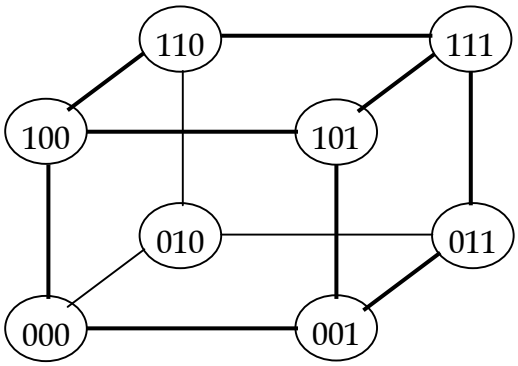
 www.VIETMATHS.com

BỘ GIÁO DỤC VÀ ĐÀO TẠO
ĐẠI HỌC HUẾ
TRƯỜNG ĐẠI HỌC KHOA HỌC

NGUYỄN GIA ĐỊNH

GIÁO TRÌNH

TOÁN RỜI RẠC



HUẾ – 2003

 www.VIETMATHS.com

LỜI NÓI ĐẦU

Được sự động viên mạnh mẽ của các đồng nghiệp trong các Khoa Toán-Cơ-Tin học, Công nghệ Thông tin và Vật lý (Trường Đại học Khoa học-Đại học Huế), các Khoa Toán và Tin học (Trường Đại học Sư phạm-Đại học Huế) và đặc biệt do nhu cầu học tập của các sinh viên trong Đại học Huế ở các Khoa nói trên và các học viên cao học ngành Phương pháp giảng dạy Toán, chúng tôi mạnh dạn viết giáo trình *Toán rời rạc* trong khi trên thị trường sách có khá nhiều tài liệu liên quan đến Toán rời rạc. Điều mà chúng tôi mong muốn là các kiến thức của học phần này phải được đưa vào đầy đủ, cô đọng, chính xác, cập nhật, bám sát theo yêu cầu đào tạo sinh viên các ngành Công nghệ Thông tin, Toán-Tin, Vật lý-Tin và một số ngành kỹ thuật khác của các trường đại học và cao đẳng.

Với sự nổ lực hết mình của bản thân, chúng tôi thiết nghĩ đây sẽ là tài liệu tham khảo tốt cho các giáo viên giảng dạy học phần toán rời rạc, các học viên cao học ngành Phương pháp giảng dạy Toán, các thí sinh thi vào cao học ngành công nghệ thông tin, các sinh viên thuộc các ngành được đề cập ở trên và các học sinh thuộc khối chuyên Toán, chuyên Tin.

Nội dung của tài liệu này được bố trí trong 4 phần, không kể lời nói đầu, mục lục, tài liệu tham khảo và phần phụ lục:

-- Phần 1 được dành cho Chương I đề cập đến *Thuật toán*;

-- Phần 2 được dành cho Chương II nói đến *bài toán đếm*;

-- Phần 3, đây là phần chiếm nhiều trang nhất trong giáo trình, bàn về *Lý thuyết đồ thị và các ứng dụng* gồm 5 chương: *Đồ thị, Đồ thị Euler và đồ thị Hamilton, Một số bài toán tối ưu trên đồ thị, Cây, Đồ thị phẳng và tô màu đồ thị*;

-- Phần 4 được dành cho Chương 8, chương cuối cùng, đề cập đến *Đại số Boole*.

Trong mỗi chương, các chứng minh của các định lý, mệnh đề được trình bày chi tiết, ngoại trừ một số định lý có phần chứng minh quá phức tạp thì được chúng tôi bỏ qua. Trong các phần của mỗi chương có nhiều ví dụ cụ thể minh hoạ cho những khái niệm cũng như những kết quả của chúng. Cuối của mỗi chương là những bài tập được chọn lọc từ dễ đến khó, bám theo nội dung của chương đó.

Chúng tôi xin chân thành cám ơn các đồng nghiệp đã động viên và góp ý cho công việc viết giáo trình *Toán rời rạc* này và lời cám ơn đặc biệt xin dành cho Khoa Công nghệ Thông tin về sự giúp đỡ quý báu và tạo điều kiện thuận lợi cho việc xuất bản giáo trình này.

Tác giả mong nhận được sự chỉ giáo của các đồng nghiệp và độc giả về những thiếu sót khó tránh khỏi của cuốn sách.

Mùa Thu năm 2003

 www.VIETMATHS.com

MỤC LỤC

 www.VIETMATHS.com

 www.VIETMATHS.com

CHƯƠNG I:
THUẬT TOÁN

1.1. KHÁI NIỆM THUẬT TOÁN.

1.1.1. Mở đầu:

Có nhiều lớp bài toán tổng quát xuất hiện trong toán học rời rạc. Chẳng hạn, cho một dãy các số nguyên, tìm số lớn nhất; cho một tập hợp, liệt kê các tập con của nó; cho tập hợp các số nguyên, xếp chúng theo thứ tự tăng dần; cho một mạng, tìm đường đi ngắn nhất giữa hai đỉnh của nó. Khi được giao cho một bài toán như vậy thì việc đầu tiên phải làm là xây dựng một mô hình dịch bài toán đó thành ngữ cảnh toán học. Các cấu trúc rời rạc được dùng trong các mô hình này là tập hợp, dãy, hàm, hoán vị, quan hệ, cùng với các cấu trúc khác như đồ thị, cây, mạng - những khái niệm sẽ được nghiên cứu ở các chương sau.

Lập được một mô hình toán học thích hợp chỉ là một phần của quá trình giải. Để hoàn tất quá trình giải, còn cần phải có một phương pháp dùng mô hình để giải bài toán tổng quát. Nói một cách lý tưởng, cái được đòi hỏi là một thủ tục, đó là dãy các bước dẫn tới đáp số mong muốn. Một dãy các bước như vậy, được gọi là một thuật toán.

Khi thiết kế và cài đặt một phần mềm tin học cho một vấn đề nào đó, ta cần phải đưa ra phương pháp giải quyết mà thực chất đó là thuật toán giải quyết vấn đề này. Rõ ràng rằng, nếu không tìm được một phương pháp giải quyết thì không thể lập trình được. Chính vì thế, thuật toán là khái niệm nền tảng của hầu hết các lĩnh vực của tin học.

1.1.2. Định nghĩa: Thuật toán là một bảng liệt kê các chỉ dẫn (hay quy tắc) cần thực hiện theo từng bước xác định nhằm giải một bài toán đã cho.

Thuật ngữ "Algorithm" (thuật toán) là xuất phát từ tên nhà toán học Ả Rập Al-Khowarizmi. Ban đầu, từ algorism được dùng để chỉ các quy tắc thực hiện các phép tính số học trên các số thập phân. Sau đó, algorism chuyển thành algorithm vào thế kỷ 19. Với sự quan tâm ngày càng tăng đối với các máy tính, khái niệm thuật toán đã được cho một ý nghĩa chung hơn, bao hàm cả các thủ tục xác định để giải các bài toán, chứ không phải chỉ là thủ tục để thực hiện các phép tính số học.

Có nhiều cách trình bày thuật toán: dùng ngôn ngữ tự nhiên, ngôn ngữ lưu đồ (sơ đồ khối), ngôn ngữ lập trình. Tuy nhiên, một khi dùng ngôn ngữ lập trình thì chỉ những lệnh được phép trong ngôn ngữ đó mới có thể dùng được và điều này thường làm cho sự mô tả các thuật toán trở nên rối rắm và khó hiểu. Hơn nữa, vì nhiều ngôn ngữ lập trình đều được dùng rộng rãi, nên chọn một ngôn ngữ đặc biệt nào đó là điều người ta không muốn. Vì vậy ở đây các thuật toán ngoài việc được trình bày bằng ngôn ngữ tự nhiên cùng với những ký hiệu toán học quen thuộc còn dùng một dạng giả mã để mô tả thuật

toán. Giả mã tạo ra bước trung gian giữa sự mô tả một thuật toán bằng ngôn ngữ thông thường và sự thực hiện thuật toán đó trong ngôn ngữ lập trình. Các bước của thuật toán được chỉ rõ bằng cách dùng các lệnh giống như trong các ngôn ngữ lập trình.

Thí dụ 1: Mô tả thuật toán tìm phần tử lớn nhất trong một dãy hữu hạn các số nguyên.

a) Dùng ngôn ngữ tự nhiên để mô tả các bước cần phải thực hiện:

1. Đặt giá trị cực đại tạm thời bằng số nguyên đầu tiên trong dãy. (Cực đại tạm thời sẽ là số nguyên lớn nhất đã được kiểm tra ở một giai đoạn nào đó của thủ tục.)

2. So sánh số nguyên tiếp sau với giá trị cực đại tạm thời, nếu nó lớn hơn giá trị cực đại tạm thời thì đặt cực đại tạm thời bằng số nguyên đó.

3. Lặp lại bước trước nếu còn các số nguyên trong dãy.

4. Dừng khi không còn số nguyên nào nữa trong dãy. Cực đại tạm thời ở điểm này chính là số nguyên lớn nhất của dãy.

b) Dùng đoạn giả mã:

---

procedure max (a_1, a_2, ..., a_n: integers)

max:= a_1

for i:= 2 **to** n

if max <a_i **then** max:= a_i

{max là phần tử lớn nhất}

Thuật toán này trước hết gán số hạng đầu tiên a_1 của dãy cho biến max. Vòng lặp "for" được dùng để kiểm tra lần lượt các số hạng của dãy. Nếu một số hạng lớn hơn giá trị hiện thời của max thì nó được gán làm giá trị mới của max.

1.1.3. Các đặc trưng của thuật toán:

-- **Đầu vào** (Input)**:** Một thuật toán có các giá trị đầu vào từ một tập đã được chỉ rõ.

-- Đầu ra (Output)**:** Từ mỗi tập các giá trị đầu vào, thuật toán sẽ tạo ra các giá trị đầu ra. Các giá trị đầu ra chính là nghiệm của bài toán.

-- **Tính dừng:** Sau một số hữu hạn bước thuật toán phải dừng.

-- **Tính xác định:** Ở mỗi bước, các bước thao tác phải hết sức rõ ràng, không gây nên sự nhập nhằng. Nói rõ hơn, trong cùng một điều kiện hai bộ xử lý cùng thực hiện một bước của thuật toán phải cho những kết quả như nhau.

-- **Tính hiệu quả:** Trước hết thuật toán cần đúng đắn, nghĩa là sau khi đưa dữ liệu vào thuật toán hoạt động và đưa ra kết quả như ý muốn.

-- **Tính phổ dụng:** Thuật toán có thể giải bất kỳ một bài toán nào trong lớp các bài toán. Cụ thể là thuật toán có thể có các đầu vào là các bộ dữ liệu khác nhau trong một miền xác định.

1.2. THUẬT TOÁN TÌM KIẾM.

1.2.1. Bài toán tìm kiếm: Bài toán xác định vị trí của một phần tử trong một bảng liệt kê sắp thứ tự thường gặp trong nhiều trường hợp khác nhau. Chẳng hạn chương trình

kiểm tra chính tả của các từ, tìm kiếm các từ này trong một cuốn từ điển, mà từ điển chẳng qua cũng là một bảng liệt kê sắp thứ tự của các từ. Các bài toán thuộc loại này được gọi là các bài toán tìm kiếm.

Bài toán tìm kiếm tổng quát được mô tả như sau: xác định vị trí của phần tử x trong một bảng liệt kê các phần tử phân biệt $a_1, a_2, ..., a_n$ hoặc xác định rằng nó không có mặt trong bảng liệt kê đó. Lời giải của bài toán trên là vị trí của số hạng của bảng liệt kê có giá trị bằng x (tức là i sẽ là nghiệm nếu $x=a_i$ và là 0 nếu x không có mặt trong bảng liệt kê).

1.2.2. Thuật toán tìm kiếm tuyến tính: Tìm kiếm tuyến tính hay tìm kiếm tuần tự là bắt đầu bằng việc so sánh x với a_1; khi $x=a_1$, nghiệm là vị trí a_1, tức là 1; khi $x\neq a_1$, so sánh x với a_2. Nếu $x=a_2$, nghiệm là vị trí của a_2, tức là 2. Khi $x\neq a_2$, so sánh x với a_3. Tiếp tục quá trình này bằng cách tuần tự so sánh x với mỗi số hạng của bảng liệt kê cho tới khi tìm được số hạng bằng x, khi đó nghiệm là vị trí của số hạng đó. Nếu toàn bảng liệt kê đã được kiểm tra mà không xác định được vị trí của x, thì nghiệm là 0. Giả mã đối với thuật toán tìm kiếm tuyến tính được cho dưới đây:

---

procedure tìm kiếm tuyến tính (x: integer, $a_1,a_2,...,a_n$: integers phân biệt)

```
i := 1
while (i ≤ n and x ≠ a_i)
        i := i + 1
if  i ≤ n then location := i
else location := 0
```

{location là chỉ số dưới của số hạng bằng x hoặc là 0 nếu không tìm được x}

1.2.3. Thuật toán tìm kiếm nhị phân: Thuật toán này có thể được dùng khi bảng liệt kê có các số hạng được sắp theo thứ tự tăng dần. Chẳng hạn, nếu các số hạng là các số thì chúng được sắp từ số nhỏ nhất đến số lớn nhất hoặc nếu chúng là các từ hay xâu ký tự thì chúng được sắp theo thứ tự từ điển. Thuật toán thứ hai này gọi là thuật toán tìm kiếm nhị phân. Nó được tiến hành bằng cách so sánh phần tử cần xác định vị trí với số hạng ở giữa bảng liệt kê. Sau đó bảng này được tách làm hai bảng kê con nhỏ hơn có kích thước như nhau, hoặc một trong hai bảng con ít hơn bảng con kia một số hạng. Sự tìm kiếm tiếp tục bằng cách hạn chế tìm kiếm ở một bảng kê con thích hợp dựa trên việc so sánh phần tử cần xác định vị trí với số hạng giữa bảng kê. Ta sẽ thấy rằng thuật toán tìm kiếm nhị phân hiệu quả hơn nhiều so với thuật toán tìm kiếm tuyến tính.

Thí dụ 2. Để tìm số 19 trong bảng liệt kê 1,2,3,5,6,7,8,10,12,13,15,16,18,19,20,22 ta tách bảng liệt kê gồm 16 số hạng này thành hai bảng liệt kê nhỏ hơn, mỗi bảng có 8 số hạng, cụ thể là: 1,2,3,5,6,7,8,10 và 12,13,15,16,18,19,20,22. Sau đó ta so sánh 19 với số hạng cuối cùng của bảng con thứ nhất. Vì 10<19, việc tìm kiếm 19 chỉ giới hạn trong bảng liệt kê con thứ 2 từ số hạng thứ 9 đến 16 trong bảng liệt kê ban đầu. Tiếp theo, ta

lại tách bảng liệt kê con gồm 8 số hạng này làm hai bảng con, mỗi bảng có 4 số hạng, cụ thể là 12,13,15,16 và 18,19,20,22. Vì 16<19, việc tìm kiếm lại được giới hạn chỉ trong bảng con thứ 2, từ số hạng thứ 13 đến 16 của bảng liệt kê ban đầu. Bảng liệt kê thứ 2 này lại được tách làm hai, cụ thể là: 18,19 và 20,22. Vì 19 không lớn hơn số hạng lớn nhất của bảng con thứ nhất nên việc tìm kiếm giới hạn chỉ ở bảng con thứ nhất gồm các số 18,19, là số hạng thứ 13 và 14 của bảng ban đầu. Tiếp theo bảng con chứa hai số hạng này lại được tách làm hai, mỗi bảng có một số hạng 18 và 19. Vì 18<19, sự tìm kiếm giới hạn chỉ trong bảng con thứ 2, bảng liệt kê chỉ chứa số hạng thứ 14 của bảng liệt kê ban đầu, số hạng đó là số 19. Bây giờ sự tìm kiếm đã thu hẹp về chỉ còn một số hạng, so sánh tiếp cho thấy19 là số hạng thứ 14 của bảng liệt kê ban đầu.

Bây giờ ta có thể chỉ rõ các bước trong thuật toán tìm kiếm nhị phân.

Để tìm số nguyên x trong bảng liệt kê $a_1,a_2,...,a_n$ với $a_1 < a_2 < ... < a_n$, ta bắt đầu bằng việc so sánh x với số hạng a_m ở giữa của dãy, với m=[(n+1)/2]. Nếu $x > a_m$, việc tìm kiếm x giới hạn ở nửa thứ hai của dãy, gồm $a_{m+1},a_{m+2},...,a_n$. Nếu x không lớn hơn a_m, thì sự tìm kiếm giới hạn trong nửa đầu của dãy gồm $a_1,a_2,...,a_m$.

Bây giờ sự tìm kiếm chỉ giới hạn trong bảng liệt kê có không hơn [n/2] phần tử. Dùng chính thủ tục này, so sánh x với số hạng ở giữa của bảng liệt kê được hạn chế. Sau đó lại hạn chế việc tìm kiếm ở nửa thứ nhất hoặc nửa thứ hai của bảng liệt kê. Lặp lại quá trình này cho tới khi nhận được một bảng liệt kê chỉ có một số hạng. Sau đó, chỉ còn xác định số hạng này có phải là x hay không. Giả mã cho thuật toán tìm kiếm nhị phân được cho dưới đây:

```
procedure tìm kiếm nhị phân (x: integer, a1,a2,...,an: integers tăng dần)
    i := 1  {i là điểm mút trái của khoảng tìm kiếm}
    j := n  {j là điểm mút phải của khoảng tìm kiếm}
    while i < j
    begin
        m:= [(i+j)/2]
        if x>am then i:=m+1
        else  j := m
    end
        if  x = ai  then  location := i
        else  location := 0
{location là chỉ số dưới của số hạng bằng x hoặc 0 nếu không tìm thấy x}
```

1.3. ĐỘ PHỨC TẠP CỦA THUẬT TOÁN.

1.3.1. Khái niệm về độ phức tạp của một thuật toán:

Thước đo hiệu quả của một thuật toán là thời gian mà máy tính sử dụng để giải bài toán theo thuật toán đang xét, khi các giá trị đầu vào có một kích thước xác định.

Một thước đo thứ hai là dung lượng bộ nhớ đòi hỏi để thực hiện thuật toán khi các giá trị đầu vào có kích thước xác định. Các vấn đề như thế liên quan đến độ phức tạp tính toán của một thuật toán. Sự phân tích thời gian cần thiết để giải một bài toán có kích thước đặc biệt nào đó liên quan đến độ phức tạp thời gian của thuật toán. Sự phân tích bộ nhớ cần thiết của máy tính liên quan đến độ phức tạp không gian của thuật toán. Vệc xem xét độ phức tạp thời gian và không gian của một thuật toán là một vấn đề rất thiết yếu khi các thuật toán được thực hiện. Biết một thuật toán sẽ đưa ra đáp số trong một micro giây, trong một phút hoặc trong một tỉ năm, hiển nhiên là hết sức quan trọng. Tương tự như vậy, dung lượng bộ nhớ đòi hỏi phải là khả dụng để giải một bài toán,vì vậy độ phức tạp không gian cũng cần phải tính đến.Vì việc xem xét độ phức tạp không gian gắn liền với các cấu trúc dữ liệu đặc biệt được dùng để thực hiện thuật toán nên ở đây ta sẽ tập trung xem xét độ phức tạp thời gian.

Độ phức tạp thời gian của một thuật toán có thể được biểu diễn qua số các phép toán được dùng bởi thuật toán đó khi các giá trị đầu vào có một kích thước xác định. Sở dĩ độ phức tạp thời gian được mô tả thông qua số các phép toán đòi hỏi thay vì thời gian thực của máy tính là bởi vì các máy tính khác nhau thực hiện các phép tính sơ cấp trong những khoảng thời gian khác nhau. Hơn nữa, phân tích tất cả các phép toán thành các phép tính bit sơ cấp mà máy tính sử dụng là điều rất phức tạp.

Thí dụ 3: Xét thuật toán tìm số lớn nhất trong dãy n số a_1, a_2, ..., a_n. Có thể coi kích thước của dữ liệu nhập là số lượng phần tử của dãy số, tức là n. Nếu coi mỗi lần so sánh hai số của thuật toán đòi hỏi một đơn vị thời gian (giây chẳng hạn) thì thời gian thực hiện thuật toán trong trường hợp xấu nhất là n-1 giây. Với dãy 64 số, thời gian thực hiện thuật toán nhiều lắm là 63 giây.

Thí dụ 4: Thuật toán về trò chơi "Tháp Hà Nội"

Trò chơi "Tháp Hà Nội" như sau: Có ba cọc A, B, C và 64 cái đĩa (có lỗ để đặt vào cọc), các đĩa có đường kính đôi một khác nhau. Nguyên tắc đặt đĩa vào cọc là: mỗi đĩa chỉ được chồng lên đĩa lớn hơn nó. Ban đầu, cả 64 đĩa được đặt chồng lên nhau ở cột A; hai cột B, C trống. Vấn đề là phải chuyển cả 64 đĩa đó sang cột B hay C, mỗi lần chỉ được di chuyển một đĩa.

Xét trò chơi với n đĩa ban đầu ở cọc A (cọc B và C trống). Gọi S_n là số lần chuyển đĩa để chơi xong trò chơi với n đĩa.

Nếu n=1 thì rõ ràng là $S_1=1$.

Nếu n>1 thì trước hết ta chuyển n-1 đĩa bên trên sang cọc B (giữ yên đĩa thứ n ở dưới cùng của cọc A). Số lần chuyển n-1 đĩa là S_{n-1}. Sau đó ta chuyển đĩa thứ n từ cọc A sang cọc C. Cuối cùng, ta chuyển n-1 đĩa từ cọc B sang cọc C (số lần chuyển là S_{n-1}).

Như vậy, số lần chuyển n đĩa từ A sang C là:

$S_n=S_{n-1}+1+S_n=2S_{n-1}+1=2(2S_{n-2}+1)+1=2^2S_{n-2}+2+1=.....=2^{n-1}S_1+2^{n-2}+...+2+1=2^n-1$.

Thuật toán về trò chơi "Tháp Hà Nội" đòi hỏi $2^{64}-1$ lần chuyển đĩa (xấp xỉ 18,4 tỉ tỉ lần). Nếu mỗi lần chuyển đĩa mất 1 giây thì thời gian thực hiện thuật toán xấp xỉ 585 tỉ năm!

Hai thí dụ trên cho thấy rằng: một thuật toán phải kết thúc sau một số hữu hạn bước, nhưng nếu số hữu hạn này quá lớn thì thuật toán không thể thực hiện được trong thực tế.

Ta nói: thuật toán trong Thí dụ 3 có độ phức tạp là n-1 và là một thuật toán hữu hiệu (hay thuật toán nhanh); thuật toán trong Thí dụ 4 có độ phức tạp là 2^n-1 và đó là một thuật toán không hữu hiệu (hay thuật toán chậm).

1.3.2. So sánh độ phức tạp của các thuật toán:

Một bài toán thường có nhiều cách giải, có nhiều thuật toán để giải, các thuật toán đó có độ phức tạp khác nhau.

Xét bài toán: Tính giá trị của đa thức $P(x)=a_nx^n+a_{n-1}x^{n-1}+ ... +a_1x+a_0$ tại x_0.

Thuật toán 1:

Procedure tính giá trị của đa thức (a_0, a_1, ..., a_n, x_0: các số thực)

sum:=a_0

for i:=1 **to** n

sum:=sum+$a_ix_0^i$

{sum là giá trị của đa thức P(x) tại x_0}

Chú ý rằng đa thức P(x) có thể viết dưới dạng:

$P(x)=(...((a_nx+a_{n-1})x+a_{n-2})x...)x+a_0$.

Ta có thể tính P(x) theo thuật toán sau:

Thuật toán 2:

Procedure tính giá trị của đa thức (a_0, a_1, ..., a_n, x_0: các số thực)

P:=a_n

for i:=1 **to** n

P:=P.x_0+a_{n-i}

{P là giá trị của đa thức P(x) tại x_0}

Ta hãy xét độ phức tạp của hai thuật toán trên.

Đối với thuật toán 1: ở bước 2, phải thực hiện 1 phép nhân và 1 phép cộng với i=1; 2 phép nhân và 1 phép cộng với i=2, ..., n phép nhân và 1 phép cộng với i=n. Vậy số phép tính (nhân và cộng) mà thuật toán 1 đòi hỏi là:

$$(1+1)+(2+1)+ ... +(n+1)=\frac{n(n+1)}{2}+n=\frac{n(n+3)}{2}.$$

Đối với thuật toán 2, bước 2 phải thực hiện n lần, mỗi lần đòi hỏi 2 phép tính (nhân rồi cộng), do đó số phép tính (nhân và cộng) mà thuật toán 2 đòi hỏi là 2n.

Nếu coi thời gian thực hiện mỗi phép tính nhân và cộng là như nhau và là một đơn vị thời gian thì với mỗi n cho trước, thời gian thực hiện thuật toán 1 là n(n+3)/2, còn thời gian thực hiện thuật toán 2 là 2n.

Rõ ràng là thời gian thực hiện thuật toán 2 ít hơn so với thời gian thực hiện thuật toán 1. Hàm $f_1(n)=2n$ là hàm bậc nhất, tăng chậm hơn nhiều so với hàm bậc hai $f_2(n)=n(n+3)/2$.

Ta nói rằng thuật toán 2 (có độ phức tạp là 2n) là thuật toán hữu hiệu hơn (hay nhanh hơn) so với thuật toán 1 (có độ phức tạp là n(n+3)/2).

Để so sánh độ phức tạp của các thuật toán, điều tiện lợi là coi độ phức tạp của mỗi thuật toán như là cấp của hàm biểu hiện thời gian thực hiện thuật toán ấy.

Các hàm xét sau đây đều là hàm của biến số tự nhiên n>0.

Định nghĩa 1: Ta nói hàm f(n) có cấp thấp hơn hay bằng hàm g(n) nếu tồn tại hằng số C>0 và một số tự nhiên n_0 sao cho

$$|f(n)| \le C|g(n)| \text{ với mọi } n \ge n_0.$$

Ta viết f(n)=O(g(n)) và còn nói f(n) thoả mãn quan hệ big-O đối với g(n).

Theo định nghĩa này, hàm g(n) là một hàm đơn giản nhất có thể được, đại diện cho "sự biến thiên" của f(n).

Khái niệm big-O đã được dùng trong toán học đã gần một thế kỷ nay. Trong tin học, nó được sử dụng rộng rãi để phân tích các thuật toán. Nhà toán học người Đức Paul Bachmann là người đầu tiên đưa ra khái niệm big-O vào năm 1892.

Thí dụ 5: Hàm $f(n)=\frac{n(n+3)}{2}$ là hàm bậc hai và hàm bậc hai đơn giản nhất là n^2. Ta có:

$$f(n)=\frac{n(n+3)}{2}=O(n^2) \text{ vì } \frac{n(n+3)}{2} \le n^2 \text{ với mọi } n\ge 3 \ (C=1, n_0=3).$$

Một cách tổng quát, nếu $f(n)=a_kn^k+a_{k-1}n^{k-1}+ \dots +a_1n+a_0$ thì $f(n)=O(n^k)$. Thật vậy, với n>1,

$$|f(n)| \le |a_k|n^k+|a_{k-1}|n^{k-1}+ \dots +|a_1|n+|a_0| = n^k(|a_k|+|a_{k-1}|/n+ \dots +|a_1|/n^{k-1}+a_0/n^k)$$
$$\le n^k(|a_k|+|a_{k-1}|+ \dots +|a_1|+a_0).$$

Điều này chứng tỏ $|f(n)| \le Cn^k$ với mọi n>1.

Cho $g(n)=3n+5n\log_2 n$, ta có $g(n)=O(n\log_2 n)$. Thật vậy,

$$3n+5n\log_2 n = n(3+5\log_2 n) \le n(\log_2 n+5\log_2 n) = 6n\log_2 n \text{ với mọi } n\ge 8 \ (C=6, n_0=8).$$

Mệnh đề: Cho $f_1(n)=O(g_1(n))$ và $f_2(n)$ là $O(g_2(n))$. Khi đó

$$(f_1 + f_2)(n) = O(\max(|g_1(n)|,|g_2(n)|)), \ (f_1f_2)(n) = O(g_1(n)g_2(n)).$$

Chứng minh. Theo giả thiết, tồn tại C_1, C_2, n_1, n_2 sao cho

$$|f_1(n)| \le C_1|g_1(n)| \text{ và } |f_2(n)| \le C_2|g_2(n)| \text{ với mọi } n > n_1 \text{ và mọi } n > n_2.$$

Do đó $|(f_1 + f_2)(n)| = |f_1(n) + f_2(n)| \le |f_1(n)| + |f_2(n)| \le C_1|g_1(n)| + C_2|g_2(n)| \le (C_1+C_2)g(n)$ với mọi $n > n_0=\max(n_1,n_2)$, ở đây $C=C_1+C_2$ và $g(n)=\max(|g_1(n)| , |g_2(n)|)$.

$|(f_1f_2)(n)| = |f_1(n)||f_2(n)| \le C_1|g_1(n)|C_2|g_2(n)| \le C_1C_2|(g_1g_2)(n)|$ với mọi $n > n_0=\max(n_1,n_2)$.

Định nghĩa 2: Nếu một thuật toán có độ phức tạp là f(n) với f(n)=O(g(n)) thì ta cũng nói thuật toán có độ phức tạp O(g(n)).

Nếu có hai thuật toán giải cùng một bài toán, thuật toán 1 có độ phức tạp $O(g_1(n))$, thuật toán 2 có độ phức tạp $O(g_2(n))$, mà $g_1(n)$ có cấp thấp hơn $g_2(n)$, thì ta nói rằng thuật toán 1 hữu hiệu hơn (hay nhanh hơn) thuật toán 2.

1.3.3. Đánh giá độ phức tạp của một thuật toán:

1) Thuật toán tìm kiếm tuyến tính:

Số các phép so sánh được dùng trong thuật toán này cũng sẽ được xem như thước đo độ phức tạp thời gian của nó. Ở mỗi một bước của vòng lặp trong thuật toán, có hai phép so sánh được thực hiện: một để xem đã tới cuối bảng chưa và một để so sánh phần tử x với một số hạng của bảng. Cuối cùng còn một phép so sánh nữa làm ở ngoài vòng lặp. Do đó, nếu $x=a_i$, thì đã có 2i+1 phép so sánh được sử dụng. Số phép so sánh nhiều nhất, 2n+2, đòi hỏi phải được sử dụng khi phần tử x không có mặt trong bảng. Từ đó, thuật toán tìm kiếm tuyến tính có độ phức tạp là O(n).

2) Thuật toán tìm kiếm nhị phân:

Để đơn giản, ta giả sử rằng có $n=2^k$ phần tử trong bảng liệt kê $a_1,a_2,...,a_n$, với k là số nguyên không âm (nếu n không phải là lũy thừa của 2, ta có thể xem bảng là một phần của bảng gồm 2^{k+1} phần tử, trong đó k là số nguyên nhỏ nhất sao cho $n < 2^{k+1}$).

Ở mỗi giai đoạn của thuật toán vị trí của số hạng đầu tiên i và số hạng cuối cùng j của bảng con hạn chế tìm kiếm ở giai đoạn đó được so sánh để xem bảng con này còn nhiều hơn một phần tử hay không. Nếu i < j, một phép so sánh sẽ được làm để xác định x có lớn hơn số hạng ở giữa của bảng con hạn chế hay không. Như vậy ở mỗi giai đoạn, có sử dụng hai phép so sánh. Khi trong bảng chỉ còn một phần tử, một phép so sánh sẽ cho chúng ta biết rằng không còn một phần tử nào thêm nữa và một phép so sánh nữa cho biết số hạng đó có phải là x hay không. Tóm lại cần phải có nhiều nhất $2k+2=2\log_2 n+2$ phép so sánh để thực hiện phép tìm kiếm nhị phân (nếu n không phải là lũy thừa của 2, bảng gốc sẽ được mở rộng tới bảng có 2^{k+1} phần tử, với $k=[\log_2 n]$ và sự tìm kiếm đòi hỏi phải thực hiện nhiều nhất $2[\log_2 n]+2$ phép so sánh). Do đó thuật toán tìm kiếm nhị phân có độ phức tạp là $O(\log_2 n)$. Từ sự phân tích ở trên suy ra rằng thuật toán tìm kiếm nhị phân, ngay cả trong trường hợp xấu nhất, cũng hiệu quả hơn thuật toán tìm kiếm tuyến tính.

3) Chú ý: Một điều quan trọng cần phải biết là máy tính phải cần bao lâu để giải xong một bài toán. Thí dụ, nếu một thuật toán đòi hỏi 10 giờ, thì có thể còn đáng chi phí thời gian máy tính đòi hỏi để giải bài toán đó. Nhưng nếu một thuật toán đòi hỏi 10 tỉ năm để giải một bài toán, thì thực hiện thuật toán đó sẽ là một điều phi lý. Một trong những hiện tượng lý thú nhất của công nghệ hiện đại là sự tăng ghê gớm của tốc độ và lượng bộ nhớ trong máy tính. Một nhân tố quan trọng khác làm giảm thời gian cần thiết để giải một

bài toán là sự xử lý song song - đây là kỹ thuật thực hiện đồng thời các dãy phép tính. Do sự tăng tốc độ tính toán và dung lượng bộ nhớ của máy tính, cũng như nhờ việc dùng các thuật toán lợi dụng được ưu thế của kỹ thuật xử lý song song, các bài toán vài năm trước đây được xem là không thể giải được, thì bây giờ có thể giải bình thường.

1. Các thuật ngữ thường dùng cho độ phức tạp của một thuật toán:

Độ phức tạp	Thuật ngữ
O(1)	Độ phức tạp hằng số
O(logn)	Độ phức tạp lôgarit
O(*n*)	Độ phức tạp tuyến tính
O(*nlogn*)	Độ phức tạp *nlogn*
$O(n^b)$	Độ phức tạp đa thức
$O(b^n)$ (b>1)	Độ phức tạp hàm mũ
O(n!)	Độ phức tạp giai thừa

2. Thời gian máy tính được dùng bởi một thuật toán:

Kích thước của bài toán	Các phép tính bit được sử dụng					
n	logn	*N*	*nlogn*	n^2	2^n	*n*!
10	3.10^{-9} s	10^{-8} s	3.10^{-8} s	10^{-7} s	10^{-6} s	3.10^{-3} s
10^2	7.10^{-9} s	10^{-7} s	7.10^{-7} s	10^{-5} s	4.10^{13}năm	*
10^3	$1,0.10^{-8}$ s	10^{-6} s	1.10^{-5} s	10^{-3} s	*	*
10^4	$1,3.10^{-8}$ s	10^{-5} s	1.10^{-4} s	10^{-1} s	*	*
10^5	$1,7.10^{-8}$ s	10^{-4} s	2.10^{-3} s	10 s	*	*
10^6	2.10^{-8} s	10^{-3} s	2.10^{-2} s	17 phút	*	*

1.4. SỐ NGUYÊN VÀ THUẬT TOÁN.

1.4.1. Thuật toán Euclide:

Phương pháp tính ước chung lớn nhất của hai số bằng cách dùng phân tích các số nguyên đó ra thừa số nguyên tố là không hiệu quả. Lý do là ở chỗ thời gian phải tiêu tốn cho sự phân tích đó. Dưới đây là phương pháp hiệu quả hơn để tìm ước số chung lớn nhất, gọi là **thuật toán Euclide**. Thuật toán này đã biết từ thời cổ đại. Nó mang tên nhà toán học cổ Hy lạp Euclide, người đã mô tả thuật toán này trong cuốn sách "*Những yếu tố*" nổi tiếng của ông. Thuật toán Euclide dựa vào 2 mệnh đề sau đây.

Mệnh đề 1 (Thuật toán chia): Cho a và b là hai số nguyên và b≠0. Khi đó tồn tại duy nhất hai số nguyên q và r sao cho

$$a = bq+r, \quad 0 \le r < |b|.$$

Trong đẳng thức trên, b được gọi là số chia, a được gọi là số bị chia, q được gọi là thương số và r được gọi là số dư.

Khi b là nguyên dương, ta ký hiệu số dư r trong phép chia a cho b là a mod b.

Mệnh đề 2: Cho a = bq + r, trong đó a, b, q, r là các số nguyên. Khi đó

$$UCLN(a,b) = UCLN(b,r).$$

(Ở đây UCLN(a,b) để chỉ ước chung lớn nhất của a và b.)

Giả sử a và b là hai số nguyên dương với $a \geq b$. Đặt $r_0 = a$ và $r_1 = b$. Bằng cách áp dụng liên tiếp thuật toán chia, ta tìm được:

$r_0 = r_1q_1 + r_2$ $\quad 0 \leq r_2 < r_1$

$r_1 = r_2q_2 + r_3$ $\quad 0 \leq r_3 < r_2$

.....

$r_{n-2} = r_{n-1}q_{n-1} + r_n$ $\quad 0 \leq r_n < r_{n-1}$

$r_{n-1} = r_nq_n$.

Cuối cùng, số dư 0 sẽ xuất hiện trong dãy các phép chia liên tiếp, vì dãy các số dư

$$a = r_0 > r_1 > r_2 > ... \geq 0$$

không thể chứa quá a số hạng được. Hơn nữa, từ Mệnh đề 2 ở trên ta suy ra:

$UCLN(a,b) = UCLN(r_0,r_1) = UCLN(r_1,r_2) = ... = UCLN(r_{n-2}, r_{n-1}) = UCLN(r_{n-1},r_n) = r_n.$

Do đó, ước chung lớn nhất là số dư khác không cuối cùng trong dãy các phép chia.

Thí dụ 6: Dùng thuật toán Euclide tìm UCLN(414, 662).

662 = 441.1 + 248

414 = 248.1 + 166

248 = 166.1+ 82

166 = 82.2 + 2

82 = 2.41.

Do đó, UCLN(414, 662) = 2.

Thuật toán Euclide được viết dưới dạng giả mã như sau:

```
procedure ƯCLN (a,b: positive integers)
    x :=  a
    y := b
    while y ≠ 0
    begin
        r  := x mod y
        x := y
        y := r
    end
{UCLN (a,b) là x}
```

Trong thuật toán trên, các giá trị ban đầu của x và y tương ứng là a và b. Ở mỗi giai đoạn của thủ tục, x được thay bằng y và y được thay bằng x mod y. Quá trình này được lặp lại chừng nào $y \neq 0$. Thuật toán sẽ ngừng khi y = 0 và giá trị của x ở điểm này, đó là số dư khác không cuối cùng trong thủ tục, cũng chính là ước chung lớn nhất của a và b.

1.4.2. Biểu diễn các số nguyên:

Mệnh đề 3: Cho b là một số nguyên dương lớn hơn 1. Khi đó nếu n là một số nguyên dương, nó có thể được biểu diễn một cách duy nhất dưới dạng:

$$n = a_k b^k + a_{k-1} b^{k-1} + ... + a_1 b + a_0.$$

Ở đây k là một số tự nhiên, a_0, a_1,..., a_k là các số tự nhiên nhỏ hơn b và $a_k \neq 0$.

