**BỘ CÔNG THƯƠNG**

**TRƯỜNG ĐẠI HỌC CÔNG NGHIỆP THỰC PHẨM TP.HCM**

**KHOA CÔNG NGHỆ THÔNG TIN**

🙜🙢🙠🙞



**ĐỒ ÁN MÔN HỌC**

**CẤU TRÚC RỜI RẠC**

**ĐỀ TÀI : ĐƯỜNG ĐI EULER**

Giảng viên hướng dẫn: Nguyễn Thị Thùy Trang

Nhóm : **∞**

Sinh viên thực hiện:

1. Nguyễn Quang Hà – 2033221091
2. Ngô Đức Hậu - 2033221334
3. Lê Văn Anh Đạt - 2033220898
4. Lư Tất Tuấn Đạt - 2001220935
5. Phạm Huy Trung - 2001225688

Thành phố Hồ Chí Minh, tháng 10 năm 2023

**LỜI CAM ĐOAN**

Chúng tôi xin cam đoan đây là công trình nghiên cứu của riêng chúng tôi. Các số liệu, kết quả nêu trong Tiểu luận/ Đồ án là trung thực và chưa từng được ai công bố trong bất kỳ công trình nào khác.

Tôi xin cam đoan rằng mọi sự giúp đỡ cho việc thực hiện Tiểu luận/ Đồ án này đã được cảm ơn và các thông tin trích dẫn trong Tiểu luận/ Đồ án đã được chỉ rõ nguồn gốc.

Nhóm sinh viên thực hiện tiểu luận

(Ký và ghi rõ họ tên)

NHÓM ∞

* Nguyễn Quang Hà - 2033221091
* Ngô Đức Hậu - 2033221334
* Phạm Huy Trung - 2001225688
* Lê Văn Anh Đạt - 2033220898
* Lư Tất Tuấn Đạt – 2001220935

**BẢNG PHÂN CHIA CÔNG VIỆC**

|  |  |  |
| --- | --- | --- |
| **TÊN** | **CÔNG VIỆC** | **PHẦN TRĂM ĐÓNG GÓP** |
| Nguyễn Quang Hà | Làm Word, tìm hiểu thông tin | 15% |
| Lê Văn Anh Đạt | 15% |
| Phạm Huy Trung | Edit video, thuyết trình code | 20% |
| Ngô Đức Hậu | Code, tìm hiểu thuật toán của đề tài | 15% |
| Lư Tất Tuấn Đạt | 35% |

**MỤC LỤC**

[I. ĐẠI CƯƠNG VỀ ĐỒ THỊ 1](#_Toc152594824)

[1.1 Giới thiệu 1](#_Toc152594825)

[1.2 Định nghĩa đồ thị 1](#_Toc152594826)

[1.3 Một số thuật ngữ cơ bản 2](#_Toc152594827)

[1.4 Đường đi, chu trinh và đồ thị liên thông 3](#_Toc152594828)

[II. CHU TRÌNH EULER VÀ BÀI TOÁN NGƯỜI ĐƯA THƯ 5](#_Toc152594829)

[2.1 Định nghĩa 5](#_Toc152594830)

[2.2 Điều kiện cần và đủ 5](#_Toc152594831)

[2.3 Các thuật toán tìm chu trình Euler 6](#_Toc152594832)

[2.4 Điều kiện để đồ thị có chu trình, đường đi euler 8](#_Toc152594833)

[2.5 Bài toán bài đưa thư 9](#_Toc152594834)

[III. ĐỒ THỊ HAMILTON, THUẬT TOÁN DIJKSTRA 13](#_Toc152594835)

**MỤC LỤC HÌNH ẢNH**

[**Hình 1** 1](#_Toc152652402)

[**Hình 2** 3](#_Toc152652403)

[**Hình 3** 4](#_Toc152652404)

[**Hình 4** 4](#_Toc152652405)

[**Hình 5** 5](#_Toc152652406)

[**Hình 6** 11](#_Toc152652407)

[**Hình 7** 15](#_Toc152652408)

[**Hình 8** 16](#_Toc152652409)

[**Hình 9** 18](#_Toc152652410)

[**Hình 10** 22](#_Toc152652411)

**GIỚI THIỆU**

Lý thuyết đồ thị là ngành khoa học được phát triển từ lâu nhưng lại có nhiều ứng dụng hiện đại. Những ý tưởng cơ bản của nó được dưa ra từ thế kỷ XVIII bởi nhà toán học Thụy sĩ tên là Leonhard Euler. Lý thuyết đồ thị được dùng để giải các bài toán trong nhiều lĩnh vực khác nhau. Chúng ta có thể xác định hai máy tính trong mạng có thể trao đổi thông tin được với nhau hay không nhờ mô hình đồ thị của mạng máy tính. Đồ thị có trọng số trên các cạnh có thể sử dụng để giải các bài toán như: Tìm đường đi ngắn nhất giữa hai thành phố trong mạng giao thông. Chúng ta cũng còn sử dụng đồ thị để giải các bài toán về lập lịch, thời khóa biểu, và phân bố tần số cho các trạm phát thanh và truyền hình... Ngày nay Lý thuyết đồ thị đã phát triển thành một ngành Toán học ứng dụng có vị trí đặc biệt quan trọng về mặt lý thuyết cũng như ứng dụng. Trong nội dung cuốn tiểu luận này nhóm chúng tôi xin trình bày về "Chu trình Euler và bài toán người đưa thư".

