CS 170 Homework 2

Due **9/13/2021, at 10:00 pm**

1 Study Group

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

In addition, we would like to share correct student solutions that are well-written with the class after each homework. Are you okay with your correct solutions being used for this purpose? Answer "Yes", "Yes but anonymously", or "No"

2 Werewolves

You are playing a party game with n other friends, who play either as werewolves or citizens. You do not know who is a citizen and who is a werewolf, but all your friends do. There are always more citizens than there are werewolves.

Your goal is to identify one player who is certain to be a citizen.

Your allowed 'query' operation is as follows: you pick two people. You ask each person if their partner is a citizen or a werewolf. When you do this, a citizen must tell the truth about the identity of their partner, but a werewolf doesn't have to (they may lie or tell the truth about their partner).

Your algorithm should work regardless of the behavior of the werewolves.

(a) Give a way to test if a single player is a citizen using $O(n)$ queries. Just an informal description of your test and a brief explanation of why it works is needed.

(b) Show how to find a citizen in $O(n \log n)$ queries (where one query is taking two people x and y and asking x to identify y and y to identify x).

There is a linear-time algorithm for this problem, but you cannot use it here, as we would like you to get practice with divide and conquer.

Hint: Split the group into two groups, and use part (a). What invariant must hold for at least one of the two groups?

Give a 3-part solution.

(c) (**Extra Credit**) Can you give a linear-time algorithm?

Hint: Don't be afraid to sometimes 'throw away' a pair of people once you've asked them to identify their partners.

Solution:

(a) To test if a player x is a citizen, we ask the other $n-1$ players what $x's$ identity is. Claim: x is a citizen if and only if at least half of the other players say x is a citizen. To see this, notice that if x is a citizen, at least half of the remaining players are also citizens, and so regardless of what the werewolves do at least half of the players will say

x is a citizen. On the other hand, if x is a werewolf, then strictly more than half of the remaining players are citizens, and so strictly less than half the players can falsely claim that $x$is a citizen.

(b) **Main idea** The divide and conquer algorithm to find a citizen proceeds by splitting the group of friends into two (roughly) equal sets A and B, and recursively calling the algorithm on A and B: $x = \mathit{citizen}(A)$ and $y = \mathit{citizen}(B)$, and checking x or y using the procedure in part (a) and returning one who is a citizen. If there is only one player, return that player.

Proof of correctness We will prove that the algorithm returns a citizen if given a group of n people of which a majority are citizens. By strong induction on n:

Base Case If $n = 1$, there is only one person in the group who is a citizen and the algorithm is trivially correct.

Induction hypothesis The claim holds for $k < n$.

Induction step After partitioning the group into two groups A and B, at least one of the two groups has more citizens than werewolves. By the induction hypothesis the algorithm correctly returns a citizen from that group, and so when the procedure from part (a) is invoked on x and y at least one of the two is identified as a citizen.

Running time analysis Two calls to problems of size $n/2$, and then linear time to compare the two people returned to each of the friends in the input group: $T(n) = 2T(\frac{n}{2}) + O(n) = O(n \log n)$ by Master Theorem.

(c) **Main idea** Split up the friends into pairs and for each pair, if either says the other is a werewolf, discard both friends; otherwise, discard any one and keep the other friend. If n was odd, use part (a) to test whether the odd man out is a citizen. If yes, you are done, else recurse on the remaining at most $n/2$ friends.

Proof of correctness After each pass through the algorithm, citizens remain in the majority if they were in the majority before the pass. To see this, let n_1, n_2 and n_3 be the number of pairs of friends with both citizens, both werewolves and one of each respectively. Then the fact that citizens are in the majority means that $n_1 > n_2$. Note that all the n_3 pairs of the third kind get discarded, and one friend is retained from each of the n_1 pairs of the first kind. So we are left with at most $n_1 + n_2$ friends of whom a majority n_1 are citizens. It is straightforward to now turn this into a formal proof of correctness by strong induction on n.

