动态规划问题

WNJXYK

Jilin University

2017 年 2 月

GG. 两个晚上赶出来的 PDF，大家凑合着看。

目录

动态规划介绍

动态规划，Dynamic Programming(DP)

用于解决一类能将原问题分解为若干个子问题求解的复杂问题。关键在于确定状态的表示和状态转移的方法。

三个性质：

1. 最优子结构性质：如果问题的最优解所包含的子问题的解也是最优的，我们就称该问题具有最优子结构性质。

2. 子问题重叠性质：子问题重叠性质是指在用递归算法自顶向下对问题进行求解时，每次产生的子问题并不总是新问题，有些子问题会被重复计算多次。

3. 无后效性：将各阶段按照一定的次序排列好之后，对于某个给定的阶段状态，它以前各阶段的状态无法直接影响它未来的决策，而只能通过当前的这个状态。

动规和分治

动态规划法与分治法类似，它们都是将问题实例归纳为更小的、相似的子问题，并通过求解子问题产生一个全局最优解。

其中分治法中的各个子问题是独立的（即不包含公共的子问题），因此一旦递归地求出各子问题的解后，便可自下而上地将子问题的解合并成问题的解。

动态规划允许这些子问题不独立，（亦即各子问题可包含公共的子子问题）也允许其通过自身子问题的解作出选择，该方法对每一个子问题只解一次，并将结果保存起来，避免每次碰到时都要重复计算。

朴素 DP：递推

这一类动态规划基本都比较简单（Really？复合上数学?）

解决这一类问题，我们只需要分清楚情况，搞清楚递推关系即可。

HDU 2084

从顶层到底层求一条路径和最大的路径，输出路径和。

$DP[i][j]$ 通过位置来表示状态，从下往上递推，

$DP[i][j] += max(DP[i+1][j], DP[i+1][j+1])$

HDU 2044

求 Fibonacci 数列

$F[i] = F[i-1] + F[i-2]$

这样简单的递推 DP 有一个专题。HDU：Source：递推求解专题练习（For Beginner）

朴素 DP：背包

背包九讲的链接:http://love-oriented.com/pack/
01 背包，多重背包，完全背包
分组背包，依赖背包，泛化物品背包
背包的优化：bitset，二进制优化，折半搜索

01 背包

一个容量为 S 的背包，给你 n 件物品，每件物品有价值 v_i，w_i。每件物品只能使用一次。

```
for (int i=1;i<=n;i++){
  for (int j=S;j>=v[i];j--){
    dp[j]=max(dp[j],dp[j-v[i]]+w[i]);
  }
}
```

完全背包

一个容量为 S 的背包，给你 n 件物品，每件物品有价值 v_i，w_i。每件物品能使用无限次。

```
1  for (int i=1;i<=n;i++){
2    for (int j=v[i];j<=S;j++){
3      dp[j]=max(dp[j],dp[j-v[i]]+w[i]);
4    }
5  }
```

多重背包

一个容量为 S 的背包，给你 n 件物品，每件物品有价值 v_i，w_i。每件物品只能使用 c_i 次。

```
1  for (int i=1;i<=n;i++){
2    for (int k=1;k<=c[i];k++){
3      for (int j=S;j>=v[i];j--){
4        dp[j]=max(dp[j],dp[j-v[i]]+w[i]);
5      }
6    }
7  }
```

分组背包

一个容量为 S 的背包，给你 K 组物品，每组有物品 cnt_k，每组中的物品有价值 $v_{k,i}$，$w_{k,i}$。每件物品只能使用一次，且同组的物品只能选择一个。

```
for (int i=1;i<=K;i++){
  for (int j=S;j>=0;j--){
    for (int k=1;k<=cnt[K];k++)
      if (j+v[i][k]<=S)
        dp[i][j+v[i][k]]=
          max(dp[i-1][j+v[i][k]] , dp[i-1][j]+w[i][k])
              ;
  }
}
```

依赖背包 I

一个容量为 S 的背包，给你 n 件物品，每件物品有价值 v_i，w_i。每件物品只能使用一次，且物品之间有单层依赖关系。

其实很简单，我们把依赖关系的物品进行组合变为一个物品组，然后做分组 DP 即可。

可以见 NOIP2016-金明的预算方案（可以去 Codevs 或者 RQNOJ 找到题目并提交）

题目描述

金明今天很开心，家里购置的新房就要领钥匙了，新房里有一间金明自己专用的很宽敞的房间。更让他高兴的是，妈妈昨天对他说：“你的房间需要购买哪些物品，怎么布置，你说了算，只要不超过N元钱就行”。今天一早，金明就开始做预算了，他把想买的物品分为两类：主件与附件，附件是从属于某个主件的，下表就是一些主件与附件的例子：

