LÝ THUYẾT ĐỒ THỊ

MỤC LỤC

MỤC LỤC 1

HƯỚNG DẪN 4

BÀI 1: ĐẠI CƯƠNG VỀ ĐỒ THỊ 6

MỘT SỐ ĐỊNH NGHĨA 6

1.1.1 Đồ thị là gì? 6

1.1.2 Cạnh song song và khuyên 8

1.1.3 Một số dạng đồ thị đặc biệt 9

1.1.4 Đỉnh kề 11

1.1.5 Bậc của đỉnh 12

1.1.6 Đẳng cấu đồ thị 14

1.1.7 Đồ thị con – Đồ thị bộ phận 15

1.1.8 Dây chuyền, chu trình, đường đi và mạch 16

BIỂU DIỄN ĐỒ THỊ 18

1.1.9 Ma trận kề 18

1.1.10 Ma trận liên thuộc 20

THÀNH PHẦN LIÊN THÔNG 21

1.1.11 Định nghĩa quan hệ liên kết ~ 21

1.1.12 Định nghĩa thành phần liên thông 21

1.1.13 Thuật toán xác định các thành phần liên thông trong đồ thị 22

TÓM TẮT 24

BÀI TẬP 25

BÀI 2: ĐỒ THỊ EULER VÀ ĐỒ THỊ HAMILTON 26

ĐỒ THỊ EULER 26

2.1.1 Một số định nghĩa 27

2.1.2 Định lý Euler 30

2.1.3 Thuật toán Fleury 30

2.1.4 Bài toán người phát thư Trung Hoa 32

ĐỒ THỊ HAMILTON 35

2.1.5 Định nghĩa 36

2.1.6 Các định lý 37

2.1.7 Quy tắc xác định chu trình Hamilton 43

TÓM TẮT 45

BÀI TẬP 46

BÀI 3: CÂY 48

ĐỊNH NGHĨA VÀ CÁC TÍNH CHẤT CƠ BẢN 48

3.1.1 Định nghĩa 48

3.1.2 Các tính chất cơ bản 49

CÂY KHUNG VÀ BÀI TOÁN CÂY KHUNG NGẮN NHẤT 51

3.1.3 Định nghĩa cây khung 51

3.1.4 Thuật toán tìm cây khung 51

3.1.5 Cây khung ngắn nhất 52

CÂY CÓ GỐC 55

3.1.6 Đồ thị có gốc 55

3.1.7 Cây có hướng 56

3.1.8 Cây có gốc 57

DUYỆT CÂY NHỊ PHÂN 59

3.1.9 Định nghĩa 59

3.1.10 Các phương pháp duyệt cây nhị phân 60

TÓM TẮT 66

BÀI TẬP 67

BÀI 4: CÁC BÀI TOÁN ĐƯỜNG ĐI 69

GIỚI THIỆU 69

THUẬT TOÁN DIJKSTRA 70

4.1.1 Ý tưởng thuật toán Dijkstra 70

4.1.2 Các bước thuật toán Dijkstra 70

THUẬT TOÁN FLOYD 76

TÓM TẮT 80

BÀI TẬP 81

BÀI 5: BÀI TOÁN GHÉP CẶP 83

BÀI TOÁN GHÉP CẶP 83

5.1.1 Giới thiệu 83

5.1.2 Định lý Hall 83

BÀI TOÁN GIAO VIỆC TỐI ƯU 87

BÀI TOÁN GIAO VIỆC CỦA GALE 90

5.1.3 Giới thiệu 90

5.1.4 Định nghĩa 90

TÓM TẮT 92

PHỤ LỤC 93

BÀI TẬP 1: NHẬP XUẤT MA TRẬN KỀ TỪ FILE 93
BÀI TẬP 2: ĐỒ THỊ LIÊN THÔNG 100
BÀI TẬP 3: CHU TRÌNH EULER 103
BÀI TẬP 4: ĐƯỜNG ĐI EULER 107
BÀI TẬP 5: TÌM KIẾM ĐƯỜNG ĐI BẰNG PHƯƠNG PHÁP DUYỆT THEO CHIỀU SÂU (DFS) 110
BÀI TẬP 6: TÌM KIẾM ĐƯỜNG ĐI BẰNG PHƯƠNG PHÁP DUYỆT THEO CHIỀU RỘNG (BFS) 114
BÀI TẬP 7: TÌM CÂY KHUNG NHỎ NHẤT VỚI THUẬT TOÁN PRIM 119
BÀI TẬP 8: TÌM CÂY KHUNG NHỎ NHẤT VỚI THUẬT TOÁN KRUSKAL 123
BÀI TẬP 9: TÌM ĐƯỜNG ĐI NGẮN NHẤT VỚI THUẬT TOÁN DIJKSTRA 127
BÀI TẬP 10: TÌM KIẾM ĐƯỜNG ĐI NGẮN NHẤT VỚI THUẬT TOÁN FLOYD 131
TÀI LIỆU THAM KHẢO 135

HƯỚNG DẪN

MÔ TẢ MÔN HỌC

Lý thuyết đồ thị là một ngành khoa học được phát triển từ lâu nhưng lại có nhiều ứng dụng hiện đại. Những ý tưởng cơ bản của nó được đưa ra từ thế kỷ 18 bởi nhà toán học Thụy Sĩ tên là Leonhard Euler. Ông đã dùng đồ thị để giải quyết bài toán 7 chiếc cầu Konigsberg nổi tiếng.

Môn học cung cấp cho sinh viên kiến thức về đồ thị và những thuật toán xử lý trên đồ thị. Từ đó có thể áp dụng vào một số bài toán mang tính thực tế cao, ví dụ như: nếu dùng mô hình đồ thị mạng máy tính có thể xác định xem hai máy tính có được nối với nhau bằng một đường truyền thông hay không, đồ thị với các trọng số được gán cho các cạnh của nó có thể dùng để giải các bài toán như bài toán tìm đường đi ngắn nhất giữa hai thành phố trong một mạng giao thông, ….

NỘI DUNG MÔN HỌC

- Bài 1: Đại cương về đồ thị
- Bài 2: Đồ thị Euler và đồ thị Hamilton
- Bài 3: Cây
- Bài 4: Các bài toán về đường đi
- Bài 5: Bài toán ghép cặp

KIẾN THỨC TIỀN ĐỀ

Môn học đòi hỏi sinh viên có một số kiến thức cơ bản về Toán rời rạc và Kỹ thuật lập trình trên ngôn ngữ C++ hoặc Java.

YÊU CẦU MÔN HỌC

Người học phải dự học đầy đủ các buổi lên lớp lý thuyết, thực hành và làm bài tập đầy đủ ở nhà.

CÁCH TIẾP NHẬN NỘI DUNG MÔN HỌC

Để học tốt môn này, sinh viên cần tập trung tiếp thu bài giảng tại lớp, theo dõi các ví dụ minh hoạ cũng như cố gắng tự hoàn thành các bài tập tương tự. Giờ thực hành cần nắm được kỹ thuật lập trình của từng thuật toán.

PHƯƠNG PHÁP ĐÁNH GIÁ MÔN HỌC

Môn học được đánh giá gồm:

- Điểm quá trình: 30%. Hình thức và nội dung do giảng viên quyết định, phù hợp với quy chế đào tạo và tình hình thực tế tại nơi tổ chức học tập.
- Điểm thi: 70%. Hình thức bài thi tự luận. Nội dung gồm các bài tập thuộc bài thứ 1 đến bài thứ 6 của môn học.

BÀI 1: ĐẠI CƯƠNG VỀ ĐỒ THỊ

Sau khi học xong bài này, học viên có thể:

- *Hiểu được đồ thị là gì? Thế nào là đồ thị có hướng và đồ thị vô hướng, một số dạng đồ thị đặc biệt như đồ thị rỗng, đồ thị đơn, đồ thị đủ, đồ thị lưỡng phân.*
- *Hiểu được thế nào là mối quan hệ kề giữa đỉnh và cạnh, giữa đỉnh và đỉnh. Từ đó có thể tính được bậc của một đỉnh trong đồ thị.*
- *Biết được phương pháp biểu diễn một đồ thị thông qua ma trận kề, ma trận liên thuộc...*
- *Hiểu được thế nào là dây chuyền, đường đi, chu trình và mạch một đồ thị.*
- *Thông qua thuật toán xác định các thành phần liên thông trong một đồ thị để có thế kết luận đồ thị đang khảo sát có phải là một đồ thị liên thông hay không.*

MỘT SỐ ĐỊNH NGHĨA

1.1.1 Đồ thị là gì?

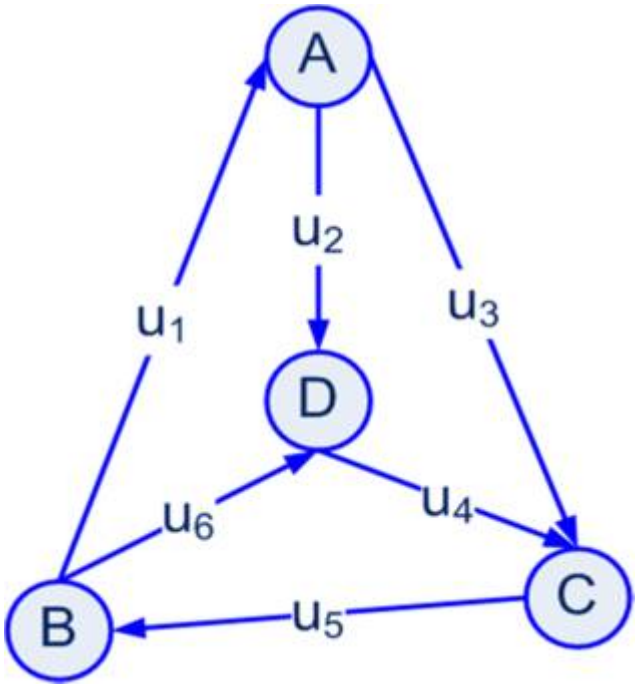
Đồ thị là một cấu trúc rời rạc gồm các đỉnh và các cạnh (vô hướng hoặc có hướng) nối các đỉnh đó. Người ta phân loại đồ thị tùy theo đặc tính và số các cạnh nối các cặp đỉnh của đồ thị.

1.1.1.1 Đồ thị có hướng

Một đồ thị có hướng G=(X, U) được định nghĩa bởi:

- Tập hợp $X \neq \varnothing$ được gọi là tập các đỉnh của đồ thị;
- Tập hợp U là tập các cạnh của đồ thị;

Mỗi cạnh $u \in U$ được liên kết với một cặp đỉnh $(i, j) \in X^2$. Trong đó mỗi cạnh u quy định một hướng đi từ đỉnh i đến đỉnh j.



Hình 1.1 Đồ thị có hướng

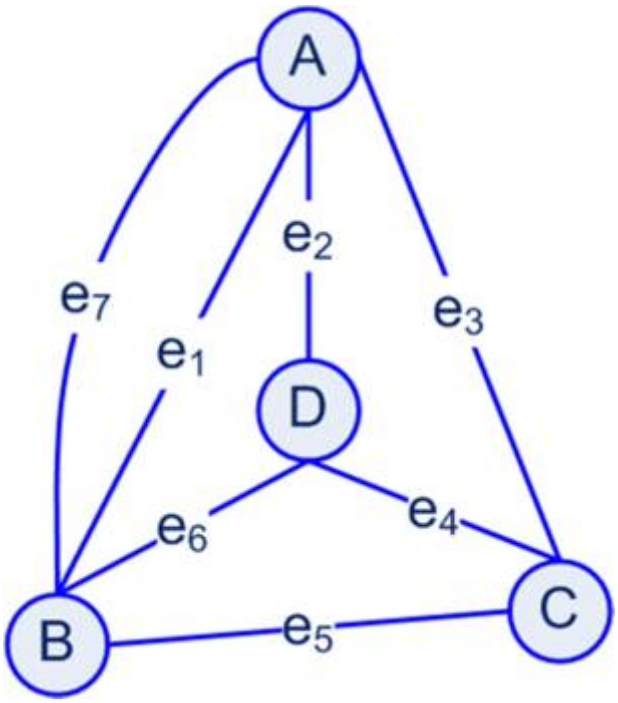
Hình 1.1 là ví dụ về một đồ thị có hướng G với tập đỉnh X gồm 4 phần tử là A, B, C, D; tập cạnh U của đồ thị gồm có 6 phần tử lần lượt là u_1, u_2, u_3, u_4, u_5 và u_6. Trong đó u_1 là cạnh nối 2 đỉnh A và B với hướng đi từ đỉnh B đến đỉnh A, ta viết là u_1= (B, A). Tương tự ta sẽ có các cạnh u_2= (A, D), u_3= (A, C)….

1.1.1.2 Đồ thị vô hướng

Một đồ thị vô hướng G=(X, E) được định nghĩa bởi:

- Tập hợp $X \neq \varnothing$ được gọi là tập các đỉnh của đồ thị;
- Tập hợp E là tập các cạnh của đồ thị;

Mỗi cạnh $e \in E$ được liên kết với một cặp đỉnh $\{i, j\} \in X^2$, không phân biệt thứ tự.



Hình 1.2 Đồ thị vô hướng

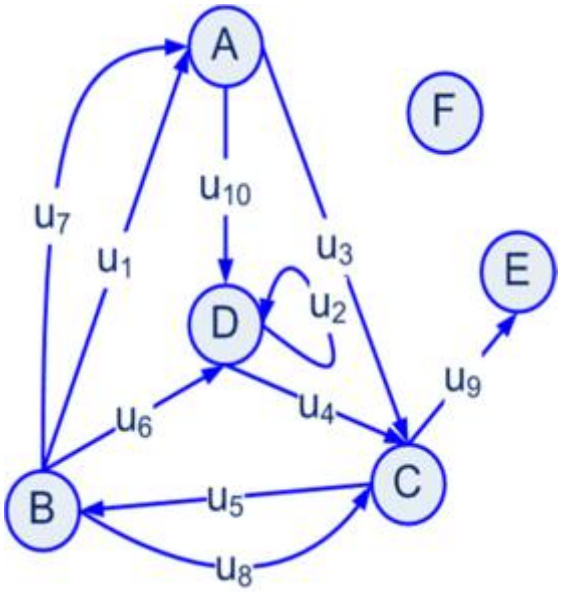
Hình 1.2 biểu diễn một đồ thị vô hướng G với tập đỉnh X gồm 4 phần tử là A, B, C, D; tập cạnh E của đồ thị gồm có 7 phần tử lần lượt là e_1, e_2, e_3, e_4, e_5, e_6 và e_7. Trong đó e_1 là cạnh nối 2 đỉnh A và B. Khác với đồ thị có hướng thì trong đồ thị vô hướng do các cạnh là không có quy định về hướng nên e_1 có thể viết là e_1= (A, B) hoặc (B, A) đều được.

1.1.2 Cạnh song song và khuyên

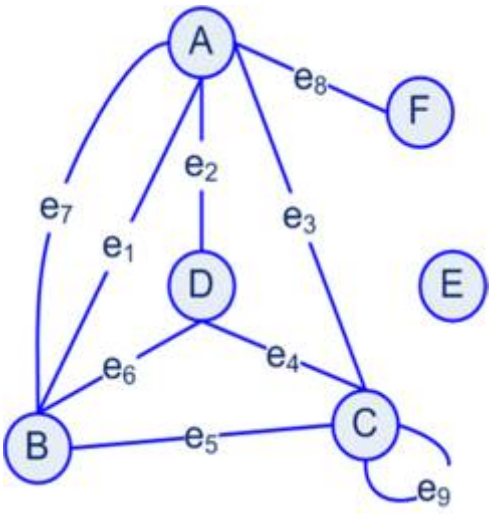
Trong một đồ thị G.

Nếu cạnh u liên kết cặp đỉnh (i, i) thì u được gọi là khuyên.

Nếu hai cạnh u và v cùng liên kết với cặp đỉnh (i, j) thì chúng được gọi là song song với nhau.



Hình 1.3 Cạnh song song và khuyên trong đồ thị có hướng



Hình 1.4 Cạnh song song và khuyên trong đồ thị vô hướng

Hình 1.3 có 2 cạnh song song với nhau là u_1 và u_7; cạnh u_2 là khuyên. Còn hình 1.4 có 2 cạnh e_1 và e_7 là song song với nhau; e_9 là khuyên.

1.1.3 Một số dạng đồ thị đặc biệt

1.1.3.1 Đồ thị rỗng

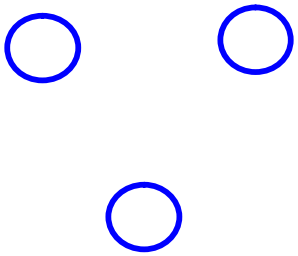
Đồ thị rỗng là đồ thị có tập cạnh là tập rỗng.

1.1.3.2 Đồ thị đơn

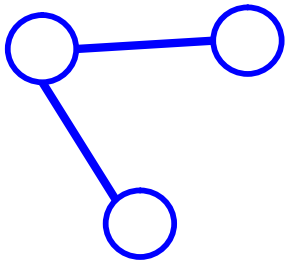
Đồ thị đơn là đồ thị không có khuyên và cạnh song song.

1.1.3.3 Đồ thị đủ

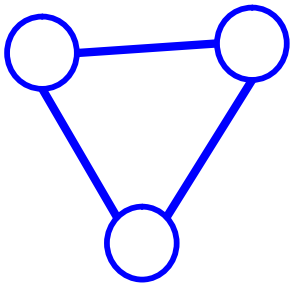
Đồ thị đủ là đồ thị vô hướng, đơn, trong đó giữa hai đỉnh bất kỳ đều có đúng một cạnh liên kết chúng với nhau.



Hình 1.5 Đồ thị rỗng



Hình 1.6 Đồ thị đơn

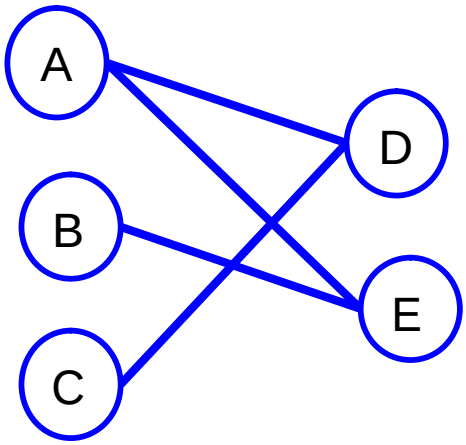


Hình 1.7 Đồ thị đủ

1.1.3.4 Đồ thị lưỡng phân

Đồ thị G=(X, E) được gọi là đồ thị lưỡng phân nếu tập X được chia thành hai tập X_1 và X_2 thỏa:

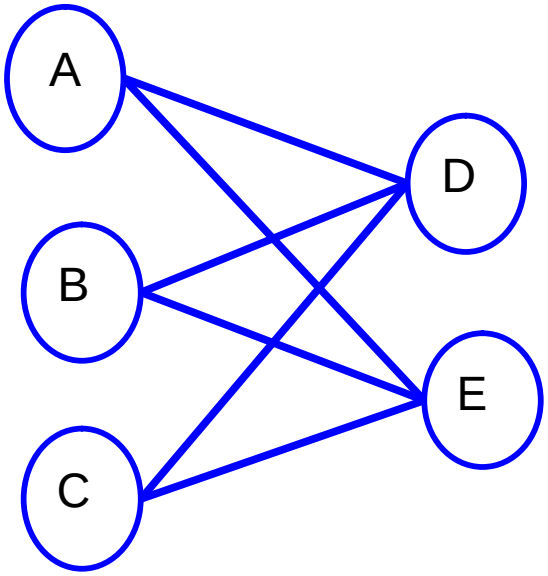
- X_1 và X_2 phân hoạch X;
- Cạnh chỉ nối giữa X_1 và X_2.



Hình 1.8 Đồ thị lưỡng phân

1.1.3.5 Đồ thị lưỡng phân đủ

Đồ thị lưỡng phân đủ: là đồ thị lưỡng phân đơn, vô hướng thỏa điều kiện $\forall$(i, j) với $i \in X_1$ và $j \in X_2$ có đúng một cạnh i và j. Nếu $|X_1| = N$ và $|X_2| = M$, ký hiệu $K_{M,N}$.

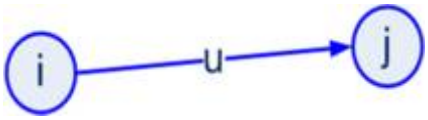


Hình 1.9 Đồ thị lưỡng phân đủ

1.1.4 Đỉnh kề

1.1.4.1 Trên đồ thị có hướng

Xét cạnh u được liên kết với cặp đỉnh (i, j):

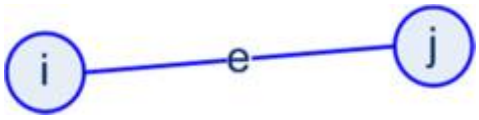


Hình 1.10 Liên kết đỉnh cạnh có hướng

- Cạnh u kề với đỉnh i và đỉnh j (hay đỉnh i và đỉnh j kề với cạnh u); có thể viết tắt u=(i, j). Cạnh u đi ra khỏi đỉnh i và đi vào đỉnh j.
- Đỉnh j được gọi là đỉnh kề của đỉnh i.

1.1.4.2 Trên đồ thị vô hướng

Xét cạnh e được liên kết với cặp đỉnh (i, j):



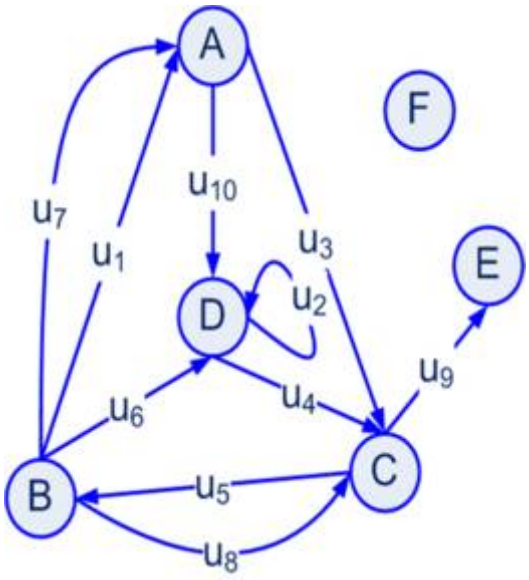
Hình 1.11 Liên kết đỉnh cạnh vô hướng

- Cạnh e kề với đỉnh i và đỉnh j (hay đỉnh i và đỉnh j kề với cạnh e); có thể viết tắt e=(i, j).
- Đỉnh i và đỉnh j được gọi là 2 đỉnh kề nhau (hay đỉnh i kề với đỉnh j và ngược lại, đỉnh j kề với đỉnh i).

1.1.5 Bậc của đỉnh

1.1.5.1 Trên đồ thị có hướng

- Nửa bậc ngoài của đỉnh x là số các cạnh đi ra khỏi đỉnh x, ký hiệu $d^+(x)$.
- Nửa bậc trong của đỉnh x là số các cạnh đi vào đỉnh x, ký hiệu $d^-(x)$.
- Bậc của đỉnh x: $d(x)=d^+(x)+d^-(x)$.

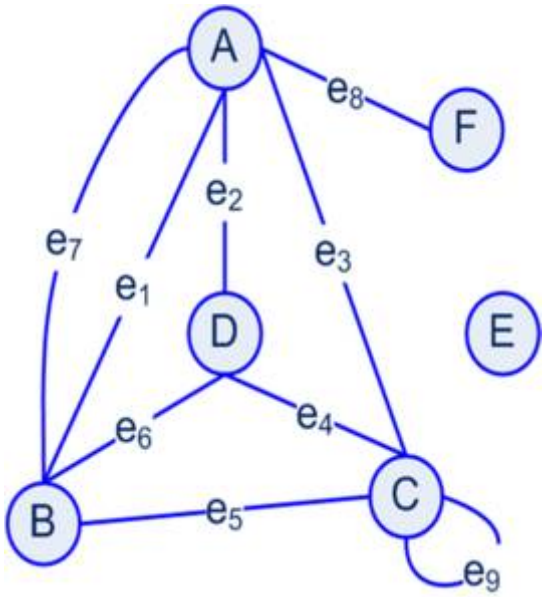


Hình 1.12 Bậc của đỉnh trong đồ thị có hướng

Đỉnh A có $d^+(A)=2$, $d^-(A)=2$ nên có bậc là 4. Đỉnh D có $d^+(D)=2$, $d^-(D)=3$ nên đỉnh D có bậc là 5. Đỉnh E có bậc là 1 và F có bậc là 0.

1.1.5.2 Trên đồ thị vô hướng

Bậc của đỉnh x trong đồ thị G là số các cạnh kề với đỉnh x, mỗi khuyên được tính hai lần, ký hiệu là dG(x) (hay d(x) nếu đang xét một đồ thị nào đó).



Hình 1.13 Bậc của đỉnh trong đồ thị vô hướng

Do đỉnh A có 5 cạnh kề nên bậc của đỉnh A là 5. Đỉnh C có 4 cạnh kề trong đó có 1 cạnh là khuyên nên theo quy ước đỉnh C sẽ có bậc bằng 5 vì 1 khuyên sẽ được tính hai lần. Đỉnh F có bậc 1 và bậc của E là 0.

1.1.5.3 Đỉnh treo và đỉnh cô lập

Từ cách tính bậc của đồ thị người ta đưa ra định nghĩa về đỉnh treo và đỉnh cô lập

- Đỉnh treo là đỉnh có bậc bằng 1.
- Đỉnh cô lập là đỉnh có bậc bằng 0.

Ở hình 1.12 có đỉnh treo là đỉnh E, đỉnh cô lập là đỉnh F.

1.1.5.4 Mối liên hệ bậc và số cạnh

Định lý

- Xét đồ thị có hướng G=(X, U). Ta có:
  - $\sum_{x\in X} d^+(x) = \sum_{x\in X} d^-(x)$ và $\sum_{x\in X} d(x) = 2|U|$
- Xét đồ thị vô hướng G=(X, E). Ta có:
  - $\sum_{x\in X} d(x) = 2|E|$

Với |U| và |E| là số cạnh của đồ thị.

Hệ quả

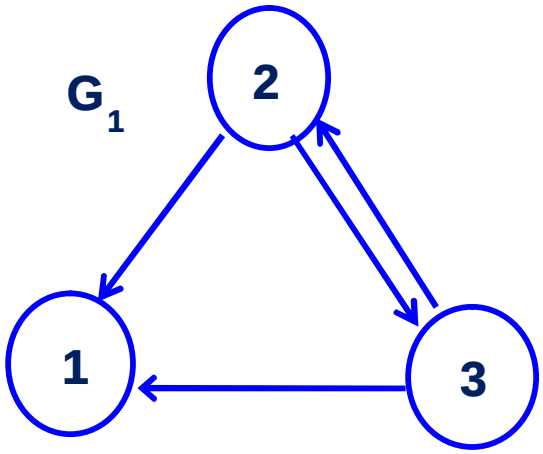
Số lượng các đỉnh có bậc lẻ trong một đồ thị là một số chẵn.

1.1.6 Đẳng cấu đồ thị

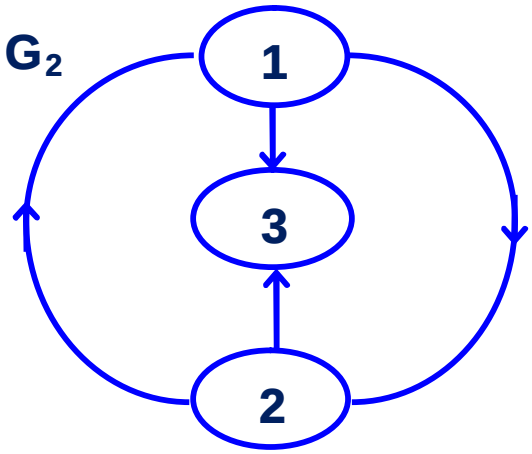
1.1.6.1 Đồ thị có hướng

Hai đồ thị có hướng $G_1=(X_1, U_1)$ và $G_2=(X_2, U_2)$ được gọi là đẳng cấu với nhau nếu tồn tại hai song ánh ψ và δ thỏa mãn điều kiện:

- ψ: X1 $\rightarrow$ X2 và δ: U1 $\rightarrow$ U2
- Nếu cạnh u $\in$ U1 liên kết với cặp đỉnh (x, y) $\in$ X1 trong G1 thì cạnh δ(u) sẽ liên kết với cặp đỉnh (ψ(x), ψ(y)) trong G2 (sự tương ứng cạnh).



Hình 1.14 Đồ thị đẳng cấu



Hình 1.15 Đồ thị đẳng cấu

Nhận xét: Ở đồ thị G_1, đỉnh 1 có bậc là 2 còn đỉnh 2 và 3 có bậc là 3. Ở đồ thị G_2, đỉnh 1 và 2 có bậc là 3 còn đỉnh 3 có bậc 2. Như vậy qua phép song ánh ψ biến đỉnh 1 trong G_1 thành đỉnh 3 trong G_2, đỉnh 3 thành đỉnh 2 và đỉnh 2 thành đỉnh 1. Tương ứng các cạnh (2,1), (3,1), (3,2), (2,3) trong G_1 sẽ trở thành (1, 3), (2, 3), (2, 1) và (1, 2). Đó là sự tương ứng cạnh. G1 và G2 là đẳng cấu với nhau.

1.1.6.2 Đồ thị vô hướng

Hai đồ thị vô hướng $G_1=(X_1, E_1)$ và $G_2=(X_2, E_2)$ được gọi là đẳng cấu với nhau nếu tồn tại hai song ánh ψ và δ thỏa mãn điều kiện:

- ψ: X1 $\rightarrow$ X2 và δ: E1 $\rightarrow$ E2

- Nếu cạnh e ∈ E1 kề với cặp đỉnh {x, y} ⊆ X1 trong G1 thì cạnh δ(e) sẽ kề với cặp đỉnh {ψ(x), ψ(y)} trong G2 (sự tương ứng cạnh).

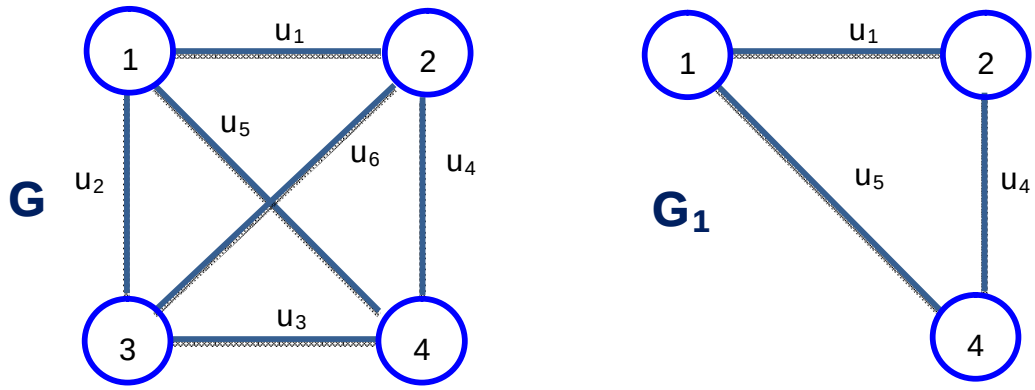
Nếu bỏ qua hướng của các cạnh trong đồ thị G_1 và G_2 ở 2 hình 1.14 và 1.15 ta sẽ có 2 đồ thị đẳng cấu với nhau với sự tương ứng cạnh như trên.

