

# 《数据结构与算法》作业三

姓名：余弦 学号：2021112893 班级：2103401

## 1. 图的存储

### 邻接矩阵存储图

```
1 int mtx[N][N];
2 DATA data[N]; //用于储存节点的数据
3 int dim;
```

其中，mtx用于储存节点之间的连接情况，data数组用于储存每个节点储存的数据，dim用于储存节点的个数。

显然，对于一个有n个节点的图，其空间占用情况为 $O(n^3)$

### 邻接表存储图

```
1 //节点的定义
2 typedef struct node{
3     DATA elm;
4     int next[N];
5     int next_num = 0;
6 }NODE;
7 NODE* head[N];
8 int dim;
```

其中，head数组用于存储每个节点的地址，dim用于储存节点的个数。next用于存储指向下一个节点的索引。对于一个有n个节点和m条边的图，需要n个节点空间存储节点地址，m个int空间用来储存节点的指向关系。所以，其空间占用情况为 $O(nm)$

## 2. 图存储类型的转换

### 邻接矩阵转邻接表

```
1 graph_link_table* matrix_to_link(graph_matrix* matrix){
2     graph_link_table* link = new graph_link_table(matrix -> dim);
3     for(int i = 1; i <= matrix -> dim; i++){
4         link -> add_node(i, matrix -> data[i]);
5     }
6     for(int i = 1; i <= matrix -> dim; i++){
7         for(int j = 1; j <= matrix -> dim; j++){
8             if(matrix -> mtx[i][j] != 0){
9                 link -> add_edge(i, j);
10            }
11        }
12    }
13    return link;
14 }
```

### 邻接表转邻接矩阵

```

1 graph_matrix* link_to_matrix(graph_link_table* link){
2     graph_matrix* matrix = new graph_matrix(link -> dim);
3     for(int i = 1; i <= (link -> dim); i++){
4         matrix -> add_node(i, link -> head[i] -> elm);
5         for(int j = 1; j <= (link -> dim); j++){
6             if(link -> head[i] -> next[j] != 0){
7                 matrix -> add_edge(i, j);
8             }
9         }
10    }
11    return matrix;
12 }

```

### 3.深度优先遍历与广度优先遍历

#### 代码

```

1 void mtx_dfs(graph_matrix &g, int x, int num){
2     st[x] = true;
3     printf_s("V%d:%d ->", x, g.get_elm(x));
4     if(num == g.dim){
5         printf_s(".\n");
6         return;
7     }
8     for(int i = 1; i <= g.dim; i++){
9         if(g.mtx[x][i] > 0 && !st[i]){
10             mtx_dfs(g, i, num + 1);
11             break;
12         }
13     }
14     if(num < g.dim){
15         for(int i = 1; i <= g.dim; i++){
16             if(!st[i]){
17                 printf_s(".\n");
18                 mtx_dfs(g, i, num + 1);
19             }
20         }
21     }
22 }
23 void mtx_bfs(graph_matrix &g, int x, int num){
24     vector<int> q;
25     q.push_back(x);
26     st[x] = true;
27     while(!q.empty()){
28         int t = q[0];
29         q.erase(q.begin());
30         printf_s("V%d:%d ->", t, g.get_elm(t));
31         for(int i = 1; i <= g.dim; i++){
32             if(g.mtx[t][i] > 0 && !st[i]){
33                 q.push_back(i);
34                 num++;
35                 st[i] = true;
36             }
37         }
38     }

```

```

39     if(num == g.dim){
40         printf_s(".\n");
41         return;
42     }
43     for(int i = 1; i <= g.dim; i++){
44         if(!st[i]){
45             printf_s(".\n");
46             mtx_bfs(g, i, num + 1);
47         }
48     }
49 }
50
51 void link_dfs(graph_link_table &g, int x, int num){
52     st[x] = true;
53     printf_s("v%d:%d ->", x, g.head[x] -> elm);
54     if(num == g.dim){
55         printf_s(".\n");
56         return;
57     }
58     for(int i = 0; i < g.head[x] -> next_num; i++){
59         if(!st[g.head[x] -> next[i]]){
60             link_dfs(g, g.head[x] -> next[i], num + 1);
61             break;
62         }
63     }
64     if(num < g.dim){
65         for(int i = 1; i <= g.dim; i++){
66             if(!st[i]){
67                 printf_s(".\n");
68                 link_dfs(g, i, num + 1);
69             }
70         }
71     }
72 }
73 void link_bfs(graph_link_table &g, int x, int num){
74     vector<int> q;
75     q.push_back(x);
76     st[x] = true;
77     while(!q.empty()){
78         int t = q[0];
79         q.erase(q.begin());
80         printf_s("v%d:%d ->", t, g.head[t] -> elm);
81         for(int i = 0; i < g.head[t] -> next_num; i++){
82             if(!st[g.head[t] -> next[i]]){
83                 q.push_back(g.head[t] -> next[i]);
84                 num++;
85                 st[g.head[t] -> next[i]] = true;
86             }
87         }
88     }
89     if(num == g.dim){
90         printf_s(".\n");
91         return;
92     }
93     for(int i = 1; i <= g.dim; i++){

