Mục lục

[Phần 1: Bài tập thực hành 3](#_Toc168163346)

[**Bài tập 1: Đảo ngược một danh sách liên kết đơn** 3](#_Toc168163347)

[**Bài tập 2: Tính diện tích tam giác** 5](#_Toc168163348)

[**Bài tập 3: Tính tích có hướng của 2 vector** 7](#_Toc168163349)

[**Bài tập 4: Thao tác với vector** 9](#_Toc168163350)

[**Bài tập 5:** 12](#_Toc168163351)

[**Viết hàm thực hiện thuật toán DFS không sử dụng đệ quy trên đồ thị biểu diễn bằng danh sách kề vector< list<int> > .** 12](#_Toc168163352)

[**Bài tập 6:** 14](#_Toc168163353)

[**Viết hàm thực hiện thuật toán BFS không sử dụng đệ quy trên đồ thị biểu diễn bằng danh sách kề vector< list<int> > .** 14](#_Toc168163354)

[**Bài tập 7:** 16](#_Toc168163355)

[**Viết các hàm thực hiện các phép giao và hợp của hai tập hợp được biểu diễn bằng set** 17](#_Toc168163356)

[**Bài tập 8:** 18](#_Toc168163357)

[**Viết các hàm thực hiện các phép giao và hợp của hai tập hợp mờ được biểu diễn bằng map.** 18](#_Toc168163358)

[**Bài tập 9:** 22](#_Toc168163359)

[**Cài đặt thuật toán Dijkstra trên đồ thị vô hướng được biểu diễn bằng danh sách kề sử dụng priority\_queue** 22](#_Toc168163360)

[Phần 2: Bài tập về nhà 25](#_Toc168163361)

[**Bài tập 10: Search Engine** 25](#_Toc168163362)

[**Xây dựng một máy tìm kiếm (search engine) đơn giản.** 25](#_Toc168163363)

[**Bài tập 11: Lịch trình chụp ảnh** 31](#_Toc168163364)

[**Bài tập 12. Lược đồ 33**](#_Toc168163365)

[**Bài tập 13: Đếm xâu con 36**](#_Toc168163366)

Mục lục hình ảnh

[Figure 1. Đảo ngược một danh sách liên kết đơn 3](#_Toc168163510)

[Figure 2. Tính diện tích tam giác 6](#_Toc168163511)

[Figure 3. Tính tích có hướng của 2 vector 8](#_Toc168163512)

[Figure 4. Thao tác với vector 9](#_Toc168163513)

[Figure 5. Thuật toán DFS không sử dụng đệ quy trên đồ thị biểu diễn bằng danh sách kề vector< list<int> > 13](#_Toc168163514)

[Figure 6. Thực hiện thuật toán BFS không sử dụng đệ quy trên đồ thị biểu diễn bằng danh sách kề vector< list<int> > 15](#_Toc168163515)

[Figure 7. Hàm thực hiện các phép giao và hợp của hai tập hợp được biểu diễn bằng set 17](#_Toc168163516)

[Figure 8. Các hàm thực hiện các phép giao và hợp của hai tập hợp mờ được biểu diễn bằng map 19](#_Toc168163517)

[Figure 9. Thuật toán Dijkstra trên đồ thị vô hướng được biểu diễn bằng danh sách kề sử dụng priority\_queue 23](#_Toc168163518)

[Figure 10. Xây dựng một máy tìm kiếm (search engine) đơn giản 29](#_Toc168163519)

[Figure 11. Lịch trình chụp ảnh 32](#_Toc168163520)

[Figure 12. Lược đồ 35](#_Toc168163521)

[Figure 13. Đếm xâu con 37](#_Toc168163522)

**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)**

Figure 1. Đảo ngược một danh sách liên kết đơn

//Nhữ Ngọc Minh- 20225046

#include <bits/stdc++.h>

using namespace std;

struct Node {

    int data;

    Node\* next;

    Node(int data) {

        this->data = data;

        next = NULL;

    }

};

// push a new element to the beginning of the list

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

    Node\* tmp1=new Node(data);

    if (head==NULL) return tmp1;

    else tmp1->next=head;

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

    # YOUR CODE HERE #

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

    return tmp1;

}

// print the list content on a line

void print(Node\* head) {

    Node\* tmp=head;

    while (tmp!=NULL)

    {

        cout<<tmp->data<<" ";

        tmp=tmp->next;

    }

    cout<<"\n";

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

    # YOUR CODE HERE #

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

}

// return the new head of the reversed list

Node\* reverse(Node\* head) {

    if (head==NULL) return NULL;

    Node\* tmp=head,\*tmp1=head->next;

    if (tmp1==NULL) return tmp;

    while(tmp1->next!=NULL)

    {

        tmp=tmp->next;

        tmp1=tmp1->next;

    };

    tmp->next=NULL;

    tmp1->next=reverse(head);

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

    # YOUR CODE HERE #

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

    return tmp1;

}

int main() {

    int n, u;

    cin >> n;

    Node\* head = NULL;

    for (int i = 0; i < n; ++i){

        cin >> u;

        head = prepend(head, u);

    }

    cout << "Original list: ";

    print(head);

    head = reverse(head);

    cout << "Reversed list: ";

    print(head);

    return 0;

}

## **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>;**

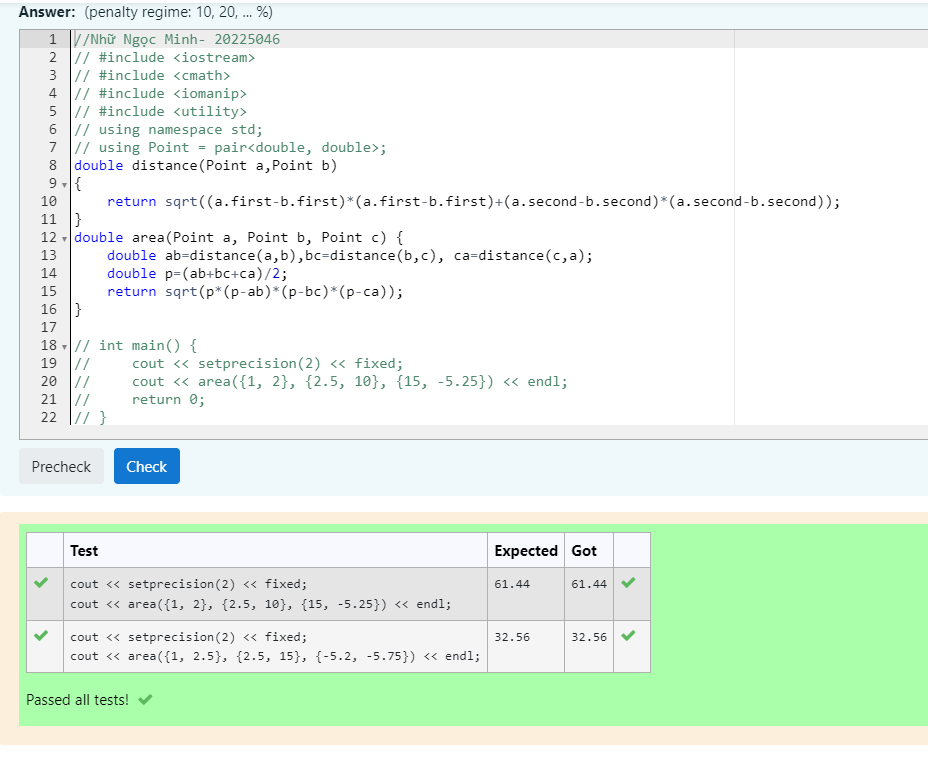


Figure 2. Tính diện tích tam giác

//Nhữ Ngọc Minh- 20225046

// #include <iostream>

// #include <cmath>

// #include <iomanip>

// #include <utility>

// using namespace std;

// using Point = pair<double, double>;

double distance(Point a,Point 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) {

    double ab=distance(a,b),bc=distance(b,c), ca=distance(c,a);

    double p=(ab+bc+ca)/2;

    return sqrt(p\*(p-ab)\*(p-bc)\*(p-ca));

}

// int main() {

//     cout << setprecision(2) << fixed;

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

//     return 0;

