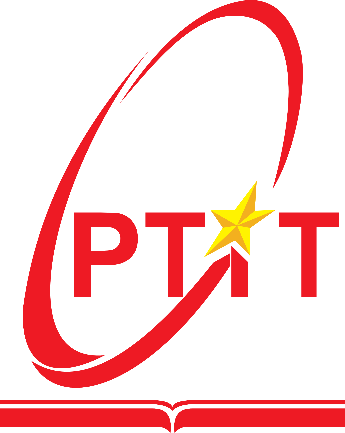
HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG CƠ SỞ TẠI

THÀNH PHỐ HỒ CHÍ MINH

KHOA CÔNG NGHỆ THÔNG TIN 2



MÔN HỌC :

CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

Giảng viên : Lê Văn Hạnh

GIẢI BÀI TOÁN GHÉP CẶP TRÊN ĐỒ THỊ (MATCHING) SỬ DỤNG THUẬT TOÁN HOPCROFT-KARP

Sinh viên thực hiện:

Huỳnh Thị Hồng Thắm N23DCCN055 <[n23dccn055@student.ptithcm.edu.vn](mailto:n23dccn055@student.ptithcm.edu.vn?subject=mail)>

MỤC LỤC

[I/ TỔNG QUAN VỀ ĐỀ TÀI 3](#_Toc199757928)

[1.1 Giới thiệu về Ghép cặp trên đồ thị (Matching) 3](#_Toc199757929)

[1.2 Cơ sở lý thuyết 3](#_Toc199757930)

[1.2.1 Đồ thị (Graph) 3](#_Toc199757931)

[1.2.2 Đồ thị 2 phía (Bipartite Graph) 4](#_Toc199757932)

[1.2.2.1 Thuật toán kiểm tra 4](#_Toc199757933)

[1.2.2.2 Định lí đồ thị hai phía 5](#_Toc199757934)

[1.2.3 Danh sách kề (Adjacency list) 5](#_Toc199757935)

[1.2.4 Đường tăng luồng (Augmenting path) 6](#_Toc199757936)

[1.2.5 BFS (Breadth-First Search) 7](#_Toc199757937)

[1.2.6 DFS (Depth-First Search) 8](#_Toc199757938)

[1.2.7 Matching(Cặp ghép) 9](#_Toc199757939)

[II/ THIẾT KẾ THUẬT TOÁN 11](#_Toc199757940)

[2.1 Thuật toán Hopcroft-Karp 11](#_Toc199757941)

[2.2 Thiết kế 12](#_Toc199757942)

[III/ TRIỂN KHAI THUẬT TOÁN 13](#_Toc199757943)

[3.1 Viết chương trình C++ 13](#_Toc199757944)

[3.2 Kết quả thử 17](#_Toc199757945)

[IV/ KẾT LUẬN 18](#_Toc199757946)

[Tài liệu tham khảo 19](#_Toc199757947)

DANH SÁCH HÌNH ẢNH

[Hình 1: Đồ thị hai phía 4](#_Toc199757910)

[Hình 2: Không phải đồ thị hai phía 4](#_Toc199757911)

[Hình 3: Thuật toán kiểm tra đồ thị liên thông là đồ thị hai phía 4](#_Toc199757912)

[Hình 4: Danh sách kề 6](#_Toc199757913)

[Hình 5: Phân biệt BFS và DFS 9](#_Toc199757914)

[Hình 6: Cặp cạnh ghép trên đồ thị vô hướng 10](#_Toc199757915)

[Hình 7: Đường tăng luông trên đồ thị vô hướng 11](#_Toc199757916)

[Hình 8: Đầu vào bài toán ghép cặp đồ thị hai phía 17](#_Toc199757917)

[Hình 9: Kết quả ghép cặp 17](#_Toc199757918)

I/ TỔNG QUAN VỀ ĐỀ TÀI

## 1.1 Giới thiệu về Ghép cặp trên đồ thị (Matching)

Bài toán ghép cặp trên đồ thị là một trong những bài toán quan trọng trong lý thuyết đồ thị, với nhiều ứng dụng thực tế trong phân công công việc, ghép cặp sinh viên với đề tài nghiên cứu, cũng như tìm cặp phù hợp trong hệ thống gợi ý.

