MỤC LỤC

[BÀI THỰC HÀNH SỐ 4 – TUẦN 15 2](#_Toc185199367)

[BÀI TẬP TRÊN LAB 2](#_Toc185199368)

[Bài 4.1: Đảo ngược một danh sách liên kết đơn. 2](#_Toc185199369)

[Bài 4.2: Một điểm trong không gian 2 chiều được biểu diễn bằng pair. Hãy viết hàm tính diện tích tam giác theo tọa độ 3 đỉnh. 6](#_Toc185199370)

[Bài 4.3: Một vector trong không gian 3 chiều được biểu diễn bằng tuple<double, double, double>. Hãy viết hàm tính tích có hướng của 2 vector: 8](#_Toc185199371)

[Bài 4.4: Cho hai std::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. 10](#_Toc185199372)

[Bài 4.5: Viết hàm void dfs(vector< list<int> > adj) thực hiện thuật toán DFS không sử dụng đệ quy trên đồ thị biểu diễn bằng danh sách kề. 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. 14](#_Toc185199373)

[Bài 4.6: Viết hàm void bfs(vector< list<int> > adj) thực hiện thuật toán BFS không sử dụng đệ quy trên đồ thị biểu diễn bằng danh sách kề. 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. 17](#_Toc185199374)

[Bài 4.7: Viết các hàm thực hiện các phép giao và hợp của hai tập hợp được biểu diễn bằng set. 20](#_Toc185199375)

[Bài 4.8: Viết các hàm thực hiện các phép giao và hợp của hai tập hợp mờ được biểu diễn bằng map. 23](#_Toc185199376)

[Bài 4.9: Cài đặt thuật toán Dijkstra trên đồ thị vô hướng được biểu diễn bằng danh sách kề sử dụng std::priority\_queue. 27](#_Toc185199377)

[BÀI TẬP VỀ NHÀ 30](#_Toc185199378)

[Bài 4.10: Xây dựng một máy tìm kiếm (search engine) đơn giản. Xây dựng một máy tìm kiếm (search engine) đơn giản. 30](#_Toc185199379)

[Bài 4.11: 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 ai tên giặc đánh vào đoạn tường thứ i. Để bảo vệ lâu đài có tất cả s lính. 45](#_Toc185199380)

[Bài 4.12: 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. 53](#_Toc185199381)

[Bài 4.13: 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. 61](#_Toc185199382)

# BÀI THỰC HÀNH SỐ 4 – TUẦN 15

## BÀI TẬP TRÊN LAB

### Bài 4.1: 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))

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

#include <iostream>

struct Node {

int data;

Node\* next;

Node(int data) {

this->data = data;

next = NULL;

}

};

// Thêm một phần tử mới vào đầu danh sách

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

Node\* temp = new Node(data);

temp->next = head;

return temp;

}

// In nội dung danh sách liên kết

void print(Node\* head) {

while (head) {

std::cout << head->data << " ";

head = head->next; // Di chuyển đến phần tử tiếp theo

}

std::cout << std::endl;

}

// Đảo ngược danh sách liên kết và trả về head mới

Node\* reverse(Node\* head) {

Node\* prev = NULL;

Node\* next = NULL;

while (head) {

next = head->next; // Lưu lại nút tiếp theo

head->next = prev; // Đảo chiều liên kết

prev = head; // Di chuyển prev lên head hiện tại

head = next; // Di chuyển head đến nút tiếp theo

}

return prev; // Trả về head mới của danh sách

}

int main() {

int n, u;

std::cin >> n;

Node\* head = NULL;

for (int i = 0; i < n; ++i) {

std::cin >> u;

head = prepend(head, u); // Gán lại head sau khi thêm phần tử

}

std::cout << "Original list: ";

print(head);

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

std::cout << "Reversed list: ";

print(head);

return 0;

}

### Bài 4.2: Một điểm trong không gian 2 chiều được biểu diễn bằng pair. Hãy viết hàm tính diện tích tam giác theo tọa độ 3 đỉnh.

