动态规划优化

贾志鹏

如何优化动态规划？

1. 预处理

如何优化动态规划？

1. 预处理
2. 重新设计状态

如何优化动态规划？

1. 预处理
2. 重新设计状态
3. 滚动数组

如何优化动态规划？

1. 预处理
2. 重新设计状态
3. 滚动数组
4. 其他高端技术

选数问题

给定 n $(n \leq 10^5)$ 个整数，选择连续一段整数，要求选择的整数个数不超过 k $(1 \leq k \leq n)$，且它们的和最大。

选数问题

给定 n $(n \leq 10^5)$ 个整数，选择连续一段整数，要求选择的整数个数不超过 k $(1 \leq k \leq n)$，且它们的和最大。

用 $s(i)$ 表示前 i 个整数的和，特别地 $s(0) = 0$。假设选择第 l 个到第 r 个整数，它们的和就是 $s(r) - s(l-1)$。

选数问题

给定 n $(n \leq 10^5)$ 个整数，选择连续一段整数，要求选择的整数个数不超过 k $(1 \leq k \leq n)$，且它们的和最大。

用 $s(i)$ 表示前 i 个整数的和，特别地 $s(0) = 0$。假设选择第 l 个到第 r 个整数，它们的和就是 $s(r) - s(l-1)$。

问题变为：选择正整数 i, j $(i > j)$ 满足 $i - j \leq k$ 且 $s(i) - s(j)$ 最大。

选数问题

给定 n $(n \leq 10^5)$ 个整数，选择连续一段整数，要求选择的整数个数不超过 k $(1 \leq k \leq n)$，且它们的和最大。

用 $s(i)$ 表示前 i 个整数的和，特别地 $s(0) = 0$。假设选择第 l 个到第 r 个整数，它们的和就是 $s(r) - s(l-1)$。

问题变为：选择正整数 i, j $(i > j)$ 满足 $i - j \leq k$ 且 $s(i) - s(j)$ 最大。

考虑只枚举 i，问题变为找出在 $[\max(0, i-k), i-1]$ 内的 j 使得 $s(j)$ 最小。

引入单调队列

固定 i，考虑区间 $[\max(0, i-k), i-1]$ 内的两个决策 j_1, j_2 $(j_1 < j_2)$，假如 $s(j_1) \geq s(j_2)$，说明 $s(j_2)$ 相比 $s(j_1)$ 更优。

引入单调队列

固定 i，考虑区间 $[\max(0, i-k), i-1]$ 内的两个决策 j_1, j_2 $(j_1 < j_2)$，假如 $s(j_1) \geq s(j_2)$，说明 $s(j_2)$ 相比 $s(j_1)$ 更优。

注意到随着 i 的增加，区间 $[\max(0, i-k), i-1]$ 的两个端点都单调不降，因此对于之后的 i，$s(j_2)$ 也一定比 $s(j_1)$ 更优。

引入单调队列

固定 i，考虑区间 $[\max(0, i-k), i-1]$ 内的两个决策 j_1, j_2 $(j_1 < j_2)$，假如 $s(j_1) \geq s(j_2)$，说明 $s(j_2)$ 相比 $s(j_1)$ 更优。

注意到随着 i 的增加，区间 $[\max(0, i-k), i-1]$ 的两个端点都单调不降，因此对于之后的 i，$s(j_2)$ 也一定比 $s(j_1)$ 更优。

由此考虑使用一个队列维护可能的决策 j，并且队列中的元素从左到右满足 s 值逐渐变大。

核心代码

```
int head = 0, tail = 0;
long long ans = -inf;
for (int i = 1; i <= n; i ++) {
    while (head < tail) {
        if (s[i - 1] > s[queue[tail - 1]]) break;
        tail --;
    }
    queue[tail ++] = i - 1;
```

核心代码

```
    while (head < tail) {
        if (i - queue[head] <= k) break;
        head ++;
    }
    if (head < tail) {
        ans = max(ans, s[i] - s[queue[head]]);
    }
}
```

单调队列适用范围

考虑一类动态规划问题，其中状态为 $f(i)$，转移大致如下

$$f(i) = h(i) + \min \text{ 或 } \max\{g(j) \mid a_i \le j \le b_i < i\}$$

其中 $h(i)$ 为仅与 i 相关的函数、$g(j)$ 为与 j 或 $f(j)$ 相关函数。

对于 $\{a_i\}$ 和 $\{b_i\}$，要求满足 $a_i \le a_{i+1}$，$b_i \le b_{i+1}$。

SCOI 2010 第一试 股票交易

最近 lxhgww 又迷上了投资股票，通过一段时间的观察和学习，他总结出了股票行情的一些规律。

通过一段时间的观察，lxhgww 预测到了未来 T 天内某只股票的走势，第 i 天的股票买入价为每股 AP_i，第 i 天的股票卖出价为每股 BP_i（数据保证对于每个 i，都有 $AP_i \geq BP_i$），但是每天不能无限制地交易，于是股票交易所规定第 i 天的一次买入至多只能购买 AS_i 股，一次卖出至多只能卖出 BS_i 股。

SCOI 2010 第一试 股票交易

另外，股票交易所还制定了两个规定。为了避免大家疯狂交易，股票交易所规定在两次交易（某一天的买入或者卖出均算是一次交易）之间，至少要间隔 W 天，也就是说如果在第 i 天发生了交易，那么从第 $i+1$ 天到第 $i+W$ 天，均不能发生交易。同时，为了避免垄断，股票交易所还规定在任何时间，一个人的手里的股票数不能超过 $MaxP$。

