Содержание

[Практическая работа №9 2](#_Toc58950156)

[Практическая работа №10 4](#_Toc58950157)

[Практическая работа №11 10](#_Toc58950158)

[Практическая работа №12 14](#_Toc58950159)

[Практическая работа №13 17](#_Toc58950160)

[Практическая работа №14 20](#_Toc58950161)

[Практическая работа №15 24](#_Toc58950162)

[Практическая работа №16 26](#_Toc58950163)

Практическая работа №9

**Вариант №3 – Префиксный калькулятор**

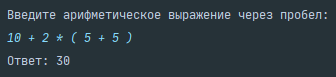
**Автор: Николаев-Аксенов И. С.**

**Группа: ИКБО-20-19**

**Код программы:**

1. **package** Calculator;
2. **import** java.util.Scanner;
3. **import** java.util.Stack;
5. **public** **class** Calculator {
6. **public** **static** **int** evaluate(String mathLine) {
7. **char**[] nums = mathLine.toCharArray();
9. Stack<Integer> values = **new** Stack<>();
10. Stack<Character> ops = **new** Stack<>();
12. **for** (**int** i = 0; i < nums.length; i++) {
13. **if** (nums[i] == ' ')
14. **continue**;
16. **if** (nums[i] >= '0' && nums[i] <= '9') {
17. StringBuffer sbuf = **new** StringBuffer();
18. **while** (i < nums.length && nums[i] >= '0' && nums[i] <= '9')
19. sbuf.append(nums[i++]);
20. values.push(Integer.parseInt(sbuf.toString()));
21. }
23. **else** **if** (nums[i] == '(')
24. ops.push(nums[i]);
26. **else** **if** (nums[i] == ')') {
27. **while** (ops.peek() != '(')
28. values.push(resultOperation(ops.pop(), values.pop(), values.pop()));
29. ops.pop();
30. }
32. **else** **if** (nums[i] == '+' || nums[i] == '-' || nums[i] == '\*' || nums[i] == '/') {
33. **while** (!ops.empty() && operationRangs(nums[i], ops.peek()))
34. values.push(resultOperation(ops.pop(), values.pop(), values.pop()));
36. ops.push(nums[i]);
37. }
38. }
40. **while** (!ops.empty())
41. values.push(resultOperation(ops.pop(), values.pop(), values.pop()));
43. **return** values.pop();
44. }
46. **public** **static** **boolean** operationRangs(**char** op1, **char** op2) {
47. **if** (op2 == '(' || op2 == ')')
48. **return** **false**;
50. **return** (op1 != '\*' && op1 != '/') || (op2 != '+' && op2 != '-');
51. }
53. **public** **static** **int** resultOperation(**char** op, **int** b, **int** a) {
54. **switch** (op) {
55. **case** '+':
56. **return** a + b;
57. **case** '-':
58. **return** a - b;
59. **case** '\*':
60. **return** a \* b;
61. **case** '/':
62. **if** (b == 0)
63. **throw** **new**
64. UnsupportedOperationException("На нуль делить нельзя!");
65. **return** a / b;
66. }
67. **return** 0;
68. }
70. **public** **static** **void** main(String[] args) {
71. Scanner input = **new** Scanner(System.in);
72. System.out.println("Введите арифметическое выражение через пробел: ");
73. String mathLine = input.nextLine();
74. System.out.print("Ответ: ");
75. System.out.println(Calculator.evaluate(mathLine));
76. }
77. }

**Результат выполнения программы:**



Практическая работа №10

**Бинарное дерево поиска. AVL дерево**

**Автор: Николаев-Аксенов И. С.**

**Группа: ИКБО-20-19**

**Код программы:**

*RedBlackTree.py:*

1. **import** sys

4. **class** Node():
5. **def** \_\_init\_\_(self, data):
6. self.data = data
7. self.parent = None
8. self.left = None
9. self.right = None
10. self.color = 1

