Contents

[Danh mục hình ảnh 2](#__RefHeading___Toc349_2734074527)

[Bài thực hành số 3: Đệ quy và khử đệ quy để giải quyết một số bài toán 3](#__RefHeading___Toc1901_2139985059)

[Phần 1. Bài tập thực hành 3](#__RefHeading___Toc353_2734074527)

[1. Bài tập 1: Đảo ngược một danh sách liên kết đơn 3](#__RefHeading___Toc355_2734074527)

[2. Bài tập 2: Tính diện tích tam giác 5](#__RefHeading___Toc357_2734074527)

[3. Bài tập 3: Tính diện tích có hướng của 2 vector 6](#__RefHeading___Toc359_2734074527)

[4. Bài tập 4: Thao tác với vector 7](#__RefHeading___Toc2013_1126995683)

[5. Bài tập 4: Thuật toán DFS 10](#__RefHeading___Toc2223_442281320)

[6. Bài tập 6: Thuật toán BFS 11](#__RefHeading___Toc2225_442281320)

[7. Bài tập 7: Phép giao và hợp 13](#__RefHeading___Toc2227_442281320)

[8. Bài tập 8: Phép giao và hợp của hai tập hợp 15](#__RefHeading___Toc2229_442281320)

[9. Bài tập 9: Thuật toán Dijkstra 16](#__RefHeading___Toc2231_442281320)

[Phần 2. Bài tập về nhà 19](#__RefHeading___Toc375_2734074527)

[1. Bài tập 10: Search Engine 19](#__RefHeading___Toc379_2734074527)

# Danh mục hình ảnh

[Hình 1 Bài 1.1 Đảo ngược một danh sách liên kết đơn 5](#Hình!0|sequence)

[Hình 2 Bài 1.2 Tính diện tích tam giác 6](#Hình!1|sequence)

[Hình 3 Bài 1.3 Tính diện tích có hướng của 2 vector 7](#Hình!2|sequence)

[Hình 4 Bài 1.4 Thao tác với vector 9](#Hình!3|sequence)

[Hình 5 Bài 1.5 Thuật toán DFS 11](#Hình!4|sequence)

[Hình 6 Bài 1.6 Thuật toán BFS 13](#Hình!5|sequence)

[Hình 7 Bài 1.7 Phép giao và hợp 14](#Hình!6|sequence)

[Hình 8 Bài 1.8 Phép giao và hợp của hai tập hợp 16](#Hình!7|sequence)

[Hình 9 Bài 1.9 Thuật toán dijkstra 18](#Hình!8|sequence)

[Hình 10 Bài 1.10 Search Engine 23](#Hình!9|sequence)

# Bài thực hành số 3**:** **Đệ quy và khử đệ quy để giải quyết một số bài toán**

## 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))

#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 \* newNode = new Node(data);

if (head == NULL) {

return newNode;

}

newNode->next = head;

return newNode;

}

// print the list content on a line

void print(Node\* head) {

Node \* u = head;

while(u != NULL) {

cout << u->data << " ";

u = u->next;

}

cout << endl;

}

// return the new head of the reversed list

Node\* reverse(Node\* head) {

if(head == NULL) return NULL;

Node \*prev = NULL, \*current = NULL, \*next = NULL;

current = head;

while(current != NULL){

next = current->next;

current->next = prev;

prev = current;

current = next;

}

// now let the head point at the last node (prev)

head = prev;

return head;

}

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);

