

Algorithm Design & Programming

NewTerm()

新增多項式的項目

Add()

將兩個多項式相加並 返回新多項式



Mult()

將兩個多項式相乘並 返回新多項式 Eval()

對多項式進行求值, 返回結果







效能分析&Source code + Comment

```
void Polynomial::NewTerm(const float newCoef, const int newExp) {
    if (this->terms == this->capacity) { // 空間不足時重新配置空間
          this->capacity *= 2;
           Term * temp = new Term[this->capacity];
           copy(this->termArray, this->termArray + terms, temp);
           delete[] this->termArray;
                                                    Polynomial拍JnewTerm()
           this->termArray = temp;
                                                          .新增項的指數: U
      this->termArray[this->terms].coef = newCoef;
                                                        曾前的多項式= 3x^5 + 6x^2 + 8
      this->termArray[this->terms++].exp = newExp;
                                                        曾後的多項式⊨ 3x^5 + 6x^2 + 8 + 1
```

Polynomial 的 NewTerm()是讓多項式類別可以增加項次,當空間不足時,我們將空間大小設為原始空間的兩倍大,將原始的資料搬移到新的資料串,並釋放存放原始資料串的記憶體,這個時間複雜度最好的狀況O(1),但最差也還是O(1);空間複雜度最好的情況是S(n+4),最壞是S(n+4+2^([lgn]+1))。

效能分析

```
float Polynomial::Eval(float f) { // 多項式求值
   float res = 0.0f;
    for (int i = 0; i < this->terms; i++) { // 走訪terms
        float temp = this->termArray[i].coef;
           for (int j = 0; j < this->termArray[i].exp; <math>j++) // f^n
               temp *= f;
                                                 Polynomial的Eval()
               res += temp;
                                                      入ſ借:
                                                    項式: 3x^5 + 6x^2 + 8 <u>+</u> 1
      return res; // 回傳最終計算值
```

Polynomial 的 Eval()是若給 f 求出多項式的值,我是先透過 res 去儲存結果值,就是去走 訪多項式,先用temp變數儲存係數,在根據項次的指數去乘上輸入的f值,以模擬計算,最 後在加到res,跑完就會知道值是多少,最後回傳結果

```
Polynomial Polynomial::Add(Polynomial poly) {
   Polynomial res;
   int* loc:
   loc = new int[poly.terms + this->terms];
   float* data:
   data = new float[poly.terms + this->terms];
   int use_len = 0;
   for (int i = 0; i < this -> terms; i++) {
       int t = -1;
       for (int j = 0; j < use_len; j++) {
           if (this->termArray[i].exp == loc[j]) {
              t = j;
               continue;
       if (t = -1) {
           loc[use_len] = this->termArray[i].exp;
           data[use_len++] = this->termArray[i].coef;
       else
            data[t] += this->termArray[i].coef;
     for (int i = 0; i < poly.terms; i++) { // 把參數poly放入陣列
        int t = -1; // -1表示未找到
           for (int j = 0; j < use_len; j++) { // 走訪重複判斷
              if (poly.termArray[i].exp == loc[j]) {
                     t = j;
                      continue;
        if (t == -1) {
           loc[use_len] = poly.termArray[i].exp;
           data[use_len++] = poly.termArray[i].coef;
         data[t] += poly.termArray[i].coef;
     for (int i = 0; i < use_len; i++) // 存入新的多項式類別
            res.NewTerm(data[i], loc[i]);
    return res;
```

Polynomial 的 Add()是自己加參數的多項式。A區域是建立一些初始設定,res是 經由多項式相加的結果,loc&data陣列是暫存放多項式加的內容,它們的大小空間最 差的情況是兩個多項式terms的大小相加。B區域是針對自己的多項式,走訪自己的所有項次,將每個內容放到暫存陣列中,t 指的是target 要存放指數的索引位置,若 t=-1,代表沒有重複的項次,所以直接放入暫存,並將use_len遞增,否則,有重複的項次,所以找到指數在loc的索引值,透過索引值對應data的值,加上這次的係數值,若不清楚可以看圖15,C區跟B區差不多,只是B區是this自己,C區是參數的poly,D區就是將暫存的data和loc存入 res 多項式。