1.1.7 Đồ thị con – Đồ thị bộ phận

1.1.7.1 Đồ thị con

Xét hai đồ thị G=(X, U) và $G_1=(X_1, U_1)$. G_1 được gọi là đồ thị con của G và ký hiệu $G_1 \leq G$ nếu:

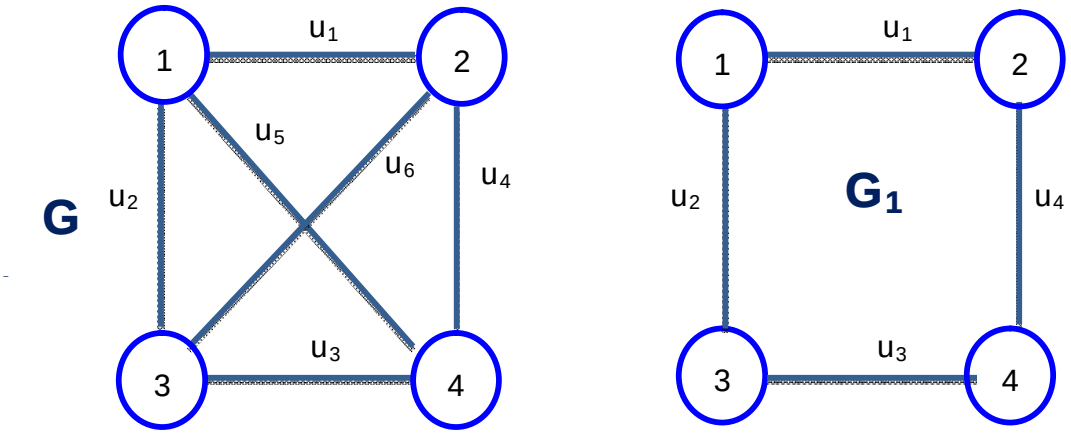
- $X1 \subseteq X$; $U1 \subseteq U$
- $\forall u=(i, j) \in U$ của G, nếu $u \in U1$ thì $i, j \in X1$



Hình 1.16 Đồ thị con

1.1.7.2 Đồ thị bộ phận

Đồ thị con $G_1=(X_1, U_1)$ của đồ thị G=(X, U) được gọi là đồ thị bộ phận của G nếu X=X1.

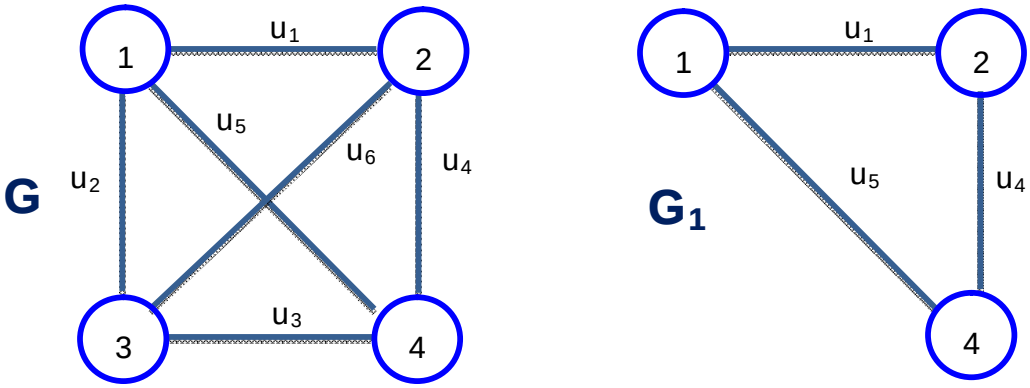


Hình 1.17 Đồ thị bộ phận

1.1.7.3 Đồ thị con sinh bởi tập đỉnh

Cho đồ thị G=(X, U) và A $\subseteq$ X. Đồ thị con sinh bởi tập đỉnh A, ký hiệu <A> (A, V), trong đó:

- Tập cạnh V $\subseteq$ U
- Gọi u=(i, j) $\in$ U là một cạnh của G, nếu i, j $\in$ A thì u $\in$ V



Hình 1.18 Đồ thị con sinh bởi tập đỉnh

1.1.8 Dây chuyền, chu trình, đường đi và mạch

1.1.8.1 Dây chuyền

Một dây chuyền trong G=(X, U) là một đồ thị con C=(V, E) của G với:

- V = {x_1, x_2, …, x_M}

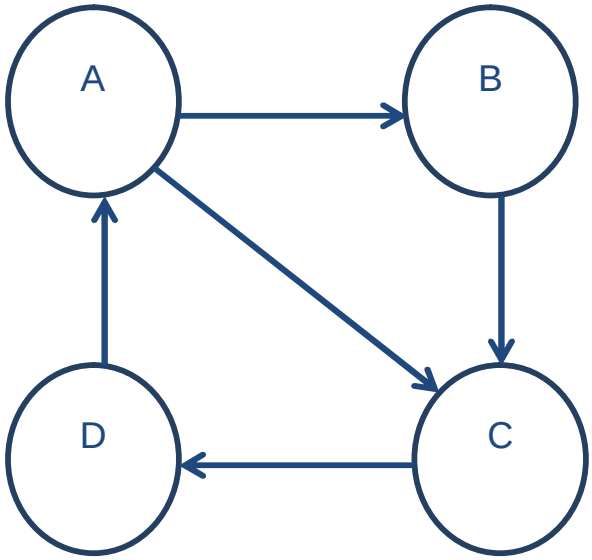
- E = {u_1, u_2, …, u_{M-1}} với $u_1=x_1x_2$, $u_2=x_2x_3$, …, $u_{M-1}=x_M$-1_{xM}; liên kết x_ix_{i+1} không phân biệt thứ tự.

Khi đó, x_1 và x_M được nối với nhau bằng dây chuyền C. x_1 là đỉnh đầu và x_M là đỉnh cuối của C.

Số cạnh của C được gọi là độ dài của C.

Khi các cạnh hoàn toàn xác định bởi cặp đỉnh kề, dây chuyền có thể viết gọn (x_1, x_2, …, x_M).

Dây chuyền sơ cấp là dây chuyền không có đỉnh lặp lại.



Hình 1.19 Dây chuyền, chu trình, đường đi và mạch

Ở đồ thị trên, do các cạnh trong dây chuyền không quan tâm đến thứ tự, hay là chiều, nên sẽ có dây chuyền sau: (A, D, C, A, B) nhưng dây chuyền này không phải là dây chuyền sơ cấp vì trong dây chuyền có đỉnh A lặp lại.

1.1.8.2 Chu trình

Chu trình là một dây chuyền có đỉnh đầu và đỉnh cuối trùng nhau.

Trong hình 1.19, dây chuyền (A, D, C, A) là một chu trình vì có đỉnh đầu và đỉnh cuối trùng nhau.

1.1.8.3 Đường đi

Một đường đi trong G=(X, U) là một đồ thị con P=(V, E) của G với:

- $V = \{x_1, x_2, \ldots, x_M\}$
- $E = \{u_1, u_2, \ldots, u_{M-1}\}$ với $u_1=x_1x_2$, $u_2=x_2x_3$, …, $u_{M-1}=x_{M-1}x_M$; liên kết x_ix_{i+1} theo đúng thứ tự.

Khi đó, có đường đi P nối từ x_1 đến x_M. x_1 là đỉnh đầu và x_M là đỉnh cuối của P.

Số cạnh của P được gọi là độ dài của P.

Khi các cạnh hoàn toàn xác định bởi cặp đỉnh kề, đường đi có thể viết gọn (x_1, x_2, …, x_M).

Đường đi sơ cấp: đường đi không có đỉnh lặp lại.

Trong hình 1.19, (D, A, B, C) là một đường đi và là đường đi sơ cấp vì trong đó không có đỉnh lặp lại.

1.1.8.4 Mạch

Mạch là một đường đi có đỉnh đầu trùng với đỉnh cuối.

Trong hình 1.19, đường đi (A, B, C, D, A) là 1 mạch.

Với đồ thị vô hướng:

- Dây chuyền ≡ đường đi, chu trình ≡ mạch.
- Do đó, thuật ngữ đường đi cũng được dùng cho đồ thị vô hướng.

Mạch trong đồ thị có hướng còn được gọi là "chu trình có hướng". Đường đi trong đồ thị có hướng cũng được gọi là "đường đi có hướng" để nhấn mạnh.

BIỂU DIỄN ĐỒ THỊ

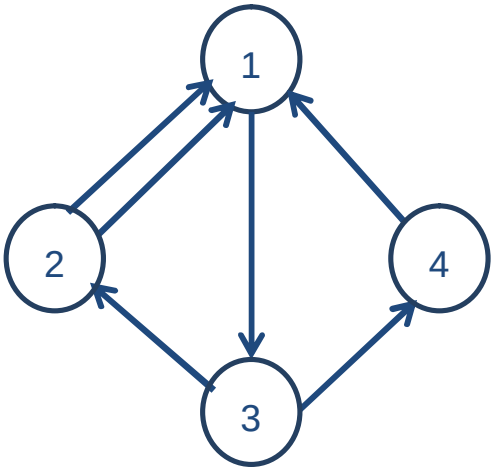
Để biểu diễn một đồ thị, có hướng và vô hướng, người ta có thể dùng dạng ma trận. Có hai dạng ma trận thường gặp để biểu diễn một đồ thị là: ma trận kề và ma trận liên thuộc. Ở những chương sau, khi tính toán với những đồ thị có trọng lượng thì chúng ta sẽ gặp thêm một dạng biểu diễn nữa là ma trận trọng lượng.

1.1.9 Ma trận kề

Xét đồ thị G=(X, U), giả sử tập X gồm N đỉnh và được sắp thứ tự $X=\{x_1, x_2, \ldots, x_N\}$, tập U gồm M cạnh và được sắp thứ tự $U=\{u_1, u_2, \ldots, u_M\}$.

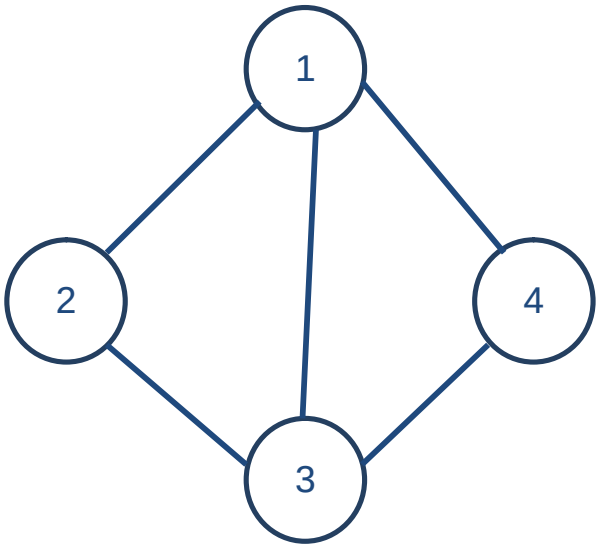
Ma trận kề của đồ thị G, ký hiệu B(G), là một ma trận nhị phân cấp NxN $B=(B_{ij})$ với B_{ij} được định nghĩa:

- Bij=1 nếu có cạnh nối x_i tới x_j.
- B_{ij}=0 trong trường hợp ngược lại.



$$B = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$

Hình 1.20 Đồ thị có hướng G và ma trận kề tương ứng



$$B=\begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{pmatrix}$$

Hình 1.21 Đồ thị vô hướng G và ma trận kề tương ứng

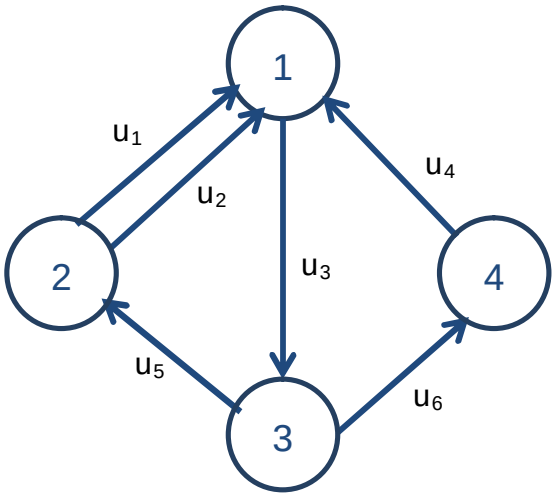
1.1.10 Ma trận liên thuộc

1.1.10.1 Ma trận liên thuộc của đồ thị có hướng

Xét đồ thị G=(X, U) có hướng, giả sử tập X gồm N đỉnh và được sắp thứ tự X={x_1, x_2, …, x_N}, tập U gồm M cạnh và được sắp thứ tự U={u_1, u_2, …, u_M}.

Ma trận liên thuộc (hay liên kết đỉnh cạnh) của G, ký hiệu A(G), là ma trận nhị phân cấp NxM A=(A_{ij}) với A_{ij} được định nghĩa:

- A_{ij}=1 nếu cạnh u_j đi ra khỏi đỉnh x_i.
- A_{ij}=-1 nếu cạnh u_j đi vào đỉnh x_i.
- A_{ij}=0 trong các trường hợp khác.



$$A=\begin{pmatrix} -1 & -1 & 1 & -1 & 0 & 0 \\ 1 & 1 & 0 & 0 & -1 & 0 \\ 0 & 0 & -1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & -1 \end{pmatrix}$$

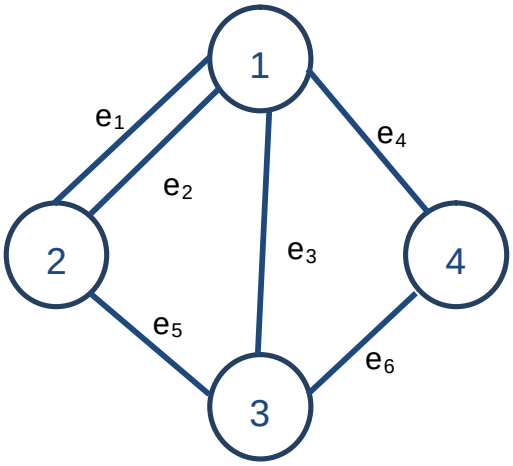
Hình 1.22 Đồ thị có hướng và ma trận liên thuộc tương ứng

1.1.10.2 Ma trận liên thuộc của đồ thị vô hướng

Xét đồ thị G=(X, U) vô hướng, giả sử tập X gồm N đỉnh và được sắp thứ tự X={x_1, x_2, …, x_N}, tập U gồm M cạnh và được sắp thứ tự U={u_1, u_2, …, u_M}.

Ma trận liên thuộc (hay liên kết đỉnh cạnh) của G, ký hiệu A(G), là ma trận nhị phân cấp NxM A=(A_{ij}) với A_{ij} được định nghĩa:

- A_{ij}=1 nếu đỉnh x_i kề với cạnh u_j.
- A_{ij}=0 nếu ngược lại.



$$A=\begin{pmatrix}1&1&1&1&0&0\\1&1&0&0&1&0\\0&0&1&0&1&1\\0&0&0&1&0&1\end{pmatrix}$$

Hình 1.23 Đồ thị vô hướng và ma trận liên thuộc tương ứng

THÀNH PHẦN LIÊN THÔNG

1.1.11 Định nghĩa quan hệ liên kết ~

Cho đồ thị G=(X, U). Ta định nghĩa một quan hệ liên kết ~ như sau trên tập đỉnh X:

$\forall i, j \in X, i \sim j \Leftrightarrow$ (i≡j hoặc có dây chuyền nối i với j).

Quan hệ này có ba tính chất: phản xạ, đối xứng và bắc cầu nên nó là một quan hệ tương đương. Do đó tập X được phân hoạch thành các lớp tương đương.

1.1.12 Định nghĩa thành phần liên thông

Một thành phần liên thông của đồ thị là một lớp tương đương được xác định bởi quan hệ liên kết ~.

Số thành phần liên thông của đồ thị là số lượng các lớp tương đương.

Đồ thị liên thông là đồ thị chỉ có một thành phần liên thông.

Khi một đồ G gồm p thành phần liên thông G_1, G_2, …, G_p thì các đồ thị G_i cũng là các đồ thị con của G và $dG(x) = dG_i(x)$, $\forall x$ của G_i.

Hình 1.24 Các thành phần liên thông của đồ thị

Ở hình 1.24, đồ thị G có 2 thành phần liên thông: thành phần liên thông thứ nhất gồm các đỉnh được tô màu, thành phần liên thông thứ 2 gồm các đỉnh không được tô màu. Đồ thị H có 1 thành phần liên thông nên đồ thị H là đồ thị liên thông.

1.1.13 Thuật toán xác định các thành phần liên thông trong đồ thị

Input: đồ thị G=(X, E), tập X gồm N đỉnh 1, 2, …, N

Output: các đỉnh của G được gán nhãn là số hiệu của thành phần liên thông tương ứng

1. Khởi tạo biến *label=0* và gắn nhãn 0 cho tất cả các đỉnh
2. Duyệt qua tất cả các đỉnh $i \in X$

Nếu nhãn của *i* là 0

1. *label* = *label* + 1.
2. Gán nhãn cho tất cả các đỉnh cùng thuộc thành phần liên thông với *i* là *label.*

Để gán nhãn các đỉnh cùng thuộc một thành phần liên thông với đỉnh i ta dùng thuật toán sau – Visit(i, label)

Input: đồ thị G=(X, E), đỉnh *i*, nhãn *label*

Output: các đỉnh cùng thuộc thành phần liên thông với i được gắn nhãn *label*

1. Gắn nhãn label cho đỉnh *i*

2. Duyệt qua tất cả các đỉnh $j \in X$ và có cạnh nối với i

   Nếu nhãn của j là 0 thì sẽ gọi lại hàm Visit(j, *label*).

TÓM TẮT

- *Đồ thị là một cấu trúc rời rạc gồm các đỉnh và các cạnh (vô hướng hoặc có hướng) nối các đỉnh đó. Nếu các cạnh có quy định hướng thì đồ thị là đồ thị có hướng còn nếu không quy định về hướng thì đồ thị là đồ thị vô hướng.*
- *Đồ thị rỗng là đồ thị có tập cạnh rỗng. Đồ thị đơn là đồ thị không có khuyên và cạnh song song.*
- *Bậc của đỉnh là số cạnh kề với đỉnh đó. Tổng bậc của tất cả các đỉnh trong đồ thị bằng hai lần số cạnh có trong đồ thị.*
- *Dây chuyền, chu trình, đường đi và mạch. Trong đồ thị vô hướng, dây chuyền và chu trình tương ứng trùng với đường đi và mạch.*
- *Một đồ thị có thể biểu diễn dưới dạng ma trận: ma trận kề biểu diễn mối quan hệ kề giữa đỉnh và đỉnh, ma trận liên thuộc biểu diễn mối quan hệ kề giữa đỉnh và cạnh.*
- *Một thành phần liên thông của đồ thị là một lớp tương đương được xác định bởi quan hệ liên kết ~. Số thành phần liên thông của đồ thị là số lượng các lớp tương đương. Đồ thị liên thông là đồ thị chỉ có một thành phần liên thông.*

BÀI TẬP

1. G là một đồ thị đơn, vô hướng có số đỉnh N>3. Chứng minh G có chứa 2 đỉnh cùng bậc.
2. Đồ thị G có đúng 2 đỉnh bậc lẻ. Chứng minh tồn tại một dây chuyền nối hai đỉnh đó với nhau.
3. Xét đồ thị G đơn, vô hướng gồm N đỉnh, M cạnh và P thành phần liên thông.

   a. Chứng minh: $M \leq (N-P)(N-P+1)/2$,

   suy ra nếu $M > (N-1)(N-2)/2$ thì G liên thông.

   b. Một đồ thị đơn có 10 đỉnh, 37 cạnh thì có chắc liên thông hay không?
4. Đồ thị G đơn, vô hướng gồm N đỉnh và $d(x) \geq (N-1)/2$ với mọi đỉnh x. Chứng minh G liên thông.
5. Đồ thị vô hướng G liên thông gồm N đỉnh. Chứng minh số cạnh của $G \geq N-1$.
6. Xét đồ thị G vô hướng đơn. Gọi x là đỉnh có bậc nhỏ nhất của G. Giả sử $d(x) \geq k \geq 2$ với k nguyên dương. Chứng minh G chứa một chu trình sơ cấp có chiều dài lớn hơn hay bằng k+1.
7. Cho G là đồ thị vô hướng liên thông. Giả sử C_1 và C_2 là 2 dây chuyền sơ cấp trong G có số cạnh nhiều nhất. Chứng minh C_1 và C_2 có đỉnh chung.
8. G là đồ thị vô hướng không khuyên và $d(x) \geq 3$ với mọi đỉnh x. Chứng minh G có chứa chu trình với số cạnh chẵn.

BÀI 2: ĐỒ THỊ EULER VÀ ĐỒ THỊ HAMILTON

Sau khi học xong bài này, học viên có thể:

- *Biết được nguồn gốc bài toán cổ điển Thành phố Konigsberg và bảy chiếc cầu.*
- *Hiểu được định nghĩa về dây chuyền, chu trình, đường đi và mạch Euler.*
- *Hiểu được thế nào được gọi là một đồ thị Euler và nửa Euler, cách xác định một đồ thị có phải là một đồ thị Euler hay không.*
- *Biết được thuật toán xác định dây chuyền Euler và chu trình Euler.*
- *Giải được bài toán Người đưa thư Trung Hoa.*
- *Hiểu được các kiến thức tương tự như trên với đồ thị Hamilton.*

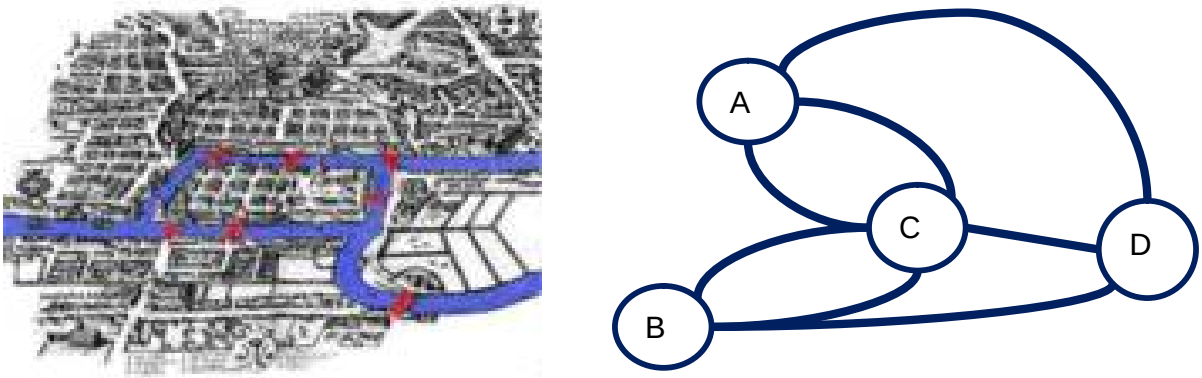
ĐỒ THỊ EULER

Bài toán 7 chiếc cầu: Thành phố Konigsberg thuộc Phổ (nay gọi là Kaliningrad thuộc Nga) được chia thành bốn vùng bằng các nhánh sông Pregel, các vùng này gồm hai vùng bên bờ sông, đảo Kneiphof và một miền nằm giữa hai nhánh của sông Pregel. Vào thế kỷ 18, người ta xây bảy chiếc cầu nối các vùng này với nhau.

Dân thành phố từng thắc mắc: "Có thể nào đi dạo qua tất cả bảy cầu, mỗi cầu chỉ một lần thôi không?". Nếu ta coi mỗi khu vực A, B, C, D như một đỉnh và mỗi cầu qua lại hai khu vực là một cạnh nối hai đỉnh thì ta có sơ đồ của Konigsberg là một đa đồ thị G như hình trên.

Bài toán tìm đường đi qua tất cả các cầu, mỗi cầu chỉ qua một lần có thể được phát biểu lại bằng mô hình này như sau: Có tồn tại chu trình đơn trong đồ thị G chứa tất cả các cạnh?

Có thể coi năm 1736 là năm khai sinh lý thuyết đồ thị, với việc công bố lời giải "Bài toán về các cầu ở Konigsberg" của nhà toán học lỗi lạc Euler (1707-1783).

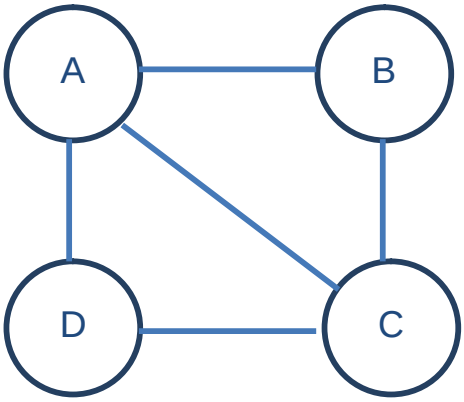


Hình 2.1 Bài toán cây cầu Konigsberg

2.1.1 Một số định nghĩa

2.1.1.1 Dây chuyền Euler

Dây chuyền Euler là dây chuyền đi qua tất cả các cạnh trong đồ thị, mỗi cạnh đi qua đúng một lần.

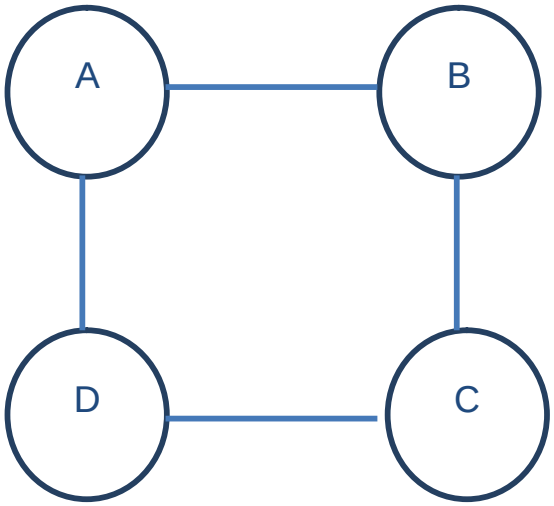


Hình 2.2 Dây chuyền Euler

Trong hình 2.2, dây chuyền (A, B, C, D, A, C) là 1 dây chuyền Euler vì dây chuyền này đi qua tất cả các cạnh của đồ thị và mỗi cạnh chỉ đi qua 1 lần. Dây chuyền Euler trong đồ thị trên cũng có thể bắt đầu từ đỉnh C và kết thúc tại đỉnh A.

2.1.1.2 Chu trình Euler

Chu trình Euler là dây chuyền Euler có đỉnh đầu và đỉnh cuối trùng nhau.

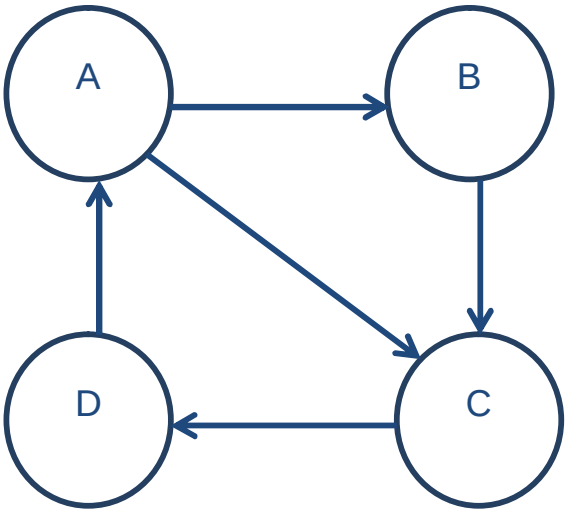


Hình 2.3 Chu trình Euler

Ở hình 2.3, chu trình (A, B, C, D, A) là một chu trình Euler. Trong trường hợp này chu trình Euler có thể bắt đầu từ bất cứ đỉnh nào của đồ thị và kết thúc ngay chính tại đỉnh bắt đầu sau khi đã đi qua hết tất cả các cạnh đúng một lần. Trong trường hợp này có thể đi theo chiều kim đồng hồ hoặc ngược lại đều được.

2.1.1.3 Đường đi Euler

Đường đi Euler là đường đi qua tất cả các cạnh của đồ thị, mỗi cạnh đúng một lần.

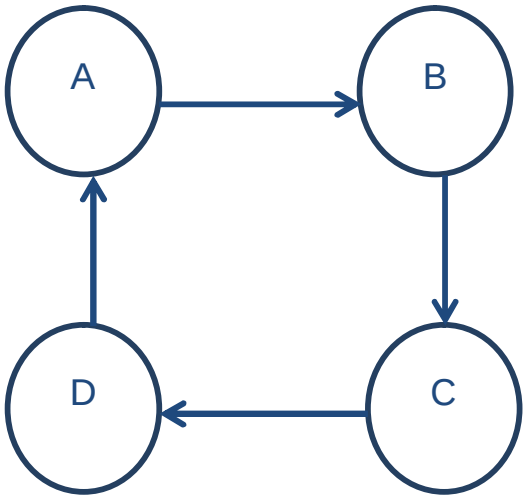


Hình 2.4 Đường đi Euler

Ở hình 2.4, có một đường đi Euler là (A, B, C, D, A, C). Khác với đồ thị ở hình 2.2 thì ở đồ thị này không có đường đi theo chiều ngược lại, tức là xuất phát từ C và kết thúc ở A.

2.1.1.4 Mạch Euler

Mạch Euler là đường đi Euler có đỉnh đầu và đỉnh cuối trùng nhau.



Hình 2.5 Mạch Euler

Nếu bắt đầu từ một đỉnh bất kỳ đi theo chiều kim đồng hồ, sau khi đi qua một lượt tất cả các cạnh của đồ thị ở hình 2.5 thì ta sẽ có được một mạch Euler. Khác với hình 2.3, ở trường hợp này không thể đi theo hướng ngược chiều kim đồng hồ.

2.1.1.5 Đồ thị Euler vô hướng

Đồ thị Euler vô hướng là đồ thị vô hướng có chứa một chu trình Euler.

Hình 2.3 là một đồ thị Euler vô hướng.

Đồ thị vô hướng có chứa một dây chuyền Euler thì được gọi là đồ thị nửa Euler vô hướng.

Hình 2.2 là một đồ thị nửa Euler vô hướng.

2.1.1.6 Đồ thị Euler có hướng

Đồ thị Euler có hướng là đồ thị có hướng có chứa một mạch Euler.

Đồ thị ở hình 2.5 là một đồ thị Euler có hướng.

Đồ thị có hướng chứa một đường đi Euler thì được gọi là đồ thị nửa Euler có hướng.

Hình 2.4 là một đồ thị nửa Euler có hướng.

2.1.2 Định lý Euler

2.1.2.1 Đồ thị có hướng

Đồ thị có hướng G là đồ thị Euler khi và chỉ khi G liên thông và $d^+(x)=d^-(x)$ với mọi $x \in X$.

Hệ quả: G là đồ thị nửa Euler khi và chỉ khi trong G tồn tại 2 đỉnh x, y thoả điều kiện $d^+(x)=d^-(x)-1$ và $d^+(y)=d^-(y)+1$, $d^+(t)=d^-(t)$ với t là tất cả các đỉnh còn lại trong đồ thị.

2.1.2.2 Đồ thị vô hướng

Đồ thị vô hướng G là đồ thị Euler khi và chỉ khi G liên thông và d(x) chẵn với mọi $x \in X$.

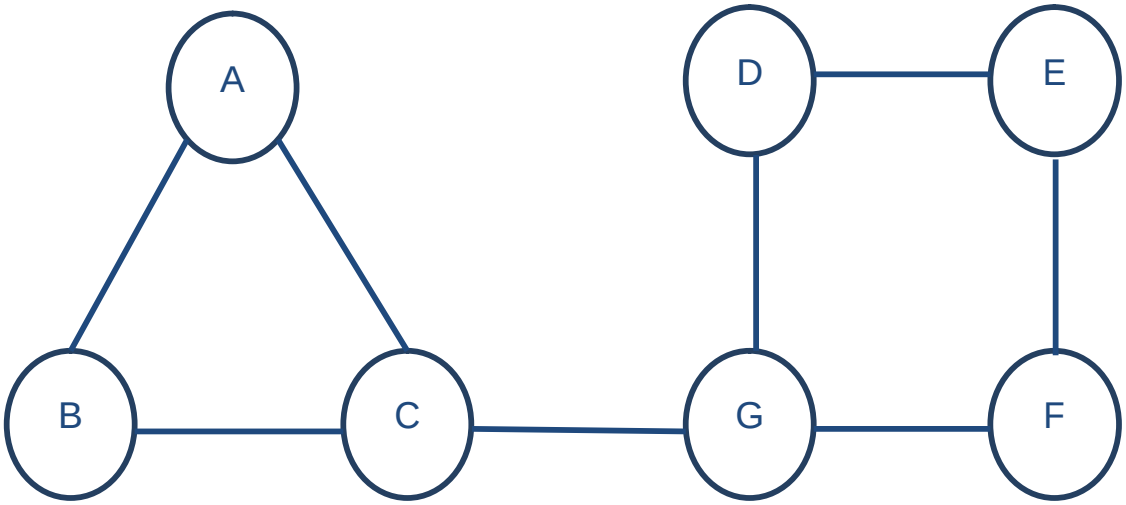
Hệ quả: G là đồ thị nửa Euler khi và chỉ khi trong G có đúng 2 đỉnh bậc lẻ.

