CS170–Fall 2022 — Homework 2 Solutions

Deming Chen cdm@pku.edu.cn

Last modify: October 3, 2022

Collaborators: None

2 Werewolves

(a) The algorithm is to pick everyone else to partner the given person, and ask them whether or not this person is a villager. Since there are always more villagers than there are werewolves, there are more truths than lies. So, if more people says this person is a villager, he or she is a villager. Otherwise, he or she is a werewolf.

However, there is one special case that the number of people who say this person is a villager equal to the number of people who say this person is a werewolf. This case can only occur when there is just one more villagers than werewolves, and the person whom we want to identify is a villager. In this case, we know that he or she is a villager.

(b) The algorithm are described as follows:

Algorithm 1. Find a villager

function search($\mathbf{S}$)
Input: a set $\mathbf{S}$ of all other players. There are more villagers than werewolves or more villagers than werewolves or equal numbers of villagers and werewolves.
Output: an element $p \in \mathbf{S}$ who is a villager.

If $|\mathbf{S}| \leqslant 2$: return any of element in $\mathbf{S}$.
Divide set $\mathbf{S}$ into 2 subsets randomly: $\mathbf{S} = \mathbf{S}_1 \bigcup \mathbf{S}_2$, s.t. $\mathbf{S}_1 \bigcap \mathbf{S}_2 = \varnothing$ and $||\mathbf{S}_1| - |\mathbf{S}_2|| \leqslant 1$.
$p_1 =$ search($\mathbf{S_1}$)
$p_2 =$ search($\mathbf{S_2}$)
Use algorithm from (1) to identify whether p_1 and p_2 are villagers or not. At least one of them is a villager.
Return the villager.

We use induction to prove that if there are more villagers than there are werewolves in $\mathbf{S}$, Algorithm 1 can give us a correct answer.

Base case: if $|\mathbf{S}| \leqslant 2$, since there are always more villagers than there are werewolves, all people are villagers.

Inductive hypothesis: for $|\mathbf{S}| < k$ the algorithm can find a villager correctly ($k \geqslant 3$).

Inductive step: we want to prove that when $|\mathbf{S}| = k$, the algorithm can give a right answer when the number of villagers is larger than the number of the werewolves. Note that at least one subset of $\mathbf{S}_1$ and $\mathbf{S}_2$ meets the condition that: the number of villagers is larger than the number of the werewolves. So at least one of the two people p_1 and p_2 is a villager. And we can identify this person using the algorithm described in (1).

The runtime of Algorithm 1 is

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

(c) A villager can be found in linear time by Algorithm 2.

Algorithm 2. Find a villager in linear time

```
function search(S)
Input:  a set S of all other players.  There are always more villagers
than there are werewolves.
Output:  an element p ∈ S who is a villager.

If |S| ⩽ 2:  return any of element in S.
If |S| is odd:
    Pick one person in S, identify he or her by the algorithm described
in (1).
    If he or she is a villager:  return this person;
    else:  ignore this person in later steps.
Divide all other people into two-people pairs, and query them.  For one
pair, if the two person identify each other as villager, we call it a
good pair.  Otherwise, we call it a bad pair.
Choose one person from each good pair to form a new group S'.
Return search(S').
```

We use induction to prove that if there are more villagers than there are werewolves in $\mathbf{S}$, Algorithm 2 can give us a correct answer.

Base case: if $|\mathbf{S}| \leqslant 2$, since there are always more villagers than there are werewolves, all people are villagers.

Inductive hypothesis: for $|\mathbf{S}| < k$ the algorithm can find a villager in linear time ($k \geqslant 3$).

Inductive step: we want to prove that when $|\mathbf{S}| = k$, the algorithm can give a right answer. Note that the two people in a good pair must be either villagers or werewolves. We can divide all good pairs into villager pairs and werewolf pairs. Since a bad pair consists of a villager and a werewolf, and There are always more villagers than there are werewolves among all people in $\mathbf{S}$, there are more villager pairs than werewolf pairs. So, There are always more villagers than werewolves in $\mathbf{S}'$. Because $|\mathbf{S}'| < |\mathbf{S}|$, according to inductive hypothesis, we can find a village in $\mathbf{S}' \subset \mathbf{S}$.

Runtime analysis: Let $|\mathbf{S}| = n$. Identifying the possible extra person and querying all pairs take time $\mathcal{O}(n)$. Since $|\mathbf{S}'| < n/2$,

$$T(n) < T(n/2) + \mathcal{O}(n). \tag{1}$$

As a result, $T(n) = \mathcal{O}(n)$.

3 The Resistance

Divide all players into $2k$ groups evenly, and choose each group to go on a mission. At least k missions succeed. For groups who fail the mission, divide them again into $2k$ groups, and repeat until there are less than $2k$ players. Then, choose each player to go on a mission individually. After that, we can identify all spies.

Runtime analysis: there are $\log_2(\frac{n}{2k})$ iterations before the number of players reduces to $2k$. In one iteration, we take $2k$ missions. So, we can identify all spies in about

$$2k \cdot \log_2\left(\frac{n}{2k}\right) = \mathcal{O}(k \log\left(\frac{n}{k}\right)) \tag{2}$$

missions.

