新增模版

ST-RMQ

ST是可以通过O(nlogn)预处理，O(1)求任一区间最值的算法

题意:

给你n(1e5)个数字，分成若干组，每组有l个(最后一组可以少一点)，使得每组的最大值的和+组数-1 <= w，求最小的符合条件的l.

类型:

ST

分析:

时间复杂度应该需要nlogn的做法。

先写了二分发现不对，因为不具有单调性。

通过wkc知道了可以用st表做。(其实只要记得st表就能知道就是用来求这类问题，但是我忘了= =)

（记得换角度思考问题 二分l (logn) * check(n) -> 枚举l (n) * check(logn))

---

stl-bitset

```
int n;
int v[2];
int a[2][105];
int f[N];
bitset<101>d[N];
bool solve(int o)
{
    MS(f, 63); f[0] = 0;
    int top = 0;
    for (int i = 1; i <= n; ++i)
    {
     for (int j = top; j >= 0; --j)
     {
          int x = min(j + a[o][i], v[o]);//下一个我方状态点
          int y = min(f[j] + a[o ^ 1][i], v[o ^ 1]);//下一个对方状态点
          if (y < f[x])
          {
               f[x] = y;
               d[x] = d[j];
               d[x][i] = 1;
          }
     }
     top = min(top + a[o][i], v[o]);
     if (f[v[o]] < v[o ^ 1])
     {
          puts("NO");
          for (int i = 1; i <= n; ++i)printf("%c", d[v[o]][i] ? '1' :
'0');
          puts("");
          return 1;
     }
```

```
    }
    return 0;
}
int main()
{
    while (~scanf("%d", &n))
    {
      scanf("%d", &v[0]);
      for (int i = 1; i <= n; ++i)scanf("%d", &a[0][i]);
      scanf("%d", &v[1]);
      for (int i = 1; i <= n; ++i)scanf("%d", &a[1][i]);
      if (!solve(0) && !solve(1))puts("YES");
    }
    return 0;
}
/*
```

【trick&&吐槽】
1，TLE了不一定是时间不够，可能是死循环了。
2，前驱记录输出方案的方法一定要顾忌到是否存在环
3，状压真是好用，bitset真是好用

【题意】
有n（100）个法官。
每个人有一个权威性a[]，
对于一次投票，如果投展成票人数的权威性之和>=p，那么就认为投票有效，否则无效。
然而，这些人的权威性被改变成了b[]，决定投票有效与否的数值也由p变成了q。

然后问你，前后的改变是否有实际区别。
如果有，输出同样的投票对应不同的结果的投票方案。

各种与权威性相关的数值a[]、b[]、p、q都在[1,1e6]范围。

【类型】
DP bitset

【分析】
首先，一道题的入手要考虑复杂度。
这道题我们有人数为n（100），各种权威性的值都在1e6，
于是，有一个2e8复杂度的显然正确的方法。
我们发现，出现了实际区别，意味着——
一种投票方案下，对应的权威值尽可能大时，
在另外一种投票方案下，对应的权威值却尽可能小。

不妨先假设p达标，q不达标
于是，我们用f[i][j]表示——
前i个人参与投票，当投票对应的a[]的权威值为j时，所对应的最小b[]
那么有状态转移——
gmin(f[i+1][j+a[i]],f[i][j]+b[i])
然而j+a[i]与f[i][j]+b[i]都可能爆炸。
我们只要控制两者的上限不超过p和q即可（因为实际意义都相同）

最后只要检测f[p]是否<q即可。
还有p不达标q达标的情况，我们对应着也DP一遍。

然而，最后我们还要输出具体的方案。
这个怎么做呢?
一种比较常见的做法，是在每个f[i][j]的j下标对应前驱。
然而，在达到顶端p的时候，这个j有可能指向自己，这样就GG了。
而里面所存的数值，更不能所有前溯的指标。

于是，我们需要用一种特殊的记录方法——
我们观察到，人数n只有100，100虽然超出了状压的范围，
然而还是可以用分开状压（2个LL）或者用bitset存储记忆这个状态是由哪些人贡献来的。
这样我们只要拆分d[p]就可以输出答案了。

【时间复杂度&&优化】
O(n*1e6)

*/

题意:
    验证给出的三个n*n的矩阵a,b,c, a*b==c? (n<=4000)
类型:

分析:
    如果直接相乘的话复杂度会是4000^3爆炸,
    于是我们引入一个n*1的列矩阵x,
    如果a*b*x==c*x的话，那证明a*b==c的概率极大。于是用随机化算法。
    a*b*x->a*(b*x)
    于是复杂度只要n^2即可。
    4000*4000直接用数组存不下， 需要用bitset。

