Advanced Computer Contest Preparation

Lecture 15

LONGEST INCREASING SUBSEQUENCE

Problem

- We are given a sequence of N numbers: $n_1, n_2, \ldots, n_N$
- Example sequence:

```
5  8  2  3  7  9  1  8  9  10
```

- To form a *subsequence* of an original sequence, we delete some elements without changing order
- A subsequence with each element strictly greater than the previous one is called an *increasing subsequence* (IS)
- What is the length of the longest IS?
  - Called the longest increasing subsequence (LIS)

Solutions?

- Greedy?
  - Since we want the *maximum* length of an IS, this could be considered an optimization problem
  - How do we compare options?
- Brute force?
  - There are 2^N possible subsequences
    - We either include an element, or exclude it
  - For each subsequence, we check if it is an increasing subsequence
  - Runtime: $O(N \times 2^N)$

DP Approach

- What is the overall problem that we are trying to solve?
  - What is the LIS of a sequence of N numbers?
- What are all of the subproblems?
  - *For* $i = 1$ *to* N, consider the sequence of the first i numbers: $n_1, n_2, \ldots, n_i$
  - What is the LIS of this sequence that contains n_i?

DP Approach

- For a problem $p(i)$, what are its subproblems?
  - To form a LIS of the first i elements with n_i, we can take an existing LIS of the first j elements ($j < i$) and append n_i to it
  - Thus, $p(i)$ depends on $p(j)$ *for* $j = 0$ *to* $i-1$

DP Approach

- What is the exact relationship between $p(i)$ and its subproblems?
  - Note that for a problem $p(j)$, n_j is the last and the greatest element of the LIS
  - In order to satisfy LIS, $n_i > n_j$ since $i > j$
  - Thus, $p(i) = max(p(j) + 1 \text{ for } j = 0 \text{ to } i\text{-}1 \text{ if } n_j < n_i)$
- What is the base case?
  - $p(0) = 0$

Example

```
Seq: 5 8 2 3 7 9 1 8 9 10

DP : 1 2 1 2 3 4 1 4 5 6
```

Answer: 6

Pseudocode

```
int N, seq[], dp[];
//get input, assume seq only contains positive integers
int len = 0;
dp[0] = 0;
for (int i = 1; i <= N; i++){
        for (int j = 0; j < i; j++)
                if (seq[j] < seq[i])
                        dp[i] = max(dp[i],dp[j]+1);
        if (dp[i] > len)
                len = dp[i];
}
print len;
```

Analysis

- How many states are there?
  - $O(N)$
- How many subproblems does each state depend on?
  - $O(N)$
- What is the time complexity needed to compute the solution to a problem?
  - $O(N)$
- Therefore, the final time complexity is $O(N^2)$

Generating the LIS

- We know how to get the length of the LIS
- Can we output the LIS given the DP array?
  - Note that there might be more than 1, but we can output any.

```
Seq: 5 8 2 3 7 9 1 8 9 10

DP : 1 2 1 2 3 4 1 4 5 6
```

