CS6013: Advanced Data Structures and Algorithms

Programming Assignment I (out of 10 marks)
(Start Date: Sunday, 18 Sep 2022)
(Submission Deadline: 11:59 pm, Saturday, 1 Oct 2022)

Multiplying two polynomials

Let $n \leq 15$. The problem is about multiplying two polynomials of degree at most n. In the main program, we declare two arrays namely, $poly\_1[16]$ and $poly\_2[16]$. Let deg_1, deg_2 be the degrees of these polynomials, respectively. The program asks the user to input the values of deg_1 and deg_2 and then input the `integer` coefficients of the two polynomials which are stored in the arrays defined above. The aim is to compute the product polynomial and print it. We compute the product in two ways. The first method is to do a straightforward (or naive) product of the coefficients of $poly\_1$ and $poly\_2$ and store the coefficients of the product polynomial in the array $naive\_prod[32]$. The second method uses the Fast Fourier Transform based technique which we learned in class. The coefficients of the product polynomial obtained using this method are stored in the array $fft\_prod[32]$. More details of the program are given below:

- In the main function, the user is asked to input the degree of polynomial 1. It is stored in the variable deg_1. Then, the user is asked to enter deg_1+1 integer coefficients of each monomial in the first polynomial in the increasing order of the degree of the monomials. These are stored in the integer array $poly\_1[16]$. Repeat the above steps to enter and store the details of the second polynomial. Its degree is stored in the integer variable deg_2 and its coefficients are stored in the integer array $poly\_2[16]$.

- Write a function $naive\_polynomial\_multiplication()$ that takes as input $poly\_1[]$ and $poly\_2[]$ and returns the product polynomial computed in a straightforward manner (we know this takes $O(n^2)$ time).

- Write a function $add\_complex\_num()$ which takes as arguments 4 integers a, b, c, and d. It returns the sum of the complex numbers $a + ib$ and $c + id$ which is stored in an integer array $sum[2]$, where $sum[0]$ contains the real part of the sum and $sum[1]$ contains the imaginary part of the sum.

- Write a function $multiply\_complex\_num()$ which takes as arguments 4 integers a, b, c, and d. It returns the product of the complex numbers $a + ib$ and $c + id$ which is stored in an integer array $mult[2]$, where $mult[0]$ contains the real part of the product and $mult[1]$ contains the imaginary part of the product.

- $find\_N()$ is a function that takes as arguments deg_1 and deg_2 and computes the smallest power of 2 greater than or equal to $deg_1 + deg_2 + 1$. The value returned by this function is stored in the main function in a variable called N.

- The function $find\_complex\_roots()$ takes N as an argument and computes the N different N-th roots of unity and store them in a `global` array named $roots[N][2]$, where $roots[i][0]$ denotes the real part of the i-th root of unity and $roots[i][1]$ denotes the imaginary part of the i-th root of unity. For example, for any $0 \leq i < N$, $root[i][0] = \cos(\frac{2\pi i}{N})$ and $root[i][1] = \sin(\frac{2\pi i}{N})$.

- The function $Eval(poly[], N)$ (the recursive function that was taught in the class) evaluates the polynomial $poly$ at the N complex roots of unity which are available in the `global` array $roots[][]$.

This function makes calls to the functions *add_complex_num*() and *multiply_complex_num*() internally. It returns a 2D array of N values which correspond to the evaluation of *poly*[] at $\omega_N^0, \omega_N^1, \ldots, \omega_N^{N-1}$.

- The main program calls $Eval(poly\_1[], N)$ and stores the values returned by this function call in an array $poly\_1\_evaluations[N][2]$ (the 2D array here is to store both the real and imaginary part of each evaluation which could be a complex number). The main program calls $Eval(poly\_2[], N)$ and stores the values returned by this function call in an array $poly\_2\_evaluations[N][2]$.
- The function $product\_polynomial\_evaluations(poly\_1\_evaluations[][], poly\_2\_evaluations[][])$ takes as input the evaluations of polynomials *poly_1* and *poly_2* and returns the evaluations of the product polynomial at the N complex roots of unity. The values returned are stored in an array named $product\_poly\_evaluations[N][2]$.
- As taught in the class, we again use the *Eval*() function (inverse FFT) to compute the coefficients of the product polynomial. The coefficients thus obtained are stored in the main program in the array $fft\_prod[32]$. Finally, the program prints the product polynomial using the function given below.
- The function *print_poly*() takes as argument the coefficient array and degree of a polynomial and prints the polynomial in an easy-to-read form on the screen.

Sample Output:

Enter the degree of the first polynomial: 4
Enter the 5 coefficients of the first polynomial in the increasing order of the degree of the monomials they belong to:
-1 0 3 0 1
Enter the degree of the second polynomial: 3
Enter the 4 coefficients of the second polynomial in the increasing order of the degree of the monomials they belong to:
0 0 2 3
The first polynomial is:
x*4 + 3x*2 -1
The second polynomial is:
3x*3 + 2x*2
The product of the two polynomials obtained via naive polynomial multiplication is:
3x*7 + 2x*6 + 9x*5 + 6x*4 - 3x*3 - 2x*2
The product of the two polynomials obtained via polynomial multiplication using FFT is:
3x*7 + 2x*6 + 9x*5 + 6x*4 - 3x*3 - 2x*2

Program Related Instructions

1. You can write your program in one of C, C++, Java, or Python.
2. The programmer has the freedom to decide whether (s)he would like to declare a variable as `local` or `global`. In other words, you don't have to strictly adhere to the function prototypes defined here. You may choose not to pass certain variables as parameters to a function by declaring them `global`.
3. You are free to define more variables and functions. What is provided above is just an outline of the program.

Submission Guidelines

1. Your submission will be one zip file named <roll-number>.zip , where you replace roll-number by your roll number (e.g. cs22mtech11003.zip), all in small letters. The compressed file should contain the below mentioned files:

(a) Programming files (please do not submit python notebooks or IDE files). **The entire source code has to be in one file named main_prog.c (or main_prog.cpp, or ...).**

(b) **No need to submit a report**. However, if you wish you may submit a text/doc file giving a detailed description of your program. No marks for this.

(c) Upload your zip file in Google Classroom at Classwork→Programming Assignment 1. No delays permitted.

2. Failure to comply with instructions (file-naming, upload, input/output specifications) will result in your submission not being evaluated (and you being awarded 0 for the assignment).

3. **Plagiarism policy**: If we find a case of plagiarism in your assignment (i.e. copying of code, either from the internet, or from each other, in part or whole), you will be awarded a zero and will lead to a FR grade for the course in line with the department Plagiarism Policy (https://cse.iith.ac.in/academics/plagiarism-policy.html). Note that we will not distinguish between a person who has copied, or has allowed his/her code to be copied; both will be equally awarded a zero for the submission.

Evaluation Scheme

Your assignment will be awarded marks based on the following aspects:

- Code clarity (includes comments, indentation, naming of variables and functions, etc.): 1 mark
- Perfect output: 3 mark
- Logic in the code of the function Eval(): 3 marks.
- Logic in the code of the functions *naive_polynomial_multiplication*(), $find\_N()$ and *print_poly*() : $1 + 1 + 1 = 3$ marks.