C语言细节

读题一定要认真。不能害怕。
排序比较的时候要写!=
读入和输出string的时候 可以用强转和.c_str();   // 用printf输出string
for(auto it : hashy) hashy[it.first] = ++cc;   // c++11的stl简便写法

交互题

每次输出之后需要添加:
fflush(stdout);

题意:

交互题。 初始有160个硬币， 可以下200次以内的赌注。 假设一次下注k，赢了硬币数量会加上k，输了会减去k。达到200就算赢，变成0就算输。机器里有个初始的数x，x的变化公式为:Xi = 487237 * Xi-1 + 1011807 % 2^20. 如果当前机器中的数x的二进制位中1的个数为奇数，那么我们就会赢。否则我们就会输。机器不会告诉我们x是多少，但是会告诉我们每次输赢的结果(即每次下完注后我们现有的硬币总数)，问下注方案。

类型:

交互_暴力出奇迹哎

分析:

每次我们可以根据输赢情况删去大概一半左右的数。所以可以直接暴力

---

ubuntu命令行

打开终端
ctrl+alt+t
终端命令:
ps-a 显示终端上的所有进程
1.23 kill 给一个进程发信号，或终止一个进程的运行
1.23.1 用pid来终止一个进程
cat创建一个进程
按下ctrl+z挂起进程，输入ps -aux查看当前进程，假设cat进程的pid为1760，输入kill -SIGKILL 1760 即可结束
1.23.2 用进程名来终止一个进程
同上创建进程并查看，输入killall -9 cat 即可结束进程
sudo reboot 重启

杂题

题意:

有一个凸包1e5，还有很多点2e4，判断这些点是否都严格在凸包内

类型:

凸包 排序

分析:

方法1:把凸包分成多个小三角形 123, 134, 145, ... 极角排序，二分查找。

法2:直接用O(logn)的模版求出每个点是否在凸包内(比第一种快了将近4倍)

题意:

给定序列a[n]，求出min (i - j) ^ 2 + (sum[j] - sum[i]) ^ 2

类型:

计算几何_最近点对

分析:

sqrt一下就变成了最近点对- -

构造

题意:

n个点的凸出度为所有子集能够组成的凸包的最多点个数。现在给你n个点，让你输出凸出度为m的方案。没有则输出-1;

$3 \le m \le 100, m \le n \le 2m$

分析:

从一条y = x * x + inf 的抛物线上取m个点，同时从一条y = -x * x - inf 的抛物线上取剩下的点。

代码:

```
while(~ scanf("%d%d", &n, &m)){
    if(n == 5 && m == 3 || n == 6 && m == 3) puts("-1");
    else{
        for(int i = 1; i <= m; i ++){
            printf("%d %d\n", i, i * i + INF);
        }
        for(int i = 1; i <= n - m; i ++){
            printf("%d %d\n", i, -i * i - INF);
        }
    }
}
```

欧拉路

判断欧拉路，欧拉回路:

注意图联通，可以DFS或者并查集

一．无向图

欧拉回路：每个顶点度数都是偶数

欧拉路：所有点度数为偶数，或者只有2个点度数为奇数

二．有向图（非混合）

欧拉回路：每个顶点入度等于出度

欧拉路：每个顶点入度等于出度；

或者只有1个点入度比出度小1，从这点出发，只有1个点出度比入度小1，从这个点结束，其他点入度等于出度

九余数定理

一个数的每位数字之和等于这个数对9取余，如果等于0就是9

思路：利用九余数定理，求(n^n)%9值，直接快速幂即可。

汉诺塔

三柱汉诺塔：T(n)=2^n−1
四柱汉诺塔：F(n)=min(2∗F(n−r)+2^r−1)，（1≤r≤n）。当r=(sqrt(8∗n+1)−1)/2时，能保证f(n)取得最小值F(n)=(n−(r^2−r+2)/2)∗2^r+1。所以算法的复杂度是
F(n)=O(sqrt(2∗n)∗2^ sqrt(2∗n))

差分约束系统

```
X1 – X2 <= 0
X1 – X5 <= –1
X2 – X5 <= 1
X3 – X1 <= 5
X4 – X1 <= 4
X4 – X3 <= –1
X5 – X3 <= –3
X5 – X4 <= –3
```

不等式组(1)