在第 1 天之前，lxhgww 手里有一大笔钱（可以认为钱的数目无限），但是没有任何股票，当然，T 天以后，lxhgww 想要赚到最多的钱，聪明的程序员们，你们能帮助他吗？

数据范围：$0 \leq W < T \leq 2000$，$1 \leq MaxP \leq 2000$，$1 \leq BP_i \leq AP_i \leq 1000$，$1 \leq AS_i, BS_i \leq MaxP$。

分析

用 $f(i,j)$ 表示经过前 i 天、目前手中股票数为 j 时的最大获利。很明显第 i 天如果进行交易，不可能既买入又卖出。由此得转移方程为

分析

用 $f(i,j)$ 表示经过前 i 天、目前手中股票数为 j 时的最大获利。很明显第 i 天如果进行交易，不可能既买入又卖出。由此得转移方程为

$$f(i,j)=\begin{cases} f(i-1,j) \\ f(i-W-1,k)-(j-k)\times AP_i & k<j,\ j-k\le AS_i \\ f(i-W-1,k)+(k-j)\times BP_i & k>j,\ k-j\le BS_i \end{cases}$$

分析

用 $f(i,j)$ 表示经过前 i 天、目前手中股票数为 j 时的最大获利。很明显第 i 天如果进行交易，不可能既买入又卖出。由此得转移方程为

$$f(i,j)=\begin{cases} f(i-1,j) \\ f(i-W-1,k)-(j-k)\times AP_i & k<j,\ j-k\le AS_i \\ f(i-W-1,k)+(k-j)\times BP_i & k>j,\ k-j\le BS_i \end{cases}$$

对于二三两种情况，将 k 需满足的条件变形，可得 $j-AS_i\le k<j$ 和 $j<k\le j+BS_i$。这样就能使用单调队列优化。

送礼物

Crash 买来了 n 件礼物，他要将这些礼物送给他的好友们。

Crash 先将礼物们排成一排，从左到右用正整数 $1 \ldots n$ 编号，每件礼物有一个正整数的价值，依次用 $w_1, w_2, \ldots, w_n$ 表示。Crash 送给每位好友的礼物一定是编号上连续的一段，满足这些礼物中任意两件礼物的价值差不会超过 m，同时送给每个好友的礼物个数不少于 k。

给出 m, k 和每件礼物的价值，问 Crash 最多能送出多少件礼物。

期望时间复杂度为 $O(n)$ 的算法。

分析

用 $f(i)$ 表示将前 i 件礼物分给好友、最多能送多少件。很明显 $f(i)$ 的转移如下

$$f(i) = \max(f(i-1), \max\{f(j) + i - j \mid b(i) \le j \le i - k\})$$

分析

用 $f(i)$ 表示将前 i 件礼物分给好友、最多能送多少件。很明显 $f(i)$ 的转移如下

$$f(i) = \max(f(i-1), \max\{f(j) + i - j \mid b(i) \le j \le i - k\})$$

注意转移是选择第 $j+1 \ldots i$ 件礼物送给某个好友，当 j 变小时更有可能出现这中间存在两件礼物价值差大于 m，因此用 $b(i)$ 表示 j 的下界。不难证明 $b(i+1) \ge b(i)$。

分析

用 $f(i)$ 表示将前 i 件礼物分给好友、最多能送多少件。很明显 $f(i)$ 的转移如下

$$f(i) = \max(f(i-1), \max\{f(j) + i - j \mid b(i) \le j \le i - k\})$$

注意转移是选择第 $j+1 \ldots i$ 件礼物送给某个好友，当 j 变小时更有可能出现这中间存在两件礼物价值差大于 m，因此用 $b(i)$ 表示 j 的下界。不难证明 $b(i+1) \ge b(i)$。

根据 $b(i)$ 的单调性，可以利用单调队列维护决策。假设当前单调队列中队首的决策为 k，还需要两个单调队列分别维护 $w_{k+1}, w_{k+2}, \ldots, w_i$ 中的最大值和最小值。

NOIP 2009 普及组 道路游戏

小新正在玩一个简单的电脑游戏。

游戏中有一条环形马路，马路上有 n 个机器人工厂，两个相邻机器人工厂之间由一小段马路连接。小新以某个机器人工厂为起点，按顺时针顺序依次将这 n 个机器人工厂编号为 $1 \sim n$，因为马路是环形的，所以第 n 个机器人工厂和第 1 个机器人工厂是由一段马路连接在一起的。小新将连接机器人工厂的这 n 段马路也编号为 $1 \sim n$，并规定第 i 段马路连接第 i 个机器人工厂和第 $i+1$ 个机器人工厂 $(1 \leq i \leq n-1)$，第 n 段马路连接第 n 个机器人工厂和第 1 个机器人工厂。

NOIP 2009 普及组 道路游戏

游戏过程中，每个单位时间内，每段马路上都会出现一些金币，金币的数量会随着时间发生变化，即不同单位时间内同一段马路上出现的金币数量可能是不同的。小新需要机器人的帮助才能收集到马路上的金币。所需的机器人必须在机器人工厂用一些金币来购买，机器人一旦被购买，便会沿着环形马路按顺时针方向一直行走，在每个单位时间内行走一次，即从当前所在的机器人工厂到达相邻的下一个机器人工厂，并将经过的马路上的所有金币收集给小新，例如，小新在 i $(1 \leq i \leq n)$ 号机器人工厂购买了一个机器人，这个机器人会从 i 号机器人工厂开始，顺时针在马路上行走，第一次行走会经过 i 号马路，到达 $i+1$ 号机器人工厂（如果 $i = n$，机器人会到达第 1 个机器人工厂），并将 i 号马路上的所有金币收集给小新。