Running time analysis In a single run of the algorithm on an input set of size n, we do $O(n)$ work to check whether f_1 is a citizen in the case that n is odd and $O(n)$ to pair up the remaining friends and prune the candidate set to at most $n/2$ people. Therefore, the runtime is given by the following recursion:

$$T(n) = T\left(\frac{n}{2}\right) + O(n) = O(n) \text{ by Master Theorem.}$$

3 Fourier Transform Basics

We will use ω_n to denote the first root of unity $\omega_n = e^{2\pi i/n}$.

Fast Fourier Transform! The *Fast Fourier Transform* FFT(p, n) is an algorithm to perform the *Discrete Fourier Transform* FT(p) which takes arguments n, some power of 2, and p, some vector $[p_0, p_1, \ldots, p_{n-1}]$.

Treating p as a polynomial $P(x) = p_0 + p_1 x + \ldots + p_{n-1}x^{n-1}$, the FFT computes the following matrix multiplication in $\mathcal{O}(n \log n)$ time:

$$\begin{bmatrix} P(1) \\ P(\omega_n) \\ P(\omega_n^2) \\ \vdots \\ P(\omega_n^{n-1}) \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega_n^1 & \omega_n^2 & \cdots & \omega_n^{(n-1)} \\ 1 & \omega_n^2 & \omega_n^4 & \cdots & \omega_n^{2(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega_n^{(n-1)} & \omega_n^{2(n-1)} & \cdots & \omega_n^{(n-1)(n-1)} \end{bmatrix} \cdot \begin{bmatrix} p_0 \\ p_1 \\ p_2 \\ \vdots \\ p_{n-1} \end{bmatrix}$$

(a) What is the Fourier transform of $(3, i, 2, 4)$?

Solution: The primitive root of unity we consider in this case is $\omega = e^{(2*i*\pi)/4} = i$. The Fourier transform is

$$\begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & i & -1 & -i \\ 1 & -1 & 1 & -1 \\ 1 & -i & -1 & i \end{pmatrix} \begin{pmatrix} 3 \\ i \\ 2 \\ 4 \end{pmatrix} = \boxed{\begin{pmatrix} 9+i \\ -4i \\ 1-i \\ 2+4i \end{pmatrix}}.$$

(b) Find x and y such that FT$(x) = (5, 2, 1, -i)$ and FT$(y) = (4, 4, i, i)$.

Solution:

(c) *Using part b*, find z such that FT$(z) = (1, -2, 1-i, -2i)$. (*Hint:* Observe that FT(v) is a linear transform of v, and recall the properties of linear transforms).

(d) Compute $(2x^2 + 1)(x + 4)$ using a Fourier transform. (*Hint:* Recall that to multiply two polynomials, first, they must both be converted by the Fourier transformation, then multiplied pointwise, and finally converted back to coefficient form)

Solution:

(a)

(b) To find the inverse Fourier transform, we just need to take the Fourier transform with $\omega = -i$ and then normalize it. In particular

$$x = \frac{1}{4}\begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -i & -1 & i \\ 1 & -1 & 1 & -1 \\ 1 & i & -1 & -i \end{pmatrix}\begin{pmatrix} 5 \\ 2 \\ 1 \\ -i \end{pmatrix} = \boxed{\frac{1}{4}\begin{pmatrix} 8-i \\ 5-2i \\ 4+i \\ 3+2i \end{pmatrix}}$$

$$y = \frac{1}{4}\begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -i & -1 & i \\ 1 & -1 & 1 & -1 \\ 1 & i & -1 & -i \end{pmatrix}\begin{pmatrix} 4 \\ 4 \\ i \\ i \end{pmatrix} = \boxed{\frac{1}{4}\begin{pmatrix} 8+2i \\ 3-5i \\ 0 \\ 5+3i \end{pmatrix}}.$$

(c) Notice first that the FT is a linear transform (since it can be expressed as just a matrix multiplication). Notice then that $\mathrm{FT}(z) = \mathrm{FT}(x) - \mathrm{FT}(y)$, so $\mathrm{FT}(z) = \mathrm{FT}(x-y)$, which tells us that $z = x - y$. Hence,

$$\boxed{z = \frac{1}{4}\begin{pmatrix} -3i \\ 2+3i \\ 4+i \\ -2-i \end{pmatrix}}.$$

For this part, only partial credit is awarded for not using part (b). Full credits for getting $z = x - y$, even if some computations are wrong in part (b) or (c).

