LỜI GIỚI THIỆU

Toán rời rạc là một lĩnh vực nghiên cứu và xử lý các đối tượng rời rạc dùng để đếm các đối tượng, và nghiên cứu mối quan hệ giữa các tập rời rạc. Một trong những yếu tố làm Toán rời rạc trở nên quan trọng là việc lưu trữ, xử lý thông tin trong các hệ thống máy tính về bản chất là rời rạc. Chính vì lý do đó, Toán học rời rạc là một môn học bắt buộc mang tính chất kinh điển của các ngành Công nghệ thông tin và Điện tử Viễn thông. Tài liệu hướng dẫn môn học Toán học rời rạc được xây dựng được xây dựng dựa trên cơ sở kinh nghiệm giảng dạy môn học và kế thừa từ giáo trình [1, 2].

Tài liệu được trình bày thành hai phần. Trong đó, phần I trình bày những kiến thức cơ bản về lý thuyết tổ hợp thông qua việc giải quyết bốn bài toán cơ bản đó là: Bài toán đếm, Bài toán tồn tại, Bài toán liệt kê và Bài toán tối ưu. Phần II trình bày những kiến thức cơ bản về Lý thuyết đồ thị: khái niệm, định nghĩa, các thuật toán trên đồ thị, đồ thị Euler, đồ thị Hamilton. Một số bài toán có ứng dụng thực tiễn quan trọng khác của lý thuyết đồ thị cũng được chú trọng giải quyết đó là Bài toán tô màu đồ thị, Bài toán tìm đường đi ngắn nhất và Bài toán luồng cực đại trong mạng.

Trong mỗi phần của tài liệu, chúng tôi cố gắng trình bày ngắn gọn trực tiếp vào bản chất của vấn đề, đồng thời cài đặt hầu hết các thuật toán bằng ngôn ngữ lập trình C nhằm đạt được hai mục tiêu chính cho người học: Nâng cao tư duy toán học trong phân tích, thiết kế thuật toán và rèn luyện kỹ năng lập trình với những thuật toán phức tạp. Mặc dù đã rất cẩn trọng trong quá trình biên soạn, tuy nhiên tài liệu không tránh khỏi những thiếu sót và hạn chế. Chúng tôi rất mong được sự góp ý quí báu của tất cả đọc giả và các bạn đồng nghiệp.

Hà nội, tháng 11 năm 2013

MỤC LỤC

CHƯƠNG 1. MỘT SỐ KHÁI NIỆM CƠ BẢN CỦA ĐỒ THỊ

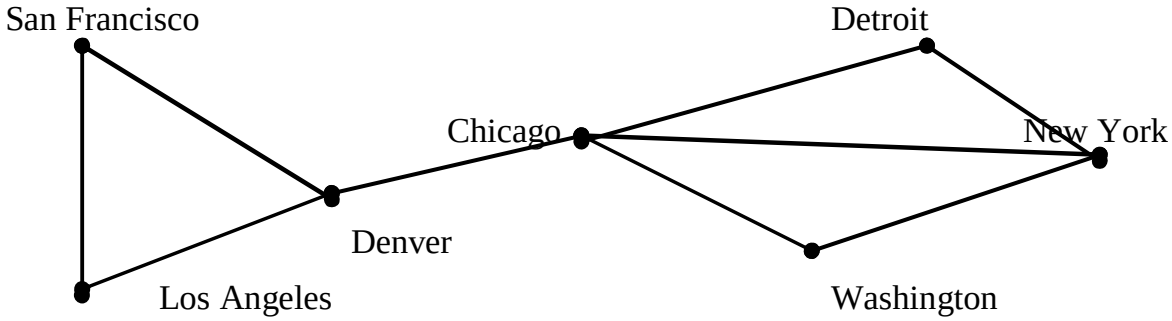
Nội dung chính của chương này đề cập đến những khái niệm cơ bản nhất của đồ thị, bao gồm:

- ✓ Định nghĩa và ví dụ.
- ✓ Phân loại đồ thị vô hướng, đồ thị có hướng, đơn đồ thị, đa đồ thị.
- ✓ Khái niệm về bậc và bán bậc của đỉnh.
- ✓ Khái niệm về đường đi, chu trình và tính liên thông của đồ thị.
- ✓ Bài tập.

Bạn đọc có thể tìm thấy những kiến thức sâu hơn và rộng hơn trong các tài liệu [1], [2], [3].

1.1. Định nghĩa và khái niệm

Đồ thị (Graph) là một cấu trúc dữ liệu rời rạc bao gồm các đỉnh và các cạnh nối các cặp đỉnh này. Chúng ta phân biệt đồ thị thông qua kiểu và số lượng cạnh và hướng của mỗi cạnh nối giữa các cặp đỉnh của đồ thị. Để minh chứng cho các loại đồ thị, chúng ta xem xét một số ví dụ về các loại mạng máy tính bao gồm: mỗi máy tính là một đỉnh, mỗi cạnh là những kênh điện thoại được nối giữa hai máy tính với nhau. Hình 1.1, là sơ đồ của mạng máy tính loại 1.

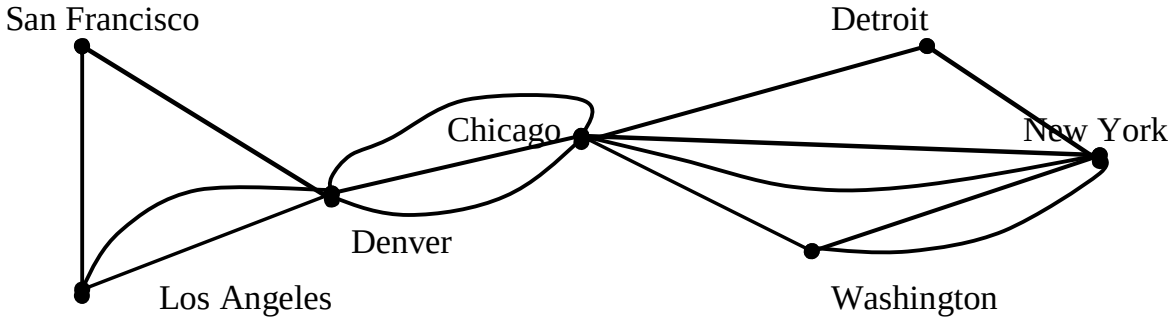


Hình 1.1. *Đơn đồ thị vô hướng.*

Trong mạng máy tính này, mỗi máy tính là một đỉnh của đồ thị, mỗi cạnh vô hướng biểu diễn các đỉnh nối hai đỉnh phân biệt, không có hai cặp đỉnh nào nối cùng một cặp đỉnh. Mạng loại này có thể biểu diễn bằng một **đơn đồ thị vô hướng**.

Định nghĩa 1. *Đơn đồ thị vô hướng G = <V, E> bao gồm V là tập các đỉnh, E là tập các cặp không có thứ tự gồm hai phần tử khác nhau của V gọi là các cạnh.*

Trong trường hợp giữa hai máy tính nào đó thường xuyên truyền tải nhiều thông tin, người ta nối hai máy tính bởi nhiều kênh thoại khác nhau. Mạng máy tính đa kênh thoại có thể được biểu diễn như Hình 1.2.



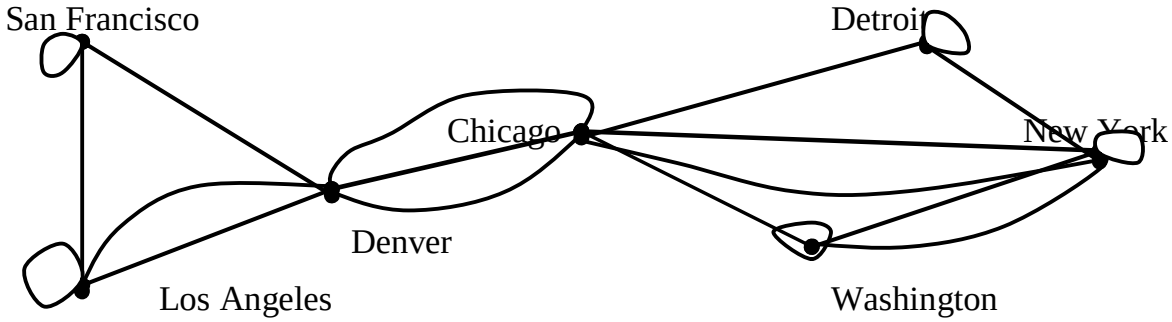
Hình 1.2. *Đa đồ thị vô hướng*.

Trên Hình 1.2, giữa hai máy tính có thể được nối với nhau bởi nhiều hơn một kênh thoại. Với mạng loại này, chúng ta không thể dùng đơn đồ thị vô hướng để biểu diễn. Đồ thị loại này là đa đồ thị vô hướng.

Định nghĩa 2. Đa đồ thị vô hướng G = <V, E> bao gồm V là tập các đỉnh, E là họ các cặp không có thứ tự gồm hai phần tử khác nhau của V gọi là tập các cạnh. e1∈E, e2∈E được gọi là cạnh bội nếu chúng cùng tương ứng với một cặp đỉnh.

Rõ ràng, mọi đơn đồ thị đều là đa đồ thị, nhưng không phải đa đồ thị nào cũng là đơn đồ thị vì giữa hai đỉnh có thể có nhiều hơn một cạnh nối giữa chúng với nhau. Trong nhiều trường hợp, có máy tính có thể nối nhiều kênh thoại với chính nó. Với loại mạng này, ta không thể dùng đa đồ thị để biểu diễn mà phải dùng giả đồ thị vô hướng. Giả đồ thị vô hướng được mô tả như trong Hình 1.3.

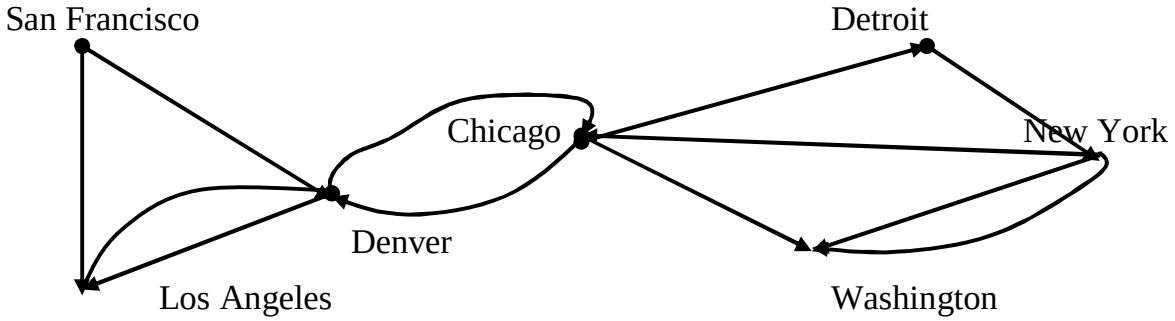
Định nghĩa 3. Giả đồ thị vô hướng G = <V, E> bao gồm V là tập đỉnh, E là họ các cặp không có thứ tự gồm hai phần tử (hai phần tử không nhất thiết phải khác nhau) trong V được gọi là các cạnh. Cạnh e được gọi là khuyên nếu có dạng e =(u, u), trong đó u là đỉnh nào đó thuộc V.



Hình 1.3. *Giả đồ thị vô hướng*.

Trong nhiều mạng, các kênh thoại nối giữa hai máy tính có thể chỉ được phép truyền tin theo một chiều. Chẳng hạn máy tính đặt tại San Francisco được phép truy nhập tới máy tính đặt tại Los Angeles, nhưng máy tính đặt tại Los Angeles không được phép

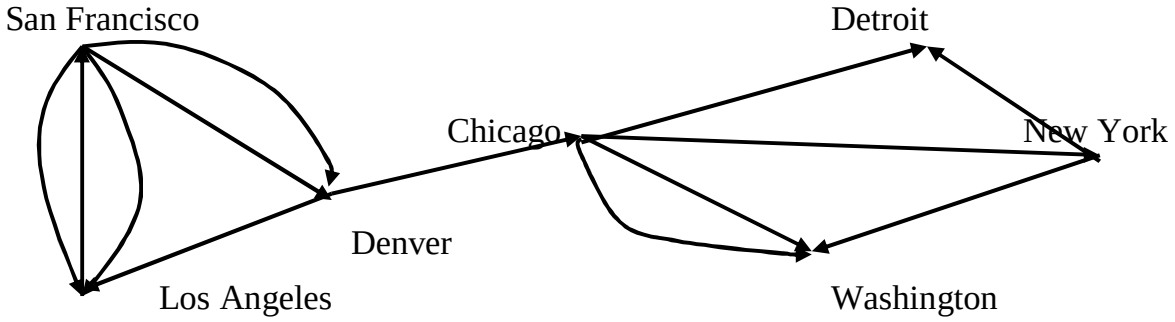
truy nhập ngược lại San Francisco. Hoặc máy tính đặt tại Denver có thể truy nhập được tới máy tính đặt tại Chicago và ngược lại máy tính đặt tại Chicago cũng có thể truy nhập ngược lại máy tính tại Denver. Để mô tả mạng loại này, chúng ta dùng khái niệm đơn đồ thị có hướng. Đơn đồ thị có hướng được mô tả như trong Hình 1.4.



Hình 1.4. Đơn đồ thị có hướng.

Định nghĩa 4. *Đơn đồ thị có hướng G = <V, E> bao gồm V là tập các đỉnh, E là tập các cặp có thứ tự gồm hai phần tử của V gọi là các cung.*

Đồ thị có hướng trong Hình 1.4 không chứa các cạnh bội. Nên đối với các mạng đa kênh thoại một chiều, đồ thị có hướng không thể mô tả được mà ta dùng khái niệm đa đồ thị có hướng. Mạng có dạng đa đồ thị có hướng được mô tả như trong Hình 1.5.



Hình 5.5. Đa đồ thị có hướng.

Định nghĩa 5. *Đa đồ thị có hướng G = <V, E> bao gồm V là tập đỉnh, E là cặp có thứ tự gồm hai phần tử của V được gọi là các cung. Hai cung e_1, e_2 tương ứng với cùng một cặp đỉnh được gọi là cung lặp.*

Từ những dạng khác nhau của đồ thị kể trên, chúng ta thấy sự khác nhau giữa các loại đồ thị được phân biệt thông qua các cạnh của đồ thị có thứ tự hay không có thứ tự, các cạnh bội, khuyên có được dùng hay không. Ta có thể tổng kết các loại đồ thị thông qua Bảng 1.

Bảng 1. Phân biệt các loại đồ thị

Loại đồ thị	Cạnh	Có cạnh bội	Có khuyên
1. Đơn đồ thị vô hướng	Vô hướng	Không	Không
2. Đa đồ thị vô hướng	Vô hướng	Có	Không
3. Giả đồ thị vô hướng	Vô hướng	Có	Có

4. Đơn đồ thị có hướng	Có hướng	Không	Không
5. Đa đồ thị có hướng	Có hướng	Có	Có

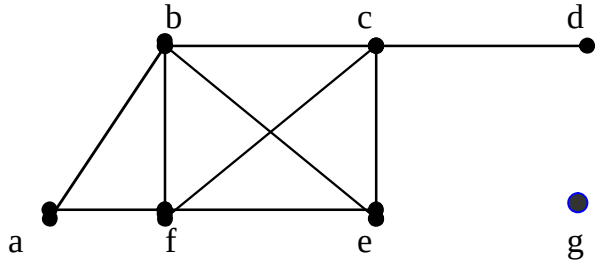
1.2. Một số thuật ngữ cơ bản trên đồ thị vô hướng

Cho đồ thị vô hướng G = <V,E>, trong đó V là tập đỉnh, E là tập cạnh. Ta bắt đầu làm quen với một số khái niệm cơ bản dưới đây.

1.2.1. Bậc của đỉnh

Định nghĩa 1. Hai đỉnh *u* và *v* của đồ thị vô hướng *G* =<*V*, *E*> được gọi là kề nhau nếu (*u,v*) là cạnh thuộc đồ thị *G*. Nếu *e* =(*u, v*) là cạnh của đồ thị *G* thì ta nói cạnh này liên thuộc với hai đỉnh *u* và *v*, hoặc ta nói cạnh *e* nối đỉnh *u* với đỉnh *v*, đồng thời các đỉnh *u* và *v* sẽ được gọi là đỉnh đầu của cạnh (*u,v*).

Định nghĩa 2. Ta gọi bậc của đỉnh *v* trong đồ thị vô hướng là số cạnh liên thuộc với nó và ký hiệu là *deg*(*v*).



Hình 1.6 Đồ thị vô hướng G.

Ví dụ 1. Xét đồ thị trong Hình 1.6, ta có:

deg(a) = 2, deg(b) =deg(c) = deg(f) = 4;

deg(e) = 3, deg(d) = 1, deg(g)=0.

Đỉnh có bậc *0* được gọi là đỉnh cô lập. Đỉnh bậc *1* được gọi là đỉnh treo. Vì vậy :

- Đỉnh ***g*** là đỉnh cô lập của đồ thị
- Đỉnh ***d*** là đỉnh treo của đồ thị.

Định lý 1. Giả sử *G* = <*V*, *E*> là đồ thị vô hướng với m cạnh. Khi đó $2m = \sum_{v \in V} \deg(v)$.

Chứng minh. Rõ ràng mỗi cạnh *e=(u,v)* bất kỳ, được tính một lần trong *deg(u)* và một lần trong *deg(v)*. Từ đó suy ra số tổng tất cả các bậc bằng hai lần số cạnh.

Hệ quả. Trong đồ thị vô hướng *G*=<*V, E*>, số các đỉnh bậc lẻ là một số chẵn.

Chứng minh. Gọi *O* là tập các đỉnh bậc chẵn và *V* là tập các đỉnh bậc lẻ. Từ định lý *1* ta suy ra:

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

Do *deg(v)* là chẵn với *v* là đỉnh trong ***O*** nên tổng thứ hai trong vế phải cũng là một số chẵn.

1.2.2. Đường đi, chu trình, đồ thị liên thông

Định nghĩa 1. Đường đi độ dài *n* từ đỉnh *u* đến đỉnh *v* trên đồ thị vô hướng $G=<V,E>$ là dãy $x_0, x_1, \ldots, x_{n-1}, x_n$, trong đó *n* là số nguyên dương, $x_0=u, x_n=v, (x_i, x_{i+1}) \in E, i = 0, 1, 2, \ldots, n-1$.

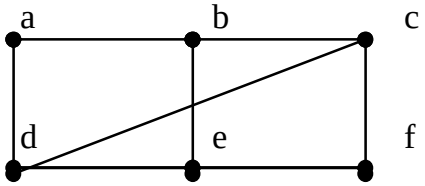
Đường đi như trên còn có thể biểu diễn thành dãy các cạnh

$$(x_0, x_1), (x_1,x_2) , \ldots, (x_{n-1}, x_n).$$

Đỉnh *u* là đỉnh đầu, đỉnh *v* là đỉnh cuối của đường đi. Đường đi có đỉnh đầu trùng với đỉnh cuối *(u=v)* được gọi là chu trình. Đường đi hay chu trình được gọi là đơn nếu như không có cạnh nào lặp lại.

Ví dụ 1. Tìm các đường đi, chu trình trong đồ thị vô hướng như trong Hình 1.7.

a, d, c, f, e là đường đi đơn độ dài *4*. *d, e, c, a* không là đường đi vì *(e,c)* không phải là cạnh của đồ thị. Dãy *b, c, f, e, b* là chu trình độ dài *4*. Đường đi *a, b, e, d, a, b* có độ dài 5 không phải là đường đi đơn vì cạnh *(a,b)* có mặt hai lần.



Hình 1.7. Đường đi trên đồ thị.

Định nghĩa 2. Đồ thị vô hướng được gọi là liên thông nếu luôn tìm được đường đi giữa hai đỉnh bất kỳ của nó.

Trong trường hợp đồ thị G=<V, E> không liên thông, ta có thể phân rã G thành một số đồ thị con liên thông mà chúng đôi một không có đỉnh chung. Mỗi đồ thị con như vậy được gọi là một thành phần liên thông của G. Như vậy, đồ thị liên thông khi và chỉ khi số thành phần liên thông của nó là 1.

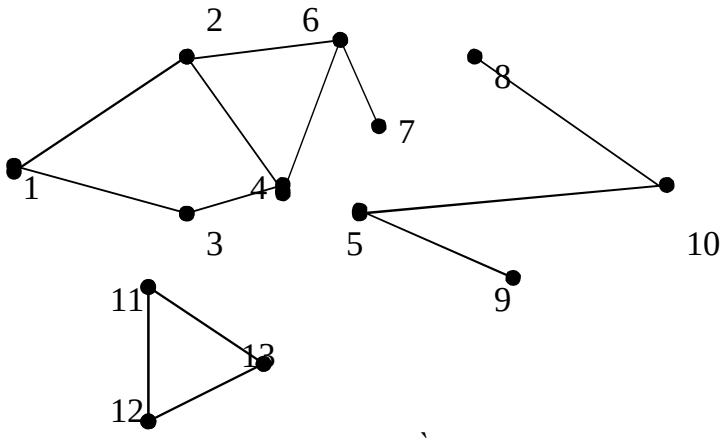
Đối với đồ thị vô hướng, đường đi từ đỉnh u đến đỉnh v cũng giống như đường đi từ đỉnh v đến đỉnh u. Chính vì vậy, nếu tồn tại đỉnh $u \in V$ sao cho u có đường đi đến tất cả các đỉnh còn lại của đồ thị thì ta kết luận được đồ thị là liên thông.

Ví dụ 2. Tìm các thành phần liên thông của đồ thị Hình 1.8 dưới đây.

Số thành phần liên thông của G là 3. Thành phần liên thông thứ nhất gồm các đỉnh 1, 2, 3, 4, 6, 7. Thành phần liên thông thứ hai gồm các đỉnh 5, 8, 9, 10. Thành phần liên thông thứ ba gồm các đỉnh 11, 12, 13.

Định nghĩa 3. Cạnh $e \in E$ được gọi là cầu nếu loại bỏ e làm tăng thành phần liên thông của đồ thị. Đỉnh $u \in V$ được gọi là đỉnh trụ nếu loại bỏ u cùng với các cạnh nối với u làm tăng thành phần liên thông của đồ thị.

Ví dụ 3. Tìm các cạnh cầu và đỉnh trụ của đồ thị Hình 1.8.



Hình 1.8. *Đồ thị vô hướng G*

Lời giải.

- Cạnh (5, 9) là cầu vì nếu loại bỏ (5, 9) thì số thành phần liên thông của đồ thị tăng từ 3 lên 4.
- Cạnh (5, 10) là cầu vì nếu loại bỏ (5, 10) thì số thành phần liên thông của đồ thị tăng từ 3 lên 4.
- Cạnh (6, 7) là cầu vì nếu loại bỏ (6, 7) thì số thành phần liên thông của đồ thị tăng từ 3 lên 4.
- Cạnh (8, 10) là cầu vì nếu loại bỏ (8, 10) thì số thành phần liên thông của đồ thị tăng từ 3 lên 4.
- Các cạnh còn lại không là cầu vì nếu loại bỏ cạnh không làm tăng thành phần liên thông của đồ thị.
- Đỉnh 5 là đỉnh trụ vì nếu loại bỏ đỉnh 5 cùng với các cạnh nối với đỉnh 5 số thành phần liên thông của đồ thị tăng từ 3 lên 4.
- Đỉnh 6 là đỉnh trụ vì nếu loại bỏ đỉnh 6 cùng với các cạnh nối với đỉnh 6 số thành phần liên thông của đồ thị tăng từ 3 lên 4.
- Đỉnh 10 là đỉnh trụ vì nếu loại bỏ đỉnh 10 cùng với các cạnh nối với đỉnh 10 số thành phần liên thông của đồ thị tăng từ 3 lên 4.
- Các đỉnh còn lại không là trụ vì nếu loại bỏ đỉnh cùng với các cạnh nối với đỉnh không làm tăng thành phần liên thông của đồ thị.

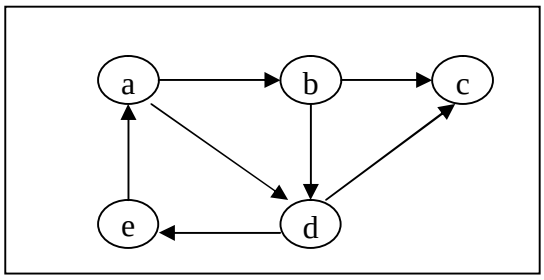
1.3. Một số thuật ngữ cơ bản trên đồ thị có hướng

Cho đồ thị có hướng G = <V,E>, trong đó V là tập đỉnh, E là tập cạnh. Ta bắt đầu làm quen với một số khái niệm cơ bản dưới đây.

1.3.1. Bán bậc của đỉnh

Định nghĩa 1. Nếu $e=(u,v)$ là cung của đồ thị có hướng G thì ta nói hai đỉnh u và v là kề nhau, và nói cung (u, v) nối đỉnh u với đỉnh v, hoặc nói cung này đi ra khỏi đỉnh u và đi vào đỉnh v. Đỉnh u được gọi là đỉnh đầu, đỉnh v được gọi là đỉnh cuối của cung (u,v).

Định nghĩa 2. Ta gọi bán bậc ra của đỉnh v trên đồ thị có hướng là số cung của đồ thị đi ra khỏi v và ký hiệu là $deg^+(v)$. Ta gọi bán bậc vào của đỉnh v trên đồ thị có hướng là số cung của đồ thị đi vào v và ký hiệu là $deg^-(v)$.



Hình 1.9. Đồ thị có hướng G.

Ví dụ 2. Xét đồ thị có hướng trong Hình 1.10, ta có

- $\deg^+(a) = 2$, $\deg^+(b) = 2$, $\deg^+(c) = 0$, $\deg^+(d) = 1$, $\deg^+(e) = 1$.
- $\deg^-(a) = 1$, $\deg^-(b) = 1$, $\deg^-(c) = 2$, $\deg^-(d) = 2$, $\deg^-(e) = 1$.

Do mỗi cung (u,v) được tính một lần trong bán bậc vào của đỉnh v và một lần trong bán bậc ra của đỉnh u nên ta có:

Định lý 1. Giả sử $G = <V, E>$ là đồ thị có hướng. Khi đó $\sum_{v\in V} \deg^+(v) = \sum_{v\in V} \deg^-(v) = |E|$.

Rất nhiều tính chất của đồ thị có hướng không phụ thuộc vào hướng trên các cung của nó. Vì vậy, trong nhiều trường hợp, ta bỏ qua các hướng trên cung của đồ thị. Đồ thị vô hướng nhận được bằng cách bỏ qua hướng trên các cung được gọi là đồ thị vô hướng tương ứng với đồ thị có hướng đã cho.

1.3.2. Đồ thị có hướng liên thông mạnh, liên thông yếu

Khái niệm đường đi và chu trình trên đồ thị có hướng được định nghĩa hoàn toàn tương tự, chỉ có điều khác biệt duy nhất là ta phải chú ý tới các cung của đồ thị.

Định nghĩa 1. Đường đi độ dài *n* từ đỉnh *u* đến đỉnh *v* trong đồ thị có hướng $G=<V,A>$ là dãy $x_0, x_1, \ldots, x_n$, trong đó, *n* là số nguyên dương, $u = x_0$, $v = x_n$, $(x_i, x_{i+1}) \in E$.

Đường đi như trên có thể biểu diễn thành dãy các cung :

$$(x_0, x_1), (x_1, x_2), \ldots, (x_{n-1}, x_n).$$

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 ($u=v$) được gọi là một chu trình. Đường đi hay chu trình được gọi là đơn nếu như không có hai cạnh nào lặp lại.

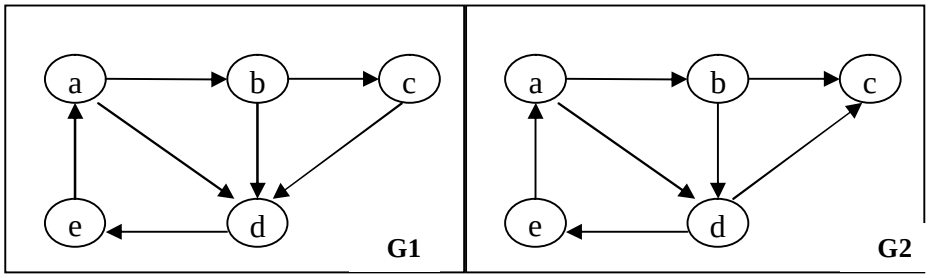
Đối với đồ thị vô hướng, đường đi từ đỉnh u đến đỉnh v cũng giống như đường đi từ đỉnh v đến đỉnh u. Đối với đồ thị có hướng, đường đi từ đỉnh u đến đỉnh v có thể không phải là đường đi từ v đến u. Chính vì vậy, đồ thị vô hướng đưa ra hai khái niệm liên thông mạnh và liên thông yếu như sau.

Định nghĩa 2. Đồ thị có hướng G=<V,E> được gọi là liên thông mạnh nếu giữa hai đỉnh bất kỳ $u \in V$, $v \in V$ đều có đường đi từ u đến v.

Như vậy, để chứng tỏ một đồ thị có hướng liên thông mạnh ta cần chứng tỏ mọi cặp đỉnh của đồ thị đều có đường đi đến nhau. Điều này hoàn toàn khác biệt với tính liên thông của đồ thị vô hướng.

Định nghĩa 3. Ta gọi đồ thị vô hướng tương ứng với đồ thị có hướng G=<V,E> là đồ thị tạo bởi G và bỏ hướng của các cạnh trong G. Khi đó, đồ thị có hướng G=<V,E> được gọi là liên thông yếu nếu đồ thị vô hướng tương ứng với nó là liên thông.

Ví dụ 1. Hình 1.10: Đồ thị G1 là liên thông mạnh, đồ thị G2 là liên thông yếu.



Hình 1.10. *Đồ thị có hướng liên thông mạnh, liên thông yếu*

Định nghĩa 4. Đồ thị vô hướng G=<V,E> được gọi là định chiều được nếu ta có thể biến đổi các cạnh trong G thành các cung tương ứng để nhận được một đồ thị có hướng liên thông mạnh.

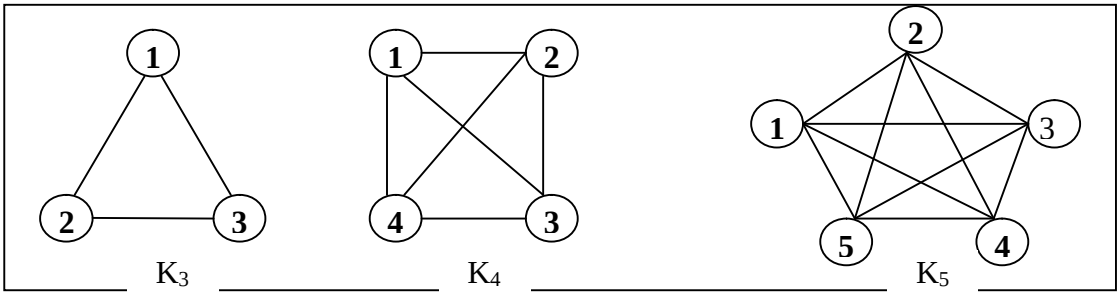
Định lý 1. Đồ thị vô hướng G=<V,E> định chiều được khi và chỉ khi các cạnh của nó không phải là cầu.

Bạn đọc có thể tìm hiểu phần chứng minh định lý trong các tài liệu [1, 2, 3].

1.4. Một số dạng đồ thị đặc biệt

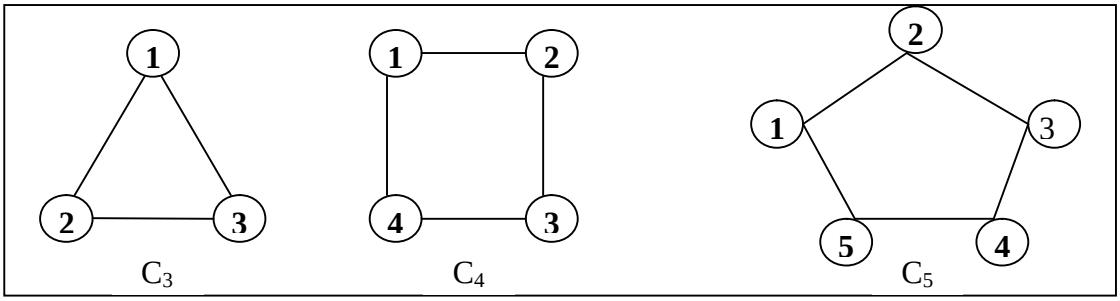
Dưới đây là một số dang đơn đồ thị vô hướng đặc biệt có nhiều ứng dụng khác nhau của thực tế.

Đồ thị đầy đủ. Đồ thị đầy đủ n đỉnh, ký hiệu là K_n, là đơn đồ thị vô hướng mà giữa hai đỉnh bất kỳ của nó đều có cạnh nối. Ví dụ đồ thị K_3, K_4, K_5 trong Hình 1.11.



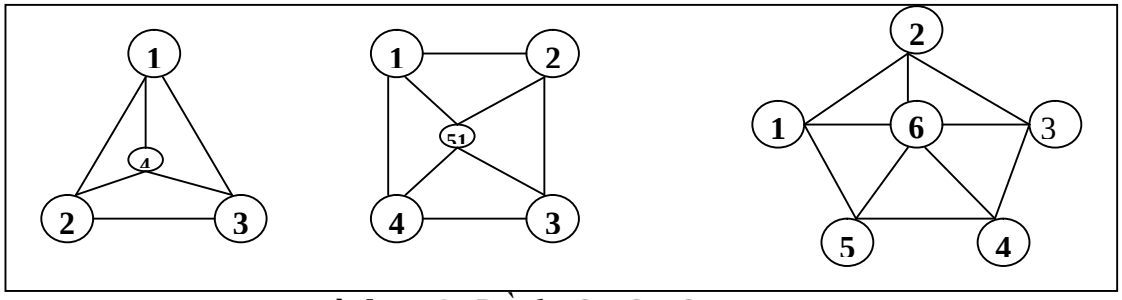
Hình 1.11. *Đồ thị K_3, K_4, K_5.*

Đồ thị vòng. Đồ thị vòng C_n ($n \geq 3$) có các cạnh (*1*,*2*), (*2*,*3*),..,(*n-1*,*n*), (*n*,*1*). Ví dụ đồ thị C_3, C_4, C_5 trong Hình 1.12.



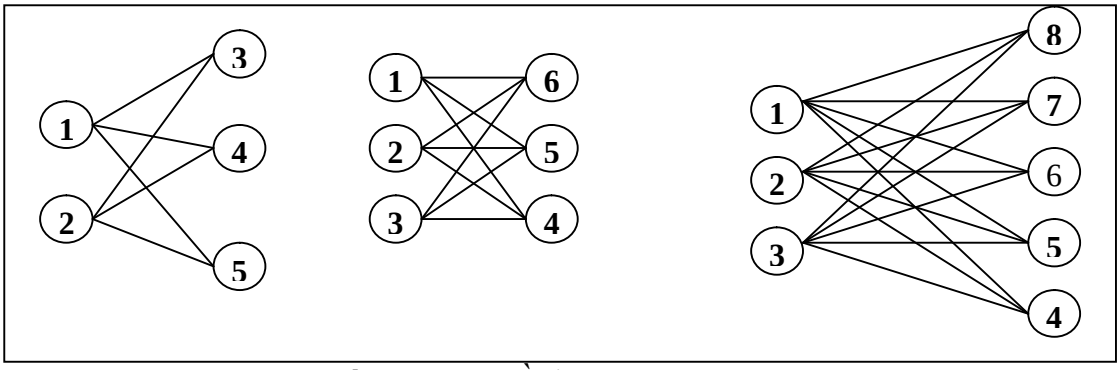
Hình 1.12. *Đồ thị C_3, C_4, C_5.*

Đồ thị bánh xe. Đồ thị bánh xe W_n thu được bằng cách bổ sung một đỉnh nối với tất cả các đỉnh của C_n. Ví dụ đồ thị W_3, W_4, W_5 trong Hình 1.13.



Hình 1.13. *Đồ thị C_3, C_4, C_5.*

Đồ thị hai phía. Đồ thị G =<V,E> được gọi là đồ thị hai phía nếu tập đỉnh V của nó có thể phân hoạch thành hai tập X và Y sao cho mỗi cạnh của đồ thị chỉ có dạng (*x*, *y*), trong đó $x \in X$ và $y \in Y$. Ví dụ đồ thị $K_{2,3}$, K_{33}, $K_{3,5}$ trong Hình 1.14.



