|  |
| --- |
| [键入公司名称] |
| 新增模板2 |
| [键入文档副标题] |

|  |
| --- |
| [选取日期] |

目录

[插头dp 2](#_Toc525068032)

[差分约束 13](#_Toc525068033)

[2-sat 15](#_Toc525068034)

# 插头dp

/\*\*

POJ 2411

题意：给你n\*m（1<=n,m<=11）的方格矩阵，要求用1\*2的多米诺骨牌去填充，问有多少种填充方法

使用2进制编码，1表示有向下延伸的插头

\*\*/

#include <cstdio>

#include <cstring>

#include <algorithm>

using namespace std;

#define N 11

typedef long long ll;

ll dp[N][1<<N];

int n,m;

bool check(int s)//状态s中的0是否能全部放横着的骨牌

{

int cnt=0;

for(int i=0;i<m;++i)

if((s>>i)&1)

{

if(cnt&1)//如果连续的0的数量为奇数，则放不下

return false;

cnt=0;

}

else ++cnt;

if(cnt&1)//如果连续的0的数量为奇数，则放不下

return false;

return true;

}

int sub(int s1,int s2)

{

if(s1==0) return -1;

return (s1-1)&s2;

}

int main()

{

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

{

memset(dp,0,sizeof(dp));

for(int s=0;s<(1<<m);++s)

if(check(s)) dp[0][s]=1;

int z=(1<<m)-1;

for(int i=0;i<n-1;++i)

for(int s1=0;s1<(1<<m);++s1)

{

if(dp[i][s1]==0) continue;

int tmp=s1^z;

for(int s2=tmp;s2>=0;s2=sub(s2,tmp))//枚举tmp的子集

if(check(s1|s2))//把s2中上方为插头的位置变为1，再判断

dp[i+1][s2]+=dp[i][s1];

}

printf("%lld\n",dp[n-1][0]);

}

return 0;

}

/\*

HDU 1693

题意：给你一个m \* n的棋盘，有的格子是障碍，找一条或多条回路使得经过每个非障碍格子恰好一次，问有多少中方法．m, n ≤ 12

使用二进制编码，1表示有插头，参考cdq的论文

\*/

#include <cstdio>

#include <cstring>

#include <algorithm>

using namespace std;

typedef long long ll;

#define N 12

ll dp[N][N][1<<N];

int g[N][N];

int n,m;

int main()

{

int ca;

scanf("%d",&ca);

for(int cas=1;cas<=ca;++cas)

{

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

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

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

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

memset(dp,0,sizeof(dp));

dp[0][m][0]=1;

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

{

for(int s=0;s<(1<<m);++s)

dp[i][0][s<<1]=dp[i-1][m][s];

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

for(int s=0;s<(1<<m+1);++s)

{

int x=(s>>j-1)&1;//(i,j)左方的插头

int y=(s>>j)&1;//(i,j)上方的插头

if(g[i][j])//如果为空白方格

{

if(x^y)//左方和上方只有一个方向有插头，此时接上插头，并向下或或向右延伸

dp[i][j][s]=dp[i][j-1][s]+dp[i][j-1][s^(1<<j-1)^(1<<j)];

else//左方和上方都有插头，此时把两个插头联通

dp[i][j][s]=dp[i][j-1][s^(1<<j-1)^(1<<j)];

}

else//如果为障碍

{

if(x==0&&y==0)//障碍方格只能从左方和上方没有插头的状态转移

dp[i][j][s]=dp[i][j-1][s];

}

}

}

printf("Case %d: There are %I64d ways to eat the trees.\n",cas,dp[n][m][0]);

}

return 0;

}

/\*

URAL 1519

题意：给你一个m \* n的棋盘，有的格子是障碍，找一条回路使得经过每个非障碍格子恰好一次，问有多少中方法．m, n ≤ 12

使用8进制编码，非0值表示插头属于的联通块，参考cdq的论文

\*/

#include <cstdio>

#include <cstring>

#include <algorithm>

using namespace std;