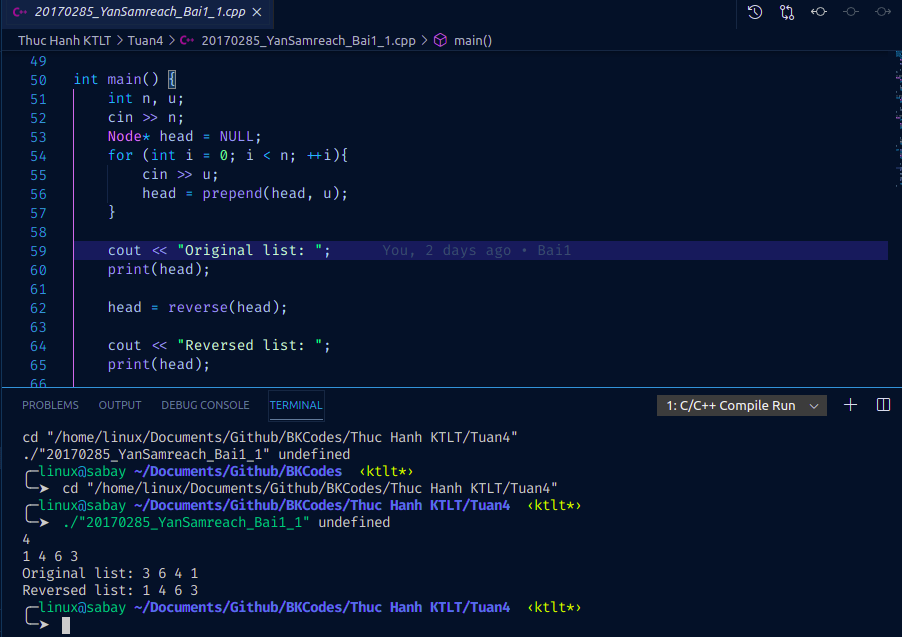
cout << "Reversed list: ";

print(head);

return 0;

}

Kết quả:



Hình 1 Bài 1.1 Đảo ngược một danh sách liên kết đơn

### 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>;

#include <iostream>

#include <cmath>

#include <iomanip>

#include <utility>

using namespace std;

using Point = pair<double, double>;

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

return 0.5 \* abs((c.first - a.first) \* (b.second - a.second) - (b.first - a.first) \* (c.second - a.second));

}

int main() {

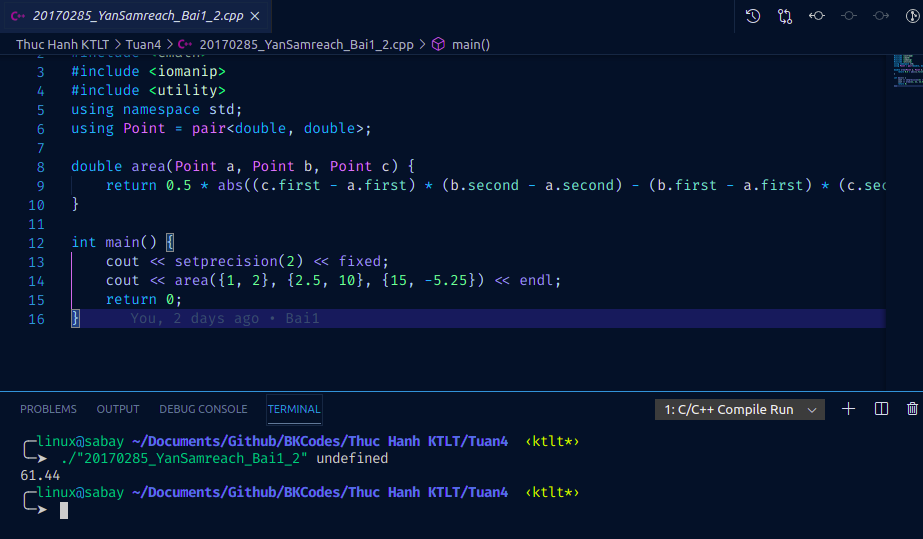
cout << setprecision(2) << fixed;

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

return 0;

}

Kết quả:



Hình 2 Bài 1.2 Tính diện tích tam giác

### Bài tập 3: **Tính diện 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>;

#include <iostream>

#include <cmath>

#include <iomanip>

using namespace std;

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

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

Vector c = { 0,0,0 };

get<0>(c) = get<1>(a) \* get<2>(b) - get<2>(a) \* get<1>(b);

get<1>(c) = get<2>(a) \* get<0>(b) - get<0>(a) \* get<2>(b);

get<2>(c) = get<0>(a) \* get<1>(b) - get<1>(a) \* get<0>(b);

return c;

}

int main() {

cout << setprecision(2) << fixed;

Vector a {1.2, 4, -0.5};

Vector b {1.5, -2, 2.5};

