最短路

在每年的校赛里, 所有进入决赛的同学都会获得一件很漂亮的T-shirt.但是每当我们的工作人员把上百件的衣服从商店运回到赛场的时候, 却是非常累的! 所以现在他们想要寻找最短的从商店到赛场的路线, 你可以帮助他们吗?

- **Standard Input**

输入包括多组数据. 每组数据第一行是两个整数N、M（N<=100, M<=10000）, N表示成都的大街上有几个路口, 标号为1的路口是商店所在地, 标号为N的路口是赛场所在地, M则表示在成都有几条路. N=M=0表示输入结束. 接下来M行, 每行包括3个整数A,B,C(1<=A,B<=N,1<=C<=1000), 表示在路口A与路口B之间有一条路, 我们的工作人员需要C分钟的时间走过这条路.

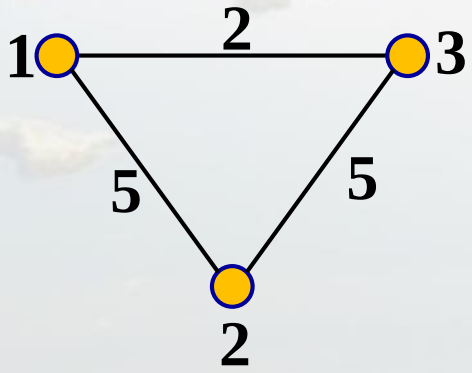
输入保证至少存在1条商店到赛场的路线.

- **Standard Output**

对于每组输入，输出一行，表示工作人员从商店走到赛场的最短时间.

- **Samples**

Input	Output
2 1 1 2 3 3 3 1 2 5 2 3 5 3 1 2 0 0	179



题目分析

- 直接使用Dijkstra算法.

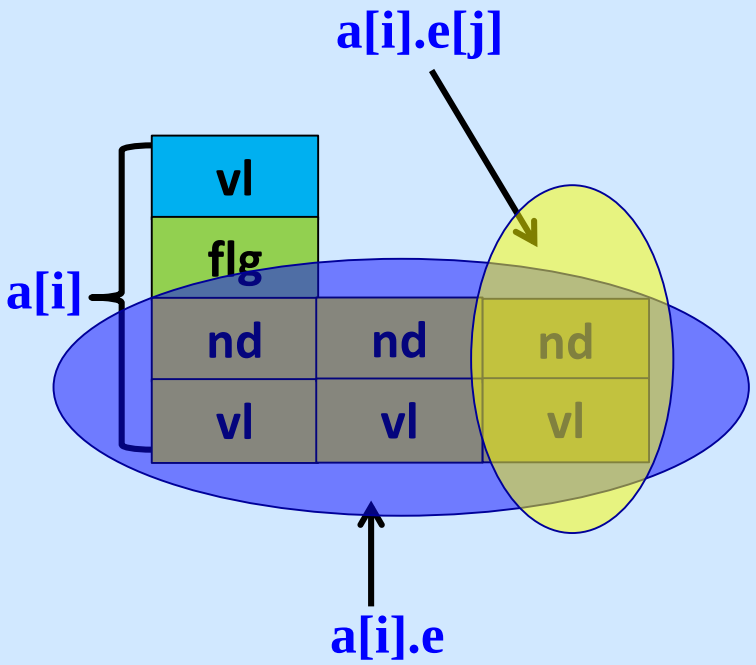
参考程序

```
#include <bits/stdc++.h>
using namespace std;
#define N 102
#define INF 10000000 //无穷
int graph[N][N];//邻接矩阵
int main()
{
    int a, b, c, i, j, n, m;
    int dis[N];//记录起点到该点的最短距离
    int f[N];//记录固定标号与临时标号
    while(scanf("%d%d", &n, &m) == 2) {
        if(n == 0 && m == 0) break;
        for (i = 0; i < N; i++)
            for (j = 0; j < N; j++)
                if(i == j) graph[i][j] = 0;
                else       graph[i][j] = INF;
        for (i = 0; i < m; i++) {
            scanf("%d%d%d", &a, &b, &c);
            a--; b--;//结点编号映射到0~n-1
            graph[a][b] = graph[b][a] = c;//无向图
        }
        for (i = 1; i < n; i++)  dis[i] = graph[0][i];
        memset(f, 0, sizeof(f));
        dis[0] = 0; f[0] = 1;//初始只有起点为固定标号
```

```
        for (i = 0; i < n-1; i++) {
            a = -1; //记录临时编号中值最小的编号
            b = INF; //记录临时编号中最小值
            for (j = 1; j < n; j++) {
                if (f[j] == 0 && dis[j] < b) {
                    b = dis[j];
                    a = j;
                }
            }
            if (a == -1) break;//说明无路到终点
            if (a == n-1) break;//已找到到终点的最短路
            f[a] = 1;//改为固定编号
            for (j = 1; j < n; j++)//更新相邻结点的标号值
                if ((f[j] == 0) && (b + graph[a][j] < dis[j]))
                    dis[j] = b + graph[a][j];
        }
        if(dis[n-1] >= INF) printf("Impossible\n");
        else  printf("%d\n",dis[n-1]);
    }
    return 0;
}
```

