資料結構報告

李柏勳

August 21, 2024

CONTENT

CHAPTER 1 解題說明

(一)從 HW2 延伸做的 polynomial 需實現環狀串列

想法陳述：為了確保敘述流暢，加法、乘法、減法、以及 Eval 的敘述及虛擬碼是使用 ChatGPT 所撰寫。

1. 加法的敘述：

加法操作對應於兩個多項式相加的過程。它會遍歷兩個多項式的項，並將具有相同指數的項相加，生成一個新的多項式作為結果。如果一個多項式中的項比另一個多項式長，剩下的項將直接附加到結果中。

```
Function Add(Polynomial A, Polynomial B) -> Polynomial Result
    Initialize Result as an empty polynomial
    i = A.first.next
    j = B.first.next

    While i != A.first and j != B.first do
        If i.exp == j.exp then
            Add a new term with coefficient (i.coef + j.coef) and exponent i.exp to Result
            Move i and j to their next terms
        Else if i.exp > j.exp then
            Add a new term with coefficient i.coef and exponent i.exp to Result
            Move i to its next term
        Else
            Add a new term with coefficient j.coef and exponent j.exp to Result
            Move j to its next term
        End If
    End While

    While i != A.first do
        Add the remaining term in A to Result
        Move i to its next term
    End While

    While j != B.first do
        Add the remaining term in B to Result
        Move j to its next term
    End While

    Return Result
End Function
```

2. 減法的敘述： 減法操作對應於兩個多項式相減的過程。類似於加法，但這次是將兩個多項式中相同指數的項的係數相減，並生成一個新的多項式作為結果。如果一個多項式中的項比另一個多項式長，剩下的項將直接附加到結果中，但係數為負。

```
Function Subtract(Polynomial A, Polynomial B) -> Polynomial Result
    Initialize Result as an empty polynomial
    i = A.first.next
    j = B.first.next

    While i != A.first and j != B.first do
        If i.exp == j.exp then
            Add a new term with coefficient (i.coef - j.coef) and exponent i.exp to Result
            Move i and j to their next terms
        Else if i.exp > j.exp then
            Add a new term with coefficient i.coef and exponent i.exp to Result
            Move i to its next term
        Else
            Add a new term with coefficient (-j.coef) and exponent j.exp to Result
            Move j to its next term
        End If
    End While

    While i != A.first do
        Add the remaining term in A to Result
        Move i to its next term
    End While

    While j != B.first do
        Add the remaining term in B to Result with negative coefficient
        Move j to its next term
    End While

    Return Result
End Function
```

3. 乘法的敘述：乘法操作將兩個多項式相乘。這涉及將第一個多項式中的每一項依次乘以第二個多項式中的每一項，並將這些結果累加到一個新的多項式中。這意味著結果中的每一項的指數是兩個相乘項的指數之和，係數是它們的乘積。

```
Function Multiply(Polynomial A, Polynomial B) -> Polynomial Result
    Initialize Result as an empty polynomial

    For each term i in A do
        Initialize Temp as an empty polynomial
        For each term j in B do
            Multiply i.coef by j.coef and add their exponents i.exp + j.exp
            Add the result as a new term to Temp
        End For
        Result = Result + Temp
    End For

    Return Result
End Function
```

4. Eval 的敘述：Evaluate 函數用於計算多項式在特定 x 值處的值。它遍歷多項式的所有項，並計算每一項的值，然後將這些值累加以得到最終的結果。

```
Function Evaluate(Polynomial P, Float x) -> Float result
    Initialize result to 0

    For each term in P do
        Calculate term_value = term.coef * x^term.exp
        Add term_value to result
    End For

    Return result
End Function
```

實作參見檔案 poly2.cpp：

```cpp
#include <iostream>
#include <cmath>
using namespace std;

struct Term {
    float coef;
    int exp;
    Term* next;
};

class Polynomial {
    friend ostream& operator<<(ostream& os, const Polynomial& p);
    friend istream& operator>>(istream& is, Polynomial& p);

private:
    Term* first;

public:
    Polynomial() {
        first = new Term;
        first->next = first;
    }

    Polynomial(const Polynomial& B) {
        first = new Term;
        first->next = first;
        *this = B;
    }

    ~Polynomial() {
        Term* p = first->next;
        while (p != first) {
            Term* q = p;
```