全都是两个未知数的差小于等于某个常数（大于等于也可以，因为左右乘以–1就可以化成小于等于）。这样的不等式组就称作差分约束系统。

这个不等式组要么无解，要么就有无数组解。因为如果有一组解{X1, X2, ..., Xn}的话，那么对于任何一个常数k，{X1 + k, X2 + k, ..., Xn + k}肯定也是一组解，因为任何两个数同时加一个数之后，它们的差是不变的，那么这个差分约束系统中的所有不等式都不会被破坏。

差分约束系统的解法利用到了单源最短路径问题中的三角形不等式。即对于任何一条边u –> v，都有：

```
d(v) <= d(u) + w(u, v)
```

其中d(u)和d(v)是从源点分别到点u和点v的最短路径的权值，w(u, v)是边u –> v的权值。

显然以上不等式就是d(v) – d(u) <= w(u, v)。这个形式正好和差分约束系统中的不等式形式相同。于是我们就可以把一个差分约束系统转化成一张图，每个未知数Xi对应图中的一个顶点Vi，把所有不等式都化成图中的一条边。对于不等式Xi – Xj <= c，把它化成三角形不等式：Xi <= Xj + c，就可以化成边Vj –> Vi，权值为c。最后，我们在这张图上求一次单源最短路径，这些三角形不等式就会全部都满足了，因为它是最短路径问题的基本性质嘛。

话说回来，所谓单源最短路径，当然要有一个源点，然后再求这个源点到其他所有点的最短路径。那么源点在哪呢？我们不妨自已造一个。以上面的不等式组为例，我们就再新加一个未知数X0。然后对原来的每个未知数都对X0随便加一个不等式（这个不等式当然也要和其它不等式形式相

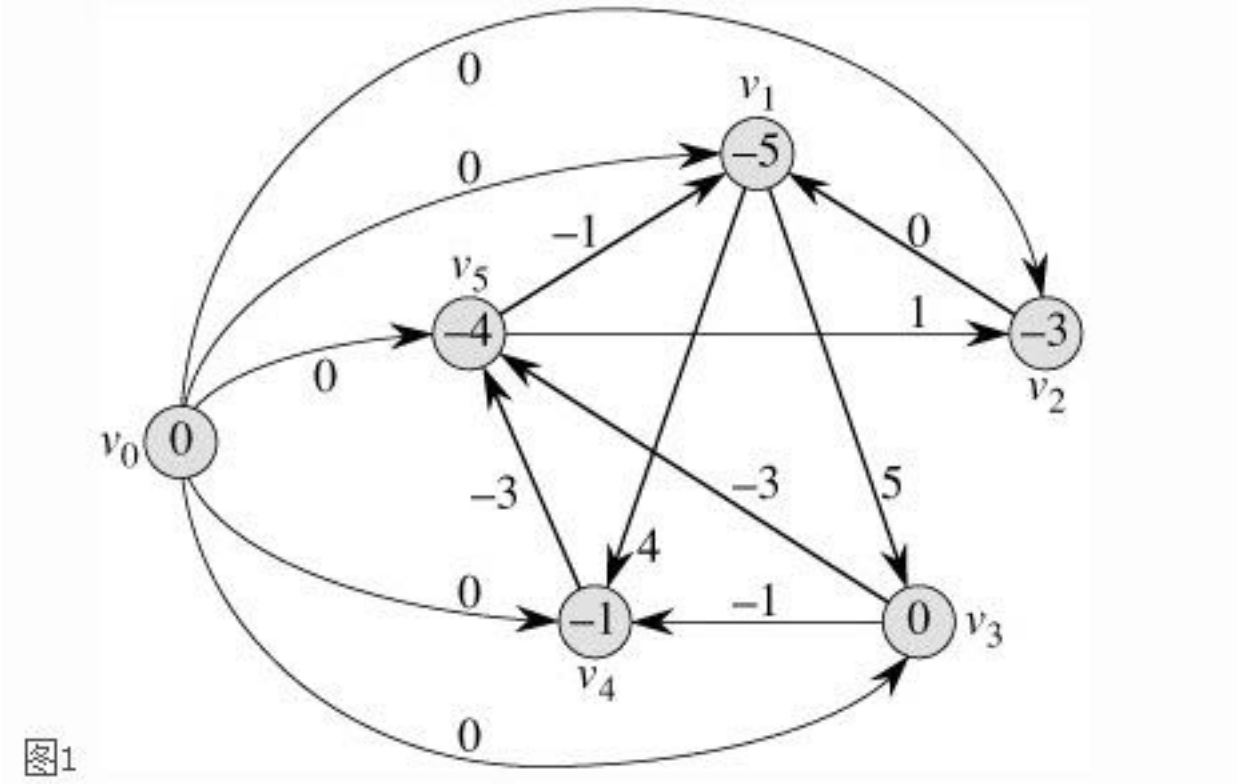
同，即两个未知数的差小于等于某个常数）。我们索性就全都写成Xn - X0 <= 0，于是这个差分约束系统中就多出了下列不等式：

```
X1 - X0 <= 0
X2 - X0 <= 0
X3 - X0 <= 0
X4 - X0 <= 0
X5 - X0 <= 0
```

