动态规划优化专题

陈小羽

uestc

术语

阶段

把问题分成几个相互联系的有顺序的几个环节，这些环节即称为阶段。

状态

某一阶段的出发位置称为状态。通常一个阶段包含若干状态。

决策

从某阶段的一个状态演变到下一个阶段某状态的选择。

策略

由开始到终点的全过程中，由每段决策组成的决策序列称为全过程策略，简称策略。

术语

状态转移方程

前一阶段的终点就是后一阶段的起点，前一阶段的决策选择导出了后一阶段的状态，这种关系描述了由 k 阶段到 k+1 阶段状态的演变规律，称为状态转移方程。

目标函数与最优化概念

目标函数是衡量多阶段决策过程优劣的准则。最优化概念是在一定条件下找到一个途径，经过按题目具体性质所确定的运算以后，使全过程的总效益达到最优。

使用动态规划的前提条件

最优化原理

一个最优化策略具有这样的性质，不论过去状态和决策如何，对前面的决策所形成的状态而言，余下的诸决策必须构成最优策略。简而言之，一个最优化策略的子策略总是最优的。

无后效性

当前的决策只取决于当前这个状态本身，而与如何到达这个状态没有任何关系。

对于动态规划的理解

递归思想

将规模为 n 的问题转化为规模为 $n-1$ 的子问题，最后转化为可以直接求解的原子问题。

一般情况下，这样的递归的方式的时间复杂度是指数级别的。但是如果所有不同的子问题的数目是多项式级别的，那么我们就只需要多项式时间就能解决问题。

动态规划三要素

1. 所有子问题组成的集合（表），这个集合的模称为问题的大小（size）
2. 问题解决的依赖关系，可以看成一个图
3. 填充问题表的顺序（实际上是（2）中得到的图的拓扑排序）

tD/eD

tD/eD 记号

如果子问题的数目为 $O(n^t)$, 且每个子问题需要依赖于 $O(n^e)$ 个其他子问题，则称这种问题为 tD/eD 的。
通过这种记号，我们可以将 dp 问题归类。

按照这样的方式，现在直接给出四种经典的动态规划方程。

方程一 $1D/1D$

定义一个函数 $w(i,j)(1 \leq i < j \leq n)$, 已知 $D(0)$, 状态转移方程为

$$E(j) = \min_{0 \leq i < j} \{D(i) + w(i,j)\}, 1 \leq j \leq n$$

其中 $D(i)$ 可以根据 $E(i)$ 在常数时间内算出来。

方程二 $2D/0D$

已知 $D(i,0)$ 和 $D(0,j)(0 \le i,j \le n)$, 状态转移方程为

$$E(i,j) = \{D(i-1,j)+a, D(i,j-1)+b, D(i-1,j-1)+c\}$$

其中 a,b,c 都能在常数时间内算出来。

方程三 $2D/1D$

定义函数 $w(i,j)(1 \le i < j \le n)$, 已知 $d(i,i)(1 \le i \le n)$, 状态转移方程为

$$C(i,j) = w(i,j) + \min_{1<k\le j}\{C(i,k-1)+C(k,j)\}, 1 \le i < j \le n$$

方程四 2D/2D

定义实函数 $w(i,j)(0 \le i < j \le 2n$, 已知 $d(i,0)$ 和 $D(0,j)(0 \le i,j \le n)$，状态转移方程为

$$E(i,j) = \min_{\substack{0 \le i_1 < i \\ 0 \le j_1 < j}} \{D(i_1,j_1) + w(i_1+j_1, i+j)\}, 1 \le i,j \le n$$

D(i,j) 可以根据 E(i,j) 在常数时间内算出来。

这些记号在之后会用到，同时也能方便你们以后的学习。

与动态规划的时间复杂度相关的因素

影响时间复杂度的因素

时间复杂度 = 状态总数 × 每个状态转移的状态数 × 每次状态转移的时间

需要注意的是，这三者并不是独立的，而是互相联系和制约的。在思考问题的时候，我们应该保有一种全局观。

改进状态的表示

一道例题

现有 n 首由 RaucousRockers 演唱组录制的珍贵的歌曲，计划从中选择一些歌曲来发行 m 张唱片，每张唱片至多包含 t 分钟的音乐，唱片中的歌曲不能重叠。按下面的标准进行选择:
(1) 这组唱片中的歌曲必须按照它们创作的顺序排序。
(2) 包含歌曲的总数尽可能多。
输入 n，m，t，和 n 首歌曲的长度，它们按照创作顺序排序，没有一首歌超出一张唱片的长度，而且不可能将所有歌曲的放在唱片中。输出所能包含的最多的歌曲数目。
$(1 \leq n, m, t \leq 20)$

改进状态的表示

设 n 首歌曲按照写作顺序排序后的长度为 $long(1 \cdots n)$

第一种想法

g[i, j, k] 表示前 i 首歌曲，用 j 张唱片另加 k 分钟来录制，最多可以录制的歌曲数目。
这样我们可以得到如下的状态转移方程:

$$\begin{cases} g(i,j,k) = max\{g(i-1,j,k-long(i))+1, g(i-1,j,k)\} \\ long(i) \leq k, 1 \leq i \\ \\ g(i,j,k) = max\{g(i-1,j-1,t-long(i))+1, g(i-1,j,k)\} \\ k < long(i), 1 \leq i \end{cases}$$

根据状态数，可以的出这种想法的时间复杂度为 $O(n \times m \times t)$

改进状态的表示

设 n 首歌曲按照写作顺序排序后的长度为 $long(1 \cdots n)$

第二种想法

$g(i,j) = (a,b)$, 表示在前 i 首歌曲中选取 j 首录制所需的最少唱片为:a 张唱片另加 b 分钟。
这样我们可以得到如下的状态转移方程和边界条件:
$g(i,j) = min\{g(i-1,j), g(i-1,j-1) + long(i)\}$

$$(a,b) + long(i) \rightarrow \begin{cases} long(i) \leq t-b : (a, b+long(i)) \\ long(i) > t-b : (a+1, long(i)) \end{cases}$$

