
# 数据结构-Dijistra

信智学院　陈琰宏

# 主要内容


01 Dijkstra算法

02 优先队列

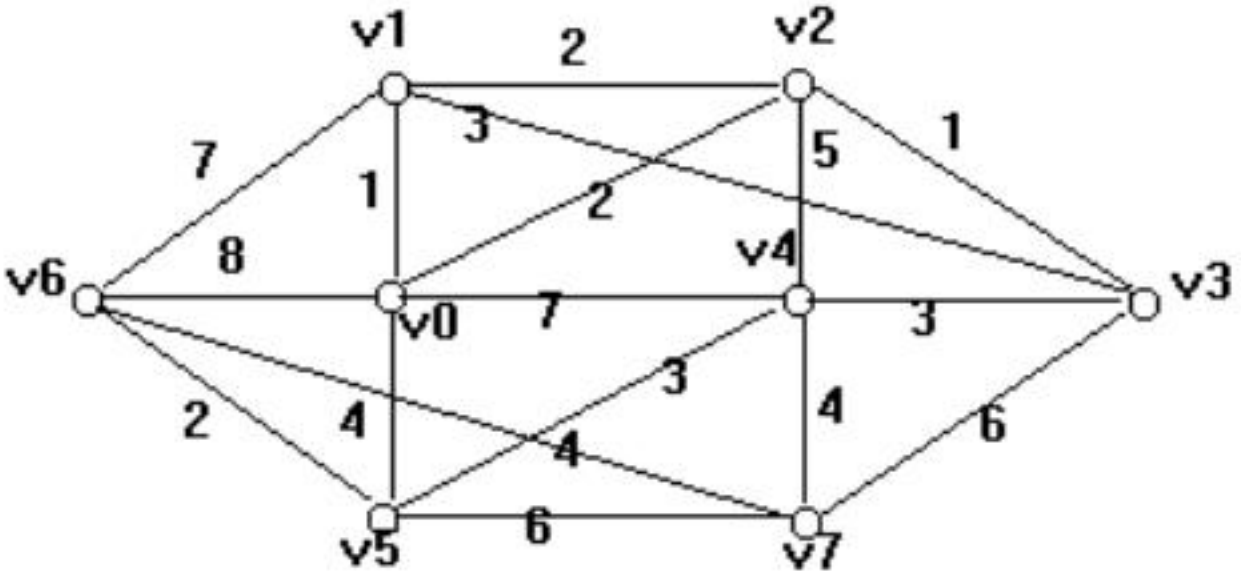
03 案例实现

# 11.1 最短路径

交通的**最短路径**选择。

# 11.1 最短路径

8个城市 $v_0$，$v_1$，…，$v_7$之间有一个公路网(如图所示)，每条公路为图中的边，边上的权数表示通过该公路所需的时间．设你处在城市 $v_0$，那么从 $v_0$到其他各城市，应选择什么路径使所需的时间最短？



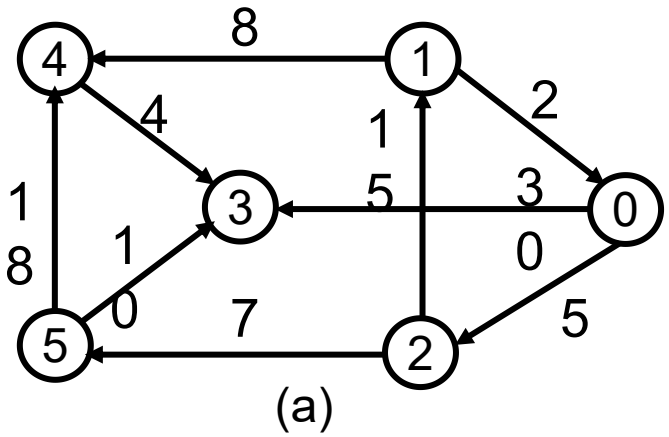
# 11.1 最短路径

- 最短路径问题：如果从图中某一顶点(称为源点)到达另一顶点(称为终点)的路径可能不止一条，如何找到一条路径，使得沿此路径各边上的权值总和达到最小。
- 问题解法：
  1. 权值为非负的单源最短路径问题(固定源点)—Dijkstra算法(迪杰斯特拉算法)；
  2. 权值为任意值的单源最短路径问题(固定源点) —Bellman-Ford算法(贝尔曼—福特算法)；
  3. 所有顶点之间的最短路径问题—Floyd-Warshall算法(弗洛伊德算法)；

# 11.1.1 Dijkstra算法

## 权值为非负的单源最短路径问题

- 问题的提出： 给定一个带权有向图G与源点v，求从v到G中其它顶点的最短路径。限定各边上的权值大于或等于0。



(a)

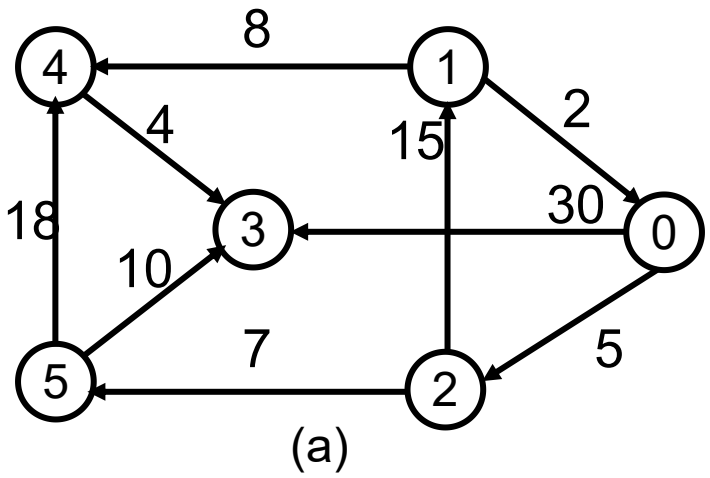
在图(a)中，考虑顶点0到其他顶点的最短距离是多少？

顶点0到顶点1的最短路径距离是：20
顶点0到顶点2的最短路径距离是： 5
顶点0到顶点3的最短路径距离是： 22
顶点0到顶点4的最短路径距离是： 28
顶点0到顶点5的最短路径距离是： 12

思考：这些最短距离是怎么求出来的？

# 11.1.1 Dijkstra算法

- 为求得这些最短路径，Dijkstra提出按路径长度的递增次序，逐步产生最短路径的算法。首先求出长度最短的一条最短路径，再参照它求出长度次短的一条最短路径，依次类推，直到从顶点v到其它各顶点的最短路径全部求出为止。



(a)

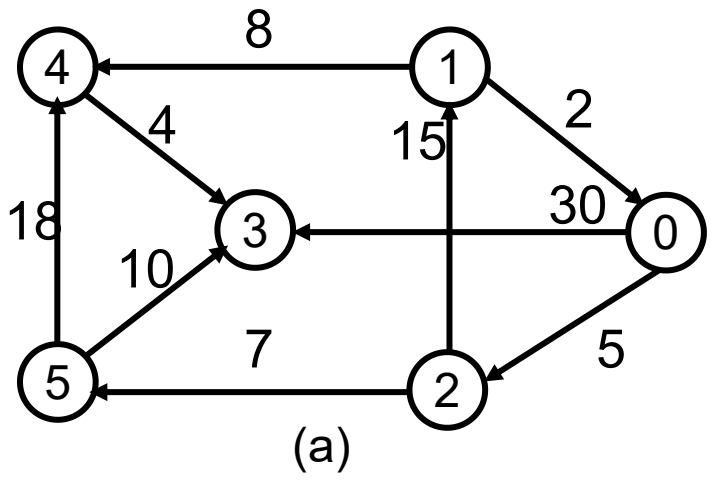
在图(a)中，考虑顶点0到其他顶点的最短距离是多少？

长度最短的最短路径是顶点0到顶点2的最短路径(就是顶点0到其他顶点的直接路径最短的路径)

长度次短的最短路径是顶点0到顶点5的最短路径$(v_0,v_2,v_5)$，其中$(v_0,v_2)$就是前面求出的长度最短的最短路径。

# 11.1.1 Dijkstra算法

- 为求得这些最短路径，Dijkstra提出**按路径长度的递增次序，逐步产生最短路径**的算法。首先求出长度最短的一条最短路径，再参照它求出长度次短的一条最短路径，依次类推，直到从顶点v到其它各顶点的最短路径全部求出为止。

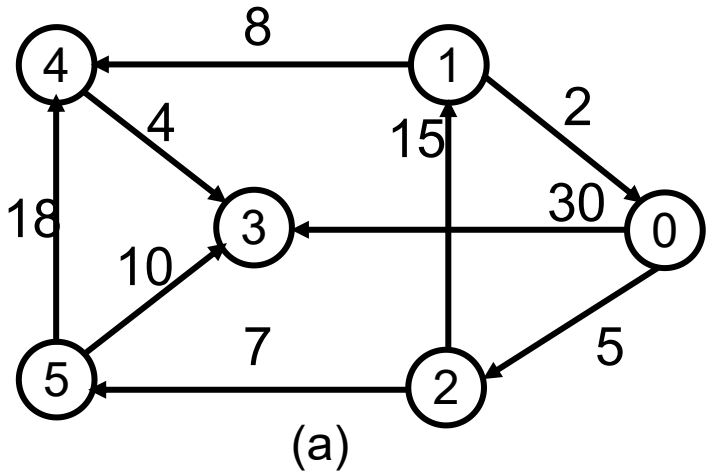


(a)

$$\begin{bmatrix} 0 & \infty & 5 & 30 & \infty & \infty \\ 2 & 0 & \infty & \infty & 8 & \infty \\ \infty & 15 & 0 & \infty & \infty & 7 \\ \infty & \infty & \infty & 0 & \infty & \infty \\ \infty & \infty & \infty & 4 & 0 & \infty \\ \infty & \infty & \infty & 10 & 18 & 0 \end{bmatrix}$$