主件 附件

电脑 打印机，扫描仪

书柜 图书

书桌 台灯，文具

工作椅 无

如果要买归类为附件的物品，必须先买该附件所属的主件。每个主件可以有0个、1个或2个附件。附件不再有从属于自己的附件。金明想买的东西很多，肯定会超过妈妈限定的N元。于是，他把每件物品规定了一个重要度，分为5等：用整数1~5表示，第5等最重要。他还从因特网上查到了每件物品的价格（都是10元的整数倍）。他希望在不超过N元（可以等于N元）的前提下，使每件物品的价格与重要度的乘积的总和最大。

设第j件物品的价格为v[j]，重要度为w[j]，共选中了k件物品，编号依次为j1，j2，……，jk，则所求的总和为：v[j1]*w[j1]+v[j2]*w[j2]+ … +v[jk]*w[jk]。（其中*为乘号）请你帮助金明设计一个满足要求的购物单。

依赖背包 II

```
/*定义*/
const int Maxn=50;
struct Good{
  int price,importance;
  int belong;
  void input(){
    scanf("%d%d%d",&price,&importance,&belong);
  }
  Good(){}
  Good(int p0,int i0){
    price=p0;
    importance=i0;
  }
};
int n,m;
Good igd[Maxn+5],gd[10],stk[3+5];
int top,tot;
int dp[32000+5];

```

依赖背包 III

```
20 int main(){
21   /*输入*/
22   scanf("%d%d",&n,&m);
23   for (int i=1;i<=m;i++) igd[i].input();
24   for (int i=1;i<=m;i++){
25     if (igd[i].belong!=0) continue;
26     /*通过依赖关系合成物品组*/
27     top=0;tot=0;
28     for (int j=1;j<=m;j++) if (igd[j].belong==i) stk[++
           top]=igd[j];
29     for (int now=0;now<(1<<top);now++){
30       int val=igd[i].price*igd[i].importance,pri=igd[i].
             price;
31       for (int j=1;j<=top;j++)
32         if ((now>>(j-1))&1) {
33           val+=stk[j].importance*stk[j].price;
34           pri+=stk[j].price;
35         }
36       gd[++tot]=Good(pri,val);
37     }
```

依赖背包 IV

```
38      /*分组DP*/
39      for (int V=n;V>=0;V--)
40        for (int j=1;j<=tot;j++)
41          if (V+gd[j].price<=n)
42            dp[V+gd[j].price]=max(dp[V+gd[j].price],dp[V]+
                  gd[j].importance);
43
44    }
45    /*输出*/
46    printf("%d\n",dp[n]);
47    return 0;
48  }
```

泛形物品

同样是背包，但是这次背包里的物品的价值会根据我们分配给他的空间而变化。这样的背包，我们考虑不停的把两个泛型物品合并称为一个物品即可。

假设，两个泛型物品根据分配空间所呈现的价值的函数分别为 $h(v)$,$l(v)$，我们可以将其合并为 $k(v)$

$$k(v) = max\{h(i) + l(v - i)\}, i \in [0, v]$$

泛形物品

同样是背包，但是这次背包里的物品的价值会根据我们分配给他的空间而变化。这样的背包，我们考虑不停的把两个泛型物品合并称为一个物品即可。

假设，两个泛型物品根据分配空间所呈现的价值的函数分别为 $h(v)$,$l(v)$，我们可以将其合并为 $k(v)$

$$k(v) = max\{h(i) + l(v - i)\}, i \in [0, v]$$

并没有代码

朴素 DP:LIS

最长不下降子序列：给你一个序列，让你求这个序列的一个子序列，保证子序列中的元素不下降。

状态表示：$DP[i]$ 表示一定选择第 i 个位置上的数字，从第一个位置到第 i 个位置所能组成的最长不下降子序列长度

转移方程：

$$DP[i] = max\{DP[j] + 1\}, num[j] \leq num[i] \ \&\& \ j < i$$

例子：13 7 9 16 38 24 37 18 44 19 21

答案：7 9 16 18 19 21

我们来画个图标演示一下 DP 过程

Pos	1	2	3	4	5	6	7	8	9	10	11
Num	13	7	9	16	38	24	37	18	44	19	21
DP	1	1	2	3	4	4	5	4	5	5	6