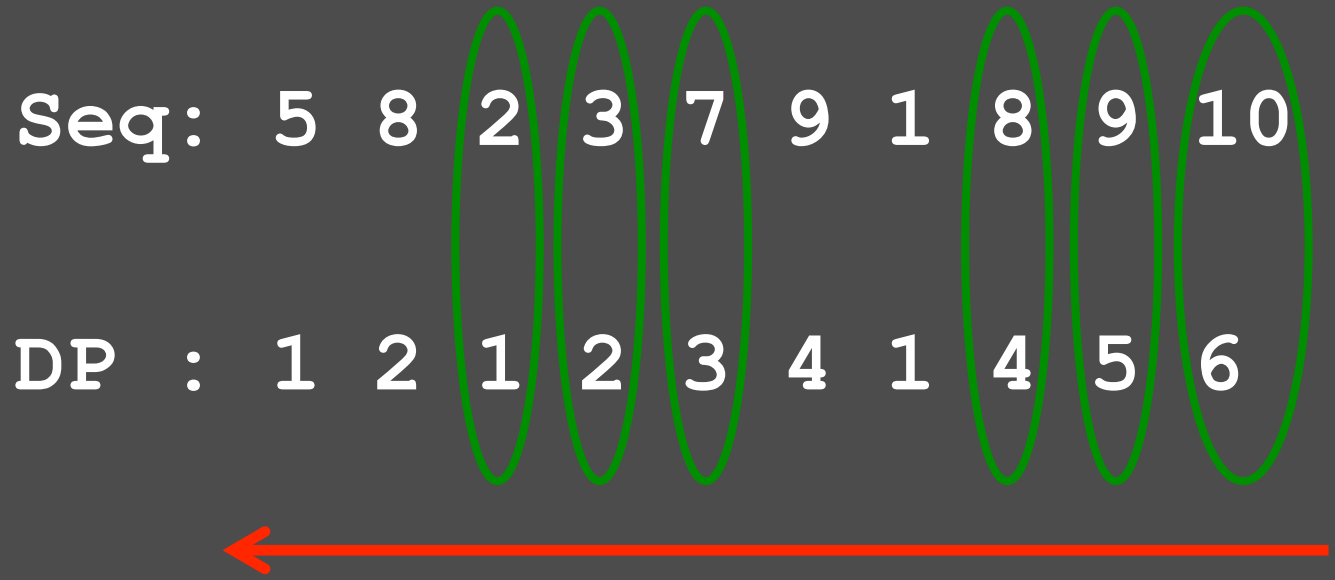
Generating the LIS

- We can use the previous state method
  - Use an additional array to remember the subproblem that was used to generate the solution to the current problem
  - Backtrack after filling in DP array
  - Runtime: $O(N)$

Generating the LIS

- We can use the subproblem check method
  - Try to find the subproblem that was used to generate solution to current problem
  - Remember that for some j, $p(i) = p(j)+1$ and $n_j < n_i$
  - If we find a $p(j)$ that is equal to $p(i)-1$ and $n_j < n_i$, do not continue to find another solution for $p(i)$
  - Runtime: $O(N)$

Generating LIS Example – Method 2



One possible LIS is: 2, 3, 7, 8, 9, 10

Generating LIS Pseudocode – Method 2

```
int N, seq[], dp[], len;
//input and DP here
vector<int> lis;
int tmp = len, cur = INF;
for (int i = N; i > 0; i--){
      if (dp[i] == tmp && seq[i] < cur){
            lis.push_back(seq[i]);
            tmp--;
            cur = seq[i];
      }
}
//LIS is in lis vector, reversed
```

Optimization

- Can we solve the LIS problem faster than $O(N^2)$?
- Let's look at the LIS problem another way by redefining the overall problem and the subproblems

Optimization

- The overall problem is:
  - What is the least end element of the LIS when considering N elements of the subsequence?
- All of the subproblems are:
  - *For* $i = 1$ *to* N, consider the sequence of the first i numbers: $n_1, n_2, \ldots, n_i$
  - What is the least end element of an IS of this sequence with length j for $j = 0$ *to* N that *might not* contain n_i?

Optimization

- $p(i,j)$ represents the least ending element of an IS of length j when the first i elements are considered
- $p(i,0) = -\infty$
- If no such element exists, $p(i,j) = \infty$
- Consider n_1:
  - $p(1,1) = n_1$, $p(1,j > 1) = \infty$
  - $p(1,0) < p(1,1) < p(1,2) \ldots$
- Consider n_c:
  - Let k be the length of the LIS of the first i elements
  - $p(c,0) < p(c,1) < p(c,2) < \ldots < p(c,k) < p(c,k+1) \ldots$
  - $p(c,k+1) = \infty$
  - $k \leq c$

Optimization

- Consider n_{c+1}:
- *for all* $j \geq 1$ and $p(c,j-1) < n_{c+1}$, $p(c+1,j) = min(n_{c+1}, p(c,j))$
  - If the last element of an IS of length $j-1$ is less than n_{c+1}, we can append n_{c+1} to form an IS with length j
- *for all* $j \geq 1$ and $p(c,j-1) \geq n_{c+1}$, $p(c+1,j) = p(c,j)$
  - It is not possible to append n_{c+1} to the end of an IS of length $j-1$, so we can't form an IS of length j

Optimization

- Lemma: There is at most one x such that $p(c+1,x) \neq p(c,x)$
  - Suppose $p(c,x-1) < n_{c+1}$, $p(c,x) \geq n_{c+1}$
  - We know that ... $p(c,x-1) < p(c,x) < p(c,x+1)$ ...
  - $p(c+1,x-1)$ cannot be equal to n_{c+1} since $p(c,x-1) < n_{c+1}$
  - $p(c+1,x+1)$ cannot be equal to n_{c+1} since $p(c,x+1-1) \geq n_{c+1}$
    - We cannot append n_{c+1} to the IS of length x since the smallest possible ending element of that IS is greater than or equal to n_{c+1}
  - Thus, only $p(c+1,x) = n_{c+1}$
- Therefore:
- $p(c+1,0) < p(c+1,1) < \ldots < p(c+1,k) < p(c+1,k+1) < p(c+1,k+2) \ldots$
- $p(c+1,k+2) = \infty$
- $k+1 \leq c+1$

