資料結構報告

張凱茗

2024/12/20

目錄

解題說明

(a): 多載>>和<<(圖 1、圖 2)

(b~i): 根據題目要求實作 Polynomial 類別成員，透過 linked lists 存放指數及係數成員(圖 3)

```
//多載輸入
friend istream& operator>>(istream& is, Polynomial& x)
{
    int n;
    is >> n; //讀取項數
    for (int i = 0; i < n; ++i)
    {
        int coef, exp;
        is >> coef >> exp;
        x.addTerm(coef, exp);
    }
    return is;
}
```

圖 1

```
//多載輸出
friend ostream& operator<<(ostream& os, const Polynomial& x)
{
    Node* curr = x.head->link;
    if (curr == x.head)
    {
        os << "0";
    }
    else
    {
        while (curr != x.head)
        {
            if (curr->coef > 0 && curr != x.head->link) os << "+";
            os << curr->coef;
            if (curr->exp != 0) os << "x^" << curr->exp;
            curr = curr->link;
        }
    }
    return os;
}
```

圖 2

```
    //新節點
    Node* createNode(int coef, int exp){ ... }

    //刪除所有節點
    void clear(){ ... }

public:
    //建構函式
    Polynomial(){ ... }

    //複製
    Polynomial(const Polynomial& other){ ... }

    //解構
    ~Polynomial(){ ... }

    //多載=
    Polynomial& operator=(const Polynomial& other){ ... }

    //新增多項式
    void addTerm(int coef, int exp){ ... }

    //多載輸入
    friend istream& operator>>(istream& is, Polynomial& x){ ... }

    //多載輸出
    friend ostream& operator<<(ostream& os, const Polynomial& x){ ... }

    //多載加法
    Polynomial operator+(const Polynomial& b) const{ ... }

    //多載減法
    Polynomial operator-(const Polynomial& b) const{ ... }

    //多載乘法
    Polynomial operator*(const Polynomial& b) const{ ... }

    //求值
    double Evaluate(double x) const{ ... }
```

演算法設計與實作

如圖 4~圖 10

```
class Node
{
public:
    int coef; //係數
    int exp;  //指數
    Node* link; //下一個節點
};

class Polynomial
{
private:
    Node* head; //頭

    //新節點
    Node* createNode(int coef, int exp)
    {
        Node* newNode = new Node;
        newNode->coef = coef;
        newNode->exp = exp;
        newNode->link = nullptr;
        return newNode;
    }
```

圖 4

```
//建構函式
Polynomial()
{
    head = new Node;
    head->link = head;
}

//複製
Polynomial(const Polynomial& other)
{
    head = new Node;
    head->link = head;
    Node* curr = other.head->link;
    while (curr != other.head)
    {
        addTerm(curr->coef, curr->exp);
        curr = curr->link;
    }
}
```

圖 5

```
//多載=
Polynomial& operator=(const Polynomial& other)
{
    if (this == &other) return *this; //相同
    clear();
    head = new Node;
    head->link = head;
    Node* curr = other.head->link;
    while (curr != other.head)
    {
        addTerm(curr->coef, curr->exp);
        curr = curr->link;
    }
    return *this;
}
```

圖 6

```cpp
//新增多項式
void addTerm(int coef, int exp)
{
    if (coef == 0) return;
    Node* prev = head;
    Node* curr = head->link;
    while (curr != head && curr->exp > exp)
    {
        prev = curr;
        curr = curr->link;
    }
    if (curr != head && curr->exp == exp)
    {
        curr->coef += coef;
        if (curr->coef == 0)
        {
            prev->link = curr->link;
            delete curr;
        }
    }
    else
    {
        Node* newNode = createNode(coef, exp);
        prev->link = newNode;
        newNode->link = curr;
    }
}
```