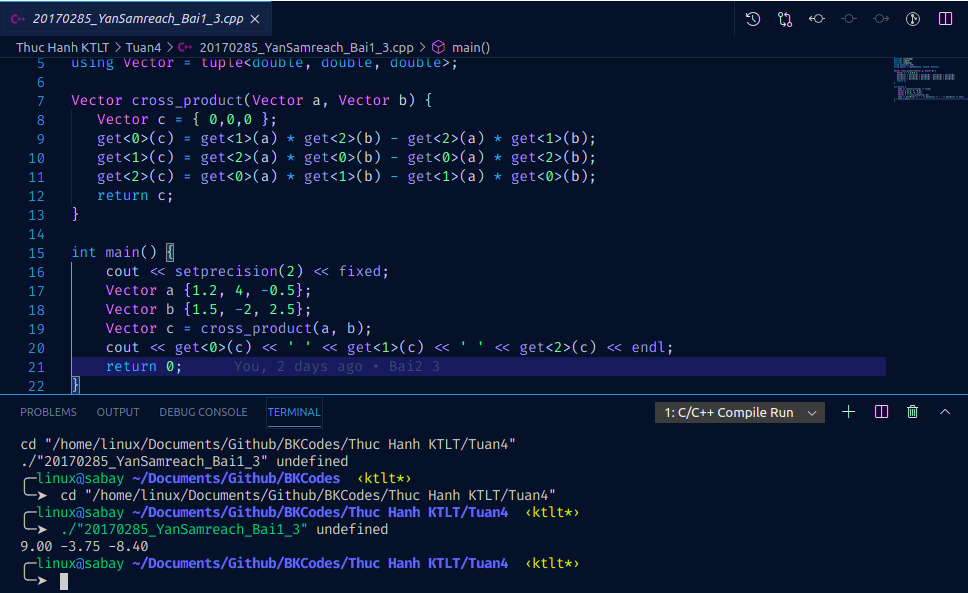
Vector c = cross\_product(a, b);

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

return 0;

}

Kết quả:



Hình 3 Bài 1.3 Tính diện tích có hướng của 2 vector

### **Bài tập 4: Thao tác với vector**

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

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

for (int v : a) if (v%2!=0) cout << v << ' ';

cout << endl;

}

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

auto new\_end = remove\_if(a.begin(), a.end(), [](int x) { return x % 2 == 0; });

a.resize(new\_end - a.begin());

}

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

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

}

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

vector<int> c;

merge(a.begin(), a.end(), b.begin(), b.end(), back\_inserter(c), [](int x, int y) { return x > y; });

return c;

}

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);

}

cout << "Odd elements of a: ";

print\_vector(a);

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);

delete\_even(a);

cout << "Odd elements of a: ";

print\_vector(a);

delete\_even(b);

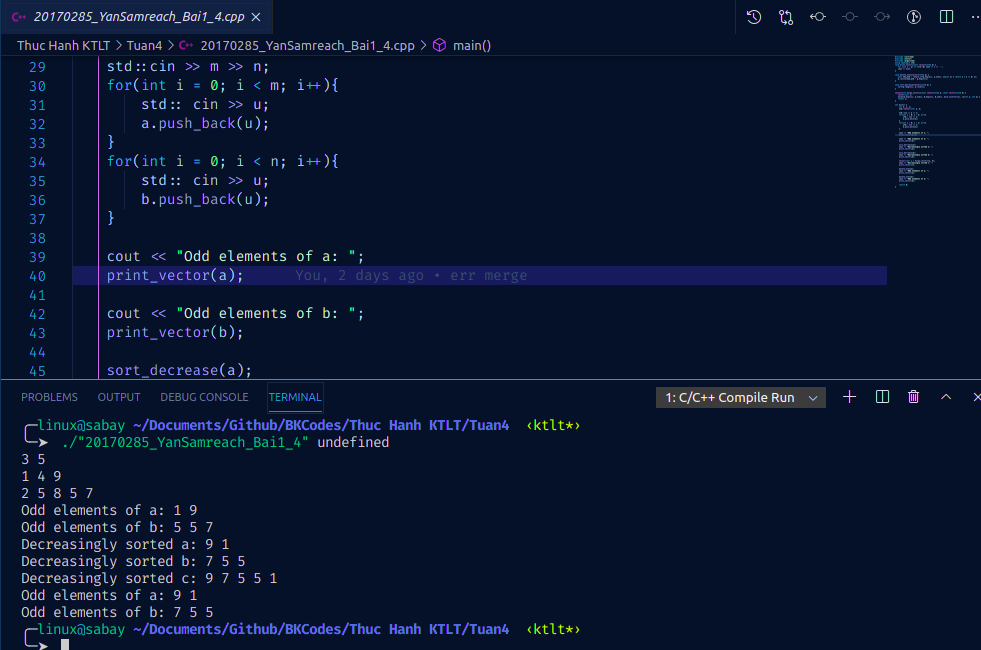
cout << "Odd elements of b: ";

