CS170– Spring 2022— Homework 2

CurMack

July 28, 2022

Fast Fourier Transform

A way to move from coefficients to values in time just $O(n \log n)$, when the points $\{x_i\}$ are complex nth roots of unity $(1, \omega, \omega^2, \ldots, \omega^{n-1})$.

$$\langle \text{values} \rangle = \textbf{FFT}(\langle \text{coefficients} \rangle, \omega)$$

$$\langle \text{coeffciients} \rangle = \frac{1}{n}\textbf{FFT}(\langle \text{values} \rangle, \omega^{-1})$$

Algorithm 1: FFT(a, ω)

Input: An array $a - (a_0, a_1, \ldots, a_{n-1})$, for n a power of 2. And ω, a primitive nth root of unity

Output: $M_n(\omega)a$

```
if ω = 1 then
    return a
(s_0, s_1, ..., s_{n/2-1}) ← FFT((a_0, a_2, ⋯ a_{n-2}), ω^2)
(s'_0, s'_1, ..., s'_{n/2-1}) ← FFT((a_1, a_3, ⋯ a_{n-1}), ω^2)
for j = 0 to n/2 − 1 do
    r_j ← s_j + ω^j s'_j
    r_{j+n/2} ← s_j − ω^j s'_j
return (r_0, r_1, ..., r_{n-1})
```

Depth-First Search

To find all nodes reachable from a particular node:

Algorithm 2: explore(G, v)

Input: $G = (V, E)$ is a graph; $v \in V$

Output: *visited*(u) is set to true for all nodes u reachable from v

```
vistited(v) ← true
previsited(v)
for each edge (v, u) ∈ E do
    if not vistited(u) then
        explore(u)
postvisited(v)
```

The `explore` procedure visits only the portion of the graph reachable from its starting point. To examine the rest of the graph, we need to restart the procedure elsewhere, at some vertex that has not yet been visited.

Algorithm 3: DFS(G)

```
for all v ∈ V do
    visited(v) ← false
for all v ∈ V do
    if not visited(v) then
        explore(v)
```

Running time: $\mathcal{O}(|V| + |E|)$

2 Werewolves

(a) To test if a single player x is a citizen, we ask the other $n-1$ players to identify x, which takes $\mathcal{O}(n)$ queries.
Claim: x is a citizen if and only if at least half of the other players say x is a citizen.

(b) Algorithm

Main Idea using devide and conquer to get runtime $\mathcal{O}(n \log n)$:

Algorithm 4: FindCitizen(G)

```
Input: G is a group of which a majority are citizens
if size(G) == 1 then
    return the only person
Split the G into two sets A and B with the (roughly) same size
x ← FindCitizen(A)
y ← FindCitizen(B)
if check(x, G) == citizen then
    return x
else
    return y
```

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(see in reference). The main idea is that after partitioning the group into two groups, at least one of the two groups still mainntens more citizens than werewolves, which will give the right answer.

Runtime Since there are two calls to the algorithm of size $n/2$, and then use linear time to identify the answer, so by using the Master Theorem:

$$T(n) = 2T(\frac{n}{2}) + \mathcal{O}(n) = \mathcal{O}(n \log n)$$

(c) Algorithm

Main Idea the algorithm to get runtime $\mathcal{O}(n)$:

Algorithm 5: FindCitizen(G)

Input: G is a group of which a majority are citizens

```
if size(G) == 1 then
    return the only person
G' ← φ
Split G into pairs
for each pair p = (x, y) do
    if Query(p) == (c, c) then
        G' ← G' ∪ x
return FindCitizen(G')
```

Correctness After each pass through the algorithm, citizens in G' remain in the majority if they were in the majority before the pass. Respectively discuss the three situations.(See in reference)

Runtime Since there is one call to the algorithm of size at most $n/2$, and then use linear time to identify the answer, so by using the Master Theorem:

$$T(n) \leq T(\frac{n}{2}) + \mathcal{O}(n) = \mathcal{O}(n)$$

3 Modular Fourier Transform

(a) $\{1, 2, 3, 4\}$ are the 4^{th} roots of unity(mod 5):

$$1^4 = 1 \equiv 1(\text{mod } 5)$$

$$2^4 = 16 \equiv 1(\text{mod } 5)$$

$$3^4 = 81 \equiv 1(\text{mod } 5)$$

$$4^4 = 256 \equiv 1(\text{mod } 5)$$

for $\omega = 2$:

$$1 + \omega + \omega^2 + \omega^3 = 1 + 2 + 4 + 8 = 15 \equiv 0(\text{mod } 5)$$

(b) Since

$$\omega^0 = 1, \omega^1 = 2, \omega^2 = 4, \omega^3 = 3$$

use the FFT algorithm at the beginning and mod 5 to get the answer: $(0, 4, 4, 2)$

(c) Notice that when $n = 4$, $4^{-1} \equiv 4 (\text{mod } 5)$ i.e. $4 \times 4 \equiv 1 (\text{mod } 5)$. And w^{-1} will be:

$$\omega^0 = 1, \omega^1 = 3, \omega^2 = 4, \omega^3 = 2$$

use the FFT algorithm at the beginning and mod 5 to get $(0, 1, 2, 2)$.
Then, $4 \times (0, 1, 2, 2) \equiv (0, 4, 3, 3) (\text{mod } 5)$

(d) Since we have $\mathbf{FFT}(0, 3, 2, 0) = (0, 4, 4, 2)$ and $\mathbf{FFT}(3, 4, 0, 0) = (2, 1, 4, 0)$.
do the dot product: $(0, 4, 4, 2). * (2, 1, 4, 0) \equiv (0, 4, 1, 0) (\text{mod } 5)$.
Also we have $\mathbf{iFFT}(0, 4, 1, 0) = (0, 4, 3, 3)$, which is the coefficients of the polynomial: $4x + 3x^2 + 3x^3$.

4 Finding Clusters

(a) The algorithm in $\mathcal{O}(|V| + |E|)$:

Algorithm 6: FindClusters(G)

```
while there are unvisted nodes in G do
    run DFS on the temp samllest unvisited node j
    for i in this search 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 $\mathcal{O}(|V| + |E|)$ since the algorithm is just a modification of **DFS**.

(b) 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.