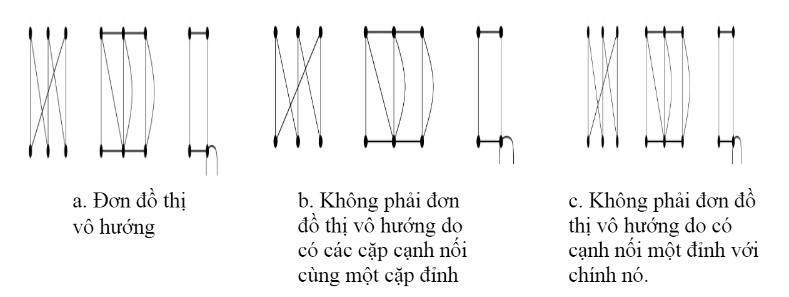
# ĐẠI CƯƠNG VỀ ĐỒ THỊ

## Giới thiệu

Lý thuyết đồ thị là một lĩnh vực nghiên cứu đã có từ lâu và có nhiều ứng dụng trong ngành công nghệ thông tin. Những tư tưởng cơ bản của lý thuyết đồ thị được đề xuất vào những năm đầu của thế kỷ 18 bởi nhà toán học lỗi lạc người Thụy Sỹ: Leonhard Euler. Chính ông là người đã sử dụng đồ thị để giải bài toán nổi tiếng về 7 cái cầu ở thành phố Konigberg. Những ứng dụng cơ bản của đồ thị như: - Xác định tính liên thông trong một mạng máy tính: hai máy tính nào đó có thể truyền dữ liệu cho nhau được không. - Tìm đường đi ngắn nhất trên mạng giao thông

## Định nghĩa đồ thị

Định nghĩa 1.2.1. Một đơn đồ thị vô hướng là một bộ G=<V,E>, trong đó: - V ‡ là tập hợp hữu hạn gồm các đỉnh của đồ thị. - E là tập hợp các cặp không có thứ tự gồm hai phần tử khác nhau của V gọi là các cạnh. Như vậy, theo định nghĩa trên, trong một đơn đồ thị không thể có các cặp cạnh nối cùng một cặp đỉnh (do E là tập hợp nên không thể có 2 cặp trùng nhau), các cạnh đều không phân biệt thứ tự nên cạnh [u.v] Đều được coi là một cạnh duy nhất, điều này phù hợp với việc biểu diễn các con đường 2 chiều, và hiển nhiên không có cặp [u,u] nào đó trong E.



**Hình 1**

Định nghĩa 1.2.2. Đa đồ thị vô hướng là một bộ G=<V,E>, trong đó - V# 0 là tập hợp hữu hạn gồm các đỉnh của đồ thị.

- E là một họ các cặp không có thứ tự của V gọi là các cung.

Định nghĩa 1.2.3. Đơn đồ thị có hướng là một bộ G=<V,E>, trong đó: - V ‡ 0 là tập hợp hữu hạn gồm các đỉnh của đồ thị. - E là tập hợp các cặp có thứ tự gồm hai phần tử khác nhau của V gọi là các cung. Ví dụ: Tiểu luận nhóm 1 Trang a. Đơn đồ thị vô hướng b. Không phải đơn đồ thị vô hướng do có các cặp cạnh nối cùng một cặp đỉnh c. Không phải đơn đồ thị vô hướng do có cạnh nối 1 đỉnh với chính nó.

Định nghĩa 1.2.4. Đa đô thị có hướng là một bộ G=<V,E>, trong đó

- V ‡ 0 là tập hợp hữu hạn gồm các đỉnh của đồ thị.

- E là một họ các cặp có thứ tự của V gọi là các cung.

Các cung nối cùng một cặp đỉnh được gọi là các cung song song. Ví dụ: Chú ý: - Đô thị sau vẫn được coi là đơn đồ thị có hướng vì e1 và e2, e3 và e4 không phải là 2 cung song song (do khác hướng).

## Một số thuật ngữ cơ bản

Định nghĩa 1.3.1. Cho đồ thị yô hướng G=<V,E>.

- Hai đình u và v của đồ thị được gọi là kể nhau nếu (u,v) là một cạnh của đồ thị.

- Nếu e=(u,v) là cạnh của đồ thị thì ta nói cạnh này là liên thuộc với hai đỉnh u và v. Cạnh được nói là nối đình u và v. Đỉnh u và v được gọi là đình đầu của cạnh e.

Định nghĩa 1.3.2. Cho đồ thị vô hướng G=<V,E>. Bậc của đình v trong đồ thị, ký hiệu là deg(v), là số cạnh liên thuộc với nó. Đình có bậc 0 được gọi là đỉnh cô lập, đỉnh có bậc 1 gọi là đinh treo.

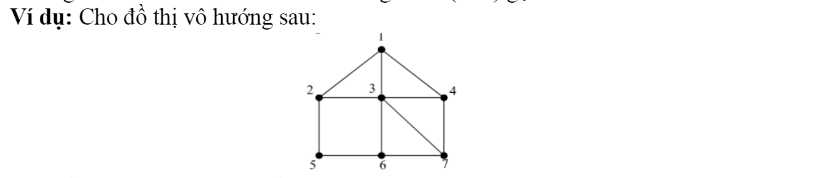
Định nghĩa 1.3.3 Cho đồ thị có hướng G=<V,E>. - Hai đỉnh u và v của đồ thị được gọi là kể nhau nếu (u,v) là một cung của đồ thị. - Nếu e=(u,v) là cung của đồ thị thi ta nói cung này đi ra khỏi đinh u vào đi vào đỉnh v. Đỉnh u được gọi là đỉnh đầu của cung e và đỉnh v được gọi là đỉnh cuối của cung e.