print\_vector(b);

return 0;

}

Kết quả:



Hình 4 Bài 1.4 Thao tác với vector

### **Bài tập 4: Thuật toán DFS**

#include <vector>

#include <list>

#include <iostream>

#include <stack>

using namespace std;

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

int n = (int)adj.size() - 1;

stack<int> S;

vector<bool> was(n + 1);

vector<bool> printed(n + 1);

for (int start = 1; start < n; start++) {

if (was[start]) continue;

S.push(start);

was[start] = true;

while (!S.empty()) {

int u = S.top();

if (!printed[u]) {

printed[u] = true;

cout << u << endl;

}

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

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

adj[u].pop\_front();

if (was[v]) continue;

was[v] = true;

S.push(v);

} else S.pop();

}

}

}

int main() {

int n = 7;

vector< list<int> > adj;

adj.resize(n + 1);

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);

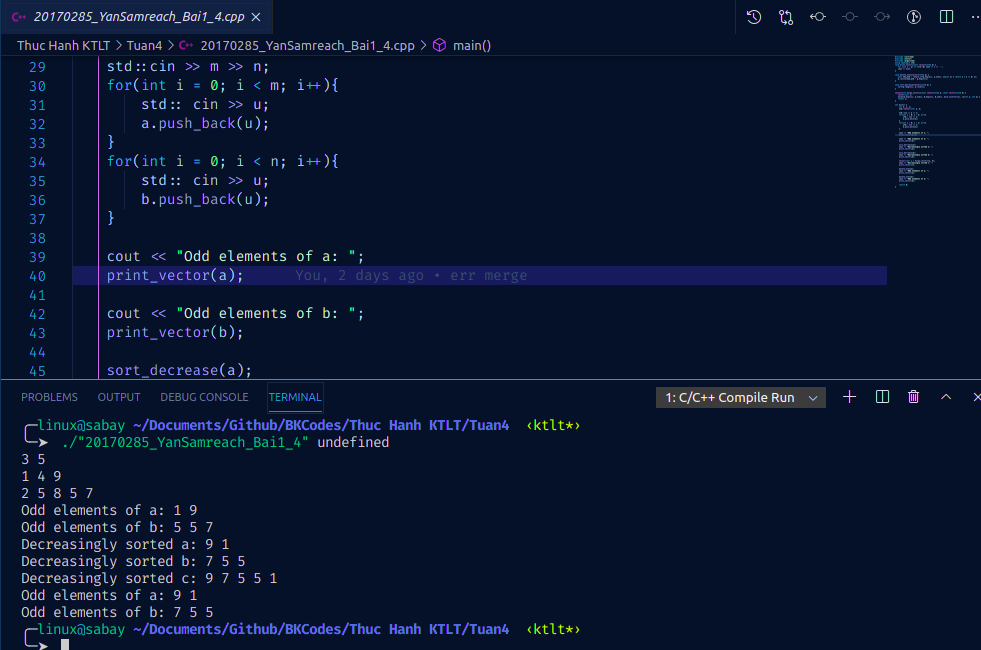
adj[2].push\_back(7);

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

dfs(adj);

}

Kết quả:



Hình 5 Bài 1.5 Thuật toán DFS

### **Bài tập 6: Thuật toán BFS**

#include <vector>

#include <list>

#include <iostream>

#include <queue>

using namespace std;