Biểu diễn của n được cho trong Mệnh đề 3 được gọi là khai triển của n theo cơ số b, ký hiệu là $(a_k a_{k-1} ... a_1 a_0)_b$. Bây giờ ta sẽ mô tả thuật toán xây dựng khai triển cơ số b của số nguyên n bất kỳ. Trước hết ta chia n cho b để được thương và số dư, tức là

$$n = bq_0 + a_0, \ 0 \leq a_0 < b.$$

Số dư a_0 chính là chữ số đứng bên phải cùng trong khai triển cơ số b của n. Tiếp theo chia q_0 cho b, ta được:

$$q_0 = bq_1 + a_1, \ 0 \leq a_1 < b.$$

Số dư a_1 chính là chữ số thứ hai tính từ bên phải trong khai triển cơ số b của n. Tiếp tục quá trình này, bằng cách liên tiếp chia các thương cho b ta sẽ được các chữ số tiếp theo trong khai triển cơ số b của n là các số dư tương ứng. Quá trình này sẽ kết thúc khi ta nhận được một thương bằng 0.

Thí dụ 7: Tìm khai triển cơ số 8 của $(12345)_{10}$.

$12345 = 8.1543 + 1$
$1543 = 8.192 + 7$
$192 = 8.24 + 0$
$24 = 8.3 + 0$
$3 = 8.0 + 3.$

Do đó, $(12345)_{10} = (30071)_8$.

Đoạn giả mã sau biểu diễn thuật toán tìm khai triển cơ số b của số nguyên n.

---

procedure *khai triển theo cơ số b* (n: positive integer)
    q := n
    k := 0
    while $q \neq 0$
    begin
        a_k := q **mod** b
        q := $[\frac{q}{b}]$
        k := k + 1
    end

1.4.3. Thuật toán cho các phép tính số nguyên:

Các thuật toán thực hiện các phép tính với những số nguyên khi dùng các khai triển nhị phân của chúng là cực kỳ quan trọng trong số học của máy tính. Ta sẽ mô tả ở

đây các thuật toán cộng và nhân hai số nguyên trong biểu diễn nhị phân. Ta cũng sẽ phân tích độ phức tạp tính toán của các thuật toán này thông qua số các phép toán bit thực sự được dùng. Giả sử khai triển nhị phân của hai số nguyên dương a và b là:

$$a = (a_{n-1}a_{n-2} \dots a_1 a_0)_2 \text{ và } b = (b_{n-1}\, b_{n-2} \dots b_1\, b_0)_2$$

sao cho a và b đều có n bit (đặt các bit 0 ở đầu mỗi khai triển đó, nếu cần).

1) Phép cộng: Xét bài toán cộng hai số nguyên viết ở dạng nhị phân. Thủ tục thực hiện phép cộng có thể dựa trên phương pháp thông thường là cộng cặp chữ số nhị phân với nhau (có nhớ) để tính tổng của hai số nguyên.

Để cộng a và b, trước hết cộng hai bit ở phải cùng của chúng, tức là:

$$a_0 + b_0 = c_0.2 + s_0.$$

Ở đây s_0 là bit phải cùng trong khai triển nhị phân của a+b, c_0 là số nhớ, nó có thể bằng 0 hoặc 1. Sau đó ta cộng hai bit tiếp theo và số nhớ

$$a_1 + b_1 + c_0 = c_1.2 + s_1.$$

Ở đây s_1 là bit tiếp theo (tính từ bên phải) trong khai triển nhị phân của a+b và c_1 là số nhớ. Tiếp tục quá trình này bằng cách cộng các bit tương ứng trong hai khai triển nhị phân và số nhớ để xác định bit tiếp sau tính từ bên phải trong khai triển nhị phân của tổng a+b. Ở giai đoạn cuối cùng, cộng a_{n-1}, b_{n-1} và c_{n-2} để nhận được $c_{n-1}.2+s_{n-1}$. Bit đứng đầu của tổng là $s_n=c_{n-1}$. Kết quả, thủ tục này tạo ra được khai triển nhị phân của tổng, cụ thể là $a+b = (s_n\, s_{n-1}\, s_{n-2} \dots s_1\, s_0)_2$.

Thí dụ 8: Tìm tổng của $a = (11011)_2$ và $b = (10110)_2$.

$a_0 + b_0 = 1 + 0 = 0.2 + 1$ ($c_0 = 0$, $s_0 = 1$), $a_1 + b_1 + c_0 = 1 + 1 + 0 = 1.2 + 0$ ($c_1 = 1$, $s_1 = 0$), $a_2 + b_2 + c_1 = 0 + 1 + 1 = 1.2 + 0$ ($c_2 = 1$, $s_2 = 0$), $a_3 + b_3 + c_2 = 1 + 0 + 1 = 1.2 + 0$ ($c_3 = 1$, $s_3 = 0$), $a_4 + b_4 + c_3 = 1 + 1 + 1 = 1.2 + 1$ ($s_5 = c_4 = 1$, $s_4 = 1$).

Do đó, $a + b = (110001)_2$.

Thuật toán cộng có thể được mô tả bằng cách dùng đoạn giả mã như sau.

---

procedure cộng (a,b: positive integers)

c := 0

for j := 0 **to** n-1

begin

$$d := \left\lfloor \frac{a_j + b_j + c}{2} \right\rfloor$$

$s_j := a_j + b_j + c - 2d$

c := d

end

s_n := c

{khai triển nhị phân của tổng là $(s_n\, s_{n-1} \dots s_1\, s_0)_2$}

Tổng hai số nguyên được tính bằng cách cộng liên tiếp các cặp bit và khi cần phải cộng cả số nhớ nữa. Cộng một cặp bit và số nhớ đòi ba hoặc ít hơn phép cộng các bit. Như vậy, tổng số các phép cộng bit được sử dụng nhỏ hơn ba lần số bit trong khai triển nhị phân. Do đó, độ phức tạp của thuật toán này là O(n).

2) Phép nhân: Xét bài toán nhân hai số nguyên viết ở dạng nhị phân. Thuật toán thông thường tiến hành như sau. Dùng luật phân phối, ta có:

$$ab = a\sum_{j=0}^{n-1} b_j 2^j = \sum_{j=0}^{n-1} a(b_j 2^j).$$

Ta có thể tính ab bằng cách dùng phương trình trên. Trước hết, ta thấy rằng ab_j=a nếu b_j=1 và ab_j=0 nếu b_j=0. Mỗi lần ta nhân một số hạng với 2 là ta dịch khai triển nhị phân của nó một chỗ về phía trái bằng cách thêm một số không vào cuối khai triển nhị phân của nó. Do đó, ta có thể nhận được $(ab_j)2^j$ bằng cách dịch khai triển nhị phân của ab_j đi j chỗ về phía trái, tức là thêm j số không vào cuối khai triển nhị phân của nó. Cuối cùng, ta sẽ nhận được tích ab bằng cách cộng n số nguyên $ab_j.2^j$ với j=0, 1, ..., n-1.

Thí dụ 9: Tìm tích của a = $(110)_2$ và b = $(101)_2$.

Ta có $ab_0.2^0 = (110)_2.1.2^0 = (110)_2$, $ab_1.2^1 = (110)_2.0.2^1 = (0000)_2$, $ab_2.2^2 = (110)_2.1.2^2 = (11000)_2$. Để tìm tích, hãy cộng $(110)_2$, $(0000)_2$ và $(11000)_2$. Từ đó ta có ab= $(11110)_2$.

Thủ tục trên được mô tả bằng đoạn giả mã sau:

---

procedure nhân (a,b: positive integers)
    for j := 0 **to** n-1
    begin
        if b_j = 1 **then** c_j := a được dịch đi j chỗ
        else c_j := 0
    end
{c_0, c_1,..., c_{n-1} là các tích riêng phần}
    p := 0
    for j := 0 **to** n-1
        p := p + c_j
{p là giá trị của tích ab}

Thuật toán trên tính tích của hai số nguyên a và b bằng cách cộng các tích riêng phần c_0, c_1, c_2, ..., c_{n-1}. Khi b_j=1, ta tính tích riêng phần c_j bằng cách dịch khai triển nhị phân của a đi j bit. Khi b_j=0 thì không cần có dịch chuyển nào vì c_j=0. Do đó, để tìm tất cả n số nguyên $ab_j.2^j$ với j=0, 1, ..., n-1, đòi hỏi tối đa là

$$0 + 1 + 2 + ... + n-1 = \frac{n(n-1)}{2}$$

phép dịch chỗ. Vì vậy, số các dịch chuyển chỗ đòi hỏi là $O(n^2)$.

Để cộng các số nguyên ab_j từ j=0 đến n–1, đòi hỏi phải cộng một số nguyên n bit, một số nguyên n+1 bit, ... và một số nguyên 2n bit. Ta đã biết rằng mỗi phép cộng đó đòi hỏi O(n) phép cộng bit. Do đó, độ phức tạp của thuật toán này là $O(n^2)$.

1.5. THUẬT TOÁN ĐỆ QUY.

1.5.1. Khái niệm đệ quy:

Đôi khi chúng ta có thể quy việc giải bài toán với tập các dữ liệu đầu vào xác định về việc giải cùng bài toán đó nhưng với các giá trị đầu vào nhỏ hơn. Chẳng hạn, bài toán tìm UCLN của hai số a, b với a > b có thể rút gọn về bài toán tìm ƯCLN của hai số nhỏ hơn, a mod b và b. Khi việc rút gọn như vậy thực hiện được thì lời giải bài toán ban đầu có thể tìm được bằng một dãy các phép rút gọn cho tới những trường hợp mà ta có thể dễ dàng nhận được lời giải của bài toán. Ta sẽ thấy rằng các thuật toán rút gọn liên tiếp bài toán ban đầu tới bài toán có dữ liệu đầu vào nhỏ hơn, được áp dụng trong một lớp rất rộng các bài toán.

Định nghĩa: Một thuật toán được gọi là đệ quy nếu nó giải bài toán bằng cách rút gọn liên tiếp bài toán ban đầu tới bài toán cũng như vậy nhưng có dữ liệu đầu vào nhỏ hơn.

Thí dụ 10: Tìm thuật toán đệ quy tính giá trị a^n với a là số thực khác không và n là số nguyên không âm.

Ta xây dựng thuật toán đệ quy nhờ định nghĩa đệ quy của a^n, đó là $a^{n+1}=a.a^n$ với n>0 và khi n=0 thì $a^0=1$. Vậy để tính a^n ta quy về các trường hợp có số mũ n nhỏ hơn, cho tới khi n=0.

---

procedure power (a: số thực khác không; n: số nguyên không âm)

if n = 0 **then** power(a,n) := 1

else power(a,n) := a * power(a,n-1)

Thí dụ 11: Tìm thuật toán đệ quy tính UCLN của hai số nguyên a,b không âm và a > b.

---

procedure UCLN (a,b: các số nguyên không âm, a > b)

if b = 0 **then** UCLN (a,b) := a

else UCLN (a,b) := UCLN (a **mod** b, b)

Thí dụ 12: Hãy biểu diễn thuật toán tìm kiếm tuyến tính như một thủ tục đệ quy.

Để tìm x trong dãy tìm kiếm $a_1,a_2,...,a_n$ trong bước thứ i của thuật toán ta so sánh x với a_i. Nếu x bằng a_i thì i là vị trí cần tìm, ngược lại thì việc tìm kiếm được quy về dãy có số phần tử ít hơn, cụ thể là dãy $a_{i+1},...,a_n$. Thuật toán tìm kiếm có dạng thủ tục đệ quy như sau.

Cho *search* (i,j,x) là thủ tục tìm số x trong dãy $a_i, a_{i+1},..., a_j$. Dữ liệu đầu vào là bộ ba (1,n,x). Thủ tục sẽ dừng khi số hạng đầu tiên của dãy còn lại là x hoặc là khi dãy còn lại chỉ có một phần tử khác x. Nếu x không là số hạng đầu tiên và còn có các số hạng khác thì lại áp dụng thủ tục này, nhưng dãy tìm kiếm ít hơn một phần tử nhận được bằng cách xóa đi phần tử đầu tiên của dãy tìm kiếm ở bước vừa qua.

procedure *search* (i,j,x)

if $a_i = x$ **then** loacation := i

else if i = j **then** loacation := 0

else *search* (i+1,j,x)

Thí dụ 13: Hãy xây dựng phiên bản đệ quy của thuật toán tìm kiếm nhị phân.

Giả sử ta muốn định vị x trong dãy a_1, a_2, ..., a_n bằng tìm kiếm nhị phân. Trước tiên ta so sánh x với số hạng giữa $a_{[(n+1)/2]}$. Nếu chúng bằng nhau thì thuật toán kết thúc, nếu không ta chuyển sang tìm kiếm trong dãy ngắn hơn, nửa đầu của dãy nếu x nhỏ hơn giá trị giữa của của dãy xuất phát, nửa sau nếu ngược lại. Như vậy ta rút gọn việc giải bài toán tìm kiếm về việc giải cũng bài toán đó nhưng trong dãy tìm kiếm có độ dài lần lượt giảm đi một nửa.

---

procedure *binary search* (x,i,j)

m := [(i+j)/2]

if $x = a_m$ **then** loacation := m

else if ($x < a_m$ and i < m) **then** *binary search* (x,i,m-1)

else if ($x > a_m$ and j > m) **then** *binary search* (x,m+1,j)

else loacation := 0

1.5.2. Đệ quy và lặp:

Thí dụ 14. Thủ tục đệ quy sau đây cho ta giá trị của n! với n là số nguyên dương.

---

procedure *factorial* (n: positive integer)

if n = 1 **then** *factorial(n)* := 1

else factorial(n) := n * factorial(n-1)

Có cách khác tính hàm giai thừa của một số nguyên từ định nghĩa đệ quy của nó. Thay cho việc lần lượt rút gọn việc tính toán cho các giá trị nhỏ hơn, ta có thể xuất phát từ giá trị của hàm tại 1và lần lượt áp dụng định nghĩa đệ quy để tìm giá trị của hàm tại các số nguyên lớn dần. Đó là thủ tục lặp.

---

procedure iterative factorial (n: positive integer)

x := 1

for i := 1 **to** n

x := i * x

{x là n!}

Thông thường để tính một dãy các giá trị được định nghĩa bằng đệ quy, nếu dùng phương pháp lặp thì số các phép tính sẽ ít hơn là dùng thuật toán đệ quy (trừ khi dùng các máy đệ quy chuyên dụng). Ta sẽ xem xét bài toán tính số hạng thứ n của dãy Fibonacci.

---

procedure *fibonacci* (n: nguyên không âm)

if n = 0 **the** *fibonacci*(n) := 0
else if n = 1 **then** *fibonacci*(n) := 1
else *fibonacci*(n) := *fibonacci*(n - 1) + *fibonacci*(n - 2)

Theo thuật toán này, để tìm f_n ta biểu diễn $f_n = f_{n-1} + f_{n-2}$. Sau đó thay thế cả hai số này bằng tổng của hai số Fibonacci bậc thấp hơn, cứ tiếp tục như vậy cho tới khi f_0 và f_1 xuất hiện thì được thay bằng các giá trị của chúng theo định nghĩa. Do đó để tính f_n cần f_{n+1}-1 phép cộng.

Bây giờ ta sẽ tính các phép toán cần dùng để tính f_n khi sử dụng phương pháp lặp. Thủ tục này khởi tạo x là $f_0 = 0$ và y là $f_1 = 1$. Khi vòng lặp được duyệt qua tổng của x và y được gán cho biến phụ z. Sau đó x được gán giá trị của y và y được gán giá trị của z. Vậy sau khi đi qua vòng lặp lần 1, ta có $x = f_1$ và $y = f_0 + f_1 = f_2$. Khi qua vòng lặp lần n-1 thì $x = f_{n-1}$. Như vậy chỉ có n – 1 phép cộng được dùng để tìm f_n khi n > 1.

---

procedure *Iterative fibonacci* (n: nguyên không âm)
if n = 0 **then** y := 0
else
begin
x := 0 ; y := 1
for i := 1 **to** n - 1
begin
z := x + y
x := y ; y := z
end
end
{y là số Fibonacci thứ n}

Ta đã chỉ ra rằng số các phép toán dùng trong thuật toán đệ quy nhiều hơn khi dùng phương pháp lặp. Tuy nhiên đôi khi người ta vẫn thích dùng thủ tục đệ quy hơn ngay cả khi nó tỏ ra kém hiệu quả so với thủ tục lặp. Đặc biệt, có những bài toán chỉ có thể giải bằng thủ tục đệ quy mà không thể giải bằng thủ tục lặp.

BÀI TẬP CHƯƠNG I:

1. Tìm một số nguyên n nhỏ nhất sao cho f(x) là $O(x^n)$ đối với các hàm f(x) sau:
a) $f(x) = 2x^3 + x^2 \log x$.
b) $f(x) = 2x^3 + (\log x)^4$.
c) $f(x) = \dfrac{x^4 + x^2 + 1}{x^3 + 1}$

d) $f(x) = \dfrac{x^5 + 5\log x}{x^4 + 1}$.

2. Chứng minh rằng
a) $x^2 + 4x + 7$ là $O(x^3)$, nhưng x^3 không là $O(x^2 + 4x + 17)$.
b) $x\log x$ là $O(x^2)$, nhưng x^2 không là $O(x\log x)$.

3. Cho một đánh giá big-O đối với các hàm cho dưới đây. Đối với hàm g(x) trong đánh giá f(x) là O(g(x)), hãy chọn hàm đơn giản có bậc thấp nhất.
a) $n\log(n^2 + 1) + n^2\log n$.
b) $(n\log n + 1)^2 + (\log n + 1)(n^2 + 1)$.
c) $n^{2^n} + n^{n^2}$.

4. Cho H_n là số điều hoà thứ n:

$$H_n = 1 + \frac{1}{2} + \frac{1}{3} + ... + \frac{1}{n}$$

Chứng minh rằng H_n là $O(\log n)$.

5. Lập một thuật toán tính tổng tất cả các số nguyên trong một bảng.

6. Lập thuật toán tính x_n với x là một số thực và n là một số nguyên.

7. Mô tả thuật toán chèn một số nguyên x vào vị trí thích hợp trong dãy các số nguyên $a_1, a_2, ..., a_n$ xếp theo thứ tự tăng dần.

8. Tìm thuật toán xác định vị trí gặp đầu tiên của phần tử lớn nhất trong bảng liệt kê các số nguyên, trong đó các số này không nhất thiết phải khác nhau.

9. Tìm thuật toán xác định vị trí gặp cuối cùng của phần tử nhỏ nhất trong bảng liệt kê các số nguyên, trong đó các số này không nhất thiết phải khác nhau.

10. Mô tả thuật toán đếm số các số 1 trong một xâu bit bằng cách kiểm tra mỗi bit của xâu để xác định nó có là bit 1 hay không.

11. *Thuật toán tìm kiếm tam phân*. Xác định vị trí của một phần tử trong một bảng liệt kê các số nguyên theo thứ tự tăng dần bằng cách tách liên tiếp bảng liệt kê đó thành ba bảng liệt kê con có kích thước bằng nhau (hoặc gần bằng nhau nhất có thể được) và giới hạn việc tìm kiếm trong một bảng liệt kê con thích hợp. Hãy chỉ rõ các bước của thuật toán đó.

12. Lập thuật toán tìm trong một dãy các số nguyên số hạng đầu tiên bằng một số hạng nào đó đứng trước nó trong dãy.

13. Lập thuật toán tìm trong một dãy các số nguyên tất cả các số hạng lớn hơn tổng tất cả các số hạng đứng trước nó trong dãy.

14. Cho đánh giá big-O đối với số các phép so sánh được dùng bởi thuật toán trong Bài tập 10.

15. Đánh giá độ phức tạp của thuật toán tìm kiếm tam phân được cho trong Bài tập 11.

16. Đánh giá độ phức tạp của thuật toán trong Bài tập 12.

17. Mô tả thuật toán tính hiệu của hai khai triển nhị phân.

18. Lập một thuật toán để xác định a > b, a = b hay a < b đối với hai số nguyên a và b ở dạng khai triển nhị phân.

19. Đánh giá độ phức tạp của thuật toán tìm khai triển theo cơ số b của số nguyên n qua số các phép chia được dùng.

20. Hãy cho thuật toán đệ quy tìm tổng n số nguyên dương lẻ đầu tiên.

21. Hãy cho thuật toán đệ quy tìm số cực đại của tập hữu hạn các số nguyên.

22. Mô tả thuật toán đệ quy tìm x^n **mod** m với n, x, m là các số nguyên dương.

23. Hãy nghĩ ra thuật toán đệ quy tính a^{2^n} trong đó a là một số thực và n là một số nguyên dương.

24. Hãy nghĩ ra thuật toán đệ quy tìm số hạng thứ n của dãy được xác định như sau: $a_0=1$, $a_1 = 2$ và $a_n = a_{n-1}\, a_{n-2}$ với n = 2, 3, 4, ...

25. Thuật toán đệ quy hay thuật toán lặp tìm số hạng thứ n của dãy trong Bài tập 24 là có hiệu quả hơn?

CHƯƠNG II
BÀI TOÁN ĐẾM

Lý thuyết tổ hợp là một phần quan trọng của toán học rời rạc chuyên nghiên cứu sự phân bố các phần tử vào các tập hợp. Thông thường các phần tử này là hữu hạn và việc phân bố chúng phải thoả mãn những điều kiện nhất định nào đó, tùy theo yêu cầu của bài toán cần nghiên cứu. Mỗi cách phân bố như vậy gọi là một cấu hình tổ hợp. Chủ đề này đã được nghiên cứu từ thế kỹ 17, khi những câu hỏi về tổ hợp được nêu ra trong những công trình nghiên cứu các trò chơi may rủi. Liệt kê, đếm các đối tượng có những tính chất nào đó là một phần quan trọng của lý thuyết tổ hợp. Chúng ta cần phải đếm các đối tượng để giải nhiều bài toán khác nhau. Hơn nữa các kỹ thuật đếm được dùng rất nhiều khi tính xác suất của các biến cố.

2.1. CƠ SỞ CỦA PHÉP ĐẾM.

2.1.1. Những nguyên lý đếm cơ bản:

1) Quy tắc cộng: Giả sử có k công việc T_1, T_2, ..., T_k. Các việc này có thể làm tương ứng bằng n_1, n_2, ..., n_k cách và giả sử không có hai việc nào có thể làm đồng thời. Khi đó số cách làm một trong k việc đó là $n_1+n_2+ ... + n_k$.

Thí dụ 1: 1) Một sinh viên có thể chọn bài thực hành máy tính từ một trong ba danh sách tương ứng có 23, 15 và 19 bài. Vì vậy, theo quy tắc cộng có 23 + 15 + 19 = 57 cách chọn bài thực hành.

2) Giá trị của biến m bằng bao nhiêu sau khi đoạn chương trình sau được thực hiện?

```
m := 0
for i1 := 1 to n1
        m := m+1
for i2 :=1 to n2
        m := m+1
.....
for ik := 1 to nk
        m := m+1
```

Giá trị khởi tạo của m bằng 0. Khối lệnh này gồm k vòng lặp khác nhau. Sau mỗi bước lặp của từng vòng lặp giá trị của k được tăng lên một đơn vị. Gọi T_i là việc thi hành vòng lặp thứ i. Có thể làm T_i bằng n_i cách vì vòng lặp thứ i có n_i bước lặp. Do các vòng lặp không thể thực hiện đồng thời nên theo quy tắc cộng, giá trị cuối cùng của m bằng số cách thực hiện một trong số các nhiệm vụ T_i, tức là $m = n_1+n_2+ ... + n_k$.

Quy tắc cộng có thể phát biểu dưới dạng của ngôn ngữ tập hợp như sau: Nếu A_1, A_2, ..., A_k là các tập hợp đôi một rời nhau, khi đó số phần tử của hợp các tập hợp này bằng tổng số các phần tử của các tập thành phần. Giả sử T_i là việc chọn một phần tử từ

tập A_i với i=1,2, ..., k. Có $|A_i|$ cách làm T_i và không có hai việc nào có thể được làm cùng một lúc. Số cách chọn một phần tử của hợp các tập hợp này, một mặt bằng số phần tử của nó, mặt khác theo quy tắc cộng nó bằng $|A_1|+|A_2|+ ... +|A_k|$. Do đó ta có:

$$|A_1 \cup A_2 \cup ... \cup A_k| = |A_1| + |A_2| + ... + |A_k|.$$

2) Quy tắc nhân: Giả sử một nhiệm vụ nào đó được tách ra thành k việc T_1, T_2, ..., T_k. Nếu việc T_i có thể làm bằng n_i cách sau khi các việc T_1, T_2, ... T_{i-1} đã được làm, khi đó có $n_1.n_2....n_k$ cách thi hành nhiệm vụ đã cho.

Thí dụ 2: 1) Người ta có thể ghi nhãn cho những chiếc ghế trong một giảng đường bằng một chữ cái và một số nguyên dương không vượt quá 100. Bằng cách như vậy, nhiều nhất có bao nhiêu chiếc ghế có thể được ghi nhãn khác nhau?

Thủ tục ghi nhãn cho một chiếc ghế gồm hai việc, gán một trong 26 chữ cái và sau đó gán một trong 100 số nguyên dương. Quy tắc nhân chỉ ra rằng có 26.100=2600 cách khác nhau để gán nhãn cho một chiếc ghế. Như vậy nhiều nhất ta có thể gán nhãn cho 2600 chiếc ghế.

2) Có bao nhiêu xâu nhị phân có độ dài n.

Mỗi một trong n bit của xâu nhị phân có thể chọn bằng hai cách vì mỗi bit hoặc bằng 0 hoặc bằng 1. Bởi vậy theo quy tắc nhân có tổng cộng 2^n xâu nhị phân khác nhau có độ dài bằng n.

3) Có thể tạo được bao nhiêu ánh xạ từ tập A có m phần tử vào tập B có n phần tử?

Theo định nghĩa, một ánh xạ xác định trên A có giá trị trên B là một phép tương ứng mỗi phần tử của A với một phần tử nào đó của B. Rõ ràng sau khi đã chọn được ảnh của i - 1 phần tử đầu, để chọn ảnh của phần tử thứ i của A ta có n cách. Vì vậy theo quy tắc nhân, ta có $n.n...n=n^m$ ánh xạ xác định trên A nhận giá trị trên B.

4) Có bao nhiêu đơn ánh xác định trên tập A có m phần tử và nhận giá trị trên tập B có n phần tử?

Nếu m > n thì với mọi ánh xạ, ít nhất có hai phần tử của A có cùng một ảnh, điều đó có nghĩa là không có đơn ánh từ A đến B. Bây giờ giả sử $m \le n$ và gọi các phần tử của A là $a_1,a_2,...,a_m$. Rõ ràng có n cách chọn ảnh cho phần tử a_1. Vì ánh xạ là đơn ánh nên ảnh của phần tử a_2 phải khác ảnh của a_1 nên chỉ có n - 1 cách chọn ảnh cho phần tử a_2. Nói chung, để chọn ảnh của a_k ta có n - k + 1 cách. Theo quy tắc nhân, ta có

$$n(n-1)(n-2)...(n-m+1) = \frac{n!}{(n-m)!}$$

đơn ánh từ tập A đến tập B.

5) Giá trị của biến k bằng bao nhiêu sau khi chương trình sau được thực hiện?

```
m := 0
for i1 := 1 to n1
        for i2 := 1 to n2
```

.....

for $i_k := 1$ **to** n_k

$k := k+1$

Giá trị khởi tạo của k bằng 0. Ta có k vòng lặp được lồng nhau. Gọi T_i là việc thi hành vòng lặp thứ i. Khi đó số lần đi qua vòng lặp bằng số cách làm các việc T_1, T_2, ..., T_k. Số cách thực hiện việc T_j là n_j (j=1, 2,..., k), vì vòng lặp thứ j được duyệt với mỗi giá trị nguyên i_j nằm giữa 1 và n_j. Theo quy tắc nhân vòng lặp lồng nhau này được duyệt qua $n_1.n_2....n_k$ lần. Vì vậy giá trị cuối cùng của k là $n_1.n_2....n_k$.

Nguyên lý nhân thường được phát biểu bằng ngôn ngữ tập hợp như sau. Nếu A_1, A_2,..., A_k là các tập hữu hạn, khi đó số phần tử của tích Descartes của các tập này bằng tích của số các phần tử của mọi tập thành phần. Ta biết rằng việc chọn một phần tử của tích Descartes A_1 x A_2 x...x A_k được tiến hành bằng cách chọn lần lượt một phần tử của A_1, một phần tử của A_2, ..., một phần tử của A_k. Theo quy tắc nhân ta có:

$$|A_1 \text{ x } A_2 \text{ x } ... \text{ x } A_k| = |A_1|.|A_2|...|A_k|.$$

2.1.2. Nguyên lý bù trừ:

Khi hai công việc có thể được làm đồng thời, ta không thể dùng quy tắc cộng để tính số cách thực hiện nhiệm vụ gồm cả hai việc. Để tính đúng số cách thực hiện nhiệm vụ này ta cộng số cách làm mỗi một trong hai việc rồi trừ đi số cách làm đồng thời cả hai việc. Ta có thể phát biểu nguyên lý đếm này bằng ngôn ngữ tập hợp. Cho A_1, A_2 là hai tập hữu hạn, khi đó

$$|A_1 \cup A_2| = |A_1| + |A_2| - |A_1 \cap A_2|.$$

Từ đó với ba tập hợp hữu hạn A_1, A_2, A_3, ta có:

$|A_1 \cup A_2 \cup A_3| = |A_1| + |A_2| + |A_3| - |A_1 \cap A_2| - |A_2 \cap A_3| - |A_3 \cap A_1| + |A_1 \cap A_2 \cap A_3|$,

và bằng quy nạp, với k tập hữu hạn A_1, A_2, ..., A_k ta có:

$$| A_1 \cup A_2 \cup ... \cup A_k| = N_1 - N_2 + N_3 - ... + (-1)^{k-1}N_k,$$

trong đó N_m ($1 \le m \le k$) là tổng phần tử của tất cả các giao m tập lấy từ k tập đã cho, nghĩa là

$$N_m = \sum_{1 \le i_1 < i_2 < ... < i_m \le k} | A_{i_1} \cap A_{i_2} \cap ... \cap A_{i_m} |$$

Bây giờ ta đồng nhất tập A_m ($1 \le m \le k$) với tính chất A_m cho trên tập vũ trụ hữu hạn U nào đó và đếm xem có bao nhiêu phần tử của U sao cho không thỏa mãn bất kỳ một tính chất A_m nào. Gọi $\overline{N}$ là số cần đếm, N là số phần tử của U. Ta có:

$$\overline{N} = N - | A_1 \cup A_2 \cup ... \cup A_k| = N - N_1 + N_2 - ... + (-1)^k N_k,$$

trong đó N_m là tổng các phần tử của U thỏa mãn m tính chất lấy từ k tính chất đã cho. Công thức này được gọi là **nguyên lý bù trừ**. Nó cho phép tính $\overline{N}$ qua các N_m trong trường hợp các số này dễ tính toán hơn.

Thí dụ 3: Có n lá thư và n phong bì ghi sẵn địa chỉ. Bỏ ngẫu nhiên các lá thư vào các phong bì. Hỏi xác suất để xảy ra không một lá thư nào đúng địa chỉ.

Mỗi phong bì có n cách bỏ thư vào, nên có tất cả n! cách bỏ thư. Vấn đề còn lại là đếm số cách bỏ thư sao cho không lá thư nào đúng địa chỉ. Gọi U là tập hợp các cách bỏ thư và A_m là tính chất lá thư thứ m bỏ đúng địa chỉ. Khi đó theo công thức về nguyên lý bù trừ ta có:

$$\overline{N} = n! - N_1 + N_2 - ... + (-1)^n N_n,$$

trong đó N_m ($1 \le m \le n$) là số tất cả các cách bỏ thư sao cho có m lá thư đúng địa chỉ. Nhận xét rằng, N_m là tổng theo mọi cách lấy m lá thư từ n lá, với mỗi cách lấy m lá thư, có (n-m)! cách bỏ để m lá thư này đúng địa chỉ, ta nhận được:

$$N_m = C_n^m (n - m)! = \frac{n!}{k!} \text{ và } \overline{N} = n!(1 - \frac{1}{1!} + \frac{1}{2!} - ... + (-1)^n \frac{1}{n!}),$$

trong đó $C_n^m = \frac{n!}{m!(n-m)!}$ là tổ hợp chập m của tập n phần tử (số cách chọn m đối tượng trong n đối tượng được cho). Từ đó xác suất cần tìm là: $1 - \frac{1}{1!} + \frac{1}{2!} - ... + (-1)^n \frac{1}{n!}$. Một điều lý thú là xác suất này dần đến e^{-1} (nghĩa là còn $> \frac{1}{3}$) khi n khá lớn.

Số $\overline{N}$ trong bài toán này được gọi là số mất thứ tự và được ký hiệu là D_n. Dưới đây là một vài giá trị của D_n, cho ta thấy D_n tăng nhanh như thế nào so với n:

n	2	3	4	5	6	7	8	9	10	11
D_n	1	2	9	44	265	1854	14833	133496	1334961	14684570

2.2. NGUYÊN LÝ DIRICHLET.

2.2.1. Mở đầu:

Giả sử có một đàn chim bồ câu bay vào chuồng. Nếu số chim nhiều hơn số ngăn chuồng thì ít nhất trong một ngăn có nhiều hơn một con chim. Nguyên lý này dĩ nhiên là có thể áp dụng cho các đối tượng không phải là chim bồ câu và chuồng chim.

Mệnh đề (Nguyên lý): Nếu có k+1 (hoặc nhiều hơn) đồ vật được đặt vào trong k hộp thì tồn tại một hộp có ít nhất hai đồ vật.

Chứng minh: Giả sử không có hộp nào trong k hộp chứa nhiều hơn một đồ vật. Khi đó tổng số vật được chứa trong các hộp nhiều nhất là bằng k. Điều này trái giả thiết là có ít nhất k + 1 vật.

Nguyên lý này thường được gọi là nguyên lý Dirichlet, mang tên nhà toán học người Đức ở thế kỷ 19. Ông thường xuyên sử dụng nguyên lý này trong công việc của mình.

Thí dụ 4: 1) Trong bất kỳ một nhóm 367 người thế nào cũng có ít nhất hai người có ngày sinh nhật giống nhau bởi vì chỉ có tất cả 366 ngày sinh nhật khác nhau.

2) Trong kỳ thi học sinh giỏi, điểm bài thi được đánh giá bởi một số nguyên trong khoảng từ 0 đến 100. Hỏi rằng ít nhất có bao nhiêu học sinh dự thi để cho chắc chắn tìm được hai học sinh có kết quả thi như nhau?

Theo nguyên lý Dirichlet, số học sinh cần tìm là 102, vì ta có 101 kết quả điểm thi khác nhau.

3) Trong số những người có mặt trên trái đất, phải tìm được hai người có hàm răng giống nhau. Nếu xem mỗi hàm răng gồm 32 cái như là một xâu nhị phân có chiều dài 32, trong đó răng còn ứng với bit 1 và răng mất ứng với bit 0, thì có tất cả 2^{32} = 4.294.967.296 hàm răng khác nhau. Trong khi đó số người trên hành tinh này là vượt quá 5 tỉ, nên theo nguyên lý Dirichlet ta có điều cần tìm.

2.2.2. Nguyên lý Dirichlet tổng quát:

Mệnh đề: Nếu có N đồ vật được đặt vào trong k hộp thì sẽ tồn tại một hộp chứa ít nhất ⌈N/k⌉ đồ vật.

(Ở đây, ⌈x⌉ là giá trị của hàm trần tại số thực x, đó là số nguyên nhỏ nhất có giá trị lớn hơn hoặc bằng x. Khái niệm này đối ngẫu với [x] – giá trị của hàm sàn hay hàm phần nguyên tại x – là số nguyên lớn nhất có giá trị nhỏ hơn hoặc bằng x.)

Chứng minh: Giả sử mọi hộp đều chứa ít hơn ⌈N/k⌉ vật. Khi đó tổng số đồ vật là

$$\le k\left(\left\lceil \frac{N}{k} \right\rceil - 1\right) < k\,\frac{N}{k} = N.$$

Điều này mâu thuẫn với giả thiết là có N đồ vật cần xếp.

Thí dụ 5: 1) Trong 100 người, có ít nhất 9 người sinh cùng một tháng.

Xếp những người sinh cùng tháng vào một nhóm. Có 12 tháng tất cả. Vậy theo nguyên lý Dirichlet, tồn tại một nhóm có ít nhất ⌈100/12⌉ = 9 người.

2) Có năm loại học bổng khác nhau. Hỏi rằng phải có ít nhất bao nhiêu sinh viên để chắc chắn rằng có ít ra là 6 người cùng nhận học bổng như nhau.

Gọi N là số sinh viên, khi đó ⌈N/5⌉ = 6 khi và chỉ khi $5 < N/5 \le 6$ hay $25 < N \le 30$. Vậy số N cần tìm là 26.

3) Số mã vùng cần thiết nhỏ nhất phải là bao nhiêu để đảm bảo 25 triệu máy điện thoại trong nước có số điện thoại khác nhau, mỗi số có 9 chữ số (giả sử số điện thoại có dạng 0XX - 8XXXXX với X nhận các giá trị từ 0 đến 9).

Có 10^7 = 10.000.000 số điện thoại khác nhau có dạng 0XX - 8XXXXX. Vì vậy theo nguyên lý Dirichlet tổng quát, trong số 25 triệu máy điện thoại ít nhất có ⌈25.000.000/10.000.000⌉ = 3 có cùng một số. Để đảm bảo mỗi máy có một số cần có ít nhất 3 mã vùng.

2.2.3. Một số ứng dụng của nguyên lý Dirichlet.

Trong nhiều ứng dụng thú vị của nguyên lý Dirichlet, khái niệm đồ vật và hộp cần phải được lựa chọn một cách khôn khéo. Trong phần nay có vài thí dụ như vậy.

Thí dụ 6: 1) Trong một phòng họp có n người, bao giờ cũng tìm được 2 người có số người quen trong số những người dự họp là như nhau.

Số người quen của mỗi người trong phòng họp nhận các giá trị từ 0 đến n – 1. Rõ ràng trong phòng không thể đồng thời có người có số người quen là 0 (tức là không quen ai) và có người có số người quen là n – 1 (tức là quen tất cả). Vì vậy theo số lượng người quen, ta chỉ có thể phân n người ra thành n –1 nhóm. Vậy theo nguyên lý Dirichlet tồn tai một nhóm có ít nhất 2 người, tức là luôn tìm được ít nhất 2 người có số người quen là như nhau.