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

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# YOUR CODE HERE #

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

}

trong đó, Point là kiểu được định nghĩa trước trong trình chấm như sau:

using Point = pair<double, double>;

A screenshot of a computer

Description automatically generated

#include <iostream>

#include <cmath>

#include <iomanip>

#include <utility>

using Point = std::pair<double, double>;

// Hàm tính diện tích tam giác từ 3 điểm

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

return fabs(

a.first \* (b.second - c.second) +

b.first \* (c.second - a.second) +

c.first \* (a.second - b.second)

) / 2.0;

}

int main() {

std::cout << std::setprecision(2) << std::fixed;

// Tính diện tích tam giác từ 3 điểm

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

return 0;

}

### Bài 4.3: Một vector trong không gian 3 chiều được biểu diễn bằng tuple<double, double, double>. Hãy viết hàm tính tích có hướng của 2 vector:

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

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# YOUR CODE HERE #

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

}

trong đó Vector là kiểu được định nghĩa sẵn trong trình chấm như sau:

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

A screenshot of a computer

Description automatically generated

#include <iostream>

#include <cmath>

#include <iomanip>

#include <tuple> // Để sử dụng std::tuple

using namespace std;

// Định nghĩa kiểu Vector

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

// Hàm tính tích có hướng của hai vector

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

return Vector(

std::get<1>(a) \* std::get<2>(b) - std::get<2>(a) \* std::get<1>(b), // c\_x

std::get<2>(a) \* std::get<0>(b) - std::get<0>(a) \* std::get<2>(b), // c\_y

std::get<0>(a) \* std::get<1>(b) - std::get<1>(a) \* std::get<0>(b) // c\_z

);

}

int main() {

// Định dạng số thập phân

cout << setprecision(2) << fixed;

// Hai vector cần tính

Vector a {1.2, 4, -0.5};

Vector b {1.5, -2, 2.5};

// Tính tích có hướng

Vector c = cross\_product(a, b);

// In kết quả

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

return 0;}

### Bài 4.4: Cho hai std::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.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

#include <iostream>

#include <vector>

#include <algorithm>

// In các phần tử của vector

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

for (int v : a)

std::cout << v << ' ';

std::cout << std::endl;

}

// Xóa các phần tử chẵn trong vector

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

a.erase(std::remove\_if(a.begin(), a.end(), [] (int x) { return x % 2 == 0; }), a.end());

}

// Sắp xếp vector theo thứ tự giảm dần

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

std::sort(a.rbegin(), a.rend());

}

// Trộn 2 vector và trả về vector mới, sắp xếp giảm dần

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

std::vector<int> c;

std::merge(a.begin(), a.end(), b.begin(), b.end(), std::back\_inserter(c), std::greater<int>());

return c;

}

int main() {

int m, n, u;

std::vector<int> a, b;

// Nhập số phần tử của vector a và b

std::cin >> m >> n;

// Nhập các phần tử của vector a

for (int i = 0; i < m; i++) {

std::cin >> u;

a.push\_back(u);

}

// Nhập các phần tử của vector b

for (int i = 0; i < n; i++) {

std::cin >> u;

b.push\_back(u);

}

// Xóa phần tử chẵn trong a và b

delete\_even(a);

std::cout << "Odd elements of a: ";

print\_vector(a);

delete\_even(b);

std::cout << "Odd elements of b: ";

print\_vector(b);

// Sắp xếp giảm dần vector a và b

sort\_decrease(a);

std::cout << "Decreasingly sorted a: ";

print\_vector(a);

sort\_decrease(b);

std::cout << "Decreasingly sorted b: ";

print\_vector(b);

// Trộn a và b thành c

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

std::cout << "Decreasingly sorted c: ";

print\_vector(c);

return 0;

}

### Bài 4.5: Viết hàm void dfs(vector< list<int> > adj) thực hiện thuật toán DFS không sử dụng đệ quy trên đồ thị biểu diễn bằng danh sách kề. 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.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

#include <vector>

#include <list>

#include <iostream>

#include <stack>

// Duyệt đồ thị theo chiều sâu (DFS)

void dfs(std::vector<std::list<int>> &adj) {

std::stack<int> S;

std::vector<bool> visited(adj.size(), false);