13. **class** RedBlackTree():
14. **def** \_\_init\_\_(self):
15. self.TNULL = Node(0)
16. self.TNULL.color = 0
17. self.TNULL.left = None
18. self.TNULL.right = None
19. self.root = self.TNULL
21. **def** \_\_pre\_order\_helper(self, node):
22. **if** node != TNULL:
23. sys.stdout.write(node.data + " ")
24. self.\_\_pre\_order\_helper(node.left)
25. self.\_\_pre\_order\_helper(node.right)
27. **def** \_\_in\_order\_helper(self, node):
28. **if** node != TNULL:
29. self.\_\_in\_order\_helper(node.left)
30. sys.stdout.write(node.data + " ")
31. self.\_\_in\_order\_helper(node.right)
33. **def** \_\_post\_order\_helper(self, node):
34. **if** node != TNULL:
35. self.\_\_post\_order\_helper(node.left)
36. self.\_\_post\_order\_helper(node.right)
37. sys.stdout.write(node.data + " ")
39. **def** \_\_search\_tree\_helper(self, node, key):
40. **if** node == TNULL **or** key == node.data:
41. **return** node
43. **if** key < node.data:
44. **return** self.\_\_search\_tree\_helper(node.left, key)
45. **return** self.\_\_search\_tree\_helper(node.right, key)
47. **def** \_\_fix\_delete(self, x):
48. **while** x != self.root **and** x.color == 0:
49. **if** x == x.parent.left:
50. s = x.parent.right
51. **if** s.color == 1:
52. s.color = 0
53. x.parent.color = 1
54. self.left\_rotate(x.parent)
55. s = x.parent.right
57. **if** s.left.color == 0 **and** s.right.color == 0:
58. s.color = 1
59. x = x.parent
60. **else**:
61. **if** s.right.color == 0:
62. s.left.color = 0
63. s.color = 1
64. self.right\_rotate(s)
65. s = x.parent.right
67. s.color = x.parent.color
68. x.parent.color = 0
69. s.right.color = 0
70. self.left\_rotate(x.parent)
71. x = self.root
72. **else**:
73. s = x.parent.left
74. **if** s.color == 1:
75. s.color = 0
76. x.parent.color = 1
77. self.right\_rotate(x.parent)
78. s = x.parent.left
80. **if** s.left.color == 0 **and** s.right.color == 0:
81. s.color = 1
82. x = x.parent
83. **else**:
84. **if** s.left.color == 0:
85. s.right.color = 0
86. s.color = 1
87. self.left\_rotate(s)
88. s = x.parent.left
90. s.color = x.parent.color
91. x.parent.color = 0
92. s.left.color = 0
93. self.right\_rotate(x.parent)
94. x = self.root
95. x.color = 0
97. **def** \_\_rb\_transplant(self, u, v):
98. **if** u.parent == None:
99. self.root = v
100. **elif** u == u.parent.left:
101. u.parent.left = v
102. **else**:
103. u.parent.right = v
104. v.parent = u.parent
106. **def** \_\_delete\_node\_helper(self, node, key):
107. z = self.TNULL
108. **while** node != self.TNULL:
109. **if** node.data == key:
110. z = node
112. **if** node.data <= key:
113. node = node.right
114. **else**:
115. node = node.left
117. **if** z == self.TNULL:
118. **print**("Данный ключ не найден на дереве")
119. **return**
121. y = z
122. y\_original\_color = y.color
123. **if** z.left == self.TNULL:
124. x = z.right
125. self.\_\_rb\_transplant(z, z.right)
126. **elif** (z.right == self.TNULL):
127. x = z.left
128. self.\_\_rb\_transplant(z, z.left)
129. **else**:
130. y = self.minimum(z.right)
131. y\_original\_color = y.color
132. x = y.right
133. **if** y.parent == z:
134. x.parent = y
135. **else**:
136. self.\_\_rb\_transplant(y, y.right)
137. y.right = z.right
138. y.right.parent = y
140. self.\_\_rb\_transplant(z, y)
141. y.left = z.left
142. y.left.parent = y
143. y.color = z.color
144. **if** y\_original\_color == 0:
145. self.\_\_fix\_delete(x)
147. **def** \_\_fix\_insert(self, k):
148. **while** k.parent.color == 1:
149. **if** k.parent == k.parent.parent.right:
150. u = k.parent.parent.left  *# uncle*
151. **if** u.color == 1:
152. u.color = 0
153. k.parent.color = 0
154. k.parent.parent.color = 1
155. k = k.parent.parent
156. **else**:
157. **if** k == k.parent.left:
158. k = k.parent
159. self.right\_rotate(k)
160. k.parent.color = 0
161. k.parent.parent.color = 1
162. self.left\_rotate(k.parent.parent)
163. **else**:
164. u = k.parent.parent.right
166. **if** u.color == 1:
167. u.color = 0
168. k.parent.color = 0
169. k.parent.parent.color = 1
170. k = k.parent.parent
171. **else**:
172. **if** k == k.parent.right:
173. k = k.parent
174. self.left\_rotate(k)
175. k.parent.color = 0
176. k.parent.parent.color = 1
177. self.right\_rotate(k.parent.parent)
178. **if** k == self.root:
179. **break**
180. self.root.color = 0
182. **def** \_\_print\_helper(self, node, indent, last):
183. **if** node != self.TNULL:
184. sys.stdout.write(indent)
185. **if** last:
186. sys.stdout.write("R----")
187. indent += "     "
188. **else**:
189. sys.stdout.write("L----")
190. indent += "|    "
192. s\_color = "RED" **if** node.color == 1 **else** "BLACK"
193. **print**(str(node.data) + "(" + s\_color + ")")
194. self.\_\_print\_helper(node.left, indent, False)
195. self.\_\_print\_helper(node.right, indent, True)
197. **def** preorder(self):
198. self.\_\_pre\_order\_helper(self.root)
200. **def** inorder(self):
201. self.\_\_in\_order\_helper(self.root)
203. **def** postorder(self):
204. self.\_\_post\_order\_helper(self.root)
206. **def** searchTree(self, k):
207. **return** self.\_\_search\_tree\_helper(self.root, k)
209. **def** minimum(self, node):
210. **while** node.left != self.TNULL:
211. node = node.left
212. **return** node
214. **def** maximum(self, node):
215. **while** node.right != self.TNULL:
216. node = node.right
217. **return** node
219. **def** successor(self, x):
220. **if** x.right != self.TNULL:
221. **return** self.minimum(x.right)
222. y = x.parent
223. **while** y != self.TNULL **and** x == y.right:
224. x = y
225. y = y.parent
226. **return** y
228. **def** predecessor(self,  x):
229. **if** (x.left != self.TNULL):
230. **return** self.maximum(x.left)
232. y = x.parent
233. **while** y != self.TNULL **and** x == y.left:
234. x = y
235. y = y.parent
237. **return** y
239. **def** left\_rotate(self, x):
240. y = x.right
241. x.right = y.left
242. **if** y.left != self.TNULL:
243. y.left.parent = x
245. y.parent = x.parent
246. **if** x.parent == None:
247. self.root = y
248. **elif** x == x.parent.left:
249. x.parent.left = y
250. **else**:
251. x.parent.right = y
252. y.left = x
253. x.parent = y
255. **def** right\_rotate(self, x):
256. y = x.left
257. x.left = y.right
258. **if** y.right != self.TNULL:
259. y.right.parent = x
261. y.parent = x.parent
262. **if** x.parent == None:
263. self.root = y
264. **elif** x == x.parent.right:
265. x.parent.right = y
266. **else**:
267. x.parent.left = y
268. y.right = x
269. x.parent = y
271. **def** insert(self, key):
272. node = Node(key)
273. node.parent = None
274. node.data = key
275. node.left = self.TNULL
276. node.right = self.TNULL
277. node.color = 1
279. y = None
280. x = self.root
282. **while** x != self.TNULL:
283. y = x
284. **if** node.data < x.data:
285. x = x.left
286. **else**:
287. x = x.right
288. node.parent = y
289. **if** y == None:
290. self.root = node
291. **elif** node.data < y.data:
292. y.left = node
293. **else**:
294. y.right = node
296. **if** node.parent == None:
297. node.color = 0
298. **return**
300. **if** node.parent.parent == None:
301. **return**
303. self.\_\_fix\_insert(node)
305. **def** get\_root(self):
306. **return** self.root
308. **def** delete\_node(self, data):
309. self.\_\_delete\_node\_helper(self.root, data)
311. **def** pretty\_print(self):
312. self.\_\_print\_helper(self.root, "", True)