(d) Observe that Fourier transform of $(1, 0, 2, 0)$ is (we take Fourier transform with $\omega = i$)

$$\begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & i & -1 & -i \\ 1 & -1 & 1 & -1 \\ 1 & -i & -1 & i \end{pmatrix}\begin{pmatrix} 1 \\ 0 \\ 2 \\ 0 \end{pmatrix} = \begin{pmatrix} 3 \\ -1 \\ 3 \\ -1 \end{pmatrix}$$

Again, the Fourier transform of $(4, 1, 0, 0)$ is

$$\begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & i & -1 & -i \\ 1 & -1 & 1 & -1 \\ 1 & -i & -1 & i \end{pmatrix}\begin{pmatrix} 4 \\ 1 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 5 \\ 4+i \\ 3 \\ 4-i \end{pmatrix}$$

Multiplying the above pointwise and taking inverse Fourier transform, we get

$$\frac{1}{4}\begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -i & -1 & i \\ 1 & -1 & 1 & -1 \\ 1 & i & -1 & -i \end{pmatrix}\begin{pmatrix} 15 \\ -4-i \\ 9 \\ -4+i \end{pmatrix} = \begin{pmatrix} 4 \\ 1 \\ 8 \\ 2 \end{pmatrix}$$

This gives us the polynomial $4 + x + 8x^2 + 2x^3$

4 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. This problem is just about understandin the Fourier transform itself; no need to use the FFT algorithm.

(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 $(1, 0, 0, 3)$ 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.

Solution:

(a) We can check that $1^4 = 1 \pmod 5$,
$2^4 = 16 = 1 \pmod 5$,
$3^4 = 81 = 1 \pmod 5$,
$4^4 = 256 = 1 \pmod 5$.

Observe that taking $\boxed{\omega = 2}$ produces the following powers: $(\omega, \omega^2, \omega^3) = (2, 4, 3)$. Verify that

$$1 + \omega + \omega^2 + \omega^3 = 1 + 2 + 4 + 3 = 10 = 0 \pmod 5.$$

(b) For $\omega = 2$:

$$M_4(2) = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 4 & 3 \\ 1 & 4 & 1 & 4 \\ 1 & 3 & 4 & 2 \end{bmatrix}.$$

Multiplying with the sequence $(1, 0, 0, 3)$ we get:

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 4 & 3 \\ 1 & 4 & 1 & 4 \\ 1 & 3 & 4 & 2 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 0 \\ 0 \\ 3 \end{bmatrix} = \begin{bmatrix} 4 \\ 0 \\ 3 \\ 2 \end{bmatrix}$$

(c) Recall that when working with the FT outside of modspace, our inverse matrix of $M_4(\omega)$ would be given by $\frac{1}{4}M_4(\omega^{-1})$.

In modspace, we can replace $\frac{1}{4}$ with the multiplicative inverse of $4 \pmod 5$ (which is 4), and ω^{-1} with the multiplicative inverse of $2 \mod 5$ (which is 3).

So for $\omega = 2$, the inverse matrix of $M_4(2)$ is the matrix

$$4 \cdot \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 3 & 4 & 2 \\ 1 & 4 & 1 & 4 \\ 1 & 2 & 4 & 3 \end{bmatrix}$$

We can verify that multiplying these two matrices mod 5 equals the identity. Also:

$$4 \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 3 & 4 & 2 \\ 1 & 4 & 1 & 4 \\ 1 & 2 & 4 & 3 \end{bmatrix} \cdot \begin{bmatrix} 4 \\ 0 \\ 3 \\ 2 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 3 \end{bmatrix}$$

(d) Just like in FFT, we will multiply $M_4(\omega)$ by the coefficient representations of the polynomials, point-wise multiply the resulting vectors, and then multiply the resulting vector by $M_4(2)^{-1}$ to get a coefficient representation for their product.

