Mục lục nội dung

**Bài thực hành 4: Thực hành sử dụng các cấu trúc dữ liệu cơ bản để giải quyết các bài toán cụ thể**

**Phần 1: Bài tập thực hành**

**Bài tập 1:** Đảo ngược một danh sách liên kết đơn

Hãy hoàn thiện các hàm thao tác trên một danh sách liên kết:

Thêm một phần tử vào đầu danh sách liên kết

In danh sách

Đảo ngược danh sách liên kết (yêu cầu độ phức tạp thời gian O(N) và chi phí bộ nhớ dùng thêm O(1))

**Bài tập 2:** Tính diện tích tam giác

Một điểm trong không gian 2 chiều được biểu diễn bằng pair. Hãy viết hàm double area(Point a, Point b, Point c) tính diện tích tam giác theo tọa độ 3 đỉnh. Trong đó, Point là kiểu được định nghĩa sẵn trong trình chấm như sau: using Point = pair<double, double>;

**Bài tập 3:** Tính tích có hướng của 2 vector

Một vector trong không gian 3 chiều được biểu diễn bằng tuple<double, double, double>. Hãy viết hàm Vector cross\_product(Vector a, Vector b) tính tích có hướng của 2 vector. Trong đó Vector là kiểu dữ liệu được định nghĩa sẵn trong trình chấm như sau: using Vector = tuple<double, double, double>;

**Bài tập 4:** Thao tác với vector

Cho hai vector, hãy xóa hết các phần tử chẵn, sắp xếp giảm dần các số trong cả 2 vector và trộn lại thành một vector cũng được sắp xếp giảm dần.

**Bài tập 5:**

Viết hàm thực hiện thuật toán DFS không sử dụng đệ quy trên đồ thị biểu diễn bằng danh sách

kề vector< list<int> > . Đồ thị có n đỉnh được đánh số từ 1 đến n. Thuật toán DFS xuất phát từ đỉnh

1. Các đỉnh được thăm theo thứ tự ưu tiên từ trái sang phải trong danh sách kề. Yêu cầu hàm trả ra thứ tự các đỉnh được thăm (những đỉnh không thể thăm từ đỉnh 1 thì không phải in ra).

**Bài tập 6:**

Viết hàm thực hiện thuật toán BFS không sử dụng đệ quy trên đồ thị biểu diễn bằng danh sách

kề vector< list<int> > . Đồ thị có n đỉnh được đánh số từ 1 đến n. Thuật toán BFS xuất phát từ đỉnh

1. Các đỉnh được thăm theo thứ tự ưu tiên từ trái sang phải trong danh sách kề. Yêu cầu hàm trả ra thứ tự các đỉnh được thăm (những đỉnh không thể thăm từ đỉnh 1 thì không phải in ra).

**Bài tập 7:**

Viết các hàm thực hiện các phép giao và hợp của hai tập hợp được biểu diễn bằng set.

**Bài tập 8:**

Viết các hàm thực hiện các phép giao và hợp của hai tập hợp mờ được biểu diễn bằng map.

**Bài tập 9:**

Cài đặt thuật toán Dijkstra trên đồ thị vô hướng được biểu diễn bằng danh sách kề sử dụng priority\_queue Cụ thể, bạn cần cài đặt hàm vector<int> dijkstra(const vector< vector< pair<int, int> > >&adj) nhận đầu vào là danh sách kề chứa các cặp pair<int, int> biểu diễn đỉnh kề và trọng số tương ứng của cạnh. Đồ thị gồm n đỉnh được đánh số từ 0 tới n-1. Hàm cần trả vector<int> chứa n phần tử lần lượt là khoảng cách đường đi ngắn nhất từ đỉnh 0 tới các đỉnh 0, 1, 2, ..., n-1

**Phần 2: Bài tập về nhà**

**Bài tập 10: Search Engine**

Xây dựng một máy tìm kiếm (search engine) đơn giản

Cho NN văn bản và QQ truy vấn. Với mỗi truy vấn, cần trả về văn bản khớp với truy vấn đó nhất.

**Bài tập 11: Lịch trình chụp ảnh**

Superior là một hòn đảo tuyệt đẹp với nn địa điểm chụp ảnh và các đường một chiều nối các điểm chụp ảnh với nhau. Đoàn khách tham quan có rr người với sở thích chụp ảnh khác nhau. Theo đó, mỗi người sẽ đưa ra danh sách các địa điểm mà họ muốn chụp. Bạn cần giúp mỗi người trong đoàn lập lịch di chuyển sao cho đi qua các điểm họ yêu cầu đúng một lần, không đi qua điểm nào khác, bắt đầu tại điểm đầu tiên và kết thúc tại điểm cuối cùng trong danh sách mà họ đưa ra, và có tổng khoảng cách đi lại là nhỏ nhất.

**Bài tập 12: Đếm đường đi**

Cho đồ thị vô hướng GG, hãy đếm số đường đi đi qua kk cạnh và không đi qua đỉnh nào quá một lần.

Mục lục hình ảnh

[Hình 1 Bài 4.1 Đảo ngược danh sách liên kết đơn 3](#_Toc161238452)

[Hình 2 Bài 4.2 Viết hàm tính diện tích tam giác 5](#_Toc161238453)

[Hình 3 Bài 4.3 Viết hàm tính tích có hướng của 2 vector 13](#_Toc161238454)

[Hình 4 Bài 4.4 Viết hàm xóa phần tử, sắp xếp vector 9](#_Toc161238455)

[Hình 5 Bài 4.5 Hàm trả thứ tự các đỉnh được thăm theo DFS 11](#_Toc161238456)

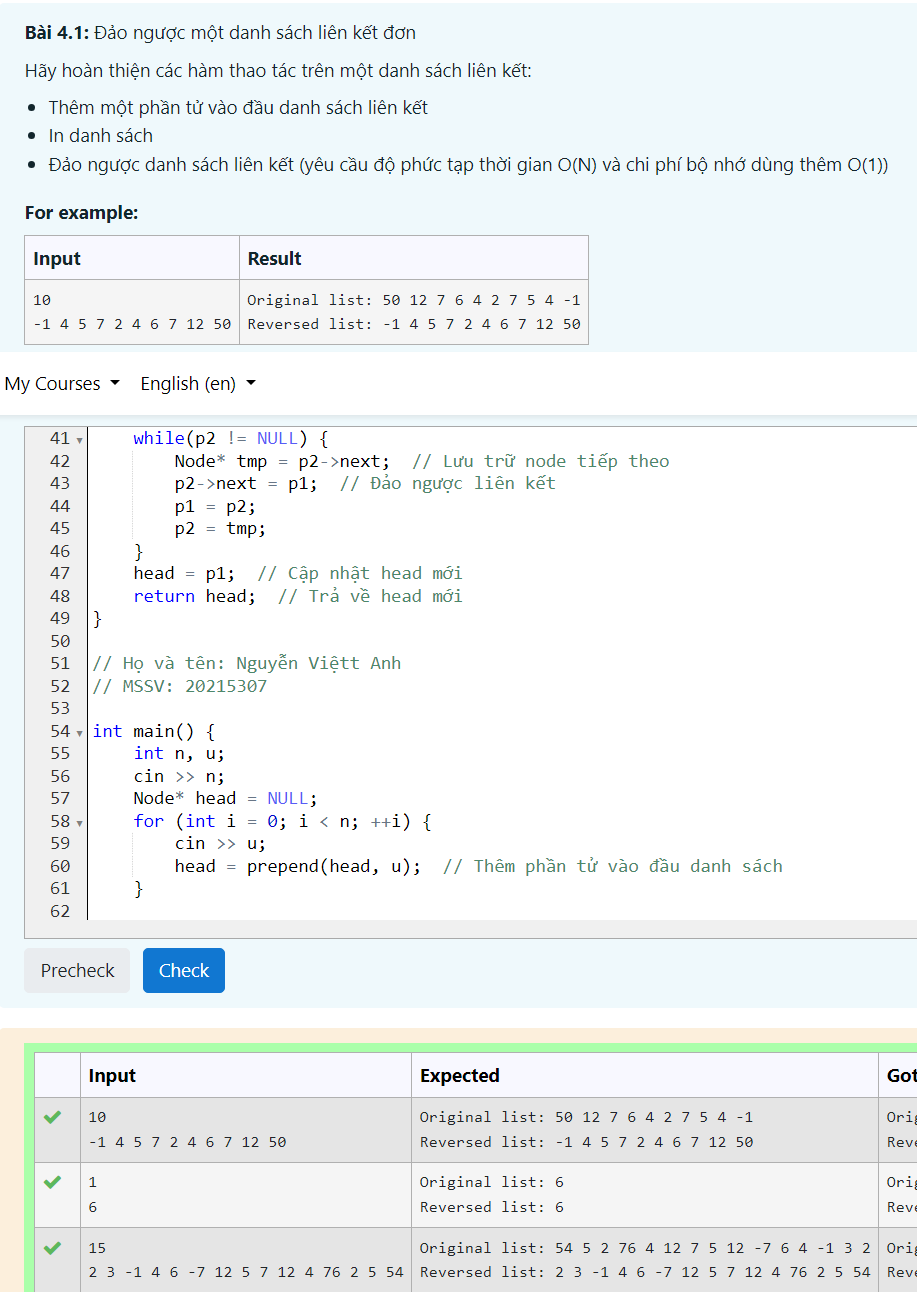
[Hình 6 Bài 4.6 Hàm trả thứ tự các đỉnh được thăm theo BFS 15](#_Toc161238457)

[Hình 7 Bài 4.7 Viết hàm thực hiện các phép giao và hợp của hai tập hợp được biểu diễn bằng set 126](#_Toc161238458)

[Hình 8 Bài 4.8 Viết hàm thực hiện các phép giao và hợp của hai tập hợp được biểu diễn bằng map 21](#_Toc161238459)

Bài thực hành số 4 – Tuần 15

# BÀI TẬP TRÊN LAP



Hình Bài 4.1: Đảo ngược danh sách liên kết đơn

#include <iostream>

using namespace std;

struct Node {

    int data;

    Node\* next;

    Node(int data) {

        this->data = data;  // Khởi tạo giá trị data của node

        next = NULL;  // Khởi tạo next là NULL

    }

};

// Thêm một phần tử mới vào đầu danh sách

Node\* prepend(Node\* head, int data) {

    Node\* q = new Node(data);  // Tạo một node mới với giá trị data

    if(head == NULL) return q;

    q->next = head;  // Liên kết node mới với head hiện tại

    head = q;

    return head;

}

// In nội dung của danh sách trên một dòng

void print(Node\* head) {

    Node\* p = head;

    if(head == NULL) return;  // Nếu danh sách rỗng, thoát khỏi hàm

    while(p != NULL) {

        cout << p->data << ' ';  // In giá trị của node hiện tại

        p = p->next;  // Di chuyển tới node tiếp theo

    }

    return;

}

// Trả về head mới của danh sách đã đảo ngược

Node\* reverse(Node\* head) {

    if(head == NULL) return NULL;  // Nếu danh sách rỗng, trả về NULL

    if(head->next == NULL) return head;  // Nếu danh sách chỉ có một phần tử, trả về head

    Node\* p1 = head;  // Khởi tạo p1 là head

    Node\* p2 = head->next;  // Khởi tạo p2 là node thứ hai

    p1->next = NULL;  // Ngắt liên kết của node đầu tiên

    while(p2 != NULL) {

        Node\* tmp = p2->next;  // Lưu trữ node tiếp theo

        p2->next = p1;  // Đảo ngược liên kết

        p1 = p2;

        p2 = tmp;

    }

    head = p1;  // Cập nhật head mới

    return head;  // Trả về head mới

}

// Họ và tên: Nguyễn Việtt Anh

// MSSV: 20215307

int main() {

    cout << "Ho va ten: Nguyen Viet Anh\n";

    cout << "MSSV: 20215307\n";

    int n, u;

    cin >> n;

    Node\* head = NULL;

    for (int i = 0; i < n; ++i) {

        cin >> u;

        head = prepend(head, u);  // Thêm phần tử vào đầu danh sách

    }

    cout << "Original list: ";

    print(head);  // In danh sách ban đầu

    head = reverse(head);  // Đảo ngược danh sách

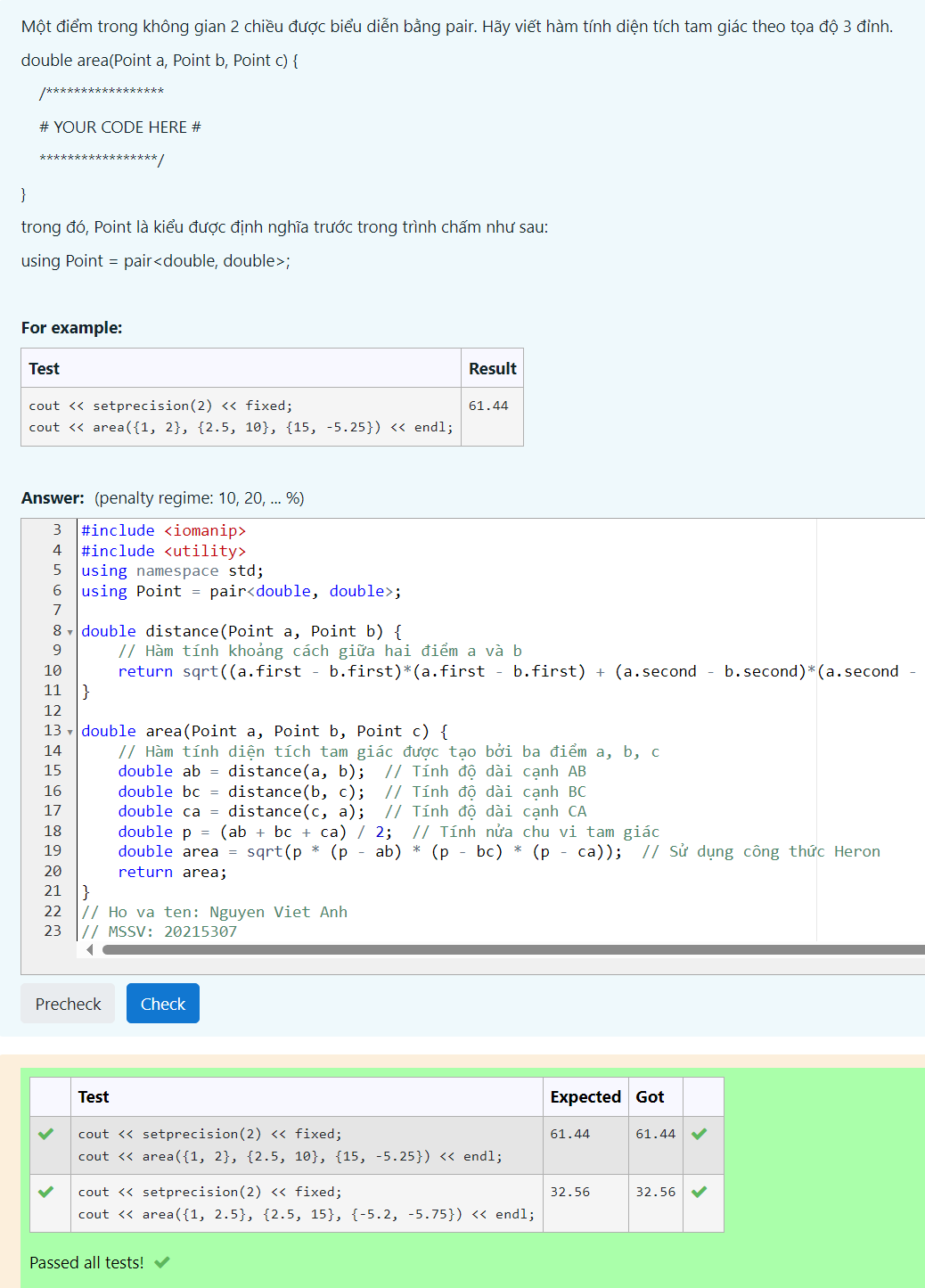
    cout << endl;

    cout << "Reversed list: ";

    print(head);  // In danh sách đã đảo ngược

    return 0;

}



Hình Bài 4.2: Viết hàm tính diện tích tam giác

#include <iostream>

#include <cmath>

#include <iomanip>

#include <utility>

using namespace std;

using Point = pair<double, double>;

double distance(Point a, Point b) {

    // Hàm tính khoảng cách giữa hai điểm a và b

    return sqrt((a.first - b.first)\*(a.first - b.first) + (a.second - b.second)\*(a.second - b.second));

}

double area(Point a, Point b, Point c) {

    // Hàm tính diện tích tam giác được tạo bởi ba điểm a, b, c

    double ab = distance(a, b);  // Tính độ dài cạnh AB

    double bc = distance(b, c);  // Tính độ dài cạnh BC

    double ca = distance(c, a);  // Tính độ dài cạnh CA

    double p = (ab + bc + ca) / 2;  // Tính nửa chu vi tam giác

    double area = sqrt(p \* (p - ab) \* (p - bc) \* (p - ca));  // Sử dụng công thức Heron

    return area;

}

// Ho va ten: Nguyen Viet Anh

// MSSV: 20215307

int main() {

    cout << "Ho va ten: Nguyen Viet Anh\n";