Optimization

- Since there is only at most one value of x such that $p(c+1,x) \neq p(c,x)$ and $p(c,x-1) < n_{c+1}$, $p(c,x) \geq n_{c+1}$, we can binary search for the value of x
- Thus, for each element, we run a binary search in $O(\log k)$ time
- We also proved that $k \leq c$ $(for\ c = 1\ to\ N)$, so the runtime for each binary search is $O(\log N)$

Optimization

- Similar to the knapsack problem, we can compress the 2-D data structure required into 1 dimension since $p(i,x)$ only depends on $p(i-1,y)$, x and y are some positive integers

Summary

- $p(i,j)$ represents the least ending element of an IS of length j when the first i elements are considered, n_i does not need to be a part of the IS
- *For* $i = 1$ *to* N, only 1 value of j could change
  - If $p(i-1,j-1) < n_i$ and $p(i-1,j) \geq n_i$ then $p(i,j) = n_i$
  - Otherwise, $p(i,j) = p(i-1,j)$
- To efficiently find the value of j, we use a binary search
- Since $p(i,x)$ only depends on $p(i-1,y)$, we can compress the 2-D array into a 1-D array
- The answer is the greatest j where $p(N,j) \neq \infty$

Example

Seq: **5** 8 2 3 7 9 1 8 9 10

DP : **5** ∞ ∞ ∞ ∞ ∞ ∞ ∞ ∞ ∞

Example

Seq: 5 **8** 2 3 7 9 1 8 9 10

DP : 5 **8** ∞ ∞ ∞ ∞ ∞ ∞ ∞ ∞

Example

Seq: 5 8 **2** 3 7 9 1 8 9 10

DP : **2** 8 ∞ ∞ ∞ ∞ ∞ ∞ ∞ ∞

Example

Seq: 5 8 2 **3** 7 9 1 8 9 10

DP : 2 **3** ∞ ∞ ∞ ∞ ∞ ∞ ∞ ∞

Example

Seq: 5 8 2 3 **7** 9 1 8 9 10

DP : 2 3 **7** ∞ ∞ ∞ ∞ ∞ ∞ ∞

Example

Seq: 5 8 2 3 7 **9** 1 8 9 10

DP : 2 3 7 **9** ∞ ∞ ∞ ∞ ∞ ∞

Example

Seq: 5 8 2 3 7 9 **1** 8 9 10

DP : **1** 3 7 9 ∞ ∞ ∞ ∞ ∞ ∞

Example

Seq: 5 8 2 3 7 9 1 **8** 9 10

DP : 1 3 7 **8** ∞ ∞ ∞ ∞ ∞ ∞

Example

Seq: 5 8 2 3 7 9 1 8 **9** 10

DP : 1 3 7 8 **9** ∞ ∞ ∞ ∞ ∞

Example

Seq: 5 8 2 3 7 9 1 8 9 **10**

DP : 1 3 7 8 9 **10** ∞ ∞ ∞ ∞

Answer: 6

Pseudocode

```
int N, seq[], dp[];
int k = 0; //current LIS length
//get input
for (int i = 1; i <= N; i++){
      int pos = lower_bound(dp,dp+k,seq[i]) - dp;
      dp[pos] = seq[i];
      if (pos == k) k++;
}
print k;
```

Analysis

- How many states are there?
  - $O(N^2)$, but only $O(N)$ are used
- How many subproblems does each state depend on?
  - $O(1)$
- What is the time complexity needed to compute the solution to a problem?
  - $O(\log N)$
- Therefore, the final time complexity is $O(N \log N)$

Generating the LIS

- Previous state method:
  - Use an additional array (`pos`) to keep track of the position in the original sequence for each number in the DP array
  - Use an additional array (`prv`) to remember the subproblem that was used to generate the solution to the current problem
  - When iterating through the i^{th} element, set `prv[i]` equal to `pos[x-1]`
  - Backtrack after filling the DP array
  - Runtime: $O(N)$

Generating the LIS

- Subproblem check method:
  - Use an additional array to keep track of the length of the LIS of the first i elements that includes n_i
  - This is the same array as the $O(N^2)$ DP array
  - Run the same algorithm as you would on the $O(N^2)$ DP array
  - Runtime: $O(N)$