NOIP 2009 普及组 道路游戏

游戏中，环形马路上不能同时存在 2 个或者 2 个以上的机器人，并且每个机器人最多能够在环形马路上行走 p 次。小新购买机器人的同时，需要给这个机器人设定行走次数，行走次数可以为 $1 \sim p$ 之间的任意整数。当马路上的机器人行走完规定的次数之后会自动消失，小新必须立刻在任意一个机器人工厂中购买一个新的机器人，并给新的机器人设定新的行走次数。

以下是游戏的一些补充说明：

1. 游戏从小新第一次购买机器人开始计时。

2. 购买机器人和设定机器人的行走次数是瞬间完成的，不需要花费时间。

3. 购买机器人和机器人行走是两个独立的过程，机器人行走时不能购买机器人，购买完机器人并且设定机器人行走次数之后机器人才能行走。

4. 在同一个机器人工厂购买机器人的花费是相同的，但是在不同机器人工厂购买机器人的花费不一定相同。

5. 购买机器人花费的金币，在游戏结束时再从小新收集的金币中扣除，所以在游戏过程中小新不用担心因金币不足，无法购买机器人而导致游戏无法进行。也因为如此，游戏结束后，收集的金币数量可能为负。

现在已知每段马路上每个单位时间内出现的金币数量和在每个机器人工厂购买机器人需要的花费，请你告诉小新，经过 m 个单位时间后，扣除购买机器人的花费，小新最多能收集到多少金币。

数据范围：$2 \leq n \leq 1000$，$1 \leq m \leq 1000$，$1 \leq p \leq m$。

分析

用 $f(i)$ 表示前 i 单位时间最多能收集到的金币数。转移需要枚举上一个机器人结束行走的时间，转移方程为

$$f(i) = \max\{f(j) + g(j+1, i-j) \mid j < i,\ i - j \le p\}$$

其中 $g(i, j)$ 表示从第 i 单位时间开始走 j 步所能获得的最大金币数，其中已经去掉了购买机器人的费用。

分析

用 $f(i)$ 表示前 i 单位时间最多能收集到的金币数。转移需要枚举上一个机器人结束行走的时间，转移方程为

$$f(i) = \max\{f(j) + g(j+1, i-j) \mid j < i,\ i - j \le p\}$$

其中 $g(i, j)$ 表示从第 i 单位时间开始走 j 步所能获得的最大金币数，其中已经去掉了购买机器人的费用。

用 $a(i, j)$ 表示第 i 个时间在第 j 条道路上的金币数、$b(i)$ 表示在第 i 个机器人工厂购买机器人的价格。因为工厂是一个环，当 $j > n$ 时，令 $a(i, j) = a(i, j - n)$。

分析

由此可得

$$g(i,j) = \max\left\{-b(k) + \sum_{t=0}^{j-1} a(i+t, k+t)\right\}$$

其中 $k = 1 \ldots n$。

由此可得

$$g(i,j) = \max\left\{-b(k) + \sum_{t=0}^{j-1} a(i+t, k+t)\right\}$$

其中 $k = 1 \ldots n$。

注意到上面的求和其实是一条斜线上的部分和，那条斜线上行列编号差为 $k-t$。用 $s(d,i)$ 表示 $\sum_{j=1}^{i} a(j, j+d)$，那么上面的式子可以变为

$$g(i,j) = \max\{-b(k) + s(k-i, i+j-1) - s(k-i, i-1)\}$$

分析

将其代入 $f(i)$ 的转移可得

$$f(i) = \max\{f(j) - b(k) + s(k-j-1, i) - s(k-j-1, j)\}$$

其中条件为 $j < i,\ i - j \le p,\ 1 \le k \le n$。

分析

将其代入 $f(i)$ 的转移可得

$$f(i) = \max\{f(j) - b(k) + s(k-j-1, i) - s(k-j-1, j)\}$$

其中条件为 $j < i,\ i - j \le p,\ 1 \le k \le n$。

令 $k' = k - j - 1$，并且转移时枚举 k'，则转移变为

$$f(i) = s(k', i) + \max\{f(j) - b(k' + j + 1) - s(k', j)\}$$

这里 $i - p \le j < i$。

分析

将其代入 $f(i)$ 的转移可得

$$f(i) = \max\{f(j) - b(k) + s(k-j-1, i) - s(k-j-1, j)\}$$

其中条件为 $j < i,\ i - j \le p,\ 1 \le k \le n$。

令 $k' = k - j - 1$，并且转移时枚举 k'，则转移变为

$$f(i) = s(k', i) + \max\{f(j) - b(k' + j + 1) - s(k', j)\}$$

这里 $i - p \le j < i$。

不难想到使用单调队列优化，时间复杂度为 $O(nm)$。

凸包优化转移

考虑大致如下的转移方程

$$f(i) = h(i) + \min \text{ 或 } \max\{y(j) - k(i)x(j) \mid j < i\}$$

其中 $h(i), k(i)$ 为仅与 i 相关的函数、$x(j), y(j)$ 为与 j 或 $f(j)$ 相关函数。

凸包优化转移

考虑大致如下的转移方程

$$f(i) = h(i) + \min \text{ 或 } \max\{y(j) - k(i)x(j) \mid j < i\}$$

其中 $h(i), k(i)$ 为仅与 i 相关的函数、$x(j), y(j)$ 为与 j 或 $f(j)$ 相关函数。

将 $(x(j), y(j))$ 想象成平面上的点，$k(i)$ 想成直线的斜率，问题就变成最大化或最小化直线的截距，使得这条直线上碰到某个点 $(x(j), y(j))$。假如是最大化截距，可以想象有条斜率固定的直线，从很远的上方向下移动，第一个碰到的点 $(x(j), y(j))$ 就是最优决策。

