

# 动态规划进阶（一）

## 1D/1D 动态规划以及区间动态规划

马玉坤

哈尔滨工业大学计算机科学与技术学院

2017 年 8 月 17 日

# 再谈 LIS

## 问题

求出一个序列的所有子序列中，满足元素单调上升（即  $a_i < a_{i+1}$ ）的最长子序列。

**子序列：**某个序列的子序列是从最初序列通过去除某些元素但不破坏余下元素的相对位置（在前或在后）而形成的新序列。比如序列 1、3、5 都是序列 1,2,3,4,5 的子序列。

## 再谈 LIS (Cont'd)

### 简单的做法

设  $f[i]$  表示以原序列的所有子序列中，以第  $i$  个元素结束的子序列中，最长的上升子序列的长度。

那么我们有：

$$f[i] = \max_{\substack{1 \leq j < i \\ a[j] < a[i]}} f[j] + 1$$

## 再谈 LIS (Cont'd)

### 更快的做法

跟之前一样，设  $f[i]$  表示以原序列的所有子序列中，以第  $i$  个元素结束的子序列中，最长的上升子序列的长度。

设  $g[i]$  为目前为止我们找到的所有长度为  $i$  的上升子序列的结尾元素中最小的那个。

实际上  $g[i]$  是单调增加的，否则如果  $g[i] \geq g[i+1]$ ，那么  $g[i+1]$  表示的那个长度为  $i+1$  的子序列中的倒数第二个元素显然小于  $g[i]$ 。

## 再谈 LIS (Cont'd)

### 更快的做法

跟之前一样, 设  $f[i]$  表示以原序列的所有子序列中, 以第  $i$  个元素结束的子序列中, 最长的上升子序列的长度。

设  $g[i]$  为目前为止我们找到的所有长度为  $i$  的上升子序列的结尾元素中最小的那个。

实际上  $g[i]$  是单调增加的, 否则如果  $g[i] \geq g[i+1]$ , 那么  $g[i+1]$  表示的那个长度为  $i+1$  的子序列中的倒数第二个元素显然小于  $g[i]$ 。

如果序列  $a[1], \dots, a[i-1]$  的  $g$  数组已经求完了, 我们看怎么能得到  $f[i]$  以及序列  $a[1], \dots, a[i]$  的  $g$  数组。

实际上, 我们只要在  $g$  数组中, 找到第大的小于  $a[i]$  的数。设这个数在  $g$  中的下标为  $k$ , 那么  $f[i] = k + 1$ 。

$g$  数组的变化也不大, 只需要使  $g[f[i]]$  赋值为  $a[i]$  即可。

## 再谈 LIS (Cont'd)

### 更快的做法 (Cont'd)

```
for (int i = 1; i <= n; i++) {  
    g[i] = INF;  
}  
for (int i = 1; i <= n; i++) {  
    f[i] = lower_bound (g+1, g+n+1, a[i]) - g;  
    g[f[i]] = a[i];  
}
```

## 对区间长度加以限制的最大区间和问题

### 问题

给定一个序列  $a_1, a_2, \dots, a_n$  以及  $L$  和  $R$ , 求最大的区间和, 满足  $L \leq \text{区间长度} \leq R$ 。