4 Modular Fourier Transform

(a)

$$\begin{aligned} 1^4 = 1 \equiv 1 \pmod 5 \\ 2^4 = 16 \equiv 1 \pmod 5 \\ 3^4 = 81 \equiv 1 \pmod 5 \\ 4^4 = 256 \equiv 1 \pmod 5 \end{aligned}$$

For $\omega = 2$:

$$1 + \omega + \omega^2 + \omega^3 = 1 + 2 + 4 + 8 = 15 \equiv 0 \pmod 5$$

(b) $f(x) = 0+2x+3x^2+0x^3 = (0+3x^2)+x(2+0x^2) = f_e(x^2)+x\cdot f_o(x^2)$, where$f_e(x) = 0+3x$, $f_o(x) = 2 + 0x$. We evaluate f_e and f_o at 1 and 4: $f_e(1) = 3, f_e(4) = 12 \equiv 2$, $f_o(1) = 2, f_o(4) = 2$.

So,

$$\begin{aligned} f(1) = f_e(1) + f_o(1) = 0 \equiv 0 \pmod 5 \\ f(2) = f_e(4) + 2f_o(4) = 6 \equiv 1 \pmod 5 \\ f(4) = f_e(1) - f_o(1) = 1 \equiv 1 \pmod 5 \\ f(3) = f_e(4) - 2f_o(4) = -2 \equiv 3 \pmod 5 \end{aligned}$$

(c)

5 Pattern Matching

(a) A simple algorithm is to compare all length-n-substrings of s. There are $(m-n+1)$ substrings, and comparing one substring takes $\mathcal{O}(m)$ time. The total time is $\mathcal{O}(nm)$.

(b) Let $a_0 a_1 \cdots a_{n-1}$ and $b_0 b_1 \cdots b_{m-1}$ denote the pattern g and the sequence s respectively. Construct two polynomials:

$$\begin{aligned} P(x) &= (-1)^{a_{n-1}} + (-1)^{a_{n-2}} x + \cdots + (-1)^{a_1} x^{n-2} + (-1)^{a_0} x^{n-1}, \\ Q(x) &= (-1)^{b_0} + (-1)^{b_1} x + \cdots + (-1)^{b_{m-2}} x^{m-2} + (-1)^{b_{m-1}} x^{m-1}. \end{aligned} \tag{3}$$

Algorithm 3 can solve the problem.

Algorithm 3. Pattern Matching by FFT

`Input:` `pattern` g`, sequence` s `and an integer` k

`Output:` `the (starting) locations of all length`$-n$ `substrings of` s `which match` g `in at least` $(n-k)$ `positions.`

`Construct` $P(x)$ `and` $Q(x)$ `as eq.(3).`

`Use FFT to mutiply` $P(x)$ `and` $Q(x)$ `and get`

$$T(x) = \sum_{i=0}^{m+n-2} t_i x^i.$$

`Return all` $j \in \{0, 1, \cdots, m-n\}$ `s.t.` $t_{j+n-1} \geqslant n-2k$`.`

The main body of this algorithm is FFT, so it takes $\mathcal{O}(m \log m)$ time.

Now we show the correctness of Algorithm 3. Consider

$$t_{n-1}, t_n, \cdots, t_{m-1}.$$

Note that for $n-1 \leqslant i \leqslant m-1$,

$$\begin{aligned} t_i &= p_0 q_i + p_1 q_{i-1} + \cdots + p_{n-1} q_{i-n+1} \\ &= (-1)^{a_{n-1}+b_i} + (-1)^{a_{n-2}+b_{i-1}} + \cdots + (-1)^{a_0+b_{i-n-1}}. \end{aligned} \tag{4}$$

If g matches perfectly with the substring starting at $i-n-1$, $t_i = n$. t_i is reduced by 2 each time one of the position in the substring fails to match. So, that the substring matches g in at least $(n-k)$ positions means $t_i \geqslant n-2k$.

As a result, all $j \in \{0, 1, \cdots, m-n\}$ s.t. $t_{j+n-1} \geqslant n-2k$ are just the locations of all length$-n$ substrings of s which match g in at least $(n-k)$ positions.

(c) We can regard A as 0, and C, T, G as 1. For the $j-$th substring of s, we compute the number of positions where g and the substring match $r_j^{(A)}$. Similarly, we can define $r_j^{(C)}$, $r_j^{(T)}$ and $r_j^{(G)}$.

We claim that the number of positions where the gene pattern and the subsequence of DNA match is

$$r_j = \frac{r_j^{(A)} + r_j^{(C)} + r_j^{(T)} + r_j^{(G)}}{2} - n. \tag{5}$$

We only consider the case that g has only one character, for these rs for multi-character strings are the summation over these for a single character. If the two characters are different, say X and Y, then $r_j^{(X)} = r_j^{(Y)} = 0$, and the other two $r_j^{(?)} = 1$. In this case, $r_j = (1 + 1 + 0 + 0)/2 - 1 = 0$. If the two characters are the same, then $r_j^{(A)} = r_j^{(C)} = r_j^{(T)} = r_j^{(G)} = 1$, so $r_j = (1 + 1 + 1 + 1)/2 - 1 = 1$.

This algorithm consists of 4 FFTs, so the runtime is also $\mathcal{O}(m \log m)$.