#define N 12

#define M 1000010

typedef long long ll;

struct Hash{

const static int mod=(1<<15)-1;

int head[mod+1],pre[M],sz;

ll status[M],f[M];

void init()

{

sz=0;

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

}

void insert(ll s,ll w)

{

ll h=s&mod;

for(int i=head[h];i!=-1;i=pre[i])

{

if(status[i]==s)

{

f[i]+=w;return;

}

}

status[sz]=s;f[sz]=w;pre[sz]=head[h];head[h]=sz++;

}

}ha[2];

bool g[N+2][N+2];

char st[N+10];

int n,m,ex,ey;

void decode(ll s,int code[],int m)//按8进制解码

{

for(int i=0;i<=m;++i)

{

code[i]=s&7;

s>>=3;

}

}

ll encode(int code[],int m)//按8进制编码，联通块用最小表示法表示

{

int fa[10],cnt=0;

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

fa[0]=0;

ll s=0;

for(int i=0;i<=m;++i)

{

if(fa[code[i]]==-1) fa[code[i]]=++cnt;

code[i]=fa[code[i]];

s|=(ll)code[i]<<i\*3;

}

return s;

}

void shift(int code[],int m)

{

for(int i=m;i>0;--i) code[i]=code[i-1];

code[0]=0;

}

void dpblank(int x,int y,int cur)//对空白方格dp

{

int code[N+1];

ll s;

for(int i=0;i<ha[cur].sz;++i)

{

decode(ha[cur].status[i],code,m);

int left=code[y-1],up=code[y];//(x,y)左方的插头和上方的插头

if(left&&up)//如果左方和右方的插头存在

{

if(left==up)//如果两个插头属于同一个联通块，则只有位于最后一个空白方格才能进行转移

{

if(x==ex&&y==ey)//(x,y)位于最后一个方格

{

code[y-1]=code[y]=0;

if(y==m) shift(code,m);//最后一列

ha[cur^1].insert(encode(code,m),ha[cur].f[i]);

//进行shift操作后，code数组改变，如果后面还要用，要重新decode

}

}

else//两个插头属于不同的联通块，则合并这两个联通块

{

code[y-1]=code[y]=0;

for(int i=0;i<=m;++i)

if(code[i]==left) code[i]=up;

if(y==m) shift(code,m);

ha[cur^1].insert(encode(code,m),ha[cur].f[i]);

//进行shift操作后，code数组改变，如果后面还要用，要重新decode

}

}

else if((left&&!up)||(!left&&up))//左方或上方有一个插头

{

int t;

if(left) t=left;

else t=up;

if(g[x][y+1])//向右延伸

{

code[y-1]=0;code[y]=t;

ha[cur^1].insert(encode(code,m),ha[cur].f[i]);

}

if(g[x+1][y])//向下延伸

{

code[y-1]=t;code[y]=0;

if(y==m) shift(code,m);//最后一列

ha[cur^1].insert(encode(code,m),ha[cur].f[i]);

//进行shift操作后，code数组改变，如果后面还要用，要重新decode

}

}

else//左方和上方都没有插头

{

if(g[x+1][y]&&g[x][y+1])//产生一个新联通块，向左和向右都要延伸

{

code[y-1]=code[y]=9;//新联通块的编号不能与当前的重复

ha[cur^1].insert(encode(code,m),ha[cur].f[i]);

}

}

}

}

void dpblock(int x,int y,int cur)//对障碍方格进行dp

{

ll s;

int code[N+1];

for(int i=0;i<ha[cur].sz;++i)

{

decode(ha[cur].status[i],code,m);

if(y==m) shift(code,m);//会于最后一列

ha[cur^1].insert(encode(code,m),ha[cur].f[i]);

//进行shift操作后，code数组改变，如果后面还要用，要重新decode

}

}

ll solve()

{

int cur=0;

ha[cur].init();

ha[cur].insert(0,1);

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

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

{

ha[cur^1].init();

if(g[i][j]) dpblank(i,j,cur);

else dpblock(i,j,cur);

cur^=1;

}

ll ans=0;

for(int i=0;i<ha[cur].sz;++i)

ans+=ha[cur].f[i];

return ans;

}

int main()

{

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

ex=0;

memset(g,0,sizeof(g));

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

{

scanf("%s",st+1);

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

if(st[j]=='.')