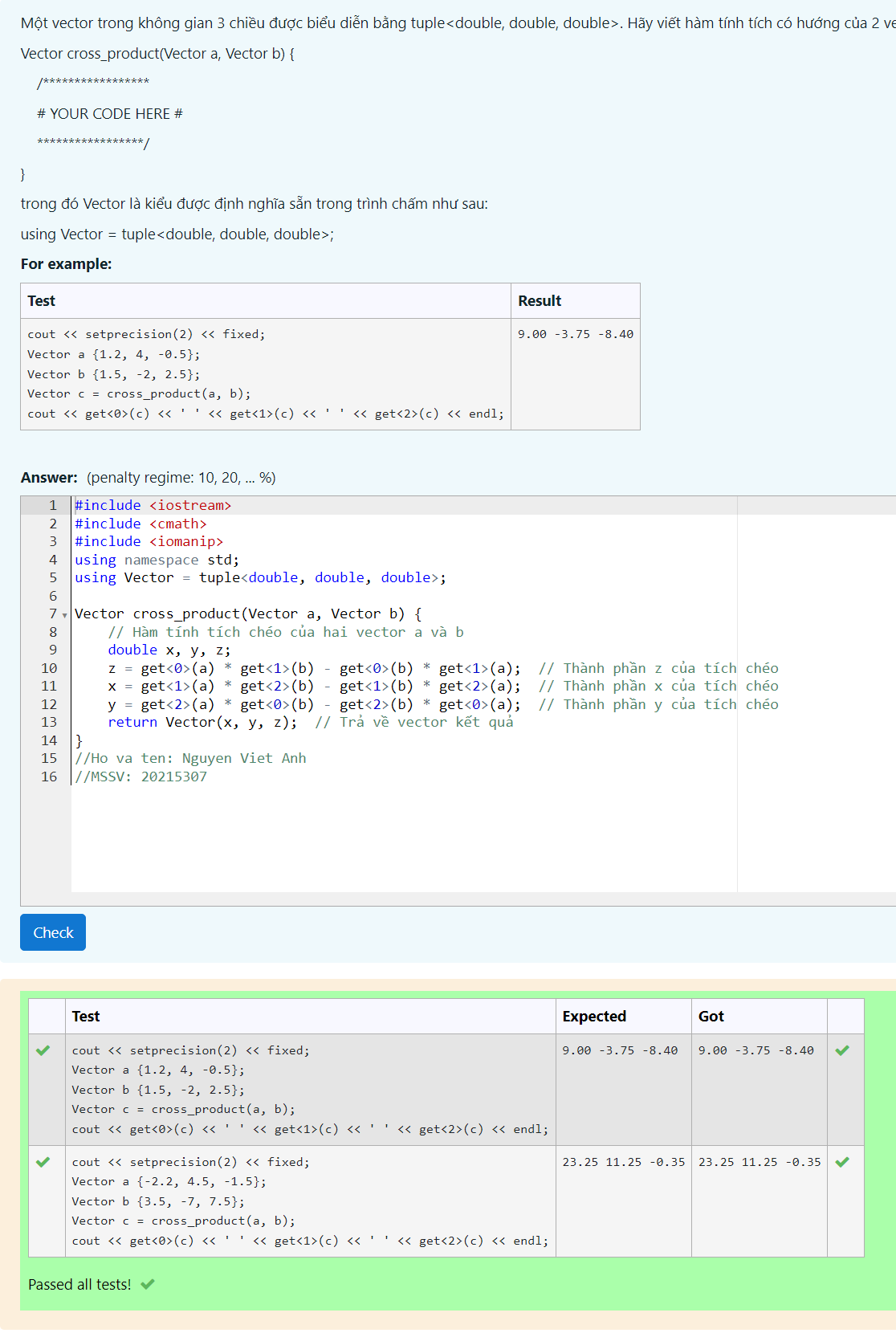
    cout << "MSSV: 20215307\n";

    cout << setprecision(2) << fixed;

    cout << area({1, 2}, {2.5, 10}, {15, -5.25}) << endl;

    return 0;

}



Hình Bài 4.3 Viết hàm tính tích có hướng của 2 vector

#include <iostream>

#include <cmath>

#include <iomanip>

using namespace std;

using Vector = tuple<double, double, double>;

Vector cross\_product(Vector a, Vector b) {

    // Hàm tính tích chéo của hai vector a và b

    double x, y, z;

    z = get<0>(a) \* get<1>(b) - get<0>(b) \* get<1>(a);  // Thành phần z của tích chéo

    x = get<1>(a) \* get<2>(b) - get<1>(b) \* get<2>(a);  // Thành phần x của tích chéo

    y = get<2>(a) \* get<0>(b) - get<2>(b) \* get<0>(a);  // Thành phần y của tích chéo

    return Vector(x, y, z);  // Trả về vector kết quả

}

//Ho va ten: Nguyen Viet Anh

//MSSV: 20215307

int main() {

    cout << "Ho va ten: Nguyen Viet Anh\n";

    cout << "MSSV: 20215307\n";

    cout << setprecision(2) << fixed;

    Vector a {1.2, 4, -0.5};

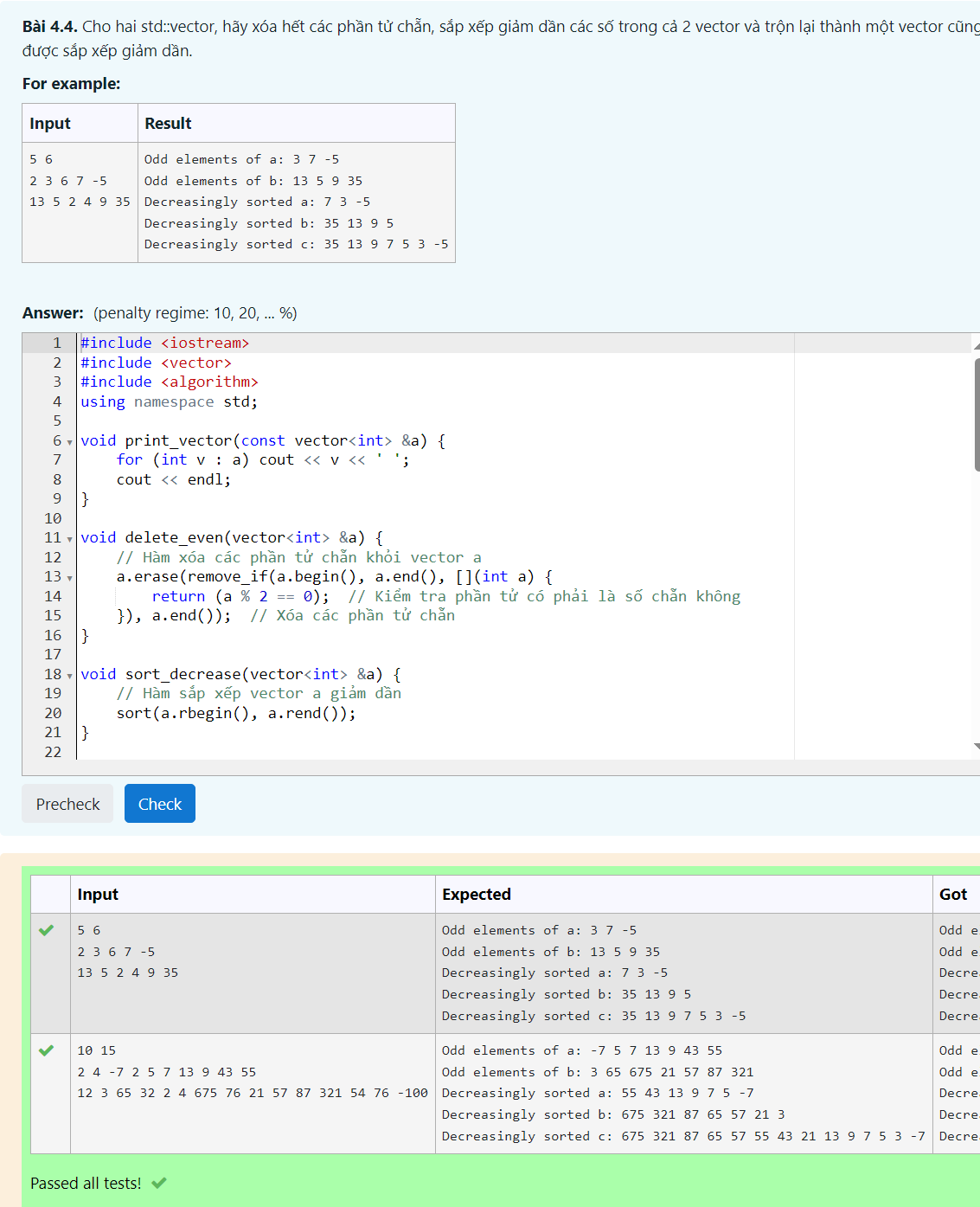
    Vector b {1.5, -2, 2.5};

    Vector c = cross\_product(a, b);

    cout << get<0>(c) << ' ' << get<1>(c) << ' ' << get<2>(c) << endl;

    return 0;

}



Hình Bài 4.4 Xóa phần tử, sắp xếp vector

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

void print\_vector(const vector<int> &a) {

    for (int v : a) cout << v << ' ';

    cout << endl;

}

void delete\_even(vector<int> &a) {

    // Hàm xóa các phần tử chẵn khỏi vector a

    a.erase(remove\_if(a.begin(), a.end(), [](int a) {

        return (a % 2 == 0);  // Kiểm tra phần tử có phải là số chẵn không

    }), a.end());  // Xóa các phần tử chẵn

}

void sort\_decrease(vector<int> &a) {

    // Hàm sắp xếp vector a giảm dần

    sort(a.rbegin(), a.rend());

}

//Ho va ten: Nguyen Viet Anh

//MSSV: 20215307

vector<int> merge\_vectors(const vector<int> &a, const vector<int> &b) {

    // Hàm hợp nhất hai vector a và b, sau đó sắp xếp theo thứ tự giảm dần

    vector<int> sum;  // Vector kết quả

    for(int v : a) {

        sum.push\_back(v);  // Thêm các phần tử của a vào sum

    }

    for(int v : b) {

        sum.push\_back(v);  // Thêm các phần tử của b vào sum

    }

    sort(sum.rbegin(), sum.rend());  // Sắp xếp vector sum theo thứ tự giảm dần

    return sum;

}

int main() {

    cout << "Ho va ten: Nguyen Viet Anh\n";

    cout << "MSSV: 20215307\n";

    int m, n, u;

    vector<int> a, b;

    cin >> m >> n;

    for(int i = 0; i < m; i++) {

        cin >> u;

        a.push\_back(u);

    }

    for(int i = 0; i < n; i++) {

        cin >> u;

        b.push\_back(u);

    }

    delete\_even(a);

    cout << "Odd elements of a: ";

    print\_vector(a);

    delete\_even(b);

    cout << "Odd elements of b: ";

    print\_vector(b);

    sort\_decrease(a);

    cout << "Decreasingly sorted a: ";

    print\_vector(a);

    sort\_decrease(b);

    cout << "Decreasingly sorted b: ";

    print\_vector(b);

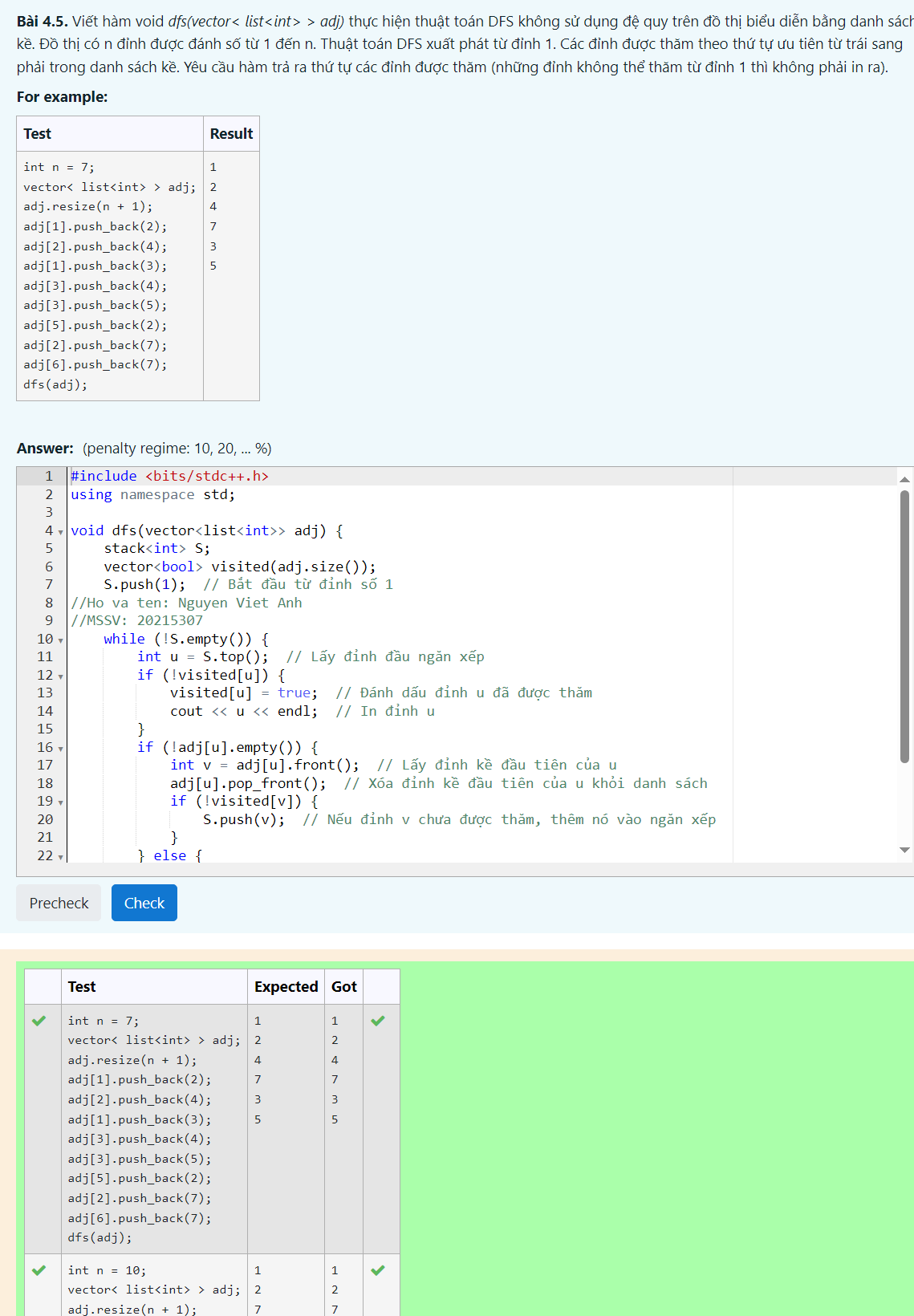
    vector<int> c = merge\_vectors(a, b);

    cout << "Decreasingly sorted c: ";

    print\_vector(c);

    return 0;

}



Hình Bài 4.5 Hàm trả thứ tự các đỉnh được thăm theo DFS

#include <bits/stdc++.h>

using namespace std;

void dfs(vector<list<int>> adj) {

    stack<int> S;

    vector<bool> visited(adj.size());