## 对区间长度加以限制的最大区间和问题 (Cont'd)

### 前缀和

设序列  $a_1, a_2, \dots, a_n$ ，他的前缀和序列为  $b_0, b_1, \dots, b_n$ ，其中

$$b_i = \sum_{j=1}^i a_j$$

那么  $b_i - b_j$  ( $i > j$ ) 又是什么含义呢？



## 对区间长度加以限制的最大区间和问题 (Cont'd)

### 前缀和

设序列  $a_1, a_2, \dots, a_n$ ，他的前缀和序列为  $b_0, b_1, \dots, b_n$ ，其中

$$b_i = \sum_{j=1}^i a_j$$

那么  $b_i - b_j$  ( $i > j$ ) 又是什么含义呢？  
 $a_{j+1}, a_{j+2}, \dots, a_i$  的和。

## 对区间长度加以限制的最大区间和问题 (Cont'd)

### 对问题的进一步分析

设  $f[i]$  表示以  $i$  结尾的满足限制的区间的最大值，那么有

$$f[i] = b[i] - \min_{i-R \leq j \leq i-L} b[j]$$

$[i-R, i-L]$  这个区间随着  $i$  不断增大，是在不断往右移动的。如图，假设给定序列的前缀和  $1, 2, 5, 4, 3, 6, 7$ ， $L=2$ ， $R=5$ 。那么对于  $b_6=6$  来说，取值区间里的值为  $1, 2, 5, 4$ ，显然  $1$  是最好的，但是对于  $a_7=7$  来说呢？显然此时选择  $2$  是最优的。这给我们一个启示：即使对于当前的  $i$  来说， $b_j$  比  $b_k$  差 ( $j > k$ )，我们在后面  $(i+1, i+2, \dots)$  也可能取  $b_j$  为决策点。所以  $b_j$  要留下。

## 对区间长度加以限制的最大区间和问题 (Cont'd)

### 对问题的进一步分析 (Cont'd)

但是  $b_3 = 4$  呢？对于  $i = 6$  以及  $i = 7$  而言， $b_3$  都不会被选中作为决策点，因为  $b_3$  后面有比它小的，这样即使可行区间向右移动， $b_3$  也永远也不会被作为决策点。

这给我们第二个启示：不要在决策集合中保留满足  $b_j > b_k$  且  $j < k$  的点。

# 对区间长度加以限制的最大区间和问题 (Cont'd)

## 单调队列

这样我们就有了初步的思路，维护一个随下标增加，前缀和单调上升的决策集合。

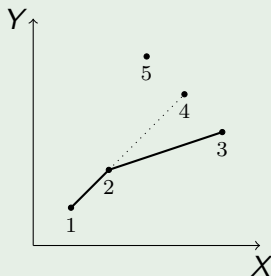
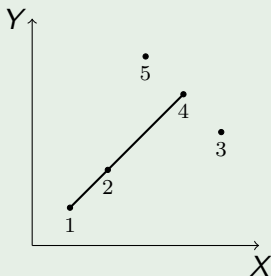


图 1: 对  $b_6 = 6$  维护的决策集合    图 2: 对  $b_7 = 7$  维护的决策集合

## 对区间长度加以限制的最大区间和问题 (Cont'd)

### 单调队列 (Cont'd)

```
for (int i = 1; i <= n; i++) {  
    int j = i-L; // 将 i-L 加入决策集合  
    while (front < rear && q[rear].value < b[j]) {  
        ---rear; // 删除队尾的前缀和更大的元素  
    }  
    ++rear;  
    q[rear].index = j;  
    q[rear].value = b[j];  
    if (q[front].index < i-R) {  
        ++front; // 删除队列中已经不在合法决策区间中的元素  
    }  
    f[i] = b[i]-q[front].value;  
}
```

# Keyboard Map

XVI Open Cup named after E.V. Pankratiev. GP of Ukraine, I

## 问题

给定  $n$  个字符，和  $m$  个按键，必须连续地把字符分配到每个按键上（类似功能机的按键上绑定的英文字母）。

每次想输入一个按键上的字符  $c$ ，就需要按这个按键若干次，次数等于编号小于等于  $c$  的字符个数。比如 'e', 'f', 'g' 在按键 2 上，那么你需要按三次按键 2，才能输入字符 'g'。

现在你知道了每个字母被输入的次数（字符  $i$  被输入  $a[i]$  次），通过设计字符的分配方案，使得按键的总次数最小。

# Keyboard Map

XVI Open Cup named after E.V. Pankratiev. GP of Ukraine, I

## 简单的做法

设  $dp[i][j]$  表示将前  $j$  个字符分配到前  $i$  个按键上时，按键的最小总次数。那么我们有

$$dp[i][j] = \min_{k=i-1}^{j-1} (dp[i-1][k] + \sum_{l=k+1}^j (l-k) \times a[l])$$

$$O(mn^3)$$

# Keyboard Map (Cont'd)

XVI Open Cup named after E.V. Pankratiev. GP of Ukraine, I

## 进一步的优化

如何快速求出  $\sum_{l=k+1}^j (l - k) a[l]$ ?