S.push(1); // Bắt đầu từ đỉnh 1 (theo giả định bài toán)

while (!S.empty()) {

int u = S.top();

if (!visited[u]) {

visited[u] = true;

std::cout << u << std::endl; // In đỉnh đã thăm

}

if (!adj[u].empty()) {

int v = adj[u].front();

adj[u].pop\_front();

if (!visited[v]) {

S.push(v);

}

} else {

S.pop();

}

}

}

int main() {

int n = 7; // Số đỉnh

std::vector<std::list<int>> adj;

adj.resize(n + 1); // Đồ thị 1-based, chứa các đỉnh từ 1 đến n

// Thêm cạnh vào đồ thị

adj[1].push\_back(2);

adj[2].push\_back(4);

adj[1].push\_back(3);

adj[3].push\_back(4);

adj[3].push\_back(5);

adj[5].push\_back(2);

adj[2].push\_back(7);

adj[6].push\_back(7);

// Gọi DFS

dfs(adj);

return 0;

}

### Bài 4.6: Viết hàm void bfs(vector< list<int> > adj) thực hiện thuật toán BFS không sử dụng đệ quy trên đồ thị biểu diễn bằng danh sách kề. 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.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

#include <vector>

#include <list>

#include <iostream>

#include <queue>

using namespace std;

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

queue<int> Q;

vector<bool> was(adj.size(), false);

Q.push(1); // Bắt đầu từ đỉnh 1 (theo giả định bài toán)

was[1] = true;

while (!Q.empty()) {

int u = Q.front();

Q.pop();

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

for (int v : adj[u]) {

if (!was[v]) {

was[v] = true;

Q.push(v);

}

}

}

cout << endl;

}

int main() {

int n = 7; // Số đỉnh

vector<list<int>> adj(n + 1); // Đồ thị 1-based, chứa các đỉnh từ 1 đến n

// Thêm cạnh vào đồ thị

adj[1].push\_back(2);

adj[2].push\_back(4);

adj[1].push\_back(3);

adj[3].push\_back(4);

adj[3].push\_back(5);

adj[5].push\_back(2);

adj[2].push\_back(7);

adj[6].push\_back(7);

// Gọi BFS

bfs(adj);

return 0;

}

### Bài 4.7: Viết các hàm thực hiện các phép giao và hợp của hai tập hợp được biểu diễn bằng set.

template<class T>

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

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# YOUR CODE HERE #

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

}

template<class T>

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

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# YOUR CODE HERE #

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

}

Lưu ý: Trong ví dụ dưới đây, hàm print\_set() là hàm được định nghĩa sẵn như sau:

template<class T>

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

for (const T &x : a) {

std::cout << x << ' ';

}

std::cout << std::endl;}

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

#include <iostream>

#include <set>

// Tính hợp của hai tập hợp

template <class T>

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

std::set<T> c = a;

c.insert(b.begin(), b.end());

return c;

}

// Tính giao của hai tập hợp

template <class T>

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

std::set<T> c;

for (const T &x : a) {

if (b.find(x) != b.end()) {

c.insert(x);

}

}

return c;

}

// In tập hợp

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() {

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;

}

### Bài 4.8: Viết các hàm thực hiện các phép giao và hợp của hai tập hợp mờ được biểu diễn bằng map.

Trong đó mỗi phần tử được gán cho một số thực trong đoạn [0..1] biểu thị độ thuộc của phần tử trong tập hợp, với độ thuộc bằng 1 nghĩa là phần tử chắc chắn thuộc vào tập hợp và ngược lại độ thuộc bằng 0 nghĩa là phần tử chắc chắn không thuộc trong tập hợp.

Phép giao và hợp của 2 tập hợp được thực hiện trên các cặp phần tử bằng nhau của 2 tập hợp, với độ thuộc mới được tính bằng phép toán min và max của hai độ thuộc.

template<class T>

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

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# YOUR CODE HERE #

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

}

template<class T>

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

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# YOUR CODE HERE #

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

}

Lưu ý: Trong ví dụ dưới đây, hàm print\_fuzzy\_set() được định nghĩa sẵn như sau:

template<class T>

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

cout << "{ ";

for (const auto &x : a) {

cout << "(" << x.first << ", " << x.second << ") ";

}

cout << "}";

cout << endl;

