子序列相关问题

Tony_Wong

November 14, 2019

目录

0 几个概念

设 $A = \{a_1, a_2, a_3, ..., a_n\}$
子序列:$B = \{a_{k_1}, a_{k_2}, a_{k_3}, ..., a_{k_p}\}$ 其中 $k_1 < k_2 < .. < k_p$
子串: $B = \{a_i, a_{i+1}, a_{i+2}, ..., a_j\}$ 即 连续子序列

1 最长上升子序列 (LIS)

1.1 DP 做法 $O(n^2)$

状态转移方程:$f[i] = \max\limits_{1 \le j < i \&\& a_j < a_i} \{f[j] + 1\}$
初值: $f[0] = 0$
结果: $\max\limits_{i \le i \le n} f[i]$

1.2 树状数组 $O(n \log n)$

对于原序列每个元素, 它有一个下标和一个权值, 最长上升子序列实质就是求**最多有多少元素它们的下标和权值都单调递增**

于是我们将 a 数组的每一个元素先记下它现在的下标, 然后按照权值从小到大排序。接着我们按从小到大的顺序枚举 a 数组,(此时权值已经默认单调递增了) 我们的转移也就变成从之前的标号比它小的状态转移过来, 这个我们只需要建立一个**以编号为下标维护长度的最大值的树状数组**即可, 枚举

a 数组时按元素的序号找到它之前序号比他小的长度最大的状态更新, 然后将它也加入树状数组中

```
struct Node {
    int val, id;
    bool operator < (const Node& a) const {
        if (val == a.val) return id > a.id;
        return val < a.val;
    }
}a[maxn];

void add(int x, int k) {
    for (; x <= n; x += x & -x) t[x] = max(t[x], k);
}
int ask(int x) {
    int res = 0;
    for (; x; x -= x & -x) res = max(res, t[x]);
    return res;
}

for (int i = 1; i <= n; ++i) {
    a[i].v = read(); a[i].id = i;
}
sort(a + 1, a + 1 + n);
for (int i = 1; i <= n; ++i) {
    add(a[i].id, ask(a[i].id) + 1);
}

Ans: ask(n)
```

1.3 贪心二分 $O(n \log n)$

```
b[1] = a[1];
int ans = 1;
```

```
for (int i = 2; i <= n; i++) {
    if (a[i] > b[len]) b[++ans] = a[i];
    else {
        int j = lower_bound(b + 1, b + len + 1, a[i]) - d;
        d[j] = a[i];
    }
}
```

2 最长不下降子序列

2.1 DP 做法 $O(n^2)$

把判断的小于号改为**小于等于号**

状态转移方程:$f[i] = \max\limits_{1\le j<i \& \& a_j \le a_i} \{f[j]+1\}$

初值: $f[0] = 0$

结果: $\max\limits_{i\le i\le n} f[i]$

2.2 贪心二分 $O(n \log n)$

大于改大于等于,lower 改 upper

```
b[1] = a[1];
int ans = 1;
for (int i = 2; i <= n; i++) {
    if (a[i] >= b[len]) b[++ans] = a[i];
    else {
        int j = upper_bound(b + 1, b + len + 1, a[i]) - d;
        d[j] = a[i];
    }
}
```

3 最长上升子串

3.1 DP 做法 $O(n^2)$

在 DP 时判断一下 `i-j==1`

3.2 扫描 $O(n)$

```
int res = 1, ans = 1;
for (int i = 2; i <= n; ++i) {
    if (a[i] > a[i - 1]) res++;
    else res = 1;
    ans = max(ans, res);
}
```

4 最长公共子序列 (LCS)

4.1 DP $O(n^2)$

$f[i][j]$ 表示 $a_{1\sim i}$ 与 $b_{1\sim j}$ 的 LCS 长度

状态转移方程: 如果 $a[i] = b[j]$ 则 $f[i][j] = max(f[i][j], f[i-1][j-1]+1)$

否则 $f[i][j] = max(f[i-1][j], f[i][j-1])$

边界:$f[i][0] = 0, f[0][j] = 0$

结果:$f[n][m]$

5 最长公共上升子序列 (LCIS)