*startRBT.py:*

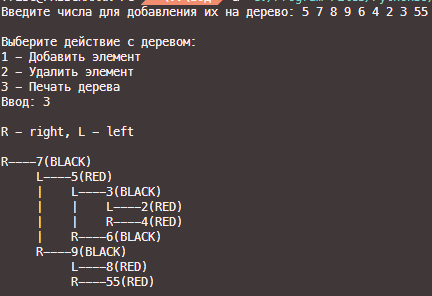
1. **from** RedBlackTree **import** RedBlackTree
2. rbt = RedBlackTree()

5. **def** menu():
6. x = int(input("**\n**Выберите действие с деревом:**\n**1 - Добавить элемент**\n**2 - Удалить элемент**\n**3 - Печать дерева**\n**Ввод: "))
7. **if** (x == 1):
8. rbt.insert(int(input("Введите число для добавления его на дерево: ")))
9. **print**("Число успешно добавлено на дерево!**\n**")
10. menu()
11. **elif** (x == 2):
12. rbt.delete\_node(
13. int(input("Введите число которое вы хотите удалить: ")))
14. **print**("Число успешно удалено из дерева!**\n**")
15. menu()
16. **elif** (x == 3):
17. **print**("**\n**R - right, L - left**\n**")
18. rbt.pretty\_print()
19. menu()
20. **else**:
21. **print**("Действие не найдено! Повторите ввод.")
22. menu()

25. **def** main():
26. numbers = list(
27. map(int, input("Введите числа для добавления их на дерево: ").split()))
29. **for** i **in** numbers:
30. rbt.insert(i)
32. menu()

35. **if** \_\_name\_\_ == "\_\_main\_\_":
36. main()

**Результат выполнения программы:**



Практическая работа №11

**Вариант №7 – Цепное хеширование.** Страховой полис: номер, компания, фамилия владельца.

**Автор: Николаев-Аксенов И. С.**

**Группа: ИКБО-20-19**

**Код программы:**

*Insurance.java:*

1. **public** **class** Insurance {
2. **int** number;
3. String company;
4. String surname;
6. **public** Insurance(**int** number, String company, String surname) {
7. **this**.number = number;
8. **this**.company = company;
9. **this**.surname = surname;
10. }
12. **public** **int** getNumber() {
13. **return** number;
14. }
16. @Override
17. **public** String toString() {
18. **return** "Insurance {" +
19. "number=" + number +
20. ", company='" + company + '**\'**' +
21. ", surname='" + surname + '**\'**' +
22. '}'+"**\n**";
23. }
24. }

