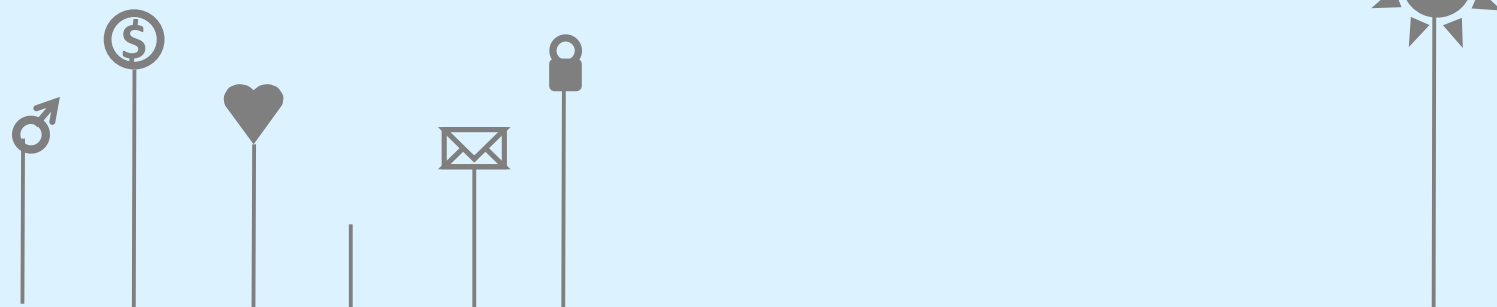


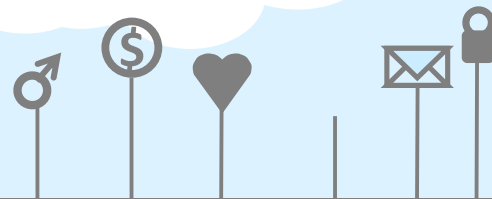
# 滚动数组

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。

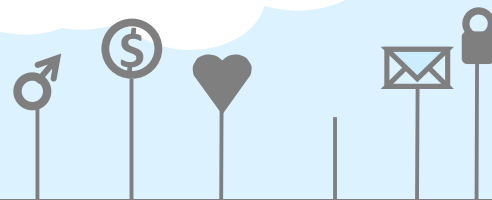
共有三种不同类型的植物，分别是**红草**、**蓝草**和**绿草**，作用分别是**攻击**、**减速**以及**下毒**。每种植物只能在僵尸通过它所在空地的这段时间内攻击到僵尸。

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。

$1 \leq N \leq 2000$     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：

$$f[x][y] = \max \left\{ \begin{array}{l} f[x-1][y] + (T+y*B)*(x-1)*G; \\ f[x][y-1] + (T+(y-1)*B)*x*G; \end{array} \right\}$$

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 $(x-1)*G$

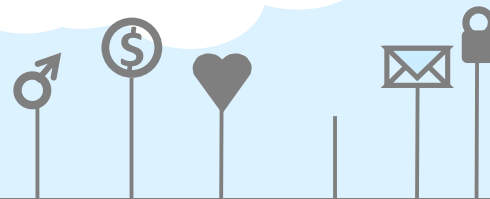
$$f[x][y-1] + (T+(y-1)*B)*x*G; \}$$

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

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

## 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

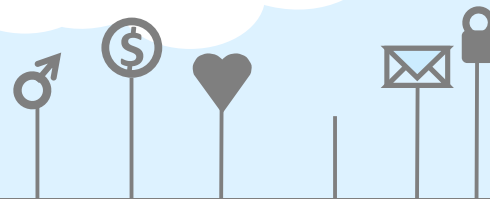
但本题空间限制只有3m,我们需要用**滚动数组**来优化空间



# 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 ( $2 < N \leq 10000$ )，求出非波拉楔数列的第N项。

//递推式 $A_i = A_{i-1} + A_{i-2}$

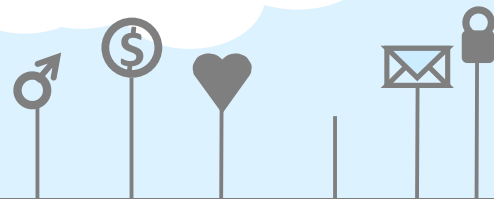
```
long long A[10001];  
A[0]=A[1]=1;  
for(int i=2;i<=N;i++)A[i]=A[i-1]+A[i-2];  
cout<<A[N];
```

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。

$1 \leq \text{序列的长度} \leq 5000$  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  $\langle x_1, x_2, \dots, x_i \rangle$  和  $\langle y_1, y_2, \dots, y_j \rangle$  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  $\langle x_1, \dots, x_{m-1} \rangle$  和  $\langle y_1, \dots, y_{n-1} \rangle$  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：

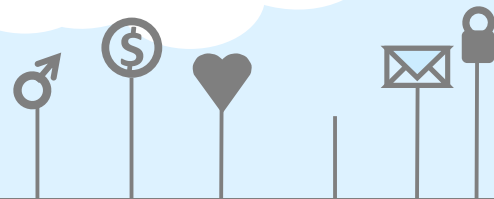
a. 找出  $\langle x_1, \dots, x_{m-1} \rangle$  和  $\langle y_1, \dots, y_n \rangle$  的最长公共子序列；

b. 找出  $\langle x_1, \dots, x_m \rangle$  和  $\langle y_1, \dots, y_{n-1} \rangle$  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；