// }

## **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>;**

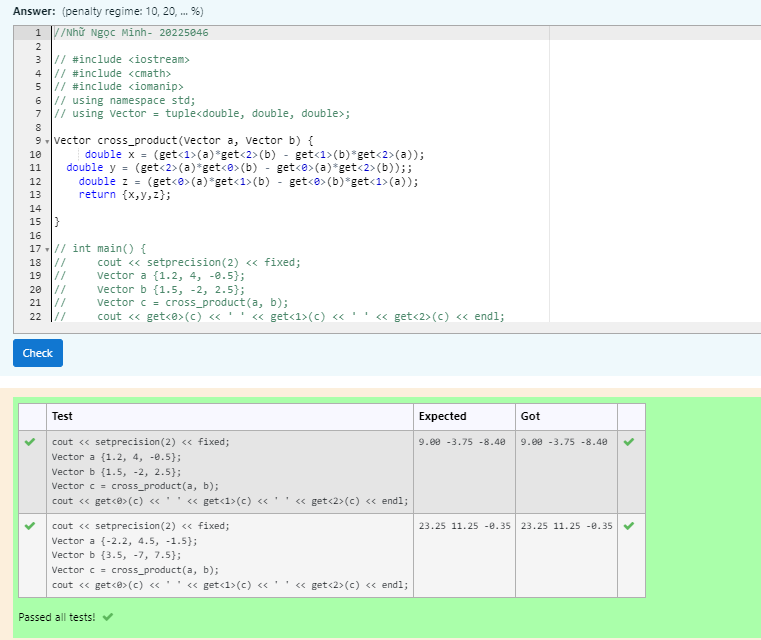


Figure 3. Tính tích có hướng của 2 vector

//Nhữ Ngọc Minh- 20225046

// #include <iostream>

// #include <cmath>

// #include <iomanip>

// using namespace std;

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

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

     double x = (get<1>(a)\*get<2>(b) - get<1>(b)\*get<2>(a));

  double y = (get<2>(a)\*get<0>(b) - get<0>(a)\*get<2>(b));;

    double z = (get<0>(a)\*get<1>(b) - get<0>(b)\*get<1>(a));

    return {x,y,z};

}

// int main() {

//     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;

// }

## **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.**

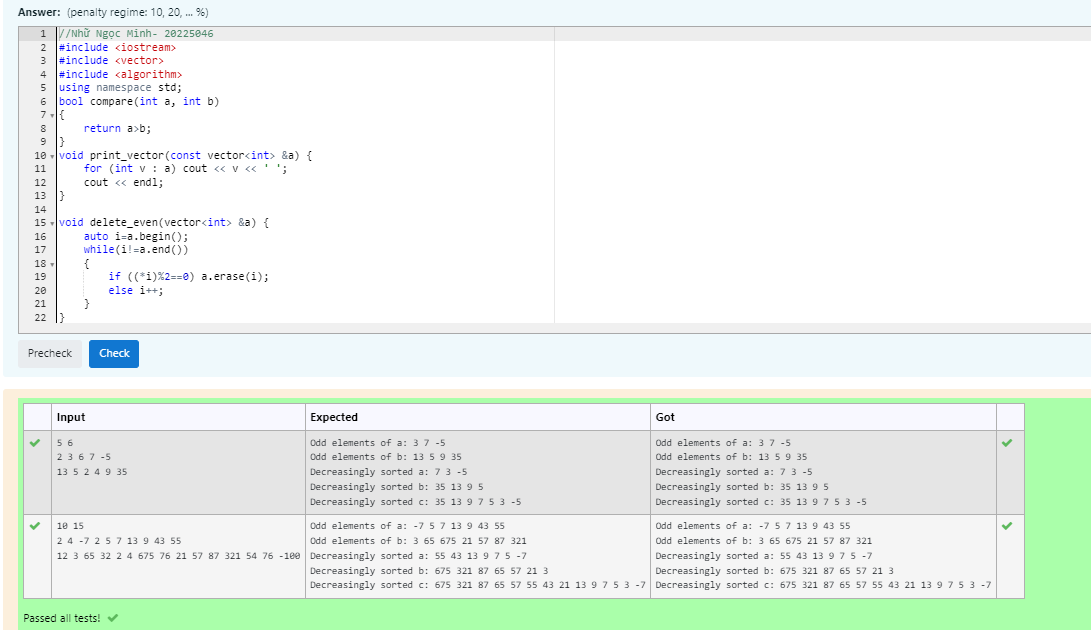


Figure 4. Thao tác với vector

//Nhữ Ngọc Minh- 20225046

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

bool compare(int a, int b)

{

    return a>b;

}

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

    for (int v : a) cout << v << ' ';

    cout << endl;

}

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

    auto i=a.begin();

    while(i!=a.end())

    {

        if ((\*i)%2==0) a.erase(i);

        else i++;

    }

}

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

    sort(a.begin(),a.end(),compare);

}

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

    vector<int> merge;

    for (int v:a ) merge.push\_back(v);

    for (int v:b) merge.push\_back(v);

    sort(merge.begin(),merge.end(),compare);

    return merge;

}

int main() {

    int m, n, u;

    std::vector<int> a, b;

    std::cin >> m >> n;

    for(int i = 0; i < m; i++){

        std:: cin >> u;

        a.push\_back(u);

    }

    for(int i = 0; i < n; i++){

        std:: 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: ";

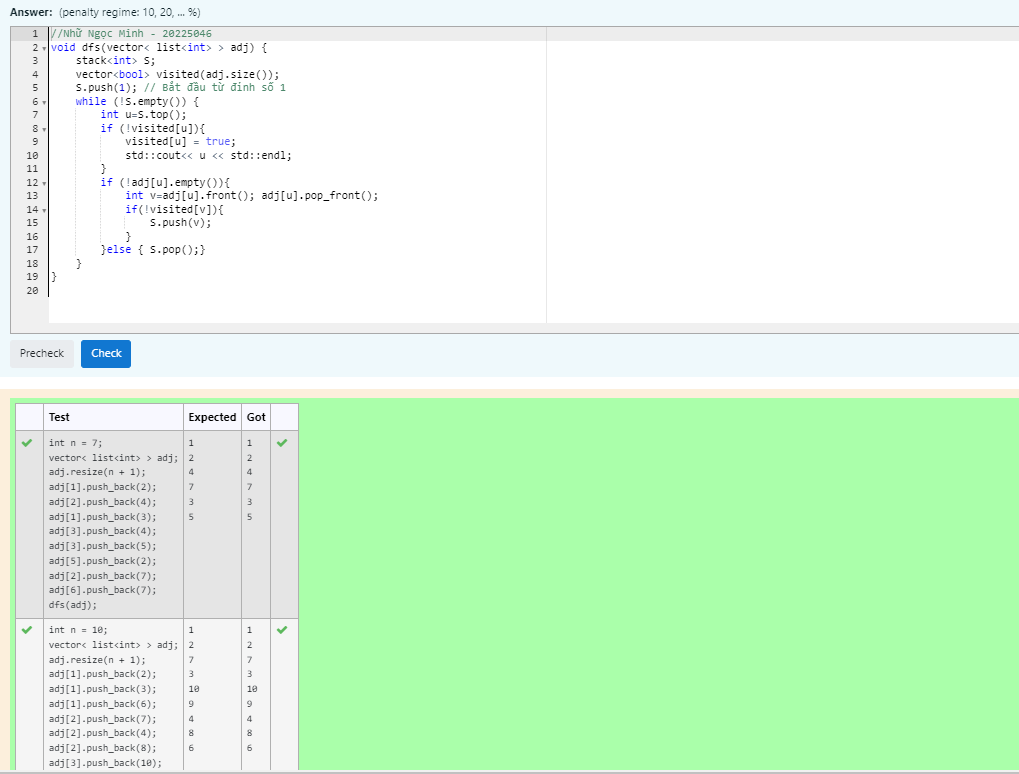
    print\_vector(c);

    return 0;

}

## **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).**



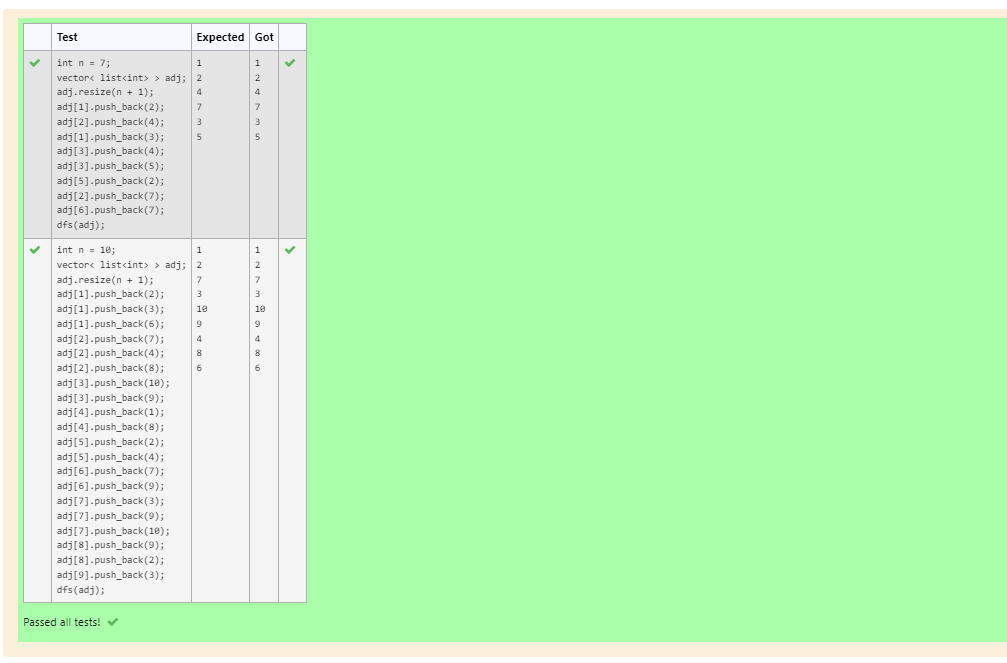


Figure 5. Thuật toán DFS không sử dụng đệ quy trên đồ thị biểu diễn bằng danh sách kề vector< list<int> >

