阅读：20,166　　作者：解学武

# 图的顺序存储结构及C语言实现

上一节　　下一节

使用图结构表示的数据元素之间虽然具有“多对多”的关系，但是同样可以采用顺序存储，也就是使用数组有效地存储图。

使用数组存储图时，需要使用两个数组，一个数组存放图中顶点本身的数据（一维数组），另外一个数组用于存储各顶点之间的关系（二维数组）。

> 存储图中各顶点本身数据，使用一维数组就足够了；存储顶点之间的关系时，要记录每个顶点和其它所有顶点之间的关系，所以需要使用二维数组。

不同类型的图，存储的方式略有不同，根据图有无权，可以将图划分为两大类：图和网 。

> 图，包括无向图和有向图；网，是指带权的图，包括无向网和有向网。存储方式的不同，指的是：在使用二维数组存储图中顶点之间的关系时，如果顶点之间存在边或弧，在相应位置用 1 表示，反之用 0 表示；如果使用二维数组存储网中顶点之间的关系，顶点之间如果有边或者弧的存在，在数组的相应位置存储其权值；反之用 0 表示。

结构代码表示：

```
#define MAX_VERtEX_NUM 20                       //顶点的最大个数
#define VRType int                              //表示顶点之间的关系的变量类型
#define InfoType char                           //存储弧或者边额外信息的指针变量类型
#define VertexType int                          //图中顶点的数据类型
typedef enum{DG,DN,UDG,UDN}GraphKind;           //枚举图的 4 种类型
typedef struct {
    VRType adj;                                 //对于无权图，用 1 或 0 表示是否相邻；对于带权图，直
    InfoType * info;                            //弧或边额外含有的信息指针
}ArcCell,AdjMatrix[MAX_VERtEX_NUM][MAX_VERtEX_NUM];

typedef struct {
    VertexType vexs[MAX_VERtEX_NUM];            //存储图中顶点数据
    AdjMatrix arcs;                             //二维数组，记录顶点之间的关系
    int vexnum,arcnum;                          //记录图的顶点数和弧（边）数
    GraphKind kind;                             //记录图的种类
}MGraph;
```

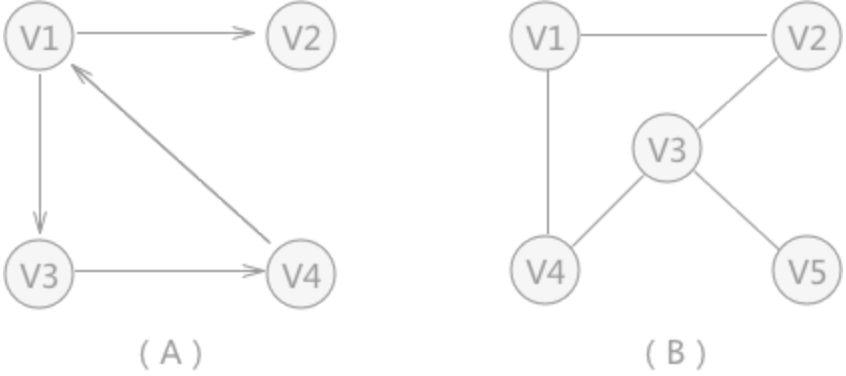


图1 有向图和无向图

例如，存储图 1 中的无向图（B）时，除了存储图中各顶点本身具有的数据外，还需要使用二维数组存储任意两个顶点之间的关系。

由于 （B） 为无向图，各顶点没有权值，所以如果两顶点之间有关联，相应位置记为 1 ； 反之记为 0 。构建的二维数组如图 2 所示。

$$\begin{pmatrix} 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

图2 无向图对应的二维数组arcs

在此二维数组中，每一行代表一个顶点，依次从 V1 到 V5 ，每一列也是如此。比如 arcs[0][1] = 1 ， 表示 V1 和 V2 之间有边存在；而 arcs[0][2] = 0，说明 V1 和 V3 之间没有边。