5.1 三重循环 DP $O(n^3)$

```
for (int i = 1; i <= n; ++i) {
    for (int j = 1; j <= m; ++j) {
        if (a[i] == b[j]) {
            for (int k = 0; k < j; ++k) {
                if (b[k] < a[i]) {
                    f[i][j] = max(f[i][j], f[i - 1][k] + 1);
```

```
                }
            }
        } else {
            f[i][j] = f[i - 1][j];
        }
    }
}

Ans: f[n][m]
```

5.2 二重循环 DP $O(n^2)$

```
for (int i = 1; i <= n; ++i) {
    int val = 0;
    if (b[0] < a[i]) val = f[i - 1][0];
    for (int j = 1; j <= m; ++j) {
        if (a[i] == b[j]) f[i][j] = val + 1;
        else f[i][j] = f[i - 1][j];
        if (b[j] < a[i]) val = max(val, f[i - 1][j]);
    }
}
```

6 最长公共子串

6.1 DP $O(n^2)$

```
for (int i = 1; i <= n; i++) {
    for (int j = 1; j <= m; j++) {
        if(a[i - 1] == b[j - 1]) {
            f[i][j] = f[i -1][j - 1] + 1;
            res = max(res, dp[i][j]);
        } else {
            dp[i][j] = 0;
```

```
        }
    }
}
```

7 最长公共前缀

将字符串插入 $Trie$ 树, 然后在树上查询 LCA 的 dep

8 三元上升子序列

$Luogu\ P1637$ **注意要离散化**

```
for (int i = 1; i <= n; ++i) {
    add1(a[i], 1);
    cntl[i] = ask1(a[i] - 1);
}
for (int i = n; i >= 1; --i) {
    add2(a[i], 1);
    cntr[i] = n - i - ask2(a[i]) + 1;
}
for (int i = 2; i < n; ++i) {
    ans += cntl[i] * cntr[i];
}
```

9 最小 m 段和

```
for (int i = 1; i <= n; ++i) {
    sum[i] = sum[i - 1] + a[i];
}
for (int i = 1; i <= n; ++i) {
    dp[i][i] = max(dp[i - 1][i - 1], a[i]);
    for (int j = 1; j <= min(i, k); ++j) {
        for (int m = j - 1; m < i; ++m) {
```

```
            dp[i][j] = min(dp[i][j], max(sum[i] - sum[m],
dp[m][j - 1]));
        }
    }
}
```

```
Ans: min(dp[i][k]) k<=i<=n
```

10 最大子段和

10.1 暴力 $O(n^3)/O(n^2)$

10.2 DP $O(n)$

$f[i]$ 为 $a_{1\sim i}$ 从 a_i 向前延伸得到的最大子段和
状态转移方程:$f[i] = max(f[i-1] + a[i], a[i])$ 且 $2 \le i \le n$
边界:$f[1] = max(0, a[1])$
结果:$\max\limits_{1 \le i \le n} f[i]$

11 逆序对

注意要**离散化**

```
for (int i = n; i; --i) {
    ans += ask(a[i] - 1);
    add(a[i], 1);
}
```

11.1 二维偏序

注意 FG 要**放在一起离散化**

```
for (int i = n; i; --i) {
    ans += ask(F[i] - 1);
    add(G[i], 1);
}
```

附. 离散化

将原数组复制一份, 再排序 (sort) 去重 (unique)

```
for (int i = 1; i <= n; ++i) old[i] = a[i];
sort(old + 1, old + 1 + n);
int len = unique(old + 1, old + 1 + n) - old - 1;
for (int i = 1; i <= n; ++i)
    a[i] = lower_bound(old + 1, old + 1 + len, a[i]) - old;
```

操作之后的数组为 `a[i]`, 原数为 `old[a[i]]` 即 `a[i]` 替换了 `old[a[i]]`