这样题目所求的最大值是 $ans = max\{k | g(n,k) \leq (m-1,t)\}$

根据状态数，可以的出这种想法的时间复杂度为 $O(n^2)$

选择适当的规划方向

一道例题

有价值分别为 $1 \cdots 6$ 的大理石各 $a(1 \cdots 6)$ 块，现要将它们分成两部分，使得两部分价值和相等，问是否可以实现。其中大理石的总数不超过 20000。

令 $S = \sum_i i \times a(i)$，若 S 为奇数，则不可能实现，否则令 $Mid = \frac{S}{2}$，则问题转化为能否从给定的大理石中选取部分大理石，使其价值和为 Mid。

选择适当的规划方向

状态

$m(i,j)$，表示能否从价值为 $1\cdots i$ 的大理石中选出部分大理石，使其价值和为 j，若能，则用 *true* 表示，否则用 *false* 表示。
则状态转移方程为:
$m(i,j) = m(i,j) \vee m(i-1, j-i\times k)(0 \le k \le a(i))$

然后我们会发现，直接按照一般的方法推的话，时间复杂度是 $O(n^2)$ 的，显然不能解决问题。

递推

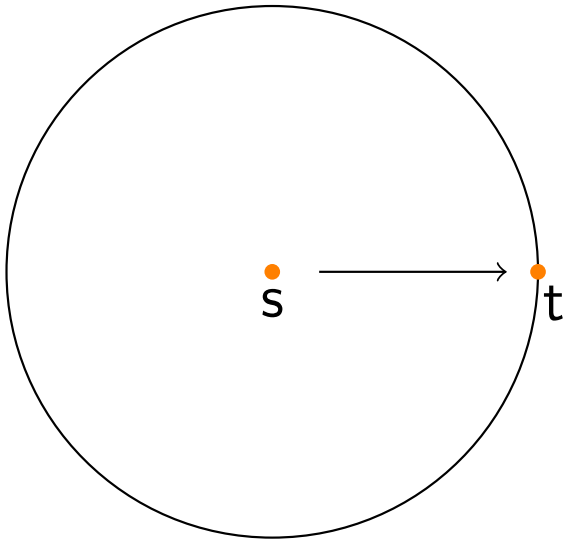
我们可以换一种思路，不去寻找当前状态的子问题。而是用当前状态去更新之后的状态。
我们使用一个队列，将值为 ture 的状态放入其中，执行宽度优先搜索。这样我们能很大程度上减小冗余的计算。时间复杂度和值为 ture 的状态数呈线性关系。

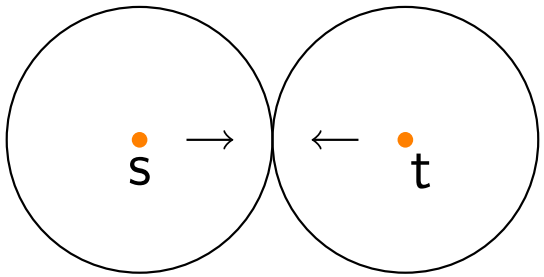
选择适当的规划方向

递推任然不能解决我们的问题。
随着物品的价值品种的增加，我们的状态数将快速增加，导致我们不能承受。要怎么办呢?

双向广搜

当我们知道了广度搜索的起始态和末态之后，其实可以考虑一种双向搜索的思路。
同时从起始态和末态最终在中间态相遇。





选择适当的规划方向

可以看到，双向的思路给我们减少了大量的状态。

在这个题上体现双向

分别求出 $1 \le k \le 3$ $(1 \to 3)$ 和 $4 \le k \le 6$ $(6 \to 4)$ 这两个区段上的 $m(i,j)$ 值。
然后再判断是不是存在 k，使得:
$m(3,k) = true \wedge m(4, Mid - k) = true$。

这样做了之后，时间复杂度发生了什么变化呢?

母函数

其实可以将原问题转化为如下形式:
$(1 + x^1 + x^2 + x^3 + \cdots x^{a(1)}) \times (1 + x^2 + x^4 + \cdots x^{2a(2)}) \times \cdots \times (1 + x^6 + x^12 + \cdots + x^{6a(6)})$
我们只需要从这个式子的结果中找到 $x^{\frac{S}{2}}$ 这一项的系数就可以做出这道题。

选择适当的规划方向

- 很显然，母函数的方法所包含的状态数和动态规划的状态数是相等的。
- 一共有 6 个括号，我们考虑将前三个括号和后 3 个括号分别先乘起来。
- 最后就能形成如下的结果:
  $(a_0+a_1x^1+a_2x^2+\cdots+a_nx^n)\times(b_0+b_1x^1+b_2x^2+\cdots+b_mx^m)$
- 如果直接乘起来的话，负杂度大概是 $O(nm)$, 但是这里我们只用判断乘起来之后，$x^{\frac{S}{2}}$ 的系数是不是 0，所以我们可以直接扫一遍判断，做到 $O(n)$。
- 相当于是把复杂度开了个方。

矩阵加速

使用矩阵的快速幂，可以加速线性递推。

线性递推

$a_n = c_1 a_{n-1} + c_2 a_{n-2} + \cdots + c_k a_{n-k}$ 可得

$$\begin{bmatrix} a_n \\ a_{n-1} \\ a_{n-2} \\ \vdots \\ a_{n-k} \end{bmatrix} = \begin{bmatrix} c_1 & c_2 & c_3 & \cdots & c_n \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 0 \end{bmatrix} \begin{bmatrix} a_{n-1} \\ a_{n-2} \\ a_{n-3} \\ \vdots \\ a_{n-k-1} \end{bmatrix}$$

递推 n 次就是左乘矩阵的 n 次幂。

矩阵加速

快速幂

如何快速计算一个数的 n 次幂?看一段代码。

```
01:int pow(int x,int n){
02:     int ans = 1, tmp = x;
03:     while(n != 0){
04:             if(n & 1){
05:                 ans *= tmp;
06:             }
07:             tmp *= tmp;
08:             n >>= 1;
09:     }
10:     return ans;
11:}
```

我们能够在短时间内算出一个数的 2^n 次幂。
然后我们看这个数的 n 次幂时，将其拆分成
$c_0 2^0 + c_1 2^1 + \cdots c_m 2^m (c_i = \{0, 1\})$

四边形不等式和决策的单调性

石子合并问题

在一个操场上摆放着一排 n($n \leq 20$) 堆石子。现要将石子有次序地合并成一堆。规定每次只能选相邻的 2 堆石子合并成新的一堆，并将新的一堆石子数记为该次合并的得分。试编程求出将 n 堆石子合并成一堆的最小得分以及相应的合并方案。

