**实 验 报 告**

|  |  |  |
| --- | --- | --- |
| **课程名称**：数据结构 | **班级**：软工21101 | **实验成绩**： |
| **实验名称**：图的应用-邻接矩阵存储结构 | **学号**：2105006207 | **批阅教师签字：董傲霜** |
| **实验编号**：实验九 | **姓名**：方福涛 | **实验日期：**2022 年 11 月12日 |
| **指导教师**：董傲霜 | **组号**：wuyanzu | **实验时间**： 时 分－ 时 分 |

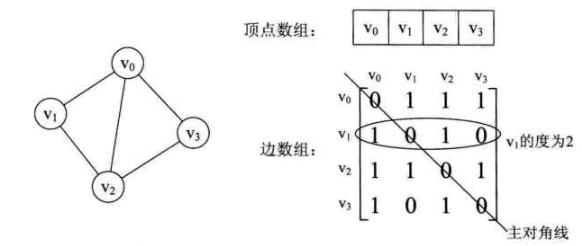
1. **实验设计思想**

说明图的邻接矩阵存储结构的设计思想。

———————————————————————————————————————

\*邻接矩阵用一个一维数组来存储各个顶点的信息，用一个二维数组来存储各个顶点之间的连接关系\*

**例如：**



我们很容易理解顶点数组的意义，那么如何理解边数组呢？下面做出解释:以顶点V0为例，保持V0所在行标不变，改变列标，由于V0与V0自己无法连接，故为0；而V0到V1由于有连接，故V0行V1列值为1，其他以此类推。