## Đường đi, chu trinh và đồ thị liên thông

Định nghĩa 1.4.1. Cho đồ thị G = <V,E> ( ký hiệu cho dùng chung cả đồ thị vô hướng và có hướng). Đường đi độ dài n từ đỉnh u đến đỉnh v (n là số nguyên dương)

Trong đồ thị trên, đỉnh 2 là đinh rẽ nhánh vì việc loại đỉnh này cùng với các cạnh (2,3), (2,1), (2,6) sẽ làm đồ thị có 2 thành phần liên thông. Cạnh (2,3) là cầu. Các cạnh còn lại đều không phải là cầu. Đối với đồ thị có hướng khái niệm liên thông khó thỏa mãn hơn do các cung bị hạn chế về chiều. Từ đó, bên cạnh khái niệm liên thông như đề cập trong đồ thị vô hướng, ta sẽ đưa thêm khái niệm liên thông nhẹ hơn: liên thông yếu

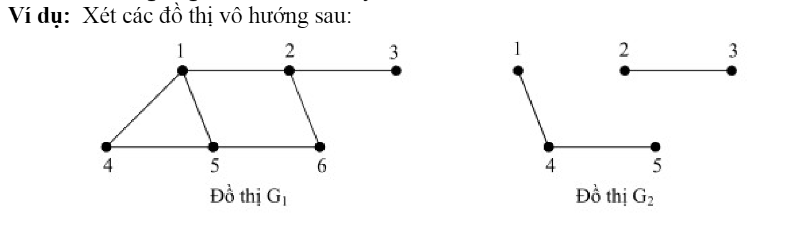
trong đó \* Ta sẽ dùng thuật ngữ đồ thị để chỉ chung cho cả đồ thị vô hướng và đồ thị có hướng Đường đi nói trên còn có thể được biểu diễn bằng dãy các cạnh/cung: Đỉnh u gọi là đỉnh đầu của đường đi, đỉnh v gọi là đỉnh cuối của đường đi. Đường đi có đỉnh đầu và đỉnh cuối trùng nhau (u=v) gọi là chu trình.



**Hình 2**

Một số đường đi từ đỉnh 2 đến đỉnh 7: - Đường đi d1: 2 3 4 7 (đường đi độ dài 3) - Đường đi d2: 2 3 4 1 3 6 7 (đường đi độ dài 6) Một số chu trình trên đồ thị trên: - Chu trình C1: 1 2 3 1 (chu trình có độ dải 3) - Chu trình C2: 1 2 3 7 6 3 4 1 (chu trỉnh có độ dài 7)

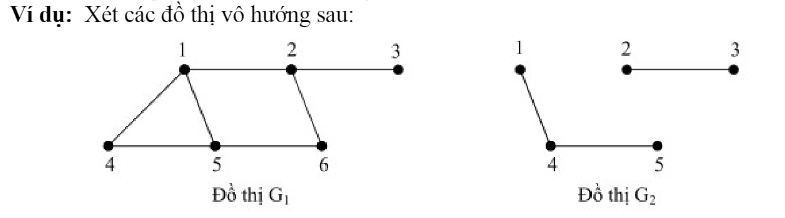
Định nghĩa 1.4.2. Đồ thị vô hướng G = <V,E> được gọi là liên thông nếu luôn tìm được đường đi giữa hai đình bất kỳ của nó.



**Hình 3**

Trong 2 đồ thị trên thì G1là đồ thị liên thông, còn G2 không phải là đồ thị liên thông vì giữa hai đình 1 và 2 không tồn tại một đường đi nào. Định nghĩa 1.4.3 Cho đồ thị G = (V,E). Đồ thị H = <W,F> được gọi là đồ thị con của G nếu và chỉ nếu W € V và F € E. Trong trường hợp một đồ thị vô hướng G không liên thông, nó sẽ được phân thành các đồ thị con độc lập nhau và chúng đều liên thông. Mỗi đồ thị con như vậy được gọi là một thành phần liên thông của G.

Định nghĩa 1.4.4 Cho đồ thị vô hướng G = <V,E>. Đình v của đồ thị được gọi là đỉnh rẽ nhánh nếu việc loại bỏ v và các cạnh liên thuộc với nó ra khỏi đồ thị sẽ làm tăng số thành phần liên thông của đồ thị. Cạnh e của đồ thị được gọi là cầu nếu việc loại bỏ nó ra khỏi đồ thị sẽ làm tăng số thành phần liên thông của đồ thị.



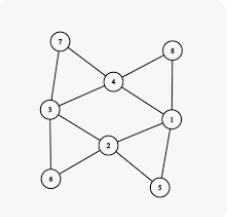
**Hình 4**

Trong 2 đồ thị trên thì G1 là đồ thị liên thông, còn G2 không phải là đồ thị liên thông vì giữa 2 đỉnh 1 và 2 không tồn tại 1 đường đi nào.

# CHU TRÌNH EULER VÀ BÀI TOÁN NGƯỜI ĐƯA THƯ

### Định nghĩa

Cho đồ thị G=(V,E), V là tập hợp các đỉnh, E là tập hợp các cạnh Chu trình Euler là chu trình qua mọi cạnh và mọi đỉnh đồ thị, mỗi cạnh không đi quá 1 lần. Đường đi Euler là đường đi qua mọi cạnh và mọi đỉnh đồ thị, mỗi cạnh không đi quá 1 lần. Cho đồ thị có hướng G=(V,E). Chu trình có hướng Euler là chu trình có hướng qua mọi cung và mọi đình đồ thị, mỗi cung không đi quá 1 lần. Đường đi có hướng Euler là đường đi có hướng qua mọi cung và mọi đỉnh đồ thị, mỗi cung không đi quá 1 lần. Đồ thị chứa chu trình Euler gọi là Đồ thị Euler.



