数据结构作业第七章——图

庄震丰 22920182204393

Nov. 17^{th}, 2019

7-15

题目要求：试在邻接矩阵存储结构上实现图的基本操作：InsertVex(G,v),InsertArc(G,v,w),DeleteVex(G,v) 和 DeleteArc(G,v,w)

算法分析：插入节点直接在原矩阵的基础上记录出入度信息可以，并且在对角线上标记该点是否存在。插入边 v,w, 则直接令 a(i,j) 为边权 value 即可。删除时相反操作就可以。

空间复杂度 $O(n^2)$, 时间复杂度 $O(m\Delta)$，m 是操作数，Δ 为最大度。

7-15.cpp

```cpp
#include<bits/stdc++.h>
using namespace std;
#define maxn 1000
#define inf 0x3f3f3f3f
struct Graph
{
    int MAP[maxn][maxn];
    int num;
}G;
void InsertVex(Graph &G,int v)
{
    int n;
    G.num++;
    int x,value;
    cout<<"input the indegree number:"<<endl;
    cin>>n;
    cout<<"input the point and value"<<endl;
    for (int i=1;i<=n;i++)
        {
            cin>>x>>value;
            G.MAP[v][x]=value;
        }
    cout<<"output the indegree number:"<<endl;
    cin>>n;
    cout<<"input the point and value"<<endl;
    for (int i=1;i<=n;i++)
        {
            cin>>x>>value;
            G.MAP[x][v]=value;
        }
}
void InsertArc(Graph &G,int v,int w)
{
    cout<<"please input the value"<<endl;
    int value;
    cin>>value;
    G.MAP[v][w]=value;
}
```

```
39      void DeleteVex(Graph &G,int v)
40      {
41          G.num--;
42          for (int i=0;i<maxn;i++)
43          {
44              G.MAP[i][v]=inf;
45              G.MAP[v][i]=inf;
46          }
47      }
48      void DeleteArc(Graph &G,int v,int w)
49      {
50          G.MAP[v][w]=inf;
51      }
52      void init()
53      {
54          for (int i=0;i<maxn;i++)
55              for (int j=0;j<maxn;j++)
56                  G.MAP[i][j]=inf;
57      }
58      void print()
59      {
60          for (int i=1;i<=G.num;i++)
61              {
62              for (int j=1;j<=G.num;j++)
63                  cout<<G.MAP[i][j]<<" ";
64                  cout<<endl;
65              }
66      }
67      void operat()
68      {
69          int type,p,q;
70          cin>>type;
71          while(type!=-1)
72          {
73              if(type==1)
74              {
75                  cout<<"Insert point:"<<endl;
76                  cin>>p;
77                  InsertVex(G,p);
78              }
79              if(type==2)
80              {
81                  cout<<"Insert Arc:"<<endl;
82                  cin>>p>>q;
83                  InsertArc(G,p,q);
84              }
85              if(type==3)
86              {
87                  cout<<"Delete point:"<<endl;
88                  cin>>p;
89                  DeleteVex(G,p);
90              }
```

```
            if(type==4)
            {
                cout<<"Delete Arc:"<<endl;
                cin>>p>>q;
                DeleteArc(G,p,q);
            }
            if (type==5)
            {
                cout<<"Matrix is:"<<endl;
                print();
            }
            cin>>type;
        }
    }
    int main()
    {
        init();
        operat();
        print();
        return 0;
    }
```

7-16

题目要求：利用邻接表实现 7-15 题的操作。

算法分析：邻接表重新构造图的结构体，每个点建立 vector 储存每个点的出度点。

插入操作时，每得到一个后继点就插入当前点的 vector，每得到一个入度点，就将当前点插入到每个前驱点的 vector。插入边的操作就是直接将 w 插入到 vector[v] 中。

值得注意的是，当进行删除操作点操作时，除了删除当前点出度点之外，也要删除所有其他点出度为当前点的边。空间复杂度 $O(E)$，插入边为 O(1), 插入点最多为 O(N)，设操作数为 M，则时间复杂度 O(MN).

