图的存储结构算法设计示例

QDU_JZR

图的邻接矩阵类型声明：

```
// ----- 图的数组(邻接矩阵)存储表示 --------
#define INFINITY INT_MAX                    // 用整型最大值代替∞
#define MAX_VERTEX_NUM 30                   // 最大顶点个数
enum GraphKind {DG, DN, UDG, UDN};          // {有向图,有向网,无向图,无向网}
typedef int VRType;
typedef char VertexType[MAX_NAME];

typedef struct {
    VRType adj; // 顶点关系类型。对无权图，用1或0表示相邻否；对带权图，则为权值
    InfoType *info; // 该弧相关信息的指针(可无)
} ArcCell, AdjMatrix[MAX_VERTEX_NUM][MAX_VERTEX_NUM]; // 二维数组

typedef struct Graph {
    VertexType vexs[MAX_VERTEX_NUM]; // 顶点向量
    AdjMatrix arcs; // 邻接矩阵
    int vexnum, arcnum; // 图的当前顶点数和弧数
    GraphKind kind; // 图的种类标志
} MGraph;
```

ODU-LZR

图的邻接矩阵类型声明：

```
#define INF    INT_MAX          // 用整型最大值代替∞
#define MAXVEX 30               //图中最大顶点个数
typedef char VertexType[10];
                                //定义VertexType为字符串类型
typedef struct vertex
{
   int adjvex;                  //顶点编号
   VertexType data;             //顶点的信息
} VType;                        //顶点类型

typedef struct graph
{  int n, e;                    //n为实际顶点数,e为实际边数
   VType vexs[MAXVEX];          //顶点集合
   int edges[MAXVEX][MAXVEX]; //边的集合
} MGraph;                       //图的邻接矩阵类型
```

ODU-LZR

在邻接矩阵上实现图的基本运算的算法示例

（1）建立图的邻接矩阵算法

由邻接矩阵数组*A*、顶点数*n*和边数*e*建立图G的邻接矩阵存储结构。

```
void CreateGraph(MGraph &g,int A[][MAXVEX],int n,int e)
{
   int i,j;
   g.n=n; g.e=e;
   for (i=0;i<n;i++)
      for (j=0;j<n;j++)
        g.edges[i][j]=A[i][j];
}
```

QDU - LZR

（2）求顶点度算法

对于无向图和有向图，求顶点度有所不同。依据定义，求无向图G中顶点v的度的算法如下：

```
int Degree1(MGraph g,int v)    //求无向图中顶点的度
{
   int i,d=0;
   if (v<0 || v>=g.n)
      return -1;                //顶点编号错误返回-1
   for (i=0;i<g.n;i++)
      if (g.edges[v][i]>0 && g.edges[v][i]<INF)
        d++;                //统计第v行既不为0也不为∞的边数即度
   return d;
}
```

QDU - LZR

求有向图G中顶点v的度的算法如下：

```
int Degree2(MGraph g, int v)  //求有向图中顶点的度
{
   int i,d1=0,d2=0,d;
   if (v<0 || v>=g.n)
      return -1;       //顶点编号错误返回-1
   for (i=0;i<g.n;i++)
      if (g.edges[v][i]>0 && g.edges[v][i]<INF)
         d1++;         //统计第v行既不为0也不为∞的边数即出度
   for (i=0;i<g.n;i++)
      if (g.edges[i][v]>0 && g.edges[i][v]<INF)
         d2++;         //统计第v列既不为0也不为∞的边数即入度
   d=d1+d2;
   return d;
}
```

QDU_LZR

一个图的邻接表存储结构的类型声明如下：

```
typedef char VertexType[10];    //VertexType为字符串类型
typedef struct edgenode
{   int adjvex;                 //相邻点序号
    int weight;                 //边的权值
    struct edgenode *nextarc;   //下一条边的顶点
} ArcNode;              //每个顶点建立的单链表中边结点的类型
typedef struct vexnode
{   VertexType data;            //存放一个顶点的信息
    ArcNode *firstarc;          //指向第一条边结点
} VHeadNode;                    //单链表的头结点类型
typedef struct
{   int n,e;                    //n为实际顶点数,e为实际边数
    VHeadNode adjlist[MAXVEX];  //单链表头结点数组
} ALGraph;                      //图的邻接表类型
```

