INTRODUCTION TO DATA STRUCTURES

Polynomial Addition & Transpose Sparse Matrix in C

HW1_D84099084_賴澍雨

Shih Yu Lai

1.Polynomial Addition

實作過程(Method 1)

資料結構

1.**多項式結構體(Polynomial Struct)**:這個結構體包含多項式的最高次數(degree)和一個浮點數陣列，該陣列代表每個非零項的係數。多項式的次數是最高非零係數項的指數。coef陣列的大小設為可以容納最大數量的項，由MAX_degree定義，即允許的最高次數加上常數項的一個空間。

程式邏輯與演算法

1.**多項式加法函數(poly_add Function)**:這個函數接受兩個多項式A和B作為輸入，返回它們的和。它初始化結果多項式，所有係數設為零。然後它確定結果多項式的次數為A或B中較高的次數。之後，它遍歷每個係數，將A和B對應的係數相加。

2.**列印多項式函數(print_poly Function)**:這個函數負責將多項式列印到標準輸出。它從最高次項遍歷到常數項(0次)。只列印非零係數以及對應的次數。first標誌用來控制格式，確保在列印第一個項之前不會有額外的字符。

輸入
(測試資料)

測試

```c
#include <stdio.h>
#define MAX_degree 101 // Maximum degree of the polynomial

typedef struct {
    int degree; // The degree of the polynomial
    float coef[MAX_degree]; // The coefficients of the polynomial
} polynomial;

polynomial poly_add(polynomial A, polynomial B) {
    polynomial result = {0}; // Initialize all coefficients to 0
    result.degree = A.degree > B.degree ? A.degree : B.degree; // Set the degree of the result to the maximum degree of the two polynomials

    // Add the coefficients of the two polynomials and store the result in the result polynomial by iterating through the coefficients
    for (int i = 0; i <= result.degree; i++) {
        result.coef[i] = A.coef[i] + B.coef[i]; // Add the coefficients
    }

    return result;
}

void print_poly(polynomial P) { // Function to print the polynomial
    int first = 1; // Flag to check if the first term has been printed
    // Iterate through the coefficients and print the non-zero coefficients
    for (int i = P.degree; i >= 0; i--) {
        if (P.coef[i] != 0) { // If the coefficient is non-zero
            if (!first) { // If it's not the first term, print a '+' sign
                printf("\n");
            }
            printf("%g %d", P.coef[i], i); // Print the coefficient and the degree
            first = 0;
        }
    }
}

int main() {
    // Initialize the polynomials A and B
    polynomial A = {9, {0, 0, -3, 0, 0, 12, 0, 0, 4, 5}};
    polynomial B = {9, {15, 0, 3, 0, 12, 0, 0, 0, 2, 3}};

    // Add the polynomials A and B
    polynomial result = poly_add(A, B);

    // Print the result
    printf("\n----------------------\n");
    printf("多項式A和B相加的結果是：\n");
    print_poly(result);
    printf("\n----------------------\n");

    // Test case 1: Polynomials with the same degree
    polynomial C = {3, {4, 0, 2, 1}};
    polynomial D = {3, {-1, 0, 2, 3}};
    polynomial result1 = poly_add(C, D);
    printf("測試1結果（相同次數的多項式相加）：\n");
    print_poly(result1);
    printf("\n----------------------\n");

    // Test case 2: Polynomials with different degrees
    polynomial E = {4, {0, 3, 1, 0, 2}};
    polynomial F = {2, {5, 0, -1}};
    polynomial result2 = poly_add(E, F);
    printf("測試2結果（不同次數的多項式相加）：\n");
    print_poly(result2);
    printf("\n----------------------\n");

    // Test case 3: Polynomials with zero coefficients
    polynomial G = {3, {0, 0, 0, 5}};
    polynomial H = {3, {0, 0, 0, -5}};
    polynomial result3 = poly_add(G, H);
    printf("測試3結果（係數為零的多項式相加）：\n");
    print_poly(result3);
    printf("\n----------------------\n");

    // Test case 4: Polynomials with boundary coefficients
    polynomial I = {0, {7}};
    polynomial J = {100, {3}};
    polynomial result4 = poly_add(I, J);
    printf("測試4結果（邊界次數的多項式相加）：\n");
    print_poly(result4);
    printf("\n----------------------\n");

    return 0;
}
```

輸出

```
----------------------
多項式A和B相加的結果是：
8 9
6 8
12 5
12 4
15 0
----------------------
測試1結果（相同次數的多項式相加）：
4 3
4 2
3 0
----------------------
測試2結果（不同次數的多項式相加）：
2 4
3 1
5 0
----------------------
測試3結果（係數為零的多項式相加）：

----------------------
測試4結果（邊界次數的多項式相加）：
10 0
----------------------
```