7-16.cpp

```
#include<bits/stdc++.h>
using namespace std;
#define maxn 1000
#define inf 0x3f3f3f3f
struct Graph
{
    vector<int> Link[maxn];
    int num;
}G;
void InsertVex(Graph &G,int v)
{
    int n;
    G.num++;
    int x;
    cout<<"input the indegree number:"<<endl;
    cin>>n;
    for (int i=1;i<=n;i++)
        {
            cin>>x;
```

```
20                  G.Link[x].push_back(v);
21              }
22          cout<<"output the indegree number:"<<endl;
23          cin>>n;
24          for (int i=1;i<=n;i++)
25              {
26                  cin>>x;
27                  G.Link[v].push_back(x);
28              }
29      }
30      void InsertArc(Graph &G,int v,int w)
31      {
32          G.Link[v].push_back(w);
33      }
34      void DeleteVex(Graph &G,int v)
35      {
36
37          while(!G.Link[v].empty())
38          {
39              G.Link[v].pop_back();
40          }
41          for (int i=1;i<=G.num;i++)
42          {
43              vector<int>::iterator it;
44              it=find(G.Link[i].begin(),G.Link[i].end(),v);
45              G.Link[i].erase(it);
46          }
47      G.num--;
48      }
49      void DeleteArc(Graph &G,int v,int w)
50      {
51          vector<int>::iterator it;
52        v  it=find(G.Link[v].begin(),G.Link[v].end(),w);
53          G.Link[v].erase(it);
54      }
55      void operat()
56      {
57          int type,p,q;
58          cin>>type;
59          while(type!=-1)
60          {
61              if(type==1)
62              {
63                  cout<<"Insert point:"<<endl;
64                  cin>>p;
65                  InsertVex(G,p);
66              }
67              if(type==2)
68              {
69                  cout<<"Insert Arc:"<<endl;
70                  cin>>p>>q;
71                  InsertArc(G,p,q);
```

```
72              }
73              if(type==3)
74              {
75                  cout<<"Delete point:"<<endl;
76                  cin>>p;
77                  DeleteVex(G,p);
78              }
79              if(type==4)
80              {
81                  cout<<"Delete Arc:"<<endl;
82                  cin>>p>>q;
83                  DeleteArc(G,p,q);
84              }
85              cin>>type;
86          }
87      }
88      int main()
89      {
90          operat();
91          return 0;
92      }
```

7-25

题目要求：假设对有向图中 n 个顶点进行自然编号，并以三个数组 $s[1\cdots max], fst[1\cdots n], lst[1\cdots n]$ 表示之。其中数组 s 存放每个顶点后继顶点的信息，第 i 个顶点后继存放在 s 中下标从 fst[i],lst[i] 的分量中 (i=1,2,...,n)。若 fst[i]>lst[i], 则第 i 个顶点无后继顶点。试编写判别该有向图中是否存在回路的算法。

算法分析：首先进行预处理，得到所有节点的入度：对 s 进行扫描。将 fst[i]...lst[i] 中点入度 +1。

得到所有点的入度后，进行以下操作：

- 找到没有入度为零的节点，删除之，将其后继节点的入读全部减去 1，同时将图中的点计数器 -1。
- 当找不到入度为 0 的点的时候，退出，否则重复以上操作。
- 退出后如果还有一个点，判断它有没有自环，如果没有自环就删除该点，计数器 -1。
- 当点计数器大于零，则存在回路，若为 0 则不存在回路。