2) Trong một tháng gồm 30 ngày, một đội bóng chuyền thi đấu mỗi ngày ít nhất 1 trận nhưng chơi không quá 45 trận. Chứng minh rằng tìm được một giai đoạn gồm một số ngày liên tục nào đó trong tháng sao cho trong giai đoạn đó đội chơi đúng 14 trận.

Gọi a_j là số trận mà đội đã chơi từ ngày đầu tháng đến hết ngày j. Khi đó

$$1 \le a_1 < a_2 < ... < a_{30} < 45$$

$$15 \le a_1+14 < a_2+14 < ... < a_{30}+14 < 59.$$

Sáu mươi số nguyên $a_1, a_2, ..., a_{30}, a_1+ 14, a_2 + 14, ..., a_{30}+14$ nằm giữa 1 và 59. Do đó theo nguyên lý Dirichlet có ít nhất 2 trong 60 số này bằng nhau. Vì vậy tồn tại i và j sao cho ai = aj + 14 (j < i). Điều này có nghĩa là từ ngày j + 1 đến hết ngày i đội đã chơi đúng 14 trận.

3) Chứng tỏ rằng trong n + 1 số nguyên dương không vượt quá 2n, tồn tại ít nhất một số chia hết cho số khác.

Ta viết mỗi số nguyên $a_1, a_2,..., a_{n+1}$ dưới dạng $a_j = 2^{k_j} q_j$ trong đó k_j là số nguyên không âm còn q_j là số dương lẻ nhỏ hơn 2n. Vì chỉ có n số nguyên dương lẻ nhỏ hơn 2n nên theo nguyên lý Dirichlet tồn tại i và j sao cho $q_i = q_j = q$. Khi đó $a_i= 2^{k_i} q$ và $aj = 2^{k_j} q$. Vì vậy, nếu $k_i \le k_j$ thì a_j chia hết cho a_i còn trong trường hợp ngược lại ta có a_i chia hết cho a_j.

Thí dụ cuối cùng trình bày cách áp dụng nguyên lý Dirichlet vào lý thuyết tổ hợp mà vẫn quen gọi là **lý thuyết Ramsey**, tên của nhà toán học người Anh. Nói chung, lý thuyết Ramsey giải quyết những bài toán phân chia các tập con của một tập các phần tử.

Thí dụ 7. Giả sử trong một nhóm 6 người mỗi cặp hai hoặc là bạn hoặc là thù. Chứng tỏ rằng trong nhóm có ba người là bạn lẫn nhau hoặc có ba người là kẻ thù lẫn nhau.

Gọi A là một trong 6 người. Trong số 5 người của nhóm hoặc là có ít nhất ba người là bạn của A hoặc có ít nhất ba người là kẻ thù của A, điều này suy ra từ nguyên lý Dirichlet tổng quát, vì $]5/2[= 3$. Trong trường hợp đầu ta gọi B, C, D là bạn của A. nếu trong ba người này có hai người là bạn thì họ cùng với A lập thành một bộ ba người bạn lẫn nhau, ngược lại, tức là nếu trong ba người B, C, D không có ai là bạn ai cả thì chứng tỏ họ là bộ ba người thù lẫn nhau. Tương tự có thể chứng minh trong trường hợp có ít nhất ba người là kẻ thù của A.

2.3. CHỈNH HỢP VÀ TỔ HỢP SUY RỘNG.

2.3.1. Chỉnh hợp có lặp.

Một cách sắp xếp có thứ tự k phần tử có thể lặp lại của một tập n phần tử được gọi là một chỉnh hợp lặp chập k từ tập n phần tử. Nếu A là tập gồm n phần tử đó thì mỗi chỉnh hợp như thế là một phần tử của tập A^k. Ngoài ra, mỗi chỉnh hợp lặp chập k từ tập n phần tử là một hàm từ tập k phần tử vào tập n phần tử. Vì vậy số chỉnh hợp lặp chập k từ tập n phần tử là n^k.

2.3.2. Tổ hợp lặp.

Một tổ hợp lặp chập k của một tập hợp là một cách chọn không có thứ tự k phần tử có thể lặp lại của tập đã cho. Như vậy một tổ hợp lặp kiểu này là một dãy không kể thứ tự gồm k thành phần lấy từ tập n phần tử. Do đó có thể là k > n.

Mệnh đề 1: Số tổ hợp lặp chập k từ tập n phần tử bằng C_{n+k-1}^{k}.

Chứng minh. Mỗi tổ hợp lặp chập k từ tập n phần tử có thể biểu diễn bằng một dãy n–1 thanh đứng và k ngôi sao. Ta dùng n – 1 thanh đứng để phân cách các ngăn. Ngăn thứ i chứa thêm một ngôi sao mỗi lần khi phần tử thứ i của tập xuất hiện trong tổ hợp. Chẳng hạn, tổ hợp lặp chập 6 của 4 phần tử được biểu thị bởi:

* * | * | | * * *

mô tả tổ hợp chứa đúng 2 phần tử thứ nhất, 1 phần tử thứ hai, không có phần tử thứ 3 và 3 phần tử thứ tư của tập hợp.

Mỗi dãy n – 1 thanh và k ngôi sao ứng với một xâu nhị phân độ dài n + k – 1 với k số 1. Do đó số các dãy n – 1 thanh đứng và k ngôi sao chính là số tổ hợp chập k từ tập n + k – 1 phần tử. Đó là điều cần chứng minh.

Thi dụ 8: 1) Có bao nhiêu cách chọn 5 tờ giấy bạc từ một két đựng tiền gồm những tờ 1000đ, 2000đ, 5000đ, 10.000đ, 20.000đ, 50.000đ, 100.000đ. Giả sử thứ tự mà các tờ tiền được chọn là không quan trọng, các tờ tiền cùng loại là không phân biệt và mỗi loại có ít nhất 5 tờ.

Vì ta không kể tới thứ tự chọn tờ tiền và vì ta chọn đúng 5 lần, mỗi lần lấy một từ 1 trong 7 loại tiền nên mỗi cách chọn 5 tờ giấy bạc này chính là một tổ hợp lặp chập 5 từ 7 phần tử. Do đó số cần tìm là $C_{7+5-1}^{5} = 462$.

2) Phương trình $x_1 + x_2 + x_3 = 15$ có bao nhiêu nghiệm nguyên không âm?

Chúng ta nhận thấy mỗi nghiệm của phương trình ứng với một cách chọn 15 phần tử từ một tập có 3 loại, sao cho có x_1 phần tử loại 1, x_2 phần tử loại 2 và x_3 phần tử loại 3 được chọn. Vì vậy số nghiệm bằng số tổ hợp lặp chập 15 từ tập có 3 phần tử và bằng $C_{3+15-1}^{15} = 136$.

2.3.3. Hoán vị của tập hợp có các phần tử giống nhau.

Trong bài toán đếm, một số phần tử có thể giống nhau. Khi đó cần phải cẩn thận, tránh đếm chúng hơn một lần. Ta xét thí dụ sau.

Thí dụ 9: Có thể nhận được bao nhiêu xâu khác nhau bằng cách sắp xếp lại các chữ cái của từ SUCCESS?

Vì một số chữ cái của từ SUCCESS là như nhau nên câu trả lời không phải là số hoán vị của 7 chữ cái được. Từ này chứa 3 chữ S, 2 chữ C, 1 chữ U và 1 chữ E. Để xác định số xâu khác nhau có thể tạo ra được ta nhận thấy có C(7,3) cách chọn 3 chỗ cho 3 chữ S, còn lại 4 chỗ trống. Có C(4,2) cách chọn 2 chỗ cho 2 chữ C, còn lại 2 chỗ trống. Có thể đặt chữ U bằng C(2,1) cách và C(1,1) cách đặt chữ E vào xâu. Theo nguyên lý nhân, số các xâu khác nhau có thể tạo được là:

$$C_7^3.C_4^2.C_2^1.C_1^1 = \frac{7!4!2!1!}{3!.4!.2!.2!.1!.1!.1!.0!} = \frac{7!}{3!.2!.1!.1!} = 420.$$

Mệnh đề 2: Số hoán vị của n phần tử trong đó có n_1 phần tử như nhau thuộc loại 1, n_2 phần tử như nhau thuộc loại 2, ..., và n_k phần tử như nhau thuộc loại k, bằng

$$\frac{n!}{n_1!.n_2!....n_k!}.$$

Chứng minh. Để xác định số hoán vị trước tiên chúng ta nhận thấy có $C_n^{n_1}$ cách giữ n_1 chỗ cho n_1 phần tử loại 1, còn lại n - n_1 chỗ trống. Sau đó có $C_{n-n_1}^{n_2}$ cách đặt n_2 phần tử loại 2 vào hoán vị, còn lại n - n_1 - n_2 chỗ trống. Tiếp tục đặt các phần tử loại 3, loại 4,..., loại k - 1vào chỗ trống trong hoán vị. Cuối cùng có $C_{n-n_1-...-n_{k-1}}^{n_k}$ cách đặt n_k phần tử loại k vào hoán vị. Theo quy tắc nhân tất cả các hoán vị có thể là:

$$C_n^{n_1}.C_{n-n_1}^{n_2}....C_{n-n_1-...-n_{k-1}}^{n_k} = \frac{n!}{n_1!.n_2!....n_k!}.$$

2.3.4. Sự phân bố các đồ vật vào trong hộp.

Thí dụ 10: Có bao nhiêu cách chia những xấp bài 5 quân cho mỗi một trong 4 người chơi từ một cỗ bài chuẩn 52 quân?

Người đầu tiên có thể nhận được 5 quân bài bằng C_{52}^5 cách. Người thứ hai có thể được chia 5 quân bài bằng C_{47}^5 cách, vì chỉ còn 47 quân bài. Người thứ ba có thể nhận được 5 quân bài bằng C_{42}^5 cách. Cuối cùng, người thứ tư nhận được 5 quân bài bằng C_{37}^5 cách. Vì vậy, theo nguyên lý nhân tổng cộng có

$$C_{52}^5.C_{47}^5.C_{42}^5.C_{37}^5 = \frac{52!}{5!.5!.5!.5!.32!}$$

cách chia cho 4 người mỗi người một xấp 5 quân bài.

Thí dụ trên là một bài toán điển hình về việc phân bố các đồ vật khác nhau vào các hộp khác nhau. Các đồ vật là 52 quân bài, còn 4 hộp là 4 người chơi và số còn lại để trên bàn. Số cách sắp xếp các đồ vật vào trong hộp được cho bởi mệnh đề sau

Mệnh đề 3: Số cách phân chia n đồ vật khác nhau vào trong k hộp khác nhau sao cho có n_i vật được đặt vào trong hộp thứ i, với i = 1, 2, ..., k bằng

$$\frac{n!}{n_1!.n_2!....n_k!.(n-n_1-...-n_k)!}.$$

2.4. SINH CÁC HOÁN VỊ VÀ TỔ HỢP.

2.4.1. Sinh các hoán vị:

Có nhiều thuật toán đã được phát triển để sinh ra n! hoán vị của tập {1,2,...,n}. Ta sẽ mô tả một trong các phương pháp đó, phương pháp liệt kê các hoán vị của tập {1,2,...,n} theo thứ tự từ điển. Khi đó, hoán vị $a_1a_2...a_n$ được gọi là đi trước hoán vị $b_1b_2...b_n$ nếu tồn tại k ($1 \le k \le n$), $a_1 = b_1$, $a_2 = b_2$,..., $a_{k-1} = b_{k-1}$ và $a_k < b_k$.

Thuật toán sinh các hoán vị của tập {1,2,...,n} dựa trên thủ tục xây dựng hoán vị kế tiếp, theo thứ tự từ điển, từ hoán vị cho trước $a_1\ a_2\ ...a_n$. Đầu tiên nếu $a_{n-1} < a_n$ thì rõ ràng đổi chỗ a_{n-1} và a_n cho nhau thì sẽ nhận được hoán vị mới đi liền sau hoán vị đã cho. Nếu tồn tại các số nguyên a_j và a_{j+1} sao cho $a_j < a_{j+1}$ và $a_{j+1} > a_{j+2} > ... > a_n$, tức là tìm cặp số nguyên liền kề đầu tiên tính từ bên phải sang bên trái của hoán vị mà số đầu nhỏ hơn số sau. Sau đó, để nhận được hoán vị liền sau ta đặt vào vị trí thứ j số nguyên nhỏ nhất trong các số lớn hơn a_j của tập $a_{j+1}, a_{j+2}, ..., a_n$, rồi liệt kê theo thứ tự tăng dần của các số còn lại của $a_j, a_{j+1}, a_{j+2}, ..., a_n$ vào các vị trí j+1, ..., n. Dễ thấy không có hoán vị nào đi sau hoán vị xuất phát và đi trước hoán vị vừa tạo ra.

Thí dụ 11: Tìm hoán vị liền sau theo thứ tự từ điển của hoán vị 4736521.

Cặp số nguyên đầu tiên tính từ phải qua trái có số trước nhỏ hơn số sau là $a_3 = 3$ và $a_4 = 6$. Số nhỏ nhất trong các số bên phải của số 3 mà lại lớn hơn 3 là số 5. Đặt số 5 vào vị trí thứ 3. Sau đó đặt các số 3, 6, 1, 2 theo thứ tự tăng dần vào bốn vị trí còn lại. Hoán vị liền sau hoán vị đã cho là 4751236.

---

procedure Hoán vị liền sau (a_1, a_2, ..., an) (hoán vị của {1,2,...,n} khác (n, n−1, ..., 2, 1))

```
    j := n – 1
    while a_j > a_{j+1}
        j := j – 1 {j là chỉ số lớn nhất mà a_j < a_{j+1}}
    k := n
    while a_j > a_k
        k := k - 1 {a_k là số nguyên nhỏ nhất trong các số lớn hơn a_j và bên phải a_j}
        đổi chỗ (a_j, a_k)
    r := n
    s := j + 1
    while r > s
        đổi chỗ (a_r, a_s)
        r := r - 1 ; s := s + 1
```

{Điều này sẽ xếp phần đuôi của hoán vị ở sau vị trí thứ j theo thứ tự tăng dần.}

2.4.2. Sinh các tổ hợp:

Làm thế nào để tạo ra tất cả các tổ hợp các phần tử của một tập hữu hạn? Vì tổ hợp chính là một tập con, nên ta có thể dùng phép tương ứng 1-1 giữa các tập con của $\{a_1,a_2,...,a_n\}$ và xâu nhị phân độ dài n.

Ta thấy một xâu nhị phân độ dài n cũng là khai triển nhị phân của một số nguyên nằm giữa 0 và $2^n - 1$. Khi đó 2^n xâu nhị phân có thể liệt kê theo thứ tự tăng dần của số nguyên trong biểu diễn nhị phân của chúng. Chúng ta sẽ bắt đầu từ xâu nhị phân nhỏ nhất 00...00 (n số 0). Mỗi bước để tìm xâu liền sau ta tìm vị trí đầu tiên tính từ phải qua trái mà ở đó là số 0, sau đó thay tất cả số 1 ở bên phải số này bằng 0 và đặt số 1 vào chính vị trí này.

---

procedure Xâu nhị phân liền sau ($b_{n-1}b_{n-2}...b_1b_0$): xâu nhị phân khác (11...11)

```
i := 0
while b_i = 1
begin
        b_i := 0
        i := i + 1
end
b_i := 1
```

Tiếp theo chúng ta sẽ trình bày thuật toán tạo các tổ hợp chập k từ n phần tử {1,2,...,n}. Mỗi tổ hợp chập k có thể biểu diễn bằng một xâu tăng. Khi đó có thể liệt kê các tổ hợp theo thứ tự từ điển. Có thể xây dựng tổ hợp liền sau tổ hợp $a_1a_2...a_k$ bằng cách sau. Trước hết, tìm phần tử đầu tiên a_i trong dãy đã cho kể từ phải qua trái sao cho $a_i \neq n - k + i$. Sau đó thay a_i bằng $a_i + 1$ và a_j bằng $a_i + j - i + 1$ với $j = i + 1, i + 2, ..., k$.

Thí dụ 12: Tìm tổ hợp chập 4 từ tập {1, 2, 3, 4, 5, 6} đi liền sau tổ hợp {1, 2, 5, 6}.

Ta thấy từ phải qua trái $a_2 = 2$ là số hạng đầu tiên của tổ hợp đã cho thỏa mãn điều kiện $a_i \neq 6 - 4 + i$. Để nhận được tổ hợp tiếp sau ta tăng a_i lên một đơn vị, tức $a_2 = 3$, sau đó đặt $a_3 = 3 + 1 = 4$ và $a_4 = 3 + 2 = 5$. Vậy tổ hợp liền sau tổ hợp đã cho là {1,3,4,5}. Thủ tục này được cho dưới dạng thuật toán như sau.

---

procedure Tổ hợp liền sau ($\{a_1, a_2, ..., a_k\}$: tập con thực sự của tập {1, 2, ..., n} không bằng $\{n - k + 1, ..., n\}$ với $a_1 < a_2 < ... < a_k$)

```
i := k
while a_i = n – k + i
        i := i – 1
        a_i := a_i + 1
for j := i + 1 to k
        a_j := a_i + j – i
```

2.5. HỆ THỨC TRUY HỒI.

2.5.1. Khái niệm mở đầu và mô hình hóa bằng hệ thức truy hồi:

Đôi khi ta rất khó định nghĩa một đối tượng một cách tường minh. Nhưng có thể dễ dàng định nghĩa đối tượng này qua chính nó. Kỹ thuật này được gọi là đệ quy. Định nghĩa đệ quy của một dãy số định rõ giá trị của một hay nhiều hơn các số hạng đầu tiên và quy tắc xác định các số hạng tiếp theo từ các số hạng đi trước. Định nghĩa đệ quy có thể dùng để giải các bài toán đếm. Khi đó quy tắc tìm các số hạng từ các số hạng đi trước được gọi là các hệ thức truy hồi.

Định nghĩa 1: Hệ thức truy hồi (hay công thức truy hồi) đối với dãy số $\{a_n\}$ là công thức biểu diễn a_n qua một hay nhiều số hạng đi trước của dãy. Dãy số được gọi là lời giải hay nghiệm của hệ thức truy hồi nếu các số hạng của nó thỏa mãn hệ thức truy hồi này.

Thí dụ 13 (Lãi kép)**: 1)** Giả sử một người gửi 10.000 đô la vào tài khoản của mình tại một ngân hàng với lãi suất kép 11% mỗi năm. Sau 30 năm anh ta có bao nhiêu tiền trong tài khoản của mình?

Gọi P_n là tổng số tiền có trong tài khoản sau n năm. Vì số tiền có trong tài khoản sau n năm bằng số có sau n – 1 năm cộng lãi suất của năm thứ n, nên ta thấy dãy $\{P_n\}$ thoả mãn hệ thức truy hồi sau:

$$P_n = P_{n-1} + 0{,}11P_{n-1} = (1{,}11)P_{n-1}$$

với điều kiện đầu $P_0 = 10.000$ đô la. Từ đó suy ra $P_n = (1{,}11)^n.10.000$. Thay n = 30 cho ta $P_{30} = 228922{,}97$ đô la.

2) Tìm hệ thức truy hồi và cho điều kiện đầu để tính số các xâu nhị phân độ dài n và không có hai số 0 liên tiếp. Có bao nhiêu xâu nhị phân như thế có độ dài bằng 5?

Gọi a_n là số các xâu nhị phân độ dài n và không có hai số 0 liên tiếp. Để nhận được hệ thức truy hồi cho $\{a_n\}$, ta thấy rằng theo quy tắc cộng, số các xâu nhị phân độ dài n và không có hai số 0 liên tiếp bằng số các xâu nhị phân như thế kết thúc bằng số 1 cộng với số các xâu như thế kết thúc bằng số 0. Giả sử $n \geq 3$.

Các xâu nhị phân độ dài n, không có hai số 0 liên tiếp kết thúc bằng số 1 chính là xâu nhị phân như thế, độ dài n – 1 và thêm số 1 vào cuối của chúng. Vậy chúng có tất cả là a_{n-1}. Các xâu nhị phân độ dài n, không có hai số 0 liên tiếp và kết thúc bằng số 0, cần phải có bit thứ n – 1 bằng 1, nếu không thì chúng có hai số 0 ở hai bit cuối cùng. Trong trường hợp này chúng có tất cả là a_{n-2}. Cuối cùng ta có được:

$$a_n = a_{n-1} + a_{n-2} \quad \text{với } n \geq 3.$$

Điều kiện đầu là $a_1 = 2$ và $a_2 = 3$. Khi đó $a_5 = a_4 + a_3 = a_3 + a_2 + a_3 = 2(a_2 + a_1) + a_2 = 13$.

2.5.2. Giải các hệ thức truy hồi.

Định nghĩa 2: Một hệ thức truy hồi tuyến tính thuần nhất bậc k với hệ số hằng số là hệ thức truy hồi có dạng:

$$a_n = c_1a_{n-1} + c_2a_{n-2} + \ ... + c_ka_{n-k} ,$$

trong đó $c_1, c_2, ..., c_k$ là các số thực và $c_k \neq 0$.

Theo nguyên lý của quy nạp toán học thì dãy số thỏa mãn hệ thức truy hồi nêu trong định nghĩa được xác định duy nhất bằng hệ thức truy hồi này và k điều kiện đầu: $a_0 = C_0, a_1 = C_1, ..., a_{k-1} = C_{k-1}$.

Phương pháp cơ bản để giải hệ thức truy hồi tuyến tính thuần nhất là tìm nghiệm dưới dạng $a_n = r^n$, trong đó r là hằng số. Chú ý rằng $a_n = r^n$ là nghiệm của hệ thức truy hồi $a_n = c_1a_{n-1} + c_2a_{n-2} + \ ... + c_ka_{n-k}$ nếu và chỉ nếu

$$r^n = c_1r^{n-1} + c_2r^{n-2} + \ ... + c_kr^{n-k} \text{ hay } r^k - c_1r^{k-1} - c_2r^{k-2} - ... \ - c_{k-1}r - c_k = 0.$$

Phương trình này được gọi là phương trình đặc trưng của hệ thức truy hồi, nghiệm của nó gọi là nghiệm đặc trưng của hệ thức truy hồi.

Mệnh đề: Cho $c_1, c_2, ..., c_k$ là các số thực. Giả sử rằng phương trình đặc trưng

$$r^k - c_1r^{k-1} - c_2r^{k-2} - ... \ - c_{k-1}r - c_k = 0$$

có k nghiệm phân biệt $r_1, r_2, ..., r_k$. Khi đó dãy $\{a_n\}$ là nghiệm của hệ thức truy hồi $a_n = c_1a_{n-1} + c_2a_{n-2} + ... + c_ka_{n-k}$ nếu và chỉ nếu $a_n = \alpha_1r_1^n + \alpha_2r_2^n + ... + \alpha_kr_k^n$, với n = 1, 2, ... trong đó $\alpha_1, \alpha_2, ..., \alpha_k$ là các hằng số.

Thí dụ 14: 1) Tìm công thức hiển của các số Fibonacci.

Dãy các số Fibonacci thỏa mãn hệ thức $f_n = f_{n-1} + f_{n-2}$ và các điều kiện đầu $f_0 = 0$ và $f_1 = 1$. Các nghiệm đặc trưng là $r_1 = \frac{1+\sqrt{5}}{2}$ và $r_2 = \frac{1-\sqrt{5}}{2}$. Do đó các số Fibonacci được cho bởi công thức $f_n = \alpha_1(\frac{1+\sqrt{5}}{2})^n + \alpha_2(\frac{1-\sqrt{5}}{2})^n$. Các điều kiện ban đầu $f_0 = 0 = \alpha_1 + \alpha_2$ và $f_1 = 1 = \alpha_1(\frac{1+\sqrt{5}}{2}) + \alpha_2(\frac{1-\sqrt{5}}{2})$. Từ hai phương trình này cho ta $\alpha_1 = \frac{1}{\sqrt{5}}$, $\alpha_2 = -\frac{1}{\sqrt{5}}$. Do đó các số Fibonacci được cho bởi công thức hiển sau:

$$f_n = \frac{1}{\sqrt{5}}(\frac{1+\sqrt{5}}{2})^n \ - \frac{1}{\sqrt{5}}(\frac{1-\sqrt{5}}{2})^n.$$

2) Hãy tìm nghiệm của hệ thức truy hồi $a_n = 6a_{n-1} - 11a_{n-2} + 6a_{n-3}$ với điều kiện ban đầu $a_0 = 2$, $a_1 = 5$ và $a_2 = 15$.

Đa thức đặc trưng của hệ thức truy hồi này là $r^3 - 6r^2 + 11r - 6$. Các nghiệm đặc trưng là r = 1, r = 2, r = 3. Do vậy nghiệm của hệ thức truy hồi có dạng

$$a_n = \alpha_1 1^n + \alpha_2 2^n + \alpha_3 3^n.$$

Các điều kiện ban đầu

$$a_0 = 2 = \alpha_1 + \alpha_2 + \alpha_3$$
$$a_1 = 5 = \alpha_1 + \alpha_2 2 + \alpha_3 3$$
$$a_2 = 15 = \alpha_1 + \alpha_2 4 + \alpha_3 9.$$

Giải hệ các phương trình này ta nhận được $\alpha_1 = 1$, $\alpha_2 = -1$, $\alpha_3 = 2$. Vì thế, nghiệm duy nhất của hệ thức truy hồi này và các điều kiện ban đầu đã cho là dãy $\{a_n\}$ với

$$a_n = 1 - 2^n + 2.3^n.$$

2.6. QUAN HỆ CHIA ĐỂ TRỊ.

2.6.1. Mở đầu:

Nhiều thuật toán đệ quy chia bài toán với các thông tin vào đã cho thành một hay nhiều bài toán nhỏ hơn. Sự phân chia này được áp dụng liên tiếp cho tới khi có thể tìm được lời giải của bài toán nhỏ một cách dễ dàng. Chẳng hạn, ta tiến hành việc tìm kiếm nhị phân bằng cách rút gọn việc tìm kiếm một phần tử trong một danh sách tới việc tìm phần tử đó trong một danh sách có độ dài giảm đi một nửa. Ta rút gọn liên tiếp như vậy cho tới khi còn lại một phần tử. Một ví dụ khác là thủ tục nhân các số nguyên. Thủ tục này rút gọn bài toán nhân hai số nguyên tới ba phép nhân hai số nguyên với số bit giảm đi một nửa. Phép rút gọn này được dùng liên tiếp cho tới khi nhận được các số nguyên có một bit. Các thủ tục này gọi là các thuật toán chia để trị.

2.6.2. Hệ thức chia để trị:

Giả sử rằng một thuật toán phân chia một bài toán cỡ n thành a bài toán nhỏ, trong đó mỗi bài toán nhỏ có cỡ $\frac{n}{b}$ (để đơn giản giả sử rằng n chia hết cho b; trong thực tế các bài toán nhỏ thường có cỡ $[\frac{n}{b}]$ hoặc $]\frac{n}{b}[$). Giả sử rằng tổng các phép toán thêm vào khi thực hiện phân chia bài toán cỡ n thành các bài toán có cỡ nhỏ hơn là g(n). Khi đó, nếu f(n) là số các phép toán cần thiết để giải bài toán đã cho thì f thỏa mãn hệ thức truy hồi sau:

$$f(n) = af(\frac{n}{b}) + g(n)$$

Hệ thức này có tên là hệ thức truy hồi chia để trị.

Thí dụ 15: 1) Thuật toán tìm kiếm nhị phân đưa bài toán tìm kiếm cỡ n về bài toán tìm kiếm phần tử này trong dãy tìm kiếm cỡ n/2, khi n chẵn. Khi thực hiện việc rút gọn cần hai phép so sánh. Vì thế, nếu f(n) là số phép so sánh cần phải làm khi tìm kiếm một phần tử trong danh sách tìm kiếm cỡ n ta có f(n) = f(n/2) + 2, nếu n là số chẵn.

2) Có các thuật toán hiệu quả hơn thuật toán thông thường để nhân hai số nguyên. Ở đây ta sẽ có một trong các thuật toán như vậy. Đó là thuật toán phân nhanh, có dùng kỹ thuật chia để trị. Trước tiên ta phân chia mỗi một trong hai số nguyên 2n bit thành hai khối mỗi khối n bit. Sau đó phép nhân hai số nguyên 2n bit ban đầu được thu về ba phép nhân các số nguyên n bit cộng với các phép dịch chuyển và các phép cộng.

Giả sử a và b là các số nguyên có các biểu diễn nhị phân độ dài 2n là

$$a = (a_{2n-1}\, a_{2n-2} \ldots a_1\, a_0)_2 \text{ và } b = (b_{2n-1}\, b_{2n-2} \ldots b_1\, b_0)_2.$$

Giả sử $a = 2^n A_1 + A_0$, $b = 2^n B_1 + B_0$, trong đó

$$A_1 = (a_{2n-1}\, a_{2n-2} \ldots a_{n+1}\, a_n)_2,\ A_0 = (a_{n-1} \ldots a_1\, a_0)_2$$

$$B_1 = (b_{2n-1}\, b_{2n-2} \ldots b_{n+1}\, b_n)_2,\ B_0 = (b_{n-1} \ldots b_1\, b_0)_2.$$

Thuật toán nhân nhanh các số nguyên dựa trên đẳng thức:

$$ab = (2^{2n} + 2^n)A_1B_1 + 2^n(A_1 - A_0)(B_0 - B_1) + (2^n + 1)A_0B_0.$$

Đẳng thức này chỉ ra rằng phép nhân hai số nguyên 2n bit có thể thực hiện bằng cách dùng ba phép nhân các số nguyên n bit và các phép cộng, trừ và phép dịch chuyển. Điều đó có nghĩa là nếu f(n) là tổng các phép toán nhị phân cần thiết để nhân hai số nguyên n bit thì

$$f(2n) = 3f(n) + Cn.$$

Ba phép nhân các số nguyên n bit cần 3f(n) phép toán nhị phân. Mỗi một trong các phép cộng, trừ hay dịch chuyển dùng một hằng số nhân với n lần các phép toán nhị phân và Cn là tổng các phép toán nhị phân được dùng khi làm các phép toán này.

Mệnh đề 1: Giả sử f là một hàm tăng thoả mãn hệ thức truy hồi $f(n) = af(\frac{n}{b}) + c$ với mọi n chia hết cho b, $a \geq 1$, b là số nguyên lớn hơn 1, còn c là số thực dương. Khi đó

$$f(n) = \begin{cases} O(n^{\log_b a}), a > 1 \\ O(\log n), a = 1 \end{cases}.$$

Mệnh đề 2: Giả sử f là hàm tăng thoả mãn hệ thức truy hồi $f(n) = af(\frac{n}{b}) + cn^d$ với mọi $n = b^k$, trong đó k là số nguyên dương, $a \geq 1$, b là số nguyên lớn hơn 1, còn c và d là các số thực dương. Khi đó

$$f(n) = \begin{cases} O(n^{\log_b a}), a > b^d \\ O(n^d \log n), a = b^d \\ O(n^d) \quad , a < b^d \end{cases}.$$

Thí dụ 16: Hãy ước lượng số phép toán nhị phân cần dùng khi nhân hai số nguyên n bit bằng thuật toán nhân nhanh.

Thí dụ 15.2 đã chỉ ra rằng f(n) = 3f(n/2) + Cn, khi n chẵn. Vì thế, từ Mệnh đề 2 ta suy ra $f(n) = O(n^{\log_2 3})$. Chú ý là $\log_2 3 \approx 1{,}6$. Vì thuật toán nhân thông thường dùng $O(n^2)$ phép toán nhị phân, thuật toán nhân nhanh sẽ thực sự tốt hơn thuật toán nhân thông thường khi các số nguyên là đủ lớn.

BÀI TẬP CHƯƠNG II:

1. Trong tổng số 2504 sinh viên của một khoa công nghệ thông tin, có 1876 theo học môn ngôn ngữ lập trình Pascal, 999 học môn ngôn ngữ Fortran và 345 học ngôn ngữ C. Ngoài ra còn biết 876 sinh viên học cả Pascal và Fortran, 232 học cả Fortran và C, 290 học cả Pascal và C. Nếu 189 sinh viên học cả 3 môn Pascal, Fortran và C thì trong trường hợp đó có bao nhiêu sinh viên không học môn nào trong 3 môn ngôn ngữ lập trình kể trên.

2. Một cuộc họp gồm 12 người tham dự để bàn về 3 vấn đề. Có 8 người phát biểu về vấn đề I, 5 người phát biểu về vấn đề II và 7 người phát biểu về vấn đề III. Ngoài ra, có đúng 1 người không phát biểu vấn đề nào. Hỏi nhiều lắm là có bao nhiêu người phát biểu cả 3 vấn đề.

3. Chỉ ra rằng có ít nhất 4 người trong số 25 triệu người có cùng tên họ viết tắt bằng 3 chữ cái sinh cùng ngày trong năm (không nhất thiết trong cùng một năm).

4. Một tay đô vật tham gia thi đấu giành chức vô địch trong 75 giờ. Mỗi giờ anh ta có ít nhất một trận đấu, nhưng toàn bộ anh ta có không quá 125 trận. Chứng tỏ rằng có những giờ liên tiếp anh ta đã đấu đúng 24 trận.

5. Cho n là số nguyên dương bất kỳ. Chứng minh rằng luôn lấy ra được từ n số đã cho một số số hạng thích hợp sao cho tổng của chúng chia hết cho n.

6. Trong một cuộc lấy ý kiến về 7 vấn đề, người được hỏi ghi vào một phiếu trả lời sẵn bằng cách để nguyên hoặc phủ định các câu trả lời tương ứng với 7 vấn đề đã nêu.

Chứng minh rằng với 1153 người được hỏi luôn tìm được 10 người trả lời giống hệt nhau.

7. Có 17 nhà bác học viết thư cho nhau trao đổi 3 vấn đề. Chứng minh rằng luôn tìm được 3 người cùng trao đổi một vấn đề.

8. Trong kỳ thi kết thúc học phần toán học rời rạc có 10 câu hỏi. Có bao nhiêu cách gán điểm cho các câu hỏi nếu tổng số điểm bằng 100 và mỗi câu ít nhất được 5 điểm.

9. Phương trình $x_1 + x_2 + x_3 + x_4 + x_5 = 21$ có bao nhiêu nghiệm nguyên không âm?

10. Có bao nhiêu xâu khác nhau có thể lập được từ các chữ cái trong từ *MISSISSIPI*, yêu cầu phải dùng tất cả các chữ?

11. Một giáo sư cất bộ sưu tập gồm 40 số báo toán học vào 4 chiếc ngăn tủ, mỗi ngăn đựng 10 số. Có bao nhiêu cách có thể cất các tờ báo vào các ngăn nếu:

1) Mỗi ngăn được đánh số sao cho có thể phân biệt được;

2) Các ngăn là giống hệt nhau?

12. Tìm hệ thức truy hồi cho số mất thứ tự D_n.

13. Tìm hệ thức truy hồi cho số các xâu nhị phân chứa xâu 01.

14. Tìm hệ thức truy hồi cho số cách đi lên n bậc thang nếu một người có thể bước một, hai hoặc ba bậc một lần.

15. 1) Tìm hệ thức truy hồi mà R_n thoả mãn, trong đó R_n là số miền của mặt phẳng bị phân chia bởi n đường thẳng nếu không có hai đường nào song song và không có 3 đường nào cùng đi qua một điểm.

b) Tính R_n bằng phương pháp lặp.

16. Tìm nghiệm của hệ thức truy hồi $a_n = 2a_{n-1} + 5a_{n-2} - 6a_{n-3}$ với $a_0 = 7$, $a_1 = -4$, $a_2 = 8$.

 www.VIETMATHS.com

CHƯƠNG III
ĐỒ THỊ

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

Đồ thị cũng được dùng để giải các bài toán trong nhiều lĩnh vực khác nhau. Thí dụ, dùng đồ thị để xác định xem có thực hiện một mạch điện trên một bảng điện phẳng được không. Chúng ta cũng có thể phân biệt hai hợp chất hóa học có cùng công thức phân tử nhưng có cấu trúc khác nhau nhờ đồ thị. Chúng ta cũng có thể xác định xem hai máy tính có được nối với nhau bằng một đường truyền thông hay không nếu dùng mô hình đồ thị mạng máy tính. Đồ thị với các trọng số được gán cho các cạnh của nó có thể dùng để giải các bài toán như bài toán tìm đường đi ngắn nhất giữa hai thành phố trong một mạng giao thông. Chúng ta cũng có thể dùng đồ thị để lập lịch thi và phân chia kênh cho các đài truyền hình.

3.1. ĐỊNH NGHĨA VÀ THÍ DỤ.

Đồ thị là một cấu trúc rời rạc gồm các đỉnh và các cạnh (vô hướng hoặc có hướng) nối các đỉnh đó. Người ta phân loại đồ thị tùy theo đặc tính và số các cạnh nối các cặp đỉnh của đồ thị. Nhiều bài toán thuộc những lĩnh vực rất khác nhau có thể giải được bằng mô hình đồ thị. Chẳng hạn người ta có thể dùng đồ thị để biểu diễn sự cạnh tranh các loài trong một môi trường sinh thái, dùng đồ thị để biểu diễn ai có ảnh hưởng lên ai trong một tổ chức nào đó, và cũng có thể dùng đồ thị để biểu diễn các kết cục của cuộc thi đấu thể thao. Chúng ta cũng có thể dùng đồ thị để giải các bài toán như bài toán tính số các tổ hợp khác nhau của các chuyến bay giữa hai thành phố trong một mạng hàng không, hay để giải bài toán đi tham quan tất cả các đường phố của một thành phố sao cho mỗi đường phố đi qua đúng một lần, hoặc bài toán tìm số các màu cần thiết để tô các vùng khác nhau của một bản đồ.

Trong đời sống, chúng ta thường gặp những sơ đồ, như sơ đồ tổ chức bộ máy, sơ đồ giao thông, sơ đồ hướng dẫn thứ tự đọc các chương trong một cuốn sách, ..., gồm những điểm biểu thị các đối tượng được xem xét (người, tổ chức, địa danh, chương mục sách, ...) và nối một số điểm với nhau bằng những đoạn thẳng (hoặc cong) hay những mũi tên, tượng trưng cho một quan hệ nào đó giữa các đối tượng. Đó là những thí dụ về đồ thị.

3.1.1. Định nghĩa: Một đơn đồ thị G = (V, E) gồm một tập khác rỗng V mà các phần tử của nó gọi là các đỉnh và một tập E mà các phần tử của nó gọi là các cạnh, đó là các cặp không có thứ tự của các đỉnh phân biệt.

3.1.2. Định nghĩa: Một đa đồ thị G = (V, E) gồm một tập khác rỗng V mà các phần tử của nó gọi là các đỉnh và một họ E mà các phần tử của nó gọi là các cạnh, đó là các cặp không có thứ tự của các đỉnh phân biệt. Hai cạnh được gọi là cạnh bội hay song song nếu chúng cùng tương ứng với một cặp đỉnh.