QDU-LZR

在邻接表上实现图的基本算法示例

（1）建立图的邻接表算法

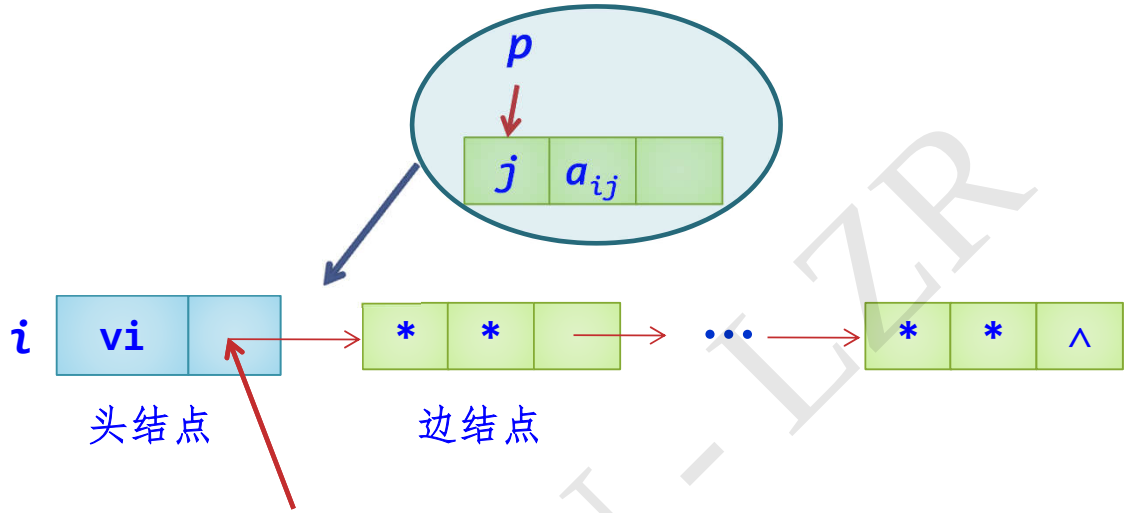
由邻接矩阵数组A、顶点数n和边数e建立图G的邻接表存储结构。

基本思路：

- 先创建邻接表头结点数组，并置所有头结点的firstarc为NULL。
- 遍历邻接矩阵数组A，当A[i][j]≠0且A[i][j]≠∞时，说明有一条从顶点i到顶点j的边，建立一个边结点p，置其adjvex域为j，其weight域为A[i][j]（a_{ij}），将p结点插入到顶点i的单链表头部。

CDU LZR

顶点i到j有边：



引用方式：G->adjlist[i].firstarc

实现语句：

```
p->nextarc=G->adjlist[i].firstarc;
G->adjlist[i].firstarc=p;
```

注意：图中每个顶点有一个头结点，每条边有一个边结点（无向图一条边对应2个边结点）。

ODU - LZR

```
void CreateGraph(ALGraph *&G,int A[][MAXVEX],int n,int e)
{
   int i,j;
   ArcNode *p;
   G=(ALGraph *)malloc(sizeof(ALGraph));
   G->n=n; G->e=e;
   for (i=0;i<G->n;i++)        //邻接表中所有头结点的指针域置空
     G->adjlist[i].firstarc=NULL;
   for (i=0;i<G->n;i++)        //检查A中每个元素
     for (j=G->n-1;j>=0;j--)
        if (A[i][j]>0 && A[i][j]<INF)              //存在一条边
        {  p=(ArcNode *)malloc(sizeof(ArcNode));   //创建结点p
           p->adjvex=j;
           p->weight=A[i][j];
           p->nextarc=G->adjlist[i].firstarc;     //头插法插入p
           G->adjlist[i].firstarc=p;
        }
}
```

CDUT-LZR

（2）销毁图运算算法

邻接表的头结点和边结点都是采用malloc函数分配的，在不再需要时应用free函数释放所有分配的空间。

基本思路：通过adjlist数组遍历每个单链表，释放所有的边结点，最后释放adjlist数组的空间。

QDU - LZR

```
void DestroyGraph(ALGraph *&G)          //销毁图
{  int i;
   ArcNode *pre,*p;
   for (i=0;i<G->n;i++)                 //遍历所有的头结点
   {  pre=G->adjlist[i].firstarc;
      if (pre!=NULL)
      {  p=pre->nextarc;
         while (p!=NULL)  //释放adjlist[i]的所有边结点
         {  free(pre);
            pre=p; p=p->nextarc;
         }
         free(pre);
      }
   }
   free(G);                  //释放G所指的头结点数组的内存空间
}
```