# 11.1.1 Dijkstra算法的基本思想

1. 设置两个顶点的集合T和S：
   a. **S中存放已找到最短路径的顶点，初始状态时，集合S中只有一个顶点，即源点$v_0$；**
   b. **T中存放当前还未找到最短路径的顶点；**
2. 以后每求得一条最短路径$(v_0,\ldots,v_k)$，就将$v_k$加入到顶点集合S中，直到所有的顶点都加入到集合S中，算法就结束了。

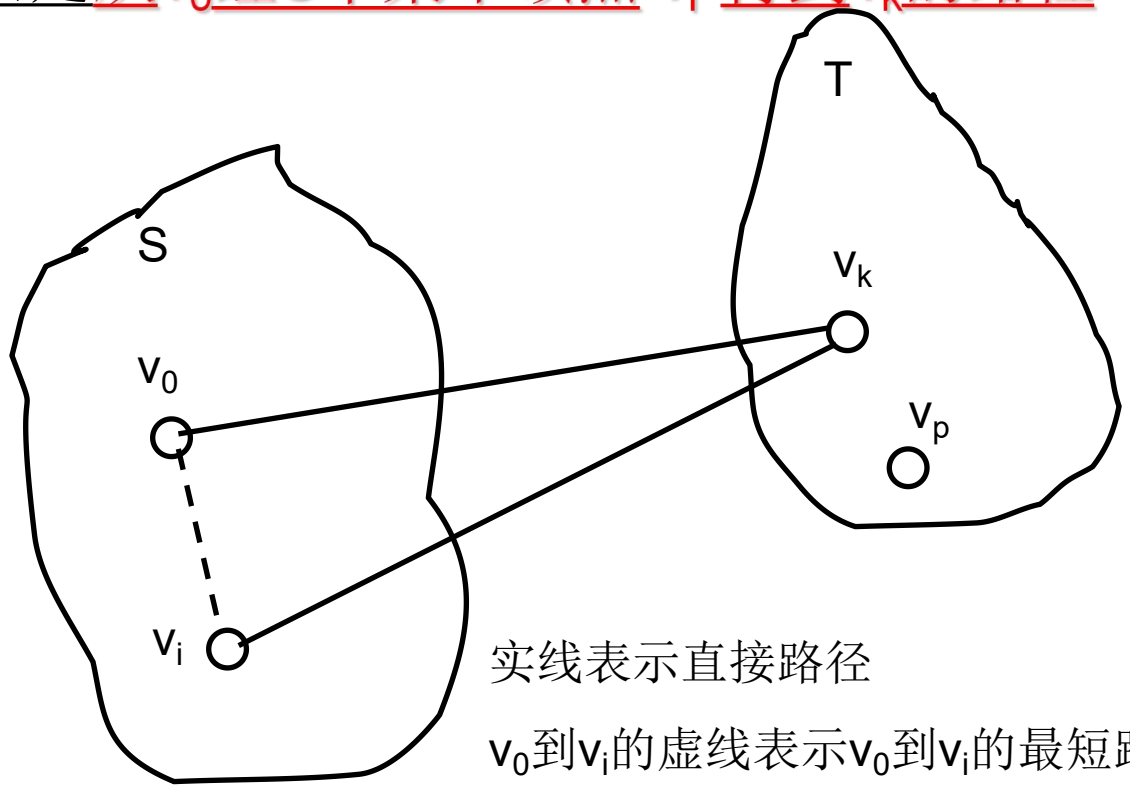


(a)

$$\begin{bmatrix} 0 & \infty & 5 & 30 & \infty & \infty \\ 2 & 0 & \infty & \infty & 8 & \infty \\ \infty & 15 & 0 & \infty & \infty & 7 \\ \infty & \infty & \infty & 0 & \infty & \infty \\ \infty & \infty & \infty & 4 & 0 & \infty \\ \infty & \infty & \infty & 10 & 18 & 0 \end{bmatrix}$$

# 11.1.1 Dijkstra算法的基本思想

- 可以证明$v_0$到T中顶点$v_k$的最短路径，要么是从$v_0$到$v_k$的直接路径；要么是从$v_0$经S中某个顶点 $v_i$ 再到$v_k$的路径。




实线表示直接路径

$v_0$到$v_i$的虚线表示$v_0$到$v_i$的最短路径

# 算法的具体步骤

1. 初始时，S中包含源点$v_0$。
2. 然后不断从T中选取到顶点$v_0$路径长度最短的顶点u，找到后：
   a) **把顶点u加入到S；**
   b) **修改顶点$v_0$到T中剩余顶点的最短路径长度值。T中各顶点新的最短路径长度值为：min{原来的最短路径长度值，顶点u的最短路径长度+u到该顶点的路径长度值}；**
3. 重复2，直到T的顶点全部加入S为止。

# 算法的具体步骤

在dijkstra算法里设置2个数组：

1) dist[n]：dist[i]表示当前找到的从源点$v_0$到终点$v_i$的最短路径的长度，初始状态下，dist[i]为$E[v_0][i]$，即邻接矩阵的第$v_0$行。
2) S[n]：S[i]为0表示顶点$v_i$还未加入到集合S中，S[i]为1表示$v_i$已经加入到集合S中。初始状态下，$S[v_0]$为1，其余为0，表示最初集合S中只有顶点$v_0$。

# 算法的具体步骤

在dijkstra算法里重复做以下工作：

1) 在dist[ ]里查找S[i]!=1，并且dist[i]最小的顶点u，
2) 将S[u]改为1，表示顶点u已经加入进来了。
3) 修改其它顶点的dist[ ]。当S[k]!=1，且顶点$v_k$到顶点$v_u$有边(E[u][k]<MAX)，且dist[u]+E[u][k]<dist[k]，则修改dist[k]为dist[u]+E[u][k]。

递推公式：

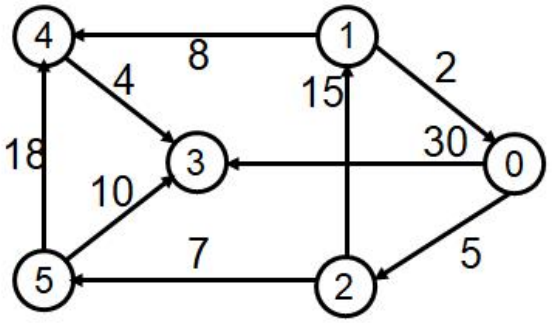
初始：*dist* [k] = Edge[$v_0$][k]，$v_0$是源点

递推：*dist* [k] = min{ *dist* [k], *dist* [u] + Edge[u][k] }, $v_k \in T$

其中$v_u$是当前dist[ ]最小的顶点。

# 算法的具体步骤

源点为0




$$\begin{bmatrix} 0 & \infty & 5 & 30 & \infty & \infty \\ 2 & 0 & \infty & \infty & 8 & \infty \\ \infty & 15 & 0 & \infty & \infty & 7 \\ \infty & \infty & \infty & 0 & \infty & \infty \\ \infty & \infty & \infty & 4 & 0 & \infty \\ \infty & \infty & \infty & 10 & 18 & 0 \end{bmatrix}$$

**1. 在T选取dist[ ]最小的顶点u**

**2. 将u加入到S**

**3. 修改T中顶点的dist[ ]**

初始状态

| | S | dist |
|---|---|---|
| 0 | 1 | 0 |
| 1 | 0 | ∞ |
| 2 | 0 | 5 |
| 3 | 0 | 30 |
| 4 | 0 | ∞ |
| 5 | 0 | ∞ |

(1)

| | S | dist |
|---|---|---|
| 0 | 1 | 0 |
| 1 | 0 | 20 |
| 2 | 1 | 5 |
| 3 | 0 | 30 |
| 4 | 0 | ∞ |
| 5 | 0 | 12 |

(2)

| | S | dist |
|---|---|---|
| 0 | 1 | 0 |
| 1 | 0 | 20 |
| 2 | 1 | 5 |
| 3 | 0 | 22 |
| 4 | 0 | 30 |
| 5 | 1 | 12 |

(3)

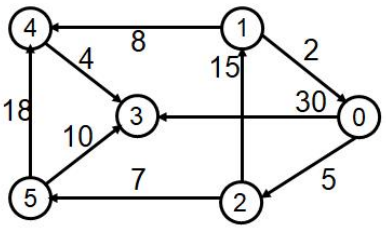
| | S | dist |
|---|---|---|
| 0 | 1 | 0 |
| 1 | 1 | 20 |
| 2 | 1 | 5 |
| 3 | 0 | 22 |
| 4 | 0 | 28 |
| 5 | 1 | 12 |

(4)

| | S | dist |
|---|---|---|
| 0 | 1 | 0 |
| 1 | 1 | 20 |
| 2 | 1 | 5 |
| 3 | 1 | 22 |
| 4 | 1 | 28 |
| 5 | 1 | 12 |

(5)

# 算法的具体步骤




$$\begin{bmatrix} 0 & \infty & 5 & 30 & \infty & \infty \\ 2 & 0 & \infty & \infty & 8 & \infty \\ \infty & 15 & 0 & \infty & \infty & 7 \\ \infty & \infty & \infty & 0 & \infty & \infty \\ \infty & \infty & \infty & 4 & 0 & \infty \\ \infty & \infty & \infty & 10 & 18 & 0 \end{bmatrix}$$

初始状态

| | S | dist | path |
|---|---|---|---|
| 0 | 1 | 0 | -1 |
| 1 | 0 | ∞ | -1 |
| 2 | 0 | 5 | 0 |
| 3 | 0 | 30 | 0 |
| 4 | 0 | ∞ | -1 |
| 5 | 0 | ∞ | -1 |

(1)

| | S | dist | path |
|---|---|---|---|
| 0 | 1 | 0 | -1 |
| 1 | 0 | **20** | **2** |
| 2 | 1 | ***5*** | 0 |
| 3 | 0 | 30 | 0 |
| 4 | 0 | ∞ | -1 |
| 5 | 0 | **12** | **2** |

(2)

| | S | dist | path |
|---|---|---|---|
| 0 | 1 | 0 | -1 |
| 1 | 0 | 20 | 2 |
| 2 | 1 | 5 | 0 |
| 3 | 0 | **22** | **5** |
| 4 | 0 | **30** | **5** |
| 5 | 1 | ***12*** | 2 |