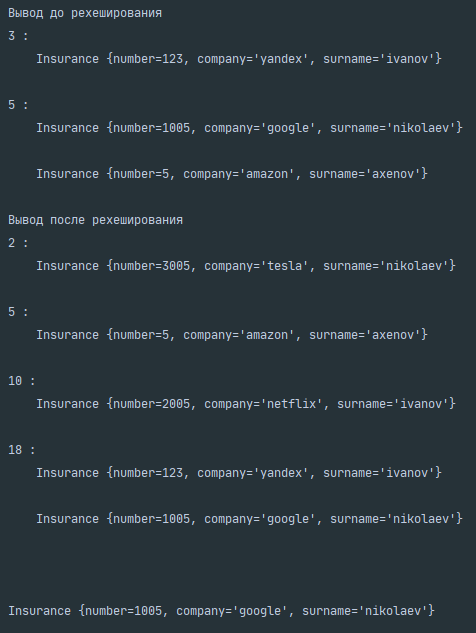
*HashTable.java:*

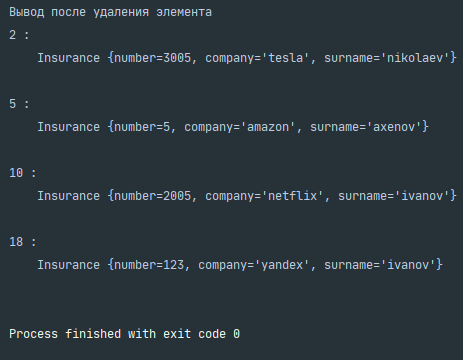
1. **import** java.util.\*;
3. **public** **class** HashTable< E> {
4. ArrayList<LinkedList<E>>table;
5. **int** size;
7. **public** HashTable(**int** size) {
8. **this**.size = size;
9. **this**.table = **new** ArrayList<>(**this**.size);
11. **for**(**int** i=0;i < 10;i++){
12. table.add(**new** LinkedList<E>());
13. }
14. }
16. **int** hash(**int** value){
17. **return** (**int**)value%size;
18. }
20. **int** hash(E n) {
21. Insurance key = (Insurance) n;
22. **return** key.number % size;
23. }
25. **void** add(E b) {
26. table.get(hash(b)).addLast(b);
27. **if** (table.get(hash(b)).size() > 2) rehash();
28. }
30. **void** rehash(){
31. ArrayDeque<E> t =**new** ArrayDeque<>();
32. **for** (**int** i = 0; i < size; ++i) {
33. **for** (E el : table.get(i)) {
34. t.add(el);
35. }
36. }
37. size = size \* 2 + 1;
38. table.clear();
39. table = **new** ArrayList<>(size);
40. **for**(**int** i=0;i<size;i++){
41. table.add(**new** LinkedList<E>());
42. }
43. **while** (!t.isEmpty()) {
44. add(t.getFirst());
45. t.pop();
46. }
47. }
48. **void** search(**int** value){
49. **for**(E t:table.get(hash(value))){
50. Insurance c= (Insurance) t;
51. **if**(c.number == value){
52. System.out.println(c);
53. }
54. }
55. }
57. **void** delete(**int** value){
58. **for**(E t:table.get(hash(value))){
59. Insurance c= (Insurance) t;
60. **if**(c.number == value){
61. table.get(hash(value)).remove(t);
62. }
63. }
64. }
65. **void** print(){
66. **for**(**int** i=0;i<size;i++){
67. **if**(!table.get(i).isEmpty()) {
68. System.out.println(i+" : ");
69. **for** (**int** j = 0; j < table.get(i).size(); j++) {
70. System.out.println("**\t**"+table.get(i).get(j));
71. }
72. }
73. }
74. }
75. }

*StartInsurance.java:*

1. **public** **class** StartInsurance {
2. **public** **static** **void** main(String[] args) {
3. HashTable<Insurance> ht = **new** HashTable<>(10);
4. ht.add(**new** Insurance(123,"yandex","ivanov"));
5. ht.add(**new** Insurance(1005,"google","nikolaev"));
6. ht.add(**new** Insurance(5,"amazon","axenov"));
8. System.out.println("Вывод до рехеширования");
9. ht.print();
10. ht.add(**new** Insurance(2005,"netflix","ivanov"));
11. ht.add(**new** Insurance(3005,"tesla","nikolaev"));
13. System.out.println("Вывод после рехеширования");
14. ht.print();
15. System.out.println("**\n**");
17. ht.search(1005);
18. System.out.println("**\n**");
20. ht.delete(1005);
21. System.out.println("**\n**");
23. System.out.println("Вывод после удаления элемента");
24. ht.print();
25. }
26. }

