象棋

一. 考察内容：

网络流 费用流

二. 题目分析：

[题目大意]

在n*m的棋盘上，给定一些棋子的位置和一些禁用的点，并规定棋子的走法，求出让所有起点走到终点的最小距离，在走的过程中不能有多个棋子同时在一个格点上。

[写题思路]

首先分析“不能有多个棋子在一个格点上”的条件，画几个图后可以得出，无论如何走，只要安排好走的顺序，都可以使得不出现有多个棋子在一个格点上的情况；故，这句话没实际用处。

现在考虑建图，在这些棋盘上，我们可以把每个格点看做一个节点，两个可以互相转移的格点看做两个点之间连接的一条边，对于初始状态有棋子的点与虚拟的源点（S）连接一条边，权值为0，容量为1，对于每个终点，对虚拟汇点（T）连接一条边，权值为0，容量为1。对两个可以互相转移的节点，连接一条无向边，权值为1（因为需要多走一步），容量无限（因为可以无限次数的转移）。

建图完成之后，考虑题目要找到所有棋子走到终点的最少步骤，所以求出最小费用最大流即可。

三. 代码实现：

```
#define _CRT_SECURE_NO_DEPRECATE
/*************************
*创建时间：2018 08 04
*文件类型：源代码文件
*题目来源：COGS
*当前状态：已通过
*备忘录：网络流 费用流 最小费用最大流
*作者：HtBest
*************************/
#include <stdio.h>
#include <iostream>
#include <string>
#include <stdlib.h>
#include <string.h>
#include <algorithm>
#include <queue>
using namespace std;
namespace IO
{
    #define SIZE 10000
    int T=0,end=0;
    char S[SIZE+1];
    int in()
    {
        T=0;
        return end=fread(S,1,SIZE,stdin);
    }
    int read(int &a)
    {
        int i=a=0,flag=1;
```

```
        while(1)
        {
            if(T==end&&!in())   return i;
            if(S[T]==' '||S[T]=='\n'||S[T]=='\r')
            {
                if(i)   return i;
            }
            else if(S[T]=='-')
            {
                flag=-1;
            }
            else
            {
                a=flag*(a*flag*10+S[T]-'0');
                ++i;
            }
            ++T;
        }
    }
    int read(char a[])
    {
        int i=0;
        while(1)
        {
            if(T==end&&!in())   return i;
            if(S[T]==' '||S[T]=='\n'||S[T]=='\r')
            {
                if(i)return i;
            }
            else    a[i++]=S[T];
            ++T;
        }
    }
}
#define MAXN 10010
#define MAXM 100010
#define INF 1000000000
int n,m,s,t,a,b,k,head[MAXN],_edge=1,way[8][2],d[MAXN],nowflow[MAXN],come[MAXN];
char map[105][105];
struct EDGE
{
    int a,b,v,flow,next;
    EDGE(int a=0,int b=0,int v=0,int flow=0,int
next=0):a(a),b(b),v(v),flow(flow),next(next){}
}edge[2*MAXM];
/* Variable explain:

*/
int ntoi(int a,int b)   {return (a-1)*m+b;}
void adde(int a,int b,int v,int flow)
{
    edge[++_edge]=EDGE(a,b,v,flow,head[a]);
    head[a]=_edge;
    edge[++_edge]=EDGE(b,a,-v,0,head[b]);
    head[b]=_edge;
}
void read()
{
    using namespace IO;
    // freopen("chessc.in","r",stdin);
    // freopen("chessc.out","w",stdout);
    // freopen("in","r",stdin);
    // freopen("out","w",stdout);
    read(n),read(m),read(k),read(a),read(b);
    s=n*m+1,t=s+1;
    way[0][0]=a,way[0][1]=b,way[1][0]=a,way[1][1]=-b,
```

```
    way[2][0]=-a,way[2][1]=b,way[3][0]=-a,way[3][1]=-b,
    way[4][0]=b,way[4][1]=a,way[5][0]=b,way[5][1]=-a,
    way[6][0]=-b,way[6][1]=a,way[7][0]=-b,way[7][1]=-a;
    for(int i=1;i<=n;++i)   read(map[i]+1);
    for(int i=1;i<=k;++i)//s向棋子连边
    {
        int ls1,ls2;
        read(ls1),read(ls2);
        adde(s,ntoi(ls1,ls2),0,1);
    }
    for(int i=1;i<=k;++i)//棋子到达的位置向t连边
    {
        int ls1,ls2;
        read(ls1),read(ls2);
        adde(ntoi(ls1,ls2),t,0,1);
    }
    for(int i=1;i<=n;++i)//棋盘通路连边
    {
        for(int j=1;j<=m;++j)
        {
            for(int k=0;k<8;++k)
            {
                if(i+way[k][0]<=0||i+way[k][0]>n||j+way[k][1]<=0||j+way[k][1]>m||
                map[i+way[k][0]][j+way[k][1]]=='*')continue;
                adde(ntoi(i,j),ntoi(i+way[k][0],j+way[k][1]),1,1e9);
            }
        }
    }
    return;
}
bool spfa()//最短路
{
    deque <int> q;
    bool vis[MAXN]={0};
    for(int i=1;i<=t;++i)d[i]=INF;
    d[s]=0;
    nowflow[s]=INF;
    q.push_back(s);
    vis[s]=1;
    while(!q.empty())
    {
        int a=q.front();
        q.pop_front();
        vis[a]=0;
        for(int i=head[a];i;i=edge[i].next)
        {
            int b=edge[i].b,v=edge[i].v,flow=edge[i].flow;
            if(d[b]>d[a]+v&&flow)
            {
                d[b]=d[a]+v;
                nowflow[b]=min(nowflow[a],flow);
                come[b]=i;
                if(!vis[b])
                {
                    q.push_back(b);
                    vis[b]=1;
                }
            }
        }
    }
    return d[t]!=INF;
}
int maxflow()
{
    int ans=0,flow=0;
```

```
    while(spfa())
    {
        ans+=nowflow[t]*d[t];
        flow+=nowflow[t];
        for(int i=t;i!=s;i=edge[come[i]].a)
        {
            edge[come[i]].flow-=nowflow[t];
            edge[come[i]^1].flow+=nowflow[t];
        }
    }
    return ans;
}
int main()
{
    read();
    printf("%d",maxflow());
    return 0;
}
```

<题目跳转> <查看代码>