{

g[i][j]=true;

ex=i;ey=j;

}

}

if(ex==0) printf("0\n");

else printf("%I64d\n",solve());

return 0;

}

/\*

FZU 1977

题意：输入一个n\*m的地图，其中字母'O'的点是必走点，字母'X'是不能走的点,字母'\*'是可走可不走的点，求经过所有点'O'的不同回路路径个数(路线相同起始点不同视为相同路径)。

可以增加一个标志位来记录是否已经形成回路。

\*/

#include <cstdio>

#include <cstring>

#include <algorithm>

using namespace std;

#define N 12

#define M 100010

typedef \_\_int64 ll;

struct Hash{

const static int mod=(1<<15)-1;

int head[mod+1],pre[M],sz;

ll status[M],f[M];

void init()

{

sz=0;

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

}

void insert(ll s,ll w)

{

ll h=s&mod;

for(int i=head[h];i!=-1;i=pre[i])

{

if(status[i]==s)

{

f[i]+=w;return;

}

}

status[sz]=s;f[sz]=w;pre[sz]=head[h];head[h]=sz++;

}

}ha[2];

int g[N+2][N+2];

char st[N+10];

int n,m,ex,ey;

bool flag;

ll ans;

int decode(ll s,int code[])//按8进制解码，返回值为标志位的值

{

for(int i=0;i<=m;++i)

{

code[i]=s&7;

s>>=3;

}

return s&1;

}

ll encode(int code[],int k)//按8进制编码，联通块用最小表示法表示

{

int fa[10],cnt=0;

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

fa[0]=0;

ll s=0;

for(int i=0;i<=m;++i)

{

if(fa[code[i]]==-1) fa[code[i]]=++cnt;

code[i]=fa[code[i]];

s|=(ll)code[i]<<i\*3;

}

if(k) s|=1LL<<(m+1)\*3;//标志位

return s;

}

void shift(int code[])

{

for(int i=m;i>0;--i) code[i]=code[i-1];

code[0]=0;

}

void dpblank(int x,int y,int cur)//对空白方格dp

{

int code[N+1];

ll s;

for(int i=0;i<ha[cur].sz;++i)

{

int has\_end=decode(ha[cur].status[i],code);//标志位，标志是否已经形成回路

int left=code[y-1],up=code[y];//(x,y)左方的插头和上方的插头

if(left&&up)//如果左方和右方的插头存在

{

if(left==up)//如果两个插头属于同一个联通块，则只有位于最后一个必选点之后的点才可以转移

{

if(flag&&!has\_end)//位于最后一个必选点之后且还没有形成回路

{

code[y-1]=code[y]=0;

if(y==m) shift(code);//最后一列

s=encode(code,1);

ha[cur^1].insert(s,ha[cur].f[i]);

//进行shift操作后，code数组改变，如果后面还要用，要重新decode

}

}

else//两个插头属于不同的联通块，则合并这两个联通块

{

code[y-1]=code[y]=0;

for(int i=0;i<=m;++i)

if(code[i]==left) code[i]=up;

if(y==m) shift(code);

s=encode(code,has\_end);

ha[cur^1].insert(s,ha[cur].f[i]);

//进行shift操作后，code数组改变，如果后面还要用，要重新decode

}

}

else if((left&&!up)||(!left&&up))//左方或上方有一个插头

{

int t;

if(left) t=left;

else t=up;

if(g[x][y+1])//向右延伸

{

code[y-1]=0;code[y]=t;

s=encode(code,has\_end);

ha[cur^1].insert(s,ha[cur].f[i]);

