

#중급반 7주차

# LIS, DP 역추적

<LIS, DP track>

<sup>2p</sup> / <LIS>

<sup>18p</sup> / <DP 역추적>

<sup>22p</sup> / <연습 문제>

## #가장 긴 증가하는 부분수열

<LIS>

가장 긴 증가하는 부분수열, Longest Increasing Subsequence(LIS)는 대표적인 DP 문제 중 하나입니다.

수열	LIS
1 2 1 3 2 5	1 2 3 5
3 2 1	1 or 2 or 3
1 4 2 1 6	1 2 6

부분수열(Subsequence): 수열에서 0개 이상의 원소를 삭제한 수열을 말합니다.

부분문자열(SubString): 문자열에서 앞과 뒤에서 0개 이상의 연속한 원소를 삭제한 문자열을 말합니다.

## #가장 긴 증가하는 부분수열

<LIS>

X로 끝나는 증가수열은, X보다 작은 수로 끝나는 증가수열에 X를 덧붙여서 만들 수 있습니다.

DP[i] : i번째 수로 끝나는 LIS의 길이로 정의해, 위 논리를 이용해서 DP 배열을 직접 채워봅시다.

i	1	2	3	4	5	6
A[i]	1	3	2	4	3	5
DP[i]						

## #가장 긴 증가하는 부분수열

<LIS>

점화식을 찾아볼까요?

먼저,  $DP[i]$ 는 '어떠한 조건'을 만족하는  $j$ 에 대해  $DP[j] + 1$ 의 최대값입니다.

$$DP[i] = \max(DP[j] + 1)$$

$A[j]$ 로 끝나는 LIS에  $A[i]$ 를 덧붙이는 것이니,  $j < i$ 이어야 합니다.

$$DP[i] = \max_{j < i} (DP[j] + 1)$$

증가수열이니,  $A[j] < A[i]$ 이어야 합니다.

$$DP[i] = \max_{j < i \ \& \ A[j] < A[i]} (DP[j] + 1)$$

## #가장 긴 증가하는 부분수열

<LIS>

$$DP[i] = \max_{j < i \ \& \ A[j] < A[i]} (DP[j] + 1)$$

코드로 옮겨봅시다.

j < i 조건이 있으니, i를 1부터 N까지 증가하는 순  
으로 계산해야 부분문제들을 올바른 순서로 해결  
할 수 있습니다.

```
for (int i = 1; i <= N; i++) {  
    DP[i] = 1;  
    for (int j = 1; j < i; j++) {  
        if (A[j] < A[i]) {  
            DP[i] = max(DP[i], DP[j] + 1);  
        }  
    }  
}  
  
int ans = 0;  
for (int i = 1; i <= N; i++) {  
    ans = max(ans, DP[i]);  
}
```

## #가장 긴 증가하는 부분수열

<LIS>

### 2 가장 긴 증가하는 부분수열 (BOJ #11053)

#### <문제 설명>

- 길이  $N$ 의 수열이 주어진다.
- 주어진 수열의 가장 긴 증가하는 부분수열의 길이를 구해라.

#### <제약 조건>

- $1 \leq N \leq 1,000$
- $1 \leq A_i \leq 1,000$

## #가장 긴 증가하는 부분수열

<LIS>

LIS 문제를 해결하는 DP를 알아보았습니다. 시간복잡도는  $O(n^2)$ 입니다.  
이번에는  $O(n \log n)$  방법을 소개하겠습니다.

## #가장 긴 증가하는 부분수열

<LIS>

하나의 배열을 더 정의합니다.

$B[i] = \min_{DP[j]=i} (A[j])$  풀어서 쓰면 LIS의 길이가 i인 LIS들 중, 가장 작은 끝나는 수를 의미합니다.

초기값은 모두 inf로 둡시다.

i	1	2	3	4	5	6
A[i]	1	3	2	4	3	5
DP[i]						
B[i]						



## #가장 긴 증가하는 부분수열

<LIS>

점화식을 찾아볼까요?

먼저,  $DP[i]$ 는 ‘어떠한 조건’을 만족하는  $j$ 에 대해  $j + 1$ 의 최대값입니다.

$$DP[i] = \max(j + 1)$$

$B[j]$ 가  $A[i]$ 보다 작아야  $B[j]$ 로 끝나는 수열 뒤에  $A[i]$ 를 붙일 수 있습니다.

$$DP[i] = \max_{B[j] < A[i]} (j + 1)$$

## #가장 긴 증가하는 부분수열

<LIS>

코드로 옮겨봅시다.