    S.push(1);  // Bắt đầu từ đỉnh số 1

//Ho va ten: Nguyen Viet Anh

//MSSV: 20215307

    while (!S.empty()) {

        int u = S.top();  // Lấy đỉnh đầu ngăn xếp

        if (!visited[u]) {

            visited[u] = true;  // Đánh dấu đỉnh u đã được thăm

            cout << u << endl;  // In đỉnh u

        }

        if (!adj[u].empty()) {

            int v = adj[u].front();  // Lấy đỉnh kề đầu tiên của u

            adj[u].pop\_front();  // Xóa đỉnh kề đầu tiên của u khỏi danh sách

            if (!visited[v]) {

                S.push(v);  // Nếu đỉnh v chưa được thăm, thêm nó vào ngăn xếp

            }

        } else {

            S.pop();  // Nếu tất cả các đỉnh kề của u đã được thăm, loại bỏ u khỏi ngăn xếp

        }

    }

}

int main() {

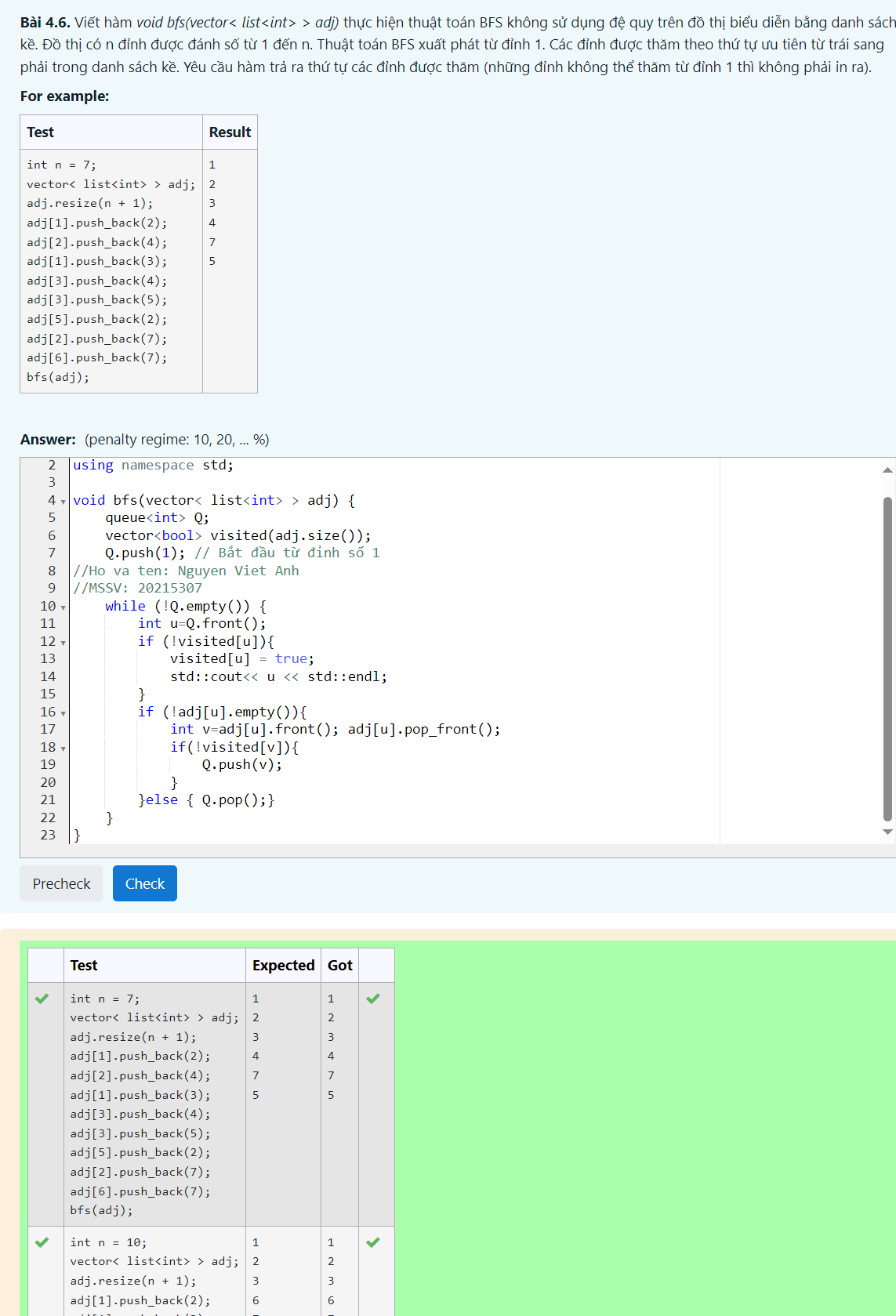
    cout << "Ho va ten: Nguyen Viet Anh\n";

    cout << "MSSV:20215307\n";

    // Chưa có mã cụ thể trong hàm main

    return 0;

}



Hình Bài 4.6 Hàm trả thứ tự các đỉnh được thăm theo BFS

#include <bits/stdc++.h>

using namespace std;

void bfs(vector<list<int>> adj) {

    queue<int> Q;

    vector<bool> visited(adj.size());

    Q.push(1);  // Bắt đầu từ đỉnh số 1

//Ho va ten:Nguyen Viet Anh

//MSSV: 20215307

    while (!Q.empty()) {

        int u = Q.front();  // Lấy đỉnh đầu hàng đợi

        if (!visited[u]) {

            visited[u] = true;  // Đánh dấu đỉnh u đã được thăm

            cout << u << endl;

        }

        if (!adj[u].empty()) {

            int v = adj[u].front();  // Lấy đỉnh kề đầu tiên của u

            adj[u].pop\_front();  // Xóa đỉnh kề đầu tiên của u khỏi danh sách

            if (!visited[v]) {

                Q.push(v);  // Nếu đỉnh v chưa được thăm, thêm nó vào hàng đợi

            }

        } else {

            Q.pop();  // Nếu tất cả đỉnh kề của u đã được thăm, bỏ u khỏi hàng đợi

        }

    }

}

int main() {

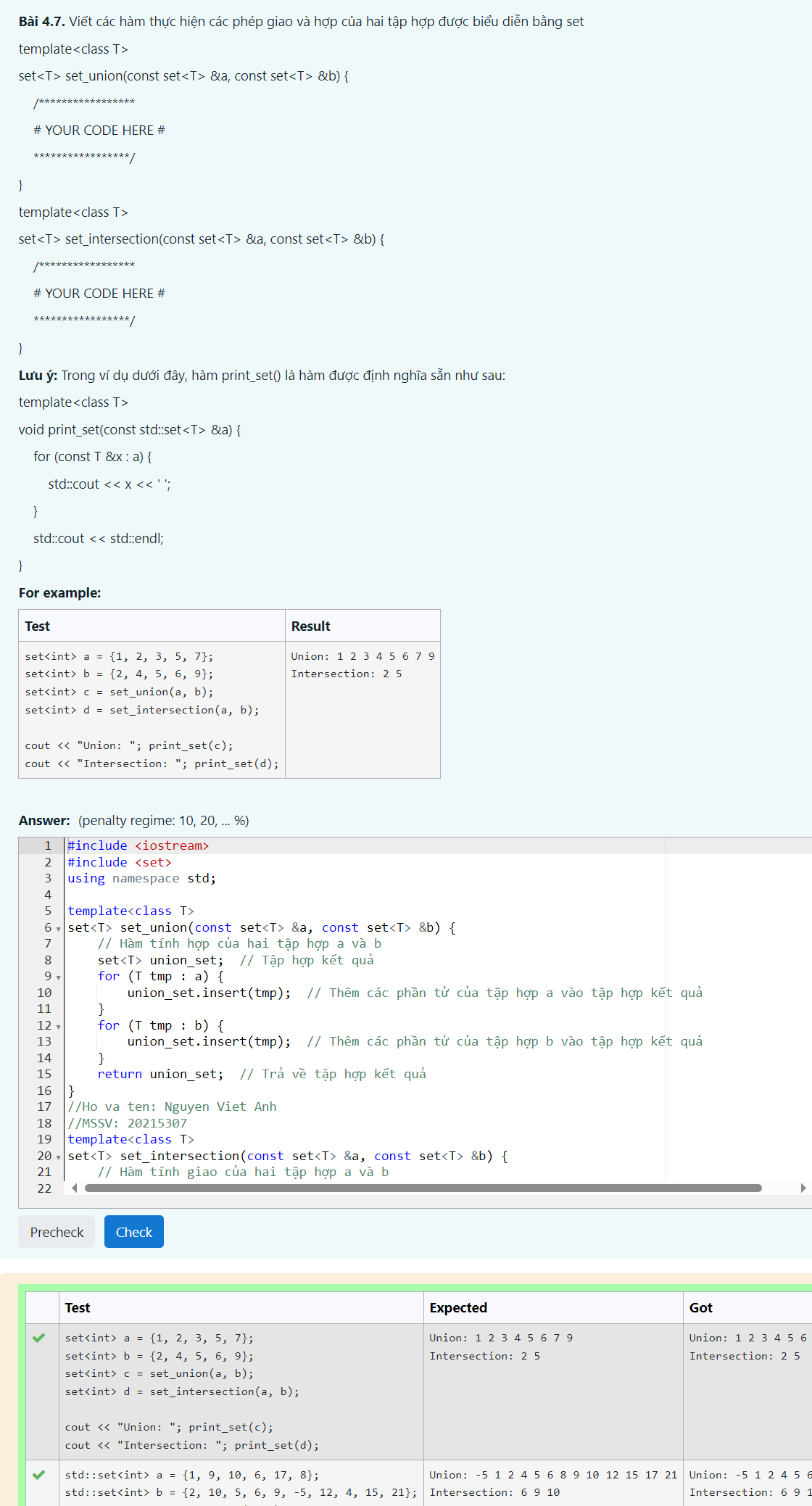
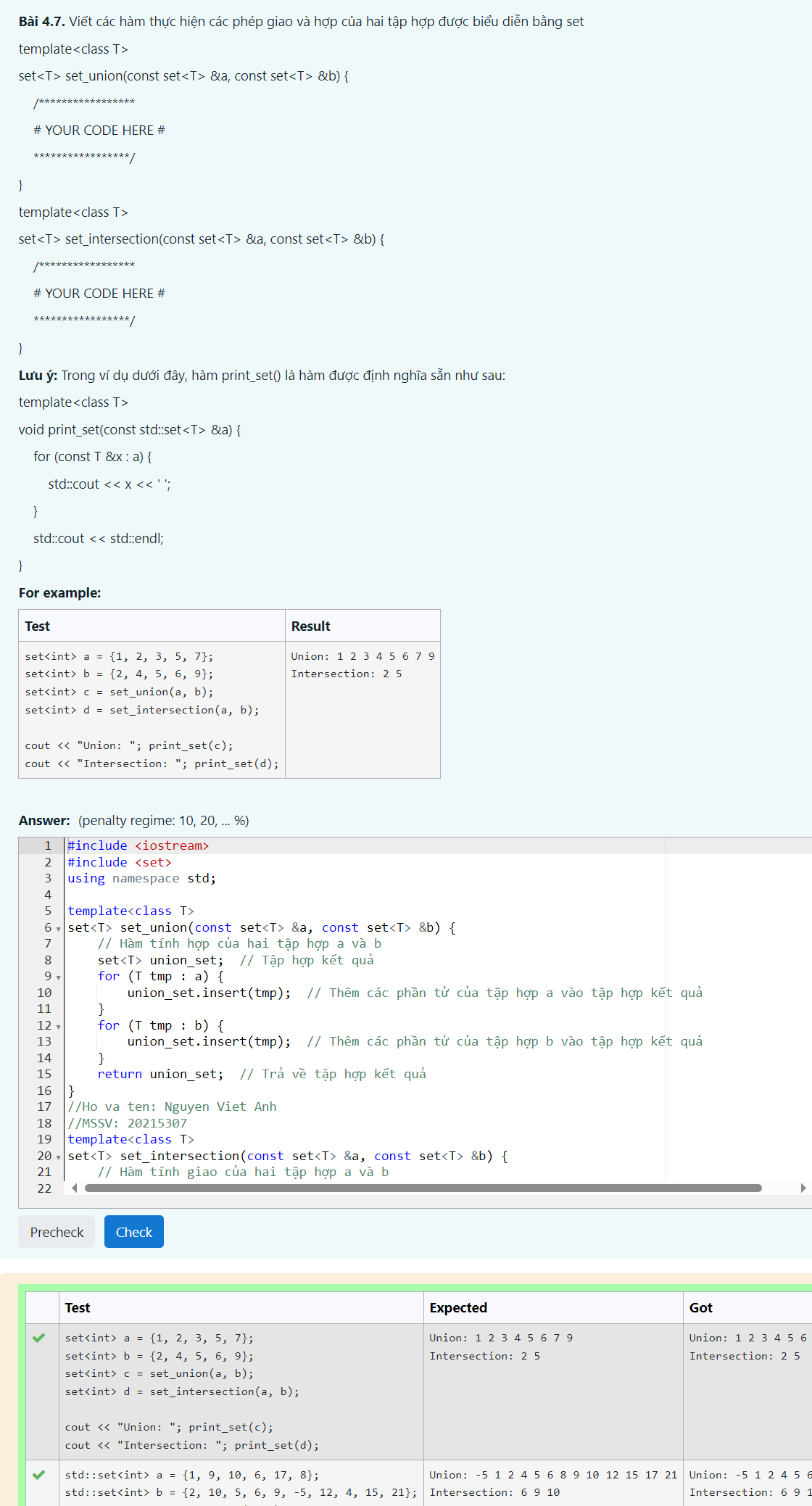
    cout << "Ho va ten: Nguyen Viet Anh\n";

    cout << "MSSV: 20215307\n";

    // Chưa có mã cụ thể trong hàm main

    return 0;

}



Hình Bài 4.7 Viết hàm thực hiện các phép giao và hợp của hai tập hợp biểu diễn bằng set

#include <iostream>

#include <set>

using namespace std;

template<class T>

set<T> set\_union(const set<T> &a, const set<T> &b) {

    // Hàm tính hợp của hai tập hợp a và b

    set<T> union\_set;  // Tập hợp kết quả

    for (T tmp : a) {

        union\_set.insert(tmp);  // Thêm các phần tử của tập hợp a vào tập hợp kết quả

    }

    for (T tmp : b) {

        union\_set.insert(tmp);  // Thêm các phần tử của tập hợp b vào tập hợp kết quả

    }

    return union\_set;  // Trả về tập hợp kết quả

}

//Ho va ten: Nguyen Viet Anh

//MSSV: 20215307

template<class T>

set<T> set\_intersection(const set<T> &a, const set<T> &b) {

    // Hàm tính giao của hai tập hợp a và b

    set<T> c;  // Tập hợp kết quả

    for (T v : a) {

        int index = 0;

        for (T u : b) {

            if (v == u) {

                index++;  // Nếu phần tử v có trong cả hai tập hợp, tăng biến đếm

                break;

            }

        }

        if (index != 0) c.insert(v);  // Nếu phần tử v có trong cả hai tập hợp, thêm vào tập hợp kết quả

    }

    return c;

}

template<class T>

void print\_set(const std::set<T> &a) {

    for (const T &x : a) {

        std::cout << x << ' ';

    }

    std::cout << std::endl;

}

int main() {

    cout << "Ho va ten: Nguyen Viet Anh\n";

    cout << "MSSV:20215307\n";

    std::set<int> a = {1, 2, 3, 5, 7};

    std::set<int> b = {2, 4, 5, 6, 9};

    std::set<int> c = set\_union(a, b);

    std::set<int> d = set\_intersection(a, b);

    std::cout << "Union: ";

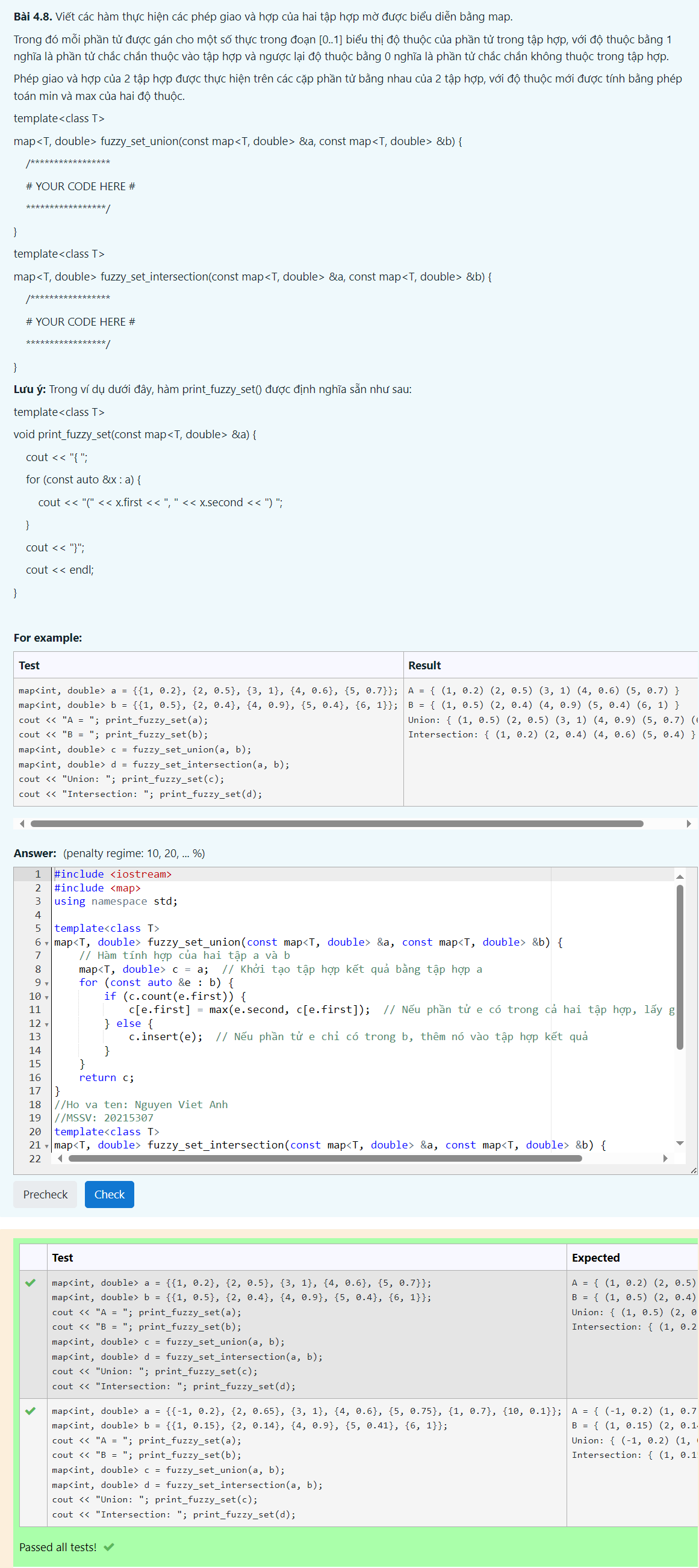
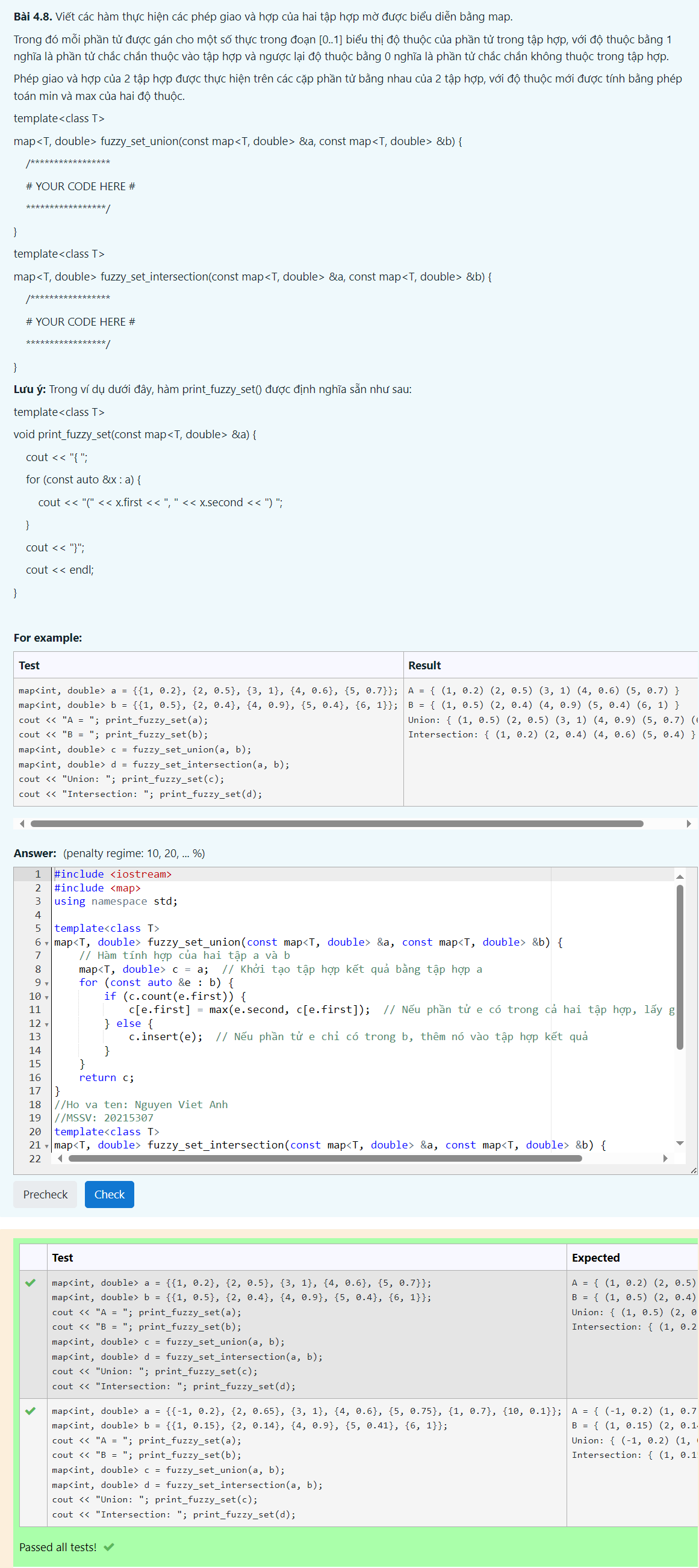
    print\_set(c);

