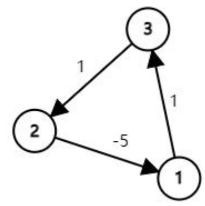
最短路（下）

3. spfa 判断负环

负环的含义：指的是一个图中存在一个环，里面包含的边的边权总和<0.



求负环的常用方法，基于$spfa$，一般采用**方法 2**：

✧ 方法一：统计每个点入队的次数，如果某个点入队 n 次，则说明存在负环.

✧ 方法二：统计当前每个点的最短路中所包含的边数，如果某点的最短路所包含的边数大于等于n，也说明存在负环.

求**起点** $start$ **出发**能否到达负环：

1. $dist[x]$ 表示起点 $start$ 到 x 的最短距离；

2. $cnt[x]$ 表示当前 x 点到起点 $start$ 最短路的边数，初始每个点到起点 $start$ 的边数为 0，只要它能再走 n 步，即 $cnt[x] \geq n$，则表示该图一定存在负环，由于从起点 $start$ 到 x 至少经过 n 条边时，则说明图中至少有 $n+1$ 个点，表示一定有点重复使用；

3. 若 $dist[j] > dist[t] + val[i]$，则表示从 t 点走到 j 点能够让权值变少，对该点 j 进行更新，并且对应的 $cnt[j] = cnt[t] + 1$，往前走一步.

代码链接

```
#include<iostream>
#include<cstring>
#include<algorithm>
#include<queue>

#define MAXN 100010

using namespace std;

const int inf=0x3f3f3f3f;
int n,m;
```

```
int head[MAXN],ed[MAXN],val[MAXN],nex[MAXN],idx;
int dist[MAXN],vis[MAXN],cnt[MAXN];

void add(int x,int y,int z){
    ed[idx]=y;
    val[idx]=z;
    nex[idx]=head[x];
    head[x]=idx++;
}

int spfa(int start){
    memset(dist,0x3f,sizeof(dist));
    memset(vis,0,sizeof(vis));
    dist[start]=0;
    vis[start]=1;
    queue<int> q;
    q.push(start);
    while(q.size()>0){
        int temp=q.front();
        q.pop();
        vis[temp]=0;
        for(int i=head[temp];i!=-1;i=nex[i]){
            if(dist[ed[i]]>dist[temp]+val[i]){
                dist[ed[i]]=dist[temp]+val[i];
                /****添加部分****/
                cnt[ed[i]]=cnt[temp]+1;
                if(cnt[ed[i]]>=n)
                    return 1;
                /****添加部分****/
                if(vis[ed[i]]==0){
                    q.push(ed[i]);
                    vis[ed[i]]=1;
                }
            }
        }
    }
    return 0;
}

int main(){
    memset(head,-1,sizeof(head));
    scanf("%d %d",&n,&m);
    for(int i=1;i<=m;i++){
        int x,y,z;
```

```
        scanf("%d %d %d",&x,&y,&z);
        add(x,y,z);
    }
    int res=spfa(1);
    if(res==1)
        printf("Yes\n");
    else printf("No\n");
    return 0;
}
```

由于一般是求**整个图是否存在负环**,需要将所有的点都加入队列中,更新周围的点.

代码链接

```
#include<iostream>
#include<cstring>
#include<algorithm>
#include<queue>

#define MAXN 100010

using namespace std;

const int inf=0x3f3f3f3f;
int n,m;
int head[MAXN],ed[MAXN],val[MAXN],nex[MAXN],idx;
int dist[MAXN],vis[MAXN],cnt[MAXN];

void add(int x,int y,int z){
    ed[idx]=y;
    val[idx]=z;
    nex[idx]=head[x];
    head[x]=idx++;
}

int spfa(){
    memset(dist,0x3f,sizeof(dist));
    memset(vis,0,sizeof(vis));
    queue<int> q;
    /****修改部分****/
    for(int i=1;i<=n;i++){
        q.push(i);
        vis[i]=1;
    }
```

```
    /****修改部分****/
    while(q.size()>0){
        int temp=q.front();
        q.pop();
        vis[temp]=0;
        for(int i=head[temp];i!=-1;i=nex[i]){
            if(dist[ed[i]]>dist[temp]+val[i]){
                dist[ed[i]]=dist[temp]+val[i];
                cnt[ed[i]]=cnt[temp]+1;
                if(cnt[ed[i]]>=n)
                    return 1;
                if(vis[ed[i]]==0){
                    q.push(ed[i]);
                    vis[ed[i]]=1;
                }
            }
        }
    }
    return 0;
}

int main(){
    memset(head,-1,sizeof(head));
    scanf("%d %d",&n,&m);
    for(int i=1;i<=m;i++){
        int x,y,z;
        scanf("%d %d %d",&x,&y,&z);
        add(x,y,z);
    int res=spfa();
    if(res==1)
        printf("Yes\n");
    else printf("No\n");
    return 0;
}
```

4. Floyd 算法

为了求出图中任意两点间的最短路径，当然可以把每个点作为起点，求解 N 次单源最短路径问题. 不过，在任意两点间最短路问题中，图一般比较稠密. 使用 $Floyd$ 算法可以在 $O(n^3)$ 的时间内完成求解.

