

Dynamic Programming

CSE 301: Combinatorial Optimization

Maximum-sum Interval

Given a sequence of real numbers a_1, a_2, \dots, a_n , find a substring (a.k.a., interval in numeric sequence) with the maximum sum. E.g.:

(2, -5, 9, -3, 1, 7, -15, 2, 3)

Maximum sum interval is: (9, -3, 1, 7)

with sum = 9-3+1+7 = 14

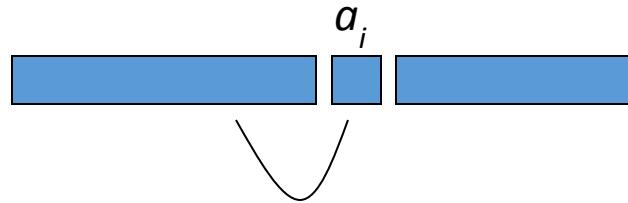
Brute force approach: For each position, we can compute the maximum-sum interval starting at that position in $O(n)$ time. Total time of this approach is $O(n^2)$.

DP Solution for Max Sum Interval

The recurrence relation: Define S_i to be the maximum sum of the intervals ending at position i . There are two cases:

Case 1: If adding a_i with S_{i-1} increases the running sum then: $S_i = S_{i-1} + a_i$

Case 2: If adding a_i with S_{i-1} decreases the running sum then: $S_i = a_i$ (i.e., consider a new substring starting from i -th index)



If $S_i < 0$, concatenating a_i with its previous interval gives less sum than a_i itself. So in this case we will take a_i instead of taking $S_{i-1} + a_i$

$$S_i \leftarrow \max \{S_{i-1} + a_i, a_i\}$$

Dynamic Programming Solution

MaxSumInterval(A,n)

1. $\text{Sum}_0 \leftarrow 0$
 2. **for** $i \leftarrow 1$ to n
 3. $\text{Sum}_i \leftarrow \text{Sum}_{i-1} + A_i$
 4. $\text{Prev}_i \leftarrow i - 1$
 5. **if** $A_i > \text{Sum}_i$
 6. $\text{Sum}_i \leftarrow A_i$
 7. $\text{Prev}_i \leftarrow 0$
- $O(n)$ time

Print_MaxSumInterval(Prev,i)

1. **if** $\text{Prev}_i > 0$
2. Print_MaxSumInterval($\text{Prev}, \text{Prev}_i$)
3. Print A_i

Dynamic Programming Solution

$$9 \quad -3 \quad 1 \quad 7 \quad -15 \quad 2 \quad 3 \quad -4 \quad 2 \quad -7 \quad 6 \quad -2 \quad 8 \quad 4 \quad -9$$

Dynamic Programming Solution

9 -3 1 7 -15 2 3 -4 2 -7 6 -2 8 4 -9

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A	0	9	-3	1	7	-15	2	3	-4	2	-7	6	-2	8	4	-9
Sum	0	9	6	7	14	-1	2	5	1	3	-4	6	4	12	16	7
Prev	0	0	1	2	3	4	0	6	7	8	9	0	11	12	13	14

The maximum sum

The maximum-sum interval: 6 -2 8 4

Longest increasing subsequence (LIS)

The longest increasing subsequence is to find a longest increasing subsequence of a given sequence of distinct integers $a_1 a_2 \dots a_n$.

e.g. 9 2 5 3 7 11 8 10 13 6

2 3 7

5 7 10 13

9 7 11

5 3 11 13

are increasing subsequences.

We want to find a longest one.

are not increasing subsequences.

Can we use LCS to solve LIS?

- Yes. How?
- $\text{LIS}(X) = \text{LCS}(X, \text{sort}(X))$
 - sort takes $O(n \lg n)$ time [assuming, $|X| = n$]
 - LCS takes $O(n^2)$ times
 - So total time to compute LIS in this approach is $O(n^2)$
- Here we converted the input of an LIS problem into the input of an LCS problem and then we output the solution of that LCS problem as a solution of our original LIS problem. In other words, an LCS problem is **reducible to** LIS problem.
- We will talk more about reducibility while discussing about NP-completeness

Can we solve LIS directly without using
LCS (and thus avoid the cost of
sorting)?