思路

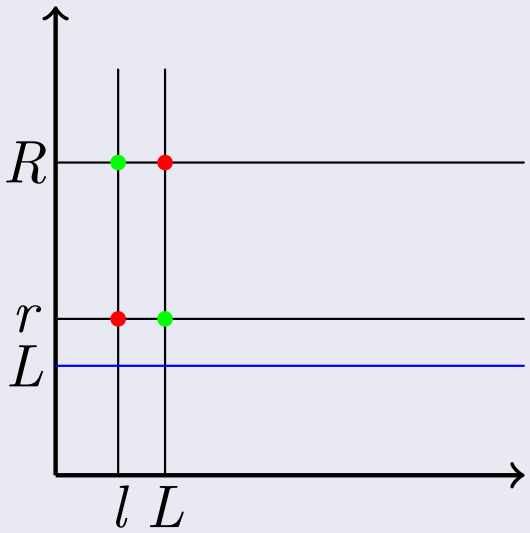
- 设 n 堆石子依次编号为 $1, 2, \cdots, n$。各堆石子数为 $d(1 \cdots n)$
- m(i,j) 表示合并 $d(1 \cdots n)$ 所得到的最小得分。
- $m(i,j) = \min_{i<k\leq j}\{m(i,k-1) + m(k,j) + \sum_{h=i}^{j} d(h)\}$
- 时间复杂度是 $O(n^3)$。

四边形不等式和决策的单调性

四边形不等式

当函数 $w(l,r)$ 满足 $w(l,r)+w(L,R)\leq w(L,r)+w(l,R), l\leq L\leq r\leq R$ 时，称 w 满足四边形不等式 (简称 w 为凸)。

理解记忆



其中，绿点的点权和要大于等于红点的点权和。

四边形不等式和决策的单调性

四边形不等式 *Monge*

当函数 $w(l,r)$ 满足
$w(l,r)+w(L,R)\leq w(L,r)+w(l,R), l\leq L\leq r\leq R$ 时，称 w 满足四边形不等式 (简称 w 为凸)。

inverseMonge

当函数 $w(l,r)$ 满足
$w(l,r)+w(L,R)\geq w(L,r)+w(l,R), l\leq L\leq r\leq R$ 时，称 w 为凹)。
所有的凸的性质都对应了类似的凹的性质，所以之后我们只研究凸的性质。

区间包含关系单调

如果函数 w 满足 $w(l,r)\leq w(L,R)((l,r)\subseteq(L,R))$
我们就说 w 关于区间包含关系单调。

四边形不等式和决策的单调性

- 回到之前的合并石子问题
  $m(i,j) = \min_{i<k\le j}\{m(i,k-1) + m(k,j) + \sum_{h=i}^{j} d(h)\}$
- 我们令 $w(i,j) = \sum_{h=i}^{j} d(h)$ 则原式可以写成
  $m(i,j) = \min_{i<k\le j}\{m(i,k-1) + m(k,j)\} + w(i,j)$
- 下面我们来证明 $m(i,j)$ 也满足四边形不等式。
- 最后形成如下定理

定理一

上式中的 w 如果同时满足四边形不等式和区间包含关系单调，那么 m 满足四边形不等式。

$m(i,j) = \min_{i<k\le j}\{m(i,k-1)+m(k,j)\} + w(i,j)$
$\Rightarrow w(l,r)+w(L,R) \le w(L,r)+w(l,R), l<L<r<R$(情况一)

证明的大概思路

- 显然 $l = R$ 时这个不等式显然是成立的。
- 假设 $R - l \le k - 1$ 的所有情况下，不等式都成立，下面证明 $R - l = k$ 的情况下，不等式也同样成立。
- 设 $y = \min\{p | m(L,r) = m(L,p-1) + m(p,r) + w(L,r)\}$
  $z = \min\{p | m(l,R) = m(l,p-1) + m(p,R) + w(l,R)\}$
- $m(l,r) + m(L,R)$
  $\le m(l,z-1) + m(z,r) + w(l,r) + m(L,y-1) + m(y,R) + w(L,R)$
  对 $w(l,r) + w(L,R)$ 使用四边形不等式的结论
  $\le w(l,R) + w(L,r) + m(L,y-1) + m(l,z-1) + m(z,r) + m(y,R)$
  不妨设 $z \le y$ ($z > y$ 时可同理讨论), 对 $m(z,r) + m(y,R)$ 使用四边形不等式的结论得:
  $\le w(l,R) + w(L,r) + m(L,y-1) + m(l,z-1) + m(z,R) + m(y,r)$
  $\le m(L,r) + m(l,R)$
- 得证。

$m(i,j) = \min_{i<k\le j}\{m(i,k-1)+m(k,j)\} + w(i,j)$
$\Rightarrow w(l,r)+w(L,R) \le w(L,r)+w(l,R), l < L = r < R$(情况二)

证明的大概思路

- 显然 $l = R$ 时这个不等式显然是成立的。
- 假设 $R - l \le k - 1$ 的所有情况下，不等式都成立，下面证明 $R - l = k$ 的情况下，不等式也同样成立。
- 设 $y = \min\{p | m(l,R) = m(l,p-1) + m(p,R) + w(l,R)\}$
- $m(l,r) + m(L,R)$
  $= m(l,r) + m(r,R) \le w(l,r) + m(l,y-1) + m(y,r) + m(r,R)$
  由于 w 区间包含关系单调。
  $\le w(l,R) + m(l,y-1) + m(y,r) + m(r,R)$
  对 $m(y,r) + m(r,R)$ 使用四边形不等式的结论得
  $\le w(l,R) + m(l,y-1) + m(y,R) + m(r,r)$
  $= m(l,R) + m(r,r)$
  $= m(l,R) + m(L,r)$
- 得证。

注意，我在证明的时候省略了一些变元取值范围的讨论，这样其实不太好。

定理一实用化

我们证明了定理一之后是不能直接使用的。怎么办呢?