朴素 DP：LCS

最长公共子序列：给你两个字符串 A 和 B，要你求出一个最长的子序列，使得它既是 A 的子序列，也是 B 的子序列

状态表示:$DP[i][j]$ 表示从 A 字符串从 1-i 的子串与 B 字符串从 1-j 的子串的最长公共子序列的长度。

状态转移方程:

$$c[i,j] = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0\,, \\ c[i-1, j-1] + 1 & \text{if } i, j > 0 \text{ and } x_i = y_j\,, \\ \max(c[i, j-1], c[i-1, j]) & \text{if } i, j > 0 \text{ and } x_i \neq y_j\,. \end{cases}$$

图示

回溯输出最长公共子序列过程：

	j	0	1	2	3	4	5	6
i		y_j	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	↑ 0	↑ 0	↑ 0	↖ 1	← 1	↖ 1
2	B	0	↖ 1	← 1	← 1	↑ 1	↖ 2	← 2
3	C	0	↑ 1	↑ 1	↖ 2	← 2	↑ 2	↑ 2
4	B	0	↖ 1	↑ 1	↑ 2	↑ 2	↖ 3	← 3
5	D	0	↑ 1	↖ 2	↑ 2	↑ 2	↑ 3	↑ 3
6	A	0	↑ 1	↑ 2	↑ 2	↖ 3	↑ 3	↖ 4
7	B	0	↖ 1	↑ 2	↑ 2	↑ 3	↖ 4	↑ 4

区间 DP

最终问题是一个区间，而区间的最优值可以由几个子区间的最优质合并得到。区间可以不断地划分一直到划分为一个单点区间，可以立即获得答案，然后我们就可以通过枚举一个区间如何分解成子区间合并他们的最优值，然后自下而上 DP。

```
1  for(int len=1;len<=n;len++){//枚举区间长度
2    for (left=1;left<=n;left++){//枚举区间的左端点
3      int right=left+len-1;//计算区间右端点
4      if (right>n) break;
5      for (int mid=left;mid<right;mid++){//枚举区间断点
6        dp[left][right]=mergeSection(dp[left][mid],dp[
             mid+1][right]);//区间合并操作
7      }
8    }
9  }
```

石子归并

https://vjudge.net/problem/CSU-1592

现在有 n 堆石子，第 i 堆有 a_i 个石子。现在要把这些石子合并成一堆，每次只能合并相邻两个，每次合并的代价是两堆石子的总石子数。求合并所有石子的最小代价。

```
Sample Input
2
4
1 2 3 4
5
3 5 2 1 4
Sample Output
19
33
```

DP 过程演示

```
1  DP[1,2]=3
2  DP[2,3]=5
3  DP[3,4]=7
4  DP[1,3]=9
5  DP[2,4]=14
6  DP[1,4]=19
7  19
8
9  DP[1,2]=8
10 DP[2,3]=7
11 DP[3,4]=3
12 DP[4,5]=5
13 DP[1,3]=17
14 DP[2,4]=11
15 DP[3,5]=10
16 DP[1,4]=22
17 DP[2,5]=22
18 DP[1,5]=33
19 33
```

树形 DP

如果出现这样的问题：给一棵树，要求以最少的代价（或取得最大收益）完成给定的操作。考虑使用 DP

1. 确立状态: 几乎所以的问题都要保存以某结点为根的子树的情况，但是要根据具体问题考虑是否要加维，加几维，如何加维。

2. 状态转移: 状态转移的变化比较多，要根据具体问题具体分析。

3. 算法实现: 由于树的结构，使用记忆化搜索比较容易实现。

由于模型建立在树上，即为树型动态规划，顾名思义，树型动态规划就是在“树”的数据结构上的动态规划。

树型动态规划是建立在树上的，一般有两个方向:

1. 根—> 叶: 既根传递有用的信息给子节点，完后根得出最优解的过程。

2. 叶-> 根: 既根的子节点传递有用的信息给根，完后根得出最优解的过程。

http://acm.hdu.edu.cn/showproblem.php?pid=2196

给你一个带权树形图，问你每个节点到最远的节点的距离是多少。

题目给的样例不太好，随手画一个。

树形 DP 代码比较玄学灵活，没有套路或一般代码

数位 DP

有这样一类问题：求给定区间中，满足给定条件的某个 D 进制数或此类数的数量。

所求的限定条件往往与数位有关，例如数位之和、指定数码个数、数的大小顺序分组等等。题目给定的区间往往很大，无法采用朴素的方法求解。此时，我们就需要利用数位的性质，设计 log(n) 级别复杂度的算法。解决这类问题最基本的思想就是“逐位确定”的方法。