Hình 1.13. *Đồ thị $K_{2,3}$, $K_{3,3}$, $K_{3,5}$.*

1.5. Những điểm cần ghi nhớ

- ✓ Nắm vững và phân biệt rõ các loại đồ thị: đơn đồ thị, đa đồ thị, đồ thị vô hướng, đồ thị có hướng, đồ thị trọng số.
- ✓ Nắm vững những khái niệm cơ bản trên đồ thị vô hướng.
- ✓ Nắm vững những khái niệm cơ bản trên đồ thị có hướng.về đồ thị.
- ✓ Nắm vững các khái niệm đường đi, chu trình, liên thông, liên thông mạnh, liên thông yếu.
- ✓ Nắm vững các loại đồ thị : đồ thị đầy đủ, đồ thị vòng, đồ thị bánh xe, đồ thị hai phía...

CHƯƠNG II. BIỂU DIỄN ĐỒ THỊ TRÊN MÁY TÍNH

Để lưu trữ đồ thị và thực hiện các thuật toán khác nhau, ta cần phải biểu diễn đồ thị trên máy tính, đồng thời sử dụng những cấu trúc dữ liệu thích hợp để mô tả đồ thị. Việc chọn cấu trúc dữ liệu nào để biểu diễn đồ thị có tác động rất lớn đến hiệu quả thuật toán. Vì vậy, lựa chọn cấu trúc dữ liệu thích hợp biểu diễn đồ thị sẽ phụ thuộc vào từng bài toán cụ thể. Nội dung chính của chương bao gồm:

- ✓ Biểu diễn đồ thị bằng ma trận kề.
- ✓ Biểu diễn đồ thị bằng danh sách cạnh.
- ✓ Biểu diễn đồ thị bằng danh sách kề.
- ✓ Biểu diễn đồ thị bằng ma trận liên thuộc.
- ✓ Bài tập Chương 2.

Bạn đọc có thể tìm thấy những kiến thức sâu hơn và rộng hơn trong các tài liệu [1], [2], [3].

2.1.Biểu diễn đồ thị bằng ma trận kề

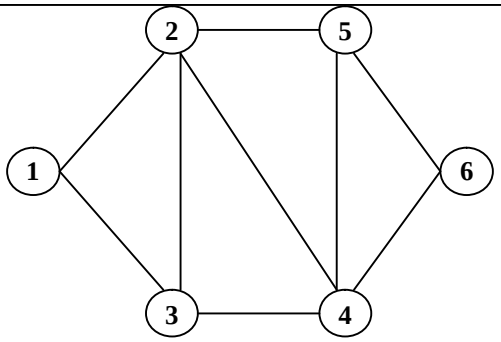
Cấu trúc dữ liệu phổ dụng nhất để biểu diễn đồ thị là biểu diễn đồ thị bằng ma trận. Về lý thuyết, người ta đã chứng minh được mỗi ma trận vuông (0,1) cấp n đều đẳng cấu với một đơn đồ thị vô hướng hoặc có hướng. Mục này, chúng ta sẽ xem xét phương pháp biểu diễn các loại đồ thị khác nhau bằng ma trận kề.

2.1.1. Ma trận kề của đồ thị vô hướng

Xét đồ thị đơn vô hướng $G = <V, E>$, với tập đỉnh $V = \{1, 2, \ldots, n\}$, tập cạnh $E = \{e_1, e_2, .., e_m\}$. Ta gọi ma trận kề của đồ thị G là ma trận có các phần tử hoặc bằng 0 hoặc bằng 1 theo qui định như sau:

$A = \{ a_{ij}: a_{ij} = 1$ nếu $(i, j) \in E, a_{ij} = 0$ nếu $(i,j) \notin E; i, j = 1, 2, \ldots, n\}$.

Ví dụ 1. Biểu diễn đồ thị trong Hình 2.1 dưới đây bằng ma trận kề.



0	1	0	0	0	0
1	0	1	1	1	0
1	1	0	1	0	0
0	1	1	0	1	1
0	1	0	1	0	1
0	0	1	1	0	0

Hình 2.1. *Ma trận kề biểu diễn đồ thị vô hướng*.

Tính chất ma trận kề đối với đồ thị vô hướng:

a) Tổng các phần tử của ma trận bằng hai lần số cạnh : $\sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij} = 2m$ (m là số cạnh của đồ thị.

b) Tổng các phần tử của hàng u là bậc của đỉnh u: $\deg(u) = \sum_{j=1}^{n} a_{uj}$. Ví dụ với ma trận kề biểu diễn đồ thị Hình 2.1, tổng các phần tử của hàng 1 là bậc của đỉnh 1, vì vậy deg(1)=2; tổng các phần tử của hàng 2 là bậc của đỉnh 2, vì vậy deg(2)=3.

c) Tổng các phần tử của cột u là bậc của đỉnh u: $\deg(u) = \sum_{j=1}^{n} a_{ju}$. Ví dụ với ma trận kề biểu diễn đồ thị Hình 2.1, tổng các phần tử của cột 1 là bậc của đỉnh 1, vì vậy deg(1)=2; tổng các phần tử của cột 2 là bậc của đỉnh 2, vì vậy deg(2)=3.

d) Nếu ký hiệu $a_{ij}^{p}, i, j = 1,2,...,n$ là các phần tử của ma trận. Khi đó,

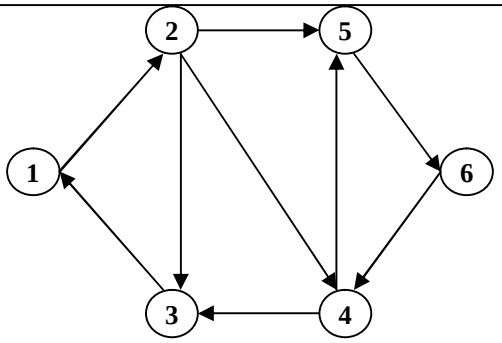
$A^p = A.A. . . A$ *(p lần);* $a_{ij}^{p}, i, j = 1,2,...,n,$

cho ta số đường đi khác nhau từ đỉnh i đến đỉnh j qua $p-1$ đỉnh trung gian.

2.1.2. Ma trận kề của đồ thị có hướng

Ma trận kề của đồ thị có hướng cũng được định nghĩa hoàn toàn tương tự, chúng ta chỉ cần lưu ý tới hướng của cạnh. Ma trận kề của đồ thị có hướng là không đối xứng.

Ví dụ 2. Tìm ma trận kề của đồ thị có hướng trong Hình 2.2.



$$\begin{vmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{vmatrix}$$

Hình 2.2. *Ma trận kề của đồ thị có hướng.*

Tính chất của ma trận kề của đồ thị có hướng:

a) Tổng các phần tử của ma trận bằng số cạnh : $\sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij} = m$ (m là số cạnh của đồ thị.

b) Tổng các phần tử của hàng *u* là bán đỉnh bậc ra của đỉnh *u*: $\deg^+(u) = \sum_{j=1}^{n} a_{uj}$.

Ví dụ với ma trận kề biểu diễn đồ thị Hình 2.2, tổng các phần tử của hàng 1 là bán đỉnh bậc a của đỉnh 1, vì vậy $\deg^+(1)=1$; tổng các phần tử của hàng 2 là bán đỉnh bậc ra của đỉnh 3, vì vậy $\deg^+(2)=3$.

c) Tổng các phần tử của cột *u* là bán đỉnh bậc vào của đỉnh *u*: $\deg^-(u) = \sum_{j=1}^{n} a_{ju}$.

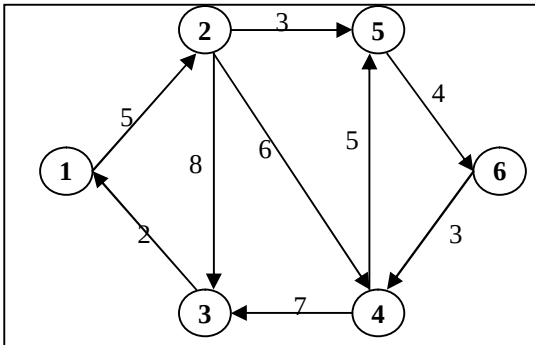
Ví dụ với ma trận kề biểu diễn đồ thị Hình 2.2, tổng các phần tử cột 1 là bán đỉnh bậc vào của đỉnh 1, vì vậy $\deg^-(1)=1$; tổng các phần tử của cột 2 là bán đỉnh bậc vào của đỉnh 2, vì vậy $\deg^-(2)=1$.

d) Nếu ký hiệu $a_{ij}^p, i, j = 1,2,...,n$ là các phần tử của ma trận. Khi đó, $A^p = A.A. . .$ *A (p lần);* $a_{ij}^p, i, j = 1,2,...,n$, cho ta số đường đi khác nhau từ đỉnh *i* đến đỉnh *j* qua *p-1* đỉnh trung gian.

2.1.3. Ma trận trọng số

Trong rất nhiều ứng dụng khác nhau của lý thuyết đồ thị, mỗi cạnh *e =(u,v)* của nó được gán bởi một số *c(e) = c(u,v)* gọi là trọng số của cạnh *e*. Đồ thị trong trường hợp như vậy gọi là đồ thị trọng số. Trong trường hợp đó, ma trận kề của đồ thị được thay bởi ma trận trọng số *c= c[i,j], i, j= 1, 2, . . ., n. c[i,j] = c(i,j)* nếu *(i, j)* $\in E$, *c[i,j] =* θ nếu *(i, j)* $\notin E$. Trong đó, θ nhận các giá trị: *0,* ∞, $-\infty$ tuỳ theo từng tình huống cụ thể của thuật toán.

Ví dụ 3. Ma trận kề của đồ thị có trọng số trong Hình 2.3.



∞	5	∞	∞	∞	∞
∞	∞	8	6	3	∞
2	∞	∞	∞	∞	∞
∞	∞	7	∞	5	∞
∞	∞	∞	∞	∞	4
∞	∞	∞	3	∞	∞

Hình 2.3. *Ma trận kề của đồ thị có hướng.*

Ưu điểm của ma trận kề:

- Đơn giản dễ cài đặt trên máy tính bằng cách sử dụng một mảng hai chiều để biểu diễn ma trận kề;
- Dễ dàng kiểm tra được hai đỉnh *u, v* có kề với nhau hay không bằng đúng một phép so sánh (a[u][v]≠0?);và chúng ta chỉ mất đúng một phép so sánh.

Nhược điểm của ma trận kề:

- Lãng phí bộ nhớ: bất kể số cạnh nhiều hay ít ta cần n2 đơn vị bộ nhớ để biểu diễn;
- Không thể biểu diễn được với các đồ thị có số đỉnh lớn (ví dụ triệu đỉnh);
- Để xem xét đỉnh đỉnh *u* có những đỉnh kề nào cần mất n phép so sánh kể cả đỉnh u là đỉnh cô lập hoặc đỉnh treo.

2.1.4. Qui ước khuôn dạng lưu trữ ma trận kề

Để thuận tiện cho những nội dung kế tiếp, ta qui ước khuôn dạng dữ liệu biểu diễn đồ thị dưới dạng ma trận kề hoặc ma trận trọng số trong file như sau:

- Dòng đầu tiên ghi lại số đỉnh của đồ thị;
- N dòng kế tiếp ghi lại ma trận kề của đồ thị. Hai phần tử khác nhau của ma trận kề được viết cách nhau một vài khoảng trống.

Ví dụ ma trận kề gồm 6 đỉnh của Hình 2.1 được tổ chức trong file dothi.in như sau:

```
dothi.in
5
0   1   0   0   0   0
1   0   1   1   1   0
1   1   0   1   0   0
0   1   0   1   0   1
0   0   1   1   0   0
```

2.2. Biểu diễn đồ thị bằng danh sách cạnh (cung)

Trong trường hợp đồ thị thưa (đồ thị có số cạnh $m \leq 6n$), người ta thường biểu diễn đồ thị dưới dạng danh sách cạnh. Trong phép biểu diễn này, chúng ta sẽ lưu trữ danh sách tất cả các cạnh (cung) của đồ thị vô hướng (có hướng). Mỗi cạnh (cung) *e(x, y)* được tương ứng với hai biến *dau*[*e*], *cuoi*[*e*]. Như vậy, để lưu trữ đồ thị, ta cần *2m* đơn vị bộ nhớ. Nhược điểm lớn nhất của phương pháp này là để nhận biết những cạnh nào kề với cạnh nào chúng ta cần *m* phép so sánh trong khi duyệt qua tất cả *m* cạnh (cung) của đồ thị. Nếu là đồ thị có trọng số, ta cần thêm *m* đơn vị bộ nhớ để lưu trữ trọng số của các cạnh.

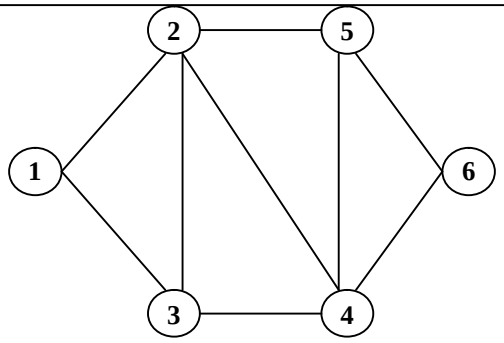
2.2.1. Biểu diễn đồ thị vô hướng bằng danh sách cạnh

Đối với đồ thị vô hướng, mỗi cạnh là bộ không tính đến thứ tự các đỉnh. Ví dụ cạnh (u,v) và cạnh (v, u) được xem là một. Do vậy, trong khi biểu diễn đồ thị vô hướng bằng danh sách cạnh ta chỉ cần liệt kê các cạnh (u,v) mà không cần liệt kê cạnh (v,u). Để tránh nhầm lẫn, ta nên liệt kê các cạnh theo thứ tự tăng dần của đỉnh đầu mỗi cạnh. Trong

trường hợp biểu diễn đa đồ thị vô hướng, ta bổ sung thêm một cột là số cạnh (socanh) nối giữa hai đỉnh của đồ thị. Hình 2.4 dưới đây mô tả chi tiết phương pháp biểu diễn đồ thị vô hướng bằng danh sách cạnh.

Tính chất danh sách cạnh của đồ thị vô hướng:

- Đỉnh đầu nhỏ hơn đỉnh cuối mỗi cạnh.
- Số đỉnh có giá trị u thuộc cả vế phải và vế trái của danh sách cạnh là bậc của đỉnh u. Ví dụ giá trị u=1 xuất hiện 2 lần từ đó ta suy ra deg(1)=2, số 2 xuất hiện 4 lần vì vậy deg(2) = 4.

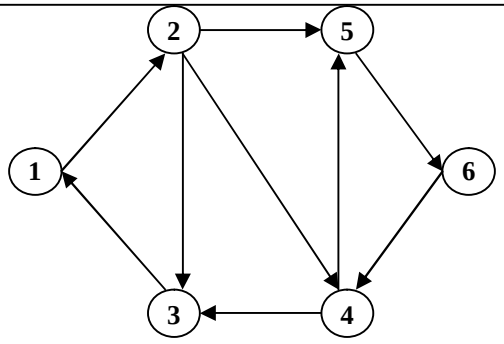


Đỉnh đầu	Đỉnh cuối
1	2
1	3
2	3
2	4
2	5
3	4
4	5
4	6
5	6

Hình 2.4. *Biểu diễn đồ thị vô hướng bằng danh sách cạnh.*

2.2.2. Biểu diễn đồ thị có hướng bằng danh sách cạnh

Trong trường hợp đồ thị có hướng, mỗi cạnh là bộ có tính đến thứ tự các đỉnh. Ví dụ cạnh (u,v) khác với cạnh (v, u). Do vậy, trong khi biểu diễn đồ thị vô hướng bằng danh sách cạnh ta đặc biệt chú ý đến hướng của các cạnh. Hình 2.5 dưới đây mô tả chi tiết phương pháp biểu diễn đồ thị có hướng bằng danh sách cạnh.



Đỉnh đầu	Đỉnh Cuối
1	2
2	3
2	4
2	5
3	1
4	3
4	5
5	6
6	4

Hình 2.5. *Biểu diễn đồ thị có hướng bằng danh sách cạnh.*

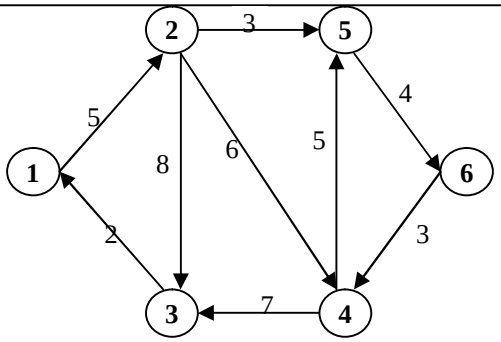
Tính chất danh sách cạnh của đồ thị vô hướng:

- Đỉnh đầu không nhất thiết phải nhỏ hơn đỉnh cuối mỗi cạnh.

- Số đỉnh có giá trị u thuộc cả vế phải các cạnh là $deg^+(u)$. Ví dụ giá trị u=1 xuất hiện 1 lần ở vế phải của tất cả các cạnh nên $deg^+(1)$ =1, giá trị u=2 xuất hiện 3 lần ở vế phải của tất cả các cạnh nên $deg^+(2)$ =3.
- Số đỉnh có giá trị u thuộc cả vế trái các cạnh là $deg^-(u)$. Ví dụ giá trị u=1 xuất hiện 1 lần ở vế trái của tất cả các cạnh nên $deg^-(1)$ =1, giá trị u=2 xuất hiện 1 lần ở vế trái của tất cả các cạnh nên $deg^-(2)$ =1.

2.2.3. Biểu diễn đồ thị trọng số bằng danh sách cạnh

Trong trường hợp đồ thị có hướng (hoặc vô hướng) có trọng số, ta bổ sung thêm một cột là trọng số của mỗi cạnh. Hình 2.6 dưới đây mô tả chi tiết phương pháp biểu diễn đồ thị trọng số bằng danh sách cạnh.



Đỉnh đầu	Đỉnh Cuối	Trọng Số
1	2	5
2	3	8
2	4	6
2	5	3
3	1	2
4	3	7
4	5	5
5	6	4
6	4	3

Hình 2.6. *Biểu diễn đồ thị có hướng bằng danh sách cạnh.*

Ưu điểm của danh sách cạnh:

- Trong trường hợp đồ thị thưa (m<6n), biểu diễn bằng danh sách cạnh tiết kiệm được không gian nhớ;
- Thuận lợi cho một số thuật toán chỉ quan tâm đến các cạnh của đồ thị.

Nhược điểm của danh sách cạnh:

- Khi cần duyệt các đỉnh kề với đỉnh u bắt buộc phải duyệt tất cả các cạnh của đồ thị. Điều này làm cho thuật toán có chi phí tính toán cao.

2.2.4. Qui ước khuôn dạng lưu trữ danh sách cạnh

Để thuận tiện cho những nội dung kế tiếp, ta qui ước khuôn dạng dữ liệu biểu diễn đồ thị dưới dạng danh sách cạnh trong file như sau:

- Dòng đầu tiên ghi lại số N, M tương ứng với số đỉnh và số cạnh của đồ thị. Hai số được viết cánh nhau một vài khoảng trống;
- M dòng kế tiếp, mỗi dòng gi lại một cạnh của đồ thị, đỉnh đầu và đỉnh cuối mỗi cạnh được viết cách nhau một vài khoảng trống.

Ví dụ với đồ thị trọng số cho bởi Hình 2.6 gồm 6 đỉnh và 9 cạnh được lưu trữ trong file dothi.in như sau:

dothi.in

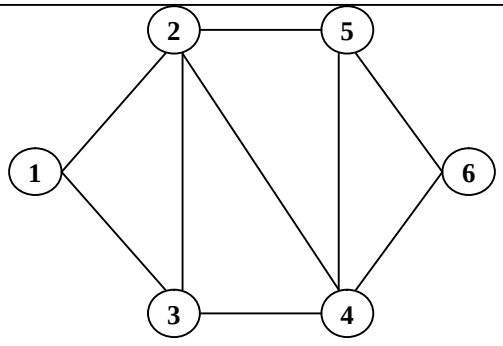
```
6   9
1   2   5
2   3   8
2   4   6
2   5   3
3   1   2
4   3   7
4   5   5
5   6   4
6   4   3
```

2.2.5. Cấu trúc dữ liệu biểu diễn danh sách cạnh

Phương pháp tốt hơn cả để biểu diễn mỗi cạnh của đồ thị là sử dụng cấu trúc. Mỗi cấu trúc gồm có hai thành viên *dau*[*e*] và cuối *cuoi*[*e*]. Khi đó, danh sách cạnh của đồ thị dễ dàng được biểu diễn bằng mảng hoặc danh sách liên kết như dưới đây.

Biểu diễn danh danh sách cạnh của đồ thị bằng mảng:

```
typedef struct {  //Định nghĩa một cạnh của đồ thị
    int   dau;
    int   cuoi;
} Edge;
Edge G[MAX]; //Danh sách các cạnh được biểu diễn trong mảng G.
```



Đỉnh đầu	Đỉnh cuối
1	2
1	3
2	3
2	4
2	5
3	4
4	5
4	6
5	6

Hình 2.7. *Biểu diễn đồ thị bằng danh sách cạnh*

Ví dụ với danh danh sách cạnh của đồ thị Hình 2.7, biểu diễn danh sách cạnh dựa vào mảng của đồ thị có dạng sau:

Cạnh:	G[1]	G[2]	G[3]	G[4]	G[5]	G[6]	G[7]	G[8]	G[9]
G[i].dau	1	1	2	2	2	3	4	4	5
G[i].cuoi	2	3	3	4	5	4	5	6	6

Đối với đồ thị có hướng cũng được biểu diễn như trên nhưng ta cần chú ý đến hướng của mỗi cung. Đối với đồ thị trọng số ta chỉ cần bổ sung vào cấu trúc Edge một thành viên là trọng số của cạnh như sau:

```
typedef struct {  //Định nghĩa một cạnh có trọng số của đồ thị
        int     dau;
        int     cuoi;
        int     trongso;
} Edge;
Edge G[MAX]; //Danh sách trọng số các cạnh biểu diễn trong mảng G.
```

Biểu diễn danh danh sách cạnh của đồ thị bằng danh sách liên kết:

```
typedef struct canh{  //Định nghĩa một cạnh của đồ thị
        int     dau;
        int     cuoi;
        struct node *next;
} *Edge;
Edge *G; //Các cạnh được của đồ thị biểu diễn bằng danh danh sách liên kết G.
```

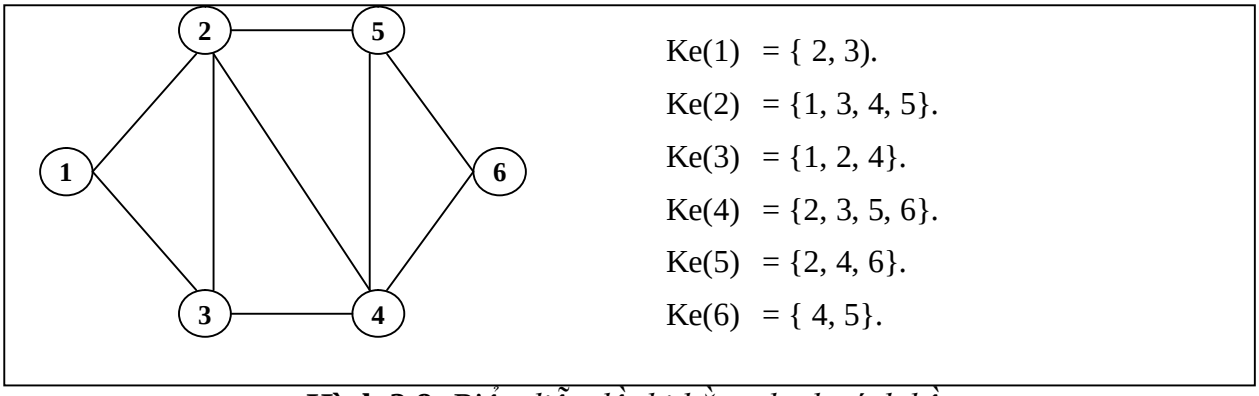
Ví dụ với danh danh sách cạnh của đồ thị Hình 2.7, biểu diễn danh sách cạnh dựa vào danh sách liên kết có dạng sau:

2.3. Biểu diễn đồ thị bằng danh sách kề

Trong rất nhiều ứng dụng, cách biểu diễn đồ thị dưới dạng danh sách kề thường được sử dụng. Trong biểu diễn này, với mỗi đỉnh *u* của đồ thị chúng ta lưu trữ danh sách các đỉnh kề với nó mà ta ký hiệu là *Ke(u)*, nghĩa là

$$Ke(u) = \{ v \in V: (u, v) \in E\},$$

Với cách biểu diễn này, mỗi đỉnh *u* của đồ thị, ta làm tương ứng với một danh sách tất cả các đỉnh kề với nó và được ký hiệu là *List*(*u*). Để biểu diễn *List*(*u*), ta có thể dùng các kiểu dữ liệu kiểu tập hợp, mảng hoặc danh sách liên kết. Hình 2.8 dưới đây đưa ra ví dụ chi tiết về biểu diễn đồ thị bằng danh sách kề.



Hình 2.8. *Biểu diễn đồ thị bằng danh sách kề.*

Ưu điểm của danh sách kề:

- Dễ dàng duyệt tất cả các đỉnh của một danh sách kề;
- Dễ dàng duyệt các cạnh của đồ thị trong mỗi danh sách kề;
- Tối ưu về phương pháp biểu diễn.

Nhược điểm của danh sách kề:

- Khó khăn cho người đọc có kỹ năng lập trình yếu.

2.3.1. Biểu diễn danh sách kề dựa vào mảng

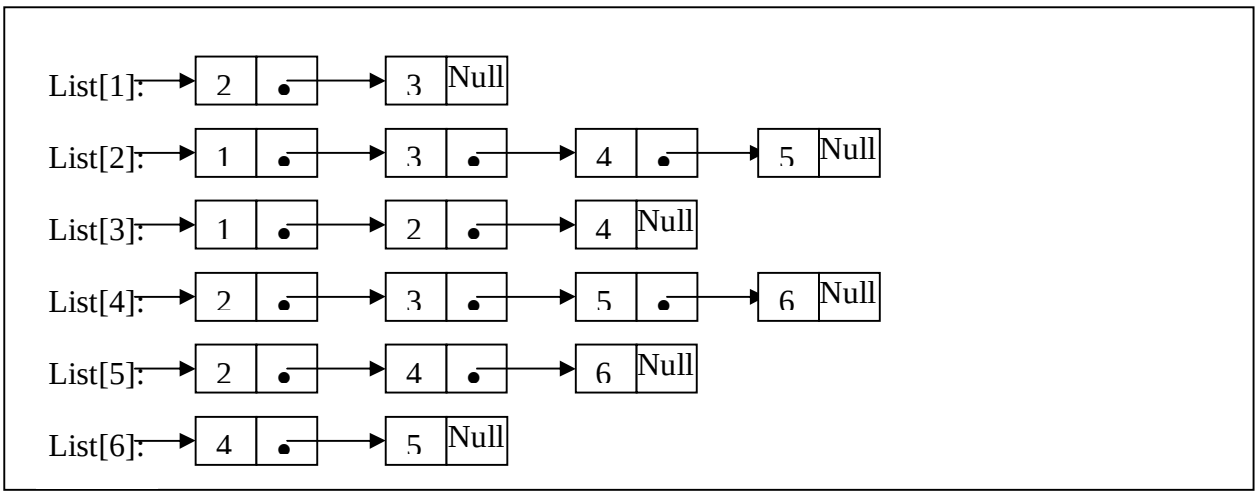
Sử dụng một mảng để lưu trữ danh sách kề các đỉnh. Trong đó, mảng được chia thành *n* đoạn, đoạn thứ *i* trong mảng lưu trữ danh sách kề của đỉnh thứ $i \in V$. Ví dụ với đồ thị được cho trong Hình 2.8 ta tổ chức mảng A[] gồm 18 phần tử, trong đó mảng A[] được chia thành 6 đoạn, mỗi đoạn lưu trữ danh sách kề của đỉnh tương ứng như dưới đây.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
A[i]=?	2	3	1	3	4	5	1	2	4	2	3	5	6	2	4	6	4	5
	Đoạn 1		Đoạn 2				Đoạn 3			Đoạn 4				Đoạn 5			Đoạn 6	

Để biết một đoạn thuộc mảng bắt đầu từ phần tử nào đến phần tử nào ta sử dụng một mảng khác dùng để lưu trữ vị trí các phần tử bắt đầu và kết thúc của đoạn. Ví dụ với danh sách kề gồm 6 đoạn như trên, ta cần xây dựng một mảng VT[6] = {0, 2, 6, 9, 13, 16, 18} để lưu trữ vị trí các đoạn trong mảng A[]. Dựa vào mảng VT[] ta có thể thấy: Ke(1) là A[1], A[2]; Ke(2) là A[3], A[4], A[5], A[6] .. ..

2.3.2. Biểu diễn danh sách kề bằng danh sách liên kết

Với mỗi đỉnh $u \in V$, ta biểu diễn mỗi danh sách kề của đỉnh bằng một danh sách liên kết List(u). Ví dụ với đồ thị trong Hình 2.8 sẽ được biểu diễn bằng 6 danh sách liên kết List[1], List[2],.., List[6] như dưới đây.



2.3.3. Qui ước khuôn dạng lưu trữ danh sách kề:

- Dòng đầu tiên ghi lại số đỉnh của đồ thị;
- N dòng kế tiếp ghi lại danh sách kề của đỉnh tương ứng theo khuôn dạng: Phần tử đầu tiên là vị trí kết thúc của đoạn, tiếp đến là danh sách các đỉnh của danh sách kề. Các phần tử được ghi cách nhau một vài khoảng trống

Ví dụ khuôn dạng lưu trữ danh sách kề của Hình 2.7 trong file dothi.in như sau:

dothi.in

```
6
2     2     3
6     1     3     4     5
9     1     2     4
13    2     3     5     6
16    2     4     6
18    4     5
```

2.4. Những điểm cần ghi nhớ

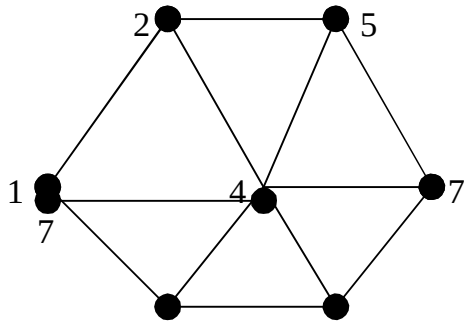
- ✓ Nắm vững và phân biệt rõ các loại đồ thị: đơn đồ thị, đa đồ thị, đồ thị vô hướng, đồ thị có hướng, đồ thị trọng số.
- ✓ Nắm vững những khái niệm cơ bản về đồ thị: đường đi, chu trình, đồ thị liên thông.
- ✓ Hiểu và nắm rõ bản chất của các phương pháp biểu diễn đồ thị trên máy tính. Phân tích ưu, nhược điểm của từng phương pháp biểu diễn.
- ✓ Chuyển đổi các phương pháp biểu diễn qua lại lẫn nhau giúp ta hiểu được cách biểu diễn đồ thị trên máy tính.

BÀI TẬP

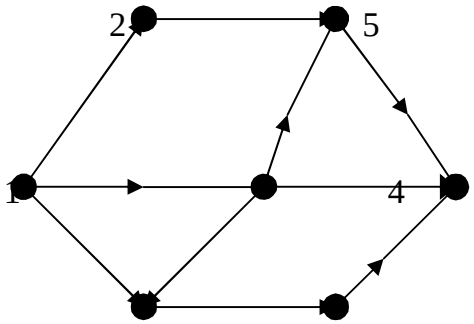
1. Trong một buổi gặp mặt, mọi người đều bắt tay nhau. Hãy chỉ ra rằng số lượt người bắt tay nhau là một số chẵn.

2. Một đơn đồ thị với n đỉnh có nhiều nhất là bao nhiêu cạnh?

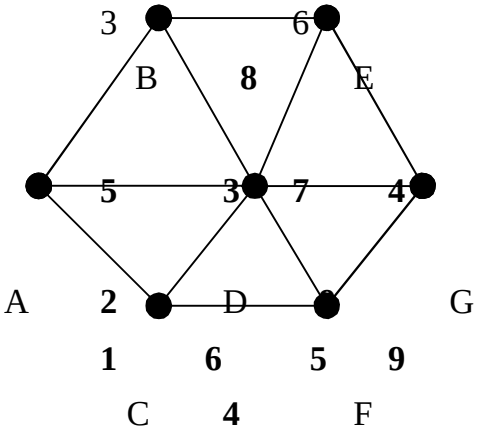
3. Hãy biểu diễn các đồ thị G1, G2, G3 dưới đây dưới dạng: ma trận kề, danh sách cạnh, danh sách kề.



a. Đồ thị vô hướng G1.



b. Đồ thị có hướng G2.



c. Đồ thị trọng số G3

4. Hãy tạo một file dữ liệu theo khuôn dạng như sau:

a. Ma trận kề:

- Dòng đầu tiên là số tự nhiên n là số các đỉnh của đồ thị.
- N dòng kế tiếp là ma trận kề của đồ thị.

b. Danh sách cạnh:

- Dòng đầu tiên ghi lại số tự nhiên n và m là số các đỉnh và các cạnh của đồ thị.
- M dòng kế tiếp ghi lại thứ tự đỉnh đầu, cuối của các cạnh.

Hãy viết chương trình chuyển đổi một đồ thị cho dưới dạng ma trận kề thành một đồ thị cho dưới dạng danh sách cạnh và danh sách kề. Ngược lại, chuyển đổi một đồ thị cho dưới dạng danh sách cạnh thành đồ thị dưới dạng ma trận kề và danh sách cạnh.

c. Danh sách kề:

- Dòng đầu tiên ghi lại số đỉnh của đồ thị;
- N dòng kế tiếp ghi lại danh sách kề của đỉnh tương ứng theo khuôn dạng: Phần tử đầu tiên là vị trí kết thúc của đoạn, tiếp đến là danh sách các đỉnh của danh sách kề. Các phần tử được ghi cách nhau một vài khoảng trống.
- M dòng kế tiếp ghi lại thứ tự đỉnh đầu, cuối của các cạnh.

Hãy viết chương trình chuyển đổi một đồ thị cho dưới dạng ma trận kề thành một đồ thị cho dưới dạng danh sách cạnh và danh sách kề. Ngược lại, chuyển đổi một đồ thị cho dưới dạng danh sách cạnh thành đồ thị dưới dạng ma trận kề và danh sách cạnh.

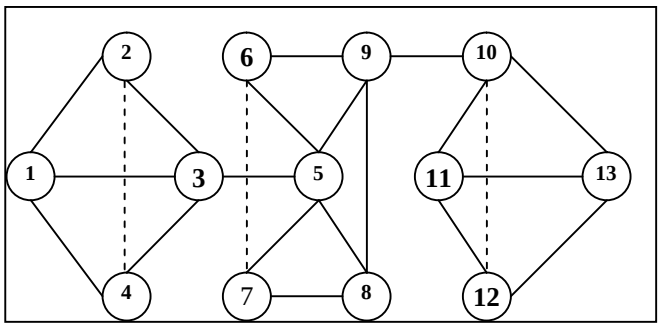
5. Một bàn cờ 8×8 được đánh số theo cách sau:

1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16
17	18	19	20	21	22	23	24
25	26	27	28	29	30	31	32
33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48
49	50	51	52	53	54	55	56
57	58	59	60	61	62	63	64

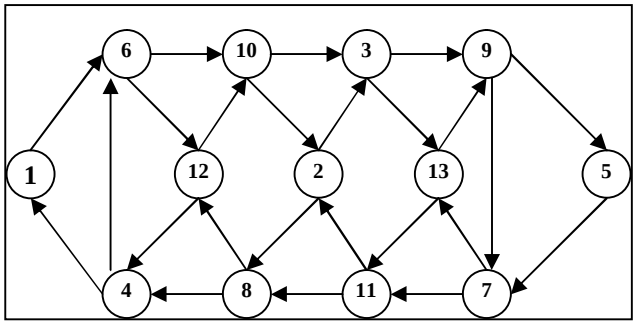
Mỗi ô có thể coi là một đỉnh của đồ thị. Hai đỉnh được coi là kề nhau nếu một con vua đặt ở ô này có thể nhảy sang ô kia sau một bước đi. Ví dụ : ô 1 kề với ô 2, 9, 10, ô 11 kề với 2, 3, 4, 10, 12, 18, 19, 20. Hãy viết chương trình tạo ma trận kề của đồ thị, kết quả in ra file king.out.