定理二

令 $s(i,j) = \min\{p|m(i,j) = m(i,p-1) + m(p,j)\} + w(i,j)$。
如果 m 满足四边形不等式，则
$s(i,j) \leq s(i,j+1) \leq s(i+1,j+1)$。

$s(i,j) = \min\{p | m(i,j) = m(i,p-1) + m(p,j)\} + w(i,j)$,$m$ 满足四边形不等式

$\Rightarrow s(i,j) \le s(i,j+1) \le s(i+1,j+1)$

证明

- 令 $m_k(i,j) = m(i,k-1) + m(k,j) + w(i,j)$
- 由于对称性，只用证明 $s(i,j) \le s(i,j+1)$。
- 分析: 要证明 $s(i,j) \le s(i,j+1)$，只要证明对于所有 $(i \le k \le k_1 \le j) \wedge (m_{k_1}(i,j) \le m_k(i,j))$，有:$m_{k_1}(i,j+1) \le m_k(i,j+1)$。
- 这样就证明了，对于 (i,j) 的答案 k_1，$\neg(\exists k)((k \le k_1) \wedge (m_k(i,j+1) \le m_{k_1}(i,j+1)))$
- 事实上，我们可以证明一个更强的不等式 $m_k(i,j) - m_{k_1}(i,j) \le m_k(i,j+1) - m_{k_1}(i,j+1)$
- 移项得 $m_k(i,j) + m_{k_1}(i,j+1) \le m_k(i,j+1) + m_{k_1}(i,j)$
- 两边展开化简得 $m(k,j) + m(k_1,j+1) \le m(k_1,j) + m(k,j+1)$，这正是 $k \le k_1 \le j < j+1$ 时的四边形不等式。
- 得证

今昔对比

初始方程

$$m(i,j) = \min_{i<k\leq j}\{m(i,k-1)+m(k,j)\}+w(i,j)$$

时间复杂度 $O(n^3)$

优化后的方程

$$m(i,j) = \min_{s(i,j-1)\leq k\leq s(i+1,j)}\{m(i,k-1)+m(k,j)\}+w(i,j)$$

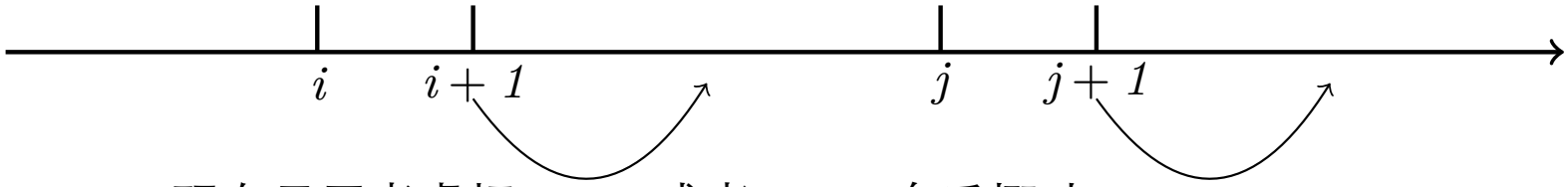
时间复杂度
$O(\sum_{i=1}^{n-1}\sum_{j=i+1}^{n}(1+s(i+1,j)-s(i,j-1)))$
$= O(\sum_{i=1}^{n}(n-i+s(i+1,n)-s(1,n-i)))$
$= O(n^2)$
实际上还存在 $O(n\log n)$ 的做法，但是超出了我们的讨论范围。

理论和实际应用存在差距

我们不可能每次做题都去证明一下 w 是凸的。怎么办呢?

定理三

w 为凸，当且仅当 $w(i,j)+w(i+1,j+1) \leq w(i+1,j)+w(i,j+1)$



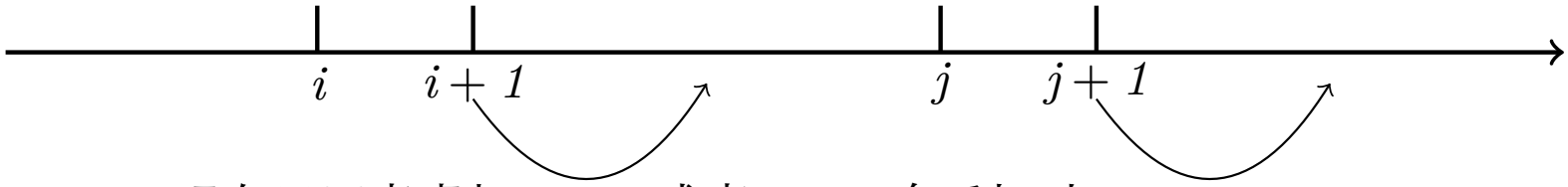
现在只用考虑把 $i+1$ 或者 $j+1$ 向后挪动

理论和实际应用存在差距

我们不可能每次做题都去证明一下 w 是凸的。怎么办呢?

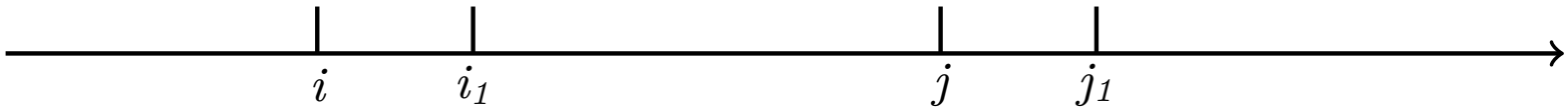
定理三

w 为凸，当且仅当 $w(i,j)+w(i+1,j+1) \leq w(i+1,j)+w(i,j+1)$



现在只用考虑把 $i+1$ 或者 $j+1$ 向后挪动

然后我们就能从这一种情况推出所有我们想要的情况



证明 (必要性 $\Rightarrow$)

定理中的式子只是四边形不等式的一个特例，所以是显然的。

$(\forall i)(\forall j)(w(i,j)+w(i+1,j+1)\leq w(i+1,j)+w(i,j+1))\Rightarrow w$ 为凸 (充分性)

证明

- 由条件, 对任意的一个 i 和 j 有:
  $w(i,j)+w(i+1,j+1)\leq w(i+1,j)+w(i,j+1)$
- 用 $i+1$ 替换 i 得:
  $w(i+1,j)+w(i+2,j+1)\leq w(i+2,j)+w(i+1,j+1)$
- 上面两式相减得: $w(i,j)-w(i+2,j)\leq w(i,j+1)-w(i+2,j+1)$
- 移向得: $w(i,j)+w(i+2,j+1)\leq w(i,j+1)+w(i+2,j)$
- 我们可以通过这种方法不停的将 i_1 和 j_1 向后移动，可以包含所有的情况。
- 也就是说有 w 为凸。

