**数据结构第七次上机实验报告**

学号：161730126 姓名：李双玖

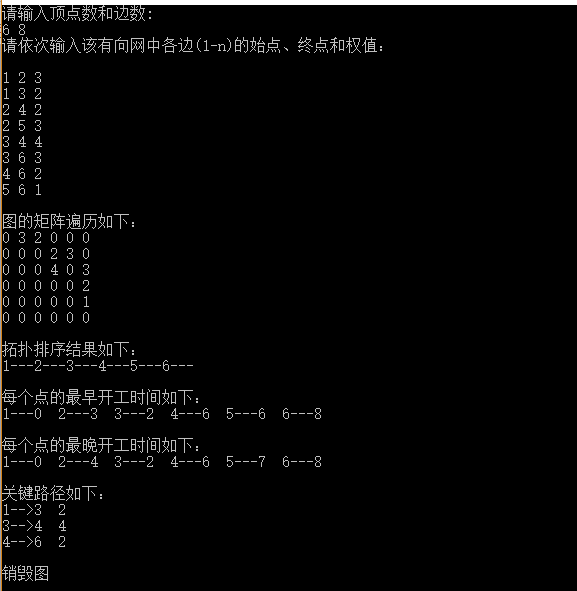
1. 调试成功程序及说明

**1、**

**题目：**关键路径。

**算法思想：**1）总体步骤为：先求出拓扑排序的结果，然后在求工程中各点的最早时间和最晚时间，最后判断每个阶段是否为关键路径。2）求最早时间时，顺序遍历拓扑排序的结果，将当前点的，所有入度点最早时间+边权重中的最大值，赋给当前点的最早时间。3）求最晚时间，逆序遍历拓扑排序结果，将当前点的，所有出度点最晚时间-边权重中的最小值，赋给当前点的最晚时间。

**运行结果：**



**结果分析：**成功。

**源程序：**

1. /\*
2. 关键路径
3. 6 8
5. 1 2 3
6. 1 3 2
7. 2 4 2
8. 2 5 3
9. 3 4 4
10. 3 6 3
11. 4 6 2
12. 5 6 1
13. \*/
14. #include<iostream>
15. #include<cstdlib>
16. #include<vector>
18. **using** **namespace** std;

21. // 有向图
22. **typedef** **struct** {
23. **char** \*vexs;             // 顶点值的向量
24. **int** vexNum, arcNum;     // 图的顶点数和边数
25. **int** \*\*arcs;             // 边的矩阵
26. }MGraph;

29. // 路径
30. **typedef** **struct** {
31. **int** from, to;
32. **int** weight;
33. }Arc;

36. // 创建图
37. **void** CreateMGraph(MGraph &G) {
38. cout<<"请输入顶点数和边数:\n";
39. cin>>G.vexNum>>G.arcNum;
41. // 动态初始化图，将每一条边的值设为0
42. G.arcs = (**int**\*\*)malloc(**sizeof**(**int**\*) \* (G.vexNum+1));
43. **for** (**int** i=0; i<=G.vexNum; i++) {
44. G.arcs[i] = (**int**\*)malloc(**sizeof**(**int**) \* (G.vexNum+1));
45. **for** (**int** j=0; j<=G.vexNum; j++) {
46. G.arcs[i][j] = 0;       // 用0表示这两条边不相邻
47. }
48. }
50. cout<<"请依次输入该有向网中各边(1-n)的始点、终点和权值：\n";
52. **int** x, y, weight;
53. **for** (**int** i=1; i<=G.arcNum; i++) {
54. cin>>x>>y>>weight;
55. **if** (x == y) {
56. cout<<"不能出现环！\n";
57. i--;
58. **continue**;
59. }
60. **if** (x > G.vexNum) {
61. cout<<"不存在该点！\n";
62. i--;
63. **continue**;
64. }
65. **if** (y > G.vexNum) {
66. cout<<"不存在该点！\n";
67. i--;
68. **continue**;
69. }
70. **if** (weight == 0) {
71. cout<<"权重不能为0！\n";
72. i--;
73. **continue**;
74. }
75. G.arcs[x][y] = weight;
76. }
77. }

80. // 销毁图
81. **void** DestroyGraph(MGraph &G) {
82. cout<<"\n销毁图\n";
83. **for** (**int** i=0; i<=G.vexNum; i++) {
84. free(G.arcs[i]);
85. }
86. free(G.arcs);
87. G.vexNum = G.arcNum = 0;
88. }