//Nhữ Ngọc Minh - 20225046

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

    stack<int> S;

    vector<bool> visited(adj.size());

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

    while (!S.empty()) {

        int u=S.top();

        if (!visited[u]){

            visited[u] = true;

            std::cout<< u << std::endl;

        }

        if (!adj[u].empty()){

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

            if(!visited[v]){

                S.push(v);

            }

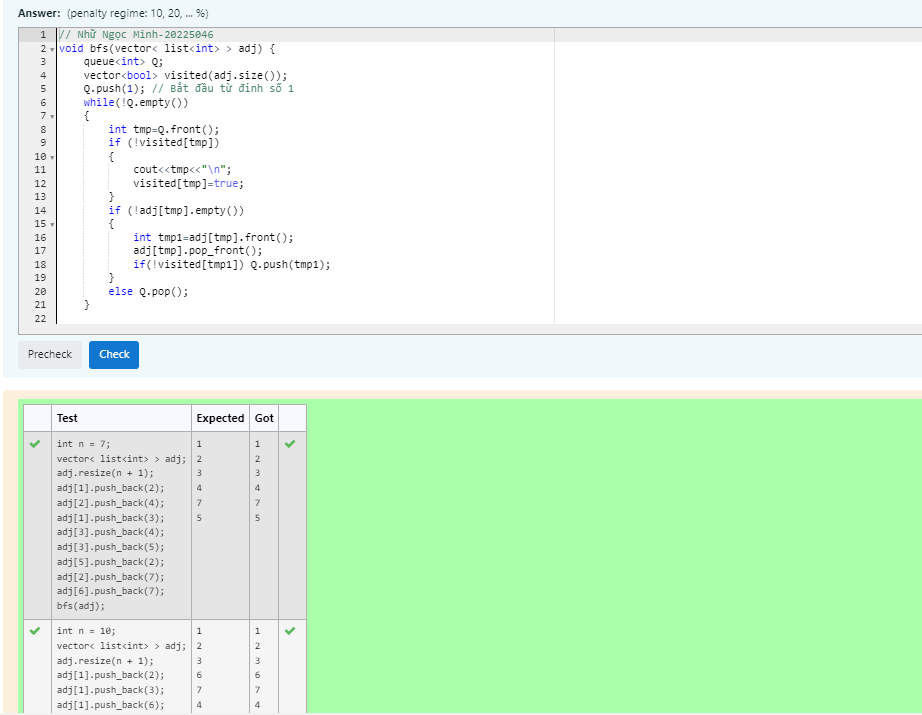
        }else { S.pop();}

    }

}

## **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).**



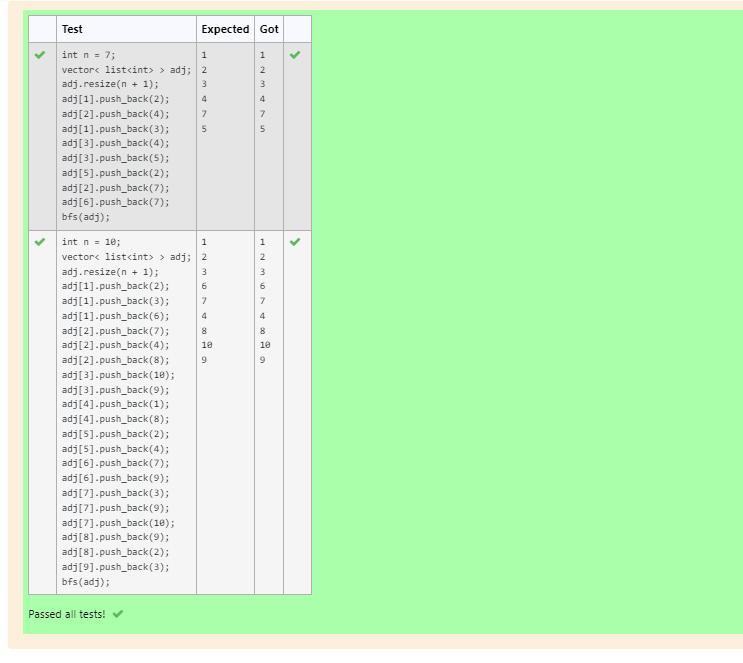


Figure 6. Thực hiện thuật toán BFS không sử dụng đệ quy trên đồ thị biểu diễn bằng danh sách kề vector< list<int> >

// Nhữ Ngọc Minh-20225046

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

    queue<int> Q;

    vector<bool> visited(adj.size());

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

    while(!Q.empty())

    {

        int tmp=Q.front();

        if (!visited[tmp])

        {

            cout<<tmp<<"\n";

            visited[tmp]=true;

        }

        if (!adj[tmp].empty())

        {

            int tmp1=adj[tmp].front();

            adj[tmp].pop\_front();

            if(!visited[tmp1]) Q.push(tmp1);

        }

        else Q.pop();

    }

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

    # YOUR CODE HERE #

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

}

## **Bài tập 7:**

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

Figure 7. Hàm thực hiện các phép giao và hợp của hai tập hợp được biểu diễn bằng set

//Nhữ Ngọc Minh- 20225046

template<class T>

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

    /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

        Ho va ten : Nguyen Quang Huy

        MSSV : 20183554

    \*/

    set<T> c;

    for(T v : a){

        c.insert(v);

    }

    for(T v  : b){

        c.insert(v);

    }

    return c;

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

}

template<class T>

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

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

    set<T> c;

    for(T v : a){

        int index = 0;

        for(T u : b){

            if(v == u) index++;

        }

        if(index != 0) c.insert(v);

    }

    return c;

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

}

## **Bài tập 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.



Figure 8. Các hàm thực hiện các phép giao và hợp của hai tập hợp mờ được biểu diễn bằng map

//Nhữ Ngọc Minh- 20225046

// #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) {

  map<T,double> tmp;

  for (auto i:a)

  {

      bool check=false; // biến check chỉ ra rằng tồn tại 1 cặp trong map b có b.key=a.key;

      pair<T,double> tmp1;

      for (auto j:b)

      {

          if (i.first==j.first)

          { tmp1=j;

              check=true;

              break;

          }

      }

      if (check) tmp.insert({i.first,max(i.second,tmp1.second)});

      else tmp.insert({i.first,i.second});

  }

  for (auto i:b)

  { bool check1=false; // check1 chỉ ra đã tồn tại b.first trong các key của tmp chưa

      for (auto j:a)

      {

          if (i.first==j.first)

          { check1=true; break;}

      }

      if (!check1) tmp.insert({i.first,i.second});

  }

  return tmp;

}

template<class T>

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

    map<T,double> tmp;

    for (auto i:a)

    {

        bool check=false; //chỉ ra sự tồn tại của 1 pair trong b mà có key bằng i.key

        pair<T,double> tmp1; // biến lưu trữ phần tử tại b mà có key bằng i.key

        for (auto j:b)

        {

            if (i.first==j.first)

            {   tmp1=j;

                check=true; break;

            }

        }

        if (check) tmp.insert({i.first,min(i.second,tmp1.second)});

    }

    return tmp;

}

// 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 << ") ";

//     }

//     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);