凸包优化转移

考虑大致如下的转移方程

$$f(i) = h(i) + \min \text{ 或 } \max\{y(j) - k(i)x(j) \mid j < i\}$$

其中 $h(i), k(i)$ 为仅与 i 相关的函数、$x(j), y(j)$ 为与 j 或 $f(j)$ 相关函数。

将 $(x(j), y(j))$ 想象成平面上的点，$k(i)$ 想成直线的斜率，问题就变成最大化或最小化直线的截距，使得这条直线上碰到某个点 $(x(j), y(j))$。假如是最大化截距，可以想象有条斜率固定的直线，从很远的上方向下移动，第一个碰到的点 $(x(j), y(j))$ 就是最优决策。

画些图可以观察出，碰到的点一定在点集的上凸壳或下凸壳上。

凸包优化转移

假如 $x(j)$ 随 j 单调，可以直接使用栈维护凸壳，查询时只需要使用二分查找即可找到需要的点，这样总时间复杂度为 $O(n \log n)$。

凸包优化转移

假如 $x(j)$ 随 j 单调，可以直接使用栈维护凸壳，查询时只需要使用二分查找即可找到需要的点，这样总时间复杂度为 $O(n \log n)$。

在前面的情况下，如果询问的斜率 $k(i)$ 随 i 单调，可以进一步使用单调队列，时间复杂度降为 $O(n)$。

凸包优化转移

假如 $x(j)$ 随 j 单调，可以直接使用栈维护凸壳，查询时只需要使用二分查找即可找到需要的点，这样总时间复杂度为 $O(n \log n)$。

在前面的情况下，如果询问的斜率 $k(i)$ 随 i 单调，可以进一步使用单调队列，时间复杂度降为 $O(n)$。

如果前面的条件都不满足，可以使用平衡树维护凸壳，总时间复杂度为 $O(n \log n)$，这种情况在实际的题目中比较少见。

HNOI 2008 第二试 玩具装箱

P 教授要去看奥运，但是他舍不下他的玩具，于是他决定把所有的玩具运到北京。他使用自己的压缩器进行压缩，其可以将任意物品变成一堆，再放到一种特殊的一维容器中。P 教授有编号为 $1 \ldots N$ 的 N $(N \leq 50000)$ 件玩具，第 i 件玩具经过压缩后变成一维长度为 C_i $(1 \leq C_i \leq 10^7)$。为了方便整理，P 教授要求在一个一维容器中的玩具编号是连续的。同时如果一个一维容器中有多个玩具，那么两件玩具之间要加入一个单位长度的填充物，形式地说如果将第 i 件玩具到第 j 个玩具放到一个容器中 $(i < j)$，那么容器的长度将为 $x = j - i + \sum_{k=i}^{j} C_k$。制作容器的费用与容器的长度有关，根据教授研究，如果容器长度为 x，其制作费用为 $(x - L)^2$，其中 L $(1 \leq L \leq 10^7)$ 是一个常量。P 教授不关心容器的数目，他可以制作出任意长度的容器，甚至超过 L。但他希望费用最小。

分析

令 $S_i = i + \sum_{k=1}^{i} C_k$，这样将第 j 件玩具到第 i 个玩具放到一个容器中时，容器长度为 $S_i - S_{j-1} - 1$。

分析

令 $S_i = i + \sum_{k=1}^{i} C_k$，这样将第 j 件玩具到第 i 个玩具放到一个容器中时，容器长度为 $S_i - S_{j-1} - 1$。

用 $f(i)$ 表示划分前 i 个玩具的最小费用，则转移方程为

$$f(i) = \min\{f(j) + (S_i - S_j - 1 - L)^2 \mid 0 \le j < i\}$$

分析

令 $S_i = i + \sum_{k=1}^{i} C_k$，这样将第 j 件玩具到第 i 个玩具放到一个容器中时，容器长度为 $S_i - S_{j-1} - 1$。

用 $f(i)$ 表示划分前 i 个玩具的最小费用，则转移方程为

$$f(i) = \min\{f(j) + (S_i - S_j - 1 - L)^2 \mid 0 \le j < i\}$$

将 $f(j) + (S_i - S_j - 1 - L)^2$ 变形可得

$$f(i) = S_i^2 + \min\{f(j) + (S_j + L + 1)^2 - 2S_i(S_j + L + 1) \mid 0 \le j < i\}$$

分析

令 $S_i = i + \sum_{k=1}^{i} C_k$，这样将第 j 件玩具到第 i 个玩具放到一个容器中时，容器长度为 $S_i - S_{j-1} - 1$。

用 $f(i)$ 表示划分前 i 个玩具的最小费用，则转移方程为

$$f(i) = \min\{f(j) + (S_i - S_j - 1 - L)^2 \mid 0 \le j < i\}$$

将 $f(j) + (S_i - S_j - 1 - L)^2$ 变形可得

$$f(i) = S_i^2 + \min\{f(j) + (S_j + L + 1)^2 - 2S_i(S_j + L + 1) \mid 0 \le j < i\}$$