A Naïve DP Approach to Solve LIS

Let $L[i]$ be the length of a longest increasing subsequence ending at position i .

$$L[i] = 1 + \max_{j=0 \dots i-1} \{L[j] \mid a_j < a_i\}$$

(use a dummy $a_0 = \inf$ imum of a , and set

Index, i	0	1	2	3	4	5	6	7	8	9	10
Input, a	0	9	2	5	3	7	11	8	10	13	6
L	0	1	1	2	2	3	4	4	5	6	3
prev	-1	0	0	2	2	4	5	5	7	8	4

LIS length is 6 and the subsequence (2,3,7,8,10,13) is an LIS.

Can you write an algorithm that uses the above recurrence to fill up L and $prev$ arrays?

LIS-Naïve-DP(X, n)

```
1.    $L[0] \leftarrow 0$ 
2.    $prev[0] \leftarrow -1$ 
3.    $maxLength \leftarrow 0$  //largest value in L array
4.    $maxIndex \leftarrow 0$  //index of the largest value in L array
5.   for  $i \leftarrow 1$  to  $n$  do
6.        $L[i] \leftarrow 0$ 
7.       for  $j \leftarrow 0$  to  $i-1$  do
8.           if  $a[j] < a[i]$  and  $L[j]+1 > L[i]$  then
9.                $L[i] \leftarrow L[j]+1$ 
10.               $prev[i] \leftarrow j$ 
11.              if  $L[i] > maxLength$  then
12.                   $maxLength \leftarrow L[i]$ 
13.                   $maxIndex \leftarrow i$ 
14.      return  $L$ ,  $prev$ ,  $maxLength$ , and  $maxIndex$ 
```

Running time: $\Theta(n^2)$

Can we design a faster algorithm?

LIS-Naïve-DP(X, n) Issue and Its Solution

5. **for** $i \leftarrow 1$ **to** n **do**
6. $L[i] \leftarrow 0$
7. **for** $j \leftarrow 0$ **to** $i-1$ **do**
8. **if** $a[j] < a[i]$ **and** $L[j]+1 > L[i]$ **then** ...

- LIS-Naïve-DP spends too much time (specifically, $O(n)$) in lines 7-8 searching in the unsorted array a (search for an $a[j]$ which satisfies the condition of line 8).
- We know that searching takes much less time (specifically, $O(\lg n)$ when we apply binary search) on a sorted array
- Since LIS is a sorted array, searching in LIS is more efficient.
- Specifically, we want to keep track of the LISs of length j , where $1 \leq j \leq |LIS(X)|$
- There may exist multiple LISs with the same length. For each length j , we want to keep track of that LIS whose last element is the smallest because this LIS is more likely (than other LISs) to be extended upon reading the next number from array a
- Instead of keeping track of each LIS, it is easier for us to keep track of $L[j] = \text{smallest number with which an LIS of length } j \text{ ends}$
- After reading $a[i]$, we will update this L array in the following way:
 - a. If $a[i] >$ last/largest element of L **then** $L = (L, a[i])$
 - b. Otherwise, replace the smallest element of L which is $\geq a[i]$ by $a[i]$
- Step b above requires searching in L which can be done in $O(\lg n)$ time
- We will also use a $prev$ array using which we can form LIS afterwards

Modified DP Approach to Solve LIS

Index, i	Input, a	L	prev = index (in a) of the element of L which comes just before $a[i]$ in L; -1 if no such element is in L
0	0	{}	-1
1	2	{2}	-1
2	5	{2,5}	Index of 2 in a = 1
3	3	{2,3}	Index of 2 in a = 1
4	6		
5	1		
6	2		
7	10		
8	7		
9	9		

Modified DP Approach to Solve LIS

Index, i	Input, a	L	prev = index (in a) of the element of L which comes just before $a[i]$ in L; -1 if no such element is in L
0	0	{}	-1
1	2	{2}	-1
2	5	{2,5}	Index of 2 in a = 1
3	3	{2,3}	Index of 2 in a = 1
4	6	{2,3,6}	Index of 3 in a = 3
5	1	{1,3,6}	-1
6	2	{1,2,6}	Index of 1 in a = 5
7	10	{1,2,6,10}	Index of 6 in a = 4
8	7	{1,2,6,7}	Index of 6 in a = 4
9	9	{1,2,6,7,9}	Index of 7 in a = 8

Modified DP Approach to Solve LIS

Index, i	Input, a	L	prev = index (in a) of the element of L which comes just before $a[i]$ in L; -1 if no such element is in L
0	0	{}	-1
1	2	{2}	-1
2	5	{2,5}	Index of 2 in a = 1
3	3	{2,3}	Index of 2 in a = 1
4	6	{2,3,6}	Index of 3 in a = 3
5	1	{1,3,6}	-1
6	2	{1,2,6}	Index of 1 in a = 5
7	10	{1,2,6,10}	Index of 6 in a = 4
8	7	{1,2,6,7}	Index of 6 in a = 4
9	9	{1,2,6,7,9}	Index of 7 in a = 8