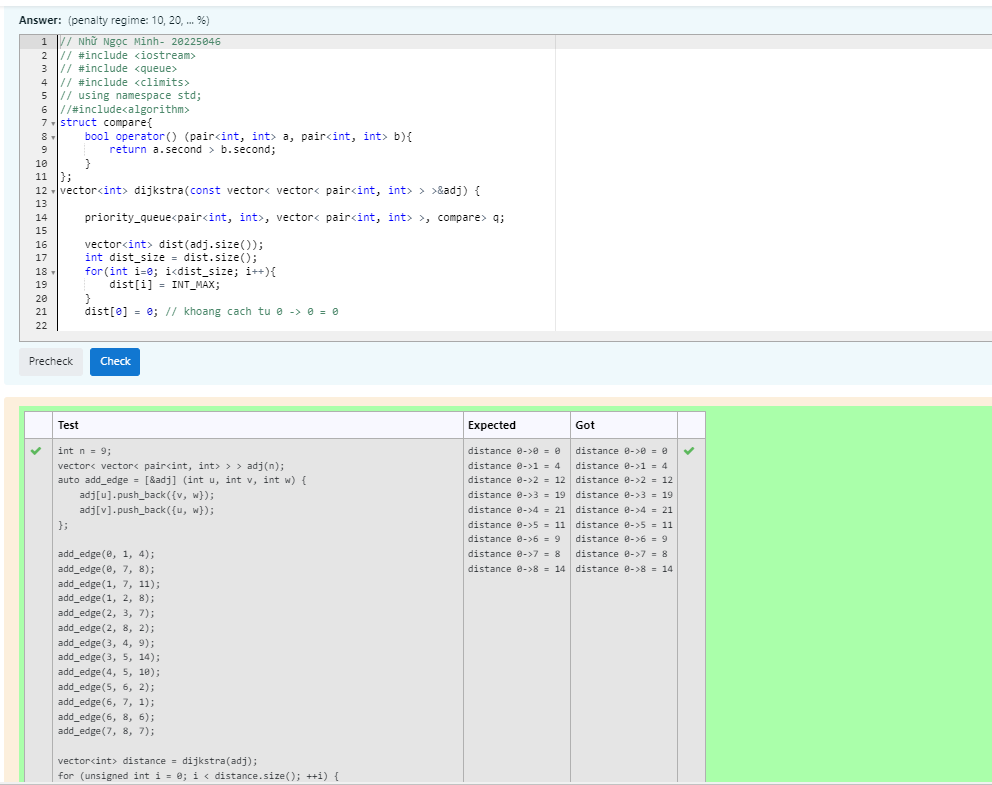
//     std::cout << "Union: "; print\_fuzzy\_set(c);

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

// }

## **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.**



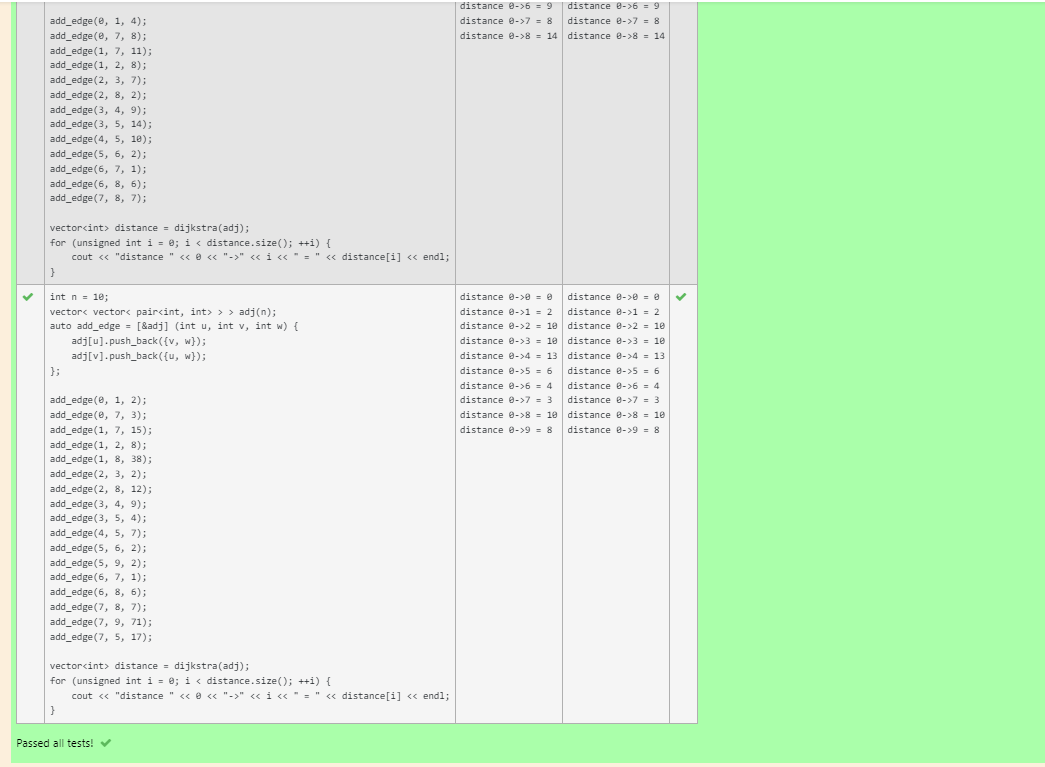


Figure 9. Thuật toán Dijkstra trên đồ thị vô hướng được biểu diễn bằng danh sách kề sử dụng priority\_queue

// Nhữ Ngọc Minh- 20225046

// #include <iostream>

// #include <queue>

// #include <climits>

// using namespace std;

//#include<algorithm>

struct compare{

    bool operator() (pair<int, int> a, pair<int, int> b){

        return a.second > b.second;

    }

};

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

    priority\_queue<pair<int, int>, vector< pair<int, int> >, compare> q;

    vector<int> dist(adj.size());

    int dist\_size = dist.size();

    for(int i=0; i<dist\_size; i++){

        dist[i] = INT\_MAX;

    }

    dist[0] = 0; // khoang cach tu 0 -> 0 = 0

    // khoi tao gia tri cho hang doi uu tien

    for(int i=0; i<dist\_size; i++){

        q.push({i, dist[i]});

    }

    while(!q.empty()){

        pair<int, int> u\_pair = q.top(); q.pop();

        int u = u\_pair.first; // lay ra dinh co thu tu uu tien nhat trong hang doi

        for(auto v\_pair : adj[u]){

            int v = v\_pair.first;

            int weight = v\_pair.second;

            if(dist[v] > dist[u] + weight){

                dist[v] = dist[u] + weight;

                q.push({v, dist[v]});

            }

        }

    }

    return dist;

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

}

// int main() {

//     int n = 9;

//     vector< vector< pair<int, int> > > adj(n);

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

//         adj[u].push\_back({v, w});

//         adj[v].push\_back({u, w});

//     };

//     add\_edge(0, 1, 4);

//     add\_edge(0, 7, 8);

//     add\_edge(1, 7, 11);

//     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);

//     vector<int> distance = dijkstra(adj);

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

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

//     }

//     return 0;

