LCS and LIS

Longest Common Subsequence

For given two sequences X and Y, a sequence Z is a common subsequence of X and Y if Z is a subsequence of both X and Y.

Write a program which finds the length of LCS of given two sequences X and Y. The sequence consists of alphabetical characters.

Source: AOJ ALDS1_10_C

Example

a b c b d a b

b d c a b a

Preparation: Substring

- Subsequence $\supset$ Substring
- Substring from i to j of S: $S[i]S[i+1]\ldots S[j-1]S[j]$
- Substring is continuous.

DP's solution of LCS

- Def.
    - $dp[i][j] = \text{Length of LCS of } X_{i-1} \text{ and } Y_{j-1}$

Here, S_i indicates the substring of S from 0 to i

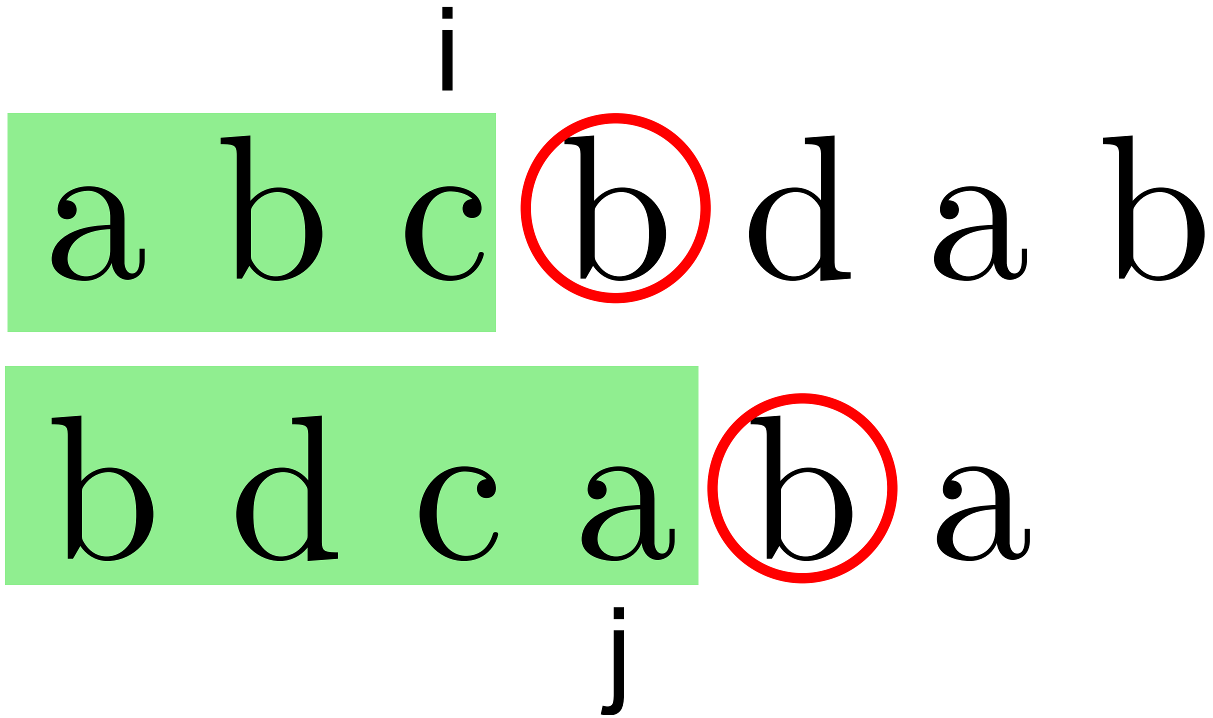
- Init.
    - $dp[i][j] = 0$

- Trans.
    - $dp[i+1][j+1] = dp[i][j] + 1 \text{ if } X[i] == Y[j]$
    - $dp[i+1][j+1] = \max(dp[i+1][j], dp[i][j+1])$

Ans.
$dp[|X|][|Y|]$

Supplement: Image of transition

- $dp[i+1][j+1] = dp[i][j] + 1 \text{ if } X[i] == Y[j]$



- $dp[i+1][j+1] = \max(dp[i+1][j], dp[i][j+1])$

i

a b c b d a b

b d c a b a

j

```cpp
for (int i = 0; i < X.size(); i++) {
  for (int j = 0; j < Y.size(); j++) {
    if (X[i] == Y[j]) dp[i + 1][j + 1] = dp[i][j] + 1;
    else dp[i + 1][j + 1] = max(dp[i + 1][j], dp[i][j + 1]);
  }
}
// the answer is dp[X.size()][Y.size()]
```

Longest Increasing Subsequence

Longest Increasing Subsequence

For a given sequence $A = a_0, a_1, \dots, a_{n-1}$, find the length of the longest increasing subsequnece (LIS) in A.

An increasing subsequence of A is defined by a subsequence $a_{i0}, a_{i1}, \dots a_{ik}$ where $0 \leq i_0 < i_1 < \cdots a_{ik} < n$ and $a_{i0} < a_{i1} < \cdots < a_{ik}$

Source: AOJ DPL_1_D

Example

$$5 \quad 1 \quad 3 \quad 2 \quad 4$$

- $1 < 2 < 4$
- $1 < 3 < 4$ is also correct.

Naive DP solution

- Def.
  - $dp[i] = \text{Length of LCS in } a[0], a[1], \ldots, a[i]$

The time complexity is $O(n^2)$ (I will skip the detail).

Good solution

- Def.
  - $dp[i] = \text{minimum number of tail of LIS with length } i + 1$
- Init
  - $dp[i] = \text{INF}$

Idea of transition

- It is good for tail to be small because of later choice
- Using binary search
  ex. 5 1 3 2 4 (DP table is right.);

0	1	2	3	4	5
INF	INF	INF	INF	INF	INF

0	1	2	3	4	5
5	INF	INF	INF	INF	INF

5

0	1	2	3	4	5
1	INF	INF	INF	INF	INF

1

0	1	2	3	4	5
1	3	INF	INF	INF	INF

$? \ 3$

0	1	2	3	4	5
1	2	INF	INF	INF	INF

$? \ 2$

0	1	2	3	4	5
1	2	4	INF	INF	INF

$? \ ? \ 4$

- Trans.
  $dp[k] = a[i]$
  Here, k is the index of the first element such that $dp[k] \geq a[i]$.

We can use lower_bound to get k

```
//When dp is a vector
k = lower_bound(dp.begin(), dp.end(), a[i]) - dp.begin();

//When dp is an array
k = lower_bound(dp, dp + n, a[i]) - dp;
```

- Ans.

  Maximum of $i+1$ such that $dp[i] < \mathrm{INF}$

The time complexity is $O(n \log n)$

```cpp
int dp[110000];
for (int i = 0; i < 110000; i++) {
  dp[i] = INF;
}

for (int i = 0; i < n; i++) {
  int idx = lower_bound(dp, dp + n, a[i]) - dp;
  dp[idx] = a[i];
}

int ans;
for (ans = 0; ans < n; ans++) {
  if (dp[ans] >= INF) break;
}
cout << ans << endl;
```