费用提前计算

DP进阶训练

例1：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。

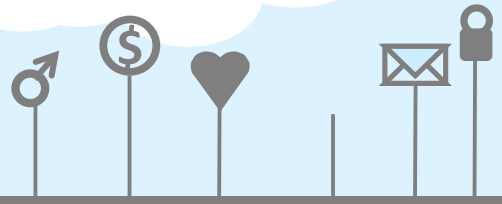
共有三种不同类型的植物，分别是**红草**、**蓝草**和**绿草**，作用分别是**攻击**、**减速**以及**下毒**。每种植物只能在僵尸通过它所在空地的这段时间内攻击到僵尸。

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1：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) }

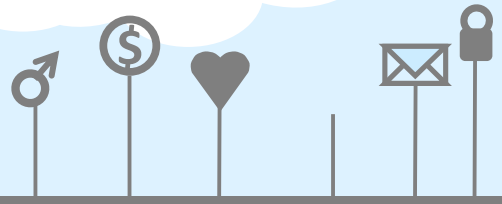
例1：植物大战僵尸 参考代码

```
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

例2：边跑边吃草 NKOJ 3693



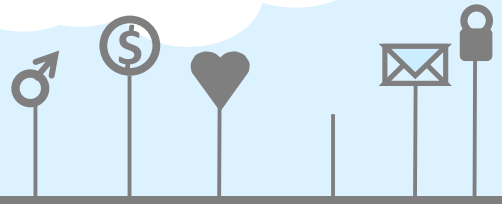
有N块草地，我们认为草地是数轴上的一些点。奶牛贝西想把它们全部吃光。于是它开始**左右行走**，吃草。贝西开始的时候站在p位置。贝西的移动速度是一个单位时间一个单位距离。

不幸的是，草如果长时间不吃，就会腐败。我们定义一堆草的腐败值是从贝西开始到吃到这堆草的总时间。

贝西可不想吃太腐败的草，请帮它安排一个路线，使得它吃完所有的草后，总腐败值最小。

1<=N<=3000

例2：边跑边吃草 问题分析1

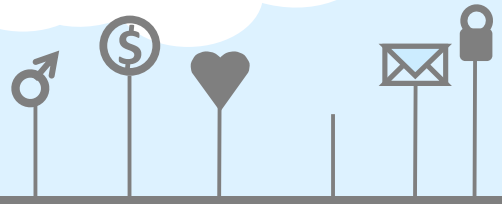


显然，贝西任何时间吃的草都是一个连续区间！
因为，它不可能路过一堆草却不吃它。
于是，这题看起来很像是一道区间DP题！

仔细观察这个问题会发现，每一次行走，都**会对未来产生影响**。
第一次行走的时间，会被累计n次；
第二个行走的时间，会被累计n-1次；
.....
我们**需要把对未来的影响纳入DP中去讨论，以消除后效性**。

如果我们按照区间DP的讨论，设定状态：
F[i][j]表示奶牛将区间i到j的草吃完，所需最小代价。
我们会发现，吃完该区间的草后，奶牛停留的位置会对未来的决策产生影响。
我们需要改进状态。

例2：边跑边吃草 问题分析2



状态：

F[i][j][0]表示奶牛将i—j区间的草都吃完，最后停留在i位置，的最小代价

F[i][j][1]表示奶牛将i—j区间的草都吃完，最后停留在j位置，的最小代价

方程：

提前计算对未来的影响

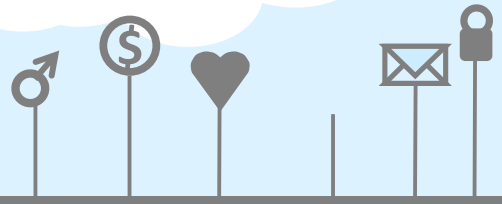
```
F[i][j][0]=min{
                F[i+1][j][0]+(n-j+i)*(a[i+1]-a[i])     //从i+1走到i,停在i
                F[i+1][j][1]+(n-j+i)*(a[j]-a[i])       //从j走到i,停在i
            }
F[i][j][1]=min{
                F[i][j-1][1]+(n-j+i)*(a[j]-a[j-1])     //从j-1走到j,停在j
                F[i][j-1][0]+(n-j+i)*(a[j]-a[i])       //从i走到j,停在j
            }
```

注意：我们事先需要将每个点按位置由小到大排序

例2：边跑边吃草 参考代码

