P1174

【算法分析】

一开始先把每一列求一个用j发子弹打出的最大值，然后一列一列的扫过去，

设f[i,j]为钱i列用了j发子弹的最大值，可以得出方程

f[i,j]:=max(f[i-1,k]+sum[i,j-k])sum为上面求出的i列用j发子弹的最大值

#include <bits/stdc++.h>

using namespace std;

const int maxn=255;

int k,i,j,n,m,ii,ans;

char c,d;

int a[maxn][maxn],b[maxn][maxn],dp[maxn][maxn],s[maxn][maxn];

bool flag[maxn][maxn];

int main()

{

cin>>n>>m>>k;

for(i=1; i<=n; i++)

for(j=1; j<=m; j++)

{

cin>>a[i][j]>>c;

flag[i][j]=c=='Y'?true:false;

}

for(int i=1; i<=m; i++)

for(int j=1; j<=n; j++)

{

b[i][j]=b[i][j-1]+a[n-j+1][i];

if(flag[n-j+1][i] && s[i][j-1]<k)

s[i][j]=s[i][j-1];

else

s[i][j]=s[i][j-1]+1;

}

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

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

dp[i][j]=-1;

for(int i=1; i<=n; i++)

if(s[1][i]<=k)

dp[1][s[1][i]]=b[1][i];

dp[1][0]=0;

for(i=1; i<=m-1; i++)

for(j=0; j<=k; j++)

{

if(dp[i][j]==-1)

continue;

dp[i+1][j]=max(dp[i+1][j],dp[i][j]);

for( ii=1; ii<=n; ii++)

{

if(s[i+1][ii]+j>k)

break;

dp[i+1][s[i+1][ii]+j] =max(dp[i][j]+b[i+1][ii],dp[i+1][s[i+1][ii]+j]);

ans=max(ans,dp[i+1][s[i+1][ii]+j]);

}

}

cout<<ans<<endl;

}

首先把最下层为标识为“Y”的砖块全打掉，反正打了也白打，不打白不打。

预处理3个数组：now[j][i]代表第j列我们要打第i个砖块可以得到的分数，注意，如果第i个砖块上有Y，要把Y也加到now里面。。。第二个数组res[j][i]，这个是个前缀和，表示第j列我们要打第i个砖块的得分(这个和now的区别在于如果i砖块上面有Y也不加进去)。第三个数组ci[j][i]表示第j列我们要打第i个砖块需要多少子弹。

DP[j][k][0]表示不从后面借子弹时前j列用k颗子弹能得到的最大分数，DP[j][k][1]表示借子弹时的最大分数。

假设第一列砖块如图所示：



如果不借子弹，那么一发子弹只能把2（N）打掉，这样得分是2，即Dp[1][1][0]=2。

如果借一个子弹，那么可以把两个砖块打掉，打掉2（Y）后奖励的一发子弹还回去就可以了，这样得分是4，即Dp[1][1][1]=4。

dp[j][k][0] = max(dp[j][k][0], max(dp[j - 1][k - ci[j][i]][1], dp[j - 1][k - ci[j][i]][0]) + res[j][i]);

就是前面j-1列(找第j列借子弹)借或不借，但是因为我们求的是dp[j][k][0]（当前第j列不去找别人借子弹），所以借到子弹就能打掉的砖块只能干瞪眼，所以是res[j][i]。

dp[j][k][0] = max(dp[j][k][0], dp[j - 1][k - ci[j][i]][0] + now[j][i]);

前面不找第j列借，但是当前第j列没有找后面借子弹，但是向前面借子弹了，所以能打掉的都打掉，所以是now[j][i]。最后得到的前i列的整体因为自给自足（j-1列因为没有欠子弹，所以可以借给i子弹），所以不用向别人借子弹而且能借给别人，所以dp第三维下标是0。

dp[j][k][1] = max(dp[j][k][1], dp[j - 1][k - ci[j][i]][1] + now[j][i]);

是j前面的找第j列借子弹，第j列又找后面列借子弹。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | #include <bits/stdc++.h>  using namespace std;  const int maxn=255;  #define M(a) memset(a,0,sizeof a)  const int mxn = 205;  int n, m, p, ans;  char ch;  int a[mxn][mxn], whe[mxn], res[mxn][mxn];  int dp[mxn][mxn][2], ci[mxn][mxn], now[mxn][mxn]; //0:不需要借，1:需要借  bool b[mxn][mxn];  int main()  {  int i, j, k;  cin>>n>>m>>p;  //子弹为0，直接输出  if (p == 0)  {  cout<<"0\n";  return 0;  }  //a[i][j]表示打掉第i行j列位置的砖块的得分  //b[i][j] = 1表示打掉第i行j列位置的砖块可以奖励一颗子弹  //读数据的时候按行翻转了，就是上下翻转了  for (i = n; i >= 1; i--)  for(j=1; j<=m; j++)  {  cin>>a[i][j]>>ch;  b[i][j]= ch == 'Y' ? 1:0;  }  //贪心打掉最下层的Y ,因为这里上下翻转了，所以打掉的是上层的  for(j=1; j<=m; j++) //j列  {  for(i=1; i<=n; i++) //i行  {  if (!b[i][j])  break;  ans += a[i][j];  }  //因为上下翻转了，whe[j]=i表示第j列第一个N的位置为i  //其实也就是贪心打掉最下层的Y之后剩下的N  whe[j] = i;  }  //now[j][i]代表第j列我们要打第i个砖块可以得到的分数，注意，如果第i个砖块上有Y，要把Y也加到now里面  //res[j][i]，这个是个前缀和，表示第j列我们要打第i个砖块的得分(这个和now的区别在于如果i砖块上面有Y也不加进去)  for(j=1; j<=m; j++)  for(i=whe[j]; i<=n; i++)  res[j][i] = res[j][i - 1] + a[i][j];  for(j=1; j<=m; j++)  for(i=whe[j]; i<=n; i++)  now[j][i] = res[j][i];  //ci[j][i]表示第j列我们要打第i个砖块需要多少子弹  //修正now[j][i]数组和填充ci[j][i]数组  for(j=1; j<=m; j++)  {  ci[j][whe[j]] = 1;//每列的第一个N肯定要打掉  for(i=whe[j]; i<=n; i++)  {  int tmp = i;  //b[i + 1][j]对应的为Y  //这部分理解可以画个小图  while (b[i + 1][j])  i++;  now[j][tmp] = res[j][i];  ci[j][i + 1] = ci[j][tmp] + 1;  }  }  //初始值 前j列在不借子弹的情况下用0发子弹打  for(j=0; j<=m; j++)  dp[j][0][0] = -1e8;  for(j=1; j<=m; j++)  for(k=1; k<=p; k++)  {  dp[j][k][0] = max(dp[j][k][0], dp[j - 1][k][0]);  dp[j][k][1] = max(dp[j][k][1], dp[j - 1][k][1]);  for(i=whe[j]; i<=n; i++)  if (!b[i][j] && k >= ci[j][i])//没奖励子弹并且没超过子弹数  {  dp[j][k][0] = max(dp[j][k][0], max(dp[j - 1][k - ci[j][i]][1], dp[j - 1][k - ci[j][i]][0]) + res[j][i]);  dp[j][k][0] = max(dp[j][k][0], dp[j - 1][k - ci[j][i]][0] + now[j][i]);  dp[j][k][1] = max(dp[j][k][1], dp[j - 1][k - ci[j][i]][1] + now[j][i]);  }  }  printf("%d\n", dp[m][p][0] + ans);//最后输出的是不欠子弹的，所以欠子弹的情况无视了  return 0;  } |