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

int n = adj.size() - 1;

vector<bool> was(n + 1);

queue<int> Q;

for (int start = 1; start < n; ++start) {

if (was[start]) continue;

Q.push(start);

was[start] = true;

while (!Q.empty()) {

int u = Q.front();

cout << u << endl;

Q.pop();

for (int v : adj[u]) {

if (was[v]) continue;

was[v] = true;

Q.push(v);

}

}

}

}

int main() {

int n = 7;

vector< list<int> > adj;

adj.resize(n + 1);

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);

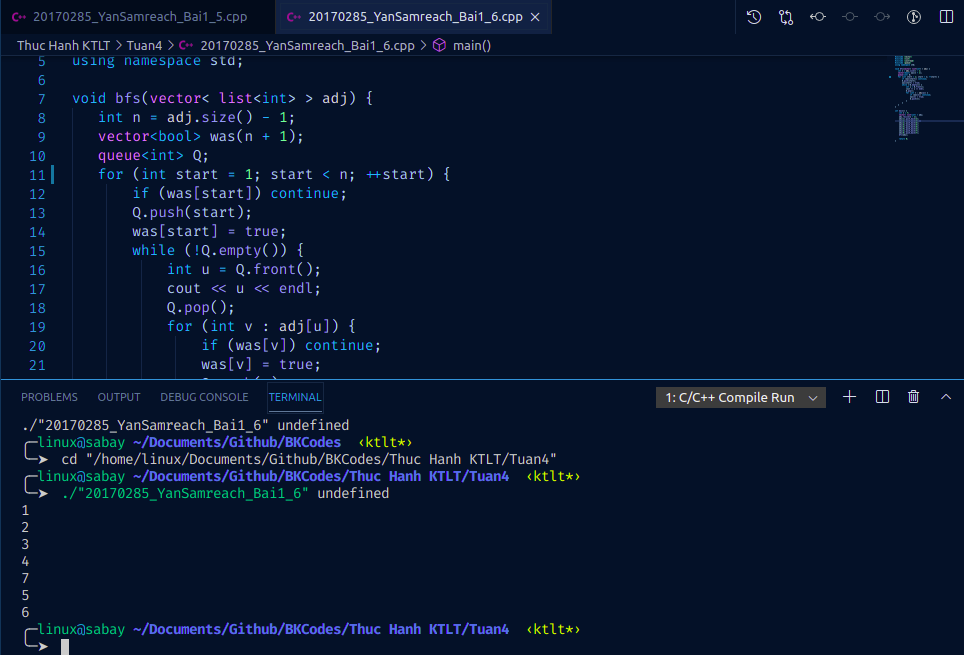
adj[6].push\_back(7);

bfs(adj);

return 0;

}

Kết quả:



Hình 6 Bài 1.6 Thuật toán BFS

### **Bài tập 7: Phép giao và hợp**

Viết các hàm thực hiện các phép giao và hợp của hai tập hợp được 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) {

set<T> c = a;

for (int 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 (int v : a) if (b.count(v))

c.insert(v);

return c;

}

template<class T>

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

for (const T& x : a) {

cout << x << ' ';

}

cout << endl;

