CS 170 HW03

Jialiang Tang
tangjialiang@berkeley.edu

9/16/2023

1. Study Group

   Jialiang Tang (myself), SID: 3039758308

   Buji Xu, SID: 3039687809

2. Inverse FFT

(a) The degree of C is at most 6.

So $n = 8$.

(b) We compute each element of $M_n(\omega^{-1}) M_n(\omega)$ as below (the subscripts i, j represents the i-th row and j-th column element of the matrix)

$$\left(M_n(\omega^{-1}) M_n(\omega)\right)_{ij} = \sum_{k=1}^{n} \left(M_n(\omega^{-1})\right)_{ik} \cdot \left(M_n(\omega)\right)_{kj}$$

$$= \sum_{k=1}^{n} \omega^{-(j-1)(k-1)} \cdot \omega^{(k-1)(j-1)}$$

$$= \sum_{k=1}^{n} \omega^{(k-1)(j-i)}$$

$$= \sum_{k=1}^{n} \omega_{j-i}^{k-1} \quad (\omega i = \omega_i^i)$$

$$= \begin{cases} n & , \text{if } j-i=0 \\ 0 & , \text{otherwise} \end{cases}$$

(Using formula in PPT of Lec 5. Page 14)

So $\frac{1}{n} M_n(\omega^{-1}) M_n(\omega) = I_n$.

Similarly $\frac{1}{n} M_n(\omega) M_n(\omega^{-1}) = I_n$.

So $\frac{1}{n} M_n(\omega^{-1})$ is the inverse of $M_n(\omega)$.

(c) The only thing we need to prove is that
$\frac{1}{n} M_n(w) M_n(w)^\dagger = \frac{1}{n} M_n(w)^\dagger M_n(w) = I_n$.

$$\left(\frac{1}{n} M_n(w) M_n(w)^\dagger\right)_{ij} = \frac{1}{n}\sum_{k=1}^{n} M_n(w)_{ik}\left(M_n(w)^\dagger\right)_{kj}$$

$$= \frac{1}{n}\sum_{k=1}^{n} w^{(i-1)(k-1)} \cdot \overline{w}^{(j-1)(k-1)}$$

$$= \frac{1}{n}\sum_{k=1}^{n} w^{(k-1)(i-j)} \qquad \left(\overline{w} = \frac{1}{w}\right)$$

$$= \begin{cases} n & , \text{if } i=j \\ 0 & , \text{otherwise} \end{cases} \qquad \text{(similar as (b))}$$

Similarly $\frac{1}{n} M_n(w)^\dagger M_n(w) = I_n$.

So $\frac{1}{\sqrt{n}} M_n(w)$ is unitary

(d)

$$\vec{c} = \begin{pmatrix} c_0 \\ c_1 \\ \vdots \\ c_{n-1} \end{pmatrix} = \frac{1}{n} M_n(w^{-1}) \cdot \begin{pmatrix} C(1) \\ C(w) \\ \vdots \\ C(w^{n-1}) \end{pmatrix}$$

$\vec{c}$ is the vector of coefficients of $C(x)$.

(e) $M_n(\omega^{-1}) = \begin{pmatrix} 1 & 1 & \cdots & 1 \\ 1 & \omega^{-1} & \cdots & \omega^{-(n-1)} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{-(n-1)} & \cdots & \omega^{-(n-1)(n-1)} \end{pmatrix}$

Rearrange $= \begin{pmatrix} 1 & 1 & \cdots & 1 \\ 1 & \omega^{-2} & \cdots & \omega^{-(n-1)+1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{-2(n-1)} & \cdots & \omega^{-(n-1)(n-1)+(n-1)} \end{pmatrix}$

Represent as matrix $= \begin{pmatrix} M_{n/2}(\omega^{-2}) & \omega^{-j} M_{n/2}(\omega^{-2}) \\ \omega^{n} \cdot M_{n/2}(\omega^{-2}) & \omega^{-j}(\omega^{-n/2} \cdot M_{n/2}(\omega^{-2})) \end{pmatrix}$

$\omega^n = 1, \omega^{n/2} = -1$ $= \begin{pmatrix} M_{n/2}(\omega^{-2}) & \omega^{-j} M_{n/2}(\omega^{-2}) \\ M_{n/2}(\omega^{-2}) & -\omega^{-j} M_{n/2}(\omega^{-2}) \end{pmatrix}$