QDU-LZR

（3）求顶点度算法

对于无向图和有向图，求顶点度有所不同。依据定义，求无向图G中顶点v的度的算法如下：

```
int Degree1(ALGraph *G,int v)   //求无向图G中顶点v的度
{  int d=0;
   ArcNode *p;
   if (v<0 || v>=G->n)
      return -1;      //顶点编号错误返回-1
   p=G->adjlist[v].firstarc;
   while (p!=NULL)    //统计v顶点的单链表中边结点个数即度
   {  d++;
      p=p->nextarc;
   }
   return d;
}
```

QDU - LZR

求有向图G中顶点v的度的算法如下：

```
int Degree2(ALGraph *G,int v)  //求有向图G中顶点v的度
{  int i,d1=0,d2=0,d; ArcNode *p;
   if (v<0 || v>=G->n)
      return -1;                    //顶点编号错误返回-1
   p=G->adjlist[v].firstarc;
   while (p!=NULL)          //统计v的单链表中边结点个数即出度
   {  d1++;
      p=p->nextarc;
   }
   for (i=0;i<G->n;i++)  //统计边结点中adjvex为v的个数即入度
   {  p=G->adjlist[i].firstarc;
      while (p!=NULL)
      {  if (p->adjvex==v) d2++;
         p=p->nextarc;
      }
   }
   d=d1+d2;
   return d;
}
```

QDU - LZR

【示例】 对于具有n个顶点的有向图G，设计以下两个算法：

（1）设计一个将邻接矩阵g转换为邻接表G的算法；

（2）设计一个将邻接表G转换为邻接矩阵g的算法。

解：（1）设计一个将邻接矩阵g转换为邻接表G的算法。

- 先分配G的内存空间并将所有头结点的firstarc域置为NULL。
- 遍历邻接矩阵g，查找元素值不为0且不为∞的元素g.edges[i][j]，找到这样的元素后创建一个边结点p，将其插入到G->adjlist[i]单链表的首部。

QDU-LZR

```
void MatToAdj(MGraph g,ALGraph *&G)
//将邻接矩阵g转换成邻接表G
{
   int i,j;  ArcNode *p;
   G=(ALGraph *)malloc(sizeof(ALGraph));
   for (i=0;i<g.n;i++)        //邻接表中所有头结点的指针域置初值
      G->adjlist[i].firstarc=NULL;
   for (i=0;i<g.n;i++)        //检查邻接矩阵中每个元素
     for (j=g.n-1;j>=0;j--)
       if (g.edges[i][j]!=0 && g.edges[i][j]!=INF) //有一条边
       {  p=(ArcNode *)malloc(sizeof(ArcNode));    //创建结点p
          p->adjvex=j;
          p->weight=g.edges[i][j];
          p->nextarc=G->adjlist[i].firstarc;   //头插法插入p
          G->adjlist[i].firstarc=p;
       }
   G->n=g.n;
   G->e=g.e;                                  //置顶点数和边数
}
```

GDU-LZR

（2）设计一个将邻接表G转换为邻接矩阵g的算法

- 先将邻接矩阵g中所有元素初始化：对角线元素置为0，其他元素置为∞。
- 然后遍历邻接表的每个单链表，当访问到G->adjlist[i]单链表的结点p时，将邻接矩阵g的元素g.edges[i][p->adjvex]修改为p->weight。

QDU - IZR

```
void AdjToMat(ALGraph *G,MGraph &g)
//将邻接表G转换成邻接矩阵g
{  int i,j;  ArcNode *p;
   for (i=0;i<G->n;i++)
     for (j=0;j<G->n;j++)
        if (i==j) g.edges[i][i]=0;   //对角线置为0
           else g.edges[i][j]=INF;
   for (i=0;i<G->n;i++)
   {  p=G->adjlist[i].firstarc;
      while (p!=NULL)
      {  g.edges[i][p->adjvex]=p->weight;
         p=p->nextarc;
      }
   }
   g.n=G->n; g.e=G->e;                  //置顶点数和边数
}
```

QDU - LZR

END

QDU-LZR