}

int main() {

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

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

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

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

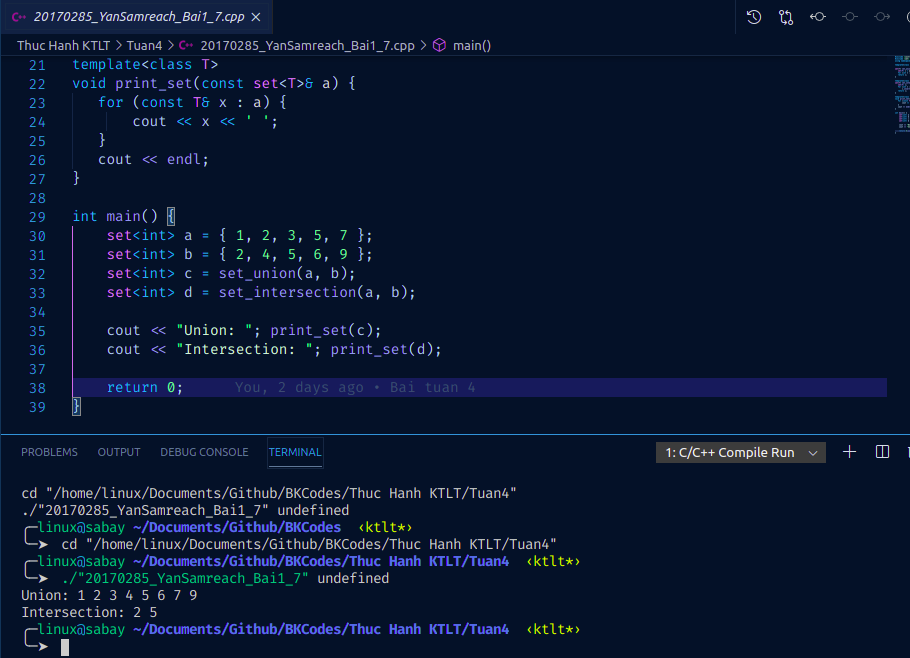
cout << "Union: "; print\_set(c);

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

return 0;

}

Kết quả:



Hình 7 Bài 1.7 Phép giao và hợp

### **Bài tập 8: Phép giao và hợp của hai tập hợp**

#include <iostream>

#include <map>

#include<algorithm>

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> c = a;

for (auto it : b) {

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

auto it2 = c.find(it.first);

if (it2 == c.end()) {

c[it.first] = it.second;

} else {

it2->second = max(it2->second, it.second);

}

}

return c;

}

template<class T>

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

map<T, double> c;

for (auto it : a) {

double v = it.second;

auto it2 = b.find(it.first);

if (it2 != b.end()) {

v = min(v, it2->second);

c[it.first] = v;

}

}

return c;

}

template<class T>

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

cout << "{ ";

for (const auto& x : a) {

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

}

cout << "}" << endl;

}

int main() {

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

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

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

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

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

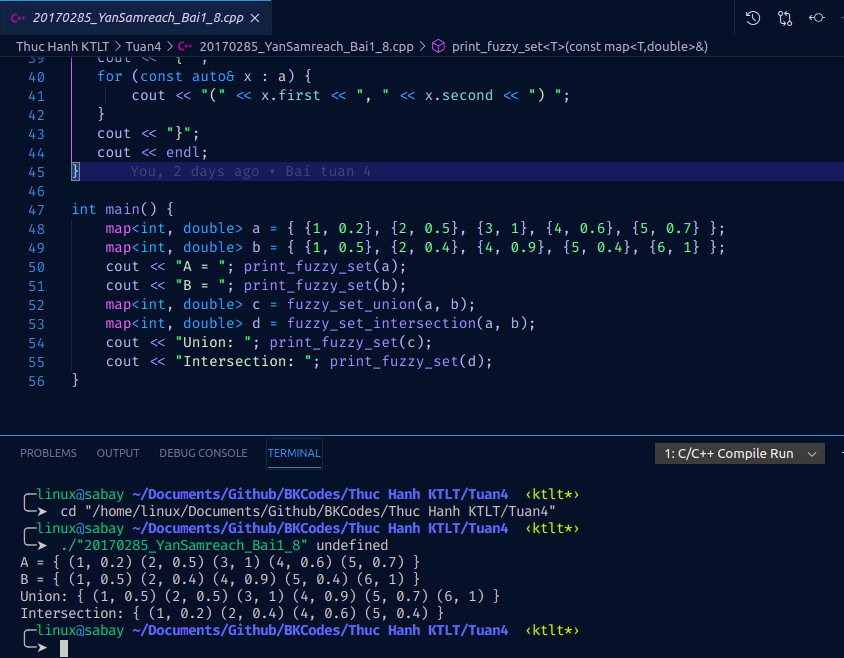
map<int, double> d = fuzzy\_set\_intersection(a, b);

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

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

}

Kết quả:



Hình 8 Bài 1.8 Phép giao và hợp của hai tập hợp

### **Bài tập 9: Thuật toán Dijkstra**

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.