以上为拓扑思路。

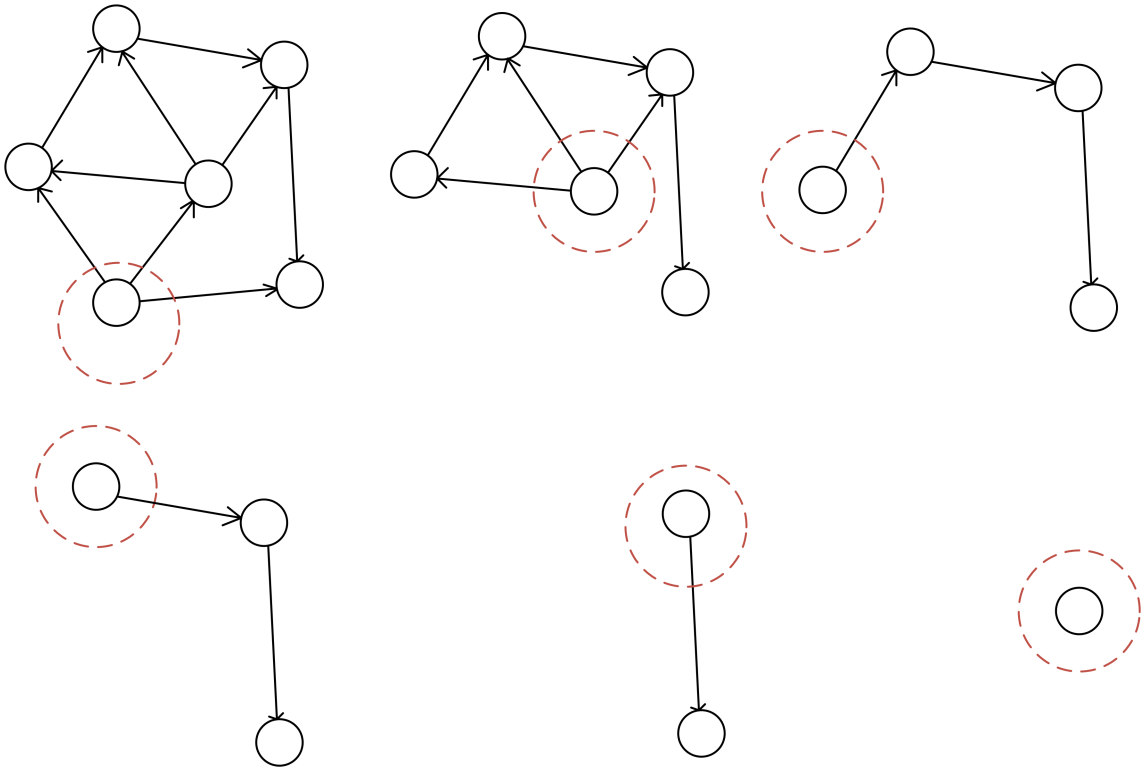
空间复杂度：$O(n)$, 时间复杂度 $O(E)$

7-25.cpp

```
1       #include<bits/stdc++.h>
2       using namespace std;
3       #define maxn 1000
4       #define inf 0x3f3f3f3f
5       int s[maxn];
6       int pre[maxn]={0};
7       int fst[maxn],lst[maxn];
8       int n;
9       void init()
10      {
11          for (int i=0;i<maxn;i++)
```

```
12          {
13              fst[i]=-1;lst[i]=-1;
14          }
15      }
16      bool fin()
17      {
18          for (int i=1;i<=n;i++)
19          {
20              if (pre[i]==0) return i;
21          }
22          return -1;
23      }
24      int main()
25      {
26          cin>>n;
27          for (int i=1;i<=n;i++)
28              cin>>s[i];
29          for (int i=1;i<=n;i++)
30              cin>>fst[i];
31          for (int i=1;i<=n;i++)
32              cin>>lst[i];
33          for (int i=1;i<=n;i++)
34              for (int j=fst[i];j<=lst[i];j++)
35                  pre[j]++;
36          int N=n;
37          int k=fin();
38          while(k)
39          {
40              N--;
41              for (int j=fst[k];j<=lst[k];j++)
42                  pre[j]--;//拓扑删点
43              k=fin();
44          }
45          if (N>0) cout<<"YES"; else cout<<"NO";
46          return 0;
47      }
```

