

**A - 梦后楼台高锁，酒醒帘幕低垂**

**Time Limit: 3000/1000MS (Java/Others)     Memory Limit: 65535/65535KB (Java/Others)**

Submit Status

给你一个有nn个点和mm条边的无向连通图,每条边都有一个权值ww.  
我们定义，对于一条路径，它的Charm valueCharm value为该路径上所有边的权值的最大值与最小值的差.  
询问从11到nn的所有路径的Charm valueCharm value的最小值.

**Input**

第一行有两个整数n,m(1≤n≤200,n−1≤m≤1000)n,m(1≤n≤200,n−1≤m≤1000)，表示该图有nn个点和mm条边.  
接下来mm行，每行三个整数u,v,w(1≤u,v≤n,1≤w≤1000000)u,v,w(1≤u,v≤n,1≤w≤1000000)，表示点uu和点vv之间有一条权值为ww的边.

**Output**

输出一个数，即从11到nn的所有路径的Charm valueCharm value的最小值.

**Sample input and output**

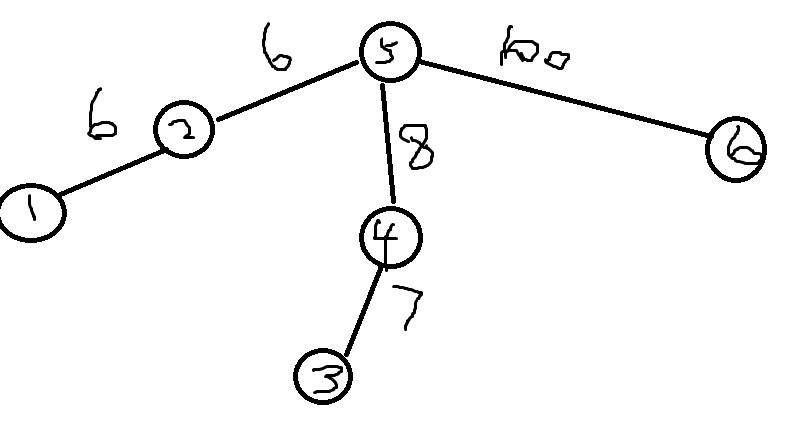
| **Sample Input** | **Sample Output** |
| --- | --- |
| 4 4  3 4 1  2 3 2  1 2 4  2 4 3 | 1 |

题意：

寻找无向联通图中两点间所有路径上边权差的最小值。

题解：

第一眼看到这道题，我便想是否需要用最短路算法来做。具体算法如下：在SPFA的迭代过程中不使路程最短，而是更新1号点到当前点的路径上的最大值和最小值，每次贪心取绝对值之差最小的那条路。可是，显然这样做是错的，因为到达某个点的路径上边权差最小并不能保证后续的路径上边权差最小。例如：



在这样的图中，在5号点贪心得路径1-2-5，然而3-4-5-6的走法比1-2-5-6的走法更优。

我们经过观察可以发现，此题的数据范围非常小，最多只有200个点和1000条边，因此我们可以随意使用较为暴力的算法。更详细地说，使用时间复杂度O(m^2)或O(n^3)(n为点数,m为边数)以内的算法都是可行的。而在题目中，只要1号点与n号点联通即可。于是，我们可以重新贪心：

将所有边按照边权值排序，从低到高依次枚举每一条边，以这条边为点1到点n路径上权值最小的边，此时找到使1号点和n号点联通要使用的权值最大边，题目的解则为最大边权减去最小边权，利用这一值更新最优解。因为共有m条边，所以总的时间复杂度是O(m^2).

如何去找每条边对应的权值最大边？我们只要将权值比当前枚举边大的边按升序依次将其的两个端点加入边集，判断此时点1和点n是否都在集合中即可。具体的实现，可以使用并查集。

如此，这道题便解了。

AC代码：

#include <cstdio>

#include <iostream>

#include <algorithm>

using namespace std;

const int maxn=205,maxk=1005;

int f[maxn],num;

struct Edge{

int from,to,dist;

};

Edge edge[maxk];

bool cmp(Edge a,Edge b) {

return a.dist<b.dist;

}

//并查集查找

int find(int now) {

if (f[now]==now) return now; else {

f[now]=find(f[now]);

return f[now];

}

}

void addedge(int f,int t,int d) {

edge[++num].from=f;

edge[num].to=t;

edge[num].dist=d;

}

int main() {

int n,m,i,j,u,v,w;

num=0;

scanf("%d%d",&n,&m);

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

scanf("%d%d%d",&u,&v,&w);

addedge(u,v,w);

}

sort(edge+1,edge+m+1,cmp);

int ans=0x3f3f3f3f;

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

for (j=1;j<=n;j++) f[j]=j;

int flag=0;

for (j=i;j<=m;j++) {

int fa=find(edge[j].from),fb=find(edge[j].to);

if (fa!=fb) {

f[fa]=fb; //并查集合并

}

if (find(1)==find(n)) { //此时1和n号点相连

flag=1;

break;

}

}

if (flag) ans=min(ans,edge[j].dist-edge[i].dist);

}

printf("%d",ans);

return 0;

}

# B - 去年春恨却来时，落花人独立，微雨燕双飞

##### Time Limit: 3000/1000MS (Java/Others)     Memory Limit: 65535/65535KB (Java/Others)

Submit Status

给你一个大小为nn的集合SS，集合里有nn个互不相同正整数.  
有qq个询问，每次询问是否能选择SS中的一些数字**( 同一个数字可以选择多次,也可以任何数字都不选)**，使它们相加的和为mm.

## Input

第一行一个数n(1≤n≤2000)n(1≤n≤2000),表示集合SS的大小.  
第二行nn个数，第ii个数ai(1≤ai≤50000)ai(1≤ai≤50000)表示集合SS中的第ii个数.  
第三行一个数q(1≤q≤10000)q(1≤q≤10000)，表示询问次数.  
接下来qq行，每行一个数m(0≤m≤1000000000)m(0≤m≤1000000000)，表示该次询问的数.

## Output

每次询问输出一行,如果存在和为mm的方法，输出YESYES,否则输出NONO.