// }

# 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.**

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

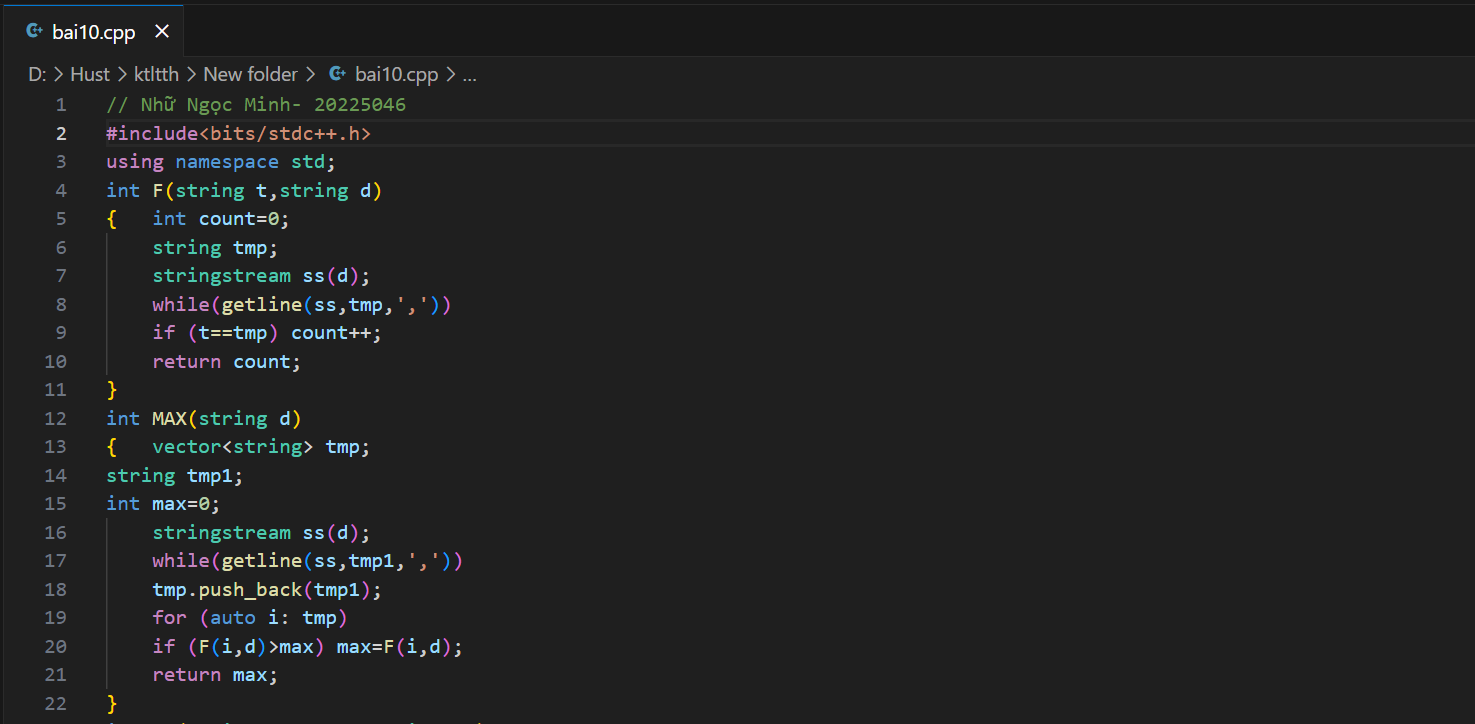
4

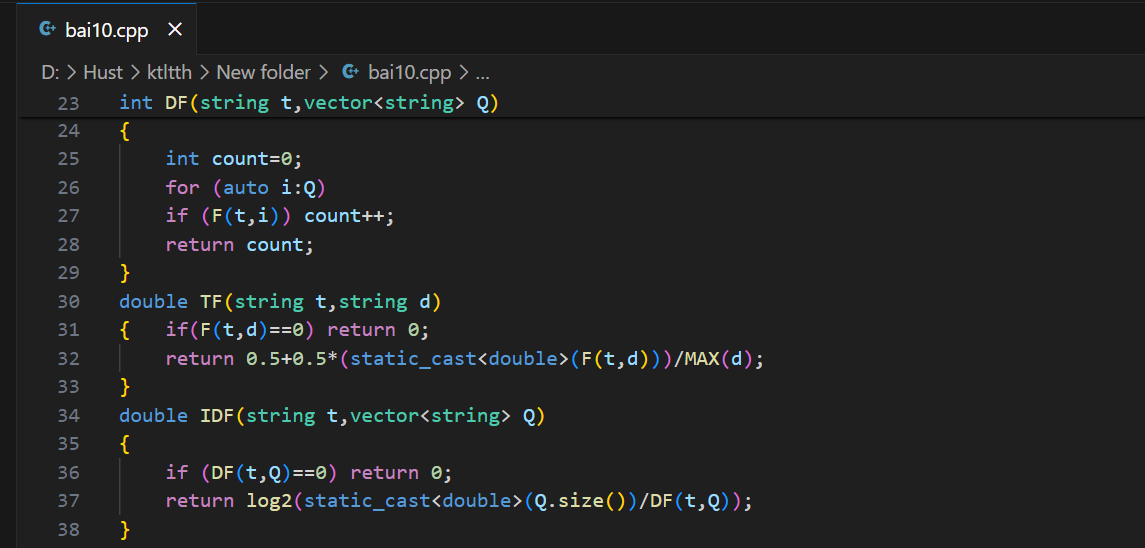
1

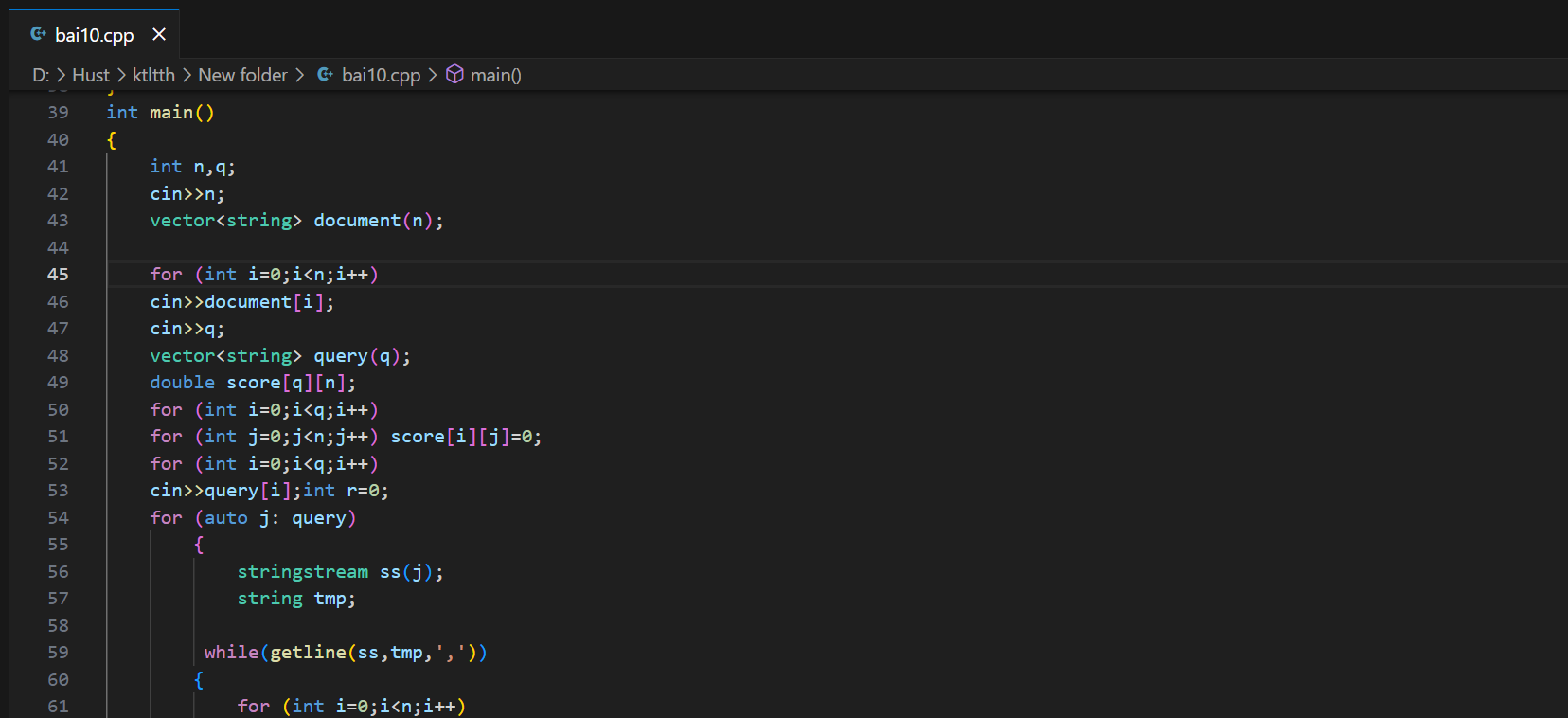
4

**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







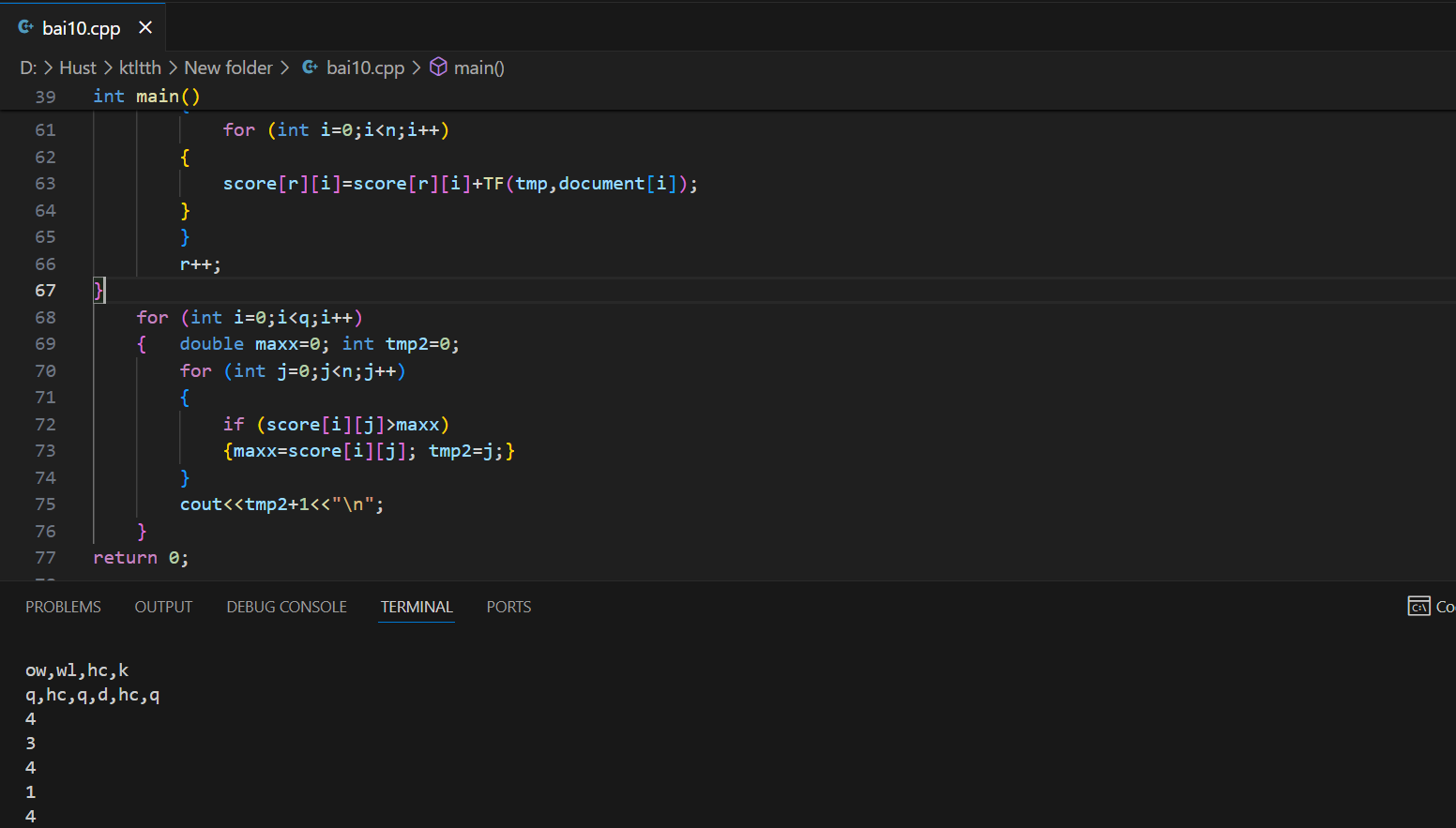


Figure 10. Xây dựng một máy tìm kiếm (search engine) đơn giản

// Nhữ Ngọc Minh- 20225046

#include<bits/stdc++.h>

using namespace std;

int F(string t,string d)

{   int count=0;

    string tmp;

    stringstream ss(d);

    while(getline(ss,tmp,','))

    if (t==tmp) count++;

    return count;

}

int MAX(string d)

{   vector<string> tmp;

string tmp1;

int max=0;

    stringstream ss(d);

    while(getline(ss,tmp1,','))

    tmp.push\_back(tmp1);

    for (auto i: tmp)

    if (F(i,d)>max) max=F(i,d);

    return max;

}

int DF(string t,vector<string> Q)

{

    int count=0;

    for (auto i:Q)

    if (F(t,i)) count++;

    return count;

}

double TF(string t,string d)

{   if(F(t,d)==0) return 0;

    return 0.5+0.5\*(static\_cast<double>(F(t,d)))/MAX(d);

}

double IDF(string t,vector<string> Q)

{

    if (DF(t,Q)==0) return 0;

    return log2(static\_cast<double>(Q.size())/DF(t,Q));

}

int main()

{