```

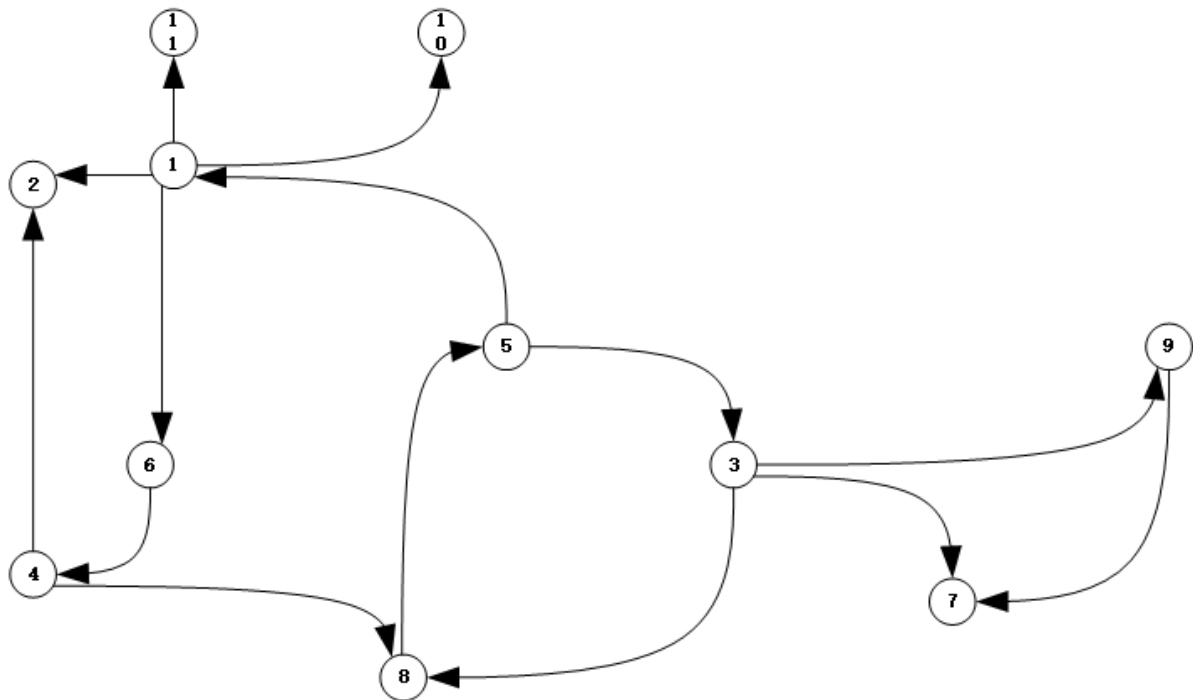
```

94         if(!st[i]){
95             printf_s("%.\\n");
96             link_bfs(g, i, num + 1);
97         }
98     }
99 }

```

## 运行结果

对于图



遍历结果为

```

DFS:
V1:1 ->V2:2 ->.
V3:3 ->V7:7 ->.
V4:4 ->V8:8 ->V5:5 ->.
V6:6 ->.
V9:9 ->.
V10:10 ->.
V11:11 ->.

BFS:
V1:1 ->V2:2 ->V6:6 ->V10:10 ->V11:11 ->V4:4 ->V8:8 ->V5:5 ->V3:3 ->V7:7 ->V9:9 ->.

DFS:
V1:1 ->V2:2 ->.
V3:3 ->V7:7 ->.
V4:4 ->V8:8 ->V5:5 ->.
V6:6 ->.
V9:9 ->.
V10:10 ->.
V11:11 ->.

BFS:
V1:1 ->V2:2 ->V6:6 ->V10:10 ->V11:11 ->V4:4 ->V8:8 ->V5:5 ->V3:3 ->V7:7 ->V9:9 ->.