对于无向图来说，二维数组构建的二阶矩阵，实际上是对称矩阵，在存储时就可以采用压缩存储的方式存储下三角或者上三角。

通过二阶矩阵，可以直观地判断出各个顶点的度，为该行（或该列）非 0 值的和。例如，第一行有两个 1，说明 V1 有两个边，所以度为 2。

存储图 1 中的有向图（A）时，对应的二维数组如图 3 所示：

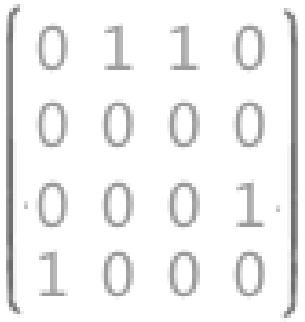

图 3 有向图对应的二维数组arcs

例如，arcs[0][1] = 1 ，证明从 V1 到 V2 有弧存在。且通过二阶矩阵，可以很轻松得知各顶点的出度和入度，出度为该行非 0 值的和，入度为该列非 0 值的和。例如，V1 的出度为第一行两个 1 的和，为 2； V1 的入度为第一列中 1 的和，为 1 。所以 V1 的出度为 2 ，入度为 1 ，度为两者的和 3 。

## 具体实现代码

```
#include <stdio.h>
#define MAX_VERtEX_NUM 20                       //顶点的最大个数
#define VRType int                              //表示顶点之间的关系的变量类型
#define InfoType char                           //存储弧或者边额外信息的指针变量类型
#define VertexType int                          //图中顶点的数据类型
typedef enum{DG,DN,UDG,UDN}GraphKind;           //枚举图的 4 种类型
typedef struct {
    VRType adj;                                 //对于无权图，用 1 或 0 表示是否相邻；对于带权图，直
    InfoType * info;                            //弧或边额外含有的信息指针
}ArcCell,AdjMatrix[MAX_VERtEX_NUM][MAX_VERtEX_NUM];

typedef struct {
    VertexType vexs[MAX_VERtEX_NUM];            //存储图中顶点数据
    AdjMatrix arcs;                             //二维数组，记录顶点之间的关系
    int vexnum,arcnum;                          //记录图的顶点数和弧（边）数
    GraphKind kind;                             //记录图的种类
}MGraph;
//根据顶点本身数据，判断出顶点在二维数组中的位置
int LocateVex(MGraph * G,VertexType v){
    int i=0;
    //遍历一维数组，找到变量v
    for (; i<G->vexnum; i++) {
        if (G->vexs[i]==v) {
            break;
        }
    }
    //如果找不到，输出提示语句，返回-1
    if (i>G->vexnum) {
        printf("no such vertex.\n");
        return -1;
```

```
    }
    return i;
}
//构造有向图
void CreateDG(MGraph *G){
    //输入图含有的顶点数和弧的个数
    scanf("%d,%d",&(G->vexnum),&(G->arcnum));
    //依次输入顶点本身的数据
    for (int i=0; i<G->vexnum; i++) {
        scanf("%d",&(G->vexs[i]));
    }
    //初始化二维矩阵，全部归0，指针指向NULL
    for (int i=0; i<G->vexnum; i++) {
        for (int j=0; j<G->vexnum; j++) {
            G->arcs[i][j].adj=0;
            G->arcs[i][j].info=NULL;
        }
    }
    //在二维数组中添加弧的数据
    for (int i=0; i<G->arcnum; i++) {
        int v1,v2;
        //输入弧头和弧尾
        scanf("%d,%d",&v1,&v2);
        //确定顶点位置
        int n=LocateVex(G, v1);
        int m=LocateVex(G, v2);
        //排除错误数据
        if (m==-1 ||n==-1) {
            printf("no this vertex\n");
            return;
        }
        //将正确的弧的数据加入二维数组
        G->arcs[n][m].adj=1;
    }
}
//构造无向图
void CreateDN(MGraph *G){
    scanf("%d,%d",&(G->vexnum),&(G->arcnum));
    for (int i=0; i<G->vexnum; i++) {
        scanf("%d",&(G->vexs[i]));
    }
    for (int i=0; i<G->vexnum; i++) {
        for (int j=0; j<G->vexnum; j++) {
            G->arcs[i][j].adj=0;
            G->arcs[i][j].info=NULL;
        }
    }
```

```
78.     for (int i=0; i<G->arcnum; i++) {
79.         int v1,v2;
80.         scanf("%d,%d",&v1,&v2);
81.         int n=LocateVex(G, v1);