邻接表实现

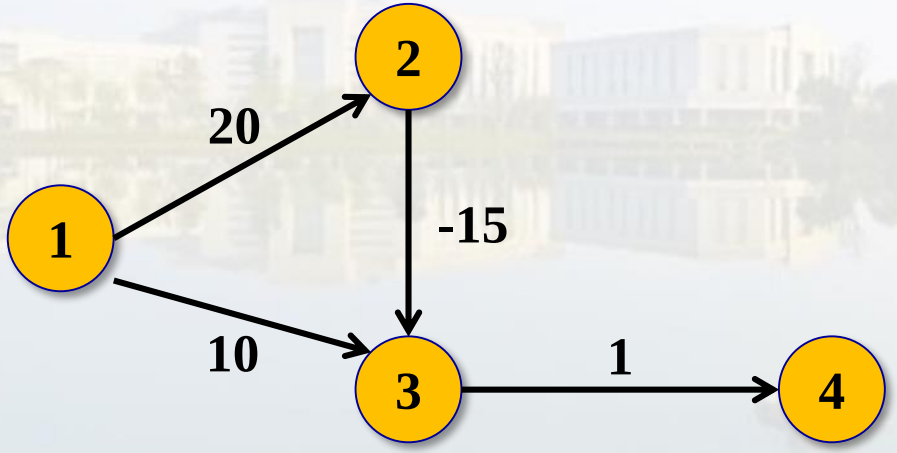
```
#include <bits/stdc++.h>
using namespace std;
#define INF 10000000 //无穷
struct edge {
    int nd, vl; //邻接边的另一个结点和边的权值
};
struct node {
    int vl;//起点到该结点的距离（目前）
    bool flg;//固定编号与临时编号标志
    vector<edge> e; //保存所有邻接边
    node():vl(0),flg(false){} //构造函数，完成初始化
};
int main() {
    int n, m, u, v, w; //结点数和边数
    edge temp;
    while(scanf("%d%d", &n, &m) == 2) {
        if(n == 0 && m == 0) break;
        vector<node> a(n);
        for(int i = 0; i < m; i++) {
            scanf("%d%d%d", &u, &v, &w);
            u--; v--;
            temp.nd = v;  temp.vl = w;
            a[u].e.push_back(temp);
            temp.nd = u; //无向图
            a[v].e.push_back(temp);
        }
```



```
        for(int i = 0; i < a[0].e.size(); i++)
            a[a[0].e[i].nd].vl = a[0].e[i].vl;
        a[0].flg = true;
        for(int i = 0; i < n-1; i++) {
            int mii = -1; //记录临时编号中值最小的编号
            int miv = INF; //记录临时编号中最小值
            for(int j = 1; j < n; j++) {
                if (!a[j].flg && a[j].vl < miv){
                    miv = a[j].vl;
                    mii = j;
                }
            }
            if (mii == -1) break;//说明无路到终点
            if (mii == n-1) break;//已找到到终点的最短路
            a[mii].flg = true;//改为固定编号
            for(int j = 0; j < a[mii].e.size(); j++)
                if(!a[j].flg && miv + a[mii].vl < a[j].vl)
                    a[j].vl = miv + a[mii].vl;
        }
        if (a[n-1].vl >= INF) printf("Impossible\n");
        else printf("%d\n", a[n-1].vl);
    }
    return 0;
}
```

算法分析

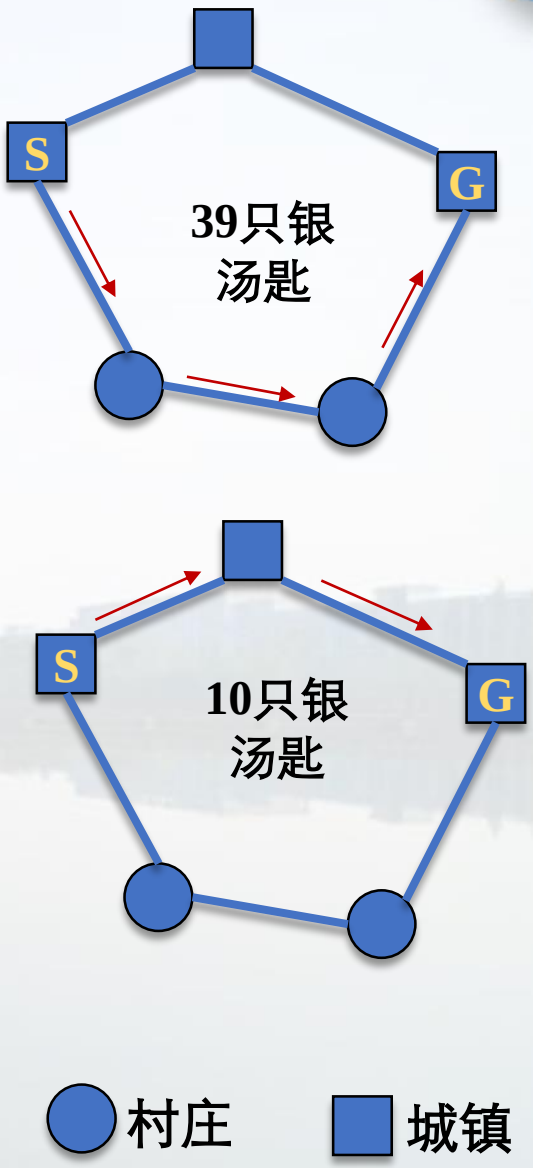
- 时间复杂度: O(|V|*|V|). 如是用优先队列维护最小值, 可以做到O(|V|*log|E|).
- 算法的局限性: 不能有负权边.



Toll

水手辛巴德把66只银汤匙出售给了撒马尔罕的苏丹. 出售相当容易, 但运货十分复杂. 这些物品要在陆路上转运, 通过若干个城镇和乡村. 而每个城镇和乡村都要收取过关费, 没有交费不准离开. 一个村庄的过关费是1个单位的货物, 而一座城镇的过关费是每20单位的货物收取1个单位的货物. 例如, 你带了70个单位的货物进入一个城镇, 则必须缴纳4个单位的货物. 城镇和村庄位于无法通行的山岩、沼泽和河流之间, 所以根本无法避免.

预测在每个村庄或城镇收取的费用很简单, 但要找到最佳路线(最便宜的路线)则是一个真正的挑战. 最佳路线取决于运送货物的数量单位. 货物的数量在20以内, 村庄和城镇收取的费用是相同的. 但是对于数量较大的货物, 就要避免通过城镇, 可以通过比较多的村庄, 右图是一个例子.