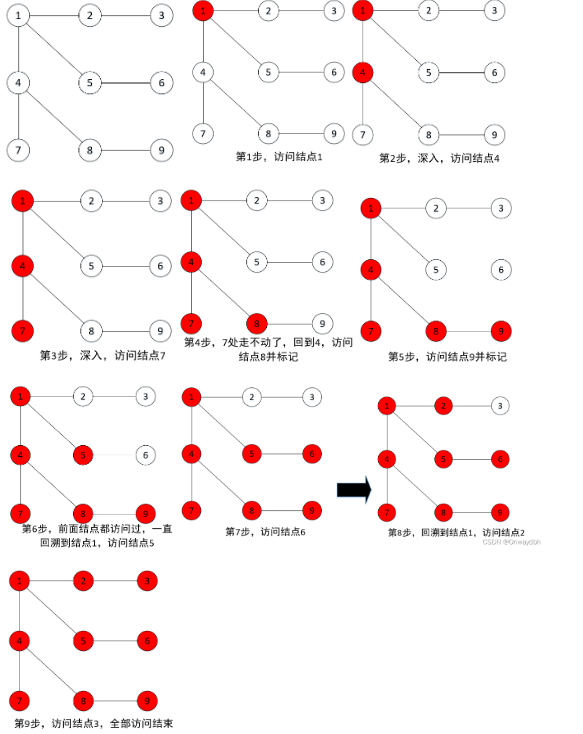
———————————————————————————————————————

**二、程序说明**

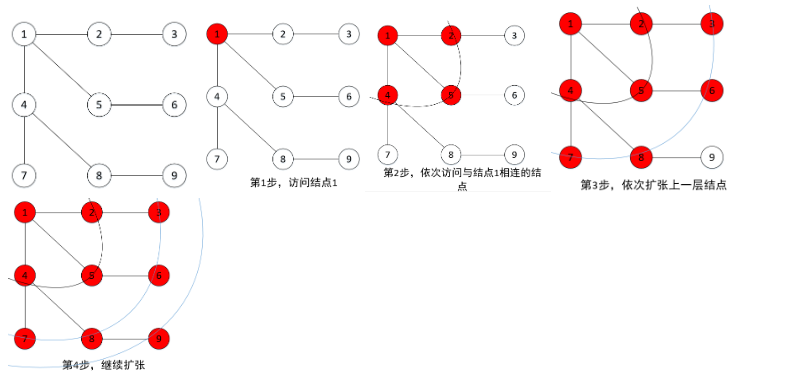
1. 给出图的邻接矩阵存储结构定义及其属性含义的说明；
2. #define MAX\_VERTEX\_NUM 50 //最大顶点数
3. typedef char VertexData; //设顶点的数据类型为字符型
4. typedef int WeightType; //假设边的权值类型为整型
5. typedef struct{
6. int data[MAX\_VERTEX\_NUM]; //存储顶点
7. int front;
8. int rear; //指向左边和右边
9. } Queue;
10. typedef struct{
11. VertexData vers[MAX\_VERTEX\_NUM]; //顶点表
12. WeightType arcs[MAX\_VERTEX\_NUM][MAX\_VERTEX\_NUM]; //邻接矩阵
13. int vernum;
14. int arcnum; //图的当前点数和边数
15. int kind; //0 direct 1 undirect
16. } AdjMatrix;
17. 给出你设计的函数的结构说明（函数，参数，执行的结果）
18. void InitQueue(Queue\* q)
19. {
20. q->front = 0;
21. q->rear = 0;
22. }
23. void enQueue(Queue \*q,int v)
24. {
25. q->rear = (q->rear + 1) % MAX\_VERTEX\_NUM;
26. q->data[q->rear] = v;
27. }
28. void deQueue(Queue \*q, int\* v)
29. {
30. q->front = (q->front + 1) % MAX\_VERTEX\_NUM;
31. \*v = q->data[q->rear];
32. }
33. bool empty(Queue \*q)
34. {
35. return q->front == q->rear;
36. }
37. int locateVertex(AdjMatrix \*G,VertexData v)
38. {
39. //v 存在返回下标 不存在返回1
40. int i;
41. for(i = 0; i < G->vernum; i++)
42. {
43. if(G->vers[i] == v)
44. {
45. break;
46. }
47. }
48. if(i < G->vernum)
49. {
50. return i;
51. }
52. else{
53. return -1;
54. }
55. }
56. void createAdjGraph(AdjMatrix \*G)
57. {
58. int i,j;
59. char v1,v2;
60. int w,k;
61. //v 存在返回下标 不存在返回1
62. scanf("%d",&(G->kind));
63. scanf("%d",&(G->vernum));
64. scanf("%d",&(G->arcnum));
65. //(2)初始化邻接矩阵
66. for(i = 0;i < G->vernum; i++)
67. {
68. for(j = 0;j < G->vernum; j++)
69. {
70. if(i==j)
71. {
72. G->arcs[i][j] = 0;
73. }
74. else{
75. G->arcs[i][j] = 88888;  //最大值
76. }
77. }
78. }
79. //(3)读入顶点个数
80. for(i = 0; i < G->vernum; i++)
81. {
82. scanf("%c",&(G->vers[i]));
83. }
84. //(4)读入边的个数
85. //AB56   ('A','B','56')
86. for(k = 0; k < G->arcnum; k++)
87. {
88. scanf("%c%c%d",&v1,&v2,&w);
89. i = locateVertex(G,v1);
90. j = locateVertex(G,v2);
91. if(i != -1 && j != -1)
92. {
93. G->arcs[i][j] = w;
94. if(G->kind == 1)
95. {
96. G->arcs[j][i] = w;
97. }
98. }
99. }
100. }
101. //深度优先遍历的递归程序
102. void rdfs(AdjMatrix \*G,int v,bool \*visited)
103. {
104. int k;
105. printf("%c ",G->vers[v]);
106. visited[v] = true;
107. for(k = 0; k < G->vernum; k++)
108. {
109. if(G->arcs[v][k] != 0 && G->arcs[v][k] != 88888)
110. {
111. if(!visited[k])
112. {
113. rdfs(G,k,visited);
114. }
115. }
116. }
117. }
118. //深度优先遍历的主程序
119. void dfs(AdjMatrix \*G)
120. {
121. int k;
122. bool visited[MAX\_VERTEX\_NUM];
123. //获取状态数组并初始化
124. for (int i = 0; i < MAX\_VERTEX\_NUM; i++)
125. visited[i] = false;
126. //检查未被访问过的顶点
127. for(k = 0; k < G->vernum; k++)
128. {
129. if(!visited[k])
130. {
131. rdfs(G, k, visited);
132. }
133. }
134. }
135. //广度优先遍历的递归程序
136. void bfs(AdjMatrix \*G,int v0,bool \*visited)
137. {
138. int k,v;
139. Queue qu;
140. InitQueue(&qu);
141. printf("%c ",G->vers[v0]);
142. visited[v0] = true;
143. enQueue(&qu,v0);
144. while(!empty(&qu))
145. {
146. deQueue(&qu,&v);
147. for(k = 0; k < G->vernum; k++)
148. {
149. if(G->arcs[v][k] !=88888 && G->arcs[v][k] !=0)
150. {
151. if(!visited[k])
152. {
153. printf("%c ",G->vers[k]);
154. visited[k] = true;
155. enQueue(&qu,k);
156. }
157. }
158. }
159. }
160. }
161. //广度优先遍历的主程序
162. void bfs(AdjMatrix \*G)
163. {
164. int k;
165. bool visited[MAX\_VERTEX\_NUM];
166. //获取状态数组并初始化
167. visited[k] = false;
168. //检查未被访问过的顶点
169. for(k = 0; k < G->vernum; k++)
170. {
171. if(!visited[k])
172. {
173. bfs(G, k, visited);
174. }
175. }
176. }