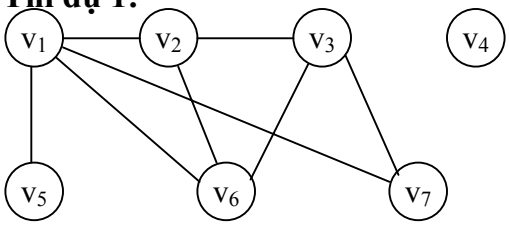
Rõ ràng mỗi đơn đồ thị là đa đồ thị, nhưng không phải đa đồ thị nào cũng là đơn đồ thị.

3.1.3. Định nghĩa: Một giả đồ thị G = (V, E) gồm một tập khác rỗng V mà các phần tử của nó gọi là các đỉnh và một họ E mà các phần tử của nó gọi là các cạnh, đó là các cặp không có thứ tự của các đỉnh (không nhất thiết là phân biệt).

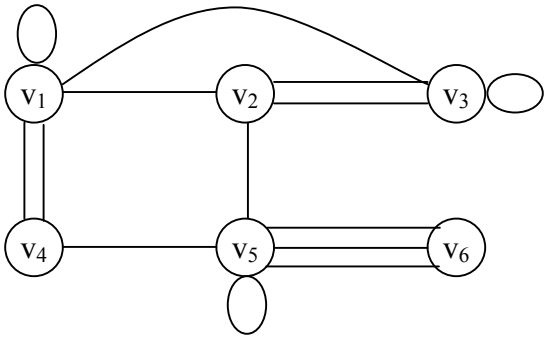
Với $v \in V$, nếu $(v,v) \in E$ thì ta nói có một khuyên tại đỉnh v.

Tóm lại, giả đồ thị là loại đồ thị vô hướng tổng quát nhất vì nó có thể chứa các khuyên và các cạnh bội. Đa đồ thị là loại đồ thị vô hướng có thể chứa cạnh bội nhưng không thể có các khuyên, còn đơn đồ thị là loại đồ thị vô hướng không chứa cạnh bội hoặc các khuyên.

Thí dụ 1:



Đơn đồ thị



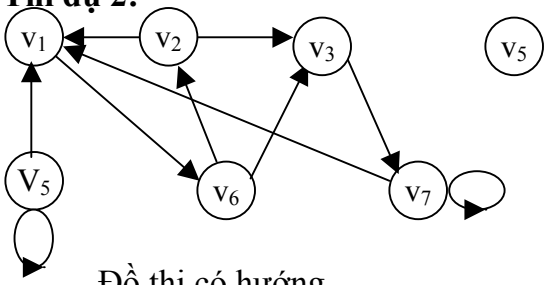
Giả đồ thị

3.1.4. Định nghĩa: Một đồ thị có hướng G = (V, E) gồm một tập khác rỗng V mà các phần tử của nó gọi là các đỉnh và một tập E mà các phần tử của nó gọi là các cung, đó là các cặp có thứ tự của các phần tử thuộc V.

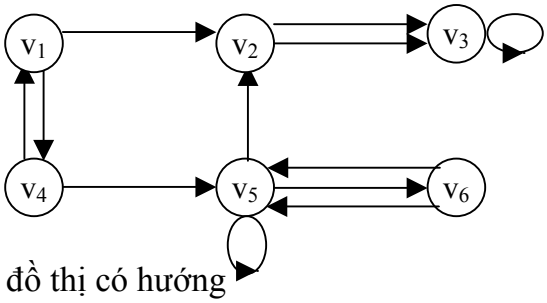
3.1.5. Định nghĩa: Một đa đồ thị có hướng G = (V, E) gồm một tập khác rỗng V mà các phần tử của nó gọi là các đỉnh và một họ E mà các phần tử của nó gọi là các cung, đó là các cặp có thứ tự của các phần tử thuộc V.

Đồ thị vô hướng nhận được từ đồ thị có hướng G bằng cách xoá bỏ các chiều mũi tên trên các cung được gọi là đồ thị vô hướng nền của G.

Thí dụ 2:



Đồ thị có hướng



Đa đồ thị có hướng

Thí dụ 3: 1) Đồ thị "lấn tổ" trong sinh thái học. Đồ thị được dùng trong nhiều mô hình có tính đến sự tương tác của các loài vật. Chẳng hạn sự cạnh tranh của các loài trong một hệ sinh thái có thể mô hình hóa bằng đồ thị "lấn tổ". Mỗi loài được biểu diễn bằng một đỉnh. Một cạnh vô hướng nối hai đỉnh nếu hai loài được biểu diễn bằng các đỉnh này là cạnh tranh với nhau.

2) Đồ thị ảnh hưởng. Khi nghiên cứu tính cách của một nhóm người, ta thấy một số người có thể có ảnh hưởng lên suy nghĩ của những người khác. Đồ thị có hướng được gọi là đồ thị ảnh hưởng có thể dùng để mô hình bài toán này. Mỗi người của nhóm được biểu diễn bằng một đỉnh. Khi một người được biểu diễn bằng đỉnh a có ảnh hưởng lên người được biểu diễn bằng đỉnh b thì có một cung nối từ đỉnh a đến đỉnh b.

3) Thi đấu vòng tròn. Một cuộc thi đấu thể thao trong đó mỗi đội đấu với mỗi đội khác đúng một lần gọi là đấu vòng tròn. Cuộc thi đấu như thế có thể được mô hình bằng một đồ thị có hướng trong đó mỗi đội là một đỉnh. Một cung đi từ đỉnh a đến đỉnh b nếu đội a thắng đội b.

4) Các chương trình máy tính có thể thi hành nhanh hơn bằng cách thi hành đồng thời một số câu lệnh nào đó. Điều quan trọng là không được thực hiện một câu lệnh đòi hỏi kết quả của câu lệnh khác chưa được thực hiện. Sự phụ thuộc của các câu lệnh vào các câu lệnh trước có thể biểu diễn bằng một đồ thị có hướng. Mỗi câu lệnh được biểu diễn bằng một đỉnh và có một cung từ một đỉnh tới một đỉnh khác nếu câu lệnh được biểu diễn bằng đỉnh thứ hai không thể thực hiện được trước khi câu lệnh được biểu diễn bằng đỉnh thứ nhất được thực hiện. Đồ thị này được gọi là **đồ thị có ưu tiên trước sau**.

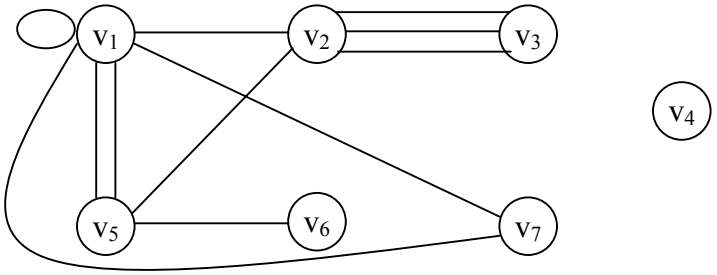
3.2. BẬC CỦA ĐỈNH.

3.2.1. Định nghĩa: Hai đỉnh u và v trong đồ thị (vô hướng) G=(V,E) được gọi là liền kề nếu (u,v)∈E. Nếu e = (u,v) thì e gọi là cạnh liên thuộc với các đỉnh u và v. Cạnh e cũng được gọi là cạnh nối các đỉnh u và v. Các đỉnh u và v gọi là các điểm đầu mút của cạnh e.

3.2.2. Định nghĩa: Bậc của đỉnh v trong đồ thị G=(V,E), ký hiệu deg(v), là số các cạnh liên thuộc với nó, riêng khuyên tại một đỉnh được tính hai lần cho bậc của nó.

Đỉnh v gọi là đỉnh treo nếu deg(v)=1 và gọi là đỉnh cô lập nếu deg(v)=0.

Thí dụ 4:



Ta có deg(v_1)=7, deg(v_2)=5, deg(v_3)=3, deg(v_4)=0, deg(v_5)=4, deg(v_6)=1, deg(v_7)=2. Đỉnh v_4 là đỉnh cô lập và đỉnh v_6 là đỉnh treo.

3.2.3. Mệnh đề: Cho đồ thị G = (V, E). Khi đó

$$2|E| = \sum_{v \in V} \deg(v).$$

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

3.2.4. Hệ quả: Số đỉnh bậc lẻ của một đồ thị là một số chẵn.

Chứng minh: Gọi V_1 và V_2 tương ứng là tập các đỉnh bậc chẵn và tập các đỉnh bậc lẻ của đồ thị G = (V, E). Khi đó

$$2|E| = \sum_{v \in V_1} \deg(v) + \sum_{v \in V_2} \deg(v)$$

Vế trái là một số chẵn và tổng thứ nhất cũng là một số chẵn nên tổng thứ hai là một số chẵn. Vì deg(v) là lẻ với mọi $v \in V_2$ nên $|V_2|$ là một số chẵn.

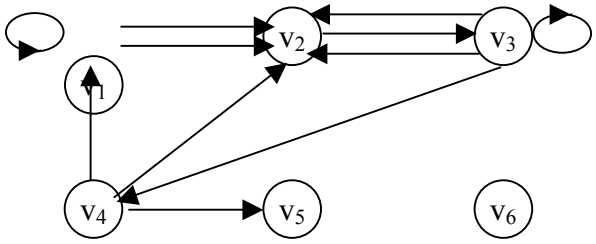
3.2.5. Mệnh đề: Trong một đơn đồ thị, luôn tồn tại hai đỉnh có cùng bậc.

Chứng minh: Xét đơn đồ thị G=(V,E) có |V|=n. Khi đó phát biểu trên được đưa về bài toán: trong một phòng họp có n người, bao giờ cũng tìm được 2 người có số người quen trong số những người dự họp là như nhau (xem Thí dụ 6 của 2.2.3).

3.2.6. Định nghĩa: Đỉnh u được gọi là nối tới v hay v được gọi là được nối từ u trong đồ thị có hướng G nếu (u,v) là một cung của G. Đỉnh u gọi là đỉnh đầu và đỉnh v gọi là đỉnh cuối của cung này.

3.2.7. Định nghĩa: Bậc vào (t.ư. bậc ra) của đỉnh v trong đồ thị có hướng G, ký hiệu $\deg_t(v)$ (t.ư. $\deg_o(v)$), là số các cung có đỉnh cuối là v.

Thí dụ 5:



$\deg_t(v_1) = 2$, $\deg_o(v_1) = 3$,
$\deg_t(v_2) = 5$, $\deg_o(v_2) = 1$,
$\deg_t(v_3) = 2$, $\deg_o(v_3) = 4$,
$\deg_t(v_4) = 1$, $\deg_0(v_4) = 3$,
$\deg_t(v_5) = 1$, $\deg_o(v_5) = 0$,
$\deg_t(v_6) = 0$, $\deg_o(v_6) = 0$.

Đỉnh có bậc vào và bậc ra cùng bằng 0 gọi là đỉnh cô lập. Đỉnh có bậc vào bằng 1 và bậc ra bằng 0 gọi là đỉnh treo, cung có đỉnh cuối là đỉnh treo gọi là cung treo.

3.2.8. Mệnh đề: Cho G =(V, E) là một đồ thị có hướng. Khi đó

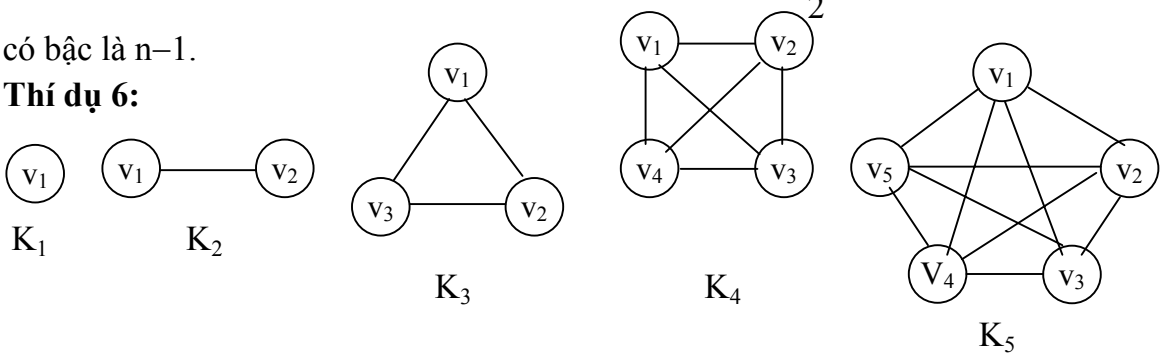
$$\sum_{v\in V}\deg_t(v) = \sum_{v\in V}\deg_o(v) = |E|.$$

Chứng minh: Kết quả có ngay là vì mỗi cung được tính một lần cho đỉnh đầu và một lần cho đỉnh cuối.

3.3. NHỮNG ĐƠN ĐỒ THỊ ĐẶC BIỆT.

3.3.1. Đồ thị đầy đủ: Đồ thị đầy đủ n đỉnh, ký hiệu là K_n, là đơn đồ thị mà hai đỉnh phân biệt bất kỳ của nó luôn liền kề. Như vậy, K_n có $\frac{n(n-1)}{2}$ cạnh và mỗi đỉnh của K_n có bậc là n–1.

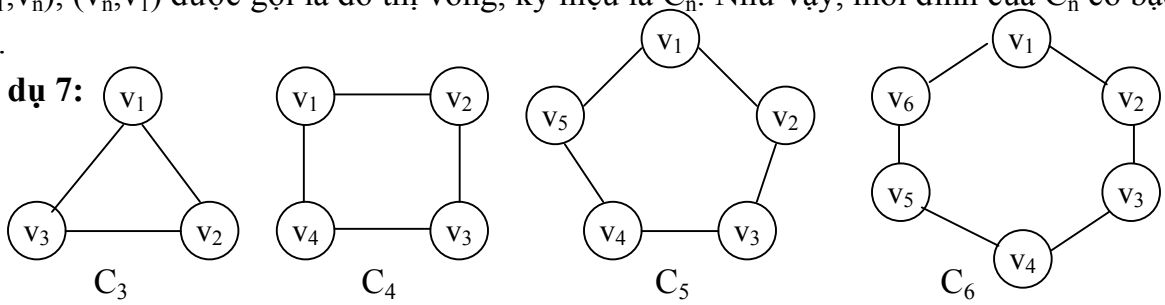
Thí dụ 6:



K_1 K_2 K_3 K_4 K_5

3.3.2. Đồ thị vòng: Đơn đồ thị n đỉnh $v_1, v_2, ..., v_n$ ($n\geq 3$) và n cạnh $(v_1,v_2), (v_2,v_3), ..., (v_{n-1},v_n), (v_n,v_1)$ được gọi là đồ thị vòng, ký hiệu là C_n. Như vậy, mỗi đỉnh của C_n có bậc là 2.

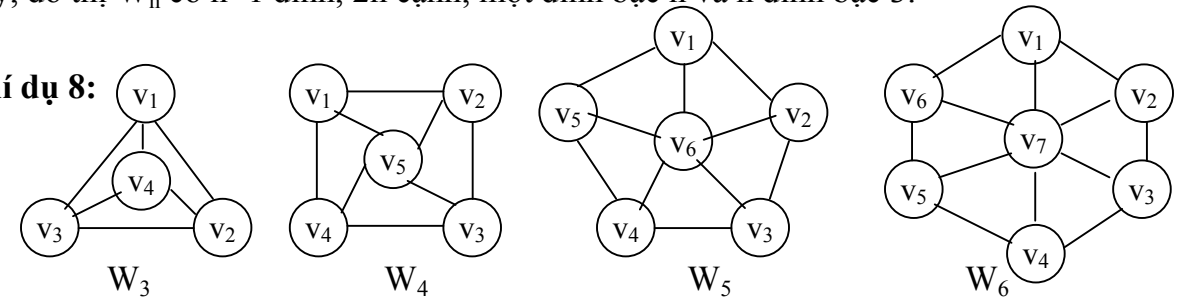
Thí dụ 7:



C_3 C_4 C_5 C_6

3.3.3. Đồ thị bánh xe: Từ đồ thị vòng C_n, thêm vào đỉnh v_{n+1} và các cạnh (v_{n+1},v_1), (v_{n+1},v_2), ..., (v_{n+1},v_n), ta nhận được đơn đồ thị gọi là đồ thị bánh xe, ký hiệu là W_n. Như vậy, đồ thị W_n có n+1 đỉnh, 2n cạnh, một đỉnh bậc n và n đỉnh bậc 3.

Thí dụ 8:

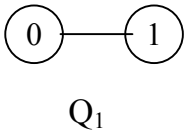


W_3 W_4 W_5 W_6

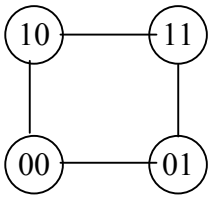
3.3.4. Đồ thị lập phương: Đơn đồ thị 2^n đỉnh, tương ứng với 2^n xâu nhị phân độ dài n và hai đỉnh kề nhau khi và chỉ khi 2 xâu nhị phân tương ứng với hai đỉnh này chỉ khác nhau đúng một bit được gọi là đồ thị lập phương, ký hiệu là Q_n. Như vậy, mỗi đỉnh của Q_n có bậc là n và số cạnh của Q_n là $n.2^{n-1}$ (từ công thức $2|E| = \sum_{v\in V}\deg(v)$).

www.VIETMATHS.com

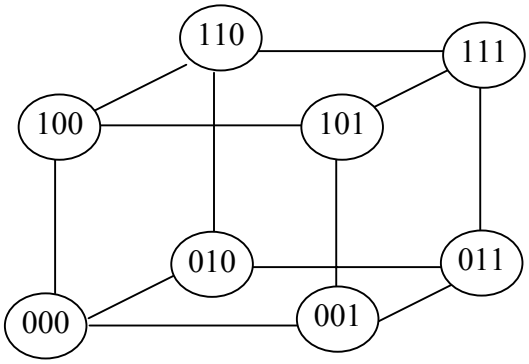
Thí dụ 9:



Q_1



Q_2

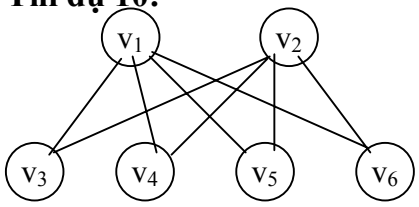


Q_3

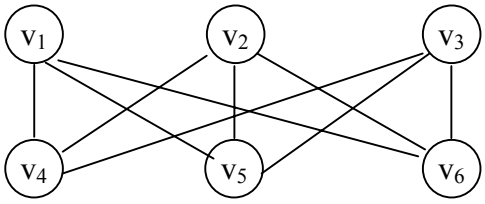
3.3.5. Đồ thị phân đôi (đồ thị hai phe): Đơn đồ thị G=(V,E) sao cho $V=V_1 \cup V_2$, $V_1 \cap V_2=\varnothing$, $V_1 \neq \varnothing$, $V_2 \neq \varnothing$ và mỗi cạnh của G được nối một đỉnh trong V_1 và một đỉnh trong V_2 được gọi là đồ thị phân đôi.

Nếu đồ thị phân đôi $G=(V_1 \cup V_2,E)$ sao cho với mọi $v_1 \in V_1$, $v_2 \in V_2$, $(v_1,v_2) \in E$ thì G được gọi là đồ thị phân đôi đầy đủ. Nếu $|V_1|=m$, $|V_2|=n$ thì đồ thị phân đôi đầy đủ G ký hiệu là $K_{m,n}$. Như vậy $K_{m,n}$ có m.n cạnh, các đỉnh của V_1 có bậc n và các đỉnh của V_2 có bậc m.

Thí dụ 10:



$K_{2,4}$

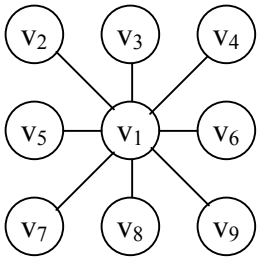


$K_{3,3}$

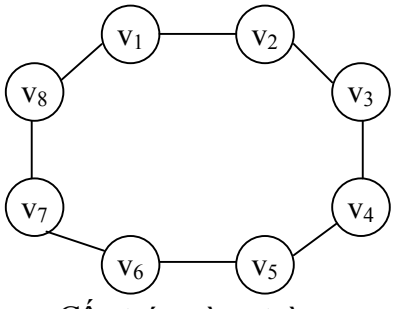
3.3.6. Một vài ứng dụng của các đồ thị đặc biệt:

1) Các mạng cục bộ (LAN): Một số mạng cục bộ dùng cấu trúc hình sao, trong đó tất cả các thiết bị được nối với thiết bị điều khiển trung tâm. Mạng cục bộ kiểu này có thể biểu diễn bằng một đồ thị phân đôi đầy đủ $K_{1,n}$. Các thông báo gửi từ thiết bị này tới thiết bị khác đều phải qua thiết bị điều khiển trung tâm.

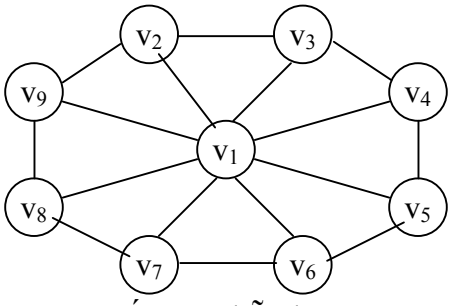
Mạng cục bộ cũng có thể có cấu trúc vòng tròn, trong đó mỗi thiết bị nối với đúng hai thiết bị khác. Mạng cục bộ kiểu này có thể biểu diễn bằng một đồ thị vòng C_n. Thông báo gửi từ thiết bị này tới thiết bị khác được truyền đi theo vòng tròn cho tới khi đến nơi nhận.



Cấu trúc hình sao



Cấu trúc vòng tròn



Cấu trúc hỗn hợp

www.VIETMATHS.com

Cuối cùng, một số mạng cục bộ dùng cấu trúc hỗn hợp của hai cấu trúc trên. Các thông báo được truyền vòng quanh theo vòng tròn hoặc có thể qua thiết bị trung tâm. Sự dư thừa này có thể làm cho mạng đáng tin cậy hơn. Mạng cục bộ kiểu này có thể biểu diễn bằng một đồ thị bánh xe W_n.

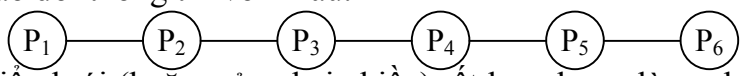
2) Xử lý song song: Các thuật toán để giải các bài toán được thiết kế để thực hiện một phép toán tại mỗi thời điểm là thuật toán nối tiếp. Tuy nhiên, nhiều bài toán với số lượng tính toán rất lớn như bài toán mô phỏng thời tiết, tạo hình trong y học hay phân tích mật mã không thể giải được trong một khoảng thời gian hợp lý nếu dùng thuật toán nối tiếp ngay cả khi dùng các siêu máy tính. Ngoài ra, do những giới hạn về mặt vật lý đối với tốc độ thực hiện các phép toán cơ sở, nên thường gặp các bài toán không thể giải trong khoảng thời gian hợp lý bằng các thao tác nối tiếp. Vì vậy, người ta phải nghĩ đến kiểu xử lý song song.

Khi xử lý song song, người ta dùng các máy tính có nhiều bộ xử lý riêng biệt, mỗi bộ xử lý có bộ nhớ riêng, nhờ đó có thể khắc phục được những hạn chế của các máy nối tiếp. Các thuật toán song song phân chia bài toán chính thành một số bài toán con sao cho có thể giải đồng thời được. Do vậy, bằng các thuật toán song song và nhờ việc sử dụng các máy tính có bộ đa xử lý, người ta hy vọng có thể giải nhanh các bài toán phức tạp. Trong thuật toán song song có một dãy các chỉ thị theo dõi việc thực hiện thuật toán, gửi các bài toán con tới các bộ xử lý khác nhau, chuyển các thông tin vào, thông tin ra tới các bộ xử lý thích hợp.

Khi dùng cách xử lý song song, mỗi bộ xử lý có thể cần các thông tin ra của các bộ xử lý khác. Do đó chúng cần phải được kết nối với nhau. Người ta có thể dùng loại đồ thị thích hợp để biểu diễn mạng kết nối các bộ xử lý trong một máy tính có nhiều bộ xử lý. Kiểu mạng kết nối dùng để thực hiện một thuật toán song song cụ thể phụ thuộc vào những yêu cầu với việc trao đổi dữ liệu giữa các bộ xử lý, phụ thuộc vào tốc độ mong muốn và tất nhiên vào phần cứng hiện có.

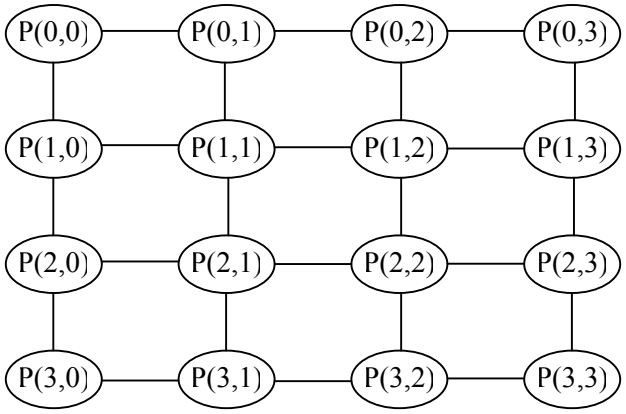
Mạng kết nối các bộ xử lý đơn giản nhất và cũng đắt nhất là có các liên kết hai chiều giữa mỗi cặp bộ xử lý. Các mạng này có thể mô hình bằng đồ thị đầy đủ K_n, trong đó n là số bộ xử lý. Tuy nhiên, các mạng liên kết kiểu này có số kết nối quá nhiều mà trong thực tế số kết nối cần phải có giới hạn.

Các bộ xử lý có thể kết nối đơn giản là sắp xếp chúng theo một mảng một chiều. Ưu điểm của mảng một chiều là mỗi bộ xử lý có nhiều nhất 2 đường nối trực tiếp với các bộ xử lý khác. Nhược điểm là nhiều khi cần có rất nhiều các kết nối trung gian để các bộ xử lý trao đổi thông tin với nhau.

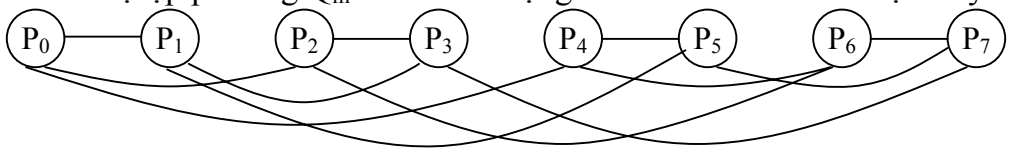


Mạng kiểu lưới (hoặc mảng hai chiều) rất hay được dùng cho các mạng liên kết. Trong một mạng như thế, số các bộ xử lý là một số chính phương, $n=m^2$. Các bộ xử lý

được gán nhãn P(i,j), $0 \le i, j \le m-1$. Các kết nối hai chiều sẽ nối bộ xử lý P(i,j) với bốn bộ xử lý bên cạnh, tức là với P(i,j±1) và P(i±1,j) chừng nào các bộ xử lý còn ở trong lưới.



Mạng kết nối quan trọng nhất là mạng kiểu siêu khối. Với các mạng loại này số các bộ xử lý là luỹ thừa của 2, $n=2^m$. Các bộ xử lý được gán nhãn là P_0, P_1, ..., P_{n-1}. Mỗi bộ xử lý có liên kết hai chiều với m bộ xử lý khác. Bộ xử lý P_i nối với bộ xử lý có chỉ số biểu diễn bằng dãy nhị phân khác với dãy nhị phân biểu diễn i tại đúng một bit. Mạng kiểu siêu khối cân bằng số các kết nối trực tiếp của mỗi bộ xử lý và số các kết nối gián tiếp sao cho các bộ xử lý có thể truyền thông được. Nhiều máy tính đã chế tạo theo mạng kiểu siêu khối và nhiều thuật toán đã được thiết kế để sử dụng mạng kiểu siêu khối. Đồ thị lập phương Q_m biểu diễn mạng kiểu siêu khối có 2^m bộ xử lý.



3.4. BIỂU DIỄN ĐỒ THỊ BẰNG MA TRẬN VÀ SỰ ĐẲNG CẤU ĐỒ THỊ:

3.4.1. Định nghĩa: Cho đồ thị G=(V,E) (vô hướng hoặc có hướng), với $V=\{v_1,v_2,...,v_n\}$. Ma trận liền kề của G ứng với thứ tự các đỉnh $v_1,v_2,...,v_n$ là ma trận

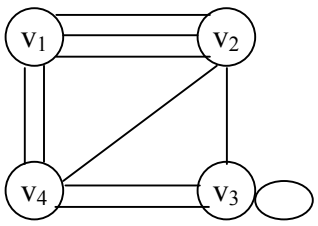
$$A=(a_{ij})_{1\le i,j\le n} \in M(n,Z),$$

trong đó a_{ij} là số cạnh hoặc cung nối từ v_i tới v_j.

Như vậy, ma trận liền kề của một đồ thị vô hướng là ma trận đối xứng, nghĩa là $a_{ij} = a_{ji}$, trong khi ma trận liền kề của một đồ thị có hướng không có tính đối xứng.

Thí dụ 11: Ma trận liền kề với thứ tự các đỉnh v_1, v_2, v_3, v_4 là:

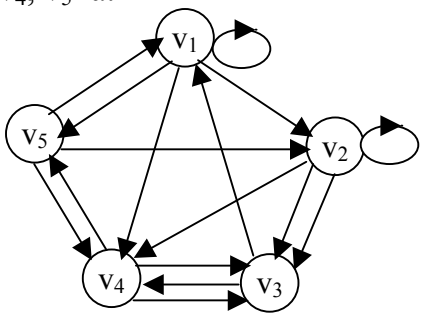
$$\begin{pmatrix} 0 & 3 & 0 & 2 \\ 3 & 0 & 1 & 1 \\ 0 & 1 & 1 & 2 \\ 2 & 1 & 2 & 0 \end{pmatrix}$$



www.VIETMATHS.com

Ma trận liền kề với thứ tự các đỉnh v_1, v_2, v_3, v_4, v_5 là:

$$\begin{pmatrix} 1 & 1 & 0 & 1 & 1 \\ 0 & 1 & 2 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 2 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 \end{pmatrix}$$



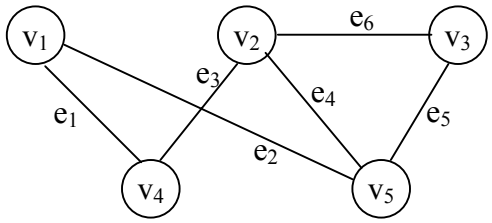
3.4.2. Định nghĩa: Cho đồ thị vô hướng G=(V,E), v_1, v_2, ..., v_n là các đỉnh và e_1, e_2, ..., e_m là các cạnh của G. Ma trận liên thuộc của G theo thứ tự trên của V và E là ma trận

$$M=(m_{ij})_{\substack{1\le i\le n \\ 1\le j\le m}} \in M(n\times m, Z),$$

m_{ij} bằng 1 nếu cạnh e_j nối với đỉnh v_i và bằng 0 nếu cạnh e_j không nối với đỉnh v_i.

Thí dụ 12: Ma trận liên thuộc theo thứ tự các đỉnh v_1, v_2, v_3, v_4, v_5 và các cạnh e_1, e_2, e_3, e_4, e_5, e_6 là:

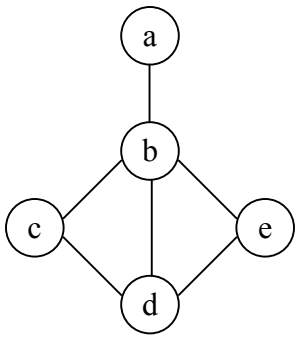
$$\begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 \end{pmatrix}$$



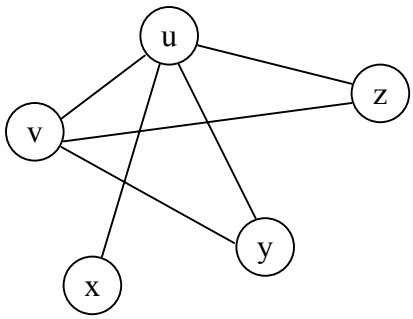
3.4.3. Định nghĩa: Các đơn đồ thị $G_1=(V_1,E_1)$ và $G_2=(V_2,E_2)$ được gọi là đẳng cấu nếu tồn tại một song ánh f từ V_1 lên V_2 sao cho các đỉnh u và v là liền kề trong G_1 khi và chỉ khi f(u) và f(v) là liền kề trong G_2 với mọi u và v trong V_1. Ánh xạ f như thế gọi là một phép đẳng cấu.

Thông thường, để chứng tỏ hai đơn đồ thị là không đẳng cấu, người ta chỉ ra chúng không có chung một tính chất mà các đơn đồ thị đẳng cấu cần phải có. Tính chất như thế gọi là một bất biến đối với phép đẳng cấu của các đơn đồ thị.

Thí dụ 13: 1) Hai đơn đồ thị G_1 và G_2 sau là đẳng cấu qua phép đẳng cấu f: $a\mapsto x$, $b\mapsto u$, $c\mapsto z$, $d\mapsto v$, $e\mapsto y$:



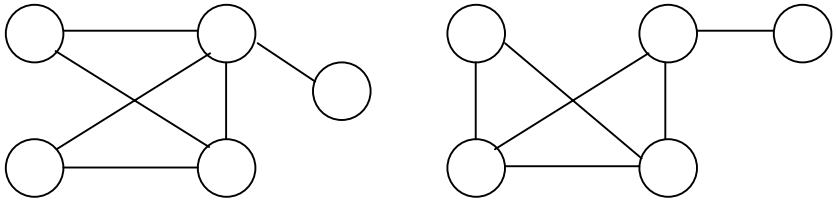
G_1



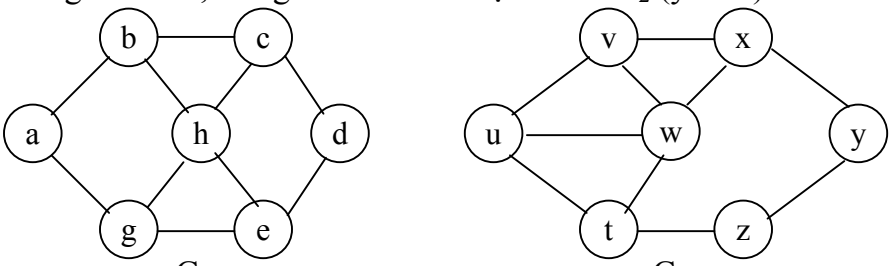
G_2

www.VIETMATHS.com

2) Hai đồ thị G_1 và G_2 sau đều có 5 đỉnh và 6 cạnh nhưng không đẳng cấu vì trong G_1 có một đỉnh bậc 4 mà trong G_2 không có đỉnh bậc 4 nào.

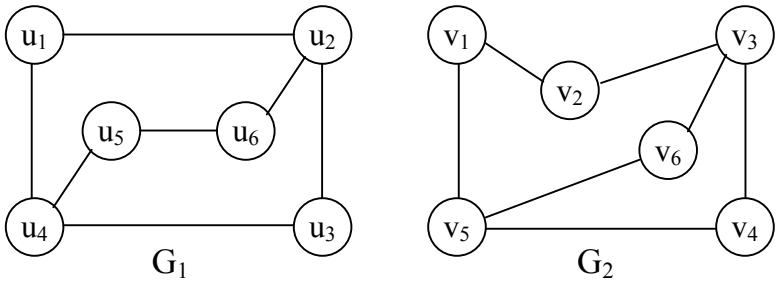


3) Hai đồ thị G_1 và G_2 sau đều có 7 đỉnh, 10 cạnh, cùng có một đỉnh bậc 4, bốn đỉnh bậc 3 và hai đỉnh bậc 2. Tuy nhiên G_1 và G_2 là không đẳng cấu vì hai đỉnh bậc 2 của G_1 (a và d) là không kề nhau, trong khi hai đỉnh bậc 2 của G_2 (y và z) là kề nhau.



G_1 G_2

4) Hãy xác định xem hai đồ thị sau có đẳng cấu hay không?



G_1 G_2

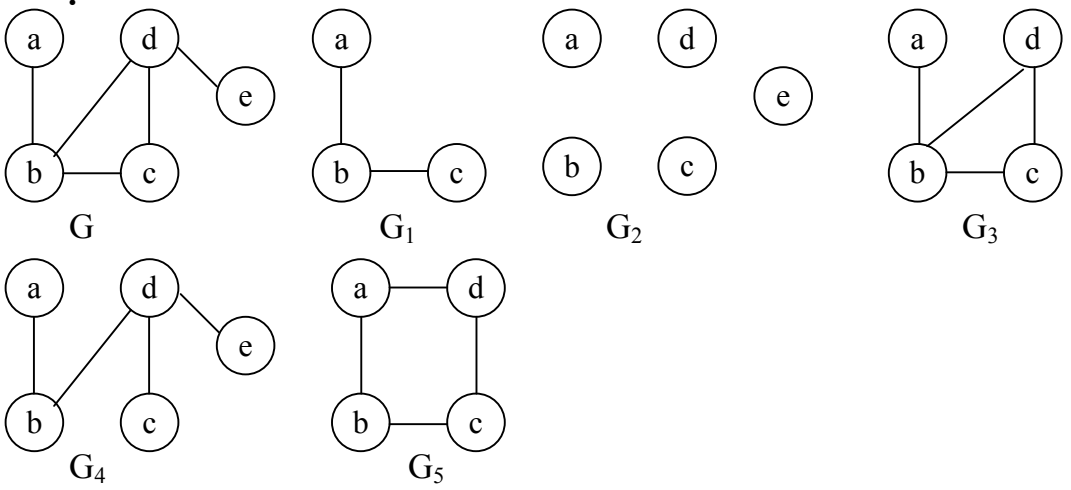
Hai đồ thị G_1 và G_2 là đẳng cấu vì hai ma trận liền kề của G_1 theo thứ tự các đỉnh $u_1, u_2, u_3, u_4, u_5, u_6$ và của G_2 theo thứ tự các đỉnh $v_6, v_3, v_4, v_5, v_1, v_2$ là như nhau và bằng:

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

3.5. CÁC ĐỒ THỊ MỚI TỪ ĐỒ THỊ CŨ.

3.5.1. Định nghĩa: Cho hai đồ thị $G_1=(V_1,E_1)$ và $G_2=(V_2,E_2)$. Ta nói G_2 là đồ thị con của G_1 nếu $V_2 \subset V_1$ và $E_2 \subset E_1$. Trong trường hợp $V_1=V_2$ thì G_2 gọi là con bao trùm của G_1.