For $\omega = 2$: We first express the polynomials as vectors of dimension 4 over the integers mod 5: $a = (3, 0, 2, 0)$, and $b = (3, -1, 0, 0) = (3, 4, 0, 0)$ respectively.

Multiplying the matrix $M_4(2)$ with both produces $(0, 1, 0, 1)$ and $(2, 1, 4, 0)$ respectively.

Then we just multiply the vectors coordinate-wise to get $(0, 1, 0, 0)$.

Now, we multiply inverse FT matrix $M_4(2)^{-1}$ that we wrote down in the previous part to get the final polynomial in the coefficient representation. The product is as follows: $\boxed{(4, 2, 1, 3)}$. This corresponds to the polynomial $\boxed{3x^3 + x^2 + 2x + 4}$.

We can verify this is correct by multiplying the two polynomials using FOIL to get $-2x^3 + 6x^2 - 3x + 9$. The coefficients of these two polynomials are equivalent $\pmod 5$.

5 Counting k-inversions

A *k-inversion* in a bitstring b is when a 1 in the bitstring appears k indices before a 0; that is, when $b_i = 1$ and $b_{i+k} = 0$, for some i. For example, the string 010010 has two 1-inversions (starting at the second and fifth bits), one 2-inversion (starting at the second bit), and one 4-inversion (starting at the second bit).

Devise an algorithm which, given a bitstring b of length n, counts all the k-inversions, for each k from 1 to $n - 1$. Your algorithm should run faster than $\Theta(n^2)$ time. You can assume arithmetic on real numbers can be done in constant time.

Give a 3-part solution.

Solution: We will use the $\Theta(n \log n)$-time FFT-based cross-correlation algorithm from lecture.

Algorithm description

Construct two vectors: vector $\vec{p} = [0, \ldots, 0, b_0, \ldots, b_{n-1}]$ is the "padded" vector which has length $2n$ and consists of every bit in b individually, preceded by n zeros; vector $\vec{f} = [(1-b_0), \ldots, (1-b_{n-1})]$ is the "flipped" vector which consists of every bit in b, flipped (replace 0 with 1 and 1 with 0).

Run the cross-correlation algorithm on these vectors. The dot product between f and $p[j : n+j]$ is the number of $(n-j)$-inversions, which we can report for j from 1 to $n-1$. (Note that the first and last dot products will be ignored, since there are no 0- or n-inversions.)

Proof of correctness

There is a k-inversion starting at index i if and only if $b_i = 1$ and $b_{i+k} = 0$; which in turn is true if and only if $(b_i)(1-b_{i+k}) = 1$. Consider our dot product $f \cdot p[j : n+j] = \sum_{i=0}^{n-1} f_i p_{i+j}$; but since $p_{i+j} = 0$ whenever $i + j < n$, this is $\sum_{i=n-j}^{n-1} f_i p_{i+j} = \sum_{i=0}^{j-1} f_{i+n-j} p_{n+i}$ which, by the definition of f and p, is $\sum_{i=0}^{j-1} (1 - b_{i+n-j}) b_i$. But by the first sentence, each term of this sum is 1 if there is a $(n-j)$-inversion starting at i, and 0 otherwise, so this sum is the number of $(n - j)$-inversions.

Since we know from lecture that the cross-correlation algorithm computes these products correctly, we can conclude that our whole algorithm is correct.

Runtime

Constructing our two vectors p and f takes a total of $\Theta(n)$ time, and we know from lecture that the cross-correlation algorithm takes $\Theta(n \log n)$ time. Arranging the results correctly may take $\Theta(n)$ time, or constant time, depending on implementation. So, since we do no other work, our algorithm as a whole takes $\Theta(n \log n)$ time.