#include <iostream>

#include <queue>

#include <climits>

using namespace std;

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

priority\_queue< pair<int, int> > Q; // {-dist, u}

vector<int> dist(adj.size(), (int)1e9);

dist[0] = 0;

Q.push({ 0, 0 });

while (!Q.empty()) {

int u = Q.top().second;

int du = -Q.top().first;

Q.pop();

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

for (auto e : adj[u]) {

int v = e.first;

int w = e.second;

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

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

Q.push({ -dist[v], 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) {

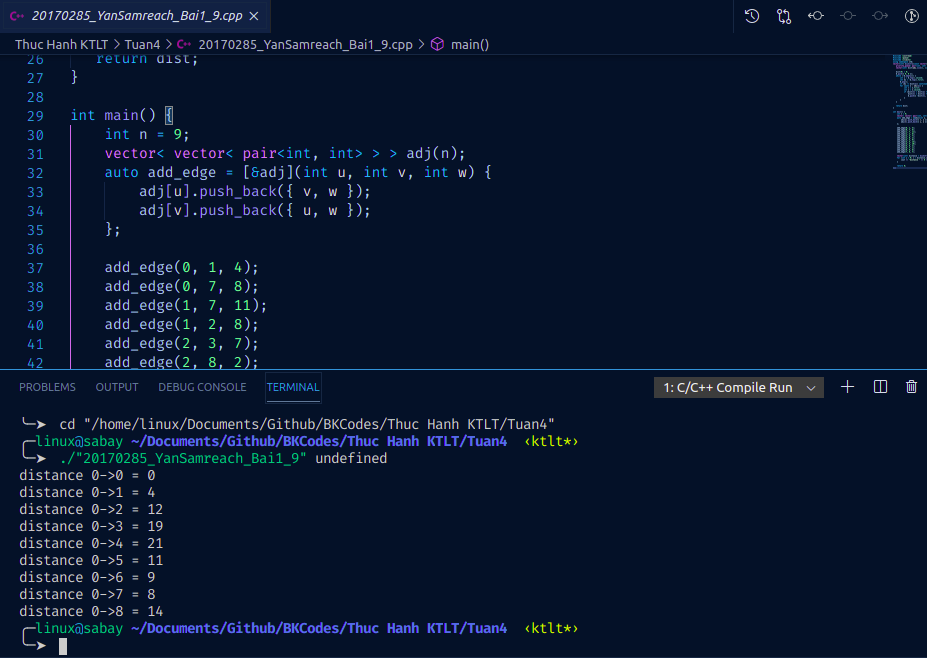
cout << "distance " << 0 << "->" << i << " = " << distance[i] << endl;

}

return 0;

}

Kết quả:



Hình 9 Bài 1.9 Thuật toán dijkstra

## Phần 2. **Bài tập về nhà**

### Bài tập 10: **Search Engine**

*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:

* f(t,d) là số lần xuất hiện của từ t trong văn bản d
* maxf(d) là giá trị lớn nhất của f(t,d) với mọi t
* df(t) là số văn bản chứa từ t
* TF(t,d)=0.5+0.5⋅f(t,d)maxf(t,d)
* IDF(t)=log2(Ndf(t))
* Điểm số của từ t trong văn bản d là score(t,d)=TF(t,d)⋅IDF(t), nếu từ t không xuất hiện trong văn bản d thì score(t,d)=0.
* Điểm số của văn bản d đối với truy vấn gồm các từ (có thể trùng nhau) t1,t2,...,tq là∑qi=1score(ti,d)

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.

**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≤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

*#include<iostream>*

#include<stdio.h>

#include<string.h>

#include<math.h>

#pragma warning(disable:4996)

using namespace std;

int N, Q;

void pop\_string(char \*\*\*save,char \*\*vb, int arr[1000],int M) {

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

int j = 0;

save[i][j] = strtok(vb[i], ",");

while (save[i][j] != NULL) {

j++;

save[i][j] = strtok(NULL, ",");

}

arr[i] = j; //so tu trong van ban

}

}

inline int calc\_f(char\* word, char\*\* doc ,int m){

int count = 0;

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

if (strcmp(word, doc[i]) == 0) count++;

}

return count;

}

int calc\_df(char\* word, char\*\*\* save, int arr[1000]) {

int index = 0;