不等式组(2)

对于这5个不等式，也在图中建出相应的边。最后形成的图如下：

图1



图中的每一条边都代表差分约束系统中的一个不等式。现在以V0为源点，求单源最短路径。最终得到的V0到Vn的最短路径长度就是Xn的一个解啦。从图1中可以看到，这组解是{-5, -3, 0, -1, -4}。当然把每个数都加上10也是一组解：{5, 7, 10, 9, 6}。但是这组解只满足不等式组(1)，也就是原先的差分约束系统；而不满足不等式组(2)，也就是我们后来加上去的那些不等式。当然这是无关紧要的，因为X0本来就是个局外人，是我们后来加上去的，满不满足与X0有关的不等式我们并不在乎。

也有可能出现无解的情况，也就是从源点到某一个顶点不存在最短路径。也说是图中存在负权的圈。这一点我就不展开了，请自已参看最短路径问题的一些基本定理。

其实，对于图1来说，它代表的一组解其实是{0, -5, -3, 0, -1, -4}，也就是说X0的值也在这组解当中。但是X0的值是无可争议的，既然是以它作为源点求的最短路径，那么源点到它的最短路径长度当然是0了。因此，实际上我们解的这个差分约束系统无形中又存在一个条件：

```
X0 = 0
```

也就是说在不等式组(1)、(2)组成的差分约束系统的前提下，再把其中的一个未知数的值定死。这样的情况在实际问题中是很常见的。比如一个问题表面上给出了一些不等式，但还隐藏着一些不等式，比如所有未知数都大于等于0或者都不能超过某个上限之类的。比如上面的不等式组(2)就规定了所有未知数都小于等于0。

对于这种有一个未知数定死的差分约束系统，还有一个有趣的性质，那就是通过最短路径算法求出来的一组解当中，所有未知数都达到最大值。下面我来粗略地证明一下，这个证明过程要结合Bellman-Ford算法的过程来说明。

假设X0是定死的；X1到Xn在满足所有约束的情况下可以取到的最大值分别为M1、M2、……、Mn（当然我们不知道它们的值是多少）；解出的源点到每个点的最短路径长度为D1、D2、……、Dn。

基本的Bellman-Ford算法是一开始初始化D1到Dn都是无穷大。然后检查所有的边对应的三角形不等式，一但发现有不满足三角形不等式的情况，则更新对应的D值。最后求出来的D1到Dn就是源点到每个点的最短路径长度。

如果我们一开始初始化D1、D2、……、Dn的值分别为M1、M2、……、Mn，则由于它们全都满足三角形不等式（我们刚才已经假设M1到Mn是一组合法的解），则Bellman-Ford算法不会再更新任合D值，则最后得出的解就是M1、M2、……、Mn。

好了，现在知道了，初始值无穷大时，算出来的是D1、D2、……、Dn；初始值比较小的时候算出来的则是M1、M2、……、Mn。大家用的是同样的算法，同样的计算过程，总不可能初始值大的算出来的结果反而小吧。所以D1、D2、……、Dn就是M1、M2、……、Mn。

那么如果在一个未知数定死的情况下，要求其它所有未知数的最小值怎么办？只要反过来求最长路径就可以了。最长路径中的三角不等式与最短路径中相反：
d(v) >= d(u) + w(u, v)
也就是d(v) - d(u) >= w(u, v)

所以建图的时候要先把所有不等式化成大于等于号的。其它各种过程，包括证明为什么解出的是最小值的证法，都完全类似。

最近几天系统得学习了一些差分约束系统的原理，特此记录如下：
所谓差分约束系统，是指一组不定方程(A,x,T,b)，其中A的每行有一个1，一个-1，其余为0，x为解向量，T为<=或>=组成的向量，b为约束矢量。具体来说，就是每行都具有 xi-xj >=|<= bi 的形式。约束的目标是使得目标函数xt-xs最大或最小。
这是典型的线性规划的个案，但是也可以转化为图论来做，利用最短路（或最长路）方法可以实现高效的解决方案。

hdu1384
题意:

给定n个限制条件x, y, v, 求一个集合，令每个区间[x, y] 之间至少有v个数。问集合中至少要有多少个数。

类型:

差分约束

分析:

```
令a[i] 表示从1 到 i中共选了多少个数
a[y] - a[x - 1] >= v
=> a[x - 1] - a[y] <= -v
=> ins(y + 1, x, -v);   // x 为0时
```

【题意】
有一家超市，实行的是24小时工作制，需要雇佣一些员工。
这家超市在不同的时间段需要不同数量的员工
每名员工被雇佣后可以在指定时间段连续工作8小时。

现在告诉你这24个小时中的每小时各需要R[i](0<=R[i]<=1000)名员工。
告诉你我们有n（1000）名可以雇佣的员工并告诉你他们的可行工作时间。

问你最少要雇佣多少员工才能满足要求。
（无解输出No Solution）

【类型】
差分约束系统

【分析】
和2SAT以及网络流一样，差分约束系统的关键也在于建图。

1，找影响问题处理的关键性要素——
我们设num[i]为i时刻能够开始工作的人数
wk[i]为i时刻实际雇佣的人数
need[i]为i时刻需要工作的人数

2，建立不等式关系
wk[i-7]+...+wk[i]>=need[i]

3，简化不等式关系
上式的wk[i-7]+...+wk[i]在实质上产生了前缀和关系，可以用于建立模型
同时，因为涉及到前缀和，我们不以0base而以1base的话会更方便。
于是，我们设sum[i]=wk[1]+...+wk[i]。
那么便有——
0<=sum[i]-sum[i-1]<=num[i]，
sum[i]-sum[i-8]>=need[i]。
问题的模型是环形的，而前缀和sum[]的累积却并非是环形的。
于是这里我们要加上约束条件——
0<=sum[i]-sum[i-1]<=num[i]，i∈[1,24]
sum[i]-sum[i-8]>=need[i]，i∈[8,24]
sum[i]+sum[24]-sum[i+16]>=need[i]，i∈[1,7]

4，转化不等式为标准形式
我们需要把式子转化成："(最多2个)未知项>=常数项"的形式
sum[i]-sum[i-1]>=0 ，i∈[1,24]
sum[i-1]-sum[i]>=-num[i] ，i∈[1,24]
sum[i]-sum[i-8]>=need[i] ，i∈[8,24]
sum[i]+sum[24]-sum[i+16]>=need[i] ，i∈[1,7]

然而，最后一个式子中出现了三个未知单位。
然而因为sum[24]是不随着i变化的。
所以我们把最后一个式子写成——
sum[i]-sum[i-1]>=0 ，i∈[1,24]
sum[i-1]-sum[i]>=-num[i] ，i∈[1,24]
sum[i]-sum[i-8]>=need[i] ，i∈[8,24]
sum[i]-sum[i+16]>=need[i]-sum[24] ，i∈[1,7]

5，建图
对于形式A-B>=C的式子，我们从节点B引出一条指向A的权值为C的有向边。

（也可以转化为：A+(-C)>=B，由A向B连一条权值为-C的有向边）
然而sum[24]的问题要怎么办呢?
我们二分sum[24]，验证可行性即可。

【时间复杂度&&优化】
如何验证可行性呢?
因为这个图是联通的，所以只要图中不会出现无限更新的环即可。

题意:
有n个任务，每个任务需要持续lst[i]天做完，每两个任务之间有四种关系:
1.SAF start after finish
2.SAS start after start
3.FAS finish after start
4.FAF finish after finish
输出一种方案，使得花最少的时间完成所有任务。

类型:
差分约束
分析:
我们设st[i] 为第i个任务的开始时间, fin[i] = st[i] + lst[i] - 1
四种关系:
1.SAF st[i] > fin[j]
st[i] > st[j] + lst[j] - 1
st[i] - st[j] >= lst[j]
2.SAS st[i] >= st[j]
st[i] >= st[j]
st[i] - st[j] >= 0
3.FAS fin[i] >= st[j]
st[i] + lst[i] - 1 >= st[j]
st[i] - st[j] >= 1 -lst[i]
4.FAF fin[i] >= fin[j]
st[i] + lst[i] - 1 >= st[j] + lst[j] - 1
st[i] - st[j] >= lst[j] - lst[i]

注意，这里要求的是最长路(所以需要加入的边值取反，画图即可明白)