## Sample input and output

| **Sample Input** | **Sample Output** |
| --- | --- |
| 3  2 4 9  4  6  7  18  25 | YES  NO  YES  YES |

## Hint

对于第一个询问，存在2+2+2=62+2+2=6,所以输出YESYES  
对于第一个询问，无法构造，输出NONO  
对于第三个询问，存在9+9=189+9=18,所以输出YESYES  
对于第四个询问，存在2+2+4+4+4+9=252+2+4+4+4+9=25,所以输出YESYES

题意：

给定最多2000个不超过50000的数，从其中取数字，每个数可以取无穷次，询问能否组成1e9以内给定的数。

题解：

看到这道题，第一想法是将其转换成完全背包问题求解。将每个数看做一件物品，数字的大小就是它的体积，也是它的价值。但是，我们注意到询问的数字范围最大可达1e9,也就是说背包的容量需要达到1e9，这样一来无论时间还是空间上使用背包问题的方法求解都是不能满足题目要求的。

此题可以转化为图论中的最短路问题。首先，我们注意到，对于任意一个数x，如果其可以被表示，那么它加上集合中任意一个数，也就是x+nAi必定也可以被表示。这样，我们只需要求对于集合A中任意一个数Ai,当X对Ai取余等于J(0<=J<=(Ai)-1)时可以表示出的最小值即可。对任意一次询问，若询问的数字为Y，只要查询余数为Y%Ai=J时取到的最小值是否不大于Y，如果成立则可以表示，反之则不可以。

对于这个问题，我们可以使用最短路算法。建立编号0,1,…(Ai-1)的Ai个点，当对Ai取余为u的数可以通过加上集合A中任意一个数Aj (j != i) 转化为余数为v的数时，用一条权值为Aj的有向边连接u点和v点。源点为0号点，最短距离为0，因为显而易见的是对于任意Ai的倍数，都可以只用Ai表示出来。

但是，我们看到Ai<=50000,N<=2000,最多时可能约有48000\*2000,即100,000,000条边，毫无疑问此时内存空间会爆炸。但是，转念一想，我们只需要每次取一个余数/点更新最短路，与Dijkstra中每次取一个点的方法完全相同，而所有的边根据选取的数与Ai之间取余的性质可以直接推出。如此，我们只需要根据选取的数字直接更新最短距离，而不需要将边全部保存下来。

最后一个问题，就是Ai取哪一个的问题了。我选取了最小的数字，这样不仅节省空间，还能减少点和边的数量，从而优化时间复杂度。此外，我的程序中还有一处优化：将数字从小到大排序，按照排序后的顺序选取数字更新最短路，若余数(Aj%Ai, j != i) 在取Aj之前已经出现过则可以不更新，因为此时边的权值(Aj)一定比之前选取的余数相同的数要大。最终AC用时为234ms。

AC代码：

#include <cstdio>

#include <iostream>

#include <queue>

#include <string.h>

#include <algorithm>

using namespace std;

const int maxn=50005;

int dist[maxn],a[maxn],visit[maxn],mark[maxn],b[maxn];

// 使用优先队列优化

struct node {

int id,dist;

node (int id,int dist): id(id),dist(dist) {}

bool operator <(const node &x)const {

return dist>x.dist;

}

};

priority\_queue<node> q;

int num,pnum;

int min(int a,int b) {

if (a<b) return a; else return b;

}

//迪杰斯特拉算法

void dijkstra (int n,int m) {

memset(visit,0,sizeof(visit));

int i;

while (!q.empty()) {

int now=q.top().id;

q.pop();

visit[now]=1;

for (i=0;i<n;i++) {

int to=(now+b[i])%m;

if (!visit[to]&&dist[now]+b[i]<dist[to]) {

dist[to]=dist[now]+b[i];

q.push(node(to,dist[to]));

}

}

}

}

int main() {

int n,i,j,x,m;

scanf("%d",&n);

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

scanf("%d",&a[i]);

}

sort(a+1,a+n+1);

m=a[1];

pnum=0;

memset(mark,0,sizeof(mark));

memset(dist,0x3f,sizeof(dist));

mark[0]=1;

q.push(node(0,0));

dist[0]=0;

b[pnum++]=a[1];

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

if (!mark[a[i]%m]) {

mark[a[i]%m]=1;

b[pnum++]=a[i];

dist[a[i]%m]=a[i];

q.push(node(a[i]%m,a[i]));

}

}

dijkstra(pnum,m);

int t;

scanf("%d",&t);

for (i=1;i<=t;i++) {

scanf("%d",&x);

if (dist[x%m]<=x) printf("YES\n"); else printf("NO\n");

}

return 0;

}

/\*

3

8 79 154

8

0 233 154 387 308 237 158 79

\*/

# C - 记得小苹初见，两重心字罗衣

##### Time Limit: 3000/1000MS (Java/Others)     Memory Limit: 65535/65535KB (Java/Others)

Submit Status

二维平面上有nn个点.  
你需要将每个点染成红色或蓝色.  
请设计一种染色方案能够使得每一行和每一列的红色的点的数量与蓝色的点的数量之差都不超过11.

## Input

第一行一个整数n(1≤n≤200000)n(1≤n≤200000),表示有nn个点.  
接下来nn行，每行两个整数x,y(1≤x,y≤200000)x,y(1≤x,y≤200000),表示第ii个点的坐标,保证任意两个点的坐标不相同. 保证输入至少存在一个可行解.

## Output

输出一行，一共nn个字母，第ii个字母表示第ii个点的颜色. 如果是红色，输出字母rr;如果是蓝色，输出字母bb.

## Sample input and output

| **Sample Input** | **Sample Output** |
| --- | --- |
| 3  1 1  1 2  2 1 | brr |

题意：

如题所述，将整数坐标系中的点用两种颜色染色使同行、同列的两种颜色点数绝对值差不大于1。

题解：

此题一开始考虑用二分图匹配做，然而始终无法建出相应的正确的图论模型。正确解法并没有这么复杂，只需要一个dfs就可以搞定。

解法如下：将同一行的n个点连成n/2条无向边，使每个点出现至多1次。Dfs时，从任意一个点开始，dfs遍历包含这个点的整个强连通分量，交替染色。即，将源点染色后，将其所有相连的未染色点染成相反色，再dfs所有子节点。这样建图后，同一个强连通分量当中不可能出现矛盾的情况，因为此时从一个点到另一个点横向和纵向移动的次数总是相等。

在我的程序中，并没有采用dfs实现，而是用到了一个队列q进行分层染色。

AC代码：

#include <cstdio>

#include <iostream>

#include <vector>

#include <string.h>

using namespace std;

const int maxn=200005;

vector<int> x[maxn],y[maxn],v[maxn];

int q[maxn];

int color[maxn];

