資料結構

HW2 Problem1 & Problem2

日期:2024/11/27

CHAPER1:解題說明

想法 (How to do?)

本題的目的是實現多項式的加法、乘法和評估，並進行正確的運算。多項式通常表達為各個項的係數和指數的組合，像是：

- 加法：將相同指數的項的係數相加。
- 乘法：將每個項的係數相乘，指數相加。
- 評估：將多項式代入某個數值，計算結果

存取和比對多項式的步驟：

1. **比較運算符**：
    - 我們需要定義一個運算符重載，以便在進行加法和乘法運算時，可以比較兩個項的指數。
    - 比較的規則是：**指數較大的項優先**，即按指數從高到低排序。
2. **存取多項式項**：
    - 使用 Term 類別來表示單一的多項式項，並存取每個項的係數和指數。
    - 透過一個數組或動態數組來存儲所有項。
3. **加法和乘法的比對邏輯**：
    - 在加法中，我們需要遍歷兩個多項式，對每個項的指數進行比較，並合併係數相同的項。
    - 在乘法中，則是對每個項與每個其他項進行比較，並根據規則相加指數，並存儲結果

範例 ：簡單的多項式加法和乘法

假設有以下兩個多項式：

1. $P(x) = 2x^2 + 3$
2. $Q(x) = x^2 + 4x + 1$

我們希望進行加法、乘法運算並評估這些多項式。

加法運算：

將兩個多項式相加時，我們會將相同指數的項的係數相加。

多項式加法：

$$P(x) + Q(x) = (2x^2 + 3) + (x^2 + 4x + 1)$$

進行加法時：

- x^2 項：$2x^2 + x^2 = 3x^2$
- $4x$ 項：$4x$
- 常數項：$3 + 1 = 4$

結果是：

$$P(x) + Q(x) = 3x^2 + 4x + 4$$

乘法運算：

將兩個多項式相乘時，我們會將每一個項與另一多項式的每一項進行相乘，並合併結果。

多項式乘法：

$$P(x) \times Q(x) = (2x^2 + 3) \times (x^2 + 4x + 1)$$

進行乘法時，我們將每個項與另一多項式的每個項相乘：

- $2x^2 \times x^2 = 2x^4$
- $2x^2 \times 4x = 8x^3$
- $2x^2 \times 1 = 2x^2$
- $3 \times x^2 = 3x^2$
- $3 \times 4x = 12x$
- $3 \times 1 = 3$