1. 状态确定：我们需要根据题目的要求合理地确定能够完整表示数位信息的状态 2. 状态转移：枚举每一位，用记忆化搜索来实现 DP

数位 DP 套路代码 I

```
1  const int Maxn=1e5;
2
3  int dp[Maxn+5][/*合理表示状态*/];
4  int dig[Maxn+5],digs;
5
6  /*初始化*/
7  int init(){
8    memset(dp,-1,sizeof(dp));
9  }
10
11 /*分割数字*/
12 inline void split(int x){
13   digs=0;
14   while(x){
15     dig[++digs]=x%10;
16     x/=10;
17   }
18 }
19
```

数位 DP 套路代码 II

```
20
21
22 /*数位DP*/
23 int dfs(int now,/*合理表示状态*/,bool isLim){
24   if (now==0) {if (/*Conditions*/) return 1; else return
         0;}
25   if (!isLim && dp[now][/*合理表示状态*/]!=-1) return dp
       [now][/*合理表示状态*/];
26
27   int lim=(isLim?dig[now]:9),ret=0;
28   for (int i=0;i<=lim;i++){
29     //if (i==4) continue;
30     ret+=dfs(now-1,/*合理表示状态*/,isLim&&i==lim);
31   }
32
33   if (!isLim) dp[now][/*合理表示状态*/]=ret;
34   return ret;
35 }
36
37
```

数位 DP 套路代码 III

```
38 /*解决问题，获得0～x符合要求数量*/
39 inline int solve(int x){
40     split(x);
41     return dfs(digs,/*合理表示状态*/,true);//第一个参数传
           入当前尝试哪一位，最后一个参数表示，前面位是否与上
           限x相同
42 }
```

概率 DP

概率 DP 主要用于求解期望、概率等题目, 转移方程比较灵活。

一般求概率是正推，求期望是逆推。

这是非常数学相关的东西！所以，我们找点例题吧。GG。

Aeroplane chess

http://acm.hdu.edu.cn/showproblem.php?pid=4405

数轴上有 N+1 个点（编号 0 N），一个人玩游戏，从 0 出发，当到达 N 或大于 N 的点则游戏结束。每次行动掷骰子一次，骰子编号 1-6，掷到多少就向前走几步，这个数轴上还有些特殊点，这些点类似飞行棋中的飞行点，只要到达这些点就可以直接飞到给定点。求总共投掷骰子次数的期望。

初始状态:$dp[x] = 0$，要求的答案：$dp[1]$

状态转移:

1. 如果没有飞行通道:$dp[v - i] += (dp[v] + 1)/6.0, i \in [1, 6]$
2. 如果有飞行通道:$dp[v] = dp[x], v \to x$

Collecting Bugs I

https://vjudge.net/problem/POJ-2096

一个软件有 s 个子系统，会产生 n 种 Bug。某人一天发现一个 Bug，这个 Bug 属于一个子系统，属于一个分类。每个 Bug 属于某个子系统的概率是 $\frac{1}{s}$, 属于某种分类的概率是 $\frac{1}{n}$。问发现 n 种 Bug, 每个子系统都发现 Bug 的天数的期望。

$dp[i][j]$ 表示已经找到 i 种 Bug，j 个系统的 Bug，达到目标状态的天数的期望。

初始状态:$dp[n][s] = 0$，要求的答案是 $dp[0][0]$

$dp[i][j]$ 可以转化成以下四种状态:

1.$dp[i][j]$, 发现一个 bug 属于已经有的 i 个分类和 j 个系统。概率为 $\frac{i}{n} \times \frac{j}{s} = p1$

2.$dp[i][j+1]$, 发现一个 bug 属于已有的分类，不属于已有的系统. 概率为 $\frac{i}{n} \times (1.0 - \frac{j}{s}) = p2$

3.$dp[i+1][j]$, 发现一个 bug 属于已有的系统，不属于已有的分类, 概率为 $(1.0 - \frac{i}{n}) \times \frac{j}{s} = p3$

Collecting Bugs II

4.$dp[i+1][j+1]$, 发现一个 bug 不属于已有的系统，不属于已有的分类, 概率为 $(1.0-\frac{i}{n})\times(1.0-\frac{j}{s})=p4$