//染色，0为b,1为r

void dfs(int s,int c) {

int i,k,j;

color[s]=c;

int top=1,tail=0;

q[1]=s;

k=c^1; //k为当前颜色

while (top>tail) {

int num=top;

for (j=tail+1;j<=num;j++) {

int p=q[j];

for (i=0;i<v[p].size();i++) {

if (color[v[p][i]]==-1) {

color[v[p][i]]=k;

q[++top]=v[p][i]; //将染色后的点压入队列

}

}

}

tail=num;

k=k^1;

}

}

int main() {

int n,i,a,b;

scanf("%d",&n);

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

scanf("%d%d",&a,&b);

//按照题解所述方法建图

if (x[a].size()%2) {

v[i].push\_back(x[a][x[a].size()-1]);

v[x[a][x[a].size()-1]].push\_back(i);

}

if (y[b].size()%2) {

v[i].push\_back(y[b][y[b].size()-1]);

v[y[b][y[b].size()-1]].push\_back(i);

}

x[a].push\_back(i);

y[b].push\_back(i);

}

memset(color,-1,sizeof(color));

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

if (color[i]==-1) dfs(i,0);

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

if (color[i]==1) printf("r"); else printf("b");

return 0;

}

**D - 琵琶弦上说相思，当时明月在，曾照彩云归**

**Time Limit: 3000/1000MS (Java/Others)     Memory Limit: 65535/65535KB (Java/Others)**

Submit Status

给你nn个仅由小写字母组成的字符串，请你找出一种[字典序](http://baike.baidu.com/link?url=hQh_mlLd0HTterU5YUtG5hFcUFasfGCizmTQ4erRgDK-bjQGzwp2kYzzCc9QiUuaXwnWzTOZ0PtYdYC65j_YA6cuFXC6q8DATSIkbZS9OI8IklqLsvuTTKJrXGdj8c41)，使得这nn个字符串在这种字典序下是从小到大排列的.

**Input**

第一行一个整数n(1≤n≤1000)n(1≤n≤1000),表示有nn个字符串.  
接下来nn行，每行一个字符串，每个字符串的长度不超过200200，不含空串.

**Output**

如果无解，输出−1−1.  
如果有解，输出一个长度为2626的字符串，其中第ii个字母表示这种字典序下第ii小的字母.  
如果有多个解，输出在字典序"abcdefghijklmnopqrstuvwxyz""abcdefghijklmnopqrstuvwxyz"下最小的那一个.

**Sample input and output**

| **Sample Input** | **Sample Output** |
| --- | --- |
| 10  petr  egor  endagorion  feferivan  ilovetanyaromanova  kostka  dmitriyh  maratsnowbear  bredorjaguarturnik  cgyforever | aghjlnopefikdmbcqrstuvwxyz |

题意：

如题所述，寻找一种新的字典序使其符合输入字符串的顺序。要求输出的字典序尽量小。

题解：

我们注意到，要使相邻两个字符串顺序符合要求，只要取它们从左到右第一个不同的字符建立大小关系。如果它们前面的字符都相同，且前面的字符串长度大于后面的字符串，则无解。

对于建立大小关系的字母，我们可以将每个字母视为图中的一个节点，从大的字母到小的字母连接一条有向边，最后对建立的图进行拓扑排序，拓扑序在前的字母排名靠前。若此时图中有环，则题目无解。

对于未参与排序的字母，因为题目要求输出字典序最小，我们可以采取这样一种策略：拓扑排序时从’a’一直找到’z’,若有未参与排序且字典序排在入度为0，出度不为0的字母之前的字母，可以优先取这个字母。这样，可以保证输出的字典序最小。

AC代码：

#include <cstdio>

#include <iostream>

#include <string.h>

#include <vector>

#include <queue>

using namespace std;

const int maxl=205;

int mark[27],d[27];

char s[27],c[27];

char a[maxl],b[maxl];

vector<int> v[27];

int main() {

int n,i,j;

scanf("%d",&n);

scanf("%s",a);

memset(d,0,sizeof(d));

memset(mark,0,sizeof(mark));

for (i=2;i<=n;i++) {

scanf("%s",b);

int lena=strlen(a),lenb=strlen(b),k=0;

while (k<lena&&k<lenb&&a[k]==b[k])

k++;

if (k>=lena||k>=lenb) { //第一种无解情况

if (lena>lenb) {

cout << -1;

return 0;

}

} else { //建图

d[b[k]-'a']++;

v[a[k]-'a'].push\_back(b[k]-'a');

}

for (j=0;j<lenb;j++) a[j]=b[j];

a[lenb]='\0';

}

i=0;

queue<int> q;

while (i<26) {

int flag=0;

for (j=0;j<26;j++) { //寻找可以排名的节点，将其排在字典序尾部

if (!mark[j]&&d[j]==0) {

s[i]=(char)('a'+j);

i++;

mark[j]=1;

q.push(j);

flag=1;

if (v[j].size()!=0) break;

}

}

if (!flag) { //图中存在环，无解

cout << -1;

return 0;

}

while (!q.empty()) { //减去已排序的节点相连的边所产生的度数

int now=q.front();

q.pop();

for (j=0;j<v[now].size();j++) d[v[now][j]]--;

}

}

s[26]='\0';

printf("%s",s);

return 0;

}

# E - 红藕香残玉簟秋，轻解罗裳，独上兰舟。

##### Time Limit: 4000/2000MS (Java/Others)     Memory Limit: 65535/65535KB (Java/Others)

Submit Status

考试结束后，为了证明自己才是最蒻的，同学们纷纷去找老师查分阅数—哦不—是查阅分数。

可老师担心学生知道自己的成绩会伤心，于是只告诉学生这样的信息：

**编号为**uu**的学生分数比编号为**vv**的学生分数高**ww**分甚至更多。**

知道这些信息后，同学们想知道自己分数可能的 **最小值** 和 **最大值** 。不过老师记性不太好，给出的信息可能有误。

## Input

第一行两个整数 nn 和 mm，表示学生个数和老师给的信息数。

接下来 mm 行每行三个整数 uu 、vv 和 ww，含义如上文所描述。

学生从 11 到 nn 编号，学生的分数为 00 到 100100 之间的整数。

1≤n≤1000001≤n≤100000，1≤m≤10000001≤m≤1000000，1≤u1≤u 、v≤nv≤n，0≤w≤1000≤w≤100 。

## Output

若老师给出的信息有误，仅输出一行 −1−1 。

否则输出 nn 行，第 ii 行为以空格隔开的两个整数，分别表示编号为 ii 的学生的分数可能的 **最小值** 和 **最大值** 。

## Sample input and output

| **Sample Input** | **Sample Output** |
| --- | --- |
| 2 2  1 2 1  2 1 1 | -1 |
| 3 2  1 2 1  2 3 1 | 2 100  1 99  0 98 |