**Результат выполнения программы:**





Практическая работа №12

**Вариант №2 – Счет в банке.**

**Автор: Николаев-Аксенов И. С.**

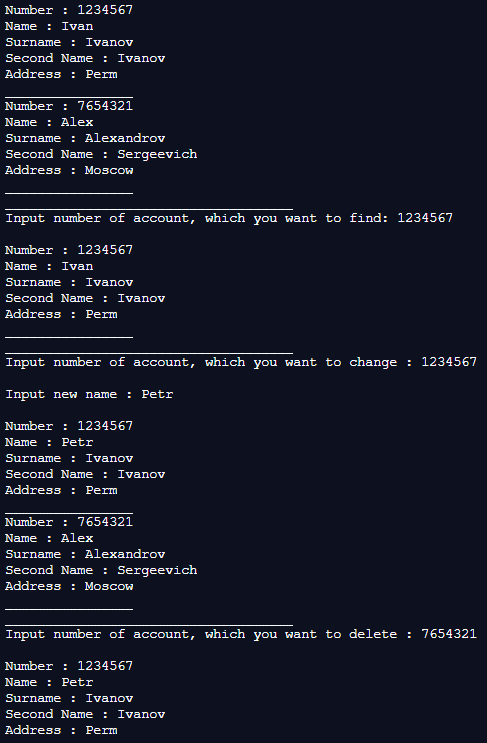
**Группа: ИКБО-20-19**

**Код программы:**

*BankAccount.cpp:*

1. #include <iostream>
2. #include <string>
3. #include <fstream>
4. #include <vector>
5. using namespace std;
7. struct bank\_account
8. {
9. int num;
10. string name;
11. string surname;
12. string second\_name;
13. string address;
14. };
15. vector<bank\_account>bk;
17. void print\_data(bank\_account a)
18. {
19. cout << "Number : " << a.num << "**\n**Name : " << a.name << "**\n**Surname : " << a.surname << "**\n**Second Name : " << a.second\_name << "**\n**Address : " << a.address << endl;
20. cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**\n**";
21. }
23. void write\_data\_to\_file()
24. {
25. ofstream ou("bank.txt");
27. if (ou.is\_open())
28. {
29. for (int i = 0; i < bk.size(); i++)
30. {
31. bank\_account temp = bk[i];
32. ou << temp.num << " " << temp.name << " "<< temp.surname << " "<< temp.second\_name << " "<< temp.address<< "**\n**";
33. }
34. }
35. ou.close();
36. }
38. void write\_data\_to\_binary()
39. {
40. bank\_account temp;
41. ofstream out("binary.txt", ostream::binary);
42. for (int i = 0; i <bk.size(); ++i)
43. out.write((char\*)&bk.at(i), sizeof(bank\_account));
45. out.close();
46. }
48. void read\_data\_from\_binary\_file()
49. {
50. ifstream fin("binary.txt", istream::binary);
51. bank\_account temp;
52. for (int i = 0; i < bk.size(); i++)
53. fin.read((char\*)&bk.at(i), sizeof(bank\_account));
55. fin.close();
56. }
58. void print\_all\_data()
59. {
60. for (int i = 0; i < bk.size(); i++)
61. print\_data(bk[i]);
62. }
64. void find()
65. {
66. int a;
67. cout << "Input number of account, which you want to find: ";
68. cin >> a;
69. cout << endl;
70. for (int i = 0; i < bk.size(); i++)
71. {
72. if (bk[i].num == a)
73. print\_data(bk[i]);
74. }
75. }
77. void change()
78. {
79. int a;
80. string b;
81. cout << "Input number of account, which you want to change : ";
82. cin >> a;
83. cout << endl;
85. cout << "Input new name : ";
86. cin >> b;
87. cout << endl;
89. read\_data\_from\_binary\_file();
90. for (int i = 0; i < bk.size(); i++)
91. {
92. if (bk[i].num == a)
93. bk[i].name = b;
94. }
95. write\_data\_to\_binary();
96. }
98. void del()
99. {
100. int a;
101. cout << "Input number of account, which you want to delete : ";
102. cin >> a;
103. cout << endl;
104. read\_data\_from\_binary\_file();
106. for (int i = 0; i < bk.size(); i++){
107. if (bk[i].num == a)
108. bk.erase(bk.begin() + i);
109. }
110. write\_data\_to\_binary();
111. }
113. int main()
114. {
115. bank\_account first{1234567, "Ivan", "Ivanov", "Ivanov", "Perm"};
116. bk.push\_back(first);
118. bank\_account second{7654321, "Alex", "Alexandrov", "Sergeevich", "Moscow"};
119. bk.push\_back(second);
121. write\_data\_to\_file();
122. print\_all\_data();
123. cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**\n**";
125. write\_data\_to\_binary();
126. read\_data\_from\_binary\_file();
127. find();
128. cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**\n**";
130. change();
131. print\_all\_data();
132. cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**\n**";
134. del();
135. print\_all\_data();
136. }

**Результат выполнения программы:**



Практическая работа №13

**Вариант №1 – Основные алгоритмы работы с графами.**

**Автор: Николаев-Аксенов И. С.**

**Группа: ИКБО-20-19**

**Задание:**

Составить программу реализации алгоритма Крускала построения остовного дерева минимального веса.

Выбрать и реализовать способ представления графа в памяти.

Предусмотреть ввод с клавиатуры произвольного графа.

Разработать доступный способ (форму) вывода результирующего дерева на экран монитора.

