TRƯỜNG ĐẠI HỌC SÀI GÒN

KHOA CÔNG NGHỆ THÔNG TIN

BÁO CÁO MÔN HỌC

CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

BÁO CÁO BÀI TẬP:

PHÂN TÍCH BÀI TOÁN, THIẾT KẾ GIẢI THUẬT VÀ KIỂM THỬ CHƯƠNG TRÌNH

SINH VIÊN THỰC HIỆN: Nguyễn Đỗ Huy – 3121411085

LỚP: DCT124C7

GVHD: ĐỖ NHƯ TÀI

Thành phố Hồ Chí Minh , Tháng 12 Năm 2024

MỤC LỤC

Bài 1. Giải Phương Trình Bậc 2

1. Phân tích bài toán
   1.1. Bài toán yêu cầu
   - Nhận hệ số a, b, c của phương trình bậc 2.
   - Giải phương trình và biện luận số nghiệm, sau đó xuất kết quả theo định dạng yêu cầu.

   1.2. Input
   - Một dòng chứa ba số a, b, c cách nhau bởi khoảng trắng.

   1.3. Output
   - Dòng 1: Thông báo về số nghiệm của phương trình, gồm:
     - "Phuong trinh co vo so nghiem" (nếu phương trình đúng với mọi giá trị x).
     - "Phuong trinh vo nghiem" (nếu phương trình không có nghiệm).
     - "Phuong trinh co 1 nghiem" (nếu có một nghiệm duy nhất).
     - "Phuong trinh co 2 nghiem" (nếu có hai nghiệm phân biệt).
   - Dòng 2: Liệt kê các nghiệm (nếu có), theo thứ tự tăng dần, định dạng số thập phân với 2 chữ số lẻ, cách nhau bởi khoảng trắng.

   1.4. Chức năng
   - Tính phương trình bậc 2.

2. Thiết kế giải thuật

```
Input a, b, c

If a == 0:
    If b == 0:
        If c == 0:
            Output "Phuong trinh co vo so nghiem"
        Else:
            Output "Phuong trinh vo nghiem"
    Else:
        x = -c / b
        Output "Phuong trinh co 1 nghiem"
        Output x with two odd numbers
Else (a ≠ 0):
    Δ = b^2 - 4*a*c
    If Δ < 0:
        Output "Phuong trinh vo nghiem"
    If Δ = 0:
        x = -b / (2*a)
        Output "Phuong trinh co 1 nghiem"
        Output x with two odd numbers
    If Δ > 0:
```

x1 = (-b - sqrt(Δ)) / (2*a)
x2 = (-b + sqrt(Δ)) / (2*a)
Arrange x1, x2 in ascending order
Output "Phuong trinh co 2 nghiem"
Output x1 and x2 with two odd numbers