6. Bàn cờ 8×8 được đánh số như bài trên. Mỗi ô có thể coi là một đỉnh của đồ thị . Hai đỉnh được gọi là kề nhau nếu một con mã đặt ở ô này có thể nhảy sang ô kia sau một nước đi. Ví dụ ô 1 kề với 11, 18, ô 11 kề với 1, 5, 17, 21, 26, 28. Hãy viết chương trình lập ma trận kề của đồ thị, kết quả ghi vào file matran.out.

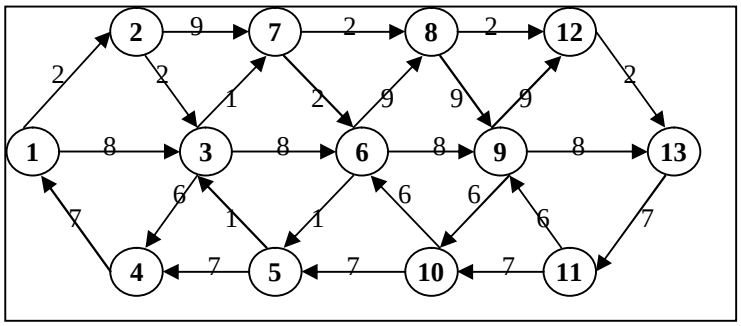
7. Hãy biểu diễn đồ thị dưới đây dưới dạng ma trận kề, danh sách cạnh, danh sách kề.



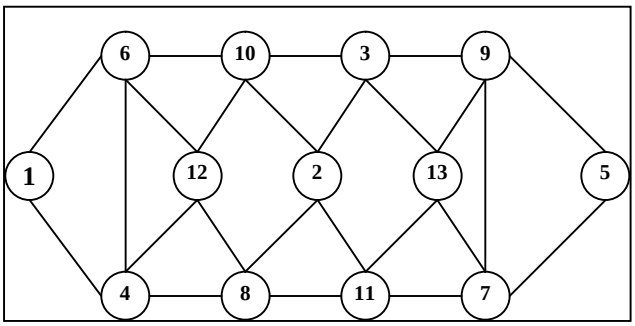
8. Hãy biểu diễn đồ thị dưới đây dưới dạng ma trận kề, danh sách cạnh, danh sách kề.



9. Hãy biểu diễn đồ thị dưới đây dưới dạng ma trận trọng số, danh sách cạnh - trọng số.



10. Hãy biểu diễn đồ thị dưới đây dưới dạng ma trận kề, danh sách cạnh, danh sách kề.



CHƯƠNG 3. TÌM KIẾM TRÊN ĐỒ THỊ

Có nhiều thuật toán trên đồ thị được xây dựng để duyệt tất cả các đỉnh của đồ thị sao cho mỗi đỉnh được viếng thăm đúng một lần. Những thuật toán như vậy được gọi là thuật toán tìm kiếm trên đồ thị. Chúng ta cũng sẽ làm quen với hai thuật toán tìm kiếm cơ bản, đó là duyệt theo chiều sâu DFS (Depth First Search) và duyệt theo chiều rộng BFS (Breath First Search). Trên cơ sở của hai phép duyệt cơ bản, ta có thể áp dụng chúng để giải quyết một số bài toán quan trọng của lý thuyết đồ thị. Nội dung chính được đề cập trong chương này bao gồm:

- ✓ Thuật toán tìm kiếm theo chiều sâu trên đồ thị.
- ✓ Thuật toán tìm kiếm theo chiều rộng trên đồ thị.
- ✓ Ứng dụng của thuật toán tìm kiếm theo chiều sâu.
- ✓ Ứng dụng của thuật toán tìm kiếm theo chiều rộng.

Bạn đọc có thể tìm hiểu sâu hơn về tính đúng đắn, độ phức tạp của các thuật toán trong các tài liệu [1, 2, 3].

3.1. Thuật toán tìm kiếm theo chiều sâu (Depth First Search)

Tư tưởng cơ bản của thuật toán tìm kiếm theo chiều sâu là bắt đầu tại một đỉnh v_0 nào đó, chọn một đỉnh *u* bất kỳ kề với v_0 và lấy nó làm đỉnh duyệt tiếp theo. Cách duyệt tiếp theo được thực hiện tương tự như đối với đỉnh v_0 với đỉnh bắt đầu là *u*.

Để kiểm tra việc duyệt mỗi đỉnh đúng một lần, chúng ta sử dụng một mảng *chuaxet[]* gồm *n* phần tử (tương ứng với *n* đỉnh), nếu đỉnh thứ *u* đã được duyệt, phần tử tương ứng trong mảng *chuaxet[u]* có giá trị *FALSE*. Ngược lại, nếu đỉnh chưa được duyệt, phần tử tương ứng trong mảng có giá trị *TRUE*.

3.1.1.Biểu diễn thuật toán DFS(u)

Thuật toán DFS(u) có thể được mô tả bằng thủ tục đệ qui như sau:

```
Thuật toán DFS (u): //u là đỉnh bắt đầu duyệt
Begin
        <Thăm đỉnh u>;//Duyệt đỉnh u
        chuaxet[u] := FALSE;//Xác nhận đỉnh u đã duyệt
        for   each v ∈ke(u) do //Lấy mỗi đỉnh v∈Ke(u).
                if  (chuaxet[v] )  then //Nếu đỉnh v chưa duyệt
                        DFS(v); //Duyệt theo chiều sâu bắt từ đỉnh v
                EndIf;
        EndFor;
End.
```

Thuật toán DFS(u) có thể khử đệ qui bằng cách sử dụng ngăn xếp như Hình 3.1 dưới đây:

```
Thuật toán DFS(u):
Begin
        Bước 1 (Khởi tạo):
                stack = ∅; //Khởi tạo stack là ∅
                Push(stack, u); //Đưa đỉnh u vào ngăn xếp
                <Thăm đỉnh u>; //Duyệt đỉnh u
                chuaxet[u] = False; //Xác nhận đỉnh u đã duyệt
        Bước 2 (Lặp) :
                while ( stack ≠ ∅ ) do
                        s = Pop(stack); //Loại đỉnh ở đầu ngăn xếp
                        for each t∈ Ke(s) do //Lấy mỗi đỉnh t∈Ke(s)
                                if ( chuaxet[t] ) then //Nếu t đúng là chưa duyệt
                                        <Thăm đỉnh t>; // Duyệt đỉnh t
                                        chuaxet[t] = False; // Xác nhận đỉnh t đã duyệt
                                        Push(stack, s);//Đưa s vào stack
                                        Push(stack, t); //Đưa t vào stack
                                        break;          //Chỉ lấy một đỉnh t
                                EndIf;
                        EndFor;
                EndWhile;
        Bước 3 (Trả lại kết quả):
                Return(<Tập đỉnh đã duyệt>);
End.
```

Hình 3.1. *Thuật toán DFS*(*u*) *dựa vào ngăn xếp*.

3.1.2. Độ phức tạp thuật toán

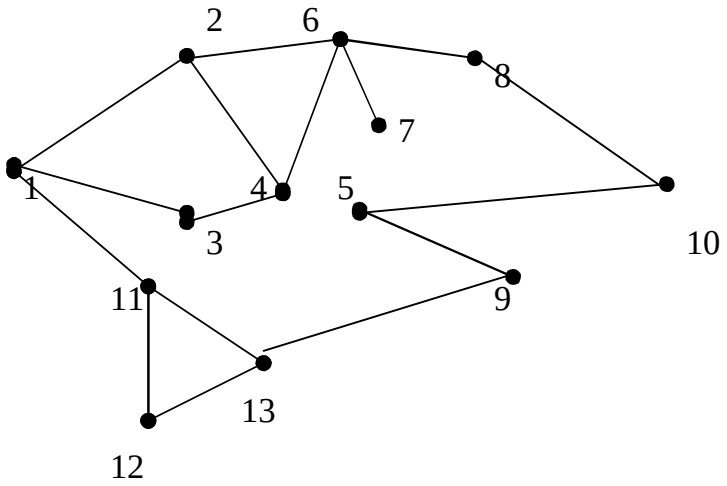
Độ phức tạp thuật toán DFS(u) phụ thuộc vào phương pháp biểu diễn đồ thị. Độ phức tạp thuật toán DFS(u) theo các dạng biểu diễn đồ thị như sau:

- Độ phức tạp thuật toán là O(n^2) trong trường hợp đồ thị biểu diễn dưới dạng ma trận kề, với *n* là số đỉnh của đồ thị.
- Độ phức tạp thuật toán là O(*n.m*) trong trường hợp đồ thị biểu diễn dưới dạng danh sách cạnh, với *n* là số đỉnh của đồ thị, *m* là số cạnh của đồ thị.
- Độ phức tạp thuật toán là O(max(*n, m*)) trong trường hợp đồ thị biểu diễn dưới dạng danh sách kề, với *n* là số đỉnh của đồ thị, *m* là số cạnh của đồ thị.

Bạn đọc tự chứng minh hoặc có thể tham khảo trong các tài liệu [1, 2, 3].

3.1.3. Kiểm nghiệm thuật toán

Ví dụ 1. Kiểm nghiệm thuật toán DFS(1) trên đồ thị gồm 13 đỉnh trong Hình 3.2 dưới đây?



Hình 3.2. *Đồ thị vô hướng G.*

Đỉnh bắt đầu duyệt	**Các đỉnh đã duyệt: chuaxet[u]=False**	**Các đỉnh chưa duyệt chuaxet[u]=True**
DFS(1)	1	2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13
DFS(2)	1, 2	3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13
DFS(4)	1, 2, 4	3, 5, 6, 7, 8, 9, 10, 11, 12, 13
DFS(3)	1,2,4, 3	5, 6, 7, 8, 9, 10, 11, 12, 13
DFS(6)	1,2,4,3, 6	5, 7, 8, 9, 10, 11, 12, 13
DFS(7)	1,2,4,3, 6,7	5, 8, 9, 10, 11, 12, 13
DFS(8)	1,2,4,3, 6,7,8	5, 9, 10, 11, 12, 13
DFS(10)	1,2,4,3, 6,7,8,10	5, 9, 11, 12, 13
DFS(5)	1,2,4,3, 6,7,8,10,5	9, 11, 12, 13
DFS(9)	1,2,4,3, 6,7,8,10,5,9	11, 12, 13
DFS(13)	1,2,4,3, 6,7,8,10,5,9,13	11, 12
DFS(11)	1,2,4,3, 6,7,8,10,5,9,13,11	12
DFS(11)	1,2,4,3, 6,7,8,10,5,9,13,11,12	ϕ

Kết quả duyệt: ***1, 2, 4, 3, 6, 7, 8, 10, 5, 9, 13, 11, 12***

Để bạn đọc làm quen với phương pháp kiểm nghiệm thuật toán dựa vào dữ liệu, chúng tôi sử dụng biểu diễn của đồ thị bằng ma trận kề như đã được trình bày trong Chương 2. Việc kiểm nghiệm thuật toán bằng các biểu diễn khác (danh sách cạnh, danh sách kề) xem như những bài tập để bạn đọc tự tìm ra lời giải.

Ví dụ 2. Cho đồ thị gồm 13 đỉnh được biểu diễn dưới dạng ma trận kề như hình bên phải. Hãy cho biết kết quả thực hiện thuật toán trong Hình 3.1 bắt đầu tại đỉnh *u=1*? Chỉ rõ trạng thái của ngăn xếp và tập đỉnh được duyệt theo mỗi bước thực hiện của thuật toán?

0	1	1	1	0	0	0	0	0	0	0	0	0
1	0	1	1	0	1	0	0	0	0	0	0	0
1	1	0	1	1	0	0	0	0	0	0	0	0
1	1	1	0	0	0	1	0	0	0	0	0	0
0	0	1	0	0	1	1	1	0	0	0	1	0
0	1	0	0	1	0	1	0	0	0	0	1	0
0	0	0	1	1	1	0	1	0	0	0	0	0
0	0	0	0	1	0	1	0	0	0	0	1	0
0	0	0	0	0	0	0	0	0	1	1	0	1
0	0	0	0	0	0	0	0	1	0	1	1	1
0	0	0	0	0	0	0	0	1	1	0	0	1
0	0	0	0	1	1	0	1	0	1	0	0	0
0	0	0	0	0	0	0	0	1	1	1	0	0

Lời giải. Trạng thái của ngăn xếp và tập đỉnh được duyệt theo thuật toán được thể hiện trong Bảng 3.1 dưới đây.

Bảng 3.1. Kiểm nghiệm thuật toán DFS(1).

STT	Trạng thái stack	Các đỉnh được duyệt
1	1	1
2	1, 2	1, 2
3	1, 2, 3	1, 2, 3
4	1, 2, 3, 4	1, 2, 3, 4
5	1, 2, 3, 4, 7	1, 2, 3, 4, 7
6	1, 2, 3, 4, 7, 5	1, 2, 3, 4, 7, 5
7	1, 2, 3, 4, 7, 5, 6	1, 2, 3, 4, 7, 5, 6
8	1, 2, 3, 4, 7, 5, 6, 12	1, 2, 3, 4, 7, 5, 6, 12
9	1, 2, 3, 4, 7, 5, 6, 12, 8	1, 2, 3, 4, 7, 5, 6, 12, 8
10	**1, 2, 3, 4, 7, 5, 6, 12, 10**	1, 2, 3, 4, 7, 5, 6, 12, 8, 10
11	1, 2, 3, 4, 7, 5, 6, 12, 10, 9	1, 2, 3, 4, 7, 5, 6, 12, 8, 10, 9
12	1, 2, 3, 4, 7, 5, 6, 12, 10, 9, 11	1, 2, 3, 4, 7, 5, 6, 12, 8, 10, 9, 11
13	1, 2, 3, 4, 7, 5, 6, 12, 10, 9, 11, 13	1, 2, 3, 4, 7, 5, 6, 12, 8, 10, 9, 11, 13
14	∅	

Kết quả duyệt DFS(1) = { 1, 2, 3, 4, 7, 5, 6, 12, 8, 10, 9, 11, 13}.

Chú ý.

- Đối với đồ thị vô hướng, nếu DFS(u) = V ta có thể kết luận đồ thị liên thông.
- Đối với đồ thị có hướng, nếu DFS(u) = V ta có thể kết luận đồ thị liên thông yếu.

3.1.4. Cài đặt thuật toán

Thuật toán được cài đặt theo khuôn dạng dữ liệu tổ chức trong file dothi.in được qui ước như được trình bày trong Mục 2.1.3 như sau:

- Dòng đầu tiên ghi lại số đỉnh của đồ thị;
- N dòng kế tiếp ghi lại ma trận kề của đồ thị. Hai phần tử khác nhau của ma trận kề được viết cách nhau một vài khoảng trống.

Chương trình được thực hiện với các thủ tục như sau:

- Hàm Init() : đọc dữ liệu theo khuôn dạng từ file dothi.in và thiết lập mảng chuaxet[u] =True (u=1, 2,..,n).
- Hàm DFS_Dequi : Cài đặt thuật toán DFS(u) bằng đệ qui.
- Hàm DFS_Stack : Cài đặt thuật toán DFS(u) dựa vào stack.

Ví dụ với file dothi.in dưới đây với u = 3 sẽ cho ta kết quả thực hiện chương trình như sau:

dothi.in

```
10
0    1    1    1    0    0    0    0    0    0
1    0    0    1    1    0    0    0    0    0
1    0    0    1    0    1    0    0    0    0
1    1    1    0    1    1    0    0    1    0
0    1    0    1    0    0    0    1    0    0
0    0    1    1    0    0    1    0    0    0
0    0    0    0    0    1    0    0    1    1
0    0    0    0    1    0    0    0    1    1
0    0    0    1    0    0    1    1    0    1
0    0    0    0    0    0    1    1    1    0
```

DFS(3) = 3, 1, 2, 4, 5, 8, 9, 7, 6, 10.

```
#include <stdio.h>
#include <conio.h>
#include <iostream.h>
#define MAX 50
#define TRUE 1
#define FALSE 0
int A[MAX][MAX], n,chuaxet[MAX];
void Init(void){
     int i,j;FILE *fp;
     fp=fopen("DOTHI.IN","r");
     fscanf(fp,"%d",&n);
     printf("\n So dinh do thi:%d",n);
     for(i=1; i<=n; i++){
          printf("\n");chuaxet[i]=TRUE;
          for(j=1; j<=n; j++){
               fscanf(fp,"%d",&A[i][j]);
               printf("%3d",A[i][j]);
          }
     }
}
void DFS_Dequi(int u){
     int v;
     printf("%3d",u);chuaxet[u]=FALSE;
     for(v=1; v<=n; v++){
          if(A[u][v] && chuaxet[v])
               DFS_Dequi(v);
     }
}
void DFS_Stack(int u){
     int Stack[MAX], dau=1, s, t;
     Stack[dau]=u;chuaxet[u]=FALSE;
     printf("%3d",u);
     while(dau>0){
          s=Stack[dau];dau--;
          for(t =1;t<=n; t++){
               if(chuaxet[t] && A[s][t]){
                    printf("%3d",t);
                    chuaxet[t] = FALSE;
                    Stack[++dau]=s;
                    Stack[++dau]=t;break;
               }
          }
     }
}
```

```
void main(void){
     int u ;clrscr();Init();
     cout<<"\n Dinh bat dau duyet:";
     cin>>u;
     DFS_Stack(u);
     //DFS_Dequi(u);
     getch();
}
```

3.2. Thuật toán tìm kiếm theo chiều rộng (Breadth First Search)

3.2.1. Biểu diễn thuật toán

Để ý rằng, với thuật toán tìm kiếm theo chiều sâu, đỉnh thăm càng muộn sẽ trở thành đỉnh sớm được duyệt xong. Đó là kết quả tất yếu vì các đỉnh thăm được nạp vào stack trong thủ tục đệ qui. Khác với thuật toán tìm kiếm theo chiều sâu, thuật toán tìm kiếm theo chiều rộng thay thế việc sử dụng stack bằng hàng đợi (queue). Trong thủ tục này, đỉnh được nạp vào hàng đợi đầu tiên là u, các đỉnh kề với u là $(v_1, v_2, . . ., v_k)$ được nạp vào hàng đợi nếu như nó chưa được xét đến. Quá trình duyệt tiếp theo được bắt đầu từ các đỉnh còn có mặt trong hàng đợi.

Để ghi nhận trạng thái duyệt các đỉnh của đồ thị, ta cũng vẫn sử dụng mảng *chuaxet[]* gồm *n* phần tử thiết lập giá trị ban đầu là *TRUE*. Nếu đỉnh *u* của đồ thị đã được duyệt, giá trị *chuaxet[u]* sẽ nhận giá trị *FALSE*. Thuật toán dừng khi hàng đợi rỗng. Hình 3.3. dưới đây mô tả chi tiết thuật toán BFS(u).

Thuật toán BFS(u):
Bước 1(Khởi tạo):
Queue = ∅; Push(Queue,u); chuaxet[u] = False;
Bước 2 (Lặp):
while (Queue ≠ ∅) do
s = Pop(Queue); <Thăm đỉnh s>;
for each t∈Ke(s) do
if (chuaxet[t]) then
Push(Queue, t); chuaxet[t] = False;
EndIf ;
EndFor ;
EndWhile ;
Bước 3 (Trả lại kết quả) :
Return(<Tập đỉnh được duyệt>) ;
End.

Hình 3.3. *Thuật toán BFS(u)*.

3.2.2. Độ phức tạp thuật toán

Độ phức tạp thuật toán BFS(u) phụ thuộc vào phương pháp biểu diễn đồ thị. Độ phức tạp thuật toán BFS(u) theo các dạng biểu diễn đồ thị như sau:

- Độ phức tạp thuật toán là $O(n^2)$ trong trường hợp đồ thị biểu diễn dưới dạng ma trận kề, với n là số đỉnh của đồ thị.
- Độ phức tạp thuật toán là $O(n.m)$ trong trường hợp đồ thị biểu diễn dưới dạng danh sách cạnh, với n là số đỉnh của đồ thị, m là số cạnh của đồ thị.
- Độ phức tạp thuật toán là $O(\max(n, m))$ trong trường hợp đồ thị biểu diễn dưới dạng danh sách kề, với n là số đỉnh của đồ thị, m là số cạnh của đồ thị.

Bạn đọc tự chứng minh hoặc có thể tham khảo trong các tài liệu [1, 2, 3].

3.2.3. Kiểm nghiệm thuật toán

Ví dụ 3. Cho đồ thị gồm 13 đỉnh được biểu diễn dưới dạng ma trận kề như hình bên phải. Hãy cho biết kết quả thực hiện thuật toán trong Hình 3.3 bắt đầu tại đỉnh *u=1*? Chỉ rõ trạng thái của hàng đợi và tập đỉnh được duyệt theo mỗi bước thực hiện của thuật toán?

0	1	1	1	0	0	0	0	0	0	0	0	0
1	0	1	1	0	1	0	0	0	0	0	0	0
1	1	0	1	1	0	0	0	0	0	0	0	0
1	1	1	0	0	0	1	0	0	0	0	0	0
0	0	1	0	0	1	1	1	0	0	0	1	0
0	1	0	0	1	0	1	0	0	0	0	1	0
0	0	0	1	1	1	0	1	0	0	0	0	0
0	0	0	0	1	0	1	0	0	0	0	1	0
0	0	0	0	0	0	0	0	0	1	1	0	1
0	0	0	0	0	0	0	0	1	0	1	1	1
0	0	0	0	0	0	0	0	1	1	0	0	1
0	0	0	0	1	1	0	1	0	1	0	0	0
0	0	0	0	0	0	0	0	1	1	1	0	0

Lời giải. Trạng thái của hàng đợi và tập đỉnh được duyệt theo thuật toán được thể hiện trong Bảng 3.2 dưới đây.

Bảng 3.2. Kiểm nghiệm thuật toán BFS(1).

STT	Trạng thái Queue	Các đỉnh được duyệt
1	1	∅
2	2, 3, 4	1
3	3, 4, 6	1, 2
4	4, 6, 5	1, 2, 3
5	6, 5, 7	1, 2, 3, 4
6	5, 7, 12	1, 2, 3, 4, 6
7	7, 12, 8	1, 2, 3, 4, 6, 5
8	12, 8	1, 2, 3, 4, 6, 5, 7
9	8, 10	1, 2, 3, 4, 6, 5, 7, 12
10	10	1, 2, 3, 4, 6, 5, 7, 12, 8

11	9, 11, 13	1, 2, 3, 4, 6, 5, 7, 12, 8,10
12	11, 13	1, 2, 3, 4, 6, 5, 7, 12, 8,10, 9
13	13	1, 2, 3, 4, 6, 5, 7, 12, 8,10, 9, 11
14	∅	1, 2, 3, 4, 6, 5, 7, 12, 8,10, 9, 11, 13

Kết quả duyệt BFS(1) = { 1, 2, 3, 4, 6, 5, 7, 12, 8,10, 9, 11, 13}.

Chú ý.

- Đối với đồ thị vô hướng, nếu BFS(u) = V ta có thể kết luận đồ thị liên thông.
- Đối với đồ thị có hướng, nếu BFS(u) = V ta có thể kết luận đồ thị liên thông yếu.

3.2.4. Cài đặt thuật toán

Thuật toán được cài đặt theo khuôn dạng dữ liệu tổ chức trong file dothi.in được qui ước như được trình bày trong Mục 2.1.3 như sau:

- Dòng đầu tiên ghi lại số đỉnh của đồ thị;
- N dòng kế tiếp ghi lại ma trận kề của đồ thị. Hai phần tử khác nhau của ma trận kề được viết cách nhau một vài khoảng trống.

Chương trình được thực hiện với các thủ tục như sau:

- Hàm Init() : đọc dữ liệu theo khuôn dạng từ file dothi.in và thiết lập mảng chuaxet[u] =True (u=1, 2,..,n).
- Hàm BFS_Dequi : Cài đặt thuật toán BFS(u) bằng hàng đợi.

Ví dụ với file dothi.in dưới đây với u = 3 sẽ cho ta kết quả thực hiện chương trình như sau:

dothi.in

```
10
0    1    1    1    0    0    0    0    0    0
1    0    0    1    1    0    0    0    0    0
1    0    0    1    0    1    0    0    0    0
1    1    1    0    1    1    0    0    1    0
0    1    0    1    0    0    0    1    0    0
0    0    1    1    0    0    1    0    0    0
0    0    0    0    0    1    0    0    1    1
0    0    0    0    1    0    0    0    1    1
0    0    0    1    0    0    1    1    0    1
0    0    0    0    0    0    1    1    1    0
```

BFS(3) = 3, 1, 4, 6, 2, 5, 9, 7, 8, 10.

```
#include <stdio.h>
#include <conio.h>
#define MAX 50
#define TRUE 1
#define FALSE 0
int A[MAX][MAX], n,chuaxet[MAX];FILE *fp;
void Init(void){
     int i,j;
     fp= fopen("dothi.in","r");
     fscanf(fp,"%d",&n);
     printf("\n So dinh do thi:%d",n);
     for(i=1; i<=n; i++){
          printf("\n");chuaxet[i]=TRUE;
          for(j=1; j<=n; j++){
               fscanf(fp,"%d",&A[i][j]);
               printf("%3d",A[i][j]);
          }
     }
}
void BFS(int u){
     int queue[MAX], low=1, high=1, v;
     queue[low]=u;chuaxet[u]=FALSE;
     printf("\n Ket qua:");
     while(low<=high){
          u = queue[low];low=low+1;
          printf("%3d", u);
          for(v=1; v<=n; v++){
               if(A[u][v] && chuaxet[v]){
                    high = high+1;
                    queue[high]=v;
                    chuaxet[v]=FALSE;
               }
          }
     }
}
void main(void){
     int u;
     Init();
     printf("\n Dinh bat dau duyet:");
     scanf("%d",&u);
     BFS(u);
}
```

3.3. Ứng dụng của thuật toán DFS và BFS

Có rất nhiều ứng dụng khác nhau của thuật toán DFS và BFS trên đồ thị. Trong khuôn khổ của giáo trình này, chúng tôi đề cập đến một vài ứng dụng cơ bản. Những ứng dụng cụ thể hơn bạn đọc có thể tìm thấy rất nhiều trong các tài liệu khác nhau hoặc Internet. Những ứng dụng cơ bản của thuật toán DFS và BFS được đề cập bao gồm:

- *Duyệt tất cả các đỉnh của đồ thị.*
- *Duyệt tất cả các thành phần liên thông của đồ thị.*
- *Tìm đường đi từ đỉnh s đến đỉnh t trên đồ thị.*
- *Duyệt các đỉnh trụ trên đồ thị vô hướng.*
- *Duyệt các đỉnh trụ trên đồ thị vô hướng.*
- *Duyệt các cạnh cầu trên đồ thị vô hướng.*
- *Định chiều đồ thị vô hướng.*
- *Duyệt các đỉnh rẽ nhánh của cặp đỉnh s, t.*
- *Xác định tính liên thông mạnh trên đồ thị có hướng.*
- *Xác định tính liên thông yếu trên đồ thị có hướng.*
- *Thuật toán tìm kiếm theo chiều rộng trên đồ thị.*
- *Xây dựng cây khung của đồ thị vô hướng liên thông…*

3.3.1. Xác định thành phần liên thông của đồ thị

a) Đặt bài toán

Cho đồ thị đồ thị vô hướng G=<V,E>, trong đó V là tập đỉnh, E là tập cạnh. Bài toán đặt ra là xác định các thành phần liên thông của G =<V,E>?

b) Mô tả thuật toán

Một đồ thị có thể liên thông hoặc không liên thông. Nếu đồ thị liên thông thì số thành phần liên thông của nó là *1*. Điều này tương đương với phép duyệt theo thủ tục *DFS*(u) hoặc *BFS*(u) được gọi đến đúng một lần. Nói cách khác, DFS(u)=V và BFS(u)=V.

Nếu đồ thị không liên thông (số thành phần liên thông lớn hơn *1*) chúng ta có thể tách chúng thành những đồ thị con liên thông. Điều này cũng có nghĩa là trong phép duyệt đồ thị, số thành phần liên thông của nó bằng đúng số lần gọi tới thủ tục *DFS*() hoặc *BFS*(). Để xác định số các thành phần liên thông của đồ thị, chúng ta sử dụng thêm biến *solt* để nghi nhận các đỉnh cùng một thành phần liên thông. Khi đó, thuật toán xác định các thành phần liên thông của đồ thị được mô tả trong Hình 3.4.

Thuật toán Duyet-TPLT:

Bước 1 (Khởi tạo):

solt = 0; //Khởi tạo số thành phần liên thông ban đầu là 0

Bước 2 (Lặp):

for (u =1; u≤ n; u++) do //lặp trên tập đỉnh

if (chuaxet[u]) then

solt = solt + 1; //Ghi nhận số thành phần liên thông

<Ghi nhận các đỉnh thuộc TPLT>;

BFS (u); //DFS(u); //

endif;

endfor;

Bước 3 (Trả lại kết quả):

Return(solt);

end.

Hình 3.4. *Thuật toán duyệt các thành phần liên thông của đồ thị.*

c) Kiểm nghiệm thuật toán

Ví dụ ta cần kiểm nghiệm thuật toán trên Hình 3.4 cho đồ thị được biểu diễn dưới dạng ma trận kề như dưới đây.

0	0	1	0	1	0	1	0	0	0	0	0	0
0	0	0	1	0	1	0	0	0	0	0	0	0
1	0	0	0	1	0	1	0	0	0	1	0	0
0	1	0	0	0	1	0	1	0	1	0	0	0
1	0	1	0	0	0	1	0	1	0	1	0	1
0	1	0	1	0	0	0	1	0	1	0	0	0
1	0	1	0	1	0	0	0	1	0	0	0	0
0	0	0	1	0	1	0	0	0	1	0	1	0
0	0	0	0	1	0	1	0	0	0	1	0	1
0	0	0	1	0	1	0	1	0	0	0	1	0
0	0	1	0	1	0	0	0	1	0	0	0	1
0	0	0	0	0	0	0	1	0	1	0	0	0
0	0	0	0	1	0	0	0	1	0	1	0	0

Thực hiện thuật toán DFS và BFS như được mô tả ở trên ta nhận được :

Thành phần liên thông 1: BFS(1) = { 1, 3, 5, 7, 9, 11, 13}.

Thành phần liên thông 2: BFS(2) = {2, 4, 6, 8, 10, 12}.

d) Cài đặt thuật toán

Chương trình duyệt các thành phần liên thông của đồ thị được cài đặt theo khuôn dạng dữ liệu biểu diễn dưới dạng ma trận kề trong Mục 2.3.1 với các thủ tục sau:

- Hàm Init() : Đọc dữ liệu theo khuôn dạng và khởi đầu mảng chuaxet[u] = True ($1 \le i \le n$).
- Hàm BFS (u), DFS(u) : Hai thuật toán duyệt theo chiều rộng và duyệt theo chiều sâu được sử dụng để xác định các thành phần liên thông.

```
#include <stdio.h>
#include <conio.h>
#define MAX 50
#define TRUE 1
#define FALSE 0
int A[MAX][MAX], n,chuaxet[MAX], solt=0;
void Init(void){
     int i,j;FILE *fp;
     fp=fopen("dothi.in","r");
     fscanf(fp,"%d",&n);
     printf("\n So dinh do thi:%d",n);
     for(i=1; i<=n; i++){
          printf("\n");chuaxet[i]=TRUE;
          for(j=1; j<=n; j++){
               fscanf(fp,"%d",&A[i][j]);
               printf("%3d",A[i][j]);
          }
     }
}
void BFS(int u){
     int queue[MAX],low=1,high=1, s,t;
     queue[low]=u;chuaxet[u]=FALSE;
     while(low<=high){
          s = queue[low];low=low+1;
          printf("%3d", s);
          for(t=1; t<=n; t++){
               if(A[s][t] && chuaxet[t]){
                    high = high+1;
                    queue[high]=t;
                    chuaxet[t]=FALSE;
               }
          }
     }
}
```

```
void DFS(int u){
     int v;printf("%3d",u);
     chuaxet[u]=FALSE;
     for(v=1; v<=n; v++){
          if(A[u][v] && chuaxet[v])
               DFS(v);
     }
}
void main(void){
     int u ; clrscr();Init();
     for(u=1;u<=n; u++){
          if(chuaxet[u]){ solt++;
               printf("\n TP.Lien thong %d:", solt);
               BFS(u);//DFS(u);
          }
     }
}
```

3.3.2. Tìm đường đi giữa các đỉnh trên đồ thị

a) Đặt bài toán

Cho đồ thị $G =<V, E>$ (*vô hướng hoặc có hướng*), trong đó V là tập đỉnh, E là tập cạnh. Bài toán đặt ra là hãy tìm đường đi từ đỉnh $s \in V$ đến đỉnh $t \in V$?

b) Mô tả thuật toán

Cho đồ thị $G =<V, E>$, s, t là hai đỉnh thuộc V. Khi đó, dễ dàng nhận thấy, nếu $t \in DFS(s)$ hoặc $t \in BFS(s)$ thì ta có thể kết luận có đường đi từ s đến t trên đồ thị. Nếu $t \notin DFS(s)$ hoặc $t \notin BFS(s)$ thì ta có thể kết luận không có đường đi từ s đến t trên đồ thị. Vấn đề còn lại là ta ghi nhận thế nào đường đi từ s đến t?

Để ghi nhận đường đi từ s đến t dựa vào hai thuật toán $DFS(s)$ hoặc $BFS(s)$ ta sử dụng một mảng *truoc*[] gồm n phần tử ($n=|V|$). Khởi tạo ban đầu *truoc*[u]=0 với mọi $u = 1, 2, .., n$. Trong quá trình thực hiện hai thuật toán DFS (s) và BFS(s), mỗi khi ta đưa đỉnh $v \in Ke(s)$ vào ngăn xếp (*trong trường hợp ta sử dụng thuật toán DFS*) hoặc hàng đợi(*trong trường hợp ta sử dụng thuật toán DFS*) ta ghi nhận *truoc*[v] = s. Điều này có nghĩa, để đi được đến v ta phải qua đỉnh s. Khi hai thuật toán DFS và BFS duyệt đến đỉnh t thì *truoc*[t] sẽ nhận giá trị là một đỉnh nào đó thuộc V hay $t \in DFS(s)$ hoặc $t \in BFS(s)$. Trong trường hợp hai thủ tục *DFS* và *BFS* không duyệt được đến đỉnh t, khi đó *truoc*[t] =0 và ta kết luận không có đường đi từ s đến t. Hình 3.5 dưới đây mô tả thuật toán tìm đường đi từ đỉnh s đến đỉnh t trên đồ thị bằng thuật toán DFS. Hình 3.6 dưới đây mô tả thuật toán tìm đường đi từ đỉnh s đến đỉnh t trên đồ thị bằng thuật toán BFS. Hình 3.7 dưới đây mô tả thuật toán ghi nhận đường đi từ đỉnh s đến đỉnh t trên đồ thị.

```
Thuật toán DFS(s):
Begin
        Bước 1 (Khởi tạo):
                stack = ∅; //Khởi tạo stack là ∅
                Push(stack, s); //Đưa đỉnh s vào ngăn xếp
                chuaxet[s] = False; //Xác nhận đỉnh u đã duyệt
        Bước 2 (Lặp) :
                while ( stack ≠ ∅ ) do
                        u = Pop(stack); //Loại đỉnh ở đầu ngăn xếp
                        for each v∈ Ke(u) do //Lấy mỗi đỉnh u∈Ke(v)
                                if ( chuaxet[v] ) then //Nếu v đúng là chưa duyệt
                                        chuaxet[v] = False; // Xác nhận đỉnh v đã duyệt
                                        Push(stack, u);//Đưa u vào stack
                                        Push(stack, v); //Đưa v vào stack
                                        truoc[v] = u; //Ghi nhận truoc[v] là u
                                        break;          //Chỉ lấy một đỉnh t
                                EndIf;
                        EndFor;
                EndWhile;
        Bước 3 (Trả lại kết quả):
                Return(<Tập đỉnh đã duyệt>);
End.
```

Hình 3.5. Thuật toán DFS tìm đường đi từ s đến t.

```
Thuật toán BFS(s):
        Bước 1(Khởi tạo):
                Queue = ∅; Push(Queue,s); chuaxet[s] = False;
        Bước 2 (Lặp):
                while (Queue ≠ ∅ ) do
                        u = Pop(Queue);
                        for each v∈Ke(u) do
                                if ( chuaxet[v] ) then
                                        Push(Queue, v);chuaxet[v]=False;truoc[v]=u;
                                EndIf ;
                        EndFor ;
                EndWhile ;
        Bước 3 (Trả lại kết quả) :
                Return(<Tập đỉnh được duyệt>) ;
End.
```

Hình 3.6. Thuật toán BFS tìm đường đi từ s đến t.

```
Thuật toán Ghi-Nhan-Duong-Di (s, t) {
        if ( truoc[t] == 0 ) {
                <Không có đường đi từ s đến t>;
        }
        else {
                <Đưa ra đỉnh t>; //Đưa ra trước đỉnh t
                u = truoc[t]; //u là đỉnh trước khi đến được t
                while (u ≠ s ) { //Lặp nếu u chưa phải là s
                        <Đưa ra đỉnh u>; //Đưa ra đỉnh u
                        u = truoc[u]; // Lần ngược lại đỉnh truoc[u].
                }
                <Đưa ra nốt đỉnh s>;
        }
}
```

Hình 3.7. Thủ tục ghi nhận đường đi từ s đến t

c) Kiểm nghiệm thuật toán

Giả sử ta cần xác định đường đi từ đỉnh 1 đến đỉnh 13 trên đồ thị được biểu diễn dưới dạng ma trận kề. Khi đó, thứ tự các bước thực hiện theo thuật toán DFS được thể hiện trong Bảng 3.3, thứ tự các bước thực hiện theo thuật toán BFS được thể hiện trong Bảng 3.4.