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

## 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。

$1 \leq \text{序列的长度} \leq 5000$     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[i][j]数组耗费了100m左右的空间  
不能满足题目要求！

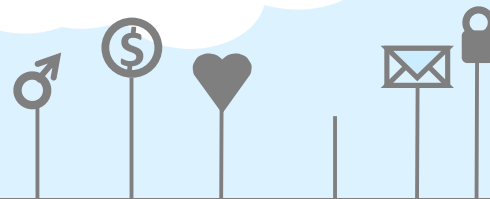
观察发现上面的f[i][j]只依赖于f[i-1][...]和f[i][...],可以考虑用滚动数组来压缩空间！

```
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[i][j]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

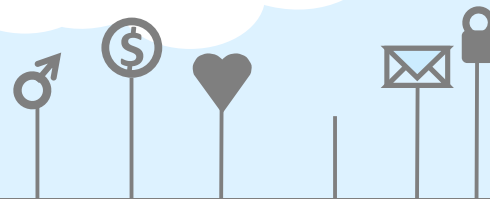


```
int f[2][5000];
for(i=1; i<=LenX; i++)
    for(j=1; j<=LenY; j++)
        if(X[i]==Y[j])f[i%2][j]=f[(i-1)%2][j-1]+1;
        else f[i%2][j]=max(f[(i-1)%2][j], f[i%2][j-1]);
```

“ $i \% 2$ ”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]);
```

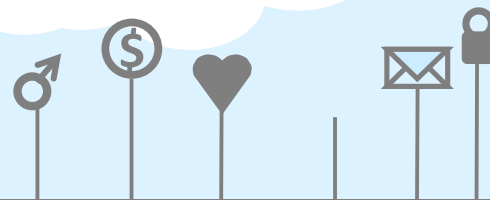
# 滚动数组的特点



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

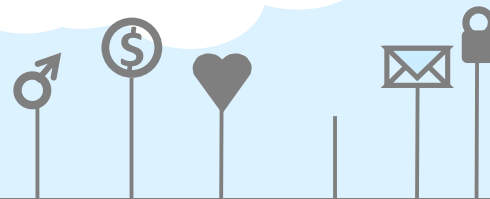


一个餐厅有三个服务员。如果某桌顾客有请求，某个服务服务员必须赶到那个地方去（那个地方没有其他服务员）某一时刻只有一个服务员能移动。被请求后，他才能移动，不允许在同样的位置出现两个服务员。从 $p$ 到 $q$ 移动一个服务员，需要花费 $\text{cost}(p,q)$ 元，这个函数没有必要对称，但是 $\text{cost}(p,p)=0$ 。

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。

$3 \leq \text{餐桌数量 } M \leq 200$ ,  $1 \leq \text{请求数量 } N \leq 1000$ , 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。

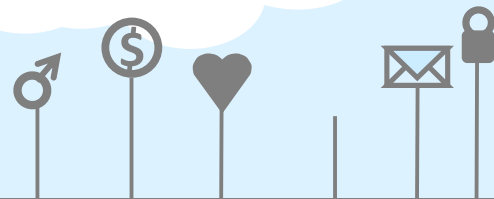
$$F[i][Q[i]][b][c] = \min\{ F[i-1][a][b][c] + \text{Cost}[a][Q[i]] \}$$

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

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

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



$$\begin{aligned} F[i][Q[i]][b][c] &= \min\{ F[i-1][a][b][c] + \text{Cost}[a][Q[i]] \} \\ F[i][a][Q[i]][c] &= \min\{ F[i-1][a][b][c] + \text{Cost}[b][Q[i]] \} \\ F[i][a][b][Q[i]] &= \min\{ F[i-1][a][b][c] + \text{Cost}[c][Q[i]] \} \end{aligned}$$

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 $Q[i]$ 。

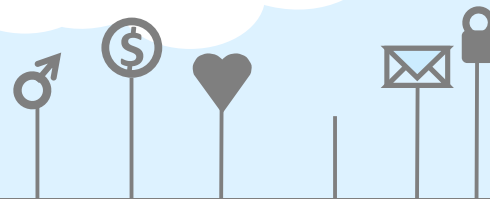
$F[i][a][b]$ 表示处理完第 $i$ 个请求后，一个服务员位于 $Q[i]$ 位置，另外两个服务员分别位于 $a, b$ 位置的最小代价。

于是动规方程变为：

$$\begin{aligned} f[i][a][b] &= \min\{ f[i-1][a][b] + \text{Cost}[Q[i-1]][Q[i]] \} \\ f[i][a][Q[i-1]] &= \min\{ f[i-1][a][b] + \text{Cost}[b][Q[i]] \} \\ f[i][b][Q[i-1]] &= \min\{ f[i-1][a][b] + \text{Cost}[a][Q[i]] \} \end{aligned}$$

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

## 例3：服务员



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

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

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

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：

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

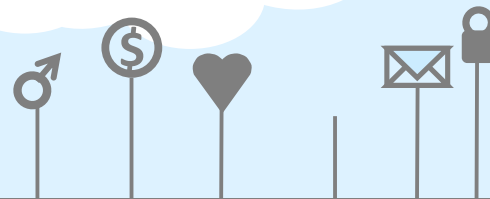
$$f[i][a][Q[i-1]] = \min\{ f[i-1][a][b] + \text{Cost}[b][Q[i]] \}$$

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

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



## 课后习题



NKOJ 3686, 4119

【思维训练】 NKoj 1548, 3559