(3)

| | S | dist | path |
|---|---|---|---|
| 0 | 1 | 0 | -1 |
| 1 | 1 | ***20*** | 2 |
| 2 | 1 | 5 | 0 |
| 3 | 0 | 22 | 5 |
| 4 | 0 | **28** | **1** |
| 5 | 1 | 12 | 2 |

(4)

| | S | dist | path |
|---|---|---|---|
| 0 | 1 | 0 | -1 |
| 1 | 1 | 20 | 2 |
| 2 | 1 | 5 | 0 |
| 3 | 1 | ***22*** | 5 |
| 4 | 0 | 28 | 1 |
| 5 | 1 | 12 | 2 |

(5)

| | S | dist | path |
|---|---|---|---|
| 0 | 1 | 0 | -1 |
| 1 | 1 | 20 | 2 |
| 2 | 1 | 5 | 0 |
| 3 | 1 | 22 | 5 |
| 4 | 1 | ***28*** | 1 |
| 5 | 1 | 12 | 2 |

# 算法的具体代码

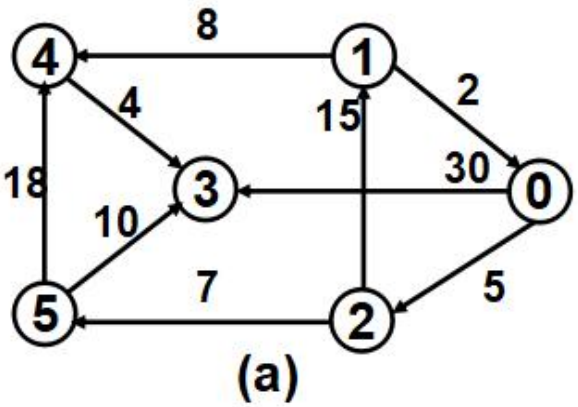
该算法可以实现的功能为：指定一个源点，指定一个终点，求源点到终点的最短路径。

```
#define max_v_num 10
#define max 1000000
int Edge[max_v_num][max_v_num];
int vexnum;
/*
dist[i]表示当前找到的从源点v0到终点vi的最短路径的长度，
初始状态下，dist[i]为E[v0][i]，即邻接矩阵的第v0行。
S[i]为0表示顶点vi还未加入到集合S中，S[i]为1表示vi已经加入到集合S中。
初始状态下，S[v0]为1，其余为0，表示最初集合S中只有顶点v0。
*/
```

# 算法的具体代码

```
13 void Dijistra(int v, int end){//v 源点 end终点
14     int dis[max_v_num],S[max_v_num];
15     int i,j,k;
16     for(i=0;i<vexnum;i++){
17         dis[i]=Edge[v][i];//初始化dist[i]
18         S[i]=0;//让所有的顶点归到一个初始集合T
19     }
20     S[v]=1,dis[v]=0;//把源点v加入到集合S中，初始化路径数组
21
22     for(i=0;i<vexnum-1;i++){//寻路除源点外的其他顶点
23         int min=max,u=v;
24         for(j=0;j<vexnum;j++){
25             if(S[j]==0&&dis[j]<min){u=j;min=dis[j];}
26         }//1 找到当前的最短路径
27
28         S[u]=1;//2 把找到的顶点加入到集合S中
29
30         for(k=0;k<vexnum;k++){//u点加入到集合T后,多出了新的路径,
31             //即通过u点可以达到新的顶点，更新dis[]数组
32             if(S[k]==0&&Edge[u][k]<max&&dis[u]+Edge[u][k]<dis[k]){
33                 dis[k]=dis[u]+Edge[u][k];
34             }
35         }
36     }
37     cout<<dis[end];
38 }
```

# 1 [3333]: 迪杰斯特拉




$$\begin{bmatrix} 0 & \infty & 5 & 30 & \infty & \infty \\ 2 & 0 & \infty & \infty & 8 & \infty \\ \infty & 15 & 0 & \infty & \infty & 7 \\ \infty & \infty & \infty & 0 & \infty & \infty \\ \infty & \infty & \infty & 4 & 0 & \infty \\ \infty & \infty & \infty & 10 & 18 & 0 \end{bmatrix}$$

输入顶点数n 边数m，m条边的顶点和权值；输出：顶点0到每个顶点的最短路径

样例输入

```
6 9
0 2 5
0 3 30
1 0 2
1 4 8
2 1 15
2 5 7
4 3 4
5 3 10
5 4 18
0 4
```

样例输出

```
28
```

```
34 int main(){
35     int i,j;
36     int n,m;//顶点数和边数
37     int u,v,w;//边的起始点和改边的权值
38     int x,y;//源点和终点
39     cin>>n>>m;
40     vexnum=n;
41     for(i=0;i<n;i++)
42         for(j=0;j<n;j++)
43             if(i==j)Edge[i][j]=0;
44             else Edge[i][j]=max;
45     for(i=0;i<m;i++){
46         cin>>u>>v>>w;
47         Edge[u][v]=w;
48     }
49     cin>>x>>y;
50     Dijistra(x,y);
51     return 0;
52 }
```

```
1  #include<iostream>
2  using namespace std;
3  #define max_v_num 10
4  #define max 1000000
5  int Edge[max_v_num][max_v_num];
6  int vexnum;
7  /*
8  dist[i]表示当前找到的从源点v0到终点vi的最短路径的长度，
9  初始状态下，dist[i]为E[v0][i]，即邻接矩阵的第v0行。
10 S[i]为0表示顶点vi还未加入到集合S中，S[i]为1表示vi已经加入到集合S中。
11 初始状态下，S[v0]为1，其余为0，表示最初集合S中只有顶点v0。
12 */
```

```
13 void Dijistra(int v, int end){//v 源点 end终点
14     int dis[max_v_num],S[max_v_num];
15     int i,j,k;
16     for(i=0;i<vexnum;i++){
17         dis[i]=Edge[v][i];//初始化dist[i]
18         S[i]=0;//让所有的顶点归到一个初始集合T
19     }
20     S[v]=1,dis[v]=0;//把源点v加入到集合S中，初始化路径数组
21
22     for(i=0;i<vexnum-1;i++){//寻路除源点外的其他顶点
23         int min=max,u=v;
24         for(j=0;j<vexnum;j++){
25             if(S[j]==0&&dis[j]<min){u=j;min=dis[j];}
26         }//1 找到当前的最短路径
27
28         S[u]=1;//2 把找到的顶点加入到集合S中
29
30         for(k=0;k<vexnum;k++){//u点加入到集合T后,多出了新的路径,
31             //即通过u点可以达到新的顶点，更新dis[]数组
32             if(S[k]==0&&Edge[u][k]<max&&dis[u]+Edge[u][k]<dis[k]){
33                 dis[k]=dis[u]+Edge[u][k];
34             }
35         }
36     }
37     cout<<dis[end];
38 }
```

# 比较： Prim和Dijkstra的区别

Prim和Dijkstra大同小异，唯一的区别是prim是维护树到其余节点的最小路径，而dijkstra是维护源节点到各个节点的最小路径。

# 比较：最小生成树和最短路径的区别


最小生成树：

最小生成树能够保证整个拓扑图的所有路径之和最小，但不能保证任意两点之间是最短路径。

最短路径：

最短路径是从一点出发，到达目的地的路径最小。

总结：

- 遇到求所有路径之和最小的问题用最小生成树&并查集解决；
- 遇到求两点间最短路径问题的用最短路，即从一个城市到另一个城市最短的路径问题。

区别：

最小生成树构成后所有的点都被连通，而最短路只要到达目的地走的是最短的路径即可，与所有的点连不连通没有关系。

# 算法复杂度分析

用邻接矩阵存储图，Dijkstra算法的复杂度为$O(n^2)$

能不能降低时间复杂度？

采用邻接表+优先队列

独立思考与判断

# 11.2 优先队列

普通的队列是一种先进先出的数据结构，元素在队列尾追加，而从队列头删除。

在优先队列中，元素被赋予优先级。当访问元素时，具有最高优先级的元素最先删除。优先队列具有最高级先出 （first in, largest out）的行为特征。

# 11.2.1 优先队列用法

**1.头文件**

**#include<queue>//与队列相同，不必引入vector的头文件**

**2.定义方式**

**priority_queue<int> p;//默认，最大值优先，是大顶堆一种简写方式**
**priority_queue<int,vector<int>,greater<int>>q1;//最小值优先，小顶堆**
**priority_queue<int,vector<int>,less<int> >q2;//最大值优先，大顶堆**

```
//对于基础类型 默认是大顶堆
priority_queue<int> a;
//等同于 priority_queue<int, vector<int>, less<int> > a;

// 这里一定要有空格，不然成了右移运算符↓ ↓
priority_queue<int, vector<int>, greater<int> > c;  //这样就是小顶堆
priority_queue<string> b;
```

**其中第一个参数是数据类型，第二个参数为容器类型。第三个参数为比较函数。**

```cpp
#include<iostream>
#include <queue>
using namespace std;
int main()
{
    //对于基础类型 默认是大顶堆
    priority_queue<int> a;
    //等同于 priority_queue<int, vector<int>, less<int> > a;

    // 这里一定要有空格，不然成了右移运算符↓ ↓
    priority_queue<int, vector<int>, greater<int> > c;  //这样就是小顶堆
    priority_queue<string> b;

    for (int i = 0; i < 5; i++){
        a.push(i);
        c.push(i);
    }
    while (!a.empty())
    {
        cout << a.top() << ' ';
        a.pop();
    }
    cout << endl;

    while (!c.empty())
    {
        cout << c.top() << ' ';
        c.pop();
    }
    cout << endl;

    b.push("abc");
    b.push("abcd");
    b.push("cbd");
    while (!b.empty())
    {
        cout << b.top() << ' ';
        b.pop();
    }
    cout << endl;
    return 0;
}
```

```
4 3 2 1 0
0 1 2 3 4
cbd abcd abc
```

# 11.2.1 优先队列用法

```
struct node
{
        int dis, num;
};
priority_queue<node>que;
bool operator<(const node &a, const node &b)
{
        return a.dis > b.dis;
}//按照dis从小到大排序
```

