线性 dp

2024 年 3 月 18 日

目录

基础

定义

具有线性阶段划分的动态规划算法被统称为线性 dp。

基础

定义

具有线性阶段划分的动态规划算法被统称为线性 dp。

在线性 dp 的问题中，动态规划都体现为“作用点在线性空间上的递推”—— dp 的阶段沿着各个维度线性增长，从一个或多个边界点开始有方向地向整个状态空间转移、扩展，最后每个状态上都保留了以自身为“目标” 的子问题的最优解。

定义

具有线性阶段划分的动态规划算法被统称为线性 dp。

在线性 dp 的问题中，动态规划都体现为“作用点在线性空间上的递推”—— dp 的阶段沿着各个维度线性增长，从一个或多个边界点开始有方向地向整个状态空间转移、扩展，最后每个状态上都保留了以自身为“目标” 的子问题的最优解。

一般来说，线性 dp 的状态定义和状态转移相对简单，对初学 dp 的同学来说比较容易接受。下面通过两个经典例题的讲解更加详细地解释线性 dp。

最长不下降子序列

最长不下降子序列问题

给定一个长度为 n 的序列 a，求出一个最长的 a 的子序列，满足该子序列的后一个元素不小于前一个元素。
数据范围：$n \leq 5000, a_i \leq 10^5$

最长不下降子序列

最长不下降子序列问题

给定一个长度为 n 的序列 a，求出一个最长的 a 的子序列，满足该子序列的后一个元素不小于前一个元素。
数据范围：$n \le 5000, a_i \le 10^5$

设 $f(i)$ 表示以 a_i 为结尾的最长不下降子序列的长度，初始化为将所有的 $f(i)$ 都为 1，最后所求即为 $\max_{i=1}^{n} f(i)$。

最长不下降子序列

最长不下降子序列问题

给定一个长度为 n 的序列 a，求出一个最长的 a 的子序列，满足该子序列的后一个元素不小于前一个元素。
数据范围：$n \le 5000, a_i \le 10^5$

设 $f(i)$ 表示以 a_i 为结尾的最长不下降子序列的长度，初始化为将所有的 $f(i)$ 都为 1，最后所求即为 $\max_{i=1}^{n} f(i)$。

考虑如何去求每个 $f(i)$，因为子序列中每个元素在原序列位置单调递增的性质，因此不下降子序列中 a_i 上一个元素 a_j 必须满足 $1 \le j < i$ 和 $a_j \le a_i$，则可以将以所有满足条件的 a_j 为结尾的最长不下降子序列后接上 a_i，取所有情况的最大值即为 $f(i)$。

最长不下降子序列问题

给定一个长度为 n 的序列 a，求出一个最长的 a 的子序列，满足该子序列的后一个元素不小于前一个元素。
数据范围：$n \leq 5000, a_i \leq 10^5$

设 $f(i)$ 表示以 a_i 为结尾的最长不下降子序列的长度，初始化为将所有的 $f(i)$ 都为 1，最后所求即为 $\max_{i=1}^{n} f(i)$。

考虑如何去求每个 $f(i)$，因为子序列中每个元素在原序列位置单调递增的性质，因此不下降子序列中 a_i 上一个元素 a_j 必须满足 $1 \leq j < i$ 和 $a_j \leq a_i$，则可以将以所有满足条件的 a_j 为结尾的最长不下降子序列后接上 a_i，取所有情况的最大值即为 $f(i)$。

于是有状态转移方程：

$$f(i) = \max_{j=1, a_j \leq a_i}^{i-1} \{f(j) + 1\}$$

最长不下降子序列

状态定义

$f(i)$ 表示以 a_i 为结尾的最长不下降子序列的长度。

最长不下降子序列

状态定义

$f(i)$ 表示以 a_i 为结尾的最长不下降子序列的长度。

状态转移方程

$$f(i) = \max_{j=1, a_j \le a_i}^{i-1} \{f(j) + 1\}$$

状态定义

$f(i)$ 表示以 a_i 为结尾的最长不下降子序列的长度。

状态转移方程

$$f(i) = \max_{j=1, a_j \le a_i}^{i-1} \{f(j) + 1\}$$

通过上述分析，便可在 $\mathcal{O}(n^2)$ 的时间复杂度内，将所有 $f(i)$ 求出，并统计出 $\max_{i=1}^{n} f(i)$。代码实现如下：