题意:
有T(30)组数据。
对于每组数据有n（1e5）个村庄，m（1e5）个战场
对于村庄i，曹操可以选择支付c[i]*num元，（0<=c[i]<=1e5）
使得这个村庄派出num个士兵，在战场x[i]为曹操作战，在战场y[i]为袁绍作战。
(1<=x[i],y[i]<=m)

对于每个战场，都有个战略价值（0,1,2）。
在战略价值为2的战场，曹操在该战场的士兵数必须严格比袁绍多
在战略价值为1的战场，曹操在该战场的士兵数必须不能比袁绍少（按照贪心原则，实际一定会相等）
在战略价值为0的战场，曹操在该战场的士兵数无所谓（按照贪心原则，实际一定会为0）

让你输出保障上述条件曹操至少需要花费的金钱。

如果无法保障这个条件，则输出-1。

类型:

差分约束思想+费用流思想+最短路

分析:

首先有一个很显然的贪心——

如果一个战场的战略价值为2，那么在这个战场，恰有：曹兵-袁兵=1

如果一个战场的战略价值为1，那么在这个战场，恰有：曹兵=袁兵=0

这题我们可以从村庄向两个战场连边。

但是——如何更加简单地设置关系呢?

为何不考虑直接从曹战场向袁战场连边呢?

这题设计到分配，尽管数据巨大不能用网络流做，我们还是可以考虑先简化数据规模，引入思想。

我们对于每个村庄所提供的(x,y,z)，从曹战场x向袁战场y连接一条流量无穷，费用为z的边。

然后，我们有一个关键性的问题就是，如何保证题目的要求呢?

首先我们简化问题，只去考虑合法性。

对于所有战略价值为2的战场，如果从这个样的点ST出发，沿着图，能够达到任意一个战略价值为0的点。

那说明，这个战略价值为2的战场，可以使得其上的曹兵比袁兵多。

为什么能说明这个呢?

很显然，我们对于一条边，使得ST的曹兵+1，下一个点的袁兵+1。

但是这样，对于下一个点是有影响的。

如果下个点重要度为0，显然这样已经可以了。

如果下个点重要度为1，我们还要继续把这1个人转移出去，一直转移到重要度为0的点上即可。

如果下个点重要度为2，我们依然一定要把这1个人转移出去。而只要转移出去了，下个点实际上其实并没有受到影响。

于是，我们枚举所有重要度为2的点，每个点沿着这个图转移一个人到重要度为0的战场上，这道题就可以AC了。

体现在网络流上，就是:

[超级源点->重要度为2的点，流量为1，费用为0]

[曹战场->袁战场，流量无限，费用为人的雇佣成本]

[重要度为0的点->超级汇点，流量无限，费用为0]

跑一个最小费用最大流，如果最大流=重要度为2的点数，那么这个最小费用就是答案了。

只是时间复杂度不允许我们跑最小费用最大流。

然而我们发现，基于这道题的特殊性，其实直接求:

枚举每个重要度为2的点，

对于每个这样的点，求它到重要度为0的点中距离最近的那个的距离。

然后这些距离求和即可。

但是这样是多源最短路。

然而我们发现，只要把边逆过来，初始化所有重要度为0的点为起点，都归为ZERO，求最短路。

然后累加所有重要度为2的点的到 ZERO的最短路即可。

最小树形图

题意:

After awarded lands to ACMers, the queen want to choose a city be her capital. This is an important event in ice_cream world, and it also a very difficult problem, because the world have N cities and M roads, every road was directed. Wiskey is a chief engineer in ice_cream world. The queen asked Wiskey must find a suitable location to establish the capital, beautify the roads which let capital can visit each city and the project's cost as less as better. If Wiskey can't fulfill the queen's require, he will be punishing.

Input

Every case have two integers N and M (N<=1000, M<=10000), the cities numbered 0…N-1, following M lines, each line contain three integers S, T and C, meaning from S to T have a road will cost C.

Output

If no location satisfy the queen's require, you must be output "impossible", otherwise, print the minimum cost in this project and suitable city's number. May be exist many suitable cities, choose the minimum number city. After every case print one blank.

Sample Input

```
3 1
0 1 1

4 4
0 1 10
0 2 10
1 3 20
2 3 30
```

Sample Output

```
impossible

40 0
```

类型:

最小树形图-hdu-wf模版

Problem Description
Do you think this is a strange problem name? That is because you don't know its full name---'Good Good Study and Day Day Up!". Very famous sentence! Isn't it?