Sample



7-27

题目要求：采用邻接表结构，编写一个判别无向图中任意两点是否一条长为 k 的简单路径的算法。

算法分析：DFS。从起点开始深搜，每当搜到无出度的点或者长度（搜索深度）大于 K，则 return, 否则沿着后继点一直进行，找到在全局变量标记。

算法复杂：空间复杂度 $O(E)$，时间复杂度 $O(E)$

7-27.cpp

```
#include<bits/stdc++.h>
using namespace std;
#define maxn 1000
#define inf 0x3f3f3f3f
int n,k,v,w;
int flag=false;
struct Graph
{
    vector<int> Link[maxn];
    int num;
}G;
bool vis[maxn];
void dfs(int x,int deep)
{
    if(x==w&&deep==k)
     {
        flag=true;
        return;
     }
     for (vector<int>::iterator it=G.Link[x].begin();it!=G.Link[x].end();it++)
        {
            vis[*it]=true;
```

```
                dfs(*it,deep+1);
                vis[*it]=false;
            }
        return;
    }
    void init()
    {
        int m,p;
        cin>>n;
        for (int i=1;i<n;i++)
        {
            cin>>m;
            for (int j=1;j<=m;j++)
            {
                cin>>p;
                G.Link[i].push_back(j);
            }
        }
        memset(vis,sizeof(vis),false);
    }
    int main()
    {
        cin>>v,w,k;
        init();
        dfs(v,1);
        if (flag) cout<<"Exist";else cout<<"Not Exist";
    }
```

7-33 题目要求：已知一无向图边集保存在某个类型 EdgeSetType 的数据结构 EdgeSet 中，并在此结构上已定义两种基本运算：

- 函数 GetMinEdge(EdgeSet,u,v)：若 EdgeSet 非空，则必存在最小边，变参 u 和 v 的最小边上两个顶点，并返回 true，否则返回 false
- 过程 DelMinEdge(EdgeSet,u,v)：从 EdgeSet 中删除依附于顶点 u 和 v 的最小边。