最长不下降子序列

状态定义

$f(i)$ 表示以 a_i 为结尾的最长不下降子序列的长度。

状态转移方程

$$f(i) = \max_{j=1, a_j \le a_i}^{i-1} \{f(j) + 1\}$$

通过上述分析，便可在 $\mathcal{O}(n^2)$ 的时间复杂度内，将所有 $f(i)$ 求出，并统计出 $\max_{i=1}^{n} f(i)$。代码实现如下：

```
int ans = 1; // 用于统计最大值的变量
for (int i = 1; i <= n; i ++) f[i] = 1; // 初始化
for (int i = 2; i <= n; i ++)
for (int j = 1; j < i; j ++)
if (a[j] <= a[i]) // 注意状态转移的条件
f[i] = max(f[i], f[j] + 1); // 状态转移
for (int i = 1; i <= n; i ++) // 统计最大值
ans = max (ans , f[i]);
```

最长不下降子序列

最长不下降子序列问题 `Plus`

给定一个长度为 n 的序列 a，求出一个最长的 a 的子序列，满足该子序列的后一个元素不小于前一个元素。

数据范围：$n \leq 10^5, a_i \leq 10^5$

最长不下降子序列

最长不下降子序列问题 Plus

给定一个长度为 n 的序列 a，求出一个最长的 a 的子序列，满足该子序列的后一个元素不小于前一个元素。
数据范围：$n \leq 10^5, a_i \leq 10^5$

将 n 的范围扩大到 10^5 时，上述做法就会 TLE，所以考虑在时间复杂度上进行优化。

最长不下降子序列

最长不下降子序列问题 Plus

给定一个长度为 n 的序列 a，求出一个最长的 a 的子序列，满足该子序列的后一个元素不小于前一个元素。
数据范围：$n \le 10^5, a_i \le 10^5$

将 n 的范围扩大到 10^5 时，上述做法就会 TLE，所以考虑在时间复杂度上进行优化。

Tips

容易发现，对于长度相同的不下降子序列，我们希望末尾元素越小越好，便更能够往后接入新的元素。

最长不下降子序列

最长不下降子序列问题 Plus

给定一个长度为 n 的序列 a，求出一个最长的 a 的子序列，满足该子序列的后一个元素不小于前一个元素。
数据范围：$n \leq 10^5, a_i \leq 10^5$

将 n 的范围扩大到 10^5 时，上述做法就会 TLE，所以考虑在时间复杂度上进行优化。

Tips

容易发现，对于长度相同的不下降子序列，我们希望末尾元素越小越好，便更能够往后接入新的元素。

设 $g(i)$ 表示长度为 i 的不下降子序列的末尾元素的最小值，同时记录现在最长不下降子序列的长度为 len。那么初始化为：

$$g(1) = a_1, len = 1$$

然后依次考虑将 a_2 到 a_n 依次加入后，$g(i)$ 和 len 的更新操作。

最长不下降子序列

状态定义

$g(i)$ 表示长度为 i 的不下降子序列的末尾元素的最小值。
len 表示现在最长不下降子序列的长度。

最长不下降子序列

状态定义

$g(i)$ 表示长度为 i 的不下降子序列的末尾元素的最小值。
len 表示现在最长不下降子序列的长度。

考虑将 a_i 加入时：

最长不下降子序列

状态定义

$g(i)$ 表示长度为 i 的不下降子序列的末尾元素的最小值。
len 表示现在最长不下降子序列的长度。

考虑将 a_i 加入时：

- 当 $a_i \geq g(len)$ 时，则可以将 a_i 放到最长不下降子序列之后，即执行 `g[++ len] = a[i];`

最长不下降子序列

状态定义

$g(i)$ 表示长度为 i 的不下降子序列的末尾元素的最小值。
len 表示现在最长不下降子序列的长度。

考虑将 a_i 加入时：

- 当 $a_i \geq g(len)$ 时，则可以将 a_i 放到最长不下降子序列之后，即执行 `g[++ len] = a[i];`
- 当 $a_i < g(len)$ 时，则找到 g 数组中从左到右第一个大于a_i 的元素（二分实现），将其替换为 a_i

最长不下降子序列

状态定义

$g(i)$ 表示长度为 i 的不下降子序列的末尾元素的最小值。
len 表示现在最长不下降子序列的长度。

考虑将 a_i 加入时：

- 当 $a_i \geq g(len)$ 时，则可以将 a_i 放到最长不下降子序列之后，即执行 `g[++ len] = a[i];`
- 当 $a_i < g(len)$ 时，则找到 g 数组中从左到右第一个大于a_i 的元素（二分实现），将其替换为 a_i