题意：

求解最多1,000,000组不等式Xi-Yi>=Bi的解，要求解在0到100之间。

题解：

这道题是典型的差分约束问题，然而我提交了将近30次。究其原因，是因为没有考虑一种特殊情况：一般的差分约束问题中，出现环则认为问题无解。然而此题中，当不等式右侧的常数为0时，此时建图可能出现边权和为0的环，也就是此时环内每个数字都相等，这时求解的结果也是符合题目要求的。

回到差分约束问题本身。对于求解最大值和最小值，方法类似：求最大值时，将不等式转换为y[i]-x[i]>=-b[i],建立从x[i]到y[i]的长度为-b[i]的边，求解最短路；求最小值时，将不等式转换为x[i]-y[i]>=b[i],建立从y[i]到x[i]的长度为b[i]的边，求解最长路。

那么如何处理开头所述的特殊情况呢？此时我们需要先对原图利用tarjan算法求出强连通分量，如果同一个强连通分量中的边权和为0则合法，否则不合法。在tarjan算法求出强连通分量并染色缩点后，再对缩点后的DAG求解最短路和最长路。

AC代码：

#include <cstdio>

#include <queue>

#include <iostream>

#include <vector>

#include <stack>

#include <string.h>

using namespace std;

const int maxn=100005,maxk=1000005;

const int inf=0x3f3f3f3f;

int du[maxn],distmax[maxn],distmin[maxn],visit[maxn],num,n,m,cnum;

int color[maxn\*2],dfn[maxn],low[maxn],instack[maxn];

int x[maxk],y[maxk],b[maxk];

vector<int> v[maxn];

vector<int> d[maxn];

vector<int> v2[maxn];

stack<int> st;

int min(int a,int b) {

if (a>b) return b; else return a;

}

int max(int a,int b) {

if (a<b) return b; else return a;

}

//tarjan强连通分量算法

void tarjan(int now,int n) {

dfn[now]=low[now]=++num;

int i;

st.push(now);

instack[now]=1;

for (i=0;i<v[now].size();i++) {

int to=v[now][i];

if (!dfn[to]) {

tarjan(to,n);

low[now]=min(low[now],low[to]);

} else if (instack[to])

low[now]=min(low[now],dfn[to]);

}

if (low[now]==dfn[now]) {

color[now+n]=++cnum;

instack[now]=0;

while (st.top()!=now) {

color[st.top()+n]=cnum;

instack[st.top()]=0;

st.pop();

}

st.pop();

}

}

//tarjan染色

bool coloring(int n,int p) {

int i;

memset(instack,0,sizeof(instack));

memset(dfn,0,sizeof(dfn));

memset(low,0,sizeof(low));

num=0;cnum=0;

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

if (!color[i+p]) tarjan(i,p);

memset(du,0,sizeof(du));

}

//缩点，拓扑排序求最短路求出最大值

bool solvemax(int n) {

coloring(n,0);

int i,j; //y[i]-x[i]>=-b[i]

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

if (color[x[i]]==color[y[i]]&&b[i]!=0) {

return false;

} else if (color[x[i]]!=color[y[i]]) {

d[color[x[i]]].push\_back(-b[i]);

v2[color[x[i]]].push\_back(color[y[i]]);

du[color[y[i]]]++;

}

}

memset(distmin,0x3f,sizeof(distmin));

queue<int> q;

int cnt=0;

for (i=1;i<=cnum;i++)

if (!du[i]) {

q.push(i);

distmin[i]=100;

}

while (!q.empty()) {

int now=q.front();

q.pop();

cnt++;

for (j=0;j<v2[now].size();j++) {

int to=v2[now][j];

distmin[to]=min(distmin[to],distmin[now]+d[now][j]);

du[to]--;

if (du[to]==0)

q.push(to);

}

}

if (cnt<cnum) return false;

for (i=1;i<=cnum;i++)

if (distmin[i]<0) return false;

return true;

}

//缩点，拓扑排序求最长路求出最小值

bool solvemin(int n) {

coloring(n,n);

int i,j; //x[i]-y[i]>=b[i]

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

if (color[x[i]+n]==color[y[i]+n]&&b[i]!=0) {

return false;

} else if (color[x[i]+n]!=color[y[i]+n]) {

d[color[y[i]+n]].push\_back(b[i]);

v2[color[y[i]+n]].push\_back(color[x[i]+n]);

du[color[x[i]+n]]++;

}

}

memset(distmax,-0x3f,sizeof(distmax));

queue<int> q;

int cnt=0;

for (i=1;i<=cnum;i++)

if (!du[i]) {

q.push(i);

distmax[i]=0;

}

while (!q.empty()) {

int now=q.front();

q.pop();

cnt++;

for (j=0;j<v2[now].size();j++) {

int to=v2[now][j];

distmax[to]=max(distmax[to],distmax[now]+d[now][j]);

du[to]--;

if (du[to]==0)

q.push(to);

}

}

if (cnt<cnum) return false;

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

if (distmax[color[i+n]]>distmin[color[i]]) return false;

return true;

}

int main() {

int i;

scanf("%d%d",&n,&m);

memset(color,0,sizeof(color));

/\* n=12;

m=n/2\*3-1;

for (i=1;i<=n/2;i++) {

x[2\*i-1]=2\*i;y[2\*i-1]=2\*i-1;

x[2\*i]=2\*i-1;y[2\*i]=2\*i;

b[2\*i-1]=b[2\*i]=0;

if (i!=n/2) {

x[n+i]=i\*2-1;y[n+i]=i\*2+1;b[n+i]=20;

}

}\*/

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

scanf("%d%d%d",&x[i],&y[i],&b[i]);

v[x[i]].push\_back(y[i]);

}

int flag1=solvemax(n);

