**BÀI 3: TÌM KIẾM TRÊN ĐỒ THỊ {1/2}**

**\*Thực hành 1a**

**Bài 1. Các đỉnh liên thông với x (LienThongBFS.\*)**

## 🔹 Phân tích đề

* Input:
  + Dòng đầu: số đỉnh n và đỉnh xuất phát x.
  + n dòng tiếp: danh sách kề của từng đỉnh (kết thúc bằng -1).
* Output:
  + Số lượng đỉnh có thể đi tới từ x.
  + Danh sách các đỉnh tìm được bằng BFS.

## 🔹 Ý tưởng BFS

1. Bắt đầu từ đỉnh x, dùng BFS duyệt toàn bộ đỉnh có thể đi tới.
2. Đánh dấu visited[].
3. Thu thập kết quả.

----------------------------------------------------------------------

#include <iostream>

#include <vector>

#include <queue>

using namespace std;

int main() {

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

cin.tie(nullptr);

int n, x;

cin >> n >> x;

vector<vector<int>> adj(n + 1);

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

while (true) {

int v;

cin >> v;

if (v == -1) break;

adj[i].push\_back(v);

}

}

vector<int> visited(n + 1, 0);

vector<int> result;

queue<int> q;

visited[x] = 1;

q.push(x);

while (!q.empty()) {

int u = q.front();

q.pop();

for (int v : adj[u]) {

if (!visited[v]) {

visited[v] = 1;

q.push(v);

result.push\_back(v); // chỉ lưu những đỉnh tìm được, KHÔNG lưu x

}

}

}

cout << result.size() << "\n";

for (int v : result) {

cout << v << " ";

}

cout << "\n";

return 0;

}

/\*

8 7

2 -1

1 5 -1

4 -1

3 7 -1

2 -1

-1

4 8 -1

7 -1

\*/

**Bài 2. Tìm đường đi (DuongDiBFS.\*)**

## 🔹 Phân tích

* Input:
  + n số đỉnh.
  + Hai đỉnh x, y.
  + Sau đó n dòng danh sách kề (mỗi dòng kết thúc bằng -1).
* Yêu cầu:
  + Tìm đường đi ngắn nhất từ x đến y bằng **BFS**.
  + In ra số đỉnh trên đường đi + liệt kê đường đi.

## 🔹 Ý tưởng BFS

1. Dùng mảng parent[] để lưu cha của mỗi đỉnh trong BFS.
2. Khi BFS từ x, nếu gặp y thì dừng.
3. Truy vết ngược từ y → x theo parent[] để lấy đường đi.
4. In kết quả.

#include <iostream>

#include <vector>

#include <queue>

#include <algorithm>

using namespace std;

int n, x, y;

vector<vector<int>> adj;

vector<bool> visited;

vector<int> parent;

bool bfs(int start, int target) {

queue<int> q;

visited[start] = true;

q.push(start);

while (!q.empty()) {

int u = q.front(); q.pop();

if (u == target) return true; // tìm thấy y

for (int v : adj[u]) {

if (!visited[v]) {

visited[v] = true;

parent[v] = u;

q.push(v);

}

}

}

return false;

}

int main() {

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

cin.tie(nullptr);

cin >> n >> x >> y;

adj.assign(n + 1, {});

visited.assign(n + 1, false);

parent.assign(n + 1, -1);

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

while (true) {

int v;

cin >> v;

if (v == -1) break;

adj[i].push\_back(v);

}

}

if (!bfs(x, y)) {

cout << "0\n"; // không có đường đi

return 0;

}

// Truy vết đường đi từ y ngược về x

vector<int> path;

for (int cur = y; cur != -1; cur = parent[cur]) {

path.push\_back(cur);

if (cur == x) break;

}

reverse(path.begin(), path.end());

cout << path.size() << "\n";

for (int v : path) cout << v << " ";

cout << "\n";

return 0;

}

/\*

8 3 8

2 -1

1 5 -1

4 -1

3 7 -1

2 -1

-1

4 8 -1

7 -1

\*/

## **3. Liệt kê thành phần liên thông**

## 🔹 Phân tích

* Input:
  + Dòng đầu: số đỉnh n.
  + n dòng tiếp theo: danh sách kề của mỗi đỉnh (kết thúc bằng -1).
* Output:
  + Số lượng thành phần liên thông k.
  + Sau đó in ra các đỉnh thuộc từng thành phần liên thông.
* Dùng **BFS**:
  + Duyệt từng đỉnh chưa thăm.
  + Thực hiện BFS để gom thành một thành phần liên thông.
  + Lưu kết quả vào mảng components.