Khái niệm cơ bản: Ghép cặp (Matching) trong đồ thị là một tập hợp các cạnh sao cho không có hai cạnh nào có chung một đỉnh. Nếu đồ thị là đồ thị lưỡng phân (bipartite graph), nghĩa là tập đỉnh có thể chia thành hai nhóm độc lập, thì bài toán tìm matching cực đại (Maximum Bipartite Matching) rất hữu ích trong việc ghép nối các đối tượng giữa hai nhóm.

Thuật toán giải quyết ghép cặp sinh viên – đề tài : Hopcroft-Karp

Ứng dụng thực tế

1. Phân công công việc: Trong một công ty, các nhân viên có những kỹ năng khác nhau và cần được ghép với các công việc phù hợp. Bài toán này có thể được mô hình hóa bằng một đồ thị lưỡng phân với một tập đỉnh đại diện cho nhân viên và tập còn lại là các công việc.
2. Ghép cặp sinh viên – đề tài: Khi sinh viên đăng ký các đề tài nghiên cứu, mỗi đề tài có một số sinh viên quan tâm. Ghép cặp phù hợp giúp đảm bảo tối ưu hóa việc phân bổ nguồn lực và tạo điều kiện cho sinh viên làm việc trên đề tài mình yêu thích.
3. Hệ thống gợi ý: Trong các ứng dụng hẹn hò, tuyển dụng hay thương mại điện tử, hệ thống gợi ý giúp tìm ra sự ghép cặp tốt nhất giữa các đối tượng dựa trên sở thích hoặc đặc điểm chung.

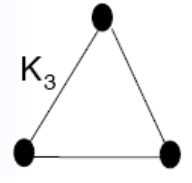
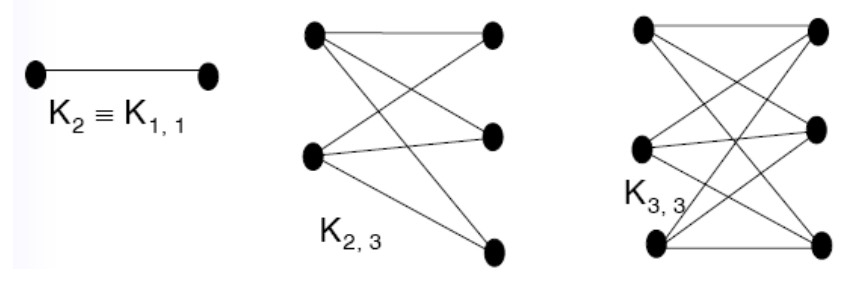
## 1.2 Cơ sở lý thuyết

1.2.1 Đồ thị (Graph)

Đồ thị không hướng (Undirected Graph): Trong đồ thị không hướng, các cạnh không có hướng cụ thể. Điều này có nghĩa là mối quan hệ giữa hai đỉnh là hai chiều. Nếu có một cạnh nối đỉnh A và đỉnh B, thì chúng ta có thể di chuyển từ A đến B và ngược lại từ B đến A. Ví dụ, một mạng lưới giao thông giữa các thành phố mà không có đường một chiều có thể được biểu diễn bằng đồ thị không hướng.

Đồ thị có hướng (Directed Graph): Trong đồ thị có hướng, các cạnh có một hướng cụ thể. Điều này có nghĩa là mối quan hệ giữa hai đỉnh là một chiều. Nếu có một cạnh hướng từ đỉnh A đến đỉnh B, thì chúng ta có thể di chuyển từ A đến B, nhưng không thể di chHình iuyển ngược lại từ B đến A (trừ khi có một cạnh hướng từ B đến A). Ví dụ, một sơ đồ dòng chảy công việc, nơi mỗi bước phải hoàn thành trước khi chuyển sang bước tiếp theo, có thể được biểu diễn bằng đồ thị có hướng.

