图的存储——链式前向星

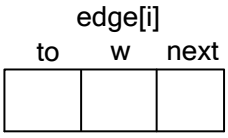
图的存储方法很多，最常见的除了邻接矩阵、邻接表和边集数组外，还有链式前向星。链式前向星是一种静态链表存储，用边集数组和邻接表相结合，可以快速访问一个顶点的所有邻接点，在算法竞赛中广泛应用。

链式前向星存储包括两种结构：

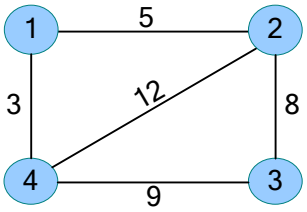
- 边集数组：edge[ ]，edge[i]表示第 i 条边；
- 头结点数组：head[ ]，head[i]存以 i 为起点的第一条边的下标(在 edge[]中的下标)

```
struct node
{
    int to,next,w;
}edge[maxe];//边集数组，边数一般要设置比 maxn*maxn 大的数，如果题目有要求除外
int head[maxn];//头结点数组
```

每一条边的结构，如图 所示。



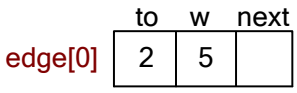
例如，一个无向图，如图 所示。



按以下顺序输入每条边的两个端点，建立的链式前向星，过程如下。

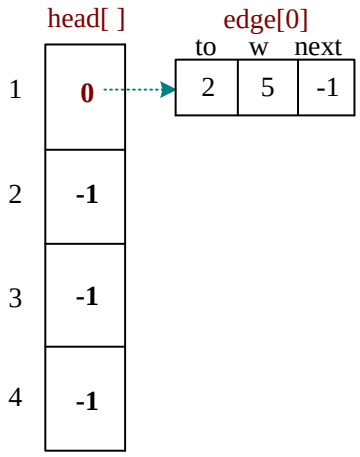
（1） 输入 1 2 5

创建一条边 1—2，权值为 5，创建第一条边 edge[0]，如图所示。

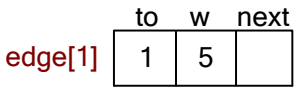


然后将该边链接到 1 号结点的头结点中。（初始时 head[]数组全部初始化为-1）

即 edge[0].next=head[1]; head[1]=0; 表示 1 号结点关联的第一个条边为 0 号边，如图所示。图中的虚线箭头仅表示他们之间的链接关系，不是指针。

