§0. MỞ ĐẦU


Leonhard Euler
(1707-1783)

Trên thực tế có nhiều bài toán liên quan tới một tập các đối tượng và những mối liên hệ giữa chúng, đòi hỏi toán học phải đặt ra một mô hình biểu diễn một cách chặt chẽ và tổng quát bằng ngôn ngữ ký hiệu, đó là đồ thị. Những ý tưởng cơ bản của nó được đưa ra từ thế kỷ thứ XVIII bởi nhà toán học Thuỵ Sĩ Leonhard Euler, ông đã dùng mô hình đồ thị để giải bài toán về những cây cầu Konigsberg nổi tiếng.

Mặc dù Lý thuyết đồ thị đã được khoa học phát triển từ rất lâu nhưng lại có nhiều ứng dụng hiện đại. Đặc biệt trong khoảng vài mươi năm trở lại đây, cùng với sự ra đời của máy tính điện tử và sự phát triển nhanh chóng của Tin học, Lý thuyết đồ thị càng được quan tâm đến nhiều hơn. Đặc biệt là các thuật toán trên đồ thị đã có nhiều ứng dụng trong nhiều lĩnh vực khác nhau như: Mạng máy tính, Lý thuyết mã, Tối ưu hoá, Kinh tế học v.v... Chẳng hạn như trả lời câu hỏi: Hai máy tính trong mạng có thể liên hệ được với nhau hay không ?; hay vấn đề phân biệt hai hợp chất hoá học có cùng công thức phân tử nhưng lại khác nhau về công thức cấu tạo cũng được giải quyết nhờ mô hình đồ thị. Hiện nay, môn học này là một trong những kiến thức cơ sở của bộ môn khoa học máy tính.

Trong phạm vi một chuyên đề, không thể nói kỹ và nói hết những vấn đề của lý thuyết đồ thị. Tập bài giảng này sẽ xem xét lý thuyết đồ thị dưới góc độ người lập trình, tức là khảo sát những **thuật toán cơ bản nhất** có thể **dễ dàng cài đặt trên máy tính** một số ứng dụng của nó. Các khái niệm trừu tượng và các phép chứng minh sẽ được diễn giải một cách hình thức cho đơn giản và dễ hiểu chứ không phải là những chứng minh chặt chẽ dành cho người làm toán. Công việc của người lập trình là đọc hiểu được ý tưởng cơ bản của thuật toán và cài đặt được chương trình trong bài toán tổng quát cũng như trong trường hợp cụ thể. Thông thường sau quá trình rèn luyện, hầu hết những người lập trình gần như phải **thuộc lòng** các mô hình cài đặt, để khi áp dụng có thể cài đặt đúng ngay và hiệu quả, không bị mất thời giờ vào các công việc gỡ rối. Bởi việc gỡ rối một thuật toán tức là phải dò lại từng bước tiến hành và tự trả lời câu hỏi: "Tại bước đó nếu đúng thì phải như thế nào ?", đó thực ra là tiêu phí thời gian vô ích để chứng minh lại tính đúng đắn của thuật toán trong trường hợp cụ thể, với một bộ dữ liệu cụ thể.

Trước khi tìm hiểu các vấn đề về lý thuyết đồ thị, bạn phải có **kỹ thuật lập trình khá tốt**, ngoài ra nếu đã có tìm hiểu trước về các kỹ thuật vét cạn, quay lui, một số phương pháp tối ưu hoá, các bài toán quy hoạch động thì sẽ giúp ích nhiều cho việc đọc hiểu các bài giảng này.

§1. CÁC KHÁI NIỆM CƠ BẢN

I. ĐỊNH NGHĨA ĐỒ THỊ (GRAPH)

Là một cấu trúc rời rạc gồm các đỉnh và các cạnh nối các đỉnh đó. Được mô tả hình thức:

$$G = (V, E)$$

V gọi là tập các **đỉnh** (Vertices) và E gọi là tập các **cạnh** (Edges). Có thể coi E là tập các cặp (u, v) với u và v là hai đỉnh của V.

