【动态规划】Ural1519 Formula 1

http://simg.sinajs.cn/blog7style/images/common/sg_trans.gif(2009-12-15 14:41:43)

[[http://simg.sinajs.cn/blog7style/images/common/sg_trans.gif](javascript:;)转载▼](javascript:;)

1519. Formula 1

Time Limit: 1.0 second  
Memory Limit: 16 MB

Background

Regardless of the fact, that Vologda could not get rights to hold the Winter Olympic games of 20\*\*, it is well-known, that the city will conduct one of the Formula 1 events. Surely, for such an important thing a new race circuit should be built as well as hotels, restaurants, international airport - everything for Formula 1 fans, who will flood the city soon. But when all the hotels and a half of the restaurants were built, it appeared, that at the site for the future circuit a lot of gophers lived in their holes. Since we like animals very much, ecologists will never allow to build the race circuit over the holes. So now the mayor is sitting sadly in his office and looking at the map of the circuit with all the holes plotted on it.

Problem

Who will be smart enough to draw a plan of the circuit and keep the city from inevitable disgrace? Of course, only true professionals - battle-hardened programmers from the first team of local technical university!.. But our heroes were not looking for easy life and set much more difficult problem: "Certainly, our mayor will be glad, if we find how many ways of building the circuit are there!" - they said.  
It should be said, that the circuit in Vologda is going to be rather simple. It will be a rectangle N\*M cells in size with a single circuit segment built through each cell. Each segment should be parallel to one of rectangle's sides, so only right-angled bends may be on the circuit. At the picture below two samples are given for N = M = 4 (gray squares mean gopher holes, and the bold black line means the race circuit). There are no other ways to build the circuit here.

Input

The first line contains the integer numbers N and M (2 ≤ N, M ≤ 12). Each of the next N lines contains M characters, which are the corresponding cells of the rectangle. Character "." (full stop) means a cell, where a segment of the race circuit should be built, and character "\*" (asterisk) - a cell, where a gopher hole is located.

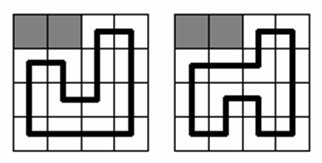
Output

You should output the desired number of ways. It is guaranteed, that it does not exceed 263-1.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

题目简意：

给你一个m \* n的棋盘，有的格子是障碍，问共有多少条回路使得经过每个非障碍格子恰好一次．m, n ≤ 12。

[](http://photo.blog.sina.com.cn/showpic.html#blogid=51cea4040100gmky&url=http://s4.sinaimg.cn/orignal/51cea404x7abee2b588b3&690)                    
如图，m = n = 4，(1, 1), (1, 2)是障碍，共有2条满足要求的回路．  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

额……先听ccy唧唧歪歪一段吧……（时间紧张的同学们请直接跳至下一块！！！）

这是个很古老的问题，陈丹琦关于状态压缩的这篇论文也是很古老的论文，ccy其实和这篇论文接触了好几次了（仅此接触而已），陈丹琦的那个ppt，ccy看了也好多好多次了，从第一次学状态压缩，然后，后来复习状态压缩。不过，陈丹琦讲的这些东西对我来说，确实是太有难度了，以至于，我一直扑在周伟的那篇没写完的论文上。这次，吴老师说，我们要提高，于是乎，我从上个星期五开始，刻苦专研陈丹琦的这篇《基于连通性状态压缩的动态规划问题》的文。（战线拉得有点长，因为周末碰到帅哥了，额……）

话说，看那文是相当痛苦，开始还知道可以怎么去尝试实现，后来，就是感慨大牛思想，仅此，感叹，毫无办法。所以，ccy在痛苦的拉了一篇后，决定从头来，一题题的试着实现。（不过不知道能不能坚持到最后一题。）

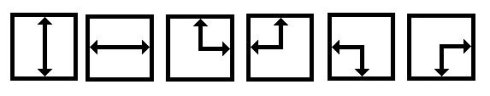
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

    这是一个古老而经典的模型，经典得以至于不用看多久就知道是个状态压缩。

    关于棋盘的状态压缩，我觉得应该分两类吧（我目前只见了这两类，囧……）。第一类：骨牌覆盖。第二类：线段覆盖。这题明显的第二类。（也是ccy一直没挑战过的一类。）

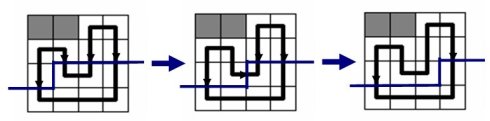
    对于线段覆盖，首先涉及到的，就是古老的名词：插头。

    曼哈顿回路，对于每个格子而言，一共有六种情况，也就是六个插头：

[](http://photo.blog.sina.com.cn/showpic.html#blogid=51cea4040100gmky&url=http://s8.sinaimg.cn/orignal/51cea404x7abf50223157&690)

