Problem Set 4

Due dates: Electronic submission of the pdf file of this homework is due on **2/16/2024 before 11.59pm** on canvas.

Name: Jawahar Sai Nathani

Resources. (All people, books, articles, web pages, etc. that have been consulted when producing your answers to this homework)

On my honor, as an Aggie, I have neither given nor received any unauthorized aid on any portion of the academic work included in this assignment. Furthermore, I have disclosed all resources (people, books, web sites, etc.) that have been used to prepare this homework. The solutions given in this homework are my own work.

Signature: Jawahar Sai Nathani

Make sure that you describe all solutions in your own words. Typeset your solutions in LaTeX. Read chapter 30 on "Polynomials and the FFT" and chapter 14 on "Dynamic Programming" in our textbook.

Problem 1. (20 points) Let ω be a primitive nth root of unity. The fast Fourier transform implements the multiplication with the matrix

$$F = (\omega^{ij})_{i,j\in[0..n-1]}.$$

Show that the inverse of the matrix F is given by

$$F^{-1} = \frac{1}{n}(\omega^{-jk})_{j,k\in[0..n-1]}$$

[Hint: $x^n - 1 = (x-1)(x^{n-1} + \cdots + x + 1)$, so any power $\omega^\ell \neq 1$ must be a root of $x^{n-1} + \cdots + x + 1$.] Thus, the inverse FFT, called IFFT, is nothing but the FFT using ω^{-1} instead of ω, and multiplying the result with $1/n$.

Solution. Let assume $F = (\omega^{ik})_{i,k\in[0..n-1]}$ and $F^{-1} = \frac{1}{n}(\omega^{-kj})_{j,k\in[0..n-1]}$ where w is nth root of unity.
According to the fundamental property of matrix multiplication and inverses $F.F^{-1} = I$

$$\begin{aligned}(F.F^{-1})_{ij} &= \frac{1}{n}\sum_{k=0}^{n-1}\omega^{ik}.\omega^{-kj} \\ &= \frac{1}{n}\sum_{k=0}^{n-1}\omega^{(i-j)k}\end{aligned}$$

$\underline{\text{If } i = j}$

$$(F.F^{-1})_{ij} = \frac{1}{n}\sum_{k=0}^{n-1}\omega^{0*k} = \frac{1}{n}\sum_{k=0}^{n-1}1 = \frac{1}{n}(n) = 1$$

$\underline{\text{If } i \neq j}$, Let $i - j = p$

$$\begin{aligned}(F.F^{-1})_{ij} &= \frac{1}{n}\sum_{k=0}^{n-1}\omega^{pk} \\ &= \frac{1}{n}(1 + \omega^p + \omega^{2.p} + \omega^{3.p} + \cdots + \omega^{(n-1).p})\end{aligned}$$

$(\omega^p - 1)(1 + \omega^p + \omega^{2.p} + \omega^{3.p} + \cdots + \omega^{(n-1).p}) = \omega^{np} - 1 = (\omega^n)^p - 1 = 1^p - 1 = 0$
As $p = i - j$ and $i \neq j$, p can never be a multiple of n. Hence $\omega^p - 1 \neq 0$ for any p. This implies $1 + \omega^p + \omega^{2.p} + \omega^{3.p} + \cdots + \omega^{(n-1).p} = 0 \; \forall \; p$.
Therefore,

$$(F.F^{-1})_{ij} = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases}$$

This is the definition of Identity matrix I. As $F.F^{-1} = I$, our assumption $F^{-1} = \frac{1}{n}(\omega^{-jk})_{j,k\in[0..n-1]}$ is correct. Hence Proved.

Problem 2. (20 points) Describe in your own words how to do a polynomial multiplication using the FFT and IFFT for polynomials $A(x)$ and $B(x)$ of degree $\leq n-1$. Make sure that you describe the length of the FFT and IFFT needed for this task. Be concise and precise. Illustrate how to multiply the polynomials $A(x) = x^2 + 2x + 1$ and $B(x) = x^3 + 2x^2 + 1$ using this approach.

Solution. Let's consider the polynomials $A(x) = x^2 + 2x + 1$ and $B(x) = x^3 + 2x^2 + 1$

- Extract the coefficients of both the polynomials. That would be (1, 2, 1) for $A(x)$ and (1, 2, 0, 1) for $B(x)$.
- Degree of $A(x)$ is 2 and of $B(x)$ is 3. Degree of resulting polynomial after multiplying these 2 would be 2 + 3 = 5.
- Length of FFT and IFFT will be d = 2^n where d $\geq$ degree of resulting polynomial. Value greater than 5 and in the form of 2^n is 8. So length of FFT and IFFT would be 8.
- Add padding to coefficients of $A(x)$ and $B(x)$ to make the length as 8.
- Implement FFT on coefficients and convert them into point-value representation.
- Perform point vise multiplication on resulting points of each polynomial generated after FFT. The new points are point-value representation for $A(x).B(x)$.
- Implement IFFT on new points to convert point-value representation into coefficient representation. The obtained values are the coefficients of $A(x).B(x)$.
- Coefficients of $A(x).B(x)$ would be (1, 4, 5, 3, 2, 1). Time complexity of above polinomial multiplication would be $O(n \log n)$.

Therefore, $A(x).B(x) = x^5 + 4x^4 + 5x^3 + 3x^2 + 2x + 1$.