82.         int m=LocateVex(G, v2);
83.         if (m==-1 ||n==-1) {
84.             printf("no this vertex\n");
85.             return;
86.         }
87.         G->arcs[n][m].adj=1;
88.         G->arcs[m][n].adj=1;//无向图的二阶矩阵沿主对角线对称
89.     }
90. }
91. //构造有向网，和有向图不同的是二阶矩阵中存储的是权值。
92. void CreateUDG(MGraph *G){
93.     scanf("%d,%d",&(G->vexnum),&(G->arcnum));
94.     for (int i=0; i<G->vexnum; i++) {
95.         scanf("%d",&(G->vexs[i]));
96.     }
97.     for (int i=0; i<G->vexnum; i++) {
98.         for (int j=0; j<G->vexnum; j++) {
99.             G->arcs[i][j].adj=0;
100.            G->arcs[i][j].info=NULL;
101.        }
102.    }
103.    for (int i=0; i<G->arcnum; i++) {
104.        int v1,v2,w;
105.        scanf("%d,%d,%d",&v1,&v2,&w);
106.        int n=LocateVex(G, v1);
107.        int m=LocateVex(G, v2);
108.        if (m==-1 ||n==-1) {
109.            printf("no this vertex\n");
110.            return;
111.        }
112.        G->arcs[n][m].adj=w;
113.    }
114. }
115. //构造无向网。和无向图唯一的区别就是二阶矩阵中存储的是权值
116. void CreateUDN(MGraph* G){
117.     scanf("%d,%d",&(G->vexnum),&(G->arcnum));
118.     for (int i=0; i<G->vexnum; i++) {
119.         scanf("%d",&(G->vexs[i]));
120.     }
121.     for (int i=0; i<G->vexnum; i++) {
122.         for (int j=0; j<G->vexnum; j++) {
123.             G->arcs[i][j].adj=0;
124.             G->arcs[i][j].info=NULL;
```

```
125.          }
126.      }
127.      for (int i=0; i<G->arcnum; i++) {
128.          int v1,v2,w;
129.          scanf("%d,%d,%d",&v1,&v2,&w);
130.          int m=LocateVex(G, v1);
131.          int n=LocateVex(G, v2);
132.          if (m==-1 ||n==-1) {
133.              printf("no this vertex\n");
134.              return;
135.          }
136.          G->arcs[n][m].adj=w;
137.          G->arcs[m][n].adj=w;//矩阵对称
138.      }
139.  }
140.  void CreateGraph(MGraph *G){
141.      //选择图的类型
142.      scanf("%d",&(G->kind));
143.      //根据所选类型，调用不同的函数实现构造图的功能
144.      switch (G->kind) {
145.          case DG:
146.              return CreateDG(G);
147.              break;
148.          case DN:
149.              return CreateDN(G);
150.              break;
151.          case UDG:
152.              return CreateUDG(G);
153.              break;
154.          case UDN:
155.              return CreateUDN(G);
156.              break;
157.          default:
158.              break;
159.      }
160.  }
161.  //输出函数
162.  void PrintGrapth(MGraph G)
163.  {
164.      for (int i = 0; i < G.vexnum; i++)
165.      {
166.          for (int j = 0; j < G.vexnum; j++)
167.          {
168.              printf("%d ", G.arcs[i][j].adj);
169.          }
170.          printf("\n");
171.      }
```

```
}
int main() {
    MGraph G;//建立一个图的变量
    CreateGraph(&G);//调用创建函数，传入地址参数
    PrintGrapth(G);//输出图的二阶矩阵
    return 0;
}
```

