C题解题报告：BFS+双端队列

首先，我们应注意到，题目所求的是“最早”什么时候得到能量。涉及到“最”之类的问题，一般先考虑用BFS（广度优先搜索），因为BFS每次搜索到的都是最短距离的缘故（当然这个得视题目具体情况）；

其次，这个题目的一个难点便是如何建图。

注意到，n×m个正方形共有(n+1)×(m+1)个顶点；

能量在每个顶点有四个移动方向（其中至少有一个方向能量无法移动：原路）；

传送门与流向相同时，移动不需要消耗能量（即两点间权重为0），相反时，移动需要消耗能量（即两点间权重为1）。

据此，我们可以建立一个二维数组，用来储存每个正方形原件传送门的激活状态。

```
int map[maxn][maxn];
```

同时还要定义方向数组如下：

```
int dir[4][4]={1,1,1,1,1,-1,1,0,-1,1,0,1,-1,-1,0,0};
```

其中dir[i][0]表示第i+1种移动方式下x的移动长度，dir[i][1]表示第i+1种移动方式下y的移动长度；而dir[i][2]和dir[i][3]用来表示第i+1种移动方式下所经过的正方形元件。

为了方便判断第i个方向是何种情况是理想的（即不消耗能量），可以再定义一个辅助数组如下：

```
int ideal_c[4]={0,1,1,0};
//别忘了'0'和'1'分别表示正方形元件的两种状态
```

这三个数组确定好之后，让点从左上角（0,0）开始搜索，每次遍历四个方向。

如果合法便加入队列中，同时更新对应方向所到达的点的消耗能量。

当队列不为空时，一直这么循环下去，直到搜索到了右下角，即点（n,m）。

此时你可能会有疑惑（当然你肯定不会有这个疑惑的），

如果题目给的情况比较好，顺着元件的初始方向走可以直接流到很接近终点的方向，用BFS的话，不得从头慢慢搜，岂不是很耗时。

确实，这是个比较严重的问题，为了解决这个问题，我们可以使用**双端队列**来对BFS进行优化。（**O（V+E）**）

双端队列，顾名思义，就是选择可以在头或尾插入元素。

```
for(int i=0;i<4;++i){//循环进入四种方向
 int nx=now.x+dir[i][0],ny=now.y+dir[i][1];//节点上的位置
 int nnx=now.x+dir[i][2],nny=now.y+dir[i][3];//正方形原件上的位置
 if(nx>=0&&nx<=n&&ny>=0&&ny<=m){
  int cost=(map[nnx][nny]==ideal_c[i]?0:1);//看正方形元件的初始状态是否‘理想’
  if(energy[nx][ny]>energy[now.x][now.y]+cost){
   energy[nx][ny]=energy[now.x][now.y]+cost;
   if(!cost)
    q.push_front(node(nx,ny));//理想，放队列前面
   else
    q.push_back(node(nx,ny));//不理想，往后面站去
  }
 }
}
```

当我们朝某个方向移动不需要消耗能量时，我们把到达的点放在队列的前面；需要消耗能量则放在后面，这样就可以优先选择不需要消耗能量的路继续进行搜索了。问题解决。

最后附上AC代码：

```
#include<iostream>
#include<cstring>
#include<queue>

using namespace std;

const int maxx=505;
const int maxy=505;
const int inf = 1e4+5;
int map[maxx][maxy];//地图 ，假设\为0，/为1
typedef struct node{
 int x;
 int y;
 node(){};
 node(int a,int b):x(a),y(b){};
 node(const node& n){
  x=n.x,y=n.y;
 }
}node;
node now;
int energy[maxx][maxy];//到达每点所需要消耗的能量
int dir[4][4]={1,1,1,1,1,-1,1,0,-1,1,0,1,-1,-1,0,0};//在每个点时可以走的方向
int ideal_c[4]={0,1,1,0};//如果地图上对应的是值与该数组对应的值相同，则不需要消耗能量便能进入
int bfs(int x,int y,int end_x,int end_y);
int n,m;

int main(){
 scanf("%d%d",&n,&m);
 if((n+m)%2)
  printf("NO SOLUTION\n");
 else{
  int c;
  for(int i=1;i<=n;++i){
   for (int j=1;j<=m;++j){
    scanf(" %c",&c);
    if(c=='/')
     map[i][j]=1;//数据预处理
```

```
      map[i][j]-1;//数据预处理
    }
   }
   memset(energy,0x3f,sizeof energy);
   int min_energy=bfs(0,0,n,m);
   printf("%d",min_energy);
  }
  return 0;
}

int bfs(int x,int y,int end_x,int end_y){
 now.x=x;
 now.y=y;
 energy[x][y]=0;
 deque <node> q;
 q.push_back(now);
 while(!q.empty()){
  now=q.front();
  if(now.x==end_x&&now.y==end_y)
   return energy[end_x][end_y];
  q.pop_front();
  for(int i=0;i<4;++i){
   int nx=now.x+dir[i][0],ny=now.y+dir[i][1];
   int nnx=now.x+dir[i][2],nny=now.y+dir[i][3];
   if(nx>=0&&nx<=n&&ny>=0&&ny<=m){
    int cost=(map[nnx][nny]==ideal_c[i]?0:1);
    if(energy[nx][ny]>energy[now.x][now.y]+cost){
     energy[nx][ny]=energy[now.x][now.y]+cost;
     if(!cost)
      q.push_front(node(nx,ny));
     else
      q.push_back(node(nx,ny));
    }
   }
  }
 }
}
```