可以列出概率的等式，然后解出 $dp[i][j]$。

$$\begin{aligned} dp[i][j] = & \ p1\times dp[i][j] \\ & + p2\times dp[i][j+1] \\ & + p3\times dp[i+1][j] \\ & + p4\times dp[i+1][j+1] \\ & + 1 \end{aligned} \tag{1}$$

状态压缩 DP

一些题目，它们具有 DP 问题的特性，但是状态中所包含的信息过多，如果要用数组来保存状态的话需要四维以上的数组。于是，我们就需要通过状态压缩来保存状态，而使用状态压缩来保存状态的 DP 就叫做状态压缩 DP。

我们可以把我们需要的若干信息压入一个 int 内部，通常的状态压缩 DP 是将若干了 01 状态压缩

状态压缩 DP 的特点：状态中的某一维会比较小，一般不会超过 15，多了的话状态数会急剧上升而无法压缩，一般来说需要状态压缩的也就是这一维。

位运算的技巧

$lowbit(x) = x\&(-x)$
$(x >> j)\&1$ 取出 x 右起第 $j+1$ 位
$x\&((1 << n) - 1)$ 将 x 右起 n 位之外都拿掉

Description

一只队伍在爬山时碰到了雪崩,他们在逃跑时遇到了一座桥,他们要尽快的过桥. 桥已经很旧了, 所以它不能承受太重的东西. 任何时候队伍在桥上的人都不能超过一定的限制. 所以这只队伍过桥时只能分批过,当一组全部过去时,下一组才能接着过. 队伍里每个人过桥都需要特定的时间,当一批队员过桥时时间应该算走得最慢的那一个,每个人也有特定的重量,我们想知道如何分批过桥能使总时间最少.

Input

第一行两个数: w — 桥能承受的最大重量(100 <= w <= 400) 和 n — 队员总数(1 <= n <= 16). 接下来 n 行每行两个数分别表示: t — 该队员过桥所需时间(1 <= t <= 50) 和 w — 该队员的重量(10 <= w <= 100).

Output

输出一个数表示最少的过桥时间.

Sample Input

100 3
24 60
10 40
18 50

Sample Output

42

状压表示:$F[ST]$ 表示 ST 所表示的人全都过桥的最小时间

我们先预处理出状态为 ST 的人一起过桥的时间和总重量，然后通过一个状态压缩的背包解决问题。

```
1  const int Maxn=20;
2  const int MaxSiz=65536;
3
4  int bin[Maxn],wei[Maxn],tim[Maxn];
5  int f[MaxSiz],teat[MaxSiz],teaw[MaxSiz];
6  int n,W;
7
8  int main(){
9
10     bin[0]=1;
11     for (int i=1;i<Maxn;i++) bin[i]=bin[i-1]<<1;
12
13     scanf("%d%d",&W,&n);
14     for (int i=1;i<=n;i++)
15         scanf("%d%d",&tim[i],&wei[i]);
16
```

```
17      /*预处理*/
18      for (int i=1;i<bin[n];i++)
19          for (int j=1;j<=n;j++){
20              if (i&bin[j-1]){
21                  teat[i]=max(tim[j],teat[i]);
22                  teaw[i]+=wei[j];
23              }
24          }
25      /*背包*/
26      for (int i=1;i<bin[n];i++){
27          f[i]=2147483647;
28          for (int j=i;j;j=i&(j-1))
29              if (teaw[j]<=W) f[i]=min(f[i],f[i^j]+teat[j
                    ]);
30      }
31      return printf("%d\n",f[bin[n]-1])*0;
32  }
```

基于连通性的 DP

GG. 我至今只会最基础的. 所以我只讲一个例题，大家自己体会体会。

在状态中要记录若干个格子的联通情况的 DP，我们成为基于连通性的 DP，一般使用状态压缩动态规划实现。

这类的题目一般要我们求一个图的环路的数量或者特定的路径的数量，更难一点的题目会让先我们把问题转化为求路径数量的问题，然后做这种 DP，数据范围一般不大!