91. // 遍历图
92. **void** Traverse(MGraph G) {
93. cout<<"\n图的矩阵遍历如下：\n";
94. **for** (**int** i=1; i<=G.vexNum; i++) {
95. **for** (**int** j=1; j<=G.vexNum; j++) {
96. cout<<G.arcs[i][j]<<" ";
97. }
98. cout<<endl;
99. }
100. cout<<endl;
101. }

104. // 拓扑排序
105. vector<**int**> TopoSort(MGraph &G) {
106. **int** \*inCount = (**int**\*)malloc(**sizeof**(**int**) \* (G.vexNum+1));    // 定义一个记录每个点入度值的数组
107. // 初始化入度值数组
108. **for** (**int** i=0; i<=G.vexNum; i++) {
109. inCount[i] = 0;
110. }
111. // 求每个点的入度
112. **for** (**int** i=1; i<=G.vexNum; i++) {
113. **for** (**int** j=1; j<=G.vexNum; j++) {
114. // 如果i，j位置不为零，则说明i->j，j点的入度加一
115. **if** (G.arcs[i][j]) inCount[j]++;
116. }
117. }
119. vector<**int**> topoRes;      // 拓扑排序后的路径
120. **bool** hasZero = **true**;
121. **while** (hasZero) {
122. hasZero = **false**;
123. **for** (**int** i=1; i<=G.vexNum; i++) {
124. // 如果存在入度为0的顶点
125. **if** (!inCount[i]) {
126. hasZero = **true**;
127. inCount[i]--;
128. topoRes.push\_back(i);
129. // 将以该点为始点的点的入度都减一
130. **for** (**int** j=1; j<=G.vexNum; j++) {
131. **if** (G.arcs[i][j]) inCount[j]--;
132. }
133. }
134. }
135. }
137. free(inCount);
139. **if** (topoRes.size() < G.vexNum) {
140. cout<<"有环！\n";
141. getchar();
142. exit(0);
143. }
145. **return** topoRes;
146. }

149. **int** main() {
150. MGraph G;
151. CreateMGraph(G);
152. Traverse(G);

155. vector<**int**> topoRes = TopoSort(G);
156. cout<<"拓扑排序结果如下：\n";
157. **for** (**int** i=0; i<topoRes.size(); i++) {
158. cout<<topoRes[i]<<"---";
159. }
160. cout<<endl<<endl;

163. **int** \*ve = (**int**\*)malloc(**sizeof**(**int**) \* (G.vexNum+1));     // 每个点最早时间
164. **int** \*vl = (**int**\*)malloc(**sizeof**(**int**) \* (G.vexNum+1));     // 每个点最晚时间
166. // 各点的最早开工时间
167. **for** (**int** n=0; n<topoRes.size(); n++) {
168. **int** j = topoRes[n];
169. // 取所将所有以j为终点的点i中，i点的最早时间加上i->j的时间中，的最大值
170. **int** max = 0;
171. **for** (**int** i=1; i<=G.vexNum; i++) {
172. **if** (G.arcs[i][j]) {
173. **if** (ve[i] + G.arcs[i][j] > max) max = ve[i] + G.arcs[i][j];
174. }
175. }
176. ve[j] = max;
177. }
178. cout<<"每个点的最早开工时间如下：\n";
179. **for** (**int** i=1; i<=G.vexNum; i++) {
180. cout<<i<<"---"<<ve[i]<<"  ";
181. }
182. cout<<endl<<endl;
184. // 各点的最晚开工时间
185. **int** last = topoRes[G.vexNum-1]; // 最后一个工程点，即终点
186. vl[last] = ve[last];    // 将终点的最晚开工时间设为最早开工时间
187. **for** (**int** n=G.vexNum-2; n>=0; n--) {
188. **int** i = topoRes[n];
189. // 取所将所有以i为始点的点j中，j点的最晚时间减去i->j的时间中，的最小值
190. **int** min = INT\_MAX;
191. **for** (**int** j=1; j<=G.vexNum; j++) {
192. **if** (G.arcs[i][j] && (vl[j] - G.arcs[i][j] < min)) {
193. min = vl[j] - G.arcs[i][j];
194. }
195. }
196. vl[i] = min;
197. }
198. cout<<"每个点的最晚开工时间如下：\n";
199. **for** (**int** i=1; i<=G.vexNum; i++) {
200. cout<<i<<"---"<<vl[i]<<"  ";
201. }
202. cout<<endl<<endl;