# 11.2.1 优先队列用法

```
struct node
{
    string name;
    int price;
    friend bool operator< (node a, node b)
    {
        return a.price < b.price;
      // 相当于less,这是大顶堆，反之则是小顶堆，最大值优先
    }
} stu; //定义结构体变量
```

这样直接可以：

```
priority_queue<node > q;
```

# 11.2.1 优先队列用法

q.push(x) //将x加入队列中，即入队操作

q.pop() //出队操作(删除队列首元素)，只是出队，没有返回值

q.top() //返回第一个元素(队首元素)优先队列的队首用top

q.size() //返回栈队列中的元素个数

q.empty() //当队列为空时，返回 true

# 11.2.3 Vector介绍

在使用数组的时候，必须指定数组的长度，一旦配置了就不能改变了，通常我们的做法是尽量配置一个大的空间，以免不够用，这样做的缺点是比较浪费空间，预估空间不当会引起很多不便。

STL实现了一个Vector容器，该容器就是来改善数组的缺点。vector是一个动态空间，随着元素的加入，它的内部机制会自行扩充以容纳新元素。因此，vector的运用对于内存的合理利用与运用的灵活性有很大的帮助，再也不必因为害怕空间不足而一开始就配置一个大容量数组了，vector是用多少就分配多少。

# 11.2.3 Vector介绍

使用vector容器之前必须加上<vector>头文件：#include<vector>;
vector成员函数

c.push_back(elem)在尾部插入一个elem数据。

```
vector<int> v;
v.push_back(1);
```

c.pop_back()删除末尾的数据。

```
vector<int> v;
v.pop_back();
```

c.assign(beg,end)将[beg,end)一个左闭右开区间的数据赋值给c。

# 11.2.3 Vector介绍

clear() 清空当前的vector
max_size() 得到vector最大可以是多大
empty() 判断vector是否为空
size() 当前使用数据的大小

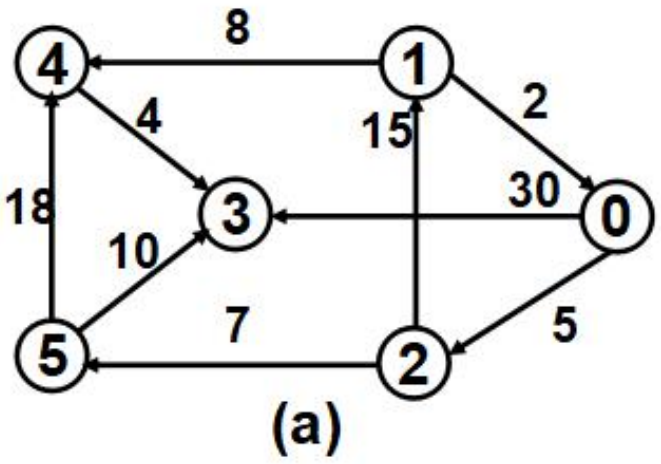
at 得到编号位置的数据
begin 得到数组头的指针
end 得到数组的最后一个单元+1的指针
front 得到数组头的引用
back 得到数组的最后一个单元的引用

capacity 当前vector分配的大小
resize 改变当前使用数据的大小，如果它比当前使用的大，者填充默认值
reserve 改变当前vecotr所分配空间的大小
erase 删除指针指向的数据项

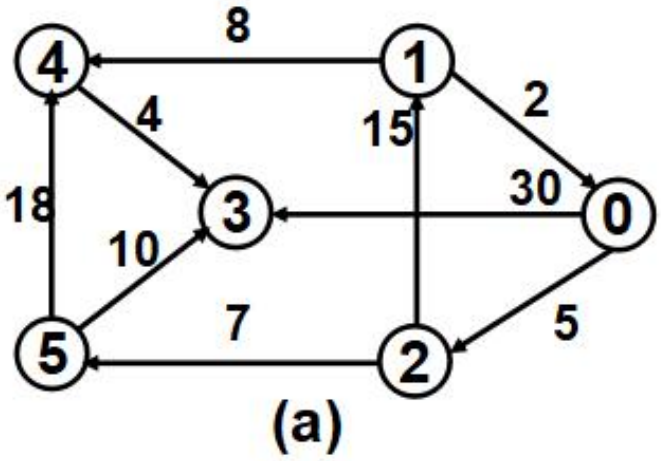
# 11.2.4 [3333]: 迪杰斯特拉

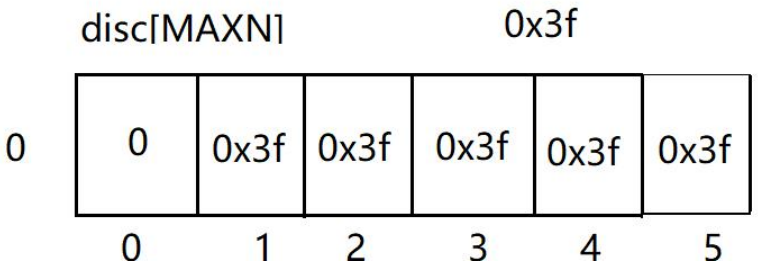
如图，求最短路径。

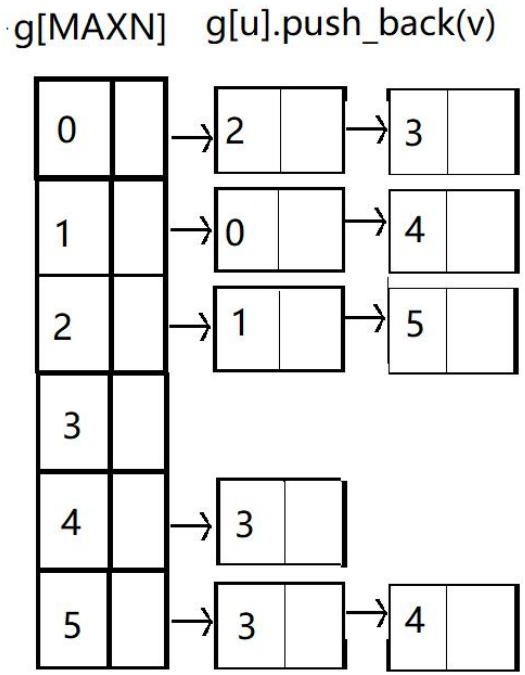
输入
顶点数n 边数m
m条边的顶点和权值
某两个顶点
输出
顶点0到每个顶点的最短路径
样例输入
6 9
0 2 5
0 3 30
1 0 2
1 4 8
2 1 15
2 5 7
4 3 4
5 3 10
5 4 18
0 4
样例输出
28

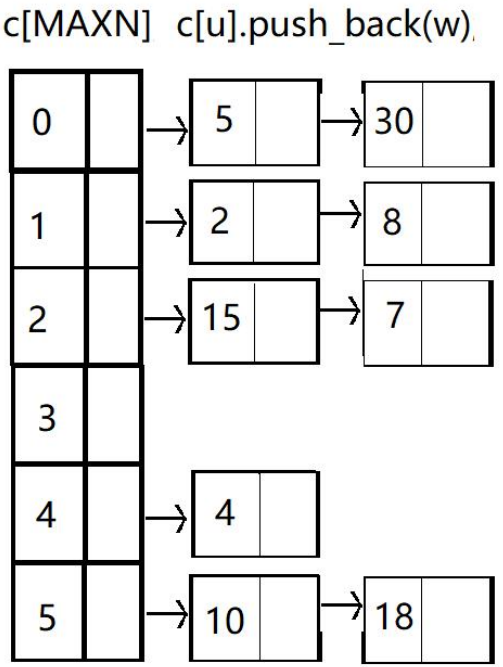


# 11.2.4.1 建立邻接表


8
4
1
4
15
2
18
3
30
0
10
5
7
2
5
(a)


disc[MAXN]
0x3f
0
0
0x3f
0x3f
0x3f
0x3f
0x3f
0
1
2
3
4
5


g[MAXN]
g[u].push_back(v)
0
2
3
1
0
4
2
1
5
3
4
3
5
3
4

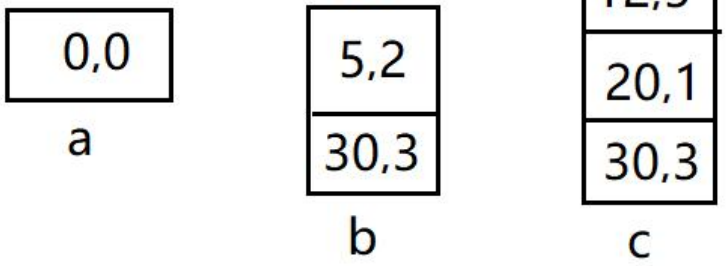

c[MAXN]
c[u].push_back(w)
0
5
30
1
2
8
2
15
7
3
4
4
5
10
18

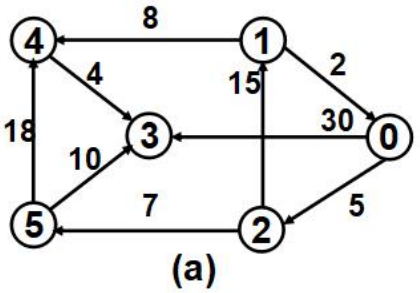
# 11.2.4.2 建立优先队列

```
#include<bits/stdc++.h>
using namespace std;
const int inf=0xffffff;
const int MAXN=100;
int edge[MAXN][MAXN];//边信息
int disc[MAXN];//保存源点到各点的最短距离
vector<int>g[MAXN], c[MAXN];
struct node
{
    int dis;//dis表示源点到当前点num的距离
    int num; //编号
};
priority_queue<node>que;//优先队列
bool operator<(const node &a, const node &b)
{
    return a.dis > b.dis;//按照dis从小到大排
}
```

# 11.2.4.2 建立优先队列

1. 初试队列为空,源点 v0 入队




2. 从 V0 出发，走 V2 、V5。将 V2,V5 入队，根据优先队列原则，dis 最小的处于队头，同时更新 disc[]数组。

disc[MAXN]

| | 0 | 0x3f | 5 | 30 | 0x3f | 0x3f |
|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 2 | 3 | 4 | 5 |