-------------------------------------------------------------------------------------------

#include <iostream>

#include <vector>

#include <queue>

using namespace std;

int n;

vector<vector<int>> adj;

vector<bool> visited;

vector<int> bfs(int start) {

vector<int> comp;

queue<int> q;

q.push(start);

visited[start] = true;

while (!q.empty()) {

int u = q.front(); q.pop();

comp.push\_back(u);

for (int v : adj[u]) {

if (!visited[v]) {

visited[v] = true;

q.push(v);

}

}

}

return comp;

}

int main() {

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

cin.tie(nullptr);

cin >> n;

adj.assign(n + 1, {});

visited.assign(n + 1, false);

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

while (true) {

int v;

cin >> v;

if (v == -1) break;

adj[i].push\_back(v);

}

}

vector<vector<int>> components;

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

if (!visited[i]) {

components.push\_back(bfs(i));

}

}

cout << components.size() << "\n";

for (auto& comp : components) {

for (int v : comp) cout << v << " ";

cout << "\n";

}

return 0;

}

**Bài 4. Cạnh cầu và đỉnh khớp**

**🔹 Ý tưởng giải**

1. **Đếm số thành phần liên thông ban đầu bằng BFS.**
2. **Kiểm tra cạnh cầu:**
   * Bỏ cạnh (x,y)(x,y)(x,y) khỏi đồ thị.
   * Đếm lại số thành phần liên thông.
   * Nếu số thành phần tăng → (x,y) là **cạnh cầu**.
3. **Kiểm tra đỉnh khớp:**
   * Bỏ đỉnh zzz (không cho BFS đi qua nó).
   * Đếm lại số thành phần liên thông.
   * Nếu số thành phần tăng → zzz là **đỉnh khớp**.

**-----------------------------------------------------------------------------**

#include <iostream>

#include <vector>

#include <queue>

using namespace std;

int n, x, y, z;

vector<vector<int>> adj;

vector<bool> visited;

void bfs(int start, int skipNode, int skipU = -1, int skipV = -1) {

queue<int> q;

q.push(start);

visited[start] = true;

while (!q.empty()) {

int u = q.front();

q.pop();

for (int v : adj[u]) {

if (v == skipNode) continue;

if ((u == skipU && v == skipV) || (u == skipV && v == skipU)) continue;

if (!visited[v]) {

visited[v] = true;

q.push(v);

}

}

}

}

int countComponents(int skipNode = -1, int skipU = -1, int skipV = -1) {

visited.assign(n + 1, false);

int cnt = 0;

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

if (i == skipNode) continue;

if (!visited[i]) {

cnt++;

bfs(i, skipNode, skipU, skipV);

}

}

return cnt;

}

int main() {

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

cin.tie(nullptr);

cin >> n >> x >> y >> z;

adj.assign(n + 1, {});

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

while (true) {

int v;

cin >> v;

if (v == -1) break;

adj[i].push\_back(v);

}

}

// Đếm số thành phần ban đầu

int initialCC = countComponents();

// Kiểm tra cạnh cầu

int withoutEdgeCC = countComponents(-1, x, y);

if (withoutEdgeCC > initialCC)

cout << "canh cau\n";

else

cout << "khong la canh cau\n";

// Kiểm tra đỉnh khớp

int withoutNodeCC = countComponents(z);

if (withoutNodeCC > initialCC)

cout << "dinh khop\n";

else

cout << "khong la dinh khop\n";

return 0;

}

**Thực hành 1b**

**Bài 1. Các đỉnh liên thông với x (LienThongDFS.\*)**

## Ý tưởng

Tìm tất cả các đỉnh liên thông với đỉnh **x** trong đồ thị vô hướng.