Một số hình ảnh của đồ thị:

Sơ đồ giao thông | Mạng máy tính

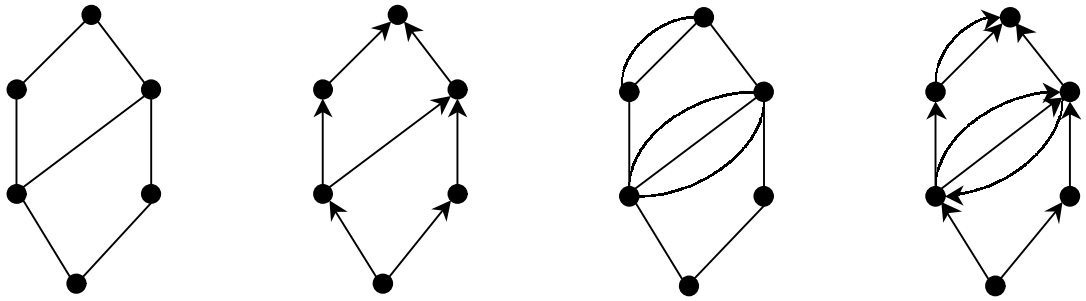
Hình 1: Ví dụ về mô hình đồ thị

Có thể phân loại đồ thị theo đặc tính và số lượng của tập các cạnh E:

Cho đồ thị G = (V, E). Định nghĩa một cách hình thức

1. G được gọi là **đơn đồ thị** nếu giữa hai đỉnh u, v của V có nhiều nhất là 1 cạnh trong E nối từ u tới v.
2. G được gọi là **đa đồ thị** nếu giữa hai đỉnh u, v của V có thể có nhiều hơn 1 cạnh trong E nối từ u tới v (Hiển nhiên đơn đồ thị cũng là đa đồ thị).
3. G được gọi là đồ thị **vô hướng** nếu các cạnh trong E là không định hướng, tức là cạnh nối hai đỉnh u, v bất kỳ cũng là cạnh nối hai đỉnh v, u. Hay nói cách khác, tập E gồm các cặp (u, v) không tính thứ tự. (u, v)≡(v, u)
4. G được gọi là đồ thị **có hướng** nếu các cạnh trong E là có định hướng, có thể có cạnh nối từ đỉnh u tới đỉnh v nhưng chưa chắc đã có cạnh nối từ đỉnh v tới đỉnh u. Hay nói cách khác, tập E gồm các cặp (u, v) có tính thứ tự: (u, v) ≠ (v, u). Trong đồ thị có hướng, các cạnh được gọi là các **cung**. Đồ thị vô hướng cũng có thể coi là đồ thị có hướng nếu như ta coi cạnh nối hai đỉnh u, v bất kỳ tương đương với hai cung (u, v) và (v, u).

Ví dụ:



Vô hướng | Có hướng — Đơn đồ thị

Vô hướng | Có hướng — Đa đồ thị

Hình 2: Phân loại đồ thị

II. CÁC KHÁI NIỆM

Như trên định nghĩa **đồ thị G = (V, E) là một cấu trúc rời rạc**, tức là các tập V và E hoặc là tập hữu hạn, hoặc là tập đếm được, có nghĩa là ta có thể đánh số thứ tự 1, 2, 3... cho các phần tử của tập V và E. Hơn nữa, đứng trên phương diện người lập trình cho máy tính thì ta chỉ quan tâm đến các đồ thị hữu hạn (V và E là tập hữu hạn) mà thôi, chính vì vậy từ đây về sau, nếu không chú thích gì thêm thì khi nói tới đồ thị, ta hiểu rằng đó là đồ thị hữu hạn.