综合一下充分性和必要性，我们就证明了定理三。
使用定理三有时能很快的证明 w 是凸的。然而我们还有更高级的技术。

暴力打表，手动检验小数据

这是网上的博主大佬们的常见做法。大家参考一下就好

推论

任意同样规模的函数 $A_1(i,j), A_2(i,j), \cdots, A_n(i,j)$ 如果都是 Monge array 的话，那么他们的线性组合也是 Monge array。前提是线性组合 $k_1A_1(i,j) + k_2A_2(i,j) + \cdots + k_nA_n(i,j)$ 中的所有 k_i 都非负。
有了定理三之后，这个推论的结论就是显然的。

目前为止能够公开的情报

定理一 $2D/1D$

有 $m(i,j) = \min_{i<k\le j}\{m(i,k-1)+m(k,j)\}+w(i,j)$
上式中的 w 如果同时满足四边形不等式和区间包含关系单调，那么 m 满足四边形不等式。

定理二

令 $s(i,j) = \min\{p|m(i,j) = m(i,p-1)+m(p,j)\}+w(i,j)$。
如果 m 满足四边形不等式，则 $s(i,j) \le s(i,j+1) \le s(i+1,j+1)$。

定理三

w 为凸，当且仅当 $w(i,j)+w(i+1,j+1) \le w(i+1,j)+w(i,j+1)$

推论

任意同样规模的函数 $A_1(i,j), A_2(i,j), \cdots, A_n(i,j)$ 如果都是 Monge array 的话，那么他们的线性组合也是 Monge array。前提是线性组合 $k_1A_1(i,j)+k_2A_2(i,j)+\cdots+k_nA_n(i,j)$ 中的所有 k_i 都非负。

四边形不等式 (Monge) 所带来的性质

一个例子

$$\begin{bmatrix} 10 & 17 & 13 & 28 & 23 \\ 17 & 22 & 16 & 29 & 23 \\ 24 & 28 & 22 & 34 & 24 \\ 11 & 13 & 6 & 17 & 7 \\ 45 & 44 & 32 & 37 & 23 \\ 36 & 33 & 19 & 21 & 6 \\ 75 & 66 & 51 & 53 & 34 \end{bmatrix}$$

凸完全单调性 convex totally monotone

Monge array A 对于所有的 a, b, c, d 都满足

$$\left.\begin{array}{l} a < b \\ c < d \\ A(a, c) \geq A(b, c) \end{array}\right\} \Rightarrow A(a, d) \geq A(b, d)$$

四边形不等式 (Monge) 所带来的性质

$$\begin{bmatrix} 10 & 17 & 13 & 28 & 23 \\ 17 & 22 & 16 & 29 & 23 \\ 24 & 28 & 22 & 34 & 24 \\ 11 & 13 & 6 & 17 & 7 \\ 45 & 44 & 32 & 37 & 23 \\ 36 & 33 & 19 & 21 & 6 \\ 75 & 66 & 51 & 53 & 34 \end{bmatrix}$$

最小值的编号性质

- 令 $a(i,j)$ 为 Monge array
  令函数 $j(i)$ 表示第 i 行满足 $a(i,j(i)) = \max_{1\le j\le n}\{a(i,j)\}$ 的最小的列标。
  则有: $j(1) \le j(2) \le j(3) \le \cdots \le j(n)$
- 同理可得: 令函数 $i(j)$ 表示第 j 列满足 $a(i(j),j) = \max_{1\le i\le n}\{a(i,j)\}$ 的最小的行标。
  则有: $i(1) \le i(2) \le i(3) \le \cdots \le i(n)$

用反证法可以很简单的证明。

两个性质在 $1D/1D$ 上的应用

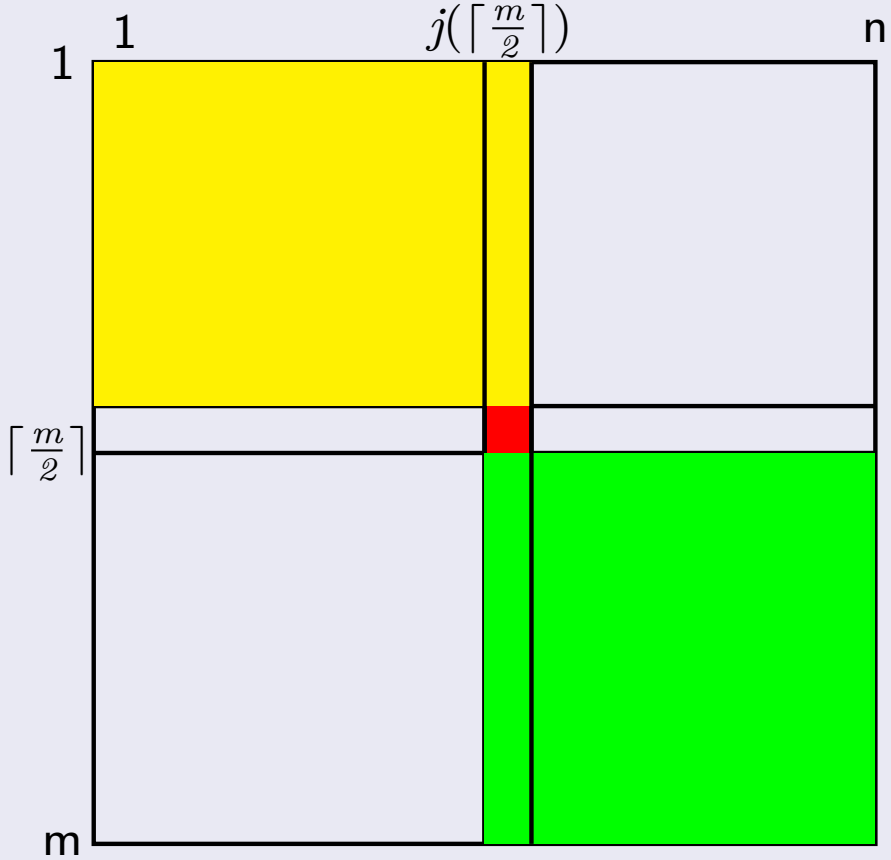
问题

考虑转移方程 $B(j) = \min_{1 \le i < j}\{d(i) + w(i,j)\}$，其中 $w(i,j)$ 是 Monge array, 问有没有优化的方法?

