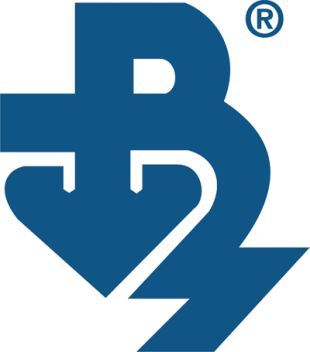
Технически университет – Варна

Катедра „Компютърни системи и технологии”



Курсова работа

по дисциплината:  
„Синтез и Анализ на Алгоритми”

на Петър Павлов Иванов

студент: спец. КСТ, 1 курс, 1б група, Ф. Номер: 19623317

Вариант 14. Съставете програма за работа с ориентиран граф, представен чрез списъци на съседство, която да прелага меню с основните операции със структурата граф. Да се напише функция, която визуализира и изтрива връх, който е с най-много изходящи дъги в дадения граф и се добави към менюто.

Сорс код на програмата:

1. #include<iostream>
2. using namespace std;
4. const int n = 15;
5. struct link
6. {
7. char key;
8. link\* next;
9. }
10. \*graph[n];
11. void init(link\* graph[n]);
12. int search\_node(link\* graph[n], char c);
13. int search\_arc(link\* graph[n], char c1, char c2);
14. void add\_node(link\* graph[n], char c);
15. void add\_arc(link\* graph[n], char c1, char c2);
16. void del\_node(link\* graph[n], char c);
17. void del\_arc(link\* graph[n], char c1, char c2);
18. void list\_node(link\* graph[n]);
19. void find\_and\_delete\_node\_with\_most\_exit\_arcs(link\* graph[n]);
20. int find\_nodes\_count(link\* graph[n]);
21. int main()
22. {
23. char node, secondNode;
24. int menu;
25. do
26. {
27. cout << "Menu**\n**";
28. cout << "1.Init**\n**";
29. cout << "2.Add node**\n**";
30. cout << "3.Add arc**\n**";
31. cout << "4.Del node**\n**";
32. cout << "5.Del arc**\n**";
33. cout << "6.Search node**\n**";
34. cout << "7.Search arc**\n**";
35. cout << "8.Print All**\n**";
36. cout << "9.Print the node with the mose exits**\n**";
37. cout << "10.Exit**\n**";
38. cin >> menu;
39. switch (menu)
40. {
41. case 1:
42. init(graph);
43. break;
44. case 2:
45. cout << "Enter node: ";
46. cin >> node;
47. add\_node(graph, node);
48. break;
49. case 3:
50. cout << "Enter first node: ";
51. cin >> node;
52. cout << "Enter second node: ";
53. cin >> secondNode;
54. add\_arc(graph, node, secondNode);
55. break;
56. case 4:
57. cout << "Enter node: ";
58. cin >> node;
59. del\_node(graph, node);
60. break;
61. case 5:
62. cout << "Enter first node: ";
63. cin >> node;
64. cout << "Enter second node: ";
65. cin >> secondNode;
66. del\_arc(graph, node, secondNode);
67. break;
68. case 6:
69. cout << "Enter node: ";
70. cin >> secondNode;
71. if (search\_node(graph, secondNode))
72. cout << "Search result : OK **\n**";
73. else
74. cout << "Search result: NO **\n**";
75. break;
76. case 7:
77. cout << "Enter first node: ";
78. cin >> node;
79. cout << "Enter second node: ";
80. cin >> secondNode;
81. if (search\_arc(graph, node, secondNode))
82. cout << "Search result : OK**\n**";
83. else
84. cout << "Search result : NO**\n**";
85. break;
86. case 8:
87. list\_node(graph);
88. break;
89. case 9:
90. find\_and\_delete\_node\_with\_most\_exit\_arcs(graph);
91. break;
92. default:
93. cout << "Invalid number. Try with numbers [1-10]!";
94. break;
95. }
96. } while (menu != 10);
97. cout << endl;
98. system("pause");
99. return 0;
100. }
101. int search\_node(link\* graph[n], char node)
102. {
103. int flag = 0;
104. for (int i = 0; i < n; i++) {
105. if (graph[i]) {
106. if (graph[i]->key == node) {
107. flag = 1;
108. }
109. }
110. }
111. return flag;
112. }
113. int search\_arc(link\* graph[n], char node, char secondNode)
114. {
115. int flag = 0;
116. if (search\_node(graph, node) && search\_node(graph, secondNode))
117. {
118. int i = 0;
119. while (graph[i]->key != node)
120. i++;
121. link\* p = graph[i];
122. while (p->key != secondNode && p->next != NULL)
123. p = p->next;
124. if (p->key == secondNode) {
125. flag = 1;
126. }
127. }
128. return flag;
129. }
130. void add\_node(link\* graph[n], char node)
131. {
132. if (search\_node(graph, node))
133. cout << "**\n** Existing node **\n**";
134. else
135. {
136. int j = 0;
137. while (graph[j] && j < n)
138. j++;
139. if (graph[j] == NULL)
140. {
141. graph[j] = new link;
142. graph[j]->key = node;
143. graph[j]->next = NULL;
144. }
145. else {
146. cout << "**\n** Overflow **\n**";
147. }
148. }
149. }
150. void add\_arc(link\* graph[n], char node, char secondNode)
151. {
152. if (search\_arc(graph, node, secondNode))
153. cout << "**\n** Existing arc **\n**";
154. else
155. {
156. if (!search\_node(graph, node)){
157. add\_node(graph, node);
158. }
159. if (!search\_node(graph, secondNode)) {
160. add\_node(graph, secondNode);
161. }
163. int i = 0;
164. link\* p;
165. while (graph[i]->key != node)
166. i++;
167. p = new link;
168. p->key = secondNode;
169. p->next = graph[i]->next;
170. graph[i]->next = p;
171. }
172. }
173. void list\_node(link\* graph[n])
174. {
175. for (int i = 0; i < n; i++)
176. if (graph[i])
177. cout << graph[i]->key;
178. cout << endl;
179. }
180. void init(link\* graph[n])
181. {
182. for (int i = 0; i < n; i++)
183. {
184. graph[i] = NULL;
185. }
186. }
187. void del\_node(link\* graph[n], char c)
188. {
189. if (search\_node(graph, c))
190. {
191. int i = 0;
192. link\* p = NULL, \* q = NULL;
193. while (graph[i]->key != c)
194. i++;
195. while (graph[i] != NULL)
196. {
197. p = graph[i];
198. graph[i] = graph[i]->next;
199. delete p;
200. }
201. for (i = 0; i < n; i++)
202. if (graph[i])
203. {
204. p = graph[i];
205. while (p->key != c && p->next != NULL)
206. {
207. q = p;
208. p = p->next;
209. }
210. if (p->key == c)
211. {
212. q->next = p->next;
213. delete p;
214. }
215. }
216. }
217. else
218. cout << "The mode is not in the graph";
219. cout << endl;
220. }
221. void del\_arc(link\* graph[n], char c1, char c2)
222. {
223. int i = 0; link\* p = NULL, \* q = NULL;
224. if (search\_arc(graph, c1, c2))
225. {
226. while (graph[i]->key != c1)
227. i++;
228. p = graph[i];
229. while (p->key != c2)
230. {
231. q = p;
232. p = p->next;
233. }
234. q->next = p->next;
235. delete p;
236. }
237. else
238. cout << "The arc is not in the graph";
239. cout << endl;
240. }
241. void find\_and\_delete\_node\_with\_most\_exit\_arcs(link\* graph[n])
242. {
243. int exit\_arcs\_per\_node[n];
244. int nodes\_count = find\_nodes\_count(graph);
245. int max = -1, index\_of\_max = 0;
247. for (int i = 0; i < nodes\_count; i++)
248. {
249. int current\_node\_arcs = 0;
250. for (int j = 0; j < n; j++)
251. {
252. if (graph[j] != NULL && j != i)
253. {
254. if (search\_arc(graph, graph[i]->key, graph[j]->key) == 1)
255. {
256. current\_node\_arcs++;
257. }
258. }
259. }
260. exit\_arcs\_per\_node[i] = current\_node\_arcs;
261. }
263. if (nodes\_count == 0) {
264. cout << "Could not find any arcs." << endl;
265. return;
266. }
267. for (int i = nodes\_count; i>=0; i--)
268. {
269. if (exit\_arcs\_per\_node[i] > max)
270. {
271. max = exit\_arcs\_per\_node[i];
272. index\_of\_max = i;
273. }
275. }
277. cout << "The node with the most exit arcs is: "<<graph[index\_of\_max]->key <<" with "<<max<<" exit arc(s).**\n**Node: " << graph[index\_of\_max]->key << " been deleted**\n**";
278. while (graph[index\_of\_max]->next != NULL) {
279. graph[index\_of\_max]->next = NULL;
280. }
281. del\_node(graph, graph[index\_of\_max]->key);
282. }
283. int find\_nodes\_count(link\* graph[n]) {
284. int i = 0;
285. while (true) {
286. if (graph[i] == NULL)
287. {
288. break;
289. }
290. i++;
291. }
292. return i;
293. }