여전히  $O(n^2)$ 입니다. 하지만 이분탐색을 활용해  $O(n \log n)$ 인 색칠된 부분을  $O(\log n)$ 으로 바꿀 수 있습니다.

```
for (int i = 1; i <= N; i++) {
    B[i] = inf;
}

for (int i = 1; i <= N; i++) {
    DP[i] = 1;
    for (int j = 1; j <= N; j++) {
        if (B[j] < A[i]) {
            DP[i] = max(DP[i], j + 1);
        }
    }
    B[DP[i]] = min(B[DP[i]], A[i]);
}

int ans = 0;
for (int i = 1; i <= N; i++) {
    ans = max(ans, DP[i]);
}
```

## #가장 긴 증가하는 부분수열

<LIS>

매 루프문에서  $i$ 가 1씩 증가할 때 마다,  $B[]$ 의 원소도 하나씩 바꿉니다.

사실,  $B[]$  배열은 항상 단조증가 상태를 유지합니다!

$B[]$  배열이 단조증가 한다면,  $\max_{B[j] < A[i]} (j + 1)$  라는 식은  $\min_{B[j] \geq A[i]} (j)$ 으로도 해석할 수 있습니다.

풀어 쓰면 ‘ $B[]$  배열에서  $A[i]$ 보다 크거나 같은 수가 처음으로 나오는 인덱스’ 라는 뜻입니다.

정렬된 배열에서  $x$ 보다 크거나 같은 최초의 인덱스 찾기... 이거 완전 이분탐색 아닌가요?

심지어 해당 연산을 정확히 수행하는 `lower_bound()` 라는 함수가 STL에 구현되어 있습니다!

## #가장 긴 증가하는 부분수열

<LIS>

```
for (int i = 1; i <= N; i++) {  
    DP[i] = 1;  
    for (int j = 1; j <= N; j++) {  
        if (B[j] < A[i]) {  
            DP[i] = max(DP[i], j + 1);  
        }  
    }  
    B[DP[i]] = min(B[DP[i]], A[i]);  
}
```



```
for (int i = 1; i <= N; i++) {  
    DP[i] = 1;  
    int idx = lower_bound(B + 1, B + N + 1, A[i]) - B;  
    DP[i] = idx;  
    B[idx] = min(B[idx], A[i]);  
}
```

## #가장 긴 증가하는 부분수열

<LIS>

$B[]$  배열은 항상 단조증가한다.

proof.

초기 배열  $B = [\text{inf}, \text{inf}, \dots, \text{inf}]$  이므로 단조증가한다.

$DP[i]$ 를 계산하는 시점에서,  $B[]$  배열이 단조증가한다고 가정하자.

$\text{idx}$ 를  $B[]$ 에서 최초로  $A[i]$ 보다 크거나 같은 원소가 등장하는 위치라고 하자.

‘최초’로 크거나 같은 원소가 등장하는 위치이므로,  $B[\text{idx}-1] < A[i] \leq B[\text{idx}]$ 이다.

가정에 의해,  $B[\text{idx}] \leq B[\text{idx}+1]$ 이다.

즉  $B[\text{idx}-1] < A[i] \leq B[\text{idx}+1]$ 이므로,  $B[\text{idx}]$ 를  $A[i]$ 로 대체해도 여전히  $B[]$  배열은 단조증가한다.

## #가장 긴 증가하는 부분수열

<LIS>

### 2 가장 긴 증가하는 부분수열 2 (BOJ #12015)

#### <문제 설명>

- 길이  $N$ 의 수열이 주어진다.
- 주어진 수열의 가장 긴 증가하는 부분수열의 길이를 구해라.

#### <제약 조건>

- $1 \leq N \leq 1,000,000$
- $1 \leq A_i \leq 1,000,000$



## #DP 역추적

<DP track>

지금까지 LIS 문제를 DP를 이용해 해결하는 방법을 알아보았습니다.

하지만 지금까지의 방법으로는 LIS의 길이만 알 수 있었지, 실제로 LIS가 어떻게 생겼는지는 모릅니다.