3. 弹出 v2，从 V2 出发，走 V1,V5,更新优先队列如 c 所示，同时更新 disc[]数组

disc[MAXN]

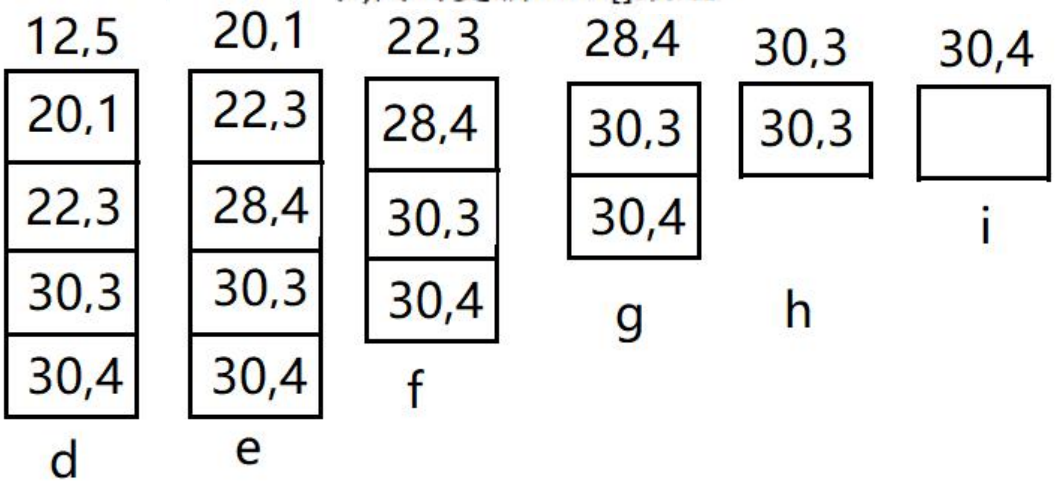
| | 0 | 20 | 5 | 30 | 0x3f | 12 |
|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 2 | 3 | 4 | 5 |

# 11.2.4.2 建立优先队列

4. 弹出 v5，从 V5 出发，走 V3, V4，更新优先队列如 d 所示,同时更新 disc[]数组

disc[MAXN]

| | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| 0 | 0 | 20 | 5 | 30 | 30 | 12 |

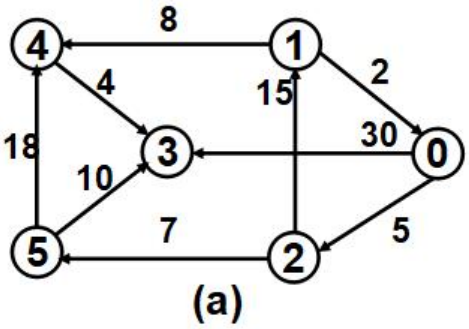



5. 弹出 v1, 从 v1 出发，走 V0,V4, 更新优先队列如 e 所示,同时更新 disc[]数组

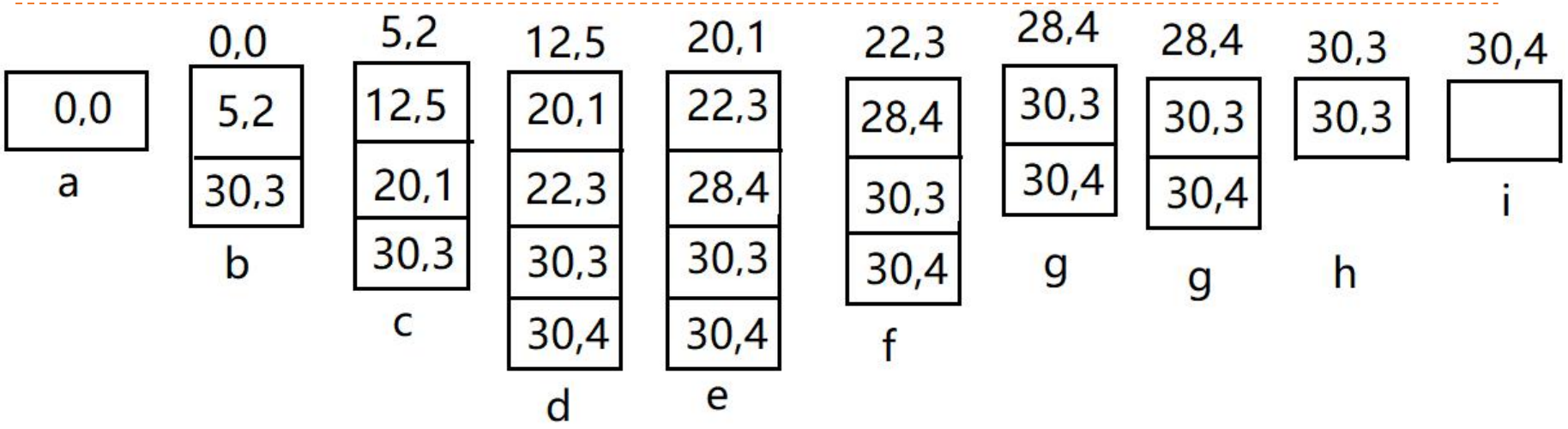
disc[MAXN]

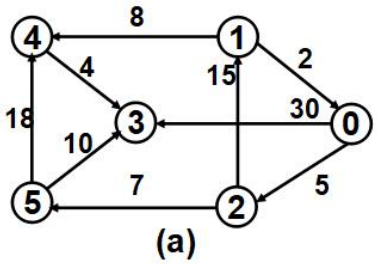
| | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| 0 | 0 | 20 | 5 | 30 | 28 | 12 |

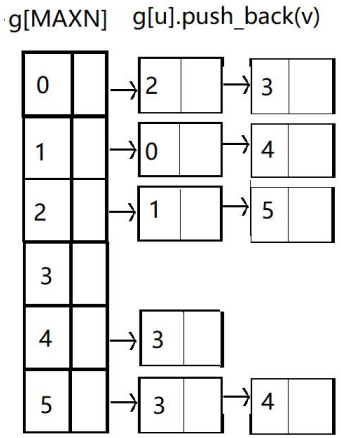
6. 弹出 V3,从 V3 出发，无路可走，更新优先队列如 f 所示。
7. 弹出 v4,从 v4 出发，disc[]没有更新,如图 g。
8. 到结束。

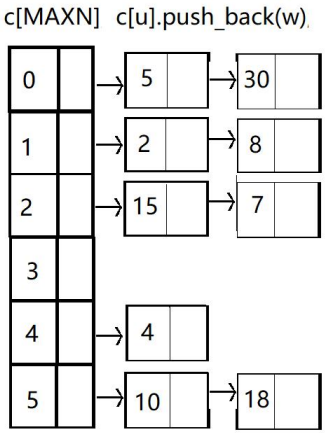


# 11.2.4.2 建立优先队列



# 11.2.4.3 代码

```
int main(){
    int n,m;
    int u,v,w;
    int st,ed;
    cin>>n>>m;
    //队尾添加元素：  vec.push_back();
    //队尾删除元素：  vec.pop_back();
    for(int i=0;i<m;i++){
        cin>>u>>v>>w;
        g[u].push_back(v);//u是邻接表出来的端点，uv是一条边
        c[u].push_back(w);//存权值
    }
    cin>>ed;
    Dijistra(0,ed);
    return 0;
}
```

# 11.2.4.3 代码

```
19  void Dijistra(int v, int end){
20      memset(disc, 0x3f, sizeof(disc));
21      disc[v] = 0;//初始化，源点到其他点初始距离为极大值
22      node tmp;
23      tmp.num = v; tmp.dis = 0;//初试源点
24      que.push(tmp);//源点入队
25      while(!que.empty())
26      {
27
28          node now = que.top();//得到dis最小点
29          que.pop();
30          if(now.dis != disc[now.num])
31              continue;//disc数组中存在的最短距离，如果取出来的边不是最小的就
32          for(int i = g[now.num].size() - 1; i >= 0; i --)
33          {//遍历邻接表
34              int to = g[now.num][i];
35              if(disc[to] > disc[now.num] + c[now.num][i])
36              {
37                  disc[to] = disc[now.num] + c[now.num][i];
38                  que.push(node{disc[to], to});
39              }
40          }
41      }
42      cout << disc[end] << endl;
43  }
```

# 11.2.4.3 代码

```
19 void Dijistra(int v, int end){
20     memset(disc, 0x3f, sizeof(disc));
21     disc[v] = 0;//初始化，源点到其他点初始距离为极大值
22     node tmp;
23     tmp.num = v; tmp.dis = 0;//初始源点
24     ____________________//源点入队
25     while(!que.empty())
26     {
27
28         node now = ____________________;//得到dis最小点
29         que.pop();
30         if(now.dis != disc[now.num])
31             continue;//disc数组中存在的最短距离，如果取出来的边不是最小的就
32         for(_____________________________)
33         {//遍历邻接表
34             int to = g[now.num][i];
35             if(_____________________________)
36             {
37                 disc[to] = _____________________________;
38                 que.push(node{disc[to], to});
39             }
40         }
41     }
42     cout << disc[end] << endl;
43 }
```

# 11.3 2 [2901]:香甜的黄油

农夫John发现做出全威斯康辛州最甜的黄油的方法：糖。把糖放在一片牧场上，他知道N（1≤N≤500）只奶牛会过来舔它，这样就能做出能卖好价钱的超甜黄油。当然，他将付出额外的费用在奶牛上。