    std::cout << "Intersection: ";

    print\_set(d);

    return 0;

}



Hình Bài 4.8 Viết hàm thực hiện các phép giao và hợp của hai tập hợp biểu diễn bằng map

#include <iostream>

#include <map>

using namespace std;

template<class T>

map<T, double> fuzzy\_set\_union(const map<T, double> &a, const map<T, double> &b) {

    // Hàm tính hợp của hai tập a và b

    map<T, double> c = a;  // Khởi tạo tập hợp kết quả bằng tập hợp a

    for (const auto &e : b) {

        if (c.count(e.first)) {

            c[e.first] = max(e.second, c[e.first]);  // Nếu phần tử e có trong cả hai tập hợp, lấy giá trị lớn hơn

        } else {

            c.insert(e);  // Nếu phần tử e chỉ có trong b, thêm nó vào tập hợp kết quả

        }

    }

    return c;

}

//Ho va ten: Nguyen Viet Anh

//MSSV: 20215307

template<class T>

map<T, double> fuzzy\_set\_intersection(const map<T, double> &a, const map<T, double> &b) {

    // Hàm tính giao của hai tập a và b

    map<T, double> c;  // Khởi tạo tập hợp kết quả

    for (const auto &x : a) {

        const auto it = b.find(x.first);  // Tìm phần tử x trong b

        if (it != b.end()) {

            c[x.first] = min(x.second, it->second);  // Nếu phần tử x có trong cả hai tập hợp, lấy giá trị nhỏ hơn

        }

    }

    return c;  // Trả về tập hợp kết quả

}

template<class T>

void print\_fuzzy\_set(const std::map<T, double> &a) {

    cout << "{ ";

    for (const auto &x : a) {

        std::cout << "(" << x.first << ", " << x.second << ") ";  // In từng phần tử của tập mờ a

    }

    cout << "}";

    std::cout << std::endl;

}

int main() {

    std::map<int, double> a = {{1, 0.2}, {2, 0.5}, {3, 1}, {4, 0.6}, {5, 0.7}};

    std::map<int, double> b = {{1, 0.5}, {2, 0.4}, {4, 0.9}, {5, 0.4}, {6, 1}};

    std::cout << "A = "; print\_fuzzy\_set(a);

    std::cout << "B = "; print\_fuzzy\_set(b);

    std::map<int, double> c = fuzzy\_set\_union(a, b);

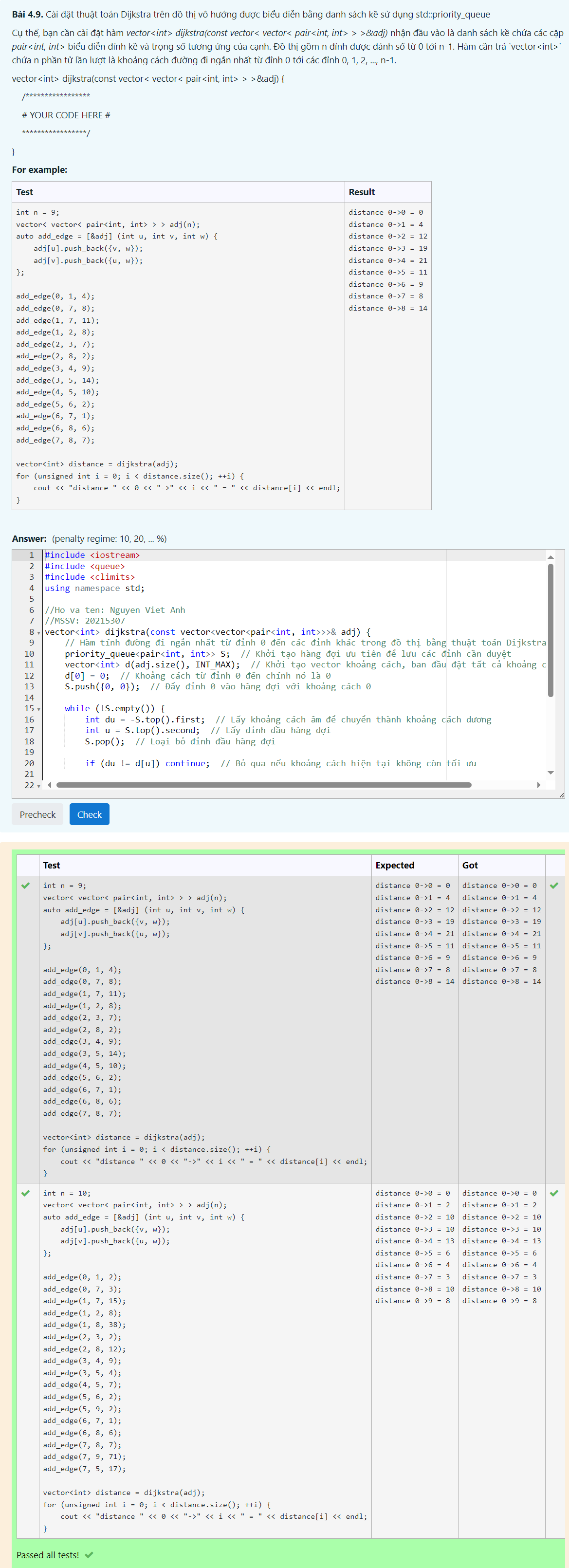
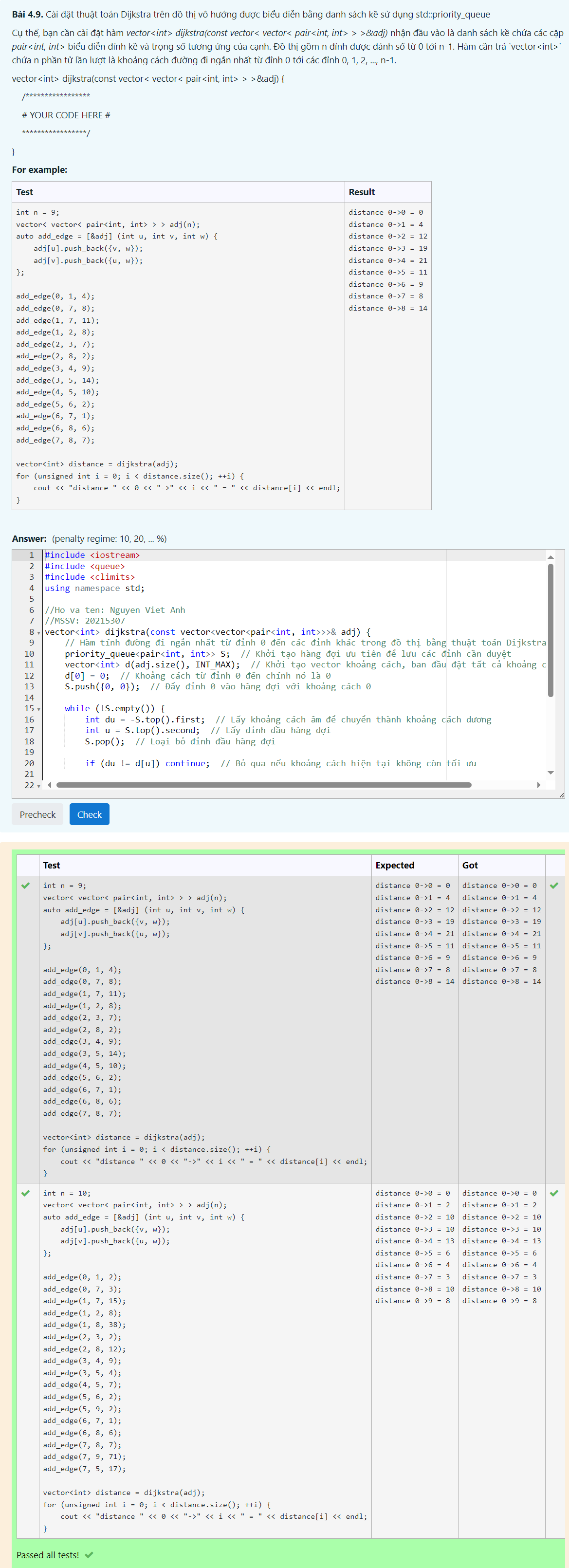
    std::map<int, double> d = fuzzy\_set\_intersection(a, b);

    std::cout << "Union: "; print\_fuzzy\_set(c);

    std::cout << "Intersection: "; print\_fuzzy\_set(d);

    return 0;

}



#include <iostream>

#include <queue>

#include <climits>

using namespace std;

//Ho va ten: Nguyen Viet Anh

//MSSV: 20215307

vector<int> dijkstra(const vector<vector<pair<int, int>>>& adj) {

    // Hàm tính đường đi ngắn nhất từ đỉnh 0 đến các đỉnh khác trong đồ thị bằng thuật toán Dijkstra

    priority\_queue<pair<int, int>> S;  // Khởi tạo hàng đợi ưu tiên để lưu các đỉnh cần duyệt

    vector<int> d(adj.size(), INT\_MAX);  // Khởi tạo vector khoảng cách, ban đầu đặt tất cả khoảng cách là vô cùng

    d[0] = 0;  // Khoảng cách từ đỉnh 0 đến chính nó là 0

    S.push({0, 0});  // Đẩy đỉnh 0 vào hàng đợi với khoảng cách 0

    while (!S.empty()) {

        int du = -S.top().first;  // Lấy khoảng cách âm để chuyển thành khoảng cách dương

        int u = S.top().second;  // Lấy đỉnh đầu hàng đợi

        S.pop();  // Loại bỏ đỉnh đầu hàng đợi

        if (du != d[u]) continue;  // Bỏ qua nếu khoảng cách hiện tại không còn tối ưu

        for (auto e : adj[u]) {  // Duyệt tất cả các đỉnh kề của u

            int v = e.first;  // Đỉnh kề

            int c = e.second;  // Trọng số cạnh

            if (d[v] > d[u] + c) {  // Nếu tìm được đường đi ngắn hơn

                d[v] = d[u] + c;  // Cập nhật khoảng cách

                S.push({-d[v], v});  // Đẩy đỉnh v vào hàng đợi với khoảng cách mới

            }

        }

    }

    return d;  // Trả về vector khoảng cách từ đỉnh 0 đến các đỉnh khác

}

# **BÀI TẬP VỀ NHÀ**

Chụp ảnh kết quả của tất cả các test.

**Bài tập 10: Search Engine**

**Xây dựng một máy tìm kiếm (search engine) đơn giản.**

**Cho NN văn bản và QQ truy vấn. Với mỗi truy vấn, cần trả về văn bản khớp với truy vấn đó nhất.**

Sử dụng phương pháp tính điểm TF-IDF:

• f(t,d)f(t,d) là số lần xuất hiện của từ tt trong văn bản dd

• maxf(d)maxf(d) là giá trị lớn nhất của f(t,d)f(t,d) với mọi tt

• df(t)df(t) là số văn bản chứa từ tt

• TF(t,d)=0.5+0.5⋅f(t,d)maxf(t,d)TF(t,d)=0.5+0.5⋅f(t,d)maxf(t,d)

• IDF(t)=log2(Ndf(t))IDF(t)=log2(Ndf(t))

• Điểm số của từ tt trong văn

bản dd là score(t,d)=TF(t,d)⋅IDF(t)score(t,d)=TF(t,d)⋅IDF(t), nếu từ tt không xuất hiện trong văn bản dd thì score(t,d)=0score(t,d)=0.

• Điểm số của văn bản dd đối với truy vấn gồm các từ (có thể trùng

nhau) t1,t2,...,tqt1,t2,...,tq là ∑qi=1score(ti,d)∑i=1qscore(ti,d)

Ta coi văn bản có điểm số càng cao thì càng khớp với truy vấn.

**Input:**

• Dòng đầu tiên chứa số NN

• Dòng thứ ii trong NN dòng tiếp theo thể hiện văn bản ii, mỗi dòng là một dãy các từ ngăn cách nhau bởi dấu phẩy

• Dòng tiếp theo chứa số QQ

• Dòng thứ ii trong QQ dòng tiếp theo thể hiện truy vấn thứ ii, mỗi dòng là một dãy các từ ngăn cách nhau bởi dấu phẩy

**Output:** Gồm QQ dòng, dòng thứ ii là chỉ số của văn bản khớp với truy vấn thứ ii nhất. Nếu có nhiều văn bản có điểm số bằng nhau, in ra văn bản có chỉ số nhỏ nhất.