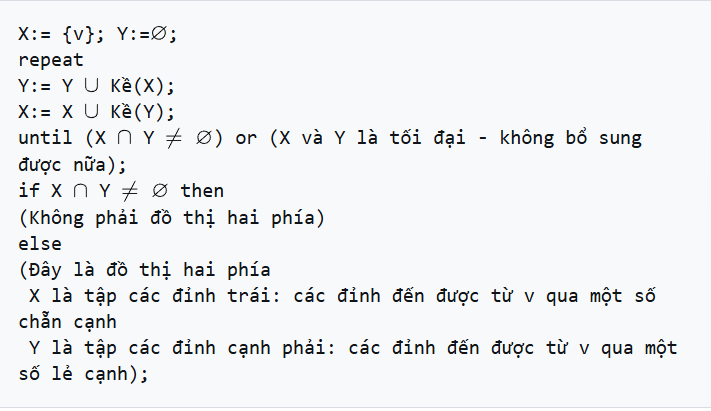
## 1.2.2 Đồ thị 2 phía (Bipartite Graph)

Đồ thị hai phía (Bipartite Graph) là một loại đồ thị đặc biệt trong Lý thuyết đồ thị, trong đó tập các đỉnh có thể được phân chia thành hai tập con không giao nhau sao cho không có cạnh nào nối hai đỉnh thuộc cùng một tập

Hình 1: Đồ thị hai phía

Hình 2: Không phải đồ thị hai phía

### 1.2.2.1 Thuật toán kiểm tra

Ta chọn 1 đỉnh v bất kì:

Hình 3: Thuật toán kiểm tra đồ thị liên thông là đồ thị hai phía

### 1.2.2.2 Định lí đồ thị hai phía

Một đồ thị A là hai phần khi và chỉ khi G không có chu trình lẻ.

Chứng minh:

Cho đồ thị liên thông G, ta chọn đỉnh u bất kỳ và thực hiện phân loại các đỉnh. Nếu đường đi từ u đến một đỉnh v có độ dài chẵn => đưa v vào tập U, nếu đường đi từ u tới v là độ dài lẻ => đưa v vào tập V.

Phân chia này không mâu thuẫn: Giả sử có một đỉnh v có hai đường đi từ u, một chẵn và một lẻ. Khi đó, hai đường này tạo thành một chu trình lẻ, mâu thuẫn với giả thiết rằng G không có chu trình lẻ. Do đó, mỗi đỉnh v chỉ có một cách phân loại duy nhất: vào tập U (nếu đường đi từ u có độ dài chẵn) hoặc vào tập V (nếu độ dài lẻ).

Không có cạnh nào nối giữa các đỉnh trong cùng một tập: Giả sử có cạnh (x, y) ∈ U, tức là x, y đều thuộc tập U. Điều này có nghĩa là x, y đều có đường đi chẵn từ u, tạo thành một chu trình lẻ. Chu trình lẻ này mâu thuẫn với giả thiết, nên không thể có cạnh nối giữa hai đỉnh trong U.

Không có cạnh nào nối giữa các đỉnh trong V: Giả sử có cạnh (x, y) ∈ V, tức là x, y đều thuộc tập V. Khi đó, cả hai đều có đường đi lẻ từ u, và cạnh này tạo thành chu trình lẻ. Mâu thuẫn với giả thiết, nên không thể có cạnh nối giữa hai đỉnh trong V.

## 1.⇒2.3 Danh sách kề (Adjacency list)

Tạo N mảng giá trị, mảng giá trị thứ u lưu danh sách các đỉnh kề với u

const int N = 1e5 + 10;

int n, m;

vector<int> adj[N];

int main() {

cin >> n >> m;

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

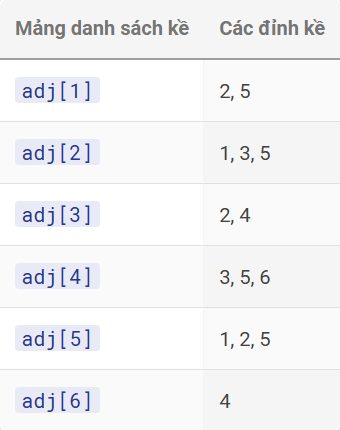
int a, b; cin >> a >> b;

adj[a].push\_back(b);

adj[b].push\_back(a); // Nếu đồ thị có hướng thì không cần viết dòng này

}

return 0;

}

Hình 4: Danh sách kề

## 1.2.4 Đường tăng luồng (Augmenting path)

Là đường đi từ đỉnh nguồn đến đỉnh đích trên đồ thị có thể chứa các cạnh được tăng giá trị luồng(bằng lượng tối đa có thể được tăng).