- 考虑将 $B(i)$ 扩展成为 $B(i,j) = d(i) + w(i,j)$ 这样的函数。
- 则我们的任务变成了求 $B(i,j)$ 在每一行上面的最小值。
- 对于这个问题我们有什么办法呢?

两个性质在 $1D/1D$ 上的应用

分治

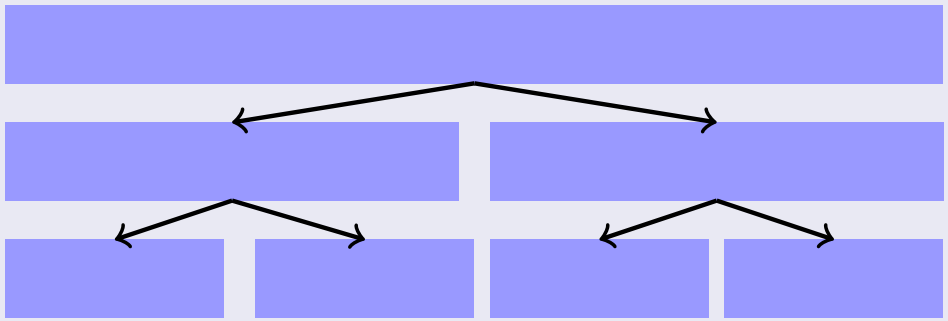


我们对 m 进行分治，对于每一个 m_i（比如 $\frac{m}{2}$）我们花费 $O(n)$ 的时间计算 $j(m_i)$ 的值。然后我们根据最小值的编号的性质（单调性），可以将原问题分解为规模更小的两个子问题。

两个性质在 $1D/1D$ 上的应用

分治法的时间复杂度

进行一个比较宽松的估计。

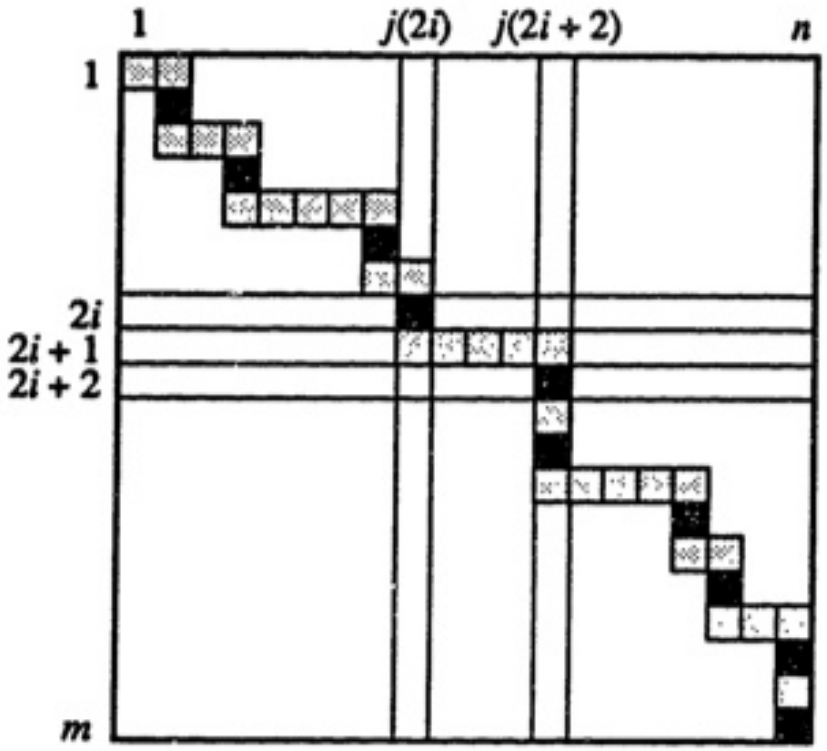


观察可以发现，树的每一层都大致会执行 n 次，总的树的深度不超过 $\log(m)$，所以总的时间复杂度为 $O(n \log m)$

向 $O(n)$ 进发

辅助定理一

给出 w 在所有偶数点的答案，我们可以在 $O(n+m)$ 的时间内算出答案。



向 $O(n)$ 进发

辅助定理二

我们可以在 $O(N)$ 的时间内算出有哪些位置一定不是我们要的答案。

使用栈

- 我们使用一个栈来记录到第 $j-1$ 列为止的所有的可能的答案记为 $s_1, s_2, \cdots s_{top}$。
  并且保证栈内的元素满足 $w(i, s_i) < w(i, s_{i+1})$。
- 假设我们已经处理完了 $j-1$ 列，则对于当前的列 j，我们将 $w(top, j)$ 和 $w(top, s_{top})$ 比较，如果 $w(top, s_{top}) < w(top, j)$ 那我们将 j 入栈，然后进入下一行。
- 否则，如果 $w(top, s_{top}) > w(top, j)$ 则由凸完全单调性得 $w(k, s_{top}) > w(k, j)(top < k \leq m)$
- 然后我们又有，$w(top-1, s_{top-1}) < w(top-1, s_{top})$，由凸完全单调性得 $w(k, s_{top-1}) < w(k, s_{top})(1 \leq k < top)$。
- 综合上面的两点，我们可以确定，s_{top} 一定不会是答案，所以我们将他从栈中 pop 掉。
- 这样的操作可以保证当前行计算完毕之后，依然可以保证栈内的元素满足 $w(i, s_i) < w(i, s_{i+1})$。

向 $O(n)$ 进发

使用栈的时间开销

- 每次每列最多进栈一次 $O(n)$
- 每个栈内元素最多出栈一次 $O(n)$
- 所以总的时间复杂度为 $O(n)$
- 顺便说一下，对于一个 $m \times n$ 的矩阵，一共可以删掉 $n - m$ 列。
- 我们可以在末尾加一行，保证这一行的数随着列数的增加而单调下降，之后我们就能保证每次最多有 m+1 个列在队列中。