將結果合併：

$$P(x) \times Q(x) = 2x^4 + 8x^3 + 5x^2 + 12x + 3$$

CHAPER2: Algorithm Design & Programming

```
4   // Term 類別定義
5   class Term {
6       friend class Polynomial; // 讓 Polynomial 類別訪問私有成員
7   private:
8       float coef; // 項的係數
9       int exp;    // 項的指數
10  public:
11      Term(float c = 0, int e = 0) : coef(c), exp(e) {}
12      float getCoef() const { return coef; }
13      int getExp() const { return exp; }
14      void setCoef(float c) { coef = c; }
15      void setExp(int e) { exp = e; }
16  };
17
```

```
// Polynomial 類別定義
class Polynomial {
private:
    Term* termArray; // 存放多項式項目
    int capacity;    // 最大容量
    int terms;       // 當前項目數量

public:
    Polynomial() {
        terms = 0;
        capacity = 10;
        termArray = new Term[capacity];
    }

    ~Polynomial() {
        delete[] termArray;
    }
```

```
  // 自定義賦值運算符
  Polynomial& operator=(const Polynomial& poly) {
      if (this == &poly) { // 防止自我賦值
          return *this;
      }
      delete[] termArray;
      capacity = poly.capacity;
      terms = poly.terms;
      termArray = new Term[capacity];
      for (int i = 0; i < terms; i++) {
          termArray[i] = poly.termArray[i];
      }
      return *this;
  }
```

```
51      // 多項式相加
52      Polynomial Add(const Polynomial& poly) {
53          Polynomial result;
54          int i = 0, j = 0;
55
56          while (i < this->terms && j < poly.terms) {
57              if (this->termArray[i].getExp() == poly.termArray[j].getExp()) {
58                  // 係數相加
59                  float newCoef = this->termArray[i].getCoef() + poly.termArray[j].getCoef();
60                  if (newCoef != 0) { // 只加不為零的項
61                      result.addTerm(newCoef, this->termArray[i].getExp());
62                  }
63                  i++;
64                  j++;
65              } else if (this->termArray[i].getExp() > poly.termArray[j].getExp()) {
66                  result.addTerm(this->termArray[i].getCoef(), this->termArray[i].getExp());
67                  i++;
68              } else {
69                  result.addTerm(poly.termArray[j].getCoef(), poly.termArray[j].getExp());
70                  j++;
71              }
72          }
```

```
74          // 處理剩餘的項
75          while (i < this->terms) {
76              result.addTerm(this->termArray[i].getCoef(), this->termArray[i].getExp());
77              i++;
78          }
79          while (j < poly.terms) {
80              result.addTerm(poly.termArray[j].getCoef(), poly.termArray[j].getExp());
81              j++;
82          }
83
84          return result;
85      }
```

```
87      // 多項式相乘
88      Polynomial Mult(const Polynomial& poly) {
89          Polynomial result;
90          for (int i = 0; i < this->terms; i++) {
91              for (int j = 0; j < poly.terms; j++) {
92                  // 係數相乘，指數相加
93                  float newCoef = this->termArray[i].getCoef() * poly.termArray[j].getCoef();
94                  int newExp = this->termArray[i].getExp() + poly.termArray[j].getExp();
95                  if (newCoef != 0) { // 只加不為零的項
96                      result.addOrCombine(newCoef, newExp);
97                  }
98              }
99          }
100         return result;
101     }
```

```
103     // 計算多項式在某點的值
104     float Eval(float x) const {
105         float result = 0;
106         for (int i = 0; i < terms; i++) {
107             float termValue = 1;
108             for (int j = 0; j < termArray[i].getExp(); j++) {
109                 termValue *= x; // 手動計算次方
110             }
111             result += termArray[i].getCoef() * termValue;
112         }
113         return result;
114     }
```

```
    // 輸入多項式
    void input() {
        int n;
        cout << "Enter the number of terms: ";
        cin >> n;
        for (int i = 0; i < n; i++) {
            float coef;
            int exp;
            cout << "Enter coefficient and exponent for term " << i + 1 << ": ";
            cin >> coef >> exp;
            addTerm(coef, exp);
        }
    }
```

```
    // 輸出多項式
    void output() const {
        bool firstTerm = true;
        for (int i = 0; i < terms; i++) {
            if (termArray[i].getCoef() == 0) continue; // 忽略係數為零的項

            if (!firstTerm && termArray[i].getCoef() > 0) {
                cout << " + ";
            }

            if (termArray[i].getCoef() < 0) {
                cout << termArray[i].getCoef();
            } else if (firstTerm) {
                cout << termArray[i].getCoef();
            } else {
                cout << "+" << termArray[i].getCoef();
            }

            if (termArray[i].getExp() > 0) {
                cout << "x^" << termArray[i].getExp();
            }
            firstTerm = false;
        }
        if (firstTerm) {
            cout << "0"; // 如果多項式全為 0
        }
        cout << endl;
    }
```

```
private:
    // 添加或合併項
    void addOrCombine(float coef, int exp) {
        for (int i = 0; i < terms; i++) {
            if (termArray[i].getExp() == exp) {
                termArray[i].setCoef(termArray[i].getCoef() + coef);
                if (termArray[i].getCoef() == 0) { // 移除係數為 0 的項
                    removeTerm(i);
                }
                return;
            }
        }
        addTerm(coef, exp);
    }

    // 添加新項
    void addTerm(float coef, int exp) {
        if (terms == capacity) {
            capacity *= 2;
            Term* newTermArray = new Term[capacity];
            for (int i = 0; i < terms; i++) {
                newTermArray[i] = termArray[i];
            }
            delete[] termArray;
            termArray = newTermArray;
        }
        termArray[terms++] = Term(coef, exp);
    }
```

```
188         // 移除項 (當係數為 0 時)
189         void removeTerm(int index) {
190             for (int i = index; i < terms - 1; i++) {
191                 termArray[i] = termArray[i + 1];
192             }
193             terms--;
194         }
195     };
```

```
197     int main() {
198         Polynomial poly1, poly2, result;
199 
200         // 輸入多項式
201         cout << "Enter the first polynomial:" << endl;
202         poly1.input();
203 
204         cout << "Enter the second polynomial:" << endl;
205         poly2.input();
206 
207         // 輸出多項式
208         cout << "First Polynomial: ";
209         poly1.output();
210         cout << "Second Polynomial: ";
211         poly2.output();
212 
213         // 計算和
214         result = poly1.Add(poly2);
215         cout << "Sum: ";
216         result.output();
217 
218         // 計算積
219         result = poly1.Mult(poly2);
220         cout << "Product: ";
221         result.output();
```

```
223         // 計算指定值的結果
224         float value;
225         cout << "Enter a value to evaluate the first polynomial: ";
226         cin >> value;
227         cout << "P1(" << value << ") = " << poly1.Eval(value) << endl;
228 
229         return 0;
230     }
```