Cạnh liên thuộc, đỉnh kề, bậc

- Đối với đồ thị vô hướng G = (V, E). Xét một cạnh e ∈ E, nếu e = (u, v) thì ta nói hai đỉnh u và v là **kề nhau** (adjacent) và cạnh e này **liên thuộc** (incident) với đỉnh u và đỉnh v.
- Với một đỉnh v trong đồ thị, ta định nghĩa **bậc** (degree) của v, ký hiệu deg(v) là số cạnh liên thuộc với v. Dễ thấy rằng trên đơn đồ thị thì số cạnh liên thuộc với v cũng là số đỉnh kề với v.

Định lý: Giả sử G = (V, E) là đồ thị vô hướng với m cạnh, khi đó tổng tất cả các bậc đỉnh trong V sẽ bằng 2m:

$$\sum_{v \in V} \deg(v) = 2m$$

Chứng minh: Khi lấy tổng tất cả các bậc đỉnh tức là mỗi cạnh e = (u, v) bất kỳ sẽ được tính một lần trong deg(u) và một lần trong deg(v). Từ đó suy ra kết quả.

Hệ quả: Trong đồ thị vô hướng, số đỉnh bậc lẻ là số chẵn

- Đối với đồ thị có hướng G = (V, E). Xét một cung e ∈ E, nếu e = (u, v) thì ta nói **u nối tới v** và **v nối từ u,** cung e là đi **ra khỏi đỉnh u và đi vào đỉnh v**. Đỉnh u khi đó được gọi là đỉnh đầu, đỉnh v được gọi là đỉnh cuối của cung e.
- Với mỗi đỉnh v trong đồ thị có hướng, ta định nghĩa: **Bán bậc ra** của v ký hiệu $\deg^+(v)$ là số cung đi ra khỏi nó; **bán bậc vào** ký hiệu $\deg^-(v)$ là số cung đi vào đỉnh đó

Định lý: Giả sử G = (V, E) là đồ thị có hướng với m cung, khi đó tổng tất cả các bán bậc ra của các đỉnh bằng tổng tất cả các bán bậc vào và bằng m:

$$\sum_{v \in V} \deg^-(v) = \sum_{v \in V} \deg^+(v) = m$$

Chứng minh: Khi lấy tổng tất cả các bán bậc ra hay bán bậc vào, mỗi cung (u, v) bất kỳ sẽ được tính đúng 1 lần trong $\deg^+(u)$ và cũng được tính đúng 1 lần trong $\deg^-(v)$. Từ đó suy ra kết quả

Một số tính chất của đồ thị có hướng không phụ thuộc vào hướng của các cung. Do đó để tiện trình bày, trong một số trường hợp ta có thể không quan tâm đến hướng của các cung và coi các cung đó là các cạnh của đồ thị vô hướng. Và đồ thị vô hướng đó được gọi là **đồ thị vô hướng nền** của đồ thị có hướng ban đầu.

§2. BIỂU DIỄN ĐỒ THỊ TRÊN MÁY TÍNH

I. MA TRẬN LIỀN KỀ (MA TRẬN KỀ)

Giả sử G = (V, E) là một **đơn đồ thị** có số đỉnh (ký hiệu $|V|$) là n, Không mất tính tổng quát có thể coi các đỉnh được đánh số 1, 2, ..., n. Khi đó ta có thể biểu diễn đồ thị bằng một ma trận vuông A = [a_{ij}] cấp n. Trong đó:

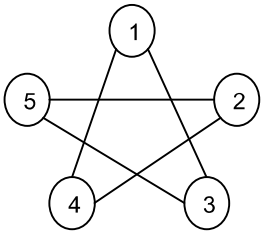
- a_{ij} = 1 nếu (i, j) ∈ E
- a_{ij} = 0 nếu (i, j) ∉ E
- Quy ước a_{ii} = 0 với ∀i;

Đối với đa đồ thị thì việc biểu diễn cũng tương tự trên, chỉ có điều nếu như (i, j) là cạnh thì không phải ta ghi số 1 vào vị trí a_{ij} mà là ghi số cạnh nối giữa đỉnh i và đỉnh j

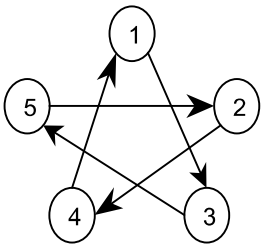
Ví dụ:

	1	2	3	4	5
1	0	0	1	1	0
2	0	0	0	1	1
3	1	0	0	0	1
4	1	1	0	0	0
5	0	1	1	0	0

⇐



	1	2	3	4	5
1	0	0	1	0	0
2	0	0	0	1	0
3	0	0	0	0	1
4	1	0	0	0	0
5	0	1	0	0	0



Các tính chất của ma trận liền kề:

1. Đối với đồ thị vô hướng G, thì ma trận liền kề tương ứng là ma trận đối xứng (a_{ij} = a_{ji}), điều này không đúng với đồ thị có hướng.
2. Nếu G là đồ thị vô hướng và A là ma trận liền kề tương ứng thì trên ma trận A:
   Tổng các số trên hàng i = Tổng các số trên cột i = Bậc của đỉnh i = deg(i)
3. Nếu G là đồ thị có hướng và A là ma trận liền kề tương ứng thì trên ma trận A:
   - Tổng các số trên hàng i = Bán bậc ra của đỉnh i = deg^+(i)
   - Tổng các số trên cột i = Bán bậc vào của đỉnh i = deg^-(i)

Trong trường hợp G là đơn đồ thị, ta có thể biểu diễn ma trận liền kề A tương ứng là các phần tử logic. a_{ij} = TRUE nếu (i, j) ∈ E và a_{ij} = FALSE nếu (i, j) ∉ E

Ưu điểm của ma trận liền kề:

- Đơn giản, trực quan, dễ cài đặt trên máy tính
- Để kiểm tra xem hai đỉnh (u, v) của đồ thị có kề nhau hay không, ta chỉ việc kiểm tra bằng một phép so sánh: $a_{uv} \neq 0$.

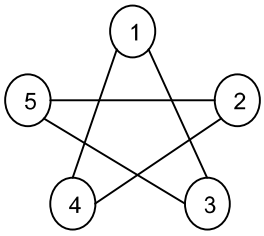
Nhược điểm của ma trận liền kề:

- Bất kể số cạnh của đồ thị là nhiều hay ít, ma trận liền kề luôn luôn đòi hỏi n^2 ô nhớ để lưu các phần tử ma trận, điều đó gây lãng phí bộ nhớ dẫn tới việc không thể biểu diễn được đồ thị với số đỉnh lớn.

Với một đỉnh u bất kỳ của đồ thị, nhiều khi ta phải xét tất cả các đỉnh v khác kề với nó, hoặc xét tất cả các cạnh liên thuộc với nó. Trên ma trận liền kề việc đó được thực hiện bằng cách xét tất cả các đỉnh v và kiểm tra điều kiện $a_{uv} \neq 0$. Như vậy, ngay cả khi đỉnh u là **đỉnh cô lập** (không kề với đỉnh nào) hoặc **đỉnh treo** (chỉ kề với 1 đỉnh) ta cũng buộc phải xét tất cả các đỉnh và kiểm tra điều kiện trên dẫn tới lãng phí thời gian

II. DANH SÁCH CẠNH

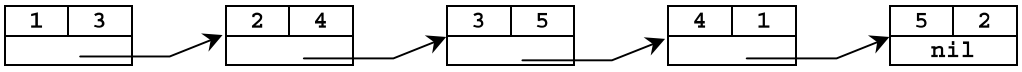
Trong trường hợp đồ thị có n đỉnh, m cạnh, ta có thể biểu diễn đồ thị dưới dạng danh sách cạnh, trong cách biểu diễn này, người ta liệt kê tất cả các cạnh của đồ thị trong một danh sách, mỗi phần tử của danh sách là một cặp (u, v) tương ứng với một cạnh của đồ thị. (Trong trường hợp đồ thị có hướng thì mỗi cặp (u, v) tương ứng với một cung, u là đỉnh đầu và v là đỉnh cuối của cung). Danh sách được lưu trong bộ nhớ dưới dạng mảng hoặc danh sách móc nối. Ví dụ với đồ thị dưới đây:



Cài đặt trên mảng:

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

Cài đặt trên danh sách móc nối:



Ưu điểm của danh sách cạnh:

- Trong trường hợp đồ thị thưa (có số cạnh tương đối nhỏ: chẳng hạn m < 6n), cách biểu diễn bằng danh sách cạnh sẽ tiết kiệm được không gian lưu trữ, bởi nó chỉ cần 2m ô nhớ để lưu danh sách cạnh.
- Trong một số trường hợp, ta phải xét tất cả các cạnh của đồ thị thì cài đặt trên danh sách cạnh làm cho việc duyệt các cạnh dễ dàng hơn. (Thuật toán Kruskal chẳng hạn)

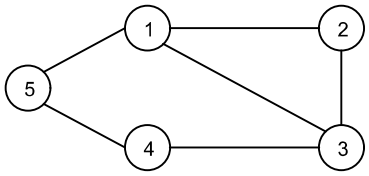
Nhược điểm của danh sách cạnh:

- Nhược điểm cơ bản của danh sách cạnh là khi ta cần duyệt tất cả các đỉnh kề với đỉnh v nào đó của đồ thị, thì chẳng có cách nào khác là phải duyệt tất cả các cạnh, lọc ra những cạnh có chứa đỉnh v và xét đỉnh còn lại. Điều đó khá tốn thời gian trong trường hợp đồ thị dày (nhiều cạnh).

III. DANH SÁCH KỀ

Để khắc phục nhược điểm của các phương pháp ma trận kề và danh sách cạnh, người ta đề xuất phương pháp biểu diễn đồ thị bằng danh sách kề. Trong cách biểu diễn này, với mỗi đỉnh v của đồ thị, ta cho tương ứng với nó một danh sách các đỉnh kề với v.

Với đồ thị G = (V, E). V gồm n đỉnh và E gồm m cạnh. Có hai cách cài đặt danh sách kề phổ biến:



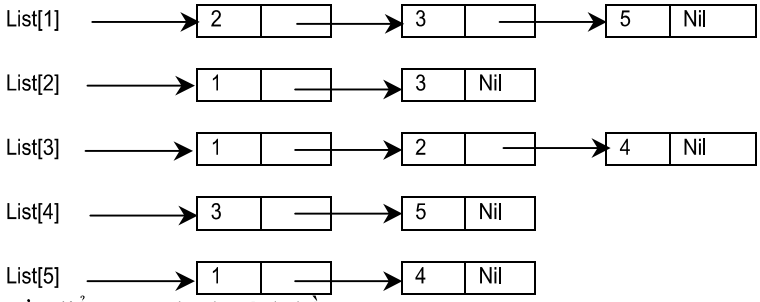
Cách 1: (Forward Star) Dùng một mảng các đỉnh, mảng đó chia làm n đoạn, đoạn thứ i trong mảng lưu danh sách các đỉnh kề với đỉnh i: Ví dụ với đồ thị sau, danh sách kề sẽ là một mảng A gồm 12 phần tử:

1	2	3	4	5	6	7	8	9	10	11	12
2	**3**	**5**	**1**	**3**	**1**	**2**	**4**	**3**	**5**	**1**	**4**
Đoạn 1			Đoạn 2		Đoạn 3			Đoạn 4		Đoạn 5	

Để biết một đoạn nằm từ chỉ số nào đến chỉ số nào, ta có một mảng lưu vị trí riêng. Ta gọi mảng lưu vị trí đó là mảng Head. Head[i] sẽ bằng chỉ số đứng liền trước đoạn thứ i. Quy ước Head[n + 1] sẽ bằng m. Với đồ thị bên thì mảng VT[1..6] sẽ là: (0, 3, 5, 8, 10, 12)