Bảng 3.3. Kiểm nghiệm thuật toán DFS(1).

STT	Trạng thái stack	Truoc[s]=?
1	1	0
2	1, 2	truoc[2] =1
3	1, 2, 3	truoc[3] = 2
4	1, 2, 3, 4	truoc[4] =3
5	1, 2, 3, 4, 7	truoc[7] =4
6	1, 2, 3, 4, 7, 5	truoc[5] =7
7	1, 2, 3, 4, 7, 5, 6	truoc[6] =5
8	1, 2, 3, 4, 7, 5, 6, 12	truoc[12] =6
9	1, 2, 3, 4, 7, 5, 6, 12, 8	truoc[8] =12
10	**1, 2, 3, 4, 7, 5, 6, 12, 10**	truoc[10] =12
11	1, 2, 3, 4, 7, 5, 6, 12, 10, 9	truoc[9] =10

12	1, 2, 3, 4, 7, 5, 6, 12, 10, 9, 11	truoc[11] =9
13	1, 2, 3, 4, 7, 5, 6, 12, 10, 9, 11, 13	truoc[13] =11
14	∅	

Kết quả đường đi từ đỉnh 1 đến đỉnh 13: *13->11-9-10-12-6-5-7-4-3-2-1*.

Bảng 3.4. Kiểm nghiệm thuật toán BFS(1).

STT	Trạng thái Queue	Truoc[s]=?
1	1	truoc[1]=0
2	2, 3, 4	truoc[2]=1; truoc[3]=1; truoc[4]=1;
3	3, 4, 6	truoc[6]= 2
4	4, 6, 5	truoc[5]=3
5	6, 5, 7	truoc[7]= 4
6	5, 7, 12	truoc[12]=6
7	7, 12, 8	truoc[8]=12
8	12, 8	
9	8, 10	truoc[10]=12;
10	10	
11	9, 11, 13	truoc[9]=10; truoc[11]=10; truoc[13]=10;
12	11, 13	
13	13	
14	∅	

Kết quả đường đi: 13-10-12-6-2-1.

Chú ý.

- Đường đi từ đỉnh s đến đỉnh t theo thuật toán BFS đi qua ít nhất các cạnh của đồ thị (có độ dài nhỏ nhất).

d) Cài đặt thuật toán

```
#include <stdio.h>
#include <conio.h>
#include <iostream.h>
#define MAX 50
#define TRUE 1
#define FALSE 0
int A[MAX][MAX], n,chuaxet[MAX], truoc[MAX], s, t;
void Init(void){//Đọc dữ liệu và khởi đầu các biến
     int i,j;FILE *fp;
     fp=fopen("dothi.in","r");
     fscanf(fp,"%d",&n);
     printf("\n So dinh do thi:%d",n);
     for(i=1; i<=n; i++){
```

```
          printf("\n");chuaxet[i]=TRUE;truoc[i]=0;
          for(j=1; j<=n; j++){
               fscanf(fp,"%d",&A[i][j]);
               printf("%3d",A[i][j]);
          }
     }
}
void DFS(int u){//Thuật toán DFS
     int v;
     printf("%3d",u);chuaxet[u]=FALSE;
     for(v=1; v<=n; v++){
          if(A[u][v] && chuaxet[v]){
               truoc[v]=u;DFS(v);
          }
     }
}
void BFS(int i){//Thuật toán BFS
     int queue[MAX], low=1, high=1, u, v;
     queue[low]=i;chuaxet[i]=FALSE;
     while(low<=high){
          u = queue[low];low=low+1;
          for(v=1; v<=n; v++){
               if(A[u][v] && chuaxet[v]){
                    high = high+1;queue[high]=v;
                    truoc[v]=u; chuaxet[v]=FALSE;
               }
          }
     }
}

void Duongdi (void){
     if (truoc[t]==0){
          printf("\n Khong ton tai duong di");
          getch(); return;
     }
     printf("\n Duong di:");
     int j = t;printf("%3d<=",j);
     while(truoc[j]!=s){
          printf("%3d<=",truoc[j]);j=truoc[j];
     }
     printf("%3d",s); getch();
}
```

```
void main (void){
     Init();
     printf("\n Dinh dau:");scanf("%d",&s);
     printf("\n Dinh cuoi:");scanf("%d",&t);
     DFS(s); //BFS(s);
     Duongdi ();
}
```

3.3.3. Tính liên thông mạnh trên đồ thị có hướng

a) Đặt bài toán

Đồ thị có hướng G=<V,E> liên thông mạnh nếu giữa hai đỉnh bất kỳ của nó đều tồn tại đường đi. Cho trước đồ thị có hướng G = <V,E>. Nhiệm vụ của ta là kiểm tra xem G có liên thông mạnh hay không?

b) Mô tả thuật toán

Đối với đồ thị vô hướng, nếu hai thủ tục DFS(u) = V hoặc BFS(u) = V thì ta kết luận đồ thị vô hướng liên thông. Đối với đồ thị có hướng, nếu DFS(u)=V hoặc BFS(u)=V thì ta mới chỉ có kết luận có đường đi từ u đến tất cả các đỉnh còn lại của đồ thị. Nhiệm vụ của ta là phải kiểm tra DFS(u)=V hoặc BFS(u)=V với mọi u∈V. Hình 3.8 dưới đây mô tả chi tiết thuật toán kiểm tra tính liên thông mạnh của đồ thị.

```
Boolean      Strong-Connective ( G =<V, E>) {
      ReInit(); //∀u∈V: chuaxet[u] = True;
      for each u∈V do {//Lấy mỗi đỉnh thuộc V
            if (DFS(u)≠V ) then //Nếu DFS(u) ≠V hoặc BFS(u) ≠V
                  return(False); //Đồ thị không liên thông mạnh
            endif;
            ReInit();//Khởi tạo lại các mảng chuaxet[]
      endfor;
      return(True);//Đồ thị liên thông mạnh
}
```

Hình 3.8. Thuật toán kiểm tra tính liên thông mạnh.

c) Kiểm nghiệm thuật toán

Giả sử ta cần xác định đồ thị có hướng G =<V,E> được biểu diễn dưới dạng ma trận kề dưới đây có liên thông mạnh hay không? Khi đó các bước thực hiện theo thuật toán Hình 3.8 được thực hiện theo Bảng 3.5 dưới đây.

0	0	0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	0	0	0	1	0	0	0	0	0
0	0	0	0	0	0	0	0	1	0	0	0	1
1	0	0	0	0	1	0	0	0	0	0	0	0
0	0	0	0	0	0	1	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1	0	1	0
0	0	0	0	0	0	0	0	0	0	1	0	1
0	0	0	1	0	0	0	0	0	0	0	1	0
0	0	0	0	1	0	1	0	0	0	0	0	0
0	1	1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	1	0	0	0	0	0
0	0	0	1	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	0	0	1	0	1	0	0

Bảng 3.5. Kiểm nghiệm thuật toán kiểm tra tính liên thông mạnh.

Đỉnh u∈V	**DFS(u)=?//BFS(u)=?**	**DFS(u) =V?**
1∈V	DFS(1) = 1, 6, 10, 2, 3, 9, 5, 7, 11, 8, 4, 12, 13	Yes
2∈V	DFS(2) = 2, 3, 9, 5, 7, 11, 8, 4, 1, 6, 10, 12, 13	Yes
3∈V	DFS(3) = 3, 9, 5, 7, 11, 2, 8, 4, 1, 6, 10, 12, 13	Yes
4∈V	DFS(4) = 4, 1, 6, 10, 2, 3, 9, 5, 7, 11, 8, 12, 13	Yes
5∈V	DFS(5) = 5, 7, 11, 2, 3, 9, 13, 8, 4, 1, 6, 10, 12	Yes
6∈V	DFS(6) = 6, 10, 2, 3, 9, 5, 7, 11, 8, 4, 1, 12, 13	Yes
7∈V	DFS(7) = 7, 11, 2, 3, 9, 5, 13, 8, 4, 1, 6, 10, 12	Yes
8∈V	DFS(8) = 8, 4, 1, 6, 10, 2, 3, 9, 5, 7, 11, 13, 12	Yes
9∈V	DFS(8) = 9, 5, 7, 11, 2, 3, 13, 8, 4, 1, 6, 10, 12	Yes
10∈V	DFS(10) = 10, 2, 3, 9, 5, 7, 11, 8, 4, 1, 6, 12, 13	Yes
11∈V	DFS(11) = 11, 2, 3, 9, 5, 7, 13, 8, 4, 1, 6, 10, 12	Yes
12∈V	DFS(12) = 12, 4, 1, 6, 10, 2, 3, 9, 5, 7, 11, 8, 13	Yes
13∈V	DFS(13) = 13, 9, 5, 7, 11, 2, 3, 8, 4, 1, 6, 10, 12	Yes

Cột ngoài cùng của Bảng có DFS(u) = V với mọi u∈V nên ta kết luận G liên thông mạnh. Nếu tại một hàng nào đó có DFS(u) ≠V thì ta kết luận đồ thị không liên thông mạnh và không cần phải kiểm tra tiếp các đỉnh còn lại.

d) Cài đặt thuật toán

Thuật toán được cài đặt theo khuôn dạng đồ thị được qui ước trong Mục 2.3.1 với các thủ tục sau:

- Thủ tục Read-Data() : Đọc ma trận kề biểu diễn đồ thị trong file dothi.in.
- Thủ tục ReInit() : Khởi tạo lại giá trị cho mảng chuaxet[].
- Thủ tục DFS(u) : Thuật toán DFS bắt đầu tại đỉnh u.
- Thủ tục BFS(u) : Thuật toán BFS bắt đầu tại đỉnh u.
- Hàm Strong-Conective(): Kiểm tra tính liên thông mạnh của đồ thị.

Chương trình kiểm tra tính liên thông mạnh của đồ thị được thể hiện như dưới đây.

```
#include <stdio.h>
#include <conio.h>
#include <iostream.h>
#define MAX 50
#define TRUE 1
#define FALSE 0
int A[MAX][MAX], n,chuaxet[MAX], solt=0;
//Doc du lieu
void Read_Data(void){
     int i,j;FILE *fp;
     fp=fopen("dothi.IN","r");
     fscanf(fp,"%d",&n);
     printf("\n So dinh do thi:%d",n);
     for(i=1; i<=n; i++){
          printf("\n");
          for(j=1; j<=n; j++){
               fscanf(fp,"%d",&A[i][j]);
               printf("%3d",A[i][j]);
          }
     }
}
//Thuat toan BFS
void BFS(int u){
     int queue[MAX],low=1,high=1, s,t;
     queue[low]=u;chuaxet[u]=FALSE;
     while(low<=high){
          s = queue[low];low=low+1;
          //printf("%3d", s);
          for(t=1; t<=n; t++){
               if(A[s][t] && chuaxet[t]){
                    high = high+1;
```

```
                    queue[high]=t;
                    chuaxet[t]=FALSE;
               }
          }
     }
}
//Thuat toan DFS
void DFS(int u){
     int v;//printf("%3d",u);
     chuaxet[u]=FALSE;
     for(v=1; v<=n; v++){
          if(A[u][v] && chuaxet[v])
               DFS(v);
     }
}
//Khoi dau lai mang chuaxet[]
void ReInit(void) {
     for (int i=1; i<=n; i++)
          chuaxet[i]=TRUE;
}
//Kiem tra so lien thong >1?
int Test_So_Lien_Thong(void) {
     for(int u=1; u<=n; u++)
          if(chuaxet[u]) return(1);
     return(0);
}
//Kiem tra tinh lien thong manh
int Strong_Conective (void) {
      Read_Data(); ReInit();
      for (int u=1; u<=n; u++){
          chuaxet[u]=FALSE;
          if(u==1) DFS(2);//BFS(2);
          esle DFS(1); //BFS(1);
          if(Test_So_Lien_Thong()) return(0);
          ReInit();
      }
      return(1);
}
void main(void){
     if(Test_LT_Manh())
          printf("\n Do thi lien thong manh");
     else
          printf("\n Do thi khong lien thong manh");
}
```

3.3.4. Duyệt các đỉnh trụ

a) Đặt bài toán

Cho đồ thị vô hướng liên thông $G = <V, E>$. Đỉnh $u \in V$ được gọi trụ nếu loại bỏ đỉnh u cùng với các cạnh nối với u làm tăng thành phần liên thông của đồ thị. Bài toán đặt ra là xây dựng thuật toán duyệt các đỉnh trụ của đồ thị vô hướng G =<V, E> cho trước?

b) Mô tả thuật toán

Không hạn chế tính tổng quát của bài toán ta có thể giả thiết đồ thị đã cho ban đầu là liên thông. Trong trường hợp đồ thị không liên thông, bài toán duyệt trụ thực chất giải quyết cho mỗi thành phần liên thông của đồ thị.

Đối với đồ thị được biểu diễn dưới dạng ma trận kề, việc loại bỏ đỉnh u cùng với các cạnh nối với u tương ứng với việc loại bỏ hàng u và cột u tương ứng trong ma trận kề. Để thực hiện việc này trong các thủ tục DFS() hoặc BFS() ta chỉ cần thiết lập giá trị chuaxet[u] = False. Quá trình duyệt sẽ được thực hiện tại một đỉnh bất kỳ v≠u. Nếu DFS(v) = V\{u} hoặc BFS(v) = V\{u} thì đồ thị mới nhận được cũng chỉ có 1 thành phần liên thông và ta kết luận v không là trụ. Trường hợp DFS(v) ≠ V\{u} hoặc BFS(v) ≠ V\{u} thì v chính là trụ vì số thành phần liên thông của đồ thị đã tăng lên. Thuật toán duyệt các đỉnh trụ của đồ thị được mô tả chi tiết trong Hình 3.9.

```
Thuật toán    Duyet-Tru ( G =<V, E>) {
        ReInit(); //∀u∈V: chuaxet[u] = True;
        for each v∈V do {//Lấy mỗi đỉnh u tập đỉnh V
                chuaxet[v] = False; //Cấm DFS hoặc BFS duyệt vào đỉnh v
                if (DFS(u) ≠ V\{v} ) then //Duyệt DFS hoặc BFS tại đỉnh u≠v
                        <Ghi nhận v là trụ>;
                endif ;
                ReInit();//Khởi tạo lại các mảng chuaxet[]
        endfor;
}
```

Hình 3.9. Thuật toán duyệt các đỉnh trụ của đồ thị.

c) Kiểm nghiệm thuật toán

Giả sử ta cần xác định đồ thị vô hướng G =<V,E> được biểu diễn dưới dạng ma trận kề dưới đây có những đỉnh trụ nào? Khi đó các bước thực hiện theo thuật toán Hình 3.9 được thực hiện theo Bảng 3.6 dưới đây.

0	1	1	1	0	0	0	0	0	0	0	0	0
1	0	1	1	0	0	0	0	0	0	0	0	0
1	1	0	1	1	0	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	1	1	1	1	0	0	0	0
0	0	0	0	1	0	1	0	1	0	0	0	0
0	0	0	0	1	1	0	1	0	0	0	0	0
0	0	0	0	1	0	1	0	1	0	0	0	0
0	0	0	0	1	1	0	1	0	1	0	0	0
0	0	0	0	0	0	0	0	1	0	1	1	1
0	0	0	0	0	0	0	0	0	1	0	1	1
0	0	0	0	0	0	0	0	0	1	1	0	1
0	0	0	0	0	0	0	0	0	1	1	1	0

Bảng 3.6. Kiểm nghiệm thuật toán duyệt các đỉnh trụ của đồ thị.

Đỉnh v∈V	**DFS(u)=?//BFS(u)=? u≠v.**	**DFS(u) ≠V\v?**
1∈V	DFS(2) = 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13	No
2∈V	DFS(1) = 1, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13	No
3∈V	DFS(1) = 1, 2, 4	Yes
4∈V	DFS(1) = 1, 2, 3, 5, 6, 7, 8, 9, 10, 11, 12, 13	No
5∈V	DFS(1) = 1, 2, 3, 4	Yes
6∈V	DFS(1) = 1, 2, 3, 4, 5, 7, 8, 9, 10, 11, 12, 13	No
7∈V	DFS(1) = 1, 2, 3, 4, 5, 6, 9, 8, 10, 11, 12, 13	No
8∈V	DFS(1) = 1, 2, 3, 4, 5, 6, 7, 9, 10, 11, 12, 13	No
9∈V	DFS(1) = 1, 2, 3, 4, 5, 6, 7, 8	Yes
10∈V	DFS(1) = 1, 2, 3, 4, 5, 6, 7, 8, 9	Yes
11∈V	DFS(1) = 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 13	No
12∈V	DFS(1) = 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 13	No
13∈V	DFS(1) = 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 13	No

Đỉnh ở hàng u có giá trị cột số 3 là Yes là những đỉnh trụ, các đỉnh có giá trị No không là đỉnh trụ.

d) Cài đặt thuật toán

Thuật toán được cài đặt theo khuôn dạng đồ thị được qui ước trong Mục 2.3.1 với các thủ tục sau:

- Thủ tục Read-Data() : Đọc ma trận kề biểu diễn đồ thị trong file dothi.in.
- Thủ tục ReInit() : Khởi tạo lại giá trị cho mảng chuaxet[].
- Thủ tục DFS(u) : Thuật toán DFS bắt đầu tại đỉnh u.
- Thủ tục BFS(u) : Thuật toán BFS bắt đầu tại đỉnh u.

Văn bản chương trình được thể hiện như dưới đây.

```
#include <stdio.h>
#include <conio.h>
#include <iostream.h>
#define MAX 50
#define TRUE 1
#define FALSE 0
int A[MAX][MAX], n,chuaxet[MAX], solt=0;
//Doc du lieu
void Read_Data(void){
  int i,j;FILE *fp;
  fp=fopen("dothi.IN","r");
  fscanf(fp,"%d",&n);
  for(i=1; i<=n; i++){
       printf("\n");
       for(j=1; j<=n; j++){
            fscanf(fp,"%d",&A[i][j]);
       }
  }
}
//Thuat toan BFS
void BFS(int u){
  int queue[MAX],low=1,high=1, s,t;
  queue[low]=u;chuaxet[u]=FALSE;
  while(low<=high){
       s = queue[low];low=low+1;
       //printf("%3d", s);
       for(t=1; t<=n; t++){
            if(A[s][t] && chuaxet[t]){
                 high = high+1;
                 queue[high]=t;
                 chuaxet[t]=FALSE;
            }
       }
  }
}
```

```
//Thuat toan DFS
void DFS(int u){
  int v;//printf("%3d",u);
  chuaxet[u]=FALSE;
  for(v=1; v<=n; v++){
       if(A[u][v] && chuaxet[v])
            DFS(v);
  }
}
//Khoi dau lai mang chuaxet[]
void ReInit(void) {
  for (int i=1; i<=n; i++)
       chuaxet[i]=TRUE;
}
//Kiem tra so lien thong >1?
int Test_So_Lien_Thong(void) {
  for(int u=1; u<=n; u++)
       if(chuaxet[u]) return(1);
  return(0);
}
//Duyệt các đỉnh trụ
void main(void) {
   Read_Data(); ReInit();
   for (int u=1; u<=n; u++){
       DFS(1);//BFS(1);
       if(Test_So_Lien_Thong())
            printf("\n Dinh %d la tru", u);
       ReInit();
   }
}
```

3.3.5. Duyệt các cạnh cầu

a) Đặt bài toán

Cho đồ thị vô hướng G =<V,E>. Cạnh e∈E được gọi là cầu nếu loại bỏ e làm tăng thành phần liên thông của đồ thị. Bài toán đặt ra là cho trước đồ thị vô hướng G = <V,E>, hãy liệt kê tất cả các cạnh cầu của đồ thị.

b) Mô tả thuật toán

Không hạn chế tính tổng quát của bài toán ta cũng giả thiết đồ thị G=<V,E> đã cho là liên thông. Trong trường hợp đồ thị không liên thông, bài toán duyệt cầu thực hiện trên mỗi thành phần liên thông của đồ thị.

Trong trường hợp đồ thị được biểu diễn dưới dạng ma trận kề, ehi loại bỏ cạnh e=(u,v)∈E ra khỏi đồ thị ta thực hiện bằng cách cho các phần tử A[u][v]=0 và A[v][u]=0 (A[][] là ma trận kề biểu diễn đồ thị G). Đối với đồ thị được biểu diễn dưới dạng danh sách kề, danh sách kề của đỉnh u ta bớt đi đỉnh v và danh sách kề của đỉnh v ta bớt đi đỉnh u (Ke(u) = Ke(u)\{v}, Ke(v) = Ke(v)\{u}). Thuật toán duyệt các cạnh cầu của đồ thị được ô tả chi tiết trong Hình 3.10.

```
Thuật toán    Duyet-Cau ( G =<V, E>) {
        ReInit(); //∀u∈V: chuaxet[u] = True;
        for each e∈E do {//Lấy mỗi cạnh thuộc đồ thị
                E = E\{e}; //Loại bỏ cạnh e ra khỏi đồ thị
                if (DFS(1) ≠ V ) then //Nếu tăng thành phần liên thông của đồ thị
                        <Ghi nhận e là cầu>;
                endif ;
                E = E∪{e} ; //Hoàn trả lại cạnh e
                ReInit();//Khởi tạo lại các mảng chuaxet[]
        endfor;
}
```

Hình 3.10. Thuật toán duyệt các cạnh cầu của đồ thị.

c) Kiểm nghiệm thuật toán

Giả sử ta cần xác định đồ thị vô hướng G =<V,E> được biểu diễn dưới dạng ma trận kề dưới đây có những cạnh cầu? Khi đó các bước thực hiện theo thuật toán Hình 3.10 được thực hiện theo Bảng 3.7 dưới đây.

0	1	1	1	0	0	0	0	0	0	0	0	0
1	0	1	1	0	0	0	0	0	0	0	0	0
1	1	0	1	1	0	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	1	1	1	1	0	0	0	0
0	0	0	0	1	0	1	0	1	0	0	0	0
0	0	0	0	1	1	0	1	0	0	0	0	0
0	0	0	0	1	0	1	0	1	0	0	0	0
0	0	0	0	1	1	0	1	0	1	0	0	0
0	0	0	0	0	0	0	0	1	0	1	1	1
0	0	0	0	0	0	0	0	0	1	0	1	1
0	0	0	0	0	0	0	0	0	1	1	0	1
0	0	0	0	0	0	0	0	0	1	1	1	0

Bảng 3.7. Kiểm nghiệm thuật toán duyệt các cạnh cầu của đồ thị.

Cạnh e∈E	**DFS(u)=?//BFS(u)=?**	**DFS(u) ≠V?**
(1,2)∈E	DFS(1) = 1, 3, 2, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13	No
(1,3)∈E	DFS(1) = 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13	No
(1,4)∈E	DFS(1) = 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13	No
(2,3)∈E	DFS(1) = 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13	No
(2,4)∈E	DFS(1) = 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13	No
(3,4)∈E	DFS(1) = 1, 2, 3, 5, 6, 7, 8, 9, 10, 11, 12, 13, 4	No
(3,5)∈E	DFS(1) = 1, 2, 3, 4, 5	Yes
(5,6)∈E	DFS(1) = 1, 2, 3, 4, 5, 7, 6, 8, 9, 10, 11, 12, 13	No
(5,7)∈E	DFS(1) = 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13	No
(5,8)∈E	DFS(1) = 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13	No
(5,9)∈E	DFS(1) = 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13	No
(6,7)∈E	DFS(1) = 1, 2, 3, 4, 5, 6, 9, 8, 7, 10, 11, 12, 13	No
(6,9) ∈E	DFS(1) = 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13	No
(7,8) ∈E	DFS(1) =1, 2, 3, 4, 5, 6, 7, 9, 8, 10, 11, 12, 13	No
(8,9) ∈E	DFS(1) =1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13	No
(9,10) ∈E	DFS(1) =1, 2, 3, 4, 5, 6, 7, 8, 9	Yes
(10,11) ∈E	DFS(1) =1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 11, 13	No
(10,12) ∈E	DFS(1) =1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13	No
(10,13) ∈E	DFS(1) =1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13	No
(11,12) ∈E	DFS(1) =1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 13, 12	No
(11,13) ∈E	DFS(1) =1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13	No
(12,13) ∈E	DFS(1) =1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13	No
Kết luận: cạnh (35), (9,10) là cầu		

d) Cài đặt thuật toán

Thuật toán được cài đặt theo khuôn dạng đồ thị được qui ước trong Mục 2.3.1 với các thủ tục sau:

- Thủ tục Read-Data() : Đọc ma trận kề biểu diễn đồ thị trong file dothi.in.
- Thủ tục ReInit() : Khởi tạo lại giá trị cho mảng chuaxet[].
- Thủ tục DFS(u) : Thuật toán DFS bắt đầu tại đỉnh u.
- Thủ tục BFS(u) : Thuật toán BFS bắt đầu tại đỉnh u.

Chương trình kiểm tra tính liên thông mạnh của đồ thị được thể hiện như dưới đây.

```
#include <stdio.h>
#include <conio.h>
#include <iostream.h>
#define MAX 50
#define TRUE 1
#define FALSE 0
int A[MAX][MAX], n,chuaxet[MAX], solt=0;
//Doc du lieu
void Read_Data(void){
     int i,j;FILE *fp;
     fp=fopen("dothi.IN","r");
     fscanf(fp,"%d",&n);
     for(i=1; i<=n; i++){
          printf("\n");
          for(j=1; j<=n; j++){
               fscanf(fp,"%d",&A[i][j]);
          }
     }
}
//Thuat toan BFS
void BFS(int u){
     int queue[MAX],low=1,high=1, s,t;
     queue[low]=u;chuaxet[u]=FALSE;
     while(low<=high){
          s = queue[low];low=low+1;
          //printf("%3d", s);
          for(t=1; t<=n; t++){
               if(A[s][t] && chuaxet[t]){
                    high = high+1;
                    queue[high]=t;
                    chuaxet[t]=FALSE;
               }
          }
     }
}
```

```
//Thuat toan DFS
void DFS(int u){
     int v;//printf("%3d",u);
     chuaxet[u]=FALSE;
     for(v=1; v<=n; v++){
          if(A[u][v] && chuaxet[v])
               DFS(v);
     }
}
//Khoi dau lai mang chuaxet[]
void ReInit(void) {
     for (int i=1; i<=n; i++)
          chuaxet[i]=TRUE;
}
//Kiem tra so lien thong >1?
int Test_So_Lien_Thong(void) {
     for(int u=1; u<=n; u++)
          if(chuaxet[u]) return(1);
     return(0);
}
//Duyệt cạnh cầu
void main(void) {
      Read_Data(); ReInit();
      for (int u=1; u<n; u++){
          for(int v=u+1;v<=n; v++){
               if(A[u][v]) { //Neu (u,v) la mot canh
                    A[u][v]=0; A[v][u]=0;//Loai canh
                    DFS(1);//BFS(1);
                    if(Test_So_Lien_Thong())
                         printf("\n Canh %d%5d ",u, v);
                    A[u][v]=1; A[v][u]=1;
                    ReInit();//Khoi tao lai mang chuaxet
               }
          }
      }
}
```

3.4. Một số bài toán quan trọng khác

2.4.1. Duyệt các thành phần liên thông mạnh của đồ thị

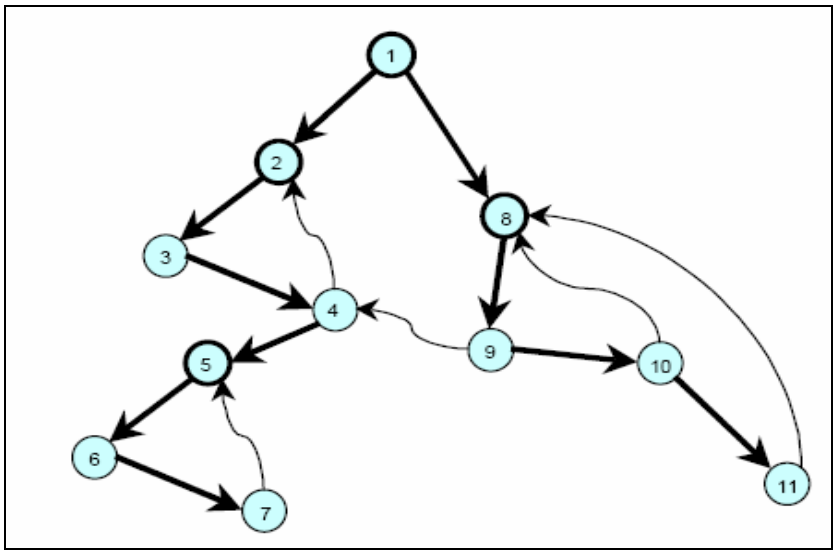
Đối với đồ thị có hướng người ta quan tâm đến việc duyệt các thành phần liên thông mạnh của đồ thị. Mỗi thành phần liên thông mạnh của đồ thị là một đồ thị con của G mà giữa hai đỉnh bất kỳ của đồ thị con đều có đường đi. Bài toán đặt ra là hãy liệt kê tất cả các thành phần liên thông mạnh của đồ thị có hướng G=<V,E>. Ví dụ với đồ thị trong Hình 3.11 dưới đây sẽ cho ta bốn thành phần liên thông mạnh.

Thành phần liên thông mạnh 1: 7, 5, 6.

Thành phần liên thông mạnh 2: 4, 3, 2.

Thành phần liên thông mạnh 3: 11, 10, 9, 8.

Thành phần liên thông mạnh 4: 1.



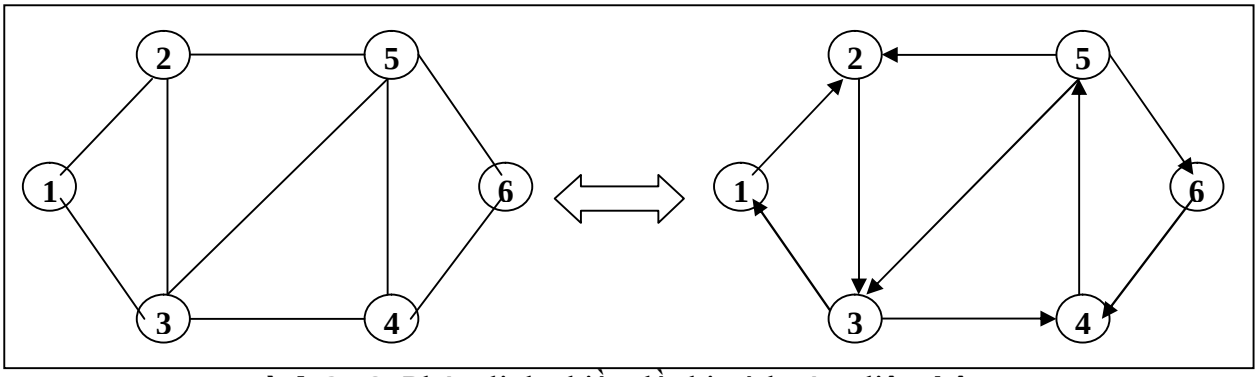
Hình 3.11. Đồ thị có hướng G =<V,E>

2.4.2. Bài toán định chiều đồ thị

Một trong những ứng dụng quan trọng của đồ thị là biểu diễn đồ thị cho các hệ thống giao thông. Đối với hệ thống giao thông người ta quan tâm đến liệu hệ thống có thể định chiều được hay không.

Định nghĩa. Phép định chiều đồ thị vô hướng liên thông là phép biến đổi đồ thị vô hướng liên thông thành đồ thị có hướng liên thông mạnh. Đồ thị vô hướng G =<V,E> có thể dịch chuyển được thành đồ thị có hướng liên thông mạnh bằng cách định chiều mỗi cạnh vô hướng thành một cung có hướng được gọi là đồ thị định chiều được.

Ví dụ đồ thị vô hướng trong Hình 3.12 dưới đây được gọi là định chiều được.



Hình 3.12. Phép định chiều đồ thị vô hướng liên thông.

Định lý. Đồ thị vô hướng liên thông G =<V, E> định chiều được khi và chỉ khi tất cả các cạnh e∈E của G đều không phải là cầu.

Bạn đọc tự tìm hiểu cách chứng minh định lý trong các tài liệu [1, 2].

Bài toán. Cho đồ thị vô hướng liên thông G = <V,E>. Hãy định chiều đồ thị G sao cho ta có thể nhận được đồ thị có hướng với ít nhất thành phần liên thông mạnh.

3.5. Một số điểm cần ghi nhớ

- Thuật toán duyệt theo chiều sâu bắt đầu tại đỉnh u∈V.
- Thuật toán duyệt theo rộng sâu bắt đầu tại đỉnh u∈V.
- Duyệt tất cả các đỉnh của đồ thị dựa vào DFS(u), BFS(u).
- Duyệt tất cả các thành phần liên thông của đồ thị dựa vào DFS(u), BFS(u).
- Tìm đường đi từ đỉn s đến t trên đồ thị dựa vào DFS(u), BFS(u).
- Kiểm tra tính liên thông mạnh của đồ thị dựa vào DFS(u), BFS(u).
- Duyệt các đỉnh trụ của đồ thị DFS(u), BFS(u).
- Duyệt các cạnh cầu của đồ thị DFS(u), BFS(u).
- Một số ứng dụng quan trọng khác của DFS và BFS.

BÀI TẬP

1. Cho đồ thị vô hướng được biểu diễn dưới dạng ma trận kề như Hình bên phải. Hãy thực hiện:

a) Trình bày thuật toán BFS(u)?

b) Kiểm nghiệm thuật toán BFS(u) bắt đầu tại đỉnh u=1? Chỉ rõ kết quả trung gian theo mỗi bước thực hiện của thuật toán.

c) Kiểm nghiệm thuật toán BFS(u) bắt đầu tại đỉnh u=7? Chỉ rõ kết quả trung gian theo mỗi bước thực hiện của thuật toán.