Problem 3. (20 points) How can you modify the polynomial multiplication algorithm based on FFT and IFFT to do multiplication of long integers in base 10? Make sure that you take care of carries in a proper way. Write your algorithm in pseudocode and give a brief explanation.

Solution. Let's consider 2 long integers in base 10 - 232 and 15. These numbers can be multiplied using FFT and IFFT as following.

- Consider both the numbers as polynomials where x will be 10. $232 = 2 * 10^2 + 3 * 10^1 + 2$ and $15 = 1 * 10^1 + 5$.
- Take the coefficients of both the polynomials - (2, 3, 2) and (1, 5).
- Perform FFT and IFFT on these coefficients to get coefficients of the product 232×15.
- Multiply each coefficient with respective power of 10 to obtain the final product of 2 integers.
- Coefficients after FFT and IFFT would be (2, 13, 17, 10). Exact value of product 232×15 is $2 * 10^3 + 13 * 10^2 + 17 * 10^1 + 10 = 3480$

Psuedocode

```
def product(A, B) {
    // Convert A and B into polynomials and extract coefficients
    A_poly = conv_poly(A)
    B_poly = conv_poly(B)

    // Add zero padding according to length of FFT
    A_pad = pad(A_poly)
    B_pad = pad(B_poly)

    // Apply FFT on A_pad and B_pad
    A_fft = FFT(A_pad)
    B_fft = FFT(B_pad)

    // Multiplly A_fft and B_fft
    C_fft = pointwise_prod(A_fft, B_fft)

    // Apply IFFT
    C = IFFT(C_fft)

    // Calculate product
    sum = 0
    for (int i = 0; i < len(C); i++)
        sum += C[len(C) - i - 1] * 10^i
    return sum
}
```

Problem 4 (20 points)**.** As stated, in dynamic programming we first solve the subproblems and then choose which of them to use in an optimal solution to the problem. Professor Capulet claims that we do not always need to solve all the subproblems in order to find an optimal solution. She suggests that we can find an optimal solution to the matrix-chain multiplication problem by always choosing the matrix A_k at which to split the subproduct $A_i A_{i+1} \cdots A_j$ (by selecting k to minimize the quantity $p_{i-1} p_k p_j$) before solving the subproblems. Find an instance of the matrix-chain multiplication problem for which this greedy approach yields a suboptimal solution.

Solution. Let's consider 3 matrices A_1 of size 4×7, A_2 of size 7×5 and A_3 of size 5×3.

- According to greedy algorithm $4 \times 5 \times 3$ gives the least value. Hence the matrices should be split between A_2 and A_3. No of scalar multiplications for this split $(A_1.A_2)A_3$ is 200.
- If we spit the matrices between A_1 and A_2 then the no of scalar multiplications for $A_1(A_2.A_3)$ would be 189. This requires less scalar multiplication compared to the greedy approach.

Therefore, the above greedy approach doesn't always yield an optimal solution.

Problem 5 (20 points)**.** Determine an LCS of $X = \langle R, H, U, B, A, R, B\rangle$ and $Y = \langle S, T, R, A, W, B, E, R, R, Y\rangle$ using the dynamic programming algorithm that was discussed in class. Make sure that you explain your answer step-by-step, in detail, rather than just giving an LCS. [Make sure that you list X vertically, and Y horizontally when constructing the table. If $x_i \neq y_j$, and $c[i-1, j] = c[i, j-1]$, choose $c[i-1, j]$. Explain row-by-row how the table is constructed.]

Solution. Let $X = \langle R, H, U, B, A, R, B\rangle$
Let $Y = \langle S, T, R, A, W, B, E, R, R, Y\rangle$
Using the following algorithm, the length of the LCS between the two sequences can be determined.

1. Initialize a table L with 0's of size $(m+1) \times (n+1)$, where m and n are the lengths of the sequences X and Y, respectively.
2. Iterate through the sequences X and Y, and for each pair of elements (x, y) do the following:
   (a) If x equals y, set $L[i][j] = L[i-1][j-1] + 1$
   (b) If x does not equal y, set $L[i][j] = \max(L[i-1][j], L[i][j-1])$
3. The length of the LCS is given by $L[m][n]$.

LCS	␣	S	T	R	A	W	B	E	R	R	Y
␣	0	0	0	0	0	0	0	0	0	0	0
R	0	0	0	1	1	1	1	1	1	1	1
H	0	0	0	1	1	1	1	1	1	1	1
U	0	0	0	1	1	1	1	1	1	1	1
B	0	0	0	1	1	1	2	2	2	2	2
A	0	0	0	1	2	2	2	2	2	2	2
R	0	0	0	1	2	2	2	2	3	3	3
B	0	0	0	1	2	2	3	3	3	3	3

To find the actual LCS string, start in the bottom right corner of the above matrix and traverse backwards through the table following the path of the larger values until you reach the top or left edge of the table.
The elements in the sequence that correspond to the table cells you visit during this backtracking process will form the LCS string.
The traversal algorithm is as follows

1. If the cell value directly to the left and directly above are the same as the current cell, go up and to the left by one.
2. If this condition fails, go directly up.

LCS string for above X and Y is $\langle R, B, R\rangle$ or $\langle R, A, R\rangle$.

Checklist:

✓ Did you add your name?

✓ Did you disclose all resources that you have used?
(This includes all people, books, websites, etc. that you have consulted)

✓ Did you sign that you followed the Aggie honor code?

✓ Did you solve all problems?

✓ Did you submit the pdf file of your homework?