Провести тестовый прогон программы для заданного графа в соответствии с индивидуальным заданием

**Код программы:**

*Graph.java:*

1. **public** **class** Graph **implements** Comparable<Graph> {
2. **public** **int** A;
3. **public** **int** mass;
4. **public** **int** B;
6. Graph(**int** A, **int** mass, **int** B) {
7. **this**.A=A;
8. **this**.B=B;
9. **this**.mass=mass;
10. }
12. @Override
13. **public** **int** compareTo(Graph o) {
14. **if**(mass!=o.mass){
15. **return** mass<o.mass? -1:1;
16. }
17. **return** 0;
18. }
19. }

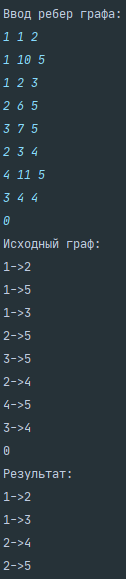
*SetGraph.java:*

1. **public** **class** SetGraph {
2. **int**[] number, rang;
4. SetGraph(**int** size) {
5. number=**new** **int**[size];
6. rang=**new** **int**[size];
7. **for**(**int** i=0;i<size;++i){
8. number[i]=i;
9. }
10. }
12. **int** set(**int** x) {
13. **return** x==number[x]? x:(number[x]=set(number[x]));
14. }
16. **boolean** merge(**int** A, **int** B){
17. **if**(set(A)==set(B))
18. **return** **false**;
20. **if**(rang[A]<rang[B])
21. number[A]=B;
23. **else** {
24. number[B]=A;
25. **if**(rang[A]==rang[B])
26. rang[A]++;
27. }
28. **return** **true**;
29. }
30. }

*StartKruskalsAlgorithm.java:*

1. **import** java.util.\*;
3. **public** **class** StartKruskalsAlgorithm {
4. **public** **static** **int** KruskalAlgorithm(ArrayList<Graph> graph) {
5. SetGraph union = **new** SetGraph(graph.size()+1);
6. Collections.sort(graph);
8. ArrayList<Graph> buff = **new** ArrayList<>();
9. **for**(Graph i: graph) {
10. **if**(union.merge(i.A,i.B)){
11. buff.add(i);
12. }
13. }
14. graph.clear();
15. graph.addAll(buff);
16. **return** 0;
17. }
19. **public** **static** **void** main(String[] args) {
20. ArrayList<Graph> graph = **new** ArrayList<>();
21. Scanner scanner = **new** Scanner(System.in);
22. **int** a,b,m;
24. System.out.println("Ввод ребер графа: ");
25. **while**(**true**){
26. a = scanner.nextInt();
27. **if**(a==0) **break**;
28. m = scanner.nextInt();
29. b = scanner.nextInt();
30. graph.add(**new** Graph(a,m,b));
31. }
32. System.out.println("Исходный граф: ");
33. **for** (Graph item : graph) {
34. System.out.println(item.A + "->" + item.B);
35. }
36. System.out.println(KruskalAlgorithm(graph));
37. System.out.println("Результат: ");
38. **for** (Graph value : graph) {
39. System.out.println(value.A + "->" + value.B);
40. }
41. System.out.println("В данной работе используется язык graphviz. Посмотреть граф можно на сайте graphvix.org.");
42. }
43. }

**Результат выполнения программы:**



|  |  |
| --- | --- |
| *Исходный граф:* | *Результат:* |
|  |  |

Практическая работа №14

**Алгоритмы сжатия и кодирования данных.**

**Автор: Николаев-Аксенов И. С.**

**Группа: ИКБО-20-19**

**Код программы:**

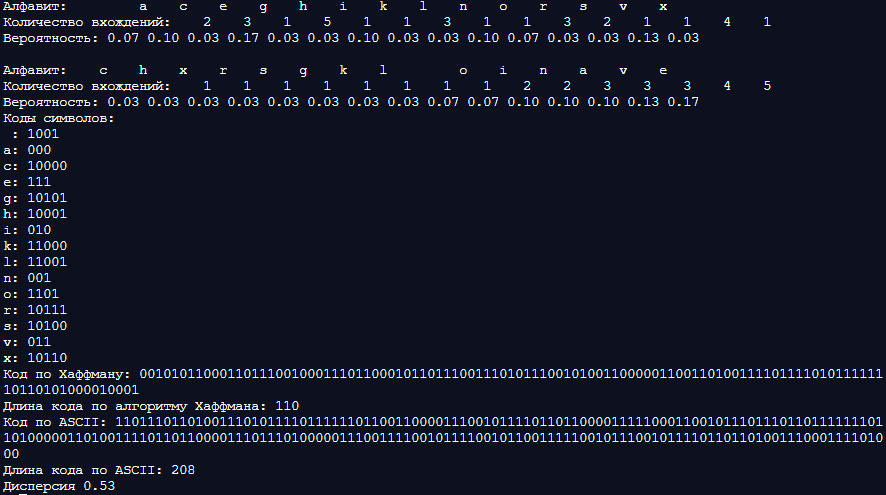
1. #include <iostream>
2. #include <vector>
3. #include <list>
4. #include <string>
5. #include <algorithm>
6. #include <clocale>
7. #include <map>
8. #include <iomanip>
9. #include <queue>
10. using namespace std;
11. map<char, string> codes;
12. map<char, int> freq;
14. struct MinHeapNode
15. {
16. char data;
17. int freq;
18. MinHeapNode\* left, \* right;
20. MinHeapNode(char data, int freq)
21. {
22. left = right = NULL;
23. this->data = data;
24. this->freq = freq;
25. }
26. };