农夫John很狡猾。像以前的巴甫洛夫，他知道他可以训练这些奶牛，让它们在听到铃声时去一个特定的牧场。他打算将糖放在那里然后下午发出铃声，以至他可以在晚上挤奶。

农夫John知道每只奶牛都在各自喜欢的牧场（一个牧场不一定只有一头牛）。给出各头牛在的牧场和牧场间的路线，找出使所有牛到达的路程和最短的牧场（他将把糖放在那）。

# 11.3 [2901]:香甜的黄油

输入

第一行：三个数：奶牛数N，牧场数P（2≤P≤800），牧场间道路数C(1≤C≤1450)。

第二行到第N+1行：1到N头奶牛所在的牧场号。第N+2行到第N+C+1行：每行有三个数：相连的牧场A、B，两牧场间距（1≤D≤255），当然，连接是双向的。

输出

一行 输出奶牛必须行走的最小的距离和。

样例输入

```
3 4 5
2
3
4
1 2 1
1 3 5
2 3 7
2 4 3
3 4 5
```

样例输出

```
8
```

分析：这是一道最短路的升级应用，根据题意， 我们需要找到一个距离所有牛最短的牧场。那么只需要枚举所有牧场为起点，求此时所有其他牧场到它的最短路之和即可。然后输出所有牧场中最短距离即为答案。

但节点数达到800，显然n^3的会超时，那么只能考虑优化，可以采用优先队+dijistra或者SPFA。

点的个数p = 800，边的个数1500
朴素版Dijkstra 复杂度是O($n^3$)
O($n^3$) $n^3$=5.12* $10^8$
堆优化版dijkstra 复杂度是O(nmlogn)
nmlogn=1.2* $10^7$
spfa 复杂度是O(nm)
O(nm) 平均是2到3倍即 3* nm=3.9* $10^6$

# 方法一：vector+优先队列

```
#include<bits/stdc++.h>
using namespace std;
const int maxn = 2e3 + 10;
vector<int>g[maxn], w[maxn];

struct node
{//dis表示起点到该点的距离，num表示该点的编号
    int dis, num;
};
priority_queue<node>que;
bool operator<(const node &a, const node &b)
{
    return a.dis > b.dis;
}
int n, p, c;
int vis[maxn];//vis表示这个区域里有几头奶牛
int disc[maxn];//表示每个点到起点的距离
```

```cpp
int main()
{
    scanf("%d %d %d", &n, &p, &c);
    //奶牛数N，牧场数P，牧场间道路数C
    for (int i = 1; i <=n; ++ i)
    {
        int x;
        scanf("%d", &x);//x区域的奶牛数量+1
        vis[x] ++;
    }
    while(c --)
    {
        int x, y, c;
            //相连的牧场A、B，两牧场间距
        scanf("%d %d %d", &x, &y, &c);
        //由于只说是有边，所以存双向边
        g[x].push_back(y);//u是邻接表出来的端点，uv是一条边
        w[x].push_back(c);//存权值
        g[y].push_back(x);
        w[y].push_back(c);
    }
    //枚举把黄油放在每一个牧场，求解，最后取一个极小值
    int ans = 1e9;
    for(int i = 1; i <= p; i ++)
        ans = min (ans, dijkstra(i));
    printf("%d\n", ans);
    return 0;
}
```

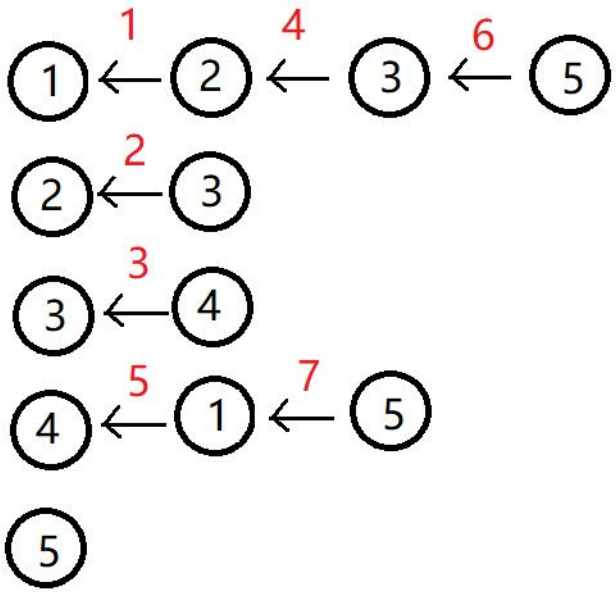
```
19  int dijkstra(int x)
20  {
21      memset(disc, 0x3f, sizeof(disc));
22      disc[x] = 0;//起点到起点的距离为0
23      que.push(node{0, x});    //起点入队
24      while(!que.empty())
25      {
26          node now = que.top();
27          que.pop();
28          if(now.dis != disc[now.num])//该点被取过了就continue
29              continue;
30          for(int i = g[now.num].size() - 1; i >= 0; i --)
31          {//遍历邻接表
32              int to = g[now.num][i];
33              if(disc[to] > disc[now.num] + w[now.num][i])
34              {
35                  disc[to] = disc[now.num] + w[now.num][i];
36                  que.push(node{disc[to], to});
37              }
38          }
39      }
40      int ans = 0;
41      //由于是求所有奶牛到该牧场的距离和最小，所以要用乘法
42      for(int i = 1; i <= p; i ++)
43          ans += disc[i] * vis[i];
44      return ans;
45  }
```

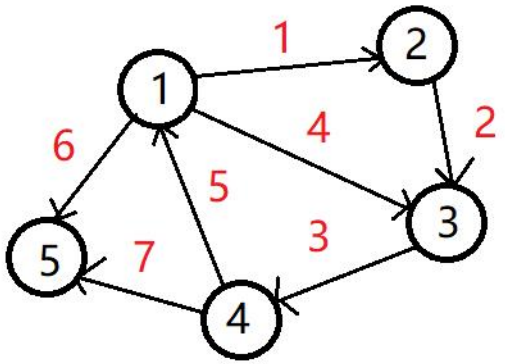
# 方法二：链式前向星+优先队列

前向星是以存储边的方式来存储图，先将边读入并存储在连续的数组中，然后按照边的起点进行排序，这样数组中起点相等的边就能够在数组中进行连续访问了。它的优点是实现简单，容易理解，缺点是需要在所有边都读入完毕的情况下对所有边进行一次排序，带来了时间开销。

链式前向星和邻接表类似，也是链式结构和线性结构的结合，每个结点i都有一个链表，链表的所有数据是从i出发的所有边的集合（对比邻接表存的是顶点集合），边的表示为一个四元组(u, v, w, next)，其中(u, v)代表该条边的有向顶点对，w代表边上的权值，next指向下一条边。

具体的，我们需要一个边的结构体数组 edge[MAXM]，MAXM表示边的总数，所有边都存储在这个结构体数组中，并且用head[i]来指向 i 结点的第一条边。




我们先对上面的7条边进行编号第一条边是1以此类推编号[1~7]，然后我们要知道两个变量的含义：

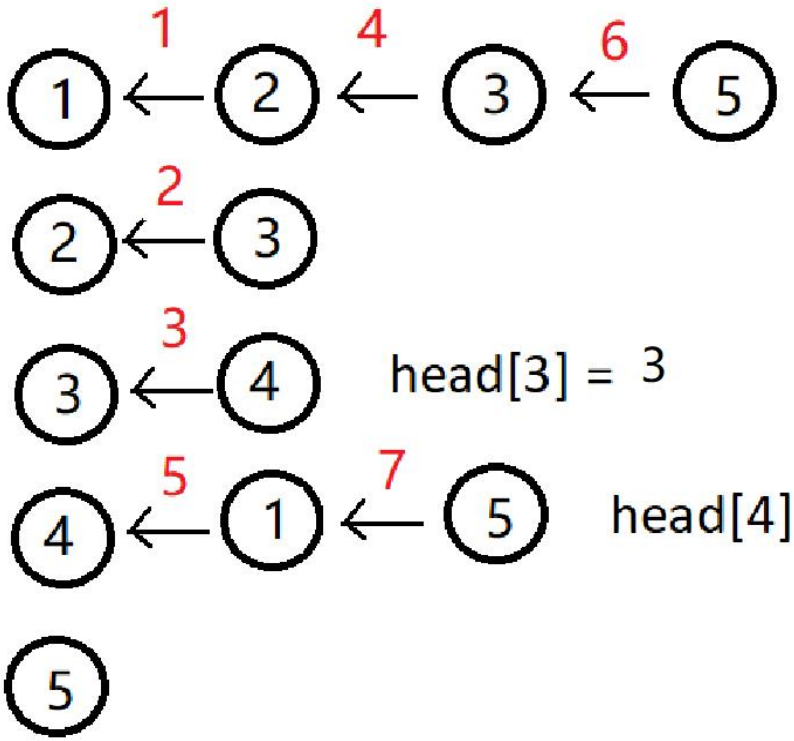
Next，表示与这个边起点相同的上一条边的编号。

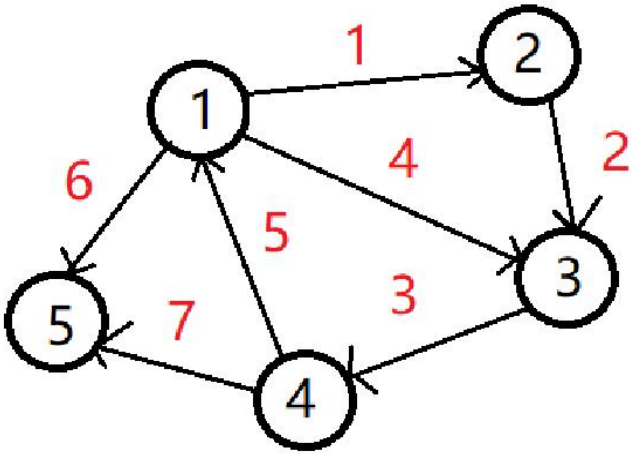
head[ i ]数组，表示以 i 为起点的最后一条边的编号。

head数组一般初始化为-1，

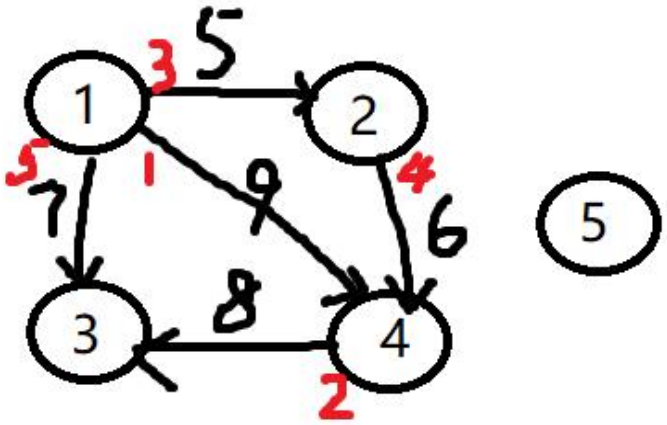
head[ i ]数组，表示以 i 为起点的最后一条边的编号。

Next，表示与这个边起点相同的上一条边的编号



# 进一步理解，画出过程图

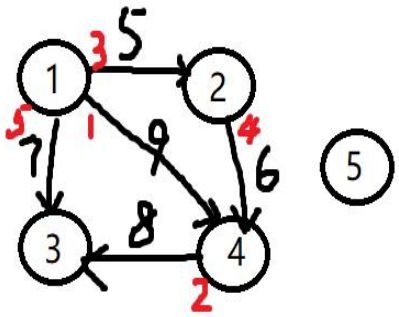



| | u | v | w |
|---|---|---|---|
| 1 | 1 | 4 | 9 |
| 2 | 4 | 3 | 8 |
| 3 | 1 | 2 | 5 |
| 4 | 2 | 4 | 6 |
| 5 | 1 | 3 | 7 |