**Ví dụ:**

Input:

5

k,k,ow

bb,ar,h

qs,qs,qs

d,bb,q,d,rj

ow

5

h,d,d,qs,q,q,ar

qs,qs

hc,d,ow,d,qs

ow,wl,hc,k

q,hc,q,d,hc,q

Output:

4

3

4

1

4

**Giới hạn:**

• N≤1000N≤1000

• Q≤1000Q≤1000

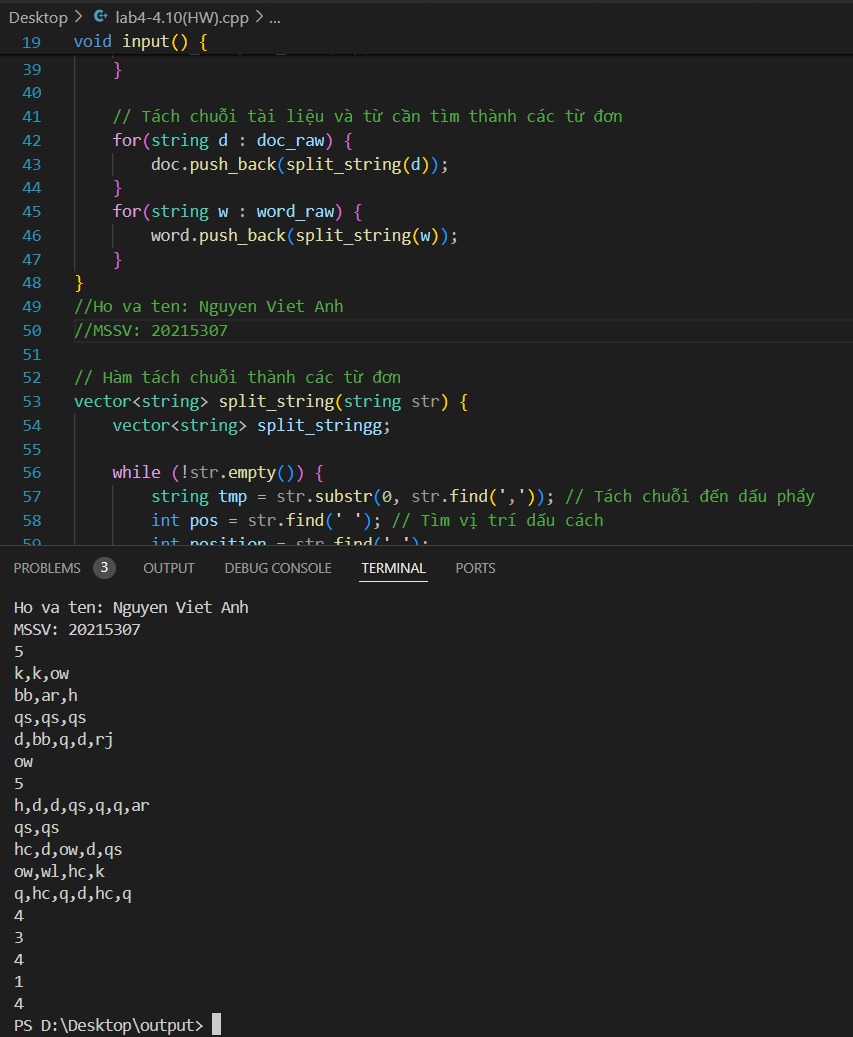
• Số từ trong mỗi văn bản không quá 10001000

• Số từ trong mỗi truy vấn không quá 1010

• Độ dài mỗi từ không quá 1010

**Tham khảo:**

• <https://en.wikipedia.org/wiki/Tf%E2%80%93idf>



#include<bits/stdc++.h>

using namespace std;

// Biến toàn cục

int n, q;

vector<vector<string>> doc; // Lưu trữ danh sách tài liệu

vector<vector<string>> word; // Lưu trữ danh sách từ cần tìm kiếm

vector<int> f\_max; // Lưu trữ tần số từ lớn nhất trong từng tài liệu

map<string, int> df; // Lưu trữ số tài liệu chứa từ khóa

map<pair<string, int>, int> fe; // Lưu trữ tần số xuất hiện của từ trong tài liệu

// Hàm tách chuỗi thành các từ

vector<string> split\_string(string str);

// Hàm xử lý tiền xử lý dữ liệu

void pre();

// Hàm nhập dữ liệu

void input() {

    vector<string> doc\_raw; // Lưu trữ tạm thời các tài liệu

    vector<string> word\_raw; // Lưu trữ tạm thời các từ cần tìm

    cin >> n; // Nhập số lượng tài liệu

    string tmp;

    getline(cin, tmp); // Đọc bỏ dòng trống

    for(int i = 0; i < n; i++) {

        string tmp;

        getline(cin, tmp); // Đọc từng tài liệu

        doc\_raw.push\_back(tmp);

    }

    cin >> q; // Nhập số lượng từ cần tìm

    getline(cin, tmp); // Đọc bỏ dòng trống

    for(int i = 0; i < q; i++) {

        string tmp;

        getline(cin, tmp); // Đọc từng từ cần tìm

        word\_raw.push\_back(tmp);

    }

    // Tách chuỗi tài liệu và từ cần tìm thành các từ đơn

    for(string d : doc\_raw) {

        doc.push\_back(split\_string(d));

    }

    for(string w : word\_raw) {

        word.push\_back(split\_string(w));

    }

}

//Ho va ten: Nguyen Viet Anh

//MSSV: 20215307

// Hàm tách chuỗi thành các từ đơn

vector<string> split\_string(string str) {

    vector<string> split\_stringg;

    while (!str.empty()) {

        string tmp = str.substr(0, str.find(',')); // Tách chuỗi đến dấu phẩy

        int pos = str.find(' '); // Tìm vị trí dấu cách

        int position = str.find(',');

        if(pos > tmp.size())

            split\_stringg.push\_back(tmp); // Thêm từ vào vector

        else {

            while(pos <= tmp.size()) {

                tmp.erase(pos, 1); // Xóa dấu cách

                pos = tmp.find(" ");

            }

        }

        if(position > str.size()) {

            break;

        } else {

            str.erase(0, position + 1); // Xóa phần đã tách khỏi chuỗi gốc

        }

    }

    return split\_stringg;

}

// Hàm tính tần số xuất hiện của từ trong tài liệu

int fre(string word\_s, int i) {

    if(fe.find({word\_s, i}) != fe.end()) {

        return fe[{word\_s, i}];

    }

    int index = 0;

    vector<string> str\_tmp = doc[i];

    for(string v : str\_tmp) {

        if(word\_s == v) index++;

    }

    fe.insert({{word\_s, i}, index});

    return index;

}

// Hàm đếm số lượng tài liệu chứa từ

int count(string word\_use) {

    if(df.find(word\_use) != df.end()) {

        return df[word\_use];

    }

    int index = 0;

    for(vector<string> str\_tmp : doc) {

        vector<string>::iterator ite = find(str\_tmp.begin(), str\_tmp.end(), word\_use);

        if(ite != str\_tmp.end()) {

            index++;

        }

    }

    df.insert({word\_use, index});

    return index;

}

// Hàm tiền xử lý dữ liệu

void pre() {

    for(vector<string> word\_str : doc) {

        map<string, int> m;

        int max\_f = 0;

        for(string word\_tmp : word\_str) {

            map<string, int>::iterator ite = m.find(word\_tmp);

            if(ite == m.end()) {

                m.insert({word\_tmp, 1});

            } else {

                ite->second += 1;

            }

            max\_f = max(m[word\_tmp], max\_f);

        }

        f\_max.push\_back(max\_f); // Thêm tần số lớn nhất vào vector

    }

}

// Hàm tìm kiếm tài liệu phù hợp nhất với danh sách từ cần tìm

int search\_engine(vector<string> list\_word) {

    double score\_max = -1000;

    int predict\_label = -1;

    for(int i = 0; i < n; i++) {

        vector<string> list\_word\_train\_doc = doc[i];

        double score = 0;

        for(string word\_tmp : list\_word) {

            if(find(list\_word\_train\_doc.begin(), list\_word\_train\_doc.end(), word\_tmp) == list\_word\_train\_doc.end()) { // từ này không xuất hiện trong văn bản

                continue;

            } else {

                int ftd = fre(word\_tmp, i);

                int dft = count(word\_tmp);

                int maxfd = f\_max[i];

                double tf\_word = 0.5 + 0.5 \* ((double) ftd / maxfd);

                double idf\_word = log2((double) n / dft);

                score += tf\_word \* idf\_word;

            }

        }

        if(score > score\_max) {

            predict\_label = i;

            score\_max = score;

        }

    }

    return predict\_label + 1;

}

int main() {

    cout << "Ho va ten: Nguyen Viet Anh\n";

    cout << "MSSV: 20215307\n";

    ios\_base::sync\_with\_stdio(false);

    cin.tie(NULL);

    input(); // Nhập dữ liệu

    pre(); // Tiền xử lý dữ liệu

    int i = 0;

    while(i < q) {

        cout << search\_engine(word[i]) << endl; // In kết quả tìm kiếm cho từng từ

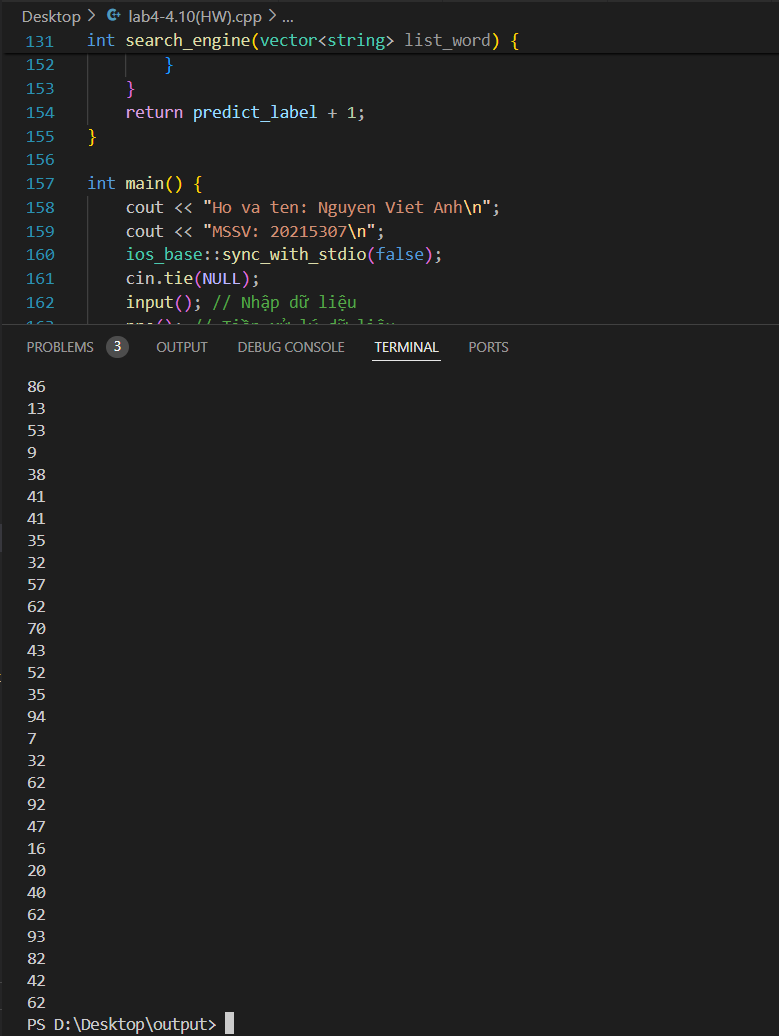
        i++;

    }

    return 0;

}

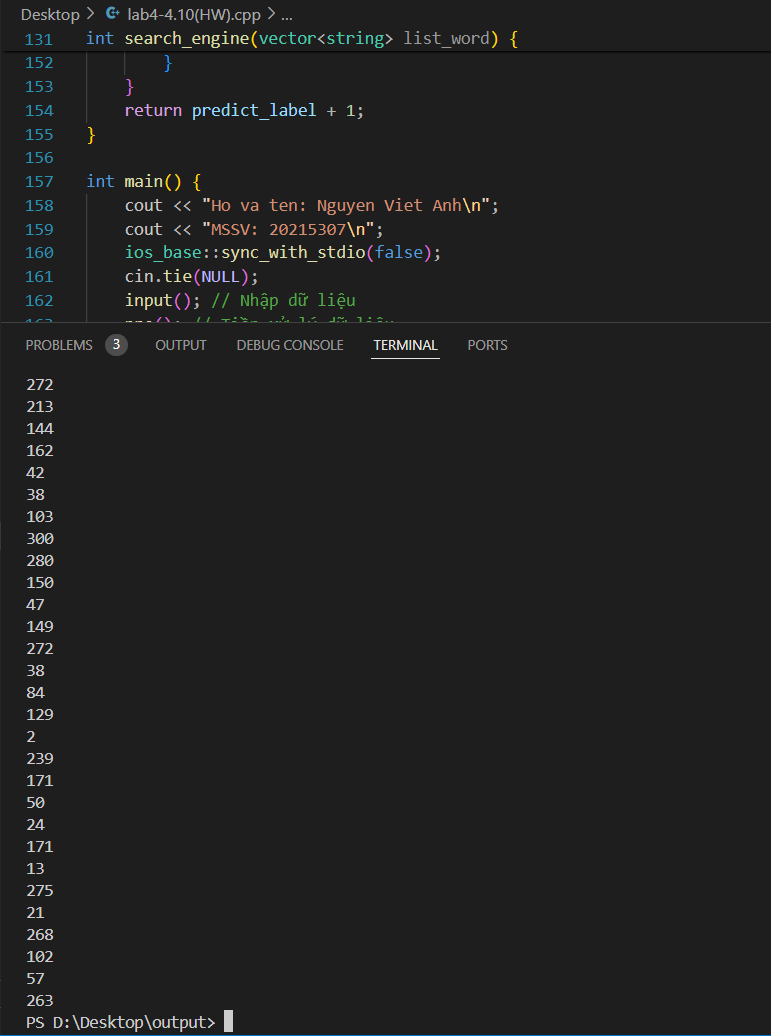
**Case 1**

****

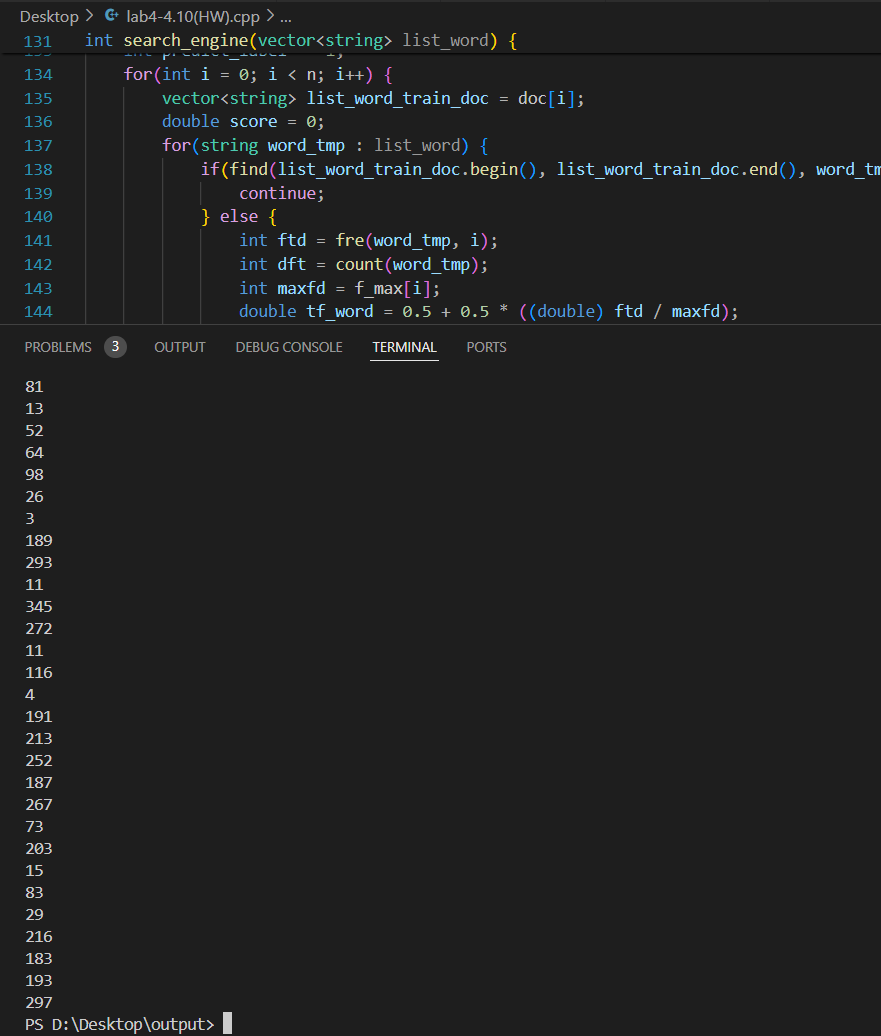
**Case 2**

****

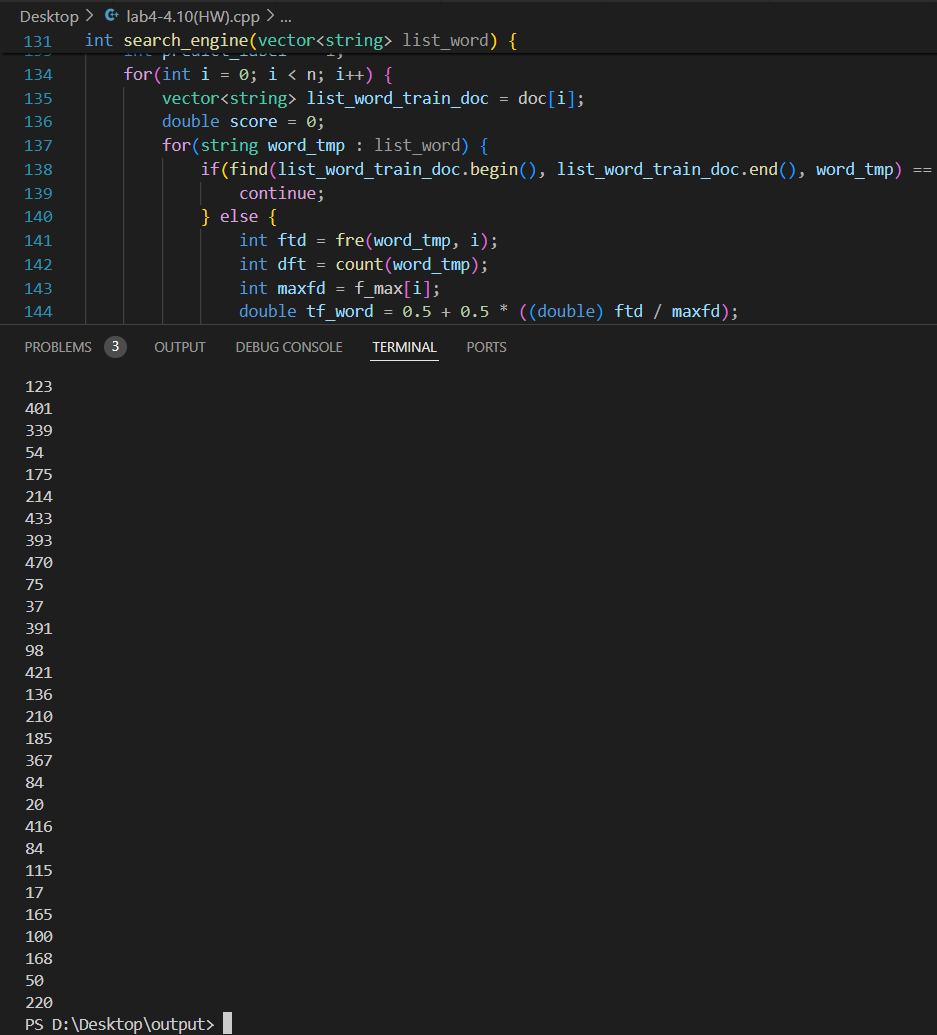
**Case 3**

****

**Case 4**

****

**Case 5**

****

**Bài tập 11**

Bức tường bao quanh một lâu đài nọ được cấu thành từ n đoạn tường được đánh số từ 1 đến n. Quân giặc lên kế hoạch tấn công lâu đài bằng cách gửi aitên giặc đánh vào đoạn tường thứ i. Để bảo vệ lâu đài có tất cả s lính.