}

if(g[x+1][y])//向下延伸

{

code[y-1]=t;code[y]=0;

if(y==m) shift(code);//最后一列

s=encode(code,has\_end);

ha[cur^1].insert(s,ha[cur].f[i]);

//进行shift操作后，code数组改变，如果后面还要用，要重新decode

}

}

else//左方和上方都没有插头

{

if(g[x+1][y]&&g[x][y+1])//产生一个新联通块，向左和向右都要延伸

{

code[y-1]=code[y]=9;//新联通块的编号不能与当前的重复

s=encode(code,has\_end);

ha[cur^1].insert(s,ha[cur].f[i]);

}

if(g[x][y]==1)//如果(x,y)位可选点，此时不选

{

code[y-1]=code[y]=0;

if(y==m) shift(code);

s=encode(code,has\_end);

ha[cur^1].insert(s,ha[cur].f[i]);

}

}

}

}

void dpblock(int x,int y,int cur)//对障碍方格进行dp

{

ll s;

int code[N+1];

for(int i=0;i<ha[cur].sz;++i)

{

int has\_end=decode(ha[cur].status[i],code);

if(y==m) shift(code);//位于最后一列

s=encode(code,has\_end);

ha[cur^1].insert(s,ha[cur].f[i]);

//进行shift操作后，code数组改变，如果后面还要用，要重新decode

}

}

ll solve()

{

if(ex==0) return 0;

int cur=0;

flag=false;

ha[cur].init();

ha[cur].insert(0,1);

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

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

{

if(i==ex&&j==ey) flag=true;

ha[cur^1].init();

if(g[i][j]) dpblank(i,j,cur);

else dpblock(i,j,cur);

cur^=1;

}

ll ans=0;

for(int i=0;i<ha[cur].sz;++i)

{

ans+=ha[cur].f[i];

}

return ans;

}

int main()

{

int ca;

scanf("%d",&ca);

for(int cas=1;cas<=ca;++cas)

{

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

ex=0;

memset(g,0,sizeof(g));

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

{

scanf("%s",st+1);

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

if(st[j]=='O')

{

g[i][j]=2;

ex=i;ey=j;

}

else if(st[j]=='\*')

g[i][j]=1;

}

printf("Case %d: %I64d\n",cas,solve());

}

return 0;

}

# 差分约束

/\*

差分约束系统：如果一个系统由n个变量和m个约束条件组成，其中每个约束条件形如 xj - xi<= bk ( i , j ∈ [1，n]，k ∈ [1，m])，则称其为差分约束系统。

例如如下的约束条件：

X1 - X2 <= 0 X1 - X5 <= -1

X2 - X5 <= 1 X3 - X1 <= 5

X4 - X1 <= 4 X4 - X3 <= -1

X5 - X3 <= -3 X5 - X4 <= -3

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。

差分约束系统求解过程：

1.新建一个图，N个变量看作N个顶点，M个约束条件作为M条边。每个顶点Vi分别对于一个未知量，每个有向边对应两个未知量的不等式。

2.为了保证图的连通性，在图中新加一个节点Vs，图中每个节点Vi都能从Vs可达，建立边w(Vs，Vi) = 0。

3.对于每个差分约束Xj - Xi <= Bk(这里是小于等于号)，则建立边w(Xi，Xj) = Bk。

4.初始化Dist[] = INF，Dist[Vs] = 0.