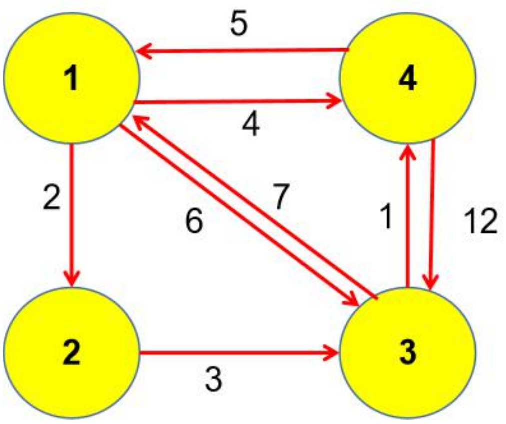
步骤：

(1). 初始化邻接矩阵(二维数组) $dist[][]$，其中 $dist[i][j]$ 表示顶点 i 到顶点 j 的权值，若顶点 i 和顶点 j 不相邻，则 $dist[i][j]=+\infty$，若顶点 i =顶点 j，则 $dist[i][j]=0$；

(2). 以第 1 个顶点为中介点，若 $dist[i][j]>dist[i][1]+dist[1][j]$，更新 $dist[i][j]$；

(3). 依次以第 $2,3,\ldots,k,\ldots,n$ 个顶点为中介点，若 $dist[i][j]>dist[i][k]+dist[k][j]$，更新 $dist[i][j]$.

举例：



j \ i	1	2	3	4
1	0	∞	7	5
2	2	0	∞	∞
3	6	3	0	12
4	4	∞	1	0

在只允许过1号顶点的情况下，任意两点之间的路程更新为:

j \ i	1	2	3	4
1	0	∞	7	5
2	2	0	9	7
3	6	3	0	11
4	4	∞	1	0

在只允许过1号和2号顶点的情况下，任意两点之间的路程更新为:

j \ i	1	2	3	4
1	0	∞	7	5
2	2	0	9	7
3	5	3	0	10
4	4	∞	1	0

在只允许过1号、2号和3号顶点的情况下，任意两点之间的路程更新为:

j \ i	1	2	3	4
1	0	10	7	5
2	2	0	9	7
3	5	3	0	10
4	4	4	1	0

在允许所有顶点作中转，任意两点之间的路程更新为:

j \ i	1	2	3	4
1	0	9	6	5
2	2	0	8	7
3	5	3	0	10
4	4	4	1	0

设 $dist[k,i,j]$ 表示”**经过若干个编号不超过 k 的节点**”从 i 到 j 的最短路长度. 该问题可划分为两个子问题, 经过编号不超过 $k-1$ 的节点从 i 到 j, 或者从 i 先到 k, 再到 j. 可得:

$$dist[k,i,j]=\min(dist[k-1,i,j],dist[k-1,i,k]+dist[k-1,k,j]);$$

初值为 $dist[0,i,j]=A[i,j]$, 其中 A 为该图的邻接矩阵.

可以看到, Floyd 算法的本质是**动态规划**, k 是阶段, 应置于最外层循环中. i 和 j 是附加状态， 应置于内层循环. 故不应该采用 i,j,k 的顺序执行循环, 会得到错误的答案.

k 这一维可省略(三维变为二维), 最初, 可直接用 $dist$ 保存邻接矩阵, 然后执行动态规划的过程. 当最外层循环到 k 时, 内层有转移方程:

$$dist[i,j]=\min(dist[i,j],dist[i,k]+dist[k,j])$$

核心代码:

```
int dist[MAXN][MAXN];

void floyd(){
    for(int k=1;k<=n;k++)
        for(int i=1;i<=n;i++)
            for(int j=1;j<=n;j++)
                dist[i][j]=min(dist[i][j],dist[i][k]+dist[k][j]);
}
```

拓扑排序

1. 问题引入

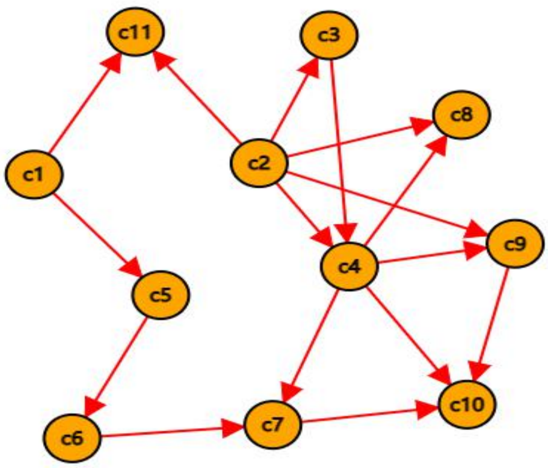
现实生活中我们遇到的要处理的事情可能要讲究先后顺序,比如说要先穿好衬衫才能穿外套,再比如说一个计算机系的学生想修完所有课程:

面对这样一张图，如何**求出做事的顺序**？这时需要**拓扑排序**.