Như vậy đoạn từ vị trí Head[i] + 1 đến Head[i + 1] trong mảng A sẽ chứa các đỉnh kề với đỉnh i. Lưu ý rằng với đồ thị có hướng gồm m cung thì cấu trúc Forward Star cần phải đủ chứa m phần tử, với đồ thị vô hướng m cạnh thì cấu trúc Forward Star cần phải đủ chứa 2m phần tử

Cách 2: Dùng các danh sách móc nối: Với mỗi đỉnh i của đồ thị, ta cho tương ứng với nó một danh sách móc nối các đỉnh kề với i, có nghĩa là tương ứng với một đỉnh i, ta phải lưu lại List[i] là chốt của một danh sách móc nối. Ví dụ với đồ thị trên, danh sách móc nối sẽ là:



Ưu điểm của danh sách kề:

- Đối với danh sách kề, việc duyệt tất cả các đỉnh kề với một đỉnh v cho trước là hết sức dễ dàng, cái tên "danh sách kề" đã cho thấy rõ điều này. Việc duyệt tất cả các cạnh cũng đơn giản vì một cạnh thực ra là nối một đỉnh với một đỉnh khác kề nó.

Nhược điểm của danh sách kề

- Về lý thuyết, so với hai phương pháp biểu diễn trên, danh sách kề tốt hơn hẳn. Chỉ có điều, trong trường hợp cụ thể mà ma trận kề hay danh sách cạnh **không thể hiện nhược điểm** thì ta nên dùng ma trận kề (hay danh sách cạnh) bởi cài đặt danh sách kề có phần dài dòng hơn.

IV. NHẬN XÉT

Trên đây là nêu các cách biểu diễn đồ thị trong bộ nhớ của máy tính, còn nhập dữ liệu cho đồ thị thì có nhiều cách khác nhau, dùng cách nào thì tuỳ. Chẳng hạn nếu biểu diễn bằng ma trận kề mà cho nhập dữ liệu cả ma trận cấp n x n (n là số đỉnh) thì khi nhập từ bàn phím sẽ rất mất thời gian, ta cho nhập kiểu danh sách cạnh cho nhanh. Chẳng hạn mảng A (nxn) là ma trận kề của một đồ thị vô hướng thì ta có thể khởi tạo ban đầu mảng A gồm toàn số 0, sau đó cho người sử dụng nhập các cạnh bằng cách nhập các cặp (i, j); chương trình sẽ tăng A[i, j] và A[j, i] lên 1. Việc nhập có thể cho kết thúc khi người sử dụng nhập giá trị i = 0. Ví dụ:

```
program Nhap_Do_Thi;
```

```
var
  A: array[1..100, 1..100] of Integer; {Ma trận kề của đồ thị}
  n, i, j: Integer;
begin
  Write('Number of vertices'); ReadLn(n);
  FillChar(A, SizeOf(A), 0);
  repeat
    Write('Enter edge (i, j) (i = 0 to exit) ');
    ReadLn(i, j);      {Nhập một cặp (i, j) tưởng như là nhập danh sách cạnh}
    if i <> 0 then
      begin            {nhưng lưu trữ trong bộ nhớ lại theo kiểu ma trận kề}
        Inc(A[i, j]);
        Inc(A[j, i]);
      end;
  until i = 0;         {Nếu người sử dụng nhập giá trị i = 0 thì dừng quá trình nhập, nếu không thì tiếp tục}
end.
```

Trong nhiều trường hợp đủ không gian lưu trữ, việc chuyển đổi từ cách biểu diễn nào đó sang cách biểu diễn khác không có gì khó khăn. Nhưng đối với thuật toán này thì làm trên ma trận kề ngắn gọn hơn, đối với thuật toán kia có thể làm trên danh sách cạnh dễ dàng hơn v.v... Do đó, với mục đích dễ hiểu, các chương trình sau này sẽ lựa chọn phương pháp biểu diễn sao cho việc cài đặt đơn giản nhất nhằm nêu bật được bản chất thuật toán. Còn trong trường hợp cụ thể bắt buộc phải dùng một cách biểu diễn nào đó khác, thì việc sửa đổi chương trình cũng không tốn quá nhiều thời gian.