Một đường tăng luồng được tạo ra bằng cách sử dụng một thuật toán tìm kiếm đường đi như DFS hoặc BFS. Khi một đường tăng luồng được tìm thấy, lượng luồng tối đa có thể được tăng trên đường đó bằng lượng luồng còn lại trên đường tăng luồng (bottleneck capacity). Một đường đi bắt đầu và kết thúc tại các đỉnh chưa ghép, xen kẽ giữa các cạnh không thuộc và thuộc matching.

Khi tìm thấy một đường tăng, ta có thể “đảo chiều” các cặp ghép dọc theo đường đó để tăng thêm 1 matching.

Ví dụ, trên một đồ thị có các đỉnh và các cạnh được gán trọng số và mỗi cạnh có một giá trị luồng tối đa, đường tăng luồng giữa đỉnh nguồn và đỉnh đích có thể được tìm bằng cách tìm kiếm đường đi từ đỉnh nguồn đến đỉnh đích. Sau đó, lượng luồng trên đường đó có thể được tăng bằng lượng luồng tối đa có thể được tăng trên đường đó.

## 1.2.5 BFS (Breadth-First Search)

BFS hay còn gọi là thuật toán tìm kiếm theo chiều rộng. BFS là một thuật toán cơ bản và được sử dụng rộng rãi trong nhiều ứng dụng thực tế.

Cách thức hoạt động của BFS

BFS hoạt động bằng cách duyệt đồ thị theo từng lớp, bắt đầu từ một đỉnh gốc. Thuật toán sẽ thăm tất cả các đỉnh kề với đỉnh gốc trước khi chuyển sang thăm các đỉnh kề của các đỉnh đã thăm trước đó. Quá trình này tiếp tục cho đến khi tất cả các đỉnh có thể truy cập được từ đỉnh gốc đã được thăm. Để thực hiện điều này, BFS sử dụng một hàng đợi (queue) để lưu trữ các đỉnh cần thăm.

Các bước thực hiện BFS

Thuật toán BFS(G, s) // G là đồ thị, s là đỉnh gốc

1. Khởi tạo hàng đợi Q rỗng.

2. Đánh dấu tất cả các đỉnh là chưa được thăm.

3. Đánh dấu đỉnh gốc s là đã thăm.

4. Thêm s vào hàng đợi Q.

5. Trong khi Q không rỗng:

6. Lấy một đỉnh u từ Q (loại bỏ khỏi hàng đợi).

7. Duyệt tất cả các đỉnh kề v của u:

8. Nếu v chưa được thăm:

9. Đánh dấu v là đã thăm.

10. Thêm v vào hàng đợi Q.

Quá trình này đảm bảo rằng các đỉnh gần với đỉnh gốc sẽ được thăm trước, do đó thuật toán được gọi là tìm kiếm theo chiều rộng.

BFS có nhiều ứng dụng quan trọng trong khoa học máy tính và các lĩnh vực khác, bao gồm:

Tìm đường đi ngắn nhất trong đồ thị không trọng số: BFS đảm bảo tìm được đường đi ngắn nhất (số cạnh ít nhất) giữa hai đỉnh trong đồ thị không trọng số.

Tìm tất cả các đỉnh có thể truy cập được từ một đỉnh gốc: BFS có thể được sử dụng để xác định các thành phần liên thông trong đồ thị.

Thu thập dữ liệu từ mạng xã hội: BFS có thể được sử dụng để thu thập thông tin từ các mối quan hệ trên mạng xã hội.

Tìm kiếm trên web: Các công cụ tìm kiếm có thể sử dụng BFS để duyệt qua các trang web liên kết với nhau

## 1.2.6 DFS (Depth-First Search)

Nếu như BFS ưu tiên việc khám phá các đỉnh gần nguồn trước, thì DFS lại đi sâu vào từng nhánh của đồ thị trước khi quay lại khám phá các nhánh khác.

DFS, hay còn gọi là tìm kiếm theo chiều sâu, là một thuật toán duyệt đồ thị bắt đầu từ một đỉnh gốc và đi sâu vào các đỉnh kề của nó trước khi quay lại và khám phá các đỉnh khác. Thuật toán này hoạt động dựa trên nguyên tắc “đi sâu nhất có thể”, tức là nó sẽ tiếp tục khám phá các đỉnh liền kề chưa được thăm cho đến khi không còn đỉnh nào có thể đi tiếp được nữa. Sau đó, nó sẽ quay lại đỉnh trước đó và tiếp tục khám phá các nhánh khác.