## Keyboard Map (Cont'd)

XVI Open Cup named after E.V. Pankratiev. GP of Ukraine, I

### 进一步的优化

如何快速求出  $\sum_{l=k+1}^j (l - k) a[l]$ ?

设  $sum1[i] = \sum_{j=1}^i a[j]$ ,  $sum2[i] = \sum_{j=1}^i j \times a[j]$ , 我们就有

$$\text{原式} = sum2[j] - sum2[k] - k \times (sum1[j] - sum1[k])$$

$$O(mn^2)$$

# Keyboard Map (Cont'd)

XVI Open Cup named after E.V. Pankratiev. GP of Ukraine, I

## 斜率优化

我们重新整理一下式子：

$$dp[i][j] = \min_{k=i-1}^{j-1} (dp[i-1][k] + sum2[j] - sum2[k] - k \times (sum1[j] - sum1[k]))$$

等等，不是应该在讲 1D/1D 动态规划，为什么 dp 数组是二维的？实际上，把  $dp[i][j]$  替换成  $f[j]$ ，把  $dp[i-1][j]$  替换成  $g[j]$ ，我们就有

$$f[i] = \min_{j=0}^{i-1} (g[j] + sum2[i] - sum2[j] - j \times (sum1[i] - sum1[j]))$$

# Keyboard Map (Cont'd)

XVI Open Cup named after E.V. Pankratiev. GP of Ukraine, I

## 斜率优化 (Cont'd)

让我们来看看，如果对于  $f[i]$  来说， $j$  比  $k$  是个更优的决策点会发生什么？设  $j > k$ 。

$$g[j] + \text{sum2}[i] - \text{sum2}[j] - j \times (\text{sum1}[i] - \text{sum1}[j]) < g[k] + \text{sum2}[i] - \text{sum2}[k] - k \times (\text{sum1}[i] - \text{sum1}[k]) \quad (1)$$

$$\frac{(g[j] - \text{sum2}[j] + j \times \text{sum1}[j]) - (g[k] - \text{sum2}[k] + k \times \text{sum1}[k])}{j - k} < \text{sum1}[i] \times (j - k) \quad (2)$$

$$\frac{(g[j] - \text{sum2}[j] + j \times \text{sum1}[j]) - (g[k] - \text{sum2}[k] + k \times \text{sum1}[k])}{j - k} < \text{sum1}[i] \quad (3)$$

$$\frac{y[j] - y[k]}{x[j] - x[k]} < \text{sum1}[i] \quad (4)$$

# Keyboard Map (Cont'd)

XVI Open Cup named after E.V. Pankratiev. GP of Ukraine, I

## 斜率优化 (Cont'd)

把每个决策点抽象成一个平面上的点  $(x[i], y[i])$ , 那么对于  $i$ , 当  $j$  优于  $k$  且  $j > k$  时, 满足点  $i$  与点  $j$  连接形成的线段的斜率  $k < \text{sum1}[i]$ 。而且值得注意的是,  $\text{sum1}[i]$  随着  $i$  增加, 是不断增加的。