因此 $h(i) = S_i^2$、$k(i) = 2S_i$、$x(j) = S_j + L + 1$、$y(j) = (S_j + L + 1)^2$。注意到 $x(j)$ 随 j 单调增，$k(i)$ 随 i 单调增，可以将时间复杂度优化为 $O(N)$。

ZJOI 2007 第二试 仓库建设

L 公司有 N ($N \leq 10^6$) 个工厂，由高到底分布在一座山上。工厂 1 在山顶，工厂 N 在山脚。由于这座山处于高原内陆地区（干燥少雨），L 公司一般把产品直接堆放在露天，以节省费用。

突然有一天，L 公司的总裁 L 先生接到气象部门的电话，被告知三天之后将有一场暴雨，于是 L 先生决定紧急在某些工厂建立一些仓库以免产品被淋坏。由于地形的不同，在不同工厂建立仓库的费用可能是不同的。第 i 个工厂目前已有成品 P_i 件，在第 i 个工厂位置建立仓库的费用是 C_i。对于没有建立仓库的工厂，其产品应被运往其他的仓库进行储藏，而由于 L 公司产品的对外销售处设置在山脚的工厂 N，故产品只能往山下运（即只能运往编号更大的工厂的仓库），当然运送产品也是需要费用的，假设一件产品运送 1 个单位距离的费用是 1。假设建立的仓库容量都都是足够大的，可以容下所有的产品。

ZJOI 2007 第二试 仓库建设

你将得到以下数据：

1. 工厂 i 距离工厂 1 的距离 X_i（其中 $X_1 = 0$ 且 $X_{i+1} > X_i$）；

2. 工厂 i 目前已有成品数量 P_i；

3. 在工厂 i 建立仓库的费用 C_i。

请你帮助 L 公司寻找一个仓库建设的方案，使得总的费用（建造费用加运输费用）最小。

分析

用 $f(i)$ 表示考虑了前 i 个工厂后的最小总费用，容易得出转移方程

$$f(i) = \min\{f(j) + w(j+1, i) + C_i \mid 0 \leq j < i\}$$

其中 $w(j+1, i)$ 表示将工厂 $j+1, j+2, \ldots, i-1$ 内的成品运到工厂 i 的代价。不难得出 $w(j+1, i)$ 的计算式为

分析

用 $f(i)$ 表示考虑了前 i 个工厂后的最小总费用，容易得出转移方程

$$f(i) = \min\{f(j) + w(j+1, i) + C_i \mid 0 \le j < i\}$$

其中 $w(j+1, i)$ 表示将工厂 $j+1, j+2, \ldots, i-1$ 内的成品运到工厂 i 的代价。不难得出 $w(j+1, i)$ 的计算式为

$$\sum_{k=j+1}^{i} (X_i - X_k)P_k = \sum_{k=j+1}^{i} X_i P_k - X_k P_k = X_i \sum_{k=j+1}^{i} P_k - \sum_{k=j+1}^{i} X_k P_k$$

分析

用 $f(i)$ 表示考虑了前 i 个工厂后的最小总费用，容易得出转移方程

$$f(i) = \min\{f(j) + w(j+1, i) + C_i \mid 0 \le j < i\}$$

其中 $w(j+1, i)$ 表示将工厂 $j+1, j+2, \ldots, i-1$ 内的成品运到工厂 i 的代价。不难得出 $w(j+1, i)$ 的计算式为

$$\sum_{k=j+1}^{i} (X_i - X_k)P_k = \sum_{k=j+1}^{i} X_i P_k - X_k P_k = X_i \sum_{k=j+1}^{i} P_k - \sum_{k=j+1}^{i} X_k P_k$$

令 $A_i = \sum_{k=1}^{i} P_k$，$B_i = \sum_{k=1}^{i} X_k P_k$，则

$$w(j+1, i) = X_i(A_i - A_j) - (B_i - B_j)$$

分析

因此转移方程变为

$$f(i) = \min\{f(j) + X_i(A_i - A_j) - (B_i - B_j) + C_i \mid 0 \leq j < i\}$$

分析

因此转移方程变为

$$f(i) = \min\{f(j) + X_i(A_i - A_j) - (B_i - B_j) + C_i \mid 0 \leq j < i\}$$

整理可得

$$f(i) = X_i A_i + C_i - B_i + \min\{f(j) + B_j - X_i A_j \mid 0 \leq j < i\}$$

分析

因此转移方程变为

$$f(i) = \min\{f(j) + X_i(A_i - A_j) - (B_i - B_j) + C_i \mid 0 \le j < i\}$$

整理可得

$$f(i) = X_i A_i + C_i - B_i + \min\{f(j) + B_j - X_i A_j \mid 0 \le j < i\}$$

由于 X_i, A_j 都单调，时间复杂度可以优化为 $O(N)$。

JSOI 2011 差额 第二试 柠檬

Flute 很喜欢柠檬。它准备了一串用树枝串起来的贝壳，打算用一种魔法把贝壳变成柠檬。

贝壳一共有 N $(1 \leq N \leq 10^5)$ 只，按顺序串在树枝上。为了方便，我们从左到右给贝壳编号 $1 \ldots N$。每只贝壳的大小不一定相同，贝壳 i 的大小为 s_i $(1 \leq s_i \leq 10^4)$。

