CS 170 HW 3

Due on 2019-02-11, at 10:00 pm

1 Study Group

List the names and SIDs of the members in your study group.

2 Maximum Subarray Sum

Given an array of n integers, the maximum subarray is the contiguous subarray (potentially empty) with the largest sum. Design an $\Theta(n \log n)$ algorithm to find the sum of the maximum subarray. For example the maximum subarray of $[-2, 1, -3, 4, -1, 2, 1, -5, 4]$ is $[4, -1, 2, 1]$, whose sum is 6.

(*Extra credit:* Try to design a $\Theta(n)$ solution)

Please give a three-part solution of the following format:

(a) **Clearly** describe your algorithm. You can include the pseudocode optionally.

(b) Write a proof of correctness.

(c) Write a runtime analysis.

3 Modular Fourier Transform

Fourier transforms (FT) have to deal with computations involving irrational numbers which can be tricky to implement in practice. Motivated by this, in this problem you will demonstrate how to do a Fourier transform in modular arithmetic, using modulo 5 as an example.

(a) There exists $\omega \in \{0, 1, 2, 3, 4\}$ such that ω are 4^{th} roots of unity (modulo 5), i.e., solutions to $z^4 = 1$. When doing the FT in modulo 5, this ω will serve a similar role to the primitive root of unity in our standard FT. Show that $\{1, 2, 3, 4\}$ are the 4^{th} roots of unity (modulo 5). Also show that $1 + \omega + \omega^2 + \omega^3 = 0 \pmod 5$ for $\omega = 2$.

(b) Using the matrix form of the FT, produce the transform of the sequence $(0, 1, 0, 2)$ modulo 5; that is, multiply this vector by the matrix $M_4(\omega)$, for the value $\omega = 2$. Be sure to explicitly write out the FT matrix you will be using (with specific values, not just powers of ω). In the matrix multiplication, all calculations should be performed modulo 5.

(c) Write down the matrix necessary to perform the inverse FT. Show that multiplying by this matrix returns the original sequence. (Again all arithmetic should be performed modulo 5.)

(d) Now show how to multiply the polynomials $2x^2 + 3$ and $-x + 3$ using the FT modulo 5.

4 Polynomial from roots

Given a polynomial with exactly n distinct roots at $r_1, \ldots, r_n$, compute the coefficient representation of this polynomial in time. Your runtime should be $O(n \log^c n)$ for some constant c (you should specify what c is in your runtime analysis). There may be multiple possible answers, but your algorithm should return the polynomial where the coefficient of the highest degree term is 1. You can give only the main idea and runtime analysis, a four part solution is not required.

Note: A root of a polynomial p is a number r such that $p(r) = 0$. The polynomial with roots $r_1, ..., r_k$ can be expressed as $\prod_i (x - r_i)$.

5 Triple sum

Design an efficient algorithm for the following problem. We are given an array $A[0..n-1]$ with n elements, where each element of A is an integer in the range $0 \leq A[i] \leq n$. The algorithm must answer the following yes-or-no question: does there exist indices i, j, k such that $A[i] + A[j] + A[k] = n$?

Design an $O(n \lg n)$ time algorithm for this problem. Note that you do not need to actually return the indices; just yes or no is enough.

Hint: define a polynomial of degree $O(n)$ based upon A, then use FFT for fast polynomial multiplication. As an example, $(x + x^2) * (x^{10} + x^{20}) = x^{11} + x^{12} + x^{21} + x^{22}$.

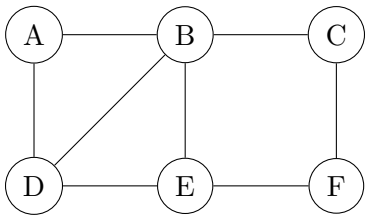
Reminder: don't forget to include explanation, pseudocode, running time analysis, and proof of correctness.

6 Searching for Viruses

Sherlock Holmes is trying to write a computer antivirus program. He starts by modeling his problem. He thinks of computer RAM as being a binary string s_2 of length m. He thinks of a virus as being a binary string s_1 of length $n < m$. His program needs to find all occurrences of s_1 in s_2 in order to get rid of the virus. Even worse, though, these viruses are still damaging if they differ slightly from s_1. So he wants to find all copies of s_1 in s_2 that differ in at most k locations for arbitrary $k \leq n$.

a) Give a $O(nm)$ time algorithm for this problem.

b) Give a $O(m \log m)$ time algorithm for any k. *(Hint: find a way to use FFT.)*
For this question we will not require a 4-part solution. Instead, please give us a clear description of the algorithm and an analysis of the running time. Give Pseudocode if you feel it will enhance your solution, but it is not required.



7 Breadth-First Search

Run breadth-first search on the above graph, breaking ties in alphabetical order (so the search starts from node A). At each step, state which node is processed and the resulting state of the queue. Draw the resulting BFS tree.

8 True Source

Design an efficient algorithm that given a directed graph G determines whether there is a vertex v from which every other vertex can be reached. (Hint: first solve this for directed acyclic graphs. Note that running DFS from every single vertex is not efficient.)