算法分析：先建立 EdgeSet 结构体，读入初始信息，用数组建立边，GetMinEdge 函数就利用其进行遍历，记录下最小边的序号，并判断两点是否为 u 和 v。
Del 函数也是进行遍历，若 E<u，v> 存在，则删除，可以直接用结构体中的 bool 变量进行标记。
空间复杂度 O(E), 两个函数的时间复杂度 O(E)
7-36.c

```

    #include <stdio.h>

    /* 类型定义 */
    typedef int MFSet;                  //并查集
    typedef struct                      //边的集合
    {
        int u, v;                     //端点
        int Weight;                     //权值
    }EdgeSetType;
```

```
11      typedef struct                          //不带权值的边的集合
12      {
13          char u;
14          char v;                             //端点
15      }Edge;
16
17      /* 函数原型 */
18      CSTree MinSpanTree_KRUSKAL_7_33(ALGraph G);
19      void InitEdgeSet_7_33(EdgeSetType EdgeSet[], ALGraph G);        //初始化边集
20      Status GetMinEdge_7_33(EdgeSetType EdgeSet[], int *u, int *v);    //获取最小边<u, v>
21      void DelMinEdge_7_33(EdgeSetType EdgeSet[], int u, int v);      //删除边<u, v>
22      void InitMFSet_7_33(MFSet S[], ALGraph G);                      //初始化并查集
23      int FindSeat_7_33(MFSet S[], int u);                           //返回顶点所在集合u
24      void Merge_7_33(MFSet S[], int u, int v);                       //将集合并入集合uv
25      CSTree CreateCSTree_7_33(Edge E[]);                             //根据边集创建树
26      void AddEdgeToTree_7_33(CSTree *T, char v, char w);             //添加边<v,w到树中，>为连接点v
27
28      int main(int argc, char *argv[])
29      {
30          ALGraph G;
31          FILE *fp;                                   //作为输入源
32          CSTree T;
33
34          printf("创建并输出无向图（带权）...\n");
35          fp = fopen("Data/Algo_7_33.txt", "r");
36          CreateGraph_AL(fp, &G);
37          fclose(fp);
38          OutputALGraph(G);
39          printf("\n");
40
41          T= MinSpanTree_KRUSKAL_7_33(G);
42          printf("此无向连通网的最小生成树为：\n");
43          Print_CS(T);
44          printf("\n");
45      }
46
47      CSTree MinSpanTree_KRUSKAL_7_33(ALGraph G)          //假设图的权值均大于0
48      {
49          MFSet S[G.vexnum+1];                        //并查集
50          EdgeSetType EdgeSet[G.arcnum+1];              //原始边集
51          Edge E[G.vexnum];                           //筛选后的最小边集
52          int u, v;
53          int k;
54
55          InitEdgeSet_7_33(EdgeSet, G);
56          InitMFSet_7_33(S, G);
57
58          k = 1;
59          E[0].u = E[0].v = G.vexnum-1;
60
61          while(k<=G.vexnum-1)
62          {
```

```
63              if(GetMinEdge_7_33(EdgeSet, &u, &v))                //找到最小边
64              {
65                  if(FindSeat_7_33(S, u)!=FindSeat_7_33(S, v))    //判断是否构成回路
66                  {
67                      E[k].u = G.vertices[u].data;
68                      E[k].v = G.vertices[v].data;
69                      Merge_7_33(S, u, v);
70                      k++;
71                  }
72
73                  DelMinEdge_7_33(EdgeSet, u, v);
74              }
75          }
76
77          return CreateCSTree_7_33(E);
78      }
79
80      void InitEdgeSet_7_33(EdgeSetType EdgeSet[], ALGraph G)
81      {
82          int k, count;
83          ArcNode *r;
84
85          EdgeSet[0].Weight = G.arcnum;
86
87          for(k=1,count=0; k<=G.vexnum; k++)
88          {
89              r = G.vertices[k].firstarc;
90              while(r && r->adjvex<k)
91                  r = r->nextarc;
92              while(r)
93              {
94                  count++;
95                  EdgeSet[count].u = k;
96                  EdgeSet[count].v = r->adjvex;
97                  EdgeSet[count].Weight = r->info.in;
98                  r = r->nextarc;
99              }
100         }
101     }
102
103     Status GetMinEdge_7_33(EdgeSetType EdgeSet[], int *u, int *v)
104     {
105         int k;
106         int min = INT_MAX;
107
108         for(k=1; k<=EdgeSet[0].Weight; k++)
109         {
110             if(EdgeSet[k].Weight<min)
111             {
112                 min = EdgeSet[k].Weight;
113                 *u = EdgeSet[k].u;
114                 *v = EdgeSet[k].v;
```

```
115             }
116         }
117 
118         if(min==INT_MAX)
119             return ERROR;
120         else
121             return OK;
122     }
123 
124     void DelMinEdge_7_33(EdgeSetType EdgeSet[], int u, int v)
125     {
126         int k;
127 
128         for(k=1; k<=EdgeSet[0].Weight; k++)
129         {
130             if((EdgeSet[k].u==u&&EdgeSet[k].v==v) || (EdgeSet[k].u==v&&EdgeSet[k].v==u))
131             {
132                 while(k+1<=EdgeSet[0].Weight)
133                 {
134                     EdgeSet[k] = EdgeSet[k+1];
135                     k++;
136                 }
137 
138                 break;
139             }
140         }
141 
142         EdgeSet[0].Weight--;
143     }
144 
145     void InitMFSet_7_33(MFSet S[], ALGraph G)
146     {
147         int k;
148 
149         S[0] = G.vexnum;
150 
151         for(k=1; k<=G.vexnum; k++)
152             S[k] = -1;
153     }
154 
155     int FindSeat_7_33(MFSet S[], int u)
156     {
157         int k;
158 
159         for(k=u; S[k]>0; k=S[k])
160             ;
161 
162         return k;
163     }
164 
165     void Merge_7_33(MFSet S[], int u, int v) //集合合并到集合uv
166     {
```

```
167         int k;
168
169         S[u] = v;
170
171         for(k=v; S[k]>0; k=S[k])
172             ;
173
174         S[k]--;
175     }
176
177     CSTree CreateCSTree_7_33(Edge E[])
178     {
179         CSTree T;
180         char Stack[E[0].u+1];              //模拟栈
181         int i, j, k;
182         char tmp;
183
184         InitTree_CS(&T);                   //初始化孩子兄弟树-
185         k = -1;
186
187         if(E[0].u)                         //边集不为空
188         {
189             Stack[++k] = E[1].v;
190             Stack[++k] = E[1].u;
191             AddEdgeToTree_7_33(&T, E[1].u, E[1].v);
192
193             while(k>=0)
194             {
195                 tmp = Stack[k--];
196                 for(i=2; i<=E[0].u; i++)
197                 {
198                     if(E[i].u==tmp || E[i].v==tmp)
199                     {
200                         if(E[i].u==tmp)
201                         {
202                             AddEdgeToTree_7_33(&T, E[i].u, E[i].v);
203                             Stack[++k] = E[i].v;
204                         }
205                         else
206                         {
207                             Stack[++k] = E[i].u;
208                             AddEdgeToTree_7_33(&T, E[i].v, E[i].u);
209                         }
210
211                         E[i].u = E[i].v = '\0';     //相当于删掉此条已访问过的边
212                     }
213                 }
214             }
215         }
216
217         return T;
218     }
```

```
219
220     void AddEdgeToTree_7_33(CSTree *T, char v, char w)
221     {
222         CSTree p, q, r;
223
224         r = (CSTree)malloc(sizeof(CSNode));
225         r->data = w;
226         r->firstchild = r->nextsibling = NULL;
227
228         if(!(*T))
229         {
230             *T = (CSTree)malloc(sizeof(CSNode));
231             (*T)->data = v;
232             (*T)->firstchild = r;
233             (*T)->nextsibling = NULL;
234         }
235         else
236         {
237             p = Order_CS(*T, v);
238             q = p->firstchild;
239
240             if(!q)
241                 p->firstchild = r;
242             else
243             {
244                 while(q && q->nextsibling)
245                     q = q->nextsibling;
246                 q->nextsibling = r;
247             }
248         }
249     }
```