3. Counting k-inversions

Description of Algorithm: Construct two polynomials using coefficients representation, noted as $A(x)$ and $B(x)$ (Both are $(n-1)$-degree). The coefficient $a_{n-1-i}=1$ if the i-th index of bitstring b is 1, $b_j=1$ if the j-th index of b is 0. Otherwise a_i and b_j are 0. Then we compte the coefficients of $A \cdot B$ using FFT and iFFT.
For all k from 1 to $n-1$, the coefficient of x^{n-1+k} is the number of k-inversions

Correctness Proof: The coefficient of $x^{n+j-i-1}$ comes from all the products of x^{n-i-1} and x^j. For each $j-i=k$, the total number equals to number of all the pairs (x^{n-i-1}, x^j) making $x^{n+j-i-1}=x^{n+k-1}$, which is exactly the coefficient of x^{n+k-1} of $A \cdot B$.

Runtime Analysis: We need $O(n)$ to define A and B, $O(n \log n)$ to compute C, $O(n)$ to output results.
In conclusion, the time complexity is $O(n \log n)$.

4. Protein Matching

(a) For each $i \in \{0, 1, \cdots, m-n-1\}$ starting points in s, check if the substring $s[i : i+n-1]$ differs with g in some positions. The check takes $O(n)$ time at each of $O(m)$ starting points, so time complexity is $O(mn)$.

(b) (It's same with T4 of discussion 3 with $k=0$).

Description of Algorithm: Let g' be g with 0's replaced by -1's. Let

$$p_1(x) = a_0 + a_1 x + \ldots + a_{n-1} x^{n-1}$$

where $a_d = g'(n-d-1)$ for all $d \in \{0, 1, \cdots, n-1\}$. Similarly, let

$$p_2(x) = b_0 + b_1 x + \ldots + b_{m-1} x^{m-1}$$

where $b_d = s'(d)$ for all $d \in \{0, 1, \cdots, m-1\}$, where again s' is s with 0's replaced by -1's.

Now consider $p_3(x) = p_1(x) \cdot p_2(x) = c_0 + c_1 x + \ldots$

The coefficient of x^{n-1+j} in $p_3(x)$ is

$$c_{n-1+j} = \sum_{i=0}^{n-1} a_{n-1-i} b_{j+i} = \sum_{i=0}^{n-1} g'(i) s'(j+i)$$

for any $j \in \{0, 1, \cdots, m-n\}$.

The coefficient will be n if and only if g perfectly match s at starting point j.

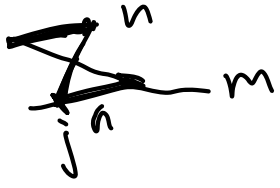
Thus, we just construct P_1, P_2 then compute P_3 using FFT and iFFT. Then output the number of $C_{n-1+j} = n$ for all $j \in \{0, 1, \cdots, m-n\}$.

Proof of correctness: Consider mapping function Φ s.t.

$\Phi(0) = -1$, $\Phi(1) = 1$.

Construct the polynomial $G(x) = \sum_i G_i x^i$ from g by setting $G[i] = \Phi(g[n-1-i])$. Similarly, construct $S(x) = \sum_i S_i x^i$ from s by setting $S[i] = \Phi(s[i])$.

Then we have



$$G(x) \cdot S(x) = \left(\sum_{i=0}^{n-1} G[i] x^i\right)\left(\sum_{j=0}^{m-1} S[j] x^j\right)$$

of x^{n-1+j}

The coefficient comes from $\sum_{i=0}^{n-1} G[n-1-i] \cdot S[j+i]$

$$= \sum_{i=0}^{n-1} \Phi(g[i])\Phi(s[j+i]) \quad (*)$$

for $j \in \{0, 1, \cdots, m-n-1\}$.

Since $\Phi(g[i])\Phi(s[j+i]) = \begin{cases} 1, & \text{if } g[i] = s[j+i] \\ 0, & \text{if } g[i] \neq s[j+i] \end{cases}$

The $(*)$ equals to n if and only if for each $i \in \{0, 1, \cdots, n-1\}$,

$g[i] = s[j+i]$, which means g matches s at starting point j.

Time Complexity: FFT and iFFT takes $O(m \log m)$, the construction of poly takes $O(m)$, finding the result in p_3 takes $O(m)$ time.