    int n,q;

    cin>>n;

    vector<string> document(n);

    for (int i=0;i<n;i++)

    cin>>document[i];

    cin>>q;

    vector<string> query(q);

    double score[q][n];

    for (int i=0;i<q;i++)

    for (int j=0;j<n;j++) score[i][j]=0;

    for (int i=0;i<q;i++)

    cin>>query[i];int r=0;

    for (auto j: query)

        {

            stringstream ss(j);

            string tmp;

         while(getline(ss,tmp,','))

        {

            for (int i=0;i<n;i++)

        {

            score[r][i]=score[r][i]+TF(tmp,document[i]);

        }

        }

        r++;

}

    for (int i=0;i<q;i++)

    {   double maxx=0; int tmp2=0;

        for (int j=0;j<n;j++)

        {

            if (score[i][j]>maxx)

            {maxx=score[i][j]; tmp2=j;}

        }

        cout<<tmp2+1<<"\n";

    }

return 0;

}

## **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.**

**Dữ liệu vào:**

Dòng đầu chứa nn và rr

Tiếp theo là ma trận n×nn×n mô tả chi phí đi lại giữa các địa điểm. Chi phí bằng 0 có nghĩa là không thể đi lại giữa hai địa điểm đó.

rr dòng tiếp theo chứa danh sách các địa điểm mà người rr đưa ra. Lưu ý là hành mỗi hành trình cần phải bắt đầu và kết thúc bởi hai đỉnh đầu và cuối của danh sách, còn các địa điểm còn lại có thể thăm theo bất kỳ thứ tự nào