而且实际上, 我们只需要维护一个下凸壳。

# Keyboard Map (Cont'd)

XVI Open Cup named after E.V. Pankratiev. GP of Ukraine, I

## 斜率优化 (Cont'd)

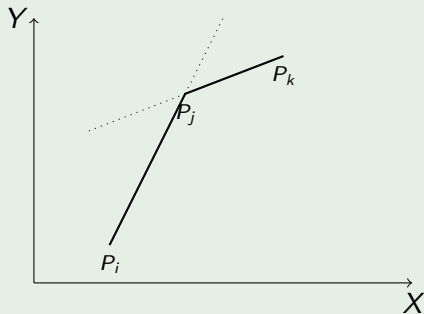


图 3: 一种情况

由图3,  $P_j$  永远不可能作为决策点。所以维护一个下凸壳即可。

# Keyboard Map (Cont'd)

XVI Open Cup named after E.V. Pankratiev. GP of Ukraine, I

## 斜率优化 (Cont'd)

- ① 每次向队尾添加元素时，通过删除队尾元素，维护双端队列下凸壳。
- ② 当队首元素与次队首元素的斜率小于等于  $sum1[i]$  时，弹出队首元素。
- ③ 每次选择队首元素作为决策点

# 石子合并

## 题目

设有  $n$  堆石子排成一排，其编号为  $1, 2, 3, \dots, n$  ( $n \leq 500$ )。每堆石子有一定的数量。现要将  $n$  堆石子合并成为一堆。归并的过程只能每次将相邻的两堆石子堆成一堆，合并的代价是两堆石子的石子数量和。这样  $n$  堆石子经过  $n-1$  次归并后成为一堆。找出将  $n$  堆石子合并成一堆石子的最小总代价。

## 石子合并 (Cont'd)

### 简单的解法

设  $dp[i][j]$  表示第  $i$  堆到第  $j$  堆石子合并成一堆所需要的最小代价，那么我们有：

$$dp[i][j] = \min_{i < k \leq j} dp[i][k-1] + dp[k][j] + w[i][j]$$

其中  $w[i][j]$  表示第  $i$  堆到第  $j$  堆石子的石子数目和。  
 $O(n^3)$



## 石子合并 (Cont'd)

### 更快的解法

设  $f[i][j]$  为另  $dp[i][j]$  的最优决策点。可以证明:

$$f[i][j-1] \leq f[i][j] \leq f[i+1][j]$$

这样, 我们每次枚举  $k$  时, 不需要从  $i+1$  枚举到  $j$ , 只需要从  $f[i][j-1]$  枚举到  $f[i+1][j]$ 。

## 石子合并 (Cont'd)

### 更快的解法 (Cont'd)

复杂度分析:

$$T(n) = \sum_{i=1}^n \sum_{j=i}^n f[i+1][j] - f[i][j-1] \quad (5)$$

$$= \sum_{k=2}^n \sum_{j-i+1=k} f[i+1][j] - f[i][j-1] \quad (6)$$

$$= \sum_{k=2}^n f[n-k+2][n] - f[1][k-1] \quad (7)$$

$$\leq \sum_{k=2}^n n \quad (8)$$

$$= O(n^2) \quad (9)$$

# 石子合并 (Cont'd)

## 定义 1

**四边形不等式** 当决策代价函数  $w$  满足

$w[a][c] + w[b][d] \leq w[a][d] + w[b][c]$  ( $a \leq b \leq c \leq d$ ) 时, 称  $w$  满足四边形不等式。

## 定义 2

**关于区间包含关系单调** 如果函数  $w$  满足  $w[i][j] \leq w[i', j']$  ( $i' \leq i \leq j \leq j'$ ), 则称  $w$  关于区间包含关系单调。

## 石子合并 (Cont'd)

### 定理 1

如果  $w$  同时满足四边形不等式和区间单调关系, 则  $dp$  也满足四边形不等式。

### 定理 2

定理1满足时,  $f[i][j-1] \leq f[i][j] \leq f[i+1][j]$ 。

### 证明

定理1与定理2的证明详见 [3]