Eat the Trees

http://acm.hdu.edu.cn/showproblem.php?pid=1693

问你一张平面图，让你用若干个回路占满非障碍格子，输出方案数。

按每个格子主动转移！记录轮廓线的联通状态！简述方法，给大家感受一下代码

```
for (int i=0;i<=n;i++){
    for (int j=1;j<=m;j++){
        if (i>0 || j==m){
            for (int nst=0;nst<=mx;nst++){
                if (j==m){
                    if (mp[i+1][1]){
                        if (dp[i][j][nst]){
                            if (nst&1){
                                if (1 < m) dp[i+1][1][(nst & (mx^1)) | (1<<m)] += dp[i][j][nst];
                                dp[i+1][1][(nst | 1) & (mx^(1<<m))] += dp[i][j][nst];
                            }else{
                                if (1 < m) dp[i+1][1][(nst | 1) | (1<<m)] += dp[i][j][nst];
                            }
                        }
                    }else{
                        if (dp[i][j][nst]){
                            if (! (nst&1) ) dp[i+1][1][(nst & (mx^1)) & (mx^(1<<m))] += dp[i][j][nst];
                        }
                    }
                }else{
                    if (mp[i][j+1]){
                        if (dp[i][j][nst]){
                            if ((nst & (1<<j)) && (nst & (1<<m))){
                                dp[i][j+1][(nst & (mx^(1<<m))) & (mx^(1<<j))] += dp[i][j][nst];
                            }else if (nst & (1<<j)){
                                if (j+1 < m) dp[i][j+1][(nst | (1<<m)) & (mx^(1<<j))] += dp[i][j][nst];
                                dp[i][j+1][(nst & (mx^(1<<m))) | (1<<j)] += dp[i][j][nst];
                            }else if(nst & (1<<m)){
                                if (j+1 < m) dp[i][j+1][(nst | (1<<m)) & (mx^(1<<j))] += dp[i][j][nst];
                                dp[i][j+1][(nst & (mx^(1<<m))) | (1<<j)] += dp[i][j][nst];
                            }else{
                                if (j+1 < m) dp[i][j+1][(nst | (1<<m)) | (1<<j)] += dp[i][j][nst];
                            }
                        }
                    }else{
                        if (dp[i][j][nst]){
                            if ((!(nst&(1<<j)) && (!(nst&(1<<m))))) dp[i][j+1][(nst & (mx^(1<<j))) & (mx^(1<<m))] += dp[i][j][nst

                        }
                    }
                }
            }
        }
    }
}
```

DP 优化 I

DP 优化实在是太多了，所以我就给大家讲一些典型的（和我印象深刻的，这就是顺带提一提了）

- 背包 DP 的二进制优化
- 单调队列优化 LIS
- 单调队列 + 斜率 DP
  http://acm.hdu.edu.cn/showproblem.php?pid=3507
  一篇文章 n 个单词，每个单词的价值为 S_i，如果一行单词从 *left* 到 *right* 的代价为 $(\sum_{i=left}^{right} S_i)^2 + M$，问如何排版使得打印文章代价最小，求代价。
  朴素转移方程：
  $dp[i] = min\{dp[j] + (sum[i] - sum[j])^2 + M\}, 0 < j < i$
  很明显的斜率优化，X 维单调，使用单调队列

DP 优化 II

- 平衡树/CDQ 分治 + 斜率 DP
  初始状态下，预知有 n 台可以盈利的机器在未来的 D 天中可以卖买到，初始你有 C 的金钱
  接来下告诉你，每台机器可以在第 D_i 天买到，需要花费 S_i 的金钱，卖出去可以得到 R_i 的金钱，持有时每日可以盈利 P_i
  机器卖出日和买入日不盈利，前一台机器卖出日和下一台机器买入日可以重合。
  朴素转移方程：$DP[i] = max\left\{DP[j] - S_j + R_j + (D_i - D_j - 1) \times P_j\right\}, DP[j] \geq S_j, i > j$
  斜率优化 DP，但是 X 维不单调使用 CDQ 分治解决。

DP 优化 III

- 线段树查询最值优化 DP
  http://acm.hdu.edu.cn/showproblem.php?pid=5324
  朴素 DP 方程:
  $DP[i] = max\{DP[j] + 1\}, j < i, L_j \geq L_i, R_j \leq R_i$
  受限于题目的形式，这题必须使用 CDQ 分治消除一维无序影响 + 线段树快速查找最值
  当然如果在某些限制条件少的题目里，这样的形式在某些情况下还可以用单调队列优化成 $O(n)$
- 线段树合并优化 DP
  http://codeforces.com/contest/750/problem/E
  有一个长度为 n 的数字串，给你 q 个询问，每个询问 $[l, r]$，问这个区间内的字符串，经过多少次变换可以使其只存在 2017 的子序列，不存在 2016 的子序列。
- DP 转移方法符合卷积过程，使用 FFT+CDQ 分治将 $O(N^2)$DP 优化到 $O(nLogn)$

End

Coda
Thanks
(2017.2.23 1:50)