CHAPTER 2 演算法設計與實作

1. 參考助教的範本，並藉由使用 ChatGPT 得出答案

```cpp
#include <iostream>
#include <cmath>
using namespace std;

struct Term {
    float coef;
    int exp;
    Term* next;
};

class Polynomial {
    friend ostream& operator<<(ostream& os, const Polynomial& p);
    friend istream& operator>>(istream& is, Polynomial& p);

private:
    Term* first;

public:
    Polynomial() {
        first = new Term;
        first->next = first;
    }

    Polynomial(const Polynomial& B) {
        first = new Term;
        first->next = first;
        *this = B;
    }

    ~Polynomial() {
        Term* p = first->next;
        while (p != first) {
            Term* q = p;
```

```cpp
        Term* p = first->next;
        // 當p不等於第一個index時，p = q，p往下指，把q刪掉，直到p指到first
        while (p != first) {
            Term* q = p;
            p = p->next;
            delete q;
        }
        delete first;
    }

    Polynomial& operator=(const Polynomial& B) {
        if (this == &B) return *this;
        Term* p = first->next;
        while (p != first) {
            Term* q = p;
            p = p->next;
            delete q;
        }
        first->next = first;
        p = B.first->next;
        while (p != B.first) {
            newTerm(p->coef, p->exp);
            p = p->next;
        }
        return *this;
    }

    Polynomial operator+(const Polynomial& B) const {
        Polynomial result;
        Term* i = first->next;
        Term* j = B.first->next;
        while (i != first && j != B.first) {
            if (i->exp == j->exp) {
```

```cpp
        while (i != first && j != B.first) {
            if (i->exp == j->exp) {
                result.newTerm(i->coef + j->coef, i->exp);
                i = i->next;
                j = j->next;
            } else if (i->exp > j->exp) {
                result.newTerm(i->coef, i->exp);
                i = i->next;
            } else {
                result.newTerm(j->coef, j->exp);
                j = j->next;
            }
        }
        while (i != first) {
            result.newTerm(i->coef, i->exp);
            i = i->next;
        }
        while (j != B.first) {
            result.newTerm(j->coef, j->exp);
            j = j->next;
        }
        return result;
    }

    Polynomial operator-(const Polynomial& B) const {
        Polynomial result;
        Term* i = first->next;
        Term* j = B.first->next;
        while (i != first && j != B.first) {
            if (i->exp == j->exp) {
                result.newTerm(i->coef - j->coef, i->exp);
                i = i->next;
                j = j->next;
```

```
                i = i->next;
                j = j->next;
            } else if (i->exp > j->exp) {
                result.newTerm(i->coef, i->exp);
                i = i->next;
            } else {
                result.newTerm(-j->coef, j->exp);
                j = j->next;
            }
        }
        while (i != first) {
            result.newTerm(i->coef, i->exp);
            i = i->next;
        }
        while (j != B.first) {
            result.newTerm(-j->coef, j->exp);
            j = j->next;
        }
        return result;
    }

    Polynomial operator*(const Polynomial& B) const {
        Polynomial result;
        for (Term* i = first->next; i != first; i = i->next) {
            Polynomial temp;
            for (Term* j = B.first->next; j != B.first; j = j->next) {
                temp.newTerm(i->coef * j->coef, i->exp + j->exp);
            }
            result = result + temp;
        }
        return result;
    }