变柠檬的魔法要求，Flute 每次从树枝一端取下一小段连续的贝壳，并选择一种贝壳的大小 s_0。如果这一小段贝壳中大小为 s_0 的贝壳有 t 只，那么魔法可以把这一小段贝壳变成 $s_0 t^2$ 只柠檬。Flute 可以取任意多次贝壳，直到树枝上的贝壳被全部取完。各个小段中，Flute 选择的贝壳大小 s_0 可以不同。而最终 Flute 得到的柠檬数，就是所有小段柠檬数的总和。

Flute 想知道，它最多能用这一串贝壳变出多少柠檬。请你帮忙解决这个问题。

分析

首先观察到，每次取下的一小段，必然满足两端点大小相同，并且这一段选择的 s_0 就是端点的大小，否则拆开必然变更优。

分析

首先观察到，每次取下的一小段，必然满足两端点大小相同，并且这一段选择的 s_0 就是端点的大小，否则拆开必然变更优。

用 $f(i)$ 表示考虑前 i 个柠檬的最优解，可得转移方程

$$f(i) = \max\{f(j) + s_i(A_i - A_{j+1} + 1)^2 \mid s_i = s_{j+1},\ 0 \le j < i\}$$

其中 A_i 表示 $s_1, s_2, \ldots, s_i$ 中与 s_i 相同的个数。

分析

对转移方程变形得

$$f(i) = s_i(A_i + 1)^2 + \max\{f(j) + s_{j+1}A_{j+1}^2 - s_i(A_i + 1)A_{j+1}\}$$

分析

对转移方程变形得

$$f(i) = s_i(A_i + 1)^2 + \max\{f(j) + s_{j+1}A_{j+1}^2 - s_i(A_i + 1)A_{j+1}\}$$

由于转移条件有 $s_i = s_{j+1}$，可以考虑维护若干个栈，时间复杂度为 $O(N)$。

四边形不等式

用 $w(i,j)$ 表示决策函数，动态规划中常常会出现如下形式的转移方程

$$f(i) = \min\{f(j) + w(j,i) \mid 0 \le j < i\}$$

四边形不等式

用 $w(i,j)$ 表示决策函数，动态规划中常常会出现如下形式的转移方程

$$f(i) = \min\{f(j) + w(j,i) \mid 0 \le j < i\}$$

如果对于任意 i,j，决策函数 $w(i,j)$ 都满足如下不等式

$$w(i,j) + w(i+1,j+1) \le w(i+1,j) + w(i,j+1)$$

可以证明 $f(i)$ 的最优决策 j 随 i 单调不降，由此可以将总时间复杂度优化到 $O(n \log n)$。

四边形不等式

每次新算出一个状态 $f(j)$ 后，考虑决策 j 可以更新哪些 $f(i)$。根据决策单调性，更新之前的决策是单调的，也就是划分成若干段，每段的最优决策依次是 $1, 2, \ldots$。

对于决策 j，当前来看以它为最优决策的状态肯定是最后连续若干个，可以二分这个转折点。

具体做法是用栈维护当前的最优决策划分，算出 $f(j)$ 后，从后完全看，对于每段计算最小的状态中，决策 j 是否更优，如果不是说明转折点就在这段中，否则将它从栈中弹出。

对于转折点所在的那一段，可以二分找到具体转折位置，最后往栈中压入新的一段。

四边形不等式

至于这样做的正确性，需要不少数学的推导，有能力的同学可以自行尝试。

实际情况中，你甚至不需要证明决策函数满足四边形不等式，如果你发现动态规划的转移具有前面描述的形式，并且没法使用斜率方式优化，数据范围大致 10^5，直接这么做一般就是正确的。

NOI 2009 第一试 诗人小 G

小 G 是一个出色的诗人，经常作诗自娱自乐。但是，他一直被一件事情所困扰，那就是诗的排版问题。

一首诗包含了若干个句子，对于一些连续的短句，可以将它们用空格隔开并放在一行中，注意一行中可以放的句子数目是没有限制的。小 G 给每首诗定义了一个行标准长度 L（行的长度为一行中符号的总个数），他希望排版后每行的长度都和行标准长度相差不远。显然排版时，不应改变原有的句子顺序，并且小 G 不允许把一个句子分在两行或者更多的行内。在满足上面两个条件的情况下，小 G 对于排版中的每行定义了一个不协调度，为这行的实际长度与行标准长度差值绝对值的 P 次方，而一个排版的不协调度为所有行不协调度的总和。

小 G 最近又作了几首诗，现在请你对这首诗进行排版，使得排版后的诗尽量协调（即不协调度尽量小），并把排版的结果告诉他。

数据范围：诗句数目 $N \leq 10^5$，$P \leq 10$，$L \leq 3000000$，每句诗长度不超过 30。

分析

令 $S_i = i + \sum_{k=1}^{i} a_k$，其中 a_i 为第 i 句诗的长度。

用 $f(i)$ 表示划分前 i 句诗的最小不和谐度，则转移方程为

$$f(i) = \min\{f(j) + |S_i - S_j - L - 1|^P \mid 0 \leq j < i\}$$

分析

令 $S_i = i + \sum_{k=1}^{i} a_k$，其中 a_i 为第 i 句诗的长度。

用 $f(i)$ 表示划分前 i 句诗的最小不和谐度，则转移方程为

$$f(i) = \min\{f(j) + |S_i - S_j - L - 1|^P \mid 0 \le j < i\}$$

假如 $w(i,j) = |S_j - S_i - L - 1|^P$ 满足四边形不等式，就可以用前面的方法直接优化了。实际比赛时可以算出一些 $w(i,j)$ 用程序来验证。