圖 7

```cpp
//多載加法
Polynomial operator+(const Polynomial& b) const
{
    Polynomial result;
    Node* currA = head->link;
    Node* currB = b.head->link;
    while (currA != head || currB != b.head)
    {
        if (currA != head && (currB == b.head || currA->exp > currB->exp))
        {
            result.addTerm(currA->coef, currA->exp);
            currA = currA->link;
        }
        else if (currB != b.head && (currA == head || currB->exp > currA->exp))
        {
            result.addTerm(currB->coef, currB->exp);
            currB = currB->link;
        }
        else
        {
            result.addTerm(currA->coef + currB->coef, currA->exp);
            currA = currA->link;
            currB = currB->link;
        }
    }
    return result;
}
```

圖 8

```cpp
//多載減法
Polynomial operator-(const Polynomial& b) const
{
    Polynomial result;
    Node* currB = b.head->link;
    while (currB != b.head)
    {
        result.addTerm(-currB->coef, currB->exp);
        currB = currB->link;
    }
    return *this + result;
}

//多載乘法
Polynomial operator*(const Polynomial& b) const
{
    Polynomial result;
    Node* currA = head->link;
    while (currA != head)
    {
        Node* currB = b.head->link;
        while (currB != b.head)
        {
            result.addTerm(currA->coef * currB->coef, currA->exp + currB->exp);
            currB = currB->link;
        }
        currA = currA->link;
    }
    return result;
}
```

圖 9

```cpp
//求值
double Evaluate(double x) const
{
    double result = 0;
    Node* curr = head->link;
    while (curr != head)
    {
        result += curr->coef * pow(x, curr->exp);
        curr = curr->link;
    }
    return result;
}
```

圖 10

效能分析

Problem 1:

時間複雜度：

輸入/輸出：**O(n)**

加法/減法：**O($n1 + n2$)**

乘法：**O(n1·n2)**

求值：**O(n)**

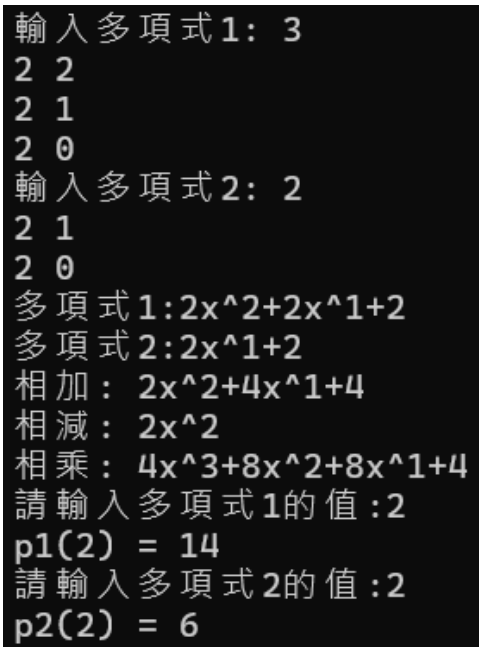
空間複雜度

單個多項式：O(n)

加法/減法結果：O(n1+n2)

乘法結果：O(n1·n2)

測試與過程



驗證

$(2x^2 + 2x + 2) + (2x + 2) = (2x^2 + 4x + 4)$

$(2x^2 + 2x + 2) - (2x + 2) = (2x^2)$

$(2x^2 + 2x + 2)(2x + 2) = (4x^3 + 8x^2 + 8x + 4)$

P1=$(2 * 2^2 + 2 * 2 + 2) = 14$

P2=$(2 * 2 + 2) = 6$

申論及開發報告

這次在寫作業時，才發現我在多載 class 的+-*=上完全忘記如何操作了，所以我又花時間去全部複習一次，並且發現 HW2 和 HW3 的時間複雜度除了輸入輸出外幾乎一致，而空間複雜度是 HW3 能夠跟著使用來緩慢增加而不是成倍增長，因此 HW3 實際運行中的空間佔用會比 HW2 少。