29. struct compare
30. {
31. bool operator()(MinHeapNode\* l, MinHeapNode\* r)
32. {
33. return (l->freq > r->freq);
34. }
36. };


40. void storeCodes(struct MinHeapNode\* root, string str)
41. {
42. if (root == NULL)
43. return;
44. if (root->data != '$')
45. codes[root->data] = str;
46. storeCodes(root->left, str + "0");
47. storeCodes(root->right, str + "1");
48. }

51. priority\_queue<MinHeapNode\*, vector<MinHeapNode\*>, compare> minHeap;
52. template<typename T> void print\_queue(T& q,int size)
53. {
54. priority\_queue<MinHeapNode\*, vector<MinHeapNode\*>, compare> q1=q;
55. priority\_queue<MinHeapNode\*, vector<MinHeapNode\*>, compare> q2 = q;
57. cout << "Алфавит: ";
58. while(!q.empty())
59. {
60. if (q.top() != NULL)
61. cout << setw(4) << q.top()->data << " ";
63. q.pop();
64. }
66. cout << endl << "Количество вхождений: ";
67. while (!q1.empty())
68. {
69. if (q1.top() != NULL)
70. cout << setw(4) << q1.top()->freq << " ";
72. q1.pop();
73. }
75. cout << endl << "Вероятность: ";
76. while (!q2.empty())
77. {
78. if (q2.top() != NULL)
79. cout << setw(4) << float(q2.top()->freq)/size << " ";
81. q2.pop();
82. }
84. cout << '**\n**';
85. }
86. void HuffmanCodes(int size)
87. {
88. struct MinHeapNode\* left, \* right, \* top;
90. for (map<char, int>::iterator v = freq.begin(); v != freq.end(); v++)
91. minHeap.push(new MinHeapNode(v->first, v->second));
92. priority\_queue<MinHeapNode\*, vector<MinHeapNode\*>, compare> minHeap2=minHeap;
93. print\_queue(minHeap2,size);
95. while (minHeap.size() != 1)
96. {
97. left = minHeap.top();
98. minHeap.pop();
99. right = minHeap.top();
100. minHeap.pop();
101. top = new MinHeapNode('$', left->freq + right->freq);
102. top->left = left;
103. top->right = right;
104. minHeap.push(top);
105. }
106. storeCodes(minHeap.top(), "");
107. }
109. void calcFreq(string str, int n)
110. {
111. for (int i = 0; i < str.size(); i++)
112. freq[str[i]]++;
113. }
115. string haffman\_code(string input)
116. {
117. string encodedString;
118. for (auto i : input)
119. encodedString += codes[i];
120. return encodedString;
121. }
123. void print\_table(string str)
124. {
125. cout << "Алфавит: ";
126. for (auto item : freq)
127. cout << setw(4) << item.first << " ";
129. cout << endl << "Количество вхождений: ";
130. for (auto item : freq)
131. cout << setw(4) << item.second << " ";
133. cout << endl << "Вероятность: ";
134. for (auto item : freq) {
135. cout.setf(std::ios::fixed);
136. cout << setprecision(2) << float(item.second)/str.length() << " ";
137. }
139. cout << endl;
140. cout << endl;
141. }
143. void sviaz\_codov()
144. {
145. for (auto s : codes)
146. cout << s.first << ": " << s.second << endl;
147. }
149. int dec2bin(int num)
150. {
151. int bin = 0, k = 1;
152. while (num)
153. {
154. bin += (num % 2) \* k;
155. k \*= 10;
156. num /= 2;
157. }
158. return bin;
159. }
161. string ascii\_code(string input) {
162. string asci = "";
163. for (int i = 0; i < input.size(); ++i) {
164. asci += to\_string(dec2bin((int(input[i]))));
165. }
166. return asci;
167. }
169. void results(string a) {
170. cout << "Коды символов: " << endl; sviaz\_codov();
171. cout << "Код по Хаффману: " << haffman\_code(a) << endl;
172. cout << "Длина кода по алгоритму Хаффмана: " << haffman\_code(a).length() << endl;
173. cout << "Код по ASCII: " << ascii\_code(a) << endl;
174. cout << "Длина кода по ASCII: " << ascii\_code(a).length() << endl;
175. cout << "Дисперсия " << ((float)haffman\_code(a).length() / ascii\_code(a).length()) << endl;
176. }
178. int main()
179. {
180. setlocale(LC\_ALL, "Russian");
181. string str = "nikolaevaxenov ivan sergeevich";
182. calcFreq(str, str.length());
183. print\_table(str);
184. HuffmanCodes(str.length());
185. results(str);
186. return 0;
187. }