}

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

#include <iostream>

#include <map>

#include <algorithm> // Để sử dụng hàm std::max và std::min

using namespace std;

// Hợp mờ của hai tập hợp

template <class T>

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

std::map<T, double> c = a;

for (const auto &e : b) {

if (c.count(e.first)) {

c[e.first] = max(c[e.first], e.second);

} else {

c.insert(e);

}

}

return c;

}

// Giao mờ của hai tập hợp

template <class T>

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

std::map<T, double> c;

for (const auto &x : a) {

const auto it = b.find(x.first);

if (it != b.end()) {

c[x.first] = min(x.second, it->second);

}

}

return c;

}

// In tập hợp mờ

template <class T>

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

cout << "{ ";

for (const auto &x : a) {

cout << "(" << x.first << ", " << x.second << ") ";

}

cout << "}" << endl;

}

int main() {

// Khởi tạo các tập hợp mờ

std::map<int, double> a = {{1, 0.2}, {2, 0.5}, {3, 1.0}, {4, 0.6}, {5, 0.7}};

std::map<int, double> b = {{1, 0.5}, {2, 0.4}, {4, 0.9}, {5, 0.4}, {6, 1.0}};

// In các tập hợp ban đầu

std::cout << "A = ";

print\_fuzzy\_set(a);

std::cout << "B = ";

print\_fuzzy\_set(b);

// Tính hợp mờ và giao mờ

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

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

// In kết quả

std::cout << "Union: ";

print\_fuzzy\_set(c);

std::cout << "Intersection: ";

print\_fuzzy\_set(d);

return 0;

}

### Bài 4.9: Cài đặt thuật toán Dijkstra trên đồ thị vô hướng được biểu diễn bằng danh sách kề sử dụng std::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.

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

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# YOUR CODE HERE #

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

}

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

#include <iostream>

#include <queue>

#include <climits>

#include <vector>

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

// priority queue stores pairs (distance from 0, vertex)

std::priority\_queue<std::pair<int, int>, std::vector<std::pair<int, int>>, std::greater<>> Q;

std::vector<int> d(adj.size(), INT\_MAX);

d[0] = 0; // start from vertex 0

Q.push({0, 0}); // push the start node with distance 0

while (!Q.empty()) {

int du = Q.top().first; // current distance

int u = Q.top().second; // current node

Q.pop();

// If this distance is not up-to-date, skip

if (du != d[u]) continue;

// Explore neighbors

for (auto e : adj[u]) {

int v = e.first; // neighbor vertex

int c = e.second; // edge cost

if (d[v] > d[u] + c) {

d[v] = d[u] + c; // update distance

Q.push({d[v], v}); // push to priority queue

}

}

}

return d;

}

int main() {

int n = 9; // Number of vertices

std::vector<std::vector<std::pair<int, int>>> adj(n);

// Function to add edges

auto add\_edge = [&adj](int u, int v, int w) {

adj[u].push\_back({v, w});

adj[v].push\_back({u, w}); // Assuming undirected graph

};

// Add edges

add\_edge(0, 1, 4);

add\_edge(1, 7, 11);

add\_edge(0, 7, 8);

add\_edge(1, 2, 8);

add\_edge(2, 3, 7);

add\_edge(2, 8, 2);

add\_edge(3, 4, 9);

add\_edge(3, 5, 14);

add\_edge(4, 5, 10);

add\_edge(5, 6, 2);

add\_edge(6, 7, 1);

add\_edge(6, 8, 6);

add\_edge(7, 8, 7);

// Run Dijkstra algorithm

std::vector<int> distance = dijkstra(adj);

// Output the distances

for (int i = 0; i < distance.size(); ++i) {

std::cout << "distance 0->" << i << " = " << distance[i] << std::endl;

}

return 0;

}

## BÀI TẬP VỀ NHÀ

### Bài 4.10: Xây dựng một máy tìm kiếm (search engine) đơn giản. Xây dựng một máy tìm kiếm (search engine) đơn giản.

Cho N văn bản và Q 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:

Ảnh có chứa văn bản

Mô tả được tạo tự động

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

**Input:**

* Dòng đầu tiên chứa số N
* Dòng thứ i trong N dòng tiếp theo thể hiện văn bản i, mỗi dòng là một dãy các từ ngăn cách nhau bởi dấu phẩy
* Dòng tiếp theo chứa số Q
* Dòng thứ i trong Q dòng tiếp theo thể hiện truy vấn thứ i, mỗi dòng là một dãy các từ ngăn cách nhau bởi dấu phẩy

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

**Giới hạn:**

* N ≤ 1000
* Q ≤ 1000
* Số từ trong mỗi văn bản không quá 1000
* Số từ trong mỗi truy vấn không quá 10
* Độ dài mỗi từ không quá 10

**Tham khảo:**

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

#include <bits/stdc++.h>

using namespace std;

typedef unordered\_map<string, int> HashMap;

const int MAX\_N = 1000;

const double EPS = 1e-9;

int N, Q;

vector<vector<string>> documents, queries;

unordered\_map<string, double> idf; // Thay đổi idf từ vector sang HashMap

HashMap df; // Số lượng văn bản chứa từ t

// Hàm tách chuỗi theo dấu phẩy

vector<string> split(const string &s) {

    vector<string> result;

    stringstream ss(s);

    string word;

    while (getline(ss, word, ',')) {

        result.push\_back(word);

    }

    return result;

}

// Tiền xử lý IDF

void preprocess\_idf() {

    for (int i = 0; i < N; i++) {

        unordered\_set<string> unique\_words;

        for (const string &word : documents[i]) {

            unique\_words.insert(word);

        }

        for (const string &word : unique\_words) {

            df[word]++;

        }

    }

    for (const auto &entry : df) {

        string word = entry.first;

        int doc\_count = entry.second;

        idf[word] = log2((double)N / doc\_count);

    }

}

// Tính TF-IDF điểm của một văn bản d với truy vấn query

double compute\_score(const vector<string> &doc, const vector<string> &query) {

    HashMap freq;

    int max\_freq = 0;

    // Tính f(t, d) và maxf(d)

    for (const string &word : doc) {

        freq[word]++;

        max\_freq = max(max\_freq, freq[word]);

    }

    double score = 0.0;

    for (const string &q\_word : query) {

        if (freq.find(q\_word) != freq.end()) {

            double tf = 0.5 + 0.5 \* (double)freq[q\_word] / max\_freq;

            score += tf \* idf[q\_word];

        }

    }

    return score;

}

int main() {

    // Đọc số lượng văn bản

    cin >> N;

    documents.resize(N);

    cin.ignore();

    for (int i = 0; i < N; i++) {

        string line;

        getline(cin, line);

        documents[i] = split(line);

    }

    // Đọc số lượng truy vấn

    cin >> Q;

    queries.resize(Q);

    cin.ignore();

    for (int i = 0; i < Q; i++) {

        string line;

        getline(cin, line);

        queries[i] = split(line);

    }

    // Tiền xử lý IDF

    preprocess\_idf();

    // Xử lý từng truy vấn

    for (int q = 0; q < Q; q++) {

        int best\_doc = -1;

        double best\_score = -1e9;

        for (int d = 0; d < N; d++) {

            double score = compute\_score(documents[d], queries[q]);

            if (score > best\_score || (fabs(score - best\_score) < EPS && d < best\_doc)) {

                best\_score = score;

                best\_doc = d;

            }

        }

        cout << best\_doc + 1 << "\n"; // In chỉ số (1-based index)

    }

    return 0;

}

Input1:

A white background with black text

Description automatically generated

Output1:

A computer screen with white text

Description automatically generated

Input2:

A white text with black letters

Description automatically generated

Output2:

A screenshot of a computer program

Description automatically generated

A computer screen with white text

Description automatically generated

Input3:

A white background with black letters

Description automatically generated

Output3:

A computer screen shot of a black screen

Description automatically generated

A computer screen shot of a black screen

Description automatically generated

Input4:

A screenshot of a computer screen

Description automatically generated

Output4:

A computer screen with white text

Description automatically generated

A black screen with white text

Description automatically generated

Input5:

A screen shot of a computer

Description automatically generated

Output5:

A screenshot of a computer program

Description automatically generated

A computer screen with white text

Description automatically generated

Input6:

A white text with black letters

Description automatically generated

Output6:

A computer screen shot of a black screen

Description automatically generated

A computer screen shot of a black screen

Description automatically generated

### Bài 4.11: 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 ai tên 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×ki 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 nn và ss (1 ≤ n ≤ 100000; 1 ≤ s ≤ 10^9).

nn dòng tiếp theo chứa hai số nguyên aiai và kiki lần lượt là số tên giặc tấn công đoạn tường thứ ii và khả năng chống trả của một lính ở đoạn tường thứ ii (1 ≤ ai,ki ≤ 10^9).