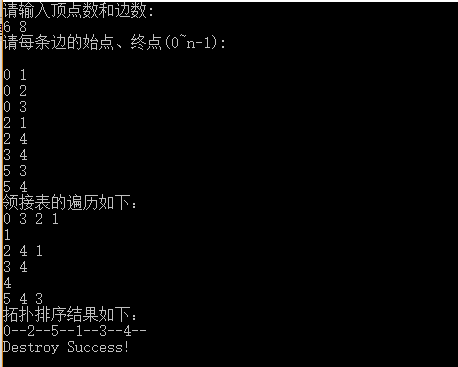
205. // 求关键路径
206. vector<Arc> importPath;
207. **for** (**int** n=0; n<topoRes.size(); n++) {
208. **int** i = topoRes[n];
209. **for** (**int** j=1; j<=G.vexNum; j++) {
210. **if** (G.arcs[i][j] && (vl[j]-G.arcs[i][j] == ve[i])) {
211. Arc arc;
212. arc.from = i;
213. arc.to = j;
214. arc.weight = G.arcs[i][j];
215. importPath.push\_back(arc);
216. }
217. }
218. }
219. cout<<"关键路径如下：\n";
220. **for** (**int** i=0; i<importPath.size(); i++) {
221. cout<<importPath[i].from<<"-->"<<importPath[i].to<<"  "<<importPath[i].weight<<endl;
222. }
224. free(ve);
225. free(vl);
227. DestroyGraph(G);
228. **return** 0;
229. }

**2、**

**题目：**编写拓扑排序算法。

**算法思想：**定义一个数组，序号代表点的序号，值代表每个点的入度值。每次遍历该数组，将入度为0的点取出放入拓扑排序好的路径最后，并将以该点为入度点的顶点的入度减一。

**运行结果：**



**结果分析：**成功。

**源程序：**

1. /\*
2. 编写拓扑排序算法
3. 6 8
5. 0 1
6. 0 2
7. 0 3
8. 2 1
9. 2 4
10. 3 4
11. 5 3
12. 5 4
13. \*/
14. #include<iostream>
15. #include<stdlib.h>
16. #include<vector>
18. **using** **namespace** std;
20. **typedef** **struct** VNode {
21. **int** data;
22. VNode\* next;
23. }VNode, \*VList;
25. // 有向图的领接表表示法
26. **typedef** **struct** {
27. **int** vexnum, arcnum;
28. VList\* vexList;
29. }ALGraph;
31. **void** CreateALGraph(ALGraph &G) {
32. cout<<"请输入顶点数和边数:\n";
33. cin>>G.vexnum>>G.arcnum;
35. // 初始化领接表
36. G.vexList = (VList\*)malloc(**sizeof**(VList) \* G.vexnum);
37. **for** (**int** i=0; i<G.vexnum; i++) {
38. VNode\* temp = (VNode\*)malloc(**sizeof**(VNode));
39. temp->data = i;
40. temp->next = NULL;
41. G.vexList[i] = temp;
42. }
44. cout<<"请每条边的始点、终点(0~n-1):\n";
45. **for** (**int** i=0; i<G.arcnum; i++) {
46. **int** x = 0, y = 0;
47. VNode\* temp = (VNode\*)malloc(**sizeof**(VNode));
48. cin>>x>>y;
49. temp->data = y;
50. temp->next = NULL;
51. temp->next = G.vexList[x]->next;
52. G.vexList[x]->next = temp;
53. }
54. }
56. **void** DestroyALGraph(ALGraph &G) {
57. **for** (**int** i=0; i<G.vexnum; i++) {
58. VNode \*p = G.vexList[i], \*temp;
59. **while** (p) {
60. temp = p;
61. p = p->next;
62. free(temp);
63. }
64. }
65. cout<<"\nDestroy Success!\n";
66. }
68. **void** TraverseALGraph(ALGraph G) {
69. cout<<"领接表的遍历如下：\n";
70. **for** (**int** i=0; i<G.vexnum; i++) {
71. VNode \* p = G.vexList[i];
72. **while** (p) {
73. cout<<p->data<<" ";
74. p = p->next;
75. }
76. cout<<endl;
77. }
78. }
80. **int** main() {
81. ALGraph G;
82. CreateALGraph(G);
83. TraverseALGraph(G);
85. **int** inCount[G.vexnum];
86. **for** (**int** i=0; i<G.vexnum; i++) {
87. inCount[i] = 0;
88. }
89. **for** (**int** i=0; i<G.vexnum; i++) {
90. VNode \* p = G.vexList[i]->next;
91. **while** (p) {
92. inCount[p->data]++;
93. p = p->next;
94. }
95. }
97. //  for (int i=0; i<G.vexnum; i++) {
98. //      cout<<inCount[i]<<"--";
99. //  }
100. //  cout<<endl;
102. vector<**int**> res;
103. **while** (1) {
104. **bool** noZero = **true**;
105. **for** (**int** i=0; i<G.vexnum; i++) {
106. **if** (inCount[i] == 0) {
107. //cout<<"===="<<i<<endl;
108. inCount[i]--;
109. noZero = **false**;
110. res.push\_back(i);
111. VNode \* p = G.vexList[i]->next;
112. **while** (p) {
113. inCount[p->data]--;
114. p = p->next;
115. }
116. }
117. }
118. **if** (noZero) **break**;
119. }
120. **if** (res.size() < G.vexnum) {
121. cout<<"有环！\n";
122. getchar();
123. **return** 0;
124. }
126. cout<<"拓扑排序结果如下：\n";
127. **for** (**int** i=0; i<G.vexnum; i++) {
128. cout<<res[i]<<"--";
129. }
131. DestroyALGraph(G);
132. **return** 0;
133. }

