滚动数组

DP的空间优化

例0：植物大战僵尸 NKOJ 2422

植物大战僵尸的游戏里有一条水平道路，道路的一端是入口，另一端是房子。僵尸会从道路的入口一端向房子一端移动。这条道路刚好穿过N块连续的空地。初始时，僵尸通过每块空地的时间是T秒。玩家可以在这N个空地中种植植物以攻击经过的僵尸，每块空地中只能种植一种植物。

共有三种不同类型的植物，分别是**红草**、**蓝草**和**绿草**，作用分别是**攻击**、**减速**以及**下毒**。每种植物只能在僵尸通过它所在空地的这段时间内攻击到僵尸。

当僵尸经过一块**红草**所在的空地时，**每秒钟生命值会减少R**点；当僵尸从一块**蓝草**所在的空地走出之后，**通过每块空地的时间延长B秒**；当僵尸从一块**绿草**所在的空地走出之后，**每秒钟会因中毒减少G点生命**值。蓝草的减速效果和绿草的下毒效果是可以累加的。也就是说，僵尸通过n块蓝草所在的空地之后，它通过每块空地的时间会变成T+B*n秒；僵尸通过n块绿草所在的空地之后，它每秒钟会因中毒失去G*n点生命值。注：**减速和中毒效果会一直持续下去**

问：怎样在这N块空地里种植各种类型的植物，才能使通过的僵尸失去的生命值最大。输出这个最大值。

1<=N<=2000　空间限制3m

例0：植物大战僵尸 问题分析

首先，一个显然的贪心是**红草一定排在最后**.

所以，若使用r个红草，只须确定其余N-r个排在前面的蓝、绿草如何摆放，可以使用动规来解决

设f[x][y]表示前面x+y个位置使用x个绿草、y个蓝草可造成的最大伤害.

求解f[x][y]只须讨论最后一格放的是何种草,可以由f[x-1][y]和f[x][y-1]递推得到。方程如下：

f[x][y]=max {

f[x-1][y]+(T+y*B)*(x-1)*G;

f[x][y-1]+(T+(y-1)*B)*x*G;

}

经过第(x+y)号格子时，中毒减少的生命值

f[x-1][y]+(T+y*B)*(x-1)*G;
表示最后一个位置放绿草,通过最后一格的时间是T+y*B，中毒造成的伤害是每时间(x-1)*G
f[x][y-1]+(T+(y-1)*B)*x*G; }
表示最后一个位置放蓝草,通过最后一格的时间是T+(y-1)*B，中毒造成的伤害是每时间x*G

对于每个f[x][y]，加上最后的N-x-y个红草带来的伤害，以及在红草的区域由于中毒带来的伤害（后者易忽视），找到最优值即可

Ans= max{ f[x][y]+(N-x-y)*(T+y*B)*(R+x*G) }

例0：植物大战僵尸 参考代码

```
for(y=0;y<=N;y++)
    for(x=0;x<=N-y;x++)
    {
        if(y>0)f[y][x]=max(f[y][x],f[y-1][x]+(T+B*(y-1))*x*G);
        if(x>0)f[y][x]=max(f[y][x],f[y][x-1]+(T+B*y)*(x-1)*G);
        if(N-y-x>0)ans=max(ans,f[y][x]+(N-y-x)*(T+y*B)*(R+x*G));
    }
cout<<ans;
```

对未来的影响

但本题空间限制只有3m,我们需要用**滚动数组**来优化空间

滚动数组的作用

滚动数组主要应用在递推或动态规划中，进行空间压缩。

例1：斐波拉楔数列

数列1, 1, 2, 3, 5, 8, 13, 21, ...称为非波拉楔数列。

从键盘输入一整数N (2<N<=10000)，求出非波拉楔数列的第N项。

```
//递推式Ai=Ai-1+Ai-2
long long A[10001];
A[0]=A[1]=1;
for(int i=2;i<=N;i++)A[i]=A[i-1]+A[i-2];
cout<<A[N];
```

左边代码耗费了10000个long long的空间，但我们发现A[i]只和A[i-1]和A[i-2]有关;可以考虑使用滚动数组来节省空间。

```
long long A[3];
A[0]=A[1]=1;
for(int i=2;i<=N;i++) A[i%3]=A[(i-1)%3]+A[(i-2)%3];
cout<<A[N%3];
```

我们只保留**最近**的3个解，数组好象在“滚动“一样，所以叫滚动数组!

例2：最长公共子序列LCS

给出两个整数序列，序列X和序列Y,求这两个序列的最长公共子序列。

1<=序列的长度<=5000　空间限制64m

状态: f[i][j]表示X序列的前i项与Y序列的前j项能构成的最长公共子序列的长度

也就是 $<x_1,x_2,\dots,x_i>$ 和 $<y_1,y_2,\dots,y_j>$ 的最长公共子序列的长度值

决策:

1. 当 $x_m==y_n$ 时，只需找出 $<x_1,\dots,x_{m-1}>$ 和 $<y_1,\dots,y_{n-1}>$ 的最长公共子序列，然后在其尾部加上 $x_m(==y_n)$ 这一项，即可得X和Y的一个最长公共子序列；

2. 当 $x_m \neq y_n$ 时，必须解两个子问题：

   a.找出 $<x_1,\dots,x_{m-1}>$ 和 $<y_1,\dots,y_n>$ 的最长公共子序列；

   b.找出 $<x_1,\dots,x_m>$ 和 $<y_1,\dots,y_{n-1}>$ 的最长公共子序列；

   这两个公共子序列中较长者即为X和Y的最长公共子序列；

方程:

$$f[i][j]=\begin{cases} f[i-1][j-1]+1 & (x[i]==y[j]) \\ \max\{f[i-1][j],\ f[i][j-1]\} & (x[i]!=y[j]) \end{cases}$$