请编写一个程序来解决辛巴德的问题. 给出要运送到某个城镇或村庄的货物的单位数量和一张路线图, 程序必须确定最廉价的路线以及在开始的时候需要带的货物的单位数量的总数.

- **Standard Input**

输入包含若干个测试用例. 每个测试用例由两部分组成: 路线地图, 然后是有关运送货物的细节.

路线的第一行给出一个整数n, 表示在地图中路线的数量 (0<=n).

后面的n行每行有两个字母, 表示一条路的两个端点. 大写字母表示城镇, 小写字母表示村庄, 两个方向中的任何一个方向都可以行走.

在路线地图后给出一行, 有关运送货物的细节, 这一行有3个元素: 整数p (0 < p <= 1000) 要运送到目的地的货物的单位数量, 一个表示开始位置的字母, 一个表示要送达的目的地位置的字母. 要求通过这样的路线地图使得这样的货物数量可以被送达.

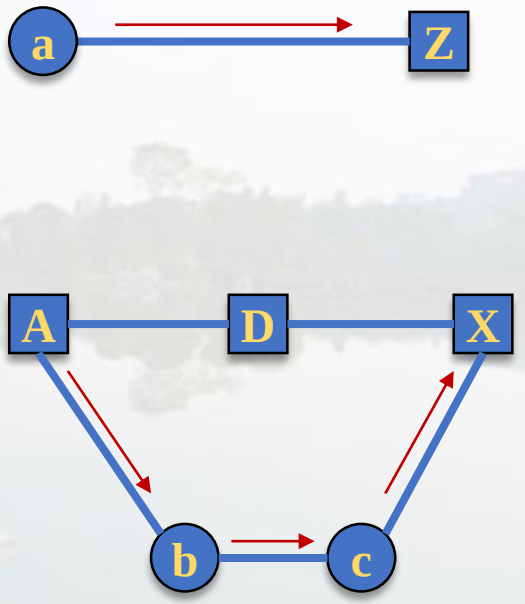
在最后一个测试用例后, 给出一行, 包含一个数字 -1.

- **Standard Output**

对每个测试用例输出一行, 给出测试用例编号和在出发的时候要带的货物的单位数量. 在下面样例中给出了输出格式.

- **Samples**

Input	Output
1 a Z 19 a Z 5 A D D X A b b c c X 39 A X -1	Case 1: 20 Case 2: 44



题目分析

- 先把路线图转化为一个无向图. 为方便, 城镇A对应结点1, ... , 城镇Z对应26; 村庄a对应结点27, ... 村庄z对应结点52.
- 如果知道出发时带的货物量k, 可以用类似于Dijkstra算法的方式求出到达终点时剩下的最大货物量p.
- 为了得到最小k, 考虑用二分的方法来求取:

把到达终点to时剩下的最大货物数p作为起点from至终点to的最长路径长度. 在p固定的情况下, 可以利用二分搜索的方法得到最小的k.

```
设区间[l,r], 初始为[p, 2^20].
 while(l != r) {
    mid = floor((l+r-1)/2);
    if(mid个货物到达to时剩下的货物数>=p)  r = mid;
    else l = mid + 1;
 }
```

- 如何求k个货物达到to时剩下的货物? 利用Dijkstra求最短路径的思想.

```
用数组g[]记录最长路, 显然g[from] = k; flag[]为结点在两个集合中的标志, 初始时所有结点的标志记为false.
while(true) {
   在标志为false的结点中找next, g[next] = w = max{g[i]}
   if(没找到) break;
   flag[next] = true;
   for(int i = 1; i <=52; i++) {
      if(结点i与next相邻) {
          if(i <= 26)  剩余货物 l = w - (w+19)/20;  //i是城镇
          else    剩余货物 l = w - 1;
          g[i] = max{l, g[i]};
      }
   }
}
```

参考程序

```
#include <stdio.h>
int tot;
int go[55][55];
int turn(char x)
{
    return (x < 'a') ? (x - 'A' + 1) : (x - 'a' + 27);
}
int check(int from, int to, int o)
{
    int temp, g[55] = {0}, flag[55] = {0};
    g[from] = o;
    while(1)
    {
        int w = 0, next = -1;
        for(int i = 1; i <= 52; i++)
            if(!flag[i] && (next==-1 || w < g[i]))
            {
                next = i;
                w = g[i];
            }
        if(next == -1)
            break;
        flag[next] = 1;
        for(int i = 1; i <=52; i++)
            if(go[next][i])
            {
                temp = w - ((i<27)?((w+19)/20):(1));
                g[i] = (temp>g[i])?(temp):(g[i]);
            }
    }
    return g[to];
}
int main()
{
    char s1[10], s2[10];
    int cnt = 0, T;
    while(scanf("%d", &T) == 1)
    {
        if(T == -1)
            break;
```

```
        for(int i = 0; i < T; i++)
        {
            scanf("%s%s", s1, s2);
            int x = turn(s1[0]), y = turn(s2[0]);
            go[x][y] = go[y][x] = 1;
        }
        scanf("%d%s%s", &tot, s1, s2);
        int from = turn(s1[0]), to = turn(s2[0]);
        int l = tot, r = (1<<20);
        while(l != r)
        {
            int mid = (l+r-1) >> 1;
            if(check(from, to, mid) >= tot)  r = mid;
            else l = mid + 1;
        }
        printf("Case %d: %d\n", ++cnt, l);
    }
    return 0;
}
```

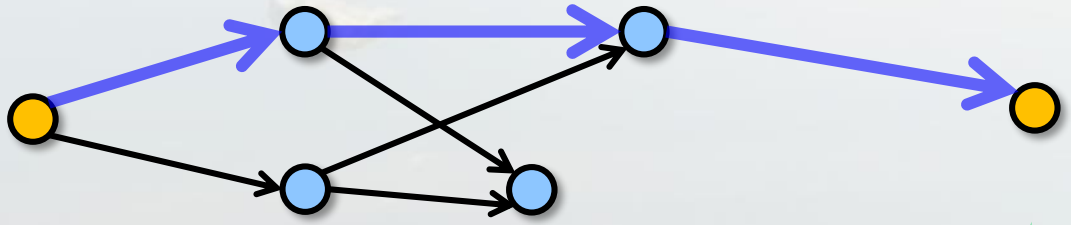
2、Bellman-Ford算法

当负权存在时, 最短路都不一定存在(有负环一定不存在).

