FZOI

信息学

单调队列优化

「TYVJ1305」最大子段和

输入一个长度为n的整数序列（可能为负数），从中找出一段不超过m 的连续子段，使得整个子段的和最大。

例如: 1, −3,5,1, −2,3
当$m = 4$时，$S = 5 + 1 - 2 + 3 = 7$
当$m = 2$或$m = 3$时，$S = 5 + 1 = 6$

$n \le m \le 300000$

「TYVJ1305」最大子序和

输入一个长度为n的整数序列（可能为负数），从中找出一段**不超过**m的连续子序列，使得整个序列的和最大。

令前缀和为：$S_i = \Sigma_{k=1}^{i} a_k$

则：

$$ans = \max_{1 \le i \le N} \left\{ S_i - \min_{i-m \le j \le i-1} \{S_j\} \right\}$$

「TYVJ1305」最大子序和

$$ans = \max_{1 \le i \le N} \left\{ S_i - \min_{i-m \le j \le i-1} \{S_j\} \right\}$$

随着i的增加，j的范围上界和下界同时增加1：

意味着不仅仅是要将$j = i$加入到候选集合，还需要将$j = i - m$从候选集合中删除掉。

此时，单调队列就非常合适了：

- 上下界均单调变化；
- 每个决策在候选集合中插入或删除至多一次；

「POJ1821」Fence

- 有 N 块木板从左至右排成一行,有 M 个工匠对这些木板进行粉刷,每块木板至多被粉刷一次。第 i 个工匠要么不粉刷,要么粉刷包含木板 S_i 的,长度不超过L_i 的连续一段木板,每粉刷一块木板可以得到 P_i 的报酬。求如何安排能使工匠们获得的总报酬最多。
- $1 \le N \le 16000, 1 \le M \le 100$

「POJ1821」Fence

- 有 N 块木板从左至右排成一行,有 M 个工匠对这些木板进行粉刷,每块木板至多被粉刷一次。第 i 个工匠要么不粉刷,要么粉刷包含木板 S_i 的,长度不超过L_i 的连续一段木板,每粉刷一块木板可以得到 P_i 的报酬。求如何安排能使工匠们获得的总报酬最多。
- $1 \le N \le 16000, 1 \le M \le 100$
- 状态:
- 将所有的工匠按照S_i 排序;
- $F[i,j]$表示安排前i个工匠粉刷前j块木板（允许有空着的木板）,工匠能获得的最大报酬;

「POJ1821」Fence

- 有 N 块木板从左至右排成一行,有 M 个工匠对这些木板进行粉刷，每块木板至多被粉刷一次。第 i 个工匠要么不粉刷,要么粉刷包含 木板 S_i 的,长度不超过L_i 的连续一段木板,每粉刷一块木板可以得 到 P_i 的报酬。求如何安排能使工匠们获得的总报酬最多。
- $1 \le N \le 16000, 1 \le M \le 100$
- 状态转移：
- 决策：第i个工匠应该刷$k+1$到第j块木板，最优的k是多少；

$$F[i,j] = \max_{j-L_i \le k \le S_i-1} \{F[i-1,k] + P_i * (j-k)\} \quad ,j \ge S_i$$

「POJ1821」Fence

- 状态转移：
- 决策：第i个工匠应该刷$k+1$到第j块木板，最优的j是多少；

$$F[i,j]=\max_{j-L_i\le k\le S_i-1}\{F[i-1,k]+P_i*(j-k)\}\quad,j\ge S_i$$

- 直接去做的时间复杂度：
- $O(mn^2)$

「POJ1821」Fence

- 状态转移：
- 决策：第i个工匠应该刷$k+1$到第j块木板，最优的j是多少；

$$F[i,j]=\max_{j-L_i\le k\le S_i-1}\{F[i-1,k]+P_i*(j-k)\}\quad ,j\ge S_i$$

- 优化：
- 当考虑内层循环j时，i可以看做是定值；
- 那么可以将方程修改下变为：

$$F[i,j]=P_i*j+\max_{j-L_i\le k\le S_i-1}\{F[i-1,k]-P_i*k\}\quad ,j\ge S_i$$

「POJ1821」Fence

$$F[i,j] = P_i * j + \max_{j-L_i \le k \le S_i - 1} \{F[i-1,k] - P_i * k\} \quad ,j \ge S_i$$

- 观察max 里面的项目：
- 令$G_i(k) = F\ [i-1, k\ ] - P_i * k$
- 发现随着j增大，k的取值范围上界不变，下界增大。
- 如果存在两个决策：$k_1 < k_2 \le S_i - 1$且$G_i(k_1) < G_i\ (k_2)$；
- 那么意味着k_2不仅仅比k_1优秀还“活”得更长。
- 那么k_1肯定是一个无用的决策，应被舍弃；
- 明显我们可以使用一个队列：
- 决策点k递增，$G_i(k)$递减的单调队列；