這個Add()函式時間複雜度為O(n+m);空間複雜度為O(4(n+m)+12)。

```
Polynomial的Add()
p1 = 3x^5 + 6x^2 + 8 + 1
p2 = 9x^4 + 8x^2 + 3x
(3x^5 + 6x^2 + 8 + 1) + (9x^4 + 8x^2 + 3x) = 3x^5 + 14x^2 + 9 + 9x^4 + 3x
Add()轜時: 0.001s
```

```
Polynomial Polynomial::Mult(Polynomial poly) {
      Polynomial res;
      int* loc = new int[poly.terms * this->terms]; // 指數的陣列
      float* data = new float[poly.terms * this->terms]; // 係數的陣列
      int use len = 0; // 陣列使用長度
     for (int i = 0; i < this->terms; i++) { // 走訪自己的Term陣列
        for (int j = 0; j < poly.terms; j++) { // 走訪參數poly的Term陣列
           float t_coef = this->termArray[i].coef * poly.termArray[j].coef; // 計算係數
                int t_exp = this->termArray[i].exp + poly.termArray[j].exp; // 計算指數
                  int t = -1; // -1表示未找到
      for (int k = 0; k < use_len; k++) { // 走訪重複判斷
          if (t_{exp} = loc[k]) {
                  t = k;
                   continue;
          if (t = -1) {
           loc[use_len] = t_exp;
            data[use_len++] = t_coef;
          else
              data[t] += t_coef;
     for (int i = 0; i < use_len; i++) {
        res.NewTerm(data[i], loc[i]);
          return res;
```

Polynomial 的 Mult()是將自己的多項式乘以參數的多項式,for 迴圈前的都是初始化的設置,跟Add()前面的一樣,只是這邊的空間大小是相乘,主要過程是透過雙層for 迴圈模擬乘法運算,第一層是走訪自己的多項式,第二層是走訪參數的多項式,內部先計算係數和指數,再根據指數去找有沒有重複,若重複找重複的地方,接來就是根據重複與否,是重複就根據t加到對應位置,若沒重複就直接放入,過程是差不多的。

Mult()函式時間複雜度是O(n*m);空間複雜 度是O(2(n+m+nm)+13)。

```
Polynomial的Mult():
p1 = 3x^5 + 6x^2 + 8 + 1
p2 = 9x^4 + 8x^2 + 3x
(3x^5 + 6x^2 + 8 + 1) * (9x^4 + 8x^2 + 3x) = 27x^9 + 24x^7 + 63x^6 + 129x^4 + 18x^3 + 72x^2 + 27x
Mult()雲時: 0.002s
```