如果最短路存在, 则最短路经过的边数不超过n-1条, 由此可有下面的算法.

```
for(int i = 0; i < n; i++) dis[i] = INF;
dis[0] = 0;
for(int k = 0; k < n-1; k++) { //控制次数
    for(int i = 0; i < m; i++) {
        if(dis[v[i]] > dis[u[i]] + w[i])
            dis[v[i]] = dis[u[i]] + w[i];
    }
}
```

完整程序如下:



```
#include <bits/stdc++.h>
using namespace std;
#define N 10000
#define INF 99999999
struct edge {
    int u, v, w;
}a[N];
int main() {
    int n, m, dis[N];
    bool flag;
    scanf("%d%d", &n, &m);
    for(int i = 0; i < m; i++)
        scanf("%d%d%d", &a[i].u, &a[i].v, &a[i].w);
    for(int i = 0; i < n; i++)  dis[i] = INF;
    dis[0] = 0; //起点
    //Bellman-Ford算法核心代码
    for(int k = 0; k < n-1; k++) {
        flag = false; //记录是否更新
        for(int i = 0; i < m; i++) {
            if(dis[a[i].v] > dis[a[i].u] + a[i].w) {
                dis[a[i].v] = dis[a[i].u] + a[i].w;  flag = true;
            }
        }
        if(!flag)  break;
    }
```

```
    //检测负权回路
    flag = false;
    for(int i = 0; i < m; i++)
        if(dis[a[i].v] > dis[a[i].u] + a[i].w) {
            flag = true; break;
        }
    if(flag)
        printf("此图含有负权回路! ");
    else {
        for(int i = 0; i < n; i++)
            printf("%d ", dis[i]);
    }
    return 0;
}
```

容易想到，如果每次仅对最短路程发生了变化的结点的相邻边执行操作，效率可能会高一些。

但如何知道当前哪些点的最短路程发生了变化呢?

可以使用一个队列来维护这些点。

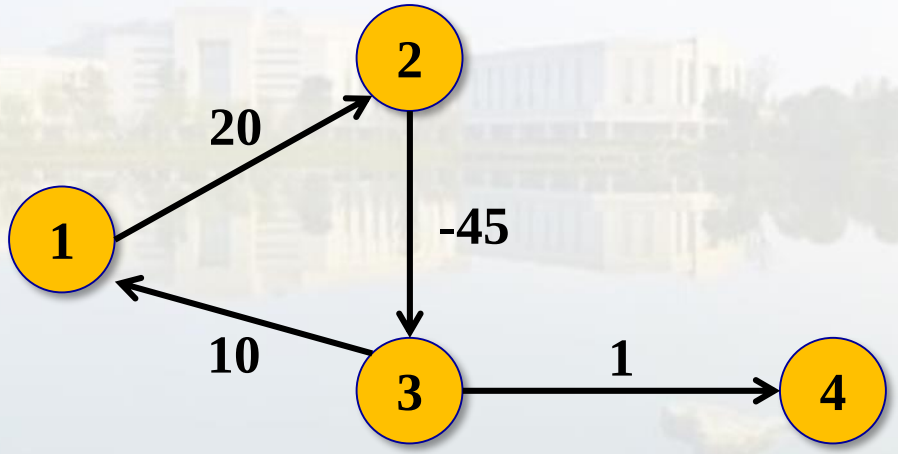
```
#include <bits/stdc++.h>
using namespace std;
const int INF = 99999999;
struct edge {
    int v, w;
};
vector<vector<edge> > a;
void add_edge(int u, int v, int w){
    edge temp;
    temp.v = v;
    temp.w = w;
    a[u].push_back(temp);
}
int main() {
    int n, m, u, v, w;
    scanf("%d%d", &n, &m);
    a.resize(n);
    for(int i = 0; i < m; i++) {
        scanf("%d%d%d", &u, &v, &w);
        add_edge(u, v, w);//有向图
    }
    vector<int> dis(n, INF);
    vector<bool> flag(n, false);
    queue<int> q;
```

```
    q.push(0);//始点入队
    dis[0] = 0;
    flag[0] = true;
    while(!q.empty()) {
        u = q.front(); q.pop(); flag[u] = false;
        for(int i = 0; i < a[u].size(); i++) {//扫描所有邻接点
            if(dis[a[u][i].v] > dis[u] + a[u][i].w) {
                dis[a[u][i].v] = dis[u] + a[u][i].w;
                if(!flag[a[u][i].v]){
                    q.push(a[u][i].v);
                    flag[a[u][i].v] = true;
                }
            }
        }
    }
    for(int i = 0; i < n; i++)
        printf("%d ", dis[i]);

    return 0;
}
```

算法分析

- 最坏时间复杂度: O(|V|*|E|). 但一般没那么大. 为了避免最坏情况的出现, 在正权图上应使用效率更高的Dijkstra算法.
- 算法的局限性: 在稠密图中复杂度比迪杰斯特拉算法高.
- 当一个结点进入队列的次数超过|V|次, 则一定存在负环.