# 读入第一条边

| | U | V | W |
|---|---|---|---|
| 1 | 1 | 4 | 9 |
| 2 | | | |
| 3 | | | |
| 4 | | | |
| 5 | | | |

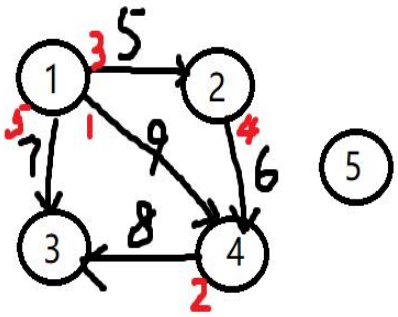
| | head | next |
|---|---|---|
| 1 | 1 | -1 |
| 2 | | |
| 3 | | |
| 4 | | |
| 5 | | |



# 读入第二条边

| | U | V | W |
|---|---|---|---|
| 1 | 1 | 4 | 9 |
| 2 | 4 | 3 | 8 |
| 3 | | | |
| 4 | | | |
| 5 | | | |

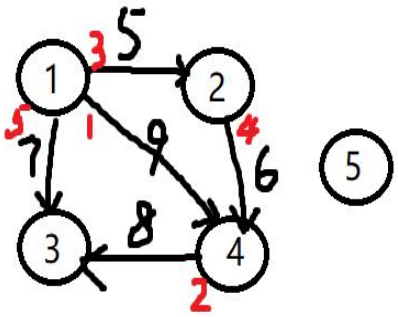
| | head | next |
|---|---|---|
| 1 | 1 | -1 |
| 2 | -1 | -1 |
| 3 | -1 | -1 |
| 4 | 2 | -1 |
| 5 | -1 | -1 |



# 读入第三条边

| | U | V | W |
|---|---|---|---|
| 1 | 1 | 4 | 9 |
| 2 | 4 | 3 | 8 |
| 3 | 1 | 2 | 5 |
| 4 | | | |
| 5 | | | |

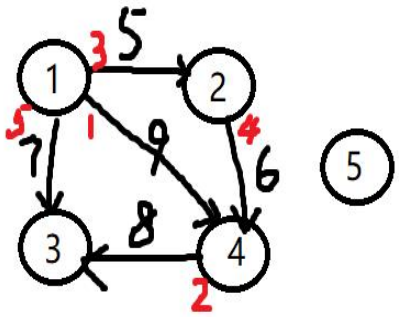
| | head | next |
|---|---|---|
| 1 | 3 | -1 |
| 2 | -1 | -1 |
| 3 | -1 | 1 |
| 4 | 2 | -1 |
| 5 | -1 | -1 |



# 读入第四条边

| | U | V | W |
|---|---|---|---|
| 1 | 1 | 4 | 9 |
| 2 | 4 | 3 | 8 |
| 3 | 1 | 2 | 5 |
| 4 | 2 | 4 | 6 |
| 5 | | | |

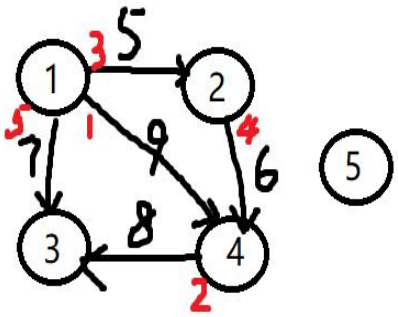
| | head | next |
|---|---|---|
| 1 | 3 | -1 |
| 2 | 4 | -1 |
| 3 | -1 | 1 |
| 4 | 2 | -1 |
| 5 | -1 | -1 |



# 读入第四条边

| | U | V | W |
|---|---|---|---|
| 1 | 1 | 4 | 9 |
| 2 | 4 | 3 | 8 |
| 3 | 1 | 2 | 5 |
| 4 | 2 | 4 | 6 |
| 5 | | | |

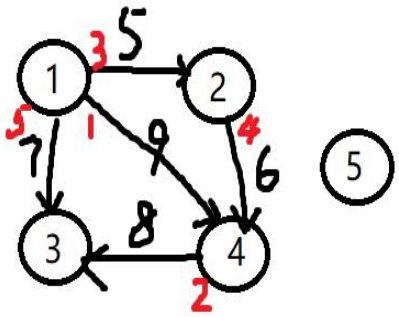
| | head | next |
|---|---|---|
| 1 | 3 | -1 |
| 2 | 4 | -1 |
| 3 | -1 | 1 |
| 4 | 2 | -1 |
| 5 | -1 | -1 |



# 读入第五条边

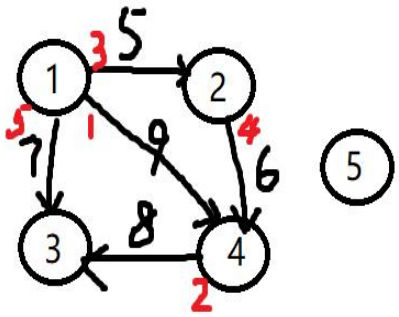
| | U | V | W |
|---|---|---|---|
| 1 | 1 | 4 | 9 |
| 2 | 4 | 3 | 8 |
| 3 | 1 | 2 | 5 |
| 4 | 2 | 4 | 6 |
| 5 | 1 | 3 | 7 |

| | head | next |
|---|---|---|
| 1 | 5 | -1 |
| 2 | 4 | -1 |
| 3 | -1 | 1 |
| 4 | 2 | -1 |
| 5 | -1 | 3 |

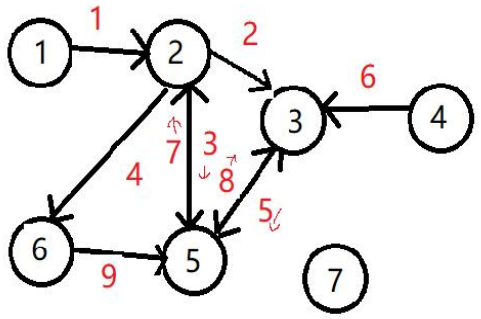


| | U | V | W |
|---|---|---|---|
| 1 | 1 | 4 | 9 |
| 2 | 4 | 3 | 8 |
| 3 | 1 | 2 | 5 |
| 4 | 2 | 4 | 6 |
| 5 | 1 | 3 | 7 |

| | head | next |
|---|---|---|
| 1 | 5 | -1 |
| 2 | 4 | -1 |
| 3 | -1 | 1 |
| 4 | 2 | -1 |
| 5 | -1 | 3 |



# 6.2.3.3 邻接表的实现-写法4（常用）




```
2 6
head[2]=4, next[4]=3
3 5
head[3]=5, next[5]=0
4 3
head[4]=6, next[6]=0
5 2
head[5]=7, next[7]=0
5 3
head[5]=8, next[8]=7
6 5
head[6]=9, next[9]=0
```

head值

| | |
|---|---|
| 1 | 1 |
| 2 | 4 |
| 3 | 5 |
| 4 | 6 |
| 5 | 8 |
| 6 | 9 |
| 7 | 0 |

