dsa-class12

Question 1. Given an array of integers, find subarray with maximum sum.

Brute force solution would be to find sum for each subarray and return max sum.

Time complexity analysis

O(total no of subarray * finding sum of each subarray) ⇒ $O(N^3)$

Can we do better?

We can reduce time complexity to $O(N^2)$ by using cumulative sum array, which will help in finding subarray sum with O(1) complexity.

Can we do better again?

Key observations.

- An element can either be a part of subarray sum or not.
- for a random index m there are 2 possibilitieseither element at index m is part of max subarray sum or max subarray sum is present in left subarray of m or right subarray of m
- With this we can say that we can apply divide and conquer approach to solve this problem.

O(N*logN) solution.

Time complexity O(N*LogN)

We can also solve this problem using Kadane's algorithm.

Algorithm

- Keep adding numbers from left to right
- if curSum > maxSum update maxSum
- if curSum is negative set it to zero.

This code will also handles when our array contains only negative numbers, I that case we will get max number as output.

Follow up question: Find start and end of subarray which has maximum sum.

returns maxSum along with start and end position

Question 2: A slight variation to the problem. Given an array of integer find max subarray sum subject to the constraint that max subarray sum should be less than given number K.

This additional constraint makes it a completely different problem and I have seen candidates trying to apply Kadane’s algorithm to this problem.

We are looking for a subarray sum closest to K and less than K.

What if we maintain cumulative sum till every index, Then the problem reduces to finding i,j such that $i<j$ and $cum[j]-cum[i]$ is as close to k but lower than it.

$cum[j]-cum[i]$ < k

cum[i] > cum[j]-k

So we are looking for a number which is just greater than *c[j]-k*.

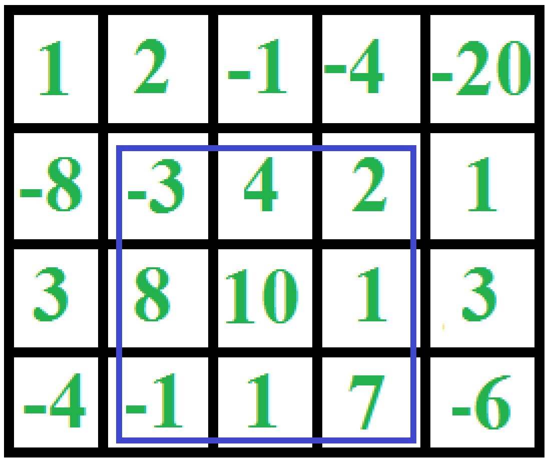
We can use ceiling function of TreeSet in java to find this.

Code.

max subarray sum up to K

Now let’s look at questions where we need to apply kadane’s algorithm.

Question 3: Find maximum sum rectangle in a 2D matrix. We are looking for a sub-matrix where sum of its element is maximum.



1	2	-1	-4	-20
-8	-3	4	2	1
3	8	10	1	3
-4	-1	1	7	-6

Example matrix

We already knows the solution for 1D matrix with kadane's algorithm.

Let's for a given column ci and cj if we have cumulative sum of rows between column ci and cj and apply 1D Kadane's algorithm on this cumulative sum array and let say we got subarray ri and rj and maximum subarray sum then we can say (ci,cj,ri,rj) forms a potential candidate for maximum submatrix sum.

Lets take an example

```
[
[1,2,-1,-4,-20],
[-8,-3,4,2,1],
[3,8,10,1,3],
[-4,-1,1,7,-6]
]lets take ci = 1 and cj = 3cumulative sum of rows between co
lumn ci and cj is
[
-3, (2-1-4)
3,  (-3+4+2)
19, (8+10+1)
7   (-1+1+7)
][-3, 3, 19, 7]
Applying kadane's algorithm on this array will given subarray
[1,3] as maximum subarray sum (29)
so ri = 1 rj = 3So sub-matrix (1,1) (3,3) forms a potential c
andidate for max sub-matrix sum.
```

Code.

Maximum sum submatrix.

Question 4. Given an `m x n` matrix `matrix` and an integer `k`, return *the max sum of a rectangle in the matrix such that its sum is no larger than* `k`.

In this problem we need to use solution of question 2 everything else remains the same.

Recursion

Refer class notes

permutation arrange R elements in N places.

```
private void backtrack(int[] boxes, int cur, int end) {
    if(cur > end) {
        print boxes
        return
    }
    for(int i = 0; i < boxes.length; i++) {
        if(boxes[i] == 0) {
            boxes[i] = cur;
            backtrack(boxes,cur+1,end);
            boxes[i] = 0;
        }
    }
}
```

generate subsets

```
private void backtrack(List<Integer> tempList, int [] nums, int
        res.add(new ArrayList<>(tempList));
        for(int i = start; i < nums.length; i++){
            tempList.add(nums[i]);
            backtrack(tempList, nums, i + 1);
            tempList.remove(tempList.size() - 1);
        }
    }
```