3、Floyd算法

如果需要求任意两点之间的距离，不必调用n次Dijstra或Bellman-ford算法，可以使用Floyd-Warshall算法。

- Floyd算法利用了动态规划
- 用d[i][j][k]表示从i到j，经过编号不超过k的点所得到的最短距离，则

$$d[i][j][k] = \min\{d[i][j][k-1],\ d[i][k][k-1] + d[k][j][k-1]$$

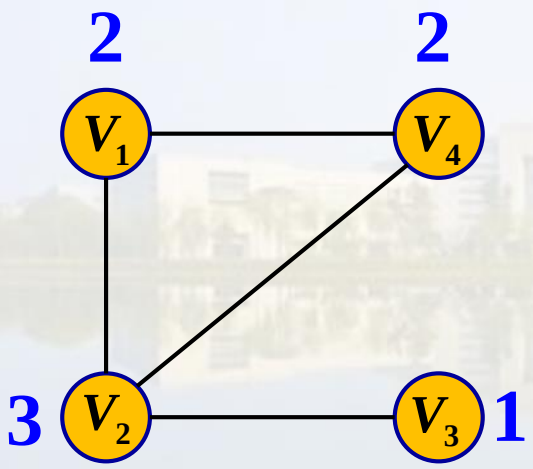
```
//if(i==j)  d[i][i] = 0;
//else if(i与j相邻)  d[i][j]=w[i][j];
//else d[i][j]=INF;

for(int k = 0; k < n; k++) //控制结点编号集合
  for(int i = 0; i < n; i++)
    for(int j = 0; j < n; j++)
      d[i][j] = min(d[i][j], d[i][k] + d[k][j]);
```

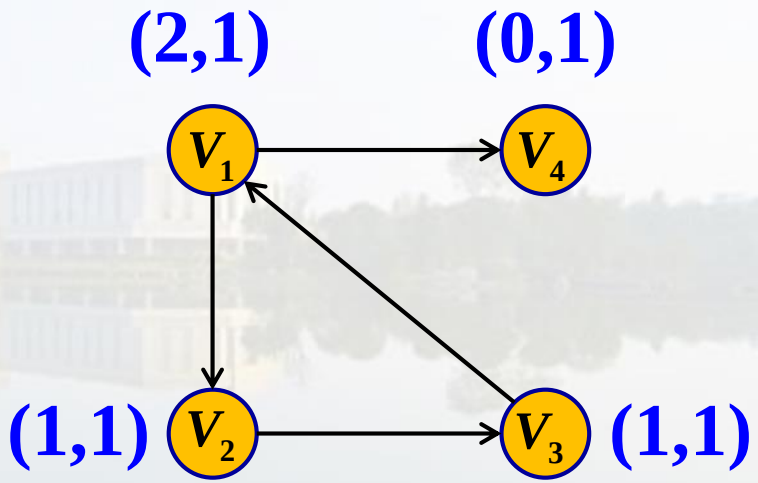
```
#include <bits/stdc++.h>
using namespace std;
const int INF = 99999999;
int main() {
    int n, m, u, v, w;
    scanf("%d%d", &n, &m);
    vector<vector<int> > dis(n);
    for(int i = 0; i < n; i++) {
        dis[i].resize(n, INF);  dis[i][i] = 0;
    }
    for(int i = 0; i < m; i++) {
        scanf("%d%d%d", &u, &v, &w);
        dis[u][v] = w;//有向图
    }
    for(int k = 0; k < n; k++)
        for(int i = 0; i < n; i++)
            for(int j = 0; j < n; j++)
                if(dis[i][j] > dis[i][k] + dis[k][j])
                    dis[i][j] = dis[i][k] + dis[k][j];
    for(int i = 0; i < n; i++) {
        for(int j = 0; j < n; j++)
            printf("%d ", dis[i][j]);
        printf("\n");
    }
    return 0;
}
```

七、度、欧拉图和中国邮递员问题

- 设G是任意图，x为G中的任意结点，与结点x关联的边数称为x的度数（自环算两度）。通常记为d(x)（或deg(x)）。
- 在有向图中进入x的边数称为x的入度，记为$d^+(x)$，由x出发的边数称为x的出度，记为$d^-(x)$。