Modified DP Approach to Solve LIS

Index, i	Input, a	L	prev = index (in a) of the element of L which comes just before $a[i]$ in L; -1 if no such element is in L
0	0	{}	-1
1	2	{2}	-1
2	5	{2,5}	Index of 2 in a = 1
3	3	{2,3}	Index of 2 in a = 1
4	6	{2,3,6}	Index of 3 in a = 3
5	1	{1,3,6}	-1
6	2	{1,2,6}	Index of 1 in a = 5
7	10	{1,2,6,10}	Index of 6 in a = 4
8	7	{1,2,6,7}	Index of 6 in a = 4
9	9	{1,2,6,7,9}	Index of 7 in a = 8

Modified DP Approach to Solve LIS

Index, i	Input, a	L	prev = index (in a) of the element of L which comes just before $a[i]$ in L; -1 if no such element is in L
0	0	{}	-1
1	2	{2}	-1
2	5	{2,5}	Index of 2 in a = 1
3	3	{2,3}	Index of 2 in a = 1
4	6	{2,3,6}	Index of 3 in a = 3
5	1	{1,3,6}	-1
6	2	{1,2,6}	Index of 1 in a = 5
7	10	{1,2,6,10}	Index of 6 in a = 4
8	7	{1,2,6,7}	Index of 6 in a = 4
9	9	{1,2,6,7,9}	Index of 7 in a = 8

Modified DP Approach to Solve LIS

Index, i	Input, a	L	prev = index (in a) of the element of L which comes just before $a[i]$ in L; -1 if no such element is in L
0	0	{}	-1
1	2	{2}	-1
2	5	{2,5}	Index of 2 in a = 1
3	3	{2,3}	Index of 2 in a = 1
4	6	{2,3,6}	Index of 3 in a = 3
5	1	{1,3,6}	-1
6	2	{1,2,6}	Index of 1 in a = 5
7	10	{1,2,6,10}	Index of 6 in a = 4
8	7	{1,2,6,7}	Index of 6 in a = 4
9	9	{1,2,6,7,9}	Index of 7 in a = 8

Modified DP Approach to Solve LIS

Index, i	Input, a	L	prev = index (in a) of the element of L which comes just before $a[i]$ in L; -1 if no such element is in L
0	0	{}	-1
1	2	{2}	-1
2	5	{2,5}	Index of 2 in a = 1
3	3	{2,3}	Index of 2 in a = 1
4	6	{2,3,6}	Index of 3 in a = 3
5	1	{1,3,6}	-1
6	2	{1,2,6}	Index of 1 in a = 5
7	10	{1,2,6,10}	Index of 6 in a = 4
8	7	{1,2,6,7}	Index of 6 in a = 4
9	9	{1,2,6,7,9}	Index of 7 in a = 8

LIS = (2,3,6,7,9)

LIS-Fast-DP(X, n)

```
1.  $L[0] \leftarrow -\infty$ ,  $L'[0] \leftarrow -1$ ,  $\text{prev}[0] \leftarrow -1$  //sentinel values
2. for  $i \leftarrow 1$  to  $n$  do
3.   if  $L[L.\text{length}] < a[i]$  then
4.     append  $a[i]$  at the end of  $L$ 
5.      $L'[L.\text{Length}] \leftarrow i$ 
6.      $\text{prev}[i] \leftarrow L'[L.\text{length}-1]$ 
7.   else
8.      $(s, h) \leftarrow (0, L.\text{length})$ 
9.     while  $s < h$  do    //binary-search for the smallest  $L[j]$  which is  $\geq a[i]$ 
0.        $m \leftarrow \lfloor (s+h)/2 \rfloor$ 
1.       if  $L[m] < a[i]$  then
2.          $s \leftarrow m+1$ 
3.       else
4.          $h \leftarrow m$ 
5.        $L[s] \leftarrow a[i]$ 
6.        $L'[s] \leftarrow i$  //save index of  $a[i]$  in  $L'$ , so that we can use it for prev later
7.        $\text{prev}[i] \leftarrow L'[s-1]$ 
8.   return  $L$ ,  $\text{prev}$ 
```

Run-time: $\Theta(n \lg n)$; as binary search (lines 9-14) takes $\Theta(\lg n)$ time in each of the $\Theta(n)$ iterations of the outer loop.