§3. CÁC THUẬT TOÁN TÌM KIẾM TRÊN ĐỒ THỊ

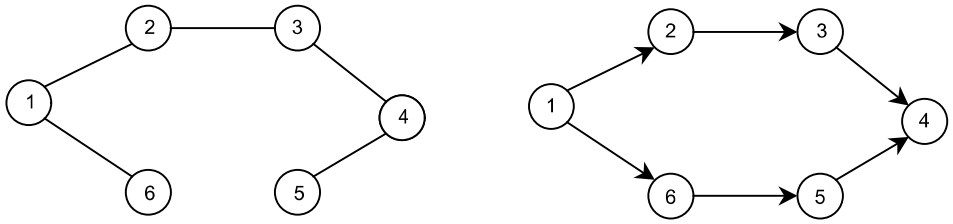
I. BÀI TOÁN

Cho đồ thị G = (V, E). u và v là hai đỉnh của G. Một **đường đi** (path) độ dài l từ đỉnh u đến đỉnh v là dãy $(u = x_0, x_1, ..., x_l = v)$ thoả mãn $(x_i, x_{i+1}) \in E$ với $\forall i: (0 \le i < l)$.

Đường đi nói trên còn có thể biểu diễn bởi dãy các cạnh: $(u = x_0, x_1), (x_1, x_2), ..., (x_{l-1}, x_l = v)$

Đỉnh u được gọi là đỉnh đầu, đỉnh v được gọi là đỉnh cuối của đường đi. Đường đi có đỉnh đầu trùng với đỉnh cuối gọi là **chu trình** (Circuit), đường đi không có cạnh nào đi qua hơn 1 lần gọi là **đường đi đơn,** tương tự ta có khái niệm **chu trình đơn.**

Ví dụ: Xét một đồ thị vô hướng và một đồ thị có hướng dưới đây:



Trên cả hai đồ thị, (1, 2, 3, 4) là đường đi đơn độ dài 3 từ đỉnh 1 tới đỉnh 4. Bởi (1, 2) (2, 3) và (3, 4) đều là các cạnh (hay cung). (1, 6, 5, 4) không phải đường đi bởi (6, 5) không phải là cạnh (hay cung).

Một bài toán quan trọng trong lý thuyết đồ thị là bài toán duyệt tất cả các đỉnh có thể đến được từ một đỉnh xuất phát nào đó. Vấn đề này đưa về một bài toán liệt kê mà yêu cầu của nó là không được bỏ sót hay lặp lại bất kỳ đỉnh nào. Chính vì vậy mà ta phải xây dựng những thuật toán cho phép **duyệt một cách hệ thống** các đỉnh, những thuật toán như vậy gọi là những thuật toán **tìm kiếm trên đồ thị** và ở đây ta quan tâm đến hai thuật toán cơ bản nhất: **thuật toán tìm kiếm theo chiều sâu** và **thuật toán tìm kiếm theo chiều rộng** cùng với một số ứng dụng của chúng.

Lưu ý:

1. Những cài đặt dưới đây là cho đơn đồ thị vô hướng, muốn làm với đồ thị có hướng hay đa đồ thị cũng không phải sửa đổi gì nhiều.
2. Dữ liệu về đồ thị sẽ được nhập từ file văn bản GRAPH.INP. Trong đó:
   - Dòng 1 chứa số đỉnh n (≤ 100), số cạnh m của đồ thị, đỉnh xuất phát S, đỉnh kết thúc F cách nhau một dấu cách.
   - m dòng tiếp theo, mỗi dòng có dạng hai số nguyên dương u, v cách nhau một dấu cách, thể hiện có cạnh nối đỉnh u và đỉnh v trong đồ thị.
3. Kết quả ghi ra file văn bản GRAPH.OUT
   - Dòng 1: Ghi danh sách các đỉnh có thể đến được từ S
   - Dòng 2: Đường đi từ S tới F được in ngược theo chiều từ F về S