```

```
    float Evaluate(const float x) const {
        float result = 0.0;
        for (Term* p = first->next; p != first; p = p->next) {
            result += p->coef * pow(x, p->exp);
        }
        return result;
    }

    void newTerm(float coef, int exp) {
        if (coef == 0) return;
        Term* p = first;
        Term* q = p->next;
        while (q != first && q->exp > exp) {
            p = q;
            q = q->next;
        }
        if (q != first && q->exp == exp) {
            q->coef += coef;
            if (q->coef == 0) {
                p->next = q->next;
                delete q;
            }
        } else {
            Term* newTerm = new Term;
            newTerm->coef = coef;
            newTerm->exp = exp;
            newTerm->next = q;
            p->next = newTerm;
        }
    }
};

ostream& operator<<(ostream& os, const Polynomial& p) {
```

```cpp

ostream& operator<<(ostream& os, const Polynomial& p) {
    Term* current = p.first->next;
    while (current != p.first) {
        if (current != p.first->next && current->coef > 0)
            os << "+";
        os << current->coef << "x^" << current->exp;
        current = current->next;
    }
    return os;
}

istream& operator>>(istream& is, Polynomial& p) {
    int n;
    is >> n;
    for (int i = 0; i < n; ++i) {
        float c;
        int e;
        is >> c >> e;
        p.newTerm(c, e);
    }
    return is;
}
```

```cpp
int main() {
    Polynomial p1, p2, p3;
    cout << "Enter the first polynomial (format: n c1 e1 c2 e2 ...): ";
    cin >> p1;
    cout << "Enter the second polynomial (format: n c1 e1 c2 e2 ...): ";
    cin >> p2;

    p3 = p1 + p2;
    cout << "Sum: " << p3 << endl;

    p3 = p1 - p2;
    cout << "Difference: " << p3 << endl;

    p3 = p1 * p2;
    cout << "Product: " << p3 << endl;

    float x;
    cout << "Enter a value for x: ";
    cin >> x;
    cout << "Evaluation of first polynomial at x = " << x << ": " << p1.Evaluate(x) << endl;

    return 0;
}
```

CHAPTER 3 效能分析

（一）

時間複雜度

1.Add 的時間複雜度為 bigO(n+m)，假設第一個多項式有 n 項，第二個多項式有 m 項。在最壞的情況下，需要遍歷兩個多項式的所有項來進行加法操作，因此時間複雜度為 bigO(n+m)。

2. Mult 的時間複雜度為 bigO(n*m)，第一個多項式有 n 項，第二個多項式有 m 項，則總共需要進行 n*m 次乘法操作。

3. Eval 的時間複雜度為 bigO(n)，n 是 terms 的長度。

4. Minus 的時間複雜度為 bigO(n+m)，減法的時間複雜度與加法類似，需要比較和操作兩個多項式的所有項，因此也是 bigO(n+m)

(二)空間複雜度

1.Add 的空間複雜度為 bigO(n+m)。

2. Mult 的空間複雜度為 bigO(n*m)。

3.Eval 的空間複雜度為 bigO(1)。

4.Minus 的空間複雜度為 bigO(n+m))。

CHAPTER 4 測試與驗證

驗證

1. 驗證加法(Add)：$(2x^4 + 4x^2) + (3x^5 + 5x^3)$ 經排序過後 $= 3x^5 + 2x^4 + 5x^3 + 4x^2$。
2. 驗證乘法(Mult)：$(2x^4 + 4x^2) * (3x^5 + 5x^3) = (6x^9 + 10x^7 + 12x^7 + 20x^5)$ 經排序過後 $= (6x^9 + 22x^7 + 20x^5)$。
3. 驗證數值(Eval)：數值代 1 ->

   $p1 = (2x^4 + 4x^2) = 6$
4. 驗證減法(Minus)：$(2x^4 + 4x^2) + (3x^5 + 5x^3)$ 經排序過後 $= -3x^5 + 2x^4 - 5x^3 + 4x^2$。

```
Enter the second polynomial (format: n c1 e1 c2 e2 ...): Sum:
Difference:
Product:
Enter a value for x: Evaluation of first polynomial at x = 0: 0
PS C:\Users\user\Desktop\school\c++> .\2.exe
Enter the first polynomial (format: n c1 e1 c2 e2 ...): 2 2 4 4 2
Enter the second polynomial (format: n c1 e1 c2 e2 ...): 2 3 5 5 3
Sum: 3x^5+2x^4+5x^3+4x^2
Difference: -3x^5+2x^4-5x^3+4x^2
Product: 6x^9+22x^7+20x^5
Enter a value for x: 1
Evaluation of first polynomial at x = 1: 6
PS C:\Users\user\Desktop\school\c++>
```