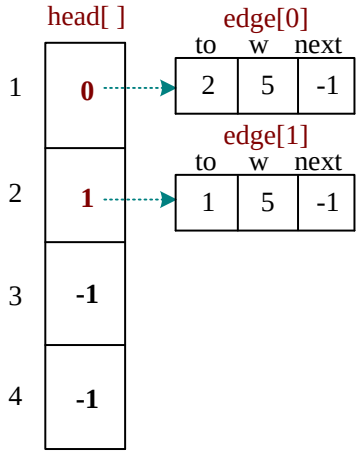


因为是无向图，还需要添加它的反向边，2—1，权值为 5。创建第二条边 edge[1]，如图所示。



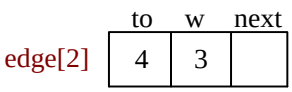
然后将该边链接到 2 号结点的头结点中。

即 edge[1].next=head[2]; head[2]=1; 表示 2 号结点关联的第一个条边为 1 号边，如图所示。



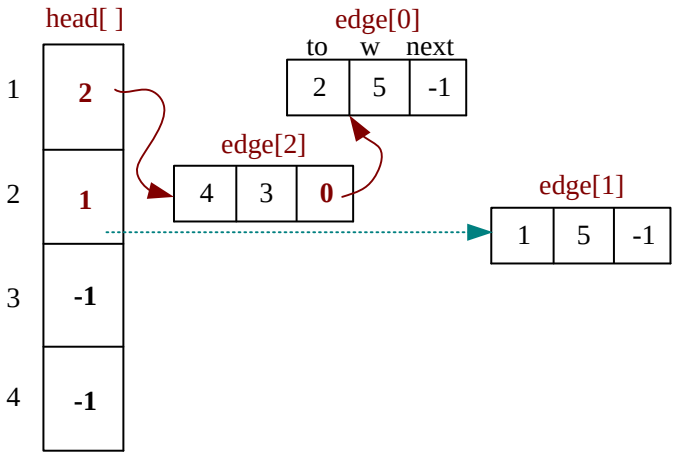
（2） 输入 1 4 3

创建一条边 1—4，权值为 3，创建第 3 条边 edge[2]，如图所示。

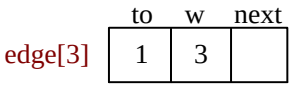


然后将该边链接到 1 号结点的头结点中（头插法）。

即 edge[2].next=head[1]; head[1]=2; 表示 1 号结点关联的第一个条边为 2 号边，如图所示。

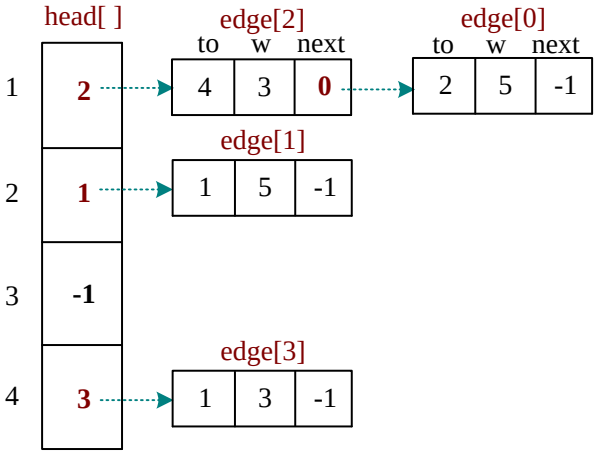


因为是无向图，还需要添加它的反向边，4—1，权值为 3。创建第 4 条边 edge[3]，如图所示。



然后将该边链接到 4 号结点的头结点中。

即 edge[3].next=head[4]; head[4]=3; 表示 4 号结点关联的第一个条边为 3 号边，如图所示。

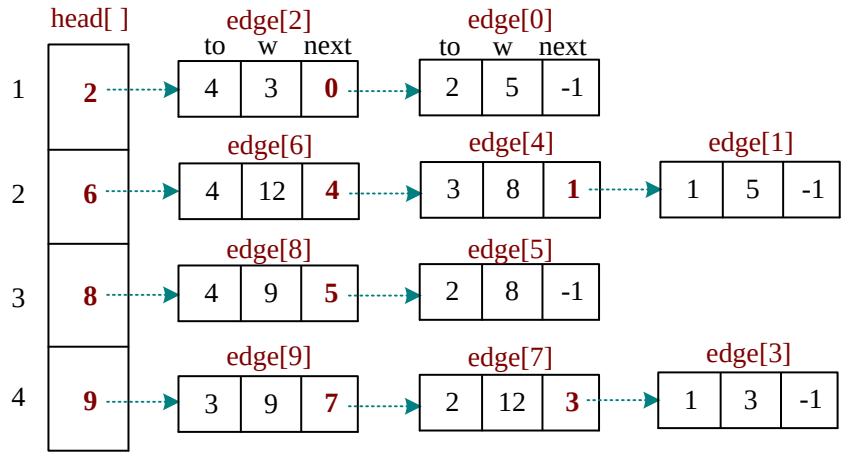


（3） 依次输入以下三条边，创建的链式前向星，如图 所示。

2 3 8

2 4 12

3 4 9



添加一条边 u v w 的代码如下：

```
void add(int u,int v,int w)//添加一条边
{
    edge[cnt].to=v;
    edge[cnt].w=w;
    edge[cnt].next=head[u];
    head[u]=cnt++;
}
```

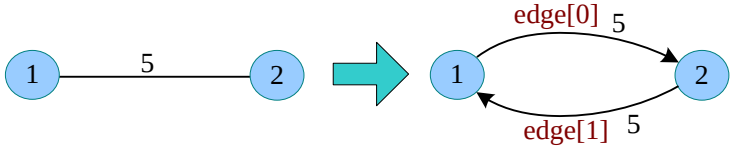
如果是有向图，每输入一条边，执行一次 add(u,v,w)即可；如果是无向图，则需要执行两次 add(u,v,w); add(v,u,w)。

如何使用链式前向星访问一个结点 u 的所有邻接点呢？

```
for(int i=head[u];i!=-1;i=edge[i].next)
{
    int v=edge[i].to;//u 的邻接点
    int w=edge[i].w;//u—v 的权值
    …
}
```

链式前向星的特性：

1) 和邻接表一样，因为采用头插法进行链接，所以边输入顺序不同，创建的链式前向星也不同。
2) 对于无向图，每输入一条边，需要添加两条边，互为反向边。例如，输入第一条边 1 2 5，实际上添加了两条边，如图 所示。



这两条边可以通过互为反向边，可以通过与 1 的异或运算得到其反向边，0^1=1，1^1=0。也就是说如果一条边的下标为 i，则其反向边为 i^1。这个特性应用在网络流中非常方便。

3) 链式前向星具有边集数组和邻接表的功能，属于静态链表，不需要频繁地创建结点，应用十分灵活。