最长公共子序列

Longest Common Subsequence

$O(nm)$

Idea：`dp[i][j]` 表示 A 的前 i 项与 B 的前 j 项的 LCS 长度，则：

$$dp[i][j]=\begin{cases}0 & i=0\text{ or }j=0\\ dp[i-1][j-1]+1 & a[i]=b[i]\\ \max\{dp[i][j-1],dp[i-1][j]\} & a[i]\neq b[i]\end{cases}$$

Code：

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

$O(m^2+|C|n)$

Idea：交换朴素做法的答案和第一维，即：`dp[i][j]` 表示与 B 的前 i 项形成的 LCS 长度为 j 的 A 的最短前缀长度。则：

$$dp[i][j]=\min\{nxt[dp[i-1][j-1]][b[i]],dp[i-1][j]\}$$

其中 `nxt[i][x]` 表示 A 中第 i 个位置之后出现的第一个 x 字符的位置，先预处理出来，复杂度 $O(|C|n)$，其中 $|C|$ 表示字符集大小。

适用情形：字符集 $|C|$ 较小（如字母集），m 较小。

Code：

```cpp
int dp[N][N], nxt[N][30], pre[30];
int LCS(){
    memset(nxt, 0x3f, sizeof nxt);
    memset(pre, 0, sizeof pre);
    for(int i = 1; i <= n; i++){
        for(int k = pre[a[i]-'a']; k < i; k++)
            nxt[k][a[i]-'a'] = i;
        pre[a[i]-'a'] = i;
    }
    for(int i = 0; i <= m; i++)
        for(int j = 0; j <= m; j++)
            dp[i][j] = j == 0 ? 0 : n + 1;
    for(int i = 1; i <= m; i++)
        for(int j = 1; j <= i; j++)
            dp[i][j] = min(nxt[dp[i-1][j-1]][b[i]-'a'], dp[i-1][j]);
    int res = 0;
    for(int i = 1; i <= m; i++)
        if(dp[m][i] <= n)   res = i;
    return res;
}
```

$O(x\lg x)$

Idea：将 A 数组中每个数依次编号，将 B 数组中的每个数换成该数对应编号的逆序，得到新的数组，设长度为 x，则对其进行 $O(x\lg x)$ 的 LIS 求解即可。

ATT：x 最坏可以达到 nm，所以此方法通常用于 A,B 是排列的情况。

Code（A,B 均是 n 的排列）：

```cpp
int dp[N];
int LIS(int a[]){
    // 长度为 i 的上升子序列的最小末尾数值是 dp[i]
    int len = 0;
    for(int i = 1; i <= n; i++){
        int p = lower_bound(dp+1, dp+len+1, a[i]) - dp;
        if(p == len + 1)    len++;
        dp[p] = a[i];
    }
    return len;
}

int main(){
    // ...
    for(int i = 1; i <= n; i++) scanf("%d", &a[i]), t[a[i]] = i;
    for(int i = 1; i <= n; i++) scanf("%d", &b[i]), b[i] = t[b[i]];
    // ...
}
```