根据 $g(i)$ 的定义和更新操作，可以发现，$g(i)$ 是单调不减的，即满足：

$$g(1) \leq g(2) \leq g(3) \leq \cdots \leq g(len)$$

最长不下降子序列

代码实现如下:

```
len = 1, g[1] = a[1]; // 初始化
for (int i = 2, j; i <= n; i ++) {
    if (a[i] >= g[len]) { // 第 1 种情况
        g[++ len] = a[i];
    }
    else { // 第 2 种情况
        j = upper_bound (g + 1, g + len + 1, a[i]) - g;
        // upper_bound 为第一个大于给定值的下标
        // lower_bound 为第一个大于等于给定值的下标
        g[j] = a[i];
    }
}
```

最长不下降子序列

最长不下降子序列问题 Plus

给定一个长度为 n 的序列 a，求出一个最长的 a 的子序列，满足该子序列的后一个元素不小于前一个元素。

数据范围：$n \le 10^5, a_i \le 10^5$

最长不下降子序列

最长不下降子序列问题 Plus

给定一个长度为 n 的序列 a，求出一个最长的 a 的子序列，满足该子序列的后一个元素不小于前一个元素。
数据范围：$n \le 10^5, a_i \le 10^5$

另一种优化方法是利用树状数组，观察初始的状态转移方程：

$$f(i) = \max_{j=1, a_j \le a_i}^{i-1} \{f(j) + 1\}$$

最长不下降子序列问题 Plus

给定一个长度为 n 的序列 a，求出一个最长的 a 的子序列，满足该子序列的后一个元素不小于前一个元素。
数据范围：$n \le 10^5, a_i \le 10^5$

另一种优化方法是利用树状数组，观察初始的状态转移方程：

$$f(i) = \max_{j=1, a_j \le a_i}^{i-1} \{f(j) + 1\}$$

树状数组以 a_i 的值作为下标，维护对应 $f(i)$ 的前缀最大值，每次插入和询问的时间复杂度都是 $\mathcal{O}(\log n)$ 的，总的时间复杂度便是 $\mathcal{O}(n \log n)$ 的。
后续学习完树状数组可以再来实现一下这个方法。

最长公共子序列

最长公共子序列问题

给定一个长度为 n 的序列 a 和一个长度为 m 的序列 b，求出一个最长的序列，使得该序列既是 a 的子序列，又是 b 的子序列。
数据范围：$n, m \le 5000$ ，$a_i, b_i \le 10^5$

最长公共子序列

最长公共子序列问题

给定一个长度为 n 的序列 a 和一个长度为 m 的序列 b，求出一个最长的序列，使得该序列既是 a 的子序列，又是 b 的子序列。
数据范围：$n, m \le 5000$ ，$a_i, b_i \le 10^5$

设 $f(i,j)$ 表示考虑序列 a 前 i 个元素和序列 b 前 j 个元素的最长公共子序列的长度，则最后所求即为 $f(n,m)$。则有状态转移方程为：

$$f(i,j) = \begin{cases} f(i-1,j-1)+1 & a_i = b_j \\ \max\{f(i-1,j), f(i,j-1)\} & a_i \neq b_j \end{cases}$$

最长公共子序列

最长公共子序列问题

给定一个长度为 n 的序列 a 和一个长度为 m 的序列 b，求出一个最长的序列，使得该序列既是 a 的子序列，又是 b 的子序列。
数据范围：$n, m \leq 5000$，$a_i, b_i \leq 10^5$

设 $f(i, j)$ 表示考虑序列 a 前 i 个元素和序列 b 前 j 个元素的最长公共子序列的长度，则最后所求即为 $f(n, m)$。则有状态转移方程为：

$$f(i,j) = \begin{cases} f(i-1, j-1) + 1 & a_i = b_j \\ \max\{f(i-1, j), f(i, j-1)\} & a_i \neq b_j \end{cases}$$

如果 $a_i = b_j$，则可以将其接到公共子序列的后面；否则，可以跳过一个 a_i，也可以跳过一个 b_j，两种情况取最大值。
于是在 $\mathcal{O}(nm)$ 的时间复杂度内便可求出 $f(n, m)$。

最长公共子序列

代码实现如下：

```
for (int i = 1; i <= n; i ++) {
    for (int j = 1; j <= m; j ++) { // 顺序枚举状态
        if (a[i] == b[j]) { // 第 1 种情况
            f[i][j] = f[i - 1][j - 1] + 1;
        }
        else { // 第 2 种情况
            f[i][j] = max(f[i - 1][j], f[i][j - 1]);
        }
    }
}
```

课后习题

习题 1

导弹拦截（NOIP1999 提高组）

习题 2

最长公共子序列