CS 170 Homework 3

Due **2/09/2021, at 10:00 pm**

1 Study Group

List the names and SIDs of the members in your study group. If you have no collaborators, write "none".

2 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.

3 Inverse FFT

Recall that in class we defined M_n, the matrix involved in the Fourier Transform, to be the following matrix:

$$M_n = \begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \omega & \omega^2 & \dots & \omega^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{n-1} & \omega^{2(n-1)} & \dots & \omega^{(n-1)(n-1)} \end{bmatrix},$$

where ω is a primitive n-th root of unity.

For the rest of this problem we will refer to this matrix as $M_n(\omega)$ rather than M_n. In this problem we will examine the inverse of this matrix.

(a) Define

$$M_n(\omega^{-1}) = \begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \omega^{-1} & \omega^{-2} & \dots & \omega^{-(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{-(n-1)} & \omega^{-2(n-1)} & \dots & \omega^{-(n-1)(n-1)} \end{bmatrix}$$

Recall that $\omega^{-1} = 1/\omega = \bar{\omega} = \exp(-2\pi i/n)$.

Show that $\frac{1}{n}M_n(\omega^{-1})$ is the inverse of $M_n(\omega)$, i.e. show that

$$\frac{1}{n}M_n(\omega^{-1})M_n(\omega) = I$$

where I is the $n \times n$ identity matrix – the matrix with all ones on the diagonal and zeros everywhere else.

(b) Let A be a square matrix with complex entries. The *conjugate transpose* $A^\dagger$ of A is given by taking the complex conjugate of each entry of A^T. A matrix A is called *unitary* if its inverse is equal to its conjugate transpose, i.e. $A^{-1} = A^\dagger$. Show that $\frac{1}{\sqrt{n}}M_n(\omega)$ is unitary.

(c) Suppose we have a polynomial $C(x)$ of degree at most $n-1$ and we know the values of $C(1), C(\omega), \dots, C(\omega^{n-1})$. Explain how we can use $M_n(\omega^{-1})$ to find the coefficients of $C(x)$.

4 Triple sum

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 elements are not necessarily distinct). We would like to know if there exist indices i, j, k (not necessarily distinct) such that

$$A[i] + A[j] + A[k] = n$$

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

Please give a 3-part solution to this problem.

5 Searching for Viruses

Sherlock Holmes is trying to write a computer antivirus program. He thinks of computer RAM as being a binary string s_2 of length m, and 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.

You do not need a 3-part solution for either part. Instead, describe the algorithms clearly and give an analysis of the running time.

6 FFT Coding

This semester, we are trying something new: questions which involve coding.

This link will take you to a python notebook, hosted on the Berkeley datahub, in which you will implement three functions (`FFT`, `calc_nth_root`, and `poly_multiply`) to investigate how FFT works. Once you have finished, download a PDF of your completed notebook via File → Download as → PDF via HTML, and append the downloaded pdf to the rest of your homework submission. Be careful when selecting pages on gradescope.

Note: Datahub does not guarantee 100% reliability when you save your notebook, and recommends downloading a local copy occasionally to backup progress (via File → Download as → Notebook (.ipynb)).