```

## 复杂度分析

若采用邻接矩阵存储图：

对于深度优先遍历，对于每个节点，需要遍历每一个节点来判断是否邻接，故其时间复杂度为 $O(n^2)$

而对于广度优先遍历，对于每个节点都需要遍历所有节点，将相连的顶点加入到队列当中，其时间复杂度为 $O(n^2)$ ， $m$ 为图边的条数。

若采用邻接表存储图：

对于深度优先遍历，顶点直接的连接关系已有，故无需遍历，时间复杂度为 $O(n)$

对于广度优先遍历，需要将某个顶点所有的相邻接的顶点加入到队列当中，故其时间复杂度为 $O(m + n)$ ，其中 $m$ 为边的条数

## 4.出度入度的计算

代码

```
1  //邻接表
2  int in_degree(int num){
3      int cnt = 0;
4      for(int i = 1; i <= this -> dim; i++){
5          for(int j = 0; j < head[i] -> next_num; j++){
6              if(head[i] -> next[j] == num) cnt ++;
7          }
8      }
9      return cnt;
10 }
11 int out_degree(int num){
12     return head[num] -> next_num;
13 }
14
15 //邻接矩阵
16 int in_degree(int num){
17     int cnt = 0;
18     for(int i = 1; i <= this->dim; i++){
19         cnt += mtx[i][num];
20     }
21     return cnt;
22 }
23
24 int out_degree(int num){
25     int cnt = 0;
26     for(int i = 1; i <= this -> dim; i++){
27         cnt += mtx[num][i];
28     }
29     return cnt;
30 }
```

测试结果

```
V1:1 -> in_degree:1, out_degree:4
V2:2 -> in_degree:2, out_degree:0
V3:3 -> in_degree:1, out_degree:3
V4:4 -> in_degree:1, out_degree:2
V5:5 -> in_degree:1, out_degree:2
V6:6 -> in_degree:1, out_degree:1
V7:7 -> in_degree:2, out_degree:0
V8:8 -> in_degree:2, out_degree:1
V9:9 -> in_degree:1, out_degree:1
V10:10 -> in_degree:1, out_degree:0
V11:11 -> in_degree:1, out_degree:0
```

```
V1:1 -> in_degree:1, out_degree:4
V2:2 -> in_degree:2, out_degree:0
V3:3 -> in_degree:1, out_degree:3
V4:4 -> in_degree:1, out_degree:2
V5:5 -> in_degree:1, out_degree:2
V6:6 -> in_degree:1, out_degree:1
V7:7 -> in_degree:2, out_degree:0
V8:8 -> in_degree:2, out_degree:1
V9:9 -> in_degree:1, out_degree:1
V10:10 -> in_degree:1, out_degree:0
V11:11 -> in_degree:1, out_degree:0
```

## 复杂度分析

### 邻接矩阵

不管是计算入度还是出度，只需要把每一行或者每一列的值相加即可，故时间复杂度为 $O(n)$

### 对于邻接表

计算入度的时候，需要扫描其他点的邻接矩阵，看是否有边进入，故时间复杂度为 $O(n + m)$

计算出度是，只需要直接读取next数组中元素的个数即可，故时间复杂度为 $O(1)$

## 5.点和边的输入方式

```
input the number of nodes:11
input the number of edges:
14
1 2
1 11
1 10
1 6
3 8
3 7
3 9
4 2
4 8
5 1
5 3
6 4
8 5
9 7
```

附录：

输入

1	11
2	14
3	1 2
4	1 11
5	1 10
6	1 6
7	3 8
8	3 7
9	3 9
10	4 2
11	4 8
12	5 1
13	5 3
14	6 4
15	8 5
16	9 7

完整源代码

```

1  #include <iostream>
2  #include "vector"
3  #include <cstring>
4  #define DATA int
5  const int N = 100;
6  using namespace std;
7  typedef struct node{
8      DATA elm;
9      int next[N];
10     int next_num = 0;
11 }NODE;
12 bool st[N];
13 class graph_matrix{
14 public:
15     int mtx[N][N];
16     DATA data[N]; //用于储存节点的数据
17     int dim;
18
19     graph_matrix(int n){
20         memset(this->mtx, -1, sizeof(this->mtx));
21         memset(this->data, 0, sizeof(this->data));
22         this->dim = n;
23         for(int i = 1; i <= n; i++){
24             for(int j = 1; j <= n; j++){
25                 this->mtx[i][j] = 0;
26             }
27         }
28     }
29
30     void add_node(int num, DATA elm){
31         if(!is_legal(num)){
32             printf_s("ERROR!\n");
33             return;
34         }
35         this->data[num] = elm;
36     }
37
38
39     bool is_legal(int num){
40         return num > 0 && num <= this->dim;
41     }
42
43     void change_node_val(int num, DATA elm){
44         if(!is_legal(num)){
45             printf_s("ERROR!\n");
46             return;
47         }
48         data[num] = elm;
49     }
50
51     void add_edge(int x, int y){
52         if(!(is_legal(x) && is_legal(y))){
53             printf_s("ERROR!\n");
54         }
55         this->mtx[x][y] ++;