分析

令 $S_i = i + \sum_{k=1}^{i} a_k$，其中 a_i 为第 i 句诗的长度。

用 $f(i)$ 表示划分前 i 句诗的最小不和谐度，则转移方程为

$$f(i) = \min\{f(j) + |S_i - S_j - L - 1|^P \mid 0 \le j < i\}$$

假如 $w(i,j) = |S_j - S_i - L - 1|^P$ 满足四边形不等式，就可以用前面的方法直接优化了。实际比赛时可以算出一些 $w(i,j)$ 用程序来验证。

如果你有一些数学基础，根据函数 $f(x) = |x|^n$ 的凸性，很容易证明 $w(i,j)$ 满足四边形不等式。

POI 2011 Stage I Lightning Conductor

有 n $(n \leq 500000)$ 座大楼，第 i 座大楼的高度为 h_i $(0 \leq h_i \leq 10^9)$。假如在第 i 座大楼上安装长度为 p 的避雷针，满足 $h_j \leq h_i + p - \sqrt{|i-j|}$ 的大楼 j 就能被保护到。

对于每座大楼 i，计算它要保护其他所有大楼时，避雷针的最短长度。

分析

实际就是对于每个 i，计算

$$\max\{h_j + \sqrt{|i-j|} \mid 1 \leq j \leq n\}$$

分析

实际就是对于每个 i，计算

$$\max\{h_j + \sqrt{|i-j|} \mid 1 \le j \le n\}$$

我们将它分成两部分计算

$$\max\{h_j + \sqrt{i-j} \mid 1 \le j < i\}$$
$$\max\{h_j + \sqrt{j-i} \mid i < j \le n\}$$

分析

实际就是对于每个 i，计算

$$\max\{h_j + \sqrt{|i-j|} \mid 1 \le j \le n\}$$

我们将它分成两部分计算

$$\max\{h_j + \sqrt{i-j} \mid 1 \le j < i\}$$
$$\max\{h_j + \sqrt{j-i} \mid i < j \le n\}$$

下面我们只考虑第一部分，第二部分做法和第一部分类似。

分析

这里虽然没有 $f(i)$ 进行转移，但还是可以使用类似方法优化。

分析

这里虽然没有 $f(i)$ 进行转移，但还是可以使用类似方法优化。

令 $w(i,j)=h_i+\sqrt{j-i}$，其中需要满足 $i\le j$。

分析

这里虽然没有 $f(i)$ 进行转移，但还是可以使用类似方法优化。

令 $w(i,j)=h_i+\sqrt{j-i}$，其中需要满足 $i\le j$。

当 $i<j$ 时，不难证明

$$w(i+1,j)+w(i,j+1)\le w(i,j)+w(i+1,j+1)$$

分析

这里虽然没有 $f(i)$ 进行转移，但还是可以使用类似方法优化。

令 $w(i,j)=h_i+\sqrt{j-i}$，其中需要满足 $i\le j$。

当 $i<j$ 时，不难证明

$$w(i+1,j)+w(i,j+1)\le w(i,j)+w(i+1,j+1)$$

其实就是证明 $\sqrt{j-i-1}+\sqrt{j-i+1}\le 2\sqrt{j-i}$，由 $f(x)=\sqrt{x}$ 的凹凸性可以直接得出。

分析

这里虽然没有 $f(i)$ 进行转移，但还是可以使用类似方法优化。

令 $w(i,j)=h_i+\sqrt{j-i}$，其中需要满足 $i\le j$。

当 $i<j$ 时，不难证明

$$w(i+1,j)+w(i,j+1)\le w(i,j)+w(i+1,j+1)$$

其实就是证明 $\sqrt{j-i-1}+\sqrt{j-i+1}\le 2\sqrt{j-i}$，由 $f(x)=\sqrt{x}$ 的凹凸性可以直接得出。

这样的话，就可以使用之前所说的方法维护决策。

All-Ukrainian School Olympiad in Informatics F Tourist

你现在站在一个数轴上，开始时时间为 0 时刻，现在有 N $(N \leq 10^5)$ 次奖励机会，每个奖励机会由两个参数 x, t $(|x| \leq 2 \cdot 10^8,\ 1 \leq t \leq 2 \cdot 10^6)$ 组成，表示如果在 t 时刻你到达了坐标为 x 的点，可以获得一分。数据保证不存在两次奖励机会的 x 和 t 都相同。你移动的速度不能超过 v $(1 \leq v \leq 1000)$，也就是说 t 的时间间隔内你最多移动 vt 的距离。求假如起点在原点和起点可以任意选两种情况下的最大得分。