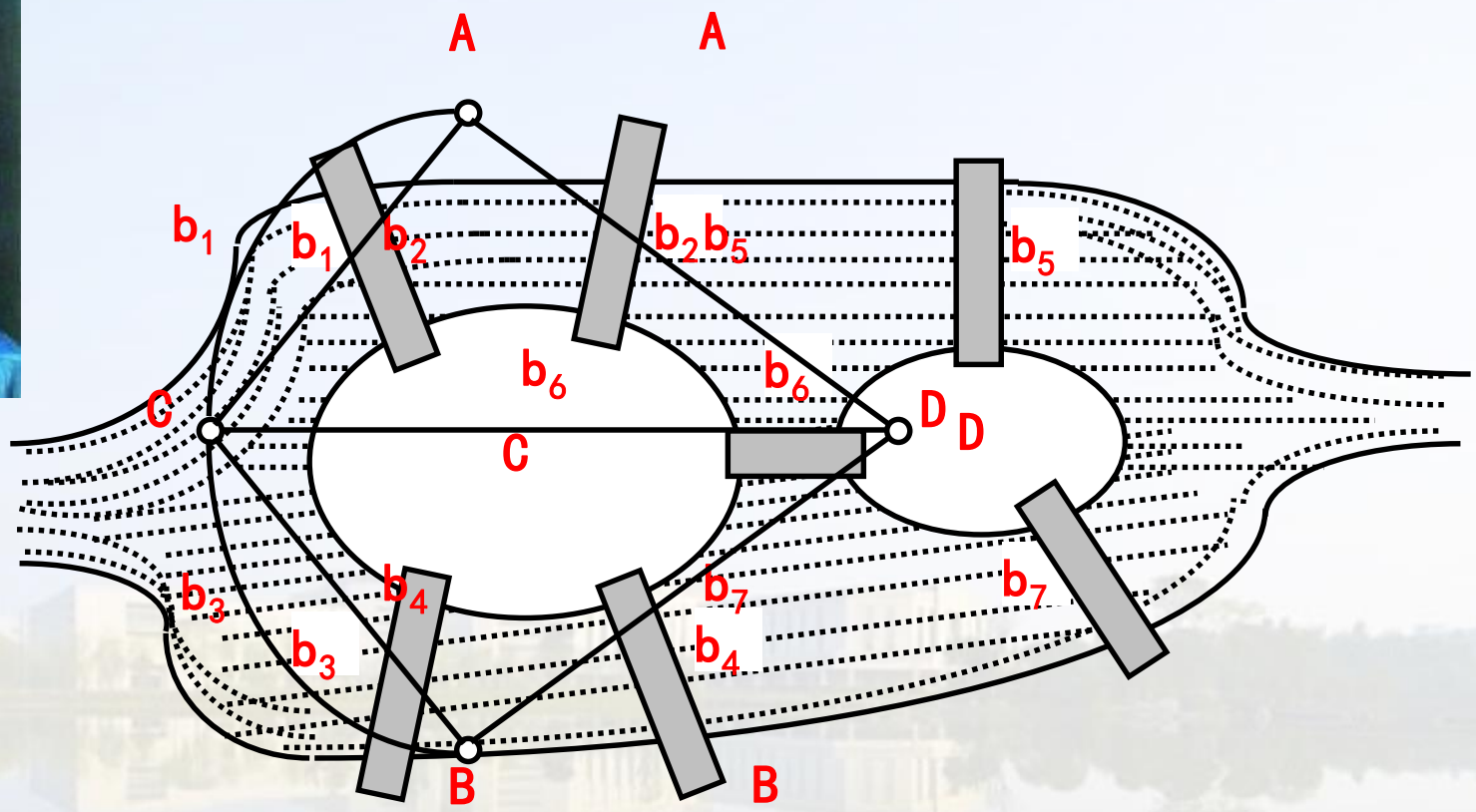


无向图的度



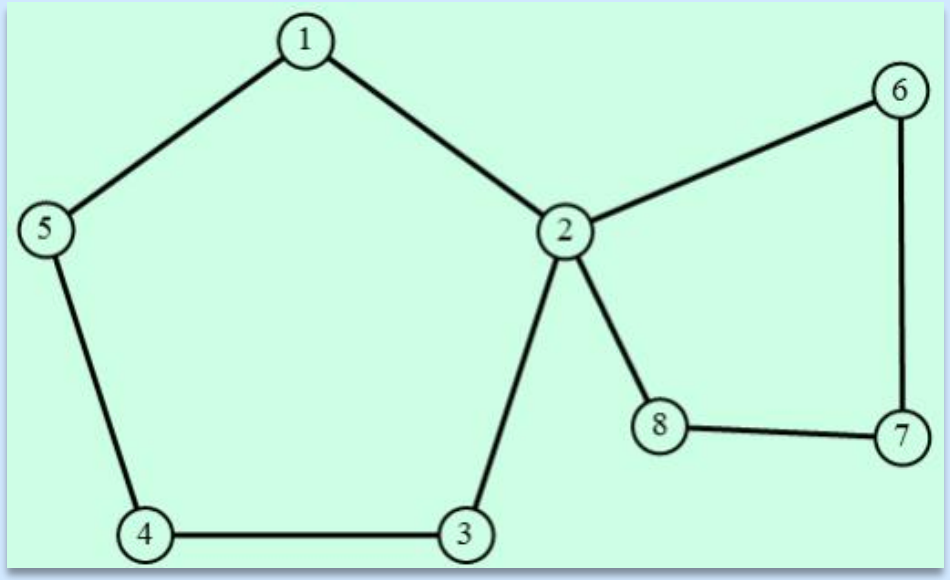
有向图的度

哥尼斯堡七桥问题与欧拉图



- 无向图欧拉回路：所有顶点度为偶数。
- 有向图欧拉回路：所有顶点入度等于出度
- 混合图欧拉回路？
- 欧拉通路

```
//递归搜索(深度优先搜索)欧拉回路
void Euler(int u){
    int v;
    for(v = 0; v < n; v++){
        if(graph[u][v] == 1){
            graph[u][v] = graph[v][u] = 0;
            Euler(v);
            path[pc++] = v;
        }
    }
}
//调用代码
pc = 0;   Euler(s);    path[pc++] = s;
for(i = pc - 1; i >= 0; i--)
    printf("%d ", path[i]+1);
```



若s取为1，则进入 path（堆栈）的顺序为：

1	5	4	3	2	8	7	6	2	1

中国邮递员问题

一个邮递员送信，要走完他负责投递的全部街道，完成任务后回到邮局，应按怎样的路线走，他所走的路程才会最短呢？

如果将这个问题抽象成图论的语言，就是**给定一个连通图，连通图的每条边的权值为对应的街道的长度(距离)，要在图中求一回路，使得回路的总权值最小。**

若图为欧拉图，只要求出图中的一条欧拉回路即可。否则，邮递员要完成任务就得在某些街道上重复走若干次。

如果重复走一次，就加一条平行边，于是原来对应的图就变成了多重图。只是要求加进的平行边的总权值最小就行了。

问题就转化为：**在一个有奇度数结点的赋权连通图中，增加一些平行边，使得新图不含奇度数结点，并且增加的边的总权值最小。**

要解决上述问题，应分下面两个大步骤。

首先，增加一些边，使得新图无奇度数结点，我们称这一步为可行方案(Feasible Scheme)；

其次，调整可行方案，使其达到增加的边的总权值最小，称这个最后的方案为最优方案(Optimal Scheme)。

结论：

I）在最优方案中，图中每条边的重数小于等于2；

II）在最优方案中，图中每个基本回路上平行边的总权值不大于该回路的权值的一半

（注：基本回路可以由生成树得到）

八、哈密尔顿图与巡回售货员问题

经过图中每个结点一次且仅一次的通路(回路)称为哈密顿通路(回路) 。存在哈密顿回路的图称为哈密顿图(Hamiltonian Graph)。

充分条件：

1. 设无向图G = <V, E>是哈密顿图，V1是V的任意非空子集，则

$$p(G-V1) \le |V1|$$

其中p(G-V1)是从G中删除V1后所得到图的连通分支数。

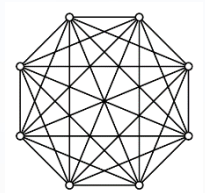
2.设G = <V, E>是具有n个结点的简单无向图。如果对任意两个不相邻的结点u, v∈V，均有