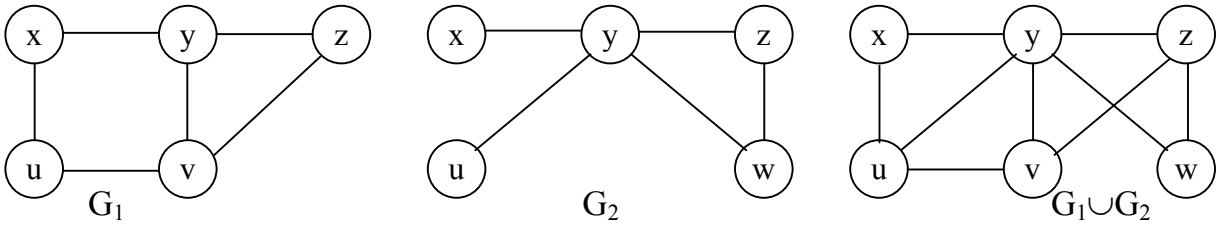
Thí dụ 14:



G_1, G_2, G_3 và G_4 là các đồ thị con của G, trong đó G_2 và G_4 là đồ thị con bao trùm của G, còn G_5 không phải là đồ thị con của G.

3.5.2. Định nghĩa: Hợp của hai đơn đồ thị $G_1=(V_1,E_1)$ và $G_2=(V_2,E_2)$ là một đơn đồ thị có tập các đỉnh là $V_1 \cup V_2$ và tập các cạnh là $E_1 \cup E_2$, ký hiệu là $G_1 \cup G_2$.

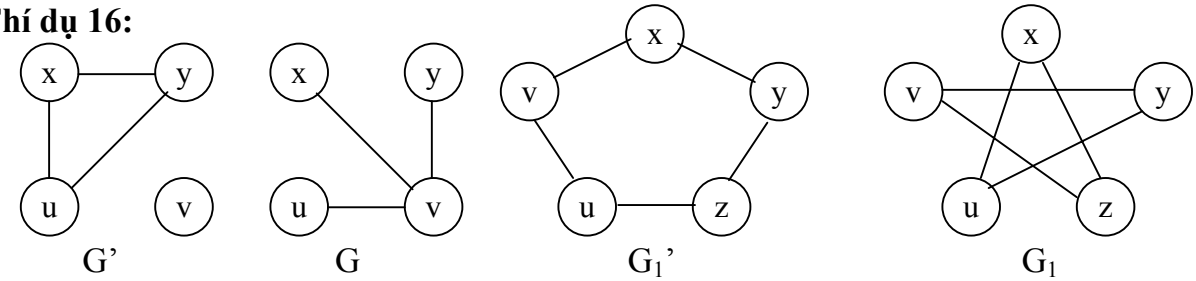
Thí dụ 15:



3.5.3. Định nghĩa: Đơn đồ thị G'=(V,E') được gọi là đồ thị bù của đơn đồ thị G=(V,E) nếu G và G' không có cạnh chung nào ($E \cap E'=\varnothing$) và $G \cup G'$là đồ thị đầy đủ.

Dễ thấy rằng nếu G' là bù của G thì G cũng là bù của G'. Khi đó ta nói hai đồ thị là bù nhau.

Thí dụ 16:



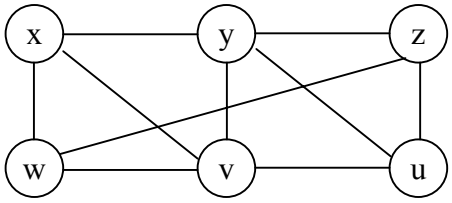
Hai đồ thị G' và G là bù nhau và hai đồ thị G_1 và G_1' là bù nhau.

3.6. TÍNH LIÊN THÔNG.

3.6.1. Định nghĩa: Đường đi độ dài n từ đỉnh u đến đỉnh v, với n là một số nguyên dương, trong đồ thị (giả đồ thị vô hướng hoặc đa đồ thị có hướng) G=(V,E) là một dãy các cạnh (hoặc cung) $e_1, e_2, ..., e_n$ của đồ thị sao cho $e_1=(x_0,x_1), e_2=(x_1,x_2), ..., e_n=(x_{n-1},x_n)$, với $x_0=u$ và $x_n=v$. Khi đồ thị không có cạnh (hoặc cung) bội, ta ký hiệu đường đi này

bằng dãy các đỉnh x_0, x_1, ..., x_n. Đường đi được gọi là chu trình nếu nó bắt đầu và kết thúc tại cùng một đỉnh. Đường đi hoặc chu trình gọi là đơn nếu nó không chứa cùng một cạnh (hoặc cung) quá một lần. Một đường đi hoặc chu trình không đi qua đỉnh nào quá một lần (trừ đỉnh đầu và đỉnh cuối của chu trình là trùng nhau) được gọi là đường đi hoặc chu trình sơ cấp. Rõ ràng rằng một đường đi (t.ư. chu trình) sơ cấp là đường đi (t.ư. chu trình) đơn.

Thí dụ 17:

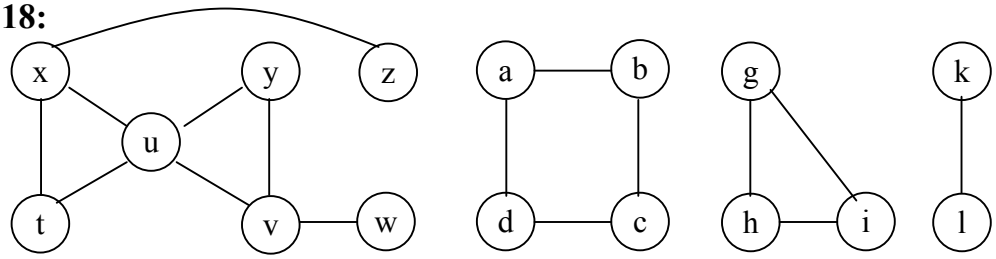


Trong đơn đồ thị trên, x, y, z, w, v, y là đường đi đơn (không sơ cấp) độ dài 5; x, w, v, z, y không là đường đi vì (v, z) không là cạnh; y, z, w, x, v, u, y là chu trình sơ cấp độ dài 6.

3.6.2. Định nghĩa: Một đồ thị (vô hướng) được gọi là liên thông nếu có đường đi giữa mọi cặp đỉnh phân biệt của đồ thị.

Một đồ thị không liên thông là hợp của hai hay nhiều đồ thị con liên thông, mỗi cặp các đồ thị con này không có đỉnh chung. Các đồ thị con liên thông rời nhau như vậy được gọi là các thành phần liên thông của đồ thị đang xét. Như vậy, một đồ thị là liên thông khi và chỉ khi nó chỉ có một thành phần liên thông.

Thí dụ 18:

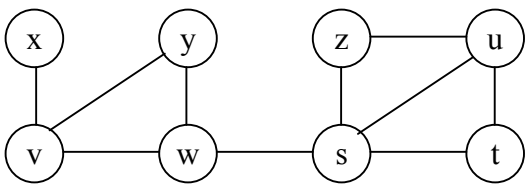


G G’

Đồ thị G là liên thông, nhưng đồ thị G’ không liên thông và có 3 thành phần liên thông.

3.6.3. Định nghĩa: Một đỉnh trong đồ thị G mà khi xoá đi nó và tất cả các cạnh liên thuộc với nó ta nhận được đồ thị con mới có nhiều thành phần liên thông hơn đồ thị G được gọi là đỉnh cắt hay điểm khớp. Việc xoá đỉnh cắt khỏi một đồ thị liên thông sẽ tạo ra một đồ thị con không liên thông. Hoàn toàn tương tự, một cạnh mà khi ta bỏ nó đi sẽ tạo ra một đồ thị có nhiều thành phần liên thông hơn so với đồ thị xuất phát được gọi là cạnh cắt hay là cầu.

Thí dụ 19:



Trong đồ thị trên, các đỉnh cắt là v, w, s và các cầu là (x,v), (w,s).

3.6.4. Mệnh đề: Giữa mọi cặp đỉnh phân biệt của một đồ thị liên thông luôn có đường đi sơ cấp.

Chứng minh: Giả sử u và v là hai đỉnh phân biệt của một đồ thị liên thông G. Vì G liên thông nên có ít nhất một đường đi giữa u và v. Gọi $x_0, x_1, ..., x_n$, với $x_0=u$ và $x_n=v$, là dãy các đỉnh của đường đi có độ dài ngắn nhất. Đây chính là đường đi sơ cấp cần tìm. Thật vậy, giả sử nó không là đường đi đơn, khi đó $x_i=x_j$ với $0 \le i < j$. Điều này có nghĩa là giữa các đỉnh u và v có đường đi ngắn hơn qua các đỉnh $x_0, x_1, ..., x_{i-1}, x_j, ..., x_n$ nhận được bằng cách xoá đi các cạnh tương ứng với dãy các đỉnh $x_i, ..., x_{j-1}$.

3.6.5. Mệnh đề: Mọi đơn đồ thị n đỉnh ($n \ge 2$) có tổng bậc của hai đỉnh tuỳ ý không nhỏ hơn n đều là đồ thị liên thông.

Chứng minh: Cho đơn đồ thị G=(V,E) có n đỉnh ($n \ge 2$) và thoả mãn yêu cầu của bài toán. Giả sử G không liên thông, tức là tồn tại hai đỉnh u và v sao cho không có đường đi nào nối u và v. Khi đó trong đồ thị G tồn tại hai thành phần liên thông là G_1 có n_1 đỉnh và chứa u, G_2 chứa đỉnh v và có n_2 đỉnh. Vì G_1, G_2 là hai trong số các thành phần liên thông của G nên $n_1+n_2 \le n$. ta có:

$$\deg(u)+\deg(v) \le (n_1 -1)+(n_2 - 1) = n_1+n_2-2 \le n-2 < n.$$

Điều mâu thuẫn ở trên dẫn đến kết luận là đồ thị G phải liên thông.

3.6.6. Hệ quả: Đơn đồ thị mà bậc của mỗi đỉnh của nó không nhỏ hơn một nửa số đỉnh là đồ thị liên thông.

3.6.7. Mệnh đề: Nếu một đồ thị có đúng hai đỉnh bậc lẻ thì hai đỉnh này phải liên thông, tức là có một đường đi nối chúng.

Chứng minh: Cho G=(V,E) là đồ thị thị có đúng hai đỉnh bậc lẻ là u và v. Giả sử u và v không liên thông với nhau. Khi đó chúng phải thuộc hai thành phần liên thông nào đó của đồ thị G, G_1 chứa u và G_2 chứa v.

Bậc của đỉnh u trong G_1 cũng chính là bậc của u trong G, nên trong G_1 đỉnh u vẫn có bậc lẻ và G_1 có duy nhất một đỉnh bậc lẻ. Điều này mâu thuẫn. Vậy hai đỉnh u và v phải liên thông.

3.6.8. Mệnh đề: Cho G=(V,E) là một đồ thị liên thông. Khi đó một đỉnh của G là điểm khớp khi và chỉ khi trong G tồn tại hai đỉnh u và v sao cho mỗi đường đi nối u và v đều phải đi qua đỉnh này.

Chứng minh: *Điều kiện cần*: Giả sử đỉnh x là điểm khớp trong đồ thị G. Khi đó đồ thị con G_1 của G nhận được bằng cách xoá x và các cạnh liên thuộc với nó là không liên thông. Giả sử G_2, G_3 là hai trong các thành phần liên thông của G_1. Lấy u là đỉnh trong G_2 và v là đỉnh trong G_3. Do u, v thuộc hai thành phần liên thông khác nhau, nên trong G_1 các đỉnh u, v không liên thông. Nhưng trong G các đỉnh u, v lại liên thông, nên mọi đường đi nối u, v đều phải đi qua đỉnh x.

Điều kiện đủ: Giả sử mọi đường đi nối u, v đều đi qua đỉnh x, nên nếu bỏ đỉnh x và các cạnh liên thuộc với x thì đồ thị con G_1 nhận được từ G chứa hai đỉnh u, v không liên thông. Do đó G_1 là đồ thị không liên thông hay đỉnh x là điểm khớp của G.

3.6.9. Định lý: Cho G là một đơn đồ thị có n đỉnh, m cạnh và k thành phần liên thông. Khi đó

$$n-k \le m \le \frac{(n-k)(n-k+1)}{2}.$$

Chứng minh: Bất đẳng thức $n-k \le m$ được chứng minh bằng quy nạp theo m. Nếu m=0 thì k=n nên bất đẳng thức đúng. Giả sử bất đẳng thức đúng đến m−1, với m ≥ 1. Gọi G' là đồ thị con bao trùm của G có số cạnh m_0 là nhỏ nhất sao cho nó có k thành phần liên thông. Do đó việc loại bỏ bất cứ cạnh nào trong G' cũng tăng số thành phần liên thông lên 1 và khi đó đồ thị thu được sẽ có n đỉnh, k+1 thành phần liên thông và m_0−1 cạnh. Theo giả thiết quy nạp, ta có $m_0-1 \ge n-(k+1)$ hay $m_0 \ge n-k$. Vậy m ≥ n-k.

Bổ sung cạnh vào G để nhận được đồ thị G'' có m_1 cạnh sao cho k thành phần liên thông là những đồ thị đầy đủ. Ta có $m \le m_1$ nên chỉ cần chứng minh

$$m_1 \le \frac{(n-k)(n-k+1)}{2}.$$

Giả sử G_i và G_j là hai thành phần liên thông của G'' với n_i và n_j đỉnh và $n_i \ge n_j > 1$ (*). Nếu ta thay G_i và G_j bằng đồ thị đầy đủ với n_i+1 và n_j-1 đỉnh thì tổng số đỉnh không thay đổi nhưng số cạnh tăng thêm một lượng là:

$$\left[\frac{(n_i+1)n_i}{2}-\frac{n_i(n_i-1)}{2}\right]-\left[\frac{n_j(n_j-1)}{2}-\frac{(n_j-1)(n_j-2)}{2}\right]=n_i-n_j+1.$$

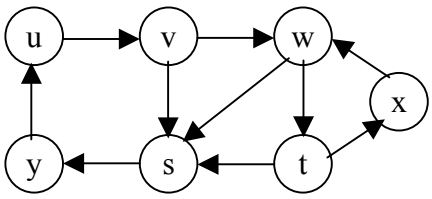
Thủ tục này được lặp lại khi hai thành phần nào đó có số đỉnh thoả (*). Vì vậy m_1 là lớn nhất (n, k là cố định) khi đồ thị gồm k-1 đỉnh cô lập và một đồ thị đầy đủ với n-k+1 đỉnh. Từ đó suy ra bất đẳng thức cần tìm.

3.6.10. Định nghĩa: Đồ thị có hướng G được gọi là liên thông mạnh nếu với hai đỉnh phân biệt bất kỳ u và v của G đều có đường đi từ u tới v và đường đi từ v tới u.

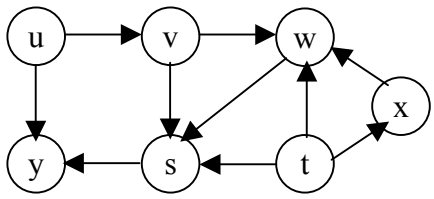
Đồ thị có hướng G được gọi là liên thông yếu nếu đồ thị vô hướng nền của nó là liên thông.

Đồ thị có hướng G được gọi là liên thông một chiều nếu với hai đỉnh phân biệt bất kỳ u và v của G đều có đường đi từ u tới v hoặc đường đi từ v tới u.

Thí dụ 20:



G



G'

Đồ thị G là liên thông mạnh nhưng đồ thị G' là liên thông yếu (không có đường đi từ u tới x cũng như từ x tới u).

3.6.11. Mệnh đề: Cho G là một đồ thị (vô hướng hoặc có hướng) với ma trận liền kề A theo thứ tự các đỉnh $v_1, v_2, ..., v_n$. Khi đó số các đường đi khác nhau độ dài r từ v_i tới v_j trong đó r là một số nguyên dương, bằng giá trị của phần tử dòng i cột j của ma trận A^r.

Chứng minh: Ta chứng minh mệnh đề bằng quy nạp theo r. Số các đường đi khác nhau độ dài 1 từ v_i tới v_j là số các cạnh (hoặc cung) từ vi tới v_j, đó chính là phần tử dòng i cột j của ma trận A; nghĩa là, mệnh đề đúng khi r=1.

Giả sử mệnh đề đúng đến r; nghĩa là, phần tử dòng i cột j của A^r là số các đường đi khác nhau độ dài r từ v_i tới v_j. Vì $A^{r+1}=A^r.A$ nên phần tử dòng i cột j của A^{r+1} bằng

$$b_{i1}a_{1j}+b_{i2}a_{2j}+ ... +b_{in}a_{nj},$$

trong đó b_{ik} là phần tử dòng i cột k của A^r. Theo giả thiết quy nạp b_{ik} là số đường đi khác nhau độ dài r từ v_i tới v_k.

Đường đi độ dài r+1 từ v_i tới v_j sẽ được tạo nên từ đường đi độ dài r từ v_i tới đỉnh trung gian v_k nào đó và một cạnh (hoặc cung) từ v_k tới v_j. Theo quy tắc nhân số các đường đi như thế là tích của số đường đi độ dài r từ v_i tới v_k, tức là b_{ik}, và số các cạnh (hoặc cung) từ v_k tới v_j, tức là a_{kj}. Cộng các tích này lại theo tất cả các đỉnh trung gian v_k ta có mệnh đề đúng đến r+1.

BÀI TẬP CHƯƠNG III:

1. Cho G là đồ thị có v đỉnh và e cạnh, còn M, m tương ứng là bậc lớn nhất và nhỏ nhất của các đỉnh của G. Chứng tỏ rằng

$$m \le \frac{2e}{v} \le M.$$

2. Chứng minh rằng nếu G là đơn đồ thị phân đôi có v đỉnh và e cạnh, khi đó

$$e \le v^2/4.$$

3. Trongmột phương án mạng kiểu lưới kết nối $n=m^2$ bộ xử lý song song, bộ xử lý P(i,j) được kết nối với 4 bộ xử lý (P(i±1) mod m, j), P(i, (j±1) mod m), sao cho các kết nối bao xung quanh các cạnh của lưới. Hãy vẽ mạng kiểu lưới có 16 bộ xử lý theo phương án này.

4. Hãy vẽ các đồ thị vô hướng được biểu diễn bởi ma trận liền kề sau:

a) $\begin{pmatrix} 1 & 2 & 3 \\ 2 & 0 & 4 \\ 3 & 4 & 0 \end{pmatrix}$, **b)** $\begin{pmatrix} 1 & 2 & 0 & 1 \\ 2 & 0 & 3 & 0 \\ 0 & 3 & 1 & 1 \\ 1 & 0 & 1 & 0 \end{pmatrix}$, **c)** $\begin{pmatrix} 0 & 1 & 3 & 0 & 4 \\ 1 & 2 & 1 & 3 & 0 \\ 3 & 1 & 1 & 0 & 1 \\ 0 & 3 & 0 & 0 & 2 \\ 4 & 0 & 1 & 2 & 3 \end{pmatrix}$.

5. Nêu ý nghĩa của tổng các phần tử trên một hàng (t.ư. cột) của một ma trận liền kề đối với một đồ thị vô hướng ? Đối với đồ thị có hướng ?

6. Tìm ma trận liền kề cho các đồ thị sau:

a) K_n , **b)** C_n, **c)** W_n , **d)** $K_{m,n}$, **e)** Q_n.

7. Có bao nhiêu đơn đồ thị không đẳng cấu với n đỉnh khi:

a) n=2, **b)** n=3, **c)** n=4.

8. Hai đơn đồ thị với ma trận liền kề sau đây có là đẳng cấu không?

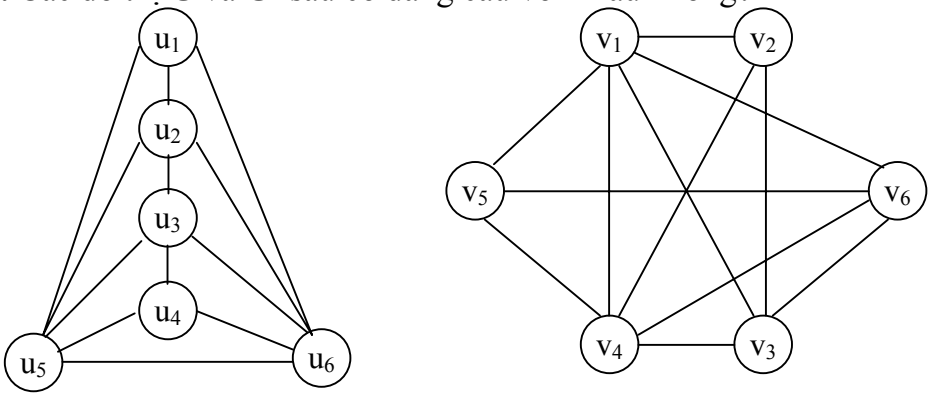
$$\begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix}, \quad \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix}.$$

9. Hai đơn đồ thị với ma trận liền kề sau đây có là đẳng cấu không?

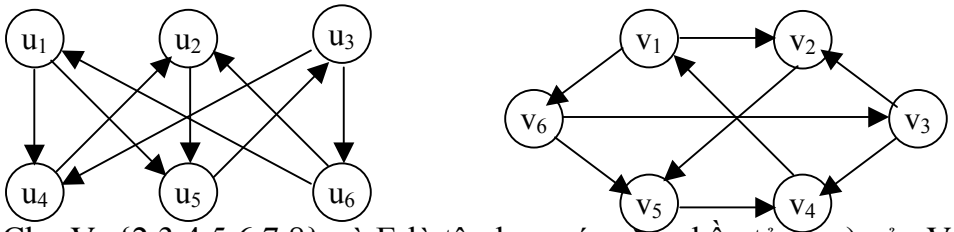
$$\begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 \end{pmatrix}, \quad \begin{pmatrix} 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \end{pmatrix}.$$

10. Các đồ thị G và G' sau có đẳng cấu với nhau không?

a)



b)



11. Cho V={2,3,4,5,6,7,8} và E là tập hợp các cặp phần tử (u,v) của V sao cho u<v và u,v nguyên tố cùng nhau. Hãy vẽ đồ thị có hướng G=(V,E). Tìm số các đường đi phân biệt độ dài 3 từ đỉnh 2 tới đỉnh 8.

12. Hãy tìm số đường đi độ dài n giữa hai đỉnh liền kề (t.ư. không liền kề) tùy ý trong $K_{3,3}$ với mỗi giá trị của n sau:

a) n=2, **b)** n=3, **c)** n=4, **d)** n=5.

14. Một cuộc họp có ít nhất ba đại biểu đến dự. Mỗi người quen ít nhất hai đại biểu khác. Chứng minh rằng có thể xếp được một số đại biểu ngồi xung quanh một bàn tròn, để mỗi người ngồi giữa hai người mà đại biểu đó quen.

15. Một lớp học có ít nhất 4 sinh viên. Mỗi sinh viên thân với ít nhất 3 sinh viên khác. Chứng minh rằng có thể xếp một số chẵn sinh viên ngồi quanh một cái bàn tròn để mỗi sinh viên ngồi giữa hai sinh viên mà họ thân.

16. Trong một cuộc họp có đúng hai đại biểu không quen nhau và mỗi đại biểu này có một số lẻ người quen đến dự. Chứng minh rằng luôn luôn có thể xếp một số đại biểu ngồi chen giữa hai đại biểu nói trên, để mỗi người ngồi giữa hai người mà anh ta quen.

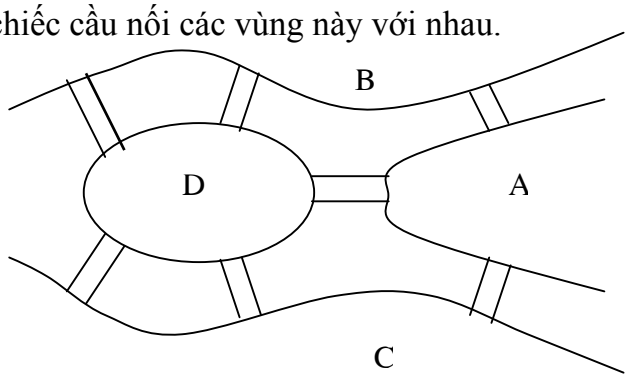
17. Một thành phố có n ($n \geq 2$) nút giao thông và hai nút giao thông bất kỳ đều có số đầu mối đường ngầm tới một trong các nút giao thông này đều không nhỏ hơn n. Chứng minh rằng từ một nút giao thông tuỳ ý ta có thể đi đến một nút giao thông bất kỳ khác bằng đường ngầm.

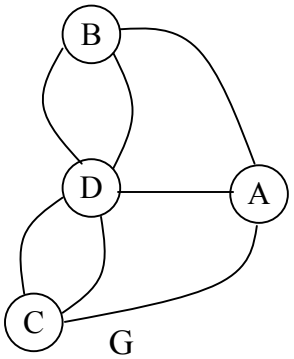
CHƯƠNG IV

ĐỒ THỊ EULER VÀ ĐỒ THỊ HAMILTON

4.1. ĐƯỜNG ĐI EULER VÀ ĐỒ THỊ EULER.

Có thể coi năm 1736 là năm khai sinh lý thuyết đồ thị, với việc công bố lời giải "bài toán về các cầu ở Konigsberg" của nhà toán học lỗi lạc Euler (1707-1783). Thành phố Konigsberg thuộc Phổ (nay gọi là Kaliningrad thuộc Nga) được chia thành bốn vùng bằng các nhánh sông Pregel, các vùng này gồm hai vùng bên bờ sông, đảo Kneiphof và một miền nằm giữa hai nhánh của sông Pregel. Vào thế kỷ 18, người ta xây bảy chiếc cầu nối các vùng này với nhau.





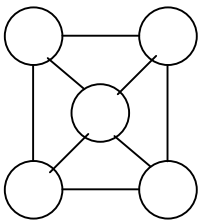
G

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

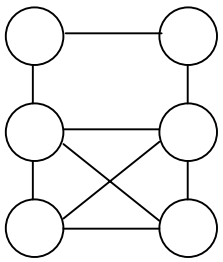
Bài toán tìm đường đi qua tất cả các cầu, mỗi cầu chỉ qua một lần có thể được phát biểu lại bằng mô hình này như sau: Có tồn tại chu trình đơn trong đa đồ thị G chứa tất cả các cạnh?

4.1.1. Định nghĩa: Chu trình (t.ư. đường đi) đơn chứa tất cả các cạnh (hoặc cung) của đồ thị (vô hướng hoặc có hướng) G được gọi là chu trình (t.ư. đường đi) Euler. Một đồ thị liên thông (liên thông yếu đối với đồ thị có hướng) có chứa một chu trình (t.ư. đường đi) Euler được gọi là đồ thị Euler (t.ư. nửa Euler).

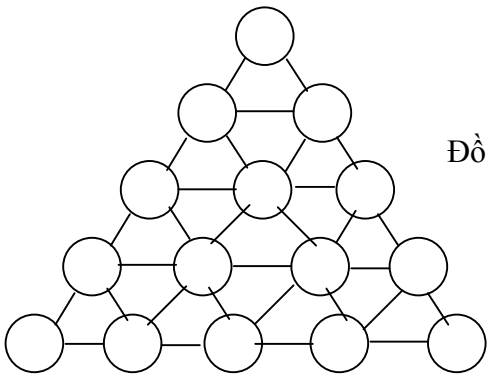
Thí dụ 1:



Đồ thị không nửa Euler

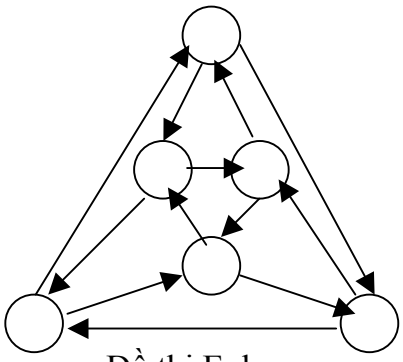


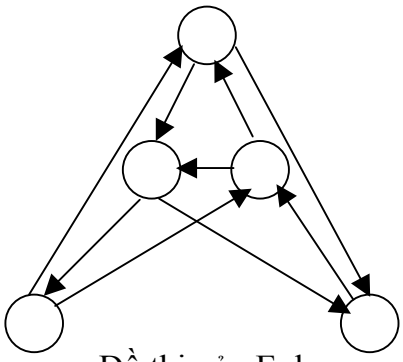
Đồ thị nửa Euler



Đồ thị Euler

www.VIETMATHS.com


Đồ thị Euler


Đồ thị nửa Euler

Điều kiện cần và đủ để một đồ thị là đồ thị Euler được Euler tìm ra vào năm 1736 khi ông giải quyết bài toán hóc búa nổi tiếng thời đó về bảy cái cầu ở Konigsberg và đây là định lý đầu tiên của lý thuyết đồ thị.

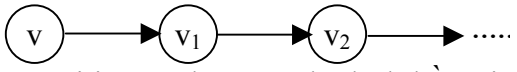
4.1.2. Định lý: Đồ thị (vô hướng) liên thông G là đồ thị Euler khi và chỉ khi mọi đỉnh của G đều có bậc chẵn.

Chứng minh:

***Điều kiện cần*:** Giả sử G là đồ thị Euler, tức là tồn tại chu trình Euler P trong G. Khi đó cứ mỗi lần chu trình P đi qua một đỉnh nào đó của G thì bậc của đỉnh đó tăng lên 2. Mặt khác, mỗi cạnh của đồ thị xuất hiện trong P đúng một lần. Do đó mỗi đỉnh của đồ thị đều có bậc chẵn.

4.1.3. Bổ đề: Nếu bậc của mỗi đỉnh của đồ thị G không nhỏ hơn 2 thì G chứa chu trình đơn.

Chứng minh: Nếu G có cạnh bội hoặc có khuyên thì khẳng định của bổ đề là hiển nhiên. Vì vậy giả sử G là một đơn đồ thị. Gọi v là một đỉnh nào đó của G. Ta sẽ xây dựng theo quy nạp đường đi

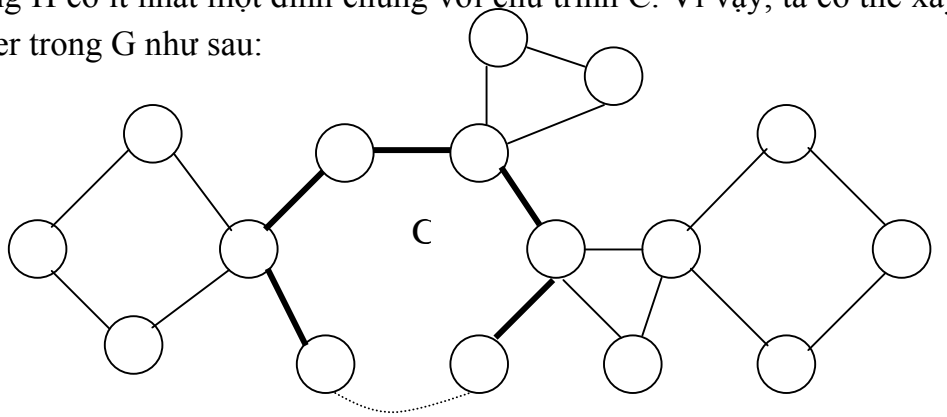


trong đó v_1 là đỉnh kề với v, còn với $i \geq 1$, chọn v_{i+1} là đỉnh kề với v_i và $v_{i+1} \neq v_{i-1}$ (có thể chọn như vậy vì $\deg(v_i) \geq 2$), $v_0 = v$. Do tập đỉnh của G là hữu hạn, nên sau một số hữu hạn bước ta phải quay lại một đỉnh đã xuất hiện trước đó. Gọi k là số nguyên dương đầu tiên để $v_k = v_i$ ($0 \leq i < k$). Khi đó, đường đi $v_i, v_{i+1}, ..., v_{k-1}, v_k$ ($= v_i$) là một chu trình đơn cần tìm.

Điều kiện đủ: Quy nạp theo số cạnh của G. Do G liên thông và bậc của mọi đỉnh là chẵn nên mỗi đỉnh có bậc không nhỏ hơn 2. Từ đó theo Bổ đề 4.1.3, G phải chứa một chu trình đơn C. Nếu C đi qua tất cả các cạnh của G thì nó chính là chu trình Euler. Giả sử C không đi qua tất cả các cạnh của G. Khi đó loại bỏ khỏi G các cạnh thuộc C, ta thu được một đồ thị mới H (không nhất thiết là liên thông). Số cạnh trong H nhỏ hơn trong G và rõ ràng mỗi đỉnh của H vẫn có bậc là chẵn. Theo giả thiết quy nạp, trong mỗi thành phần liên thông của H đều tìm được chu trình Euler. Do G liên thông nên mỗi thành

www.VIETMATHS.com

phần trong H có ít nhất một đỉnh chung với chu trình C. Vì vậy, ta có thể xây dựng chu trình Euler trong G như sau:



Bắt đầu từ một đỉnh nào đó của chu trình C, đi theo các cạnh của C chừng nào chưa gặp phải đỉnh không cô lập của H. Nếu gặp phải đỉnh như vậy thì ta đi theo chu trình Euler của thành phần liên thông của H chứa đỉnh đó. Sau đó lại tiếp tục đi theo cạnh của C cho đến khi gặp phải đỉnh không cô lập của H thì lại theo chu trình Euler của thành phần liên thông tương ứng trong H, ... Quá trình sẽ kết thúc khi ta trở về đỉnh xuất phát, tức là thu được chu trình đi qua mỗi cạnh của đồ thị đúng một lần.

4.1.4. Hệ quả: Đồ thị liên thông G là nửa Euler (mà không là Euler) khi và chỉ khi có đúng hai đỉnh bậc lẻ trong G.

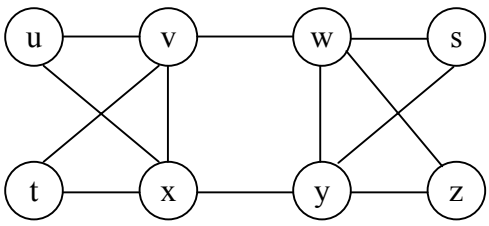
Chứng minh: Nếu G là nửa Euler thì tồn tại một đường đi Euler trong G từ đỉnh u đến đỉnh v. Gọi G' là đồ thị thu được từ G bằng cách thêm vào cạnh (u,v). Khi đó G' là đồ thị Euler nên mọi đỉnh trong G' đều có bậc chẵn (kể cả u và v). Vì vậy u và v là hai đỉnh duy nhất trong G có bậc lẻ.

Đảo lại, nếu có đúng hai đỉnh bậc lẻ là u và v thì gọi G' là đồ thị thu được từ G bằng cách thêm vào cạnh (u,v). Khi đó mọi đỉnh của G' đều có bậc chẵn hay G' là đồ thị Euler. Bỏ cạnh (u,v) đã thêm vào ra khỏi chu trình Euler trong G' ta có được đường đi Euler từ u đến v trong G hay G là nửa Euler.

4.1.5. Chú ý: Ta có thể vạch được một chu trình Euler trong đồ thị liên thông G có bậc của mọi đỉnh là chẵn theo thuật toán Fleury sau đây.

Xuất phát từ một đỉnh bất kỳ của G và tuân theo hai quy tắc sau:

1. Mỗi khi đi qua một cạnh nào thì xoá nó đi; sau đó xoá đỉnh cô lập (nếu có);
2. Không bao giờ đi qua một cầu, trừ phi không còn cách đi nào khác.



 www.VIETMATHS.com

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

4.1.6. Bài toán người phát thư Trung Hoa:

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

Bài toán được nhà toán học Trung Hoa Guan nêu lên đầu tiên (1960), vì vậy thường được gọi là "bài toán người phát thư Trung Hoa". Ta xét bài toán ở một dạng đơn giản như sau.

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

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

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

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

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

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

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

$$q + m(G),$$

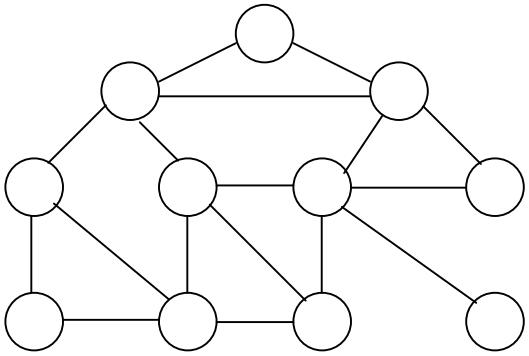
trong đó m(G) là số cạnh mà hành trình đi qua hai lần và được xác định như sau:

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

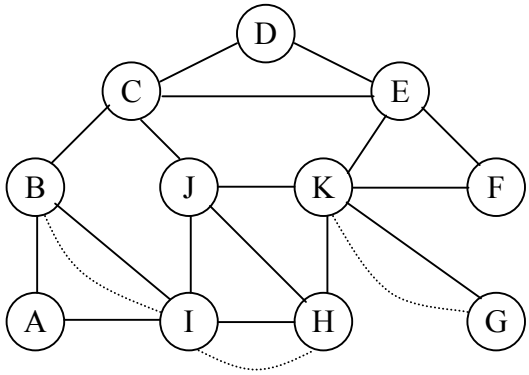
Ta gọi độ dài đường đi ngắn nhất từ u đến v là khoảng cách d(u,v). Đối với mọi phân hoạch cặp P_i, ta tính khoảng cách giữa hai đỉnh trong từng cặp, rồi tính tổng $d(P_i)$. Số m(G) bằng cực tiểu của các $d(P_i)$:

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

Thí dụ 2: Giải bài toán người phát thư Trung Hoa cho trong đồ thị sau:



G



G_T

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

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

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

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

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

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

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

4.1.7. Định lý: Đồ thị có hướng liên thông yếu G là đồ thị Euler khi và chỉ khi mọi đỉnh của G đều có bậc vào bằng bậc ra.

Chứng minh: Chứng minh tương tự như chứng minh của Định lý 4.1.2 và điều kiện đủ cũng cần có bổ đề dưới đây tương tự như ở Bổ đề 4.1.3.

4.1.8. Bổ đề: Nếu bậc vào và bậc ra của mỗi đỉnh của đồ thị có hướng G không nhỏ hơn 1 thì G chứa chu trình đơn.

4.1.9. Hệ quả: Đồ thị có hướng liên thông yếu G là nửa Euler (mà không là Euler) khi và chỉ khi tồn tại hai đỉnh x và y sao cho:

$$deg_o(x) = deg_t(x)+1,\ deg_t(y) = deg_o(y)+1,\ deg_t(v) = deg_o(v),\ \forall v\in V,\ v \neq x,\ v \neq y.$$

Chứng minh: Chứng minh tương tự như ở Hệ quả 4.1.4.

4.2. ĐƯỜNG ĐI HAMILTON VÀ ĐỒ THỊ HAMILTON.