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

v[i].clear();

v2[i].clear();

d[i].clear();

}

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

v[y[i]].push\_back(x[i]);

}

int flag2=solvemin(n);

if (!flag1||!flag2) {

printf("-1");

} else {

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

printf("%d %d\n",distmax[color[i+n]],distmin[color[i]]);

}

return 0;

}

/\*

6 5

1 2 20

2 3 20

3 4 20

4 5 20

5 6 20

\*/

# F - 云中谁寄锦书来？雁字回时，月满西楼。

##### Time Limit: 6000/3000MS (Java/Others)     Memory Limit: 65535/65535KB (Java/Others)

Submit Status

在干完 TheBigOneTheBigOne（一票大的）之后，劫犯们得准备逃跑路线了。

城市可以看作 nn 个点 mm 条边的无向图，节点编号为 00 到 n−1n−1，其中有 kk 个点为安全屋，劫犯们只要到达其中一个安全屋就能摆脱警察的追捕。

劫犯们从联合储蓄（节点 00）出发，希望能在最短的时间内到达安全屋。

但是警察对劫犯紧追不舍，每当劫犯到达一个节点，警察就立刻封锁与该点相连的边。

由于警力有限，对于当前点警察最多能够封锁与其相连的 dd 条边。

现在劫犯想知道，在最坏情况下，他们能到达安全屋的最短时间。

## Input

第一行四个整数 nn 、mm 、kk 和 dd，含义如上文所描述。

接下来 mm 行每行三个整数 uu 、vv 和 ww，表示节点 uu 和 vv 之间有一条边，且通过该条边要花费 ww 的时间。

接下来一行有 kk 个整数，表示安全屋所在节点编号。

1≤n≤1000001≤n≤100000，1≤m≤10000001≤m≤1000000，0≤k≤n0≤k≤n，0≤d≤m0≤d≤m，0≤u0≤u 、v<nv<n，0≤w≤100000≤w≤10000 。

## Output

若劫犯们不能到达安全屋，则输出 −1−1 。

否则输出最坏情况下劫犯们到达安全屋的最短时间。

## Sample input and output

| **Sample Input** | **Sample Output** |
| --- | --- |
| 3 3 1 0  0 1 1  1 2 1  0 2 3  2 | 2 |
| 3 3 1 1  0 1 1  1 2 1  0 2 3  2 | -1 |

题意：  
 最多100,000个点，1,000,000条边的无向图，源点唯一，汇点有多个，且在每个点可以用最优方法(尽量使路径长)使路径不通过与其相连的d条边，询问源点到任意一个汇点的最短路的最小值。

题解：

此题的一个难点是：如何确认在每个点封锁哪些路是最优的？如果我们从源点出发寻找路径，我们很难判断这一点，因为某点之前的最短路并不等价于之后的路径也最短，更何况这题有多个汇点。这时，我们可以转换思维，从汇点开始寻找最短路径。

具体方法如下：每次从已经找到最短路的值的点开始逆推，将与相连的所有边加入优先队列，根据边权与初始点到该点最短距离的和排序，每次取和最小的边更新最短路。如果边所指向的节点已经被更新d次，则可以更新；若少于d次，则不更新。我的程序中用t数组储存每个点已经被更新多少次。最开始时，我们默认多个汇点处的最短路径值已经确定且为0.

这一算法实际上是迪杰斯特拉算法的一种变形。

AC代码：

#include <cstdio>

#include <iostream>

#include <string.h>

#include <queue>

using namespace std;

const int maxn=100005,maxk=2000005,inf=0x3f3f3f3f;

int dist[maxn],t[maxn],head[maxn],num,safe[maxn],mark[maxk];

struct node {

int id,dist;

node(int id,int dist):id(id),dist(dist) {}

bool operator <(const node &x)const {

return dist>x.dist;

}

};

priority\_queue<node> pq;

struct Edge {

int from,to,dist,pre;

};

Edge edge[maxk];

void addedge(int f,int t,int d) {

edge[num].from=f;edge[num].to=t;edge[num].dist=d;

edge[num].pre=head[f];head[f]=num++;

edge[num].from=t;edge[num].to=f;edge[num].dist=d;

edge[num].pre=head[t];head[t]=num++;

}

int dj(int k) {

int ans=inf,j,i;

while (!pq.empty()) {

i=pq.top().id;

int from,to;

from=edge[i].from;

to=edge[i].to;

pq.pop();

if (t[to]==k&&edge[i].dist+dist[from]<dist[to]) {

dist[to]=edge[i].dist+dist[from];

for (j=head[to];j!=-1;j=edge[j].pre) {

if (!mark[j]) {

if (edge[j].from==to)

pq.push(node(j,edge[j].dist+dist[to]));

else

pq.push(node(j^1,edge[j].dist+dist[to]));

mark[j]=mark[j^1]=1;

}

}

} else t[to]++;

}

return dist[0];

}

int main() {

int n,m,k,i,j,x,y,w,d;

num=0;

memset(head,-1,sizeof(head));

memset(t,0,sizeof(t));

memset(mark,0,sizeof(mark));

memset(dist,0x3f,sizeof(dist));

scanf("%d%d%d%d",&n,&m,&k,&d);

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

scanf("%d%d%d",&x,&y,&w);

addedge(x,y,w);

}

for (i=1;i<=k;i++) {

scanf("%d",&safe[i]);

t[safe[i]]=d;

dist[safe[i]]=0;

for (j=head[safe[i]];j!=-1;j=edge[j].pre) {

if (!mark[j]) {

if (edge[j].from==safe[i]) pq.push(node(j,edge[j].dist));

else pq.push(node(j^1,edge[j].dist));

mark[j]=mark[j^1]=1;

}

}

}

int ans=dj(d);

if (ans==inf) cout << -1; else printf("%d\n",ans);

return 0;

