

# 最长上升子序列 (LIS)

最长不下降及下降子序列请自行考虑关系运算符变化

## 问题描述

给定 (非负) 整数序列  $\{a_1, a_2, \dots, a_n\}$ , 求其一个子序列, 使得元素呈升序, 且子序列长度最长, 输出长度 (输出序列)

例:  $\{1, 6, 4, 2, 3, 9, 8\}$  的 LIS 为  $\{1, 2, 3, 9\}$  或  $\{1, 2, 3, 8\}$

## 解题思路

### Solution1 : DP

设  $dp[i]$  表示以  $a[i]$  为结尾的 LIS 长度, 初始化为 1 (子序列至少包含自己)

为保证升序, 状态转移需从  $a[i]$  之前比  $a[i]$  小的元素转移而来。你说, 是不是啊

则状态转移方程  $dp[i] = \max(dp[i], dp[j] + 1)$  其中  $(j < i) \wedge a[j] < a[i]$  复杂度  $O(n^2)$  你说, 是不是啊\*2

如需输出任意一解, 可设置一追踪标记  $tr$  与  $ans$  同步更新为当前位置  $i$

设  $ans$  在  $dp[k]$  处取到, 则对于  $i < k$  都有  $dp[i] < dp[k]$  (废话), 且至少有一个序列满足  $dp$  数组以 1 为单位递增 你说, 是不是啊\*3

如  $\{1, 6, 4, 2, 3, 9, 8\}$  的  $dp$  数组元素为  $\{1, 2, 2, 2, 3, 4, 4\}$

那么只需从  $tr$  遍历到 1, 遇到  $a[tr] == ans$  则  $a[tr]$  压栈, 同时  $ans -= 1$  最后弹栈输出。你说, 是不是啊\*4

```
for(int i = 1; i <= n; i++) dp[i] = 1;
int ans = -1, tr;
for(int i = 1; i <= n; ++i){
    for(int j = 1; j < i; ++j){
        if(a[j] < a[i]) dp[i] = max(dp[i], dp[j] + 1);
        ans = max(ans, dp[i]);
        if(dp[i] >= ans) tr = i, ans = dp[i]; /*也可不加等号*/
    }
}

printf("%d\n", ans);
while(tr){
    if(dp[tr] == ans) z.push(a[tr]), --ans;
    --tr;
}
```

```
while(!z.empty()) printf("%d ", z.top()), z.pop();
```

## Solution 2 : 二分+贪心

因为贪心所以某些条件下会受限制。你说，是... 草，不说了

设  $f[i]$  表示长度为  $i$  的 LIS 末尾元素的最小值，这个数组一定是单调不下降的。

置  $f[1] = a[1]$  取  $cnt = 1$  表示 LIS 的长度,遍历数组  $a$  :

$a[i] > f[cnt]$  则  $f[++cnt] = a[i]$

$a[i] < f[cnt]$  找到  $f$  中第一个大于  $a[i]$  的位置并替换为  $a[i]$

这样做的替换并不会影响当前 LIS 的长度变化，只改变了数组  $f$  中某个位置的值并让其尽可能小，以便于不断获得更优解，所以不影响 LIS 长度的正确性，但不能用来追踪解，可想而知。

因为数组  $f$  有序，所以可用二分，复杂度  $O(n \log n)$

```
f[1] = a[1];
int cnt = 1;
for(int i = 2; i <= n; ++i){
    if(a[i] > f[cnt]) f[++cnt] = a[i];
    else if(a[i] < f[cnt]){
        int j = upper_bound(f + 1, f + cnt + 1, a[i]) - f; /*得到数组f中第一个大于a[i]的位置,也可手写二分*/
        f[j] = a[i];
    }
}
```

最终  $f[cnt]$  即为答案

## Solution 3 : 转LCS

顾名思义，将数组  $a$  排序得到新数组  $b$  问题转化为求  $a, b$  的最长公共子序列，长度分别为  $n, m$

此处只给出其 DP 状态转移方程，以  $dp[i][j]$  表示  $a[i]$  和  $b[j]$  的 LCS 长度，状态转移方程  $dp[i][j] =$

```
0 ;(i == 0 || j == 0)
dp[i-1][j-1] + 1 ;(a[i] == b[j])
max{dp[i][j-1], dp[i-1][j]} ;(a[i] != b[j])
```

解的追踪（不予详解）：设一标记数组  $tr[n][m]$  标记依据  $dp$  方程设立，最后从  $tr[n][m]$  回溯寻找  $a[i] == b[j]$  的标记，压栈即可

代码

```
for(int i = 1; i <= n; ++i){
```

```
for(int j = 1; j <= m; ++j){
    if(a[i] == b[j]){
        dp[i][j] = dp[i-1][j-1] + 1;
        tr[i][j] = 3;
    }
    else{
        dp[i][j] = max(dp[i-1][j], dp[i][j-1]);
        tr[i][j] = dp[i-1][j] > dp[i][j-1] ? 1 : 2;
    }
}
}

int x = n, y = m, t;
int dx[4] = {INF, -1, 0, -1};
int dy[4] = {INF, 0, -1, -1};
stack <int>z;
while(tr[x][y]){
    if(tr[x][y] == 3) z.push(a[x]);
    t = tr[x][y];
    x += dx[t];
    y += dy[t];
}
while(!z.empty()){
    printf("%d ", z.top());
    z.pop();
}
```

答案即为  $dp[n][m]$

## Solution 4 : 据说可以用树状数组 ?

吓得我赶紧去学 先前的[树状数组](#)有待完善...

From: <https://wiki.cvbbacm.com/> - CVBB ACM Team

Permanent link: <https://wiki.cvbbacm.com/doku.php?id=2020-2021:teams:manespace:%E6%9C%80%E9%95%BF%E4%B8%8A%E5%8D%87%E5%AD%90%E5%BA%8F%E5%88%97&rev=1590985145>

Last update: 2020/06/01 12:19