

資料結構報告

張哲維

AUG, 22, 2024

CONTENTS

1 解題說明

2 演算法設計與實作

3 效能分析

4 測試與過程

CHAPTER 1

解題說明

用Polynomial來表示多項式，進行加法、乘法 處理多項式的輸入 輸出使用運算符號來方便這些操作。

Ex:

兩個多項式 $p1(x) = 4x^2 + 3x + 1$ 和 $p2(x) = x^2 + 2$ ，我們希望實現以下功能：

1. **加法:** $p1 + p2 = (4+1)x^2 + 3x + (1+2) = 5x^2 + 3x + 3$
2. **減法:** $p1 - p2 = 3x^2 + 3x -$
3. **乘法:** $p1 * p2 = (4x^2 + 3x + 1)(x^2 + 2) = 8x + x^2 + 4 = 3x^4 + 2x^3 + 13x^2 + 8x + 4$

◀ CHAPTER 2

解題說明

```
1  #include <iostream>
2  #include <algorithm>
3  #include <cmath>
4  using namespace std;
5
6  class Polynomial; // 前向宣告
7
8  class Term {
9  |   friend class Polynomial;
10 | public:
11 |     float coef;
12 |     int exp;
13 | };
14
15 class Polynomial {
16 | private:
17 |     Term* termArray; // 非零項的數組
18 |     int capacity;     // termArray 的大小
19 |     int terms;        // 非零項的數量
20 |
21 | public:
22 |     Polynomial(int cap = 10) : capacity(cap), terms(0) {
23 |         termArray = new Term[capacity];
24 |     }
25
26 |     ~Polynomial() {
27 |         delete[] termArray;
28 |     }
29
30 |     friend istream& operator>>(istream& input, Polynomial& p);
31 |     friend ostream& operator<<(ostream& output, const Polynomial& p);
32
33 |     Polynomial operator+(const Polynomial& other) const;
34 |     Polynomial operator-(const Polynomial& other) const;
```

```
},

istream& operator>>(istream& input, Polynomial& p) {
    int numTerms;
    cout << "輸入項數: ";
    input >> numTerms;

    for (int i = 0; i < numTerms; ++i) {
        float coef;
        int exp;
        cout << "輸入係數和指數: ";
        input >> coef >> exp;
        p.NewTerm(coef, exp);
    }

    sort(p.termArray, p.termArray + p.terms, [](const Term& a, const Term& b) {
        return a.exp > b.exp;
    });

    return input;
}

ostream& operator<<(ostream& output, const Polynomial& p) {
    for (int i = 0; i < p.terms; ++i) {
        output << p.termArray[i].coef << "x^" << p.termArray[i].exp;
        if (i != p.terms - 1) output << " + ";
    }
    return output;
}

void Polynomial::NewTerm(float theCoeff, int theExp) {
    for (int i = 0; i < terms; ++i) {
        if (termArray[i].exp == theExp) {
            termArray[i].coef += theCoeff;
            return;
        }
    }
    termArray[terms] = Term(theCoeff, theExp);
    terms++;
}
```

```
76     }
77     if (terms == capacity) {
78         capacity *= 2;
79         Term* newArray = new Term[capacity];
80         copy(termArray, termArray + terms, newArray);
81         delete[] termArray;
82         termArray = newArray;
83     }
84     termArray[terms].coef = theCoeff;
85     termArray[terms++].exp = theExp;
86 }
87
88 Polynomial Polynomial::operator+(const Polynomial& other) const {
89     Polynomial c;
90     int aPos = 0, bPos = 0;
91     while (aPos < terms && bPos < other.terms) {
92         if (termArray[aPos].exp == other.termArray[bPos].exp) {
93             float t = termArray[aPos].coef + other.termArray[bPos].coef;
94             if (t) c.NewTerm(t, termArray[aPos].exp);
95             aPos++;
96             bPos++;
97         } else if (termArray[aPos].exp < other.termArray[bPos].exp) {
98             c.NewTerm(other.termArray[bPos].coef, other.termArray[bPos].exp);
99             bPos++;
100         } else {
101             c.NewTerm(termArray[aPos].coef, termArray[aPos].exp);
102             aPos++;
103         }
104     }
105     for (; aPos < terms; aPos++)
106         c.NewTerm(termArray[aPos].coef, termArray[aPos].exp);
107     for (; bPos < other.terms; bPos++)
108         c.NewTerm(other.termArray[bPos].coef, other.termArray[bPos].exp);
109 }
```

```
115     while (aPos < terms && bPos < other.terms) {
116         if (termArray[aPos].exp == other.termArray[bPos].exp) {
117             float t = termArray[aPos].coef - other.termArray[bPos].coef;
118             if (t) c.NewTerm(t, termArray[aPos].exp);
119             aPos++;
120             bPos++;
121         } else if (termArray[aPos].exp < other.termArray[bPos].exp) {
122             c.NewTerm(-other.termArray[bPos].coef, other.termArray[bPos].exp);
123             bPos++;
124         } else {
125             c.NewTerm(termArray[aPos].coef, termArray[aPos].exp);
126             aPos++;
127         }
128     }
129     for (; aPos < terms; aPos++)
130         c.NewTerm(termArray[aPos].coef, termArray[aPos].exp);
131     for (; bPos < other.terms; bPos++)
132         c.NewTerm(-other.termArray[bPos].coef, other.termArray[bPos].exp);
133     return c;
134 }
135
136 Polynomial Polynomial::operator*(const Polynomial& other) const {
137     Polynomial c;
138     for (int i = 0; i < terms; ++i) {
139         for (int j = 0; j < other.terms; ++j) {
140             float newCoef = termArray[i].coef * other.termArray[j].coef;
141             int newExp = termArray[i].exp + other.termArray[j].exp;
142             c.NewTerm(newCoef, newExp);
143         }
144     }
145     return c;
146 }
147 }
```

```
7
8 float Polynomial::evaluate(float x) const {
9     float result = 0.0;
10    for (int i = 0; i < terms; ++i) {
11        result += termArray[i].coef * pow(x, termArray[i].exp);
12    }
13    return result;
14 }
15
16 int main() {
17     Polynomial p1, p2;
18
19     cout << "輸入第一個多項式:\n";
20     cin >> p1;
21     cout << "輸入第二個多項式:\n";
22     cin >> p2;
23
24     cout << "多項式1: " << p1 << endl;
25     cout << "多項式2: " << p2 << endl;
26
27     Polynomial p3 = p1 + p2;
28     Polynomial p4 = p1 - p2;
29     Polynomial p5 = p1 * p2;
30
31     cout << "多項式加法結果: (" << p1 << ") + (" << p2 << ") = " << p3 << endl;
32     cout << "多項式減法結果: (" << p1 << ") - (" << p2 << ") = " << p4 << endl;
33     cout << "多項式乘法結果: (" << p1 << ") * (" << p2 << ") = " << p5 << endl;
34
35     return 0;
36 }
```


CHAPTER3

效能分析

時間複雜度

加法 減法: $O(N+M)$
乘法 $O(N*M)$

空間複雜度
 $O(n)$

心得

讓我們了解多項式的運算 讓我們更方便在生活上