> 注意：在此程序中，构建无向网和有向网时，对于之间没有边或弧的顶点，相应的二阶矩阵中存放的是 0。目的只是为了方便查看运行结果，而实际上如果顶点之间没有关联，它们之间的距离应该是无穷大（∞）。

例如，使用上述程序存储图 4（a）的有向网时，存储的两个数组如图 4（b）所示：

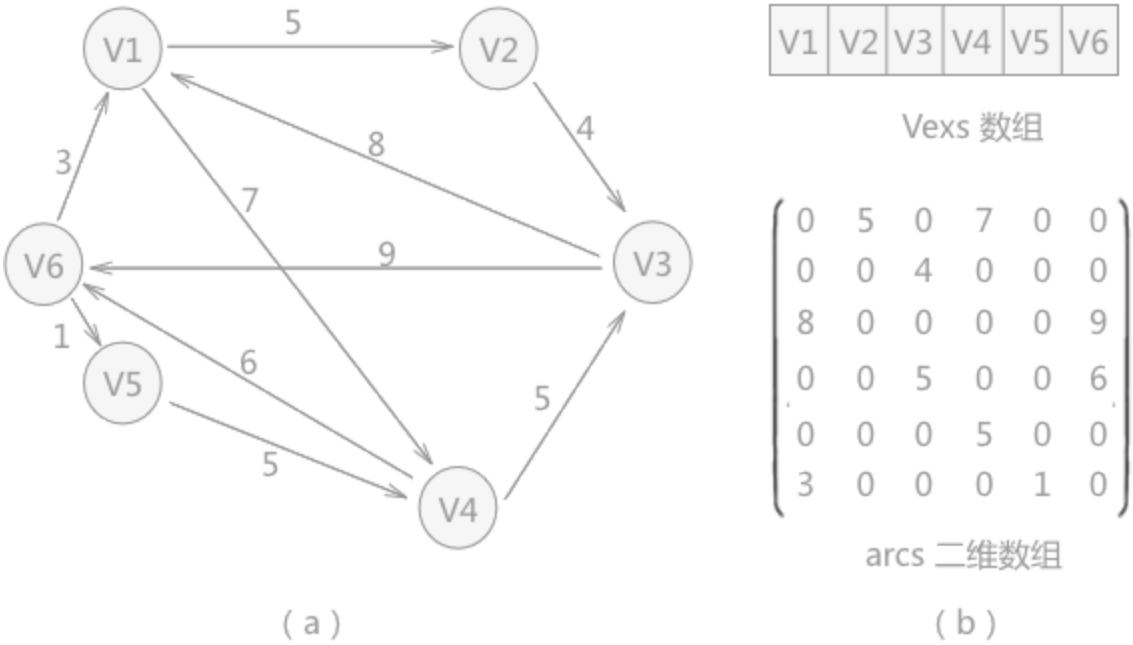


图 4 有向网

相应地运行结果为：

```
2
6,10
1
2
3
4
5
6
1,2,5
2,3,4
3,1,8
1,4,7
```

```
4,3,5
3,6,9
6,1,3
4,6,6
6,5,1
5,4,5
0 5 0 7 0 0
0 0 4 0 0 0
8 0 0 0 0 9
0 0 5 0 0 6
0 0 0 5 0 0
3 0 0 0 1 0
```

## 总结

本节主要详细介绍了使用数组存储图的方法，在实际操作中使用更多的是链式存储结构，例如邻接表、十字链表和邻接多重表，这三种存储图的方式放在下一节重点去讲。

上一节 下一节

联系方式 购买教程（带答疑）