0	1	1	1	1	0	0	0	0	0	0	0	0
1	0	1	1	0	0	0	0	0	0	0	0	0
1	1	0	1	0	0	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	1	1	1	1	0	0	0	0
0	0	0	0	1	0	1	0	1	0	0	0	0
0	0	0	0	1	1	0	1	0	1	0	0	0
0	0	0	0	1	0	1	0	1	0	0	0	0
0	0	0	0	1	1	0	1	0	0	0	0	0
0	0	0	0	0	0	1	0	0	0	1	1	1
0	0	0	0	0	0	0	0	0	1	0	1	1
0	0	0	0	0	0	0	0	0	1	1	0	1
0	0	0	0	0	0	0	0	0	1	1	1	0

2. Cho đồ thị vô hướng được biểu diễn dưới dạng ma trận kề như Hình bên phải. Hãy thực hiện:

a) Trình bày thuật toán DFS(u)?

b) Kiểm nghiệm thuật toán DFS(u) bắt đầu tại đỉnh u=1? Chỉ rõ kết quả trung gian theo mỗi bước thực hiện của thuật toán.

c) Kiểm nghiệm thuật toán DFS(u) bắt đầu tại đỉnh u=7? Chỉ rõ kết quả trung gian theo mỗi bước thực hiện của thuật toán.

0	1	1	1	1	0	0	0	0	0	0	0	0
1	0	1	1	0	0	0	0	0	0	0	0	0
1	1	0	1	0	0	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	1	1	1	1	0	0	0	0
0	0	0	0	1	0	1	0	1	0	0	0	0
0	0	0	0	1	1	0	1	0	1	0	0	0
0	0	0	0	1	0	1	0	1	0	0	0	0
0	0	0	0	1	1	0	1	0	0	0	0	0
0	0	0	0	0	0	1	0	0	0	1	1	1
0	0	0	0	0	0	0	0	0	1	0	1	1
0	0	0	0	0	0	0	0	0	1	1	0	1
0	0	0	0	0	0	0	0	0	1	1	1	0

3. Cho đồ thị vô hướng được biểu diễn dưới dạng ma trận kề như Hình bên phải. Hãy thực hiện:

a) Trình bày thuật toán duyệt các thành phần liên thông của đồ thị?

b) Kiểm nghiệm thuật toán trên đồ thị đã cho? Chỉ rõ kết quả trung gian theo mỗi bước thực hiện của thuật toán.

0	0	1	0	1	0	1	0	0	0	0	0	0
0	0	0	1	0	1	0	0	0	0	0	0	0
1	0	0	0	1	0	1	0	0	0	1	0	0
0	1	0	0	0	1	0	1	0	1	0	0	0
1	0	1	0	0	0	1	0	1	0	1	0	1
0	1	0	1	0	0	0	1	0	1	0	0	0
1	0	1	0	1	0	0	0	1	0	0	0	0
0	0	0	1	0	1	0	0	0	1	0	1	0
0	0	0	0	1	0	1	0	0	0	1	0	1
0	0	0	1	0	1	0	1	0	0	0	1	0
0	0	1	0	1	0	0	0	1	0	0	0	1
0	0	0	0	0	0	0	1	0	1	0	0	0
0	0	0	0	1	0	0	0	1	0	1	0	0

4. Cho đồ thị vô hướng được biểu diễn dưới dạng ma trận kề như Hình bên phải. Hãy thực hiện:

a) Dựa vào thuật toán BFS, xây dựng thuật toán tìm đường đi từ đỉnh s đến đỉnh t trên đồ thị?

b) Tìm đường đi từ đỉnh s=1 đến đỉnh t =13 trên đồ thị đã cho? Chỉ rõ kết quả trung gian theo mỗi bước thực hiện của thuật toán.

c) Viết chương trình tìm đường đi từ s đến t dựa vào biểu diễn đồ thị dưới dạng ma trận kề.

0	1	1	1	1	0	0	0	0	0	0	0	0
1	0	1	1	0	0	0	0	0	0	0	0	0
1	1	0	1	0	0	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	1	1	1	1	0	0	0	0
0	0	0	0	1	0	1	0	1	0	0	0	0
0	0	0	0	1	1	0	1	0	1	0	0	0
0	0	0	0	1	0	1	0	1	0	0	0	0
0	0	0	0	1	1	0	1	0	0	0	0	0
0	0	0	0	0	0	1	0	0	0	1	1	1
0	0	0	0	0	0	0	0	0	1	0	1	1
0	0	0	0	0	0	0	0	0	1	1	0	1
0	0	0	0	0	0	0	0	0	1	1	1	0

5. Cho đồ thị vô hướng được biểu diễn dưới dạng ma trận kề như Hình bên phải. Hãy thực hiện:

a) Dựa vào thuật toán DFS, xây dựng thuật toán tìm đường đi từ đỉnh s đến đỉnh t trên đồ thị?

b) Tìm đường đi từ đỉnh s=1 đến đỉnh t =13 trên đồ thị đã cho? Chỉ rõ kết quả trung gian theo mỗi bước thực hiện của thuật toán.

c) Viết chương trình tìm đường đi từ s đến t dựa vào biểu diễn đồ thị dưới dạng ma trận kề.

0	1	1	1	1	0	0	0	0	0	0	0	0
1	0	1	1	0	0	0	0	0	0	0	0	0
1	1	0	1	0	0	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	1	1	1	1	0	0	0	0
0	0	0	0	1	0	1	0	1	0	0	0	0
0	0	0	0	1	1	0	1	0	1	0	0	0
0	0	0	0	1	0	1	0	1	0	0	0	0
0	0	0	0	1	1	0	1	0	0	0	0	0
0	0	0	0	0	0	1	0	0	0	1	1	1
0	0	0	0	0	0	0	0	0	1	0	1	1
0	0	0	0	0	0	0	0	0	1	1	0	1
0	0	0	0	0	0	0	0	0	1	1	1	0

6. Cho đồ thị có hướng được biểu diễn dưới dạng ma trận kề như Hình bên phải. Hãy thực hiện:

a) Dựa vào thuật toán DFS, xây dựng thuật toán kiểm tra tính liên thông mạnh của đồ thị?

b) Kiểm nghiệm thuật toán trên đồ thị đã cho? Chỉ rõ kết quả trung gian theo mỗi bước thực hiện của thuật toán.

c) Viết chương trình kiểm tra tính liên thông mạnh của đồ thị dựa vào biểu diễn ma trận kề.

0	0	0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	0	0	0	1	0	0	0	0	0
0	0	0	0	0	0	0	0	1	0	0	0	1
1	0	0	0	0	1	0	0	0	0	0	0	0
0	0	0	0	0	0	1	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1	0	1	0
0	0	0	0	0	0	0	0	0	0	1	0	1
0	0	0	1	0	0	0	0	0	0	0	1	0
0	0	0	0	1	0	1	0	0	0	0	0	0
0	1	1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	1	0	0	0	0	0
0	0	0	1	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	0	0	1	0	1	0	0

7. Cho đồ thị có hướng được biểu diễn dưới dạng ma trận kề như Hình bên phải. Hãy thực hiện:

a) Dựa vào thuật toán BFS, xây dựng thuật toán kiểm tra tính liên thông mạnh của đồ thị?

b) Kiểm nghiệm thuật toán trên đồ thị đã cho? Chỉ rõ kết quả trung gian theo mỗi bước thực hiện của thuật toán.

c) Viết chương trình kiểm tra tính liên thông mạnh của đồ thị dựa vào biểu diễn ma trận kề.

0	0	0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	0	0	0	1	0	0	0	0	0
0	0	0	0	0	0	0	0	1	0	0	0	1
1	0	0	0	0	1	0	0	0	0	0	0	0
0	0	0	0	0	0	1	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1	0	1	0
0	0	0	0	0	0	0	0	0	0	1	0	1
0	0	0	1	0	0	0	0	0	0	0	1	0
0	0	0	0	1	0	1	0	0	0	0	0	0
0	1	1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	1	0	0	0	0	0
0	0	0	1	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	0	0	1	0	1	0	0

8. Cho đồ thị vô hướng được biểu diễn dưới dạng ma trận kề như Hình bên phải. Hãy thực hiện:

a) Dựa vào thuật toán BFS, xây dựng thuật toán duyệt các đỉnh trụ của đồ thị?

b) Kiểm nghiệm thuật toán trên đồ thị đã cho? Chỉ rõ kết quả trung gian theo mỗi bước thực hiện của thuật toán.

c) Viết chương trình kiểm tra tính liên thông mạnh của đồ thị dựa vào biểu diễn ma trận kề.

0	1	0	0	0	0	1	1	1	1	0	0	0
1	0	1	0	0	0	1	0	1	0	0	0	0
0	1	0	1	1	1	0	0	0	0	0	0	0
0	0	1	0	1	1	0	0	0	0	0	0	0
0	0	1	1	0	1	0	0	0	0	0	0	0
0	0	1	1	1	0	0	0	0	0	0	0	0
1	1	0	0	0	0	0	1	0	0	0	0	0
1	0	0	0	0	0	1	0	1	0	0	0	0
1	1	0	0	0	0	0	1	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	1	1	1
0	0	0	0	0	0	0	0	0	1	0	1	1
0	0	0	0	0	0	0	0	0	1	1	0	1
0	0	0	0	0	0	0	0	0	1	1	1	0

9. Cho đồ thị vô hướng được biểu diễn dưới dạng ma trận kề như Hình bên phải. Hãy thực hiện:

a) Dựa vào thuật toán DFS, xây dựng thuật toán duyệt các đỉnh trụ của đồ thị?

b) Kiểm nghiệm thuật toán trên đồ thị đã cho? Chỉ rõ kết quả trung gian theo mỗi bước thực hiện của thuật toán.

c) Viết chương trình kiểm tra tính liên thông mạnh của đồ thị dựa vào biểu diễn ma trận kề.

0	1	0	0	0	0	1	1	1	1	0	0	0
1	0	1	0	0	0	1	0	1	0	0	0	0
0	1	0	1	1	1	0	0	0	0	0	0	0
0	0	1	0	1	1	0	0	0	0	0	0	0
0	0	1	1	0	1	0	0	0	0	0	0	0
0	0	1	1	1	0	0	0	0	0	0	0	0
1	1	0	0	0	0	0	1	0	0	0	0	0
1	0	0	0	0	0	1	0	1	0	0	0	0
1	1	0	0	0	0	0	1	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	1	1	1
0	0	0	0	0	0	0	0	0	1	0	1	1
0	0	0	0	0	0	0	0	0	1	1	0	1
0	0	0	0	0	0	0	0	0	1	1	1	0

10. Cho đồ thị vô hướng được biểu diễn dưới dạng ma trận kề như Hình bên phải. Hãy thực hiện:

a) Dựa vào thuật toán BFS, xây dựng thuật toán duyệt các cạnh cầu của đồ thị?

b) Kiểm nghiệm thuật toán trên đồ thị đã cho? Chỉ rõ kết quả trung gian theo mỗi bước thực hiện của thuật toán.

c) Viết chương trình kiểm tra tính liên thông mạnh của đồ thị dựa vào biểu diễn ma trận kề.

0	1	0	0	0	0	1	1	1	1	0	0	0
1	0	1	0	0	0	1	0	1	0	0	0	0
0	1	0	1	1	1	0	0	0	0	0	0	0
0	0	1	0	1	1	0	0	0	0	0	0	0
0	0	1	1	0	1	0	0	0	0	0	0	0
0	0	1	1	1	0	0	0	0	0	0	0	0
1	1	0	0	0	0	0	1	0	0	0	0	0
1	0	0	0	0	0	1	0	1	0	0	0	0
1	1	0	0	0	0	0	1	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	1	1	1
0	0	0	0	0	0	0	0	0	1	0	1	1
0	0	0	0	0	0	0	0	0	1	1	0	1
0	0	0	0	0	0	0	0	0	1	1	1	0

11. Cho đồ thị vô hướng liên thông G =<V, E> như dưới đây:

Ke(1) = { 2, 3, 4}.	Ke(5) = {3, 6, 7, 8, 12}.	Ke(9) = {10, 11, 13}.
Ke(2) = {1, 3, 4, 6}.	Ke(6) = {2, 5, 7, 12}.	Ke(10) = {9, 11, 12, 13}.
Ke(3) = {1, 2, 4, 5}.	Ke(7) = {4, 5, 6, 8}.	Ke(11) = {9, 10, 13}.
Ke(4) = {1, 2, 3, 7}.	Ke(8) = {5, 7, 12}.	Ke(12) = {5, 6, 8, 10}.
		Ke(13) = {9, 10, 11}.

Hãy thực hiện:

a) Tìm BFS(1) =?

b) Tìm BFS(5) =?

c) Tìm DFS(1) =?

d) Tìm DFS(5) =?

d) Tìm đường đi từ 1 đến 13 bằng thuật toán BFS?

e) Tìm đường đi từ 1 đến 13 bằng thuật toán DFS?

12. Cho đồ thị vô hướng liên thông G =<V, E>. Ta gọi đỉnh $s \in V$ là đỉnh "*thắt*" của cặp đỉnh $u, v \in V$ nếu mọi đường đi từ *u* đến *v* đều phải qua *s*. Dựa vào thuật toán duyệt theo chiều sâu (hoặc chiều rộng), hãy thực hiện:

a) Xây dựng thuật toán tìm tất cả các đỉnh *thắt* $s \in V$ của cặp đỉnh $u, v \in V$?

b) Tìm tập đỉnh *thắt* $s \in V$ của cặp đỉnh u=1, $v=12$ trên đồ thị đã cho, chỉ rõ kết quả theo mỗi bước thực hiện của thuật toán?

c) Tìm tập đỉnh *thắt* $s \in V$ của cặp đỉnh $u = 1$, $v = 13$ trên đồ thị được biểu diễn dưới dạng danh sách kề dưới đây, chỉ rõ kết quả theo mỗi bước thực hiện của thuật toán?

Ke(1) = { 2, 3, 4}.	Ke(5) = {3, 6, 7, 8, 12}.	Ke(9) = {10, 11, 13}.
Ke(2) = {1, 3, 4, 6}.	Ke(6) = {2, 5, 7, 12}.	Ke(10) = {9, 11, 12, 13}.
Ke(3) = {1, 2, 4, 5}.	Ke(7) = {4, 5, 6, 8}.	Ke(11) = {9, 10, 13}.
Ke(4) = {1, 2, 3, 7}.	Ke(8) = {5, 7, 12}.	Ke(12) = {5, 6, 8, 10}.
		Ke(13) = {9, 10, 11}.

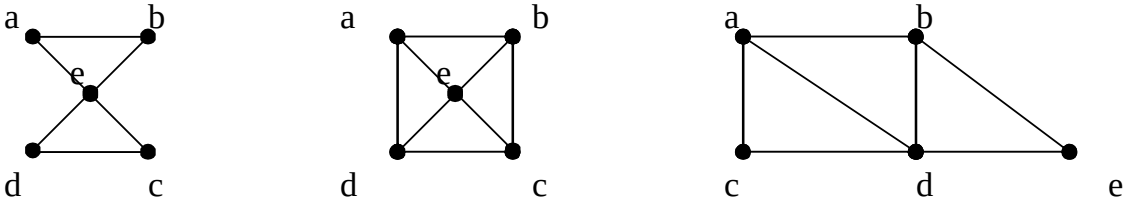
CHƯƠNG 4. ĐỒ THỊ EULER, ĐỒ THỊ HAMIL TON

4.1. Đồ thị Euler, đồ thị nửa Euler

Định nghĩa. Chu trình đơn trong đồ thị G đi qua mỗi cạnh của đồ thị đúng một lần được gọi là chu trình Euler. Đường đi đơn trong G đi qua mỗi cạnh của nó đúng một lần được gọi là đường đi Euler. Đồ thị được gọi là đồ thị Euler nếu nó có chu trình Euler. Đồ thị có đường đi Euler được gọi là nửa Euler.

Rõ ràng, mọi đồ thị Euler đều là nửa Euler nhưng điều ngược lại không đúng.

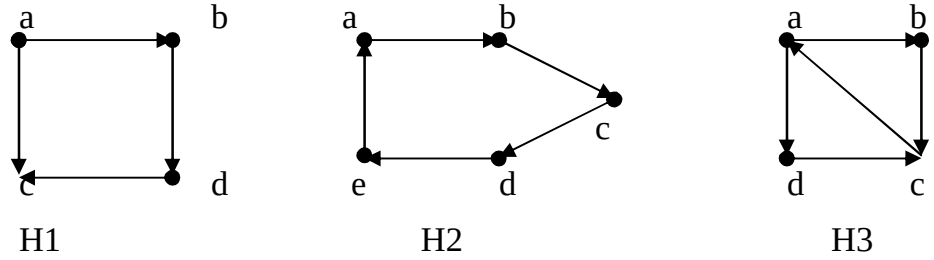
Ví dụ 1. Xét các đồ thị G1, G2, G3 trong Hình 4.1.



Hình 6.1. Đồ thị vô hướng G1, G2, G3.

Đồ thị G1 là đồ thị Euler vì nó có chu trình Euler a, e, c, d, e, b, a. Đồ thị G3 không có chu trình Euler nhưng chứa đường đi Euler a, c, d, e, b, d, a, b vì thế G3 là nửa Euler. G2 không có chu trình Euler cũng như đường đi Euler.

Ví dụ 2. Xét các đồ thị có hướng H1, H2, H3 trong Hình 4.2.



Hình 4.2. Đồ thị có hướng H1, H2, H3.

Đồ thị H2 là đồ thị Euler vì nó chứa chu trình Euler a, b, c, d, e, a vì vậy nó là đồ thị Euler. Đồ thị H3 không có chu trình Euler nhưng có đường đi Euler a, b, c, a, d, c nên nó là đồ thị nửa Euler. Đồ thị H1 không chứa chu trình Euler cũng như chu trình Euler.

4.2. Thuật toán tìm chu trình Euler

Để tìm một chu trình Euler của đồ thị ta sử dụng kết quả của định lý sau.

Định lý 1. *Điều kiện cần và đủ để đồ thị G=<V,E> là Euler***.** Đồ thị vô hướng liên thông G=<V, E> là đồ thị Euler khi và chỉ khi mọi đỉnh của G đều có bậc chẵn. Đồ thị có

hướng liên thông yếu G=<V, E> là đồ thị Euler khi và chỉ khi tất cả các đỉnh của nó đều có bán đỉnh bậc ra bằng bán đỉnh bậc vào (điều này làm cho đồ thị là liên thông mạnh).

4.2.1. Chứng minh đồ thị là Euler

Đối với đồ thị vô hướng, để chứng minh đồ thi có là Euler hay không ta chỉ cần thực hiện:

- Kiểm tra đồ thị có liên thông hay không? Điều này dễ dàng thực hiện bằng cách kiểm tra DFS(u) = V hoặc BFS(u) = V thì ta kết luận đồ thị là liên thông (u là đỉnh bất kỳ của đồ thị).
- Sử dụng tính chất của ma trận kề biểu đồ thị vô hướng để tính toán bậc của các đỉnh.

- Vì BFS(1) = { 1, 2, 6, 3, 5, 7, 4, 11, 8, 10, 12, 9, 13} = V. Do vậy, G liên thông.
- Ta lại có :

deg(1) = deg(13) = 2.
deg (2) = deg(3) = 4
deg(4) = deg(5) = 4
deg(6) = deg(7) = 4
deg(8) = deg(9) = 4
deg(10) = deg(11) = deg(12) = 4

Chú ý: Tổng các phần tử của hàng u (cột u) là bậc của đỉnh u. Ví dụ tổng các phần tử của hàng 1 là 2 nên deg(1) = 2.

0	1	0	0	0	1	0	0	0	0	0	0	0
1	0	1	0	1	1	0	0	0	0	0	0	0
0	1	0	1	1	0	0	0	0	0	1	0	0
0	0	1	0	0	0	1	1	0	0	1	0	0
0	1	1	0	0	1	1	0	0	0	0	0	0
1	1	0	0	1	0	1	0	0	0	0	0	0
0	0	0	1	1	1	0	1	0	0	0	0	0
0	0	0	1	0	0	1	0	1	1	0	0	0
0	0	0	0	0	0	0	1	0	1	0	1	1
0	0	0	0	0	0	0	1	1	0	1	1	0
0	0	1	1	0	0	0	0	0	1	0	1	0
0	0	0	0	0	0	0	0	1	1	1	0	1
0	0	0	0	0	0	0	0	1	0	0	1	0

Đối với đồ thị có hướng, để chứng minh đồ thi có là Euler hay không ta chỉ cần thực hiện:

- Kiểm tra đồ thị có liên thông yếu hay không? Điều này dễ dàng thực hiện bằng cách kiểm tra nếu tồn tại đỉnh u$\in$ V để DFS(u) = V hoặc BFS(u) = V thì ta kết luận đồ thị là liên thông yếu.
- Sử dụng tính chất của ma trận kề biểu đồ thị có hướng để tính bán đỉnh bậc ra và bán đỉnh bậc vào của các đỉnh. Bán đỉnh bậc ra của đỉnh u là deg+(u) là số các số 1 của hàng u. Bán đỉnh bậc vào của đỉnh u là deg-(u) là số các số 1 của cột u.

Ví dụ để chứng minh đồ thị có hướng được biểu diễn dưới dạng ma trận kề như dưới đây ta thực hiện như sau:

+ Vì BFS(1) = { 1, 2, 3, 5, 4, 11, 6, 7, 10, 12, 8, 9, 13} = V. Do vậy, G liên thông yếu.
+ Ta lại có:

$deg^+(2)= deg^-(2)=deg^+(3)= eg^-(3) =2$
$deg^+(4)=deg^-(4)=deg^+(5)=deg^-(5) =2$
$deg^+(6)=deg^-(6)=deg^+(7)=deg^-(7) =2$
$deg^+(8)=deg^-(8)=deg^+(9)=deg^-(9) =2$
$deg^+(10) = deg^-(10) = 2$
$deg^+(11) = deg^-(11) =2$
$deg^+(12) = deg^-(12) =2$
$deg^+(1)=deg^-(1)=deg^-(13)=deg^+(13) =1$

0	1	0	0	0	0	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0	0	0	1	0	0
0	0	0	0	0	0	1	0	0	0	1	0	0
0	0	1	0	0	1	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	1	0	0	0	0	0	0	0
0	0	0	1	0	0	1	0	0	0	0	0	0
0	0	0	0	0	0	0	1	0	1	0	0	0
0	0	0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	0	0	0	0	0	1	0	1	0
0	0	0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	0	0	0	1	0	0	0	0

G liên thông yếu và có bán đỉnh bậc ra bằng bán đỉnh bậc vào nên G là đồ thị Euler.

4.2.2. Biểu diễn thuật toán tìm chu trình Euler

Để tìm một chu trình Euler trên đồ thị, ta thực hiện theo thuật toán trong Hình 4.3 dưới đây:

Thuật toán Euler-Cycle(u):

Bước 1 (Khởi tạo) :

```
stack = ∅ ;    //Khởi tạo một stack bắt đầu là ∅
CE = ∅ ;       //Khởi tạo mảng CE bắt đầu là ∅
Push (stack, u) ; //Đưa đỉnh u vào ngăn xếp
```

Bước 2 (Lặp):

```
while (stack≠∅ ) do { //Lặp cho đến khi stack rỗng
    s = get(stack); //Lấy đỉnh ở đầu ngăn xếp
    if ( Ke(s) ≠ ∅ ) then { // Nếu danh sách Ke(s) chưa rỗng
        t =< Đỉnh đầu tiên trong Ke(s)>;
        Push(stack, t)' //Đưa t vào stack;
        E = E \ (s,t); // Loại bỏ cạnh (s,t);
    }
    else { //Trường hợp Ke(s)=∅
        s = Pop(stack);// Đưa s ra khỏi ngăn xếp
        s ⇒CE; //Đưa s sang CE
    }
}
```

Bước 3 (Trả lại kết quả) :

```
<Lật ngược lại các đỉnh trong CE ta được chu trình Euler> ;
```

Hình 4.3. Thuật toán tìm chu trình Euler bắt đầu tại đỉnh u

4.2.3. Kiểm nghiệm thuật toán

Ví dụ ta cần tìm một chu trình Euler bắt đầu tại đỉnh u=1 trên đồ thị G = <V, E> được biểu diễn dưới dạng ma trận kề như dưới đây. Khi đó, các bước thực hiện của thuật toán được thực hiện như Bảng 4.1 (chú ý phần chứng minh ta đã thực hiện ở trên).

Bước	**Trạng thái Stack**	**Giá trị CE**
1	1	ϕ
2	1, 2	ϕ
3	1, 2, 3	ϕ
4	1, 2, 3, 4	ϕ
5	1, 2, 3, 4,7	ϕ
6	1, 2, 3, 4,7,5	ϕ
7	1, 2, 3, 4,7,5,2	ϕ
8	1, 2, 3, 4,7,5,2,6	ϕ
9	1, 2, 3, 4,7,5,2,6,1	ϕ
10	1, 2, 3, 4,7,5,2,6	1
11	1, 2, 3, 4,7,5,2,6,5	1
12	1, 2, 3, 4,7,5,2,6,5,3	1
13	1, 2, 3, 4,7,5,2,6,5,3,11	1
14	1, 2, 3, 4,7,5,2,6,5,3,11,4	1
15	1, 2, 3, 4,7,5,2,6,5,3,11,4,8	1
16	1, 2, 3, 4,7,5,2,6,5,3,11,4,8,7	1
17	1, 2, 3, 4,7,5,2,6,5,3,11,4,8,7,6	1
18	1, 2, 3, 4,7,5,2,6,5,3,11,4,8,7	1,6
19	1, 2, 3, 4,7,5,2,6,5,3,11,4,8	1,6,7
20	1, 2, 3, 4,7,5,2,6,5,3,11,4,8,9	1,6,7
21	1, 2, 3, 4,7,5,2,6,5,3,11,4,8,9,10	1,6,7
22	1, 2, 3, 4,7,5,2,6,5,3,11,4,8,9,10,8	1,6,7
23	1, 2, 3, 4,7,5,2,6,5,3,11,4,8,9,10	1,6,7,8
24	1, 2, 3, 4,7,5,2,6,5,3,11,4,8,9,10,11	1,6,7,8
25	1, 2, 3, 4,7,5,2,6,5,3,11,4,8,9,10,11,12	1,6,7,8
26	1, 2, 3, 4,7,5,2,6,5,3,11,4,8,9,10,11,12,9	1,6,7,8
27	1, 2, 3, 4,7,5,2,6,5,3,11,4,8,9,10,11,12,9,13	1,6,7,8
28	1, 2, 3, 4,7,5,2,6,5,3,11,4,8,9,10,11,12,9,13,12	1,6,7,8
29	1, 2, 3, 4,7,5,2,6,5,3,11,4,8,9,10,11,12,9,13,12,10	1,6,7,8
Đưa lần lượt các đỉnh trong Stack sang CE cho đến khi stack=∅		
30 -..	CE = 1, 6, 7, 8, 10, 12, 13, 9, 12, 11, 10, 9, 8, 4, 11, 3, 5, 6, 2, 5, 7, 4, 3, 2, 1	
Lật ngược lại các đỉnh trong CE ta được chu trình Euler		
1- 2- 3- 4- 7-5-2-6-5-3-11-4-8-9-10-11-12-9-13-12-10-8-7-6-1		

4.2.4. Cài đặt thuật toán

Chương trình tìm một chu trình Euler của đồ thị bắt đầu tạo đỉnh u trên đồ thị vô hướng liên thông được cài đặt theo khuôn dạng đồ thị biểu diễn dưới dạng ma trận kề. Các thủ tục chính bao gồm:

- Thủ tục Init() : đọc dữ liệu theo khuôn dạng biểu diễn ma trận kề.
- Thủ tục Kiemtra(): Kiểm tra xem G có là Euler hay không.
- Thủ tục Euler-Cycle (u) : Xây dựng chu trình Euler bắt đầu tại đỉnh u.

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <math.h>
#define MAX 50
#define     TRUE 1
#define FALSE     0
int A[MAX][MAX], n, u=1;
void Init(void){
        int i, j;FILE *fp;
        fp = fopen("CTEULER.IN", "r");
        fscanf(fp,"%d", &n);
        printf("\n So dinh do thi:%d",n);
        printf("\n Ma tran ke:");
        for(i=1; i<=n;i++){
                printf("\n");
                for(j=1; j<=n;j++){
                        fscanf(fp,"%d", &A[i][j]);
                        printf("%3d", A[i][j]);
                }
        }
        fclose(fp);
}
int Kiemtra(void){
        int i, j, s, d;    d=0;
        for(i=1; i<=n;i++){
                s=0;
                for(j=1; j<=n;j++)
                        s+=A[i][j];
                if(s%2) d++;
        }
        if(d>0) return(FALSE);
        return(TRUE);
}
void Euler-Cycle( int u){
```

```
        int v, x, top, dCE;
        int stack[MAX], CE[MAX];
        top=1; stack[top]=u;dCE=0;
        do {
                v = stack[top];x=1;
                while (x<=n && A[v][x]==0)
                        x++;
                if (x>n) {
                        dCE++; CE[dCE]=v; top--;
                }
                else {
                        top++; stack[top]=x;
                        A[v][x]=0; A[x][v]=0;
                }
        } while(top!=0);
        printf("\n Co chu trinh Euler:");
        for(x=dCE; x>0; x--)
                printf("%3d", CE[x]);
}
void main(void){
        clrscr(); Init();
        if(Kiemtra())
                Tim();
        else printf("\n Khong co chu trinh Euler");
}
```

4.3. Thuật toán tìm đường đi Euler

Một đồ thị không có chu trình Euler nhưng vẫn có thể có đường đi Euler. Để tìm một đường đi Euler trên đồ thị vô hướng ta sử dụng kết quả của định lý 2. Để tìm một đường đi Euler trên đồ thị có hướng ta sử dụng kết quả của định lý 3.

Định lý 2. Đồ thị vô hướng liên thông G =<V,E> là đồ thị nửa Euler khi và chỉ khi G có 0 hoặc 2 đỉnh bậc lẻ. Trong trường hợp G có hai đỉnh bậc lẻ, đường đi Euler xuất phát tại một đỉnh bậc lẻ và kết thúc tại đỉnh bậc lẻ còn lại. Trong trường hợp G có 0 đỉnh bậc lẻ G chính là đồ thị Euler.

Định lý 3. Đồ thị có hướng liên thông yếu G =<V,E> là đồ thị nửa Euler khi và chỉ khi tồn tại đúng hai đỉnh u, v $\in$ V sao cho $deg^+(u) - deg^-(u) = deg^-(v) - deg^+(v)=1$, các đỉnh s$\neq$ u, s$\neq$ v còn lại có $deg^+(s) = deg^-(s)$. Đường đi Euler sẽ xuất phát tại đỉnh u và kết thúc tại đỉnh v.

4.3.1. Chứng minh đồ thị là nửa Euler

Để chứng tỏ đồ thị vô hướng G =<V,E> là nửa Euler ta cần thực hiện:

- Chứng tỏ đồ thị đã cho liên thông. Điều này dễ ràng thực hiện được bằng cách sử dụng hai thủ tục DFS(u) hoặc BFS(u).
- Có 0 hoặc hai đỉnh bậc lẻ. Sử dụng tính chất của các phương pháp biểu diễn đồ thị để tìm ra bậc của mỗi đỉnh.

Ví dụ. Chứng minh rằng, đồ thị vô hướng liên thông G =<V,E> được biểu diễn dưới dạng ma trận kề dưới đây là đồ thị nửa Euler.

Chứng minh. Theo tính chất của ma trận kề, tổng các phần tử hàng u là bậc của đỉnh u. Vì vậy ta có:

deg(1) = deg(13) = 3
deg (2) = deg(3) = deg(11) = 4
deg(12) = deg(6) = deg(7) = 4
deg(8) = deg(9) = 4
deg(5) = deg(4) = deg(10) = 6

G liên thông và có 2 đỉnh bậc lẻ u=1 và u=13 nên G là nửa Euler.

0	1	0	0	1	1	0	0	0	0	0	0	0
1	0	1	0	1	1	0	0	0	0	0	0	0
0	1	0	1	1	0	0	0	0	0	1	0	0
0	0	1	0	1	0	1	1	0	1	1	0	0
1	1	1	1	0	1	1	0	0	0	0	0	0
1	1	0	0	1	0	1	0	0	0	0	0	0
0	0	0	1	1	1	0	1	0	0	0	0	0
0	0	0	1	0	0	1	0	1	1	0	0	0
0	0	0	0	0	0	0	1	0	1	0	1	1
0	0	0	1	0	0	0	1	1	0	1	1	1
0	0	1	1	0	0	0	0	0	1	0	1	0
0	0	0	0	0	0	0	0	1	1	1	0	1
0	0	0	0	0	0	0	0	1	1	0	1	0

Để chứng tỏ đồ thị có hướng G =<V,E> là nửa Euler ta cần thực hiện:

- Chứng tỏ đồ thị đã cho liên thông yếu. Điều này dễ ràng thực hiện được bằng cách sử dụng hai thủ tục DFS(u) hoặc BFS(u).
- Có hai đỉnh u và v thỏa mãn $deg^+(u) - deg^-(u) = deg^-(v) - deg^+(v) = 1$.
- Các đỉnh $s \neq u$, $s \neq v$ còn lại có $deg^+(s) = deg^-(s)$.

Ví dụ. Chứng minh rằng, đồ thị có hướng liên thông yếu G =<V,E> được biểu diễn dưới dạng ma trận kề dưới đây là đồ thị nửa Euler?

Chứng minh. Theo tính chất của ma trận kề, $deg^+(u)$ là tổng các phần tử hàng u, $deg^-(u)$ là tổng các phần tử cột u. Vì vậy ta có:

$deg^+(2) = deg^-(2) = deg^+(3) = deg^-(3) = 2$
$deg^+(6) = deg^-(6) = deg^+(7) = deg^-(7) = 2$
$deg^+(8) = deg^-(8) = deg^+(9) = deg^-(9) = 2$
$deg^+(11) = deg^-(11) = deg^+(12) = deg^-(12) = 2$
$deg^+(5) = deg^-(5) = deg^+(4) = deg^-(4) = deg^+(10) = deg^-(10) = 3$
$deg^+(1) - deg^-(1) = deg^-(13) - deg^+(13) = 1$

G liên thông yếu và có 2 đỉnh u=1 và u=13 thỏa mãn điều kiện nên G là nửa Euler.

0	1	0	0	1	0	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0	0	0	1	0	0
0	0	0	0	0	0	1	0	0	1	1	0	0
0	0	1	1	0	1	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	1	0	0	0	0	0	0	0
0	0	0	1	0	0	1	0	0	0	0	0	0
0	0	0	0	0	0	0	1	0	1	0	0	0
0	0	0	0	0	0	0	1	0	0	0	1	1
0	0	0	0	0	0	0	0	0	1	0	1	0
0	0	0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	0	0	0	1	0	0	0	0

4.3.2. Thuật toán tìm đường đi Euler

Thuật toán tìm đường đi Euler và chu trình Euler chỉ khác nhau duy nhất ở một điểm đó là đầu vào của thuật toán. Đối với thuật toán tìm chu trình Euler, đầu vào thuật toán là đỉnh u∈V bất kỳ. Đối với thuật toán tìm đường đi trình Euler, đầu vào thuật toán là đỉnh u∈V là đỉnh bậc lẻ đầu tiên trong trường hợp đồ thị vô hướng. Đối với đồ thị có hướng, đỉnh u∈V là đỉnh có $deg^+(u)-deg^-(u)=1$. Thuật toán tìm đường đi Euler trên đồ thị vô hướng hoặc có hướng được mô tả chi tiết trong Hình 4.4.

Thuật toán Euler-Path (u):

- u là đỉnh bậc lẻ đầu tiên nếu G là đồ thị vô hướng
- u là đỉnh có $deg^+(u) - deg^-(u) =1$.

Bước 1 (Khởi tạo) :

```
stack = ∅ ;    //Khởi tạo một stack bắt đầu là ∅
dCE = ∅ ;      //Khởi tạo mảng dCE bắt đầu là ∅
Push (stack, u) ; //Đưa đỉnh u vào ngăn xếp
```

Bước 2 (Lặp):

```
while (stack≠∅ ) do { //Lặp cho đến khi stack rỗng
        s = get(stack); //Lấy đỉnh ở đầu ngăn xếp
        if ( Ke(s) ≠ ∅ ) then { // Nếu danh sách Ke(s) chưa rỗng
                t =< Đỉnh đầu tiên trong Ke(s)>;
                Push(stack, t)' //Đưa t vào stack;
                E = E \ (s,t); // Loại bỏ cạnh (s,t);
        }
        else { //Trường hợp Ke(s)=∅
                s = Pop(stack);// Đưa s ra khỏi ngăn xếp
                s ⇒dCE; //Đưa s sang dCE
        }
}
```

Bước 3 (Trả lại kết quả) :

