CS 170 Homework 2

Due **2/07/2022, 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.

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: $x = citizen(A)$ and $y = citizen(B)$, 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 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 FFT, produce the transform of the sequence $(0,3,2,0)$ modulo 5; that is, evaluate the polynomial $3x+2x^2$ at $\{1,2,4,3\}$ using the recursive FFT algorithm defined in class, but with $\omega = 2$ and in modulo 5 instead of with $\omega = i$ in the complex numbers. All calculations should be performed modulo 5. *Hint: You can verify your calculation by evaluating the polynomial at the roots of unity using the slow method, if you like.*

(c) Now perform the inverse FFT on the sequence $(0,4,1,0)$, also using the recursive algorithm. Recall that the inverse FFT is the same as the forward FFT, but using ω^{-1} instead of ω, and with an extra multiplication by 4^{-1} for normalization.

(d) Now show how to multiply the polynomials $3x + 2x^2$ and $3 - x$ using the FFT modulo 5. You may use the fact that the FT of $(3,4,0,0)$ modulo 5 is $(2,1,4,0)$ without doing your own calculation. Note that although we would normally have to use the 8th roots of unity to do this multiplication, we can get away with just the 4th roots of unity in this case (why?).

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) The FFT divide-and-conquer algorithm gives us

$$\text{FFT}(0,3,2,0) = (\text{FFT}(0,2) + (1,2)\text{FFT}(3,0), \text{FFT}(0,2) - (1,2)\text{FFT}(3,0))$$

where the multiplication by $(1,2)$ is elementwise.

Then, $\text{FFT}(0,2) = (\text{FFT}(0) + 1 * \text{FFT}(2), \text{FFT}(0) - 1 * \text{FFT}(2))$ and $\text{FFT}(3,0) = (\text{FFT}(3) + 1 * \text{FFT}(0), \text{FFT}(3) - 1 * \text{FFT}(0))$.

The FFT of one element does nothing, so $\text{FFT}(0) = 0$, $\text{FFT}(2) = 2$, and $\text{FFT}(3) = 3$. This means $\text{FFT}(0,2) = (2,-2) = (2,3) \pmod 5$, and $\text{FFT}(3,0) = (3,3)$. Then $\text{FFT}(0,3,2,0) = (2+3, 3+6, 2-3, 3-6) \pmod 5 = (0,4,4,2)$.

(c) $4^{-1} = 4 \pmod 5$ and $-2 = 3 \pmod 5$, so our answer will be $4 * \text{iFFT}(0,4,1,0) = 4 * (\text{iFFT}(0,1) + (1,3)\text{iFFT}(4,0), \text{iFFT}(0,1) - (1,3)\text{iFFT}(4,0))$.

We also have that $\text{iFFT}(0,1) = (0+1, 0-1) = (1,4) \pmod 5$ and $\text{iFFT}(4,0) = (4,4) \pmod 5$ (note that the unnormalized inverse FFT at $n = 2$ and below is identical to the forward FFT).

This means iFFT$(0, 4, 1, 0) = (1 + 4, 4 + 12, 1 - 4, 4 - 12) = (0, 1, 2, 2)$, giving us a final answer of $4 * (0, 1, 2, 2) = (0, 4, 3, 3)$.

(d) We will run our FFT on the coefficient representations of the polynomials, point-wise multiply the resulting vectors, and then run the inverse FFT on the resulting vector by to get a coefficient representation for their product.

We already have that the FFT of $3x + 2x^2$ is $(0, 4, 4, 2)$ from our result in part (b); we have that the FFT of $3 - x$ is $(2, 1, 4, 0)$ given in the question (since $-1 = 4$ modulo 5, and so $(3, 4, 0, 0)$ is the coefficient representation of $3 - x$).

Multiplying these elementwise gives us $(0, 4, 1, 0)$ as our answer modulo 5.

We already know from our answer to part (c) that the inverse FFT of this is $(0, 4, 3, 3)$, so our final answer is $4x + 3x^2 + 3x^3$. We can check that this is correct by multiplying the polynomials directly using FOIL.

4 Finding Clusters

We are given a directed graph $G = (V, E)$, where $V = \{1, \ldots, n\}$, i.e. the vertices are integers in the range 1 to n. For every vertex i we would like to compute the value $m(i)$ defined as follows: $m(i)$ is the smallest j such from which you can reach vertex i. (As a convention, we assume that i is reachable from i.)

(a) Show that the values $m(1)$, ..., $m(n)$ can be computed in $O(|V| + |E|)$ time.

Please give a 3-part solution to this problem.

Solution: The algorithm is as follows.

procedure DFS-Clusters(G)
  while there are unvisited nodes in G **do**
    Run DFS on G starting from the numerically-first unvisited node j
    for i visited by this DFS **do** $m(i) := j$

To see that this algorithm is correct, note that if a vertex i is assigned a value then that value is the smallest of the nodes that can reach it in G, and every node is assigned a value because the loop does not terminate until this happens.

The running time is $O(|V| + |E|)$ since the algorithm is just a modification of DFS.

(b) Suppose we instead define $m(i)$ to be the smallest j that can be reached from i, instead of the smallest j from which you can reach i. How should you modify your answer to part (a) to work in this case? **Solution:** We start by reversing all edges in G, i.e. replacing each edge (u, v) with the edge (v, u), and then just using our algorithm from the previous part. This works because in the reversed graph, we can reach j from i if and only if we can reach i from j in the original.