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

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

Trước Hamilton, có thể là từ thời Euler, người ta đã biết đến một câu đố hóc búa về "đường đi của con mã trên bàn cờ". Trên bàn cờ, con mã chỉ có thể đi theo đường chéo của hình chữ nhật 2 x 3 hoặc 3 x 2 ô vuông. Giả sử bàn cờ có 8 x 8 ô vuông. Hãy tìm đường đi của con mã qua được tất cả các ô của bàn cờ, mỗi ô chỉ một lần rồi trở lại ô xuất phát.

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

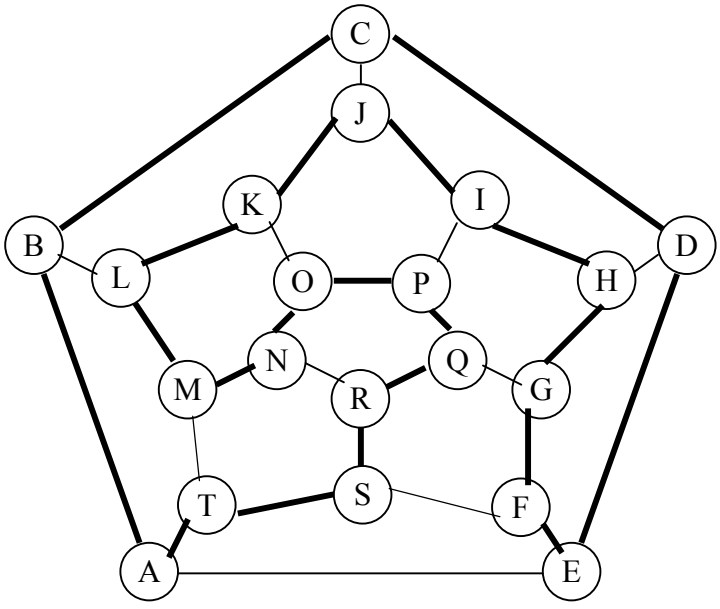
Hiện nay đã có nhiều lời giải và phương pháp giải cũng có rất nhiều, trong đó có quy tắc: mỗi lần bố trí con mã ta chọn vị trí mà tại vị trí này số ô chưa dùng tới do nó khống chế là ít nhất.

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

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

4.2.1. Định nghĩa: Chu trình (t.ư. đường đi) sơ cấp chứa tất cả các đỉnh của đồ thị (vô hướng hoặc có hướng) G được gọi là chu trình (t.ư. đường đi) Hamilton. Một đồ thị có chứa một chu trình (t.ư. đường đi) Hamilton được gọi là đồ thị Hamilton (t.ư. nửa Hamilton).

Thí dụ 3: 1)



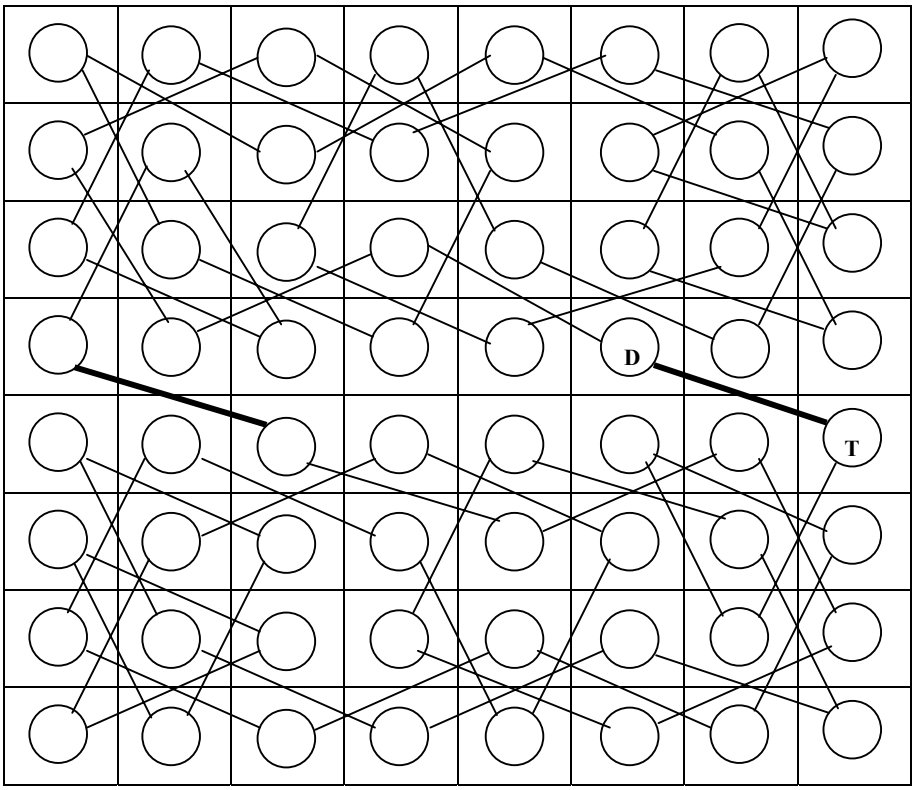
Đồ thị Hamilton (hình thập nhị diện đều biểu diễn trong mặt phẳng) với chu trình Hamilton A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, A (đường tô đậm).

2) Trong một đợt thi đấu bóng bàn có n ($n \geq 2$) đấu thủ tham gia. Mỗi đấu thủ gặp từng đấu thủ khác đúng một lần. Trong thi đấu bóng bàn chỉ có khả năng thắng hoặc thua.

Chứng minh rằng sau đợt thi đấu có thể xếp tất cả các đấu thủ đứng thành một hàng dọc, để người đứng sau thắng người đứng ngay trước anh (chị) ta.

Xét đồ thị có hướng G gồm n đỉnh sao cho mỗi đỉnh ứng với một đấu thủ và có một cung nối từ đỉnh u đến đỉnh v nếu đấu thủ ứng với u thắng đấu thủ ứng với v. Như vậy, đồ thị G có tính chất là với hai đỉnh phân biệt bất kỳ u và v, có một và chỉ một trong hai cung (u,v) hoặc (v,u), đồ thị như thế được gọi là đồ thị có hướng đầy đủ. Từ Mệnh đề 4.2.2 dưới đây, G là một đồ thị nửa Hamilton. Khi đó đường đi Hamilton trong G cho ta sự sắp xếp cần tìm.

3) Một lời giải về hành trình của con mã trên bàn cờ 8 x 8:



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

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

4.2.2. Định lý (Rédei): Nếu G là một đồ thị có hướng đầy đủ thì G là đồ thị nửa Hamilton.

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

-- Nếu α đã đi qua tất cả các đỉnh của G thì nó là một đường đi Hamilton của G.

-- Nếu trong G còn có đỉnh nằm ngoài α, thì ta có thể bổ sung dần các đỉnh này vào α và cuối cùng nhận được đường đi Hamilton.

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

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

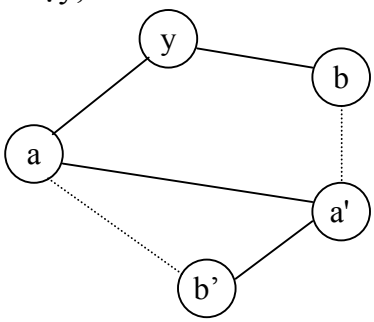
b) Nếu tồn tại chỉ số i ($1 \le i \le k-1$) mà từ v_i có cung nối tới v và từ v có cung nối tới v_{i+1} thì ta chen v vào giữa v_i và v_{i+1} để được đường đi sơ cấp $\alpha_2=(v_1, v_2, ..., v_i, v, v_{i+1}, ..., v_k)$.

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

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

4.2.3. Định lý (Dirac, 1952): Nếu G là một đơn đồ thị có n đỉnh và mọi đỉnh của G đều có bậc không nhỏ hơn $\frac{n}{2}$ thì G là một đồ thị Hamilton.

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



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

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

Như vậy, với mỗi đỉnh kề với a, ta có một đỉnh không kề với b, tức là số đỉnh không kề với b không thể ít hơn số đỉnh kề với a (số đỉnh kề với a không nhỏ hơn $\frac{n}{2}$+k).

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

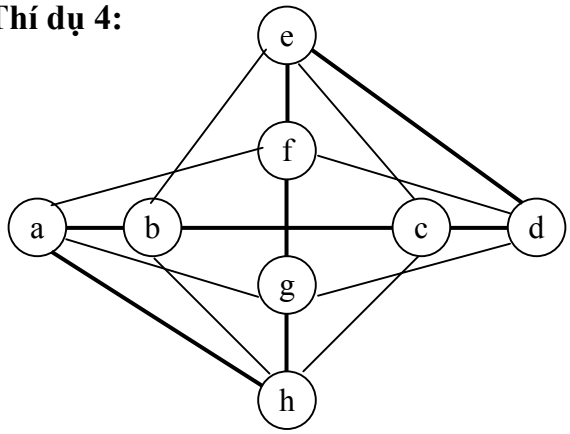
4.2.4. Hệ quả: Nếu G là đơn đồ thị có n đỉnh và mọi đỉnh của G đều có bậc không nhỏ hơn $\frac{n-1}{2}$ thì G là đồ thị nửa Hamilton.

Chứng minh: Thêm vào G một đỉnh x và nối x với mọi đỉnh của G thì ta nhận được đơn đồ thị G' có n+1 đỉnh và mỗi đỉnh có bậc không nhỏ hơn $\frac{n+1}{2}$. Do đó theo Định lý 4.2.3, trong G' có một chu trình Hamilton. Bỏ x ra khỏi chu trình này, ta nhận được đường đi Hamilton trong G.

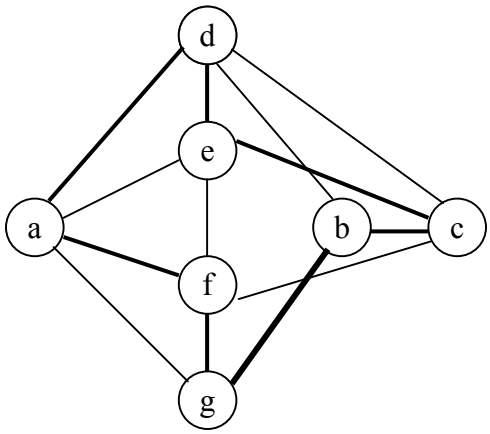
4.2.5. Định lý (Ore, 1960): Nếu G là một đơn đồ thị có n đỉnh và bất kỳ hai đỉnh nào không kề nhau cũng có tổng số bậc không nhỏ hơn n thì G là một đồ thị Hamilton.

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

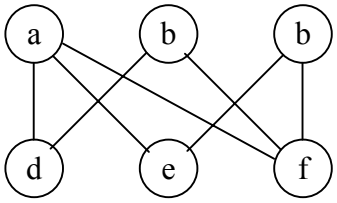
Thí dụ 4:



Đồ thị G này có 8 đỉnh, đỉnh nào cũng có bậc 4, nên theo Định lý 4.2.3, G là đồ thị Hamilton.



Đồ thị G' này có 5 đỉnh bậc 4 và 2 đỉnh bậc 2 kề nhau nên tổng số bậc của hai đỉnh không kề nhau bất kỳ bằng 7 hoặc 8, nên theo Định lý 4.2.5, G' là đồ thị Hamilton.



Đồ thị phân đôi này có bậc của mỗi đỉnh bằng 2 hoặc 3 (> 3/2), nên theo Định lý 4.2.6, nó là đồ thị Hamilton.

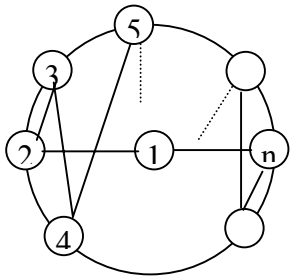
4.2.7. Bài toán sắp xếp chỗ ngồi:

Có n đại biểu từ n nước đến dự hội nghị quốc tế. Mỗi ngày họp một lần ngồi quanh một bàn tròn. Hỏi phải bố trí bao nhiêu ngày và bố trí như thế nào sao cho trong mỗi ngày, mỗi người có hai người kế bên là bạn mới. Lưu ý rằng n người đều muốn làm quen với nhau.

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

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

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



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

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

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

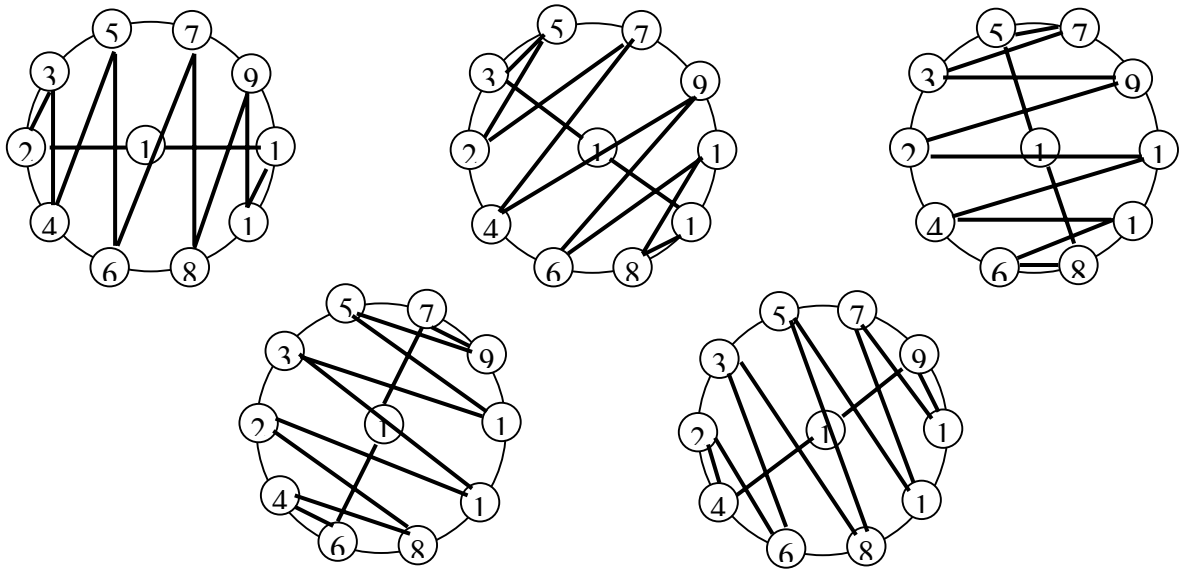
Thí dụ 5: Giải bài toán sắp xếp chỗ ngồi với n=11.

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

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

www.VIETMATHS.com

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



BÀI TẬP CHƯƠNG IV:

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

a) $\mathbf{K_n}$, **b)** $\mathbf{C_n}$, **c)** $\mathbf{W_n}$, **d)** $\mathbf{Q_n}$.

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

a) chu trình Euler ? **b)** đường đi Euler ?

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

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

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

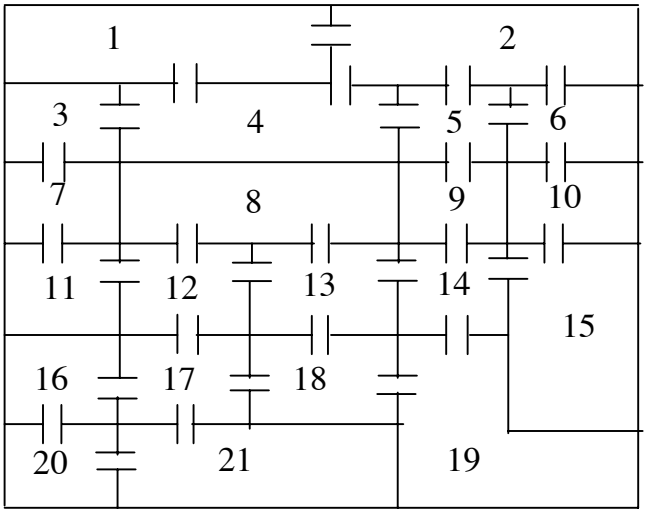
6. Hiệu trưởng mời 2n ($n \geq 2$) sinh viên giỏi đến dự tiệc. Mỗi sinh viên giỏi quen ít nhất n sinh viên giỏi khác đến dự tiệc. Chứng minh rằng luôn luôn có thể xếp tất cả các sinh viên giỏi ngồi xung quanh một bàn tròn, để mỗi người ngồi giữa hai người mà sinh viên đó quen.

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

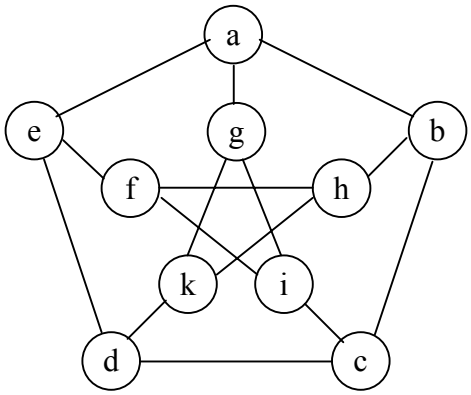
Hãy tìm nơi giấu báu vật

www.VIETMATHS.com

www.VIETMATHS.com

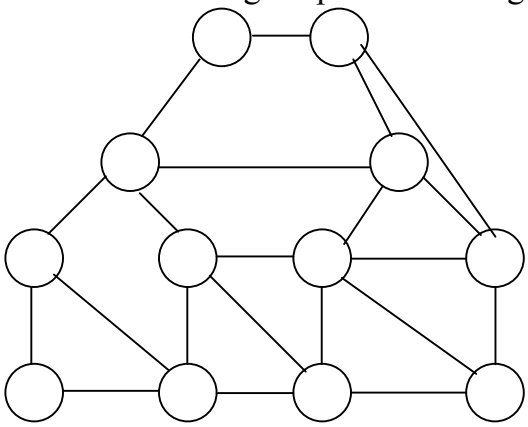


8. Đồ thị cho trong hình sau gọi là đồ thị Peterson P.

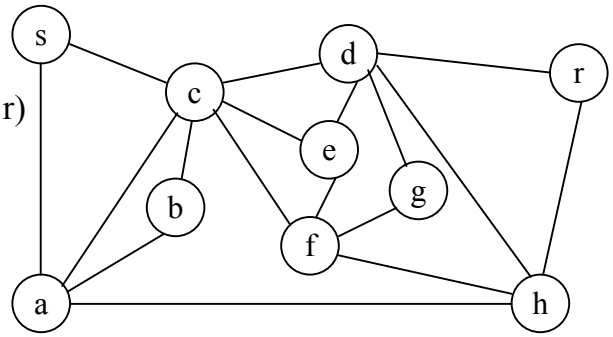


a) Tìm một đường đi Hamilton trong P.
b) Chứng minh rằng P \ {v}, với v là một đỉnh bất kỳ của P, là một đồ thị Hamilton.

9. Giải bài toán người phát thư Trung Hoa với đồ thị cho trong hình sau:



10. Chứng minh rằng đồ thị G cho trong hình sau có đường đi Hamilton (từ s đến r) nhưng không có chu trình Hamilton.



www.VIETMATHS.com

11. Cho thí dụ về:

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

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

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

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

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

 www.VIETMATHS.com

CHƯƠNG V
MỘT SỐ BÀI TOÁN TỐI ƯU TRÊN ĐỒ THỊ

5.1. ĐỒ THỊ CÓ TRỌNG SỐ VÀ BÀI TOÁN ĐƯỜNG ĐI NGẮN NHẤT.

5.1.1. Mở đầu:

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

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

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

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

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

5.1.2. Bài toán tìm đường đi ngắn nhất:

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

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

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

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

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

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

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

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

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

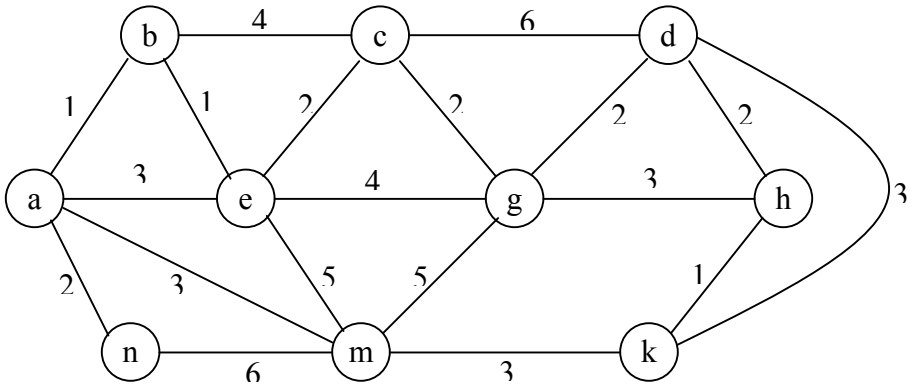
5.1.3. Thuật toán Dijkstra:

---

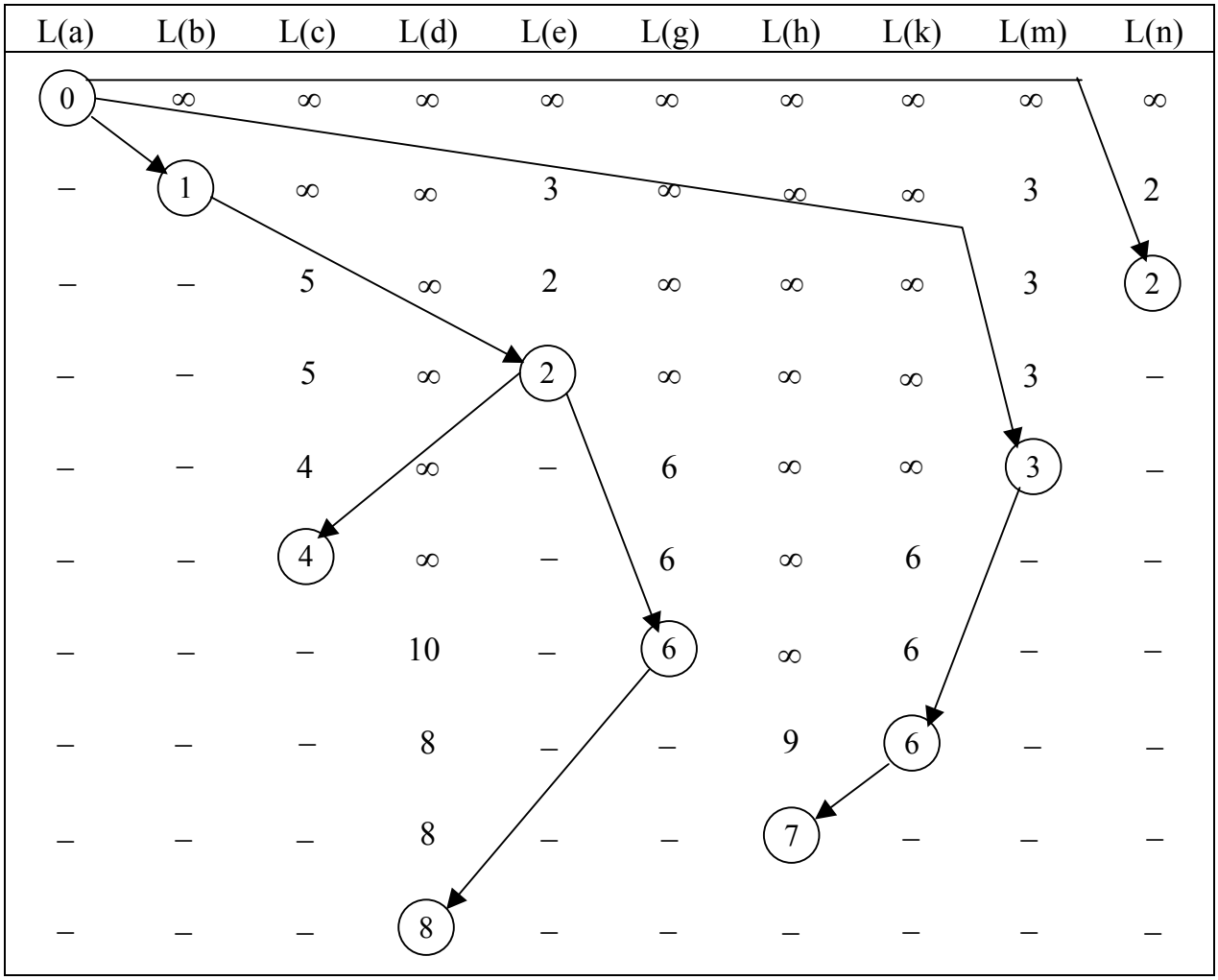
procedure *Dijkstra* (G=(V,E) là đơn đồ thị liên thông, có trọng số với trọng số dương)
{G có các đỉnh $a=u_0, u_1, ..., u_n=z$ và trọng số $m(u_i,u_j)$, với $m(u_i,u_j) = \infty$ nếu (u_i,u_j) không là một cạnh trong G}

for i := 1 **to** n
$L(u_i) := \infty$
L(a) := 0
S := V \ {a}
u := a
while $S \neq \varnothing$
begin
for tất cả các đỉnh v thuộc S
if L(u) +m(u,v) < L(v) **then** L(v) := L(u)+m(u,v)
u := đỉnh thuộc S có nhãn L(u) nhỏ nhất
{L(u): độ dài đường đi ngắn nhất từ a đến u}
S := S \ {u}
end

Thí dụ 1: Tìm khoảng cách d(a,v) từ a đến mọi đỉnh v và tìm đường đi ngắn nhất từ a đến v cho trong đồ thị G sau.



www.VIETMATHS.com



5.1.4. Định lý: Thuật toán Dijkstra tìm được đường đi ngắn nhất từ một đỉnh cho trước đến một đỉnh tuỳ ý trong đơn đồ thị vô hướng liên thông có trọng số.

Chứng minh: Định lý được chứng minh bằng quy nạp. Tại bước k ta có giả thiết quy nạp là:

(i) Nhãn của đỉnh v không thuộc S là độ dài của đường đi ngắn nhất từ đỉnh a tới đỉnh này;

(ii) Nhãn của đỉnh v trong S là độ dài của đường đi ngắn nhất từ đỉnh a tới đỉnh này và đường đi này chỉ chứa các đỉnh (ngoài chính đỉnh này) không thuộc S.

Khi k=0, tức là khi chưa có bước lặp nào được thực hiện, S=V \ {a}, vì thế độ dài của đường đi ngắn nhất từ a tới các đỉnh khác a là ∞ và độ dài của đường đi ngắn nhất từ a tới chính nó bằng 0 (ở đây, chúng ta cho phép đường đi không có cạnh). Do đó bước cơ sở là đúng.

Giả sử giả thiết quy nạp là đúng với bước k. Gọi v là đỉnh lấy ra khỏi S ở bước lặp k+1, vì vậy v là đỉnh thuộc S ở cuối bước k có nhãn nhỏ nhất (nếu có nhiều đỉnh có nhãn nhỏ nhất thì có thể chọn một đỉnh nào đó làm v). Từ giả thiết quy nạp ta thấy rằng

www.VIETMATHS.com

trước khi vào vòng lặp thứ k+1, các đỉnh không thuộc S đã được gán nhãn bằng độ dài của đường đi ngắn nhất từ a. Đỉnh v cũng vậy phải được gán nhãn bằng độ dài của đường đi ngắn nhất từ a. Nếu điều này không xảy ra thì ở cuối bước lặp thứ k sẽ có đường đi với độ dài nhỏ hơn $L_k(v)$ chứa cả đỉnh thuộc S (vì $L_k(v)$ là độ dài của đường đi ngắn nhất từ a tới v chứa chỉ các đỉnh không thuộc S sau bước lặp thứ k). Gọi u là đỉnh đầu tiên của đường đi này thuộc S. Đó là đường đi với độ dài nhỏ hơn $L_k(v)$ từ a tới u chứa chỉ các đỉnh không thuộc S. Điều này trái với cách chọn v. Do đó (i) vẫn còn đúng ở cuối bước lặp k+1.

Gọi u là đỉnh thuộc S sau bước k+1. Đường đi ngắn nhất từ a tới u chứa chỉ các đỉnh không thuộc S sẽ hoặc là chứa v hoặc là không. Nếu nó không chứa v thì theo giả thiết quy nạp độ dài của nó là $L_k(v)$. Nếu nó chứa v thì nó sẽ tạo thành đường đi từ a tới v với độ dài có thể ngắn nhất và chứa chỉ các đỉnh không thuộc S khác v, kết thúc bằng cạnh từ v tới u. Khi đó độ dài của nó sẽ là $L_k(v)+m(v,u)$. Điều đó chứng tỏ (ii) là đúng vì $L_{k+1}(u)=\min(L_k(u), L_k(v)+m(v,u))$.

5.1.5. Mệnh đề: Thuật toán Dijkstra tìm đường đi ngắn nhất từ một đỉnh cho trước đến một đỉnh tuỳ ý trong đơn đồ thị vô hướng liên thông có trọng số có độ phức tạp là $O(n^2)$.

Chứng minh: Thuật toán dùng không quá n−1 bước lặp. Trong mỗi bước lặp, dùng không hơn 2(n−1) phép cộng và phép so sánh để sửa đổi nhãn của các đỉnh. Ngoài ra, một đỉnh thuộc S_k có nhãn nhỏ nhất nhờ không quá n−1 phép so sánh. Do đó thuật toán có độ phức tạp $O(n^2)$.

5.1.6. Thuật toán Floyd:

Cho G=(V,E) là một đồ thị có hướng, có trọng số. Để tìm đường đi ngắn nhất giữa mọi cặp đỉnh của G, ta có thể áp dụng thuật toán Dijkstra nhiều lần hoặc áp dụng thuật toán Floyd được trình bày dưới đây.

Giả sử $V=\{v_1, v_2, ..., v_n\}$ và có ma trận trọng số là $W \equiv W_0$. Thuật toán Floyd xây dựng dãy các ma trận vuông cấp n là W_k ($0 \le k \le n$) như sau:

```
procedure Xác định W_n
    for i := 1 to n
        for j := 1 to n
            W[i,j] := m(v_i,v_j)  {W[i,j] là phần tử dòng i cột j của ma trận W_0}
            for k := 1 to n
                if W[i,k] +W[k,j] < W[i,j] then W[i,j] := W[i,k] +W[k,j]
                {W[i,j] là phần tử dòng i cột j của ma trận W_k}
```

5.1.7. Định lý: Thuật toán Floyd cho ta ma trận $W^*=W_n$ là ma trận khoảng cách nhỏ nhất của đồ thị G.

Chứng minh: Ta chứng minh bằng quy nạp theo k mệnh đề sau:

$W_k[i,j]$ là chiều dài đường đi ngắn nhất trong những đường đi nối đỉnh v_i với đỉnh v_j đi qua các đỉnh trung gian trong $\{v_1, v_2, ..., v_k\}$.

Trước hết mệnh đề hiển nhiên đúng với k=0.

Giả sử mệnh đề đúng với k-1.

Xét $W_k[i,j]$. Có hai trường hợp:

1) Trong các đường đi chiều dài ngắn nhất nối v_i với v_j và đi qua các đỉnh trung gian trong $\{v_1, v_2, ..., v_k\}$, có một đường đi γ sao cho $v_k \notin \gamma$. Khi đó γ cũng là đường đi ngắn nhất nối v_i với v_j đi qua các đỉnh trung gian trong $\{v_1, v_2, ..., v_{k-1}\}$, nên theo giả thiết quy nạp,

$$W_{k-1}[i,j] = \text{chiều dài } \gamma \le W_{k-1}[i,k]+W_{k-1}[k,j].$$

Do đó theo định nghĩa của W_k thì $W_k[i,j]=W_{k-1}[i,j]$.

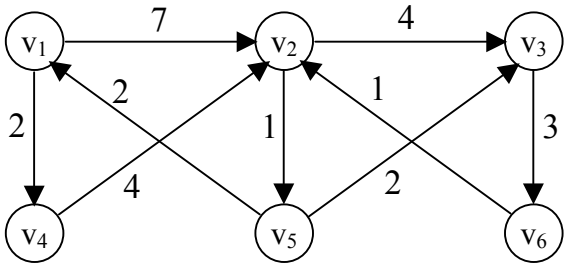
2) Mọi đường đi chiều dài ngắn nhất nối v_i với v_j và đi qua các đỉnh trung gian trong $\{v_1, v_2, ..., v_k\}$, đều chứa v_k. Gọi $\gamma = v_i ... v_k ... v_j$ là một đường đi ngắn nhất như thế thì $v_1 ... v_k$ và $v_k ... v_j$ cũng là những đường đi ngắn nhất đi qua các đỉnh trung gian trong $\{v_1, v_2, ..., v_{k-1}\}$ và

$$W_{k-1}[i,k]+W_{k-1}[k,j] = \text{chiều dài}(v_1 ... v_k) + \text{chiều dài}(v_k ... v_j)$$
$$= \text{chiều dài } \gamma < W_{k-1}[i,j].$$

Do đó theo định nghĩa của W_k thì ta có:

$$W_k[i,j] = W_{k-1}[i,k]+W_{k-1}[k,j] .$$

Thí dụ 2: Xét đồ thị G sau:



Áp dụng thuật toán Floyd, ta tìm được (các ô trống là ∞)

$$W = W_0 = \begin{pmatrix} & 7 & 2 & \\ & & 4 & 1 \\ & & & 3 \\ & 4 & & \\ 2 & & 2 & \\ & 1 & & \end{pmatrix}$$

$$W_1 = \begin{pmatrix} & 7 & & 2 & & \\ & & 4 & & 1 & \\ & & & & & 3 \\ & 4 & & & & \\ 2 & 9 & & 2 & 4 & \\ & 1 & & & & \end{pmatrix}, \quad W_2 = \begin{pmatrix} & 7 & 11 & 2 & 8 & \\ & & 4 & & 1 & \\ & & & & & 3 \\ & 4 & 8 & & 5 & \\ 2 & 9 & 2 & 4 & 10 & \\ & 1 & 5 & & 2 & \end{pmatrix}$$

$$W_3 = \begin{pmatrix} & 7 & 11 & 2 & 8 & 14 \\ & & 4 & & 1 & 7 \\ & & & & & 3 \\ & 4 & 8 & & 5 & 11 \\ 2 & 9 & 2 & 4 & 10 & 5 \\ & 1 & 5 & & 2 & 8 \end{pmatrix}, \quad W_4 = \begin{pmatrix} & 6 & 10 & 2 & 7 & 13 \\ & & 4 & & 1 & 7 \\ & & & & & 3 \\ & 4 & 8 & & 5 & 11 \\ 2 & 8 & 2 & 4 & 9 & 5 \\ & 1 & 5 & & 2 & 8 \end{pmatrix}$$

$$W_5 = \begin{pmatrix} 9 & 6 & 9 & 2 & 7 & 12 \\ 3 & 9 & 3 & 5 & 1 & 6 \\ & & & & & 3 \\ 7 & 4 & 7 & 9 & 5 & 10 \\ 2 & 8 & 2 & 4 & 9 & 5 \\ 4 & 1 & 4 & 6 & 2 & 7 \end{pmatrix}, \quad W^* = W_6 = \begin{pmatrix} 9 & 6 & 9 & 2 & 7 & 12 \\ 3 & 7 & 3 & 5 & 1 & 6 \\ 7 & 4 & 7 & 9 & 5 & 3 \\ 7 & 4 & 7 & 9 & 5 & 10 \\ 2 & 6 & 2 & 4 & 7 & 5 \\ 4 & 1 & 4 & 6 & 2 & 7 \end{pmatrix}.$$

Thuật toán Floyd có thể áp dụng cho đồ thị vô hướng cũng như đồ thị có hướng. Ta chỉ cần thay mỗi cạnh vô hướng (u,v) bằng một cặp cạnh có hướng (u,v) và (v,u) với m(u,v)=m(v,u). Tuy nhiên, trong trường hợp này, các phần tử trên đường chéo của ma trận W cần đặt bằng 0.

Đồ thị có hướng G là liên thông mạnh khi và chỉ khi mọi phần tử nằm trên đường chéo trong ma trận trọng số ngắn nhất W* đều hữu hạn.

5.2. BÀI TOÁN LUỒNG CỰC ĐẠI.

5.2.1. Luồng vận tải:

5.2.1.1. Định nghĩa: Mạng vận tải là một đồ thị có hướng, không có khuyên và có trọng số G=(V,E) với V={v_0, v_1, ..., v_n} thoả mãn:

1) Mỗi cung e ∈ E có trọng số m(e) là một số nguyên không âm và được gọi là khả năng thông qua của cung e.

2) Có một và chỉ một đỉnh v_0 không có cung đi vào, tức là $deg_t(v_0)=0$. Đỉnh v_0 được gọi là lối vào hay đỉnh phát của mạng.

3) Có một và chỉ một đỉnh v_n không có cung đi ra, tức là $deg_o(v_n)=0$. Đỉnh v_n được gọi là lối ra hay đỉnh thu của mạng.

5.2.1.2. Định nghĩa: Để định lượng khai thác, tức là xác định lượng vật chất chuyển qua mạng vận tải G=(V,E), người ta đưa ra khái niệm luồng vận tải và nó được định nghĩa như sau.

Hàm φ xác định trên tập cung E và nhận giá trị nguyên được gọi là luồng vận tải của mạng vận tải G nếu φ thoả mãn:

1) $\varphi(e) \geq 0, \forall e \in E$.

2) $\sum_{e\in\Gamma^-(v)} \varphi(e) = \sum_{e\in\Gamma^+(v)} \varphi(e)$, $\forall v \in V$, $v \neq v_0$, $v \neq v_n$. Ở đây, $\Gamma^-(v)=\{e\in E \mid e \text{ có đỉnh cuối là } v\}$, $\Gamma^+(v)=\{e\in E \mid e \text{ có đỉnh đầu là } v\}$.

3) $\varphi(e) \leq m(e), \forall e \in E$.

Ta xem φ(e) như là lượng hàng chuyển trên cung e=(u,v) từ đỉnh u đến đỉnh v và không vượt quá khả năng thông qua của cung này. Ngoài ra, từ điều kiện 2) ta thấy rằng nếu v không phải là lối vào v_0 hay lối ra v_n, thì lượng hàng chuyển tới v bằng lượng hàng chuyển khỏi v.

Từ quan hệ 2) suy ra:

4) $\sum_{e\in\Gamma^+(v_0)} \varphi(e) = \sum_{e\in\Gamma^-(v_n)} \varphi(e) =: \varphi_{v_n}$.

Đại lượng φ_{v_n} (ta còn ký hiệu là φ_n) được gọi là luồng qua mạng, hay cường độ luồng tại điểm v_n hay giá trị của luồng φ. Bài toán đặt ra ở đây là tìm φ để φ_{v_n} đạt giá trị lớn nhất, tức là tìm giá trị lớn nhất của luồng.

5.2.1.3. Định nghĩa: Cho mạng vận tải G=(V,E) và $A \subset V$. Ký hiệu

$$\Gamma^-(A)=\{(u,v)\in E \mid v\in A, u\notin A\},\ \Gamma^+(A)=\{(u,v)\in E \mid u\in A, v\notin A\}.$$

Đối với tập cung M tuỳ ý, đại lượng $\varphi(M)=\sum_{e\in M}\varphi(e)$ được gọi là luồng của tập cung M.