```
<Lật ngược lại các đỉnh trong dCE ta được đường đi Euler> ;
```

Hình 4.4. Thuật toán tìm đường đi Euler trên đồ thị.

4.3.3. Kiểm nghiệm thuật toán

Ví dụ ta cần tìm đường đi Euler trên đồ thị có hướng liên thông yếu được biểu diễn dưới dạng ma trận kề trong Mục 4.3.1. Khi đó, đỉnh u có deg+(u)-deg-(u)=1 là đỉnh 1. Kết quả thực hiện của thuật toán Hình 4.4 được thể hiện trong Bảng 4.2 dưới đây.

Bước	Trạng thái Stack	Giá trị dCE
1	1	ϕ
2	1, 2	
3	1, 2, 3	
4	1, 2, 3, 4	
5	1, 2, 3, 4,7	
6	1, 2, 3, 4,7,5	
7	1, 2, 3, 4,7,5,3	
8	1, 2, 3, 4,7,5,3,11	
9	1, 2, 3, 4,7,5,3,11,10	
10	1, 2, 3, 4,7,5,3,11,10,8	
11	1, 2, 3, 4,7,5,3,11,10,8,4	
12	1, 2, 3, 4,7,5,3,11,10,8,4,10	
13	1, 2, 3, 4,7,5,3,11,10,8,4,10,12	
14	1, 2, 3, 4,7,5,3,11,10,8,4,10,12,9	
15	1, 2, 3, 4,7,5,3,11,10,8,4,10,12,9,8	
16	1, 2, 3, 4,7,5,3,11,10,8,4,10,12,9,8,7	
17	1, 2, 3, 4,7,5,3,11,10,8,4,10,12,9,8,7,6	
18	1, 2, 3, 4,7,5,3,11,10,8,4,10,12,9,8,7,6,1	
19	1, 2, 3, 4,7,5,3,11,10,8,4,10,12,9,8,7,6,1,5	
20	1, 2, 3, 4,7,5,3,11,10,8,4,10,12,9,8,7,6,1,5,4	
21	1, 2, 3, 4,7,5,3,11,10,8,4,10,12,9,8,7,6,1,5,4,11	
22	1, 2, 3, 4,7,5,3,11,10,8,4,10,12,9,8,7,6,1,5,4,11,12	
23	1, 2, 3, 4,7,5,3,11,10,8,4,10,12,9,8,7,6,1,5,4,11,12,13	
24	1, 2, 3, 4,7,5,3,11,10,8,4,10,12,9,8,7,6,1,5,4,11,12,13,9	
25	1, 2, 3, 4,7,5,3,11,10,8,4,10,12,9,8,7,6,1,5,4,11,12,13,9,10	
26	1, 2, 3, 4,7,5,3,11,10,8,4,10,12,9,8,7,6,1,5,4,11,12,13,9,10,13	
27	1, 2, 3, 4,7,5,3,11,10,8,4,10,12,9,8,7,6,1,5,4,11,12,13,9,10	13,
28	1, 2, 3, 4,7,5,3,11,10,8,4,10,12,9,8,7,6,1,5,4,11,12,13,9	13,10
29	1, 2, 3, 4,7,5,3,11,10,8,4,10,12,9,8,7,6,1,5,4,11,12,13	13,10,9
30	1, 2, 3, 4,7,5,3,11,10,8,4,10,12,9,8,7,6,1,5,4,11,12	13,10,9,13
31	1, 2, 3, 4,7,5,3,11,10,8,4,10,12,9,8,7,6,1,5,4,11	13,10,9,13,12
32	1, 2, 3, 4,7,5,3,11,10,8,4,10,12,9,8,7,6,1,5,4	13,10,9,13,12,11
33	1, 2, 3, 4,7,5,3,11,10,8,4,10,12,9,8,7,6,1,5	13,10,9,13,12,11,4
34	1, 2, 3, 4,7,5,3,11,10,8,4,10,12,9,8,7,6,1,5,6	13,10,9,13,12,11,4
35	1, 2, 3, 4,7,5,3,11,10,8,4,10,12,9,8,7,6,1,5,6,2	13,10,9,13,12,11,4
36	1, 2, 3, 4,7,5,3,11,10,8,4,10,12,9,8,7,6,1,5,6,2,5	13,10,9,13,12,11,4
Đưa lần lượt các đỉnh trong Stack sang dCE		
37 ...	dCE = 13,10,9,13,12,11,4, 5, 2, 6, 5, 1, 6, 7, 8, 9, 12, 10, 4, 8, 10, 11, 3, 5, 7, 4, 3, 2, 1	
Lật ngược lại các đỉnh trong CE ta được đường đi Euler		
1- 2- 3- 4- 7- 5- 3- 11- 10- 8 -4- 10- 12- 9- 8- 7- 6- 1-5- 6- 2- 5- 4-11- 12- 13-9-10-13		

4.3.4. Cài đặt thuật toán

Chương trình tìm một đường đi Euler của đồ thị bắt đầu tạo đỉnh u trên đồ thị vô hướng liên thông được cài đặt theo khuôn dạng đồ thị biểu diễn dưới dạng ma trận kề. Các thủ tục chính bao gồm:

- Thủ tục Init() : đọc dữ liệu theo khuôn dạng biểu diễn ma trận kề.
- Thủ tục Kiemtra(): Kiểm tra xem G có là nửa Euler hay không.
- Thủ tục Euler-Cycle (u) : Xây dựng đường đi Euler bắt đầu tại đỉnh u (đỉnh bậc lẻ đầu tiên).

Chương trình tìm đường đi Euler được thể hiện như sau:

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <math.h>
#include <dos.h>
#define MAX 50
#define    TRUE 1
#define FALSE   0
void Init(int A[][MAX], int *n){
    int i, j;FILE *fp;
    fp = fopen("DDEULER.IN", "r");
    fscanf(fp,"%d", n);
    printf("\n So dinh do thi:%d",*n);
    printf("\n Ma tran ke:");
    for(i=1; i<=*n;i++){
            printf("\n");
            for(j=1; j<=*n;j++){
                    fscanf(fp,"%d", &A[i][j]);
                    printf("%3d", A[i][j]);
            }
    }
    fclose(fp);
}
int Kiemtra(int A[][MAX], int n, int *u){
    int i, j, s, d;
    d=0;
    for(i=1; i<=n;i++){
            s=0;
            for(j=1; j<=n;j++)
                    s+=A[i][j];
            if(s%2){
```

```
                d++;*u=i;
            }
        }
        if(d!=2) return(FALSE);
        return(TRUE);
    }
    void DDEULER(int A[][MAX], int n, int u){
        int v, x, top, dCE;
        int stack[MAX], CE[MAX];
        top=1; stack[top]=u;dCE=0;
        do {
            v = stack[top];x=1;
            while (x<=n && A[v][x]==0)
                x++;
            if (x>n) {
                dCE++; CE[dCE]=v; top--;
            }
            else {
                top++; stack[top]=x;
                A[v][x]=0; A[x][v]=0;
            }
        } while(top!=0);
        printf("\n Co duong di Euler:");
        for(x=dCE; x>0; x--)
            printf("%3d", CE[x]);
    }
    void main(void){
        int A[MAX][MAX], n, u;
        clrscr(); Init(A, &n);
        if(Kiemtra(A,n,&u))
            DDEULER(A,n,u);
        else printf("\n Khong co duong di Euler");
        getch();
    }
```

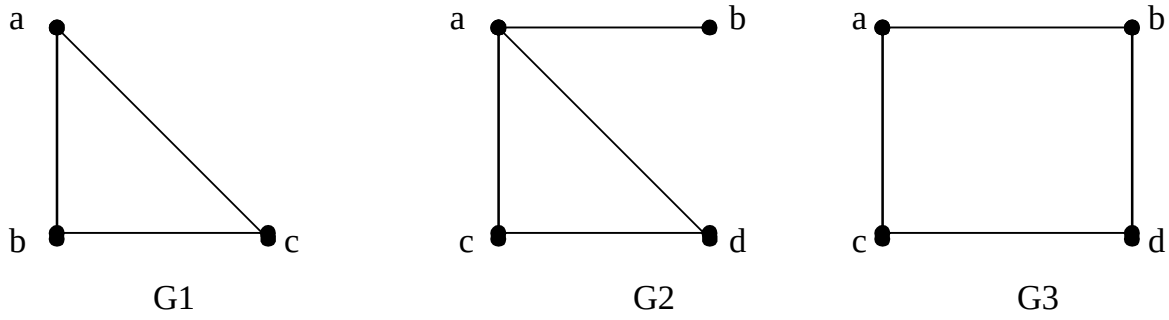
4.4. Đồ thị Hamilton

Với đồ thị Euler, chúng ta quan tâm tới việc duyệt các cạnh của đồ thị mỗi cạnh đúng một lần, thì trong mục này, chúng ta xét đến một bài toán tương tự nhưng chỉ khác nhau là ta chỉ quan tâm tới các đỉnh của đồ thị, mỗi đỉnh đúng một lần. Sự thay đổi này tưởng như không đáng kể, nhưng thực tế có nhiều sự khác biệt trong khi giải quyết bài toán.

Định nghĩa. Đường đi qua tất cả các đỉnh của đồ thị mỗi đỉnh đúng một lần được gọi là đường đi Hamilton. Chu trình bắt đầu tại một đỉnh *v* nào đó qua tất cả các đỉnh còn lại mỗi đỉnh đúng một lần sau đó quay trở lại *v* được gọi là chu trình Hamilton. Đồ thị có chu trình Hamilton được gọi là đồ thị Hamilton. Đồ thị có đường đi Hamilton được gọi là đồ thị nửa Hamilton.

Như vậy, một đồ thị Hamilton bao giờ cũng là đồ thị nửa Hamilton nhưng điều ngược lại không luôn luôn đúng. Ví dụ sau sẽ minh họa cho nhận xét này.

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



Hình 4.5. Đồ thị đồ thi hamilton G3, nửa Hamilton G2 và G1.

4.4.1. Thuật toán tìm tất cả các chu trình Hamilton

Cho đến nay, việc tìm ra một tiêu chuẩn để nhận biết đồ thị Hamilton vẫn còn mở, mặc dù đây là vấn đề trung tâm của lý thuyết đồ thị. Cho đến nay cũng vẫn chưa có thuật toán hiệu quả để kiểm tra một đồ thị có phải là đồ thị Hamilton hay không. Hình 4.6 dưới đây mô tả thuật toán liệt kê tất cả chu trình Hamilton bắt đầu tại đỉnh k.

```
Thuật toán Hamilton( int k) {
/* Liệt kê các chu trình Hamilton của đồ thị bằng cách phát triển dãy đỉnh
(X[1], X[2], . . ., X[k-1] ) của đồ thị G = (V, E) */
        for y∈ Ke(X[k-1]) {
                if (k==n+1) and (y == v0) then
                    Ghinhan(X[1], X[2], . . ., X[n], v0);
                else {
                        X[k]=y; chuaxet[y] = false;
                        Hamilton(k+1);
                        chuaxet[y] = true;
                }
        }
}
```

Hình 4.6. Thuật toán liệt kê các chu trình Hamilton bắt đầu tại đỉnh k.

Khi đó, việc liệt kê chu trình Hamilton được thực hiện như sau:

Begin

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

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

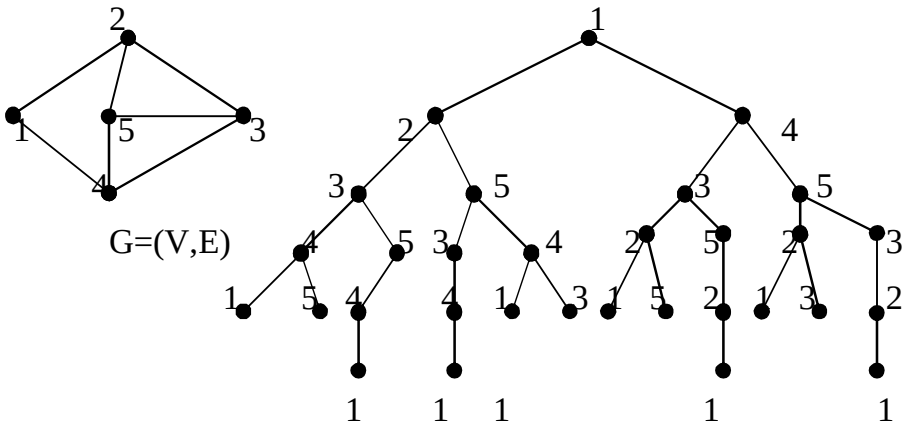
chuaxet[v0] = false;

Hamilton(2);

End.

4.4.2. Kiểm nghiệm thuật toán

Ví dụ với đồ thị G=<V, E> dưới đây sẽ cho ta cây tìm kiếm chu trình Hamilton thể hiện thuật toán trên được mô tả như trong Hình 4.6.



Hình 4.7. Cây tìm kiếm chu trình Hamilton.

4.4.3. Cài đặt thuật toán

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

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <math.h>
#include <dos.h>
#define MAX 50
#define     TRUE 1
#define FALSE     0
int A[MAX][MAX], C[MAX], B[MAX];
int n,i, d;
void Init(void){
    int i, j;FILE *fp;
```

```
        fp= fopen("CCHMTON.IN", "r");
        if(fp==NULL){
                printf("\n Khong co file input");
                getch(); return;
        }
        fscanf(fp,"%d",&n);
        printf("\n So dinh do thi:%d", n);
        printf("\n Ma tran ke:");
        for(i=1; i<=n; i++){
                printf("\n");
                for(j=1; j<=n; j++){
                        fscanf(fp, "%d", &A[i][j]);
                        printf("%3d", A[i][j]);
                }
        }
        fclose(fp);
        for (i=1; i<=n;i++)
                C[i]=0;
}
void Result(void){
        int i;
        printf("\n ");
        for(i=n; i>=0; i--)
                printf("%3d", B[i]);
        d++;
}
void Hamilton(int *B, int *C, int i){
        int j, k;
        for(j=1; j<=n; j++){
                if(A[B[i-1]][j]==1 && C[j]==0){
                        B[i]=j; C[j]=1;
                        if(i<n) Hamilton(B, C, i+1);
                        else if(B[i]==B[0]) Result();
                        C[j]=0;
                }
        }
}
void main(void){
        B[0]=1; i=1;d=0;    Init();
        Hamilton(B,C,i);
        if(d==0)                printf("\n Khong co chu trinh Hamilton");
        }
```

4.4.3. Cài đặt thuật toán

Cũng giống như thuật toán tìm chu trình Hamilton, thuật toán tìm đường đi Hamilton được cài đặt như sau:

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <math.h>
#include <dos.h>
#define MAX 50
#define	TRUE 1
#define FALSE	0
int A[MAX][MAX], C[MAX], B[MAX];
int n,i, d;
void Init(void){
	int i, j;FILE *fp;
	fp= fopen("DDHMTON.IN", "r");
	if(fp==NULL){
		printf("\n Khong co file input");
		getch(); return;
	}
	fscanf(fp,"%d",&n);
	printf("\n So dinh do thi:%d", n);
	printf("\n Ma tran ke:");
	for(i=1; i<=n; i++){
		printf("\n");
		for(j=1; j<=n; j++){
			fscanf(fp, "%d", &A[i][j]);
			printf("%3d", A[i][j]);
		}
	}
	fclose(fp);
	for (i=1; i<=n;i++)
		C[i]=0;
}
void Result(void){
	int i;
	printf("\n ");
	for(i=n; i>0; i--)
		printf("%3d", B[i]);
	d++;
}
void Hamilton(int *B, int *C, int i){
```

```
        int j, k;
        for(j=1; j<=n; j++){
                if(A[B[i-1]][j]==1 && C[j]==0){
                        B[i]=j; C[j]=1;
                        if(i<n) Hamilton(B, C, i+1);
                        else  Result();
                        C[j]=0;
                }
        }
}
void main(void){
        B[0]=1; i=1;d=0;
   Init();
        Hamilton(B,C,i);
        if(d==0)
                printf("\n Khong co duong di Hamilton");
        getch();
}
```

4.5. Những điểm cần ghi nhớ

- ✓ Khái niệm và định nghĩa về đồ thị Euler, đồ thị nửa Euler, đồ thị Hamilton, đồ thị nửa Hamilton.
- ✓ Nắm vững và phân biệt rõ sự khác biệt giữa chu trình (đường đi) Euler và chu trình (đường đi Hamilton).
- ✓ Phương pháp hiểu rõ bản chất của thuật toán là cài đặt và kiểm chứng thuật toán bằng cách viết chương trình.

BÀI TẬP

1. Cho đồ thị vô hướng liên thông G=<V,E> như hình bên phải. Hãy thực hiện:

a) Chứng minh đồ thị đã cho là Euler?

b) Xây dựng thuật toán tìm một chu trình Euler của đồ thị bắt đầu tại đỉnh u∈V?

c) Tìm một chu trình Euler bắt đầu tại đỉnh u=1? Chỉ rõ kết quả trung gian theo mỗi bước thực hiện của thuật toán?

d) Tìm một chu trình Euler bắt đầu tại đỉnh u=5? Chỉ rõ kết quả trung gian theo mỗi bước thực hiện của thuật toán?

e) Viết chương trình tìm một chu trình Euler của đồ thị bắt đầu tại đỉnh u?

0	0	0	1	0	1	0	0	0	0	0	0	0
0	0	1	0	0	0	0	1	0	1	1	0	0
0	1	0	0	0	0	0	0	1	1	0	0	1
1	0	0	0	0	1	0	1	0	0	0	1	0
0	0	0	0	0	0	1	0	1	0	0	0	0
1	0	0	1	0	0	0	0	0	1	0	1	0
0	0	0	0	1	0	0	0	1	0	1	0	1
0	1	0	1	0	0	0	0	0	0	1	1	0
0	0	1	0	1	0	1	0	0	0	0	0	1
0	1	1	0	0	1	0	0	0	0	0	1	0
0	1	0	0	0	0	1	1	0	0	0	0	1
0	0	0	1	0	1	0	1	0	1	0	0	0
0	0	1	0	0	0	1	0	1	0	1	0	0

2. Cho đồ thị vô hướng liên thông G=<V,E> như hình bên phải. Hãy thực hiện:

a) Chứng minh đồ thị đã cho là nửa Euler?

b) Xây dựng thuật toán tìm một đường đi Euler của đồ thị?

c) Tìm một đường đi Euler của đồ thị? Chỉ rõ kết quả trung gian theo mỗi bước thực hiện của thuật toán?

d) Viết chương trình tìm một đường đi Euler của đồ thị?

0	0	0	1	0	1	0	0	0	0	0	0	0
0	0	1	0	0	0	0	1	0	1	1	0	0
0	1	0	0	0	0	0	0	1	1	0	0	1
1	0	0	0	0	1	0	1	0	0	0	1	0
0	0	0	0	0	0	1	0	1	0	0	0	0
1	0	0	1	0	0	0	0	0	1	0	1	0
0	0	0	0	1	0	0	0	0	0	1	0	1
0	1	0	1	0	0	0	0	0	0	1	1	0
0	0	1	0	1	0	0	0	0	0	0	0	1
0	1	1	0	0	1	0	0	0	0	0	1	0
0	1	0	0	0	0	1	1	0	0	0	0	1
0	0	0	1	0	1	0	1	0	1	0	0	0
0	0	1	0	0	0	1	0	1	0	1	0	0

3. Cho đồ thị có hướng liên thông yếu G=<V,E> như hình bên phải. Hãy thực hiện:

a) Chứng minh đồ thị đã cho là Euler?

b) Xây dựng thuật toán tìm một chu trình Euler của đồ thị bắt đầu tại đỉnh u∈V?

c) Tìm một chu trình Euler bắt đầu tại đỉnh u=1? Chỉ rõ kết quả trung gian theo mỗi bước thực hiện của thuật toán?

d) Tìm một chu trình Euler bắt đầu tại đỉnh u=5? Chỉ rõ kết quả trung gian theo mỗi bước thực hiện của thuật toán?

e) Viết chương trình tìm một chu trình Euler của đồ thị bắt đầu tại đỉnh u?

0	0	0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	0	0	0	1	0	0	0	0	0
0	0	0	0	0	0	0	0	1	0	0	0	1
1	0	0	0	0	1	0	0	0	0	0	0	0
0	0	0	0	0	0	1	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1	0	1	0
0	0	0	0	0	0	0	0	0	0	1	0	1
0	0	0	1	0	0	0	0	0	0	0	1	0
0	0	0	0	1	0	1	0	0	0	0	0	0
0	1	1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	1	0	0	0	0	0
0	0	0	1	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	0	0	1	0	1	0	0

4. Cho đồ thị có hướng liên thông yếu G=<V,E> như hình bên phải. Hãy thực hiện:

a) Chứng minh đồ thị đã cho là nửa Euler?

b) Xây dựng thuật toán tìm một đường đi của đồ thị?

c) Tìm một đường đi Euler của đồ thị? Chỉ rõ kết quả trung gian theo mỗi bước thực hiện của thuật toán?

e) Viết chương trình tìm một đường đi duler của đồ thị?

0	0	0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	0	0	0	1	0	0	0	0	0
0	0	0	0	0	0	0	0	1	0	0	0	1
1	0	0	0	0	1	0	0	0	0	0	0	0
0	0	0	0	0	0	1	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1	0	1	0
0	0	0	0	0	0	0	0	0	0	1	0	1
0	0	0	1	0	0	0	0	0	0	0	1	0
0	0	0	0	1	0	0	0	0	0	0	0	0
0	1	1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	1	0	0	0	0	0
0	0	0	1	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	0	0	1	0	1	0	0

5. Cho đồ thị vô hướng liên thông được biểu diễn dưới dạng danh sách kề như dưới đây:

Ke(1) = { 4, 6 }. Ke(5) = { 7, 9 }. Ke(9) = { 3, 5, 7, 13 }.
Ke(2) = { 3, 8, 10, 11}. Ke(6) = { 1, 4, 10, 12 }. Ke(10) = { 2, 3, 6, 12 }.
Ke(3) = { 2, 9, 10, 13 }. Ke(7) = { 5, 9, 11, 13 }. Ke(11) = { 2, 7, 8, 13 }.
Ke(4) = { 1, 6, 8, 12 }. Ke(8) = { 2, 4, 11, 12 }. Ke(12) = { 4, 6, 8, 10 }.
Ke(13) = { 3, 7, 9, 11 }.

Hãy thực hiện:

a) Tìm một chu trình Euler bắt đầu tại đỉnh u=1? Chỉ rõ kết quả trung gian theo mỗi bước thực hiện của thuật toán?

b) Tìm một chu trình Euler bắt đầu tại đỉnh u=5? Chỉ rõ kết quả trung gian theo mỗi bước thực hiện của thuật toán?

c) Viết chương trình tìm một chu trình Euler của đồ thị bắt đầu tại đỉnh u?

6. Cho đồ thị vô hướng liên thông được biểu diễn dưới dạng danh sách kề như dưới đây:

Ke(1) = { 4, 6 }. Ke(5) = { 7, 9 }. Ke(9) = { 3, 5, 7, 13 }.
Ke(2) = { 3, 8, 10, 11}. Ke(6) = { 1, 10, 12 }. Ke(10) = { 2, 3, 6, 12 }.

Ke(3) = { 2, 9, 10, 13 }. Ke(7) = { 5, 9, 11, 13 }. Ke(11) = { 2, 7, 8, 13 }.
Ke(4) = { 1, 8, 12 }. Ke(8) = { 2, 4, 11, 12 }. Ke(12) = { 4, 6, 8, 10 }.
Ke(13) = { 3, 7, 9, 11 }.

Hãy thực hiện:

a) Xây dựng thuật toán tìm một đường đi Euler của đồ thị?

b) Tìm một đường đi Euler của đồ thị? Chỉ rõ kết quả trung gian theo mỗi bước thực hiện của thuật toán?

c) Viết chương trình tìm một đường đi của đồ thị bắt đầu tại đỉnh u?

7. Cho đồ thị có hướng liên thông yếu được biểu diễn dưới dạng danh sách kề như dưới đây:

Ke(1) = { 6 }. Ke(5) = { 7 }. Ke(9) = { 5, 7 }.
Ke(2) = { 3, 8}. Ke(6) = { 10, 12 }. Ke(10) = { 2, 3 }.
Ke(3) = { 9, 13 }. Ke(7) = { 11, 13 }. Ke(11) = { 2, 8 }.
Ke(4) = { 1, 6 }. Ke(8) = { 4, 12 }. Ke(12) = { 4, 10 }.
Ke(13) = { 9, 11 }.

Hãy thực hiện:

a) Tìm một chu trình Euler bắt đầu tại đỉnh u=1? Chỉ rõ kết quả trung gian theo mỗi bước thực hiện của thuật toán?

b) Tìm một chu trình Euler bắt đầu tại đỉnh u=7? Chỉ rõ kết quả trung gian theo mỗi bước thực hiện của thuật toán?

c) Viết chương trình tìm một chu trình Euler của đồ thị bắt đầu tại đỉnh u?

8. Cho đồ thị có hướng liên thông yếu được biểu diễn dưới dạng danh sách kề như dưới đây:

Ke(1) = { 6 }. Ke(5) = { 7 }. Ke(9) = { 5, 7 }.
Ke(2) = { 3, 8}. Ke(6) = { 10, 12 }. Ke(10) = { 2, 3 }.
Ke(3) = { 9, 13 }. Ke(7) = { 11, 13 }. Ke(11) = { 2, 8 }.
Ke(4) = { 1 }. Ke(8) = { 4, 12 }. Ke(12) = { 4, 10 }.
Ke(13) = { 9, 11 }.

Hãy thực hiện:

a) Trình bày thuật toán tìm một đường đi Euler trên đồ thị có hướng?

b) Tìm một đường đi Euler của đồ thị?

c) Viết chương trình tìm một đường đi Euler của đồ thị?

CHƯƠNG 5. CÂY KHUNG CỦA ĐỒ THỊ

Nội dung chính của chương này đề cập đến một loại đồ thị đơn giản nhất đó là cây. Cây được ứng dụng rộng rãi trong nhiều lĩnh vực khác nhau của tin học như tổ chức các thư mục, lưu trữ dữ liệu, biểu diễn tính toán, biểu diễn quyết định và tổ chức truyền tin. Những nội dung được trình bày bao gồm:

- ✓ Cây và các tính chất cơ bản của cây.
- ✓ Cây khung của đồ thị & các thuật toán cơ bản xây dựng cây khung của đồ thị.
- ✓ Bài toán tìm cây khung nhỏ nhất & các thuật toán tìm cây khung nhỏ nhất.
- ✓ Thuật toán Kruskal tìm cây bao trùm nhỏ nhất.
- ✓ Thuật toán Prim tìm cây bao trùm nhỏ nhất.

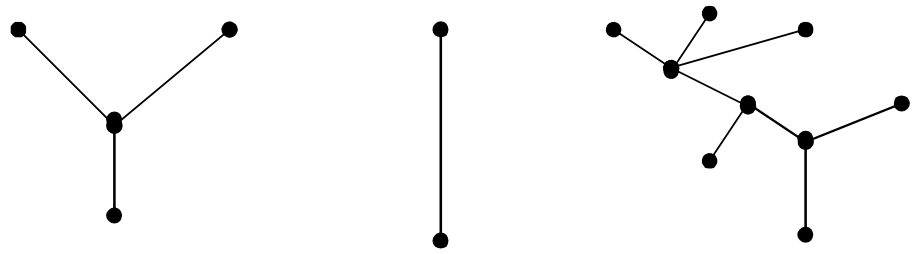
Bạn đọc có thể tìm thấy những chứng minh cụ thể cho các định lý, tính đúng đắn và độ phức tạp các thuật toán thông qua các tài liệu [1], [2].

5.1. Cây và một số tính chất cơ bản

Định nghĩa 1. Ta gọi cây là đồ thị vô hướng liên thông không có chu trình. Đồ thị không liên thông được gọi là rừng.

Như vậy, rừng là đồ thị mà mỗi thành phần liên thông của nó là một cây.

Ví dụ. Rừng gồm 3 cây trong hình 7.1.



Hình 5.1 . *Rừng gồm 3 cây T1, T2, T3.*

Cây được coi là dạng đồ thị đơn giản nhất của đồ thị. Định lý sau đây cho ta một số tính chất của cây.

Định lý. Giả sử T= <V, E> là đồ thị vô hướng n đỉnh. Khi đó những khẳng định sau là tương đương

a) T là một cây;

b) T không có chu trình và có n-1 cạnh;

c) T liên thông và có đúng n-1 cạnh;

d) T liên thông và mỗi cạnh của nó đều là cầu;

e) Giữa hai đỉnh bất kỳ của T được nối với nhau bởi đúng một đường đi đơn;

f) T không chứa chu trình nhưng hễ cứ thêm vào nó một cạnh ta thu được đúng một chu trình;

Chứng minh. Định lý được chứng minh định lý thông qua các bước (a) =>(b) =>(c) => (d) =>(e) => (f) => (a). Những bước cụ thể của quá trình chứng minh bạn đọc có thể tìm thấy trong các tài liệu [1], [2].

Định nghĩa 2. Cho G là đồ thị vô hướng liên thông. Ta gọi đồ thị con T của G là một cây khung của G (Cây bao trùm) nếu T thoả mãn hai điều kiện:

a) T là một cây;

b) Tập đỉnh của T bằng tập đỉnh của G.

Trong lý thuyết đồ thị, người ta qua tâm đến hai bài toán cơ bản về cây:

Bài toán 1. Cho đồ thị vô hướng G =<V,E>. Hãy xây dựng một cây khung của đồ thị bắt đầu tại đỉnh u∈V.

Bài toán 2. Cho đồ thị vô hướng G =<V,E> có trọng số. Hãy xây dựng cây khung có độ dài nhỏ nhất.

Bài toán 1 được giải quyết bằng các thuật toán tìm kiếm cơ bàn: thuật toán DFS hoặc BFS. Bài toán 2 được giải quyết bằng thuật toán Kruskal hoặc PRIM.

5.2. Xây dựng cây khung của đồ thị dựa vào thuật toán DFS

Để tìm một cây khung trên đồ thị vô hướng liên thông ta có thể sử dụng kỹ thuật tìm kiếm theo chiều sâu. Giả sử ta cần xây dựng một cây bao trùm xuất phát tại đỉnh *u* nào đó. Trong cả hai trường hợp, mỗi khi ta đến được đỉnh *v* tức (*chuaxet[v] = False*) từ đỉnh u thì cạnh (*u,v*) được kết nạp vào cây khung. Kỹ thuật xây dựng cây khung bắt đầu tại đỉnh u dựa vào thuật toán DFS được mô tả trong Hình 5.2.

5.2.1. Mô tả thuật toán

```
Thuật toán Tree-DFS(u) {
        chuaxet[u] = False; //Bật trạng thái đỉnh u từ True trở thành False
        for v∈Ke(u) do { //Duyệt trên danh sách kề của đỉnh u
                if (chuaxet[v]) { //Nếu đỉnh v chưa được xét đến
                        T = T∪(u,v); //Hợp cạnh (u,v) vào cây khung
                        DFS(v); //Duyệt theo chiều sâu bắt đầu tại đỉnh v
                }
        }
}
```

Hình 5.2. Thuật toán Tree-DFS(u).

Khi đó, quá trình xây dựng cây khung bắt đầu tại đỉnh u được thực hiện như thuật toán trong Hình 5.3.

```
Thuật toán Tree-Graph-DFS() {
        for each u∈V do //Khởi tạo các đỉnh chưa xét
                chuaxet[u]= True;
        endfor;
        roof = <Đỉnh bất kỳ của đồ thị>; //Lấy một đỉnh bất kỳ làm gốc
        T = ∅;          //Thiết lập tập cạnh ban đầu của cây là ∅
        Tree-DFS(roof); //Thực hiện thuật toán Tree-DFS(roof)
        if (|T| <n-1) <Đồ thị không liên thông>;
        else <Ghi nhận tập cạnh Tcủa cây khung>
}
```

Hình 5.3. Thuật toán xây dựng cây khung dựa vào DFS.

5.2.2. Kiểm nghiệm thuật toán

Giả sử ta cần kiểm nghiệm thuật toán Tree-Graph-DFS với đỉnh bắt đầu u=1 trên đồ thị được biểu diễn dưới dạng ma trận kề dưới đây. Khi đó các bước thực hiện của thuật toán được thể hiện trong Bảng 5.1.

0	1	1	1	0	0	0	0	0	0	0	0	0
1	0	1	1	0	0	0	0	0	0	0	0	0
1	1	0	1	1	0	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	1	1	1	1	0	0	0	0
0	0	0	0	1	0	1	0	1	0	0	0	0
0	0	0	0	1	1	0	1	0	0	0	0	0
0	0	0	0	1	0	1	0	1	0	0	0	0
0	0	0	0	1	1	0	1	0	1	0	0	0
0	0	0	0	0	0	0	0	1	0	1	1	1
0	0	0	0	0	0	0	0	0	1	0	1	1
0	0	0	0	0	0	0	0	0	1	1	0	1
0	0	0	0	0	0	0	0	0	1	1	1	0

Bảng 5.1. Kiểm nghiệm thuật toán Tree-Graph-DFS

Bước	Tree-DFS(u) =?	T =?
1	1	T = ∅
2	1, 2	T = T∪(1,2)
3	1, 2, 3	T = T∪(2,3)

4	1, 2, 3, 4	T = T∪(3, 4)
5	1, 2, 3, 4, 5	T = T∪(3, 5)
6	1, 2, 3, 4, 5, 6	T = T∪(5, 6)
7	1, 2, 3, 4, 5, 6, 7	T = T∪(6, 7)
8	1, 2, 3, 4, 5, 6, 7, 8	T = T∪(7, 8)
9	1, 2, 3, 4, 5, 6, 7, 8, 9	T = T∪(8, 9)
10	1, 2, 3, 4, 5, 6, 7, 8, 9, 10	T = T∪(9, 10)
11	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11	T = T∪(10, 11)
12	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12	T = T∪(11, 12)
13	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13	T = T∪(12, 13)
Kết luận T = {(1,2), (2,3), (3,4), (3,5), (5,6), (6,7), (7,8), (8,9), (9,10), (10,11), (11,12), (12,13)		

5.2.3. Cài đặt thuật toán

Thuật toán Tree-Graph-DFS được cài đặt đối với đồ thị được biểu diễn dưới dạng ma trận kề. Các thủ tục chính được cài đặt bao gồm:

- Thủ tục Init() : đọc dữ liệu và thiết lập giá trị của mảng chuaxet[].
- Thủ tục Tree-DFS (u) : thuật toán DFS bắt đầu tại đỉnh u.
- Thủ tục Result(): ghi nhận tập cạnh của cây khung.

Chương trình xây dựng một cây khung được thể hiện như sau:

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#define MAX 50
#define    TRUE 1
#define FALSE  0
int CBT[MAX][2], n, A[MAX][MAX], chuaxet[MAX], sc, QUEUE[MAX];
void Init(void){
    int i, j;FILE *fp;
    fp= fopen("BAOTRUM1.IN", "r");
    if(fp==NULL){
            printf("\n Khong co file input");
            getch(); return;
    }
    fscanf(fp,"%d",&n);
```

```
    printf("\n So dinh do thi:%d", n);
    printf("\n Ma tran ke:");
    for(i=1; i<=n; i++){
            printf("\n");
            for(j=1; j<=n; j++){
                    fscanf(fp, "%d", &A[i][j]);
                    printf("%3d", A[i][j]);
            }
    }
    fclose(fp);
    for (i=1; i<=n;i++)
            chuaxet[i]=TRUE;
}
void TREE_DFS(int i){
    int j; chuaxet[i] = False;
    if(sc==n-1) return;
    for(j=1; j<=n; j++){
            if (chuaxet[j] && A[i][j]){  sc++;
                    CBT[sc][1]=i; CBT[sc][2]=j;
                    if(sc==n-1) return;
                    STREE_DFS(j);
            }
    }
}
void Result(void){
    int i, j;
    for(i=1; i<=sc; i++){
            printf("\n Canh %d:", i);
            for(j=1; j<=2; j++)
                    printf("%3d", CBT[i][j]);
    }
}
void main(void){
    int i; Init(); sc=0; i=1; /* xây dựng cây bao trùm tại đỉnh 1*/
    TREE_DFS(i);
    if (sc<n-1) printf("\n Đồ thị không liên thông");
    else Result();
}
```

5.3. Xây dựng cây khung của đồ thị dựa vào thuật toán BFS

Để tìm một cây khung trên đồ thị vô hướng liên thông ta có thể sử dụng kỹ thuật tìm kiếm theo chiều rộng. Giả sử ta cần xây dựng một cây bao trùm xuất phát tại đỉnh *u*

nào đó. Trong cả hai trường hợp, mỗi khi ta đến được đỉnh *v* tức (*chuaxet[v] = False*) từ đỉnh u thì cạnh (*u,v*) được kết nạp vào cây khung.

5.3.1. Cài đặt thuật toán

Thuật toán xây dựng cây khung của đồ thị được mô tả như Hình 5.4.