3. Mã nguồn chương trình

```
#include <iostream>
#include <cmath>

using namespace std;

// Hàm giải phương trình bậc 2
void giaiptbac2(double a, double b, double c) {
    // Trường hợp đặc biệt: a = 0
    if (a == 0) {
        if (b == 0) {
            if (c == 0) {
                cout << "Phuong trinh co vo so nghiem" << endl;
            } else {
                cout << "Phuong trinh vo nghiem" << endl;
            }
        } else {
            // Phương trình bậc 1: bx + c = 0
            double x = -c / b;
            cout << "Phuong trinh co 1 nghiem" << endl;
            printf("%.2f\n", x);
        }
    } else {
        // Phương trình bậc 2: ax^2 + bx + c = 0
        double delta = b * b - 4 * a * c;

        if (delta < 0) {
            cout << "Phuong trinh vo nghiem" << endl;
        } else if (delta == 0) {
            // Phương trình có 1 nghiệm kép
            double x = -b / (2 * a);
            cout << "Phuong trinh co 1 nghiem" << endl;
            printf("%.2f\n", x);
        } else {
            // Phương trình có 2 nghiệm phân biệt
            double x1 = (-b - sqrt(delta)) / (2 * a);
            double x2 = (-b + sqrt(delta)) / (2 * a);

            // Sắp xếp nghiệm theo thứ tự tăng dần
            if (x1 > x2) {
```

```
                swap(x1, x2);
            }

            cout << "Phuong trinh co 2 nghiem" << endl;
            printf("%.2f %.2f\n", x1, x2);
        }
    }
}

// Hàm chứa tập testcase
void runTestCases() {
    // cout << "Running test cases...\n";

    // Testcase 1: Phương trình có vô số nghiệm
    // cout << "Testcase 1: ";
    giaiptbac2(0, 0, 0);
    cout << " --- \n";

    // Testcase 2: Phương trình vô nghiệm
    // cout << "Testcase 2: ";
    giaiptbac2(1, 2, 10);
    cout << " --- \n";

    // Testcase 3: Phương trình có 1 nghiệm (bậc 1)
    // cout << "Testcase 3: ";
    giaiptbac2(0, 2, -4);
    cout << " --- \n";

    // Testcase 4: Phương trình có 1 nghiệm kép
    // cout << "Testcase 4: ";
    giaiptbac2(1, 2, 1);
    cout << " --- \n";

    // Testcase 5: Phương trình có 2 nghiệm phân biệt
    // cout << "Testcase 5: ";
    giaiptbac2(1, -5, 6);
    cout << " --- \n";
}

// Hàm main
int main() {
    // // Chạy các testcase
    // runTestCases();

    // // Nhập liệu từ bàn phím
    // double a, b, c;
    // cout << "\nNhap a, b, c: ";
```

```
    // cin >> a >> b >> c;

    // // Giải phương trình với dữ liệu người dùng nhập
    // giaiptbac2(a, b, c);

    return 0;
}
```

4. Tập các testcase kiểm tra

Testcase	Input (a, b, c)	Kết quả mong đợi
1	0, 0, 0	Phuong trinh co vo so nghiem
2	1, 2, 10	Phuong trinh vo nghiem
3	1, 2, 1	Phuong trinh co 1 nghiem -1.00
4	1, -5, 6	Phuong trinh co 2 nghiem 2.00 3.00

Bài 2. Giải Phương Trình Trùng Phương

1. Phân tích bài toán
   1.1. Bài toán yêu cầu
   - Nhận hệ số a, b, c của phương trình trùng phương.
   - Giải và biện luận số nghiệm phương trình, sau đó xuất kết quả theo định dạng yêu cầu.

   1.2. Input
   - Một dòng chứa ba số a, b, c cách nhau bởi khoảng trắng.

   1.3. Output
   - Dòng 1: Thông báo về số nghiệm của phương trình, gồm:
     - "Phuong trinh co vo so nghiem" (nếu phương trình đúng với mọi giá trị x).
     - "Phuong trinh vo nghiem" (nếu phương trình không có nghiệm).
     - "Phuong trinh co [xx] nghiem" (với xx = 1, 2, 3 hay 4).
   - Dòng 2: Liệt kê các nghiệm (nếu có), theo thứ tự tăng dần, định dạng số thập phân với 2 chữ số lẻ, cách nhau bởi khoảng trắng.

   1.4. Chức năng
   - Tính phương trình trùng phương.

2. Thiết kế giải thuật

```
Input a, b, c

If a == 0:
   If b == 0:
      If c == 0:
         Output "Phuong trinh co vo so nghiem"
      Else:
         Output "Phuong trinh vo nghiem"
   Else:
      t = -c / b
      If t < 0:
         Output "Phuong trinh vo nghiem"
      Else:
         x1 = -sqrt(t)
         x2 = sqrt(t)
         Output "Phuong trinh co 2 nghiem"
         Output x1, x2 (sorted)

Else:
   Compute Δ = b^2 - 4*a*c
   If Δ < 0:
      Output "Phuong trinh vo nghiem"
```

