指定一个点（源点）到其余各个顶点的最短路径。

首先是数据结构，先要存数据的嘛

用二维数组存顶点之间边的关系

比如说a【1】【2】就是顶点1到顶点2的距离，a【1】【2】= 1，为1.（懒得画图）

再来一个book数组，记录源点到该点是否确定了最短路径

初始化两个数组为无穷大。然后再读取初始化的数据。

还需要一个dis数组来存储源点到其余各点的初始路程，也用来存储最终路程。

一开始dis数组只存了源点直接到各个点的值，既然是求源点到各个顶点的最短距离，我们先从dis数组开始，遍历dis数组，找到源点到各个顶点的最短距离。

具体的算法如下：

//这里我们假设1是源点

//输入n（顶点）数量，m（边）数量

.....（输入略）

//初始化数组（就是把二维数组和dis数组设为无穷大既不可到达）

for(int i = 1;i <= n;i++)

{

for(int j = 1;j <=n;j++)

{

If(i == j)

e[i][j] = 0;

else

e[i][j] = inf;

}

dis[i] = inf;

book[i] = 0;//标记数组，标记源点到i点已经为最短路径

}

book[1]=1;

//输入数据

for(int i = 1;i <= m;i++)

{

......（输入略）

}

for(int i = 1;i <= n;i++)

dis[i] = e[1][i];//存储一开始源点到各点的最短路径

---------------------------------------------初始化工作结束------------------------------------------------------

遍历dis数组

for(int i = 2;i <= n;i++)//从2开始遍历（这里我们假设1是源点）

{

min = inf(9999999999999);//inf为变量，设置成我们存放最小值，这里是初始化

for(int j = 1;j <= n;j++)//寻找dis数组中最小的值，找到的最小值就是已确定的最短路径。

{

//为什么这么说呢，因为dis数组记录源点到各个点的最短路径（是直达的）。比如说dis【2】= 2；说明距离顶点1最近的点是顶点2，距离为2。读者可能会问，能不能顶点1先到其他点再到顶点2呢？从而缩短顶点1到顶点2的距离。不可能，因为顶点1可以直接到顶点2了，而且是最小的（距离顶点1最近），所以从顶点1绕路到顶点2只能增加距离。

If(book[j] == 0 && dis[j] < min)

{

min = dis[j];

u = j;

}

book[u] = 1;//下次寻找不再找它（u）。

//上面确定了一个最短路径，接下来就是从确定的那个顶点拓展，比如确定了1到2的最短路径，那么从2拓展，比如说2-3，2-4；比对dis[3]跟e[u][v] + dis[2]哪个大，更新dis数组。

for(v = 1;v <= n;v++)//接着上面的例子说，这里是直接循环找到2的出边，因为e二维数组储存了顶点之间的信息。然后比对dis[3]跟e[u][v] + dis[2]哪个大，更新dis数组

{

If(e[u][v] < inf)

{

If(dis[v] > dis[u] = e[u][v])

dis[v] = dis[u] + e[u][v];

}

}

}

}

好了，循环结束后，dis数组里面就存储了源点到各个顶点的最短距离了。

这里的时间复杂度是n的平方，优化的方法有用堆来优化，可以降到logn

还可以用邻接表代替邻接矩阵（就是存储数据的那个二维数组）

可以降到（m+n）log n的平方，需要注意的是当边等于顶点数的平方（也就是稀疏图，边数远小于顶点数的平方，相对的，叫稠密图），时间复杂度会比n的平方还高，所以要看情况使用。最好用堆。

而且这个是基于贪心策略的求最短路径算法，每次更新拓展一个最短路径的点。所以就要求所以边权为正，不能为负。为负的话每次更新拓展的可能就不是最短的。所以用这个算法算要求不能有负权边。