$$deg(u)+deg(v) \ge n-1$$

则G中存在哈密顿通路。

巡回售货员问题

$G = <V, E, W>$是n个结点的赋权完全图，这里$V = \{v_1, v_2, \cdots, v_n\}$是城市的集合，E是连接城市的道路的集合，W是从E到正实数集合的一个函数(即$W(v_i, v_j)$是城市v_i与v_j之间的距离)，显然对V中任意三个城市v_i, v_j, v_k，显然它们之间的距离应满足三角不等式：$W(v_i, v_j) + W(v_j, v_k) \geq W(v_i, v_k)$，试求出该赋权图上的最短哈密顿回路。

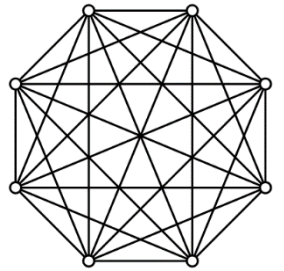


最邻近算法：

① 以v_i为始点，在其余n-1个结点中，找出与始点最邻近的结点v_j(如果与v_i最邻近的结点不唯一，则任选其中的一个作为v_j)，形成具有一条边的通路v_iv_j；

② 假设x是最新加入到这条通路中的结点，从不在通路上的结点中选取一个与x最邻近的结点，把连接x与此结点的边加到这条通路中。重复这一步，直到G中所有结点都包含在通路中；

③ 把始点和最后加入的结点之间的边放入，就得到一条回路。



抄近路算法：

① 求G中的一棵最小生成树T；

② 将T中各边均加一条与原边权值相同的平行边，设所得图为G’，显然G’是欧拉图；

③ 求G’中的一条欧拉回路E；

④ 在E中按如下方法求从结点v出发的一个哈密顿回路H：从v出发，沿E访问G’中各个结点，在没有访问完所有结点之前，一旦出现重复出现的结点，就跳过它走到下一个结点，称这种走法为抄近路走法。W(H)作为最短哈密顿回路的长度(设为d_0)的近似值。

可以证明：若赋权完全图$K_n(n\geq 3)$满足三角不等式，d_0是K_n中最短哈密顿回路的长度，H是用抄近路算法求出的K_n中的哈密顿回路，则

$$W(H)<2d_0。$$

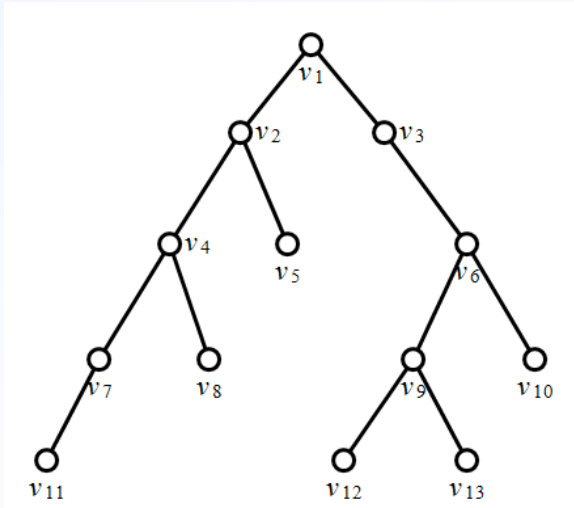
九、几种特殊的图

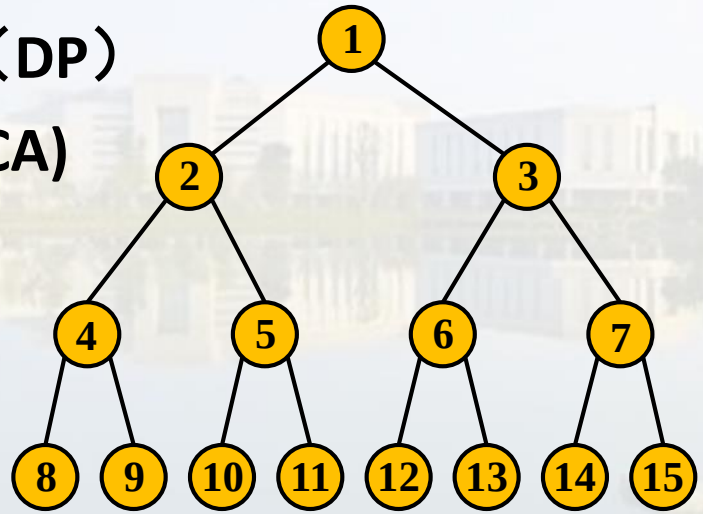
1、树

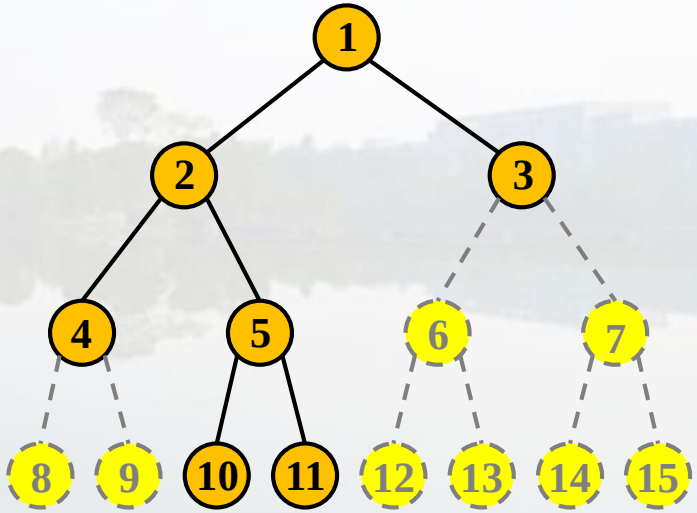
- n个结点，n-1条边的连通图
- 没有回路（环），任意两结点间有唯一路径
- 天然的递归结构