Else If Δ == 0:
t = -b / (2*a)
If t < 0:
Output "Phuong trinh vo nghiem"
Else:
x1 = -sqrt(t)
x2 = sqrt(t)
Output "Phuong trinh co 2 nghiem"
Output x1, x2 (sorted)
Else:
t1 = (-b - sqrt(Δ)) / (2*a)
t2 = (-b + sqrt(Δ)) / (2*a)
List solutions = []
If t1 >= 0:
Add -sqrt(t1) and sqrt(t1) to solutions
If t2 >= 0:
Add -sqrt(t2) and sqrt(t2) to solutions
If solutions is empty:
Output "Phuong trinh vo nghiem"
Else:
Sort solutions
Output "Phuong trinh co [len(solutions)] nghiem"
Output solutions

3. Mã nguồn chương trình

```cpp
#include <iostream>
#include <cmath>

using namespace std;

// Hàm giải phương trình trùng phương
void giaipttrungphuong(double a, double b, double c) {
    // Kiểm tra xem phương trình có phải là phương trình bậc 2 hay không.
    if (a == 0) {
        if (b == 0) {
            if (c == 0) {
                cout << "Phuong trinh co vo so nghiem" << endl;
            } else {
                cout << "Phuong trinh vo nghiem" << endl;
            }
        } else {
            // Tính giá trị trung gian t để kiểm tra nghiệm của phương trình khi a
= 0.
            double t = -c / b;
```

```
            if (t < 0) {
                cout << "Phuong trinh vo nghiem" << endl;
            } else {
                double x1 = -sqrt(t);
                double x2 = sqrt(t);
                cout << "Phuong trinh co 2 nghiem" << endl;
                cout << x1 << " " << x2 << endl;
            }
        }
    } else {
        // Tính delta để kiểm tra nghiệm của phương trình bậc 4.
        double delta = b * b - 4 * a * c;
        if (delta < 0) {
            cout << "Phuong trinh vo nghiem" << endl;
        } else if (delta == 0) {
            double t = -b / (2 * a);
            if (t < 0) {
                cout << "Phuong trinh vo nghiem" << endl;
            } else {
                double x1 = -sqrt(t);
                double x2 = sqrt(t);
                cout << "Phuong trinh co 2 nghiem" << endl;
                cout << x1 << " " << x2 << endl;
            }
        } else {
            double t1 = (-b - sqrt(delta)) / (2 * a);
            double t2 = (-b + sqrt(delta)) / (2 * a);
            double solutions[4]; // Mảng tĩnh để lưu nghiệm của phương trình (tối
đa 4 nghiệm).
            int count = 0; // Biến đếm số lượng nghiệm tìm được.

            if (t1 >= 0) {
                solutions[count++] = -sqrt(t1); // Thêm nghiệm \(-\sqrt{t1}\) vào
mảng solutions và tăng biến đếm.
                solutions[count++] = sqrt(t1);
            }
            if (t2 >= 0) {
                solutions[count++] = -sqrt(t2);
                solutions[count++] = sqrt(t2);
            }

            if (count == 0) {
                cout << "Phuong trinh vo nghiem" << endl;
            } else {
                cout << "Phuong trinh co " << count << " nghiem" << endl;
                // Duyệt qua các nghiệm trong mảng để sắp xếp chúng theo thứ tự
tăng dần.
```

```
                for (int i = 0; i < count; i++) {
                    for (int j = i + 1; j < count; j++) {
                        if (solutions[i] > solutions[j]) {
                            double temp = solutions[i];
                            solutions[i] = solutions[j];
                            solutions[j] = temp;
                        }
                    }
                }
                for (int i = 0; i < count; i++) {
                    cout << solutions[i] << " ";
                }
                cout << endl;
            }
        }
    }
}

// Hàm chứa tập testcase
void runTestCases() {
    // cout << "Running test cases..." << endl;

    // Testcase 1: Phương trình có vô số nghiệm
    // cout << "Testcase 1: ";
    giaipttrungphuong(0, 0, 0);
    cout << " --- \n";

    // Testcase 2: Phương trình vô nghiệm
    // cout << "Testcase 2: ";
    giaipttrungphuong(1, 2, 10);
    cout << " --- \n";

    // Testcase 3: Phương trình có 2 nghiệm
    // cout << "Testcase 3: ";
    giaipttrungphuong(1, -2, 1);
    cout << " --- \n";

    // Testcase 4: Phương trình có 4 nghiệm
    // cout << "Testcase 4: ";
    giaipttrungphuong(1, -5, 4);
    cout << " --- \n";

    // Testcase 5: Phương trình vô nghiệm do t < 0
    // cout << "Testcase 5: ";
    giaipttrungphuong(1, 2, 1);
    cout << " --- \n";
}
```

```
// Hàm main
int main() {
    // // Chạy các testcase
    // runTestCases();

    // // Nhập liệu từ bàn phím
    // double a, b, c;
    // cout << "\nNhap a, b, c: ";
    // cin >> a >> b >> c;

    // // Giải phương trình với dữ liệu người dùng nhập
    // giaipttrungphuong(a, b, c);

    return 0;
}
```