Do các đoạn tường có chất lượng khác nhau nên khả năng bảo vệ tại các đoạn tường cũng khác nhau. Cụ thể tại đoạn tường thứ i, mỗi lính có thể đẩy lùi tấn công của ki tên giặc. Giả sử đoạn tường thứ i có xi lính. Khi đó nếu số tên giặc không vượt quá xi thì không có tên giặc nào lọt vào được qua đoạn tường này. Ngược lại sẽ có ai - xi \* ki tên giặc lọt vào lâu đài qua đoạn tường này.

Yêu cầu hãy viết chương trình phân bố lính đứng ở các đoạn tường sao cho tổng số lính là s và tổng số lượng tên giặc lọt vào lâu đài là nhỏ nhất.

**Dữ liệu vào:**

Dòng thứ nhất chứa các số nguyên n và s ( 1≤ n ≤ 100000;1 ≤ s ≤ 109).

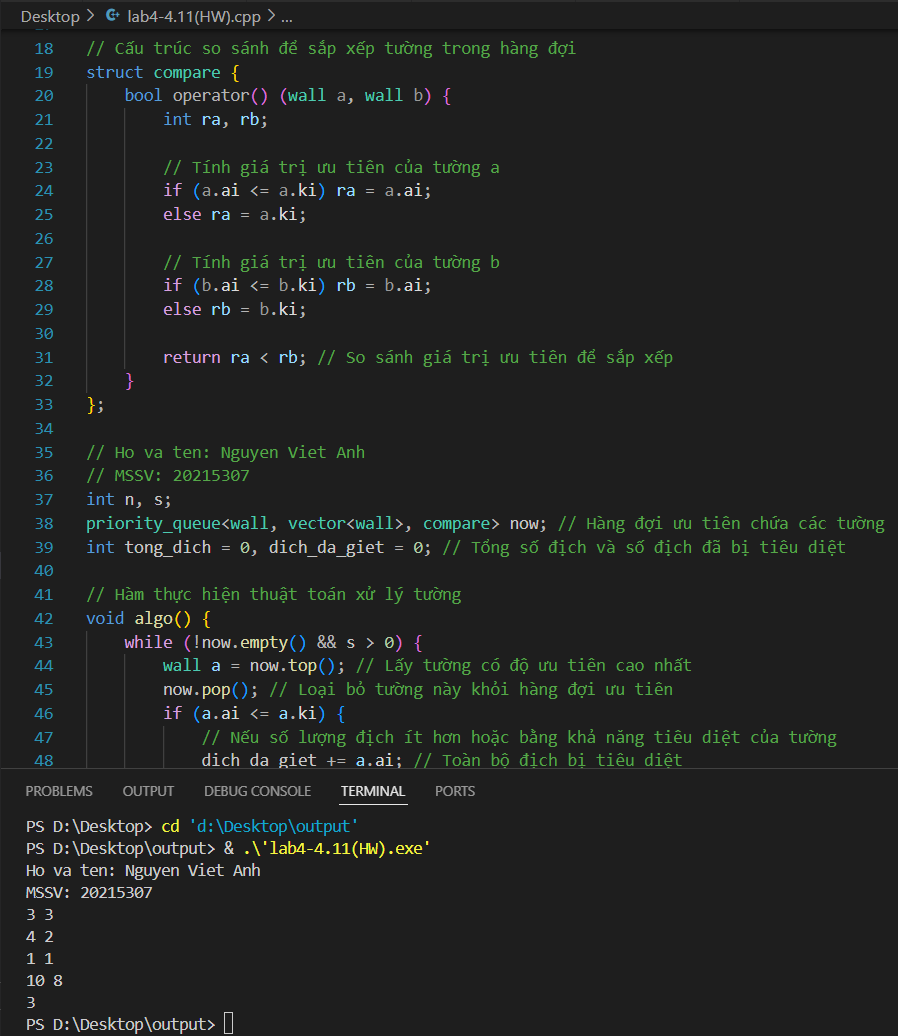
n dòng tiếp theo chứa hai số nguyên ai và ki lần lượt là số tên giặc tấn công đoạn tường thứ i và khả năng chống trả của một lính ở đoạn tường thứ i ( 1≤ ai, ki ≤ 109).

**Kết quả:**

Ghi ra một số nguyên duy nhất là số lượng tên giặc tối thiểu có thể lọt vào lâu đài.

**Ví dụ:**

| **Dữ liệu vào** | **Kết quả** |
| --- | --- |
| 3 3 4 2 1 1 10 8 | 3 |
| 1 10 8 1 | 0 |



#include <iostream>

#include <queue>

#include <vector>

#include <map>

using namespace std;

// Định nghĩa cấu trúc lưu trữ thông tin của một wall

struct wall {

    int ai; // Số lượng địch

    int ki; // Khả năng tiêu diệt của tường

    wall(int ai, int ki) {

        this->ai = ai;

        this->ki = ki;

    }

};

// Cấu trúc so sánh để sắp xếp tường trong hàng đợi

struct compare {

    bool operator() (wall a, wall b) {

        int ra, rb;

        // Tính giá trị ưu tiên của tường a

        if (a.ai <= a.ki) ra = a.ai;

        else ra = a.ki;

        // Tính giá trị ưu tiên của tường b

        if (b.ai <= b.ki) rb = b.ai;

        else rb = b.ki;

        return ra < rb; // So sánh giá trị ưu tiên để sắp xếp

    }

};

// Ho va ten: Nguyen Viet Anh

// MSSV: 20215307

int n, s;

priority\_queue<wall, vector<wall>, compare> now; // Hàng đợi ưu tiên chứa các tường

int tong\_dich = 0, dich\_da\_giet = 0; // Tổng số địch và số địch đã bị tiêu diệt

// Hàm thực hiện thuật toán xử lý tường

void algo() {

    while (!now.empty() && s > 0) {

        wall a = now.top(); // Lấy tường có độ ưu tiên cao nhất

        now.pop(); // Loại bỏ tường này khỏi hàng đợi ưu tiên

        if (a.ai <= a.ki) {

            // Nếu số lượng địch ít hơn hoặc bằng khả năng tiêu diệt của tường

            dich\_da\_giet += a.ai; // Toàn bộ địch bị tiêu diệt

        } else {

            // Nếu số lượng địch lớn hơn khả năng tiêu diệt của tường

            int now\_enemy = a.ai - a.ki; // Số địch còn lại sau khi tiêu diệt

            now.push(wall(now\_enemy, a.ki)); // Đẩy lại số địch còn lại vào hàng đợi với tường có cùng khả năng tiêu diệt

            dich\_da\_giet += a.ki; // Cập nhật số địch bị tiêu diệt

        }

        s -= 1; // Giảm số lượng lượt tấn công còn lại

    }

}

// Hàm nhập dữ liệu

void input() {

    cin >> n >> s; // Nhập số lượng tường và số lượt tấn công

    int i = 0;

    while (i < n) {

        int ai, ki;

        cin >> ai >> ki; // Nhập số địch và khả năng tiêu diệt của từng tường

        now.push(wall(ai, ki)); // Đẩy tường vào hàng đợi ưu tiên

        tong\_dich += ai; // Cập nhật tổng số địch

        i++;

    }

}

int main() {

    cout << "Ho va ten: Nguyen Viet Anh\n";

    cout << "MSSV: 20215307\n";

    ios\_base::sync\_with\_stdio(false);

    cin.tie(NULL);

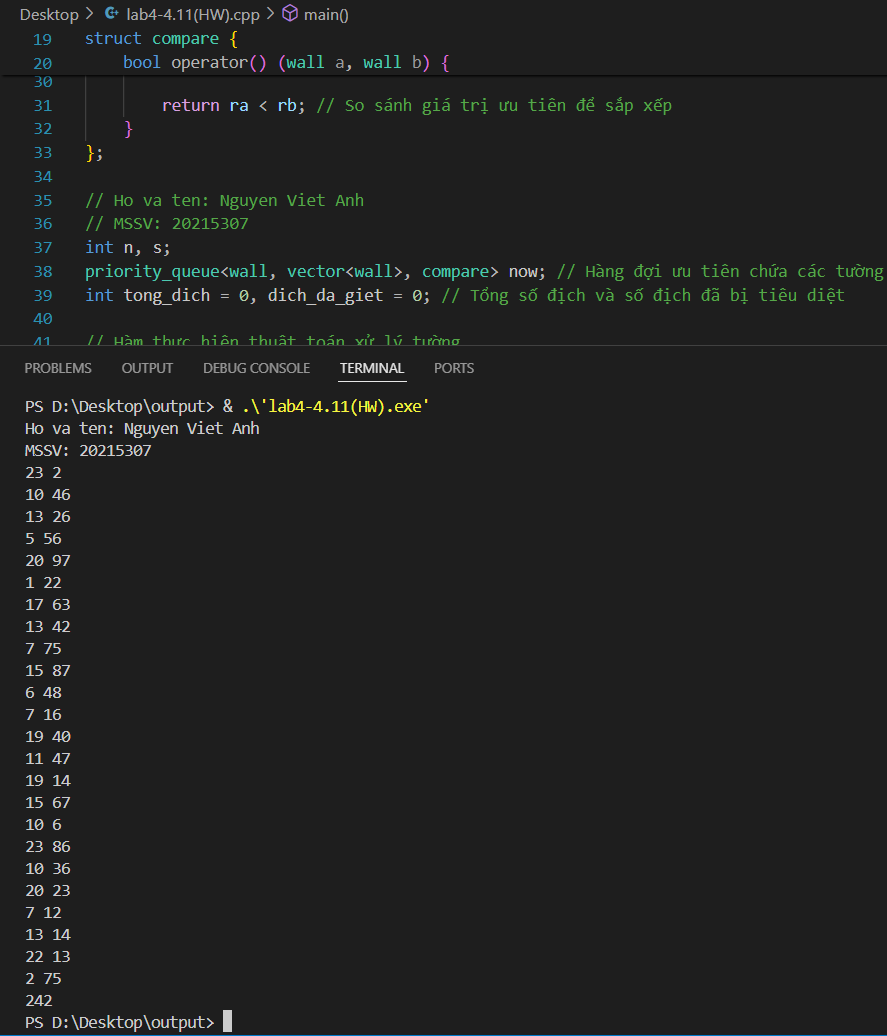
    input(); // Nhập dữ liệu

    algo(); // Thực hiện thuật toán

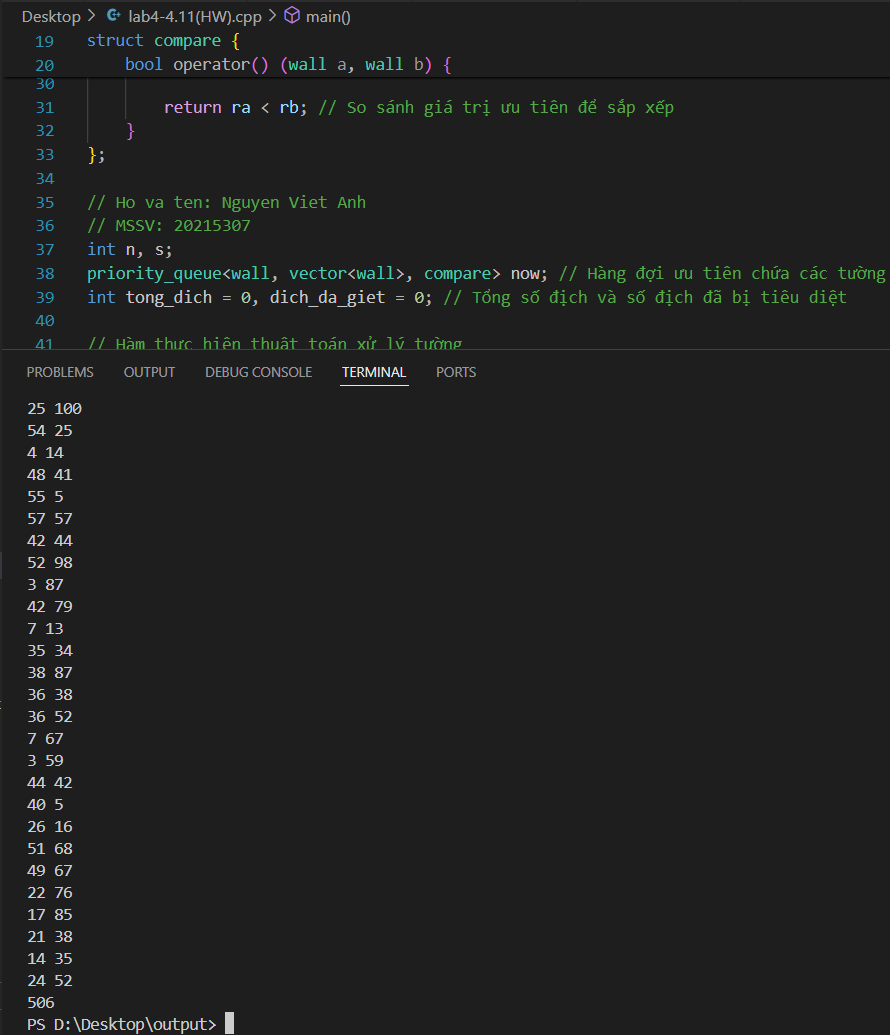
    cout << tong\_dich - dich\_da\_giet; // In ra số địch còn lại chưa bị tiêu diệt