* void init(link\* graph[n]) – Инициализира граф.
* int search\_node(link\* graph[n], char c) – търси даден връх от посочения граф и връща 1 ако го намери и 0 ако не го намери.
* int search\_arc(link\* graph[n], char c1, char c2) – търси ориентирана дъга от посочените граф и върхове и връща 1 ако я намери или 0 ако не я намери.
* void add\_node(link\* graph[n], char c) – Добавя връх в посочения граф.
* void add\_arc(link\* graph[n], char c1, char c2) – добавя ориентирана дъга в посочения граф между подадените върхове.
* void del\_node(link\* graph[n], char c) – изтрива даден връх от посочения граф.
* void del\_arc(link\* graph[n], char c1, char c2) – изтрива ориентирана дъга между посочените върхове в дадения граф.
* void list\_node(link\* graph[n]) – принтира върховете на графът.
* void find\_and\_delete\_node\_with\_most\_exit\_arcs(link\* graph[n]) – намира върхът с най много изходни дъги и го изтрива.
* int find\_nodes\_count(link\* graph[n]) – намира количеството върхове в посочения граф.
* int main() – меню с опции от [1-10].

**Използвана литература**

* Синтез и Анализ на Алгоритми – ръководство за лабораторни упражнения – Е. Рачева, Н. Николов, А. Иванова - Варна 2008
* stackoverflow.com
* w3schools.com