**Результат выполнения программы:**



Практическая работа №15

**Вариант №2 – Расстановка скобок.**

**Автор: Николаев-Аксенов И. С.**

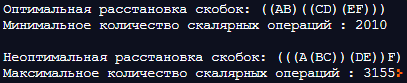
**Группа: ИКБО-20-19**

**Код программы:**

1. #include <iostream>
2. #include <limits.h>
3. using namespace std;
5. void printParenthesis(int i, int j, int n, int \*bracket, char &name)
6. {
7. if (i == j)
8. {
9. cout << name++;
10. return;
11. }
13. cout << "(";
15. printParenthesis(i, \*((bracket + j \* n) + i), n, bracket, name);
17. printParenthesis(\*((bracket + j \* n) + i) + 1, j, n, bracket, name);
18. cout << ")";
19. }
21. void matrixChainOrder(int p[], int n)
22. {
23. int min[n][n];
24. int max[n][n];
26. for (int i = 1; i < n; i++)
27. {
28. min[i][i] = 0;
29. max[i][i] = 0;
30. }
32. for (int L = 2; L < n; L++)
33. {
34. for (int i = 1; i < n - L + 1; i++)
35. {
36. int j = i + L - 1;
37. min[i][j] = INT\_MAX;
38. max[i][j] = INT\_MIN;
39. for (int k = i; k <= j - 1; k++)
40. {
41. int q = min[i][k] + min[k + 1][j] + p[i - 1] \* p[k] \* p[j];
42. if (q < min[i][j])
43. {
44. min[i][j] = q;
45. min[j][i] = k;
46. }
48. if (q >= max[i][j])
49. {
50. max[i][j] = q;
51. max[j][i] = k;
52. }
53. }
54. }
55. }
57. char matrixName = 'A';
58. cout << "Оптимальная расстановка скобок: ";
59. printParenthesis(1, n - 1, n, (int \*) min, matrixName);
60. cout << "**\n**Минимальное количество скалярных операций : " << min[1][n - 1] << endl;
62. matrixName = 'A';
63. cout << "**\n**Неоптимальная расстановка скобок: ";
64. printParenthesis(1, n - 1, n, (int \*) max, matrixName);
65. cout << "**\n**Максимальное количество скалярных операций : " << max[1][n - 1];
66. }

69. int main()
70. {
71. setlocale(LC\_ALL, "Russian");
73. int arr[] = {5, 10, 3, 12, 5, 50, 6};
75. int n = sizeof(arr) / sizeof(arr[0]);
76. matrixChainOrder(arr, n);
78. return 0;
79. }

**Результат выполнения программы:**



Практическая работа №16

**Вариант №6 – Реализовать задачу о рюкзаке методом ветвей и границ.**

**Автор: Николаев-Аксенов И. С.**

**Группа: ИКБО-20-19**

**Код программы:**

1. #include <iostream>
2. #include <algorithm>
3. #include <queue>
4. using namespace std;
6. struct Item
7. {
8. float weight;
9. int value;
10. };
12. struct Node
13. {
14. int level, profit, bound;
15. float weight;
16. };
18. bool cmp(Item a, Item b)
19. {
20. double r1 = (double) a.value / a.weight;
21. double r2 = (double) b.value / b.weight;
22. return r1 > r2;
23. }
25. int bound(Node u, int n, int knapsackWeight, Item arr[])
26. {
27. if (u.weight >= knapsackWeight)
28. return 0;
30. int profit\_bound = u.profit;
32. int j = u.level + 1;
33. int totalWeight = u.weight;
35. while ((j < n) && (totalWeight + arr[j].weight <= knapsackWeight))
36. {
37. totalWeight += arr[j].weight;
38. profit\_bound += arr[j].value;
39. j++;
40. }
42. if (j < n)
43. profit\_bound += (knapsackWeight - totalWeight) \* arr[j].value / arr[j].weight;
45. return profit\_bound;
46. }
48. int knapsack(int W, Item arr[], int n)
49. {
50. sort(arr, arr + n, cmp);
52. queue<Node> Q;
53. Node u, v;
55. u.level = -1;
56. u.profit = u.weight = 0;
57. Q.push(u);
59. int maxProfit = 0;
60. while (!Q.empty())
61. {
62. u = Q.front();
63. Q.pop();
65. if (u.level == -1)
66. v.level = 0;
68. if (u.level == n - 1)
69. continue;
71. v.level = u.level + 1;
72. v.weight = u.weight + arr[v.level].weight;
73. v.profit = u.profit + arr[v.level].value;
75. if (v.weight <= W && v.profit > maxProfit)
76. maxProfit = v.profit;
78. v.bound = bound(v, n, W, arr);
80. if (v.bound > maxProfit)
81. Q.push(v);
83. v.weight = u.weight;
84. v.profit = u.profit;
85. v.bound = bound(v, n, W, arr);
86. if (v.bound > maxProfit)
87. Q.push(v);
88. }
90. return maxProfit;
91. }
93. int main() {
94. setlocale(LC\_ALL, "Russian");
96. int W = 10;
97. Item arr[] = {{2,    40}, {3.14, 50}, {1.98, 120}, {5,    95}, {3,    30}};
98. int n = sizeof(arr) / sizeof(arr[0]);
100. cout << "Маскимальная стоимость: " << knapsack(W, arr, n);
102. return 0;
103. }

**Результат выполнения программы:**

