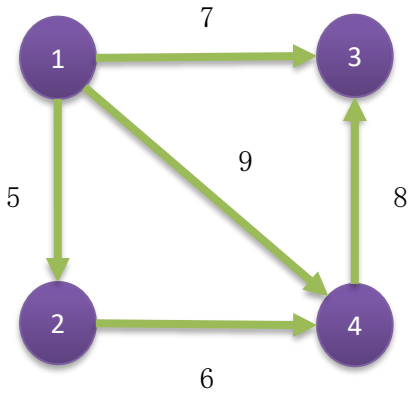
邻接表

邻接表主要用来存储稀疏图，当边的数量M远远小于顶点的平方的时候，就可以考虑使用这种方式。

举例，现有如下数据和图像：

```
4  5
1  4  9
4  3  8
1  2  5
2  4  6
1  3  7
```



输入数字的意思如下

第一行两个整数n m。n表示顶点个数（顶点编号为1~n），m表示边的条数。接下来m行表示，每行有3个数x y z，表示顶点x到顶点y的边的权值为z。

现在，我们来研究一下如何使用邻接表来存储这个图像。

首先我们按照读入的顺序为每一条边进行编号（1~m）。比如第一条边“1 4 9”的编号就是1，“1 3 7”这条边的编号是5。

这里用u、v和w三个数组用来记录每条边的具体信息，即u[i]、v[i]和w[i]表示第i条边是从第u[i]号顶点到v[i]号顶点（u[i]à v[i]），且权值为w[i]。

转化成表格如下：

	u	v	w
1			
2			
3			
4			
5			

接下来，我们需要定义一个first数组。它的索引的含义是顶点的编号，值的含义是当前顶点所连接的边的号码。

此外，我们还需要定一个next数组。它的索引含义是边的编号，值的含义也是边的编号。至于它的用途，我们边填写表格，边来理解。

两个数组的初始结构如下：

first

1	-1
2	-1
3	-1
4	-1

next

1	0
2	0
3	0
4	0

现在，我们开始读入边，首先是1 4 9。将数据存储后，图像表示如下：

	u	v	w
1	1	4	9
2			
3			
4			
5			

first

1	1
2	-1
3	-1
4	-1

next

1	-1
2	0
3	0
4	0

1号顶点连接的是1号边，因此first填入1,next数组为什么填写-1呢？且看下文。

继续填入第二条边4 3 8。将数据存储后，图像表示如下：

	u	v	w
1	1	4	9
2	4	3	8
3			
4			
5			

first

1	1
2	-1
3	-1
4	2

next

1	-1
2	-1
3	0
4	0

同样，我们对next数组的填写先忽略。现在看是填入地3条边1 2 5。图像如下：

	u	v	w
1	1	4	9
2	4	3	8
3	1	2	5
4			
5			

first

1	3
2	-1
3	-1
4	2

next

1	-1
2	-1
3	1
4	0

看到这里，答案即将揭晓！

first数组中索引1原本的值是1，但是当第3条边填入之后，它却变成了3！对，在这里我们将与顶点1连接的边进行了修改，由1变成了3，即用最新的边的索引进行替换。

再看next数组，此时3对应的值不再是-1，而是变成了1，这个1也是边的编号，意思是与3号边共顶点的前一条边的编号是1。

继续来填入第4条边：2 4 5。

	u	v	w
1	1	4	9
2	4	3	8
3	1	2	5
4	2	4	5
5			

first

1	3
2	4
3	-1
4	2

next

1	-1
2	-1
3	1
4	-1

现在来填入最后一条边：1 3 7

	u	v	w
1	1	4	9
2	4	3	8
3	1	2	5
4	2	4	5
5	1	3	7

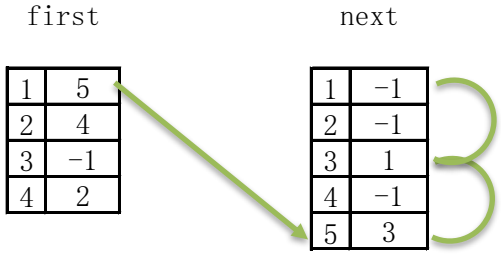
first

1	5
2	4
3	-1
4	2

next

1	-1
2	-1
3	1
4	-1
5	3

此时，如果我们想遍历1号顶点的每一条边就很简单了。1号顶点的其中一条边的编号存储在first[1]中。其余的边则可以通过next数组寻找到。请看下图。



细心的同学会发现，此时遍历边某个顶点边的时候的遍历顺序正好与读入时候的顺序相反。因为在为每个顶点插入边的时候都直接插入“链表”的首部而不是尾部。不过这并不会产生任何问题，这正是这种方法的其妙之处。

编码

```
#include<bits/stdc++.h>

using namespace std;

int main() {
    int n, m, i;
    //u、v和w的数组大小要根据实际情况来设置，要比m的最大值要大1
```

```
    int u[6], v[6], w[6];
    //first和next的数组大小要根据实际情况来设置，要比n的最大值要大1
    int first[5], next[5];
    scanf("%d %d", &n, &m);
    //初始化first数组下标1~n的值为-1，表示1~n顶点暂时都没有边
    for (i = 1; i <= n; i++) {
        first[i] = -1;
    }
    for (i = 1; i <= m; i++) {
        //读入每一条边
       scanf("%d %d %d", &u[i], &v[i], &w[i]);
        //更新first数组和next数组
       next[i] = first[u[i]];
        first[u[i]] = i;
    }
    //找出一个顶点所关联的全部的边
    // 1号顶点其中的一条边的编号（其实也是最后读入的边）
    int k = first[1];
    //其余的边都可以在next数组中依次找到
    while (k != -1) {
        printf("%d %d %d ", u[k], v[k], w[k]);
        k = next[k];
    }
    //遍历每个顶点的所有边的代码如下。
    for (i = 1; i <= n; i++) {
        k = first[i];
        while (k != -1) {
            printf("%d %d %d ", u[k], v[k], w[k]);
            k = next[k];
        }
    }
    return 0;
}
```