**Kết quả:**

Gồm rr dòng ghi chi phí đi lại ít nhất của rr người theo thứ tự đầu vào

**Ví dụ:**

**Dữ liệu mẫu:**

6 3

1. 1 2 0 1 1
2. 0 1 1 1 0

0 2 0 1 3 0

4 3 1 0 0 0

1. 0 1 1 0 0
2. 0 0 0 0 0

1 3 5

6 3 2 5

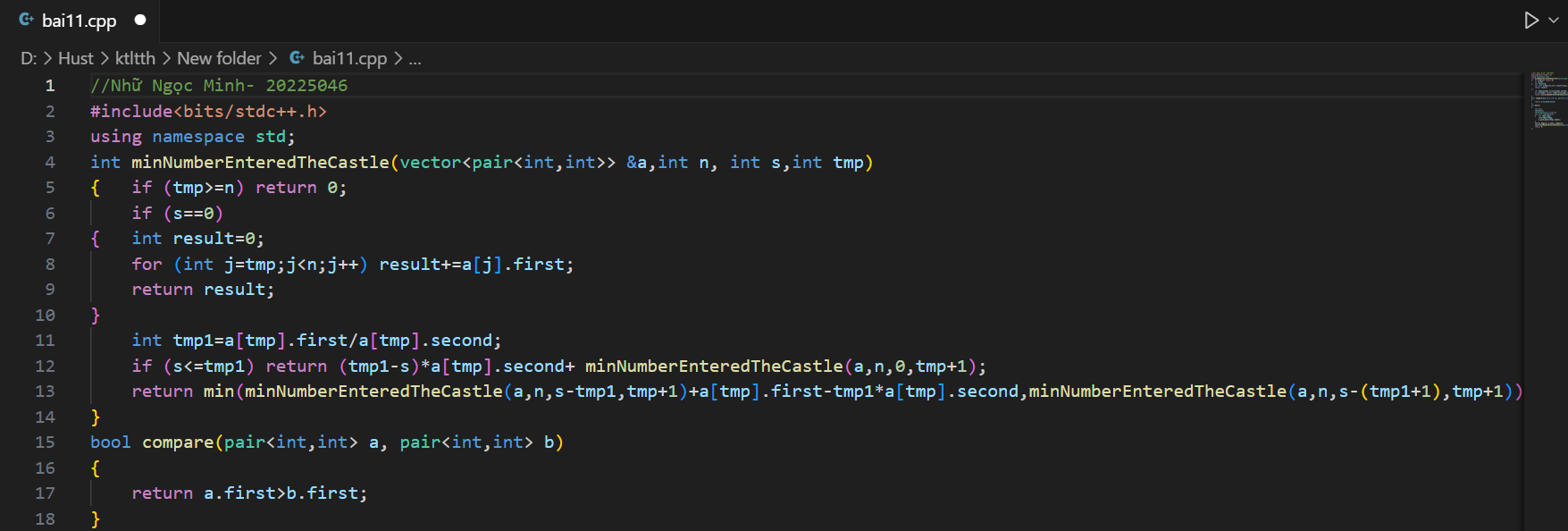
6 1 2 3 4 5

**Kết quả mẫu:**

5

0

7



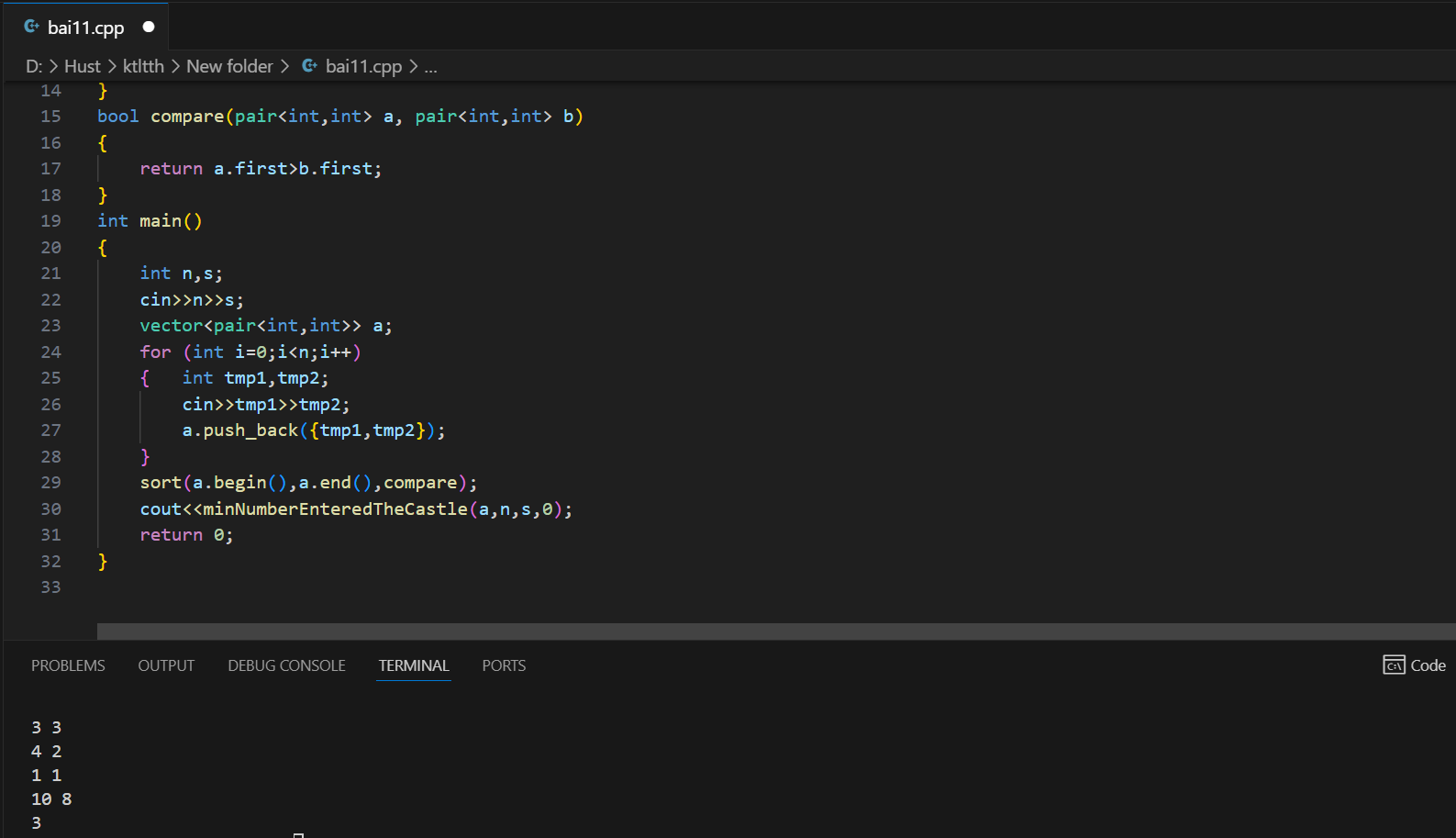


Figure 11. Lịch trình chụp ảnh

//Nhữ Ngọc Minh- 20225046

#include<bits/stdc++.h>

using namespace std;

int minNumberEnteredTheCastle(vector<pair<int,int>> &a,int n, int s,int tmp)

{   if (tmp>=n) return 0;

    if (s==0)

{   int result=0;

    for (int j=tmp;j<n;j++) result+=a[j].first;

    return result;

}

    int tmp1=a[tmp].first/a[tmp].second;

    if (s<=tmp1) return (tmp1-s)\*a[tmp].second+ minNumberEnteredTheCastle(a,n,0,tmp+1);

    return min(minNumberEnteredTheCastle(a,n,s-tmp1,tmp+1)+a[tmp].first-tmp1\*a[tmp].second,minNumberEnteredTheCastle(a,n,s-(tmp1+1),tmp+1));

}

bool compare(pair<int,int> a, pair<int,int> b)

{

    return a.first>b.first;

}

int main()

{

    int n,s;

    cin>>n>>s;

    vector<pair<int,int>> a;

    for (int i=0;i<n;i++)

    {   int tmp1,tmp2;

        cin>>tmp1>>tmp2;

        a.push\_back({tmp1,tmp2});

    }

    sort(a.begin(),a.end(),compare);

    cout<<minNumberEnteredTheCastle(a,n,s,0);

    return 0;

}

## Bài tập 12. Lược đồ

Cho một lược đồ gồm n𝑛 cột chữ nhật liên tiếp nhau có chiều rộng bằng 1 và chiều cao lần lượt là các số nguyên không âm h1,h2,…,hnℎ1,ℎ2,…,ℎ𝑛. Hãy xác định hình chữ nhật có diện tích lớn nhất có thể tạo thành từ các cột liên tiếp.

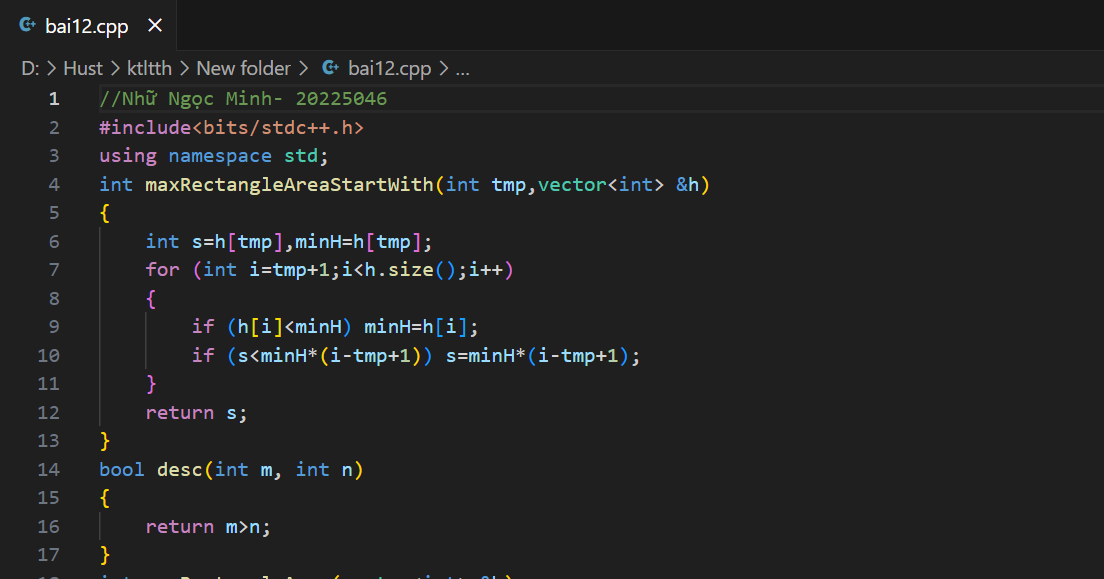
**Dữ liệu vào:**  
Dòng thứ nhất chứa số nguyên dương n𝑛 (1≤n≤1061≤𝑛≤106).  
Dòng thứ hai chứa n𝑛 số nguyên không âm h1,h2,…,hnℎ1,ℎ2,…,ℎ𝑛 cách nhau bởi dấu cách (0≤hi≤1090≤ℎ𝑖≤109).

**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:  
7  
6 2 5 4 5 1 6

Kết quả:  
12





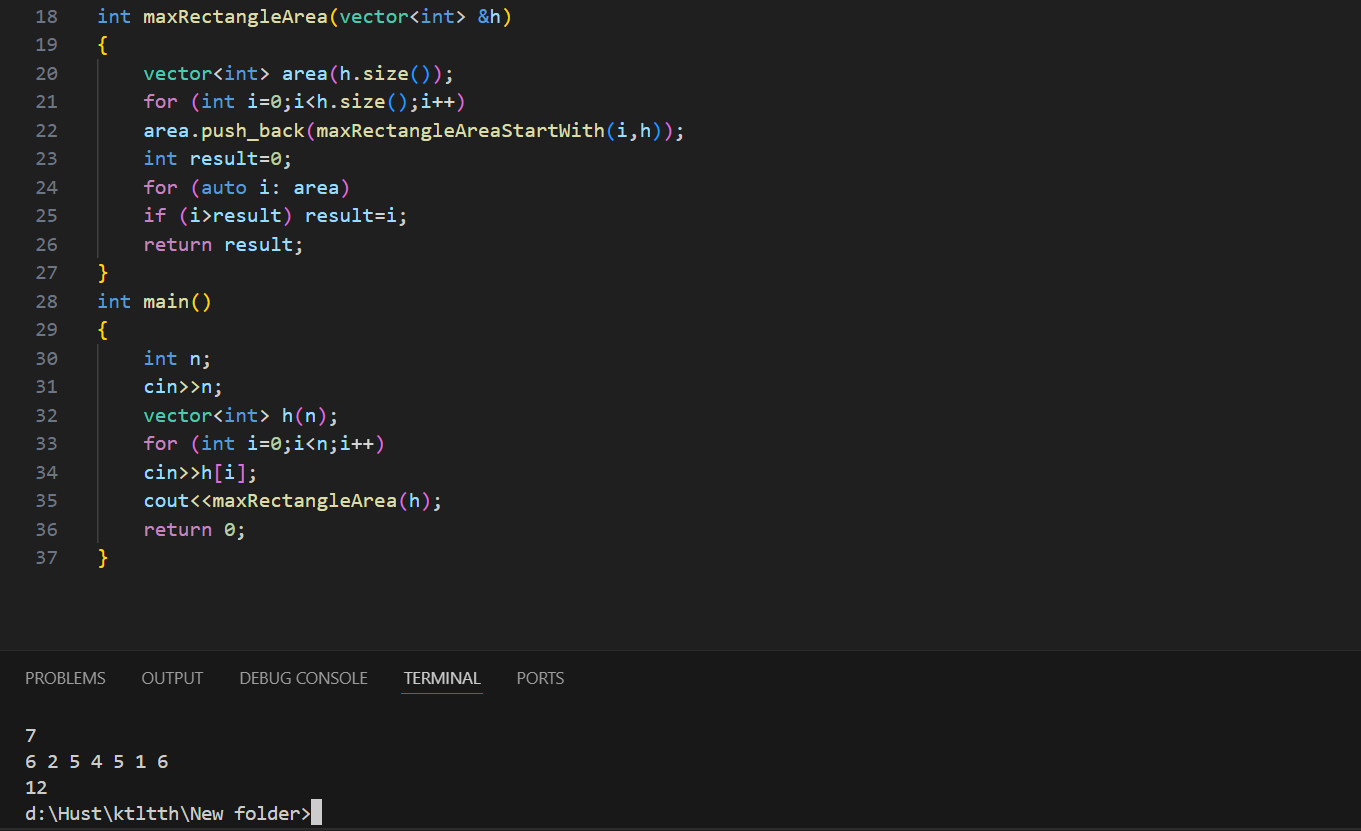


Figure 12. Lược đồ

//Nhữ Ngọc Minh- 20225046

#include<bits/stdc++.h>

using namespace std;