例2：最长公共子序列LCS

给出两个整数序列，序列X和序列Y,求这两个序列的最长公共子序列。

1<=序列的长度<=5000　空间限制64m

```
int f[5000][5000];
for(i=1; i<=LenX; i++)
    for(j=1; j<=LenY; j++)
        if(X[i]==Y[j])f[i][j]=f[i-1][j-1]+1;  else f[i][j]=max(f[i-1][j],f[i][j-1]);
```

左边代码f[ ][ ]数组耗费了100m左右的空间
不能满足题目要求！

观察发现上面的f[i][j]只依赖于f[i-1][...]和f[i][...],可以考虑用滚动数组来压缩空间！

```
int f[2][5000];
for(i=1; i<=LenX; i++)
    for(j=1; j<=LenY; j++)
        if(X[i]==Y[j])f[i%2][j]=f[(i-1)%2][j-1]+1;
        else f[i%2][j]=max(f[(i-1)%2][j], f[i%2][j-1]);
```

左边代码f[ ][ ]数组耗费了0.04m左右的空间
较上面代码大大优化了空间！

例2：最长公共子序列LCS

```
int f[2][5000];
for(i=1; i<=LenX; i++)
    for(j=1; j<=LenY; j++)
        if(X[i]==Y[j])f[i%2][j]=f[(i-1)%2][j-1]+1;
        else f[i%2][j]=max(f[(i-1)%2][j], f[i%2][j-1]);
```

“i % 2”相当于是“判奇偶，而“%”运算是比较耗时的，所以上面滚动数组代码还可以考虑用位运算来进行优化！

```
int f[2][5000];
for(i=1; i<=LenX; i++)
    for(j=1; j<=LenY; j++)
        if(X[i]==Y[j])f[i&1][j]=f[(i-1)&1][j-1]+1;
        else f[i&1][j]=max(f[(i-1)&1][j],f[i&1][j-1]);
```

滚动数组的特点

通过前面的例子我们看出：

滚动数组几乎不能带来时间上的优化，但在空间上的优化是相当明显的。

例3：服务员

一个餐厅有三个服务员。如果某桌顾客有请求，某个服务服务员必须赶到那个地方去（那个地方没有其他服务员）某一时刻只有一个服务员能移动。被请求后，他才能移动，不允许在同样的位置出现两个服务员。从p到q移动一个服务员，需要花费cost(p,q)元，这个函数没有必要对称，但是cost(p,p)=0。

餐厅必须满足所有的请求。目标在满足所有请求的情况下，使得总花费最少。

3<=餐桌数量M<=200, 1<=请求数量N<=1000,　时限3s　空间 64m

例3：服务员

很容易设计出朴素的动规方程：

Q[i]为第i个请求。F[i][a][b][c]表示满足前i个请求时三个服务员位置分别为a,b,c时花费代价的最小值。

F[i][Q[i]][b][c]=min{ F[i-1][a][b][c]+Cost[a][Q[i]] }

F[i][a][Q[i]][c]=min{ F[i-1][a][b][c]+Cost[b][Q[i]] }

F[i][a][b][Q[i]]=min{ F[i-1][a][b][c]+Cost[c][Q[i]] }

观察数据规模，这个动规方程的时间复杂度为1000*200*200*200=$8*10^9$，肯定超时。

例3：服务员

F[i][Q[i]][b][c]=min{ F[i-1][a][b][c]+Cost[a][Q[i]] }
F[i][a][Q[i]][c]=min{ F[i-1][a][b][c]+Cost[b][Q[i]] }
F[i][a][b][Q[i]]=min{ F[i-1][a][b][c]+Cost[c][Q[i]] }

观察会发现：F[i][a][b][c]中记录的三个服务员的位置必有一个为Q[i]，所以，优化的动机又出现了：

我们可以将记录三个服务员位置的F改为记录其中两个服务员的位置，另一个服务员的位置为Q[i]。

F[i][a][b]表示处理完第i个请求后，一个服务员位于Q[i]位置，另外两个服务员分别位于a,b位置的最小代价。

于是动规方程变为：

f[i][a][b] = min{ f[i - 1][a][b] + Cost[Q[i - 1]][Q[i]] }

f[i][a][Q[i - 1]] = min{ f[i - 1][a][b] + Cost[b][Q[i]] }

f[i][b][Q[i - 1]] = min{ f[i - 1][a][b] + Cost[a][Q[i]] }

这个动规方程的时间复杂度为1000*200*200不会超时。
但空间复杂度为1000*200*200，会超空间限制

例3：服务员

$f[i][a][b] = \min\{ f[i-1][a][b] + Cost[Q[i-1]][Q[i]] \}$
$f[i][a][Q[i-1]] = \min\{ f[i-1][a][b] + Cost[b][Q[i]] \}$
$f[i][b][Q[i-1]] = \min\{ f[i-1][a][b] + Cost[a][Q[i]] \}$

观察会发现：F数组只与F[i][ ][ ]和F[i-1][ ][ ]有关，考虑滚动数组优化空间

于是动规方程变为：

$f[i][a][b] = \min\{ f[i \wedge 1][a][b] + Cost[Q[i-1]][Q[i]] \}$
$f[i][a][Q[i-1]] = \min\{ f[i \wedge 1][a][b] + Cost[b][Q[i]] \}$
$f[i][b][Q[i-1]] = \min\{ f[i \wedge 1][a][b] + Cost[a][Q[i]] \}$

这样就以空间复杂度为2*200*200、时间复杂度为1000*200*200完美地解决了。

课后习题

NKOJ 3686，4119

【思维训练】 NKOJ 1548，3559