# Do Geese See God?

Asia Tsukuba Regional Contest 2015 G

## 题目

给定一个字符串，添加最少的字符使得这个字符串变成回文串。

# Do Geese See God?

Asia Tsukuba Regional Contest 2015 G

## 题目

给定一个字符串，添加最少的字符使得这个字符串变成回文串。  
还没完。

在所有这样的最短的回文串中，找到字典序第  $k$  小的并输出。字符串长度  $\leq 2000$ ， $k \leq 10^{18}$ 。

# Do Geese See God? (Cont'd)

Asia Tsukuba Regional Contest 2015 G

## 第一个问题

给定一个字符串，添加最少的字符，使得字符串变为回文串。

# Do Geese See God? (Cont'd)

Asia Tsukuba Regional Contest 2015 G

## 第一个问题

给定一个字符串，添加最少的字符，使得字符串变为回文串。  
设  $f[i][j]$  表示子串  $S[i \dots j]$  最少需要添加多少个字符变成回文串。  
我们有

$$f[i][j] = \begin{cases} f[i+1][j-1], & S[i] = S[j] \\ \min(f[i+1][j], f[i][j-1]) + 1, & S[i] \neq S[j] \end{cases}$$



# Do Geese See God? (Cont'd)

Asia Tsukuba Regional Contest 2015 G

## 第二个问题

给定一个字符串，问有多少种添加字符的方式，满足添加的字符数最少，而且使得字符串变为回文串。

# Do Geese See God? (Cont'd)

Asia Tsukuba Regional Contest 2015 G

## 第二个问题

给定一个字符串，问有多少种添加字符的方式，满足添加的字符数最少，而且使得字符串变为回文串。

这个问题跟第一个问题类似。设  $g[i][j]$  为子串  $S[i \dots j]$  的方式数目。我们有

$$f[i][j] = \begin{cases} g[i+1][j-1], & S[i] = S[j] \\ g[i+1][j], & S[i] \neq S[j] \wedge f[i+1][j]+1 = f[i][j] \\ g[i][j-1], & S[i] \neq S[j] \wedge f[i][j-1]+1 = f[i][j] \\ g[i+1][j] + f[i][j-1], & S[i] \neq S[j] \wedge f[i+1][j]+1 \neq f[i][j] \wedge f[i][j-1]+1 \neq f[i][j] \end{cases}$$

# Do Geese See God? (Cont'd)

Asia Tsukuba Regional Contest 2015 G

## 原问题

解决了前面两个问题，我们就可以计算原问题了。

我们定义  $T(i, j, k)$  表示子串  $S[i \dots j]$  的最短生成回文串中字典序第  $k$  小的串。 $T(1, n, k)$  就是答案。但  $T(i, j, k)$  应该怎么求呢？我们就只求第一个字符 (同时也是最后一个)。

# Do Geese See God? (Cont'd)

Asia Tsukuba Regional Contest 2015 G

## 原问题

解决了前面两个问题，我们就可以计算原问题了。

我们定义  $T(i, j, k)$  表示子串  $S[i \dots j]$  的最短生成回文串中字典序第  $k$  小的串。 $T(1, n, k)$  就是答案。但  $T(i, j, k)$  应该怎么求呢？我们就只求第一个字符 (同时也是最后一个)。

首先，可以肯定的是  $T(i, j, k)$  的第一个字符要么是  $S[i]$ ，要么是  $S[j]$ 。

什么时候选择  $S[i]$ ？什么时候选择  $S[j]$ ？选择了  $S[i]$  或者  $S[j]$  之后，子问题变成了什么样子？

# Do Geese See God? (Cont'd)

Asia Tsukuba Regional Contest 2015 G

## 原问题

- 1  $S[i] = S[j]$ : 第一个字符肯定是  $S[i]$ , 子问题  $T(i+1, j-1, k)$ 。

# Do Geese See God? (Cont'd)

Asia Tsukuba Regional Contest 2015 G

## 原问题

- ①  $S[i] = S[j]$ : 第一个字符肯定是  $S[i]$ , 子问题  $T(i+1, j-1, k)$ 。
- ②  $S[i] < S[j]$ :