2.1.3 Thuật toán Fleury

Thuật toán Fleury dùng để xác định chu trình Euler trong đồ thị G liên thông và có tất cả các đỉnh đều là đỉnh bậc chẵn.

2.1.3.1 Định nghĩa về cầu

Cạnh e của đồ thị G được gọi là cầu nếu xóa e khỏi đồ thị thì làm tăng số thành phần liên thông của G.



Hình 2.6 Cầu trong đồ thị

Trong hình 2.6, cạnh (C, G) được gọi là cầu vì khi xoá cạnh này làm tăng số thành phần liên thông của đồ thị.

2.1.3.2 Thuật toán Fleury.

Gọi chu trình Euler cần tìm là C. Thuật toán sẽ tiến hành theo các bước sau:

Khởi tạo: Chọn một đỉnh bất kỳ cho vào *C*.

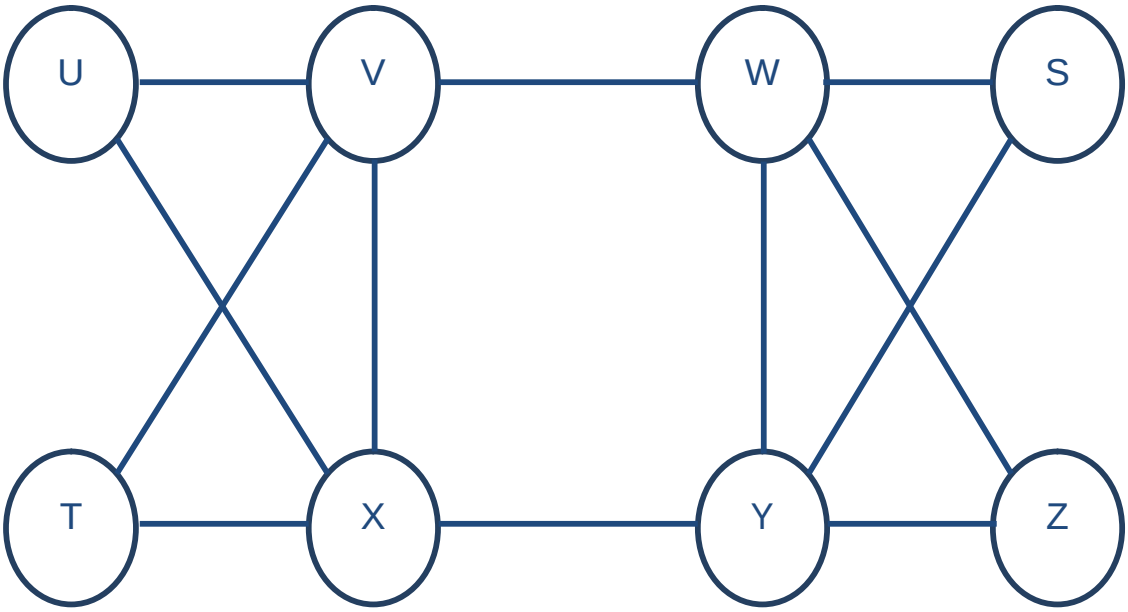
Lặp trong khi G vẫn còn cạnh

Chọn cạnh e nối đỉnh vừa chọn với một đỉnh kề với nó theo nguyên tắc: chỉ chọn cầu nếu không còn cạnh nào khác để chọn.

Bổ sung e và đỉnh cuối của nó vào C.

Xóa e khỏi G. Sau đó xoá đỉnh cô lập (nếu có).

2.1.3.3 Ví dụ



Hình 2.7 Chu trình Euler

Xuất phát từ u, ta có thể đi theo cạnh (u, v) hoặc (u, x), giả sử là (u, v) (xoá (u, v)). Từ v có thể đi qua một trong các cạnh (v, w), (v, x), (v, t), giả sử (v, w) (xoá (v, w)). Tiếp tục, có thể đi theo một trong các cạnh (w, s), (w, y), (w, z), giả sử (w, s) (xoá (w, s)).

Đi theo cạnh (s, y) (xoá (s, y) và s). Vì (y, x) là cầu nên có thể đi theo một trong hai cạnh (y, w), (y, z), giả sử (y, w) (xoá (y, w)). Đi theo (w, z) (xoá (w, z) và w) và theo (z, y) (xoá (z, y) và z). Tiếp tục đi theo cạnh (y, x) (xoá (y, x) và y). Vì (x, u) là cầu nên đi theo cạnh (x, v) hoặc (x, t), giả sử (x, v) (xoá (x, v)).

Tiếp tục đi theo cạnh (v, t) (xoá (v, t) và v), theo cạnh (t, x) (xoá cạnh (t, x) và t), cuối cung đi theo cạnh (x, u) (xoá (x, u), x và u).

Sau khi thực hiện xong thuật toán ta có chu trình Euler như sau: (u, v, w, s, y, w, z, y, x, v, t, x, u)

2.1.4 Bài toán người phát thư Trung Hoa

2.1.4.1 Phát biểu bài toán

Một nhân viên đi từ Sở Bưu Điện, qua một số đường phố để phát thư, rồi quay về Sở. Người ấy phải đi qua các đường theo trình tự nào để đường đi là ngắn nhất?

Bài toán được nhà toán học Trung Hoa Guan nêu lên đầu tiên (1960), vì vậy thường được gọi là "bài toán người phát thư Trung Hoa".

2.1.4.2 Giải bài toán

Ta xét bài toán ở một dạng đơn giản như sau.

Cho đồ thị liên thông G. Một chu trình qua mọi cạnh của G gọi là một hành trình trong G. Trong các hành trình đó, hãy tìm hành trình ngắn nhất, tức là qua ít cạnh nhất.

Rõ ràng rằng nếu G là đồ thị Euler (mọi đỉnh đều có bậc chẵn) thì chu trình Euler trong G (qua mỗi cạnh của G đúng một lần) là hành trình ngắn nhất cần tìm.

Chỉ còn phải xét trường hợp G có một số đỉnh bậc lẻ (số đỉnh bậc lẻ là một số chẵn). Khi đó, mọi hành trình trong G phải đi qua ít nhất hai lần một số cạnh nào đó.

Dễ thấy rằng một hành trình qua một cạnh (u, v) nào đó quá hai lần thì không phải là hành trình ngắn nhất trong G. Vì vậy, ta chỉ cần xét những hành trình T đi qua hai lần một số cạnh nào đó của G.

Ta quy ước xem mỗi hành trình T trong G là một hành trình trong đồ thị Euler G_T, có được từ G bằng cách vẽ thêm một cạnh song song đối với những cạnh mà T đi qua hai lần. Bài toán đặt ra được đưa về bài toán sau:

Trong các đồ thị Euler G_T, tìm đồ thị có số cạnh ít nhất (khi đó chu trình Euler trong đồ thị này là hành trình ngắn nhất).

Định lý (Gooodman và Hedetniemi, 1973). Nếu G là một đồ thị liên thông có q cạnh thì hành trình ngắn nhất trong G có chiều dài

$q + m(G)$,

trong đó m(G) là số cạnh mà hành trình đi qua hai lần và được xác định như sau:

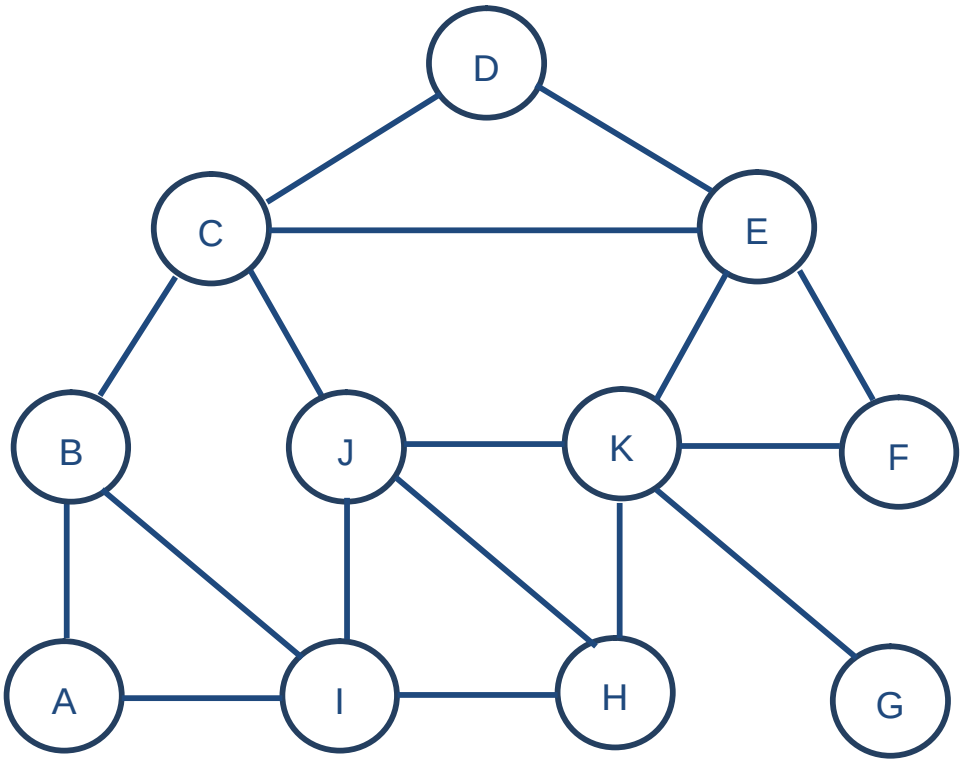
Gọi $V_0(G)$ là tập hợp các đỉnh bậc lẻ (2k đỉnh) của G. Ta phân 2k phần tử của G thành k cặp, mỗi tập hợp k cặp gọi là một phân hoạch cặp của $V_0(G)$.

Ta gọi độ dài đường đi ngắn nhất từ u đến v là khoảng cách d(u, v). Đối với mọi phân hoạch cặp P_i, ta tính khoảng cách giữa hai đỉnh trong từng cặp, rồi tính tổng $d(P_i)$. Số m(G) bằng cực tiểu của các $d(P_i)$:

$m(G) = \min d(P_i)$.

2.1.4.3 Ví dụ

Giải bài toán người phát thư Trung Hoa với đồ thị sau:



Hình 2.8 Bài toán người phát thư Trung Hoa

Tập hợp các đỉnh bậc lẻ $V_O(G) = \{B, G, H, K\}$ và tập hợp các phân hoạch cặp là $P = \{P_1, P_2, P_3\}$, trong đó

$P_1 = \{(B, G), (H, K)\} \rightarrow d(P_1) = d(B, G) + d(H, K) = 4 + 1 = 5$,

$P_2 = \{(B, H), (G, K)\} \rightarrow d(P_2) = d(B, H) + d(G, K) = 2 + 1 = 3$,

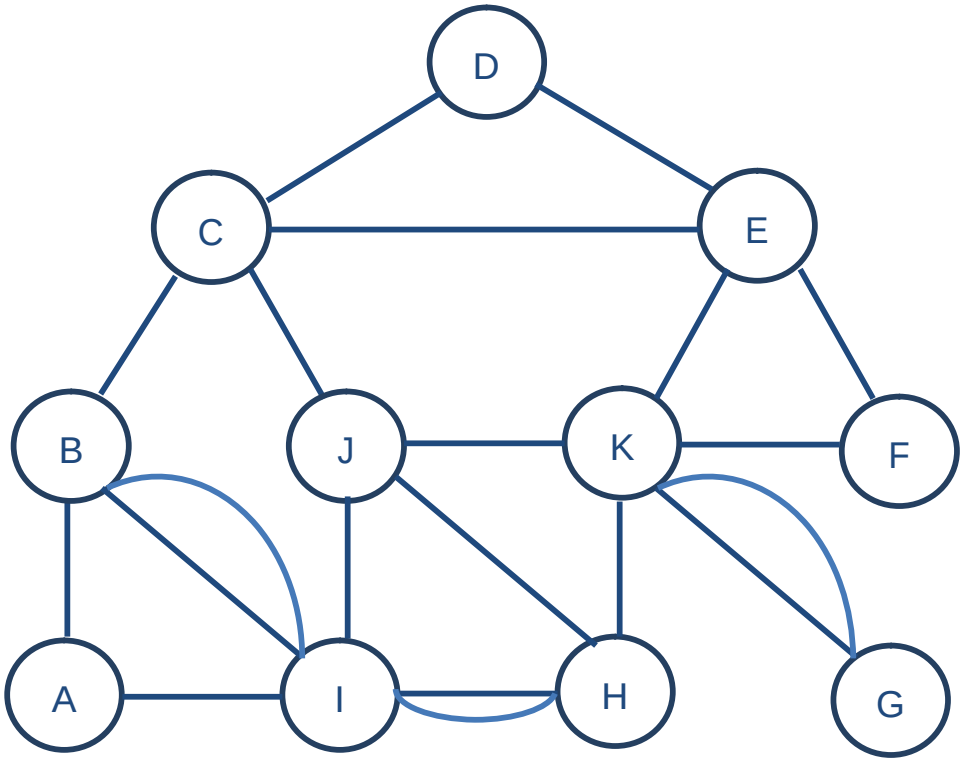
$P_3 = \{(B, K), (G, H)\} \rightarrow d(P_3) = d(B, K) + d(G, H) = 3 + 2 = 5$.

Vậy $m(G) = \min(d(P_1), d(P_2), d(P_3)) = 3$.

Do đó G_T có được từ G bằng cách thêm vào 3 cạnh: (B, I), (I, H), (G, K) và G_T là đồ thị Euler. Vậy hành trình ngắn nhất cần tìm là đi theo chu trình Euler trong G_T:

(A, B, C, D, E, F, K, G, K, E, C, J, K, H, J, I, H, I, B, I, A)

Đồ thị sau khi thêm cạnh:



Hình 2.9 Bài toán người phát thư Trung Hoa

ĐỒ THỊ HAMILTON

Năm 1857, nhà toán học người Ailen là Hamilton(1805-1865) đưa ra trò chơi "đi vòng quanh thế giới" như sau.

Cho một hình thập nhị diện đều (đa diện đều có 12 mặt, 20 đỉnh và 30 cạnh), mỗi đỉnh của hình mang tên một thành phố nổi tiếng, mỗi cạnh của hình (nối hai đỉnh) là đường đi lại giữa hai thành phố tương ứng. Xuất phát từ một thành phố, hãy tìm đường đi thăm tất cả các thành phố khác, mỗi thành phố chỉ một lần, rồi trở về chỗ cũ.

Trước Hamilton, có thể là từ thời Euler, người ta đã biết đến một câu đố hóc búa về "đường đi của con mã trên bàn cờ". Trên bàn cờ, con mã chỉ có thể đi theo đường chéo của hình chữ nhật 2 x 3 hoặc 3 x 2 ô vuông. Giả sử bàn cờ có 8 x 8 ô vuông.

Hãy tìm đường đi của con mã qua được tất cả các ô của bàn cờ, mỗi ô chỉ một lần rồi trở lại ô xuất phát.

Bài toán này được nhiều nhà toán học chú ý, đặc biệt là Euler, De Moivre, Vandermonde, ...

Hiện nay đã có nhiều lời giải và phương pháp giải cũng có rất nhiều, trong đó có quy tắc: mỗi lần bố trí con mã ta chọn vị trí mà tại vị trí này số ô chưa dùng tới do nó khống chế là ít nhất.

Một phương pháp khác dựa trên tính đối xứng của hai nửa bàn cờ. Ta tìm hành trình của con mã trên một nửa bàn cờ, rồi lấy đối xứng cho nửa bàn cờ còn lại, sau đó nối hành trình của hai nửa đã tìm lại với nhau.

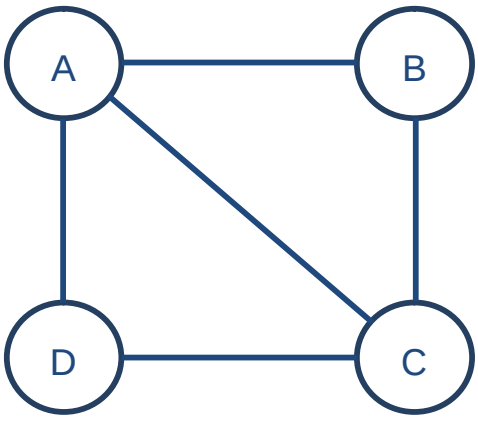
Trò chơi và câu đố trên dẫn tới việc khảo sát một lớp đồ thị đặc biệt, đó là đồ thị Hamilton.

2.1.5 Định nghĩa

Cho một đồ thị vô hướng G(X, E).

2.1.5.1 Dây chuyền Hamilton

Dây chuyền Hamilton là dây chuyền đi qua tất cả các đỉnh của đồ thị mỗi đỉnh đúng một lần.



Hình 2.10 Dây chuyền Hamilton

Trong hình 2.10, dây chuyền (A, B, C, D) là một dây chuyền Hamilton vì nó đi qua 4 đỉnh của đồ thì và mỗi đỉnh chỉ đi qua 1 lần.

2.1.5.2 Chu trình Hamilton

Chu trình Hamilton là một dây chuyền Hamilton và một cạnh trong đồ thị nối đỉnh đầu của dây chuyền với đỉnh cuối của nó.

Theo hình số 2.10 thì dây chuyền (A, B, C, D) và cạnh AD tạo nên một chu trình Hamilton là (A, B, C, D, A).

2.1.5.3 Đồ thị Hamilton

Đồ thị Hamilton là đồ thị có chứa một chu trình Hamilton.

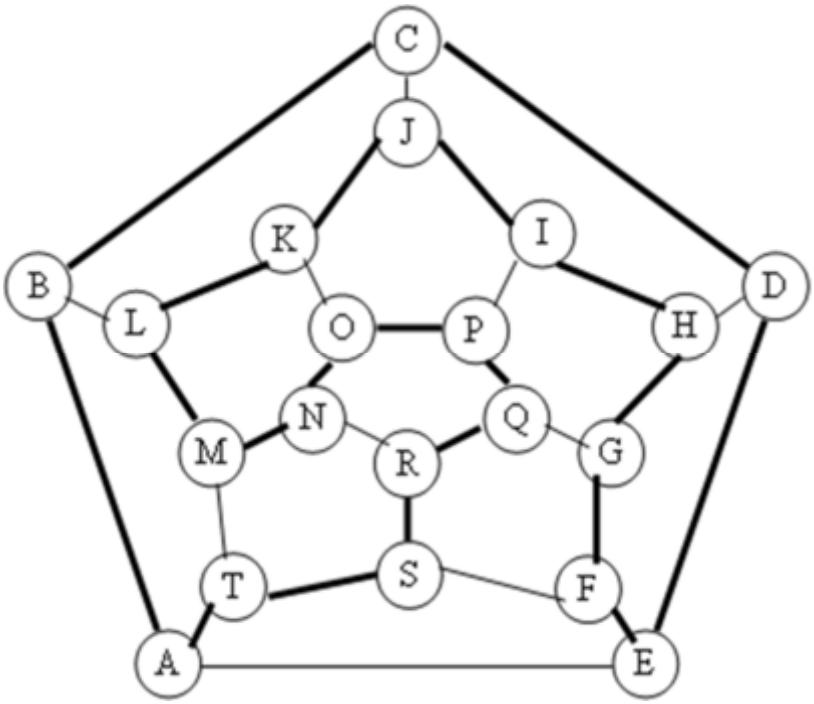
Đồ thị ở hình 2.10 có chứa một chu trình Hamilton nên nó là một đồ thị Hamilton.

Đồ thị nửa Hamilton là đồ thị có chứa một dây chuyền Hamilton.

2.1.6 Các định lý

2.1.6.1 Một vài bài toán điển hình

a) Bài toán thập nhị diện đều



Hình 2.11 Bài toán thập nhị diện đều

Ở hình 2.11, đồ thị Hamilton (hình thập nhị diện đều biểu diễn trong mặt phẳng) với chu trình Hamilton A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, A (đường tô đậm).

b) Bài toán giải bóng bàn

Trong một đợt thi đấu bóng bàn có n ($n \geq 2$) đấu thủ tham gia. Mỗi đấu thủ gặp từng đấu thủ khác đúng một lần. Trong thi đấu bóng bàn chỉ có khả năng thắng hoặc thua. Chứng minh rằng sau đợt thi đấu có thể xếp tất cả các đấu thủ đứng thành một hàng dọc, để người đứng sau thắng người đứng ngay trước anh (chị) ta.

Xét đồ thị có hướng G gồm n đỉnh sao cho mỗi đỉnh ứng với một đấu thủ và có một cung nối từ đỉnh u đến đỉnh v nếu đấu thủ ứng với u thắng đấu thủ ứng với v. Như vậy, đồ thị G có tính chất là với hai đỉnh phân biệt bất kỳ u và v, có một và chỉ một trong hai cung (u, v) hoặc (v, u), đồ thị như thế được gọi là đồ thị có hướng đầy đủ. Từ định lý Rédei sẽ trình bày sau đây, G là một đồ thị nửa Hamilton. Khi đó đường đi Hamilton trong G cho ta sự sắp xếp cần tìm.

c) Nhận xét

Đường đi Hamilton tương tự đường đi Euler trong cách phát biểu: Đường đi Euler qua mọi cạnh (cung) của đồ thị đúng một lần, đường đi Hamilton qua mọi đỉnh của đồ thị đúng một lần. Tuy nhiên, nếu như bài toán tìm đường đi Euler trong một đồ thị đã được giải quyết trọn vẹn, dấu hiệu nhận biết một đồ thị Euler là khá đơn giản và dễ sử dụng, thì các bài toán về tìm đường đi Hamilton và xác định đồ thị Hamilton lại khó hơn rất nhiều. Đường đi Hamilton và đồ thị Hamilton có nhiều ý nghĩa thực tiễn và đã được nghiên cứu nhiều, nhưng vẫn còn những khó khăn lớn chưa ai vượt qua được.

Người ta chỉ mới tìm được một vài điều kiện đủ để nhận biết một lớp rất nhỏ các đồ thị Hamilton và đồ thị nửa Hamilton. Sau đây là một vài kết quả.

2.1.6.2 Định lý Rédei

Nếu G là một đồ thị có hướng đầy đủ thì G là đồ thị nửa Hamilton.

Chứng minh: Giả sử G=(V,E) là đồ thị có hướng đầy đủ và $\alpha=(v_1,v_2, ..., v_{k-1}, v_k)$ là đường đi sơ cấp bất kỳ trong đồ thị G.

- Nếu α đã đi qua tất cả các đỉnh của G thì nó là một đường đi Hamilton của G.
- Nếu trong G còn có đỉnh nằm ngoài α, thì ta có thể bổ sung dần các đỉnh này vào α và cuối cùng nhận được đường đi Hamilton.

Thật vậy, giả sử v là đỉnh tuỳ ý không nằm trên α.

a) Nếu có cung nối v với v_1 thì bổ sung v vào đầu của đường đi α để được $\alpha_1=(v, v_1, v_2, ..., v_{k-1}, v_k)$.

b) Nếu tồn tại chỉ số i ($1 \le i \le k-1$) mà từ v_i có cung nối tới v và từ v có cung nối tới v_{i+1} thì ta chen v vào giữa v_i và v_{i+1} để được đường đi sơ cấp $\alpha_2=(v_1, v_2, ..., v_i, v, v_{i+1}, ..., v_k)$.

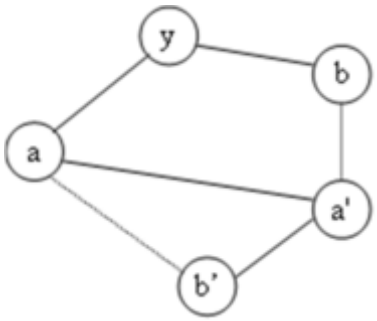
c) Nếu cả hai khả năng trên đều không xảy ra nghĩa là với mọi i ($1 \le i \le k$) v_i đều có cung đi tới v. Khi đó bổ sung v vào cuối của đường đi α và được đường đi $\alpha_3=(v_1, v_2, ..., v_{k-1}, v_k, v)$.

Nếu đồ thị G có n đỉnh thì sau n-k bổ sung ta sẽ nhận được đường đi Hamilton.

2.1.6.3 Định lý Dirac (1952)

Nếu G là một đơn đồ thị có n đỉnh và mọi đỉnh của G đều có bậc không nhỏ hơn $\frac{n}{2}$ thì G là một đồ thị Hamilton.

Chứng minh: Định lý được chứng minh bằng phản chứng. Giả sử G không có chu trình Hamilton. Ta thêm vào G một số đỉnh mới và nối mỗi đỉnh mới này với mọi đỉnh của G, ta được đồ thị G'. Giả sử k (>0) là số tối thiểu các đỉnh cần thiết để G' chứa một chu trình Hamilton. Như vậy, G' có n+k đỉnh.



Hình 2.12 Định lý Dirac

Gọi P là chu trình Hamilton ayb ...a trong G', trong đó a và b là các đỉnh của G, còn y là một trong các đỉnh mới. Khi đó b không kề với a, vì nếu trái lại thì ta có thể bỏ đỉnh y và được chu trình ab ...a, mâu thuẫn với giả thiết về tính chất nhỏ nhất của k.

Ngoài ra, nếu a' là một đỉnh kề nào đó của a (khác với y) và b' là đỉnh nối tiếp ngay a' trong chu trình P thì b' không thể là đỉnh kề với b, vì nếu trái lại thì ta có thể thay P bởi chu trình aa' ...bb' ... a, trong đó không có y, mâu thuẫn với giả thiết về tính chất nhỏ nhất của k.

Như vậy, với mỗi đỉnh kề với a, ta có một đỉnh không kề với b, tức là số đỉnh không kề với b không thể ít hơn số đỉnh kề với a (số đỉnh kề với a không nhỏ hơn $\frac{n}{2}$ +k). Mặt khác, theo giả thiết số đỉnh kề với b cũng không nhỏ hơn $\frac{n}{2}$+k. Vì không có đỉnh nào vừa kề với b lại vừa không kề với b, nên số đỉnh của G' không ít hơn 2($\frac{n}{2}$ +k)=n+2k, mâu thuẫn với giả thiết là số đỉnh của G' bằng n+k (k>0). Định lý được chứng minh.

Hệ quả: Nếu G là đơn đồ thị có n đỉnh và mọi đỉnh của G đều có bậc không nhỏ hơn $\frac{n-1}{2}$ thì G là đồ thị nửa Hamilton.

2.1.6.4 Định lý Ore (1960)

Nếu G là đồ thị có n đỉnh và bất kỳ hai đỉnh nào không kề nhau cũng có tổng số bậc không nhỏ hơn n thì G là một đồ thị Hamilton.

2.1.6.5 Định lý đồ thị lưỡng phân

Nếu G là đồ thị lưỡng phân với hai tập đỉnh là V_1, V_2 có số đỉnh cùng bằng n ($n \geq 2$) và bậc của mỗi đỉnh lớn hơn $\frac{n}{2}$ thì G là một đồ thị Hamilton.

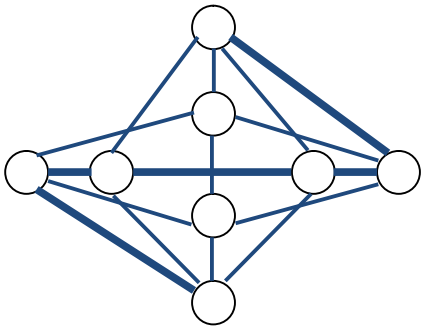
2.1.6.6 Định lý

Đồ thị vô hướng đơn G gồm n đỉnh và m cạnh. Nếu $m \geq (n^2-3n+6)/2$ thì G là đồ thị Hamilton.

2.1.6.7 Định lý

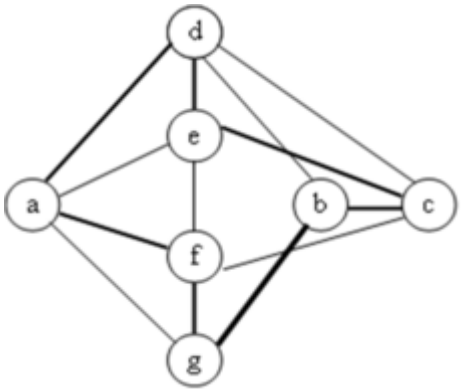
Đồ thị đầy đủ K_n với n lẻ và $n \geq 3$ có đúng $\frac{n-1}{2}$ chu trình Hamilton phân biệt.

2.1.6.8 Ví dụ



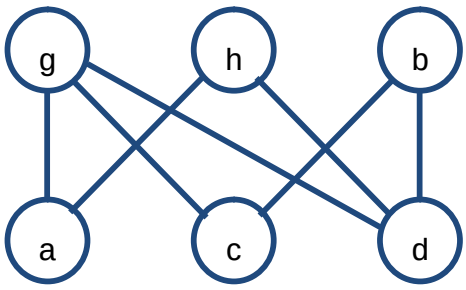
Hình 2.13 Ví dụ 1

Đồ thị hình 2.13 có 8 đỉnh, đỉnh nào cũng có bậc 4 nên theo định lý Dirac thì đồ thị này là đồ thị Hamilton.



Hình 2.14 Ví dụ 2

Đồ thị hình 2.14 có 5 đỉnh bậc 4 và 2 đỉnh bậc 3 kề nhau nên tổng bậc 2 đỉnh bất kì không kề nhau bằng 7 hoặc 8 nên theo định lý Ore thì đồ thị trên là đồ thị Hamilton.



Hình 2.15 Ví dụ 3

Ở hình 2.15 là đồ thị lưỡng phân có các đỉnh có bậc 2 hoặc 3 (lớn hơn 3/2) nên theo định lý đồ thị lưỡng phân thì đồ thị này là đồ thị Hamilton.

Bài toán sắp xếp chỗ ngồi: Có n đại biểu từ n nước đến dự hội nghị quốc tế. Mỗi ngày họp một lần ngồi quanh một bàn tròn. Hỏi phải bố trí bao nhiêu ngày và bố trí

như thế nào sao cho trong mỗi ngày, mỗi người có hai người kế bên là bạn mới. Lưu ý rằng n người đều muốn làm quen với nhau.

Xét đồ thị gồm n đỉnh, mỗi đỉnh ứng với mỗi người dự hội nghị, hai đỉnh kề nhau khi hai đại biểu tương ứng muốn làm quen với nhau. Như vậy, ta có đồ thị đầy đủ K_n. Đồ thị này là Hamilton và rõ ràng mỗi chu trình Hamilton là một cách sắp xếp như yêu cầu của bài toán. Bái toán trở thành tìm các chu trình Hamilton phân biệt của đồ thị đầy đủ K_n (hai chu trình Hamilton gọi là phân biệt nếu chúng không có cạnh chung).

K_n có $\frac{n(n-1)}{2}$ cạnh và mỗi chu trình Hamilton có n cạnh, nên số chu trình Hamilton phân biệt nhiều nhất là $\frac{n-1}{2}$.

Giả sử các đỉnh của K_n là 1, 2, ..., n. Đặt đỉnh 1 tại tâm của một đường tròn và các đỉnh 2, ..., n đặt cách đều nhau trên đường tròn (mỗi cung là $360^0/(n-1)$ sao cho đỉnh lẻ nằm ở nửa đường tròn trên và đỉnh chẵn nằm ở nửa đường tròn dưới. Ta có ngay chu trình Hamilton đầu tiên là 1,2, ..., n,1. Các đỉnh được giữ cố định, xoay khung theo chiều kim đồng hồ với các góc quay:

$$\frac{360^0}{n-1}, 2.\frac{360^0}{n-1}, 3.\frac{360^0}{n-1}, \ldots, \frac{n-3}{2}.\frac{360^0}{n-1},$$

Ta nhận được $\frac{n-3}{2}$ khung phân biệt với khung đầu tiên. Do đó ta có $\frac{n-1}{2}$ chu trình Hamilton phân biệt.