}

# G - 花自飘零水自流，一种相思，两处闲愁。

##### Time Limit: 2000/1000MS (Java/Others)     Memory Limit: 65535/65535KB (Java/Others)

Submit Status

在一座城市中，有 nn 个地区，地区之间由 mm 条单行道连接。

某游客由机场出发，通过这些单行道游览城市，**最后回到机场**。

由于游客可能不熟悉道路，该城市的法律能够容忍游客在单行道上逆行一次。

现在该游客想知道他最多能游历多少个不同的地区。

## Input

第一行两个整数 nn 和 mm，表示地区和单行道的数量。

接下来 mm 行每行两个整数 xx 和 yy，表示有一条 xx 到 yy 的单行道。

地区由 11 到 nn 进行编号，且**机场位于**11**号地区**。

1≤n1≤n 、m≤100000m≤100000，1≤x1≤x 、y≤ny≤n 。

## Output

输出一个整数，即最多逆行一次的情况下游历的地区数的最大值。

## Sample input and output

| **Sample Input** | **Sample Output** |
| --- | --- |
| 3 3  1 2  2 3  3 1 | 3 |
| 3 3  1 2  2 3  1 3 | 3 |

# 题意：

# 在有向图中，可以将其中一条边变为无向边，每条边可以走多次，询问从源点开始最多遍历多少个地区再回到源点。

# 题解： 一个显而易见的事实是：同一个强连通分量内的所有点可以互相到达，逆行同一个强连通分量内的边毫无意义。那么对于题目中所给的图，我们可以先使用tarjan算法将其转化为DAG，令缩点后的每个点点权为该强连通分量之内的点数，之后再对这个图进行处理。

# 对于缩点后的DAG的每条边，如果逆行从u到v的边，即添加从v到u的边并经过该边，那么我们可以计算从源点到v、u到源点所经过的最多点数再相加，所得结果取最大值就是答案。由于上述过程中源点所在的联通块被计算了两次，所以最后还需要将源点所在联通块的点数减掉。

# 源点到每个点经过的最大点权，可以通过spfa求出。每个点到源点的最大点权应如何求解呢？此时将原图中所有边反向，再求源点到每个点的最大点权即是答案。

# AC代码：

#include <vector>

#include <cstdio>

#include <iostream>

#include <string.h>

#include <queue>

#include <stack>

using namespace std;

const int maxn=100005;

int low[maxn],color[maxn],dfn[maxn];

int x[maxn],y[maxn],val[maxn],comed[maxn],god[maxn];

bool instack[maxn],inque[maxn];

vector<int> v[maxn];

vector<int> e1[maxn];

vector<int> e2[maxn];

stack<int> st;

int num,n,m,cnum;

int min(int a,int b) {

if (a>b) return b; else return a;

}

int max(int a,int b) {

if (a<b) return b; else return a;

}

//tarjan算法求强连通分量

void tarjan(int now) {

num++;

dfn[now]=low[now]=num;

instack[now]=1;

st.push(now);

for (int i=0;i<v[now].size();i++) {

int to=v[now][i];

if (!dfn[to]) {

tarjan(to);

low[now]=min(low[now],low[to]);

}

else if (instack[to])

low[now]=min(low[now],dfn[to]);

}

if (dfn[now]==low[now]) {

instack[now]=0;

color[now]=++cnum;

val[cnum]++;

while (st.top()!=now) {

color[st.top()]=cnum;

val[cnum]++;

instack[st.top()]=0;

st.pop();

}

st.pop();

}

}

//预处理源点到每个点经过的最大点权

void spfato(int s){

int i;

memset(inque,0,sizeof(inque));

memset(comed,0,sizeof(comed));

inque[s]=1;

queue<int> q;

q.push(s);

comed[s]=val[s];

while (!q.empty()) {

int now=q.front();

q.pop();

inque[now]=0;

for (i=0;i<e1[now].size();i++) {

int to=e1[now][i];

if (comed[now]+val[to]>comed[to]) {

if (!inque[to]) {

inque[to]=1;

q.push(to);

}

comed[to]=comed[now]+val[to];

}

}

}

}

//预处理每个点到源点的最大点权

void spfafrom(int s){

int i;

memset(inque,0,sizeof(inque));

memset(god,0,sizeof(god));

inque[s]=1;

queue<int> q;

q.push(s);

god[s]=val[s];

while (!q.empty()) {

int now=q.front();

q.pop();

inque[now]=0;

for (i=0;i<e2[now].size();i++) {

int to=e2[now][i];

if (god[now]+val[to]>god[to]) {

if (!inque[to]) {

inque[to]=1;

q.push(to);

}

god[to]=god[now]+val[to];

}

}

}

}

int main() {

int i;

scanf("%d%d",&n,&m);

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

scanf("%d%d",&x[i],&y[i]);

v[x[i]].push\_back(y[i]);

}

memset(dfn,0,sizeof(dfn));

memset(low,0,sizeof(low));

memset(instack,0,sizeof(instack));

memset(color,0,sizeof(color));

memset(val,0,sizeof(val));

num=cnum=0;

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

if (!color[i]) tarjan(i);

num=0;

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

if (color[x[i]]!=color[y[i]])

e1[color[x[i]]].push\_back(color[y[i]]);

spfato(color[1]);

num=0;

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

if (color[x[i]]!=color[y[i]])

e2[color[y[i]]].push\_back(color[x[i]]);

spfafrom(color[1]);

int ans=1;

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

if (color[x[i]]!=color[y[i]]&&god[color[x[i]]]!=0&&comed[color[y[i]]]!=0)

ans=max(ans,comed[color[y[i]]]+god[color[x[i]]]-val[color[1]]);

printf("%d",ans);

return 0;

}

/\*

8 9

1 5

1 2

4 5

2 7

7 8

8 2

2 5

4 3

3 6

ans=5

\*/

# H - 此情无计可消除，才下眉头，却上心头。

##### Time Limit: 2000/1000MS (Java/Others)     Memory Limit: 65535/65535KB (Java/Others)

Submit Status

有一个含 nn 项的数列 A1A1 、A2A2 、... 、AnAn 。

虽然不知道每一项的具体值，但我们知道每一项的值要么为 00，要么为 11 。

而且我们可以花 Ci,jCi,j 的费用询问数列中第 ii 项到第 jj 项的异或和。

第 ii 项到第 jj 项的异或和为 AiAi ⊕ Ai+1Ai+1 ⊕ ... ⊕ AjAj 。

其中 00 ⊕ 00 = 00 、00 ⊕ 11 = 11 、11 ⊕ 00 = 11 、11 ⊕ 11 = 00 。

那么我们至少要花多少费用才能知道数列中每一项的具体值呢？

## Input

第一行一个整数 nn，表示数列的项数。

接下来 nn 行中，第 ii 行有 n+1−in+1−i 个整数，即 Ci,iCi,i 、Ci,i+1Ci,i+1 、... 、Ci,nCi,n 。

1≤n≤10001≤n≤1000，1≤Ci,j≤10000001≤Ci,j≤1000000 。

## Output

输出一个整数，即要花的最小费用。

## Sample input and output

| **Sample Input** | **Sample Output** |
| --- | --- |
| 2  1 2  2 | 3 |
| 3  1 2 3  2 1  3 | 4 |