```
Thuật toán Tree-BFS(u):
Begin
        Bước 1 (Khởi tạo):
                T = ∅; //Tập cạnh cây khung ban đầu.
                Queue = ∅; //Thiết lập hàng đợi ban đầu;
                Push(Queue, u); //Đưa u vào hàng đợi;
                chuaxet[u] = False;//Bật trạng thái đã xét của đỉnh u
        Bước 2 (Lặp):
                while (Queue≠∅ ) do { //Lặp cho đến khi hàng đợi rỗng
                        s = Pop(Queue); Lấy s ra khỏi hàng đợi
                        for each t∈Ke(s) do { //Lặp trên danh sách Ke(s)
                                if (chuaxet[t] ) then { //Nếu đỉnh t chuaxet
                                        Push(Queue, t);// Đưa t vào hàng đợi
                                        T = T∪(s,t); //Kết nạp (s,t) vào cây khung
                                        chuaxet[t] = False; //Ghi nhận t đã xét
                                endif ;
                        endfor ;
                endwwhile ;
        Bước 3 (Trả lại kết quả) :
                if (| T | < n-1 ) <Đồ thị không liên thông> ;
                else <Ghi nhận tập cạnh T của cây khung" ;
end.
```

Hình 5.4. Thuật toán Tree-BFS(u).

5.3.2. Kiểm nghiệm thuật toán

Giả sử ta cần kiểm nghiệm thuật toán Tree- BFS với đỉnh bắt đầu u=1 trên đồ thị được biểu diễn dưới dạng ma trận kề dưới đây. Khi đó các bước thực hiện của thuật toán được thể hiện trong Bảng 5.2.

0	1	1	1	0	0	0	0	0	0	0	0	0
1	0	1	1	0	0	0	0	0	0	0	0	0
1	1	0	1	1	0	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	1	1	1	1	0	0	0	0
0	0	0	0	1	0	1	0	1	0	0	0	0
0	0	0	0	1	1	0	1	0	0	0	0	0
0	0	0	0	1	0	1	0	1	0	0	0	0
0	0	0	0	1	1	0	1	0	1	0	0	0
0	0	0	0	0	0	0	0	1	0	1	1	1
0	0	0	0	0	0	0	0	0	1	0	1	1
0	0	0	0	0	0	0	0	0	1	1	0	1
0	0	0	0	0	0	0	0	0	1	1	1	0

Bảng 5.2. Kiểm nghiệm thuật toán Tree-BFS		
Bước	Trạng thái hàng đợi: Tree-BFS(u) =?	T =?
1	1	T = ∅
2	2, 3, 4	T = T∪{(1,2), (1,3), (1,4) }
3	3, 4	T=T∪∅
4	4, 5	T=T∪(3,5)
5	5	T=T∪∅
6	6, 7, 8, 9	T = T∪{(5,6), (5,7), (5,8), (5,9)}
7	7, 8, 9	T = T∪∅
8	8, 9	T = T∪∅
9	9	T = T∪∅
10	10	T = T∪(9,10)
11	11, 12, 13	T = T∪{(10,11), (10,12), (10,13)}
12	12, 13	T = T∪∅
13	13	T = T∪∅
14	∅	T = T∪∅
Kết luận T = {(1,2), (1,3), (1,4), (3,5), (5,6), (5,7), (5,8), (5,9), (9,10), (10,11), (10,12), (10,13)		

5.3.3. Cài đặt thuật toán

Thuật toán Tree-BFS được cài đặt đối với đồ thị được biểu diễn dưới dạng ma trận kề. Các thủ tục chính được cài đặt bao gồm:

- Thủ tục Init() : đọc dữ liệu và thiết lập giá trị của mảng chuaxet[].
- Thủ tục Tree-BFS (u) : thuật toán BFS bắt đầu tại đỉnh u.
- Thủ tục Result(): ghi nhận tập cạnh của cây khung.

Chương trình xây dựng một cây khung được thể hiện như sau:

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#define MAX 50
#define    TRUE 1
#define FALSE   0
int CBT[MAX][2], n, A[MAX][MAX], chuaxet[MAX], sc, QUEUE[MAX];
void Init(void){
    int i, j;FILE *fp;
    fp= fopen("BAOTRUM1.IN", "r");
    if(fp==NULL){
            printf("\n Khong co file input");
            getch(); return;
    }
    fscanf(fp,"%d",&n);
    printf("\n So dinh do thi:%d", n);
    printf("\n Ma tran ke:");
    for(i=1; i<=n; i++){
            printf("\n");
            for(j=1; j<=n; j++){
                    fscanf(fp, "%d", &A[i][j]);
                    printf("%3d", A[i][j]);
            }
    }
    fclose(fp);
    for (i=1; i<=n;i++)
            chuaxet[i]=TRUE;
}
void TREE_BFS(int u){
    int dauQ, cuoiQ, v, p;
    dauQ=1; cuoiQ=1; QUEUE[dauQ]=u;chuaxet[u]=FALSE;
    while(dauQ<=cuoiQ){
            v= QUEUE[dauQ]; dauQ=dauQ+1;
            for(p=1; p<=n; p++){
                    if(chuaxet[p] && A[v][p]){
                            chuaxet[p]=FALSE; sc++;
                            CBT[sc][1]=v; CBT[sc][2]=p;
```

```
                        cuoiQ=cuoiQ+1;
                        QUEUE[cuoiQ]=p;
                        if(sc==n-1) return;
                    }
                }
            }
        }

    void Result(void){
        int i, j;
        for(i=1; i<=sc; i++){
                printf("\n Canh %d:", i);
                for(j=1; j<=2; j++)
                        printf("%3d", CBT[i][j]);
        }
    }
    void main(void){
        int i; Init(); sc=0; i=1; /* xây dựng cây bao trùm tại đỉnh 1*/
        TREE_BFS(i);
        if (sc<n-1) printf("\n Đồ thị không liên thông");
        else Result();
    }
```

5.4. Bài toán xây dựng cây khung có độ dài nhỏ nhất

Bài toán tìm cây khung nhỏ nhất là một trong những bài toán tối ưu trên đồ thị có ứng dụng trong nhiều lĩnh vực khác nhau của thực tế. Bài toán được phát biểu như dưới đây.

5.4.1. Đặt bài toán

Cho $G=<V, E>$ là đồ thị vô hướng liên thông với tập đỉnh $V = \{1, 2, . . ., n\}$ và tập cạnh E gồm m cạnh. Mỗi cạnh e của đồ thị được gán với một số không âm $c(e)$ được gọi là độ dài cạnh. Giả sử $H=<V, T>$ là một cây khung của đồ thị G. Ta gọi độ dài $c(H)$ của cây khung H là tổng độ dài các cạnh: $c(H) = \sum_{e \in T} c(e)$. Bài toán được đặt ra là, trong số các cây khung của đồ thị hãy tìm cây khung có độ dài nhỏ nhất của đồ thị.

Để minh họa cho những ứng dụng của bài toán này, chúng ta có thể tham khảo hai mô hình thực tế của bài toán.

Bài toán nối mạng máy tính. Một mạng máy tính gồm n máy tính được đánh số từ *1, 2, . . ., n*. Biết chi phí nối máy *i* với máy *j* là *c[i, j], i, j = 1, 2, . . ., n*. Hãy tìm cách nối mạng sao cho chi phí là nhỏ nhất.

Bài toán xây dựng hệ thống cable. Giả sử ta muốn xây dựng một hệ thống cable điện thoại nối *n* điểm của một mạng viễn thông sao cho điểm bất kỳ nào trong mạng đều

có đường truyền tin tới các điểm khác. Biết chi phí xây dựng hệ thống cable từ điểm *i* đến điểm *j* là *c[i,j]*. Hãy tìm cách xây dựng hệ thống mạng cable sao cho chi phí là nhỏ nhất.

Để giải bài toán cây khung nhỏ nhất, chúng ta có thể liệt kê toàn bộ cây khung và chọn trong số đó một cây nhỏ nhất. Phương án như vậy thực sự không khả thi vì số cây khung của đồ thị là rất lớn cỡ n^{n-2}, điều này không thể thực hiện được với đồ thị với số đỉnh cỡ chục.

Để tìm một cây khung ta có thể thực bằng hai thuật toán: Thuật toán Kruskal và thuật toán PRIM.

5.4.2. Thuật toán Kruskal

Thuật toán sẽ xây dựng tập cạnh *T* của cây khung nhỏ nhất *H*=<*V*, *T*> theo từng bước được mô tả trong Hình 5.5 như dưới đây.

a) Mô tả thuật toán

```
Thuật toán Kruskal:
Begin
    Bước 1 (Khởi tạo):
        T = ∅; //Khởi tạo tập cạnh cây khung là ∅
        d(H) = 0' //Khởi tạo độ dài nhỏ nhất cây khung là 0
    Bước 2 (Sắp xếp):
        <Sắp xếp các cạnh của đồ thị theo thứ tự giảm dần của trọng số>;
    Bước 3 (Lặp):
        while (|T<n-1| && E≠∅ ) do { // Lặp nếu E≠∅ và |T|<n-1
            e = <Cạnh có độ dài nhỏ nhất>;
            E = E \ {e}; //Loại cạnh e ra khỏi đồ thị
            if (T∪{e} không tạo nên chu trình ) then {
                T = ∪{e}; // Kết nạp e vào tập cạnh cây khung
                d(H) = d(H) + d(e); //Độ dài của tập cạnh cây khung
            endif;
        endwhile;
    Bước 4 (Trả lại kết quả):
        if (|T| < n-1) then <Đồ thị không liên thông>;
        else
            Return( T, d(H));
end.
```

Hình 5.5. Thuật toán Kruskal tìm cây khung nhỏ nhất.

b) Kiểm nghiệm thuật toán

Ví dụ ta cần kiểm nghiệm thuật toán Kruskal trong Hình 5.5 trên đồ thị được biểu diễn dưới dạng ma trận kề như dưới đây. Thực hiện tuần tự các bước của thuật toán ta sẽ được kết quả như sau:

∞	2	1	3	∞	∞	∞	∞	∞	∞	∞	∞	∞
2	∞	2	∞	∞	5	5	∞	∞	∞	∞	∞	∞
1	2	∞	4	∞	5	∞	∞	∞	∞	∞	∞	∞
3	∞	4	∞	5	5	∞	∞	∞	∞	∞	∞	∞
∞	∞	∞	5	∞	6	∞	∞	∞	6	∞	∞	∞
∞	5	5	5	6	∞	6	6	6	6	∞	∞	∞
∞	5	∞	∞	∞	6	∞	6	∞	∞	∞	∞	∞
∞	∞	∞	∞	∞	6	6	∞	7	∞	∞	7	7
∞	∞	∞	∞	∞	6	∞	7	∞	7	7	∞	∞
∞	∞	∞	∞	6	6	∞	∞	7	∞	7	7	∞
∞	∞	∞	∞	∞	∞	∞	∞	7	7	∞	8	∞
∞	∞	∞	∞	∞	∞	∞	7	∞	7	8	∞	8
∞	∞	∞	∞	∞	∞	∞	7	∞	∞	∞	8	∞

Bước 1: T = ϕ; D(T) = 0;

Bước 2. Sắp xếp các cạnh theo thứ tự tăng dần của trọng số

Đầu	Cuối	Tr.Số
1	2	2
1	3	1
1	4	3
2	3	2
2	6	5
2	7	5
3	4	4
3	6	5
4	5	5
4	6	5
5	6	6
5	10	6
6	7	6
6	8	6
6	9	6
6	10	6
7	8	6
8	9	7
8	12	7
8	13	7
9	10	7
9	11	7
10	11	7
10	12	7
11	12	8
12	13	8

Đầu	Cuối	Tr.Số
1	3	1
1	2	2
2	3	2
1	4	3
3	4	4
2	6	5
2	7	5
3	6	5
4	5	5
4	6	5
5	6	6
5	10	6
6	7	6
6	8	6
6	9	6
6	10	6
7	8	6
8	9	7
8	12	7
8	13	7
9	10	7
9	11	7
10	11	7
10	12	7
11	12	8
12	13	8

Bước 3 (lặp) :

STT	Cạnh được xét	T ∪e
1	E \(1,3)	T = T∪(1,3); D(T) = 1
2	E = E\(1,2)	T = T∪(1,2) ; D(T) = 1+2 =3
3	E = E\(2,3)	*Tạo nên chu trình*
4	E = E\(1,4)	T = T∪(1,4); D(T) = 3 +3 =6
5	E = E\(3,4)	*Tạo nên chu trình*
6	E = E\(2,6)	T = T∪(2,6); D(T) = 6+5=11
7	E = E\(2,7)	T = T∪(2,7); D(T) = 11+5 =16
8	E = E\(3,6)	*Tạo nên chu trình*
9	E = E\(4,5)	T = T∪ (4,5); D(T) = 16+5 =21
10	E = E\(4,6)	*Tạo nên chu trình*
11	E = E\(5,6)	*Tạo nên chu trình*
12	E = E\(5,10)	T = T∪(5,10); D(T) = 21+6 =27
13	E = E\(6,7)	*Tạo nên chu trình*
14	E = E\(6,8)	T = T∪(6,8); D(T) = 27+6 =33
15	E = E\(6,9)	T = T∪(6,9); D(T) = 33+6 =39
16	E = E\(6,10)	*Tạo nên chu trình*
17	E = E\(7,8)	*Tạo nên chu trình*
18	E = E\(8,9)	*Tạo nên chu trình*
19	E = E\(8,12)	T = T∪(8,12); D(T) = 39+7 =46
20	E = E\(8,13)	T = T∪(8,13); D(T) = 46+7 =53
21	E = E\(9,10)	*Tạo nên chu trình*
22	E = E\(9,11)	T = T∪(9,11); D(T) = 53+7 =60
Bước lặp kết thúc vì |T|> N-1 =12		

Bước 4 : Trả lại kết quả:

T = { (1,3), (1,2), (1,4), (2,6), 2,7), (4,5), (5,10), (6,8),(6,9), (8,12), (8,13), (9,11) }

D(T) = 1 + 2 + 3 + 5 + 5 + 5 + 6 + 6 + 6 + 7 + 7 +7 = 60

c) Cài đặt thuật toán

Chương trình tìm cây khung nhỏ nhất theo thuật toán Kruskal cho đồ thị biểu diễn dưới dạng danh sách trọng số được thể hiện dưới đây với các thủ tục:

- Thủ tục Init(): đọc dữ liệu biểu diễn bằng danh sách trọng số.
- Thủ tục Heap(): sắp xếp các cạnh theo thứ tự tăng dần của trọng số bằng thuật toán Heap Sort.
- Thủ tục Find(), Union() : tìm và kiểm tra khi kết nào cạnh vào cây khung có tạo nên chu trình hay không.
- Thủ tục Result() : đưa ra tập cạnh và độ dài nhỏ nhất của cây khung.

```
#include <stdio.h>
#include <conio.h>
#include <dos.h>
#define MAX 50
#define    TRUE 1
#define FALSE   0
int n, m, minl, connect;
int          dau[500],cuoi[500], w[500];
int           daut[50], cuoit[50], father[50];
void Init(void){
    int i; FILE *fp;
    fp=fopen("baotrum1.in","r");
    fscanf(fp, "%d%d", &n,&m);
    printf("\n So dinh do thi:%d", n);
    printf("\n So canh do thi:%d", m);
    printf("\n Danh sach ke do thi:");
    for(i=1; i<=m;i++){
            fscanf(fp, "%d%d%d", &dau[i], &cuoi[i], &w[i]);
            printf("\n Canh %d: %5d%5d%5d", i, dau[i], cuoi[i], w[i]);
    }
    fclose(fp);getch();
}
void Heap(int First, int Last){
    int j, k, t1, t2, t3;
    j=First;
    while(j<=(Last/2)){
            if( (2*j)<Last && w[2*j + 1]<w[2*j])
                    k = 2*j +1;
            else
                    k=2*j;
            if(w[k]<w[j]){
                    t1=dau[j]; t2=cuoi[j]; t3=w[j];
                    dau[j]=dau[k]; cuoi[j]=cuoi[k]; w[j]=w[k];
                    dau[k]=t1; cuoi[k]=t2; w[k]=t3;  j=k;
            }
            else j=Last;
    }
}


int Find(int i){
    int tro=i;
```

```
    while(father[tro]>0) tro=father[tro];
    return(tro);
}
void Union(int i, int j){
    int x = father[i]+father[j];
    if(father[i]>father[j]) {father[i]=j;father[j]=x; }
    else {
        father[j]=i;   father[i]=x;
    }
}
void Krusal(void){
    int i, last, u, v, r1, r2, ncanh, ndinh;
    for(i=1; i<=n; i++)         father[i]=-1;
    for(i= m/2;i>0; i++)
        Heap(i,m);
    last=m; ncanh=0; ndinh=0;minl=0;connect=TRUE;
    while(ndinh<n-1 && ncanh<m){
        ncanh=ncanh+1;     u=dau[1]; v=cuoi[1];
        r1= Find(u); r2= Find(v);
        if(r1!=r2) {
            ndinh=ndinh+1; Union(r1,r2);
            daut[ndinh]=u; cuoit[ndinh]=v;
            minl=minl+w[1];
        }
        dau[1]=dau[last];   cuoi[1]=cuoi[last]; w[1]=w[last]; last=last-1;
        Heap(1, last);
    }
    if(ndinh!=n-1) connect=FALSE;
}
void Result(void){
    int i;
    printf("\n Do dai cay khung nho nhat:%d", minl);
    printf("\n Cac canh cua cay khung nho nhat:");
    for(i=1; i<n; i++)
        printf("\n %5d%5d",daut[i], cuoit[i]);
 }
void main(void){
    Init(); Krusal();Result(); getch();
}
```

5.4.2. Thuật toán Prim

Thuật toán Kruskal làm việc kém hiệu quả đối với những đồ thị có số cạnh khoảng m=n(n-1)/2. Trong những tình huống như vậy, thuật toán Prim tỏ ra hiệu quả hơn.

a) Mô tả thuật toán

Thuật toán Prim còn được mang tên là người láng giềng gần nhất. Trong thuật toán này, bắt đầu tại một đỉnh tuỳ ý *s* của đồ thị, nối *s* với đỉnh *y* sao cho trọng số cạnh *c[s, y]* là nhỏ nhất. Tiếp theo, từ đỉnh *s* hoặc *y* tìm cạnh có độ dài nhỏ nhất, điều này dẫn đến đỉnh thứ ba *z* và ta thu được cây bộ phận gồm *3* đỉnh *2* cạnh. Quá trình được tiếp tục cho tới khi ta nhận được cây gồm *n-1* cạnh, đó chính là cây bao trùm nhỏ nhất cần tìm. Thuật toán Prim được mô tả trong Hình 5.6.

Thuật toán PRIM (s):
Begin:
Bước 1 (Khởi tạo):
V_H = {*s*}; //Tập đỉnh cây khung thiết lập ban đầu là s
V = V\{s}; //Tập đỉnh V được bớt đi s
T = ∅; //Tập cạnh cây khung thiết lập ban đầu là ∅
d(H) = 0; //Độ dài cây khung được thiết lập là 0
Bước 2 (Lặp):
while (V≠ ∅) do {
e = <u, v>: cạnh có độ dài nhỏ nhất thỏa mãn u∈V, v∈V_H;
d(H) = d(H) + d(e); // Thiết lập đồ dài cây khung nhỏ nhất
T = T ∪ {e}; //Kết nạp e vào cây khung
V = V \{u}; // Tập đỉnh V bớt đi đỉnh u
V_H = V_H∪{u}; // Tập đỉnh V_H thêm vào đỉnh u
endwhile;
Bước 3 (Trả lại kết quả):
if (|T|<n-1) then <Đồ thị không liên thông>;
else Return(T, d(H));
End.

Hình 5.6. Thuật toán PRIM xây dựng cây khung nhỏ nhất.

b) Kiểm nghiệm thuật toán

Giả sử ta cần kiểm nghiệm thuật toán cho đồ thị trọng số Mục 5.4.1. Khi đó các bước thực hiện theo thuật toán PRIM như trong Bảng dưới đây.

Bước khởi tạo: T =ϕ; D(T)=0; V = 2,3,4,5,6,7,8,9,10,11,12,13; V_H =1

e=(v,t)| v∈V, t∈V_T có độ dài nhỏ nhất	V \v = ?	V_H ∪v=?	T, D(T)
(1,3)	2,4,5,6,7,8,9,10,11,12,13	1,3	T = T∪(1,3) D(T) = 0 +1
(1,2)	4,5,6,7,8,9,10,11,12,13	1,2,3	T = T∪(1,2) D(T) = 1+2=3
(1,4)	5,6,7,8,9,10,11,12,13	1,2,3,4	T = T∪(1,4) D(T) = 3+3=6
(2,6)	5, 7,8,9,10,11,12,13	1,2,3,4,6	T = T∪(2,6) D(T) = 6+5=11
(2,7)	5, 8,9,10,11,12,13	1,2,3,4,6,7	T = T∪(2,7) D(T) = 11+5=16
(4,5)	8,9,10,11,12,13	1,2,3,4,5, 6,7	T = T∪(4,5) D(T) = 16+5=21
(5,10)	8,9,11,12,13	1,2,3,4,5, 6,7,10	T = T∪(5,10) D(T) = 21+6=27
(6,8)	9,11,12,13	1,2,3,4,5, 6,7,8,10	T = T∪(6,8) D(T) = 27+6=33
(6,9)	11,12,13	1,2,3,4,5, 6,7,8,9,10	T = T∪(6,9) D(T) = 33+6=39
(8,12)	11,13	1,2,3,4,5, 6,7,8,9,10,12	T = T∪(8,12) D(T) = 39+7=46
(8,13)	11	1,2,3,4,5, 6,7,8,9,10,12,13	T = T∪(8,13) D(T) = 46+7=53
(9,11)	ϕ	1,2,3,4,5, 6,7,8,9,10,12,13,11	T = T∪(9,11) D(T) = 53+7=60
V = ϕ : kết thúc bước lặp			

Kết quả: T = { (1,3), (1,2), (1,4), (2,6), 2,7), (4,5), (5,10), (6,8),(6,9), (8,12), (8,13), (9,11) }
D(T) = 1 + 2 + 3 + 5 + 5 + 5 + 6 + 6 + 6 + 7 + 7 +7 = 60

c) Cài đặt thuật toán

Chương trình tìm cây khung nhỏ nhất theo thuật toán PRIM cho đồ thị biểu diễn dưới dạng danh sách trọng số được thể hiện dưới đây với các thủ tục:

- Thủ tục Init(): đọc dữ liệu biểu diễn bằng danh sách trọng số.
- Thủ tục Prim: Thuật toán PRIM xây dựng cây khung nhỏ nhất.
- Thủ tục Result() : đưa ra tập cạnh và độ dài nhỏ nhất của cây khung.

Chương trình cài đặt thuật toán Prim tìm cây bao trùm nhỏ nhất được thực hiện như sau:

```
#include <stdio.h>
#include <conio.h>
#define         TRUE 1
#define FALSE        0
#define MAX 10000
int   a[100][100];
int  n,m, i,sc,w;
int   chuaxet[100];
int   cbt[100][3];
FILE *f;
void Init (void){
        int  p,i,j,k;
        for(i=1; i<=n; i++)
                for(j=1; j<=n;j++)
                        a[i][j]=0;
        f=fopen("baotrum.in","r");
        fscanf(f,"%d%d",&n,&m);
        printf("\n So dinh: %3d  ",n);
        printf("\n So canh: %3d", m);
        printf("\n Danh sach canh:");
        for(p=1; p<=m; p++){
                fscanf(f,"%d%d%d",&i,&j,&k);
                printf("\n %3d%3d%3d", i, j, k);
                a[i][j]=k; a[j][i]=k;
        }
        for (i=1; i<=n; i++){
                printf("\n");
                for (j=1; j<=n; j++){
                        if (i!=j && a[i][j]==0)
                                a[i][j]=MAX;
                        printf("%7d",a[i][j]);
                }
        }
        fclose(f);getch();
}
void Result(void){
         for(i=1;i<=sc; i++)
                printf("\n %3d%3d", cbt[i][1], cbt[i][2]);
}
void PRIM(void){
        int i,j,k,top,min,l,t,u;
        int s[100];
```

```
        sc=0;w=0;u=1;
        for(i=1; i<=n; i++)
                chuaxet[i]=TRUE;
        top=1;s[top]=u;
        chuaxet[u]=FALSE;
        while (sc<n-1) {
                min=MAX;
                for (i=1; i<=top; i++){
                        t=s[i];
                        for(j=1; j<=n; j++){
                                if (chuaxet[j] && min>a[t][j]){
                                        min=a[t][j];
                                        k=t;l=j;
                                }
                        }
                }
                sc++;w=w+min;
                cbt[sc][1]=k;cbt[sc][2]=l;
                chuaxet[l]=FALSE;a[k][l]=MAX;
                a[l][k]=MAX;top++;s[top]=l;
                printf("\n");
        }
}
void main(void){
        Init();PRIM();Result();
}
```

5.5. Những nội dung cần ghi nhớ

- ✓ Cây là đồ thị vô hướng liên thông không có chu trình. Do vậy, mọi đồ thị vô hướng liên thông đều có ít nhất một cây khung của nó.
- ✓ Hiểu cách biểu diễn và cài đặt được các loại cây: cây nhị phân tìm kiếm, cây quyết định, cây mã tiền tố và cây mã Huffman.
- ✓ Nắm vững phương pháp xây dựng cây khung của đồ thị bằng hai thuật toán duyệt theo chiều rộng và duyệt theo chiều sâu.
- ✓ Hiểu và cài đặt được các thuật toán Kruskal và Prim tìm cây bao trùm nhỏ nhất.

BÀI TẬP

1. Cho đồ thị vô hướng được biểu diễn dưới dạng ma trận kề như Hình bên phải. Hãy thực hiện:

a) Trình bày thuật toán xây dựng một cây khung của đồ thị bắt đầu tại đỉnh u∈V dựa vào thuật toán BFS(u)?

b) Kiểm nghiệm thuật toán BFS(u) bắt đầu tại đỉnh u=1? Chỉ rõ kết quả trung gian theo mỗi bước thực hiện của thuật toán.

c) Kiểm nghiệm thuật toán BFS(u) bắt đầu tại đỉnh u=7? Chỉ rõ kết quả trung gian theo mỗi bước thực hiện của thuật toán.

0	1	1	1	1	0	0	0	0	0	0	0	0
1	0	1	1	0	0	0	0	0	0	0	0	0
1	1	0	1	0	0	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	1	1	1	1	0	0	0	0
0	0	0	0	1	0	1	0	1	0	0	0	0
0	0	0	0	1	1	0	1	0	1	0	0	0
0	0	0	0	1	0	1	0	1	0	0	0	0
0	0	0	0	1	1	0	1	0	0	0	0	0
0	0	0	0	0	0	1	0	0	0	1	1	1
0	0	0	0	0	0	0	0	0	1	0	1	1
0	0	0	0	0	0	0	0	0	1	1	0	1
0	0	0	0	0	0	0	0	0	1	1	1	0

2. Cho đồ thị vô hướng được biểu diễn dưới dạng ma trận kề như Hình bên phải. Hãy thực hiện:

a) Trình bày thuật toán xây dựng một cây khung của đồ thị bắt đầu tại đỉnh u∈V dựa vào thuật toán DFS(u)?

b) Kiểm nghiệm thuật toán DFS(u) bắt đầu tại đỉnh u=1? Chỉ rõ kết quả trung gian theo mỗi bước thực hiện của thuật toán.

c) Kiểm nghiệm thuật toán DFS(u) bắt đầu tại đỉnh u=7? Chỉ rõ kết quả trung gian theo mỗi bước thực hiện của thuật toán.

0	1	1	1	1	0	0	0	0	0	0	0	0
1	0	1	1	0	0	0	0	0	0	0	0	0
1	1	0	1	0	0	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	1	1	1	1	0	0	0	0
0	0	0	0	1	0	1	0	1	0	0	0	0
0	0	0	0	1	1	0	1	0	1	0	0	0
0	0	0	0	1	0	1	0	1	0	0	0	0
0	0	0	0	1	1	0	1	0	0	0	0	0
0	0	0	0	0	0	1	0	0	0	1	1	1
0	0	0	0	0	0	0	0	0	1	0	1	1
0	0	0	0	0	0	0	0	0	1	1	0	1
0	0	0	0	0	0	0	0	0	1	1	1	0

3. Cho đồ thị vô hướng được biểu diễn dưới dạng danh sách kề như dưới đây

Ke(1) = { 2, 3, 4, 5 }.
Ke(2) = { 1, 3, 4 }.
Ke(3) = { 1, 2, 4 }.
Ke(4) = { 1, 2, 3 }.
Ke(5) = { 1, 6, 7, 8, 9 }.
Ke(6) = { 5, 7, 9 }.
Ke(7) = { 5, 6, 8, 10 }.
Ke(8) = { 5, 7, 9 }.
Ke(9) = { 5, 6, 8 }.
Ke(10) = { 7, 11, 12, 13 }.
Ke(11) = { 10, 12, 13 }.
Ke(12) = { 10, 11, 13 }.
Ke(13) = { 10, 11, 12 }.

Hãy thực hiện:

a) Trình bày thuật toán xây dựng cây khung của đồ thị bắt đầu tại đỉnh u dựa vào thuật toán DFS?

b) Xây dựng cây khung của đồ thị bắt đầu tại đỉnh u=3? Chỉ rõ kết quả theo mỗi bước thực hiện của thuật toán?

c) Viết chương trình xây dựng cây khung của đồ thị bắt đầu tại đỉnh u∈V?

4. Cho đồ thị vô hướng được biểu diễn dưới dạng danh sách kề như dưới đây

Ke(1) = { 2, 3, 4, 5 }.	Ke(5) = { 1, 6, 7, 8, 9 }.	Ke(9) = { 5, 6, 8 }.
Ke(2) = { 1, 3, 4 }.	Ke(6) = { 5, 7, 9 }.	Ke(10) = { 7, 11, 12, 13 }.
Ke(3) = { 1, 2, 4 }.	Ke(7) = { 5, 6, 8, 10 }.	Ke(11) = { 10, 12, 13 }.
Ke(4) = { 1, 2, 3 }.	Ke(8) = { 5, 7, 9 }.	Ke(12) = { 10, 11, 13 }.
		Ke(13) = { 10, 11, 12 }.

Hãy thực hiện:

a) Trình bày thuật toán xây dựng cây khung của đồ thị bắt đầu tại đỉnh u dựa vào thuật toán DFS?

b) Xây dựng cây khung của đồ thị bắt đầu tại đỉnh u=3? Chỉ rõ kết quả theo mỗi bước thực hiện của thuật toán?

c) Viết chương trình xây dựng cây khung của đồ thị bắt đầu tại đỉnh u∈V?

5. Cho đồ thị vô hướng có trọng số G =<V,E> được biểu diễn dưới dạng ma trận trọng số như hình bên phải. Hãy thực hiện:

a) Trình bày thuật toán Prim tìm cây khung nhỏ nhất trên đồ thị vô hướng có trọng số?

b) Áp dụng thuật toán, tìm cây khung nhỏ nhất tại đỉnh số 1 của đồ thị G, chỉ rõ kết quả theo từng bước thực hiện của thuật toán?

c) Viết chương trình tìm cây khung nhỏ nhất của đồ thị bằng thuật toán PRIM?

∞	2	1	3	∞	∞	∞	∞	∞	∞	∞	∞	∞
2	∞	2	∞	∞	5	5	∞	∞	∞	∞	∞	∞
1	2	∞	4	∞	5	∞	∞	∞	∞	∞	∞	∞
3	∞	4	∞	5	5	∞	∞	∞	∞	∞	∞	∞
∞	∞	∞	5	∞	6	∞	∞	∞	6	∞	∞	∞
∞	5	5	5	6	∞	6	6	6	6	∞	∞	∞
∞	5	∞	∞	∞	6	∞	6	∞	∞	∞	∞	∞
∞	∞	∞	∞	∞	6	6	∞	7	∞	∞	7	7
∞	∞	∞	∞	∞	6	∞	7	∞	7	7	∞	∞
∞	∞	∞	∞	6	6	∞	∞	7	∞	7	7	∞
∞	∞	∞	∞	∞	∞	∞	∞	7	7	∞	8	∞
∞	∞	∞	∞	∞	∞	∞	7	∞	7	8	∞	8
∞	∞	∞	∞	∞	∞	∞	7	∞	∞	∞	8	∞

6. Cho đồ thị vô hướng có trọng số G =<V,E> được biểu diễn dưới dạng ma trận trọng số như hình bên phải. Hãy thực hiện:

a) Trình bày thuật toán Kruskal tìm cây khung nhỏ nhất trên đồ thị vô hướng có trọng số?

b) Áp dụng thuật toán, tìm cây khung nhỏ nhất của đồ thị G, chỉ rõ kết quả theo từng bước thực hiện của thuật toán?

c) Viết chương trình tìm cây khung nhỏ nhất của đồ thị bằng thuật toán Kruskal?

∞	2	1	3	∞	∞	∞	∞	∞	∞	∞	∞	∞
2	∞	2	∞	∞	5	5	∞	∞	∞	∞	∞	∞
1	2	∞	4	∞	5	∞	∞	∞	∞	∞	∞	∞
3	∞	4	∞	5	5	∞	∞	∞	∞	∞	∞	∞
∞	∞	∞	5	∞	6	∞	∞	∞	6	∞	∞	∞
∞	5	5	5	6	∞	6	6	6	6	∞	∞	∞
∞	5	∞	∞	∞	6	∞	6	∞	∞	∞	∞	∞
∞	∞	∞	∞	∞	6	6	∞	7	∞	∞	7	7
∞	∞	∞	∞	∞	6	∞	7	∞	7	7	∞	∞
∞	∞	∞	∞	6	6	∞	∞	7	∞	7	7	∞
∞	∞	∞	∞	∞	∞	∞	∞	7	7	∞	8	∞
∞	∞	∞	∞	∞	∞	∞	7	∞	7	8	∞	8
∞	∞	∞	∞	∞	∞	∞	7	∞	∞	∞	8	∞

CHƯƠNG 6. BÀI TOÁN TÌM ĐƯỜNG ĐI NGẮN NHẤT

Trong chương này chúng ta sẽ đề cập đến bài toán tìm đường đi ngắn nhất trên đồ thị. Đây là một trong những bài toán có ý nghĩa về lý thuyết và thực tế. Bạn đọc có thể tìm hiểu thêm về phương pháp chứng minh tính đúng đắn cũng như độ phức tạp của các thuật toán thông qua tài liệu [1, 2].

6.1. Phát biểu bài toán

Xét đồ thị $G=<V, E>$; trong đó $|V| = n$, $|E| = m$. Với mỗi cạnh $(u, v)\in E$, ta đặt tương ứng với nó một số thực A[u][v] được gọi là trọng số của cạnh. Ta sẽ đặt $A[u,v]=\infty$ nếu $(u, v)\notin E$. Nếu dãy $v_0, v_1, \ldots, v_k$ là một đường đi trên G thì $\sum_{i=1}^{p} A[v_{i-1}, v_i]$ được gọi là độ dài của đường đi.

