瑰丽华尔兹

一. 考察内容：

动态规划 单调队列优化

二. 题目分析：

[写题思路]

考虑用动态规划实现，由于T太大，考虑用k做状态的一维，设f[i][j][k]为第i次倾斜之后，钢琴在(j,k)位置时的最大移动距离。

转移：对于每次倾斜，可以向倾斜的位置移动(0~倾斜时间)格，对于f[i][j][k]，倾斜后在(j,k)的状态，每次倾斜都需要通过倾斜之前的状态f[i-1]来转移，j和k的变化由倾斜方向决定，时间复杂度O(n^3*k)≈1.6e9。

考虑对转移进行优化，对于每一个状态f[i][j][k]，他的转移需要考察f[i-1]中的一行，取他们的最优值，这个考察是线性的，考虑用单调队列维护，这样可以使得每转移棋盘上的一行(或一列)，需要的时间复杂度是线性的，即不需要对于每个状态都枚举他之前的状态转移，只需要集体枚举一遍即可，这样的时间复杂度是O(n^2*k)≈8e6。

三. 代码实现：

```
#define _CRT_SECURE_NO_DEPRECATE
/*************************
*创建时间：2018 08 09
*文件类型：源代码文件
*题目来源：洛谷
*当前状态：已通过
*备忘录：动态规划
*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 1
    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 201
int n,m,x,y,K,f[MAXN][MAXN][MAXN];
char a[MAXN][MAXN];
struct MOVE
{
    int d,len;//方向、持续时间
}b[MAXN];
struct DDDL
{
    int q[MAXN],head,tail,d[MAXN],flag;
    void set(int v)
    {
        flag=v;
        head=0;
        tail=-1;
    }
    void push(int x)
    {
        while(head<=tail&&d[q[tail]]+flag*x<=d[x]+flag*q[tail])--tail;
        q[++tail]=x;
    }
    void update(int x)
    {
        if(flag==1)
            while(head<=tail&&q[head]<x)++head;
        else
            while(head<=tail&&q[head]>x)++head;
    }
    int query(){return d[q[head]];}
}q;
/* Variable explain:

*/
void read()
{
    using namespace IO;
    int ls1,ls2;
    // freopen("in","r",stdin);
    // freopen("out","w",stdout);
    read(n);
    read(m);
```

```
    read(x);
    read(y);
    read(K);
    for(int i=1;i<=n;++i)
    {
        read(a[i]+1);
    }
    for(int i=1;i<=K;++i)
    {
        read(ls1);
        read(ls2);
        read(b[i].d);
        b[i].len=ls2-ls1+1;
    }
    return;
}
void dp()
{
    int ans=-10;
    //状态f[i][j][k]:完成第i次平移状态之后，在j,k时的最大移动距离。
    for(int k=0;k<=K;++k)for(int i=1;i<=n;++i)for(int j=1;j<=m;++j)f[k][i][j]=-1e9;
    f[0][x][y]=0;
    for(int i=1;i<=K;++i)
    {
        if(b[i].d==3)
        {
            for(int j=1;j<=n;++j)
            {
                q.set(-1);
                for(int k=m;k;--k)
                {
                    if(a[j][k]=='x'){q.set(-1);continue;}
                    q.d[k]=f[i-1][j][k];
                    q.push(k);
                    q.update(k+b[i].len);
                    f[i][j][k]=q.query()-k+q.q[q.head];
                    // printf("%d %d %d %d\n",i,j,k,f[i][j][k]);
                }
            }
        }
        if(b[i].d==4)
        {
            for(int j=1;j<=n;++j)
            {
                q.set(1);
                for(int k=1;k<=m;++k)
                {
                    if(a[j][k]=='x'){q.set(1);continue;}
                    q.d[k]=f[i-1][j][k];
                    q.push(k);
                    q.update(k-b[i].len);
                    f[i][j][k]=q.query()+k-q.q[q.head];
                    // printf("%d %d %d %d\n",i,j,k,f[i][j][k]);
                }
            }
        }
        if(b[i].d==1)
        {
            for(int j=1;j<=m;++j)
            {
                q.set(-1);
                for(int k=n;k;--k)
                {
                    if(a[k][j]=='x'){q.set(-1);continue;}
                    q.d[k]=f[i-1][k][j];
                    q.push(k);
```

```
                    q.update(k+b[i].len);
                    f[i][k][j]=q.query()-k+q.q[q.head];
                    // printf("%d %d %d %d\n",i,k,j,f[i][k][j]);
                }
            }
        }
        if(b[i].d==2)
        {
            for(int j=1;j<=m;++j)
            {
                q.set(1);
                for(int k=1;k<=n;++k)
                {
                    if(a[k][j]=='x'){q.set(1);continue;}
                    q.d[k]=f[i-1][k][j];
                    q.push(k);
                    q.update(k-b[i].len);
                    f[i][k][j]=q.query()+k-q.q[q.head];
                    // printf("%d %d %d %d\n",i,k,j,f[i][k][j]);
                }
            }
        }
    }
    for(int i=1;i<=n;++i)for(int j=1;j<=m;++j)ans=max(ans,f[K][i][j]);
    printf("%d\n",ans);
}
int main()
{
    read();
    dp();
    return 0;
}
```

<题目跳转> <查看代码>