Dynamic Programming 3

Topics:

- DP Problems.

Q) Given an integer array find the length of maximum increasing subsequence.

2 5 15 3 4 20 9

2 5 15 20 = 4 ➔ increasing order

2 3 4 20 = 4 ➔ increasing order

5 15 20 = 3 ➔ increasing order

Among all, the max length and which is increasing.

One of the brute force ways to solve is to generate all the subsequences and based on which ever subsequence submit it.

So, how to generate subsequence of array? Recursion.

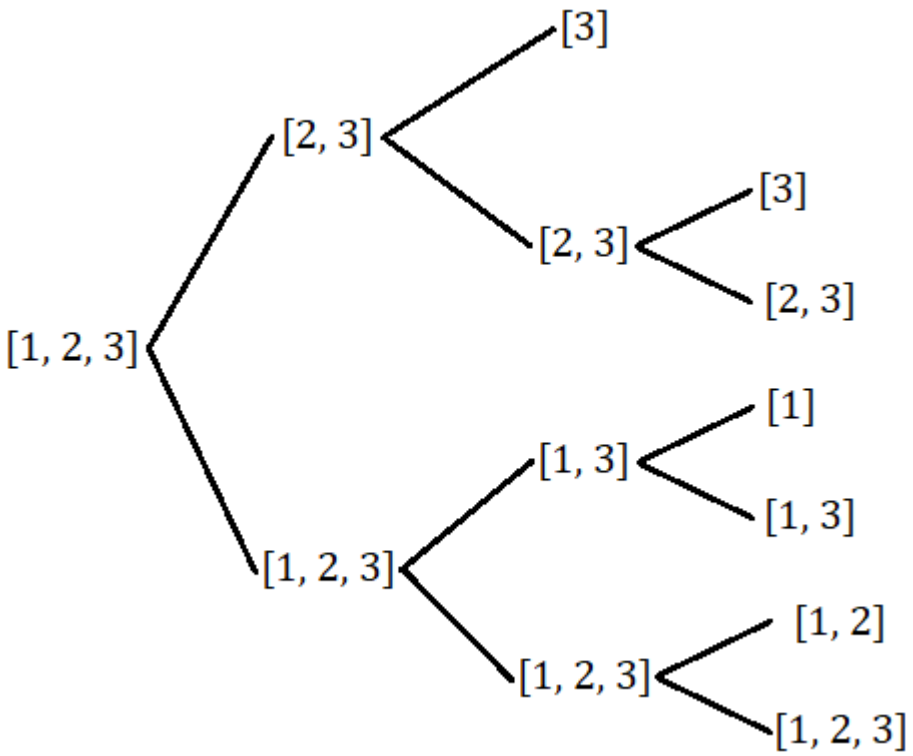
If given an array how can we find all subsequences of it.

Ex: for [1, 2, 3], the ***subsequences*** would be

1
2
3
12
23
13
123

NOTE: Subsequence is not a contiguous chunk of elements.

If we have an array of size 3, and we being at 0; we can take the value at 0^{th} index or not. All the choices that can be taken are show in the below diagram. So, for each character we can take it or leave it.




```python
def generateSubsequence(arr, idx, current_subsequence):

    if idx == len(arr):
        print(current_subsequence)
        return

    # i am taking element at idx position
    current_subsequence.append(arr[idx])
    generateSubsequence(arr, idx + 1, current_subsequence)

    # if i don't take element at idx position
    current_subsequence.pop()
    generateSubsequence(arr, idx + 1, current_subsequence)

if __name__ == "__main__":
    a = [100, 2, 3,4, 5]
    print(generateSubsequence(a))
```

For an arr = [5, 20, 15], we are at 0^{th} index. Now, the current_subsequence is empty []. Since index is 0 and length of array is 3. Hence, `idx != len(arr)`. If we are taking a particular element, (5) and calling recursion generateSubsequence.

In the stack, idx = 1 and current_subsequence = [5].

Now, at 1^{st} index, since idx != len(arr), the element at 1^{st} index position is appended. We are calling `generateSubsequence(arr, idx + 1, current_subsequence)` so the current_subsequence = [5, 20] and the idx = 2.

Now at 2^{nd} index, since idx != len(arr), the element at 2^{nd} index position is appended. We are calling the recursion function `generateSubsequence` so the current_subsequence = [5, 20, 15] and the idx = 2

Now from the stack, we pop last element by `current_subsequence.pop()` and will call `generateSubsequence`.

generateSub([1, 3]) idx = 3
generateSub([1]) idx = 2 cur_subseq = [1, 3]
generateSub([1, 2]) idx = 3
generateSub([1, 2, 3])
generateSub([1, 2]) cur_subseq = [1, 2, idx = 2 3]
generateSub([1]) idx = 1 cur_subseq = [1, 2]
generateSub([ ]) idx = 0 cur_subseq = [1]

since idx == len(arr)
print(current_subsequence)

Time Complexity would be $O(2^n)$ Space Complexity is $O(n)$

Ex 02:

[5, 1, 7, 8, 2, 15] ➔ [1, 7, 8, 15] & [5, 7, 8, 15] so the answer should be 4.