}

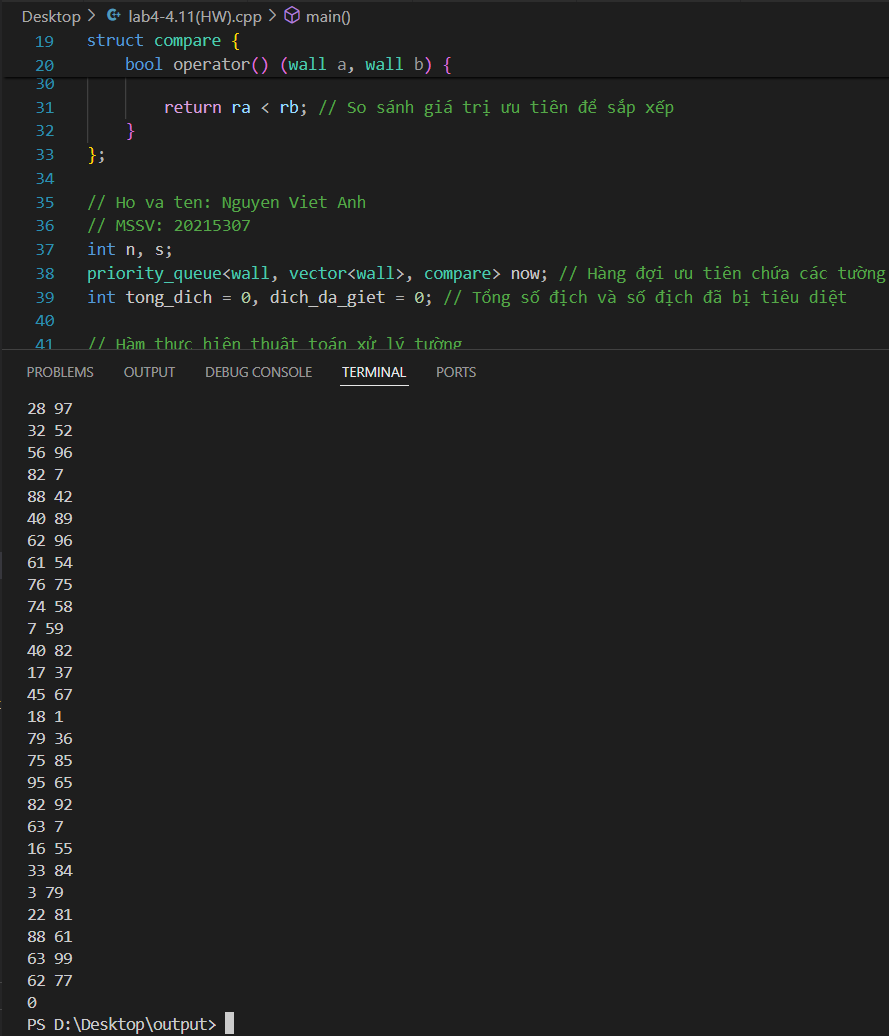
**Case 1**

****

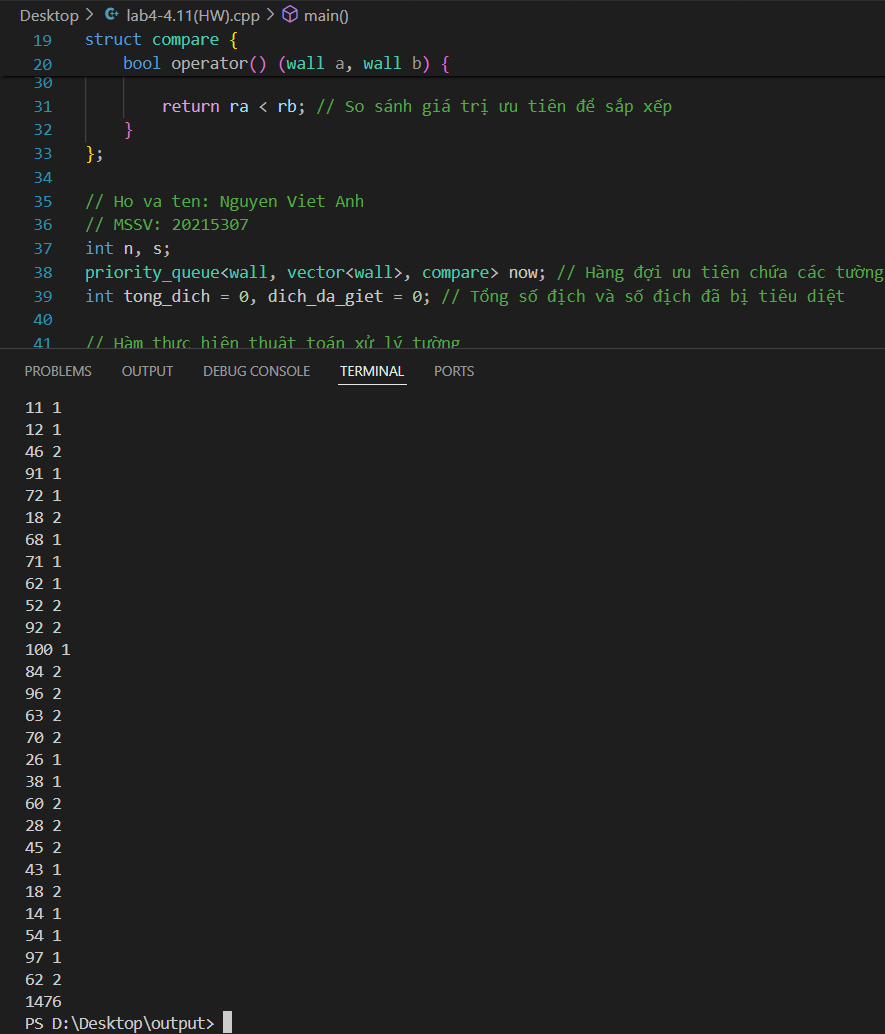
**Case 2**

****

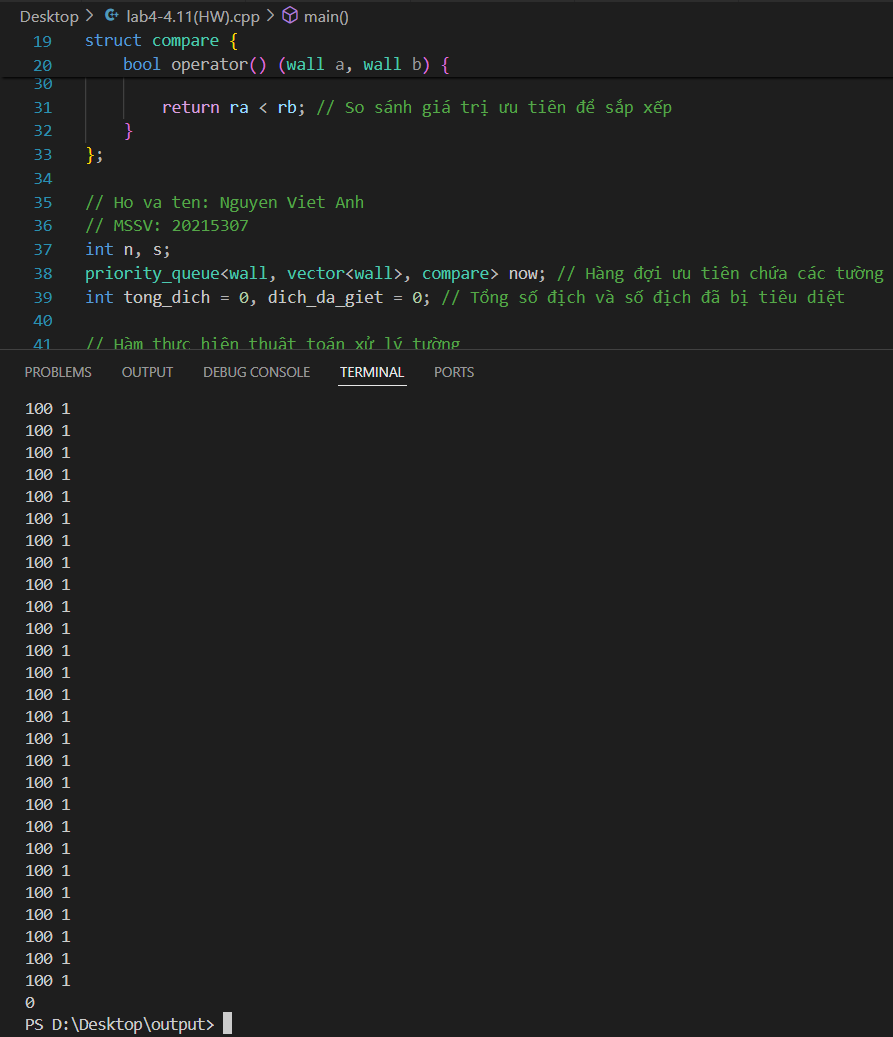
**Case 3**

****

**Case 4**

****

**Case 5**

****

**Bài tập 12**

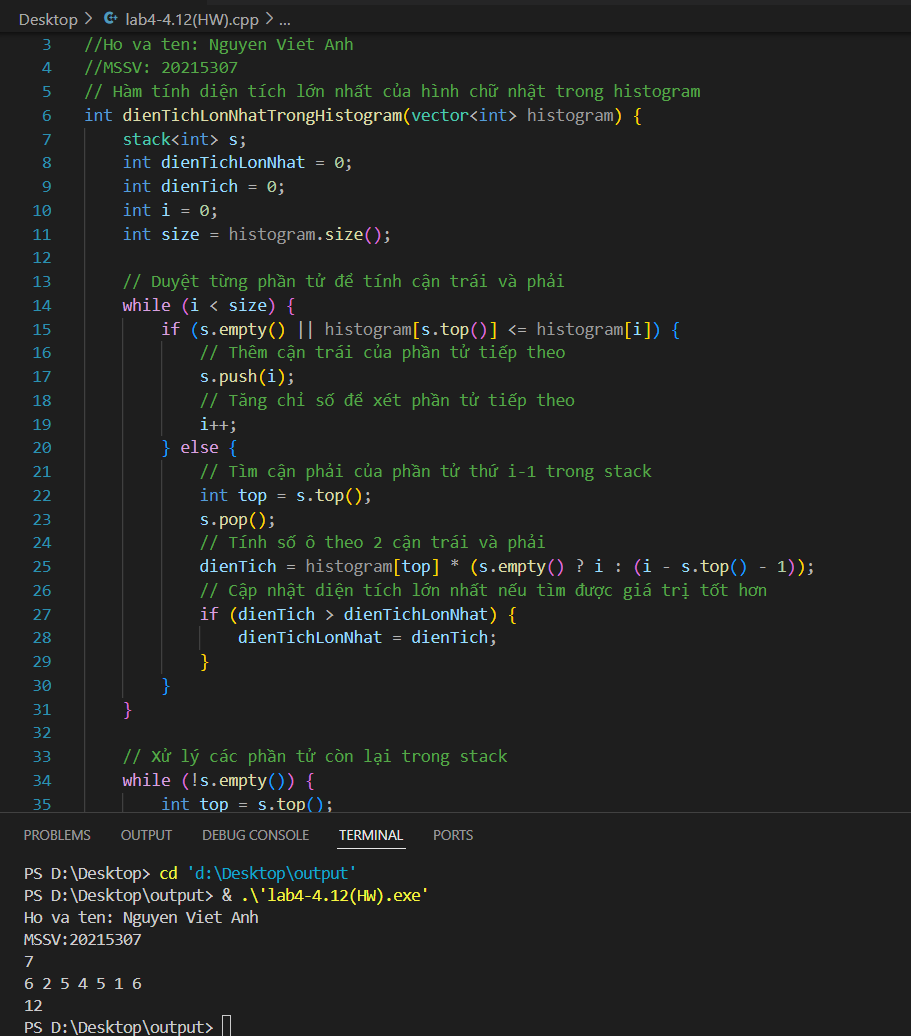
Cho một lược đồ gồm n cột chữ nhật liên tiếp nhau có chiều rộng bằng 1 và chiều cao lần lượt là các số nguyên không âm h1,h2,…,hn . Hãy xác định hình chữ nhật có diện tích lớn nhất có thể tạo thành từ các cột liên tiếp.

**Dữ liệu vào:**Dòng thứ nhất chứa số nguyên dương n (1 ≤ n ≤ 106). Dòng thứ hai chứa n số nguyên không âm h1,h2,…,hn cách nhau bởi dấu cách (0 ≤ hi≤ 109).

**Kết quả:** In ra số nguyên duy nhất là diện tích hình chữ nhật lớn nhất có thể tạo thành từ các cột liên tiếp của lược đồ.

**Ví dụ:**

| Dữ liệu vào | Kết quả |
| --- | --- |
| 7 6 2 5 4 5 1 6 | 12 |

****

#include <bits/stdc++.h>

using namespace std;

//Ho va ten: Nguyen Viet Anh

//MSSV: 20215307

// Hàm tính diện tích lớn nhất của hình chữ nhật trong histogram

int dienTichLonNhatTrongHistogram(vector<int> histogram) {

    stack<int> s;

    int dienTichLonNhat = 0;

    int dienTich = 0;

    int i = 0;

    int size = histogram.size();

    // Duyệt từng phần tử để tính cận trái và phải

    while (i < size) {

        if (s.empty() || histogram[s.top()] <= histogram[i]) {

            // Thêm cận trái của phần tử tiếp theo

            s.push(i);

            // Tăng chỉ số để xét phần tử tiếp theo

            i++;

        } else {

            // Tìm cận phải của phần tử thứ i-1 trong stack

            int top = s.top();

            s.pop();

            // Tính số ô theo 2 cận trái và phải

            dienTich = histogram[top] \* (s.empty() ? i : (i - s.top() - 1));

            // Cập nhật diện tích lớn nhất nếu tìm được giá trị tốt hơn

            if (dienTich > dienTichLonNhat) {

                dienTichLonNhat = dienTich;

            }

        }

    }

    // Xử lý các phần tử còn lại trong stack

    while (!s.empty()) {

        int top = s.top();

        s.pop();

        dienTich = histogram[top] \* (s.empty() ? i : (i - s.top() - 1));

        if (dienTich > dienTichLonNhat) {

            dienTichLonNhat = dienTich;

        }

    }

    // Trả về diện tích lớn nhất

    return dienTichLonNhat;

}

//Ho va ten: Nguyen Viet Anh

//MSSV: 20215307

int main() {

    cout << "Ho va ten: Nguyen Viet Anh\n";

    cout << "MSSV:20215307\n";

    int n, in;

    cin >> n;

    vector<int> chieuCao;

    // Nhập danh sách chiều cao của các cột trong histogram

    for (int i = 0; i < n; ++i) {

        cin >> in;

        chieuCao.push\_back(in);

    }

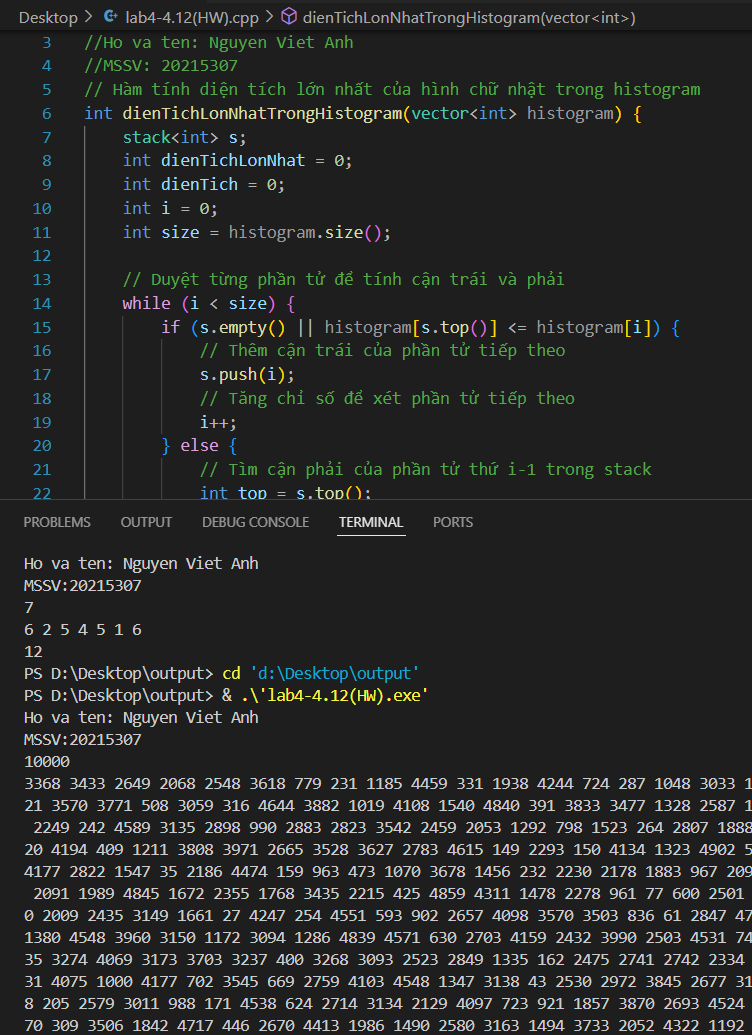
    // In ra diện tích lớn nhất của hình chữ nhật trong histogram

    cout << dienTichLonNhatTrongHistogram(chieuCao) << endl;

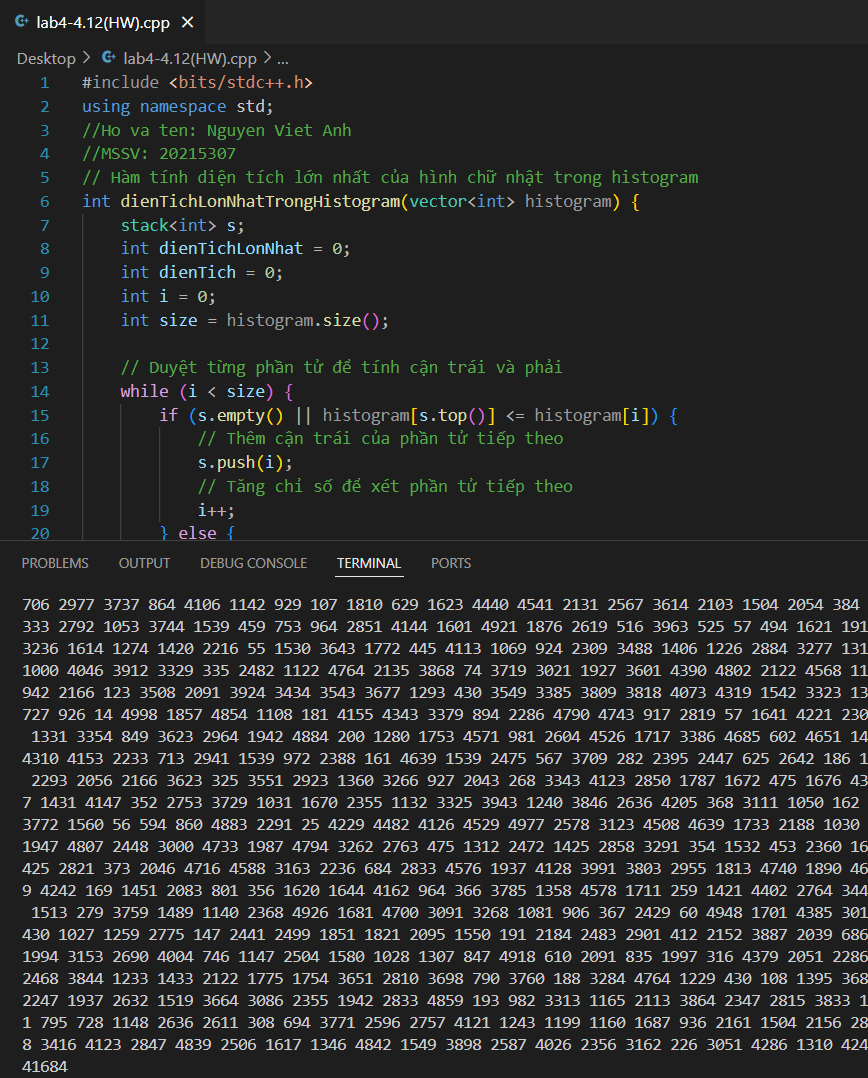
    return 0;