**Kết quả:**

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

#include<bits/stdc++.h>

#define ll long long

using namespace std;

const int MAX = 1e5 + 7;

int n, s, quantities[MAX], values[MAX];

ll result = 0;

multiset<pair<int, int>, greater<pair<int, int>>> itemSet;

int main()

{

// Đọc vào số lượng sản phẩm và số lượng cần chọn

cin >> n >> s;

// Đọc vào thông tin các sản phẩm: số lượng và giá trị

for(int i = 1; i <= n; i++)

{

cin >> quantities[i] >> values[i];

// Thêm vào multiset theo thứ tự giảm dần của giá trị

itemSet.insert({values[i], quantities[i]});

result += quantities[i];

}

// Lặp cho đến khi hết sản phẩm hoặc không còn sản phẩm để chọn

while (!itemSet.empty() && s > 0)

{

// Lấy phần tử có giá trị cao nhất

auto top = itemSet.begin();

auto currentItem = (\*top);

itemSet.erase(top);

// Nếu số lượng cần chọn nhỏ hơn số lượng của sản phẩm hiện tại

if (s < (currentItem.second / currentItem.first))

{

result -= currentItem.first \* s;

s = 0;

}

else

{

result -= currentItem.first \* (currentItem.second / currentItem.first);

s -= (currentItem.second / currentItem.first);

}

// Cập nhật lại số lượng của sản phẩm và nếu còn lại, thêm lại vào multiset

currentItem.second -= currentItem.first \* (currentItem.second / currentItem.first);

if (currentItem.second > 0)

itemSet.insert({currentItem.second, currentItem.second});

}

// In ra kết quả cuối cùng

cout << result;

}

Input1:

Output1:

A screenshot of a computer

Description automatically generated

Input2:

Output2:

A computer screen shot of a black screen

Description automatically generated

Input3:

A screenshot of a computer

Description automatically generated

Output3:

A computer screen with white text

Description automatically generated

Input4:

A screenshot of a computer

Description automatically generated

Output4:

A black screen with white text

Description automatically generated

Input5:

A screenshot of a computer

Description automatically generated

Output5:

A computer screen with white text

Description automatically generated

Input6:

A screenshot of a computer

Description automatically generated

Output6:

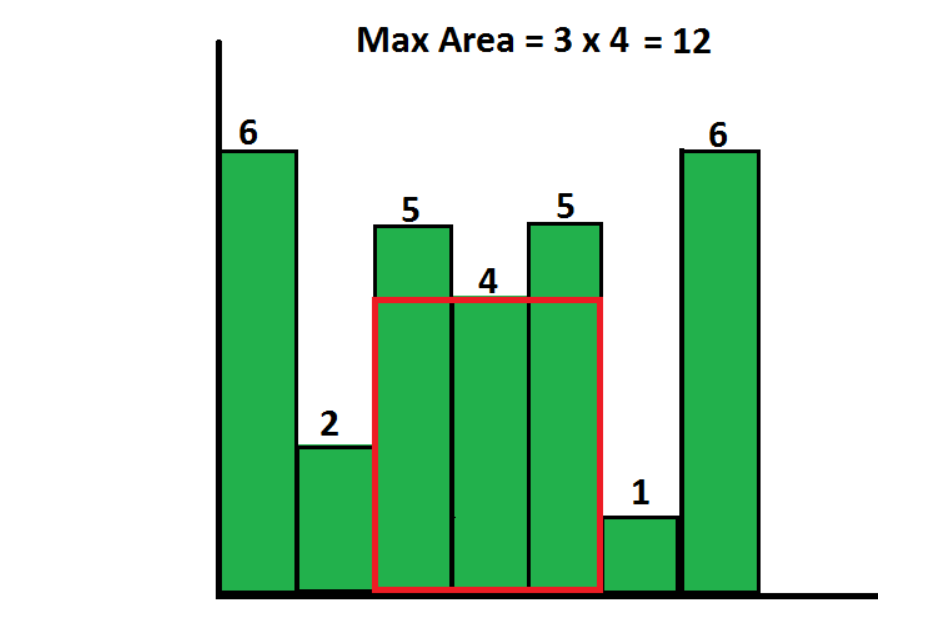
A computer screen shot of a black screen