分析

考虑将所有奖励机会按照 t 排序，用 $f(i)$ 表示现在在坐标为 x_i 的点、时间为 t_i 时的最大得分，容易得出如下转移：

$$f(i) = \max\{f(j) + 1 \mid |x_i - x_j| \leq v(t_i - t_j)\}$$

分析

考虑将所有奖励机会按照 t 排序，用 $f(i)$ 表示现在在坐标为 x_i 的点、时间为 t_i 时的最大得分，容易得出如下转移：

$$f(i) = \max\{f(j) + 1 \mid |x_i - x_j| \leq v(t_i - t_j)\}$$

对于起点的限制，两种情况只是边界不太一样，转移是一样的。

分析

考虑将所有奖励机会按照 t 排序，用 $f(i)$ 表示现在在坐标为 x_i 的点、时间为 t_i 时的最大得分，容易得出如下转移：

$$f(i) = \max\{f(j) + 1 \mid |x_i - x_j| \leq v(t_i - t_j)\}$$

对于起点的限制，两种情况只是边界不太一样，转移是一样的。

这样时间复杂度为 $O(N^2)$，考虑如何优化。

分析

尝试将 $|x_i - x_j| \leq v(t_i - t_j)$ 变形找出突破口：

$$
\begin{aligned}
& |x_i - x_j| \leq v(t_i - t_j) \\
\Leftrightarrow \quad & -v(t_i - t_j) \leq x_i - x_j \leq v(t_i - t_j) \\
\Leftrightarrow \quad & \begin{cases} vt_j - x_j \leq vt_i - x_i \\ vt_j + x_j \leq vt_i + x_i \end{cases}
\end{aligned}
$$

分析

尝试将 $|x_i - x_j| \le v(t_i - t_j)$ 变形找出突破口：

$$
\begin{aligned}
& |x_i - x_j| \le v(t_i - t_j) \\
\Leftrightarrow \quad & -v(t_i - t_j) \le x_i - x_j \le v(t_i - t_j) \\
\Leftrightarrow \quad & \begin{cases} vt_j - x_j \le vt_i - x_i \\ vt_j + x_j \le vt_i + x_i \end{cases}
\end{aligned}
$$

继续观察可以发现，$|x_i - x_j| \le v(t_i - t_j)$ 蕴含了 $t_i \ge t_j$，因此可以将所有奖励机会按照 $vt_i + x_i$ 排序，然后变为经典的 LIS 问题（$vt_i - x_i$ 单调不降）。

Codeforces Beta Round #72 (Div. 1 Only) E Two Subsequences

给定 N ($N \leq 200000$) 个长度为 L ($L \leq 20$) 的 01 串 $\{a_i\}$，将要这 N 个串分为两组 $\{b_i\}$ 和 $\{c_i\}$，每组内还按照原来的顺序排列。定义 $F(\{a_i\})$ 为 $\{a_i\}$ 中的串按顺序排列，并且相邻两项会尽可能利用公共部分，例如 $F(\{001, 011\}) = 0011$、$F(\{000, 011, 110\}) = 000110$，同时定义 $F(\emptyset) = \emptyset$。求一个分组方案，使得 $|F(\{b_i\})| + |F(\{c_i\})|$ 最小。

分析

观察 $F(\{a_i\})$ 的计算方法，能发现是将 $\{a_i\}$ 中最后一个串拼上去，因为所有串长度都是 L，因此只需要考虑和前一个串的公共部分。设 $w(s_1, s_2)$ 为最大的 k 满足 s_1 的 $L-k$ 后缀和 s_2 的 $L-k$ 前缀相等，由于 $L \leq 20$，不难想到用位运算加速，这样求 $w(s_1, s_2)$ 可以只要 $O(L)$ 的时间，进而可以观察出 $|F(\{a_i\})| = L + \sum_{i=2}^{|\{a_i\}|} w(a_{i-1}, a_i)$。

分析

观察 $F(\{a_i\})$ 的计算方法，能发现是将 $\{a_i\}$ 中最后一个串拼上去，因为所有串长度都是 L，因此只需要考虑和前一个串的公共部分。设 $w(s_1, s_2)$ 为最大的 k 满足 s_1 的 $L-k$ 后缀和 s_2 的 $L-k$ 前缀相等，由于 $L \leq 20$，不难想到用位运算加速，这样求 $w(s_1, s_2)$ 可以只要 $O(L)$ 的时间，进而可以观察出 $|F(\{a_i\})| = L + \sum_{i=2}^{|\{a_i\}|} w(a_{i-1}, a_i)$。

然后回到原题要求的东西，很容易想到用状态 $f(i, j)$ 表示一个子序列以 a_i 结尾、另一个以 a_j 结尾时的最优解，这样时间复杂度为 $O(N^2 L)$。注意到这里状态数就是二维的，优化状态通常难度很大，因此考虑重新设计状态。

分析

观察一下可以发现，分成两组其实就是将 $a_1, a_2, \ldots, a_N$ 分成若干段，编号为奇数的段属于一组、编号为偶数的段属于另外一组，可以先累加 $w(a_{i-1}, a_i)$，然后只需要考虑分隔的地方对代价的改变量。用 $f(i)$ 表示前 i 个数分为若干段、并且在 a_{i-1} 和 a_i 处分开时的最优代价，不难写出如下的转移方程：

$$f(i) = \min\{f(j) + w(a_{j-1}, a_i)\} - w(a_{i-1}, a_i)$$

分析

虽然看上去暴力转移的话，时间复杂度和之前的算法一样，但是状态数减少后优化时间复杂度就成为了可能。注意到 $L \leq 20$ 并且 $w(a_{j-1}, a_i) \leq L$，就想到枚举 $w(a_{j-1}, a_i)$ 的值。用 $g(l,t)$ 表示当 j 满足满足 a_{j-1} 的后 l 位为 t 时，$f(j)$ 的最小值。有了 $g(l,t)$ 后，$f(i)$ 的转移只需要 $O(L)$ 的时间，并且可以在 $O(L)$ 的时间内更新 $g(l,t)$，这样整个算法的时间复杂度优化为 $O\left(NL + 2^L\right)$。