CHAPER3:效能分析

Time Complexity:

- **加法 (Add)**: 每個多項式的項數是 n 和 m，因此加法的時間複雜度是 O(n+m)O(n + m)O(n+m)，其中 n 和 m 分別是兩個多項式的項數。
- **乘法 (Mult)**: 每個多項式的項數是 n 和 m，因此乘法的時間複雜度是 O(n×m)O(n \times m)O(n×m)，其中 n 和 m 是兩個多項式的項數。
- **評估 (Eval)**: 評估多項式的時間複雜度是 O(n)O(n)O(n)，其中 n 是多項式的項數，因為每項需要計算其次方。

Space Complexity:

- **空間複雜度**：空間複雜度主要取決於多項式的項數，最壞情況下是 O(n+m)O(n + m)O(n+m) 空間，其中 n 和 m 分別是兩個多項式的項數。
-

CHAPER4:測試與認證

```
PS D:\資料結構\homework2> .\problem1and2.exe
Enter the first polynomial:
Enter the number of terms: 2
Enter coefficient and exponent for term 1: 2 3
Enter coefficient and exponent for term 2: 5 0
Enter the second polynomial:
Enter the number of terms: 3
Enter coefficient and exponent for term 1: 1 3
Enter coefficient and exponent for term 2: 2 2
Enter coefficient and exponent for term 3: -4 0
First Polynomial: 2x^3 + 5
Second Polynomial: 1x^3 + 2x^2-4
Sum: 3x^3 + 2x^2 + 1
Product: 2x^6 + 4x^5-3x^3 + 10x^2-20
Enter a value to evaluate the first polynomial: 2
P1(2) = 21
PS D:\資料結構\homework2>
```

CHAPER5:效能量測

可根據項數量級測量效能。例如，在多項式的項數量為 1000 和 10000 時進行性能測量，並測量加法、乘法和評估的執行時間。

CHAPER6:心得討論

這次的作業讓我深入了解多項式的運算，雖然這個作業的設計主要是基於加法和乘法的基本運算，但在實際應用中，我學到了如何處理多項式中可能出現的各種邊界情況。例如，處理某些項的係數為零的情況，避免它們在最終結果中出現。

將每個功能模塊（加法、乘法、評估等）封裝為獨立的成員函數，這樣不僅讓程式結構清晰，還能夠讓其他人容易理解和使用。

在這次作業中，我使用了 Term 類別來表示多項式的單個項，這讓我能夠將每個項的係數和指數封裝成對象。這樣的設計讓處理和儲存多項式變得更加直觀，並且使後續的加法、乘法等運算變得簡單可控。