Bài toán tìm đường đi ngắn nhất trên đồ thị dưới dạng tổng quát có thể được phát biểu dưới dạng sau: tìm đường đi ngắn nhất từ một đỉnh xuất phát $s\in V$ (đỉnh nguồn) đến đỉnh cuối $t\in V$ (đỉnh đích). Đường đi như vậy được gọi là đường đi ngắn nhất từ *s* đến *t*, độ dài của đường đi $d(s,t)$ được gọi là khoảng cách ngắn nhất từ *s* đến *t* (trong trường hợp tổng quát $d(s,t)$ có thể âm). Nếu như không tồn tại đường đi từ *s* đến *t* thì độ dài đường đi $d(s,t)=\infty$. Dưới đây là một số thể hiện cụ thể của bài toán.

Trường hợp 1. Nếu *s* cố định và *t* thay đổi, khi đó bài toán được phát biểu dưới dạng tìm đường đi ngắn nhất từ *s* đến tất cả các đỉnh còn lại trên đồ thị. Đối với đồ thị có trọng số không âm, bài toán luôn có lời giải bằng thuật toán Dijkstra. Đối với đồ thị có trọng số âm nhưng không tồn tại chu trình âm, bài toán có lời giải bằng thuật toán Bellman-Ford. Trong trường hợp đồ thị có chu trình âm, bài toán không có lời giải.

Trường hợp 2. Nếu s thay đổi và t cũng thay đổi, khi đó bài toán được phát biểu dưới dạng tìm đường đi ngắn nhất giữa tất cả các cặp đỉnh của đồ thị. Bài toán luôn có lời giải trên đồ thị không có chu trình âm. Đối với đồ thị có trọng số không âm, bài toán được giải quyết bằng cách thực hiện lặp lại n lần thuật toán Dijkstra. Đối với đồ thị không có chu trình âm, bài toán có thể giải quyết bằng thuật toán Floyed.

Các thuật toán cụ thể giải quyết bài toán tìm đường đi ngắn nhất được thực hiện như dưới đây.

6.2. Thuật toán Dijkstra

Thuật toán tìm đường đi ngắn nhất từ đỉnh *s* đến các đỉnh còn lại được Dijkstra đề nghị áp dụng cho trường hợp đồ thị có hướng với trọng số không âm. Thuật toán được thực hiện trên cơ sở gán tạm thời cho các đỉnh. Nhãn của mỗi đỉnh cho biết cận trên của độ dài đường đi ngắn nhất tới đỉnh đó. Các nhãn này sẽ được biến đổi (tính lại) nhờ một

thủ tục lặp, mà ở mỗi bước lặp một số đỉnh sẽ có nhãn không thay đổi, nhãn đó chính là độ dài đường đi ngắn nhất từ s đến đỉnh đó.

6.2.1. Mô tả thuật toán

Thuật toán Dijkstra tìm đường đi ngắn nhất từ s đến tất cả các đỉnh còn lại của đồ thị được mô tả chi tiết trong Hình 6.1.

```
Thuật toán Dijkstra (s): //s∈ V là một đỉnh bất kỳ của G = <V,E>
Begin
    Bước 1 (Khởi tạo):
        d[s]=0; //Gán nhãn của đỉnh s là 0
        T = V\{s}; // T là tập đỉnh có nhãn tạm thời
        for each  v∈ V do { //Sử dụng s gán nhãn cho các đỉnh còn lại
            d[v] = A[s,v];
            truoc[v]=s;
        endfor;
    Bước 2 (Lặp):
        while (T≠ ∅ ) do {
            Tìm đỉnh u∈T sao cho d[u] = min { d[z] | z∈T};
            T= T\{u}; //cố định nhãn đỉnh u
            for each v∈ T do { //Sử dụng u, gán nhãn lại cho các đỉnh
                if ( d[v] > d[u] + A[u, v] ) then {
                    d[v] =  d[u] + A[u, v]; //Gán lại nhãn cho đỉnh v;
                    truoc[v] = u;
                endif;
            endfor;
        endwhlie;
    Bước 3 (Trả lại kết quả):
        Return (d[s], truoc[s]);
End.
```

Hình 6.1. Thuật toán Dijkstra.

6.2.2. Kiểm nghiệm thuật toán

Đầu vào của thuật toán :

- Ma trận trọng số không âm $A[u,v] = \begin{cases} d(u,v) & if \quad (u,v) \in E \\ \infty & if \quad (u,v) \notin E \end{cases}$

- s là đỉnh bất kỳ của đồ thị.

Ví dụ ta cần kiểm nghiệm thuật toán cho đồ thị được biểu diễn dưới dạng ma trận trọng số dưới đây. Khi đó, các bước thực hiện theo thuật toán Dijkstra tại đỉnh s =1 được thể hiện như Bảng 6.1.

∞	2	8	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
∞	∞	2	∞	∞	∞	9	∞	∞	∞	∞	∞	∞
∞	∞	∞	6	∞	8	1	∞	∞	∞	∞	∞	∞
7	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
∞	∞	1	7	∞	∞	∞	∞	∞	∞	∞	∞	∞
∞	∞	∞	∞	1	∞	∞	9	8	∞	∞	∞	∞
∞	∞	∞	∞	∞	2	∞	2	∞	∞	∞	∞	∞
∞	∞	∞	∞	∞	∞	∞	∞	9	∞	∞	2	∞
∞	∞	∞	∞	∞	∞	∞	∞	∞	6	∞	9	8
∞	∞	∞	∞	7	6	∞	∞	∞	∞	∞	∞	∞
∞	∞	∞	∞	∞	∞	∞	∞	6	7	∞	∞	∞
∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	2
∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	7	∞	∞

Bảng 6.1. Các bước thực hiện thuật toán Dijkstra tại s =1

Bước	Đỉnh 1	Đỉnh 2	Đỉnh 3	Đỉnh 4	Đỉnh 5	Đỉnh 6	Đỉnh 7	Đỉnh 8	Đỉnh 9	Đỉnh 10	Đỉnh 11	Đỉnh 12	Đỉnh 13
1	<0,1>	<2,1>	<8,1>	<∞,1>	<∞,1>	<∞,1>	<∞,1>	<∞,1>	<∞,1>	<∞,1>	<∞,1>	<∞,1>	<∞,1>
2	*	<2,1>	<4,2>	<∞,1>	<∞,1>	<∞,1>	<11,2>	<∞,1>	<∞,1>	<∞,1>	<∞,1>	<∞,1>	<∞,1>
3	*	*	<4,2>	<10,3>	<∞,1>	<12,3>	<5,3>	<∞,1>	<∞,1>	<∞,1>	<∞,1>	<∞,1>	<∞,1>
4	*	*	*	<10,3>	<∞,1>	<7, 7>	<5,3>	<7, 7>	<∞,1>	<∞,1>	<∞,1>	<∞,1>	<∞,1>
5	*	*	*	<10,3>	<8,6>	<7, 7>	*	<7,7>	<15,6>	<∞,1>	<∞,1>	<∞,1>	<∞,1>
6	*	*	*	<10,3>	<8,6>	*	*	<7,7>	<15,6>	<∞,1>	<∞,1>	<9,8>	<∞,1>
7	*	*	*	<10,3>	<8,6>	*	*	*	<15,6>	<∞,1>	<∞,1>	<9,8>	<∞,1>
8	*	*	*	<10,3>	*	*	*	*	<15,6>	<∞,1>	<∞,1>	<9,8>	<11,12>
9	*	*	*	<10,3>	*	*	*	*	<15,6>	<∞,1>	<∞,1>	*	<11,12>
10	*	*	*	*	*	*	*	*	<15,6>	<∞,1>	<18,13>	*	<11,12>
11	*	*	*	*	*	*	*	*	<15,6>	<21,9>	<18,13>	*	*
12	*	*	*	*	*	*	*	*	*	<21,9>	<18,13>	*	*
13	*	*	*	*	*	*	*	*	*	<21,9>	*	*	*

Kết quả :

Đường đi ngắn nhất từ đỉnh 1 đến đỉnh 2: 2. Đường đi: 1-2.

Đường đi ngắn nhất từ đỉnh 1 đến đỉnh 3: 4. Đường đi: 1-2-3.

Đường đi ngắn nhất từ đỉnh 1 đến đỉnh 4: 10. Đường đi: 1-2-3-10.

Đường đi ngắn nhất từ đỉnh 1 đến đỉnh 5: 8. Đường đi: 1-2-3-7-6-5.

Đường đi ngắn nhất từ đỉnh 1 đến đỉnh 6: 7. Đường đi: 1-2-3-7-6.

Đường đi ngắn nhất từ đỉnh 1 đến đỉnh 7: 5. Đường đi: 1-2-3-7.

Đường đi ngắn nhất từ đỉnh 1 đến đỉnh 8: 7. Đường đi: 1-2-3-7-8.

Đường đi ngắn nhất từ đỉnh 1 đến đỉnh 9: 15. Đường đi: 1-2-3-7-6-9.

Đường đi ngắn nhất từ đỉnh 1 đến đỉnh 10: 21. Đường đi: 1-2-3-7-6-9-10.

Đường đi ngắn nhất từ đỉnh 1 đến đỉnh 11: 18. Đường đi: 1-2-3-7-8-12-13-11.

Đường đi ngắn nhất từ đỉnh 1 đến đỉnh 12: 18. Đường đi: 1-2-3-7-8-12.

Đường đi ngắn nhất từ đỉnh 1 đến đỉnh 13: 11. Đường đi: 1-2-3-7-8-12-13.

6.2.3. Cài đặt thuật toán

Chương trình cài đặt thuật toán Dijkstra tìm đường đi ngắn nhất từ một đỉnh đến tất cả các đỉnh khác của đồ thị có hướng với trọng số không âm được thực hiện như sau:

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <math.h>
#include <dos.h>
#define MAX 50
#define     TRUE 1
#define FALSE     0
int n, s, t;
char chon;
int truoc[MAX], d[MAX], CP[MAX][MAX];
int     final[MAX];
void Init(void){
        FILE * fp;int i, j;
        fp = fopen("ijk1.in","r");
        fscanf(fp,"%d", &n);
        printf("\n So dinh :%d",n);
        printf("\n Ma tran khoang cach:");
        for(i=1; i<=n;i++){
                printf("\n");
                for(j=1; j<=n;j++){
                        fscanf(fp, "%d", &CP[i][j]);
                        printf("%3d",CP[i][j]);
                        if(CP[i][j]==0) CP[i][j]=32000;
                }
        }
        fclose(fp);
}
```

```
void Result(void){
        int i,j;
        printf("\n Duong di ngan nhat tu %d den %d la\n", s,t);
        printf("%d<=",t);
        i=truoc[t];
        while(i!=s){
                printf("%d<=",i);
                i=truoc[i];
        }
        printf("%d",s);
        printf("\n Do dai duong di la:%d", d[t]);
        getch();
}
void Dijkstra(void){
        int v, u, minp;
        printf("\n Tim duong di tu s=");scanf("%d", &s);
        printf("    den ");scanf("%d", &t);
        for(v=1; v<=n; v++){
                d[v]=CP[s][v];
                truoc[v]=s;
                final[v]=FALSE;
        }
        truoc[s]=0;   d[s]=0;final[s]=TRUE;
        while(!final[t]) {
                minp=2000;
                for(v=1; v<=n; v++){
                        if((!final[v]) && (minp>d[v]) ){
                                u=v;
                                minp=d[v];
                        }
                }
                final[u]=TRUE;// u- la dinh co nhan tam thoi nho nhat
                if(!final[t]){
                         for(v=1; v<=n; v++){
                                if ((!final[v]) && (d[u]+ CP[u][v]< d[v])){
                                        d[v]=d[u]+CP[u][v];
                                        truoc[v]=u;
                                }
                        }
                }

        }
}
```

```
void main(void){
        clrscr();Init();        Dijkstra();
        Result(); getch();
}
```

6.3.Thuật toán Bellman-Ford

Thuật toán Bellman-Ford dùng để tìm đường đi ngắn nhất trên đồ thị không có chu trình âm. Do vậy, trước khi thực hiện thuật toán Bellman-Ford ta cần kiểm tra đồ thị có chu trình âm hay không. Trong trường hợp đồ thị có chu trình âm, bài toán sẽ không có lời giải.

6.3.1. Mô tả thuật toán

Thuật toán được thực hiện theo k = n - 2 vòng lặp (n là số đỉnh của đồ thị) chi tiết trong Hình 6.2.

Thuật toán Bellman-Ford (s): //*s∈ V là đỉnh bất kỳ của đồ thị*
Begin:
Bước 1 (Khởi tạo):
for v∈V do { //*Sử dụng s gán nhãn cho các đỉnh v∈V*
D[v] = A[s][v];
Truoc[v] = s;
}
Bước 2 (Lặp) :
D[s] = 0; K=1;
while (K<=N-2) { //N-2 vòng lặp
for v∈V\{s} do { //*Lấy mỗi đỉnh v∈V\s*
for u∈V do { //*Gán nhãn cho v*
if (D[v] > D[u] + A[u][v]) {
D[v]= D[u] + A[u][v];
Truoc[v] = u;
endif;
endfor;
endfor;
endwlie;
Bước 3 (Trả lại kết quả):
Return(D[v], Truoc[v]: v∈U);
End.

Hình 6.2. Thuật toán Bellman-Ford.

6.3.2. Kiểm nghiệm thuật toán

Ví dụ ta cần kiểm nghiệm thuật toán Bellman-Ford cho đồ thị được biểu diễn dưới dạng ma trận trọng số sau:

$$A = \begin{vmatrix} \infty & 1 & \infty & \infty & 3 \\ \infty & \infty & 3 & 3 & 8 \\ \infty & \infty & \infty & 1 & -5 \\ \infty & \infty & 2 & \infty & \infty \\ \infty & \infty & \infty & 4 & \infty \end{vmatrix}$$

Khi đó, kết quả thực hiện theo thuật toán ta được kết quả sau:

Vòng lặp K=1:

v=2; D[2] = 1

D[1] + A[1, 2] = 0+1 (Không nhỏ hơn 1)

D[2] + A[2, 2] = 1 + ∞>1

D[3] + A[3, 2] = ∞ + ∞>1

D[4] + A[4, 2] = ∞ + ∞>1

D[5] + A[5, 2] = ∞ + ∞>1

v=3; D[3] = ∞

D[1] + A[1,3] = 0+∞

D[2] + A[2, 3] = 1 + 3 = 4<∞ (Thay D[3] = 4, Truoc[3] = 2)

D[3] + A[3, 3] = 4 + ∞>4

D[4] + A[4, 3] = ∞ + 2>4

D[5] + A[5, 3] = ∞ + ∞>4

v=4; D[4] = ∞

D[1] + A[1,4] = 0+∞

D[2] + A[2, 4] = 1 + 3 = 4<∞ (Thay D[4] = 4, Truoc[4] = 2)

D[3] + A[3, 4] = 4 + 1=5>4

D[4] + A[4, 4] = 4 + ∞ >4

D[5] + A[5, 4] = ∞ + 4>4

v=5; D[5] = 3

D[1] + A[1,5] = 0+3 (Không nhỏ hơn 3)

D[2] + A[2, 5] = 1 + 8 = 9>3

D[3] + A[3, 5] = 4 -5=-1<3 (Thay D[5] = -1, Truoc[5] =3)

D[4] + A[4, 5] = 4 + ∞ >-1

D[5] + A[5, 5] = -1 + ∞>-1

Vòng lặp K=2:

v=2; D[2] = 1

D[1] + A[1, 2] = 0+1 (Không nhỏ hơn 1)

D[2] + A[2, 2] = 1 + ∞>1

D[3] + A[3, 2] = 4 + ∞>1

D[4] + A[4, 2] = 4 + ∞>1

D[5] + A[5, 2] = -1 + ∞>1

v=3; D[3] = 4

D[1] + A[1, 3] = 0+∞>4

D[2] + A[2, 3] = 1 + 3 =4 (Không nhỏ hơn 4)

D[3] + A[3, 3] = 4 + ∞>4

D[4] + A[4, 3] = 4 + 2>4

D[5] + A[5, 3] = -1 + ∞>4

v=4; D[4] = 4

D[1] + A[1, 4] = 0+∞>4

D[2] + A[2, 4] = 1 + 3 =4 (Không nhỏ hơn 4)

D[3] + A[3, 4] = 4 + 1>4

D[4] + A[4, 4] = 4 + ∞>4

D[5] + A[5, 4] = -1 + 4=3< 4 (Thay D[4] = 5, Truoc[4] = 5

v=5; D[5] = -1

D[1] + A[1, 5] = 0+∞>-1

D[2] + A[2, 5] = 1 + 3 =-1

D[3] + A[3, 5] = 4 + 1>-1

D[4] + A[4, 5] = 3 + ∞>-1

D[5] + A[5, 5] = -1 + ∞>-1

Vòng lặp K=3:

v=2; D[2] = 1

D[1] + A[1, 2] = 0+1 (Không nhỏ hơn 1)

D[2] + A[2, 2] = 1 + ∞>1

D[3] + A[3, 2] = 4 + ∞>1

D[4] + A[4, 2] = 3 + ∞>1

D[5] + A[5, 2] = -1 + ∞>1

v=3; D[3] = 4

D[1] + A[1, 3] = 0+∞>4

D[2] + A[2, 3] = 1 + 3 =4 (Không nhỏ hơn 4)

D[3] + A[3, 3] = 4 + ∞>4

D[4] + A[4, 3] = 3 + 2>4

D[5] + A[5, 3] = -1 + ∞>4

v=4; D[4] = 3

D[1] + A[1, 4] = 0+∞>3

D[2] + A[2, 4] = 1 + 3 =3

D[3] + A[3, 4] = 4 + 1>3
D[4] + A[4, 4] = 3 + ∞>3
D[5] + A[5, 4] = -1 + 4=3(Không nhỏ hơn 3)

v=5; D[5] = -1
D[1] + A[1, 5] = 0+∞>-1
D[2] + A[2, 5] = 1 + 3 =-1
D[3] + A[3, 5] = 4 + 1>-1
D[4] + A[4, 5] = 3 + ∞>-1
D[5] + A[5, 5] = -1 + ∞>-1

Kết quả cuối cùng ta nhận được Bảng 6.2 dưới đây.

Bảng 6.2. Kết quả kiểm nghiệm theo thuật toán Bellman-Ford

K=?	D[1], Truoc[1]	D[2], Truoc[2]	D[3], Truoc[3]	D[4], Truoc[4]	D[5], Truoc[5]
	<0,1>	<1,1>	<∞,1>	<∞,1>	<3,1>
1	<0,1>	<1,1>	<4,2>	<4,2>	<-1,3>
2	<0,1>	<1,1>	<4,2>	<3,5>	<-1,3>
3	<0,1>	<1,1>	<4,2>	<3,5>	<-1,3>

6.3.3. Cài đặt thuật toán

Chương trình cài đặt thuật toán Bellman-Ford tìm đường đi ngắn nhất từ một đỉnh đến tất cả các đỉnh khác của đồ thị có hướng, không có chu trình âm được thực hiện như sau:

```
#include <iostream.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#define MAX  100
#define  MAXC 10000

int C[MAX][MAX]; //Ma tran trong so bieu dien do thi
int D[MAX];      //Do dai duong di
int Trace[MAX];   //Luu lai vet duong di
int n, m, S, F;     // n:So dinh; S: Dinh bat dau; F:
Dinh ket thuc
FILE *fp;
void Read_Data(void){
     int i, u, v;fp = fopen("dothi.in","r");
     fscanf(fp,"%d%d%d%d",&n,&m,&S,&F);
     for(u=1; u<=n; u++)
        for(v=1; v<=n; v++)
```

```
            if (u==v) C[u][v]=0;
            else C[u][v]=MAXC;
     for(i=1; i<=m; i++)
         fscanf(fp,"%d%d%d",&u,&v,&C[u][v]);
     fclose(fp);
}
void Init(void){
    int i;
    for( i=1; i<=n; i++){
        D[i] = C[S][i];
        Trace[i]=S;
    }
}
void Result(void){
    if (D[F]==MAXC) printf("\n Khong co duong di");
    else {
         printf("\n Do dai %d den %d: %d", S, F, D[F]);
         while (F!=S ){
               printf("%d <--",F);
               F = Trace[F];
         }
    }
}
void Ford_Bellman(void){
    int k, u, v;D[S]=0;
    for( k=1; k<=n-2; k++){
         for(v=1; v<=n; v++){
            // if (v!=S ){
                for( u=1; u<=n; u++){
                     if (D[v]>D[u]+C[u][v]){
                        D[v] = D[u]+C[u][v];
                        Trace[u]=v;
                     }
                }
            // }
         }
    }
}
int main()
{
      Read_Data();Init();
      Ford_Bellman(); Result();
      system("PAUSE");
      return 0;
```

}

6.4.Thuật toán Floy

Để tìm đường đi ngắn nhất giữa tất cả các cặp đỉnh của đồ thị, chúng ta có thể sử dụng *n* lần thuật toán *Ford_Bellman* hoặc *Dijkstra* (trong trường hợp trọng số không âm). Tuy nhiên, trong cả hai thuật toán được sử dụng đều có độ phức tạp tính toán lớn (chí ít là $O(n^3)$). Trong trường hợp tổng quát, người ta thường dùng thuật toán *Floy*.

6.4.1. Mô tả thuật toán

Thuật toán Floy được mô tả chi tiết trong Hình 6.3.

```
Thuật toán Floy:
Begin:
    Bước 1 (Khởi tạo):
        for (i=1; i≤ n; i++) {
            for (j =1; j≤ n; j++) {
                d[i,j] = a[i, j];
                p[i,j] = i;
            }
        }
    Bước 2 (lặp) :
        for (k=1; k≤ n; k++) {
            for (i=1; i≤ n; i++){
                for (j =1; j≤ n; j++) {
                    if (d[i,j] > d[i, k] + d[k, j]) {
                        d[i, j] = d[i, k] + d[k, j];
                        p[i,j] = p[k, j];
                    }
                }
            }
        }
    }
    Bước 3 (Trả lại kết quả):
        Return (p([i,j], d[i,j]: i, j∈V);
```

Hình 6.3. Thuật toán Floy.

6.4.2. Cài đặt thuật toán

Chương trình cài đặt thuật toán Foly tìm đường đi ngắn nhất giữa tất cả các cặp đỉnh của đồ thị được thể hiện như sau:

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <math.h>
#include <dos.h>
#define MAX 10000
#define     TRUE 1
#define FALSE     0
int A[50][50], D[50][50], S[50][50];
int n, u, v, k;FILE *fp;
void Init(void){
    int i, j, k;
    fp=fopen("FLOY.IN","r");
    if(fp==NULL){
        printf("\n Khong co file input");
        getch(); return;
    }
    for(i=1; i<=n; i++)
        for(j=1; j<=n; j++)
            A[i][j]=0;
    fscanf(fp,"%d%d%d",&n,&u,&v);
    printf("\n So dinh do thi:%d",n);
    printf("\n Di tu dinh:%d den dinh %d:",u,v);
    printf("\n Ma tran trong so:");
    for(i=1; i<=n; i++){
        printf("\n");
        for(j=1; j<=n; j++){
            fscanf(fp,"%d", &A[i][j]);
            printf("%5d",A[i][j]);
            if(i!=j && A[i][j]==0)
                A[i][j]=MAX;
        }
    }
    fclose(fp);getch();
}
void Result(void){
    if(D[u][v]>=MAX) {
        printf("\n Khong co duong di");
        getch(); return;
```

```
        }
        else {
                printf("\n Duong di ngan nhat:%d", D[u][v]);
                printf("\n Dinh %3d", u);
                while(u!=v) {
                        printf("%3d",S[u][v]);
                        u=S[u][v];
                }
        }
}
void Floy(void){
        int i, j,k, found;
        for(i=1; i<=n; i++){
                for(j=1; j<=n;j++){
                        D[i][j]=A[i][j];
                        if (D[i][j]==MAX) S[i][j]=0;
                        else S[i][j]=j;
                }
        }
        /* Mang D[i,j] la mang chua cac gia tri khoan cach ngan nhat tu i den j
        Mang S la mang chua gia tri phan tu ngay sau cua i tren duong di
        ngan nhat tu i->j  */
        for (k=1; k<=n; k++){
                for (i=1; i<=n; i++){
                        for (j=1; j<=n; j++){
                                if (D[i][k]!=MAX && D[i][j]>(D[i][k]+D[k][j]) ){
                                // Tim D[i,j] nho nhat co the co
                                        D[i][j]=D[i][k]+D[k][j];
                                        S[i][j]=S[i][k];
                                        //ung voi no la gia tri cua phan tu ngay sau i
                                }
                        }
                }
        }
}
void main(void){
        clrscr();Init();
        Floy();Result();
}
```

6.5. Những nội dung cần ghi nhớ

- ✓ Hiểu bài toán tìm đường đi ngắn nhất và các dạng cụ thể của bài toán.
- ✓ Hiểu thuật toán, kiểm nghiệm thuật toán và cài đặt được thuật toán Dijkstra.
- ✓ Hiểu thuật toán, kiểm nghiệm thuật toán và cài đặt được thuật toán Bellman-Ford.
- ✓ Hiểu thuật toán, kiểm nghiệm thuật toán và cài đặt được thuật toán Floy.

BÀI TẬP

1. Cho đồ thị gồm 7 đỉnh cho bởi ma trận trọng số

00	11	65	17	65	65	65
65	00	12	65	65	10	16
65	65	00	13	14	65	19
65	65	65	00	65	65	18
65	65	65	65	00	65	15
65	13	18	65	65	00	10
65	65	65	65	65	65	00

Tìm đường đi ngắn nhất từ đỉnh 1 đến đỉnh 7. Yêu cầu chỉ rõ những kết quả trung gian trong quá trình thực hiện thuật toán.

2. Cho Cơ sở dữ liệu ghi lại thông tin về N ***Tuyến bay*** (N<=100) của một hãng hàng không. Trong đó, thông tin về mỗi tuyến bay được mô tả bởi: Điểm khởi hành (departure), điểm đến (destination), khoảng cách (lenght). Departure, destination là một xâu kí tự độ dài không quá 32, không chứa dấu trống ở giữa, Length là một số nhỏ hơn 32767.

Ta gọi "***Hành trình bay***" từ điểm khởi hành A tới điểm đến B là dãy các hành trình [A, A_1, n_1], [A_1, A_2, n_2] . . .[A_k, B,n_k] với A_i là điểm đến của tuyến i nhưng lại là điểm khởi hành của tuyến i +1, n_i là khoảng cách của tuyến bay thứ i (1<=i<k). Trong đó, khoảng cách của hành trình là tổng khoảng cách của các tuyến mà hành trình đi qua (n_1+n_2+. .+n_k).

Cho file dữ liệu kiểu text hanhtrinh.in được ghi theo từng dòng, số các dòng trong file dữ liệu không vượt quá N, trên mỗi dòng ghi lại thông tin về một tuyến bay, trong đó departure, destination, length được phân biệt với nhau bởi một hoặc vài dấu trống. Hãy tìm giải pháp để thoả mãn nhu cầu của khách hàng đi từ A đến B theo một số tình huống sau:

Tìm hành trình có khoảng cách bé nhất từ A đến B. In ra màn hình từng điểm mà hành trình đã qua và khoảng cách của hành trình. Nếu hành trình không tồn tại hãy đưa ra thông báo "Hành trình không tồn tại".

Ví dụ về Cơ sở dữ liệu hanhtrinh.in

New_York	Chicago	1000	
Chicago	Denver		1000
New_York	Toronto	800	
New_York	Denver		1900

Toronto	Calgary	1500
Toronto	Los_Angeles	1800
Toronto	Chicago	500
Denver	Urbana	1000
Denver	Houston	1500
Houston	Los_Angeles	1500
Denver	Los_Angeles	1000

Với điểm đi : New_York, điểm đến : Los_Angeles ; chúng ta sẽ có kết quả sau:

Hành trình ngắn nhất:

New_York to Toronto to Los_Angeles; Khoảng cách: 2600.

3. Kế tục thành công với khối lập phương thần bí, Rubik sáng tạo ra dạng phẳng của trò chơi này gọi là trò chơi các ô vuông thần bí. Đó là một bảng gồm 8 ô vuông bằng nhau như hình 1. Chúng ta qui định trên mỗi ô vuông có một màu khác nhau. Các màu được kí hiệu bởi 8 số nguyên tương ứng với tám màu cơ bản của màn hình EGA, VGA như hình 1. Trạng thái của bảng các màu được cho bởi dãy kí hiệu màu các ô được viết lần lượt theo chiều kim đồng hồ bắt đầu từ ô góc trên bên trái và kết thúc ở ô góc dưới bên trái. Ví dụ: trạng thái trong hình 1 được cho bởi dãy các màu tương ứng với dãy số (1, 2, 3, 4, 5 , 6, 7, 8). Trạng thái này được gọi là trạng thái khởi đầu.

Biết rằng chỉ cần sử dụng 3 phép biến đổi cơ bản có tên là ‘A’, ‘B’, ‘C’ dưới đây bao giờ cũng chuyển được từ trạng thái khởi đầu về trạng thái bất kỳ:

‘A’ : đổi chỗ dòng trên xuống dòng dưới. Ví dụ sau phép biến đổi A, hình 1 sẽ trở thành hình 2:

‘B’ : thực hiện một phép hoán vị vòng quanh từ trái sang phải trên từng dòng. Ví dụ sau phép biển đổi B hình 1 sẽ trở thành hình 3:

‘C’ : quay theo chiều kim đồng hồ bốn ô ở giữa. Ví dụ sau phép biến đổi C hình 1 trở thành hình 4:

Hình 1

1	2	3	4
8	7	6	5

Hình 2

8	7	6	5
1	2	3	4

Hình 3

4	1	2	3
5	8	7	6

Hình 4

1	7	2	4
8	6	3	5

Cho file dữ liệu Input.txt ghi lại 8 số nguyên trên một dòng, mỗi số được phân biệt với nhau bởi một dấu trống ghi lại trạng thái đích. Hãy tìm dãy các phép biến đổi sơ bản để đưa trạng thái khởi đầu về trạng thái đích sao cho số các phép biến đổi là ít nhất có thể được.

Dữ liệu ra được ghi lại trong file Output.txt, dòng đầu tiên ghi lại số các phép biến đổi, những dòng tiếp theo ghi lại tên của các thao tác cơ bản đã thực hiện, mỗi thao tác cơ bản được viết trên một dòng.

Bạn sẽ được thêm 20 điểm nếu sử dụng bảng màu thích hợp của màn hình để mô tả lại các phép biến đổi trạng thái của trò chơi. Ví dụ với trạng thái đích dưới đây sẽ cho ta kết quả như sau:

Input.txt	Output.txt
2 6 8 4 5 7 3 1	7 B C A B C C B

4. Cho một mạng thông tin gồm N nút. Trong đó, đường truyền tin hai chiều trực tiếp từ nút i đến nút j có chi phí truyền thông tương ứng là một số nguyên A[i,j] = A[j,i], với A[i,j]>=0, i ≠ j. Nếu đường truyền tin từ nút i_1 đến nút i_k phải thông qua các nút i_2, . . i_{k-1} thì chi phí truyền thông được tính bằng tổng các chi phí truyền thông A[i_1,i_2], A[i_2,i_3], . . . A[i_{k-1},i_k]. Cho trước hai nút i và j. Hãy tìm một đường truyền tin từ nút i đến nút j sao cho chi phí truyền thông là thấp nhất.

Dữ liệu vào được cho bởi file TEXT có tên INP.NN. Trong đó, dòng thứ nhất ghi ba số N, i, j, dòng thứ k + 1 ghi k-1 số A[k,1], A[k,2], . . , A[k,k-1], 1<=k<=N.
Kết quả thông báo ra file TEXT có tên OUT.NN. Trong đó, dòng thứ nhất ghi chi phí truyền thông thấp nhất từ nút i đến nút j, dòng thứ 2 ghi lần lượt các nút trên đường truyền tin có chi phí truyền thông thấp nhất từ nút i tới nút j.

5. Cho một mạng thông tin gồm N nút. Trong đó, đường truyền tin hai chiều trực tiếp từ nút i đến nút j có chi phí truyền thông tương ứng là một số nguyên A[i,j] = A[j,i], với A[i,j]>=0, i ≠ j. Nếu đường truyền tin từ nút i_1 đến nút i_k phải thông qua các nút i_2, . . i_{k-1} thì chi phí truyền thông được tính bằng tổng các chi phí truyền thông A[i_1,i_2], A[i_2,i_3], . . . A[i_{k-1},i_k]. Biết rằng, giữa hai nút bất kỳ của mạng thông tin đều tồn tại ít nhất một đường truyền tin.

Để tiết kiệm đường truyền, người ta tìm cách loại bỏ đi một số đường truyền tin mà vẫn đảm bảo được tính liên thông của mạng. Hãy tìm một phương án loại bỏ đi những đường truyền tin, sao cho ta nhận được một mạng liên thông có chi phí tối thiểu nhất có thể được.

Dữ liệu vào được cho bởi file TEXT có tên INP.NN. Trong đó, dòng thứ nhất ghi số N, dòng thứ k + 1 ghi k-1 số A[k,1], A[k,2], . . , A[k,k-1], 1<=k<=N.

Kết quả thông báo ra file TEXT có tên OUT.NN trong đó dòng thứ nhất ghi chi phí truyền thông nhỏ nhất trong toàn mạng. Từ dòng thứ 2 ghi lần lượt các nút trên đường truyền tin, mỗi đường truyền ghi trên một dòng.

5. Cho đồ thị có hướng có trọng số được biểu diễn dưới dạng ma trận trọng số như dưới đây. Hãy thực hiện:

a) Trình bày thuật toán Dijkstra tìm đường đi ngắn nhất từ đỉnh $s \in V$ đến các đỉnh còn lại của đồ thị?

b) Tìm đường đi ngắn nhất từ đỉnh 1 đến tất cả các đỉnh còn lại của đồ thị? Chỉ rõ kết quả theo mỗi bước thực hiện của thuật toán?

c) Tìm đường đi ngắn nhất từ đỉnh 5 đến tất cả các đỉnh còn lại của đồ thị? Chỉ rõ kết quả theo mỗi bước thực hiện của thuật toán?

d) Viết chương trình tìm đường đi ngắn nhất từ đỉnh s đến tất cả các đỉnh còn lại của đồ thị?

∞	2	8	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
∞	∞	2	∞	∞	∞	9	∞	∞	∞	∞	∞	∞
∞	∞	∞	6	∞	8	1	∞	∞	∞	∞	∞	∞
7	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
∞	∞	1	7	∞	∞	∞	∞	∞	∞	∞	∞	∞
∞	∞	∞	∞	1	∞	∞	9	8	∞	∞	∞	∞
∞	∞	∞	∞	∞	2	∞	2	∞	∞	∞	∞	∞
∞	∞	∞	∞	∞	∞	∞	∞	9	∞	∞	2	∞
∞	∞	∞	∞	∞	∞	∞	∞	∞	6	∞	9	8
∞	∞	∞	∞	7	6	∞	∞	∞	∞	∞	∞	∞
∞	∞	∞	∞	∞	∞	∞	∞	6	7	∞	∞	∞
∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	2
∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	7	∞	∞

6. Cho đồ thị có hướng có trọng số được biểu diễn dưới dạng ma trận trọng số như dưới đây. Hãy thực hiện:

a) Trình bày thuật toán Bellman-Ford tìm đường đi ngắn nhất từ đỉnh $s \in V$ đến các đỉnh còn lại của đồ thị?

b) Tìm đường đi ngắn nhất từ đỉnh 1 đến tất cả các đỉnh còn lại của đồ thị? Chỉ rõ kết quả theo mỗi bước thực hiện của thuật toán?

c) Tìm đường đi ngắn nhất từ đỉnh 5 đến tất cả các đỉnh còn lại của đồ thị? Chỉ rõ kết quả theo mỗi bước thực hiện của thuật toán?

d) Viết chương trình tìm đường đi ngắn nhất từ đỉnh s đến tất cả các đỉnh còn lại của đồ thị?

∞	7	∞	9	4	∞	∞	∞	∞
∞	∞	3	∞	-4	∞	∞	∞	∞
∞	∞	∞	∞	-8	∞	-3	∞	∞
∞	∞	∞	∞	∞	∞	∞	-4	∞
∞	∞	∞	5	∞	2	∞	3	∞
∞	∞	∞	∞	∞	∞	5	∞	2
∞	∞	∞	∞	∞	∞	∞	∞	-7
∞	∞	∞	∞	∞	-2	∞	∞	-3
∞	∞	∞	∞	∞	∞	∞	∞	∞