向 $O(n)$ 进发

计算答案, 前提为 $m \leq n$

- 不妨设 $m = n$
- 选择所有的偶数行, 构成一个子矩阵 $B_{\lfloor \frac{m}{2} \rfloor \times n}$，显然，这个子矩阵也是 Monge array。
- 对 $B_{\lfloor \frac{n}{2} \rfloor \times n}$ 使用辅助定理二。得到 $B_{\lfloor \frac{n}{2} \rfloor \times \lfloor \frac{n}{2} \rfloor}$。
- 计算 $B_{\lfloor \frac{n}{2} \rfloor \times \lfloor \frac{n}{2} \rfloor}$ 的所有行的答案（子问题）。
- 通过辅助定理一和 B 得到的偶数行的答案计算 $w(i, j)$ 的所有的答案。

分析时间

- 设规模为 n 的问题的时间复杂度为 $T(n)$，根据过程的递归定义我们有
- $$T(n) = \begin{cases} O(1) & n = 1 \\ T(\frac{n}{2}) + O(n) & n \neq 1 \end{cases}$$
- 加 $O(n)$ 是因为每次用辅助定理一和二，还有构造 $B_{\lfloor \frac{n}{2} \rfloor \times n}$ 的时间都是 O(n) 的

向 $O(n)$ 进发

分析时间 (接上页)

- 最后得到的时间复杂度是

$$\sum_{i=0}^{+\infty} O(\frac{n}{2^i}) = O(\sum_{i=0}^{+\infty} \frac{n}{2^i})$$

- $= O(n)$

注意这里有个 $m \leq n$ 的前提条件。
这里讲的只是 Monge array 的一种离线的情况，实际应用之中，主要是用的在线的方法，这里不做过多的讲解。思想都是类似的。

目前为止能够公开的情报

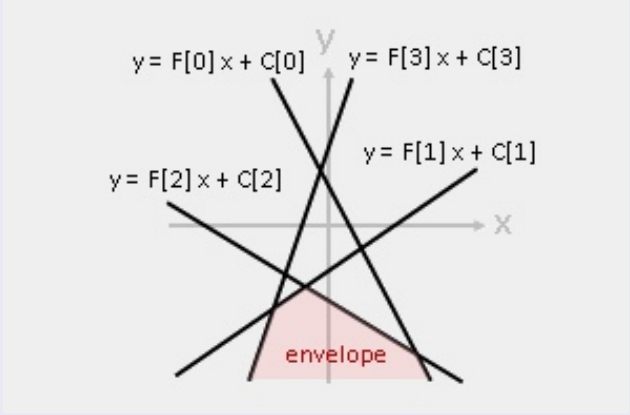
- 前面讲了 $2D/1D$ 和 $1D/1D$ 两种类型的四边形不等式 Monge。
- 由三大定理和一些 Monge 导出的性质分别优化了 $2D/1D$ 和 $1D/1D$ 的时间复杂度。
- 注意 $1D/1D$ 的 $O(n)$ 优化里有个 $m \leq n$ 的前提条件。
- 做不出来？先打个表再说。

斜率

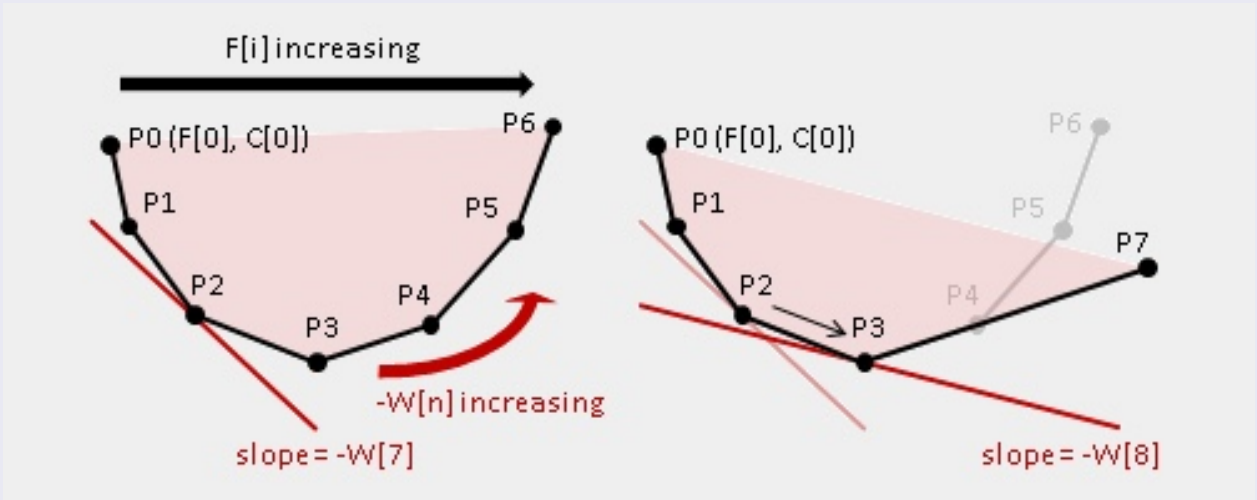
- 有时我们会遇到如下情况的转移式 $(1D/1D)$。
  $f(i) = (min|max)\{f(j) \times w(i) + c(j)\}(1 \leq j < i)$
  不妨我们这里只考虑 min。
- 对于这种转移方程我们可以换一个视角。
- 我们将 $f(j)$ 看作直线的斜率, 将 $c(j)$ 看作直线的纵截距。那么我们就得到了一系列的直线。最后我们再将 w(i) 看作横坐标
- 现在的问题就变成了如何快速求一系列直线在一个给定的 x 的最小值的问题。
- 有没有解决这类问题的方法呢? 答案是肯定的。