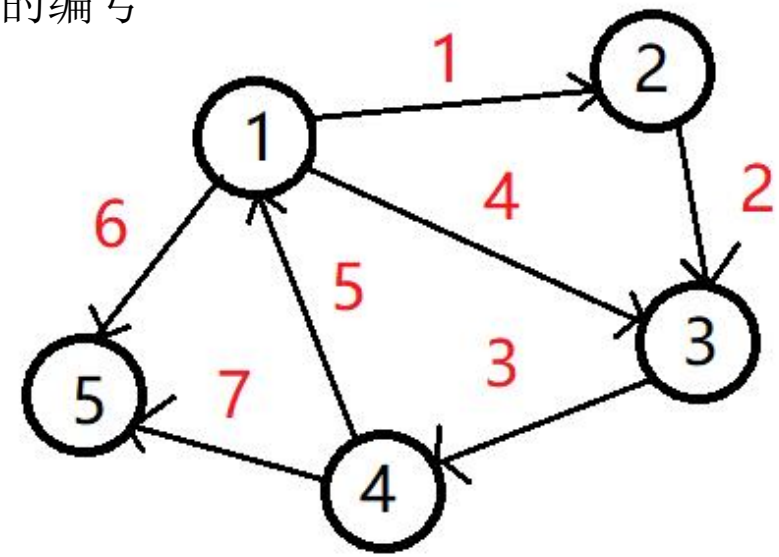
next 数组

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 2 | 3 | 0 | 0 | 0 | 7 | 0 |

```
for(int j=head1[i];j!=0;j=next1[j])
    start++;
```

```
void add(int f, int t){
    next1[cnt]=head1[f];//next指向下一条边
                        //(原头结点指向的边)
    head1[f]=cnt;//把当前边插入都头结点后面，头插法
    cnt++;
}
```

```
void add_edge(int u, int v, int w)//加边，u起点，v终点，w边权
{
   edge[cnt].to = v; //终点
   edge[cnt].w = w; //权值
   edge[cnt].next = head[u];
                        //以u为起点上一条边的编号，
                        //也就是与这个边起点相同的上一条边的编号
   head[u] = cnt++;//更新以u为起点上一条边的编号
}
```



```cpp
#include<bits/stdc++.h>
using namespace std;
const int maxn = 2e3 + 10;
//2-4行的变量都是正常的链式前向星数组
int Head[1000], Next[maxn * 2], w[maxn * 2], v[maxn * 2], num_edge;
//dis表示起点到该点的距离，num表示该点的编号
struct node
{
    int dis, num;
};
void ins(int x, int y, int c)
{//建边
    v[++ num_edge] = y;
    Next[num_edge] = Head[x];
    w[num_edge] = c;
    Head[x] = num_edge;
}
priority_queue<node>que;
bool operator<(const node &a, const node &b)
{
    return a.dis > b.dis;
}
int n, p, c;
int vis[maxn];//vis表示这个区域里有几头奶牛
//表示每个点到起点的距离
int disc[maxn];
```

```
61  int main()
62  {
63      memset(Head, -1, sizeof(Head));
64      scanf("%d %d %d", &n, &p, &c);
65      for (int i = 1; i <=n; ++ i)
66      {
67          int x;
68          scanf("%d", &x);
69          vis[x] ++;          //x区域的奶牛数量+1
70      }
71      while(c --)
72      {
73          int x, y, c;
74          scanf("%d %d %d", &x, &y, &c);
75          //由于只说是有边，所以存双向边
76          ins(x, y, c); ins(y, x, c);
77      }
78      //枚举把黄油放在每一个牧场，求解，最后取一个极小值
79      int ans = 1e9;
80      for(int i = 1; i <= p; i ++)
81      {
82          ans = min (ans, dijkstra(i));
83      }
84      printf("%d\n", ans);
85      return 0;
86  }
```

```
27  int dijkstra(int x)
28  {
29      memset(disc, 0x3f, sizeof(disc));
30      disc[x] = 0;//起点到起点的距离为0
31      que.push(node{0, x});    //起点入队
32      while(!que.empty())
33      {
34          node now = que.top();
35          que.pop();
36          if(now.dis != disc[now.num])
37              continue;//该点被取过了就continue
38          for(int i = Head[now.num]; i != -1; i = Next[i])
39          {
40              int to = v[i];
41              //如果能通过now.num这个点使得起点到to这个点更近，就更新距离
42              if(disc[to] > disc[now.num] + w[i])
43              {
44                  disc[to] = disc[now.num] + w[i];
45                  que.push(node{disc[to], to});
46              }
47          }
48      }
49      int ans = 0;//由于是求所有奶牛到该牧场的距离和最小，所以要用乘法
50      for(int i = 1; i <= p; i ++)
51          ans += disc[i] * vis[i];
52      return ans;
53  }
```

# 小结

Dijkstra求最短路径的算法是一种基于贪心策略的算法。每次新扩展一个路程最短的点，更新与其相邻的点的路程。当所有边权都为正时，由于不会存在一个路程更短的没扩展过的点，所以这个点的路程永远不会再被改变，因而保证了算法的正确性。不过根据这个原理，用本算法求最短路径的图是不能有负权边的，因为扩展到负权边的时候会产生更短的路程，有可能就破坏了已经更新的点路程不会改变的性质。

# 3 [2892] 最短路径问题

平面上有n个点（n≤100），每个点的坐标均在-10000~10000之间。其中的一些点之间有连线。若有连线，则表示可从一个点到达另一个点，即两点间有通路，通路的距离为两点间的直线距离。现在的任务是找出从一点到另一点之间的最短路径。

输入
共n+m+3行，其中:
第1行为整数n。第2行到第n+1行（共n行），每行两个整数x和y，描述了一个点的坐标。第n+2行为一个整数m，表示图中连线的个数。
此后的m 行，每行描述一条连线，由两个整数i和j组成，表示第i个点和第j个点之间有连线。
最后一行：两个整数s和t，分别表示源点和目标点。
输出
一行，一个实数（保留两位小数），表示从s到t的最短路径长度。

样例输入

```
5
0 0
2 0
2 2
0 2
3 1
5
1 2
1 3
1 4
2 5
3 5
1 5
```

样例输出

```
3.41
```

```cpp
#include <bits/stdc++.h>
using namespace std;
const int inf=1e9+7;

struct point
{
    double x,y;
}a[105];//每个点坐标

struct node
{
    int to;//出边顶点
    double cost;//权值
};

int n,m,s,e;
vector <node> G[1005];
double dj[1005];
int vis[1005];//默认为0，表示未处理

void dijstla(int s)
{
```

```cpp

double cal(int x,int y)
{
    return sqrt((a[x].x-a[y].x)*(a[x].x-a[y].x)+(a[x].y-a[y].y)*(a[x].y-a[y].y));
}
```

```
52  int main()
53  {
54      cin>>n;
55      for(int i=0;i<=n;i++)
56          dj[i]=inf;//最短距离初始无穷大
57      int x,y;
58      for(int i=1;i<=n;i++)
59          cin>>a[i].x>>a[i].y;
60      cin>>m;
61      for(int i=1;i<=m;i++)
62      {
63          cin>>x>>y;//计算距离，建立邻接表
64          G[x].push_back((node){y,cal(x,y)});
65          G[y].push_back((node){x,cal(x,y)});
66      }
67      cin>>s>>e;
68      dijstla(s);
69      printf("%.2f\n",dj[e]);
70      return 0;
71  }
```

```
21  void dijstla(int s)
22  {
23      dj[s]=0;
24      while(1)
25      {
26          int u=-1;
27          for(int i=0;i<n;i++)
28          {
29              if(!vis[i]&&(u==-1||(dj[i]<dj[u])))
30              {
31                  u=i;
32              }
33          }
34          if(u==-1) break;
35          vis[u]=1;
36          for(int i=0;i<G[u].size();i++)
37          {
38              node v=G[u][i];
39              if(!vis[v.to]&&dj[v.to]>dj[u]+v.cost)
40              {
41                  dj[v.to]=dj[u]+v.cost;
42              }
43          }
44      }
45  }
```

# 4 [3581] 最短路计数

题目描述
给出一个N个顶点M条边的无向无权图，顶点编号为1 ~ N。问从顶点1开始，到其他每个点的最短路有几条。
输入
输入第一行包含2个正整数N，M，为图的顶点数与边数。
接下来M行，每行两个正整数x, y，表示有一条顶点x连向顶点y的边，请注意可能有自环与重边。

输出
输出包括N行，每行一个非负整数，第i行输出从顶点1到顶点i有多少条不同的最短路，由于答案有可能会很大，你只需要输出mod 100003后的结果即可。如果无法到达顶点i则输出0。

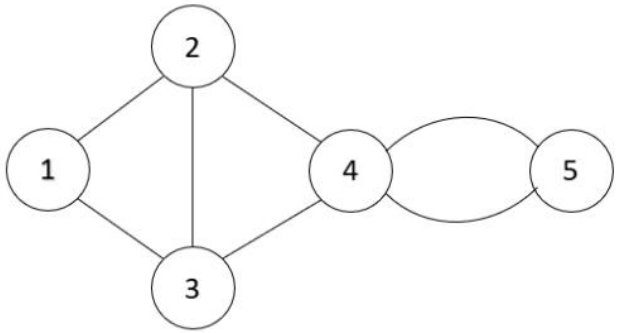
样例输入

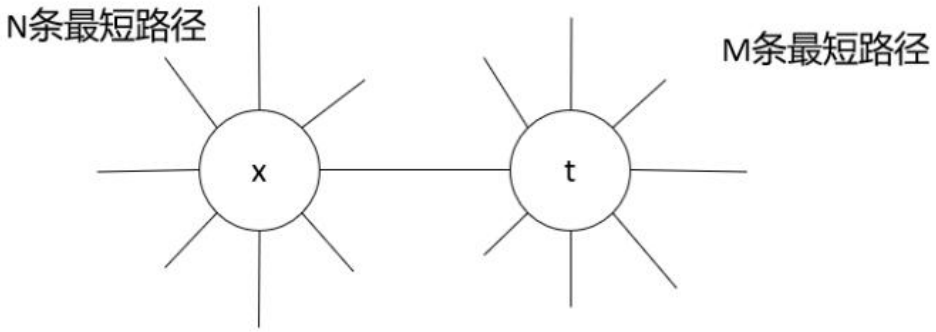
```
5 7
1 2
1 3
2 4
3 4
2 3
4 5
4 5
```

样例输出

```
1
1
1
2
4
```

用了bfs




```
if(dep[t]==dep[x]+1)
        cnt[t]=(cnt[t]+cnt[x])
```

```cpp
#include<bits/stdc++.h>
using namespace std;
const int maxn=1000000+1,maxm=2000000+1,INF=0x7f7f7f7f,MOD=100003;
vector<int>G[maxn];
int dep[maxn];bool vis[maxn];int cnt[maxn];
//cnt存最短路个数，vis记录节点是否访问，dep表示距离
int main(){
    int n,m;scanf("%d%d",&n,&m);
    for(int i=1;i<=m;i++){
        int x,y;scanf("%d%d",&x,&y);
        G[x].push_back(y);
        G[y].push_back(x);
    }
    queue<int>Q;
    dep[1]=0;vis[1]=1;Q.push(1);cnt[1]=1;
    while(!Q.empty()){  //队列为空
        int x=Q.front();Q.pop();  //从前往后搜索
        for(int i=0;i<G[x].size();i++){
            int t=G[x][i];
            if(!vis[t]){vis[t]=1;dep[t]=dep[x]+1;Q.push(t);}  //若未识别
            if(dep[t]==dep[x]+1){cnt[t]=(cnt[t]+cnt[x])%MOD;}  //取模
        }
    }
    for(int i=1;i<=n;i++){
        printf("%d\n",cnt[i]);  //输出
    }
    return 0;
}
```

# 今天的课程结束啦……

**下课了…**
**同学们再见！**