The max increasing subsequence is 1. If we have only a single element, then

[1, 1, 1, 1, 1, 1]

We will put one pointer at 1st index and another at 0th index. Since 5 > 1, the subsequence will not be ending at 1. Now, the pointer will be at 7 and another at 5. Since 5 < 7, this 5 will increase the subsequence ending at 7 by 2. The max (1, 2) = 2.

The pointer is moved forward to 1 from 5. Since 1 < 7, this can update the subsequence ending at 7. The length of subsequence ending at 1 = 1 and at 7 is 2. As max(1, 2) = 2, so the value will not change.

Now, the pointer is at 8 and the pink pointer is at 5. Since 5 < 8, this can update the subsequence ending at 8. The length of subsequence ending at 1 = 2 and at 7 is 3. As max(3, 2) = 3, so the value will not change.

Code:

```
def maxLengthSubsequence(arr):
    n = len(arr)
    dp = [1] * n

    for j in range(1, n):
        for i in range(0, j):
            if arr[i] < arr[j]:
                dp[j] = max(dp[j], dp[i] + 1)

    return max(dp)

if __name__ == "__main__":
    a = [100, 2, 3, 4, 5]
    print(maxLengthSubsequence(a))
```

Ex 03: [1, 5, 2, 3]

[1] i = 0, j = 0
[1, 5] i = 0, j = 1
[1, 5, 2] i = 0, j = 2
[1, 5, 2, 3] i = 0, j = 3

[5] i = 1, j = 1
[5, 2] i = 1. j = 2

```
for i in range (n):
     for j in range (a2 + i, n):
          for k in range (I, j + 1):
               sum += A[k]
```

Time Complexity = $O(n^3)$

```
def findMaxSumSubarray(arr):
    n = len(arr)
    max_sum = 0
    for i in range(0, n):
        for j in range(i, n):
            sum = 0
            for k in range(i, j+1):
                sum += arr[i]
            max_sum = max(max_sum, sum)
    return max_sum
```

How to optimize?

Array = [1, 10, 20, 30, 5, 15]

Running sum array = [1, 11, 31, 61, 66, 81]

Tell me sum from i = 2 to j = 4, will be sum of values from 2 – 4.

```
def findMaxSumSubarray(arr):
    n = len(arr)
    max_sum = 0
    for i in range(0, n):
        for j in range(i, n):
            sum = presum[j] - presum[i-1]
            max_sum = max(max_sum, sum)
    return max_sum
```

Kadane Algorithm:

Q) Given an array A = [], find the maximum sum of contiguous subarray

Ex 04: [1, 2, 3, -2, 5]

At 1, cur_sum = 1 and max_sum = 1.
At 2, cur_sum = 3 and max_sum = 3
At 3, cur_sum = 6 and max_sum = 6
At -2, cur_sum = 4 and max_sum = 6
At 5, cur_sum = 9 and max_sum = 9

So, the max_sum = 9 in the end.

Ex 05: [-2, -3, 4, -1, -2, 1, 5, -3]

At 0, cur_sum = -2; max_sum = -2

At 1, cur_sum = -3; max_sum = -5

If we start our sub-array from -3, then cur_sum = -3. So the cur_sum will always be **`max (cur_sum + a[i], a[i])`** So, between -2 & -3, the max_sum is -2. So max_sum is max(max_sum, cur_sum)

At 4, cur_sum = 1; max_sum = 1

But, it we start our sub-array from 4, then cur_sum = 4. So the max_sum (-2, 4) is 4.

At -1, cur_sum = 0; max_sum = 4

This process will continue for all the elements.

```
def kadaneAlgo(arr):
    cur_sum = arr[0]
    max_sum = arr[0]
    for i in range(1, len(arr)):
        cur_sum = max(cur_sum + arr[i], arr[i])
        max_sum = max(cur_sum, max_sum)
    return max_sum
```

Time Complexity = ***O(n)***

Resources:
For Graphs

1. https://www.geeksforgeeks.org/graph-and-its-representations/
2. https://www.geeksforgeeks.org/breadth-first-search-or-bfs-for-a-graph/
3. https://www.geeksforgeeks.org/depth-first-search-or-dfs-for-a-graph/