CS255 HOMEWORK 2

Q1: Longest simple path in a directed acyclic graph

Suppose that we are given a directed acyclic graph G =(V,E) with real-valued edge weights and two distinguished vertices s and t. Describe a dynamic programming approach for finding the longest weighted simple path from s to t. What does the subproblem graph look like? What is the efficiency of your algorithm?

Q2: Maximum Subarray (or largest sum contiguous subarray) problem (Section 4.1 of CLRS)

Suppose you are a freelancer and that you plan to work at a Mykonos resort for some part of the n-day summer season next year. Unfortunately, there isn't enough work for you to be paid every day and you need to cover your own expenses (but you want to go). Fortunately, you know in advance that if you are at the resort on the ith day of the season, you'll make pi euros where pi could be negative (if expenses are more than earnings) or positive (if expenses are less than earnings). To maximize your earning you should choose carefully which day you arrive and which day you leave; the days you work should be consecutive and you don't need to work all season. For example, if n = 8 and p1 = −9 , p2 = 10 , p3 = −8 , p4 = 10 , p5 = 5 , p6 = −4 , p7 = −2 , p8 = 5 then if you worked from day 2 to day 5, you would earn 10 − 8 + 10 + 5 = 17 euros in total. Assume the resort pays your airtickets. Give the pseudocode of the following three approaches.
Facts: When all numbers are positive, then all days is the answer, but when all numbers are negative; no days are selected and the total is 0. Your result should always be 0 or a positive number. Hint: use arrays. The output is the maximum sum, as well as the arriving and departing day.
A. The first algorithm, is a brute force $O(n^2)$ algorithm that considers all possible pairs of arriving and departing dates. The outer loop picks the beginning element, the inner loop finds the maximum possible sum with first element picked by outer loop and compares this maximum with the overall maximum.
B. The second algorithm, is a divide and conquer O(n lg n) algorithm. The basic idea follows:
Algorithm D&C (pseudocode)
Use a Divide and Conquer approach. The idea follows
1) Divide the given array in two halves
2) Return the maximum of the following three
….**a)** Maximum subarray sum in left half (Make a recursive call)
….**b)** Maximum subarray sum in right half (Make a recursive call)
….**c)** Maximum subarray sum such that the subarray crosses the midpoint For 2c) we can easily find the crossing sum as follows: simply combine the left part and the right part.
Basically, that method finds the maximum sum starting from mid point and ending at some point on left of mid, then finds the maximum sum starting from mid + 1 and ending with sum point on right of mid + 1 and adding them together.
C. The third algorithm, is a (dynamic programming) O(n) algorithm, known as Kadane's

Algorithm. The basic idea is that you keep only positive contiguous summations that we compute by adding one element each time, kept in maxTemp, and at the end keep the best, and save in maxSum. In the next pseudocode, also the arrive and depart index are computed.

Kadane's Algorithm:

```
//Initialize
 maxSum = 0
 maxTemp= 0
 tempArrive=0
for i=0 to n-1 in array a
 (a) maxTemp = maxTemp + a[i]
 (b) if(maxTemp < 0)
 maxTemp = 0
(c) if(maxSum < maxTemp)
 maxSum = maxTemp
return (maxSum, arrive, depart)
```

returns depart -1 if sum is <0

Q3: Coin changing

Consider the problem of making change for n cents using the fewest number of coins. Assume that each coin's value is an integer.

a. Describe a greedy algorithm to make change consisting of quarters, dimes, nickels, and pennies. Prove that your algorithm yields an optimal solution.

b. Suppose that the available coins are in the denominations that are powers of c, i.e., the denominations are c0, c1, … , ck for some integers c > 1 and k >= 1. Show that the greedy algorithm always yields an optimal solution.

c. Give a set of coin denominations for which the greedy algorithm does not yield an optimal solution. Your set should include a penny so that there is a solution for every value of n.

d. Give an O(nk)-time algorithm that makes change for any set of k different coin denominations, assuming that one of the coins is a penny.

Q4: A k-way merge operation.

Suppose you have k sorted arrays, each with n elements, and you want to combine them into a single sorted array of kn elements.

a) Here's one strategy: Using the Merge procedure from Mergesort, merge the first two arrays, then merge in the third, then merge in the fourth, and so on. What is the time complexity of this algorithm, in terms of k and n?

b) Give a more efficient solution to this problem, using divide-and-conquer. Hint: merge pairwise. What is the time complexity of this algorithm, in terms of k and n?

Q5: Parallel Minimum Spanning Tree

Provide a pseudocode description of how you would implement a Parallel Minimum

spanning Tree. Hint: Your solution need not be complex, maybe based on Kruskal, and should take advantage of parallel algorithms covered in class