**Hình 5**

### Điều kiện cần và đủ

#### Định lý 1 (Định lý Euler)

Đồ thị G có chu trình Euler khi và chỉ khi G liên thông và mọi đình có bậc chẵn khác 0. Chứng minh

(i) (=) : Giả sử G có chu trình Euler và v là đình bất kỳ của G. Khi đó chu trình Euler đến v theo cạnh e thì ra khỏi v bằng cạnh e' ‡ e. Do đó bậc của v phải là số chẵn. G hiển nhiên là liên thông.

(ii) (<=): Già sử G liên thông và mọi đỉnh có bậc chắn khác 0. Ta chứng minh G có chu trình Euler quy nạp theo số cạnh m của G. • m = 1: Vì G liên thông và mọi đỉnh bậc chẵn nên G chỉ có 1 đình và 1 khuyên. Khuyên đó cũng tạo thành chu trình Euler. • Giả sử G có m cạnh, số đỉnh n > 0 và mọi đồ thị liên thông có số cạnh nhỏ hơn m với mọi đình bậc chẵn đều có chu trình euler.

+ Trường hợp n = 1 hoặc 2 thị hiển nhiên tồn tại chu trình Euler .

+ Trường hợp n > 2. Vi bậc của các đỉnh chẵn ≥ 2, bao giờ cũng chọn được 3 đình a, b, c với các cạnh x = (a,b), Y = (a,c). - Già sử G chứa cạnh z = (b,c). Xét đồ thị G' thu được từ G bằng cách loại bỏ ba cạnh x,y,z. Sẽ xảy ra 1 trong ba khả năng sau: Tiểu luận nhỏm 1 Trang Toán ứng dụng Chu trình Euler và bải toán người đưa thư • G' liên thông. Vì số cạnh của G' nhỏ hơn m và các đình vẫn có bậc chẵn nên theo giả thiết quy nạp tồn tại chu trình Euler C' của G'. Nối chu trình con (x,y,z) với C' ta thu được chu trình Euler C của G.

Ta xây dựng chu trình Euler C của G như sau. Xuất phát từ đỉnh a đi theo chu trình C 1 quay về a, sau đó đi theo cạnh x = (a,b) đến đỉnh b, từ b đi theo chu trình 2 quay về b, sau đỏ đi theo cạnh z = (b,c) và y = (c,a) quay về a.

#### Định lý 2

Cho đồ thị G có k đình bậc lẻ. Khi đó số đường đi tối thiểu phủ G là k/2. Chứng minh. Ta đã biết số đình bậc lẻ là chẵn, k=2n. Chứng minh quy nạp theo n. (i) n=1: Nối 2 đỉnh bậc lẻ với nhau bằng cạnh z ta thu được đồ thị G' thoả định lý Euler. Như vậy G' có chu trình Euler C'. Bỏ cạnh z trên C' ta thu được đường đi Euler phủ G. (ii) Già sử G có số đình bậc lẻ là 2n và định lý đúng với k<2n. Nôi 2 đình bậc lẻ a,b nào đó với nhau bằng cạnh z ta thu được đồ thị G' có 2n-2 đình bậc lẻ. Theo giả thiết quy nạp G' có n-1 đường đi phù G'. Gọi P là đường đi qua cạnh z. Hiển nhiên a, b không phải đình đầu hoặc cuối của P, vỉ vậy nếu bỏ cạnh z ta thu được 2 đường đi P 1 và P 2 cùng với n-2 đường đi còn lại phủ đồ thị G. (đpcm) Bây giờ xét đồ thị có hướng G = (V, A).

### Các thuật toán tìm chu trình Euler

Có một số thuật toán để tìm chu trình Euler trong đồ thị. Chu trình Euler là một chu trình (đường đi đóng) đi qua tất cả các cạnh của đồ thị ít nhất một lần. Dưới đây là một biến phổ thuật toán số để tìm chu trình Euler:

Thuật toán Hierholzer :

Đây là thuật toán phổ biến và hiệu quả nhất để tìm kiếm chu trình Euler trong đồ thị có hướng. Thuật toán này hoạt động bằng cách tìm kiếm các chu trình con (chu trình con) trong sơ đồ và sau đó chúng kết hợp để tạo chu trình Euler.

1. Chọn một đỉnh xuất phát:

- Bắt đầu từ một đỉnh bất kỳ trong đồ thị.

2. Lặp cho đến khi không còn cạnh nào:

Chọn một cạnh kề có thể đi được.

Nếu đỉnh đó không phải là cầu, thực hiện bước tiếp theo.

Nếu đỉnh đó là cầu, chọn một cạnh thay thế khác.

- Loại bỏ cạnh đã chọn và di chuyển đến đỉnh kề đó.

3. Kết quả:

- Khi thuật toán kết thúc, tất cả các cạnh đã được duyệt đúng một lần, và ta có một chu trình Euler.

Fleury toán học : Đây là một thuật toán để tìm chu trình Euler trong đồ thị vô hướng. Thuật toán này tương tự như thuật toán Hierholzer, nhưng có một số chế độ giới hạn về việc chọn cạnh để đi qua. Thuật toán này hoạt động bằng cách chọn cạnh sao cho không tạo ra các cầu (cạnh duy nhất kết nối hai thành phần liên thông khác nhau) cho đến khi không còn cạnh nào để đi qua.