3.測試與測試資料設計

相同次數:確保加法正確處理符號。

不同次數:應該測試次數不同的多項式，以確保程序可以處理一個多項式的次數高於另一個的情況。

零係數:檢查程序是否正確地跳過打印零係數的項。

邊界條件:應該測試次數在程序可處理範圍極限(0和MAX_degree - 1)的多項式，以確保沒有越界錯誤。

2. Transpose Sparse Matrix

實作過程

使用的資料結構

1.**term 結構體**: 代表稀疏矩陣中的一個非零元素，包含三個整數型態的成員：行標(row)、列標(col)和該位置的值(value)。

2.**a 和 b 陣列**: 用來存放稀疏矩陣和其轉置矩陣的非零元素。大小為 MAX_TERMS 的陣列足夠存放所有可能的非零元素。

程式邏輯與演算法

1.**transpose 函式**: 這個函式的目標是將稀疏矩陣a 轉置，然後存放到 b。函式首先讀取a 的第一個元素以獲取整個矩陣的行數、列數和非零元素的總數。隨後，函式通過列遍歷a，將每個元素的行標和列標交換後存入b，從而達到轉置的效果。

2.**print_matrix 函式**: 這個函式將傳入的稀疏矩陣陣列列印出來，格式為行標、列標和值。

測試程式的正確性

測試

1.**基礎測試**:使用範例中矩陣作為輸入資料創建矩陣a，並計算其轉置，然後與函式transpose 的輸出比較。

2.**隨機測試**: 隨機生成稀疏矩陣的非零元素，然後使用transpose 函式，並檢查輸出是否滿足轉置的基本規則(行列交換)。

3.**特殊情況**: 包括空矩陣(沒有非零元素)、只有一行或一列的矩陣、每行或每列只有一個非零元素的矩陣等情況，以確保函式在這些邊界條件下也能正常運作。

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define MAX_TERMS 100 /* Size of terms array */
 // A term is a non-zero element in the matrix
typedef struct {
    int row;
    int col;
    int value;
} term;
// The first element of the array is a special element that contains the number of rows, columns, and non-zero elements in the matrix
term a[MAX_TERMS];
term b[MAX_TERMS];
// Transpose the matrix 'a' and store the result in 'b'
void transpose(term a[], term b[]) {
    int n, i, j, currentb;
    n = a[0].value;
    b[0].row = a[0].col;
    b[0].col = a[0].row;
    b[0].value = n;
    // If the matrix is not empty, transpose the matrix
    if (n > 0) {
        currentb = 1;
        for (i = 0; i < a[0].col; i++) {
            for (j = 1; j <= n; j++) {
                if (a[j].col == i) {
                    b[currentb].row = a[j].col;
                    b[currentb].col = a[j].row;
                    b[currentb].value = a[j].value;
                    currentb++;
                }
            }
        }
    }
}
// Print the matrix
void print_matrix(term m[]) {
    int num_elements = m[0].value;
    printf("row col value\n");
    for (int i = 1; i <= num_elements; i++) {
        printf("%d  %d  %d\n", m[i].row, m[i].col, m[i].value);
    }
}
// Test the transpose function by initializing the matrix in the provided example
void basic_test() {
    printf("Basic Test\n");
    // Initialize matrix 'a' as in the provided example
    a[0].row = 6; a[0].col = 6; a[0].value = 8;
    a[1].row = 0; a[1].col = 0; a[1].value = 15;
    a[2].row = 0; a[2].col = 3; a[2].value = 22;
    a[3].row = 0; a[3].col = 5; a[3].value = -15;
    a[4].row = 1; a[4].col = 1; a[4].value = 11;
    a[5].row = 1; a[5].col = 2; a[5].value = 3;
    a[6].row = 2; a[6].col = 3; a[6].value = -6;
    a[7].row = 4; a[7].col = 0; a[7].value = 91;
    a[8].row = 5; a[8].col = 2; a[8].value = 28;

    transpose(a, b);
    print_matrix(b);
}
// Test the transpose function by initializing the matrix with random values
void random_test() {
    printf("\nRandom Test\n");
    srand((unsigned)time(NULL));
    int n = rand() % 10 + 1;
    a[0].row = 6; a[0].col = 6; a[0].value = n;
    for (int i = 1; i <= n; i++) {
        a[i].row = rand() % 6;
        a[i].col = rand() % 6;
        a[i].value = rand() % 20 + 1;
    }

    transpose(a, b);
    print_matrix(b);
}
// Test the transpose function with special cases
void special_cases_test() {
    printf("\nSpecial Cases Test - Empty Matrix\n");
    a[0].row = 6; a[0].col = 6; a[0].value = 0;
    transpose(a, b);
    print_matrix(b);

    printf("\nSpecial Cases Test - Single Row\n");
    a[0].row = 1; a[1].col = 6; a[0].value = 3;
    a[1].row = 0; a[1].col = 2; a[1].value = 5;
    a[2].row = 0; a[2].col = 5; a[2].value = -3;
    transpose(a, b);
    print_matrix(b);
}
// Main function to test the transpose function
int main() {
    basic_test();
    random_test();
    special_cases_test();

    return 0;
}
```

基礎測試

輸入測試資料

隨機測試

特殊測試

輸出

```
Basic Test
row col value
0   0   15
0   4   91
1   1   11
2   1   3
2   5   28
3   0   22
3   2   -6
5   0   -15

Random Test
row col value
1   2   14

Special Cases Test - Empty Matrix
row col value

Special Cases Test - Single Row
row col value
2   0   5
5   0   -3
```