```
//先把所有点按由小到大排序，包括起点。
 f[Start][Start][0]=f[Start][Start][1]=0;       //Start表示起点
 for(int i=Start;i>=1;i--)
     for(int j=Start;j<=n;j++)
     {
         f[i][j][0]=min(f[i][j][0],f[i+1][j][0]+(n-(j-i))*(a[i+1]-a[i]));
         f[i][j][0]=min(f[i][j][0],f[i+1][j][1]+(n-(j-i))*(a[j]-a[i]));
         f[i][j][1]=min(f[i][j][1],f[i][j-1][1]+(n-(j-i))*(a[j]-a[j-1]));
         f[i][j][1]=min(f[i][j][1],f[i][j-1][0]+(n-(j-i))*(a[j]-a[i]));
     }
 printf("%d\n",min(f[1][n][0],f[1][n][1]));
```

例3：任务安排 NKOJ 1047

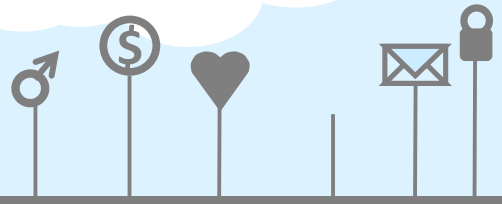


N个任务排成一个序列在一台机器上等待完成（顺序不得改变），这N个任务可被分成若干批，每批包含相邻的若干任务。从时刻0开始，这些任务被分批加工，第i个任务单独完成所需的时间是Ti。**在每批任务开始前，机器需要启动时间S**，而完成这批任务所需的时间是各个任务需要时间的总和（同一批任务将在同一时刻完成）。每个任务的费用是它的完成时刻乘以一个费用系数Fi。请确定一个分组方案，使得总费用最小。

例如：S=1；T={1,3,4,2,1}；F={3,2,3,3,4}。如果分组方案是{1,2}、{3}、{4,5}，则完成时间分别为{5,5,10,14,14}，费用C={15,10,30,42,56}，总费用就是153。

1<=N<=5000

例3：任务安排 问题分析1



采用类似1006”护卫队”的分析方法：讨论如何分组？

第1个任务的最小费用 dp[1]=(s+t1)*f1 time[1]=s+t1

前2个任务的最小费用

dp[2]=dp[1]+(time[1]+s+t2)*f2; time[2]=time[1]+s+t2
dp[2]=(s+t1+t2)*(f1+f2); time[2]=ts+t1+t2

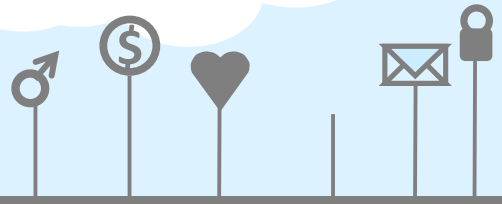
前3个任务的最小费用

dp[3]=dp[2]+(time[2]+s+t3)*f3; time[3]=time[2]+s+t3
dp[3]=dp[1]+(time[1]+s+t2+t3)*(f2+f3) time[3]=time[1]+s+t2+t3
dp[3]=(s+t1+t2+t3)*(f1+f2+f3) time[3]=s+t1+t2+t3

前4个任务的最小费用

dp[4]=dp[3]+(time[3]+s+t4)*f4; time[4]=time[3]+s+t4
dp[4]=dp[2]+(time[2]+s+t3+t4)*(f3+f4) time[4]=time[2]+s+t3+t4
dp[4]=dp[1]+(time[1]+s+t2+t3+t4)*(f2+f3+f4) time[4]=time[1]+s+t2+t3+t4
dp[4]=(s+t1+t2+t3+t4)*(f1+f2+f3+f4) time[4]=s+t1+t2+t3+t4

例3：任务安排 问题分析2



采用类似1006”护卫队”的分析方法：讨论如何分组？

前i个任务的最小费用

dp[i]= **min{** dp[j]+(time[j]+s+t(j+1,...,i))*f(j+1,...,i) **}** 0<=j<i

