1) {
278. bfs\_func();
279. }
280. };

283. void Tree::bfs() {
284. queue<Struct\*> q;
286. q.push(head);
288. if (!head)
289. return;
291. while (!q.empty()) {
292. Struct\* cur = q.front(); q.pop();
294. if (cur) {
295. cout << "Element: " << cur->value << endl;
296. }
298. if (cur->left)
299. q.push(cur->left);
300. if (cur->right)
301. q.push(cur->right);
302. }
303. }

**Контрольные примеры:**

****

****

****

**Вывод:**

В ходе лабораторной работы я изучил деревья поиска и получил практические навыки их использования.