Thí dụ 5: Giải bài toán sắp xếp chỗ ngồi với n=11.

Có (11–1)/2=5 cách sắp xếp chỗ ngồi phân biệt như sau:

1 2 3 4 5 6 7 8 9 10 11 1

1 3 5 2 7 4 9 6 11 8 10 1

1 5 7 3 9 2 11 4 10 6 8 1

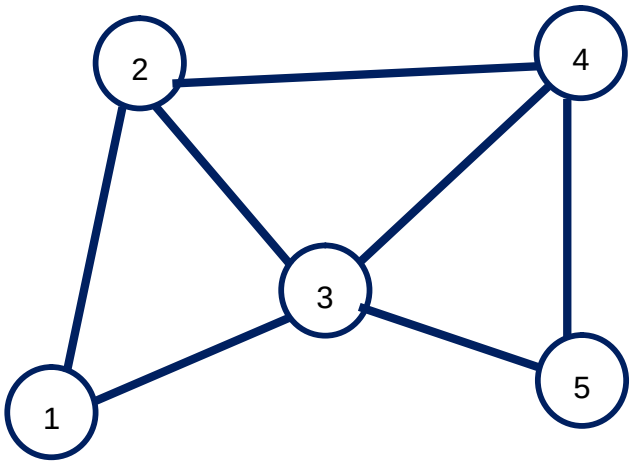
1 7 9 5 11 3 10 2 8 4 6 1

1 9 11 7 10 5 8 3 6 2 4 1

2.1.7 Quy tắc xác định chu trình Hamilton

1. Nếu G có đỉnh bậc < 2 thì G không có chu trình Hamilton
2. Nếu đỉnh có bậc 2 thì 2 cạnh kề với nó phải nằm trong chu trình Hamilton
3. Các cạnh thừa (ngoài 2 cạnh đã chọn trong chu trình Hamilton) phải được bỏ đi trong quá trình xác định chu trình
4. Nếu quá trình xây dựng tạo nên một chu trình con thì đồ thị không có chu trình Hamilton

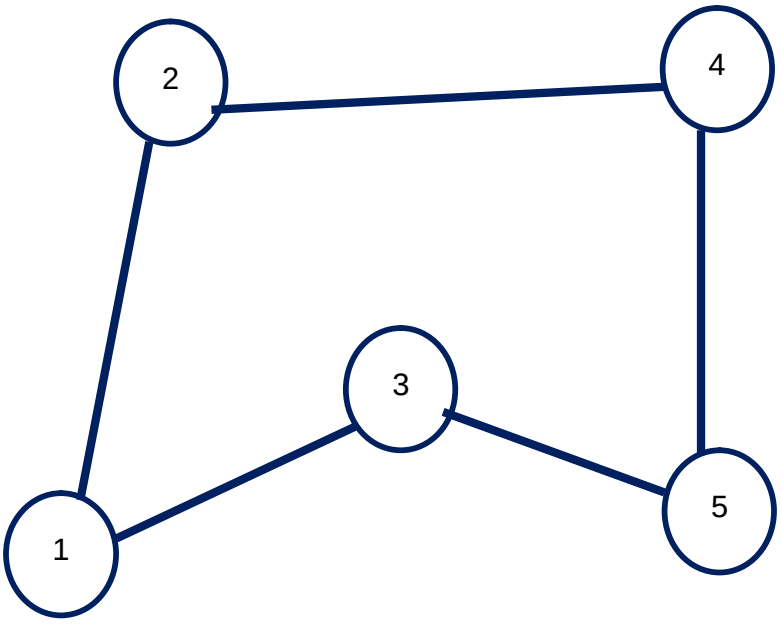
Ví dụ: Xác định chu trình Hamilton trên đồ thị sau



Hình 2.16 Ví dụ chu trình Hamilton

Gọi C là chu trình cần xác định. Ban đầu C là tập rỗng.

Theo nguyên tắc xác định chu trình Hamilton thì 2 đỉnh có bậc 2 là 1 và 5 thì 2 cạnh kề của 2 đỉnh đó sẽ nằm trong chu trình Hamilton. Như vậy ta sẽ chọn các cạnh (1, 2), (1, 3), (5,3), (5,4) cho vào chu trình C. Lúc này đỉnh 3 đã chọn được 2 cạnh kề với nó là (1, 3), (5,3) nên có thể bỏ 2 cạnh (3, 2) và (3, 4) đi. Và cuối cùng chọn cạnh (2, 4) cho vào C ta có được chu trình Hamilton như hình 2.17.



Hình 2.17 Ví dụ chu trình Hamilton

TÓM TẮT

- *Dây chuyền Euler là dây chuyền đi qua tất cả các cạnh trong đồ thị, mỗi cạnh đi qua đúng một lần. Chu trình Euler là dây chuyền Euler có đỉnh đầu và đỉnh cuối trùng nhau.*
- *Đường đi Euler là đường đi qua tất cả các cạnh của đồ thị, mỗi cạnh đúng một lần. Mạch Euler là đường đi Euler có đỉnh đầu và đỉnh cuối trùng nhau.*
- *Đồ thị Euler vô hướng là đồ thị vô hướng có chứa một chu trình Euler. Đồ thị vô hướng G là đồ thị Euler khi và chỉ khi G liên thông và d(x) chẵn với mọi $x \in X$.*
- *Đồ thị có hướng G là đồ thị Euler khi và chỉ khi G liên thông và $d^+(x)=d^-(x)$ với mọi $x \in X$. Đồ thị Euler có hướng là đồ thị có hướng có chứa một mạch Euler.*
- *Cạnh e của đồ thị G được gọi là cầu nếu xóa e khỏi đồ thị thì làm tăng số thành phần liên thông của G.*
- *Thuật toán Fleury dùng để xác định chu trình Euler trong đồ thị G liên thông và có tất cả các đỉnh đều là đỉnh bậc chẵn.*
- *Dây chuyền Hamilton là dây chuyền đi qua tất cả các đỉnh của đồ thị mỗi đỉnh đúng một lần.*
- *Chu trình Hamilton là một dây chuyền Hamilton và một cạnh trong đồ thị nối đỉnh đầu của dây chuyền với đỉnh cuối của nó.*
- *Đồ thị Hamilton là đồ thị có chứa một chu trình Hamilton. Đồ thị nửa Hamilton là đồ thị có chứa một dây chuyền Hamilton.*
- *Quy tắc xác định chu trình Hamilton.*

BÀI TẬP

1. Với giá trị nào của n các đồ thị sau đây có chu trình Euler ?

   a) K_n

   b) C_n

   c) W_n

   d) Q_n.

2. Với giá trị nào của m và n các đồ thị phân đôi đầy đủ $K_{m,n}$ có:

   a) chu trình Euler ?

   b) đường đi Euler ?

3. Với giá trị nào của m và n đồ thị phân đôi đầy đủ $K_{m,n}$ có chu trình Hamilton ?

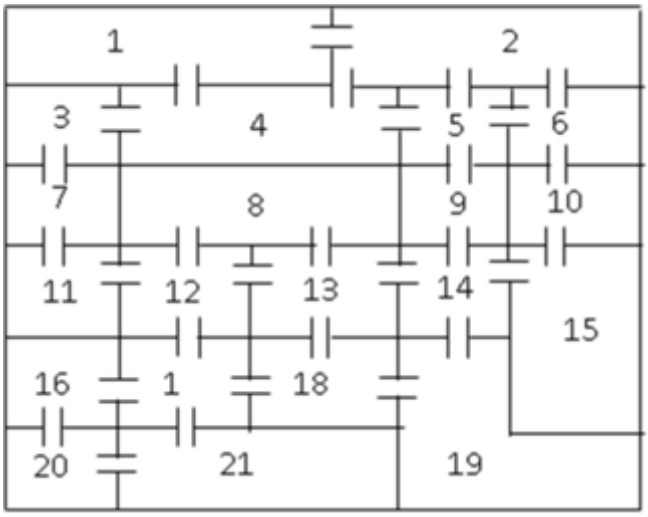
4. Chứng minh rằng đồ thị lập phương Q_n là một đồ thị Hamilton. Vẽ cây liệt kê tất cả các chu trình Hamilton của đồ thị lập phương Q_3.

5. Trong một cuộc họp có 15 người mỗi ngày ngồi với nhau quanh một bàn tròn một lần. Hỏi có bao nhiêu cách sắp xếp sao cho mỗi lần ngồi họp, mỗi người có hai người bên cạnh là bạn mới, và sắp xếp như thế nào ?

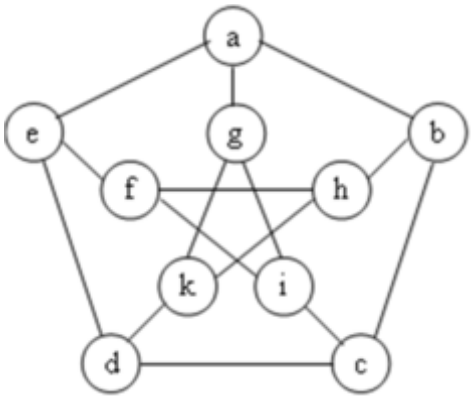
6. Hiệu trưởng mời 2n ($n \geq 2$) sinh viên giỏi đến dự tiệc. Mỗi sinh viên giỏi quen ít nhất n sinh viên giỏi khác đến dự tiệc. Chứng minh rằng luôn luôn có thể xếp tất cả các sinh viên giỏi ngồi xung quanh một bàn tròn, để mỗi người ngồi giữa hai người mà sinh viên đó quen.

7. Chứng minh rằng con mã không thể đi qua tất cả các ô của một bàn cờ có 4x 4 hoặc 5 x 5 ô vuông, mỗi ô chỉ một lần, rồi trở về chỗ cũ.

8. Một ông vua đã xây dựng một lâu đài để cất báu vật. Người ta tìm thấy sơ đồ của lâu đài (hình sau) với lời dặn: muốn tìm báu vật, chỉ cần từ một trong các phòng bên ngoài cùng (số 1, 2, 6, 10, ...), đi qua tất cả các cửa phòng, mỗi cửa chỉ một lần; báu vật được giấu sau cửa cuối cùng. Hãy tìm nơi giấu báu vật.



9. Đồ thị cho trong hình sau gọi là đồ thị Peterson P.



a) Tìm một đường đi Hamilton trong P.

b) Chứng minh rằng P \ {v}, với v là một đỉnh bất kỳ của P, là một đồ thị Hamilton.

10. Cho thí dụ về:

a) Đồ thị có một chu trình vừa là chu trình Euler vừa là chu trình Hamilton;

b) Đồ thị có một chu trình Euler và một chu trình Hamilton, nhưng hai chu trình đó không trùng nhau;

c) Đồ thị có 6 đỉnh, là đồ thị Hamilton, nhưng không phải là đồ thị Euler;

d) Đồ thị có 6 đỉnh, là đồ thị Euler, nhưng không phải là đồ thị Hamilton.

BÀI 3: CÂY

Sau khi học xong bài này, học viên có thể:

- *Hiểu được định nghĩa như thế nào là cây. Các tính chất cơ bản của cây.*
- *Hiểu được như thế nào là cây khung của một đồ thị.*
- *Làm quen với đồ thị có trọng lượng (hay còn gọi là trọng số).*
- *Hiểu được cách tìm cây khung và cây khung có tổng trọng lượng nhỏ nhất bằng hai thuật toán là Prim và Kruskal.*
- *Hiểu được cây có gốc là gì. Ôn lại các phép duyệt cây.*

ĐỊNH NGHĨA VÀ CÁC TÍNH CHẤT CƠ BẢN

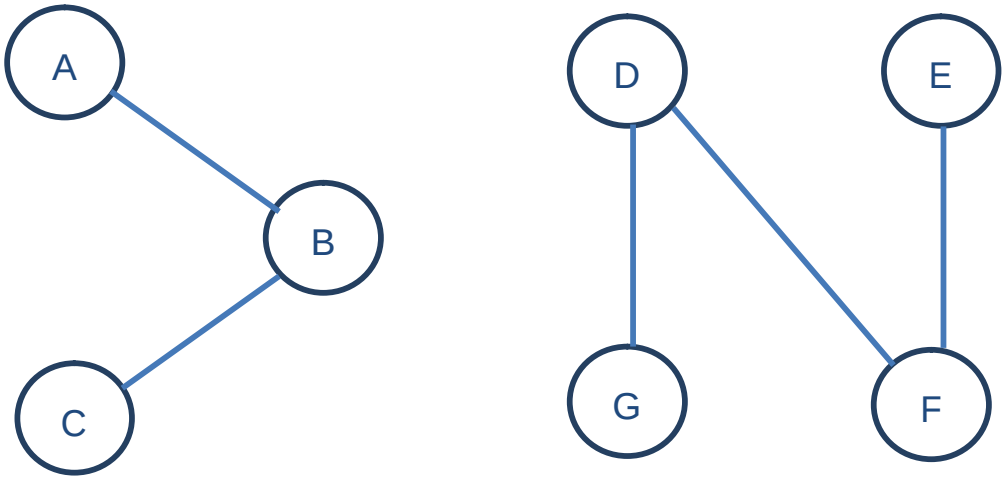
Cây đã được dùng từ năm 1857, khi nhà toán học Anh tên là Arthur Cayley dùng cây để xác định những dạng khác nhau của hợp chất hoá học. Từ đó cây đã được dùng để giải nhiều bài toán trong nhiều lĩnh vực khác nhau. Cây rất hay được sử dụng trong tin học. Chẳng hạn, người ta dùng cây để xây dựng các thuật toán rất có hiệu quả để định vị các phần tử trong một danh sách. Cây cũng dùng để xây dựng các mạng máy tính với chi phí rẻ nhất cho các đường điện thoại nối các máy phân tán. Cây cũng được dùng để tạo ra các mã có hiệu quả để lưu trữ và truyền dữ liệu. Dùng cây có thể mô hình các thủ tục mà để thi hành nó cần dùng một dãy các quyết định. Vì vậy cây đặc biệt có giá trị khi nghiên cứu các thuật toán sắp xếp.

3.1.1 Định nghĩa

Cây là một đồ thị vô hướng liên thông, không chứa chu trình và có ít nhất hai đỉnh.

Lưu ý: cây không chứa khuyên và cạnh song song.

Rừng là một đồ thị gồm p thành phần liên thông, trong đó mỗi thành phần liên thông là một cây.



Hình 3.1 Cây

Đồ thị gồm 7 đỉnh A, B, C, D, E, F, G là một rừng trong đó có 2 cây: cây thứ 1 là thành phần liên thông gồm 3 đỉnh A, B, C; cây thứ 2 là thành phần liên thông gồm 4 đỉnh D, E, F, G.

3.1.2 Các tính chất cơ bản

3.1.2.1 Định lý về sự tồn tại đỉnh treo

Nếu T là một cây có n đỉnh thì T có ít nhất hai đỉnh treo.

Chứng minh: Lấy một cạnh (a, b) tuỳ ý của cây T. Trong tập hợp các đường đi sơ cấp chứa cạnh (a,b), ta lấy đường đi từ u đến v dài nhất. Vì T là một cây nên u $\neq$ v. Mặt khác, u và v phải là hai đỉnh treo, vì nếu một đỉnh, u chẳng hạn, không phải là đỉnh treo thì u phải là đầu mút của một cạnh (u, x), với x là đỉnh không thuộc đường đi từ u đến v. Do đó, đường đi sơ cấp từ x đến v, chứa cạnh (a, b), dài hơn đường đi từ u đến v, trái với tính chất đường đi từ u đến v đã chọn.

3.1.2.2 Định lý

Cho T là một đồ thị có n $\geq$ 2 đỉnh. Các điều sau là tương đương:

1) T là một cây.

2) T liên thông và có n–1 cạnh.

3) T không chứa chu trình và có n–1 cạnh.

4) T liên thông và mỗi cạnh là cầu.

5) Giữa hai đỉnh phân biệt bất kỳ của T luôn có duy nhất một đường đi sơ cấp.

6) T không chứa chu trình nhưng khi thêm một cạnh mới thì có được một chu trình duy nhất.

Chứng minh: 1)⇒2) Chỉ cần chứng minh rằng một cây có n đỉnh thì có n–1 cạnh. Ta chứng minh bằng quy nạp. Điều này hiển nhiên khi n=2. Giả sử cây có k đỉnh thì có k–1 cạnh, ta chứng minh rằng cây T có k+1 đỉnh thì có k cạnh. Thật vậy, trong T nếu ta xoá một đỉnh treo và cạnh treo tương ứng thì đồ thị nhận được là một cây k đỉnh, cây này có k–1 cạnh, theo giả thiết quy nạp. Vậy cây T có k cạnh.

2)⇒3) Nếu T có chu trình thì bỏ đi một cạnh trong chu trình này thì T vẫn liên thông. Làm lại như thế cho đến khi trong T không còn chu trình nào mà vẫn liên thông, lúc đó ta được một cây có n đỉnh nhưng có ít hơn n–1 cạnh, trái với 2).

3)⇒4) Nếu T có k thành phần liên thông T_1, ..., T_k lần lượt có số đỉnh là n_1, ..., n_k (với $n_1+n_2+ \dots +n_k=n$) thì mỗi T_i là một cây nên nó có số cạnh là $n_i–1$. Vậy ta có

$$n–1=(n_1–1)+(n_2–1)+ \dots +(n_k–1)=(n_1+n_2+ \dots +n_k)–k=n–k.$$

Do đó k=1 hay T liên thông. Hơn nữa, khi bỏ đi một cạnh thì T hết liên thông, vì nếu còn liên thông thì T là một cây n đỉnh với n–2 cạnh, trái với điều đã chứng minh ở trên.

4)⇒5) Vì T liên thông nên giữa hai đỉnh phân biệt bất kỳ của T luôn có một đường đi sơ cấp, nhưng không thể được nối bởi hai đường đi sơ cấp vì nếu thế, hai đường đó sẽ tạo ra một chu trình và khi bỏ một cạnh thuộc chu trình này, T vẫn liên thông, trái với giả thiết.

5)⇒6) Nếu T chứa một chu trình thì hai đỉnh bất kỳ trên chu trình này sẽ được nối bởi hai đường đi sơ cấp. Ngoài ra, khi thêm một cạnh mới (u, v), cạnh này sẽ tạo nên với đường đi sơ cấp duy nhất nối u và v một chu trình duy nhất.

6)⇒1) Nếu T không liên thông thì thêm một cạnh nối hai đỉnh ở hai thành phần liên thông khác nhau ta không nhận được một chu trình nào. Vậy T liên thông, do đó nó là một cây.

CÂY KHUNG VÀ BÀI TOÁN CÂY KHUNG NGẮN NHẤT

3.1.3 Định nghĩa cây khung

Cho G=(X, E) là một đồ thị liên thông và T=(X, F) là một đồ thị bộ phận của G. Nếu T là cây thì T được gọi là một cây khung của G.

Cây khung còn có thể được gọi bằng các tên khác như cây bao trùm, cây phủ hoặc là cây tối đại.

Mọi đồ thị liên thông đều có chứa ít nhất một cây khung.

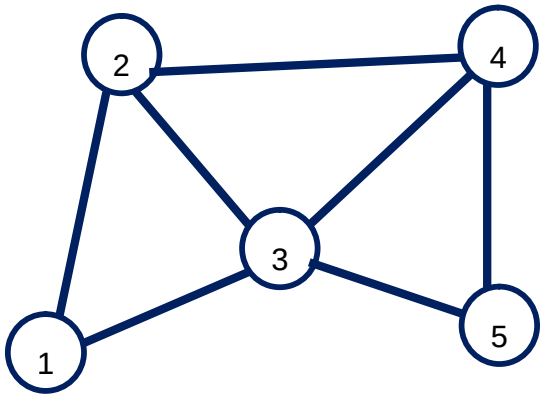
3.1.4 Thuật toán tìm cây khung

Input: đồ thị liên thông G=(X, E), X gồm N đỉnh.

Output: cây khung T=(V, U) của G.

1. Chọn tùy ý 1 đỉnh $v \in X$ và khởi tạo V := { v }; U := ∅;
2. Chọn $w \in X \setminus V$ sao cho $\exists e \in E$, e nối w với một đỉnh trong V.
3. $V := V \cup \{w\}$; $U := U \cup \{e\}$
4. Nếu U đủ N-1 cạnh thì dừng, ngược lại lặp từ thao tác số 2.

Ví dụ: Tìm cây khung của đồ thị sau



Hình 3.2 Ví dụ cây khung của đồ thị

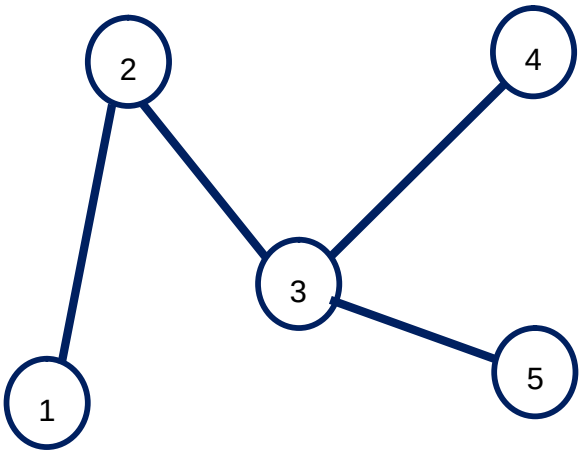
Bước 0 (bước khởi tạo): V={1}, U=∅; X\V= {2, 3, 4, 5}.

Bước 1: Các cạnh tạo bởi 1 đỉnh trong V và 1 đỉnh trong X\V là (1, 2), (1, 3). Chọn cạnh (1, 2), bỏ đỉnh 2 vào V, có V={1, 2}, U={(1, 2)}, X\V={3, 4, 5}.

Bước 2: Các cạnh tạo bởi 1 đỉnh trong V và 1 đỉnh trong X\V là (1, 3), (2, 3), (2, 4). Chọn cạnh (2, 3), bỏ đỉnh 3 vào V, có V={1, 2, 3}, U={(1, 2), (2, 3)}, X\V={4, 5}.

Bước 3: Các cạnh tạo bởi 1 đỉnh trong V và 1 đỉnh trong X\V là (2, 4), (3, 4), (3, 5). Chọn cạnh (3, 4), bỏ đỉnh 4 vào V, có V={1, 2, 3, 4}, U={(1, 2), (2, 3), (3, 4)}, X\V={5}.

Bước 4: Các cạnh tạo bởi 1 đỉnh trong V và 1 đỉnh trong X\V là (3, 5) và (4, 5). Chọn cạnh (3, 5), bỏ 5 vào V, có V={1, 2, 3, 4, 5}, U={(1, 2), (2, 3), (3, 4), (3, 5)}, X\V=∅. Thuật toán dừng. Cây khung của đồ thị như sau:



Hình 3.3 Ví dụ cây khung của đồ thị

3.1.5 Cây khung ngắn nhất

3.1.5.1 Đồ thị có trọng lượng

Cho G= (X, E). G được gọi là đồ thị có trọng lượng nếu mỗi cạnh của G được tương ứng với một số thực, nghĩa là có một ánh xạ như sau:

$$L: E \rightarrow R$$

$$e \rightarrow L(e)$$

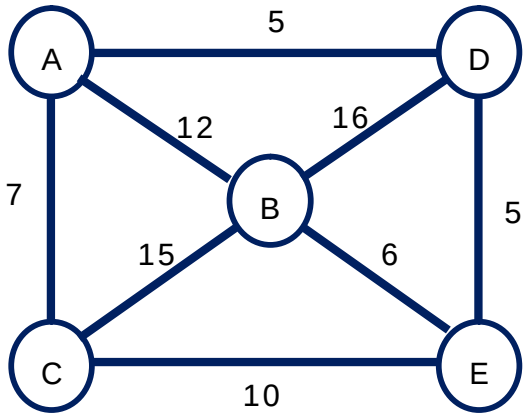
Trọng lượng của một cây T của G bằng với tổng trọng lượng các cạnh trong cây: L(T) = ΣL(e) với (e∈T)

Cây khung ngắn nhất là cây khung có trọng lượng nhỏ nhất của G.

3.1.5.2 Ma trận trọng lượng

Trong các thuật toán tìm cây tối đại ngắn nhất chúng ta có thể bỏ đi hướng các cạnh và các khuyên; đối với các cạnh song song thì có thể bỏ đi và chỉ để lại một cạnh trọng lượng nhỏ nhất trong chúng. Vì vậy đồ thị có thể biểu diễn bằng ma trận trọng lượng L_{NxN} được qui ước như sau:

- L_{ij} = trọng lượng cạnh nhỏ nhất nối i đến j (nếu có)
- L_{ij} = ∞ nếu không có cạnh nối i đến j



Hình 3.4 Đồ thị có trọng lượng

Ma trận trọng lượng của đồ thị trên là

$$L = \begin{pmatrix} \infty & 12 & 7 & 5 & \infty \\ 12 & \infty & 15 & 16 & 6 \\ 7 & 15 & \infty & \infty & 10 \\ 5 & 16 & \infty & \infty & 5 \\ \infty & 6 & 10 & 5 & \infty \end{pmatrix}$$

3.1.5.3 Thuật toán Prim

Input: đồ thị liên thông G=(X, E), X gồm N đỉnh.

Output: cây khung ngắn nhất T=(V, U) của G.

1. Chọn tùy ý 1 đỉnh v∈X và khởi tạo V := { v }; U := ∅;
2. Chọn cạnh e có trọng lượng nhỏ nhất trong các cạnh (w, v) mà w∈X\V và v∈V.
3. V := V ∪ {w}; U := U ∪ {e}
4. Nếu U đủ N-1 cạnh thì dừng, ngược lại lặp từ thao tác số 2.

Ví dụ: Tìm cây khung ngắn nhất của đồ thị hình 3.4 bằng thuật toán Prim

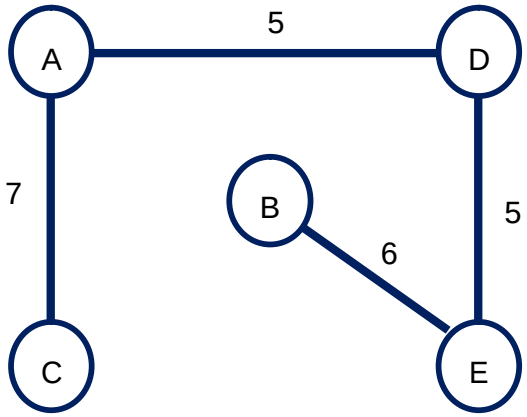
Bước 0: V={A}, U=∅; X\V= {B, C, D, E}

Bước 1: Các cạnh tạo bởi 1 đỉnh trong V và 1 đỉnh trong X\V là (A, D), (A, B) và (A, C). Chọn cạnh có trọng lượng nhỏ nhất trong 3 cạnh trên là (A, D) có trọng lượng bằng 5, bỏ đỉnh D vào V, có V={A, D}, U={(A, D)}, X\V={B, C, E}.

Bước 2: Chọn cạnh (D, E) có trọng lượng là 5, bỏ E vào V, có V={A, D, E}, U={(A, D), (D, E)}, X\V={B, C}.

Bước 3: Chọn cạnh (B, E) có trọng lượng là 6, bỏ B vào V, có V={A, D, E, B}, U={(A, D), (D, E), (B, E)}, X\V={C}.

Bước 4: Chọn cạnh (A, C) có trọng lượng là 7, bỏ C vào V, có V={A, D, E, B, C}, U={(A, D),(D, E), (B, E), (A, C)}, X\V=∅. Thuật toán kết thúc. Ta có L(T)= 5+5+6+7=23.



Hình 3.5 Cây khung của đồ thị

3.1.5.4 Thuật toán Kruskal

Input: đồ thị G=(X, E) liên thông, X gồm N đỉnh

Output: cây khung ngắn nhất T=(V, U) của G

1. Sắp xếp các cạnh trong G tăng dần theo trọng lượng; khởi tạo T := ∅.
2. Lần lượt lấy từng cạnh e thuộc danh sách đã sắp xếp. Nếu T+{e} không chứa chu trình thì kết nạp e vào T, T := T+{e}.
3. Nếu T đủ N-1 cạnh thì dừng; ngược lại, lặp thao tác 2.

Ví dụ: Tìm cây khung ngắn nhất của đồ thị hình 3.4 bằng thuật toán Kruskal

Bước 0: Sắp xếp các cạnh theo thứ tự tăng dần của trọng lượng (A, D), (D, E), (E, B), (A, C), (E, C), (A, B), (C, B), (B, D). T=∅.

Bước 1: Bỏ (A, D) vào T không tạo thành chu trình nên chọn cạnh (A, D). T={(A, D)}.

Bước 2: Bỏ (D, E) vào T không tạo thành chu trình nên chọn cạnh (D, E). T={(A, D), (D, E)}.

Bước 3: Bỏ (E, B) vào T không tạo thành chu trình nên chọn cạnh (E, B). T={(A, D), (D, E), (E, B)}.

Bước 4: Bỏ (A, C) vào T không tạo thành chu trình nên chọn cạnh (A, C). T={(A, D), (D, E), (E, B),(A, C)}. Thuật toán kết thúc, cây khung ngắn nhất T có L(T)=23.

CÂY CÓ GỐC

3.1.6 Đồ thị có gốc

3.1.6.1 Định nghĩa

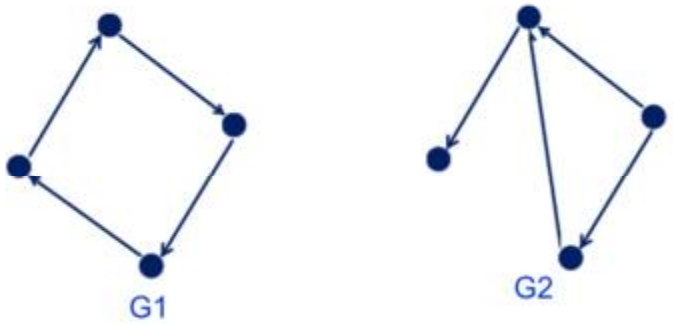
Cho đồ thị có hướng G=(X, E). Ta nói G là một đồ thị có gốc nếu tồn tại đỉnh r∈X sao cho từ r có đường đi đến v, ∀∈X.

Hình 3.6 Đồ thị có gốc

Ở hình trên, G1 là đồ thị có gốc còn G2 thì không vì trên G2 không tìm thấy được đỉnh mà từ đó có thể đi đến tất cả các đỉnh còn lại.

3.1.6.2 Đồ thị liên thông mạnh

Cho đồ thị có hướng G=(X, E). Ta nói G là đồ thị liên thông mạnh khi và chỉ khi $\forall i, j \in X$ luôn tồn tại đường đi từ i đến j và đường đi từ j đến i.

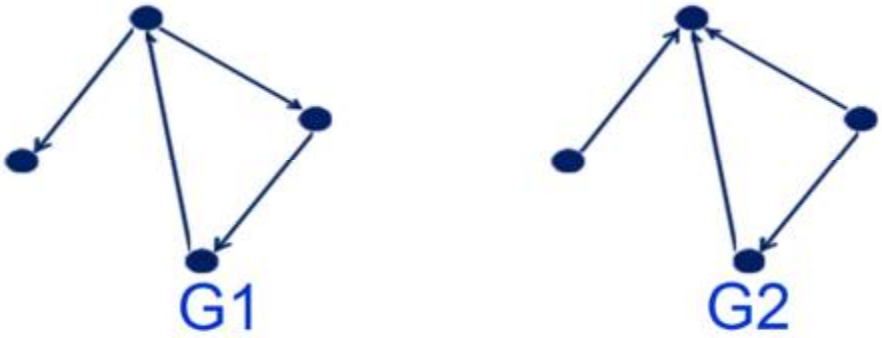


Hình 3.7 Đồ thị liên thông mạnh

G1 là đồ thị liên thông mạnh. G2 không phải là đồ thị liên thông mạnh.