心得討論

提供更多操作, 例如多項式的除 法與微分 實現了多項式運 算的核心功能, 增強了對動態記 憶體與演算法的 理解

4 整體感想

程式碼

```
#include <iestreamb
                                                                                              #Polynomial Polynomial::Mult(Polynomial poly) {
#include <string>
                                                                                                     Polynomial ros:
#include <exception>
                                                                                                     int" loc = new int[poly.terms " this->terms]: // 指数的瞳列
#include <etime>
                                                                                                     float* data = mow float[poly.torms * this->torms]; // 像數的應列
using namespace std:
                                                                                                     int use 1cm = 0- // 陰功使用异度
                                                                                                     for (int i = 0; i < this->torms; i++) { // 走的自己的Torm随列
class Polynomial; // 前向宣告
                                                                                                        for (int j = 0; j < poly.torms; j++) { // 走數參數poly的Torm總列
                                                                                                           float t_coof = this->tormArray[i].coof * poly.tormArray[j].coof; // 計算係數
class Torm {
                                                                                                                int t_cxp = this->tcrmArray[i].cxp + poly.tcrmArray[j].cxp; // 計算指數
   friend Polynomial:
                                                                                                                   i=t t = -1; // -1表示未找到
   friend estream& operator<<(estream& es, const Polynomial& p);
                                                                                                     for (int k = 0; k < usc_lcn; k++) { // 走鯋重複判斷
                                                                                                          if (t_exp - loc[k]) {
   float cocf: // 像數
                                                                                                                   t = k:
   int cap: // 指数
                                                                                                                   comtinuo:
class Polynomial {
                                                                                                          if (t -- -1) {
   friend ostroam& operator<<(ostroam& os, const Polynomial& p);
   friend istream& operator>>(istream& input, Polynomial& p);
                                                                                                            loc[usc_lcn] = t_cmp;
                                                                                                            datafusc lcm++1 = t cocf:
   Torn" tornArray: // 多項式中非写項的陰列
   int capacity: // tormArray 的大小
   int toms: // 非写項的數量
                                                                                                              data[t] += t_cocf:
   Polynomial(): // Construct the polynomial p(x) = 0.
   Polynomial Add(Polynomial poly); // Ecturn the sum of the polynomial "this and poly.
                                                                                                     for (int i = 0; i < usc_lcm; i++) {
   Polynomial Mult(Polynomial poly): // Ecturn the product of the polynomials this and poly.
                                                                                                        rcs.NcwTcrm(data[i], loc[i]);
   float Eval(float f): // Evaluate the polynomial this at f and return the result.
   void NewTorm(const float newCoof, const int newExp): // 新增項目
                                                                                                          roturn rost
                                                                                               float Polynomial::Eval(float f) { // 多項式求值
Polynomial::Polynomial(): capacity(2), torms(0) {
                                                                                                   float ros = 0.0f;
   this->tormArray = now Torm[capacity];
                                                                                                    for (int i = 0; i < this->torms; i++) { // 走勤torms
                                                                                                       float tomp = this->tornArray[i].coof;
                                                                                                          for (int j = 0; j < this->tornArray[i].cap; j++) // f^n
Polynomial Polynomial::Add(Polynomial poly) {
                                                                                                              tomp *= f:
   Polynomial ros:
                                                                                                               ros 🖛 tomp:
   int* loc:
   loc = new int[poly.terms + this->terms]:
                                                                                                     roturn ros: // 回傳品銘計算值
   data = new float[poly.torms + this->terms];
                                                                                               void Polynomial::NowTorm(const float nowCoof, const int nowExp) {
   int usc_lcn = 0:
                                                                                                   if (this->torms - this->capacity) { // 空間不足時重新配置空間
   for (int i = 0; i < this->torms; i++) {
       int t = -1;
                                                                                                          this->capacity *= 2:
       for (int j = 0; j < usc_lon; j++) {
                                                                                                          Torm * tomp = new Torm[this->capacity]:
          if (this->tormArray[i].csp == loc[j]) {
                                                                                                          copy(this->tormArray, this->tormArray + torms, tomp);
                                                                                                          delete[] this->termArray:
              continuo:
                                                                                                          this->tornArray = tomp:
                                                                                                     this->tormArray[this->torms].coof = mowCoof;
                                                                                                      this->termArray[this->terms++1.cap = newExp;
          loc[usc_lon] = this->tormArray[i].cxp;
          data[usc_lcn++] = this->tcrmArray[i].cocf;
                                                                                              ostroamă oporator≪(ostroamă output, const Polynomială p) {
                                                                                                   for (int i = 0; i < p.tcrms; i++) {
                                                                                                      if (p.tc=mArray[i].cocf = 0) continuc:// 判斷係數是否為 0、說過該項
           data[t] += this->tormArray[i].coof;
                                                                                                        if (i - 0) { // 遊艇第一項
                                                                                                          output << p.tcrmArray[i].cocf;
     for (int i = 0; i < poly.tcrms; i++) { // 把參數poly放入陰列
        int t = -1: // -1表示未找到
          for (int j = 0; j < usc lcn; j++) { // 走動量複判斷
                                                                                                         if (p.tormArray[i].cocf > 0) {// 藍塑後爐項:根據像數的正負,決定是否如 "+"
             if (poly.tormArray[i].cxp == loc[j]) {
                                                                                                             output << " + " << p.tcrmArray[i].cocf;
                     continuo:
                                                                                                          clsc {
                                                                                                              output << " - " << -p.tcrmArray[i].cocf;
        if (t -- -1) {
          loc[usc_lon] = poly.tormArray[i].exp;
          data[usc_lcn++] = poly.tcrmArray[i].cocf;
                                                                                                       if (p.tcrmArray[i].cxp != 0) {// 庭超抱歉
                                                                                                          if (p.tormArray[i].cmp != 1) {
        data[t] += poly.tormArray[i].coof;
                                                                                                             output << "A" << p.tcrmArray[i].cxp;
     for (int i = 0; i < usc_lcm; i++) // 存人新的多項式類別
           rcs.NowTorm(data[i], loc[i]);
                                                                                                   sctuse output:
```

```
istrcam & operator>>(istrcam & input. Polynomial & p) {
   float t_cocf;
   int t_cmp:
   char tmp;
   bool plus - truc:
   while (1) {
      imput >> t_cocf;
      if (!plus) {
          t_coof *- -1;
         plus - truc:
      p.NowTorm(t_coof, 0); // 所有項都存储完惠
           break: // 跳出翅画
      input.ignorc(1): // 忽略指數前的符號
      imput >> t_cmp; // 這人指數
      p.NowTorm(t_coof, t_cxp);
      imput.gct(tmp);
      if (tmp == '\m') brcak : // 若下一個項目為負,代表下一項係款為負
      if (tmp == '-') plus = false: // 判斷是否多項式結束
ecture input:
int main() {
  clock_t start, finish;
   cout << "輸入格式 sx^n2+bx^n1+cx^n0+d (若為常數,可省略x^0)\n":
  Polynomial pl. p2:
  cout << "pl: ":
  cin >> pl;
   cout << "p2: ":
   cin \gg p2:
   cout << "pl = " << pl << cmdl;
   cout << "p2 = " << p2 << cmd1;
   cout << "-----\n";
   cout << "Polynomial@SncwTorm()\n";
  float t_coof = 0.0f;
   int t_cmp = 0:
   cout << "諧蝓人新增項的像數: ":
   cin >> t_cocf:
   cout « "謗轍人新增項的指數: ":
   cin >> t cap:
   cout « "新增前的多項式= " « pl « cadl;
  pl.NewTerm(t_coof, t_cmp);
  cout << "新增後的多項式= " << pl << cndl:
   cout << "----\n";
   cout << "Polynomial@SEval()\n":
   float f = 0.0f:
  cout << "論輸人=值: ":
  cin >> f;
   cout << "多項式: " << pl << cndl:
   cout << "结果= " << pl.Eval(f) << cmdl;
   cout << "----\n"-
   cout << "Polynomial@SAdd()\=":
   cout << "pl = " << pl << cadl;
   cout << "p2 = " << p2 << cmd1;
   start = clock():
   cout << "(" << pl << ") + (" << p2 << ") = " << p1.Add(p2) << cndl;
   finish = clock():
   cout << "Add() 網絡: " << (double)(finish - start) /
      CLOCKS FER SEC << "s" << cmdl;
    cout << "----\n";
   cout << "Polynomial@SWelt():\n":
   cout << "pl = " << pl << cmdl:
   cout << "p2 = " << p2 << cmd1:
   start = clock():
   cout << "(" << pl << ") * (" << p2 << ") * " << p1.Mult(p2) << cmdl;
   finish - clock();
   cout << "Mult()開時: " << (double)(finish - start) / CLOCKS_PER_SEC <<
      "s" << cmdl:
   cout << "-----\n":
   roturn 0:
```