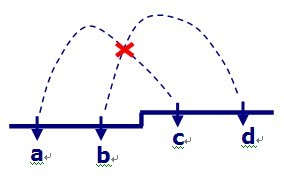
    至于棋盘里状态的递推，有两种，一种是逐行，一种是逐格。对于这题，逐格可以省掉逐行里的很多无用状态，所以，下面说的是逐格。

    如图所示：

[](http://photo.blog.sina.com.cn/showpic.html#blogid=51cea4040100gmky&url=http://s2.sinaimg.cn/orignal/51cea404x7abf652c4f31&690)

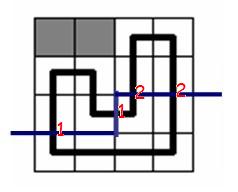
    陈丹琦画的这个相当清楚，我就不累赘了。

    下面，是状态的表示。因为这是个曼哈顿回路，有很很神奇的现象，联通路径不交叉。

[](http://photo.blog.sina.com.cn/showpic.html#blogid=51cea4040100gmky&url=http://s11.sinaimg.cn/orignal/51cea404x7abf72b3723a&690)

    状态要包含的东西有两点，一是插头方向，二是连通性，因为，这里有神奇的不交叉，所以，我们可以巧妙地用括号匹配。

    如表示这个状态：

[](http://photo.blog.sina.com.cn/showpic.html#blogid=51cea4040100gmky&url=http://s1.sinaimg.cn/orignal/51cea404x7abf968c0820&690)

    相同的数字表示这两个插头联通，当前做到的是第3行第4列，有状态（1，1，2，2）。我们用括号表示：(#)()。

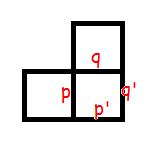
   （：表示左插头。

   ）：表示右插头。

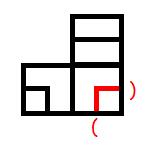
    #：表示无插头。

    说到这里，相信都看出括号的巧妙吧，他成功地化解了插头的繁琐，同时，能够通过匹配，判断出联通性。并且，降低了进制。这里有3种情况，鉴于位运算的速度可爱，我们用4进制。

    下面，来看格子具体的扩展情况。

[](http://photo.blog.sina.com.cn/showpic.html#blogid=51cea4040100gmky&url=http://s2.sinaimg.cn/orignal/51cea404x72accb885b01&690)我们要把p、q的插头状态转为p'和q',pq --> p'q'。

    CASE 1:   ## -->（）

[](http://photo.blog.sina.com.cn/showpic.html#blogid=51cea4040100gmky&url=http://s10.sinaimg.cn/orignal/51cea404x7abfbfc50009&690)

    这时需要判断方格(i,j+1)和方格(i+1,j)是否还可以覆盖线段，以免生出无用状态。

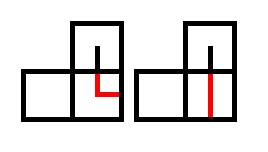
    CASE 2:

    2.1  #（ -->  # (

         #（ --> （ #

         # ) -->  # )

         # ) -->  ) #

[](http://photo.blog.sina.com.cn/showpic.html#blogid=51cea4040100gmky&url=http://s14.sinaimg.cn/orignal/51cea404x7abfd47e3cfd&690)

    第一种：判断方格(i,j+1)是否可以覆盖线段。

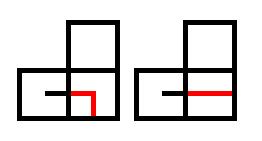
    第二种：判断方格(i+1,j)是否可以覆盖线段。

    2.2  ) # --> ) #

         ) # --> # )

         ( # --> ( #

         ( # --> # (

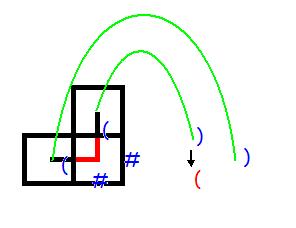
[](http://photo.blog.sina.com.cn/showpic.html#blogid=51cea4040100gmky&url=http://s9.sinaimg.cn/orignal/51cea404x72acca68af78&690)