题意：

如题所述。

题解：

一开始无脑写了个最小费用最大流，果然WA了。

仔细观察后，我们发现要求出结果，至少需要查询n次，且每个节点必须作为一段区间的边界被查询，因为异或操作的结果相当于告诉你区间内每个数位上0和1的个数是奇数还是偶数。将每个数视为节点，N个节点选取N条边且覆盖所有点，答案可能会与最小生成树有关。

正解如下：对于每次查询区间[l,r]，建立从l-1到r号节点的权值为查询花费的边，最后直接求从0到N号节点的最小生成树即可。一旦想到方法，实现起来就是一个裸的最小生成树，使用kruskal或prim均可。

AC代码：

#include <cstdio>

#include <iostream>

#include <vector>

#include <algorithm>

#include <string.h>

using namespace std;

const int maxn=1005,maxm=1000005;

int head[maxn],num,f[maxn];

struct Edge{

int dist,from,to,pre;

};

Edge edge[maxm];

void addedge(int s,int t,int d) {

Edge e;

e.from=s;e.to=t;e.dist=d;e.pre=head[s];

edge[num]=e;head[s]=num++;

}

bool cmp(Edge a,Edge b) {

return a.dist<b.dist;

}

//并查集

int find (int now) {

if (f[now]==now) return now; else {

f[now]=find(f[now]);

return f[now];

}

}

//kruskal算法求最小生成树

int kruskal(int n) {

int ans=0,cnt=0,i;

for (i=0;i<=n;i++) f[i]=i;

for (i=0;i<num;i++) {

int fa=find(edge[i].from);

int fb=find(edge[i].to);

if (fa!=fb) {

cnt++;

ans+=edge[i].dist;

if (cnt==n) return ans;

f[fa]=fb;

}

}

return 0;

}

int main() {

int n,i,j,x;

scanf("%d",&n);

num=0;

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

for (j=i;j<=n;j++) {

scanf("%d",&x);

addedge(i-1,j,x);

}

}

sort(edge,edge+num,cmp);

int ans=kruskal(n);

printf("%d",ans);

return 0;

}

**I - 老当益壮， 宁移白首之心？**

**Time Limit: 3000/1000MS (Java/Others)     Memory Limit: 65535/65535KB (Java/Others)**

Submit Status

请构造一个01串，使其满足以下条件：

* 环状（即首尾相连）
* 每一位取值为0或1
* 长度是2n2n
* 对于每个(2n2n个)位置，从其开始沿逆时针方向的连续的n位01串（包括自己）构成的数均不相同，即0到2n−12n−1中的数各出现一次

**Input**

输入一个整数n(1≤n≤15)n(1≤n≤15)

**Output**

输出任一一个长度为2n2n且满足题意的01串（顺逆时针均可），保证输入有解。

**Sample input and output**

| **Sample Input** | **Sample Output** |
| --- | --- |
| 3 | 00010111 |

**Hint**

样例的0001011100010111,对于每个位置，沿逆时针方向连续长度为3的01串有：000,001,010,101,011,111,110,100000,001,010,101,011,111,110,100,即为0−70−7的所有数字

题意：

如题目所述。

题解：

此题的正解是欧拉回路。不过很诡异的是，我用Dfs也顺利AC，并且在没有优化的情况下5ms AC.

搜索的思路很简单：观察可知0一定会作为数字出现，此时一定会出现n个数字0连在一起的情况，我们先将01串前n位确定为0.接着，对于剩余的每一位只需穷举取0或取1，并判断是否合法(01串最后n位组成的数字是否出现过)，不合法时直接回溯修改之前的数字即可。

值得注意的是，搜索到最后一位时，需要多判断一下最后n位与开头n位相连的情况。

欧拉回路解法中将最后n位数组成的数字添加0或1能转变的数字视为一条边，从而建立图论模型，也是一种合理的解法。

AC代码：

#include <cstdio>

#include <string.h>

#include <iostream>

using namespace std;

int a[1<<16],mark[1<<16];

int n,ans;

bool dfs(int depth,int num) {

if (mark[num]) return false;

else mark[num]=1;

if (depth>(1<<n)){

int j=num;

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

j=(j%(1<<(n-1)))\*2+a[i];

if (mark[j]) {

mark[num]=0;

return false;

}

}

return true;

} else {

if (dfs(depth+1,(num%(1<<(n-1)))\*2)) return true;

a[depth]=1;

if (dfs(depth+1,(num%(1<<(n-1)))\*2+1)) return true;

a[depth]=0;

}

mark[num]=0;

return false;

}

int main() {

int i;

cin >> n;

memset(a,0,sizeof(a));

memset(mark,0,sizeof(mark));

if (dfs(n+1,0))

for (i=1;i<=1<<n;i++) cout << a[i];

return 0;

}

/\*

4

0000100110101111

\*/

# J - 穷且益坚， 不坠青云之志。

##### Time Limit: 6000/2000MS (Java/Others)     Memory Limit: 65535/65535KB (Java/Others)

Submit Status

求一个有nn个元素的数列，满足任意连续pp个数的和不小于ss，任意连续qq个数的和不大于tt。

## Input

输入包括五个数n,p,q,s,t(1≤n,p,q≤500,1≤s,t≤100000)n,p,q,s,t(1≤n,p,q≤500,1≤s,t≤100000)

## Output

如果不能做到，输出"No"

否则输出"Yes"，并在第二行输出一个可行解，包括n个数,保证在intint范围内有解。

## Sample input and output

| **Sample Input** | **Sample Output** |
| --- | --- |
| 4 2 2 1 2 | Yes  1 0 1 0 |

题意：

如题所述。

题解：

与E题类似，这题也可以通过差分约束求解。只不过，这题相比之下少了很多坑。

令sum[i]表示数列中从开头到i号元素的和，则题目的约束条件可以表示为：

sum[p+i]-sum[i]>=s (0<=i<=n-p)

sum[i]-sum[q+i]>=-t (0<=i<=n-q)

建立从i到i+p，权值为-s的边；建立从i到i+q，权值为t的边，接着利用spfa算法求出源点到各个点的最短路，保存在sum数组中就可以了。源点为0号点，初始值为0.