```



```

56     }
57
58     DATA get_elm(int num){
59         if(!is_legal(num)){
60             printf_s("ERROR!\n");
61             return -1;
62         }
63         return this -> data[num];
64     }
65
66     int in_degree(int num){
67         int cnt = 0;
68         for(int i = 1; i <= this->dim; i++){
69             cnt += mtx[i][num];
70         }
71         return cnt;
72     }
73
74     int out_degree(int num){
75         int cnt = 0;
76         for(int i = 1; i <= this -> dim; i++){
77             cnt += mtx[num][i];
78         }
79         return cnt;
80     }
81 };
82 void mtx_dfs(graph_matrix &g, int x, int num){
83     st[x] = true;
84     printf_s("V%d:%d ->", x, g.get_elm(x));
85     if(num == g.dim){
86         printf_s(".\n");
87         return;
88     }
89     for(int i = 1; i <= g.dim; i++){
90         if(g.mtx[x][i] > 0 && !st[i]){
91             mtx_dfs(g, i, num + 1);
92             break;
93         }
94     }
95     if(num < g.dim){
96         for(int i = 1; i <= g.dim; i++){
97             if(!st[i]){
98                 printf_s(".\n");
99                 mtx_dfs(g, i, num + 1);
100             }
101         }
102     }
103 }
104 void mtx_bfs(graph_matrix &g, int x, int num){
105     vector<int> q;
106     q.push_back(x);
107     st[x] = true;
108     while(!q.empty()){
109         int t = q[0];
110         q.erase(q.begin());

```

```

111         printf_s("v%d:%d ->", t, g.get_elm(t));
112         for(int i = 1; i <= g.dim; i++){
113             if(g.mtx[t][i] > 0 && !st[i]){
114                 q.push_back(i);
115                 num++;
116                 st[i] = true;
117             }
118         }
119     }
120     if(num == g.dim){
121         printf_s(".\n");
122         return;
123     }
124     for(int i = 1; i <= g.dim; i++){
125         if(!st[i]){
126             printf_s(".\n");
127             mtx_bfs(g, i, num + 1);
128         }
129     }
130 }
131
132 class graph_link_table{
133 public:
134     //利用邻接表实现图
135     NODE* head[N];
136     int dim;
137     graph_link_table(int n){
138         this -> dim = n;
139         for(int i = 1; i <= n; i++){
140             this -> head[i] = nullptr;
141         }
142     }
143     bool if_legal(int num){
144         return (num > 0 && num <= this ->dim);
145     }
146     bool is_node(int num){
147         if(!head[num]){
148             printf_s("ERROR:The node has not been created!\n");
149             return 0;
150         }
151         return 1;
152     }
153     void add_node(int num, DATA data){
154         if(!if_legal(num)){
155             printf_s("ERROR!\n");
156         }
157         if(!head[num]){
158             head[num] = (NODE*) malloc(sizeof(NODE));
159             head[num] -> elm = data;
160             head[num] -> next_num = 0;
161         }
162         else{
163             printf_s("ERROR:Node has existed!\n");
164         }
165     }

```

```

166 void change_node_val(int num, DATA data){
167     if(!if_legal(num)){
168         printf_s("ERROR!\n");
169         return;
170     }
171     if(!head[num]){
172         printf_s("ERROR:The node has not been created!\n");
173         return;
174     }
175     head[num] -> elm = data;
176 }
177 void add_edge(int x, int y){
178     if(is_node(x) and is_node(y)){
179         head[x] -> next[head[x] -> next_num++] = y;
180     }
181 }
182
183 void delete_edge(int x, int y){
184     if(is_node(x) and is_node(y)){
185         if(head[x] -> next[y] > 0) head[x] -> next[y] --;
186         else printf_s("ERROR:There is no edge between %d and %d!\n", x,
y);
187     }
188 }
189
190 int in_degree(int num){
191     int cnt = 0;
192     for(int i = 1; i <= this -> dim; i++){
193         for(int j = 0; j < head[i] -> next_num; j++){
194             if(head[i] -> next[j] == num) cnt ++;
195         }
196     }
197     return cnt;
198 }
199
200 int out_degree(int num){
201     return head[num] -> next_num;
202 }
203
204 };
205 void link_dfs(graph_link_table &g, int x, int num){
206     st[x] = true;
207     printf_s("V%d:%d ->", x, g.head[x] -> elm);
208     if(num == g.dim){
209         printf_s(".\n");
210         return;
211     }
212     for(int i = 0; i < g.head[x] -> next_num; i++){
213         if(!st[g.head[x] -> next[i]]){
214             link_dfs(g, g.head[x] -> next[i], num + 1);
215             break;
216         }
217     }
218     if(num < g.dim){
219         for(int i = 1; i <= g.dim; i++){