### Hướng tiếp cận

1. **Mô hình hóa đồ thị**
   * Dùng **danh sách kề**: adj[i] chứa các đỉnh kề với đỉnh i.
   * Vì đồ thị **vô hướng**, khi đọc cạnh (i, j) thì phải thêm cả j vào adj[i] và i vào adj[j].
2. **Duyệt DFS (hoặc BFS)**
   * Tạo mảng visited[n+1] để đánh dấu các đỉnh đã thăm.
   * Bắt đầu từ đỉnh x, duyệt tất cả các đỉnh có thể đi được bằng **DFS** hoặc **BFS**.
   * Trong quá trình duyệt, lưu lại các đỉnh đã thăm vào một danh sách result.
3. **Kết quả**
   * Số lượng đỉnh liên thông với x: chính là result.size().
   * Các đỉnh liên thông: in ra theo thứ tự duyệt.

#include <iostream>

#include <vector>

#include <cstring>

using namespace std;

const int MAXN = 100005;

vector<int> adj[MAXN];

bool visited[MAXN];

vector<int> result;

int n, x;

void dfs(int u) {

visited[u] = true;

result.push\_back(u);

for (int v : adj[u]) {

if (!visited[v]) {

dfs(v);

}

}

}

int main() {

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

cin.tie(0);

cin >> n >> x;

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

while (true) {

int v;

cin >> v;

if (v == -1) break;

adj[i].push\_back(v);

adj[v].push\_back(i); // vì đồ thị vô hướng

}

}

memset(visited, false, sizeof(visited));

dfs(x);

cout << result.size() << "\n";

for (int v : result) cout << v << " ";

cout << "\n";

return 0;

}

**Bài 2. Tìm đường đi (DuongDiDFS.\*)**

## Ý tưởng

1. Đọc vào số đỉnh n, hai đỉnh x, y.
2. Dùng **danh sách kề** để lưu đồ thị.
3. Dùng **DFS** từ x, lưu cha của mỗi đỉnh.
4. Khi tìm thấy y, dừng lại.
5. Truy ngược cha từ y về x để dựng lại đường đi.

#include <iostream>

#include <vector>

#include <queue>

#include <cstring>

using namespace std;

const int MAXN = 100005;

vector<int> adj[MAXN];

bool visited[MAXN];

int parent[MAXN];

int n, x, y;

void bfs(int start) {

queue<int> q;

q.push(start);

visited[start] = true;

parent[start] = -1;

while (!q.empty()) {

int u = q.front();

q.pop();

if (u == y) return; // dừng nếu đã tới y

for (int v : adj[u]) {

if (!visited[v]) {

visited[v] = true;

parent[v] = u;

q.push(v);

}

}

}

}

int main() {

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

cin.tie(0);

cin >> n >> x >> y;

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

while (true) {

int v;

cin >> v;

if (v == -1) break;

adj[i].push\_back(v);

adj[v].push\_back(i); // thêm cạnh ngược lại vì đồ thị vô hướng

}

}

memset(visited, false, sizeof(visited));

memset(parent, -1, sizeof(parent));

bfs(x);

if (!visited[y]) {

cout << 0 << "\n"; // không có đường đi

return 0;

}

// dựng lại đường đi từ y về x

vector<int> path;

for (int cur = y; cur != -1; cur = parent[cur]) {

path.push\_back(cur);

}

cout << path.size() << "\n";

for (int i = path.size() - 1; i >= 0; i--) {

cout << path[i];

if (i > 0) cout << " ";

}

cout << "\n";

return 0;