따라서 DP를 이용하는 문제에서, 실제 답의 형태를 구하는 방법을 알아보시다.



## #DP 역추적

<DP track>

### 1로 만들기 (BOJ #1463)

#### <문제 설명>

- 3가지 연산이 있다.
  1. 3으로 나누어 떨어지면, 3으로 나눈다
  2. 2로 나누어 떨어지면, 2로 나눈다
  3. 1을 뺀다
- 정수 N이 주어졌을 때, 1로 만들기 위해 필요한 최소 연산 횟수는?

#### <제약 조건>

- $1 \leq N \leq 1,000,000$





## #DP 역추적

<DP track>

### 1로 만들기 (BOJ #1463)

#### <문제 해설>

DP[i] : i를 1로 만들기 위해 필요한 최소 연산 횟수.

$DP[i] = \min( DP[i/3] + 1, DP[i/2] + 1, DP[i-1] + 1 )$

## #DP 역추적

&lt;DP track&gt;

1로 만들기 문제에서 역추적을 하는 방법을 알아보시다.

DP[i]를 계산하기 위해 가져온 하위 문제에 화살표를 그어봅시다.

i	1	2	3	4	5	6	7	8	9
DP[i]	0	1	1	2	3	2	3	3	2

어느 숫자에서 출발해 화살표를 따라가다 보면 1에 도달할 것입니다.

N에서 출발해 화살표를 따라 이동하면서 방문하는 모든 숫자들을 적으면 그게 1이 되는 과정입니다.

## #DP 역추적

<DP track>

이러한 ‘화살표’ 를 코드로 표현하기 위해,  
track[i] : DP[i]를 계산하기 위해 가져온 이전 인덱스  
와 같은 배열을 활용할 수 있습니다.

```
for (int i = 2; i <= N; i++) {  
    DP[i] = DP[i - 1] + 1;  
    track[i] = i - 1;  
    if (i % 2 == 0 && DP[i / 2] + 1 < DP[i]) {  
        DP[i] = DP[i / 2] + 1;  
        track[i] = i / 2;  
    }  
    if (i % 3 == 0 && DP[i / 3] + 1 < DP[i]) {  
        DP[i] = DP[i / 3] + 1;  
        track[i] = i / 3;  
    }  
}
```

## #DP 역추적

<DP track>

화살표를 ‘따라간다’는 것은, 반복문을 이용하면 됩니다.  
while문이 종료되면, res vector에 1이 되는 과정이 저장  
되어 있게 됩니다.

```
vector<int> res;  
int curr = N;  
while (true) {  
    res.push_back(curr);  
    if (curr == 1)  
        break;  
    curr = track[curr];  
}
```



## #DP 역추적

<DP track>

### 가장 긴 증가하는 부분수열 4 (BOJ #14002)

#### <문제 설명>


- 길이  $N$ 의 수열이 주어진다.
- 주어진 수열의 가장 긴 증가하는 부분수열을 구해라.


#### <제약 조건>


- $1 \leq N \leq 1,000$
- $1 \leq A_i \leq 1,000$





## #연습 문제 도전

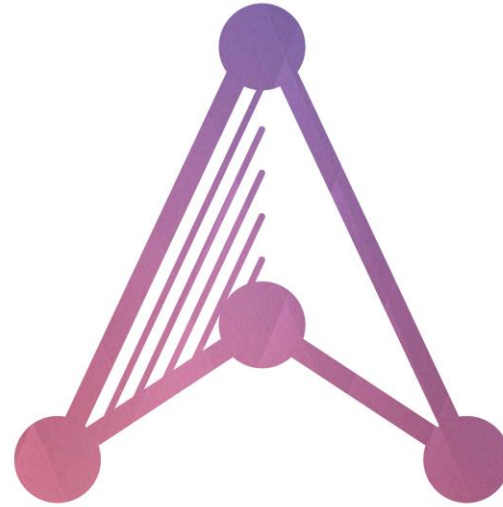
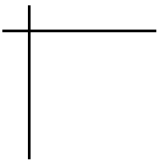
 가장 긴 바이토닉 부분수열 (BOJ #11054)

 전깃줄 2 (BOJ #2568)

 가장 긴 증가하는 부분수열 5 (BOJ #14003)

 숨바꼭질 3 (BOJ #13913)

 전깃줄 (BOJ #2565)



***A L O H A***  
*The algorithm club.*