7-36

题目要求：在图的邻接表存储结构中，为每个定点增加一个 MPL 域，试写一算法，求有向无环图 G 的每个顶点出发的最长路径的长度，并把它存进 MPL 域，给出算法复杂度。

算法分析：FLOYD. 首先将邻接存储结构化为邻接矩阵，减少访问边权时间。利用 FLOYD 求出所有点的最长路。再对每个点进行遍历，得到最长 distance，将其 push 进 MPL 域。

空间复杂度 $O(E+N^2)$，时间复杂度 $O(N^3)$

7-36.cpp

```
1     #include<bits/stdc++.h>
2     #define MAX 1000000000
3     using namespace std;
4     int d[1000][1000],path[1000][1000];//FLOYD
5     vector<int> MPL;
6     int main()
7     {
8         int i,j,k,m,n;
9         int x,y,z;
10        scanf("%d%d",&n,&m);
```

```
11          for(i=1;i<=n;i++)
12              for(j=1;j<=n;j++)
13              {
14                  d[i][j]=MAX;
15                  path[i][j]=j;
16              }
17          for(i=1;i<=m;i++)
18          {
19                  scanf("%d%d%d",&x,&y,&z);
20                  d[x][y]=z;
21                  d[y][x]=z;
22          }
23          for(k=1;k<=n;k++)
24              for(i=1;i<=n;i++)
25                  for(j=1;j<=n;j++) {
26                      if(d[i][k]+d[k][j]<d[i][j]) {
27                          d[i][j]=d[i][k]+d[k][j];
28                          path[i][j]=path[i][k];
29                      }
30                  }
31          for(i=1;i<=n;i++)
32              for(j=i+1;j<=n;j++)
33                if (i!=j) printf("%d->%d:%d\n",i,j,d[i][j]);
34          int f, en;
35          scanf("%d%d",&f,&en);
36          while (f!=en)
37          {
38              printf("%d->",f);
39              f=path[f][en];
40          }
41          printf("%d\n",en);
42          for (int i=1;i<=n;i++)
43          {
44              int t=-MAX, fl=1;
45              for (int j=1;j<=n;j++)
46              {
47                  if (d[i][j]!=MAX)
48                  {
49                      if (d[i][j]>t)
50                      {
51                          t=d[i][j];
52                          fl=j;
53                      }
54                  }
55              }
56              MPL.push_back(fl);
57          }
58          return 0;
59      }
```