}

**Case 1**

****

**Case 2**

****

**Bài 13**

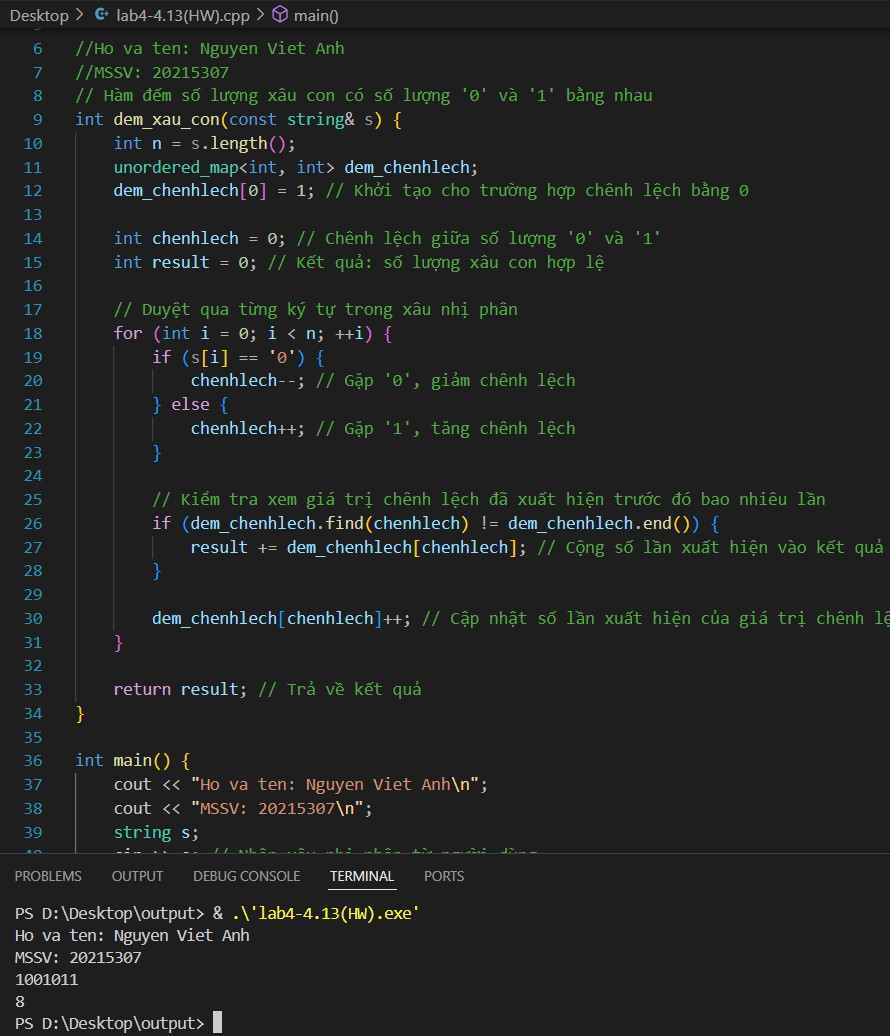
Cho một xâu nhị phân độ dài n. Hãy viết chương trình đếm số lượng xâu con chứa số ký tự 0 và số ký tự 1 bằng nhau.

**Dữ liệu vào:**Một dòng duy nhất chứa một xâu nhị phân độ dài n ( 1 ≤ n ≤ 106 ).

**Kết quả:**Ghi ra một số nguyên duy nhất là số lượng xâu con có số ký tự 0 và số ký tự 1 bằng nhau.

**Ví dụ:**

| Dữ liệu vào | Kết quả |
| --- | --- |
| 1001011 | 8 |



#include <iostream>

#include <unordered\_map>

#include <vector>

using namespace std;

//Ho va ten: Nguyen Viet Anh

//MSSV: 20215307

// Hàm đếm số lượng xâu con có số lượng '0' và '1' bằng nhau

int dem\_xau\_con(const string& s) {

    int n = s.length();

    unordered\_map<int, int> dem\_chenhlech;

    dem\_chenhlech[0] = 1; // Khởi tạo cho trường hợp chênh lệch bằng 0

    int chenhlech = 0; // Chênh lệch giữa số lượng '0' và '1'

    int result = 0; // Kết quả: số lượng xâu con hợp lệ

    // Duyệt qua từng ký tự trong xâu nhị phân

    for (int i = 0; i < n; ++i) {

        if (s[i] == '0') {

            chenhlech--; // Gặp '0', giảm chênh lệch

        } else {

            chenhlech++; // Gặp '1', tăng chênh lệch

        }

        // Kiểm tra xem giá trị chênh lệch đã xuất hiện trước đó bao nhiêu lần

        if (dem\_chenhlech.find(chenhlech) != dem\_chenhlech.end()) {

            result += dem\_chenhlech[chenhlech]; // Cộng số lần xuất hiện vào kết quả

        }

        dem\_chenhlech[chenhlech]++; // Cập nhật số lần xuất hiện của giá trị chênh lệch hiện tại

    }

    return result; // Trả về kết quả

}

int main() {

    cout << "Ho va ten: Nguyen Viet Anh\n";

    cout << "MSSV: 20215307\n";

    string s;

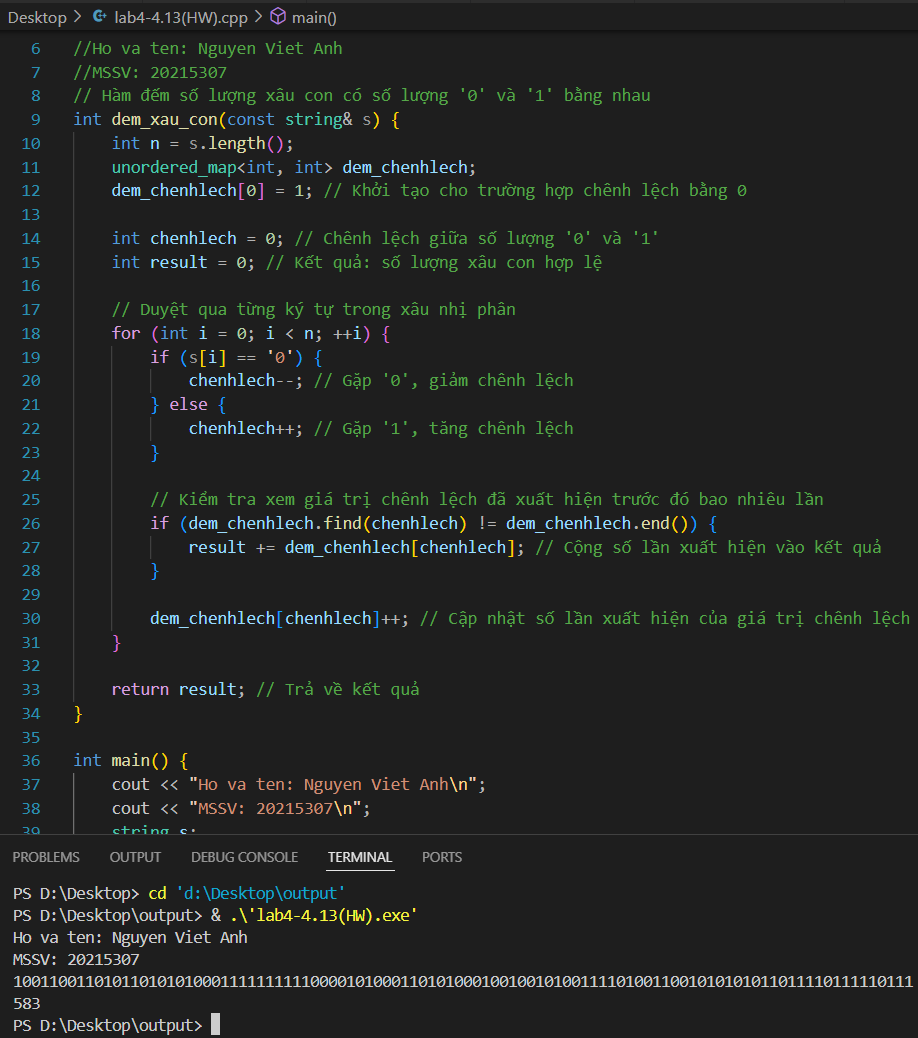
    cin >> s; // Nhập xâu nhị phân từ người dùng

    cout << dem\_xau\_con(s) << endl; // In ra số lượng xâu con hợp lệ

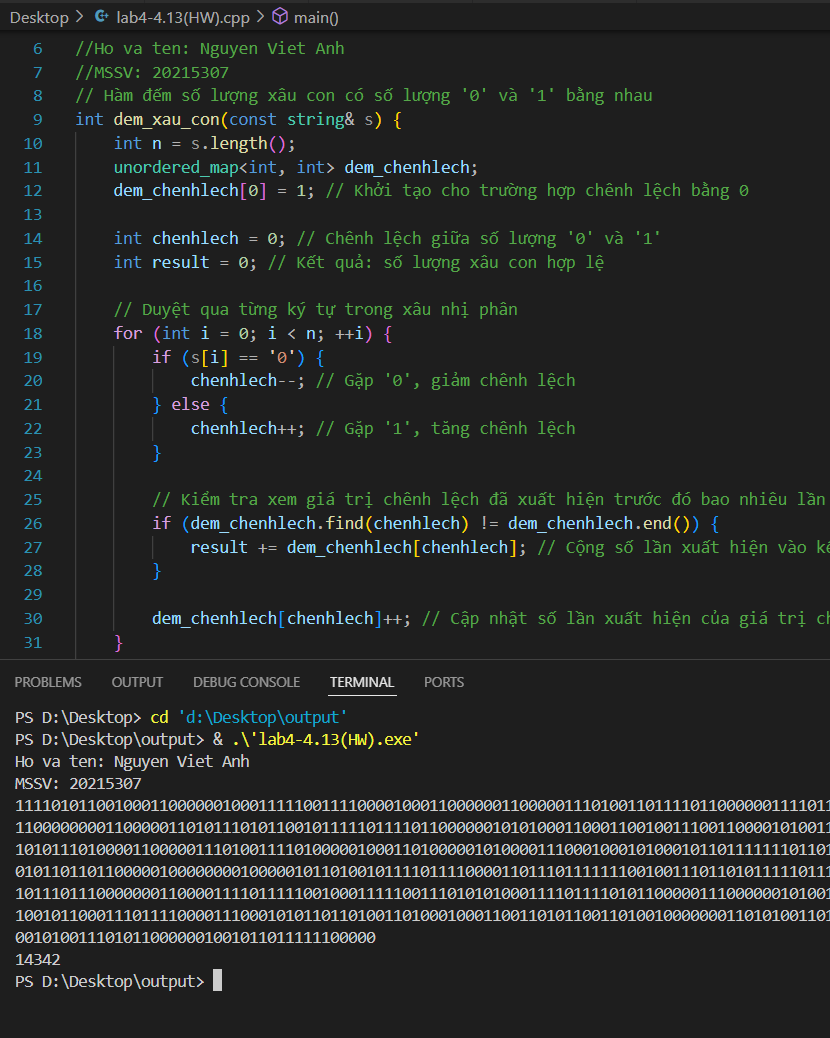
    return 0;

}

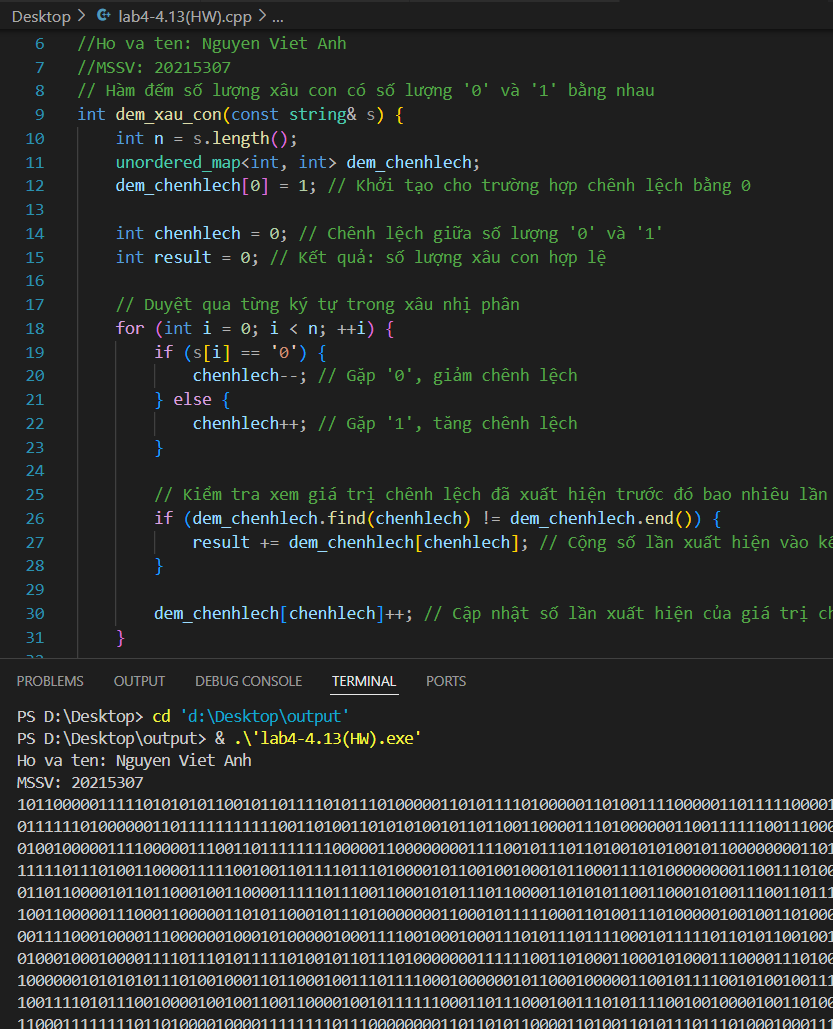
**Case 1**

****

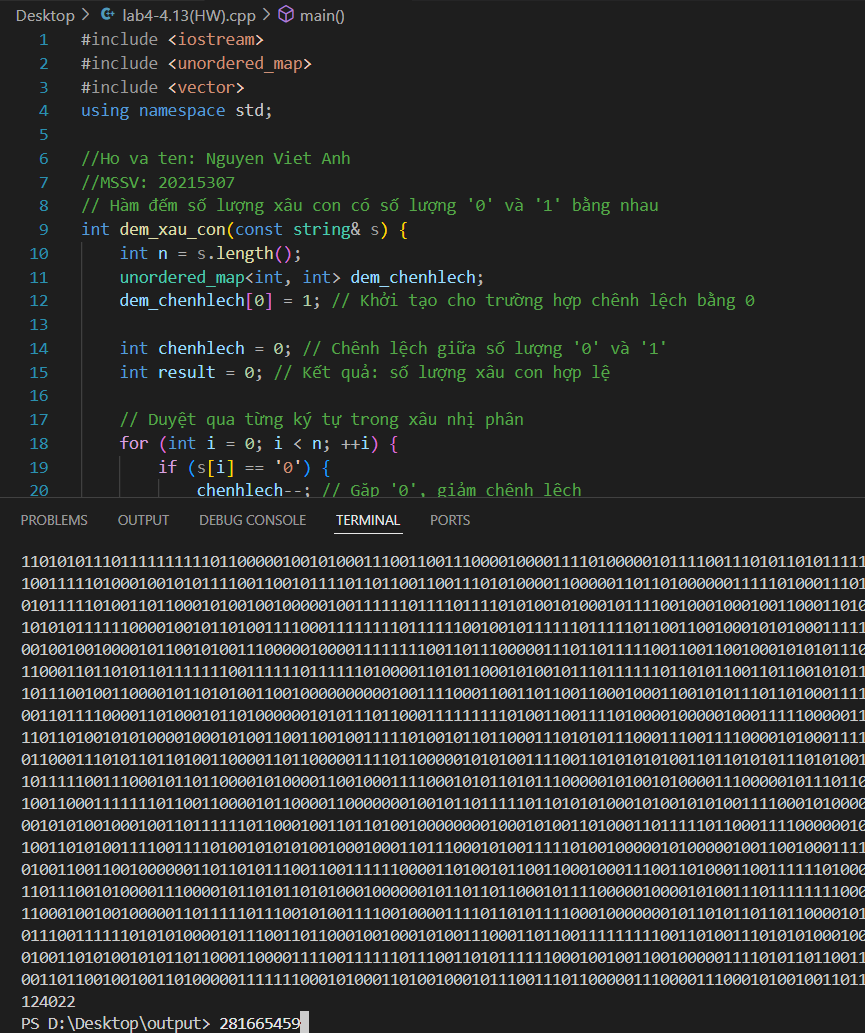
**Case 2**

****

**Case 3**

****

**Case 4**

****