5.2.2图的存储

回忆: 路由协议设问题中, 需要存储路由器和网关的信息,路由相互之间是否有通路及通路长度三个信息. 即使部分问题不牵涉通路长度(图的权值), 也至少需要存储图的顶点和边的两方面信息,如何存储?

仍然有顺序存储和链式存储2种方法!

图的存储2-邻接表

1. 无向图邻接表

对图中每个顶点Vi建立一个单链表，链表中的结点表示依附于顶点Vi的边，每个链表结点为两个域：

adjvex	nextarc

其中：邻接点域（adjvex）记载与顶点Vi邻接的顶点信息；

链域（nextarc）指向下一个与顶点Vi邻接的链表结点

每个链表附设一个头结点，头结点结构为：

其中：vexdata存放顶点信息（姓名、编号等）；

fristarc指向链表的第一个结点。

vexdata	firstarc

图的存储2-邻接表

1. 无向图邻接表

对图中每个 中的结点表
示依附于顶 个域：

```
struct edge
{
        int v;
        struct edge *nextarc;
};
```

其中：邻接点域（adjvex）指出与顶点vi邻接的顶点信息；
链域 邻接的链表结点

每个链表附 为：

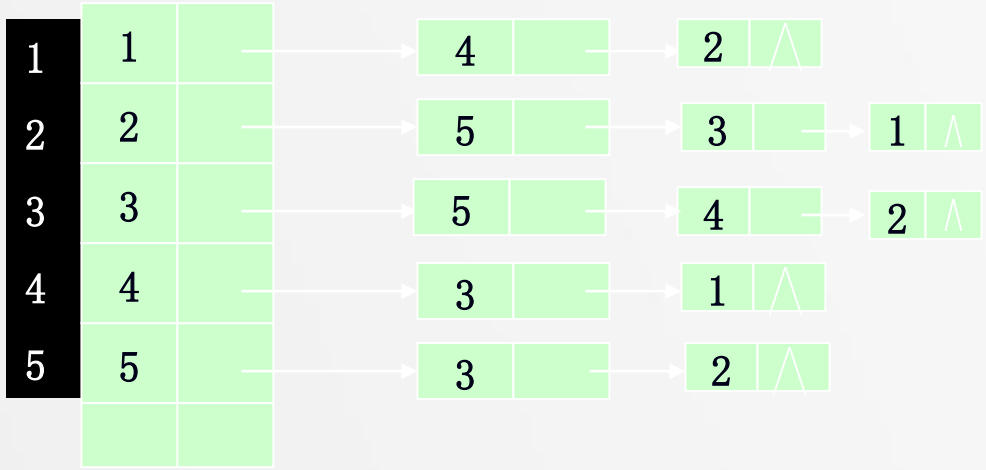
```
struct vex
{
        ElemType data;
        struct edge *firstarc;
};
```

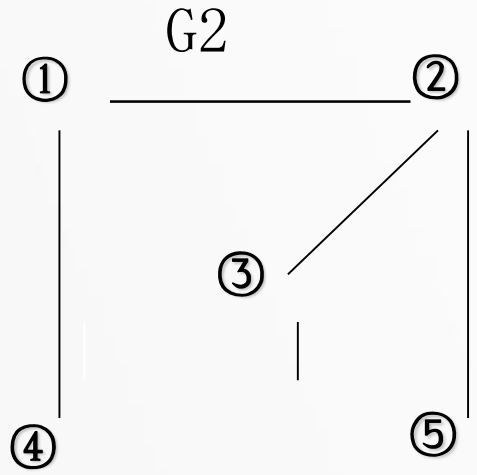
其中：vexda ；
fristarc指

data	firstarc

图的存储2-邻接表

如图G2的邻接表为：





无向图邻接表特点：

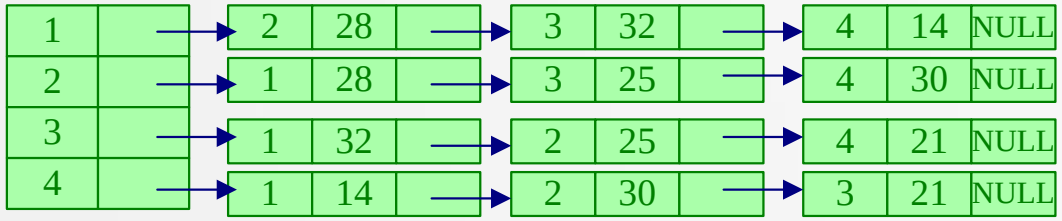
1.n个顶点，e条边的无向图，需n个头结点和2e个链表结点

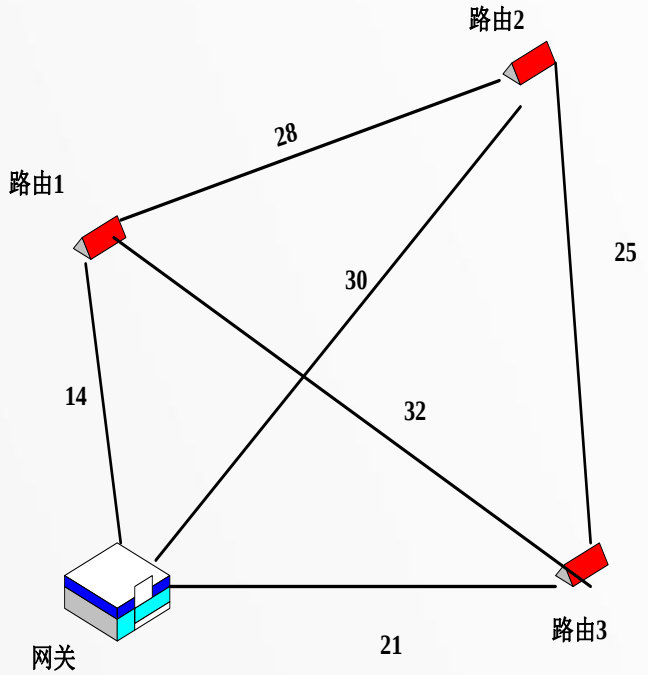
2.顶点Vi的度 TD（Vi）=链表i中的链表结点数

讨 论

请采用邻接表存储右图数据

答:



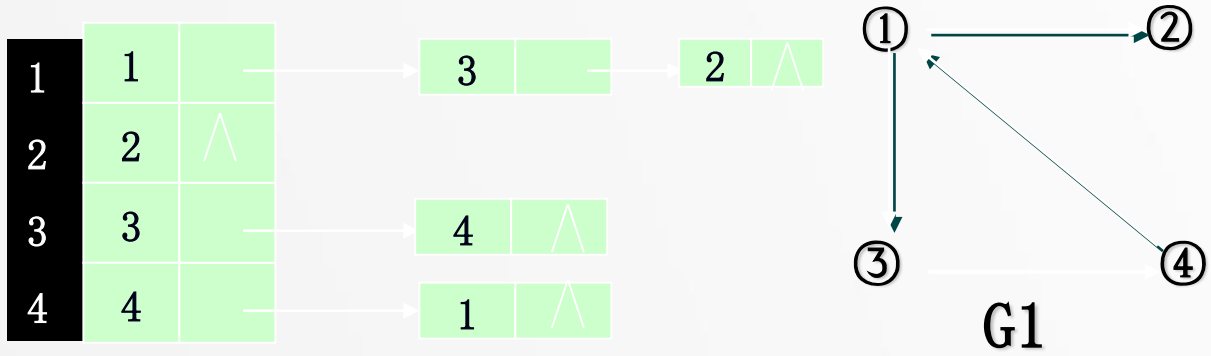


图的存储2-邻接表

2. 有向图邻接表

与无向图的邻接表结构一样。只是在第i条链表上的结点是以Vi为弧尾的各个弧头顶点

G1的邻接表



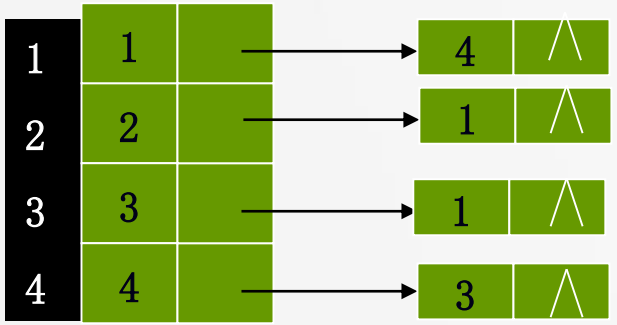
有向图邻接表特点：
1. n个顶点，e条弧的有向图，需n个表头结点，e 个链表结点
2. 第i条链表上的链表结点数，为Vi的出度(求顶点的出度易，求入度难)

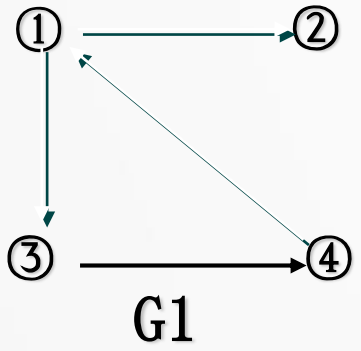
图的存储2-邻接表

3. 有向图逆邻接表

与无向图的邻接表结构一样。只是在第i条链表上的结点是以Vi为弧头的各个弧尾顶点

G1的逆邻接表





此时，第i条链表上的结点数，为Vi的入度

邻接表的结构定义和建立算法：

```
typedef  struct node{ //边表结点
  int adjtex;         //邻接点域
  struct node *next;  //链域
 }EdgeNode;           //若也表示边上的权，增加一个数据域
typedef  struct vnode{ //顶点表结点
  VertexType  vertex;  //顶点域
  EdgeNode *firstedeg; // 边表头指针
} VertexNode;
typedef VertexNode AdjList[MaxNodeNum];
typedef  struct{
   AdjList  adjlist;// 邻接表
   int n,e;         //顶点数和边数
}ALGraph;  //对于简单应用无需定义此类型，直接使用AdjList
                                    类型。
```

建立无向图的邻接表

```
void CreateALGraph(ALGraph *G)
 { int i.j.k; EdgeNode *s;
```

```
scanf( “%d%d” ,&G->n,&G->e); //读入顶点数和边数
for(i=0;i<G->n;i++){                    //建立顶点表
 G->adjlist[i].vertex=getchar( );  //读入顶点信息
 G->adjlist[i].firstedge=NULL;} //边表置空
for(k=0;k<G->e;k++){                    //建立边表
 scanf( “%d%d” ,&i,&j);                  //读入边(vi,vj)的顶点对序
 s=(EdgeNode  *)malloc(sizeof(EdgeNode)); //生成边表结点
s->adjvex=j;     //邻结点的序号为j
 s->next=G-> adjlist[i].firstedge;//前插入
 G-> adjlist[i].firstedge=s; //将新结点*s插入vi头部
 s= (EdgeNode  *)malloc(sizeof(EdgeNode));
 s->adjvex=i;  //邻接序号为i
 s->next=G-> adjlist[j].firstedge;
 G-> adjlist[j].firstedge=s;  //将新结点*s插入vj头部
 }
}
```