Từ điều kiện 2) dễ dàng suy ra hệ quả sau.

5.2.1.4. Hệ quả: Cho φ là luồng của mạng vận tải G=(V,E) và $A \subset V \setminus \{v_0,v_n\}$. Khi đó:

$$\varphi(\Gamma^-(A))=\varphi(\Gamma^+(A)).$$

5.2.2. Bài toán luồng cực đại:

Cho mạng vận tải G=(V,E). Hãy tìm luồng φ để đạt φ_{v_n} max trên mạng G.

Nguyên lý của các thuật toán giải bài toán tìm luồng cực đại là như sau.

5.2.2.1. Định nghĩa: Cho $A \subset V$ là tập con tuỳ ý không chứa lối vào v_0 và chứa lối ra v_n. Tập $\Gamma^-(A)$ được gọi là một thiết diện của mạng vận tải G.

Đại lượng $m(\Gamma^-(A))=\sum_{e\in\Gamma^-(A)} m(e)$ được gọi là khả năng thông qua của thiết diện $\Gamma^-(A)$.

Từ định nghĩa thiết diện và khả năng thông qua của nó ta nhận thấy rằng: mỗi đơn vị hàng hoá được chuyển từ v_0 đến v_n ít nhất cũng phải một lần qua một cung nào đó của thiết diện $\Gamma^-(A)$. Vì vậy, dù luồng φ và thiết diện $\Gamma^-(A)$ như thế nào đi nữa cũng vẫn thoả mãn quan hệ:

$$\varphi_n \le m(\Gamma^-(A)).$$

Do đó, nếu đối với luồng φ và thiết diện W mà có:

$$\varphi_n = m(W)$$

thì chắc chắn rằng luồng φ đạt giá trị lớn nhất và thiết diện W có khả năng thông qua nhỏ nhất.

5.2.2.2. Định nghĩa: Cung e trong mạng vận tải G với luồng vận tải φ được goi là cung bão hoà nếu $\varphi(e)=m(e)$.

Luồng φ của mạng vận tải G được gọi là luồng đầy nếu mỗi đường đi từ v_0 đến v_n đều chứa ít nhất một cung bão hoà.

Từ định nghĩa trên ta thấy rằng, nếu luồng φ trong mạng vận tải G chưa đầy thì nhất định tìm được đường đi α từ lối vào v_0 đến lối ra v_n không chứa cung bão hoà. Khi đó ta nâng luồng φ thành φ' như sau:

$$\varphi'(e) = \begin{cases} \varphi(e)+1 \ khi \ e\in\alpha, \\ \varphi(e) \quad khi \ e\notin\alpha. \end{cases}$$

Khi đó φ' cũng là một luồng, mà giá trị của nó là:

$$\varphi'_n = \varphi_n + 1 > \varphi_n.$$

Như vậy, đối với mỗi luồng không đầy ta có thể nâng giá trị của nó và nâng cho tới khi nhận được một luồng đầy.

Tuy vậy, thực tế cho thấy rằng có thể có một luồng đầy, nhưng vẫn chưa đạt tới giá trị cực đại. Bởi vậy, cần phải dùng thuật toán Ford-Fulkerson để tìm giá trị cực đại của luồng.

5.2.2.3. Thuật toán Ford-Fulkerson:

Để tìm luồng cực đại của mạng vận tải G, ta xuất phát từ luồng tuỳ ý φ của G, rồi nâng luồng lên đầy, sau đó áp dụng thuật toán Ford-Fulkerson hoặc ta có thể áp dụng thuật toán Ford-Fulkerson trực tiếp đối với luồng φ.

Thuật toán gồm 3 bước:

Bước 1 (đánh dấu ở đỉnh của mạng): Lối vào v_0 được đánh dấu bằng 0.

1) Nếu đỉnh v_i đã được đánh dấu thì ta dùng chỉ số +i để đánh dấu cho mọi đỉnh y chưa được đánh dấu mà $(v_i,y)\in E$ và cung này chưa bão hoà ($\varphi(v_i,y)<m(v_i,y)$).

2) Nếu đỉnh v_i đã được đánh dấu thì ta dùng chỉ số –i để đánh dấu cho mọi đỉnh z chưa được đánh dấu mà $(z,v_i)\in E$ và luồng của cung này dương ($\varphi(z,v_i)>0$).

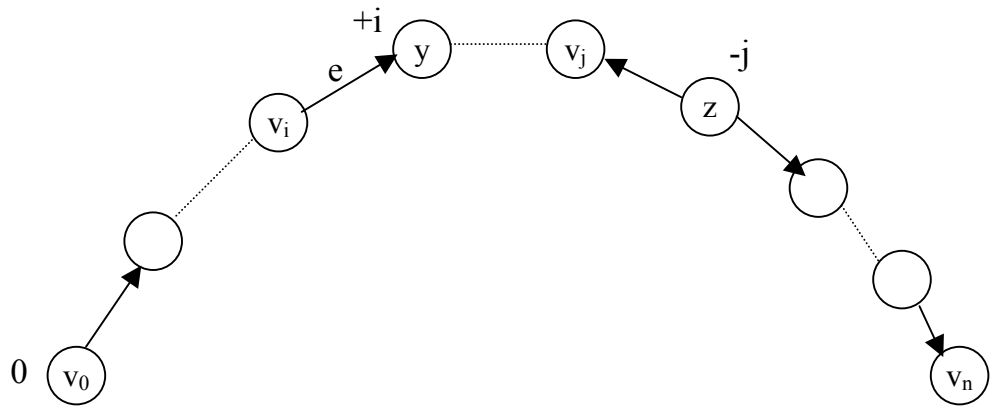
Nếu với phương pháp này ta đánh dấu được tới lối ra v_n thì trong G tồn tại giữa v_0 và v_n một xích α, mọi đỉnh đều khác nhau và được đánh dấu theo chỉ số của đỉnh liền trước nó (chỉ sai khác nhau về dấu). Khi đó chắc chắn ta nâng được giá trị của luồng.

Bước 2 (nâng giá trị của luồng): Để nâng giá trị của luồng φ, ta đặt:

$\varphi'(e) = \varphi(e)$, nếu $e \notin \alpha$,

$\varphi'(e) = \varphi(e)+1$, nếu $e \in \alpha$ được định hướng theo chiều của xích α đi từ v_o đến v_n,

$\varphi'(e) = \varphi(e)-1$, nếu $e \in \alpha$ được định hướng ngược với chiều của xích α đi từ v_o đến v_n.



φ' thoả mãn các điều kiện về luồng, nên φ' là một luồng và ta có:

$$\varphi'_n = \varphi_n + 1.$$

Như vậy, ta đã nâng được luồng lên một đơn vị.

Sau đó lặp lại một vòng mới. Vì khả năng thông qua của các cung đều hữu hạn, nên quá trình phải dừng lại sau một số hữu hạn bước.

Bước 3: Nếu với luồng φ^0 bằng phương pháp trên ta không thể nâng giá trị của luồng lên nữa, nghĩa là ta không thể đánh dấu được đỉnh v_n, thì ta nói rằng quá trình nâng luồng kết thúc và φ^0 đã đạt giá trị cực đại, đồng thời gọi φ^0 là luồng kết thúc.

Khi mạng vận tải G=(V,E) đạt tới luồng φ^0, thì bước tiếp theo ta không thể đánh dấu được tới lối ra v_n. Trên cơ sở hiện trạng được đánh dấu tại bước này, ta sẽ chứng minh rằng luồng φ^0 đã đạt được giá trị cực đại.

5.2.2.4. Bổ đề: Cho luồng φ của mạng vận tải G=(V,E) và $A \subset V$, chứa lối ra v_n và không chứa lối vào v_0. Khi đó:

$$\varphi_{v_n} = \varphi(\Gamma^-(A)) - \varphi(\Gamma^+(A)).$$

Chứng minh: Đặt $A_1 = A \setminus \{v_n\}$. Theo Hệ quả 5.2.1.4, ta có:

$$\varphi(\Gamma^-(A_1)) = \varphi(\Gamma^+(A_1)) \quad (1).$$

Đặt $C_1 = \{(a,v_n) \in E \mid a \notin A\}$. Khi đó $\Gamma^-(A) = \Gamma^-(A_1) \cup C_1$ và $\Gamma^-(A_1) \cap C_1 = \varnothing$, nên

$$\varphi(\Gamma^-(A)) = \varphi(\Gamma^-(A_1)) + \varphi(C_1) \quad (2).$$

Đặt $C_2 = \{(b,v_n) \in E \mid b \in A_1\}$. Khi đó $C_2 = \{(b,v_n) \in E \mid b \in A\}$, $\Gamma^+(A_1) = \Gamma^+(A) \cup C_2$ và $\Gamma^+(A) \cap C_2 = \varnothing$, nên

$$\varphi(\Gamma^+(A)) = \varphi(\Gamma^+(A_1)) - \varphi(C_2) \quad (3).$$

Ngoài ra, $\Gamma^-(v_n) = C_1 \cup C_2$ và $C_1 \cap C_2 = \varnothing$, nên

$$\varphi_{v_n} = \varphi(\Gamma^-(v_n)) = \varphi(C_1) + \varphi(C_2) \quad (4).$$

Từ (1), (2), (3) và (4), ta có:

$$\varphi_{v_n} = \varphi(\Gamma^-(A)) - \varphi(\Gamma^+(A)).$$

5.2.2.5. Định lý (Ford-Fulkerson): Trong mạng vận tải G=(V,E), giá trị lớn nhất của luồng bằng khả năng thông qua nhỏ nhất của thiết diện, nghĩa là

$$\max_{\varphi} \varphi_{v_n} = \min_{A \subset V, v_0 \notin A, v_n \in A} m(\Gamma^-(A)).$$

Chứng minh: Giả sử trong mạng vận tải G, φ^0 là luồng cuối cùng, mà sau đó bằng phương pháp đánh dấu của thuật toán Ford-Fulkerson không đạt tới lối ra v_n. Trên cơ sở hiện trạng được đánh dấu lần cuối cùng này, ta dùng B để ký hiệu tập gồm các đỉnh của G không được đánh dấu. Khi đó $v_0 \notin B$, $v_n \in B$. Do đó $\Gamma^-(B)$ là một thiết diện của mạng vận tải G và theo Bổ đề 5.2.2.4, ta có:

$$\varphi^0_{v_n} = \varphi^0(\Gamma^-(B)) - \varphi^0(\Gamma^+(B)) \quad (1).$$

Đối với mỗi cung $e=(u,v) \in \Gamma^-(B)$ thì $u \notin B$ và $v \in B$, tức là u được đánh dấu và v không được đánh dấu, nên theo nguyên tắc đánh dấu thứ nhất, e đã là cung bão hoà:

$$\varphi^0(e) = m(e).$$

Do đó,
$$\varphi^0(\Gamma^-(B)) = \sum_{e \in \Gamma^-(B)} \varphi^0(e) = \sum_{e \in \Gamma^-(B)} m(e) = m(\Gamma^-(B)) \quad (2).$$

Đối với mỗi cung $e=(s,t) \in \Gamma^+(B)$ thì $s \in B$ và $t \notin B$, tức là s không được đánh dấu và t được đánh dấu, nên theo nguyên tắc đánh dấu thứ hai:

$$\varphi^0(e) = 0.$$

Do đó,
$$\varphi^0(\Gamma^+(B)) = \sum_{e \in \Gamma^+(B)} \varphi^0(e) = 0 \quad (3).$$

Từ (1), (2) và (3) ta suy ra:

$$\varphi^0_{v_n} = m(\Gamma^-(B)).$$

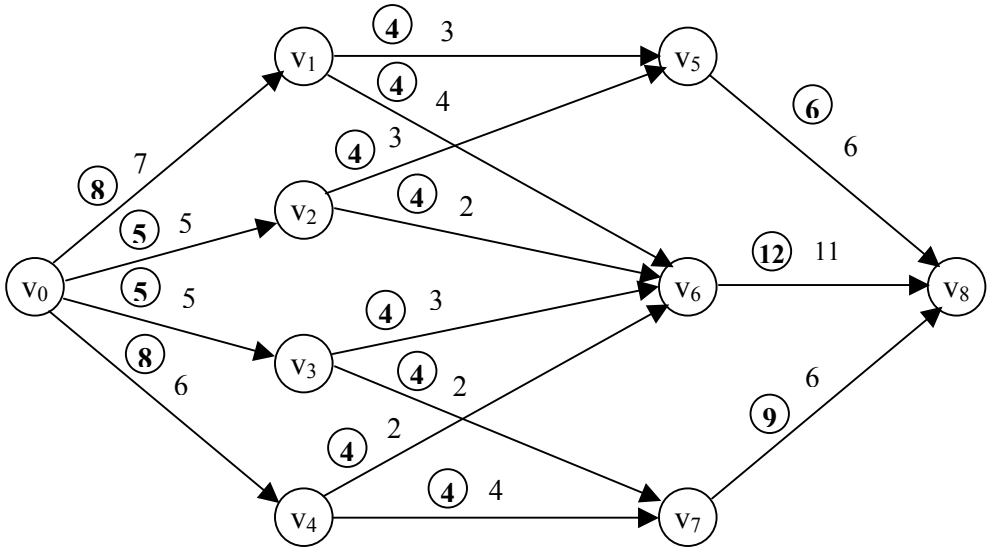
Vì vậy, $\varphi^0_{v_n}$ là giá trị lớn nhất của luồng đạt được, còn $m(\Gamma^-(B))$ là giá trị nhỏ nhất trong các khả năng thông qua của các thiết diện thuộc mạng vận tải G.

Thí dụ 3: Cho mạng vận tải như hình dưới đây với khả năng thông qua được đặt trong khuyên tròn, luồng được ghi trên các cung. Tìm luồng cực đại của mạng này.

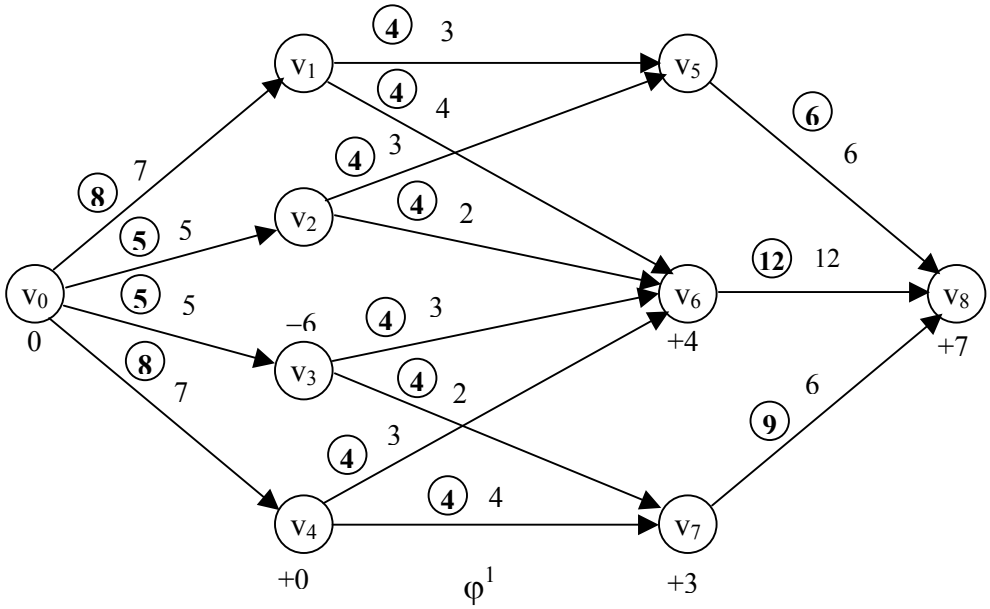
Luồng φ có đường đi (v_0,v_4), (v_4,v_6), (v_6,v_8) gồm các cung chưa bão hoà nên nó chưa đầy. Do đó có thể nâng luồng của các cung này lên một đơn vị, để được φ^1.

Do mỗi đường xuất phát từ v_0 đến v_8 đều chứa ít nhất một cung bão hoà, nên luồng φ^1 là luồng đầy. Song nó chưa phải là luồng cực đại.

Áp dụng thuật toán Ford-Fulkerson để nâng luồng φ^1.

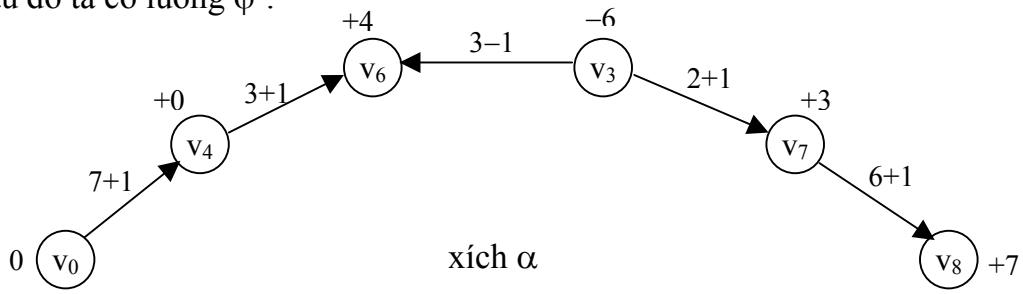


φ



φ^1

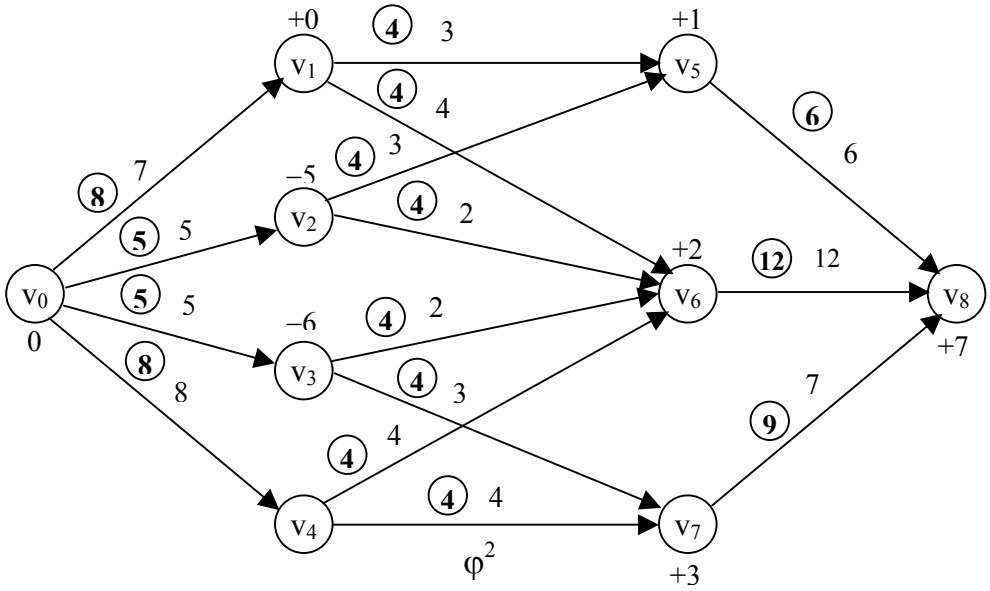
Xét xích $\alpha=(v_0, v_4, v_6, v_3, v_7, v_8)$. Quá trình đánh dấu từ v_0 đến v_8 để có thể nâng luồng φ^1 lên một đơn vị bằng cách biến đổi luồng tại các cung thuộc xích α được đánh dấu. Sau đó ta có luồng φ^2.



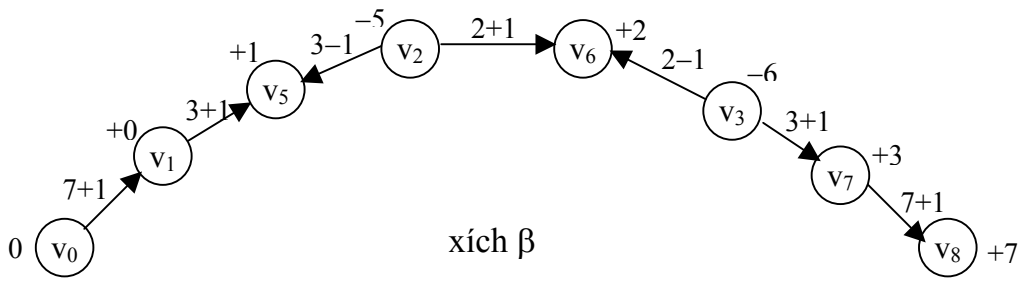
xích α

Xét xích $\beta=(v_0, v_1, v_5, v_2, v_6, v_3, v_7, v_8)$. Quá trình đánh dấu từ v_0 đến v_8 để có thể nâng luồng φ^2 lên một đơn vị bằng cách biến đổi luồng tại các cung thuộc xích β được đánh dấu. Sau đó ta có luồng φ^3.

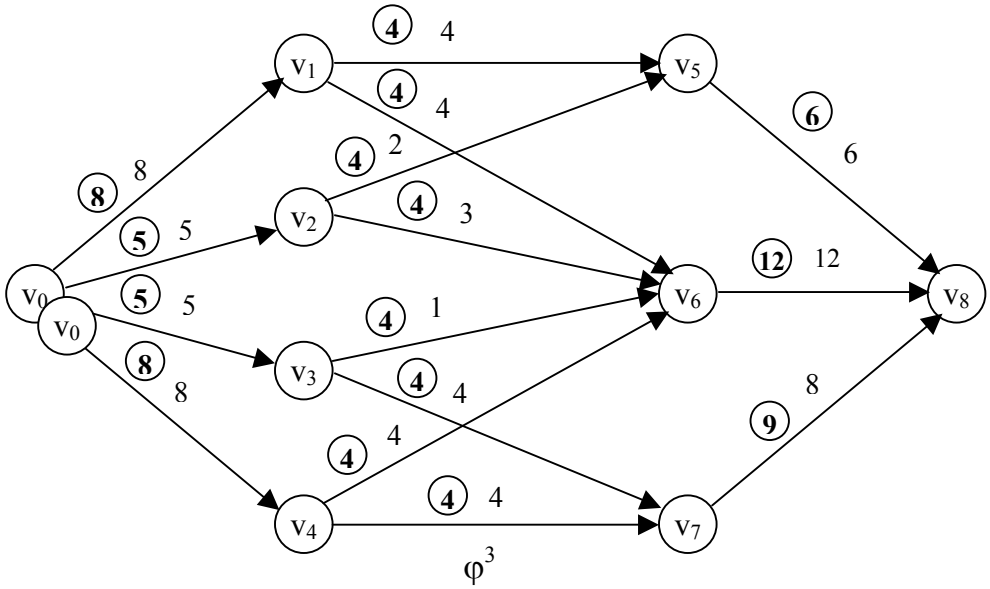
www.VIETMATHS.com



φ^2



xích β



φ^3

Tiếp theo ta chỉ có thể đánh dấu được đỉnh v_0 nên quá trình nâng luồng kết thúc và ta được giá trị của luồng cực đại là:

$$\varphi^3_{v_8} = 6+12+8 = 26.$$

Mặt khác, thiết diện nhỏ nhất $\Gamma^-(B)$ với $B=\{v_1, v_2, ..., v_8\}$ là

$$\Gamma^-(B)=\{(v_0,v_1), (v_0,v_2), (v_0,v_3), (v_0,v_4)\}.$$

www.VIETMATHS.com

5.3. BÀI TOÁN DU LỊCH.

5.3.1. Giới thiệu bài toán:

Một người xuất phát từ một thành phố nào đó muốn tới thăm n–1 thành phố khác, mỗi thành phố đúng một lần, rồi quay về thành phố ban đầu. Hỏi nên đi theo trình tự nào để độ dài tổng cộng các đoạn đường đi qua là ngắn nhất (khoảng cách giữa hai thành phố có thể hiểu là cự ly thông thường hoặc thời gian cần đi hoặc chi phí của hành trình, ... và xem như cho trước).

Xét đồ thị đầy đủ G=(V,E), với V={1, 2, ..., n}, có trọng số với trọng số m_{ij}= m(i,j) có thể khác m_{ji} = m(j,i). Như vậy, ta có thể xem G như là một đồ thị có hướng đầy đủ "mạnh" theo nghĩa với mọi i, j=1, 2, ..., n, i≠j, luôn có (i,j), (j,i)∈E. Bài toán trở thành tìm chu trình Hamilton có độ dài ngắn nhất trong G.

Bài toán nổi tiếng này đã có lời giải bằng cách sử dụng phương pháp "nhánh và cận".

5.3.2. Phương pháp nhánh và cận: Giả sử trong một tập hữu hạn các phương án của bài toán, ta phải chọn ra được một phương án tối ưu theo một tiêu chuẩn nào đó (thí dụ làm cho hàm mục tiêu đạt giá trị nhỏ nhất). Ta sẽ tìm cách phân chia tập phương án đang xét thành hai tập con không giao nhau. Với mỗi tập này, ta sẽ tính "cận dưới" (chặn dưới đủ tốt) của các giá trị hàm mục tiêu ứng với các phương án trong đó. Mang so sánh hai cận dưới vừa tính được, ta có thể phán đoán xem tập con nào có nhiều triển vọng chứa phương án tối ưu và tiếp tục phân chia tập con đó thành hai tập con khác không giao nhau, lại tính các cận dưới tương ứng ... Lặp lại quá trình này thì sau một số hữu hạn bước, cuối cùng sẽ được một phương án tốt, nói chung là tối ưu. Nếu không thì lặp lại quá trình phân chia để kiểm tra và sau một vài bước, ta sẽ được phương án tối ưu.

Người ta thường mô tả quá trình phân chia đó bằng một "cây có gốc" mà gốc sẽ tượng trưng cho tập toàn bộ các phương án, còn các đỉnh ở phía dưới lần lượt tượng trưng cho các tập con trong quá trình "phân nhánh nhị phân". Vì vậy, phương pháp này mang tên nhánh và cận.

5.3.3. Cơ sở lý luận của phép toán: Nếu không xác định thành phố xuất phát thì có n! hành trình, mỗi hành trình ứng với một hoán vị nào đó của tập {1, 2, ..., n}. Còn nếu cho trước thành phố xuất phát thì có tất cả là (n–1)! hành trình.

Giả sử h=(π(1), π(2), ..., π(n), π(1)) (π là một hoán vị) là một hành trình qua các thành phố π(1), ..., π(n) theo thứ tự đó rồi quay về π(1) thì hàm mục tiêu

$$f(h) = m_{\pi(1)\pi(2)} + \cdots + m_{\pi(n-1)\pi(n)} + m_{\pi(n)\pi(1)} = \sum_{(i,j)\in h} m_{ij},$$

sẽ biểu thị tổng độ dài đã đi theo hành trình h, trong đó (i,j) ký hiệu một chặng đường của hành trình, tức là một cặp thành phố kề nhau theo hành trình h.

5.3.4. Ma trận rút gọn: Quá trình tính toán sẽ được thực hiện trên các ma trận suy từ ma trận trọng số M=(m_{ij}) ban đầu bằng những phép biến đổi rút gọn để các số liệu được đơn giản.

Phép trừ phần tử nhỏ nhất của mỗi dòng (t.ư. cột) vào tất cả các phần tử của dòng (t.ư. cột) đó được gọi là phép rút gọn dòng (t.ư. cột). Phần tử nhỏ nhất đó được gọi là hằng số rút gọn dòng (t.ư. cột) đang xét. Ma trận với các phần tử không âm và có ít nhất một phần tử bằng 0 trên mỗi dòng và mỗi cột được gọi là ma trận rút gọn của ma trận ban đầu.

Thí dụ 4:

$$M = \begin{pmatrix} 4 & 3 & 5 \\ 6 & 2 & 7 \\ 9 & 10 & 5 \end{pmatrix} \begin{matrix} 3 \\ 2 \\ 5 \end{matrix} \longrightarrow \begin{pmatrix} 1 & 0 & 2 \\ 4 & 0 & 5 \\ 4 & 5 & 0 \end{pmatrix} \longrightarrow M' = \begin{pmatrix} 0 & 0 & 2 \\ 3 & 0 & 5 \\ 3 & 5 & 0 \end{pmatrix},$$
$$1 \quad 0 \quad 0$$

tất nhiên có thể rút gọn cách khác

$$M = \begin{pmatrix} 4 & 3 & 5 \\ 6 & 2 & 7 \\ 9 & 10 & 5 \end{pmatrix} \begin{matrix} 0 \\ 0 \\ 0 \end{matrix} \longrightarrow M'' = \begin{pmatrix} 0 & 1 & 0 \\ 2 & 0 & 2 \\ 5 & 8 & 0 \end{pmatrix}.$$
$$4 \quad 2 \quad 5$$

5.3.5. Mệnh đề: Phương án tối ưu xét trên ma trận trọng số ban đầu cũng là phương án tối ưu của bài toán xét trên ma trận rút gọn và đảo lại.

Chứng minh: Có thể xem việc đi tìm chu trình Hamilton của người du lịch như là một bài toán vận tải đặc biệt dưới dạng bảng. Như vậy thì trong bảng (ma trận trọng số hoặc ma trận rút gọn) ta phải có đúng n ô chọn, mỗi ô chọn tượng trưng cho một cặp thành phố trên hành trình cần tìm, trên mỗi dòng và mỗi cột có đúng một ô chọn. Mỗi hành trình h sẽ tương ứng một–một với một tập n ô chọn xác định. f(h) chính là tổng các trọng số ban đầu ghi trong n ô chọn đang xét.

Với mỗi hành trình h bất kỳ, nếu ký hiệu f′(h)= $\sum_{(i,j)\in h} m'_{ij}$ là giá trị của hàm mục tiêu ứng với ma trận rút gọn M’ và s là tổng các hằng số rút gọn thì ta có:

$$f(h) = f'(h)+s.$$

Gọi X là tập toàn bộ các phương án đang xét ở một giai đoạn nào đó, h_0 là phương án tối ưu của bài toán xét trên ma trận trọng số ban đầu M, ta có:

$$f(h_0) \le f(h), \forall h \in X$$

hay $f(h_0)-s \le f(h)-s, \forall h \in X$ hay $f'(h_0) \le f'(h), \forall h \in X$ hay h_0 là phương án tối ưu của bài toán xét trên ma trận rút gọn M’.

5.3.6. Phân nhánh: Sự phân hoạch tập hợp tất cả các hành trình ở một giai đoạn nào đó thành hai tập con rời nhau được biểu diễn bằng sự phân nhánh của một cây. Trên cây, mỗi đỉnh được biểu diễn thành một vòng tròn và sẽ tượng trưng cho môt tập hành trình nào đó. Đỉnh X đầu tiên là tập toàn bộ các hành trình. Đỉnh (i,j) biểu diễn tập các hành trình có chứa cặp (i,j) kề nhau. Đỉnh $\overline{(i,j)}$ biểu diễn tập các hành trình không chứa cặp (i,j) kề nhau. Tại đỉnh (i,j) lại có sự phân nhánh: đỉnh (k,l) biểu diễn tập các hành trình có chứa cặp (i,j) và cặp (k,l), đỉnh $\overline{(k,l)}$ biểu diễn tập các hành trình có chứa cặp (i,j) nhưng không chứa cặp (k,l) ...

Nếu quá trình diễn ra đủ lớn thì cuối cùng sẽ có những đỉnh chỉ biểu diễn một hành trình duy nhất.

Vấn đề đặt ra là nên chọn cặp thành phố nào để tiến hành phân nhánh xuất phát từ một đỉnh cho trước trên cây? Một cách tự nhiên ta nên chọn cặp thành phố nào gần nhau nhất để phân nhánh trước, trên ma trận rút gọn thì những cặp thành phố (i,j) như vậy đều có m'_{ij}=0 và những hành trình nào chứa cặp (i,j) đều có triển vọng là tốt.

Trên ma trận rút gọn thường có nhiều cặp thành phố thoả mãn điều kiện đó (m'_{ij}=0). Để quyết định ta phải tìm cách so sánh. Vì thành phố i nhất thiết phải nối liền với một thành phố nào đó nên các hành trình h không chứa (i,j) tức là h$\in \overline{(i,j)}$ phải ứng với những độ dài hành trình ít ra có chứa phần tử nhỏ nhất trong dòng i không kể m'_{ij}=0 và phần tử nhỏ nhất trong cột j không kể m'_{ij}=0 vì thành phố j nhất thiết phải nối liền với một thành phố nào đó ở trước nó trên hành trình. Ký hiệu tổng của hai phần tử nhỏ nhất đó là θ_{ij} thì ta có f'(h) $\geq \theta_{ij}$, $\forall$h$\in \overline{(i,j)}$.

Vì lý do trên, số θ_{ij} có thể dùng làm tiêu chuẩn so sánh giữa các cặp thành phố (i,j) cùng có m'_{ij}=0. Một cách tổng quát, ở mỗi giai đoạn ta sẽ chọn cặp thành phố (i,j) có m'_{ij}=0 trong ma trận rút gọn và có θ_{ij} lớn nhất để tiến hành phân nhánh từ một đỉnh trên cây.

5.3.7. Tính cận: Với mỗi đỉnh của cây phân nhánh, ta phải tính cận dưới của các giá trị hàm mục tiêu ứng với tập phương án mà đỉnh đó biểu diễn. Cận dưới tính được sẽ ghi bên dưới đỉnh đang xét.

Theo công thức f(h)=f'(h)+s và do f'(h) $\geq$ 0 nên f(h) $\geq$ s, $\forall$h$\in$X. Vì vậy tổng các hằng số rút gọn của ma trận ban đầu có thể lấy làm cận dưới của đỉnh X đầu tiên trên cây. Mặt khác, ta lại có f'(h) $\geq \theta_{ij}$, $\forall$h$\in \overline{(i,j)}$, do đó f(h)=f'(h)+s $\geq \theta_{ij}$+s, $\forall$h$\in \overline{(i,j)}$. Vì vậy tổng θ_{ij}+s có thể lấy làm cận dưới cho đỉnh $\overline{(i,j)}$. Sau khi chọn (i,j) để phân nhánh xuất phát từ đỉnh X thì trên bảng có thể xoá dòng i và cột j vì trên đó ô chọn (i,j) là duy nhất. Sau khi bỏ dòng i và cột j thì ma trận M' lại có thể rút gọn thành ma trận M'' với s' là tổng các hằng số rút gọn, f''(h) là giá trị của hàm mục tiêu xét trên M''. Khi đó ta có f'(h)=f''(h)+s', $\forall$h$\in$(i,j), do đó f(h)=f'(h)+s=f''(h)+s+s', $\forall$h$\in$(i,j). Do f''(h) $\geq$ 0 nên

f(h) ≥ s+s', ∀h∈(i,j), nghĩa là tổng s+s' có thể lấy làm cận dưới cho đỉnh (i,j) trong cây phân nhánh.

Nếu tiếp tục phân nhánh thì cận dưới của các đỉnh tiếp sau được tính toán tương tự, vì đây là một quá trình lặp. Ta chỉ cần xem đỉnh xuất phát của các nhánh giống như đỉnh X ban đầu.. Để tiết kiệm khối lượng tính toán, người ta thường chọn đỉnh có cận dưới nhỏ nhất để phân nhánh tiếp tục.

5.3.8. Thủ tục ngăn chặn hành trình con: Một đường đi hoặc chu trình Hamilton không thể chứa chu trình con với số cạnh tạo thành nhỏ hơn n. Vì vậy ta sẽ đặt $m_{ii}=\infty$ (i=1, ..., n) để tránh các khuyên.

Với i≠j và nếu (i,j) là ô chọn thì phải đặt ngay $m'_{ji}=\infty$ trong ma trận rút gọn.

Nếu đã chọn (i,j) và (j,k) và n>3 thì phải đặt ngay $m'_{ji}=m'_{kj}=m'_{ki}=\infty$.

Chú ý rằng việc đặt $m'_{ij}=\infty$ tương đương với việc xoá ô (i,j) trong bảng hoặc xem (i,j) là ô cấm, nghĩa là hai thành phố i và j không được kề nhau trong hành trình định kiến thiết. Ở mỗi giai đoạn của quá trình đều phải tiến hành thủ tục ngăn chặn này trước khi tiếp tục rút gọn ma trận.

5.3.9. Tính chất tối ưu: Quá trình phân nhánh, tính cận, ngăn chặn hành trình con, rút gọn ma trận phải thực hiện cho đến khi nào có đủ n ô chọn để kiến thiết một hành trình Hamilton, nói cách khác là cho đến khi trên cây phân nhánh đã xuất hiện một đỉnh chỉ biểu diễn một hành trình duy nhất và đã xoá hết được mọi dòng mọi cột trong bảng. Cận dưới của đỉnh cuối cùng này chính là độ dài của hành trình vừa kiến thiết.

a) Nếu cận dưới của đỉnh này không lớn hơn các cận dưới của mọi đỉnh treo trên cây phân nhánh thì hành trình đó là tối ưu.

b) Nếu trái lại thì phải xuất phát từ đỉnh treo nào có cận dưới nhỏ hơn để phân nhánh tiếp tục và kiểm tra xem điều kiện a) có thoả mãn không.