斜率

convex hull 凸包（其实就是一个凸多边形）

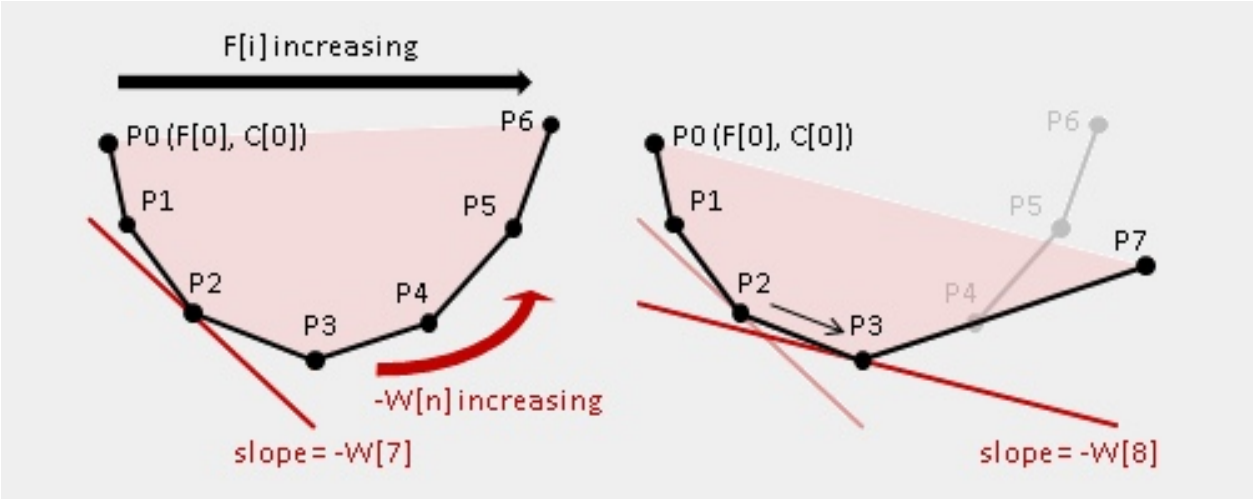


这种叫做包络线



这种叫凸包

斜率



convex hull trick

观察图形可以发现，原来的问题其实是包络线问题。
而且可以转化为凸包问题。
因为凸包和包络线都可以维护图中的红色区域。
所以对于某个 x 的最小值，其实是在凸包边界上 y 的对应取值。
下面的问题就是

- 如何维护凸包 (去掉冗余的直线)
- 如何在凸包上查询某个 x 对应的 y

斜率

Query

假定凸包上的点都是有序的，则我们可以使用二分法，找到第一个不大于 x 的凸包上的点。然后我们把 x 带入对应的直线。

Maintian

- 不妨维护下凸壳（要求最大值）
- 假设直线事先都按照斜率升序排列。
- 考虑第 i 条直线。假设前 $i-1$ 条直线都已经按照顺序排在了栈 S 中。
- 设第 i 条直线于 S_{top-1} 的直线的交点为 A，与 S_{top} 的交点为 B。
- 如果 $A_x > B_x$ 就将 S_{top} pop 掉。
- 重复这个操作，直到不能弹。
- 然后第 i 条直线入栈。
- 可以发现，计算完第 i 条直线之后，S 内的直线斜率依然是有序的。我们的目的达到了。

斜率

例题

给你 n 个矩形。让你把这些矩形分成一些不相交的集合。
每个集合的费用为这个集合中所有矩形的最大的长和最大的宽的乘积。
问最小费用。

事实一

对于一个矩形 B 如果存在矩形 A 并且 $(A_w \geq B_w) \wedge (A_h \geq b_h)$，那么 B 的存在对答案没有影响。
之后的讨论都假设这样的 B 不存在。

事实二

将矩形的宽度 w 作为第一关键字（升序），将矩形的高度 h 作为第二关键字（降序）排序。
那么集合的划分就变成了区间的划分，因为我们会在区间的最左端取到 h 的最大值，在最右端取到 w 的最大值。
所以这个区间中除了端点的其他的矩形都对答案没有任何的贡献。但是把他们放在这个区间里一定比较优。

斜率

- $cost(i) = min(cost(i), cost(j) + rect(i).h * rect(j+1).w)$ $(1 \leq j < i)$ 其中 $cost(i)$ 表示前 i 个矩形的花费的最小值。
- 设 $b_j = cost(j), k_j = rect(j+1).w$，则原问题转化为斜率问题。
- 这个问题有一个很好的性质，就是 k_j 是单调递增的，因为我们之前排了序。
- 应用刚才讲到的 convex hull trick，我们可以将复杂度做到 $(n \log n)$。
- 这个题的 w 是单调递增的。
- 我们将凸包中所有端点小于 $rect(i).h$ 的线段对应的直线从凸包上删除，他们已经用不到了。
- 要做到这一点只需将栈 S 改为双端队列 Deq。
- 改进后时间复杂度为 $O(n)$。

斜率

扩展

如果 $F_i = k_j x + b_j$ 中，k_j 是无序的，怎么办?

使用 set

- 使用两个 set
- Set_1 的每个节点：$\{key = slope, value = \{y_{intercept}, x_{min}\}\}$
- Set_2 的每个节点：$\{key = x_{min}, value = \{slope, y_{intercept}\}\}$
- $y_{intercept}$ 表示纵截距
- x_{min} 表示每个线段上 x 的最小值。
- $slope$ 表示斜率。

斜率

Query

- 在 Set_2 中查询比当前 x 小的最大的 x_j(lower_bound-1)
- 查询到之后把 x 带进去。
- 复杂度 $O(\log n)$

Maintian

- 将待加入的 L 加入 Set_1 和 Set_2。
- 在 Set_1 可以在 $O(log)$ 的时间得到 L 的前驱，L 前驱的前驱。后继同理
- 然后我们发现这些东西的斜率是有序的，所以可以用之前的方法同样的处理。
- 注意在 Set_1 中删掉的直线，可以藉由 x_{min} 的值在 Set_2 上同样的删除。
- 复杂度 $O(\log n)$

总时间

如果查询和修改的次数是 $O(n)$ 的，则总时间是 $O(n \log n)$

减少状态转移花费的时间

概念

这部分主要是在说，当题目中的某个数据，有某种性质的时候，可以用一些数据结构来维护 这部分主要是一些技巧性的东西，不存在什么广泛适用的算法，所以不讲。
具体的东西，大家回去做题的时候就能遇得到了。
和前面比起来这些东西不算很难。

现在能够适用的工具

- 改进状态的表示
- 选好规划的方向
- 矩阵加速
- Monge array
  - 三大定理和推论
  - 定理的两个性质
  - 从性质引出的两个辅助定理
  - $1D/1D$ 分治法
  - 适用辅助定理 $1D/1D \rightarrow O(n)$
- convex hull trick
- 用数据结构加速

End