3.1.6.3 Đồ thị tựa liên thông mạnh

Cho đồ thị có hướng G=(X, E). Ta nói G là đồ thị tựa liên thông mạnh khi và chỉ khi $\forall i, j \in X, \exists k \in X$ sao cho có đường đi từ k đến i và có đường đi từ k đến j.



Hình 3.8 Đồ thị tựa liên thông mạnh

Đồ thị G1 là đồ thị tựa liên thông mạnh còn đồ thị G2 thì không.

3.1.7 Cây có hướng

3.1.7.1 Định nghĩa

Cây có hướng là đồ thị có hướng mà đồ thị vô hướng nền của nó là một cây.

3.1.7.2 Định lý

Cho G là đồ thị có hướng.

- Nếu G có chứa một đồ thị bộ phận là cây có hướng thì G tựa liên thông mạnh.
- Nếu G tựa liên thông mạnh thì G có chứa một đồ thị bộ phận là cây có hướng.

Nếu G tựa liên thông mạnh, T là một cây có hướng là đồ thị bộ phận G thì T cũng được gọi là cây có hướng tối đại của G.

3.1.8 Cây có gốc

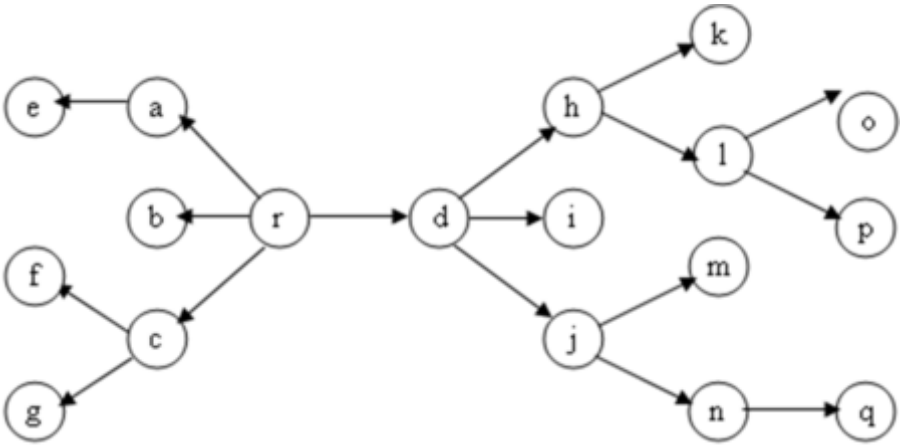
Cây có gốc là một cây có hướng, trong đó có một đỉnh đặc biệt, gọi là gốc, từ gốc có đường đi đến mọi đỉnh khác của cây.

Trong cây có gốc thì gốc r có bậc vào bằng 0, còn tất cả các đỉnh khác đều có bậc vào bằng 1.

Một cây có gốc thường được vẽ với gốc r ở trên cùng và cây phát triển từ trên xuống, gốc r gọi là đỉnh mức 0. Các đỉnh kề với r được xếp ở phía dưới và gọi là đỉnh mức 1. Đỉnh ngay dưới đỉnh mức 1 là đỉnh mức 2, ...

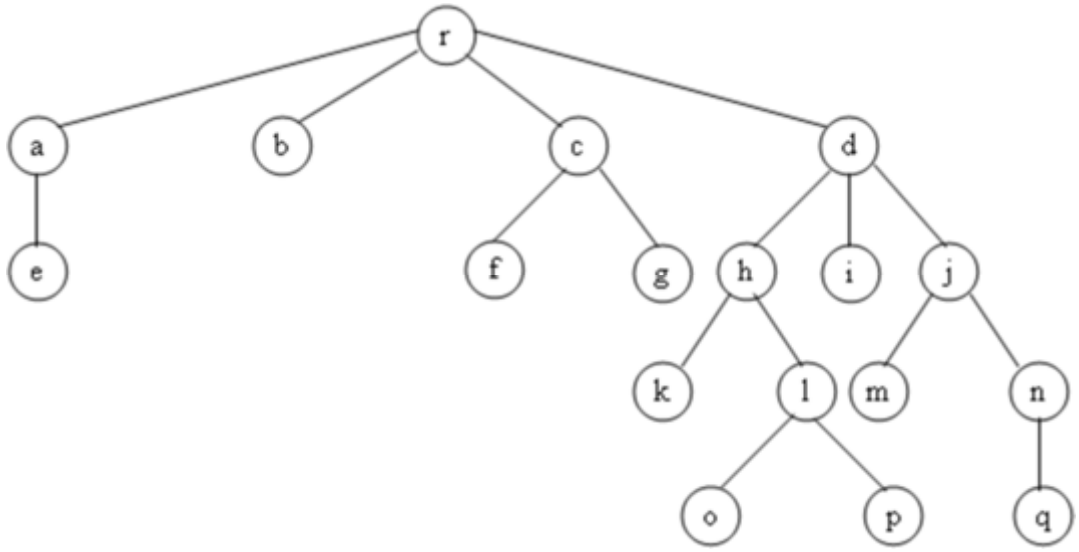
Tổng quát, trong một cây có gốc thì v là đỉnh mức k khi và chỉ khi đường đi từ r đến v có độ dài bằng k.

Mức lớn nhất của một đỉnh bất kỳ trong cây gọi là chiều cao của cây



Hình 3.9 Cây có gốc

Cây có gốc ở hình trên thường được vẽ như trong hình dưới đây để làm rõ mức của các đỉnh.



Hình 3.10 Cây có gốc

Trong cây có gốc, mọi cung đều có hướng từ trên xuống, vì vậy vẽ mũi tên để chỉ hướng đi là không cần thiết; do đó, người ta thường vẽ các cây có gốc như là cây nền của nó.

Cho cây T có gốc $r=v_0$. Giả sử v_0, v_1, ..., v_{n-1}, v_n là một đường đi trong T. Ta gọi:

- v_{i+1} là con của v_i và v_i là cha của v_{i+1}.
- v_0, v_1, ..., v_{n-1} là các tổ tiên của v_n và v_n là dòng dõi của v_0, v_1, ..., v_{n-1}.

Đỉnh treo v_n là đỉnh không có con; đỉnh treo cũng gọi là lá hay đỉnh ngoài; một đỉnh không phải lá là một đỉnh trong.

Một cây có gốc T được gọi là cây m-phân nếu mỗi đỉnh của T có nhiều nhất là m con. Với m=2, ta có một cây nhị phân.

Trong một cây nhị phân, mỗi con được chỉ rõ là con bên trái hay con bên phải; con bên trái (t.ư. phải) được vẽ phía dưới và bên trái (t.ư. phải) của cha.

Cây có gốc T được gọi là một cây m-phân đầy đủ nếu mỗi đỉnh trong của T đều có m con.

Mệnh đề: Một cây m-phân đầy đủ có i đỉnh trong thì có mi+1 đỉnh và có (m−1)i+1 lá.

Chứng minh: Mọi đỉnh trong của cây m-phân đầy đủ đều có bậc ra là m, còn lá có bậc ra là 0, vậy số cung của cây này là mi và do đó số đỉnh của cây là mi+1. Gọi *l* là số lá thì ta có *l*+i=mi+1, nên *l*=(m−1)i+1.

Mệnh đề:

1) Một cây m-phân có chiều cao h thì có nhiều nhất là m^h lá.

2) Một cây m-phân có *l* lá thì có chiều cao $h \geq [\log_m l]$.

Chứng minh:

1) Mệnh đề được chứng minh bằng quy nạp theo h. Mệnh đề hiển nhiên đúng khi h=1. Giả sử mọi cây có chiều cao $k \leq h-1$ đều có nhiều nhất m^{k-1} lá (với $h \geq 2$). Xét cây T có chiều cao h. Bỏ gốc khỏi cây ta được một rừng gồm không quá m cây con, mỗi cây con này có chiều cao $\leq h-1$. Do giả thiết quy nạp, mỗi cây con này có nhiều nhất là m^{h-1} lá. Do lá của những cây con này cũng là lá của T, nên T có nhiều nhất là $m.m^{h-1}=m^h$ lá.

2) $l \leq m^h \Leftrightarrow h \geq [\log_m l]$.

DUYỆT CÂY NHỊ PHÂN

3.1.9 Định nghĩa

Trong nhiều trường hợp, ta cần phải "điểm danh" hay "thăm" một cách có hệ thống mọi đỉnh của một cây nhị phân, mỗi đỉnh chỉ một lần. Ta gọi đó là việc duyệt cây nhị phân hay đọc cây nhị phân.

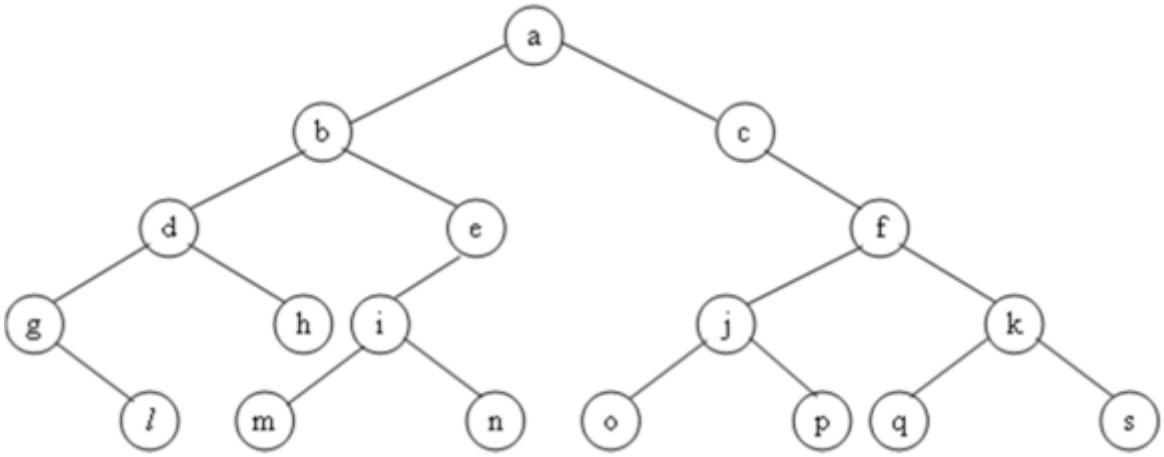
Có nhiều thuật toán duyệt cây nhị phân, các thuật toán đó khác nhau chủ yếu ở thứ tự thăm các đỉnh.

Cây nhị phân T có gốc r được ký hiệu là T(r). Giả sử r có con bên trái là u, con bên phải là v. Cây có gốc u và các đỉnh khác là mọi dòng dõi của u trong T gọi là cây con bên trái của T, ký hiệu T(u). Tương tự, ta có cây con bên phải T(v) của T. Một cây T(r) có thể không có cây con bên trái hay bên phải.

Sau đây là ba trong các thuật toán duyệt cây nhị phân T(r). Các thuật toán đều được trình bày đệ quy. Chú ý rằng khi cây T(x) chỉ là môt đỉnh x thì "duyệt T(x)" có nghĩa là "thăm đỉnh x".

3.1.10 Các phương pháp duyệt cây nhị phân

Áp dụng các phương pháp duyệt cây nhị phân cho đồ thị sau



Hình 3.11 Cây nhị phân

3.1.10.1 Thuật toán tiền thứ tự

1. Thăm gốc r.

2. Duyệt cây con bên trái của T(r) theo tiền thứ tự.

3. Duyệt cây con bên phải của T(r) theo tiền thứ tự.

Duyệt cây nhị phân T(a) trong hình trên theo tiền thứ tự:

1. Thăm a

2. Duyệt T(b)

2.1. Thăm b

2.2. Duyệt T(d)

2.2.1. Thăm d

2.2.2. Duyệt T(g)

2.2.2.1. Thăm g

2.2.2.3. Duyệt T(*l*): Thăm *l*

2.2.3. Duyệt T(h): Thăm h

2.3. Duyệt T(e)

2.3.1. Thăm e

2.3.2. Duyệt T(i)

2.3.2.1. Thăm i

2.3.2.2. Duyệt T(m): Thăm m

2.3.2.3. Duyệt T(n): Thăm n

3. Duyệt T(c)

3.1. Thăm c

3.3. Duyệt T(f)

3.3.1.Thăm f

3.3.2. Duyệt T(j)

3.3.2.1. Thăm j

3.3.2.2. Duyệt T(o): Thăm o

3.3.2.3. Duyệt T(p): Thăm p

3.3.3. Duyệt T(k)

3.3.3.1. Thăm k

3.3.3.2. Duyệt T(q): Thăm q

3.3.3.3. Duyệt T(s): Thăm s

Kết quả duyệt cây T(a) theo tiền thứ tự là:

a, b, d, g, *l*, h, e, i, m, n, c, f, j, o, p, k, q, s.

3.1.10.2 Thuật toán trung thứ tự

1. Duyệt cây con bên trái của T(r) theo trung thứ tự.

2. Thăm gốc r.

3. Duyệt cây con bên phải của T(r) theo trung thứ tự.

Duyệt cây nhị phân T(a) trong hình trên theo trung thứ tự:

1. Duyệt T(b)

1.1. Duyệt T(d)

1.1.1. Duyệt T(g)

1.1.1.2. Thăm g

1.1.1.3. Duyệt T(*l*): thăm *l*

1.1.2. Thăm d

1.1.3. Duyệt T(h): Thăm h

1.2. Thăm b

1.3. Duyệt T(e)

1.3.1. Duyệt T(i)

1.3.1.1. Duyệt T(m): Thăm m

1.3.1.2. Thăm i

1.3.1.3. Duyệt T(n): Thăm n

1.3.2. Thăm e

2. Thăm a

3. Duyệt T(c)

3.2. Thăm c

3.3. Duyệt T(f)

3.3.1. Duyệt T(j)

3.3.1.1. Duyệt T(o): Thăm o

3.3.1.2. Thăm j

3.3.1.3. Duyệt T(p): Thăm p

3.3.2. Thăm f

3.3.3. Duyệt T(k)

3.3.3.1. Duyệt T(q): Thăm q

3.3.3.2. Thăm k

3.3.3.3. Duyệt T(s): Thăm s

Kết quả duyệt cây T(a) theo trung thứ tự là:

g, *l*, d, h, b, m, i, n, e, a, c, o, j, p, f, q, k, s.

3.1.10.3 Thuật toán hậu thứ tự

1. Duyệt cây con bên trái của T(r) theo hậu thứ tự.

2. Duyệt cây con bên phải của T(r) theo hậu thứ tự.

3. Thăm gốc r.

Duyệt cây nhị phân T(a) trong hình trên theo hậu thứ tự:

1. Duyệt T(b)

1.1. Duyệt T(d)

1.1.1. Duyệt T(g)

1.1.1.2. Duyệt T(*l*): thăm *l*

1.1.1.3. Thăm g

1.1.2. Duyệt T(h): thăm h

1.1.3. Thăm d

1.2. Duyệt T(e)

1.2.1. Duyệt T(i)

1.2.1.1. Duyệt T(m): Thăm m

1.2.1.2. Duyệt T(n): Thăm n

1.2.1.3. Thăm i

1.2.3. Thăm e

1.3. Thăm b

2. Duyệt T(c)

2.2. Duyệt T(f)

2.2.1. Duyệt T(j)

2.2.1.1. Duyệt T(o): Thăm o

2.2.1.2. Duyệt T(p): Thăm p

2.2.1.3. Thăm j

2.2.2. Duyệt T(k)

2.2.2.1. Duyệt T(q): Thăm q

2.2.2.2. Duyệt T(s): Thăm s

2.2.2.3. Thăm k

2.2.3. Thăm f

2.3. Thăm c

3. Thăm a

Kết quả duyệt cây T(a) theo hậu thứ tự là:

l, g, h, d, m, n, i, e, b, o, p, j, q, s, k, f, c, a.

3) Thuật toán hậu thứ tự:

1. Duyệt cây con bên trái của T(r) theo hậu thứ tự.

2. Duyệt cây con bên phải của T(r) theo hậu thứ tự.

3. Thăm gốc r.

Duyệt cây nhị phân T(a) trong hình trên theo hậu thứ tự:

1. Duyệt T(b)

1.1. Duyệt T(d)

1.1.1. Duyệt T(g)

1.1.1.2. Duyệt T(*l*): thăm *l*

1.1.1.3. Thăm g

1.1.2. Duyệt T(h): thăm h

1.1.3. Thăm d

1.2. Duyệt T(e)

1.2.1. Duyệt T(i)

1.2.1.1. Duyệt T(m): Thăm m

1.2.1.2. Duyệt T(n): Thăm n

1.2.1.3. Thăm i

1.2.3. Thăm e

1.3. Thăm b

2. Duyệt T(c)

2.2. Duyệt T(f)

2.2.1. Duyệt T(j)

2.2.1.1. Duyệt T(o): Thăm o

2.2.1.2. Duyệt T(p): Thăm p

2.2.1.3. Thăm j

2.2.2. Duyệt T(k)

2.2.2.1. Duyệt T(q): Thăm q

2.2.2.2. Duyệt T(s): Thăm s

2.2.2.3. Thăm k

2.2.3. Thăm f

2.3. Thăm c

3. Thăm a

Kết quả duyệt cây T(a) theo trung thứ tự là:

l, g, h, d, m, n, i, e, b, o, p, j, q, s, k, f, c, a.

TÓM TẮT

- *Cây là một đồ thị vô hướng liên thông, không chứa chu trình và có ít nhất hai đỉnh. Rừng là một đồ thị gồm p thành phần liên thông, trong đó mỗi thành phần liên thông là một cây.*
- *Nếu T là một cây có n đỉnh thì T có ít nhất hai đỉnh treo.*
- *Cho T là một đồ thị có $n \geq 2$ đỉnh. Các điều sau là tương đương:*
    - *T là một cây.*
    - *T liên thông và có $n-1$ cạnh.*
    - *T không chứa chu trình và có $n-1$ cạnh.*
    - *T liên thông và mỗi cạnh là cầu.*
    - *Giữa hai đỉnh phân biệt bất kỳ của T luôn có duy nhất một đường đi sơ cấp.*
    - *T không chứa chu trình nhưng khi thêm một cạnh mới thì có được một chu trình duy nhất.*
- *Cho G=(X, E) là một đồ thị liên thông và T=(X, F) là một đồ thị bộ phận của G. Nếu T là cây thì T được gọi là một cây khung của G.*
- *Ma trận trọng lượng biểu diễn cho đồ thị có trọng lượng.*
- *Hai thuật toán dùng để tìm cây khung ngắn nhất cho một đồ thị là Prim và Kruskal.*
- *Cho đồ thị có hướng G=(X, E). Ta nói G là một đồ thị có gốc nếu tồn tại đỉnh $r \in X$ sao cho từ r có đường đi đến v, $\forall \in X$.*
- *Cây có hướng là đồ thị có hướng mà đồ thị vô hướng nền của nó là một cây.*
- *Cây có gốc là một cây có hướng, trong đó có một đỉnh đặc biệt, gọi là gốc, từ gốc có đường đi đến mọi đỉnh khác của cây.*
- *Ba phương pháp duyệt cây: tiền thứ tự, trung thứ tự và hậu thứ tự.*

BÀI TẬP

1. Vẽ tất cả các cây (không đẳng cấu) có:

a) 4 đỉnh

b) 5 đỉnh

c) 6 đỉnh

2. Một cây có n_2 đỉnh bậc 2, n_3 đỉnh bậc 3, …, n_k đỉnh bậc k. Hỏi có bao nhiêu đỉnh bậc 1?

3. Tìm số tối đa các đỉnh của một cây m-phân có chiều cao h.

4. Có thể tìm được một cây có 8 đỉnh và thoả điều kiện dưới đây hay không? Nếu có, vẽ cây đó ra, nếu không, giải thích tại sao:

a) Mọi đỉnh đều có bậc 1.

b) Mọi đỉnh đều có bậc 2.

c) Có 6 đỉnh bậc 2 và 2 đỉnh bậc 1.

d) Có đỉnh bậc 7 và 7 đỉnh bậc 1.

5. Chứng minh hoặc bác bỏ các mệnh đề sau đây.

a) Trong một cây, đỉnh nào cũng là đỉnh cắt.

b) Một cây có số đỉnh không nhỏ hơn 3 thì có nhiều đỉnh cắt hơn là cầu.

6. Có bốn đội bóng đá A, B, C, D lọt vào vòng bán kết trong giải các đội mạnh khu vực. Có mấy dự đoán xếp hạng như sau:

a) Đội B vô địch, đội D nhì.

b) Đội B nhì, đội C ba.

c) Đội A nhì, đội C tư.

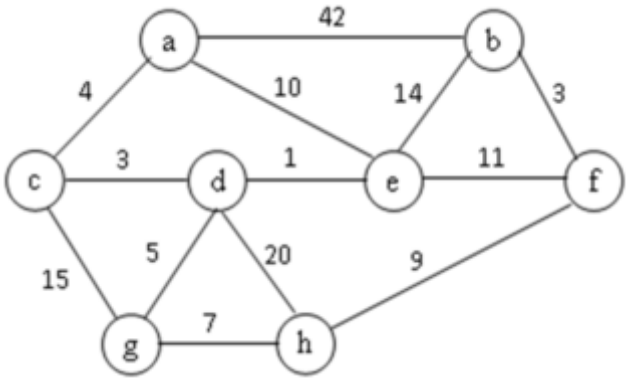
Biết rằng mỗi dự đoán trên đúng về một đội. Hãy cho biết kết quả xếp hạng của các đội.

7. Cây *Fibonacci* có gốc T_n được dịnh nghĩa bằng hồi quy như sau. T_1 và T_2 đều là cây có gốc chỉ gồm một đỉnh và với n=3,4, … cây có gốc T_n được xây dựng từ gốc với T_{n-1} như là cây con bên trái và T_{n-2} như là cây con bên phải.

a) Hãy vẽ 7 cây *Fibonacci* có gốc đầu tiên.

b) Cây *Fibonacci* T_n có bao nhiêu đỉnh, lá và bao nhiêu đỉnh trong. Chiều cao của nó bằng bao nhiêu?

8. Tìm cây khung nhỏ nhất của đồ thị sau theo thuật toán Kruskal và Prim.



9. Tìm cây khung nhỏ nhất bằng thuật toán Prim của đồ thị gồm các đỉnh A, B, C, D, E, F, H, I được cho bởi ma trận trọng số sau.

$$\begin{pmatrix} \infty & 16 & 15 & 23 & 19 & 18 & 32 & 20 \\ 16 & \infty & 13 & 33 & 24 & 20 & 19 & 11 \\ 15 & 13 & \infty & 13 & 29 & 21 & 20 & 19 \\ 23 & 33 & 13 & \infty & 22 & 30 & 21 & 12 \\ 19 & 24 & 29 & 22 & \infty & 34 & 23 & 21 \\ 18 & 20 & 21 & 30 & 34 & \infty & 17 & 14 \\ 32 & 19 & 20 & 21 & 23 & 17 & \infty & 18 \\ 20 & 11 & 19 & 12 & 21 & 14 & 18 & \infty \end{pmatrix}$$

BÀI 4: CÁC BÀI TOÁN ĐƯỜNG ĐI

Sau khi học xong bài này, học viên có thể:

- *Tìm được lời giải phù hợp cho từng bài toán tìm đường đi tối ưu với các điều kiện cho trước.*
- *Hiểu được ý tưởng và các bước thực hiện các thuật toán tìm đường đi như Dijkstra, Floyd.*

GIỚI THIỆU

Trong đời sống, chúng ta thường gặp những tình huống như sau: để đi từ địa điểm A đến địa điểm B trong thành phố, có nhiều đường đi, nhiều cách đi; có lúc ta chọn đường đi ngắn nhất (theo nghĩa cự ly), có lúc lại cần chọn đường đi nhanh nhất (theo nghĩa thời gian) và có lúc phải cân nhắc để chọn đường đi rẻ tiền nhất (theo nghĩa chi phí), v.v...

Có thể coi sơ đồ của đường đi từ A đến B trong thành phố là một đồ thị, với đỉnh là các giao lộ (A và B coi như giao lộ), cạnh là đoạn đường nối hai giao lộ. Trên mỗi cạnh của đồ thị này, ta gán một số dương, ứng với chiều dài của đoạn đường, thời gian đi đoạn đường hoặc cước phí vận chuyển trên đoạn đường đó, ...

Đồ thị có trọng số là đồ thị G=(V,E) mà mỗi cạnh (hoặc cung) $e \in E$ được gán bởi một số thực m(e), gọi là trọng số của cạnh (hoặc cung) e.

Trong phần này, trọng số của mỗi cạnh được xét là một số dương và còn gọi là chiều dài của cạnh đó. Mỗi đường đi từ đỉnh u đến đỉnh v, có chiều dài là m(u,v), bằng tổng chiều dài các cạnh mà nó đi qua. Khoảng cách d(u,v) giữa hai đỉnh u và v là chiều dài đường đi ngắn nhất (theo nghĩa m(u,v) nhỏ nhất) trong các đường đi từ u đến v.

Có thể xem một đồ thị G bất kỳ là một đồ thị có trọng số mà mọi cạnh đều có chiều dài 1. Khi đó, khoảng cách d(u,v) giữa hai đỉnh u và v là chiều dài của đường đi từ u đến v ngắn nhất, tức là đường đi qua ít cạnh nhất.

THUẬT TOÁN DIJKSTRA

4.1.1 Ý tưởng thuật toán Dijkstra

Cho đơn đồ thị liên thông, có trọng số G=(V,E). Tìm khoảng cách $d(u_0,v)$ từ một đỉnh u_0 cho trước đến một đỉnh v bất kỳ của G và tìm đường đi ngắn nhất từ u_0 đến v.

Có một số thuật toán tìm đường đi ngắn nhất; ở đây, ta có thuật toán do E. Dijkstra, nhà toán học người Hà Lan, đề xuất năm 1959. Trong phiên bản mà ta sẽ trình bày, người ta giả sử đồ thị là vô hướng, các trọng số là dương. Chỉ cần thay đổi đôi chút là có thể giải được bài toán tìm đường đi ngắn nhất trong đồ thị có hướng.

Phương pháp của thuật toán Dijkstra là: xác định tuần tự đỉnh có khoảng cách đến u_0 từ nhỏ đến lớn.

Trước tiên, đỉnh có khoảng cách đến a nhỏ nhất chính là a, với $d(u_0,u_0)=0$. Trong các đỉnh $v \neq u_0$, tìm đỉnh có khoảng cách k_1 đến u_0 là nhỏ nhất. Đỉnh này phải là một trong các đỉnh kề với u_0. Giả sử đó là u_1. Ta có:

$$d(u_0,u_1) = k_1.$$

Trong các đỉnh $v \neq u_0$ và $v \neq u_1$, tìm đỉnh có khoảng cách k_2 đến u_0 là nhỏ nhất. Đỉnh này phải là một trong các đỉnh kề với u_0 hoặc với u_1. Giả sử đó là u_2. Ta có:

$$d(u_0,u_2) = k_2.$$

Tiếp tục như trên, cho đến bao giờ tìm được khoảng cách từ u_0 đến mọi đỉnh v của G. Nếu $V=\{u_0, u_1, ..., u_n\}$ thì:

$$0 = d(u_0,u_0) < d(u_0,u_1) < d(u_0,u_2) < ... < d(u_0,u_n).$$

4.1.2 Các bước thuật toán Dijkstra

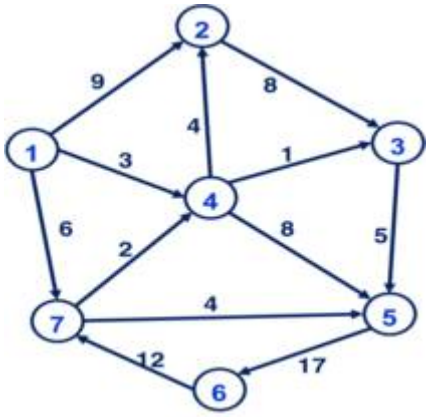
Input: N, L, s, t – số đỉnh, ma trận trọng lượng, đỉnh xuất phát, đỉnh kết thúc

Output: D, Labels – D[k]: trọng lượng ĐĐNN s→k, Labels[k]: đỉnh ngay trước k trong ĐĐNN s→k

1. V=X; D[s]=0; D[k]=∞, ∀k∈X\{s}; Labels[k]=-1, ∀k∈X.
2. Trong khi t∈V:
    1. Chọn đỉnh v∈V với D[v] nhỏ nhất;
    2. V := V\{v};
    3. Với mọi đỉnh k∈V và có cạnh nối từ v đến k,

    Nếu D[k] > D[v]+L_{vk} thì D[k]= D[v]+L_{vk} và Labels[k]=v

Ví dụ: Đồ thị G gồm 7 đỉnh, 12 cạnh như hình bên. Tìm đường đi ngắn nhất từ đỉnh 1 đến đỉnh 5



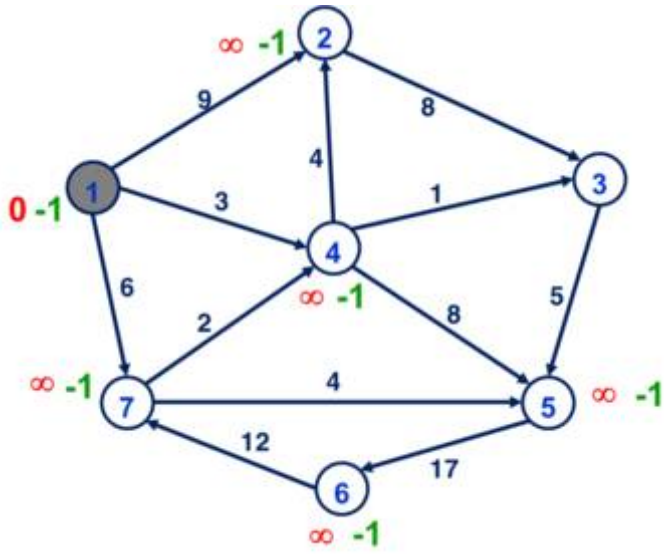
Hình 4.1 Thuật toán Dilkstra

Ma trận trọng lượng của đồ thị trên là

$$\begin{pmatrix} 0 & 9 & \infty & 3 & \infty & \infty & 6 \\ \infty & 0 & 8 & \infty & \infty & \infty & \infty \\ \infty & \infty & 0 & \infty & 5 & \infty & \infty \\ \infty & 4 & 1 & 0 & 8 & \infty & \infty \\ \infty & \infty & \infty & \infty & 0 & 17 & \infty \\ \infty & \infty & \infty & \infty & \infty & 0 & 12 \\ \infty & \infty & \infty & 2 & 4 & \infty & 0 \end{pmatrix}$$

Hình 4.2 Ma trận trọng lượng của đồ thị

Bước 0: Khởi tạo. Tập V là tập các đỉnh chưa được tô màu. D[k] là số bên trái, Labels[k]: số bên phải. Bắt đầu đi từ đỉnh 1 với D[1]=0; Labels[1]=-1.

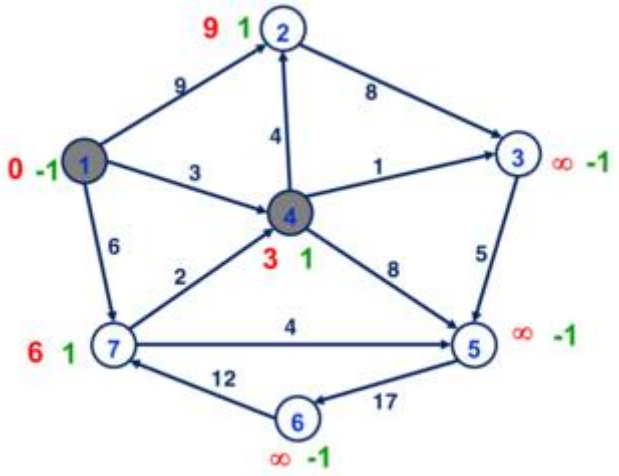


Hình 4.3 Đồ thị sau bước khởi tạo

Bước 1: Từ đỉnh 1 có thể đi đến các đỉnh chưa tô màu:

- Đỉnh 2, D[2] = ∞ > D[1] + L_{12}= 0 + 9 = 9. Do đó cập nhật lại D[2] = 9; Labels[2] = 1.
- Đỉnh 4, D[4] = ∞ > D[1] + L_{14} = 0 + 3 = 3. Cập nhật lại D[4] = 3, Labels[4] = 1.
- Đỉnh 7, D[7] = ∞ > D[1]+L_{17} = 0 + 6 = 6. Cập nhật lại D[7] = 6, Labels[7] = 1.