AC代码：

#include <cstdio>

#include <queue>

#include <deque>

#include <iostream>

#include <vector>

#include <string.h>

using namespace std;

const int maxn=505,maxk=1005;

const int inf=0x3f3f3f3f;

int dist[maxn],num,n,m;

int head[maxn],t[maxn];

bool inque[maxn];

vector<int> v[maxn];

vector<int> d[maxn];

//spfa算法

bool spfa(int s,int n) {

deque<int> q;

memset(inque,0,sizeof(inque));

memset(t,0,sizeof(t));

q.push\_back(s);

dist[s]=0;

inque[s]=1;

t[s]=1;

int i;

while (!q.empty()) {

int now=q.front();

q.pop\_front();

inque[now]=0;

for (i=0;i<v[now].size();i++) {

int to=v[now][i];

if (dist[now]+d[now][i]<dist[to]) {

dist[to]=dist[now]+d[now][i];

if (!inque[to]) {

if (!q.empty()) {

if (dist[to]<dist[q.front()]) //small label first优化

q.push\_front(to);

else q.push\_back(to);

} else q.push\_back(to);

t[to]++;

if (t[to]>n) return false;

inque[to]=1;

}

}

}

}

return true;

}

int main() {

int n,p,q,s,t,i,j;

scanf("%d%d%d%d%d",&n,&p,&q,&s,&t);

memset(dist,0x3f,sizeof(dist));

//sum[p+i]-sum[i]>=s

for (i=0;p+i<=n;i++) {

v[i+p].push\_back(i);

d[i+p].push\_back(-s);

}

//sum[i]-sum[q+i]>=-t

for (i=0;q+i<=n;i++) {

v[i].push\_back(q+i);

d[i].push\_back(t);

}

int flag=1;

flag=flag&spfa(0,n);

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

if (dist[i]==inf) flag=flag&spfa(i,n);

}

if (flag) {

cout << "Yes" << endl;

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

printf("%d ",dist[i]-dist[i-1]);

}

} else cout << "No";

return 0;

}

# K - 酌贪泉而觉爽， 处涸辙以犹欢。

##### Time Limit: 6000/2000MS (Java/Others)     Memory Limit: 65535/65535KB (Java/Others)

Submit Status

在UESTCUESTC，你会学到众多的FTFT，像DFTDFT，CTFTCTFT，DTFTDTFT，FFTFFT等，但是，这里，我们考虑一种名叫CFTCFT的东西，它的作用与前面的不太一样，在这里，是将一个01串，映射为另一个等长的01串。

CFT:S1−>S2,(S1,S2为01串)CFT:S1−>S2,(S1,S2为01串)

CFTCFT的映射规则有mm条，在下面给出；同时，每计算一个CFTCFT需要一定的时间tt，现在给你初始的01串S0S0，其长度为nn,并且元素全为0，即00...000...0，问你，至少需要多长的时间，可以通过CFTCFT将S0S0变为S1S1,S1S1长度也为nn,且元素全为1(即11...111...1)

## Input

首行两个整数n,m(1≤n≤20,1≤m≤100000)n,m(1≤n≤20,1≤m≤100000).

​接下来mm行，每行包括两个01串(长度为nn)和一个整数si,wi,ti,(1≤ti≤1000000000)si,wi,ti,(1≤ti≤1000000000)，表示sisi经过CFTCFT变为wiwi需要titi 的时间

## Output

如果不能得到S1S1，输出-1；

​否则输出将S0S0经过CFTCFT变为S1S1所需最少时间。

## Sample input and output

| **Sample Input** | **Sample Output** |
| --- | --- |
| 3 5  000 110 2  000 010 4  000 101 2  110 111 1  000 111 3 | 3 |

题意：

一共最多2^n个节点，100,000条有向边，求0号节点到2^n号节点的最短路。

题解：

此题乍一看似乎是个裸的最短路，然而我们发现，本题中节点数较多，可达2^20=1048576个，且边权很大，需要使用long long类型，可能会使内存爆炸。在这种情况下，我们需要优化空间。

观察到题目中最多有100,000条边，意味着其实最多涉及到的节点只有200,000个。我们可以将点的编号离散化，将它们分别表示为0,1,2…n号点，再求解最短路。

AC代码：

#include <cstdio>

#include <iostream>

#include <queue>

#include <stack>

#include <vector>

#include <string.h>

using namespace std;

typedef long long ll;

const int maxn=100005,maxk=1<<21;

ll dist[maxk],num;

int inque[maxk],head[maxk],a[maxn],b[maxn];

ll c[maxn];

struct Edge{

int from,to,dist,pre;

};

Edge edge[maxk];

char aa[25],bb[25];

void addedge(int f,int t,int d) {

edge[++num].from=f;

edge[num].to=t;

edge[num].dist=d;

edge[num].pre=head[f];

head[f]=num;

}

//二进制转十进制

int convert(string s,int len){

int v=0,count=1,i;

for (i=len-1;i>=0;i--) {

v+=(s[i]-'0')\*count;

count\*=2;

}

return v;

}

//spfa算法求解最短路

ll spfa(int s,int des) {

int i;

memset(inque,0,sizeof(inque));

memset(dist,0x3f,sizeof(dist));

dist[0]=0;

inque[0]=1;

int top=1;

queue<int> q;

q.push(0);

while (!q.empty()) {

int now=q.front();

q.pop();

inque[now]=0;

for (i=head[now];i!=-1;i=edge[i].pre) {

int to=edge[i].to;

if (dist[edge[i].from]+edge[i].dist<dist[to]) {

if (!inque[to]) {

inque[to]=1;

q.push(to);

}

dist[to]=dist[edge[i].from]+edge[i].dist;

}

}

}

return dist[des];

}

int main() {

int n,m,i,x,y;

scanf("%d%d",&n,&m);

num=0;

for (i=0;i<(1<<n);i++) head[i]=-1;

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

scanf("%s%s%lld",aa,bb,&c[i]);

a[i]=convert(aa,n);

b[i]=convert(bb,n);

addedge(a[i],b[i],c[i]);

}

ll ans=spfa(0,(1<<n)-1);

if (ans==0x3f3f3f3f3f3f3f3f) cout << -1;

else printf("%lld",ans);

return 0;

}