「POJ1821」Fence

$$F[i,j] = P_i * j + \max_{j-L_i \le k \le S_i - 1} \{F[i-1,k] - P_i * k\} \quad ,j \ge S_i$$

- 观察max里面的项目：
- 令$G_i(k) = F[i-1,k] - P_i * k$
- 维护：决策点k递增，$G_i(k)$递减的单调队列；
- 具体操作：

1. 当j变大时，检查队头元素，把小于$j - L_i$的决策出队；
2. 当要求最优的时候，队头为所求；
3. 有一个新的决策入队时，队尾查询$G_i(k)$的单调性，将无用决策出队；

「POJ1821」Fence

$$F[i,j] = P_i * j + \max_{j-L_i \le k \le S_i - 1} \{F[i-1,k] - P_i * k\} \quad , j \ge S_i$$

- 具体实现中：
- 当内层循环开始时($j = S_i$)，建立一个空的单调队列，再按照上一页步骤进行实现；
- 时间复杂度分析：
- 由于每个决策最多入队一次、出队一次，因此转移的时间均摊$O(1)$；
- 整个算法时间复杂度为$O(NM)$。

稍做迁移

$$F[i,j] = P_i * j + \min_{j-L_i \le k \le S_i-1} \{F[i-1,k] - P_i * k\} \ , j \ge S_i$$

- 应该如何修改?
- 维护单调队列，单调递增即可;

「POJ3017」Cut the Sequence

- 给定一个长度为N的序列A，要求把该序列分成若干段，在满足"每段中所有数的和"不超过M的前提下，让"每段中所有数的最大值"之和最小。试计算这个最小值
- $N \le 10^5, 0 \le A \le 10^6, M \le 10^{11}$

「POJ3017」Cut the Sequence

- 给定一个长度为N的序列A，要求把该序列分成若干段，在满足“每段中所有数的和”不超过M的前提下，让“每段中所有数的最大 值”之和最小。试计算这个最小值
- $N \le 10^5, 0 \le A \le 10^6, M \le 10^{11}$
- 状态：
- $f[i]$表示把前i个数分成若干段，在满足条件的情况下，各段的最大值之和最小时，最小值是多少
- 决策：
- 最后一段的开始位置

「POJ3017」Cut the Sequence

- 给定一个长度为N的序列A，要求把该序列分成若干段，在满足“每段中所有数的和”不超过M的前提下，让“每段中所有数的最大 值”之和最小。试计算这个最小值
- $N \le 10^5, 0 \le A \le 10^6, M \le 10^{11}$
- 状态：
- $f[i]$表示把前i个数分成若干段，在满足条件的情况下，各段的最大值之和最小时，最小值是多少
- 状态转移方程：

$$f[i] = \min_{0 \le j \le i \&\& \sum_{k=j+1}^{i} A_k < M} \left\{ f[j] + \max_{j+1 \le k \le i} \{A_k\} \right\}$$

「POJ3017」Cut the Sequence

- 状态转移方程：

$$f[i] = \min_{0\le j\le i \&\& \sum_{k=j+1}^{i} A_k < M} \left\{ f[j] + \max_{j+1\le k\le i} \{A_k\} \right\}$$

- 暴力求解：
- $O(n^3)$
- 发现 $\max\limits_{j+1\le k\le i} \{A_k\}$这个的求解可以优化：
- ST表预处理后，可$O(1)$完成询问
- 复杂度变为$O(n^2)$

「POJ3017」Cut the Sequence

- 状态转移方程：

$$f[i] = \min_{0 \le j \le i \&\& \sum_{k=j+1}^{i} A_k < M} \left\{ f[j] + \max_{j+1 \le k \le i} \{A_k\} \right\}$$

- 再优化：
- **中心思想：**
- 及早地将不需要的状态（不可能的决策）排除掉，让候选集合保 持有效性和有序性

- 考虑在什么时候j才是必要的

「POJ3017」Cut the Sequence

- 状态转移方程：

$$f[i]=\min_{0\le j\le i \&\& \sum_{k=j+1}^{i} A_k<M}\left\{f[j]+\max_{j+1\le k\le i}\{A_k\}\right\}$$

- j必须满足以下条件之一：
- $A_j = \max\limits_{j\le k\le i}\{A_k\}$
- $\sum_{k=j}^{i} A_k > M$(即j是满足$\sum_{k=j+1}^{i} A_k \le M$的最小$j$)