Do từ 1 đến các đỉnh i khác có L_{1i} = ∞, D[i] = ∞, D[1] + L_{1i} = ∞ nên không cập nhật lại. Ở bước này ta thấy D[4] có giá trị nhỏ nhất nên chọn đỉnh tiếp theo là 4. Tô màu đỉnh 4

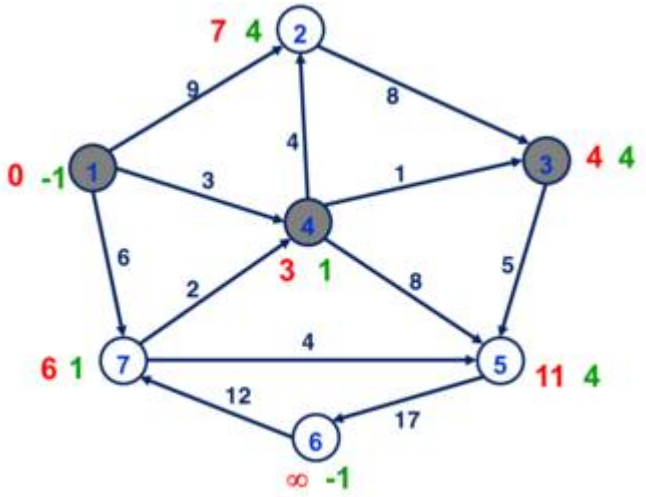


Hình 4.4 Đồ thị sau khi bước 1

Bước 2: Từ đỉnh 4 có thể đi đến các đỉnh chưa được tô màu:

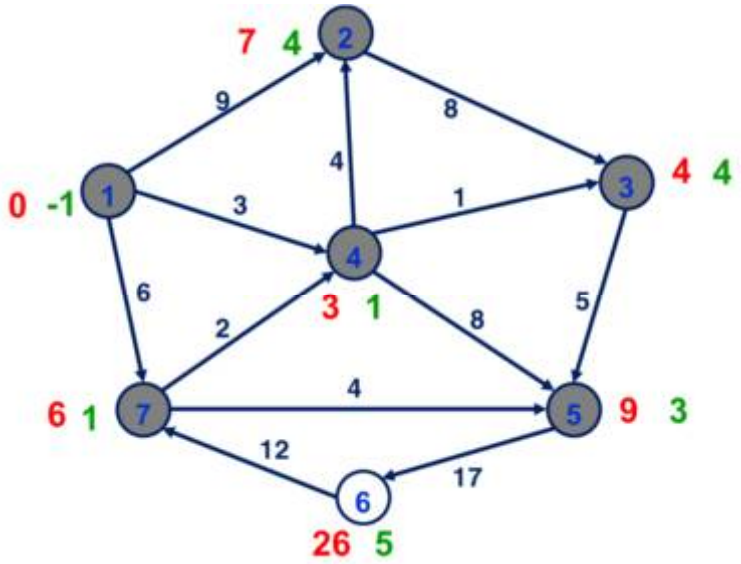
- Đỉnh 2, D[2] = 9 > D[4] + L_{42} = 3 + 4 = 7. Cập nhật lại D[2] = 7; Labels[2] = 4.
- Đỉnh 3, D[3] = ∞ > D[4] + L_{43} = 3 + 1 = 4. Cập nhật lại D[3] = 4; Labels[3] = 4.
- Đỉnh 5, D[5] = ∞ > D[4] + L45 = 3 + 8 =11. Cập nhật lại D[5] = 11; Labels[5] = 4.

Các đỉnh còn lại không cần cập nhật. D[3] = 4 là nhỏ nhất nên chọn đỉnh 3 là đỉnh kế tiếp. Tô màu đỉnh 3.



Hình 4.5 Đồ thị sau bước 2

Các bước còn lại tương tự như trên. Sau đi thực hiện xong sẽ có kết quả như sau:



Hình 4.6 Đồ thị sau khi thuật toán kết thúc

Đường đi ngắn nhất từ 1 đến 5 có trọng lượng D[5]=9: 5 ← 3 ← 4 ← 1.

Giá trị của D[i] qua các bước:

Bước	1	2	3	4	5	6	7
0	**0**	∞	∞	∞	∞	∞	∞
1		9	∞	**3**	∞	∞	6
2		7	**4**		11	∞	6
3		7			9	∞	**6**
4		**7**			9	∞	
5					**9**	∞	

Hình 4.7 Bảng biểu diễn trọng lượng qua từng bước

Giá trị của Label[i] qua các bước:

Bước	1	2	3	4	5	6	7
0	**-1**	-1	-1	-1	-1	-1	-1
1		1	-1	**1**	-1	-1	1
2		4	**4**		4	-1	1
3		4			3	-1	**1**
4		**4**			3	-1	
5					**3**	-1	

Hình 4.8 Bảng biểu diễn giá trị label qua từng bước

Kết hợp 2 bảng lại:

Bước	1	2	3	4	5	6	7
0	(0, -1)	(∞, -1)	(∞, -1)	(∞, -1)	(∞, -1)	(∞, -1)	(∞, -1)
1		(9, 1)	(∞, -1)	**(3, 1)**	(∞, -1)	(∞, -1)	(6, 1)
2		(7, 4)	**(4, 4)**		(11, 4)	(∞, -1)	(6, 1)
3		(7, 4)			(9, 3)	(∞, -1)	**(6, 1)**
4		**(7, 4)**			(9, 3)	(∞, -1)	
5					**(9, 3)**	(∞, -1)	

Hình 4.9 Bảng kết hợp trọng lượng và Label qua từng bước

THUẬT TOÁN FLOYD

Sau khi áp dụng thuật toán Floyd, ta sẽ có đường đi ngắn nhất giữa 2 đỉnh bất kì trong đồ thị.

Thuật toán như sau:

Input: N, L – *số đỉnh, ma trận trọng lượng của G(X, E)*

Output*: L, Nexts* – L[u, v]: trọng lượng ĐĐNN u→v, Nexts[u, v]: đỉnh ngay sau u trong ĐĐNN u→v

1. Nếu $L[u, v] \neq \infty$: Nexts[u, v]=v

   Ngược lại: Nexts[u, v]=-1 , $\forall (u, v) \in X^2$.

2. Với mọi $t \in X$

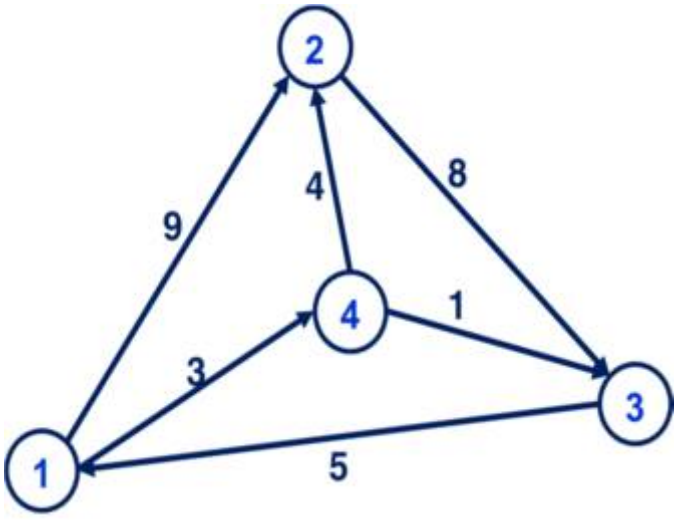
   Với mọi $u \in X$ có $L[u, t] \neq \infty$

   Với mọi $v \in X$ có $L[t, v] \neq \infty$

   Nếu L[u, v] > L[u, t] + L[t, v] thì

   1. L[u, v] = L[u, t] + L[t, v]
   2. Nexts[u, v] = Nexts[u, t]

Ví dụ: áp dụng thuật toán Floyd cho đồ thị 4 đỉnh và 6 cạnh sau



Hình 4.10 Thuật toán Floyd

Bước 0: khởi tạo bảng như hình dưới. Ô (u, v) chứa thông tin đường đi ngắn nhất từ đỉnh u đến đỉnh v, (L[u, v], Nexts[u,v]). Những ô chưa có thông tin sẽ mang giá trị (∞, -1).

	1	2	3	4
1		(9,2)		(3,4)
2			(8,3)	
3	(5,1)			
4		(4,2)	(1,3)	

Hình 4.11 Bước khởi tạo thuật toán Floyd

Bước 1: chọn đỉnh 1 làm đỉnh trung gian. Tính L[u, 1] + L[1, v], nếu tổng này bé hơn L[u, v] ở bước trên thì cập nhật lại thông tin ô (u, v): L[u, v] = L[u, 1] + L[1, v]; Nexts[u, v] = 1. Những ô có giá trị in đậm là những ô được cập nhật.

	1	2	3	4
1		(9, 2)		(3, 4)
2			(8,3)	
3	(5, 1)	**(14, 1)**		**(8, 1)**
4		(4, 2)	(1, 3)	

Hình 4.12 Bước 1 thuật toán Floyd

Bước 2: chọn đỉnh 2 làm đỉnh trung gian

	1	2	3	4
1		(9, 2)	**(17, 2)**	(3, 4)
2			(8,3)	
3	(5, 1)	(14, 1)		(8, 1)
4		(4, 2)	(1, 3)	

Hình 4.13 Bước 2 thuật toán Floyd

Bước 3: lấy đỉnh 3 làm đỉnh trung gian

	1	2	3	4
1		(9, 2)	(17, 2)	(3, 4)
2	**(13, 3)**		(8,3)	**(16, 3)**
3	(5, 1)	(14, 1)		(8, 1)
4	**(6, 3)**	(4, 2)	(1, 3)	

Hình 4.14 Bước 3 thuật toán Floyd

Bước 4: lấy đỉnh 4 làm trung gian

	1	2	3	4
1		(9, 2)	**(4,4)**	(3, 4)
2	(13, 3)		(8,3)	(16, 3)
3	(5, 1)	**(12, 4)**		(8, 1)
4	(6, 3)	(4, 2)	(1, 3)	

Hình 4.15 Bước 4 thuật toán Floyd

Giả sử cần tìm đường đi ngắn nhất từ đỉnh 2 đến đỉnh 4 thì dựa vào thông tin ô (2, 4): L[2, 4] = 16, Nexts[2, 4] = 3, tức là tổng trọng lượng nhỏ nhất là 16, trên con đường đi ngắn nhất đến 4 thì phải đi qua 3. Ta nhìn vào ô (2, 3) và (3, 4). Ô (2, 3) có L[2, 3] = 8, Nexts[2, 3] = 3, tức là đường đi ngắn nhất từ 2 đến 3 là 8 và không đi qua đỉnh trung gian nào cả. Ô (3, 4) có L[3, 4] = 8, Nexts[3, 4] = 1, nghĩa là trước khi đi từ 3 đến 4 thì phải đi qua 1. Dựa vào ô (3,1) ta có đường đi ngắn nhất là 5 và không đi qua đỉnh trung gian nào khác. Tương tự với (1, 4), đường đi ngắn nhất có trọng số là 3 và không đi qua đỉnh trung gian nào khác.

Vậy đường đi ngắn nhất từ 2 đến 4 là: 2 → 3 → 1 → 4 với tổng trọng lượng là 16.

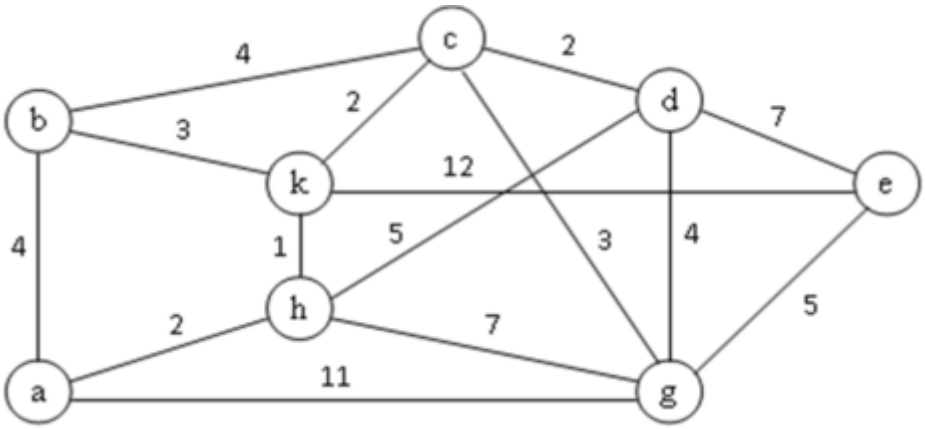
TÓM TẮT

- *Thuật toán Dijkstra dùng để xác định đường đi ngắn nhất từ một định đến tất cả các đỉnh khác trong đồ thị có trọng số không âm.*
- *Thuật toán Floyd dùng để xác định đường đi ngắn nhất giữa ha đỉnh bất kì trong đồ thị.*

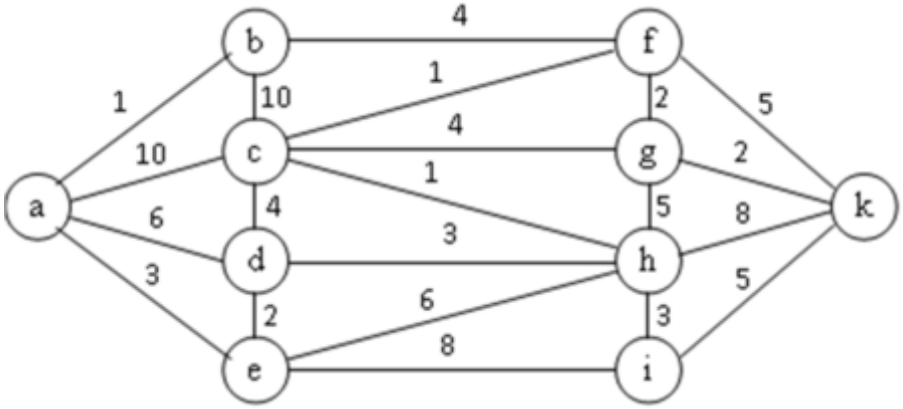
BÀI TẬP

1. Dùng thuật toán Dijkstra tìm đường đi ngắn nhất từ đỉnh a đến các đỉnh khác trong đồ thị sau:

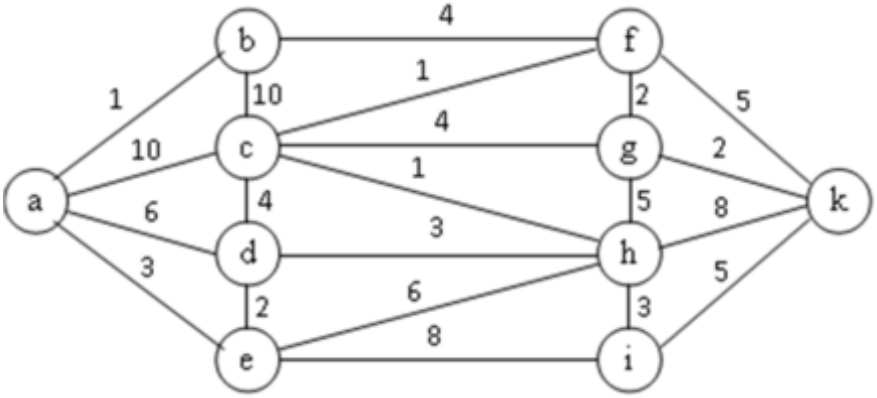
a)



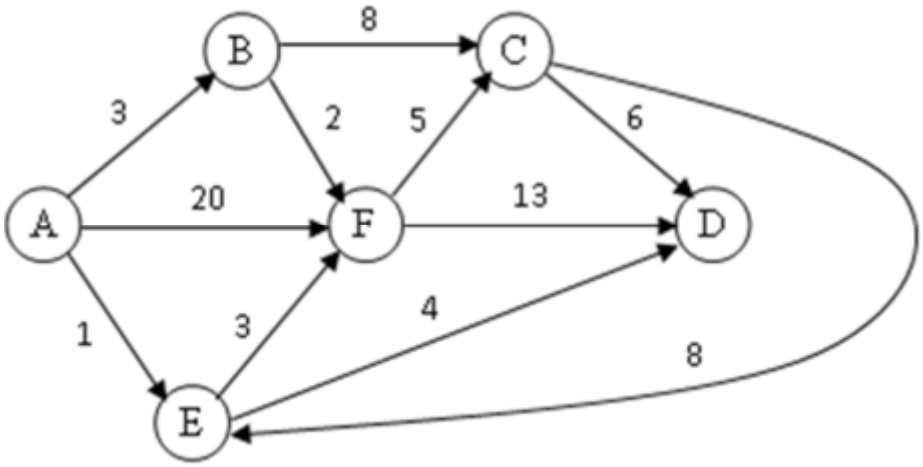
b)



c)



2. Dùng thuật toán Floyd tìm đường đi ngắn nhất giữa tất cả các đỉnh trong đồ thị sau:



BÀI 5: BÀI TOÁN GHÉP CẶP

Sau khi học xong bài này, học viên có thể:

- *Hiểu được thế nào là bài toán ghép cặp và các thuật toán sử dụng trong bài toán này.*
- *Tìm hiểu bài toán lập lịch.*

BÀI TOÁN GHÉP CẶP

5.1.1 Giới thiệu

Các bài toán ghép cặp có thể chia thành 2 loại:

- Thứ nhất: Xét việc ghép mỗi phần tử của tập hợp này với một phần tử của tập hợp khác như phân công n việc khác nhau cho n người, mỗi người một việc, đó chính là bài toán hôn nhân.
- Thứ hai: Xét việc phân chia 2k phần tử của một tập hợp thành k cặp, chẳng hạn như sắp xếp 2k sinh viên vào k phòng trong ký túc xá (mỗi phòng có 2 sinh viên).

5.1.2 Định lý Hall

Bài toán hôn nhân do Philip Hall giải quyết năm 1935 có rất nhiều ứng dụng và được phát biểu dưới nhiều dạng khác nhau.

Ví dụ 1. Có 5 việc cần tuyển người đảm nhiệm, mỗi người một việc. Gọi S_i là tập hợp các ứng viên thích hợp cho việc thứ i và giả sử rằng ta có

$S_1=\{A, B, C\}$, $S_2=\{D, E\}$, $S_3=\{D\}$, $S_4=\{E\}$, $S_5=\{A, E\}$

Có thể tuyển đủ 5 người thích hợp cho 5 việc nói trên không?

Câu trả lời là không vì 3 việc thứ 2, thứ 3, thứ 4 chỉ có 2 ứng viên.

Ví dụ 2. Có 4 chàng trai B_1, B_2, B_3, B_4 và 5 cô gái G_1, G_2, G_3, G_4, G_5; mỗi chàng có một danh sách các cô gái thích hợp như sau:

B_1: {G_1, G_4, G_5} B2: {G_1}

B3: {G_2, G_3, G_4} B4: {G_2, G_4}

Mỗi chàng trai có thể kết hôn với một cô gái thích hợp hay không?

Ta dễ dàng thấy rằng một lời giải cho bài toán này là các cặp sau:

(B_1, G_4), (B_2, G_1), (B_3, G_3), (B_4, G_2)

Hai bài toán trong 2 ví dụ trên có cùng một dạng. Vậy khi nào những bài toán như thế có lời giải? Đó là nội dung của **Định lý Hall**

5.1.2.1 Định nghĩa

Cho S là một tập hợp hữu hạn và đa tập hợp A={A_1, …, A_m} là một họ các tập con của S.

Ta bảo A thỏa điều kiện Hall nếu với mọi k ∈{1, …, m} và k phần tử bất kỳ $A_{i_1}, \dots, A_{i_k} \in A$, ta có

$$\left|\boldsymbol{A_{i_1}} \cup \boldsymbol{A_{i_2}} \cup \dots \cup \boldsymbol{A_{i_k}}\right| \geq \boldsymbol{k}$$

Một hệ đại diện riêng biệt hay một SDR của A là một bộ (a_1, … , a_m) gồm m phần tử khác nhau của S sao cho ai ∈ A_i; mỗi ai gọi là một đại diện của A_i.

Ví dụ 3. Cho A = {A_1, A_2, A_3, A_4} với A_1={1, 2}, A_2={2, 3, 4}, A_3={1}, A_4={2} thì vì $|A_1 \cup A_2 \cup A_3|$ = 2 nên A không thỏa điều kiện Hall.

Hiển nhiên, nếu A có một SDR thì A thỏa điều kiện Hall. Điều thú vị là đảo lại cũng đúng.

5.1.2.2 Định lý

Cho S là một tập hợp hữu hạn và đa tập hợp A={A_1, …, A_m} là một họ các tập con của S. A có một SDR nếu và chỉ nếu A thỏa điều kiện Hall.

Thuật toán tìm SDR

Chọn một phần tử a_1 tùy ý của A_1 làm đại diện cho A_1.

Với i>=2 và $A_i - \{a_1, \dots, a_{i-1}\} \neq \emptyset$, ta chọn một phần tử ai bất kỳ của $A_i - \{a_1, \dots, a_{i-1}\}$

Nếu ta chọn được $a_1, \dots, a_m$ thì chúng tạo thành một SDR của A

Ngược lại, tức là có k sao cho $A_k - \{a_1, \dots, a_{k-1}\} = \emptyset$ hay $A_k \subseteq \{a_1, \dots, a_{k-1}\}$ nghĩa là mỗi phần tử của A_k đã là đại diện. Gọi S(b) là phần tử của A có đại diện là b.

Đặt $B_1=A_k$ và sắp các phần tử của B_1 thành 1 dãy $U_1=b_1b_2\dots b_h$. Nếu $S(b_1) - B_1=\emptyset$ thì đặt $U_2=U_1$; nếu $S(b_1) - B_1=\{b_{h+1}\dots b_p\}$, ta lập dãy

$U_2=U_1\ b_{h+1}\dots b_p=b_1b_2\dots b_p$

Giả sử đã có U_i và B_i là tập hợp các phần tử xuất hiện trong U_i. Nếu $S(b_i)-B_i=\emptyset$ thì đặt $U_{i+1}= U_i$; nếu ngược lại $S(b_i) - B_i = \{b_{i_1}, \dots, b_{i_t}\}$, thì ta lập dãy

$U_{i+1}= U_i b_{i_1}, \dots, b_{i_t}$

Nếu cuối cùng ta có dãy U_q với các phần tử $b_1, \dots, b_r$ đều đã là đại diện của $S(b_i)$, i=1, …, r thì

$$|A_k \cup S(b_1) \cup \dots \cup S(b_r)| = r < r + 1,$$

điều này trái giả thiết là A thỏa điều kiện Hall. Vậy có r mà trong U_r có một phần tử b_s chưa là đại diện. Theo định nghĩa của U_1 thì b_s không có trong U_1 và có $s_1 < s$ sao cho $b_s \in S(b_{s_1})$ và $b_s \notin U_{s_1-1}$ (ta biết $b \in U_i$ nếu b là một thành phần của U_i). Nếu $b_{s_1} \notin U_1$ thì có $s_2 < s_1$ sao cho $b_{s_1} \in S(b_{s_2})$… Tiếp tục quá trình trên, ta có dãy $s_t, \dots, s_1$ với $s_t < \dots < s_2 < s_1 < s$ sao cho

$b_s \in S(b_{s_1}),\ b_{s_1} \in S(b_{s_2}), \dots, b_{t-1} \in S(b_{s_t}),\ b_{s_t} \in A_k$

Ta chọn b_s làm đại diện cho $S(b_{s_1})$, b_{s_1} làm đại diện cho $S(b_{s_2})$, …, và b_{t-1} làm đại diện cho $S(b_{s_t})$, cuối cùng b_{s_t} làm đại diện cho A_k.

Lặp lại quá trình trên ta có một SDR của A.

Ví dụ 4. Cho $A = \{A_1,\dots,A_7\}$ với $A_1=\{1,2,3\}$, $A_2=\{2,4\}$, $A_3=\{2,5\}$, $A_4=\{1,3\}$, $A_5=\{1,2,4\}$, $A_6=\{2,3,4\}$, $A_7=\{3,6,7\}$. Tìm SDR của A.

Chọn $1 \in A_1$, $2 \in A_2$, $5 \in A_3$, $3 \in A_4$, $4 \in A_5$. Các phần tử của A_6 đều đã là đại diện. Ta lập dãy U_1, U_2, … như sau:

$U_1 = 234$; $S(2) = A_2 = \{2,4\}$

U_2 = 234 ; S(3) = A_4 = {1,3}

U_3 = 2341 ; S(4) = A_5 = {1,2,4}

U_4 = 2341 ; S(1) = A_1 = {1,2,3}

Ta có $|A_6 \cup A_2 \cup A_4 \cup A_5 \cup A_1| = 4$ ➔ A không có SDR.

Ví dụ 5. Cho A = {A_1 ,…, A_9} với A_1={a,b}, A_2={b,d}, A_3={b,c}, A_4={a,c}, A_5={e}, A_6={f,g}, A_7={f,g,h}, A_8={e,h,i}, A_9={e,f}. Tìm SDR của A.

Chọn a$\in A_1$, b$\in A_2$, c$\in A_3$. Các phần tử của A_4 ={a,c} đều đã là đại diện. Ta lập dãy U_1, U_2, … như sau:

U_1 = ac ; S(a) = A_1 = {a,b}

U_2 = acb ; S(c) = A_3 = {b,c}

U_3 = acb ; S(b) = A_2 = {b,d}

d$\in$S(b) và d chưa là đại diện, b$\in$S(a)=A_1. Chọn d làm đại diện cho S(b)= A_2, b làm đại diện cho S(a)= A_1 và a làm đại diện cho A_4.

Chọn e$\in A_5$, f$\in A_6$, g$\in A_7$, h$\in A_8$. Các phần tử của A_9 ={e,f} đều đã là đại diện. Ta lập dãy V_1, V_2, … như sau:

V_1 = ef ; S(e) = A_5 = {e}

V_2 = ef ; S(f) = A_6 = {f,g}

V_3 = efg ; S(g) = A_7 = {f,g,h}

V_4 = efgh ; S(h) = A_8 = {e,h,i}

Trong S(h) thì I chưa là đại diện và

i$\in$S(h), h$\in$S(g), g$\in$S(f), f$\in A_9$

Chọn i$\in$S(h) = A_8, h$\in$S(g) = A_7, g$\in$S(f) = A_6, và f$\in A_9$. Vậy, **một SDR của A là {b, c, d, a, e, g, h, i, f}.**

5.1.2.3 Định lý hôn nhân

Mỗi chàng trai a_i, i=1, …, m có một danh sách A_i các cô gái thích hợp với mình. Mỗi chàng trai có thể kết hôn với một cô gái thích hợp nếu và chỉ nếu A = {A_1, …, A_m} thỏa điều kiện Hall.

BÀI TOÁN GIAO VIỆC TỐI ƯU

Có n công việc, n ứng viên và ta muốn phân công mỗi người một việc, khả năng của mỗi ứng viên đối với mỗi công việc được xác định bằng điểm thể hiện trên một bảng chữ nhật, điểm càng cao khả năng càng kém. Tìm phương án phân công tốt nhất, tức là tìm n số ở trên n dòng, n cột khác nhau sao cho tổng của chúng bé nhất.

Nhận xét rằng bài toán sẽ không đổi nếu mỗi dòng hay mỗi cột đều trừ đi một hằng số.

Ví dụ 6. Giả sử ta có 4 việc x, y, z, t và 4 ứng viên A, B, C, D mà ta muốn phân công mỗi người một việc. Khả năng của mỗi ứng viên tương ứng với mỗi công việc cho trong bảng sau đây, trong đó điểm càng cao khả năng càng thấp.

	A	B	C	D
x	5	7	15	12
y	8	3	9	10
z	4	14	2	5
t	6	3	1	14

Hình 5.1 Bảng phân công công việc

Trừ mỗi dòng cho số nhỏ nhất của dòng đó

	A	B	C	D
x	0	2	10	7
y	5	0	6	7
z	2	12	0	3
t	5	2	0	13

Hình 5.2 Bảng phân công công việc sau khi trừ mỗi dòng

Trừ mỗi cột cho số nhỏ nhất của cột đó

	A	B	C	D
x	0	2	10	4
y	5	0	6	4
z	2	12	0	0
t	5	2	0	10

Hình 5.3 Bảng phân công công việc sau khi trừ mỗi cột

Bốn số ở trên 4 dòng, 4 cột khác nhau ở trên bảng cuối cùng có tổng nhỏ nhất là 0, tương ứng với phương án phân công (A, x), (B, y), (C, t), (D, z) với tổng số điểm thật sự là 5+3+1+5=14.

Nhưng có phải lúc nào ta cũng chọn được các số 0 ở trên những dòng và những cột khác nhau ?

Thuật toán tìm phương án phân công tốt nhất

1. Trừ mỗi dòng cho số nhỏ nhất của dòng đó.
2. Trừ mỗi cột cho số nhỏ nhất của cột đó.
3. Xác định k, số đường nhỏ nhất chứa tất cả các zero của A và chỉ rõ k đường này.

   Nếu k<n thì đến Bước 4. Nếu k=n thì đến bước 5.
4. Gọi a là số nhỏ nhất không xuất hiện trong các đường xác định ở B3

- Trừ a cho tất cả các số không ở trên đường nào
- Cộng a cho tất cả các số ở trên 2 đường (giao điểm của dòng và cột)
- Giữ nguyên các số chỉ ở trên một đường

Trở lại B3

5. Đặt Ai = {j: A_{ij}=0}. Dùng thuật toán Hall để tìm ra SDR của A={A_1, …, A_n}, tức là tìm n zero độc lập (Các số 0 ở trên những dòng và những cột khác nhau của ma trận A được gọi là các *zero độc lập*). Suy ra kết quả. Dừng.

Ví dụ: Xét bài toán giao việc sau đây

	A	B	C	D
x	6	8	2	7
y	5	8	13	9
z	2	7	8	9
t	4	11	7	10

Hình 5.4 Bảng phân công công việc

Thực hiện theo các bước ở trên, ta có kết quả như sau:

Bước 1:

	A	B	C	D
x	4	6	0	5
y	0	3	8	4
z	0	5	6	7
t	0	7	3	6

Hình 5.5 Kết quả sau khi áp dụng bước 1

Bước 2:

	A	B	C	D
x	4	3	0	1
y	0	0	8	0
z	0	2	6	3
t	0	4	3	2

Hình 5.6 Kết quả sau khi áp dụng bước 2

Bước 3: k = 3

	A	B	C	D
x	4	3	0	1
y	0	0	8	0
z	0	2	6	3
t	0	4	3	2

Hình 5.7 Kết quả sau khi áp dụng bước 3

Bước 4: a = 2

	A	B	C	D
x	6	3	0	1
y	2	0	8	0
z	0	0	4	1
t	0	2	1	0

Hình 5.8 Kết quả sau khi áp dụng bước 4

Lặp lại bước 3 với k=4

	A	B	C	D
x	6	3	0	1
y	2	0	8	0
z	0	0	4	1
t	0	2	1	0

Hình 5.9 Kết quả sau khi lặp lại bước 3

Bước 5:

	A	B	C	D
x	6	3	0*	1
y	2	0*	8	0
z	0*	0	4	1
t	0	2	1	0*

Hình 5.10 Kết quả sau bước 5

Phương án tối ưu là

(A, z), (B, y), (C, x), (D, t)

với tổng số điểm là: 2+8+2+10=22

Một phương án tối ưu khác là:

(A, t), (B, z), (C, x), (D, y)

BÀI TOÁN GIAO VIỆC CỦA GALE

5.1.3 Giới thiệu

Một loại bài toán giao việc khác là xét tầm quan trọng của công việc. Việc nào quan trọng hơn sẽ được ưu tiên nhận người.