1. Tìm một chu trình hoặc đường đi Euler:

- Bắt đầu từ một đỉnh bất kỳ, tìm một chu trình hoặc đường đi Euler thông qua việc thêm cạnh vào đường đi khi đi qua đỉnh chưa được xử lý.

2. Lặp lại cho đến khi không còn cạnh nào chưa xử lý:

- Khi có đỉnh nào đó có cạnh chưa xử lý, bắt đầu một chu trình hoặc đường đi Euler mới từ đỉnh đó.

3. Kết quả:

- Khi thuật toán kết thúc, ta có thể kết hợp các chu trình Euler hoặc đường đi Euler thành một chu trình Euler lớn hơn.

**Đặc điểm chung:**

Cả hai thuật toán đều đảm bảo rằng mỗi cạnh sẽ được duyệt đúng một lần.

Đối với đồ thị có hướng, Fleury có thể không xử lý được trường hợp của cầu, trong khi Hierholzer không gặp vấn đề này và cũng áp dụng được cho đồ thị vô hướng.

Thuật toán Floyd - Tìm đường đi ngắn nhất giữa mọi cặp đỉnh: Floyd hay còn gọi là Floyd-Warshall là thuật toán để tìm đường đi ngắn nhất giữa mọi cặp đỉnh. Floyd hoạt động được trên đồ thị có hướng, có thể có trọng số âm, tuy nhiên không có chu trình âm. Ngoài ra, Floyd còn có thể được dùng để phát hiện chu trình âm ( tìm đường đi ngắn nhất, truy vết đường đi, phát hiện chu trình âm, liệt kê tất cả đường đi đơn.)

1. Khởi tạo ma trận:

Bước đầu tiên là khởi tạo ma trận trọng số, trong đó trọng số của cạnh (i, j) là chi phí trực tiếp từ đỉnh i đến đỉnh j, nếu có. Nếu không có cạnh nối trực tiếp, thì trọng số tương ứng được đặt là vô cực (infinity).

2.Lặp qua tất cả các đỉnh:

Với mỗi đỉnh trung gian k từ 1 đến số đỉnh trong đồ thị, thuật toán sẽ kiểm tra xem có cách nào ngắn hơn để đi từ đỉnh i đến đỉnh j thông qua đỉnh k hay không.

3.Cập nhật ma trận:

Nếu có một đường đi ngắn hơn từ i đến j thông qua đỉnh k, thì cập nhật giá trị của ma trận tại (i, j) bằng chi phí của đường đi ngắn nhất mới.

4.Kết quả:

Sau khi lặp qua tất cả các đỉnh trung gian, ma trận kết quả sẽ chứa chi phí ngắn nhất giữa mọi cặp đỉnh trong đồ thị.

### Điều kiện để đồ thị có chu trình, đường đi euler

* **Đồ Thị Vô Hướng**

Đồ thị vô hướng có chu trình Euler nếu :

- Các đỉnh có bậc khác 0 của đồ thị liên thông với nha

-Tất cả các đỉnh của đồ thị đều có bậc chẵ

Đồ thị vô hướng có đường đi Euler nếu :

- Các đỉnh có bậc khác 0 của đồ thị liên thông với nhau

- Đồ thị có 0 hoặc 2 đỉnh có bậc lẻ, trong trường hợp có 2 đỉnh bậc lẻ thì đường đi sẽ bắt đầu từ đỉnh bậc lẻ thứ 1 và kết thúc ở đỉnh bậc lẻ thứ 2

* **Đồ Thị Có Hướng**

Đồ thị có hướng có chu trình Euler nếu :

- Các đỉnh có bậc khác 0 của đồ thị thuộc cùng 1 thành phần liên thông -

- Mọi đỉnh thuộc đồ thị đều có bán bậc ra bằng bán bậc vào

Đồ thị có hướng có đường đi Euler nếu :

- Các đỉnh có bậc khác 0 của đồ thị thuộc cùng 1 thành phần liên thông

- Tồn tại 2 đỉnh u, v mà deg+(u) - deg-(u) = 1 và deg-(v) - deg+(v) = 1, mọi đỉnh còn lại đều có bán bậc ra bằng bán bậc vào. Khi đó đường đi bắt đầu từ đỉnh u và kết thúc ở đỉnh v.

### Bài toán bài đưa thư

Bài toán "bài đưa thư" (The Postman problem) là một đồ thị bài toán trong đó bạn cần tìm một đường đi ngắn nhất hoặc đường đi với độ dài nhỏ nhất để đi qua tất cả các cạnh (hoặc đỉnh) ít nhất một lần trong một sơ đồ không kín (không phải là chương trình Euler). Bài toán này có thể xuất ra nhiều câu hỏi thực tế, ví dụ như một người đưa thư muốn đi qua tất cả các ngõ để phân tích thư.



**Hình 6**

Có hai biến chính của thư đưa ra bài toán:

Bài toán đưa ra ngắn nhất (Vấn đề của người đưa thư ngắn nhất) : Trong trường hợp này, bạn cần tìm đường đi ngắn nhất để đi qua tất cả các cạnh một lần.

Bài toán thư nhẹ nhất (Vấn đề người đưa thư tối thiểu) : Trong trường hợp này, bạn cần tìm đường đi có tổng số cộng nhỏ nhất để đi qua tất cả các cạnh ít nhất một lần.