课程代号	课程名	前导课程
c_1	高等数学	无
c_2	高级语言程序设计	无
c_3	离散数学	c_2
c_4	算法与数据结构	c_2 c_3
c_5	数字电路基础	c_1
c_6	计算机组成原理	c_5
c_7	操作系统	c_4 c_6
c_8	编译原理	c_2 c_4
c_9	算法分析与设计	c_2 c_4
c_{10}	软件工程学	c_4 c_7 c_9
c_{11}	数值分析	c_1 c_2

软件工程专业的一组必学课程

将**事件**视为点,**先后关系**视为边,问题转化为在图中求一个有先后关系的排序,可得到如下图:

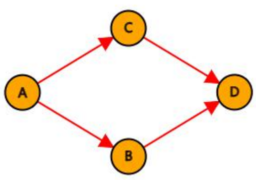


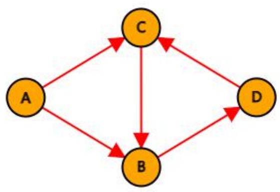
2. 拓扑排序的**概念**

在图论中,拓扑排序是一个**有向无环图**的所有顶点的线性序列,且该序列必须满足以下两个条件:

1. 每个顶点出现且出现一次.
2. 若存在一条从顶点 A 到顶点 B 的路径,那么在序列中顶点 A 出现在顶点 B 的前面.

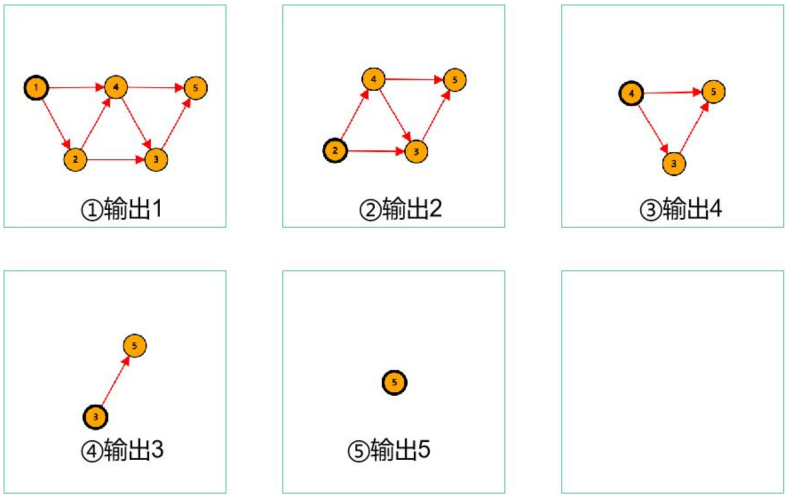
注：有向无环图（DAG）才有拓扑排序,非 DAG 无拓扑排序.例如下左图中存在拓扑排序（A,B,C,D 或 A,C,B,D——此处可说明**拓扑排序可能不唯一**),而下有图中不存在拓扑排序(存在 B,C,D 构成的环——此处可推出拓扑排序可**判断有向图中是否有环**).





3. 拓扑排序的**求法及步骤**

1. 在该图中选择一个没有前驱（即入度为 0）的顶点并记录该顶点；
2. 从图中删除该顶点和所有以它为起点的有向边；
3. 重复步骤 1 和步骤 2，直到当前的图为**空**或者**当前图中不存在没有前驱（入度为 0）的顶点**为止，如果满足后一种情况说明**有向图必然存在环**.



代码链接 时间复杂度为 $O(n+m)$，其中 n 为顶点数，m 为边数.

```
int deg[MAXN];//记录每个点的入度
int topsort(){
    queue<int> q;
    for(int i=1;i<=n;i++)//将所有入度为0 的点加入队列
        if(deg[i]==0)
            q.push(i);
    while(q.size()>0){
        int temp=q.front();
        res[++cnt]=temp;
        q.pop();

        for(int i=head[temp];i!=-1;i=nex[i]){//将选中的点的所有从该点出发的边删除
            deg[ed[i]]--;
            if(deg[ed[i]]==0)//将删除后入度为0 的点加入队列中
                q.push(ed[i]);
        }
    }
    if(cnt<n)//该图中存在环
        return -1;
    return 1;
}
```