# Do Geese See God? (Cont'd)

Asia Tsukuba Regional Contest 2015 G

## 原问题

- ①  $S[i] = S[j]$ : 第一个字符肯定是  $S[i]$ , 子问题  $T(i+1, j-1, k)$ 。
- ②  $S[i] < S[j]$ :
  - ①  $f[i+1][j] + 1 = f[i][j]$

# Do Geese See God? (Cont'd)

Asia Tsukuba Regional Contest 2015 G

## 原问题

- ①  $S[i] = S[j]$ : 第一个字符肯定是  $S[i]$ , 子问题  $T(i+1, j-1, k)$ 。
- ②  $S[i] < S[j]$ :
  - ①  $f[i+1][j] + 1 = f[i][j]$ 
    - ①  $g[i+1][j] < k$ : 此时如果我们选  $S[i]$ , 那么会凑不出第  $k$  个, 所以选  $S[j]$ 。子问题变成  $T(i, j-1, k - g[i+1][j])$ 。



# Do Geese See God? (Cont'd)

Asia Tsukuba Regional Contest 2015 G

## 原问题

- ①  $S[i] = S[j]$ : 第一个字符肯定是  $S[i]$ , 子问题  $T(i+1, j-1, k)$ 。
- ②  $S[i] < S[j]$ :
  - ①  $f[i+1][j] + 1 = f[i][j]$ 
    - ①  $g[i+1][j] < k$ : 此时如果我们选  $S[i]$ , 那么会凑不出第  $k$  个, 所以选  $S[j]$ 。子问题变成  $T(i, j-1, k - g[i+1][j])$ 。
    - ②  $g[i+1][j] \geq k$ : 选  $S[i]$  即可。

# Do Geese See God? (Cont'd)

Asia Tsukuba Regional Contest 2015 G

## 原问题

- ①  $S[i] = S[j]$ : 第一个字符肯定是  $S[i]$ , 子问题  $T(i+1, j-1, k)$ 。
- ②  $S[i] < S[j]$ :
  - ①  $f[i+1][j] + 1 = f[i][j]$ 
    - ①  $g[i+1][j] < k$ : 此时如果我们选  $S[i]$ , 那么会凑不出第  $k$  个, 所以选  $S[j]$ 。子问题变成  $T(i, j-1, k - g[i+1][j])$ 。
    - ②  $g[i+1][j] \geq k$ : 选  $S[i]$  即可。
  - ②  $f[i+1][j] + 1 \neq f[i][j]$ : 此时选  $S[j]$ 。





# Do Geese See God? (Cont'd)

Asia Tsukuba Regional Contest 2015 G

## 原问题

- ①  $S[i] = S[j]$ : 第一个字符肯定是  $S[i]$ , 子问题  $T(i+1, j-1, k)$ 。
- ②  $S[i] < S[j]$ :
  - ①  $f[i+1][j] + 1 = f[i][j]$ 
    - ①  $g[i+1][j] < k$ : 此时如果我们选  $S[i]$ , 那么会凑不出第  $k$  个, 所以选  $S[j]$ 。子问题变成  $T(i, j-1, k - g[i+1][j])$ 。
    - ②  $g[i+1][j] \geq k$ : 选  $S[i]$  即可。
  - ②  $f[i+1][j] + 1 \neq f[i][j]$ : 此时选  $S[j]$ 。
- ③  $S[i] > S[j]$ : 类似

## 参考资料

-  汪一宁. 1D/1D 动态规划优化初步.
-  周源. 浅谈数形结合思想在信息学竞赛中的应用.
-  赵爽. 动态规划加速原理之四边形不等式.
-  刘汝佳, 黄亮. 算法艺术与信息学竞赛 [M]. 清华大学出版社, 2004.