Cách thức hoạt động của DFS:

Bắt đầu: Chọn một đỉnh bất kỳ làm đỉnh xuất phát.

Thăm đỉnh: Đánh dấu đỉnh hiện tại là đã được thăm.

Khám phá các đỉnh kề: Duyệt qua các đỉnh kề của đỉnh hiện tại. Nếu một đỉnh kề chưa được thăm, hãy đệ quy (hoặc sử dụng stack) để tiếp tục thăm đỉnh đó.

Quay lui: Khi không còn đỉnh kề nào chưa được thăm, quay lại đỉnh trước đó và tiếp tục quá trình.

Kết thúc: Thuật toán kết thúc khi tất cả các đỉnh có thể truy cập từ đỉnh bắt đầu đã được thăm.

Thuật toán DFS(G, s) // G là đồ thị, s là đỉnh bắt đầu

1. Khởi tạo một stack S rỗng.

2. Đánh dấu tất cả các đỉnh là chưa được thăm.

3. Đưa đỉnh bắt đầu s vào stack.

4. Trong khi S không rỗng:

5. Lấy một đỉnh u từ S (pop).

6. Nếu u chưa được thăm:

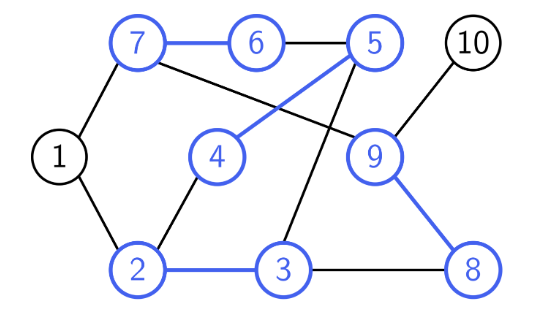
7. Đánh dấu u là đã được thăm.

8. Duyệt tất cả các đỉnh kề v của u:

 9. Nếu v chưa được thăm, đưa v vào stack (push).

## 1.2.7 Matching(Cặp ghép)

Hình 5: Phân biệt BFS và DFS

Matching là một tập hợp các cạnh sao cho không có hai cạnh nào có chung một đỉnh. Giả sử cho một đồ thị vô hướng G=(V, E) V đỉnh, E cạnh. Một tập cạnh M thuộc E được gọi là cặp cạnh ghép nếu không có đỉnh v ∈ V là một đỉnh tự do không thuộc bất kỳ cạnh nào trong tập cạnh M. Một cạnh e ∈ E là cạnh tự do nếu e không xuất hiện trong tập cạnh ghép. Đỉnh v ∈ V là đỉnh tự do nếu v không thuộc bất kỳ cạnh nào trong tập cạnh ghép.

Hình 6: Cặp cạnh ghép trên đồ thị vô hướng

- Đường tăng của luồng là một đường có hai đỉnh đầu mút của đường đi là các đỉnh tự do, đường đi bắt đầu và kết thúc bằng cạnh tự do. Nếu tồn tại đường tăng luồng, ta có thể mở rộng cặp ghép bằng cách thay thế các cạnh hiện có bằng các cạnh mới trên đường đi đó.

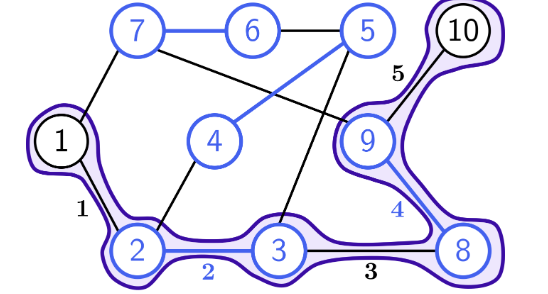
- Đường đi π = v₁, v₂, ..., v₂k+₂ là đường dẫn xen kẽ đối với M tương ứng có những tính chất sau:

+ Mỗi cạnh thuộc đường đi π đều thuộc tập cạnh E. eᵢ = (vᵢ, vᵢ₊₁) ∈ E ∀ 1 ≤ i ≤ 2k + 1.