Now "GGS-DDU" is lzqxh's target! He has N courses and every course is divided into a plurality of levels. Just like College English have Level 4 and Level 6.

To simplify the problem, we suppose that the i-th course has Levels from level 0 to level a[i]. And at the beginning, lzqxh is at Level 0 of every course. Because his target is "GGS-DDU", lzqxh wants to reach the highest Level of every course.

Fortunately, there are M tutorial classes. The i-th tutoial class requires that students must reach at least Level L1[i] of course c[i] before class begins. And after finishing the i-th tutorial class, the students will reach Level L2[i] of course d[i]. The i-th tutoial class costs lzqxh money[i].

For example, there is a tutorial class only students who reach at least Level 5 of "Tiyu" can apply. And after finishing this class, the student's "MeiShu" will reach Level 10 if his "MeiShu"'s Level is lower than 10. (Don't ask me why! Supernatural class!!!")

Now you task is to help lzqxh to compute the minimum cost!

Input
The input contains multiple test cases.

The first line of each case consists of two integers, N (N<=50) and M (M<=2000).
The following line contains N integers, representing a[1] to a[N]. The sum of a[1] to a[N] will not exceed 500.
The next M lines, each have five integers, indicating c[i], L1[i], d[i], L2[i] and money[i] (1<=c[i], d[i]<=N, 0<=L1[i]<=a[c[i]], 0<=L2[i]<=a[d[i]], money[i]<=1000) for the i-th tutorial class. The courses are numbered from 1 to N.

The input is terminated by N = M = 0.

Output
Output the minimum cost for achieving lzqxh's target in a line. If his target can't be achieved, just output -1.

Sample Input
```
3 4
3 3 1
1 0 2 3 10
2 1 1 2 10
```

```
1 2 3 1 10
3 1 1 3 10
0 0
```

Sample Output
40

分析: 将每门course的每个level看作一个节点，每门course的第i个level向第i – 1个level连一条费用为0的有向边，如果有一个class需要选修人具有course a 的level l1， 修完后能使course b 达到level l2，那么也连一条从course a 的level l1 指向course b 的level l2的有向边，费用为这个class的学费；每门course的level 0 合并作为根节点，这样问题就转换成求以根节点出发， 到达其它所有节点的最小费用，这样就把问题转化成有向图的最小生成树(即最小树形图)问题。

---

floyd优化

```
inline void floyd(int x, int y)
{
    int qxnum = 0;
    int qynum = 0;
    for (int i = 1; i <= n; ++i)if (b[i][x] && !b[i][y])qx[++qxnum] = i;
    for (int i = 1; i <= n; ++i)if (b[y][i] && !b[x][i])qy[++qynum] = i;
    for (int i = 1; i <= qxnum; ++i)
    {
      for (int j = 1; j <= qynum; ++j)
      {
            b[qx[i]][qy[j]] = 1;
      }
    }
    b[x][y] = 1;
}
int main()
{
    while (~scanf("%d%d", &n, &m), n || m)
    {
      MS(b, 0);
      int ans = 0;
      for (int i = 1; i <= n; ++i)b[i][i] = 1;
      for (int i = 1; i <= m; ++i)
      {
            scanf("%d%d", &x, &y);
            if (b[y][x])++ans;
            else if (!b[x][y])floyd(x, y);
      }
      printf("%d. %d\n", ++casei, ans);
    }
    return 0;
}
```

/*
【trick&&吐槽】
1，if (b[y][x])++ans;
要先判定这个，因为我们也要避免自环出现。
2，一个比较厉害的优化，是针对floyd的。
我们可以找到哪些点只能连接x，哪些点只能连接y。
因为x-y这条边的加入，这两部分的点可以连通了。

【题意】
有n（234）个点，m（1e5）条有向边。
这m条边依次加入。
如果某一条加入的时候会形成环，我们就不能加入这条边。
问你有多少条边是不能被加入的。

【类型】
有向图找环

【分析】
有向图判环？tarjan？
然而tarjan的复杂度可是可以达到单次O(m)
这个图上同时最多是可能有n^2条边，于是tarjan的复杂度可达O(mn^2)爆炸！
那我们怎么办呢。。。dfs！
我们加了一条从x到y的边，我们就看看之前能否找到一个从y到x的环即可。
其实复杂度即时还是O(mn^2)
然而我们优化一下还是可以AC哒~