5.求解以Vs为源点的单源最短路径，推荐用SPFA，因为一般可能存在负值。

如果图中存在负权回路，则该差分约束系统不存在可行解。

Vs到某点如果不存在最短路径，即最短路为INF，则对于该点表示的变量可以取任意值，都能满足差分约束的要求，如果存在最短路径，则得到该变量的最大值。

上述过程最终得到的解为满足差分约束系统各项的最大值。

注意点：

1. 如果要求最大值想办法把每个不等式变为标准 x - y <= k 的形式,然后建立一条从 y 到 x 权值为 k 的边，求出最短路径即可，变得时候注意 x - y < k => x - y <= k-1。

2. 如果要求最小值的话，变为 x - y >= k 的标准形式，然后建立一条从 y到 x 权值为 k 的边，求出最长路径即可。

3. 如果权值为正，用Dijkstra，SPFA，BellmanFord都可以，如果为负不能用Dijkstra，并且需要判断是否有负环，有的话就不存在。

\*/

/\*

POJ 3169

\*/

#include <cstdio>

#include <cstring>

#include <algorithm>

#include <queue>

using namespace std;

#define N 1010

#define M 20010

const int inf=0x3f3f3f3f;

int head[N],to[M],w[M],pre[M],num[N],d[N];

bool inque[N];

int n,m1,m2,e;

void addedge(int x,int y,int z)

{

to[e]=y;w[e]=z;pre[e]=head[x];head[x]=e++;

}

int spfa(int s,int t,int K)//K为图中点的数量，用来判负环

{

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

memset(d,0x3f,sizeof(d));//求最短路赋值为inf，最长路则赋值为-inf

deque<int> que;

que.push\_back(s);d[s]=0;inque[s]=true;num[s]=1;

while(!que.empty())

{

int x=que.front();

que.pop\_front();

inque[x]=false;

for(int i=head[x];i!=-1;i=pre[i])

{

int y=to[i],cost=d[x]+w[i];

if(cost<d[y])//求最短路时用小于号，最长路则用大于号

{

d[y]=cost;

num[y]=num[x]+1;

if(num[y]>K) return -1;//如果起点到y之间点的数量超过K，形成负环

if(!inque[y])

{

inque[y]=true;

if(!que.empty()&&d[y]<=d[que.front()])//求最短路用小于等于号，求最长路用大于等于号

que.push\_front(y);

else que.push\_back(y);

}

}

}

}

if(d[t]==inf) return -2;//s无法到达t，如果求最长路，则改为-inf

else return d[t];

}

int main()

{

scanf("%d%d%d",&n,&m1,&m2);

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

e=0;

int x,y,z;

for(int i=1;i<=m1;++i)

{

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

if(x>y) swap(x,y);

addedge(x,y,z);

}

for(int i=1;i<=m2;++i)

{

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

if(x>y) swap(x,y);

addedge(y,x,-z);

}

printf("%d\n",spfa(1,n,n));

return 0;

}

# 2-sat

/\*

2-sat 输出任意解

题意：有一对新人结婚，邀请n对夫妇去参加婚礼。有一张很长的桌子，人只能坐在桌子的两边，还要满足下面的要求：

1.每对夫妇不能坐在同一侧

2.m对夫妇之中可能有通奸关系（包括男男，男女，女女），有通奸关系的不能同时坐在新娘的对面，可以分开坐，可以同时坐在新娘这一侧。如果存在一种可行的方案，输出与新娘同侧的人。

求解的时候去选择和新郎同一侧的人，输出的时候换一下就是新娘同一侧的人。如果i和j有奸情，则增加一条i到j',j到i'的边，同时增加一条新娘到新郎的边，表示必须选新郎。

\*/

#include <cstdio>

#include <cstring>

#include <algorithm>

#include <vector>

using namespace std;

#define N 20010

#define M 100010

int head[N],to[M],pre[M];

int dfn[N],low[N],sta[N],block[N],ru[N],opp[N],col[N];

bool insta[N];

int n,m,e,cnt,scnt,top;

vector<int> g[N];

void addedge(int x,int y)

{

to[e]=y;pre[e]=head[x];head[x]=e++;

}

void tarjan(int x)