+ v₁ và v₂k+₂ là các đỉnh tự do, trong khi v₂, v₃, ..., v₂k+₁ là các đỉnh được ghép.

+ Các cạnh xen kẽ giữa nằm trong M và nằm ngoài M. e₁, e₃, ..., e₂k+₁ ∉ M và e₂, e₄, ..., e₂k ∈ M. Có k cạnh khớp nhau và k + 1 cạnh tự do.

+ M ∩ π = {e₂, e₄, ..., e₂k}

Nói một cách đơn giản hơn, đường đi π bắt đầu và kết thúc tại các đỉnh không khớp và xen kẽ giữa các cạnh là một phần của M khớp và các cạnh không khớp. Có k cạnh là một phần của khớp và k+1 cạnh không khớp, và tập hợp các cạnh nằm trong cả đường đi và khớp là e₂, e₄, ..., e₂k.

Hình 7: Đường tăng luông trên đồ thị vô hướng

Bản chất của bài toán

**Tìm cặp ghép cực đại trong đồ thị phân đôi** (Maximum Bipartite Matching): Có một đồ thị phân đôi gồm hai tập đỉnh riêng biệt (gọi là bên trái và bên phải). Mỗi cạnh nối một đỉnh bên trái với một đỉnh bên phải.

Nhiệm vụ là tìm một tập hợp các cạnh (cặp ghép) sao cho:

* Không có hai cạnh nào trong tập hợp này chia sẻ cùng một đỉnh.
* Số lượng cạnh trong tập hợp là lớn nhất có thể (cực đại).

# II/ THIẾT KẾ THUẬT TOÁN

## 2.1 Thuật toán Hopcroft-Karp

B1: Tìm kiếm nhiều nhất số đường đi tăng luồng có đường đi ngắn nhất

B2: Cập nhật cạnh ghép

B3: Kết thúc thuật toán nếu không còn đường tăng luồng nữa, nếu còn quay lại bước 1

**Các ý tưởng chính:**

* Level graph (BFS phase)

Gán khoảng cách (dist[]) cho các đỉnh bên trái chưa ghép.

Tạo “level graph” để biết khoảng cách từ đỉnh chưa ghép đến đỉnh có thể ghép.

Dùng **BFS** để xây dựng cấp độ (level) cho từng đỉnh.

* Tìm đường tăng (DFS phase)

Với mỗi đỉnh chưa ghép bên trái, dùng **DFS** để tìm **đường tăng** (augmenting path).

Nếu tìm được đường tăng, hoán đổi các cặp ghép trên đường đó để tăng kích thước matching.

* Lặp cho đến khi không tìm được đường tăng mới

Nếu trong một lượt BFS không còn đường tăng nào, thuật toán kết thúc.

***Giải thích về bài toán ghép sinh viên với đề tài***

Tập bên trái: Các sinh viên (mỗi sinh viên là một đỉnh).

Tập bên phải: Các đề tài (mỗi đề tài là một đỉnh).

Cạnh nối: Nếu sinh viên A có thể làm đề tài A, B, C…ta nối một cạnh từ sinh viên A đến các đề tài đó.

Mục đích: Ghép mỗi sinh viên với một đề tài sao cho số lượng sinh viên được nhận đề tài tối đa, mỗi đề tài chỉ được giao cho 1 sinh viên, không có đề tài nào được giao cho 2 sinh viên cùng lúc.

Độ phức tạp của thuật toán Hopcroft – Karp là O(E.)

## 2.2 Thiết kế

1. Biến và cấu trúc dữ liệu chính:

g[u][i]: danh sách kề — các đỉnh bên phải kết nối với đỉnh u bên trái.

degree[u]: số đỉnh kề của u bên trái.

matchl[u]: đỉnh bên phải được ghép với u (0 nếu chưa ghép).

matchr[v]: đỉnh bên trái được ghép với v (0 nếu chưa ghép).

dist[u]: khoảng cách (cấp độ) trong đồ thị cấp độ.

queue\_[]: dùng để BFS.

2. Hàm bfs(n) — Xây dựng đồ thị cấp độ:

Khởi tạo dist[u] = 0 nếu u chưa được ghép (matchl[u] == 0), và đẩy u vào hàng đợi.