    第一种：判断方格(i+1,j)是否可以覆盖线段。

    第二种：判断方格(i,j+1)是否可以覆盖线段。

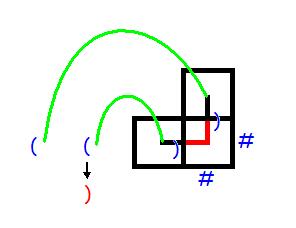
    CASE 3:

    3.1 （（ --> # #

[](http://photo.blog.sina.com.cn/showpic.html#blogid=51cea4040100gmky&url=http://s9.sinaimg.cn/orignal/51cea404x7ac00f47efd8&690)

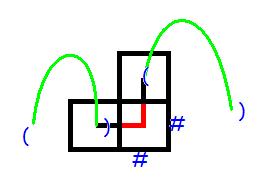
    q对应的右括号变为左括号。

    3.2  ）） --> # #

[](http://photo.blog.sina.com.cn/showpic.html#blogid=51cea4040100gmky&url=http://s13.sinaimg.cn/orignal/51cea404x7ac01c60e68c&690)

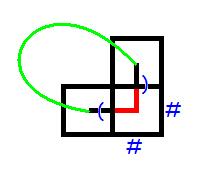
    p对应的左括号变为右括号。

    3.3 )( --> # #

[](http://photo.blog.sina.com.cn/showpic.html#blogid=51cea4040100gmky&url=http://s1.sinaimg.cn/orignal/51cea404x7ac0265e1190&690)

    pq对应的括号不变。

    3.4 （）--> # #

[](http://photo.blog.sina.com.cn/showpic.html#blogid=51cea4040100gmky&url=http://s7.sinaimg.cn/orignal/51cea404x7ac02e5fafe6&690)

    这种情况下，只能是棋盘的最后一格。

    ccy此题小记：

    （以前虽然看过线段覆盖的题ppt，但是从来没有实现过，也觉得有诸多困难，这次，终于解开以前的疑虑。）

    1、关于轮廓线。

       一直以为第一格需要特殊处理，比如单独写段代码来解决之类的，实际不需要。因为第一格的左插头始终是0。

    2、状态一般都用HASH来解决。

    3、关于状态转移。

       因为以前研究的一直是骨牌覆盖问题，长期是先找出所有状态，再行行递推，直接用。所以对于逐格转移的状态很迷茫。其实，我们总是从上次有用的状态推出当前有效的状态。陈丹琦讲的队列扩展，应该就这意思。

    4、关于2^n进制的处理。

       白痴都知道用2进制运算时就可以用位运算，有速度。而在遇到x进制，在大小允许的情况下，我们一般都转成2^n进制做。不过，ccy一直以为都没有实际操作过这个。这次理了一下。比如4进制：

       (1203)4 = 1\*4^3 + 2\*4^2 + 0\*4^1 + 3\*4^0

               = 1\*2^6 + 2\*2^4 + 0\*2^2 + 3\*2^0

    额……，看了一天半论文，想了半天这题，写了半上午代码，再写了半下午以上的东东（画图花时间呀！！！）。终于完工。

    合掌，微笑。O(∩\_∩)O哈哈~

ccy代码：

#include<iostream>  
using namespace std;

const int Limitn=15;  
const int Limithash=1999997;

int a[Limitn][Limitn];//地图，判断线段可放不  
int k,n,m,nn,mm;//k：滚动用。

int jz[Limitn];//进制移动的位数

int hash[Limithash];//hash里存的东东是当前状态在state里第几个位置  
unsigned long long state[2][600000],sum[2][600000];//state状态，sum:该状态的和  
int total[2];//该格子状态的总数

int ans=0;

void init()  
{  
    char ch;  
    scanf("%d%d",&n,&m);  
    for (int i=1;i<=n;i++)  
    {  
        scanf("%c",&ch);  
        for (int j=1;j<=m;j++)  
        {  
            scanf("%c",&ch);  
            a[i][j]=ch=='.';  
            if (ch=='.') {nn=i; mm=j;}//nn,mm记录的是最后一个可覆盖线段的格子的位置  
        }  
    }  
}

void prepared\_jinzhi()  
{  
    jz[0]=0;  
    for (int i=1;i<=n;i++)  
        jz[i]=i<<1;  
}