```

```

220         if(!st[i]){
221             printf_s(".\n");
222             link_dfs(g, i, num + 1);
223         }
224     }
225 }
226 }
227 void link_bfs(graph_link_table &g, int x, int num){
228     vector<int> q;
229     q.push_back(x);
230     st[x] = true;
231     while(!q.empty()){
232         int t = q[0];
233         q.erase(q.begin());
234         printf_s("v%d:%d ->", t, g.head[t] -> elm);
235         for(int i = 0; i < g.head[t] -> next_num; i++){
236             if(!st[g.head[t] -> next[i]]){
237                 q.push_back(g.head[t] -> next[i]);
238                 num++;
239                 st[g.head[t] -> next[i]] = true;
240             }
241         }
242     }
243     if(num == g.dim){
244         printf_s(".\n");
245         return;
246     }
247     for(int i = 1; i <= g.dim; i++){
248         if(!st[i]){
249             printf_s(".\n");
250             link_bfs(g, i, num + 1);
251         }
252     }
253 }
254 //将图的邻接表转换为邻接矩阵
255 graph_matrix* link_to_matrix(graph_link_table* link){
256     graph_matrix* matrix = new graph_matrix(link -> dim);
257     for(int i = 1; i <= (link -> dim); i++){
258         matrix -> add_node(i, link -> head[i] -> elm);
259         for(int j = 1; j <= (link -> dim); j++){
260             if(link -> head[i] -> next[j] != 0){
261                 matrix -> add_edge(i, j);
262             }
263         }
264     }
265     return matrix;
266 }
267 //将图的邻接矩阵转换为邻接表
268 graph_link_table* matrix_to_link(graph_matrix* matrix){
269     graph_link_table* link = new graph_link_table(matrix -> dim);
270     for(int i = 1; i <= matrix -> dim; i++){
271         link -> add_node(i, matrix -> data[i]);
272     }
273     for(int i = 1; i <= matrix -> dim; i++){
274         for(int j = 1; j <= matrix -> dim; j++){

```

```

275         if(matrix -> mtx[i][j] != 0){
276             link -> add_edge(i, j);
277         }
278     }
279 }
280 return link;
281 }
282
283 void set_false(bool st[]){
284     for(int i = 1; i <= N; i++){
285         st[i] = false;
286     }
287 }
288 //将边保存到vector中
289 int main() {
290     cout<<"input the number of nodes:";
291     int nn;
292     cin>>nn;
293     graph_matrix G(nn);
294     /**添加点和边**//
295     //添加点
296     for(int i = 1; i <= nn; i++){
297         G.add_node(i, i);
298     }
299     //添加边
300     int tmp;
301     cout<<"input the number of edges:"<<endl;
302     cin>>tmp;
303     for(int i = 0; i < tmp; i++){
304         int x, y;
305         cin>>x>>y;
306         G.add_edge(x, y);
307     }
308     graph_link_table* g = matrix_to_link(&G);
309     set_false(st);
310     printf_s("DFS:\n");
311     mtx_dfs(G, 1, 1);
312     printf_s("\n");
313     set_false(st);
314     printf_s("BFS:\n");
315     mtx_bfs(G, 1, 1);
316     printf_s("\n");
317     set_false(st);
318     printf_s("DFS:\n");
319     link_dfs(*g, 1, 1);
320     printf_s("\n");
321     set_false(st);
322     printf_s("BFS:\n");
323     link_bfs(*g, 1, 1);
324
325     //每个节点的入度和出度
326     for(int i = 1; i <= G.dim; i++){
327         printf_s("v%d:%d -> in_degree:%d, out_degree:%d\n", i, G.data[i],
328             G.in_degree(i), G.out_degree(i));

```

```
329     puts("");
330     for(int i = 1; i <= g -> dim; i++){
331         printf_s("V%d:%d -> in_degree:%d, out_degree:%d\n", i, g -> head[i]
332     -> elm, g -> in_degree(i), g -> out_degree(i));
333     }
334     return 0;
335 }
```