Dưới đây là một phương pháp đơn giản để giải quyết bài toán đưa thư (Vấn đề người đưa thư ngắn nhất) trong một sơ đồ vô hướng:

* Tính toán các đỉnh có bậc thang (đỉnh có số lẻ cạnh nối) : Bắt đầu bằng cách xác định tất cả các bậc cao cấp trong sơ đồ có bậc thang (số cạnh nối là số lẻ). Điều này đảm bảo rằng bạn phải đạt được những phẩm chất này ít nhất một lần.
* Tìm tất cả các cặp đỉnh có bậc lẻ : Tạo một danh sách các cặp đỉnh có bậc lẻ mà bạn cần kết nối bằng cách tìm các cạnh ngắn nhất hoặc đường đi ngắn nhất giữa chúng (ví dụ: Dijkstra hoặc thuật toán tìm đường đi ngắn nhất).

- Kết quả hợp nhất tất cả các đường đi : Sử dụng một thuật toán tìm đường đi ngắn nhất như thuật toán Dijkstra để tìm đường đi ngắn nhất từ ​​một trong các đỉnh chưa kết nối tới mọi cặp đỉnh có bậc lẻ được tìm thấy trong bước trước. Kết hợp tất cả các đường đi này để tạo đường đi tổng hợp ngắn nhất.

**3.1 Tìm đường đi, chu trình Euler**

Để tìm đường đi hoặc chu trình Euler trong một đồ thị, bạn có thể sử dụng thuật toán Hierholzer nếu đồ thị có hướng hoặc thuật toán Fleury nếu đồ thị không hướng. Dưới đây là cách tìm đường đi hoặc chu trình Euler bằng cả hai phương pháp này:

\*Tìm đường đi hoặc chu trình Euler trong đồ thị có hướng (Thuật toán Hierholzer):

Kiểm tra tính tồn tại: Trước tiên, hãy kiểm tra xem đồ thị có tồn tại chu trình Euler hoặc đường đi Euler không. Điều này đòi hỏi đồ thị phải liên thông và có tối đa một đỉnh có bậc ra - bậc vào khác nhau (nếu đồ thị có chu trình Euler) hoặc không có đỉnh nào có bậc ra - bậc vào khác nhau (nếu đồ thị có đường đi Euler).

Lựa chọn đỉnh xuất phát: Chọn một đỉnh xuất phát, bắt đầu từ đó.

**Thực hiện thuật toán Hierholzer:**

* Bắt đầu từ đỉnh xuất phát, đi theo các cạnh chưa được thăm.
* Nếu bạn gặp một đỉnh chưa từng thăm, hãy theo đó và tiếp tục tạo một chu trình con.
* Nếu bạn không còn cạnh nào để đi, quay lại các chu trình con đã tạo và nối chúng lại với nhau tại các đỉnh chung để tạo ra chu trình Euler hoặc đường đi Euler.

\*Tìm đường đi hoặc chu trình Euler trong đồ thị không hướng (Thuật toán Fleury):

Kiểm tra tính tồn tại: Kiểm tra xem đồ thị có tồn tại chu trình Euler hoặc đường đi Euler không. Điều này đòi hỏi đồ thị phải liên thông và có tối đa hai đỉnh có bậc lẻ (đỉnh có số lẻ cạnh nối).

Lựa chọn đỉnh xuất phát: Chọn một đỉnh bất kỳ làm đỉnh xuất phát.

**Thực hiện thuật toán Fleury:**

* Bắt đầu từ đỉnh xuất phát, đi qua một cạnh chưa được thăm và loại bỏ cạnh đó (đánh dấu cạnh này để không sử dụng lại).
* Lặp lại quá trình trên cho tất cả các cạnh chưa được thăm, nhưng cẩn thận để không tạo ra cầu (nếu tạo ra cầu, hãy sử dụng cạnh khác).
* Khi bạn không thể đi tiếp nữa, quay lại và tiếp tục tìm đường đi cho đến khi bạn tạo ra chu trình Euler hoặc đường đi Euler.

**3.2 Giải thoát bài toán đưa thư**

Bài toán "đưa thư" hoặc "bài đưa thư" là một dạng bài toán tối ưu hóa trong lĩnh vực quản lý chuỗi cung ứng và phân phát. Mục tiêu của bài toán này là tìm cách phân phát thư hoặc hàng hóa từ một điểm đến nhiều điểm một cách hiệu quả, sao cho tổng chi phí hoặc thời gian là tối thiểu.

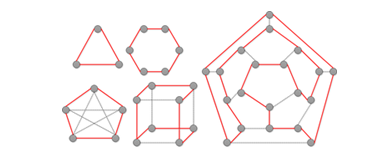
Dưới đây là một số cách để giải quyết bài toán đưa thư:

* Giải thuật TSP (Traveling Salesman Problem): Bài toán đưa thư có thể coi là một biến thể của bài toán du lịch (TSP), trong đó một người phải du lịch qua tất cả các địa điểm khách hàng và quay trở về nơi xuất phát. Bạn có thể sử dụng giải thuật TSP hoặc các biến thể của nó để tìm đường đi tối ưu.
* Giải thuật Dijkstra và A cho đường đi ngắn nhất\*: Nếu bạn quan tâm đến việc tìm đường đi ngắn nhất từ một điểm xuất phát đến tất cả các điểm đến, bạn có thể sử dụng giải thuật Dijkstra hoặc giải thuật A\*.
* Giải thuật phân phối và lập lịch: Nếu bạn đang quản lý một mạng lưới phân phối hoặc lập kế hoạch phân phát, bạn có thể sử dụng các giải thuật tối ưu hóa dựa trên giải thuật lập lịch và phân phối. Các giải thuật như giải thuật Dễ dàng nhất trước (EDD) và giải thuật lập lịch xe (Vehicle Scheduling) có thể được sử dụng để tối ưu hóa lịch trình phân phát.
* Sử dụng phần mềm tối ưu hóa: Có nhiều phần mềm tối ưu hóa dành riêng cho việc giải quyết bài toán đưa thư và quản lý chuỗi cung ứng. Các phần mềm này cung cấp các công cụ mạnh mẽ để tối ưu hóa lịch trình phân phát và quản lý việc đưa thư hoặc hàng hóa.
* Sử dụng hệ thống thông tin địa lý (GIS): Công nghệ thông tin địa lý (GIS) có thể được sử dụng để xác định tối ưu hóa lịch trình và đường đi dựa trên thông tin địa lý, giao thông và các ràng buộc khác.

Bài toán đưa thư có thể rất phức tạp và đa dạng, tùy thuộc vào quy mô và yêu cầu cụ thể. Việc chọn giải pháp phụ thuộc vào quy mô của bài toán, yêu cầu về thời gian, khoảng cách, và ràng buộc khác.

# ĐỒ THỊ HAMILTON, THUẬT TOÁN DIJKSTRA

Lý thuyết thị đồ là một lĩnh vực quan trọng trong toán học, và đồ thị Hamilton là một khái niệm đặc biệt quan trọng trong lĩnh vực này. Đồ thị Hamilton liên quan đến việc tìm kiếm đường đi hoặc chu trình đi qua mỗi đỉnh cao của đồ thị một lần và chỉ một lần.



**Hình 7**

##### Đường đi Hamilton

Đường Đi Hamilton là một loại đường đi trong lý thuyết sơ đồ, nơi một công cụ đỉnh cao của đồ thị được đi qua một lần và chỉ một lần. Đường Đi Hamilton được đặt tên theo nhà toán học và nhà vật lý William Rowan Hamilton.

Một Đường Đi Hamiltonian trên một sơ đồ là một dãy các đỉnh khác nhau sao cho có một cạnh giữa mọi cặp đỉnh tuần nhau trong dãy. Nếu xuất phát và kết thúc của đường đi đó là cùng một chất lượng, ta gọi nó là Chu trình Hamilton (Chu trình Hamilton).

Chú ý rằng tìm đường đi Hamilton trên một sơ đồ là một vấn đề NP-đầy đủ, tức là không có giải thuật đa thức nào được biết để giải quyết nó trong thời gian hợp lý khi số lượng đồ thị lớn nhất.

Đường Đi Hamilton có ứng dụng trong nhiều lĩnh vực, có giới hạn như quy hoạch động, thiết kế mạch điện và các vấn đề liên quan đến đường đi tối ưu hóa qua các điểm trong mạng .

##### **Chu Trình Hamilton**

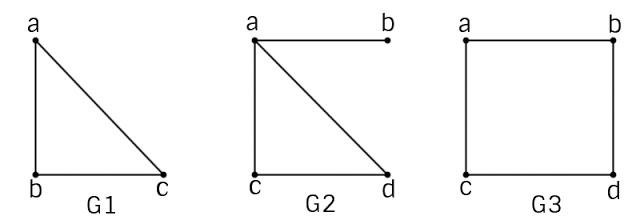
Chu Trình Hamilton là một khái niệm trong lý thuyết đồ thị, đặc biệt là trong lý thuyết đồ thị không hướng. Một Chu Trình Hamilton trên một đồ thị là một chuỗi các đỉnh khác nhau sao cho có một cạnh giữa mọi đỉnh viền quanh nhau trong chuỗi, và đỉnh xuất phát và kết thúc là cùng một đỉnh.

Nói một cách khác, một Chu Trình Hamilton là một đường đi Hamiltonian đóng, nơi mỗi đỉnh cao trong đồ thị được đi qua một lần và chỉ một lần, và cuối cùng quay trở lại xuất phát đỉnh cao để tạo thành một chu trình.

Tìm kiếm Chu Trình Hamilton trên một sơ đồ là một vấn đề NP-đầy đủ, điều này có nghĩa là không có giải pháp hiệu quả nhất được biết để giải quyết nó trong thời gian đa thức đối với tất cả các sơ đồ. Do đó, việc xác định sự tồn tại hoặc không tồn tại trong chu trình Hamilton là một nhiệm vụ sơ bộ và Đòi hỏi các phương pháp thông minh và tối ưu hóa.

Chu Trình Hamilton có ứng dụng trong nhiều lĩnh vực như quy trình lịch trình, mạng lưới tối ưu hóa và các vấn đề liên quan đến kết nối trong hệ thống phức tạp

**Ví dụ**. Đồ thị hamilton G1, G3, nửa Hamilton G2.

**

*Đường đi và chu trình Hamilton*

**Hình 8**

Cho đến nay, việc tìm ra một tiêu chuẩn để nhận biết đồ thị Hamilton vẫn còn mở, mặc dù đây là vấn đề trung tâm của lý thuyết đồ thị. Hơn thế nữa, cho đến nay cũng vẫn chưa có thuật toán hiệu quả để kiểm tra một đồ thị có phải là đồ thị Hamilton hay không.

