## [POJ 3133 Manhattan Wiring （插头DP）](https://www.cnblogs.com/kuangbin/archive/2012/09/28/2707416.html)

Manhattan Wiring

|  |  |  |
| --- | --- | --- |
| **Time Limit:** 5000MS |  | **Memory Limit:** 65536K |
| **Total Submissions:** 1110 |  | **Accepted:** 634 |

Description

There is a rectangular area containing n × m cells. Two cells are marked with “2”, and another two with “3”. Some cells are occupied by obstacles. You should connect the two “2”s and also the two “3”s with non-intersecting lines. Lines can run only vertically or horizontally connecting centers of cells without obstacles.

Lines cannot run on a cell with an obstacle. Only one line can run on a cell at most once. Hence, a line cannot intersect with the other line, nor with itself. Under these constraints, the total length of the two lines should be minimized. The length of a line is defined as the number of cell borders it passes. In particular, a line connecting cells sharing their border has length 1.

Fig. 1(a) shows an example setting. Fig. 1(b) shows two lines satisfying the constraints above with minimum total length 18.

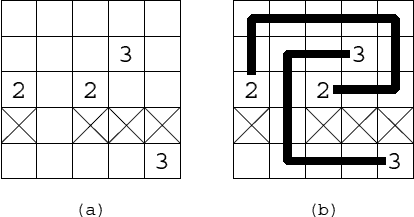


Figure 1: An example of setting and its solution

Input

The input consists of multiple datasets, each in the following format.

|  |  |
| --- | --- |
| n | m |
| row1 | |
| … | |
| rown | |

n is the number of rows which satisfies 2 ≤ n ≤ 9. m is the number of columns which satisfies 2 ≤ m ≤ 9. Each rowi is a sequence of m digits separated by a space. The digits mean the following.

0: Empty

1: Occupied by an obstacle

2: Marked with “2”

3: Marked with “3”

The end of the input is indicated with a line containing two zeros separated by a space.

Output

For each dataset, one line containing the minimum total length of the two lines should be output. If there is no pair of lines satisfying the requirement, answer “0” instead. No other characters should be contained in the output.

Sample Input

5 5

0 0 0 0 0

0 0 0 3 0

2 0 2 0 0

1 0 1 1 1

0 0 0 0 3

2 3

2 2 0

0 3 3

6 5

2 0 0 0 0

0 3 0 0 0

0 0 0 0 0

1 1 1 0 0

0 0 0 0 0

0 0 2 3 0

5 9

0 0 0 0 0 0 0 0 0

0 0 0 0 3 0 0 0 0

0 2 0 0 0 0 0 2 0

0 0 0 0 3 0 0 0 0

0 0 0 0 0 0 0 0 0

9 9

3 0 0 0 0 0 0 0 2

0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

2 0 0 0 0 0 0 0 3

9 9

0 0 0 1 0 0 0 0 0

0 2 0 1 0 0 0 0 3

0 0 0 1 0 0 0 0 2

0 0 0 1 0 0 0 0 3

0 0 0 1 1 1 0 0 0

0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

9 9

0 0 0 0 0 0 0 0 0

0 3 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 2 3 2

0 0

Sample Output

18

2

17

12

0

52

43

Source

[Japan 2006](http://poj.org/searchproblem?field=source&key=Japan+2006)

/\*

POJ 3133

G++ 782ms 1436K\*/

#include<stdio.h>

#include<iostream>

#include<string.h>

#include<algorithm>using namespace std;

const int hash\_size=60007;const int INF=100000;

int n,m;int map[20][20];int Pow[40];

struct Node{

int hash\_chart[hash\_size],sz;

int msk[hash\_size];

int dp[hash\_size];

int next[hash\_size];

void clear(){

sz=0;

memset(hash\_chart,-1,sizeof(hash\_chart));

}

inline void push(int \_msk,int val){

int x=\_msk%hash\_size;

for(int i=hash\_chart[x];i!=-1;i=next[i]){

if(msk[i]==\_msk){

if(dp[i]>val)dp[i]=val;

return;

}

}

msk[sz]=\_msk;

dp[sz]=val;

next[sz]=hash\_chart[x];

hash\_chart[x]=sz++;

}

inline int res(){//得到答案

int x=0;

for(int i=hash\_chart[x];i!=-1;i=next[i])

if(!msk[i])

return dp[i]-2;

return 0;

}

}hh[2];//两个循环转移状态

void solve(){

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

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

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

int now,pre;

pre=0;

now=1;

hh[pre].clear();

hh[pre].push(0,0);

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

for(int j=0;j<=m;j++){

hh[now].clear();

for(int p=0;p<hh[pre].sz;p++){//从pre的所有状态推到now的状态

int k=hh[pre].msk[p];//3进制表示的当前的插头状态

int t=hh[pre].dp[p];

if(j==m){

if(k/Pow[m])continue;//最后有插头不能转移

hh[now].push(k\*3,t);

continue;

}

int t1=(k/Pow[j])%3;//左

int t2=(k/Pow[j+1])%3;//上

int tk;

if(map[i][j]==0){//当前格子为空

if(t1==0&&t2==0){//没有插头

tk=k+Pow[j]+Pow[j+1];//增加2号插头

hh[now].push(tk,t+1);

tk=k+(Pow[j]<<1)+(Pow[j+1]<<1);//增加3号插头

hh[now].push(tk,t+1);

tk=k;//不加插头

hh[now].push(tk,t);

}

else if((t1&&(!t2))||(t2&&(!t1))){//只有一个插头

int temp=k-t1\*Pow[j]-t2\*Pow[j+1];

int temps=(!t1)?t2:t1;

tk=temp+temps\*Pow[j];//插头从下边出来

hh[now].push(tk,t+1);

tk=temp+temps\*Pow[j+1];//插头从右边出来

hh[now].push(tk,t+1);

}

else if((t1==t2)&&t1){//有两个一样的插头

tk=k-t1\*Pow[j]-t2\*Pow[j+1];//把插头消去

hh[now].push(tk,t+1);

}

}

else if(map[i][j]==1){//障碍

if(t1==0&&t2==0){//不能有插头

tk=k;

hh[now].push(tk,t);

}

}

else if(map[i][j]==2){//2号

if(t1==0&&t2==0){

tk=k+Pow[j];

hh[now].push(tk,t+1);

tk=k+Pow[j+1];

hh[now].push(tk,t+1);

}

else if((t1==1&&t2==0)||(t1==0&&t2==1)){

tk=k-Pow[j]\*t1-Pow[j+1]\*t2;

hh[now].push(tk,t+1);

}

}

else if(map[i][j]==3){

if(t1==0&&t2==0){

tk=k+(Pow[j]<<1);

hh[now].push(tk,t+1);

tk=k+(Pow[j+1]<<1);

hh[now].push(tk,t+1);

}

else if((t1==2&&t2==0)||(t1==0&&t2==2)){

tk=k-Pow[j]\*t1-Pow[j+1]\*t2;

hh[now].push(tk,t+1);

}

}

}

swap(now,pre);

}

printf("%d\n",hh[pre].res());

}

int main(){

Pow[0]=1;

for(int i=1;i<20;i++)Pow[i]=3\*Pow[i-1];

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

if(n==0&&m==0)break;

solve();

}

return 0;

}