void Hash\_in(unsigned long long s,int data)//哈希  
{  
    int hashpos=s%Limithash;  
    while (hash[hashpos])  
    {  
        if (state[k][hash[hashpos]]==s)//如果状态存在  
        {  
            sum[k][hash[hashpos]]+=data;  
            return;  
        }  
        hashpos++;  
        if (hashpos==Limithash) hashpos=0;  
    }  
    total[k]++;//新状态  
    hash[hashpos]=total[k];  
    state[k][total[k]]=s; sum[k][total[k]]=data;  
}

void work()  
{  
    int tmps;  
    total[0]=1; sum[0][1]=1;  
    for (int i=1;i<=n;i++)  
    {  
        for (int j=1;j<=m;j++)  
        {  
            k^=1;//滚动  
            memset(hash,0,sizeof(hash));  
            memset(sum[k],0,sizeof(sum[k]));  
            memset(state[k],0,sizeof(state[k]));  
            total[k]=0;//清空滚动数组上次的记录  
            for (int u=1;u<=total[1-k];u++)//对于前一格每个有效的状态分别扩展  
            {  
                int s=state[1-k][u];//取出状态  
                int data=sum[1-k][u];//到达前一格该状态下的方案总数

                int p=(s>>jz[j-1])%4;//取第j-1位的括号  
                int q=(s>>jz[j])%4;//取第j位的括号

                if (!a[i][j])//如果不能覆盖线段  
                {  
                    if (p==0 && q==0) Hash\_in(s,data);//如果连线可以绕开障碍，则新增该状态。  
                }  
                else//如果能覆盖线段  
                {  
                    if (p==0 && q==0)  
                    {  
                        if (a[i][j+1] && a[i+1][j])  
                        {  
                            tmps=s+1\*(1<<jz[j-1])+2\*(1<<jz[j]);  
                            Hash\_in(tmps,data);  
                        }  
                        continue;  
                    }

                    if (p==0 && q>0)  
                    {  
                        if (a[i][j+1])  
                            Hash\_in(s,data);  
                        if (a[i+1][j])  
                        {  
                            tmps=s-q\*(1<<jz[j]);  
                            tmps+=q\*(1<<jz[j-1]);  
                            Hash\_in(tmps,data);  
                        }  
                        continue;  
                    }

                    if (p>0 && q==0)  
                    {  
                        if (a[i+1][j])  
                            Hash\_in(s,data);  
                        if (a[i][j+1])  
                        {  
                            tmps=s-p\*(1<<jz[j-1]);  
                            tmps+=p\*(1<<jz[j]);

                            Hash\_in(tmps,data);  
                        }  
                        continue;  
                    }

                    if (p==1 && q==1)  
                    {  
                        int brackets=1;//寻找匹配的括号  
                        for (int v=j+1;v<=n;v++)  
                        {  
                            int w=(s>>jz[v])%4;  
                            if (w==1) brackets++;  
                            if (w==2) brackets--;  
                            if (brackets==0)  
                            {  
                                tmps=s-2\*(1<<jz[v])+1\*(1<<jz[v]);  
                                break;  
                            }  
                        }  
                        tmps=tmps-1\*(1<<jz[j-1])-1\*(1<<jz[j]);  
                        Hash\_in(tmps,data);  
                        continue;  
                    }

                    if (p==2 && q==2)  
                    {  
                        int brackets=1;  
                        for (int v=j-2;v>=1;v--)  
                        {  
                            int w=(s>>jz[v])%4;  
                            if (w==2) brackets++;  
                            if (w==1) brackets--;  
                            if (brackets==0)  
                            {  
                                tmps=s-1\*(1<<jz[v])+2\*(1<<jz[v]);  
                                break;  
                            }  
                        }  
                        tmps=tmps-2\*(1<<jz[j-1])-2\*(1<<jz[j]);  
                        Hash\_in(tmps,data);  
                        continue;  
                    }

                    if (p==2 && q==1)  
                    {  
                        tmps=s-2\*(1<<jz[j-1])-1\*(1<<jz[j]);  
                        Hash\_in(tmps,data);  
                        continue;  
                    }

                    if (p==1 && q==2)  
                    {  
                        if (i==nn && j==mm) ans+=data;  
                        continue;  
                    }  
                }  
            }  
        }  
        for (int i=1;i<=total[k];i++)//每一行做完后，应该乘以4，这个用笔算下，就很清楚了  
            state[k][i]=state[k][i]<<2;  
    }  
}

int main()  
{  
    freopen("data.in","r",stdin);  
    freopen("data.out","w",stdout);

    init();  
    prepared\_jinzhi();  
    work();  
    printf("%d\n",ans);

    fclose(stdout);  
    return 0;  
}