Để liệt kê tất cả các chu trình Hamilton của đồ thị, chúng ta có thể sử dụng thuật toán sau:

void Hamilton( int k) {

/\* Liệt kê các chu trình Hamilton của đồ thị bằng cách phát triển dãy đỉnh

(X[1], X[2],..., X[k-1] ) của đồ thị G = (V, E) \*/

for y∈Ke(X[k-1]) {

if (k==n+1) and (y == v0) then

Ghinhan(X[1], X[2],..., X[n], v0);

else {

X[k]=y; chuaxet[y] = false;

Hamilton(k+1);

chuaxet[y] = true;

}

}

}

**Chương trình chính chính thể hiện như sau:**

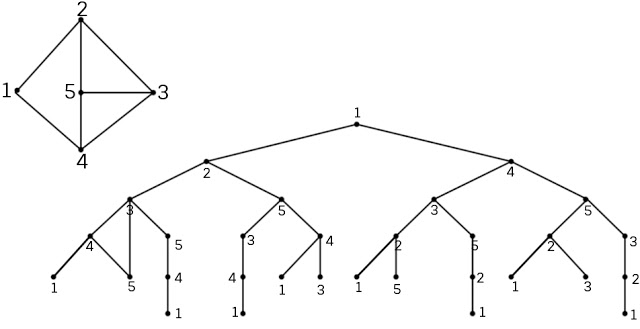
for (v∈V ) chuaxet[v] = true; /\*thiết lập trạng thái các đỉnh\*/

X[1] = v0; (\*v0 là một đỉnh nào đó của đồ thị\*)

chuaxet[v0] = false;

Hamilton(2);

### Cây tìm kiếm chu trình Hamilton thể hiện thuật toán trên được mô tả như sau:

**

*Thuật toán tìm chu trình Hamition*

**Hình 9**

### Chương trình liệt kê các chu trình Hamilton được thể hiện như sau:

[#include](https://expressmagazine.net/hashtags/include)<iostream>

[#include](https://expressmagazine.net/hashtags/include)<conio.h>

using namespace std;

[#define](https://expressmagazine.net/hashtags/define) MAX 50

[#define](https://expressmagazine.net/hashtags/define) TRUE 1

[#define](https://expressmagazine.net/hashtags/define) FALSE 0

int A[MAX][MAX];//ma trận liền kề.

int C[MAX], B[MAX];

int n;//số đỉnh của đồ thị.

int d;//đếm số lượng chu trình hamilton.

void Init(void){

freopen("CCHMTON.IN", "r",stdin);

cin>>n;

//nhập ma trận kề

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

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

cin>>A[i][j];

}

}

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

C[i]=0;

}

void Result(void){

cout<<"Chu trinh Hamilton: ";

for(int i=n; i>=0; i--)

cout<<B[i]<<" ";

d++;

cout<<endl;

}

void Hamilton(int \*B, int \*C, int i){

int j, k;

for(j=1; j<=n; j++){

if(A[B[i-1]][j]==1 && C[j]==0){

B[i]=j; C[j]=1;

if(i<n) Hamilton(B, C, i+1);

else

if(B[i]==B[0]) Result();

C[j]=0;

}

}

}

void main(void){

B[0]=1;

int i=1;

d=0;

Init();

Hamilton(B,C,i);

if(d==0)

cout<<"Khong co chu trinh Hamilton";

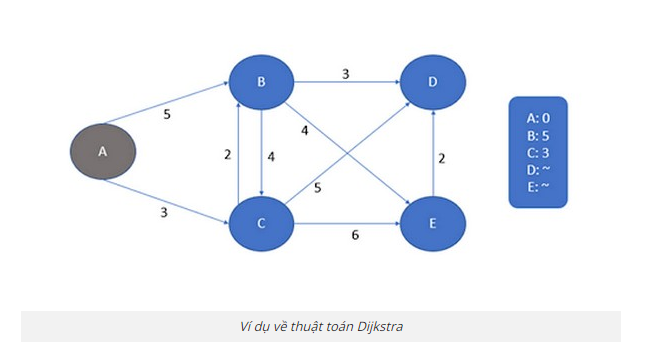
getch();

}

##### Thuật toán tìm đường đi ngắn nhất Dijkstra

Thuật toán tìm kiếm đường đi ngắn nhất Dijkstra là một thuật toán được quyết định sử dụng để tìm đường đi ngắn nhất từ ​​​​một sản phẩm xuất phát đến tất cả các hạng còn lại trong sơ đồ có số dương quan trọng. Đây là một công cụ tìm kiếm thuật toán không có đường dẫn định nghĩa trạng thái ngắn nhất giữa các đỉnh cao trong sơ đồ

**Ví dụ:**



**Hình 10**

Ví dụ, để biểu diễn đường đi ngắn nhất từ thành phố A đến thành phố B, chúng ta dùng các đỉnh của đồ thị để thị phạm các thành phố và các cạnh để biểu diễn các đường nối giữa chúng. Trọng số các cạnh sẽ được xem như độ dài của các con đường, vì vậy mà chúng không âm, nhờ đó thuật toán sẽ chỉ ra con đường ngắn nhất.

Dưới đây là mô tả chi tiết của thuật toán:

import heapq

def dijkstra(graph, start):

dist = {node: float('infinity') for node in graph}

dist[start] = 0

pq = [(0, start)]

while pq:

current\_dist, current\_node = heapq.heappop(pq)

if current\_dist > dist[current\_node]:

continue

for neighbor, weight in graph[current\_node].items():

distance = current\_dist + weight

if distance < dist[neighbor]:

dist[neighbor] = distance

heapq.heappush(pq, (distance, neighbor))

return dist