```
3
1
2 3
3 5
1 7
```

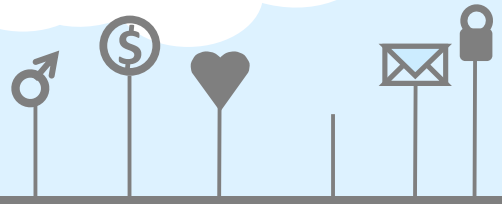
```
dp[1]=9
3*3=9                                   time[1]=3    (1)

dp[2]=44
dp[1]+(time[1]+1+3)*5=44                time[2]=7    (1)(2)
(1+2+3)*(3+5)=48                        time[2]=6    (1,2)

dp[3]=105
dp[2]+(time[2]+1+1)*7=107               time[3]=9    ((1)(2))(3)
dp[1]+(time[1]+1+3+1)*(5+7)=105         time[3]=8    (1)(2,3)
(1+2+3+1)*(3+5+7)=105                   time[3]=7    (1,2,3)
```

```
(1,2)(3)        48+(6+1+1)*7=104     dp[3]=104
```

例3：任务安排 问题分析3



此题属于动规中**费用提前**计算一类

dp[i]表示(完成工作1到i的费用)+(因增加了s导致的后面的工作增加的费用)的总和的最小值

t[i]表示工作1到i的时间的累加(前缀和)；
f[i]表示工作1到i的费用系数的累加(前缀和)。

dp[i]=min{ dp[j]+t[i]*(f[i]-f[j])+s*(f[n]-f[j]) }

j用来分组，表示把工作j+1到工作i分到同一组中。
t[i]*(f[i]-f[j]) 表示完成j+1到i这一组工作的费用
s*(f[n]-f[j])表示因为增加了一次开机，导致从工作j+1到工作n增加的费用。

练习题：**NKOJ2521，1984**

有挑战的思维训练：**NKOJ 3067**

下面是3067的题解，慎入！

思维训练：小车的颜色nkoj3067

题目大意：

输入两个长度分别为 n 和 m（$n,m \leqslant 5000$）的颜色序列，要求按顺序合并成同一个序列，即每次可以把一个序列开头的颜色放到新序列的尾部。

例如，两个颜色序列 GBBY 和 YRRGB，至少有两种合并结果：GBYBRYRGB 和 YRRGGBBYB。对于每个颜色 c 来说，其跨度 $L(c)$等于最大位置和最小位置之差。例如，对于上面两种合并结果，每个颜色的 $L(c)$和所有 $L(c)$的总和如图 9-8 所示。

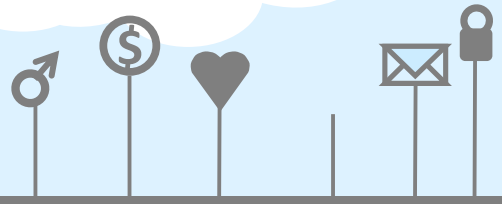
Color	G	Y	B	R	Sum
$L(c)$: Scenario 1	7	3	7	2	19
$L(c)$: Scenario 2	1	7	3	1	12

图 9-8　每个颜色的 $L(c)$和 $L(c)$的总和

你的任务是找一种合并方式，使得所有 $L(c)$的总和最小。

最长公共子序列(LCS)？

思维训练：小车的颜色nkoj3067



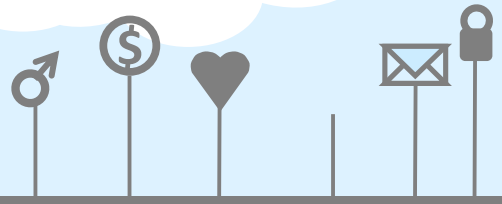
序列1：GBBY
序列2：YRRGB

假设序列1和序列2分别移走了2个和3个元素，构成了新序列: YRGRGB

这时的情况是，G、Y、B在新序列中出现了，但还有没有结束。
R已经出现，并且也结束了，它的跨度值是2。

现在如果从序列1中移走一个元素(即B),B是一个开始了但还没结束的字母，它会使得前面所有开始了但没有结束的字母的跨度值都增加1,也就是B、Y和G的跨度值都需要+1。