*/

---

hdu1452 积性函数+逆元

题目意思：2004^x的所有正因数的和(S)对29求余；输出结果；
原题链接
题目解析：解析参照来源：点击打开链接
因子和
6的因子是1,2,3,6; 6的因子和是s(6)=1+2+3+6=12;
20的因子是1,2,4,5,10,20; 20的因子和是s(20)=1+2+4+5+10+20=42;
2的因子是1,2; 2的因子和是s(2)=1+2=3;
3的因子是1,3; 3的因子和是s(3)=1+3=4;
4的因子和是 s(4)=1+2+4=7;
5的因子和是 s(5)=1+5=6;
s(6)=s(2)*s(3)=3*4=12;
s(20)=s(4)*s(5)=7*6=42;
这是巧合吗？
再看 s(50)=1+2+5+10+25+50=93=3*31=s(2)*s(25),s(25)=1+5+25=31.
这在数论中叫积性函数，当gcd(a,b)=1时s(a*b)=s(a)*s(b);

如果p是素数
s(p^n)=1+p+p^2+...+p^n=(p^(n+1)-1) /(p-1) (1)
例 hdu1452 Happy2004
计算 因子和 s(2004^X) mod 29,
2004=2^2 *3 *167
s(2004^X)) = (s(2^2X))) *(s(3^X))) * (s(167^X)))
167)=22;
s(2004^X)) = (s(2^2X))) *(s(3^X))) * (s(22^X)))
a=s(2^2X)=(2^(2X+1)-1)//根据 (1)
b=s(3^X)= (3^(X+1)-1)/2//根据 (1)
c=s(22^X)= (22^(X+1)-1)/21//根据 (1)
%运算法则 1. (a*b) %p= (a%p) *(b%p)
%运算法则 2. (a/b) %p= (a *b^(-1)%p)
b^(-1)是 b的逆元素 (%p)
2的逆元素是15 ()) ，因为2*15=30 % 29=1 % 29
21的逆元素是18 ()) ，因为21*18=378% 29 =1 % 29
因此
a= (powi(2,2*x+1,29)-1) %29;
b=(powi(3,x+1,29)-1)*15 %29;
c=(powi(22,x+1,29)-1)*18 %29;
ans= (a*b) % 29*c % 29;
资料拓展： 1. 高次幂快速取模链接
2.积性函数：在数论中的积性函数：对于正整数n的一个算术函数 f(n)，若f(1)=1，且当a,b互质时f(ab)=f(a)f(b)，在数论上就称它为积性函数。若对于某积性函数f(n) ，就算a, b不互质，也有f(ab)=f(a)f(b)，则称它为**完全积性**的。若将n表示成质因子分解式

$$n = p_1^{a_1} p_2^{a_2} \cdots p_n^{a_n}$$

则有

$$f(n) = f(p_1^{a_1}) f(p_2^{a_2}) \ldots f(p_n^{a_n})$$

3.求逆元：

在计算(a/b)%Mod时，往往需要先计算b%Mod的逆元p（b有逆元的条件是gcd(b,Mod)==1，显然素数肯定有逆元），然后由(a*p)%Mod得结果c。这里b的逆元p满足(b*p)%Mod=1。先来简单证明一下：
(a/b)%Mod=c; (b*p)%Mod=1; ==》 (a/b)*(b*p) %Mod=c; ==》 (a*p)%Mod=c;
从上面可以看出结论的正确性，当然这里b需要是a的因子。接下来就需要知道根据b和Mod，我们怎么计算逆元p了。扩展欧几里德算法，大家应该都知道，就是已知a、b，求一组解(x,y)使得a*x+b*y=1。这里求得的x即为a%b的逆元，y为b%a的逆元（想想为什么？把方程两边都模上b或a看看）。调用ExtGcd (b,Mod,x,y)，x即为b%Mod的逆元p。

求b%Mod的逆元p还有另外一种方法，即p=b^(Mod-2)%Mod，因为b^(Mod-1)%Mod=1(这里需要Mod为素数)。

Ubuntu 16.04 中的 Eclipse Mars 菜单栏选项失效

在 eclipse.ini 配置文件中加入以下这行:

```
--launcher.GTK_version
2
```

示例:

```
-startup
plugins/org.eclipse.equinox.launcher_1.3.100.v20150511-1540.jar
--launcher.GTK_version
2
-product
org.eclipse.epp.package.cpp.product
--launcher.defaultAction
openFile
-showsplash
org.eclipse.platform
--launcher.XXMaxPermSize
256m
--launcher.defaultAction
openFile
--launcher.appendVmargs
-vmargs
-Dosgi.requiredJavaVersion=1.7
-XX:MaxPermSize=256m
-Xms256m
-Xmx1024m
```