「POJ3017」Cut the Sequence

- $A_j = \max\limits_{j \le k \le i} \{A_k\}$
- $\Sigma_{k=j}^{i} A_k > M$(即j是满足$\Sigma_{k=j+1}^{i} A_k \le M$的最小$j$)
- 证明:
- 如果都不满足，那么$\max\limits_{j \le k \le i} \{A_k\} = \max\limits_{j+1 \le k \le i} \{A_k\}$，有因为显然 $f[j-1] \le f[j]$
- 因此： $f[j-1] + \max\limits_{j \le k \le i} \{A_k\} \le f[j] + \max\limits_{j+1 \le k \le i} \{A_k\}$
- 也就是$j-1$比j更优；
- 得证。

「POJ3017」Cut the Sequence

- 状态转移方程：

$$f[i] = \min_{0\le j\le i \,\&\&\, \sum_{k=j+1}^{i} A_k < M} \left\{ f[j] + \max_{j+1\le k\le i} \{A_k\} \right\}$$

- j必须满足以下条件之一：

1. $A_j = \max\limits_{j\le k\le i} \{A_k\}$

2. $\sum_{k=j}^{i} A_k > M$（即是满足$\sum_{k=j+1}^{i} A_k \le M$的最小$j$）

- 怎么运用呢？
- 对于条件2，可以一开始就对每个i预处理出来，存为$c[i]$；
- 后面转移的时候讨论下$c[i]$即可

「POJ3017」Cut the Sequence

- 状态转移方程：

$$f[i] = \min_{0\le j\le i \&\& \sum_{k=j+1}^{i} A_k < M} \left\{ f[j] + \max_{j+1\le k\le i} \{A_k\} \right\}$$

- j必须满足以下条件之一：

1. $A_j = \max_{j\le k\le i} \{A_k\}$

对于条件1，我们会自然而然想到，用一个单调队列进行维护：

j递增A_j递减

那么决策集合都在这个单调队列中

「POJ3017」Cut the Sequence

- 状态转移方程：

$$f[i]=\min_{0\le j\le i \&\& \sum_{k=j+1}^{i} A_k<M}\left\{f[j]+\max_{j+1\le k\le i}\{A_k\}\right\}$$

- 用一个单调队列进行维护：j递增A_j递减
- 那么决策集合都在这个单调队列中
- 但是，队首并不是最优的决策，队列中的元素集合是$\{A_j\}$
- 我们需要找到的是$f[j]+\max_{j+1\le k\le i}\{A_k\}$中最小的；
- 那么我们将在单调队列中的j，形成一个新的$f[j]+\max_{j+1\le k\le i}\{A_k\}$集合，拿一个数据结构维护（堆或者平衡树或者set）
- 该数据结构和单调队列的元素保持一致；

「POJ3017」Cut the Sequence

- 状态转移方程：

$$f[i]=\min_{0\le j\le i\&\&\sum_{k=j+1}^{i}A_k<M}\left\{f[j]+\max_{j+1\le k\le i}\{A_k\}\right\}$$

- 求解：$f[j]+\max\limits_{j+1\le k\le i}\{A_k\}$
- 如果我们要做j的决策，我们如何才能将该值求解出来：
- 即如何找到$\max\limits_{j+1\le k\le i}\{A_k\}$
- 方法一：
- ST表

「POJ3017」Cut the Sequence

- 状态转移方程：

$$f[i]=\min_{0\le j\le i \&\& \sum_{k=j+1}^{i} A_k<M}\left\{f[j]+\max_{j+1\le k\le i}\{A_k\}\right\}$$

- 求解：$f[j]+\max_{j+1\le k\le i}\{A_k\}$
- 如果我们要做j的决策，我们如何才能将该值求解出来：
- 方法二：
- 如果j在单调队列中的位置为x，那么该值为$f[q[x]]+A[q[x+1]]$
- 仔细体会下，会觉得挺显然的。

「POJ3017」Cut the Sequence

- 参考:
- 《蓝书》
- http://blog.leanote.com/post/okami/df2a864dff6d
- PS：网上有些代码是没有加堆的，直接遍历单调队列，
- 严格的来说，不可取；