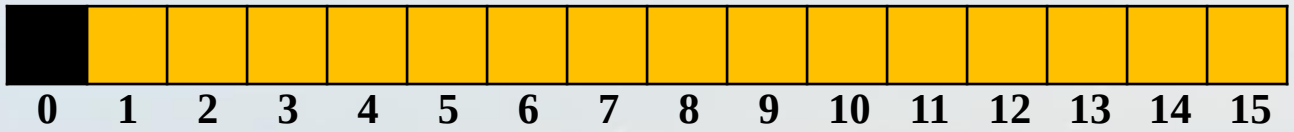
常见问题：

- 树上的动态规划（DP）
- 最近公共祖先（LCA)
- 树形转线形
- 生成树
- 树上的分治

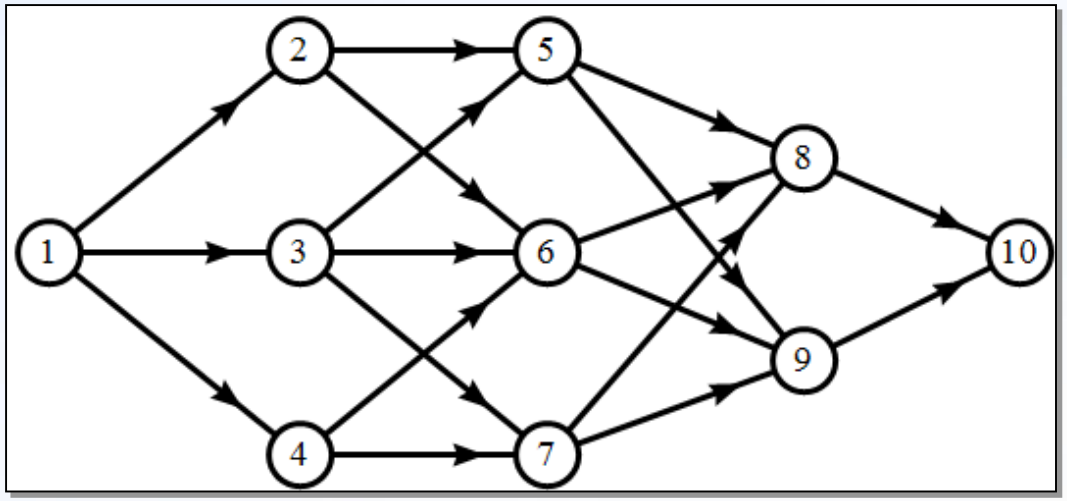








2、有向无环图（DAG）



常见问题：

- DAG上的的动态规划（DP）
- 拓扑排序

嵌套矩形问题：有n个矩形，每个矩形可以用两个整数a、b描述，表示它的长和宽。矩形X(a,b)可以嵌套在Y(c,d)中，当且仅当a<c,b<d，或者b<c,a<d（相当于把矩形X旋转90º)。

硬币问题：有n种硬币，面值分别为$V_1,V_2,\ldots,V_n$，每种都有无限多。给定非负整数S，可以选用多少个硬币，使得面值之和恰好为S？输出硬币数目的最小值和最大值。1<=n<=100,0<=S<=10000,1<=V_i<=S。

拓扑排序

考虑如下问题

1、学习物理必须要有微积分基础

2、学习模拟电路必须要有电路分析基础

3、学习电路分析要先学物理和微积分

4、学习概论率论要有微积分基础

5、学习随机信号分析要先学概率论

请确定一种选课顺序。

分析：先建图

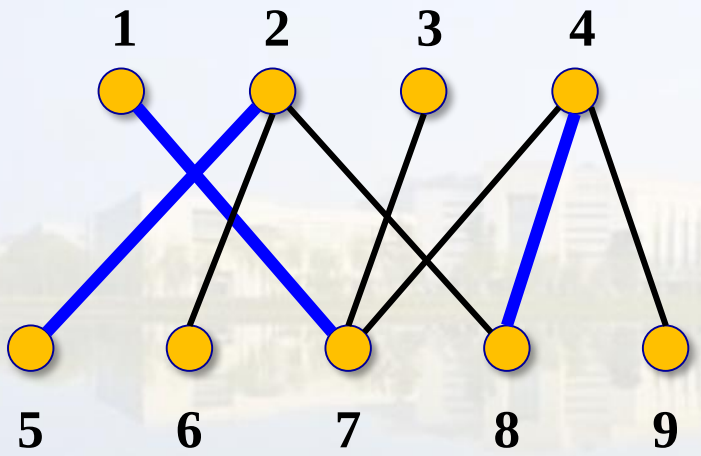
方法：先选入度为0的结点，选完之后把由其出发的边去掉，……

拓扑排序实现代码

```
queue<int> q;
for(int i = 0; i < n; i++)
    if(deg[i] == 0)
        q.push(i);
while(!q.empty())
{
    int u = q.front();
    q.pop();
    for(int v = 0; v < n; v++)
    {
        if(g[u][v])
        {
            deg[v]--;
            if(deg[v] == 0)
                q.push(v);
        }
    }
}
```

3、二分图（偶图）

如果图的结点可以分为两部分X和Y，并且图中**任意一条边的端点一个在X中，另一个一定在Y中**，这样的图称为二分图或偶图。



定理：图G是二分图当且仅当图G内没有含奇数个节点的回路。

常见问题：匹配

算法：匈牙利算法，网络流