Example

Seq: **5** 8 2 3 7 9 1 8 9 10

DP : **5** ∞ ∞ ∞ ∞ ∞ ∞ ∞ ∞ ∞

Len: 1

Example

Seq: 5 **8** 2 3 7 9 1 8 9 10

DP : 5 **8** ∞ ∞ ∞ ∞ ∞ ∞ ∞ ∞

Len: 1 2

Example

Seq: 5 8 **2** 3 7 9 1 8 9 10

DP : **2** 8 ∞ ∞ ∞ ∞ ∞ ∞ ∞ ∞

Len: 1 2 1

Example

Seq: 5 8 2 **3** 7 9 1 8 9 10

DP : 2 **3** ∞ ∞ ∞ ∞ ∞ ∞ ∞ ∞

Len: 1 2 1 2

Example

Seq: 5 8 2 3 **7** 9 1 8 9 10

DP : 2 3 **7** ∞ ∞ ∞ ∞ ∞ ∞ ∞

Len: 1 2 1 2 3

Example

Seq: 5 8 2 3 7 **9** 1 8 9 10

DP : 2 3 7 **9** ∞ ∞ ∞ ∞ ∞ ∞

Len: 1 2 1 2 3 4

Example

Seq: 5 8 2 3 7 9 **1** 8 9 10

DP : **1** 3 7 9 ∞ ∞ ∞ ∞ ∞ ∞

Len: 1 2 1 2 3 4 1

Example

Seq: 5 8 2 3 7 9 1 **8** 9 10

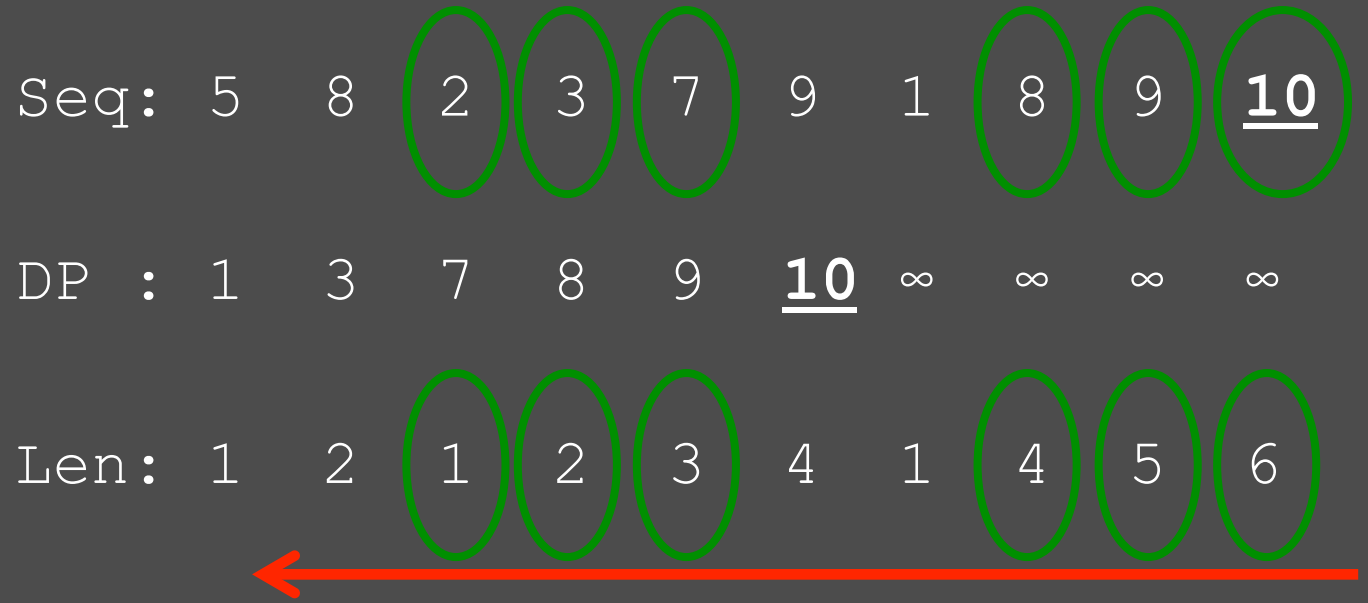
DP : 1 3 7 **8** ∞ ∞ ∞ ∞ ∞ ∞

Len: 1 2 1 2 3 4 1 4

Example

Seq:	5	8	2	3	7	9	1	8	**9**	10
DP :	1	3	7	8	**9**	∞	∞	∞	∞	∞
Len:	1	2	1	2	3	4	1	4	5	

Example



The LIS is: 2,3,7,8,9,10

THANK YOU!