執行結果

```
Microsoft Visual Studio 偵錯主控台
輸入格式 ax^n2+bx^n1+cx^nO+d (若為常數,可省略x^O)
p1: 3x^5+6x^2+8
p2: 9x^4+8x^2+3x^1
p1 = 3x^5 + 6x^2 + 8
p2 = 9x^4 + 8x^2 + 3x
Polynomial的newTerm()
  前入新增項的係數: 1
輸入新增項的係數: 0
輸入新增項的指數: 0
增前的多項式= 3x^5 + 6x^2 + 8
增後的多項式= 3x^5 + 6x^2 + 8 + 1
Polynomial的Eval()
 請輸入f值: 1
多項式: 3x^5 + 6x^2 + 8 + 1
Polynomial的Add()
p1 = 3x^5 + 6x^2 + 8 + 1
p2 = 9x^4 + 8x^2 + 3x
(3x^5 + 6x^2 + 8 + 1) + (9x^4 + 8x^2 + 3x) = 3x^5 + 14x^2 + 9 + 9x^4 + 3x
Add()需時: 0.001s
Polynomial的Mult():
p1 = 3x^5 + 6x^2 + 8 + 1
p2 = 9x^4 + 8x^2 + 3x
(3x^5 + 6x^2 + 8 + 1) * (9x^4 + 8x^2 + 3x) = 27x^9 + 24x^7 + 63x^6 + 129x^4 + 18x^3 + 72x^2 + 27x
Mult()需時: 0.002s
```