Algorytmy i struktury danych
Projekt zaliczeniowy

Temat: „Implementacja wielomianów na bazie tablic"

Autor: Tomasz Różycki

Wstęp:

Niniejszy projekt ma na celu zaimplementowanie wielomianów na bazie tablic oraz działań: dodawania, odejmowania, mnożenia i obliczania wartości wielomianu schematem Hornera.

Definicja wielomianu:

Wielomian (inaczej suma algebraiczna) – jest to wyrażenie algebraiczne będące sumą jednomianów. Dla nieujemnej liczby całkowitej n *wielomianem stopnia n zmiennej x* jest wyrażenie postaci:

$$a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0 = \sum_{i=0}^{n} a_i x^i$$

gdzie $a_0, a_{1,} \dots, a_n$ są współczynnikami wielomianu oraz $a_n \neq 0$.

Algorytm Hornera:

Jest to sposób obliczania wartości wielomianu dla danej wartości argumentu **wykorzystujący minimalną liczbę mnożeń**. Mając wielomian w postaci:

$$W(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$$

To obliczając jego wartość dla zadanego ***x*** bezpośrednio z zadanego wzoru należy wykonać $1 + 2 + 3 + \cdots + (n-1) + n = \frac{n(n+1)}{2}$ mnożeń i dodawań.

Tymczasem proste przekształcenie:
$W(x) = a_0 + x(a_1 + x(a_2 + \cdots + x(a_{n-1} + x a_n) \dots))$

Sprawia, że wystarczy jedynie $\boldsymbol{n}$ mnożeń i $\boldsymbol{n}$ dodawań.

Dodawanie, odejmowanie i mnożenie wielomianów:

Dodawanie i odejmowanie są to metody intuicyjne, wykonujemy działania na współczynnikach przy odpowiednich potęgach zmiennej *x.* W przypadku mnożenia warto zauważyć, że stopień nowopowstałego wielomianu jest sumą stopni mnożonych wielomianów. Operacje te można wykonywać w sposób analogiczny do operacji na pozycyjnym systemie liczbowym.

Opis programu:

Definicja typów:

```
class Polynomial
{
private:
    double *coefficient_arr;
    int arr_size; //rozmiar tablicy jest o jeden wiekszy od stopnia wielomianu
    //przyklad:
    //int stopien = 2
    //x^2 + x + c --> trzy wspolczynniki

public:
    //konstruktor - tworzy wielomian zainicjalizowany na zero
    Polynomial(int degree);

    //konstruktor - tworzy wielomian i inicjalizuje wartosci
    Polynomial(double *arr, int coefficient);

    //destruktor
    ~Polynomial();

    //dodaje jeden wielomian do drugiego
    Polynomial Add(Polynomial &other);

    //odejmuje wielomiany
    Polynomial Substract(Polynomial &other);

    //mnozenie wielomanow
    Polynomial Multiply(Polynomial &other);

    //zwraca stopien wielomanu
    int degree();

    //obliczanie wartosci wielomianu metoda Hornera
    double Horner(double value);

    //wyswietla wielomian
    void displayPolynomial();
};
```

Najważniejsze algorytmy:

1) Dodawanie i odejmowanie

```
Polynomial Polynomial::Add(Polynomial &other)
{
    //sprawdzamy ktory wielomian ma wiekszy stopien
    int resultSize = (arr_size >= other.arr_size) ? arr_size : other.arr_size;
    //tworzymy nowy obiekt, stopien o 1 mniejszy od arr_size
    Polynomial result(resultSize - 1);

    if (arr_size >= other.arr_size)
    {
        for (int i = 0; i < other.arr_size; i++)
        {
            //jesli other.arr ma mniejszy lub rowny stopien to dodaje tylko
wspolczynniki do rozmiaru other.arrsize
            result.coefficient_arr[i] = coefficient_arr[i] +
other.coefficient_arr[i];
        }
        for (int i = other.arr_size; i < arr_size; i++)
        {
            //pozosotale wspolczynniki przepisuje
            result.coefficient_arr[i] = coefficient_arr[i];
        }
    }
    else
    {
        for (int i = 0; i < arr_size; i++)
        {
            result.coefficient_arr[i] = coefficient_arr[i] +
other.coefficient_arr[i];
        }
        for (int i = arr_size; i < other.arr_size; i++)
        {
            result.coefficient_arr[i] = other.coefficient_arr[i];
        }
    }
    return result;
}
```

Odejmowanie zostało zaimplementowane w sposób analogiczny do dodawania, lecz zamiast dodawać odpowiednie współczynniki, odejmujemy je. Wymusiło to zmianę znaku + na – w trzech miejscach.

Złożoność:

Niech N = stopień większego wielomianu + 1

O(N)
$\Delta(N) = 0$

2) *Mnożenie*

```
Polynomial Polynomial::Multiply(Polynomial &other)
{
    //najwyzsza potega wielomianu to suma rozmiaru dwoch tablic - 2(bo
wspolczynnik x^0)
    int result_degree = arr_size + other.arr_size - 2;
    //tworzymy nowy obiekt
    Polynomial result(result_degree);

    //mnozenie polega na wymnozeniu kazdego elementu wielomianu_1 z
wielomianem_2
    for (int i = 0; i < arr_size; i++) //przegladamy wszystkie elementy
wielomianu_1
    {
        for (int j = 0; j < other.arr_size; j++) //przegladamy wszyskie
elementy wielomianu_2
        {
            //[i+j] poniewaz przy mnozeniu poteg, jest to suma wspolczynnikow
            result.coefficient_arr[i + j] += (coefficient_arr[i] *
other.coefficient_arr[j]);
        }
    }
    return result;
}
```

Złożoność:

Oznaczmy rozmiar tablicy przechowujący współczynniki wielomianu:

1. n = arr_size
2. m = other.arr_size

O(n *m)

$\Delta(n) = 0$

3) *Algorytm Hornera*

```
double Polynomial::Horner(double x)
{
    double result = 0;
    if (arr_size == 1){
        result = coefficient_arr[0];
    }
    else
    {
        result = coefficient_arr[arr_size - 1];
        for (int i = arr_size - 2; i >= 0; i--)
        {
            //a_n * x + a_(n-1)
            result = result * x + coefficient_arr[i];
        }
    }
    return result;
}
```

Złożoność:

Niech N = rozmiar tablicy współczynników

O(N)

$\Delta(N) = 0$

Analiza algorytmu:

Przypomnijmy uproszczenie postaci wielomianu:

$$W(x) = a_0 + x(a_1 + x(a_2 + \cdots + x(a_{n-1} + xa_n) \ldots))$$

Wykonywanie działań zaczynamy od „najbardziej zagłębionego” nawiasu, mnożymy współczynnik przy najwyższej potędze razy argument x oraz dodajemy współczynnik przy o jeden mniejszej potędze.

<u>Kompilacja i uruchomienie:</u>

g++ main.cpp library.cpp -o main

main.exe

Następnie program w sposób intuicyjny prosi użytkownika o wprowadzenie danych.