int maxRectangleAreaStartWith(int tmp,vector<int> &h)

{

    int s=h[tmp],minH=h[tmp];

    for (int i=tmp+1;i<h.size();i++)

    {

        if (h[i]<minH) minH=h[i];

        if (s<minH\*(i-tmp+1)) s=minH\*(i-tmp+1);

    }

    return s;

}

bool desc(int m, int n)

{

    return m>n;

}

int maxRectangleArea(vector<int> &h)

{

    vector<int> area(h.size());

    for (int i=0;i<h.size();i++)

    area.push\_back(maxRectangleAreaStartWith(i,h));

    int result=0;

    for (auto i: area)

    if (i>result) result=i;

    return result;

}

int main()

{

    int n;

    cin>>n;

    vector<int> h(n);

    for (int i=0;i<n;i++)

    cin>>h[i];

    cout<<maxRectangleArea(h);

    return 0;

}

## Bài tập 13: Đếm xâu con

Cho một xâu nhị phân độ dài n𝑛. 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 n𝑛 (1≤n≤1061≤𝑛≤106).

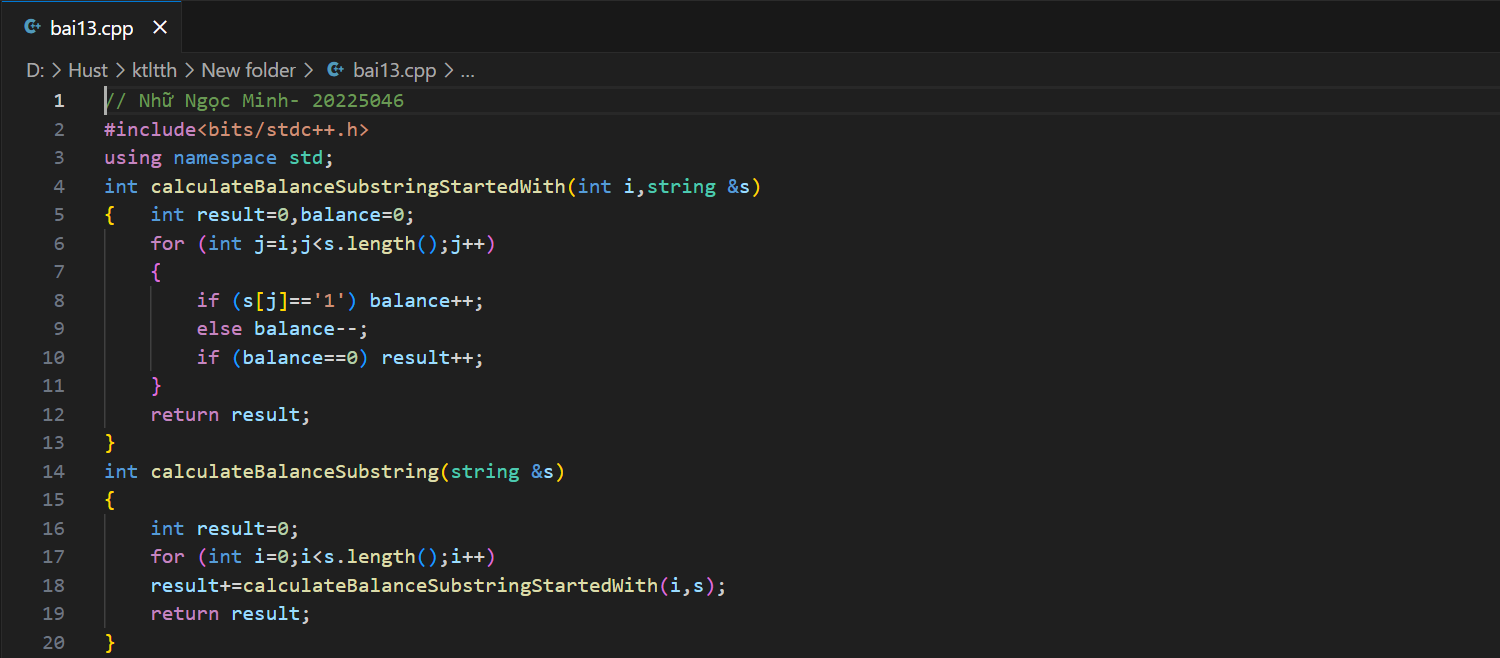
**Kết quả:**  
Ghi ra một số nguyên duy nhất là số lượng xâu con có số ký tự 0 và số ký tự 1 bằng nhau.

**Ví dụ:**

Dữ liệu vào:  
1001011

Kết quả:  
8

Giải thích: Trong ví dụ trên có 8 xâu con thỏa mãn là 10, 01, 10, 01, 1001, 0101, 100101, 001011



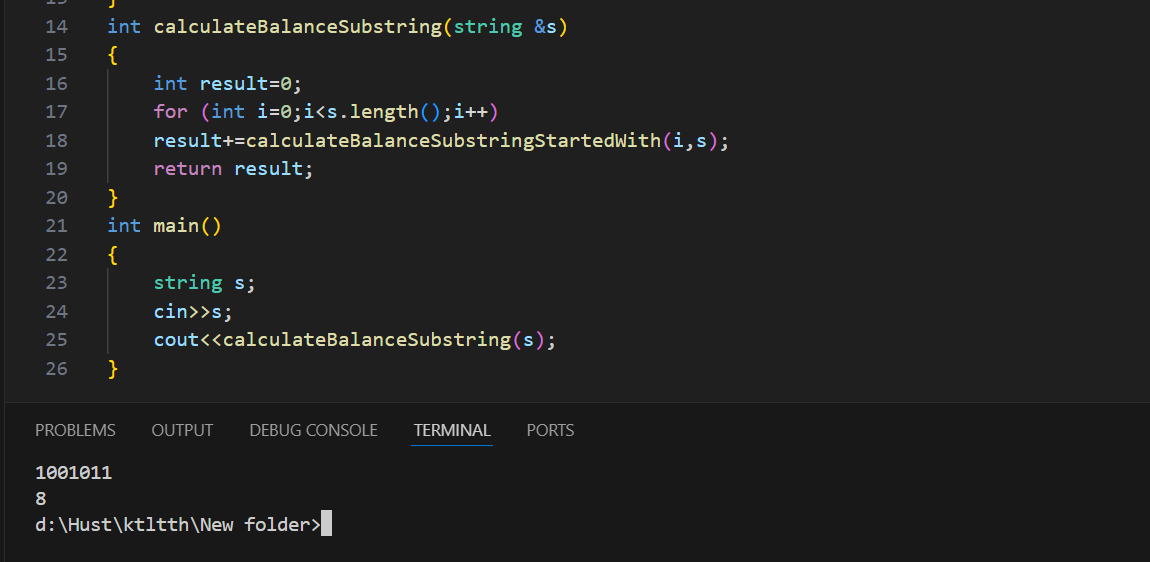


Figure 13. Đếm xâu con

// Nhữ Ngọc Minh- 20225046

#include<bits/stdc++.h>

using namespace std;

int calculateBalanceSubstringStartedWith(int i,string &s)

{   int result=0,balance=0;

    for (int j=i;j<s.length();j++)

    {

        if (s[j]=='1') balance++;

        else balance--;

        if (balance==0) result++;

    }

    return result;

}

int calculateBalanceSubstring(string &s)

{

    int result=0;

    for (int i=0;i<s.length();i++)

    result+=calculateBalanceSubstringStartedWith(i,s);

    return result;

}

int main()

{

    string s;

    cin>>s;

    cout<<calculateBalanceSubstring(s);

}