}

**Bài 3. Đồ thị phân đôi (DoThiPhanDoi.\*)**

## Ý tưởng chính

* Bản chất của **đồ thị phân đôi** = có thể tô màu đồ thị bằng **2 màu** mà không có 2 đỉnh kề nhau cùng màu.
* Thuật toán cơ bản:
  1. Chọn một đỉnh chưa được tô màu → tô màu (0).
  2. Duyệt BFS hoặc DFS từ đỉnh đó.
  3. Với mỗi cạnh (u,v)(u, v)(u,v):
     + Nếu vvv chưa được tô màu → tô bằng màu ngược lại của uuu.
     + Nếu vvv đã được tô màu mà **bằng màu với uuu** → **mâu thuẫn → không phân đôi**.
  4. Nếu duyệt xong toàn bộ đồ thị mà không mâu thuẫn → **đồ thị phân đôi**.

#include <iostream>

#include <vector>

#include <queue>

using namespace std;

bool isBipartite(int n, vector<vector<int>>& adj) {

vector<int> color(n + 1, -1); // -1 = chưa tô, 0 và 1 là 2 màu

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

if (color[start] == -1) { // Nếu thành phần liên thông chưa duyệt

queue<int> q;

q.push(start);

color[start] = 0;

while (!q.empty()) {

int u = q.front(); q.pop();

for (int v : adj[u]) {

if (color[v] == -1) {

color[v] = 1 - color[u]; // tô màu đối lập

q.push(v);

}

else if (color[v] == color[u]) {

return false; // xung đột màu → không phân đôi

}

}

}

}

}

return true;

}

int main() {

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

cin.tie(nullptr);

int n, m;

cout << "Nhap so dinh va so canh: ";

cin >> n >> m;

vector<vector<int>> adj(n + 1);

cout << "Nhap cac canh (u v), ket thuc bang -1 -1:\n";

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

int u, v;

cin >> u >> v;

if (u < 1 || v < 1 || u > n || v > n) {

continue;

}

// Nếu đồ thị vô hướng:

adj[u].push\_back(v);

adj[v].push\_back(u);

// Nếu đồ thị có hướng thì chỉ cần:

// adj[u].push\_back(v);

}

if (isBipartite(n, adj)) {

cout << "Do thi phan doi\n";

}

else {

cout << "Do thi khong phan doi\n";

}

return 0;

}

**Bài 4. DFS Không đệ quy**

## 1. Ý tưởng chính

* Bình thường DFS hay viết bằng **đệ quy**. Nhưng có thể thay thế lời gọi đệ quy bằng một **ngăn xếp (stack)** để quản lý các đỉnh cần duyệt.
* Thuật toán:
  1. Bắt đầu từ đỉnh start.
  2. Đánh dấu start là đã thăm và đẩy nó vào stack.
  3. Trong khi stack chưa rỗng:
     + Lấy đỉnh trên cùng ra (giống như quay lui đệ quy).
     + In nó ra.
     + Duyệt tất cả các đỉnh kề v của nó, nếu v chưa được thăm → đánh dấu và đưa vào stack.
  4. Kết thúc khi stack rỗng.

#include <iostream>

#include <vector>

#include <stack>

using namespace std;

void DFS\_iterative(int start, vector<vector<int>>& adj, int n) {

vector<bool> visited(n + 1, false);

stack<int> st;

st.push(start);

visited[start] = true;

cout << "Thu tu DFS: ";

while (!st.empty()) {

int u = st.top();

st.pop();

cout << u << " ";

// Duyệt ngược để thứ tự giống DFS đệ quy

for (int i = (int)adj[u].size() - 1; i >= 0; --i) {

int v = adj[u][i];

if (!visited[v]) {

visited[v] = true;

st.push(v);

}

}

}

cout << "\n";

}

int main() {

int n, start;

cout << "Nhap so dinh va dinh bat dau: ";

cin >> n >> start;

vector<vector<int>> adj(n + 1);

cout << "Nhap danh sach ke (ket thuc moi dong bang -1):\n";

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

while (true) {

int v;

cin >> v;

if (v == -1) break;

adj[i].push\_back(v);

}

}

DFS\_iterative(start, adj, n);

system("pause");

return 0;

}