Đỉnh đã ghép thì đặt dist[u] = -1.

Duyệt hàng đợi BFS, mở rộng sang các đỉnh kề v bên phải.

Nếu v đã ghép với một đỉnh matchr[v], tiếp tục xét đỉnh đó (nếu chưa xét), đặt cấp độ tăng lên 1 so với u.

Nếu tìm được đỉnh v chưa ghép (matchr[v] == 0), đánh dấu found = true — tức là còn đường tăng.

Hàm trả về true nếu tồn tại đường tăng, false nếu không còn.

3. Hàm dfs(u) — Tìm đường tăng

Với đỉnh u chưa ghép, duyệt các đỉnh v kề.

Nếu v chưa ghép (matchr[v] == 0), hoặc có thể tiếp tục tìm đường tăng qua matchr[v] (đỉnh ghép với v) với cấp độ tăng dần (đảm bảo tìm đường ngắn nhất), thì ghép u với v.

Trả về true nếu tìm được đường tăng.

4. Hàm hopcroft\_karp(n, m)

Khởi tạo các mảng matchl và matchr bằng 0 (chưa ghép).

Lặp lại:

Gọi bfs để xây dựng đồ thị cấp độ. Nếu không còn đường tăng thì dừng.

Dùng dfs tìm tất cả đường tăng có thể trong đồ thị cấp độ và tăng số cặp ghép.

Trả về số cặp ghép cực đại.

5. Hàm main

Nhập dữ liệu: số đỉnh 2 bên, số cạnh.

Đọc các cạnh, lưu vào danh sách kề.

Gọi hopcroft\_karp để tìm số cặp ghép cực đại.

In ra kết quả và các cặp ghép.

# III/ TRIỂN KHAI THUẬT TOÁN

## 3.1 Viết chương trình C++

#include <iostream>

using namespace std;

const int maxn = 10005;

int g[maxn][maxn];

int degree[maxn];

int matchl[maxn], matchr[maxn];

int dist[maxn];

int queue\_[maxn];

bool bfs(int n) {

int front = 0, back = 0;

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

if (matchl[u] == 0) {

dist[u] = 0;

queue\_[back++] = u;

} else {

dist[u] = -1;

}

}

bool found = false;

while (front < back) {

int u = queue\_[front++];

for (int i = 0; i < degree[u]; ++i) {

int v = g[u][i];

if (matchr[v] != 0 && dist[matchr[v]] == -1) {

dist[matchr[v]] = dist[u] + 1;

queue\_[back++] = matchr[v];

}

if (matchr[v] == 0) found = true;

}

}

return found;

}

bool dfs(int u) {

for (int i = 0; i < degree[u]; ++i) {

int v = g[u][i];

if (matchr[v] == 0 || (dist[matchr[v]] == dist[u] + 1 && dfs(matchr[v]))) {

matchl[u] = v;

matchr[v] = u;

return true;

}

}

return false;

}

int hopcroft\_karp(int n, int m) {

for (int i = 1; i <= n; ++i) matchl[i] = 0;

for (int i = 1; i <= m; ++i) matchr[i] = 0;

int result = 0;

while (bfs(n)) {

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

if (matchl[u] == 0 && dfs(u)) {

result++;

}

}

}

return result;

}

int main() {

int n, m, e;

cout << "Nhap so dinh tap trai (x) va tap phai (y): ";

cin >> n >> m;

cout << "Nhap so canh: ";

cin >> e;

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

int u, v;

cin >> u >> v;

g[u][degree[u]++] = v;

}

int res = hopcroft\_karp(n, m);

cout << "So luong cap ghep cuc dai la: " << res << endl;

cout << "Cac cap ghep:" << endl;

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

if (matchl[u] != 0) {

cout << u << " - " << matchl[u] << endl;