20.说明图的深度优先和广度优先遍历实现步骤（处理步骤，可用文字也可画流程图）。

**深度优先遍历：**



**广度优先遍历：**

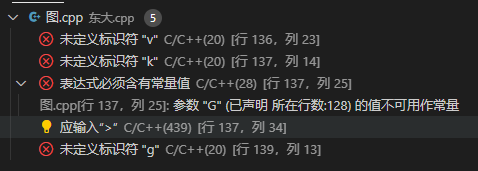


**三、实验环境**

Windows10、DEV-C++

**四、实验过程分析**

实验调试过程中的问题及改正的事件举例说明（只要调程序就不可能没有错误，现象、原因、修正方法，可以截图但不能过多）。



定义字符问题较多，都是细节问题，最后逐一排查。Debug完毕。

**五、实验结果总结**

1、阐述图的邻接矩阵存储结构上实现深度优先和广度优先遍历操作的算法时间复杂度和空间复杂度。

**深度优先遍历：**

　　　　数组表示：查找所有顶点的所有邻接点所需时间为O(n2)，n为顶点数，算法时间复杂度：O(n2) 、空间复杂度：O（n）。

**广度优先遍历：**

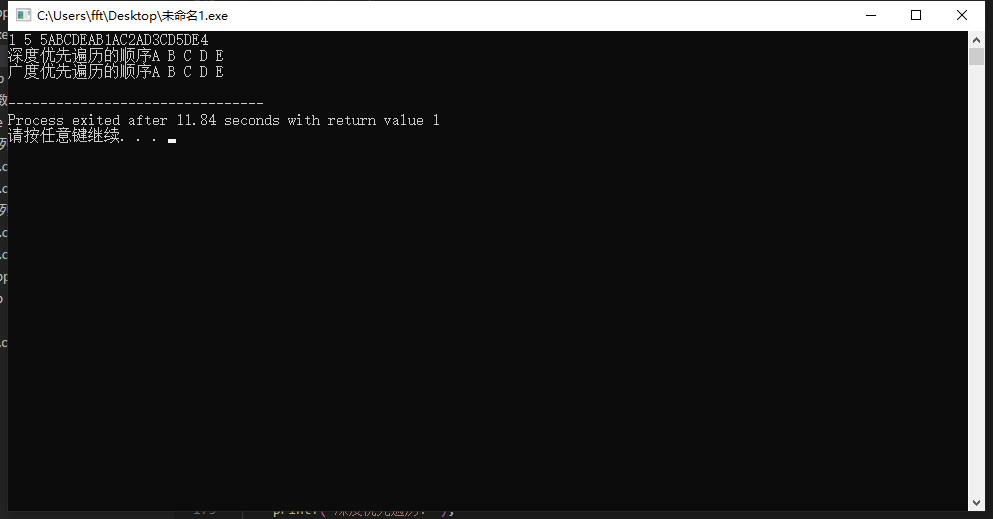
　　　　数组表示：查找每个顶点的邻接点所需时间为O(n2)，n为顶点数，算法的时间复杂度为O(n2) 、空间复杂度：O（n）。

2、给出你的测试方法和测试截图（不能超过三个截图）；

1 5 5ABCDEAB1AC2AD3CD5DE4

深度优先遍历的顺序A B C D E

广度优先遍历的顺序A B C D E



**六、附录**

1. 意见和建议（没有可不写）。

2. 思考题：

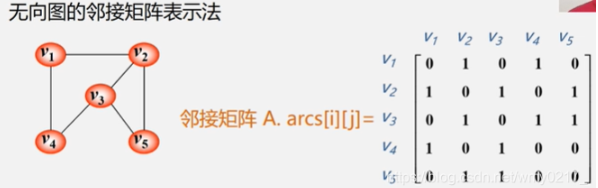
回答以下问题：

1. 图的邻接矩阵存储结构适合什么样的图进行存储？

有向图、无向图、网图

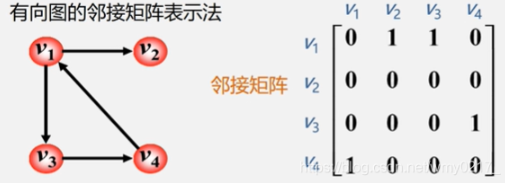
1. 无向图和有向图的邻接矩阵各有什么特点？

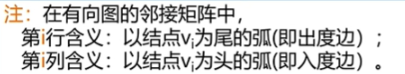
**无向图：**



分析1：无向图的邻接矩阵是对称的  
分析2：**顶点 i 的度=第 i 行(列)中 1 的个数**  
特别：完全图的邻接矩阵中，对角元素为0，其余为1

**有向图：**





分析1：有向图的邻接矩阵可能是不对称的  
分析2：  
**顶点的出度=第 i 行元素之和**  
**顶点的入度=第 i 列元素之和**  
**顶点的度=第 i 行元素之和+第 i 列元素之和**

**七、打分表**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 项目 | 考核点 | 分数 | **得分** | 备注 |
| 程序 | 逻辑是否正确  程序可读性  创新点 | 50 | **0** |  |
| 报告完整性 | 实验过程阐述是否完整  测试数据设计是否合理  运行结果是否正确 | 40 | **0** |  |
| 调试问题及解决方法 | 是否对调试过程问题进行阐述 | 5 | **0** |  |
| 思考题目 | 回答是否正确 | 5 | **0** |  |
| 合计 |  | 100 | **0** |  |