4. Tập các testcase kiểm tra

Testcase	Input (a, b, c)	Kết quả mong đợi
1	0, 0, 0	Phuong trinh co vo so nghiem
2	1, 2, 10	Phuong trinh vo nghiem
3	1, -2, 1	Phuong trinh co 2 nghiem
4	1, -5, 4	Phuong trinh co 4 nghiem

Bài 3. Phần tử chung

1. Phân tích bài toán

1.1. Bài toán yêu cầu

- Nhận ba dãy số nguyên dương x, y, z lần lượt có nx, ny, nz phần tử.
- Tìm phần tử chung của ba dãy số trên, sau đó xuất kết quả theo định dạng yêu cầu.

1.2. Input

- Số lượng phần tử nx, ny, nz của ba dãy số x, y, z.
- Các phần tử của ba dãy số: x, y, z.

1.3. Output

- Số lượng phần tử chung.
- Danh sách các phần tử chung (theo thứ tự tăng dần, không trùng lặp).

1.4. Ràng buộc

- $0 \leq nx, ny, nz \leq 1000$: Số phần tử trong mỗi dãy tối đa là 1000.
- $0 \leq xi, yi, zi \leq 10000$: Giá trị mỗi phần tử nằm trong [0, 10000].

1.5. Chức năng

- Tìm phần tử chung của các dãy.

2. Thiết kế giải thuật

FUNCTION findCommonElements(nx, x, ny, y, nz, z):

// Tạo mảng đếm cho từng dãy (kích thước cố định 10001)

array count_x[10001] <- tất cả giá trị khởi tạo 0

array count_y[10001] <- tất cả giá trị khởi tạo 0

array count_z[10001] <- tất cả giá trị khởi tạo 0

// Đếm số lần xuất hiện của từng phần tử trong mỗi dãy

FOR i FROM 0 TO nx - 1:

count_x[x[i]] <- 1

FOR i FROM 0 TO ny - 1:

count_y[y[i]] <- 1

FOR i FROM 0 TO nz - 1:

```
        count_z[z[i]] <- 1

    // Tìm phần tử chung bằng cách kiểm tra mảng đếm
    common_elements <- empty list
    FOR i FROM 0 TO 10000:
        IF count_x[i] == 1 AND count_y[i] == 1 AND count_z[i] == 1:
            ADD i TO common_elements

    // In kết quả
    PRINT size of common_elements
    FOR element IN common_elements:
        PRINT element

END FUNCTION

// Main
MAIN:
    // Nhập dữ liệu
    READ nx
    READ array x of size nx
    READ ny
    READ array y of size ny
    READ nz
    READ array z of size nz

    // Gọi hàm xử lý
    CALL findCommonElements(nx, x, ny, y, nz, z)
END MAIN
```