Ta sẽ viết $J_1<J_2$ nếu công việc J_1 quan trọng hơn (ưu tiên hơn) J_2.

5.1.4 Định nghĩa

Cho $A = \{a_1, \dots, a_n\}$ là tập hợp các công việc $a_1< \dots <a_n$. Ta bảo A phân công được nếu có thể phân công những người khác nhau phụ trách tất cả những công việc khác nhau trong A.

Cho $A = \{a_1, \dots, a_n\}$ phân công được. A được gọi là tập hợp phân công được tối ưu hay OAS nếu với mọi tập hợp phân công được $B=\{b_1, \dots, b_n\}$, ta có $m\leq n$ và $a_i\leq b_i$, $i=1,\dots,m$.

Ví dụ Cho $J_1<J_2<J_3<J_4<J_5$ và danh sách ứng viên cho các công việc này như sau

Công việc	Ứng viên
J_1	A, B
J_2	B, C
J_3	B
J_4	A, C
J_5	B, C,D

Hình 5.11 Bảng công việc và các ứng viên

Có thể kiểm chứng các tập hợp $\{J_2, J_3, J_4\}$, $\{J_1, J_4, J_5\}$, $\{J_1, J_2, J_3, J_5\}$ là phân công được, trong đó tập hợp $A=\{J_1, J_2, J_3, J_5\}$ là tối ưu.

Làm thế nào để xác định một tập hợp phân công được tối ưu?

Thuật toán tìm tập hợp phân công được tối ưu

Thuật toán này chính là thuật toán tìm một hệ đại diện riêng biệt cho $A=\{J_1,\ldots,J_s\}$ với $J_1< \ldots <J_s$. Chọn đại diện theo thứ tự ưu tiên. Nếu đến J_r mà không tìm được đại diện cho J_r sau khi đã đổi việc thì bỏ qua công việc này (chưa có người phụ trách) để xét đến J_{r+1}.

Ví dụ Tìm OAS cho các công việc trong bảng trong ví dụ trên.

Chọn $A \in J_1$, $B \in J_2$ thì không thể chọn đại diện cho $J_3=\{B\}$.

Đặt $U_1=B$, ta có $S(B)=J_2=\{B,C\}$ nên $U_2=BC$ và C chưa là đại diện.

Vì $C \in S(B)$, $B \in U_1$ nên ta thay đổi đại diện như sau: C là đại diện cho J_2 và B là đại diện cho J_3.

Hình 5.12 Ứng viên tương ứng cho các công việc

Tất cả các phần tử của J_4 đều đã là đại diện.

Đặt $U_1=AC$, ta có $S(A)=J_1=\{A,B\}$ nên $U_2=ACB$, $S(C)=J_2=\{B,C\}$ → $U_3=ACB$, $S(B)=J_3=\{B\}$ → $U_4=ACB$ → Không có đại diện cho J_4 nên ta xét đến J_5 (loại J_4 ra khỏi A) và chọn D làm đại diện cho J_5. Vậy ta có OAS là $\{J_1, J_2, J_3, J_5\}$ với các phân công sau đây: $A \rightarrow J_1$, $C \rightarrow J_2$, $B \rightarrow J_3$, $D \rightarrow J_5$.

TÓM TẮT

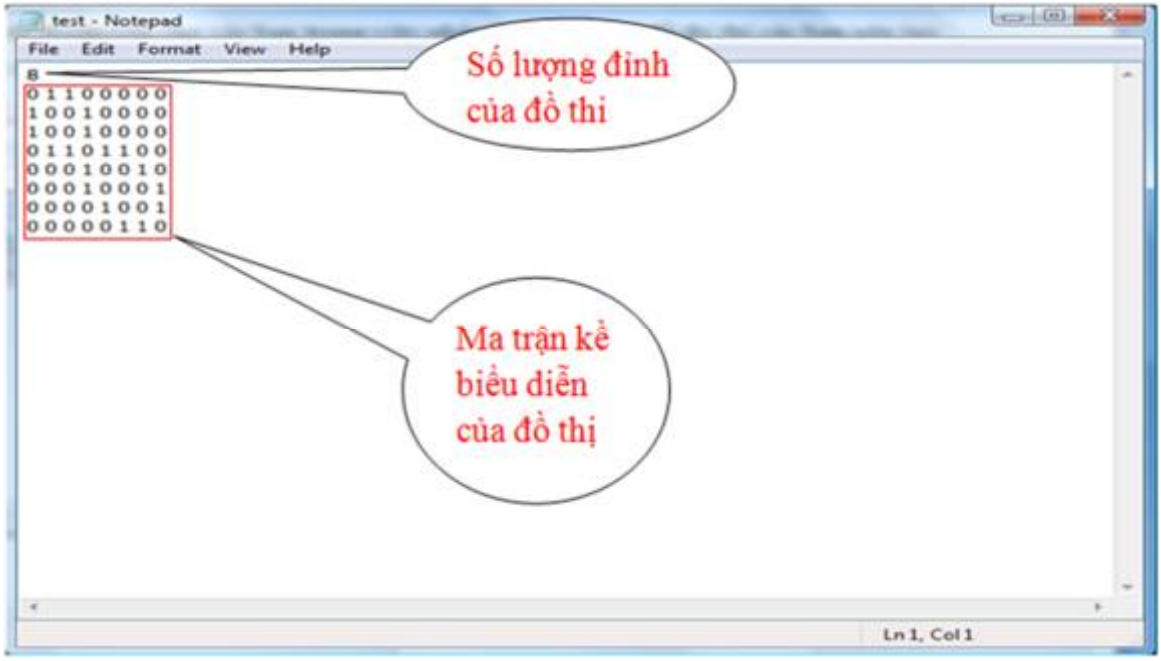
- *Bài toán hôn nhân do Philip Hall giải quyết năm 1935 có rất nhiều ứng dụng và được phát biểu dưới nhiều dạng khác nhau.*
- *Cho S là một tập hợp hữu hạn và đa tập hợp $A=\{A_1, \ldots, A_m\}$ là một họ các tập con của S. A có một SDR nếu và chỉ nếu A thỏa điều kiện Hall.*
- *Cho S là một tập hợp hữu hạn và đa tập hợp $A=\{A_1, \ldots, A_m\}$ là một họ các tập con của S. A có một SDR nếu và chỉ nếu A thỏa điều kiện Hall.*
- *Mỗi chàng trai a_i, $i=1, \ldots, m$ có một danh sách A_i các cô gái thích hợp với mình. Mỗi chàng trai có thể kết hôn với một cô gái thích hợp nếu và chỉ nếu $A = \{A_1, \ldots, A_m\}$ thỏa điều kiện Hall.*
- *Bài toán giao việc tối ưu của Gale.*

PHỤ LỤC

BÀI TẬP 1: NHẬP XUẤT MA TRẬN KỀ TỪ FILE

Hãy viết chương trình để:

- Đọc thông tin của đồ thị gồm số đỉnh n, ma trận kề từ một file nào đó, chẳng hạn như C:/test.txt và file đó có cấu trúc như sau:



- Xuất thông tin của đồ thị ra màn hình:
  - Dòng đầu tiên là số đỉnh n của đồ thị.
  - Những dòng tiếp theo (n dòng) là thông tin ma trận kề của đồ thị.
- Kiểm tra đồ thị được đọc vào từ file đó có hợp lệ không?
- Nếu thông tin đồ thị đó là hợp lệ, thì bạn hãy cho biết đồ thị đó là có hướng hay vô hướng?

HƯỚNG DẪN

Đầu tiên: tạo một cấu trúc lưu trữ đồ thị như sau:

```
#define MAX 10 // định nghĩa giá trị MAX

#define inputfile "C:/test.txt" // định nghĩa đường dẫn tuyệt đối đến file chứa thông tin của đồ thị
```

```
typedef struct GRAPH {
  int n; // số đỉnh của đồ thị
  int a[MAX][MAX]; // ma trận kề của đồ thị
}DOTHI;
```

Giải thích: struct dùng để khai một cấu trúc DOTHI (đồ thị) gồm có số đỉnh của đồ thị là n, và ma trận kề là mảng 2 chiều a. Cách sử dụng cấu trúc struct này hoàn toàn tương tự cách sử dụng đối với kiểu dữ liệu thông thường cơ bản như int a, char c, …..

Ví dụ: bạn muốn tạo một biến đồ thị thì bạn làm như sau:

```
DOTHI g;
```

Còn bạn muốn gán giá trị số đỉnh của đồ thị vào g (chẳng hạn đồ thị có 8 đỉnh) thì bạn làm như sau:

g.n = 8; // giống như bạn sử dụng một biến int, char, float thông thường nhưng chỉ khác chỗ là có thêm ở đằng trước **g.**

Điều này có nghĩa là nếu bạn muốn truy cập vào thành phần nào của đồ thị thì bạn dùng <tên_biến_thuộc_kiểu_đồ_thị> và dấu chấm.

Ví dụ: **g.**n //truy cập vào giá trị n của biến g

g.a[i][j] //truy cập vào giá trị hàng i cột j của mảng a của biến g.

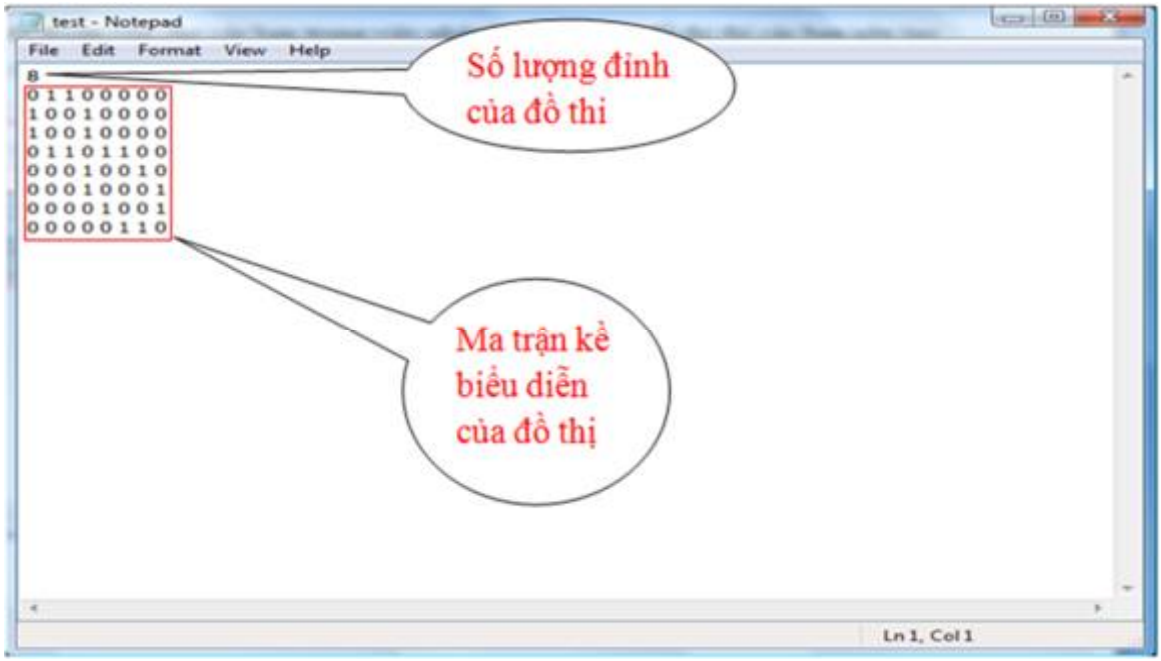
Còn nếu bạn muốn sử dụng các giá trị n (số đỉnh của đồ thị), a[i][j] (giá trị ma trận kề) thì bạn chỉ việc dùng <tên_biến_thuộc_kiểu_đồ_thị> và dấu chấm như trên

Ví dụ: lấy số đỉnh của đồ thị g gán vào biến int h và giá trị đầu tiên (chỉ số i = 0, j = 0) trong ma trận kề của đồ thị g vào biến int k thì làm như sau:

```
h = g.n;
k = g.a[0][0];
```

Do đó, để thuận tiện cho việc nhập thông tin của đồ thị thì các bạn nên tạo một cái file có cấu trúc như sau lưu trữ thông tin của đồ thị bạn cần test.

Bước 1: bạn mở NOTEPAD lên tạo một file có cấu trúc tổ chức như sau:



Sau đó bạn lưu lại với tên file bất kì gì đó chẳng hạn như bạn lưu với tên test.txt ở ổ **C**.

Bước 2: Lấy/đọc những thông tin đồ thị từ file bạn đã lưu ở trên (ví dụ C:/test.txt) gán vào một biến DOTHI g. Việc đó được làm thông qua hàm DocMaTranKe như sau:

```
int DocMaTranKe(char TenFile[100], DOTHI &g)
{
  FILE* f; // một biến FILE
  f = fopen(TenFile, "rt"); // mở một file có đường dẫn lưu ở biến TenFile
  if (f == NULL) // nếu file mở được thì biến f != NULL và file không mở được thì biến f = NULL
  {
      printf("Khong mo duoc file\n");
      return 0; // không đọc được file trả về kết quả 0
  }
  // Đọc giá trị đỉnh của đồ thị vào biến n của cấu trúc DOTHI g
  fscanf(f, "%d", &g.n);
```

```
    // Đọc giá trị của ma trận a từ file vào (dùng 2 vòng for, dòng trước, cột sau để đọc từng phần tử của ma trận)
    // với a[i][j] là giá trị ma trận tại dòng i, cột j
    int i, j;
    for (i=0; i<g.n; i++)
    {
        for (j=0; j<g.n; j++)
        {
            fscanf(f, "%d", &g.a[i][j]); // đọc từng giá trị và gán vào ma trận kề a
        }
    }
    // đóng file đã mở ở trên.
    fclose(f);
  return 1; // trả về kết quả 1 cho biết đã đọc file và sử lý nhập thông tin đồ thị xong
  }
```

Tuy nhiên, để xem kết quả đọc thông tin đồ thị từ file có đúng hay không? bạn viết hàm XuatMaTranKe như sau để xuất thông tin của đồ thị g hiện tại:

```
  void XuatMaTranKe (DOTHI g)
  {
    printf("So dinh cua do thi la %d\n", g.n);
    printf("Ma tran ke cua do thi la\n");
    for (int i = 0; i < g.n; i++)
    {
        printf ("\t");
```

```
        for (int j = 0; j < g.n; j++)
        {
            printf("%d ",g.a[i][j]);
        }
        printf("\n");
    }
}
```

Bước 3: Kiểm tra tính hợp lệ của ma trận kề nhập vào: nó có phải là biểu diễn của một trong 2 dạng đồ thị mà giáo viên hướng dẫn đã trình bày cho các bạn không? Thì các bạn viết hàm **KiemTraMaTranHopLe** như sau:

```
int KiemTraMaTranKeHopLe(DOTHI g)
{
  int i;
  for (i=0; i<g.n; i++)
  {
if (g.a[i][i] != 0) /* kiểm tra nếu tồn tại một giá trị trên đường chéo khác 0 nghĩa là a[i][j] != 0 Thì trả về giá trị 0 (ma trận kề không hợp lệ) */
            return 0;
  }
  return 1; // trả về 1 nếu tất cả các giá trị trên đường chéo là 0
}
```

Bước 4: kiểm tra đồ thị bạn nhập vào có là **vô hướng hay là có hướng**, bạn dùng hàm sau KiemTraDoThiVoHuong sau.

```
int KiemTraDoThiVoHuong(DOTHI g)
{
  int i, j;
```

```
    for (i=0; i<g.n; i++)
    {
        for (j=0; j<g.n; j++)
        {
            if (g.a[i][j] != g.a[j][i]) /* kiểm tra nếu tồn tại một giá trị a[i][j] != a[j][i] thì tức ma trận kề không đối xứng, lúc đó đồ thị không phải là vô hướng. trả về kết quả 0 (đồ thị không phải là vô hướng) */
                return 0;
        }
    }
    return 1; // trả về kết quả 1 nếu đồ thị là vô hướng
}
```

Bước 5: Code trong hàm main để gọi hàm các hàm tương ứng và chạy. Có thể làm như sau:

```
void main()
{
  DOTHI g;
  clrscr();
  if (DocMaTranKe(inputfile, g) == 1)
  {
      printf("Da lay thong tin do thi tu file thanh cong.\n\n");
      XuatMaTranKe(g);
      printf("Bam 1 phim bat ki de tien hanh kiem tra do thi ...\n\n");
      getch();
      if (KiemTraMaTranKeHopLe(g) == 1)
          printf ("Do thi hop le.\n");
```

```
        else
            printf ("Do thi khong hop le.\n");

        if (KiemTraDoThiVoHuong(g) == 1)
            printf ("Do thi vo huong.\n");
        else
            printf ("Do thi co huong.\n");
    }
    getch();
}
```

BÀI TẬP 2: ĐỒ THỊ LIÊN THÔNG

Bạn hãy viết chương trình để xác định số thành phần liên thông của một đồ thị vô hướng, và cho biết với mỗi thành phần liên thông bao gồm những đỉnh nào của đồ thị.

HƯỚNG DẪN

Chú ý: bạn **đã làm được việc đọc thông tin** của đồ thị từ một file nào đó vào chương trình của bạn rồi.

Bước 1: xét tính liên thông của đồ thị, viết một hàm DiTimCacDinhLienThong như sau để đi tìm các đỉnh j liên thông với đỉnh i trong đồ thị và gán nhãn cho những đỉnh liên thông j đó bằng nhãn của đỉnh i.

```
void DiTimCacDinhLienThong (DOTHI g, int nhan[MAX], int i)
{
  for (int j = 0; j < g.n; j++)
  {
      if (g.a[i][j] != 0 && nhan[j] != nhan[i]) // nếu tồn tại cạnh giữa đỉnh i và đỉnh j, đồng thời nhãn của đỉnh j khác với nhãn của đỉnh i (nhãn thành phần liên thông) thì thực hiện gán nhãn của đỉnh j = nhãn của đỉnh i và DiTimCacDinhLienThong với đỉnh j
      {
          nhan[j] = nhan[i]; // gán nhãn cho đỉnh j
      DiTimCacDinhLienThong (g, nhan, j); // tiếp tục DiTimCacDinhLienThong với đỉnh j và gán nhãn tương ứng tiếp.
      }
  }
}
```

Bước 2: Sau đó viết hàm XetLienThong như sau:

```
void XetLienThong(DOTHI g)
```

```
{
    int Nhan[MAX]; // tạo một mảng Nhãn để lưu lại nhãn của các đỉnh trong đồ thị g
    int i;
    for (i=0;i<g.n;i++) // gán nhãn ban đầu cho tất cả các đỉnh của đồ thị g là 0
        Nhan[i] =0;
    int SoThanhPhanLT = 0; // lưu lại số thành phần liên thông trong đồ thị g, ban đầu là 0. Tức chưa có thành phần nào.
    // duyệt lần lượt tất cả các đỉnh và chọn đỉnh có nhãn là 0. Ta bắt đầu xét
    for (i=0; i<g.n; i++)
    {
        if (Nhan[i] == 0) // có một đỉnh trong đồ thị có nhãn là 0
        {
            SoThanhPhanLT ++; // tăng số thành phần liên thông lên
            Nhan[i] = SoThanhPhanLT; // gán nhãn cho đỉnh đó bởi SoThanhPhanLT
            DiTimCacDinhLienThong(g, Nhan, i); // gọi hàm đi tìm các đỉnh liên thông với đỉnh i và gán nhãn cho nó. Hàm này được khai báo ở sau.
        }
    }
    printf ("So thanh phan lien thong la %d\n",SoThanhPhanLT);
    /*Tới đây là coi như bạn đã hoàn tất quá trình xét tính liên thông của đồ thị rồi đó. Vấn đề còn lại là bạn chỉ cần code để hiển thị ra những thành phần liên thông của nó ra. Gợi ý dựa vào mảng Nhãn */
  for (i = 1; i <= SoThanhPhanLT; i++)
    {
        printf("Thanh phan lien thong thu %d gom cac dinh ", i);
```

```
        /* các bạn code phần này để xử lý hiển thị ra các đỉnh trong thành phần liên thông thứ i*/
        printf("\n");
    }
}
```

Bước 3: Code trong hàm main để gọi hàm các hàm tương ứng và chạy. Có thể làm như sau:

```
void main()
{
    DOTHI g;
    clrscr();
    if (DocMaTranKe("C:\\test2.txt",g) == 1)
    {
        printf("Da lay thong tin do thi tu file thanh cong.\n\n");
        XuatMaTranKe(g);
        printf("Bam 1 phim bat ki de bat dau xet tinh lien thong cua do thi ...\n\n");
        getch();
        XetLienThong(g);
    }
    getch();
}
```

BÀI TẬP 3: CHU TRÌNH EULER

Bạn hãy viết chương trình để kiểm tra xem đồ thị của bạn có chu trình euler hay không? nếu có thì hãy xuất ra chu trình euler đó?

HƯỚNG DẪN

Bước 1: Tạo một cấu trúc Stack như sau để lưu lại các đỉnh trong chu trình euler:

```
struct STACK
{
  int array[100]; // lưu lại thứ tự các đỉnh trong chu trình euler có tối đa 100 đỉnh
  int size; // số lượng các đỉnh trong chu trình euler.
};
```

Để khởi tạo một stack rỗng, ta tạo một hàm khoitaoStack như sau:

```
void khoitaoStack (STACK &stack)
{
  stack.size = 0; // khởi tạo stack thì kích thước của stack bằng 0
}
```

Để đẩy một giá trị value vào stack, ta gọi hàm DayGiaTriVaoStack như sau:

```
void DayGiaTriVaoStack (STACK &stack, int value)
{
  if(stack.size + 1 >= 100) // nếu stack đã đầy thì không đẩy giá trị đó vào được vì kích thước của stack chỉ có chứa tối đa 100 phần tử.
        return; //thoát không thực hiện đẩy giá trị vào stack nữa
  stack.array[stack.size] = value; // đẩy giá trị value vào stack
  stack.size++; // tăng kích thước stack lên
}
```

Bước 2: Viết hàm **tìm đường đi** từ một đỉnh i đối với đồ thị g theo dạng đệ qui.

```
void TimDuongDi (int i, DOTHI &g, STACK & stack)
{
  for (int j = 0; j < g.n; j++)
  {
      if (g.a[i][j] != 0) // vì đồ thị vô hướng nên đối xứng do đó chỉ cần kiểm tra g.a[i][j]!=0 thôi, không cần kiểm tra g.a[j][i] != 0
      {
          g.a[i][j] = g.a[j][i] =0; // loại bỏ cạnh nối đỉnh i tới đỉnh j khỏi đồ thị
          TimDuongDi(j,g,stack); // gọi đệ quy tìm đường đi tại đỉnh j
      }
  }
  DayGiaTriVaoStack(stack,i); // đẩy đỉnh i vào trong stack
}
```

Bước 3: Để kiểm tra đồ thị g, có chu trình euler không thì ta viết hàm kiểm tra chu trình Euler như sau:

```
int KiemTraChuTrinhEuler (DOTHI g)
{
  int i,j;
  int x = 0; // x là giá trị đỉnh bắt đầu xét chu trình euler, điều kiện x là đỉnh phải có bậc > 0
  /* bạn phải code chỗ này để tìm 1 đỉnh x bắt đầu tìm chu trình euler, đỉnh x này phải có bậc > 0*/
```

```
DOTHI temp = g; // tạo ra một bản copy của đồ thị để thực hiện thuật toán, vì trong quá trình thi hành thuật toán ta có xóa cạnh nên ta tạo ra bảng copy này để thực hiện việc xóa đó mà không ảnh hưởng đến đồ thị gốc (ban đầu).

STACK stack; // tạo một stack như thuật toán đã trình bày

khoitaoStack (stack); // ban đầu stack chưa có gì nên phải khởi tạo nó về 0.

TimDuongDi(x,temp, stack); // bắt đầu tìm chu trình euler từ đỉnh x trong đồ thị temp, và thứ tự các đỉnh trong chu trình euler được lưu vào stack này.

/* Bạn cần phải kiểm tra xem hàm TimDuongDi có tìm thấy chu trình euler trong đồ thị temp không? Bạn kiểm tra bằng cách nào? Vì đối với thuật toán của mình thì có xóa cạnh (tức cạnh đã đi qua rồi không đi lại được nữa), do đó để làm điều này, đơn giản bạn kiểm tra xem có tồn tại cung hay đường đi nào trong đồ thị temp không? Nếu tồn tại một cung hay cạnh thì đồ thị temp hay đồ thị ban đầu không có chu trình euler và trả về kết quả 0(sai_ tương ứng là không tìm thấy chu trình euler). Bạn viết code kiểm tra điều này*/

/* Nếu có chu trình euler thì bắt buộc đỉnh đầu và đỉnh cuối trong stack phải bằng nhau. Điều này tương đương với việc đỉnh đầu và đỉnh cuối trùng nhau trong chu trình Euler. Nếu đỉnh đầu và đỉnh cuối không trùng nhau thì bạn trả về kết quả 0 (sai_đồ thị temp hay đồ thị ban đầu không có chu trình euler). Bạn viết code kiểm tra điều này*/

    printf("\n Chu Trinh Euler : ");

/* Nếu không rơi vào 2 trường hợp trên thì bạn đã có chu trình euler rồi đó, hãy code mà xuất ra. Dựa vào stack đó*/

return 1; // trả về kết quả 1 tức có chu trình euler

}
```

Bước 4: Code trong hàm main để gọi hàm các hàm tương ứng và chạy. Có thể làm như sau:

```
void main()

{

  DOTHI g;
```

```
    clrscr();
    if (DocMaTranKe(inputfile, g) == 1)
    {
        printf("Da lay thong tin do thi tu file thanh cong.\n\n");
        XuatMaTranKe(g);
        printf("Bam 1 phim bat ki de bat dau xet tim chu trinh euler ...\n\n");
        getch();
        if (!KiemTraChuTrinhEuler(g))
        {
            printf("Khong co chu trinh Euler trong do thi cua ban\n");
            getch();
        }
    }
    getch();
}
```

BÀI TẬP 4: ĐƯỜNG ĐI EULER

Bạn hãy viết chương trình để kiểm tra xem đồ thị của bạn có đường đi euler không? nếu có thì hãy xuất ra đường đi euler đó.

HƯỚNG DẪN

Bước 1 và 2: Làm tương tự ở bài tập 3 (chu trình euler)

Bước 3: Trường hợp nếu như đồ thị không tồn tại chu trình euler thì đồ thị đó có thể tồn tại đường đi euler không?. Để làm điều này tiến hành viết hàm KiemTraDuongDiEuler như sau:

```
int KiemTraDuongDiEuler (DOTHI g)
{
  int i,j;
  int x = 0; // x là giá trị đỉnh bắt đầu xét đường đi euler, điều kiện x là đỉnh phải có bậc lẽ.
  /* bạn phải code chỗ này để tìm 1 đỉnh x bắt đầu tìm chu trình euler, đỉnh x này phải có bậc lẽ*/
  DOTHI temp = g; // tạo ra một bản copy của đồ thị để thực hiện thuật toán, vì trong quá trình thi hành thuật toán ta có xóa cạnh nên ta tạo ra bảng copy này để thực hiện việc xóa đó mà không ảnh hưởng đến đồ thị gốc (ban đầu).
  STACK stack; // tạo một stack như thuật toán đã trình bày
  khoitaoStack (stack); // ban đầu stack chưa có gì nên phải khởi tạo nó về 0.
  TimDuongDi(x,temp, stack); // bắt đầu tìm đường đi euler từ đỉnh x trong đồ thị temp, và thứ tự các đỉnh trong đương đi euler được lưu vào stack này.
  /* Bạn cần phải kiểm tra xem hàm TimDuongDi có tìm thấy đường đi euler trong đồ thị temp không? Bạn kiểm tra bằng cách nào? Vì đối với thuật toán của mình thì có xóa cạnh (tức cạnh đã đi qua rồi không đi lại được nữa), do đó để làm điều này, đơn giản bạn kiểm tra xem có tồn tại cung hay đường đi nào trong đồ thị temp không? Nếu tồn tại một cung hay cạnh thì đồ thị temp hay đồ thị ban đầu
```

```
không có đường đi euler và trả về kết quả 0(sai_ tương ứng là không tìm thấy đường đi euler). Bạn viết code kiểm tra điều này*/

    /* Nếu có đường đi euler thì bắt buộc đỉnh đầu và đỉnh cuối trong stack không giống nhau. Điều này tương đương với việc đỉnh đầu và đỉnh cuối không trùng nhau trong đường đi Euler. Nếu đỉnh đầu và đỉnh cuối trùng nhau thì bạn trả về kết quả 0 (sai_đồ thị temp hay đồ thị ban đầu không có đường đi euler). Bạn viết code kiểm tra điều này*/

    printf("\nĐường đi Euler : ");

    /* Nếu không rơi vào 2 trường hợp trên thì bạn đã có đường đi euler rồi đó, hãy code mà xuất ra. Dựa vào stack đó */

  return 1;

  }
```

Bước 4: Code trong hàm main để gọi hàm các hàm tương ứng và chạy. Có thể làm như sau:

```
  void main()

  {

    DOTHI g;

    clrscr();

    if (DocMaTranKe(inputfile, g) == 1)

    {

        printf("Da lay thong tin do thi tu file thanh cong.\n\n");

        XuatMaTranKe(g);

        printf("Bam 1 phim bat ki de bat dau xet tim chu trinh euler ...\n\n");

        getch();

        if (!KiemTraChuTrinhEuler(g))

        {

            printf("Khong co chu trinh Euler trong do thi cua ban\n");
```

```
            printf("Bam 1 phim bat ki de bat dau xet tim duong di euler
...\n\n");
            getch();
            if (!KiemTraDuongDiEuler(g))
            {
                printf("Khong co duong di Euler trong do thi cua ban \n");
            }
        }
    }
    getch();
  }
```

BÀI TẬP 5: TÌM KIẾM ĐƯỜNG ĐI BẰNG PHƯƠNG PHÁP DUYỆT THEO CHIỀU SÂU (DFS)

Bạn có một đồ thị g (gồm đỉnh n và ma trận kề a), bạn muốn tìm đường đi từ một đỉnh S (Start) đến một đỉnh F (Finish) trong đồ thị đó. Bạn hãy viết chương trình tìm đường đi có thể theo thuật toán duyệt theo chiều sâu (DFS). Nếu tìm có đường đi từ đỉnh S (Start) đến đỉnh F (Finish) trong đồ thị g này thì bạn xuất ra đường đi, còn nếu không có đường đi thì bạn thông báo không có đường đi từ đỉnh S (Start) đến đỉnh F (Finish).