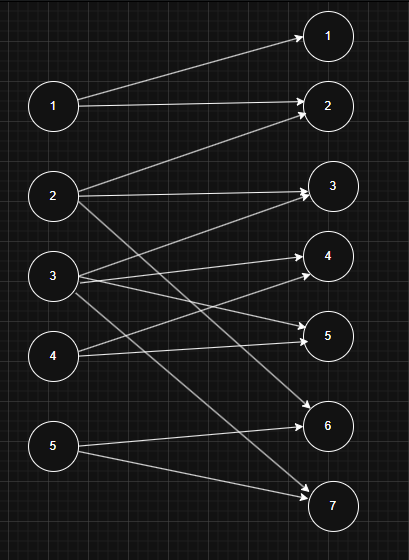
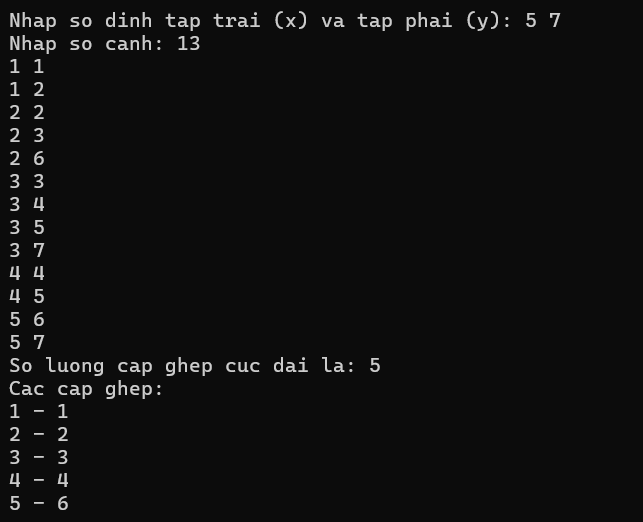
}

}

return 0;

}

## 3.2 Kết quả thử



Hình 8: Đầu vào bài toán ghép cặp đồ thị hai phía

Hình 9: Kết quả ghép cặp

# IV/ KẾT LUẬN

Sau khi tìm hiểu và triển khai, em nhận thấy thuật toán Hopcroft–Karp là một phương pháp rất hiệu quả để giải bài toán ghép cặp cực đại trong đồ thị hai phía.

Trong quá trình làm bài, em đã hiểu rõ hơn về cách hoạt động của thuật toán, cách tổ chức dữ liệu, cũng như cách triển khai bằng ngôn ngữ lập trình C++. Đồng thời, em cũng thấy được rằng đây không chỉ là một bài toán mang tính lý thuyết, mà còn có thể áp dụng vào các tình huống thực tế như phân công công việc, ghép cặp, lập lịch, v.v.

Tuy ban đầu có hơi khó khăn do thuật toán khá trừu tượng, nhưng nhờ việc phân tích từng bước và thử nghiệm trên dữ liệu nhỏ, em đã dần hiểu và hoàn thành việc cài đặt. Em tin rằng việc học thuật toán này sẽ giúp ích rất nhiều trong việc tiếp cận các bài toán tối ưu khác trong tương lai.

Một số hạn chế như thiếu phân tích cho trường hợp đồ thị có trọng số hoặc đồ thị không phải hai phía. Có khả năng gặp khó khăn nếu làm việc với dữ liệu, đồ thị lớn và chưa sử dụng được thư viện chuyên dụng.

Ứng dụng thực tế của thuật toán được đề xuất khá phù hợp, như bài toán ghép cặp sinh viên và đề tài. Cách phân chia đề tài cho sinh viên khá hợp lý, đảm bảo cho mỗi sinh viên một đề tài mà không trùng nhau.

# Tài liệu tham khảo

Brilliant.org. (n.d.). *Matching Algorithms*. Retrieved from Brilliant.org: https://brilliant.org/wiki/matching-algorithms/

Luanvan.net.vn. (n.d.). *Bài toán ghép cặp và ứng dụng*. Retrieved from Luanvan.net.vn: https://luanvan.net.vn/luan-van/bai-toan-ghep-cap-va-ung-dung-47727/

Wiki, V. (n.d.). *Lý thuyết đồ thị - Bài toán Matching cực đại*. Retrieved from VNOI Wiki: https://wiki.vnoi.info/algo/graph-theory/max-matching

Wikipedia. (n.d.). *Đồ thị hai phía*. Retrieved from Wikipedia: https://vi.wikipedia.org/wiki/Đồ\_thị\_hai\_phía

Wikipedia. (n.d.). *Hopcroft–Karp algorithm*. Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Hopcroft%E2%80%93Karp\_algorithm