Description automatically generated

### Bài 4.12: 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 nn (1 ≤ n ≤ 10^6).  
Dòng thứ hai chứa nn số nguyên không âm h1,h2,…,hnh1,h2,…,hn cách nhau bởi dấu cách (0 ≤ hi ≤ 10^9).

**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.



#include <bits/stdc++.h>

using namespace std;

using ll = long long;

ll largestRectangleArea(vector<ll> &heights) {

stack<int> st; // Stack lưu chỉ số cột

ll max\_area = 0;

int n = heights.size();

for (int i = 0; i < n; ++i) {

// Khi cột hiện tại thấp hơn cột trên đỉnh Stack

while (!st.empty() && heights[st.top()] > heights[i]) {

ll h = heights[st.top()];

st.pop();

ll width = st.empty() ? i : (i - st.top() - 1);

max\_area = max(max\_area, h \* width);

}

st.push(i);

}

// Xử lý các cột còn lại trong Stack

while (!st.empty()) {

ll h = heights[st.top()];

st.pop();

ll width = st.empty() ? n : (n - st.top() - 1);

max\_area = max(max\_area, h \* width);

}

return max\_area;

}

int main() {

int n;

cin >> n;

vector<ll> heights(n);

for (int i = 0; i < n; ++i) {

cin >> heights[i];

}

cout << largestRectangleArea(heights) << endl;

return 0;

}

Input1:

Output1:

A black screen with white text

Description automatically generated

Input2:

A number pattern on a white background

Description automatically generated

Output2:

A screenshot of a computer

Description automatically generated

Input3:

A screenshot of a computer screen

Description automatically generated

Output3:

A screenshot of a computer

Description automatically generated

Input4:

A number pattern with black numbers

Description automatically generated with medium confidence

Output4:

A screen shot of a computer

Description automatically generated

Input5:

A screenshot of a computer code

Description automatically generated

Output5:

A screenshot of a computer

Description automatically generated

Input6:

A screenshot of a computer screen

Description automatically generated

Output6:

A screenshot of a computer screen

Description automatically generated

### Bài 4.13: 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 ≤ 10^6).

**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.

#include <bits/stdc++.h>

using namespace std;

using ll = long long;

ll countBalancedSubstrings(const string &s) {

unordered\_map<int, ll> balanceCount; // Map lưu tần suất của từng giá trị balance

balanceCount[0] = 1; // Giá trị ban đầu của balance = 0

ll balance = 0, count = 0;

for (char c : s) {

// Cập nhật giá trị balance

if (c == '1') {

balance++;

} else { // c == '0'

balance--;

}

// Nếu balance đã xuất hiện, cộng số lần xuất hiện trước đó vào kết quả

if (balanceCount.find(balance) != balanceCount.end()) {

count += balanceCount[balance];

}

// Cập nhật số lần xuất hiện của balance

balanceCount[balance]++;

}

return count;

}

int main() {

string s;

cin >> s;

cout << countBalancedSubstrings(s) << endl;

return 0;

}

Input1:

Output1: A screen shot of a computer

Description automatically generated

Input2:

Output2:A black screen with white text

Description automatically generated

Input3:

Output3: A computer screen with numbers

Description automatically generated

Input4:A screenshot of a computer

Description automatically generated

Output4:A computer screen with numbers

Description automatically generated

Input5:

A screenshot of a computer

Description automatically generated

Output5: A computer screen with numbers and digits

Description automatically generated

Input6:

A screenshot of a computer

Description automatically generated

Output6:A computer screen with numbers and text

Description automatically generated