Question 1

Devansh Kumar Jha

2021–08–30

1 Programming problem 1

1.1 Problem Description

We are required to add the polynomials according to the given input where we are first given two integers n and m which are followed by 2n and 2m integers in the next two lines. The main complexities and boundary cases in the problem were to design a O(m+n) algorithm and to recognize that zero polynomials are the exceptional cases which do not directly fit into the algorithm and so are to be specifically handled. To take care of integer overflow we by default use big integers as our storage containers.

1.2 Data Structure Usage

We will be using a dynamically allocated doubly linked list with a sentinel node for easier and error free implementation.

1.3 Algorithm Explanation

The algorithm is quite simple. We just take two temprorary node pointers which will correspond to the current term of that particular polynomial in the addition process. There are 3 major cases as follows -

- **1st polynomial already traversed**
  In this case we just add the term of 2nd polynomial to the answer polynomials doubly linked list. Here we don't need to check anything as the question statement says that the coefficients will anyways be non-zero.
- **2nd polynomail already traversed**
  Similar to the case above with the difference that this time we will be traversing the 1st polynomial and adding new nodes to the resultant polynomial.
- **Both polynomials are being traversed**
  This is the most complex case of the mentioned three with some exceptions to be handled within. While we are simultaneously traversing the 1st and

2nd polynomial the currently checked term might have different variable exponents. In that case we cannot add them and we simply need to add one of them to the list. We chose to add the element with smaller exponent into the list as it ensures the automatic sorting of the resultant polynomial. In case of the exponents being equal we need to confirm that the coefficients don't add to zero. In case they do than we just skip these terms however if not than we add them. So it can easily be seen that in this case the iteration would be a bit more costly as compared 1st and 2nd case iteration.

1.4 Pseudo Code

These are the variables used -

- **HEAD** - The first element of addition polynomial.
- **HEAD1** - The first element of first polynomial.
- **HEAD2** - The first element of second polynomial.
- **SENT** - Sentinel node for addition polynomial.

****** - Similar goes for TAIL,TAIL1,TAIL2,SENT1,SENT2

Algorithm 1: List Making Algorithm

```
Input  : The two linked lists showing p(x) and q(x) and pointers to
         make output polynomial r(x)
Output: No output
temp1 = HEAD1 and temp2 = HEAD2
/* Temprorary pointers to traverse along the available
   polynomials.                                                  */
while temp1! = SENT1 or temp2! = SENT2 do
    if temp1 == SENT1 then
        TAIL.insert − node(temp2)    // insert-node() is a function
         which makes a new node and adds to the list
        temp2 ← (temp2 → next)
    else
        if temp2 == SENT2 then
            TAIL.insert − node(temp1)
            temp1 ← (temp1 → next)
        else
            if temp1 → exp == temp2 → exp then
                if temp1 → cof! = temp2 → cof then
                    a=make-
                      node(temp1 → cof + temp2 → cof,temp1 → cof)
                      /* make-node() function allocates memory for a
                      new node and fills with the parameters given */
                    TAIL.insert − node(a)
                else
                end if
                temp1 ← (temp1 → next)
                temp2 ← (temp2 → next)
            else
                if temp1 → exp < temp2 → exp then
                    TAIL.insert − node(temp1)
                    temp1 ← (temp1 → next)
                else
                    TAIL.insert − node(temp2)
                    temp2 ← (temp2 → next)
                end if
            end if
        end if
    end if
end while
return
```