{

dfn[x]=low[x]=++cnt;

insta[x]=true;

sta[++top]=x;

int y;

for(int i=head[x];i!=-1;i=pre[i])

{

y=to[i];

if(!dfn[y])

{

tarjan(y);

low[x]=min(low[x],low[y]);

}

else if(insta[y]) low[x]=min(low[x],dfn[y]);

}

if(low[x]==dfn[x])

{

++scnt;

do

{

y=sta[top--];

insta[y]=false;

block[y]=scnt;

}while(y!=x);

}

}

int main()

{

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

{

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

e=0;

addedge(0,1);

int x,y;

char st[20];

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

{

scanf("%s",st);

int l=strlen(st);

char c=st[l-1];

st[l-1]='\0';

x=atoi(st)<<1;

if(c=='h') x|=1;

scanf("%s",st);

l=strlen(st);

c=st[l-1];

st[l-1]='\0';

y=atoi(st)<<1;

if(c=='h') y|=1;

addedge(x,y^1);

addedge(y,x^1);

}

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

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

memset(insta,0,sizeof(insta));

cnt=scnt=top=0;

for(int i=0;i<n\*2;++i)

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

bool flag=true;

for(int i=0;i<n\*2;i+=2)//判断是否有可行解

if(block[i]==block[i^1])

{

flag=false;break;

}

else//记录相反的联通块

{

opp[block[i]]=block[i^1];

opp[block[i^1]]=block[i];

}

if(!flag)//无可行解

{

puts("bad luck");

continue;

}

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

{

g[i].clear();ru[i]=0;

}

for(int x=0;x<n\*2;++x)

for(int i=head[x];i!=-1;i=pre[i])

{

int y=to[i];

if(block[x]==block[y]) continue;

g[block[y]].push\_back(block[x]);//建立反向边

++ru[block[x]];

}

memset(col,0,sizeof(col));

//拓扑排序

top=0;

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

if(ru[i]==0) sta[++top]=i;

while(top)

{

int x=sta[top--];

if(!col[x])//0表示为染色，-1表示假，1表示真

{

col[x]=1;col[opp[x]]=-1;

}

for(int i=0;i<g[x].size();++i)

{

int y=g[x][i];

if(--ru[y]==0) sta[++top]=y;

}

}

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

if(col[block[i<<1]]==-1) printf("%dw ",i);

else printf("%dh ",i);

printf("\n");

}

return 0;

}

/\*

2-sat 求字典序最小的解，代码量比求任意解小，但效率慢一点

题意：每个党派需要在两个代表中选一个，这2\*n个代表中有彼此讨厌的m对人，输出n个去开会的代表

\*/

#include <cstdio>

#include <cstring>

#include <algorithm>

using namespace std;

#define N 16010

#define M 100010

int head[N],to[M],pre[M],sta[N];

bool mark[N];

int n,m,e,top;

void addedge(int x,int y)

{

to[e]=y;pre[e]=head[x];head[x]=e++;

}

bool dfs(int x)

{

if(mark[x^1]) return false;

if(mark[x]) return true;

mark[x]=true;

sta[++top]=x;

for(int i=head[x];i!=-1;i=pre[i])

if(!dfs(to[i])) return false;

return true;

}

bool solve()

{

for(int i=0;i<n\*2;i+=2)

{

if(!mark[i]&&!mark[i^1])

{

top=0;

if(!dfs(i))

{

while(top>0) mark[sta[top--]]=false;

if(!dfs(i+1)) return false;

}

}

}

return true;

}

int main()

{

while(scanf("%d%d",&n,&m)!=EOF)

{

for(int i=0;i<n\*2;++i)

{

head[i]=-1;mark[i]=0;

}

e=0;

int x,y;

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

{

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

--x;--y;

addedge(x,y^1);

addedge(y,x^1);

}

if(!solve()) puts("NIE");

else

{

for(int i=0;i<n\*2;++i)

if(mark[i]) printf("%d\n",i+1);

}

}

return 0;

}