CS 170 Homework 2

Due **9/14/2020, 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 citizen.

Your allowed 'query' operation is as follows: you pick two people. You ask each person if their partner is a citizen or 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 asking a pair of people to identify the other person).

You cannot use a linear-time algorithm 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.

3 Agent Meetup

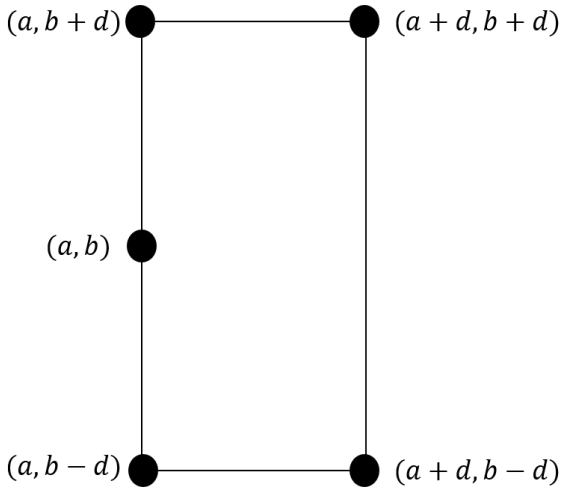
Manhattan has an "amazing" road system where streets form a checkerboard pattern, and all roads are either straight North-South or East-West. We simplify Manhattan's roadmap by assuming that each pair x, y, where x and y are integers, corresponds to an intersection.

As a result, the distance between any two intersections can be measured as the *Manhattan distance* between them, i.e. $|x_i - x_j| + |y_i - y_j|$. You, working as a mission coordinator at the CS 170 Secret Service Agency, have to arrange a meeting between two of n secret agents located at intersections across Manhattan. Hence, your goal is to find the two agents that are the closest to each other, as measured by their Manhattan distance.

In this problem, you will devise an efficient algorithm for this special purpose. As mentioned before, you can assume that the coordinates of the agents are integer values, i.e. the ith agent is at location (x_i, y_i) where x_i, y_i are integers.

Note: This problem is very geometric, we suggest you draw examples when working on it!

(a) Let (a, b) be an arbitary intersection. Suppose all agents i for which $x_i > a$ are Manhattan distance strictly greater than d apart from each other, where $d > 0$. Give an upper bound on the number of agents i that satisfy $a \leq x_i \leq a + d$ and $b - d \leq y_i \leq b + d$. In other words, how many agents can fit in this rectangle (visualized below) without two of these agents being Manhattan distance d or less apart?



Briefly justify your answer. A reasonable bound suffices, you are not expected to provide the tightest possible bound.

(b) Design a divide and conquer algorithm to find the minimum Manhattan distance between any two agents. **Give a three-part solution.** A solution that has runtime within logarithmic factors of the optimal runtime will still get full credit.

Hint: Think about Maximum Subarray Sum from HW1 and the cases we used in that problem. Can similar cases be used here? How can we use part (a) to handle one of those cases efficiently?

4 Practice with Polynomial Multiplication with FFT

(a) Suppose that you want to multiply the two polynomials $x + 1$ and $2x + 1$ using the FFT. Choose an appropriate power of two n and corresponding root of unity ω_n to use in FFT,

find the FFT of the two sequences, multiply the results componentwise, and compute the inverse FFT to get the final result.

(b) Repeat for the pair of polynomials $1 + x + 2x^2$ and $2 + 3x$.

5 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.

6 Sparse FFT

Given an input vector $(p_0, p_1, \ldots, p_{n-1})$, the FFT algorithm computes the following matrix-vector product:

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

in time $O(n \log n)$. Sometimes we want to apply FFT to a *sparse* vector, i.e. a vector where most p_i values are 0. In this case, it is not necessary to do $O(n \log n)$ work.

Modify the FFT algorithm so that it runs in $O(n \log k)$ time rather than $O(n \log n)$ time, where k is the number of p_i that are non-zero. Give a description of your algorithm and a runtime analysis; no proof of correctness is necessary, as long as your main idea is clear.

7 Polynomial from roots

A root of a polynomial p is a number r such that $p(r) = 0$. Given a polynomial with exactly n distinct roots at $r_1, \ldots, r_n$, compute the coefficient representation of this polynomial. Your runtime should be $\mathcal{O}(n \log^c n)$ for some constant $c > 0$ (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 algorithm description and runtime analysis, a three-part solution is not required.

Hint: One polynomial with roots $r_1, ..., r_k$ is

$$p(x) = \prod_{i=1}^{k} (x - r_i)$$

. Recall that you can use FFT as a black-box algorithm for multiplying polynomials in time $O(n \log n)$.