3. Mã nguồn chương trình

```cpp
#include <iostream>
#include <cmath>

using namespace std;

// Hàm tìm phần tử chung của ba dãy
void findCommonElements(int nx, int x[], int ny, int y[], int nz, int z[]) {
    // Khởi tạo mảng đếm
    int count_x[10001] = {0};
    int count_y[10001] = {0};
    int count_z[10001] = {0};

    // Đếm số lần xuất hiện của các phần tử trong mỗi dãy
    for (int i = 0; i < nx; i++) {
        count_x[x[i]] = 1;
    }
    for (int i = 0; i < ny; i++) {
        count_y[y[i]] = 1;
    }
    for (int i = 0; i < nz; i++) {
        count_z[z[i]] = 1;
    }

    // Tìm phần tử chung
    int common_count = 0;
    int common_elements[10001];
    for (int i = 0; i <= 10000; i++) {
        if (count_x[i] == 1 && count_y[i] == 1 && count_z[i] == 1) {
            common_elements[common_count++] = i; // Lưu phần tử chung
        }
    }

    // Xuất kết quả
    cout << common_count << endl; // In số lượng phần tử chung
    for (int i = 0; i < common_count; i++) {
        cout << common_elements[i] << " ";
    }
    if (common_count > 0) {
        cout << endl;
    }
}

// Hàm chứa tập testcase
void runTestCases() {
    // Testcase 1
    // cout << "Testcase 1:\n";
```

```
    int nx1 = 5;
    int x1[] = {1, 2, 5, 4, 3};
    int ny1 = 4;
    int y1[] = {5, 6, 1, 4};
    int nz1 = 4;
    int z1[] = {5, 3, 5, 1};
    findCommonElements(nx1, x1, ny1, y1, nz1, z1);
    cout << " --- \n";

    // Testcase 2
    // cout << "Testcase 2:\n";
    int nx2 = 6;
    int x2[] = {10, 20, 30, 40, 50, 60};
    int ny2 = 5;
    int y2[] = {15, 25, 35, 40, 50};
    int nz2 = 7;
    int z2[] = {5, 10, 20, 30, 40, 50, 60};
    findCommonElements(nx2, x2, ny2, y2, nz2, z2);
    cout << " --- \n";

    // Testcase 3
    // cout << "Testcase 3:\n";
    int nx3 = 3;
    int x3[] = {1, 1, 1};
    int ny3 = 3;
    int y3[] = {1, 1, 1};
    int nz3 = 3;
    int z3[] = {1, 1, 1};
    findCommonElements(nx3, x3, ny3, y3, nz3, z3);
    cout << " --- \n";

    // Testcase 4
    // cout << "Testcase 4:\n";
    int nx4 = 0; // Không có phần tử
    int x4[] = {};
    int ny4 = 0;
    int y4[] = {};
    int nz4 = 0;
    int z4[] = {};
    findCommonElements(nx4, x4, ny4, y4, nz4, z4);
    cout << " --- \n";

    // Testcase 5
    // cout << "Testcase 5:\n";
    int nx5 = 4;
    int x5[] = {10000, 9999, 9998, 9997};
    int ny5 = 4;
```

```
    int y5[] = {9997, 9998, 10000, 9999};
    int nz5 = 4;
    int z5[] = {9999, 9997, 10000, 9998};
    findCommonElements(nx5, x5, ny5, y5, nz5, z5);
    cout << " --- \n";
    cout << endl;
}

int main() {
    // Chạy các testcase
    // runTestCases();

    // // Nhập dữ liệu từ bàn phím
    // int nx, ny, nz;
    // cin >> nx; // Số phần tử của dãy x
    // int x[nx];
    // for (int i = 0; i < nx; i++) {
    //     cin >> x[i];
    // }

    // cin >> ny; // Số phần tử của dãy y
    // int y[ny];
    // for (int i = 0; i < ny; i++) {
    //     cin >> y[i];
    // }

    // cin >> nz; // Số phần tử của dãy z
    // int z[nz];
    // for (int i = 0; i < nz; i++) {
    //     cin >> z[i];
    // }

    // // Gọi hàm xử lý và in kết quả
    // findCommonElements(nx, x, ny, y, nz, z);

    return 0;
}
```

4. Tập các testcase kiểm tra

Testcase	Input	Kết quả mong đợi
1	5 1 2 5 4 3 4 5 6 1 4 4	2 1 5

	5 3 5 1	
2	6 10 20 30 40 50 60 5 15 25 35 40 50 7 5 10 20 30 40 50 60	2 40 50
3	3 1 1 1 3 1 1 1 3 1 1 1	1 1
4	0 0 0	0
5	4 10000 9999 9998 9997 4 9997 9998 10000 9999 4 9999 9997 10000 9998	4 9997 9998 9999 10000