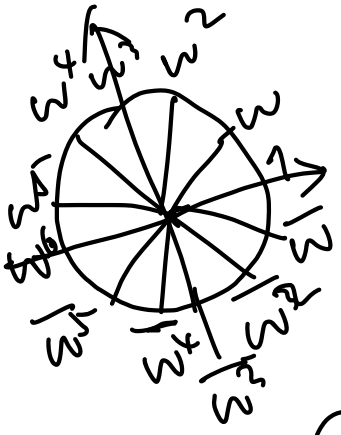
Overall, it takes $O(m \log m)$ time.

Note: The answer refers to solution of discussion 3.

(c) Description of Algorithm:

Construct mapping function Φ_g: $\Phi_g(c_i) = w^{i-1}$, where w is α-root complex number s.t. $z^\alpha = 1$. While $\Phi_s(c_i) = \overline{w^{i-1}}$

Let g' be the string g with each character η replaced by $\Phi_g(\eta)$. s' be the string s with each character replaced by $\Phi_s(\$)$. Let $p_1(x) = a_0 + a_1 x + \ldots + a_{n-1} x^{n-1}$, where $a_d = g'(n-d-1)$ for $d \in \{0, 1, \cdots, n-1\}$



$p_2(x) = b_0 + b_1 + \ldots + b_{m-1} x^{m-1}$ where $b_d = s'(d)$ for $d \in \{0, 1, \cdots, m-1\}$.

Consider $p_3(x) = p_1(x) \cdot p_2(x) = c_0 + c_1 x + \cdots$

The coefficient of x^{n-1+j} in $p_3(x)$ is

$$c_{n-1+j} = \sum_{i=0}^{n-1} a_{n-1-i} b_{j+i} = \sum_{i=0}^{n-1} g'(i)\, s'(j+i)$$

for all $j \in \{0, 1, \cdots, m-n\}$.

The coefficient will be n if and only if g perfectly match s at starting point j.

Thus, we just construct p_1, p_2 then compute p_3 using FFT and iFFT. Then output the number of $c_{n-1+j} = n$ for all $j \in \{0, 1, \cdots, m-n\}$.

Proof or correctness: The proof is similar to that in (b). We only need to prove that $c_{n-1+j} = n$ if and only if g matches s at starting point j.

Since $c_{n-1+j} = \sum_{i=0}^{n-1} g'(i)\, s'(j+i)$, we analyse the real part of c_{n-1+j}:

$$\begin{aligned} \mathrm{Re}(c_{n-1+j}) &= \sum_{i=0}^{n-1} \mathrm{Re}(g'(i)\, s'(j+i)) \\ &\leq \sum_{i=0}^{n-1} 1 = n \end{aligned}$$

The equalvalance holds only when $\mathrm{Re}(g'(i)\, s'(j+i)) = 1$ for each i

Recall that $|g'(i)| = |s'(j+i)| = 1$, we have:

$$\mathrm{Re}(g'(i)s'(j+i)) = 1 \iff g'(i)s'(j+i) = 1$$

$$\iff s'(j+i) = \overline{g'(i)}$$

$$\iff s(j+i) = g(i)$$

for each $i \in \{0, 1, \cdots, n-1\}$

So $C_{n-1+j} = n$ iff g matches s at starting point j.

5. Triple sum

Description of Algorithm: Construct 3 (all-the same) polynomials:

$$P_1(x) = \sum_{a \in A} x^a$$
$$\| \quad P_2(x)$$
$$\| \quad P_3(x)$$

Consider $p(x) = P_1(x) \cdot P_2(x) \cdot P_3(x)$, the coefficient of x^n is the number of tuple (i, j, k) satisfying $A[i] + A[j] + A[k] = n$.

Proof of Correctness: For each tuple (i, j, k) satisfying $A[i]+A[j]+A[k]=n$, $x^{A[i]} \cdot x^{A[j]} \cdot x^{A[k]} = x^n$ so the tuple contributes 1 to the coefficient of x^n of $p(x)$. On the other hand, for each (i, j, k) not satisfying $A[i] + A[j] + A[k] = n$, $x^{A[i]} \cdot x^{A[j]} \cdot x^{A[k]} \neq x^n$ so it won't influence the coefficient of x^n.

To sum, the coefficient of x^n is exactly the number of tuple (i, j, k) satisfying $A[i] + A[j] + A[k]$.

Runtime Analysis: Construction of polys need $O(n)$, FFT and iFFT need $O(n \log n)$, so the overall runtime is $O(n \log n)$.