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

for (int j = 0; j < arr[i]; j++) {

if (strcmp(word, save[i][j]) == 0) {

index++;

break;

}

}

}

return index;

}

int calc\_maxf(char\*\* word, char \*\*doc, int m) {

int max = 0;

int k = 0;

char sv[10];

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

if (strcmp(sv, doc[i]) == 0)

continue;

k = calc\_f(doc[i], doc, m);

if (k > max) {

max = k;

strcpy(sv, doc[i]);

}

}

return max;

}

float score(char\*\* word, char\*\* doc, int m, int n, char \*\*\*save, int arr[1000]) {

float score = 0;

float maxf = calc\_maxf(word, doc, m);

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

float f = calc\_f(word[i], doc, m);

if (f == 0)

continue;

float df = calc\_df(word[i], save, arr);

if (df == 0)

continue;

score += (0.5 + 0.5 \* f / maxf) \* log((float)(N) / df) / log(2.0);

}

return score;

}

int main() {

cin >> N;

char \*\*\*saveText;

char \*\*\*saveQuery;

int numText[1000];// luu so tu trong 1 van ban

int numQuery[1000];// luu so tu trong 1 truy van

char\*\* text;

char\*\* query;

saveText = new char\*\* [N];

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

saveText[i] = new char\* [1000];

for (int j = 0; j < 1000; j++) {

saveText[i][j] = new char[10];

}

}

text = new char\* [N];

getchar();

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

text[i] = new char[11000];

cin >> text[i];

}

cin >> Q;

getchar();

saveQuery = new char\*\* [Q];

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

saveQuery[i] = new char\* [1000];

for (int j = 0; j < 1000; j++) {

saveQuery[i][j] = new char[10];

}

}

query = new char\* [Q];

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

query[i] = new char[110];

cin >> query[i];

}

pop\_string(saveText, text, numText,N);

pop\_string(saveQuery, query, numQuery,Q);

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

float max = 0;

float mark = 0;

int index = 0;

for (int j = 0; j < N; j++) {

mark = score(saveQuery[i], saveText[j], numText[j], numQuery[i], saveText,numText);

if (mark > max) {

max = mark;

index = j+1;

}

}

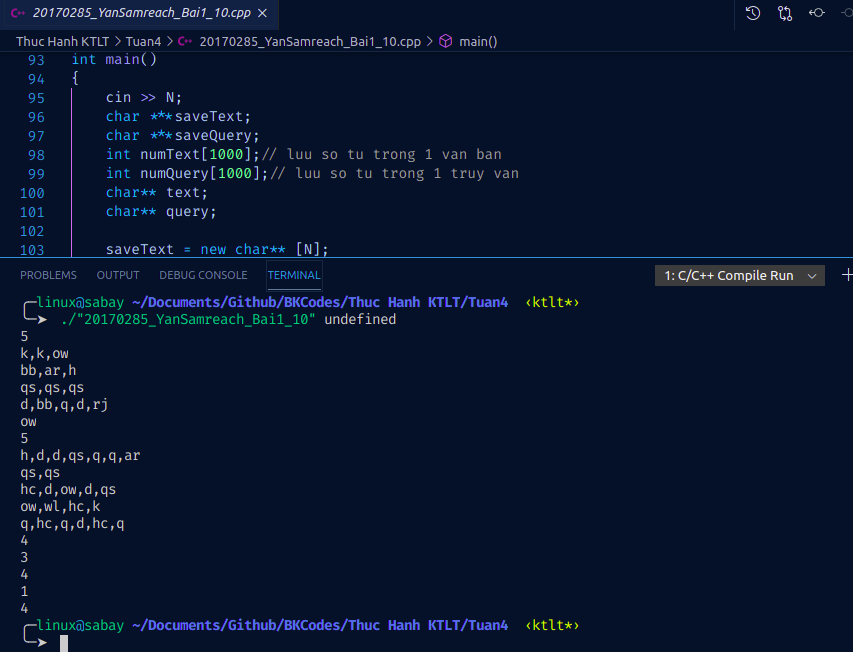
cout << index << endl;

}

return 0;

}

Kết quả:



Hình 10 Bài 1.10 Search Engine