多重背包

- 现在有一个背包容量为M的背包，有N种物品，分别的数量为c_i，体积为v_i，价值为w_i，将N种物品装入背包，问最大价值是多少

- 传统解法：
- 决策每种物品放多少个：

$$f[j] = \max_{1 \le cnt \le c_i} \{f[j - cnt * v_i] + cnt * w_i\}$$

- j从后往前遍历

多重背包

$$f[j] = \max_{1 \le cnt \le c_i} \{f[j - cnt * v_i] + cnt * w_i\}$$

- 当要转移到j的时候，f[]下标值候选集合是$\{j - cnt * v_i\}$：

		$j-2v_i$			$j-v_i$			j	

- 当要转移到$j-1$的时候，f[]下标值候选集合是$\{j - 1 - cnt * v_i\}$：

	$j-1-2v_i$			$j-1-v_i$			j-1		

- 发现没有重合的

多重背包

- 当要转移到j的时候，$f[]$下标值候选集合是$\{j - cnt * v_i\}$：

		$j-2v_i$			$j-v_i$			j	

- 但是，当要转移到$j-3$时要转移到j的时候

		$j-3-v_i$			$j-3$				

- 发现就用重合的了，
- 因此我们考虑，将状态j按照除以v_i的余数分组，对每一组分别计算，不同组之间的状态相互没有影响

:多重背包

- 发现就用重合的了，因此我们考虑，将状态j按照除以v_i的余数分组，对每一组分别计算，不同组之间的状态相互没有影响：
- 余数为0——$0, v_i, 2v_i, \ldots$
- 余数为1——$1, 1+v_i, 1+2v_i, \ldots$
- ……
- 余数为v_i-1——$(v_i-1), (v_i-1)+v_i, (v_i-1)+2v_i, \ldots$
- 令余数为$u \in [0, v_i-1]$，那么状态$j=u+p*v_i$
- 那么倒序j的过程，就可以变成倒序p的过程

- 令余数为$u \in [0, v_i - 1]$，那么状态$j = u + p * v_i$
- 那么倒序j的过程，就可以变成倒序p的过程

```
for(j = M;j>=v[i];j--)
```

```
for(u=v[i]-1;u>=0;u--)
        for(p=(M-u)/v[i];p>=0;p--)
            j=u+p*v[i];
```

- 新的状态转移方程：

$$f[u + p * v_i] = \max_{p-c_i \le k \le p-1} \{f[u + k * v_i] + (p - k) * w_i\}$$

- 此时是否就适合优化了呢？

多重背包

$$f[u + p * v_i] = \max_{p-c_i \le k \le p-1} \{f[u + k * v_i] + (p-k) * w_i\}$$

- 我们可以将外层循环i, u都看作定值，只看内层循环p，然后再快 速做k的决策
- 右边的式子可以分离成$(f[u + k * v_i] - k * w_i) + p * w_i$
- 因此，我们建立一个k单调递增，数值$f[u + k * v_i] - k * w_i$单调递减的队列，即可每次$O(1)$进行转移。
- 具体的：

$$f[u+p*v_i] = \max_{p-c_i \le k \le p-1} \{f[u+k*v_i] + (p-k)*w_i\}$$

- 对于每个p:
- 检查队头的合法性，将大于$p-1$的出队
- 取队头为最优决策，更新$f[u+p*v_i]$
- 将新的可用的决策$k=p-c_i-1$插入到队尾，并检查队尾的单调性，排查无用决策
- 整体时间复杂度：
- $O(NM)$

- 题外话：
- 该写法时间复杂度上更优了。
- 但是由于常数会比较大，因此并不一定会比二进制分组的多重背包写法更优。
- 这种提供一种优化思路。

总结下

状态转移方程	定值(外层循环)	状态变量(内层循环)	决策变量
$ans = \max\limits_{1\le i\le N}\left\{S[i] - \min\limits_{i-M\le j\le i-1}\{S[j]\}\right\}$		i	j
$F[i,j] = \max\limits_{j-L_i\le k\le S_i-1}\{F[i-1,k] + P_i * (j-k)\}$	i	j	k
$F[i] = \min\limits_{0\le j<i 并且 \sum_{k=j+1}^{i} A_k\le M}\left\{F[j] + \max\limits_{j+1\le k\le i}\{A_k\}\right\}$		i	j
$F[u+p*V_i] = \max\limits_{p-C_i\le k\le p-1}\{F[u+k*V_i] + (p-k)*W_i\}$	i,u	p	k

都可以归结于以下形式：

$$f[i] = \min_{L(i)\le j\le R(i)}\{f[j] + val(i,j)\}$$

1D/1D动态规划

1D/1D动态规划

- 指的是状态数为 $O(n)$，每一个状态决策量为 $O(n)$的动态规划方程。 直接求解的时间复杂度为 $O(n^2)$，但是，绝大多数这样的方程通过合理的组织与优化都是可 以优化到 $O(nlogn)$乃至 $O(n)$的时间复杂度的。

总结下

$$f[i] = \min_{L(i)\le j\le R(i)}\{f[j] + val(i,j)\}$$

- 最优化问题，$L(i)$和$R(i)$是关于变量i的一次函数，限制了决策j的取值范围
- 保证上下界的变化是单调的
- $val(i, j)$是关于i, j的多项式
- 单调队列优化的**基本条件**：
- $val(i, j)$可以拆分成两部分，每部分只与i, j中的一项有关。

总结下

$$f[i] = \min_{L(i) \le j \le R(i)} \{f[j] + val(i,j)\}$$

- 基本套路：
- 将$val(i, j)$分离成两部分：1）只含有i 2)只含有j
- 第一部分不会影响决策，使用单调队列维护第二部分；
- 及时将不可能的决策排出掉，并迅速找到最优解