思维训练：小车的颜色nkoj3067



最长公共子序列，状态转移方程：

dp[i][j]表示 <x1,x2,...,xi>和<y1,y2,...,yj>的最长公共子串的长度值

即：X的前i个和Y的前j个元素能够构成的最长子序列长度

$$dp[i][j]=\begin{cases} dp[i-1][j-1]+1 & (x[i]==y[j]) \\ \max(dp[i-1][j],dp[i][j-1]) & (x[i]!=y[j]) \\ 0 & (x[i]==0 \text{ 或 } y[j]==0) \end{cases}$$

若此题采取类似LCS的方式定义状态：

dp[i][j]表示第一个序列的前i个元素和第二个序列的前j个元素，构成的新序列的最小权值

dp[i][j] = ?

当某个字母第一次出现在新串时，不知道它什么时候会结束，

当某个字母最后一次出现在新串时，又忘了它第一次的出现位置。

当某个字母既不是该字母的第一个也不是最后一个出现在新串中，**它会使得前面还没有结束的所有的字母跨度都+1。**

动规过程是动态的，方案随时都在变。

思维训练：小车的颜色nkoj3067

费用提前计算的解法：

dp[i][j]第1个序列的前i个元素和第2个序列的前j个元素，构成了新序列后，**还需要的最少代价。**

dp[i][j]=min(dp[i-1][j], dp[i][j-1]) + cnt[i][j]

cnt[i][j]表示当前还有多少个颜色是已经在新序列中出现过但尚未结束的状态，因为每添加一个元素进新序列， 这cnt[i][j]个颜色的跨度都会+1。(也就是对未来状态的影响)

现在的问题是，怎样计算cnt[i][j]?

思维训练：小车的颜色 参考代码

cnt[i][j]表示当前已经在新序列中出现过但尚未结束的颜色的个数。

```
//伪代码.....
  for(int i=0;i<len1;i++)
   {
        if(first[strA[i]-'A'][1]==inf) first[strA[i]-'A'][1]=i;   //first[x][1]字符x在序列1中第一次出现的位置
        last[strA[i]-'A'][1]=i;                                  //last[x][1]字符x在序列1中最后出现的位置
   }
   for(int i=0;i<len2;i++)
   {

        if(first[strB[i]-'A'][2]==inf) first[strB[i]-'A'][2]=i;   //first[x][2]字符x在序列2中第一次出现的位置
        last[strB[i]-'A'][2]=i;                                  //last[x][2]字符x在序列2中最后出现的位置
   }
   for(int i=0;i<=len1;i++)
      for(int j=0;j<=len2;j++)
      {
          int tot=0;
          for(int k=0;k<26;k++)   //讨论k是否是已经出现了但还没有结束的颜色
              if( (i>=first[k][1] || j>=first[k][2])&&(i<last[k][1] || j<last[k][2]) )tot++;
          cnt[i][j]=tot;
      }
```