**3、**

**题目：**问题描述：小刘承包了很多片麦田，为了灌溉这些麦田，小刘在第一个麦田挖了一口很深的水井，所有的麦田都从这口井来引水灌溉。 为了灌溉，小刘需要建立一些水渠，以连接水井和麦田，小刘也可以利用部分麦田作为“中转站”，利用水渠连接不同的麦田，这样只要一片麦田能被灌溉，则与其连接的麦田也能被灌溉。现在小刘知道哪些麦田之间可以建设水渠和建设每个水渠所需要的费用（注意不是所有麦田之间都可以建立水渠）。请问灌溉所有麦田最少需要多少费用来修建水渠。

输入格式：输入的第一行包含两个正整数n, m，分别表示麦田的片数和小刘可以建立的水渠的数量。麦田使用1, 2, 3, ……依次标号。 接下来m行，每行包含三个整数ai, bi, ci，表示第ai片麦田与第bi片麦田之间可以建立一条水渠，所需要的费用为ci。

输出格式：输出一个整数，表示灌溉所有麦田所需要的最小费用，及水渠连接说明。

**问题分析：**这个问题可以用最小生成树算法实现。

输入样例:

4 4

1 2 1

2 3 4

2 4 2

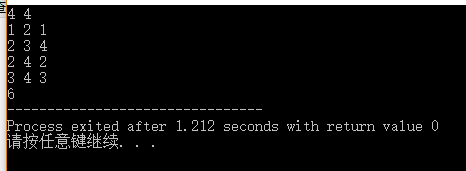
3 4 3

输出样例:

6

**算法思想：**Kruskal算法，用并查集来判断连通性。

**运行结果：**



**结果分析：**成功。

**源程序：**

1. #include<cstdio>
2. #include<algorithm>
4. **using** **namespace** std;
6. **typedef** **struct** {
7. **int** from, to;
8. **int** weight;
9. }Arc;
11. Arc arcs[100005];
12. **int** points[1005];       // 并查集
14. // 找爸爸
15. **int** find(**int** n) {
16. **return** n == points[n] ? n : points[n] = find(points[n]);
17. }
19. **bool** cmp(Arc a, Arc b) {
20. **return** a.weight < b.weight;
21. }
23. **int** main() {
24. **long** **long** n=0, m=0;
25. scanf("%d%d", &n, &m);
27. **for** (**int** i=0; i<n; i++) points[i] = i;
29. **for** (**int** i=0; i<m; i++) {
30. scanf("%d%d%d", &arcs[i].from, &arcs[i].to, &arcs[i].weight);   // 用cin比scanf慢
31. arcs[i].from--;
32. arcs[i].to--;
33. }
35. // 对m条边，进行快速排序
36. sort(arcs, arcs+m, cmp);
38. **int** sum = 0;
39. **for** (**int** i=0; i<m; i++) {
40. **int** x = find(arcs[i].from);
41. **int** y = find(arcs[i].to);
42. **if** (x != y) {
43. points[x] = y;
44. sum += arcs[i].weight;
45. }
46. }
48. printf("%d", sum);
49. **return** 0;
50. }