HƯỚNG DẪN

Bước 1: Tạo 1 mảng 1 chiều LuuVet dùng để lưu vết đường đi từ S→ F, và 1 mảng 1 chiều khác với tên là ChuaXet dùng để đánh dấu đỉnh nào trong đồ thị đã xét rồi, đỉnh nào chưa xét trong quá trình tìm đường đi từ S→F.

```
int LuuVet[MAX]; // LuuVet[i] = đỉnh liền trước i trên đường đi từ S → i
int ChuaXet[MAX]; // ChuaXet[i] = 0 là đỉnh i chưa được xét đến trong quá trình tìm đường đi, còn ChuaXet[i] = 1 là đỉnh I được xét đến rồi trong quá trình tìm đường đi.
```

Bước 2: **Code phần duyệt theo chiều sâu (DFS)** theo dạng đệ qui.

```
void DFS(int v, GRAPH g) // hàm xét tại đỉnh v của đồ thị g
{
  ChuaXet[v] = 1; // gán lại giá trị đỉnh v trong mảng ChuaXet là 1, điều này có nghĩa là đỉnh v đã và đang được xét hay duyệt đến theo thuật toán.
  int u;
  for(u = 0; u < g.n ; u++)
  {
      if(g.a[v][u] != 0 && ChuaXet[u] == 0) // Xem xét có cạnh nào nối từ đỉnh v đến đỉnh u trong đồ thị g không (điều này tương ứng với g.a[v][u] != 0) và
```

đỉnh u đã được xét hay duyệt đến hay chưa? Nếu có cạnh nối từ đỉnh v đến đỉnh u và đỉnh u chưa được xét hay duyệt đến thì tiến hành duyệt đỉnh u

```
        {
            LuuVet[u] = v; // đánh dấu lại đỉnh u được đi đến từ đỉnh v trong quá trình duyệt theo thuật toán DFS
            DFS(u,g); // tiến hành nhảy tới đỉnh u và xét duyệt đỉnh u
        }
    }
}
```

Bước 3: Trước khi tiến hành duyệt DFS để tìm đường đi S→ F thì cần phải khởi tạo các giá trị thích hợp cho các mảng LuuVet và ChuaXet. Quá trình đó như sau:

```
void duyettheoDFS (int S, int F, GRAPH g) // hàm này dùng để tìm đường đi từ S→F trong đồ thị g theo thuật toán DFS
{
  int i;
  // khởi tạo lại các giá trị thích hợp cho mảng LuuVet và ChuaXet
  /* các bạn tự viết code khởi tạo các giá trị cho mạng LuuVet và ChuaXet. Gợi ý vì ban đầu chưa chạy thuật toán nên các đỉnh i đều chưa có vết đi đến đó nên đặt giá trị -1. Và ban đầu chưa chạy thuật toán nên đỉnh i trong đồ thị g đều chưa được xét nên giá trị là 0 */
  DFS(S,g); // tiến hành thuật toán duyệt theo chiều sâu
  // xuất đường đi
  if (ChuaXet[F] == 1) // sau khi thuật toán duyệt xong mà đỉnh F được xét hay duyệt đến thì nhãn của nó = 1 điều này có nghĩa là có đường đi từ S →F. Tiến hành xuất đường đi.
  {
      printf("Duong di tu dinh %d den dinh %d la: \n\t",S,F);
```

```
        i = F;
        printf("%d ", F);
        /* các bạn tự viết code xuất ra đường đi từ F← S hén. Gợi ý dựa vào mảng LuuVet để tiến hành truy vết ra đường đi từ S→F*/
    }
    Else // sau khi thuật toán duyệt xong mà đỉnh F chưa được xét hay duyệt đến thì nhãn của nó = 0 điều này có nghĩa là không có đường đi từ S →F. Thông báo không có đường đi từ S→F
    {
        printf("Khong co duong di tu dinh %d den dinh %d \n",S,F);
    }
}
```

Bước 4: Code trong hàm main để gọi hàm các hàm tương ứng và chạy. Có thể làm như sau:

```
void main()
{
  DOTHI g;
  clrscr();
  if (DocMaTranKe(inputfile, g) == 1)
  {
        printf("Da lay thong tin do thi tu file thanh cong.\n\n");
        XuatMaTranKe(g);
        printf("Bam 1 phim bat ki de bat dau duyet theo DFS ...\n\n");
        getch();
        duyettheoDFS(0,2,g);
  }
```

```
    getch();
}
```

BÀI TẬP 6: TÌM KIẾM ĐƯỜNG ĐI BẰNG PHƯƠNG PHÁP DUYỆT THEO CHIỀU RỘNG (BFS)

Bạn có một đồ thị g (gồm đỉnh n và ma trận kề a), bạn muốn tìm đường đi từ một đỉnh S (Start) đến một đỉnh F (Finish) trong đồ thị đó. Bạn hãy viết chương trình tìm đường đi có thể theo thuật toán duyệt theo chiều rộng (BFS). Nếu tìm có đường đi từ đỉnh S (Start) đến đỉnh F (Finish) trong đồ thị g này thì bạn xuất ra đường đi, còn nếu không có đường đi thì bạn thông báo không có đường đi từ đỉnh S (Start) đến đỉnh F (Finish).

HƯỚNG DẪN

Bước 1: Tạo 1 mảng 1 chiều LuuVet dùng để lưu vết đường đi từ S→ F, và 1 mảng 1 chiều khác với tên là ChuaXet dùng để đánh dấu đỉnh nào trong đồ thị đã xét rồi, đỉnh nào chưa xét trong quá trình tìm đường đi từ S→F.

```
int LuuVet[MAX]; // LuuVet[i] = đỉnh liền trước i trên đường đi từ S → i
int ChuaXet[MAX]; // ChuaXet[i] = 0 là đỉnh i chưa được xét đến trong quá trình tìm đường đi, còn ChuaXet[i] = 1 là đỉnh I được xét đến rồi trong quá trình tìm đường đi.
```

Bước 2: **Code phần duyệt theo chiều rộng (BFS)** không dùng đệ qui.

Đầu tiên tạo một cấu trúc hàng đợi như sau dùng để lưu thứ tự danh sách các đỉnh cần được duyệt.

```
struct QUEUE
{
  int size; // kích thước hiện tại hay số phần tử có trong hàng đợi
int array[MAX]; //mảng lưu các giá trị trong hàng đợi
};
```

Để khởi tạo một hàng đợi thì dùng hàm KhoiTaoQueue như sau:

```
void KhoiTaoQueue(QUEUE &Q)
```

```
{
  Q.size = 0; // vì ban đầu hàng đợi rỗng nên size của nó là 0
}
```

Để đẩy một giá trị vào hàng đợi, ta dùng hàm DayGiaTriVaoQueue như sau:

```
int DayGiaTriVaoQueue(QUEUE &Q,int value)
{
  if(Q.size + 1 >= 100) // nếu hàng đợi đã đầy rồi thì không thể đẩy thêm giá trị vào hàng đợi được nữa
      return 0; // trả về kết quả 0 báo cho người lập trình biết là không thể đẩy thêm giá trị vào hàng đợi được nữa.
  Q.array[Q.size] = value; // đẩy giá trị vào hàng đợi.
  Q.size++; // tăng số lượng phần tử trong hàng đợi
  return 1; // trả về kết quả 1 báo cho người lập trình biết là đã đẩy thêm giá trị vào hàng đợi thành công.
}
```

Để lấy một gái trị ra từ hàm đợi ta dùng hàm LayGiaTriRaKhoiQueue như sau:

```
int LayGiaTriRaKhoiQueue(QUEUE &Q,int &value)
{
  if(Q.size <= 0) // nếu hàng đợi rỗng thì không thể lấy ra giá trị trị nào được
      return 0; // trả về kết quả 0 báo cho người lập trình biết là hàng đợi rỗng, không thể lấy giá trị nào
  value = Q.array[0]; // lấy giá trị đầu tiên trong hàng đợi.
  for(int i = 0; i< Q.size - 1 ; i++) // tiến hành loại bỏ giá trị đầu tiên ra khỏi hàng đợi
      Q.array[i] = Q.array[i+1];
  Q.size--; // giảm kích thước hàng đợi vì đã lấy ra 1 giá trị hay phần tử
```

```
    return 1; // trả về kết quả 1 báo cho người lập trình biết là đã lấy giá trị ra khỏi hàng đợi thành công.
  }
```

Để kiểm tra hàm đợi có rỗng hay không, ta dùng hàm KiemTraQueueRong như sau:

```
  int KiemTraQueueRong(QUEUE Q)
  {
    if(Q.size <= 0) // nếu hàm đợi rỗng thì trả về kết quả 1, ngược lại không rỗng là 0
        return 1;
    return 0;
  }
```

Bước 3: Để tiến hành duyệt BFS tại một đỉnh v nào đó của đồ thị g, ta viết hàm BFS sau:

```
  void BFS(int v, GRAPH g)
  {
    QUEUE Q;
    KhoiTaoQueue(Q); // khởi tạo hàng đợi vì ban đầu thuật toán hàng đợi ta chưa có gì
    DayGiaTriVaoQueue(Q,v); // đẩy đỉnh v vào hàng đợi theo như thuật toán đã trình bày
    while(!KiemTraQueueRong(Q)) // trong khi hàm đợi chưa rỗng thì tiến hành các bước sau
    {
        LayGiaTriRaKhoiQueue(Q,v); //lấy phần tử đầu tiên trong hàng đợi ra và gán giá trị đó vào v
```

```
        ChuaXet[v] = 1; //bắt đầu xét đỉnh v nên nhãn ChuaXet của đỉnh v được gán là 1, tức đang xét hoặc đã xét trong thuật toán BFS
        for(int u = 0; u < g.n ; u++)
        {
            if(g.a[v][u] != 0 && ChuaXet[u] == 0) // Xem xét có cạnh nào nối từ đỉnh v đến đỉnh u trong đồ thị g không (điều này tương ứng với g.a[v][u] != 0) và đỉnh u đã được xét hay duyệt đến hay chưa? Nếu có cạnh nối từ đỉnh v đến đỉnh u và đỉnh u chưa được xét hay duyệt đến thì tiến hành như sau
            {
                DayGiaTriVaoQueue(Q,u); // đẩy đỉnh u vào hàng đợi theo như thuật toán đã trình bày
                if (LuuVet[u] == -1)
  LuuVet[u] = v; // đánh dấu lại đỉnh u được đi đến từ đỉnh v trong quá trình duyệt theo thuật toán BFS
            }
        }
    }
  }
```

Bước 4: Trước khi tiến hành duyệt BFS để tìm đường đi S→ F thì cần phải khởi tạo các giá trị thích hợp cho các mảng LuuVet và ChuaXet. Quá trình đó như sau

```
void duyettheoBFS (int S, int F, GRAPH g) // hàm này dùng để tìm đường đi từ S→F trong đồ thị g theo thuật toán BFS
{
  // khởi tạo lại các giá trị thích hợp cho mảng LuuVet và ChuaXet
  /* các bạn tự viết code khởi tạo các giá trị cho mạng LuuVet và ChuaXet. Gợi ý vì ban đầu chưa chạy thuật toán nên các đỉnh i đều chưa có vết đi đến đó nên đặt giá trị -1. Và ban đầu chưa chạy thuật toán nên đỉnh i trong đồ thị g đều chưa được xét nên giá trị là 0 */
```

```
    BFS(S,g); // tiến hành thuật toán BFS
    // xuất đường đi
    if (ChuaXet[F] == 1) // sau khi thuật toán duyệt xong mà đỉnh F được xét hay duyệt đến thì nhãn của nó = 1 điều này có nghĩa là có đường đi từ S →F. Tiến hành xuất đường đi.
    {
        printf("Duong di tu dinh %d den dinh %d la: \n\t",S,F);
        i = F;
        printf("%d ", F);
        /* các bạn tự viết code xuất ra đường đi từ F← S hén. Gợi ý dựa vào mảng LuuVet để tiến hành truy vết ra đường đi từ S→F*/
    }
    Else // sau khi thuật toán duyệt xong mà đỉnh F chưa được xét hay duyệt đến thì nhãn của nó = 0 điều này có nghĩa là không có đường đi từ S →F. Thông báo không có đường đi từ S→F
    {
        printf("Khong co duong di tu dinh %d den dinh %d \n",S,F);
    }
}
```

Bước 5: Code trong hàm main để gọi hàm các hàm tương ứng và chạy.

BÀI TẬP 7: TÌM CÂY KHUNG NHỎ NHẤT VỚI THUẬT TOÁN PRIM

Với một đồ thị cho trước gồm số đỉnh n và ma trận kề. Bạn hãy viết chương trình thi hành thuật toán Prim trên đồ thị đó để tìm cây khung nhỏ nhất. Nếu tìm được cây khung nhỏ nhất, thì bạn hãy xuất ra k là tổng các giá trị của cạnh trong cây khung nhỏ nhất (trọng lượng của cây khung nhỏ nhất) và dòng tiếp theo là các cạnh (u,v) thuộc cây khung này. Còn nếu không tìm được, xuất ra thông báo không tìm được cây khung nhỏ nhất hay đồ thị không liên thông.

HƯỚNG DẪN

Bước 1: Trước khi thi hành thuật toán Prim cần phải kiểm tra đồ thị có liên thông hay không? Nếu đồ thị không liên thông thì không có cây khung nhỏ nhất. Còn ngược lại thì có.

Lật lại bài tâp 2 "**xét tính liên thông**", xem và code phần xét liên thông trên đồ thị. Tuy nhiên, bạn cần có một sự thay đổi nhỏ là hàm liên thông phải trả về kết quả để mình biết đường (đồ thị liên thông hay không) mà xử lý.

```
int XetLienThong(DOTHI g)
{
  /*code như phần xét liên thông chỉ thêm trả kết quả về */
  return SoThanhPhanLT;
}
```

Bước 2: Tiến hành code thuật toán **Prim** như sau:

```
int ChuaXet[MAX]; // giá trị 0 là chưa xét, giá trị 1 là xét rồi.
typedef struct EDGE // khai báo 1 cấu trúc CANH cho cạnh của đồ thị
{
  int u;
  int v;
  int value;
```

```
}CANH;
CANH T[MAX]; // mảng lưu các cạnh trong thuật toán Prim
void Prim (DOTHI g)
{
  if (XetLienThong(g) != 1) // đi kiểm tra đồ thị có liên thông không, nếu kết quả trả về là 1 thì đồ thị liên thông (tức chỉ có 1 thành phần liên thông), còn khác 1 thì đồ thị không liên thông
  {
      printf ("Do thi khong lien thong, do do khong thuc hien duoc thuat toan Prim tim cay khung nho nhat\n");
      return;
  }
  int nT = 0; // dùng để lưu số cạnh trong thuật toán Prim
  for (int i = 0; i < g.n; i++) // ban đầu thuật toán Prim, chưa làm gì nên gán giá trị chưa xét cho tất cả các đỉnh bằng 0
      ChuaXet[i] = 0;
  ChuaXet[0] = 1; // bắt đầu thuật toán Prim, xuất phát từ đỉnh 1
  while (nT < g.n - 1) // nếu thuật toán đã thu thập đủ g.n-1 cạnh thì thuật toán dừng
  {
      CANH CanhNhoNhat; // dùng để lưu cạnh nhỏ nhất nối từ tập những đỉnh đã xét (giá trị chưa xét == 1) đến 1 đỉnh chưa xét nào đó trong đồ thị.
      int GiaTriNhoNhat = 100; // khởi tạo giá trị nhỏ nhất của cạnh nhỏ nhất là 100, bạn có thể thay đổi số này sao cho phù hợp với bài toán của bạn.
      for (int i = 0; i < g.n; i++) // duyệt các đỉnh trong đồ thị
      {
          if (ChuaXet[i] == 1) // xem đỉnh nào đã xét
```

```
            {
                for (int j = 0; j < g.n; j++)
                    if (ChuaXet[j] == 0 && g.a[i][j] != 0 && GiaTriNhoNhat > g.a[i][j]) // tìm đỉnh chưa xét j, có cạnh nối từ đỉnh i đến đỉnh j và giá trị của cạnh đó nhỏ hơn giá trị nhỏ nhất ở trên
                    {
                        CanhNhoNhat.u = i; // cập nhập lại giá trị trong CanhNhoNhat
                        CanhNhoNhat.v = j;
                        CanhNhoNhat.value = g.a[i][j];
                        GiaTriNhoNhat = g.a[i][j]; // cập nhập lại GiaTriNhoNhat
                    }
            }
        }
        T[nT] = CanhNhoNhat; //Thêm cạnh nhỏ nhất vào tập cạnh T của mình
        nT++; // tăng số cạnh lên 1
        ChuaXet[CanhNhoNhat.v] = 1; // gán lại giá trị chưa xét của đỉnh v là 1, tức đã xét rồi
    }

    int TongTrongSoCuaCayKhung = 0; // trọng số của cây khung nhỏ nhất
    // xuất cây khung nhỏ nhất bởi thuật toán Prim và giá trị của cây khung
    printf ("Cay khung nho nhat cua do thi la \n");
    for (i = 0; i < nT - 1; i++)
    {
```

```
        printf ("(%d,%d), ", T[i].u, T[i].v);
        TongTrongSoCuaCayKhung += T[i].value;
    }
    printf ("(%d,%d).\n", T[nT - 1].u, T[nT - 1].v);
    TongTrongSoCuaCayKhung += T[nT - 1].value;
    printf ("Tong gia tri cua cay khung la %d\n",TongTrongSoCuaCayKhung);
}
```

Bước 3: Code trong hàm main để gọi hàm các hàm tương ứng và chạy.

BÀI TẬP 8: TÌM CÂY KHUNG NHỎ NHẤT VỚI THUẬT TOÁN KRUSKAL

Với một đồ thị cho trước gồm số đỉnh n và ma trận kề. Bạn hãy viết chương trình thi hành thuật toán Kruskal trên đồ thị đó để tìm cây khung/bao trùm nhỏ nhất. Nếu tìm được cây khung/bao trùm nhỏ nhất, thì bạn hãy xuất ra k là tổng các giá trị/trọng số của các cạnh trong cây khung/bao trùm nhỏ nhất và dòng tiếp theo là các cạnh (u,v) thuộc cây khung/bao trùm này. Còn nếu không tìm được, xuất ra thông báo không tìm được cây khung/bao trùm nhỏ nhất hay đồ thị không liên thông.

HƯỚNG DẪN

Bước 1: Định nghĩa cấu trúc cạnh như sau:

```
typedef struct EDGE // khai báo 1 cấu trúc CANH cho cạnh của đồ thị
{
  int u;
  int v;
  int value;
}CANH;
```

Bước 2: viết hàm sắp xếp các cạnh trong danh sách các cạnh theo trọng số tăng dần.

```
void SapXepTang(CANH E[100],int tongsocanh)
{
  CANH canhtam;
  for(int i = 0 ; i < tongsocanh - 1 ; i++)
  {
      for(int j = i + 1 ; j < tongsocanh ; j++)
          if(E[i].value > E[j].value)
          {
```

```
                canhtam = E[i];
                E[i] = E[j];
                E[j] = canhtam;
            }
    }
}
```

Bước 3: viết hàm Kruskal để thi hành thuật toán Kruskal như sau

```
void Kruskal (DOTHI g)
{
  CANH listEdge[MAX]; // chứa danh sách tất cả các cạnh của đồ thị
  int tongsocanh = 0; // chứa tổng số cạnh trong đồ thị
  int i,j;
  for(i = 0 ; i < g.n ; i++) // tiến hành thêm các cạnh trong đồ thị vào listEdge
  {
      for( j = i+1 ; j< g.n ; j++)
          if(g.a[i][j] > 0)
          {
              listEdge[tongsocanh].u = i;
              listEdge[tongsocanh].v = j;
              listEdge[tongsocanh].value = g.a[i][j];
              tongsocanh++;
          }
  }
  // tiến hành sắp xếp các cạnh trong listEdge theo trọng số tăng dần
  SapXepTang(listEdge, tongsocanh);
```

```
int nT = 0; // số lượng các cạnh trong cây khung
CANH T[MAX]; // chứa các cạnh của cây khung
int nhan[MAX]; // chứa nhãn của các đỉnh trong đồ thị theo thuật toán
for (i = 0; i < g.n ; i++)
    nhan[i] = i;
int canhdangxet = 0; // lưu lại thuật toán đang xét cạnh thứ mấy trong danh sách listEdge
while(nT < g.n && canhdangxet < tongsocanh)
{
    // tiến hành thêm một cạnh vào cây khung mà không tạo chu trình bằng cách chọn cạnh mà đỉnh tại nhãn khác nhau
    if (nhan[listEdge[canhdangxet].u] != nhan[listEdge[canhdangxet].v])
    {
        T[nT] = listEdge[canhdangxet];
        nT++;
        // tiến hành sửa nhãn của tất cả các đỉnh có cùng giá trị với nhãn của đỉnh v thành nhãn của đỉnh u
        int giatri = nhan[listEdge[canhdangxet].v];
        for (j = 0; j < g.n; j++)
            if (nhan[j] == giatri)
                nhan[j] = nhan[listEdge[canhdangxet].u];
    }
    canhdangxet++; // xét cạnh kế tiếp trong danh sách cạnh listEdge
}
if(nT != g.n -1) // nếu số cạnh trong cây khung không đủ n-1 cạnh thì suy ra đồ thị không liên thông
```

```
        printf("\nDo thi khong lien thong \n");
    else // có cây khung, tiến hành xuất
    {
        int TongTrongSoCuaCayKhung = 0;
        printf("\nDo thi lien thong \n");
        printf ("Cay khung nho nhat cua do thi la \n");
        for (i = 0; i < nT; i++)
        {
            printf ("(%d,%d), ", T[i].u, T[i].v);
            TongTrongSoCuaCayKhung += T[i].value;
        }
        printf ("\nTong gia tri cua cay khung la
%d\n",TongTrongSoCuaCayKhung);
    }
  }
```

Bước 4: Code trong hàm main để gọi hàm các hàm tương ứng và chạy.

BÀI TẬP 9: TÌM ĐƯỜNG ĐI NGẮN NHẤT VỚI THUẬT TOÁN DIJKSTRA

Bạn có một đồ thị g (gồm đỉnh n và ma trận kề a), bạn muốn tìm đường đi ngắn nhất từ một đỉnh S (Start) đến một đỉnh F (Finish) trong đồ thị đó. Bạn hãy viết chương trình tìm đường đi ngắn nhất có thể theo thuật toán Dijkstra. Nếu tìm có đường đi từ đỉnh S (Start) đến đỉnh F (Finish) trong đồ thị g này thì bạn xuất ra đường đi, còn nếu không có đường đi thì bạn thông báo không có đường đi từ đỉnh S (Start) đến đỉnh F (Finish).

HƯỚNG DẪN

Bước 1: Định nghĩa VOCUC, tạo 1 mảng 1 chiều LuuVet dùng để lưu vết đường đi từ S→ F, 1 mảng 1 chiều khác với tên là ChuaXet dùng để đánh dấu đỉnh nào trong đồ thị đã xét rồi, đỉnh nào chưa xét trong quá trình tìm đường đi từ S→F, 1 mảng DoDaiDuongDiToi để lưu lại độ dài nhỏ nhất trong quá trình tìm đường đi từ S→F.

```
#define VOCUC 1000
int LuuVet[MAX]; // LuuVet[i] = đỉnh liền trước i trên đường đi từ S → i
int ChuaXet[MAX]; // ChuaXet[i] = 0 là đỉnh i chưa được xét đến trong quá trình tìm đường đi, còn ChuaXet[i] = 1 là đỉnh I được xét đến rồi trong quá trình tìm đường đi.
 int DoDaiDuongDiToi[MAX]; // Lưu lại độ dài nhỏ nhất tới đỉnh i
```

Bước 2: viết hàm TimDuongDiNhoNhat để tìm đỉnh chưa xét có giá trị đường đi nhỏ nhất, rồi tiếp tục thi hành thuật toán, căn cứ vào mảng DoDaiDuongDiToi.

```
int TimDuongDiNhoNhat(DOTHI g)
{
  int li = -1; // nếu không tìm thấy đỉnh nào thỏa điều kiện thì trả về -1
  float p = VOCUC;
  for(int i = 0 ; i< g.n ; i++)
  {
```

```
        if(ChuaXet[i] == 0 && DoDaiDuongDiToi[i] < p)
        {
            p = DoDaiDuongDiToi[i];
            li = i;
        }
    }
    return li; // trả về đỉnh chưa xét có giá trị đường đi nhỏ nhất
}
```

Bước 3: viết hàm CapNhatDuongDi để tiến hành cập nhập lại giá trị đường đi trong quá trình thi hành thuật toán tại đỉnh u.

```
void CapNhatDuongDi(int u, DOTHI g)
{
    ChuaXet[u] = 1;// đỉnh u đã được chọn nên phải gán giá trị ChuaXet của nó = 1
    for(int v = 0; v < g.n ; v++)
    {
        if(ChuaXet[v]==0 && g.a[u][v] >0 && g.a[u][v] < VOCUC) // tìm một đỉnh v chưa xét và có cạnh nối từ u → v
            if(DoDaiDuongDiToi[v] > DoDaiDuongDiToi[u] + g.a[u][v]) //nếu như độ dài đường đi tới đỉnh v từ đỉnh khác mà lớn hơn độ dài đường đi tới đỉnh u + cạnh (u,v) thì tiến hành cập nhập lại
            {
                DoDaiDuongDiToi[v] = DoDaiDuongDiToi[u] + g.a[u][v];
                LuuVet[v] = u;
            }
    }
```

```
}
```

Bước 4: viết hàm Dijkstra để tiến hành thuật toán Dijkstra tìm đường đi ngắn nhất từ đỉnh S→ đỉnh F.

```
void Dijkstra (int S, int F, DOTHI g)
{
  //Khởi tạo các giá trị cần thiết cho thuật toán
  int i;
  for (i = 0; i< g.n ; i++)
  {
      ChuaXet[i] = 0;
      DoDaiDuongDiToi[i] = VOCUC;
      LuuVet[i] = -1;
  }
  DoDaiDuongDiToi[S] = 0;

  //Thi hành thuật toán Dijkstra
  while(ChuaXet[F] == 0) // trong khi thuật toán tìm đường đi vẫn chưa xét đến đỉnh F thì tiếp tục
  {
      int u = TimDuongDiNhoNhat(g); // tìm đỉnh mà có độ dài đường đi nhỏ nhất ở bước hiện tại
      if(u == -1) break; // nếu như không tìm được đỉnh nào thì dừng thuật toán và kết quả không tìm thấy đường đi. Ngược lại tiến hành cập nhập lại độ dài đường đi.
      CapNhatDuongDi(u,g); // cập nhập lại độ dài đường đi
  }
```

```
    if (ChuaXet[F] == 1) //Xuất kết quả đường đi nếu quá trình thi hành thuật toán đã tới điểm F
    {
        printf("Duong di ngan nhat tu dinh %d den dinh %d la: \n\t",S,F);
        i = F;
        printf("%d ", F);
        while (LuuVet[i] != S)
        {
            printf ("<-%d", LuuVet[i]);
            i = LuuVet[i];
        }
        printf ("<-%d\n", LuuVet[i]);
        printf("\tVoi do dai la %d\n",DoDaiDuongDiToi[F]);
    }
    else // ngược lại thì không có đường đi từ S→F
    {
        printf("Khong co duong di tu dinh %d den dinh %d \n",S,F);
    }
}
```

Bước 5: Code trong hàm main để gọi hàm các hàm tương ứng và chạy.

BÀI TẬP 10: TÌM KIẾM ĐƯỜNG ĐI NGẮN NHẤT VỚI THUẬT TOÁN FLOYD

Bạn có một đồ thị g (gồm đỉnh n và ma trận kề a), bạn muốn tìm đường đi ngắn nhất từ một đỉnh S (Start) đến một đỉnh F (Finish) trong đồ thị đó. Bạn hãy viết chương trình tìm đường đi ngắn nhất có thể theo thuật toán Floyd. Nếu tìm có đường đi từ đỉnh S (Start) đến đỉnh F (Finish) trong đồ thị g này thì bạn xuất ra đường đi, còn nếu không có đường đi thì bạn thông báo không có đường đi từ đỉnh S (Start) đến đỉnh F (Finish).

HƯỚNG DẪN

Bước 1: Định nghĩa VOCUC, tạo 1 mảng 2 chiều Sau_Nut dùng để lưu vết đường đi từ bất kỳ đỉnh i đến đỉnh j nào, 1 mảng 2 chiều khác với tên là L dùng để lưu lại độ dài đường đi ngắn nhất từ đỉnh i tới đỉnh j trong đồ thị.

```
#define VOCUC 1000
int Sau_Nut[MAX][MAX]; // Sau_Nut[i][j] = đỉnh liền sau i trên đường đi từ i → j
int L[MAX][MAX]; // L[i][j] = lưu lại độ dài đường đi ngắn nhất từ đỉnh i tới đỉnh j trong đồ thị
```

Bước 2: viết hàm Floyd để tiến hành thuật toán Floyd tìm đường đi ngắn nhất từ đỉnh giữa hai đỉnh bất kỳ i và j.

```
void Floyd(DOTHI g)
{
  int i,j;
  // khởi tạo giá trị cho 2 mảng 2 chiều L và Sau_Nut như trong thuật toán
  for(i = 0; i< g.n ; i++)
  {
      for( j = 0; j < g.n ; j++)
      {
```

```
                if(g.a[i][j] > 0) // nếu có cạnh nối đỉnh i với đỉnh j
                {
                    Sau_Nut[i][j] = j; // khởi tạo đỉnh liền sau i trên đường tìm đường đi từ i tới j là j.
                    L[i][j] = g.a[i][j]; // lưu lại độ dài ngắn nhất tại thời điểm hiện tại từ điểm i đến đỉnh j là cạnh nối đỉnh i với đỉnh j.
                }
                else // không có cạnh nối từ đỉnh i đến đỉnh j
                {
                    Sau_Nut[i][j] = -1; // khởi tạo đỉnh liền sau i trên đường tìm đường đi từ i tới j là -1.
                    L[i][j] = VOCUC; // lưu lại độ dài ngắn nhất tại thời điểm hiện tại từ điểm i đến đỉnh j là VOCUC (không có đường đi).
                }
            }
        }

        // Thi hành thuật toán Floyd
        for(int k = 0; k< g.n ; k++)
        {
            for(i = 0 ; i < g.n ; i++)
            {
                for(j = 0; j< g.n ; j++)
                {
                    if(L[i][j] > L[i][k] + L[k][j]) // nếu tồn tại một đỉnh trung gian k sao cho đường đi từ đỉnh i qua đỉnh k tới đỉnh j ngắn hơn đường đi từ đỉnh i đến đỉnh j thì tiến hành chọn đường đi đó.
```

```
                    {
                        L[i][j] = L[i][k] + L[k][j]; // cập nhập lại độ dài đường đi từ i tới j.
                        Sau_Nut[i][j] = Sau_Nut[i][k]; // lưu lại đỉnh liền sau i trên đường tìm đường đi từ i tới j là k.
                    }
                }
            }
        }
        // xuất kết quả tìm đường đi ngắn nhất từ S --> F
        int S,F;
        printf ("nhap vao dinh bat dau: ");
        scanf("%d",&S);
        printf("Nhap vao dinh ket thuc: ");
        scanf("%d",&F);
        if (Sau_Nut[S][F] == -1) // nếu giá trị Sau_Nut[S][F] là -1 thì điều này có nghĩa là đỉnh liền sau S là -1 trên đường tìm đường đi từ S tới F. Tương đương với việc không có đường đi từ đỉnh S đến F.
        {
            printf ("Khong co duong di tu dinh %d den dinh %d la :\n",S,F);
        }
        else // ngược lại có đường đi từ S tới F.

        {
            printf ("Duong di ngan nhat tu dinh %d den dinh %d la :\n",S,F);
            printf ("\t%d",S);
```

```
        int u = S;
        while(Sau_Nut[u][F] != F) // trong khi giá trị đỉnh liền sau u trên đường tìm đường đi ngắn nhất từ S tới F không phải là F thì tiếp tục theo vết đi tới đến khi nào gặp F thì dừng.
        {
            u = Sau_Nut[u][F];
            printf (" --> %d",u);
        }
        printf (" --> %d",F);
        printf ("\n\tVoi tong trong so la %d",L[S][F]);
    }
}
```

Bước 3: Code trong hàm main để gọi hàm các hàm tương ứng và chạy.

TÀI LIỆU THAM KHẢO

1) Bài giảng môn Lý thuyết đồ thị của giảng viên.

2) Trần Đan Thư - Dương Anh Đức, *Giáo trình lý thuyết đồ thị*, Đại học Khoa học Tự nhiên Thành phố Hồ Chí Minh - Nxb. ĐHQG Thành phố Hồ Chí Minh.

3) Nguyễn Đức Nghĩa - Nguyễn Tô Thành, *Toán rời rạc*, Nhà xuất bản Đại học Quốc gia Hà Nội, 1997

4) Vũ Đình Hòa, *Định lý và vấn đề về đồ thị hữu hạn*, Nhà xuất bản Giáo dục, 2001

5) *Lý thuyết đồ* thị *và ứng dụng*, Nhà xuất bản Khoa học và Kỹ thuật - Hà Nội 2000.

6) GS Nguyễn Hữu Anh, "Toán rời rạc", Nhà xuất bản lao động xã hội