Thí dụ 5: Xét bài toán với 6 thành phố, các số liệu cho theo bảng sau:

$$M=\begin{pmatrix} \infty & 27 & 43 & 16 & 30 & 26 \\ 7 & \infty & 14 & 1 & 30 & 25 \\ 20 & 13 & \infty & 35 & 5 & 0 \\ 21 & 16 & 25 & \infty & 18 & 18 \\ 12 & 46 & 27 & 48 & \infty & 5 \\ 23 & 5 & 5 & 9 & 5 & \infty \end{pmatrix} \begin{matrix} 16 \\ 1 \\ 0 \\ 16 \\ 5 \\ 5 \end{matrix}$$

$$\begin{matrix} 5 & 0 & 0 & 0 & 0 & 0 \end{matrix}$$

Tổng các hằng số rút gọn bước đầu là s=48. Trong ma trận rút gọn ta có:

$$m'_{14}=m'_{24}=m'_{36}=m'_{41}=m'_{42}=m'_{56}=m'_{62}=m'_{63}=m'_{65}=0$$

và $\theta_{14}=10$, $\theta_{24}=1$, $\theta_{36}=5$, $\theta_{41}=1$, $\theta_{42}=0$, $\theta_{56}=2$, $\theta_{62}=0$, $\theta_{63}=9$, $\theta_{65}=2$. Sau khi so sánh ta thấy $\theta_{14}=10$ là lớn nhất nên ta chọn ô (1,4) để phân nhánh. Cận dưới của đỉnh $\overline{(1,4)}$ là $s+\theta_{14}=58$. Xoá dòng 1 cột 4 rồi đặt $m'_{41}=\infty$.

$$M' = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array} \begin{array}{c} \begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \end{array} \\ \begin{pmatrix} \infty & 11 & 27 & 0 & 14 & 10 \\ 1 & \infty & 13 & 0 & 29 & 24 \\ 15 & 13 & \infty & 35 & 5 & 0 \\ 0 & 0 & 9 & \infty & 2 & 2 \\ 2 & 41 & 22 & 43 & \infty & 0 \\ 13 & 0 & 0 & 4 & 0 & \infty \end{pmatrix} \end{array}.$$

$$\begin{array}{c} \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 5 & 6 \end{array} \\ \begin{pmatrix} 1 & \infty & 13 & 29 & 24 \\ 15 & 13 & \infty & 5 & 0 \\ \infty & 0 & 9 & 2 & 2 \\ 2 & 41 & 22 & \infty & 0 \\ 13 & 0 & 0 & 0 & \infty \end{pmatrix} \end{array} \longrightarrow M'' = \begin{array}{c} \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 5 & 6 \end{array} \\ \begin{pmatrix} 0 & \infty & 12 & 28 & 23 \\ 15 & 13 & \infty & 5 & 0 \\ \infty & 0 & 9 & 2 & 2 \\ 2 & 41 & 22 & \infty & 0 \\ 13 & 0 & 0 & 0 & \infty \end{pmatrix} \end{array}.$$

Tổng hằng số rút gọn là $s'=1$. Vậy cận dưới của đỉnh (1,4) là $s+s'=49$. Vì $49<58$ nên tiếp tục phân nhánh tại đỉnh (1,4). Trong ma trận còn lại, sau khi rút gọn ta có

$$m''_{21}=m''_{36}=m''_{42}=m''_{56}=m''_{62}=m''_{63}=m''_{65}=0.$$

Ở giai đoạn này, sau khi tính toán ta thấy $\theta_{21}=14$ là lớn nhất nên chọn tiếp ô (2,1). Cận dưới của đỉnh $\overline{(2,1)}$ là $49+\theta_{21}=63$. Xoá dòng 2 cột 1. Đặt $m''_{42}=\infty$. Rút gọn ma trận còn lại, ta có:

$$\begin{array}{c} \\ 3 \\ 4 \\ 5 \\ 6 \end{array} \begin{array}{c} \begin{array}{cccc} 2 & 3 & 5 & 6 \end{array} \\ \begin{pmatrix} 13 & \infty & 5 & 0 \\ \infty & 9 & 2 & 2 \\ 41 & 22 & \infty & 0 \\ 0 & 0 & 0 & \infty \end{pmatrix} \end{array} \longrightarrow M''' = \begin{array}{c} \\ 3 \\ 4 \\ 5 \\ 6 \end{array} \begin{array}{c} \begin{array}{cccc} 2 & 3 & 5 & 6 \end{array} \\ \begin{pmatrix} 13 & \infty & 5 & 0 \\ \infty & 7 & 0 & 0 \\ 41 & 22 & \infty & 0 \\ 0 & 0 & 0 & \infty \end{pmatrix} \end{array}.$$

Tổng hằng số rút gọn là 2. Cận dưới của đỉnh (2,1) là $49+2=51$.

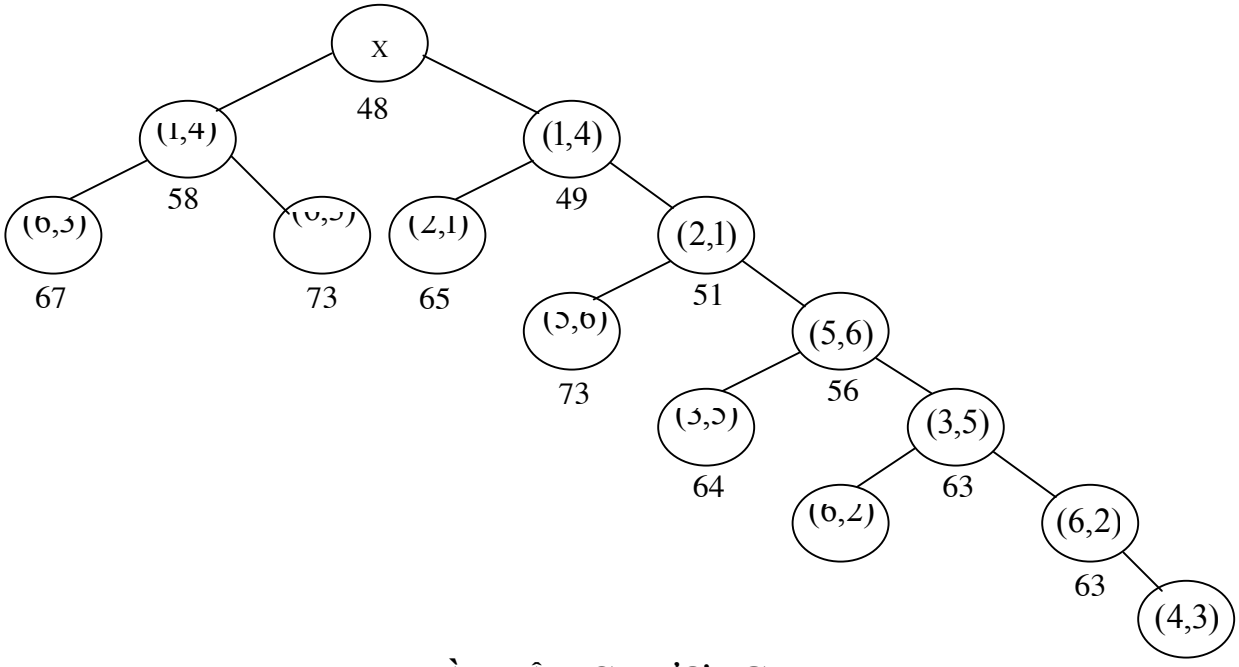
Tiếp tục như vậy cuối cùng ta được 6 ô chọn là:

(1,4), (2,1), (5,6), (3,5), (4,3), (6,2)

và kiến thiết hành trình h_0=(1 4 3 5 6 2 1) với $f(h_0)=63$ là cận dưới của đỉnh cuối cùng. cận dưới của đỉnh cuối cùng là 63, trong khi đó đỉnh treo $\overline{(1,4)}$ có cận dưới là $58<63$ nên phải tiếp tục phân nhánh từ đó để kiểm tra. Sau sự phân nhánh này thì mọi đỉnh treo đều có cận dưới không nhỏ hơn 63 nên có thể khẳng định rằng hành trình h_0=(1 4 3 5 6 2 1) là tối ưu.

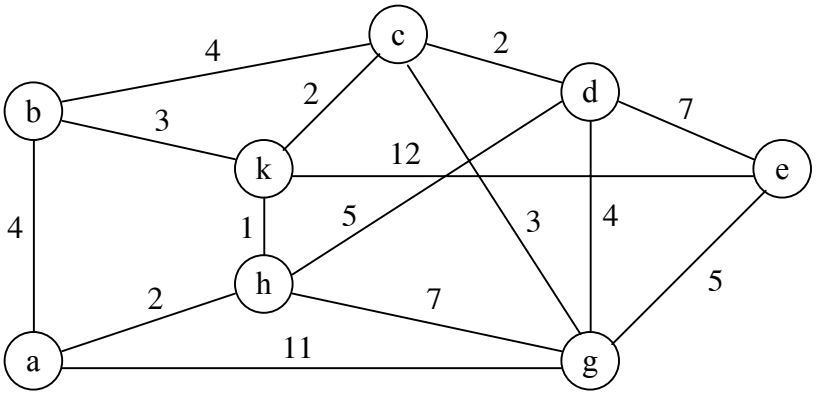
Sự phân nhánh từ đỉnh $\overline{(1,4)}$ được làm như sau: trong ma trận rút gọn đợt 1, ta đặt $m'_{14}=\infty$ vì xem ô (1,4) là ô cấm, $\theta_{63}=9$ là lớn nhất trong các θ_{ij}, do đó chọn ô (6,3) để

phân nhánh. Cận dưới của đỉnh $\overline{(6,3)}$ là 58+9=67. Đặt $m'_{36}=\infty$. Rút gọn ma trận với tổng hằng số rút gọn là 15. Cận dưới của đỉnh (6,3) là 58+15=73.

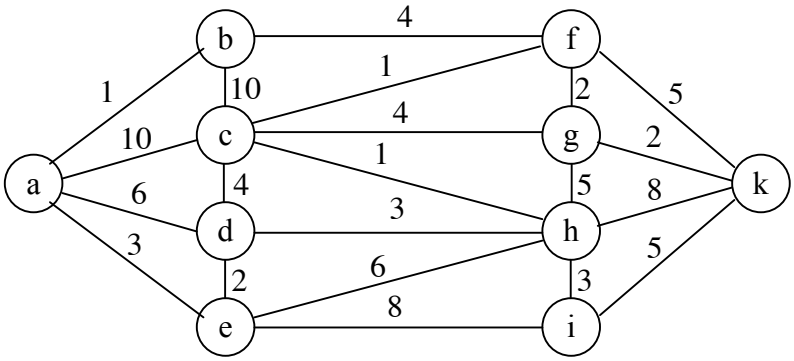


BÀI TẬP CHƯƠNG V:

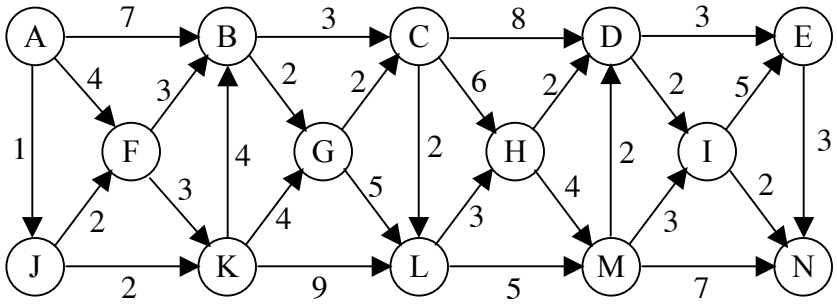
1. Dùng thuật toán Dijkstra tìm đường đi ngắn nhất từ đỉnh a đến các đỉnh khác trong đồ thị sau:



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



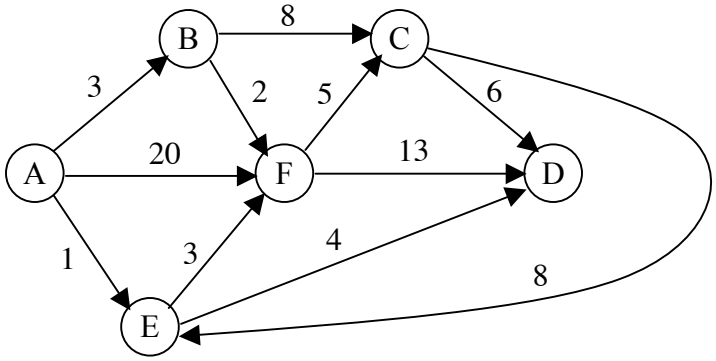
3. Cho đồ thị có trọng số như hình dưới đây. Hãy tìm đường đi ngắn nhất từ đỉnh A đến đỉnh N.



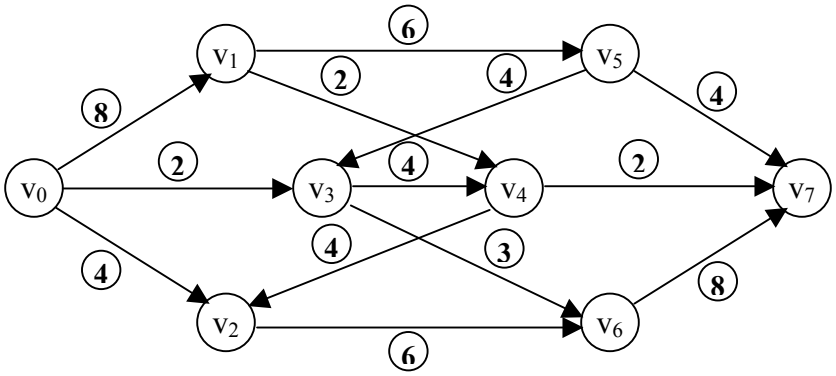
4. Tìm đường đi ngắn nhất từ B đến các đỉnh khác của đồ thị có ma trận trọng số là (các ô trống là ∞):

	A	B	C	D	E	F	G
A		3	6				
B	3		2	4			
C	6	2		1	4	2	
D		4	1		2		4
E			4	2		2	1
F			2		2		4
G				4	1	4	

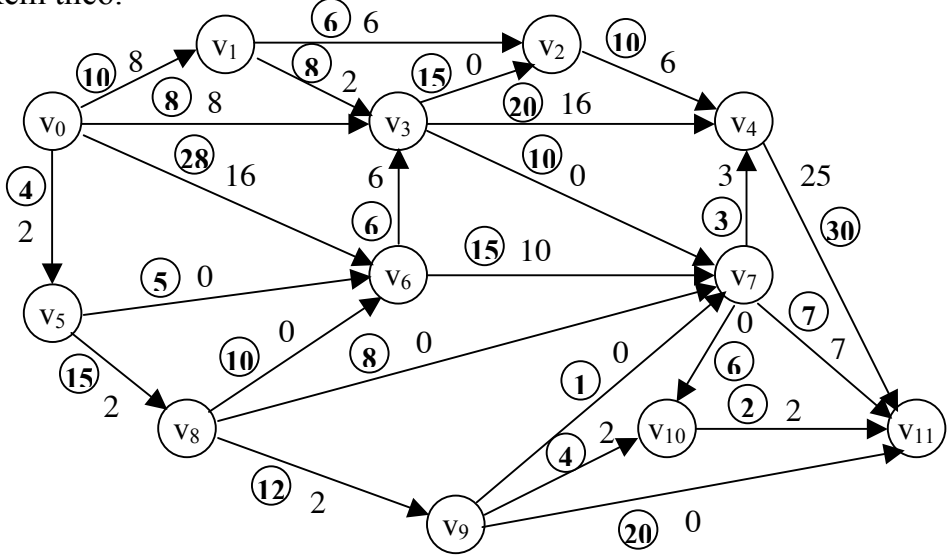
5. Tìm W* bằng cách áp dụng thuật toán Floyd vào đồ thị sau:



6. Giải bài toán mạng vận tải sau bằng thuật toán Ford-Fulkerson với luồng vận tải khởi đầu bằng 0.



7. Giải bài toán mạng vận tải sau bằng thuật toán Ford-Fulkerson với luồng vận tải khởi đầu được cho kèm theo.



8. Hãy giải bài toán *người du lịch* với 6 thành phố, có số liệu cho trong ma trận trọng số sau:

$$\begin{pmatrix} \infty & 25 & 45 & 14 & 32 & 24 \\ 9 & \infty & 16 & 2 & 34 & 23 \\ 22 & 11 & \infty & 33 & 7 & 0 \\ 23 & 14 & 27 & \infty & 20 & 21 \\ 14 & 44 & 29 & 46 & \infty & 3 \\ 25 & 3 & 4 & 7 & 8 & \infty \end{pmatrix}.$$

CHƯƠNG VI
CÂY

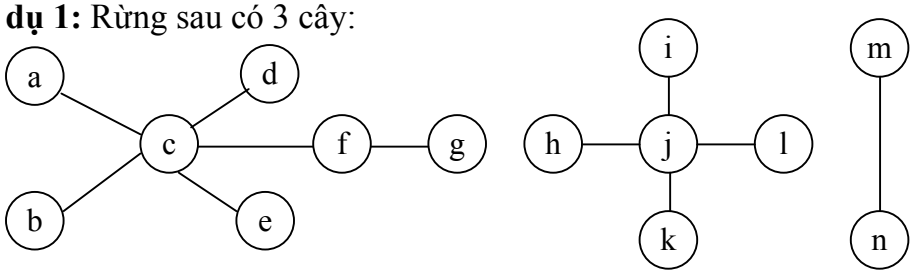
Một đồ thị liên thông và không có chu trình được gọi là cây. Cây đã được dùng từ năm 1857, khi nhà toán học Anh tên là Arthur Cayley dùng cây để xác định những dạng khác nhau của hợp chất hoá học. Từ đó cây đã được dùng để giải nhiều bài toán trong nhiều lĩnh vực khác nhau. Cây rất hay được sử dụng trong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

6.1. ĐỊNH NGHĨA VÀ CÁC TÍNH CHẤT CƠ BẢN.

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

Một đồ thị vô hướng không chứa chu trình và có ít nhất hai đỉnh gọi là một rừng. Trong một rừng, mỗi thành phần liên thông là một cây.

Thí dụ 1: Rừng sau có 3 cây:



6.1.2. Mệnh đề: Nếu T là một cây có n đỉnh thì T có ít nhất hai đỉnh treo.

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

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

1) T là một cây.

2) T liên thông và có n−1 cạnh.

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

4) T liên thông và mỗi cạnh là cầu.

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

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

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

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

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

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

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

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

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

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

6.2. CÂY KHUNG VÀ BÀI TOÁN TÌM CÂY KHUNG NHỎ NHẤT.

6.2.1. Định nghĩa: Trong đồ thị liên thông G, nếu ta loại bỏ cạnh nằm trên chu trình nào đó thì ta sẽ được đồ thị vẫn là liên thông. Nếu cứ loại bỏ các cạnh ở các chu trình khác cho đến khi nào đồ thị không còn chu trình (vẫn liên thông) thì ta thu được một cây nối các đỉnh của G. Cây đó gọi là cây khung hay cây bao trùm của đồ thị G.

Tổng quát, nếu G là đồ thị có n đỉnh, m cạnh và k thành phần liên thông thì áp dụng thủ tục vừa mô tả đối với mỗi thành phần liên thông của G, ta thu được đồ thị gọi là rừng khung của G. Số cạnh bị loại bỏ trong thủ tục này bằng m−n+k, số này ký hiệu là ν(G) và gọi là chu số của đồ thị G.

6.2.2. Bài toán tìm cây khung nhỏ nhất: Bài toán tìm cây khung nhỏ nhất của đồ thị là một trong số những bài toán tối ưu trên đồ thị tìm được ứng dụng trong nhiều lĩnh

 www.VIETMATHS.com

vực khác nhau của đời sống. Trong phần này ta sẽ có hai thuật toán cơ bản để giải bài toán này. Trước hết, nội dung của bài toán được phát biểu như sau.

Cho G=(V,E) là đồ thị vô hướng liên thông có trọng số, mỗi cạnh $e \in E$ có trọng số $m(e) \geq 0$. Giả sử $T=(V_T,E_T)$ là cây khung của đồ thị G ($V_T=V$). Ta gọi độ dài m(T) của cây khung T là tổng trọng số của các cạnh của nó:

$$m(T)= \sum_{e \in E_T} m(e) .$$

Bài toán đặt ra là trong số tất cả các cây khung của đồ thị G, hãy tìm cây khung có độ dài nhỏ nhất. Cây khung như vậy được gọi là cây khung nhỏ nhất của đồ thị và bài toán đặt ra được gọi là bài toán tìm cây khung nhỏ nhất.

Để minh hoạ cho những ứng dụng của bài toán cây khung nhỏ nhất, dưới đây là hai mô hình thực tế tiêu biểu cho nó.

Bài toán xây dựng hệ thống đường sắt: Giả sử ta muốn xây dựng một hệ thống đường sắt nối n thành phố sao cho hành khách có thể đi từ bất cứ một thành phố nào đến bất kỳ một trong số các thành phố còn lại. Mặt khác, trên quan điểm kinh tế đòi hỏi là chi phí về xây dựng hệ thống đường phải là nhỏ nhất. Rõ ràng là đồ thị mà đỉnh là các thành phố còn các cạnh là các tuyến đường sắt nối các thành phố tương ứng, với phương án xây dựng tối ưu phải là cây. Vì vậy, bài toán đặt ra dẫn về bài toán tìm cây khung nhỏ nhất trên đồ thị đầy đủ n đỉnh, mỗi đỉnh tương ứng với một thành phố với độ dài trên các cạnh chính là chi phí xây dựng hệ thống đường sắt nối hai thành phố.

Bài toán nối mạng máy tính: Cần nối mạng một hệ thống gồm n máy tính đánh số từ 1 đến n. Biết chi phí nối máy i với máy j là m(i,j) (thông thường chi phí này phụ thuộc vào độ dài cáp nối cần sử dụng). Hãy tìm cách nối mạng sao cho tổng chi phí là nhỏ nhất. Bài toán này cũng dẫn về bài toán tìm cây khung nhỏ nhất.

Bài toán tìm cây khung nhỏ nhất đã có những thuật toán rất hiệu quả để giải chúng. Ta sẽ xét hai trong số những thuật toán như vậy: thuật toán Kruskal và thuật toán Prim.

6.2.3. Thuật toán Kruskal: Thuật toán sẽ xây dựng tập cạnh E_T của cây khung nhỏ nhất $T=(V_T, E_T)$ theo từng bước. Trước hết sắp xếp các cạnh của đồ thị G theo thứ tự không giảm của trọng số. Bắt đầu từ $E_T=\varnothing$, ở mỗi bước ta sẽ lần lượt duyệt trong danh sách cạnh đã sắp xếp, từ cạnh có độ dài nhỏ đến cạnh có độ dài lớn hơn, để tìm ra cạnh mà việc bổ sung nó vào tập E_T không tạo thành chu trình trong tập này. Thuật toán sẽ kết thúc khi ta thu được tập E_T gồm n−1 cạnh. Cụ thể có thể mô tả như sau:

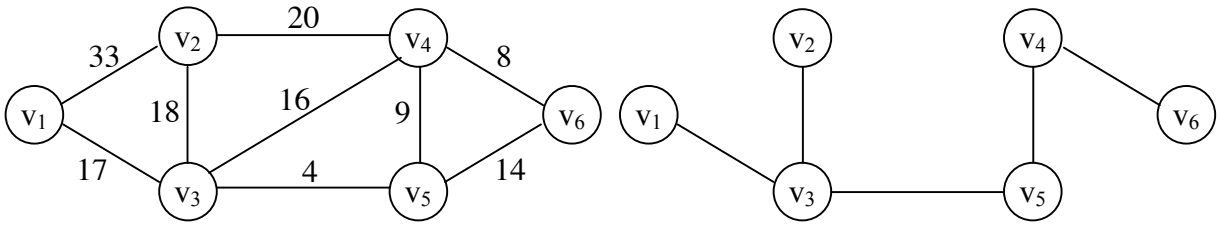
1. Bắt đầu từ đồ thị rỗng T có n đỉnh.

2. Sắp xếp các cạnh của G theo thứ tự không giảm của trọng số.

3. Bắt đầu từ cạnh đầu tiên của dãy này, ta cứ thêm dần các cạnh của dãy đã được xếp vào T theo nguyên tắc cạnh thêm vào không được tạo thành chu trình trong T.

4. Lặp lại Bước 3 cho đến khi nào số cạnh trong T bằng n−1, ta thu được cây khung nhỏ nhất cần tìm.

Thí dụ 2: Tìm cây khung nhỏ nhất của đồ thị cho trong hình dưới đây:



Bắt đầu từ đồ thị rỗng T có 6 đỉnh.

Sắp xếp các cạnh của đồ thị theo thứ tự không giảm của trọng số:

$\{(v_3, v_5), (v_4, v_6), (v_4, v_5), (v_5, v_6), (v_3, v_4), (v_1, v_3), (v_2, v_3), (v_2, v_4), (v_1, v_2)\}$.

Thêm vào đồ thị T cạnh (v_3, v_5).

Do số cạnh của T là $1<6-1$ nên tiếp tục thêm cạnh (v_4, v_6) vào T. Bây giờ số cạnh của T đã là 2 vẫn còn nhỏ hơn 6, ta tiếp tục thêm cạnh tiếp theo trong dãy đã sắp xếp vào T. Sau khi thêm cạnh (v_4, v_5) vào T, nếu thêm cạnh (v_5, v_6) thì nó sẽ tạo thành với 2 cạnh (v_4, v_5), (v_4, v_6) đã có trong T một chu trình. Tình huống tương tự cũng xãy ra đối với cạnh (v_3, v_4) là cạnh tiếp theo trong dãy. Tiếp theo ta bổ sung cạnh (v_1, v_3), (v_2, v_3) vào T và thu dược tập E_T gồm 5 cạnh:

$$\{(v_3, v_5), (v_4, v_6), (v_4, v_5), (v_1, v_3), (v_2, v_3)\}.$$

Tính đúng đắn của thuật toán: Rõ ràng đồ thị thu được theo thuật toán có n−1 cạnh và không có chu trình. Vì vậy theo Định lý 6.1.3, nó là cây khung của đồ thị G. Như vậy chỉ còn phải chỉ ra rằng T có độ dài nhỏ nhất. Giả sử tồn tại cây khung S của đồ thị mà $m(S)<m(T)$. Ký hiệu e_k là cạnh đầu tiên trong dãy các cạnh của T xây dựng theo thuật toán vừa mô tả không thuộc S. Khi đó đồ thị con của G sinh bởi cây S được bổ sung cạnh e_k sẽ chứa một chu trình duy nhất C đi qua e_k. Do chu trình C phải chứa cạnh e thuộc S nhưng không thuộc T nên đồ thị con thu được từ S bằng cách thay cạnh e của nó bởi e_k, ký hiệu đồ thị này là S', sẽ là cây khung. Theo cách xây dựng, $m(e_k)\le m(e)$, do đó $m(S')\le m(S)$, đồng thời số cạnh chung của S' và T đã tăng thêm một so với số cạnh chung của S và T. Lặp lại quá trình trên từng bước một, ta có thể biến đổi S thành T và trong mỗi bước tổng độ dài không tăng, tức là $m(T)\le m(S)$. Mâu thuẩn này chứng tỏ T là cây khung nhỏ nhất của G.

Độ phức tạp của thuật toán Kruskal được đánh giá như sau. Trước tiên, ta sắp xếp các cạnh của G theo thứ tự có chiều dài tăng dần; việc sắp xếp này có độ phức tạp $O(p^2)$, với p là số cạnh của G. Người ta chứng minh được rằng việc chọn e_{i+1} không tạo nên chu trình với i cạnh đã chọn trước đó có độ phức tạp là $O(n^2)$. Do $p\le n(n-1)/2$, thuật toán Kruskal có độ phức tạp là $O(p^2)$.

6.2.4. Thuật toán Prim: Thuật toán Kruskal làm việc kém hiệu quả đối với những đồ thị dày (đồ thị có số cạnh $m \approx n(n-1)/2$). Trong trường hợp đó, thuật toán Prim tỏ ra hiệu quả hơn. Thuật toán Prim còn được gọi là phương pháp lân cận gần nhất.

1. $V_T := \{v_*\}$, trong đó v_* là đỉnh tuỳ ý của đồ thị G.

$E_T := \varnothing$.

2. Với mỗi đỉnh $v_j \notin V_T$, tìm đỉnh $w_j \in V_T$ sao cho

$$m(w_j, v_j) = \min_{x_i \in V_T} m(x_i, v_j) =: \beta_j$$

và gán cho đỉnh v_j nhãn $[w_j, \beta_j]$. Nếu không tìm được w_j như vậy (tức là khi v_j không kề với bất cứ đỉnh nào trong V_T) thì gán cho v_j nhãn $[0, \infty]$.

3. Chọn đỉnh v_{j*} sao cho

$$\beta_{j*} = \min_{v_j \notin V_T} \beta_j$$

$V_T := V_T \cup \{v_{j*}\}$,

$E_T := E_T \cup \{(w_{j*}, v_{j*})\}$.

Nếu $|V_T| = n$ thì thuật toán dừng và (V_T, E_T) là cây khung nhỏ nhất.

Nếu $|V_T| < n$ thì chuyển sang Bước 4.

4. Đối với tất cả các đỉnh $v_j \notin V_T$ mà kề với v_{j*}, ta thay đổi nhãn của chúng như sau:

Nếu $\beta_j > m(v_{j*}, v_j)$ thì đặt $\beta_j := m(v_{j*}, v_j)$ và nhãn của v_j là $[v_{j*}, \beta_j]$. Ngược lại, ta giữ nguyên nhãn của v_j. Sau đó quay lại Bước 3.

Thí dụ 3: Tìm cây khung nhỏ nhất bằng thuật toán Prim của đồ thị gồm các đỉnh A, B, C, D, E, F, H, I được cho bởi ma trận trọng số sau.

	A	B	C	D	E	F	H	I
A	∞	15	16	19	23	20	32	18
B	15	∞	33	13	34	19	20	12
C	16	33	∞	13	29	21	20	19
D	19	13	13	∞	22	30	21	11
E	23	34	29	22	∞	34	23	21
F	20	19	21	30	34	∞	17	18
H	32	20	20	21	23	17	∞	14
I	18	12	19	11	21	18	14	∞

Yêu cầu viết các kết quả trung gian trong từng bước lặp, kết quả cuối cùng cần đưa ra tập cạnh và độ dài của cây khung nhỏ nhất.

V.lặp	**A**	**B**	**C**	**D**	**E**	**F**	**H**	**I**	$\mathbf{V_T}$	$\mathbf{E_T}$
K.tạo	–	**[**A,15**]**	[A,16]	[A,19]	[A,23]	[A,20]	[A,32]	[A,18]	A	∅
1	–	–	[A,16]	[B,13]	[A,23]	[B,19]	[B,20]	**[**B,12**]**	A, B	(A,B)
2	–	–	[A,16]	**[**I,11**]**	[I,21]	[I,18]	[I,14]	–	A, B, I	(A,B), (B,I)
3	–	–	**[**D,13**]**	–	[I,21]	[I,18]	[I,14]	–	A, B, I, D	(A,B), (B,I), (I,D)
4	–	–	–	–	[I,21]	[I,18]	**[**I,14**]**	–	A, B, I, D, C	(A,B), (B,I), (I,D), (D,C)
5	–	–	–	–	[I,21]	**[**H,17**]**	–	–	A, B, I, D, C, H	(A,B), (B,I), (I,D), (D,C), (I,H)
6	–	–	–	–	**[**I,21**]**	–	–	–	A, B, I, D, C, H, F	(A,B), (B,I), (I,D), (D,C), (I,H), (H,F)
7	–	–	–	–	–	–	–	–	A, B, I, D, C, H, F, E	(A,B), (B,I), (I,D), (D,C), (I,H), (H,F), (I,E)

Vậy độ dài cây khung nhỏ nhất là:

15 + 12 + 11 + 13 + 14 + 17 + 21 = 103.

Tính đúng đắn của thuật toán: Để chứng minh thuật toán Prim là đúng, ta chứng minh bằng quy nạp rằng T(k) (k=1, 2, ...,n), đồ thị nhận được trong vòng lặp thứ k, là một đồ thị con của cây khung nhỏ nhất của G, do đó T(n) chính là một cây khung nhỏ nhất của G.

T(1) chỉ gồm đỉnh v_* của G, do đó T(1) là đồ thị con của mọi cây khung của G. Giả sử T(i) ($1 \le i < n$) là một đồ thị con của một cây khung nhỏ nhất của G. Ta chứng minh rằng T(i+1) cũng là đồ thị con của một cây khung nhỏ nhất.

Thật vậy, theo thuật toán Prim $E_{T(i+1)}=E_{T(i)} \cup \{e_{i+1}\}$, với e_{i+1} là cạnh ngắn nhất trong tất cả các cạnh có một đầu mút thuộc $V_{T(i)}$, đầu mút kia không thuộc $V_{T(i)}$.

Nếu e_{i+1} là một cạnh của T thì T_{i+1} là đồ thị con của T.

Nếu e_{i+1} không phải là một cạnh của T thì T_{i+1} là đồ thị con $T'=(V_T, E_T \cup \{e_{i+1}\})$. Đồ thị T' chứa một chu trình sơ cấp duy nhất C (theo tính chất 6 của định lý về cây). Ta chọn trong C một cạnh e_j có một đỉnh thuộc T(i) và đỉnh kia không thuộc T(i) và $e_j \ne e_{i+1}$. Ta bỏ e_j trong C. Khi đó

$$T''=(V_T, E_{T'} \setminus \{e_j\})$$

là một cây khung của G và T(i+1) là đồ thị con của T' nên cũng là đồ thị con của T''. Theo cách chọn e_{i+1} của thuật toán Prim, ta có

$$m(e_{i+1}) \le m(e_j) \text{ do đó } m(T'') \le m(T).$$

Nhưng T'' là một cây khung của G, còn T là cây khung nhỏ nhất, vì vậy phải có m(T'')=m(T), tức là T'' cũng là cây khung nhỏ nhất của G.

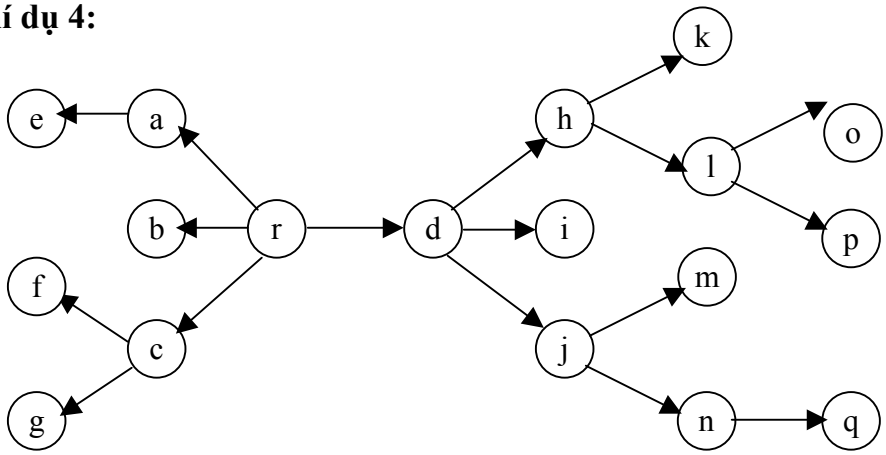
Độ phức tạp của thuật toán Prim là $O(n^3)$. Thật vậy, nếu T(k) có k đỉnh thì có n–k đỉnh không thuộc T(k), do đó ta phải chọn chiều dài nhỏ nhất của nhiều nhất là k(n–k) cạnh. Do $k(n-k) < (n-1)^2$, nên độ phức tạp của bước chọn e_{k+1} là $O(n^2)$. Vì phải chọn n–1 cạnh, nên độ phức tạp của thuật toán Prim là $O(n^3)$.

6.3. CÂY CÓ GỐC.

6.3.1. Định nghĩa: Cây có hướng là đồ thị có hướng mà đồ thị vô hướng nền của nó là một cây.

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

Thí dụ 4:



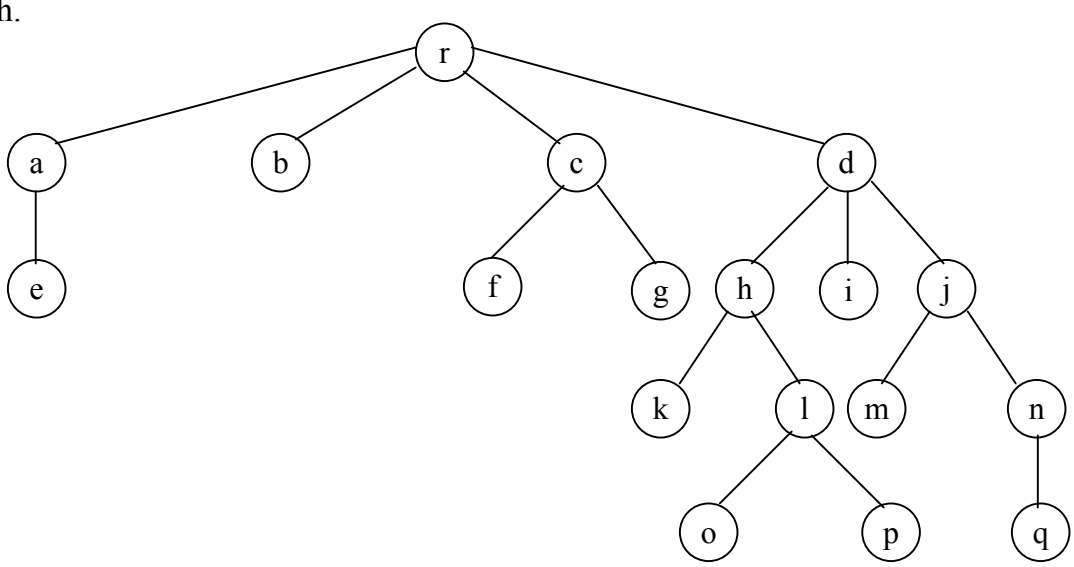
Trong cây có gốc thì gốc r có bậc vào bằng 0, còn tất cả các đỉnh khác đều có bậc vào bằng 1.

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

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

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

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



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

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

– v_{i+1} là con của v_i và v_i là cha của v_{i+1}.

– $v_0, v_1, ..., v_{n-1}$ là các tổ tiên của v_n và v_n là dòng dõi của $v_0, v_1, ..., v_{n-1}$.

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

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

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

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

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

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

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

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

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

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

6.4. DUYỆT CÂY NHỊ PHÂN.

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

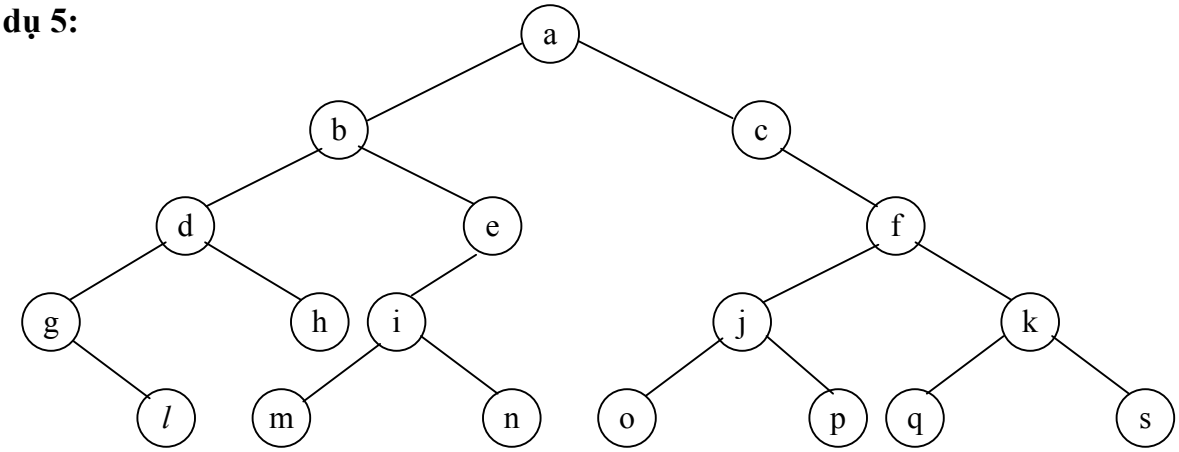
Có nhiều thuật toán duyệt cây nhị phân, các thuật toán đó khác nhau chủ yếu ở thứ tự thăm các đỉnh.

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

con bên trái của T, ký hiệu T(u). Tương tự, ta có cây con bên phải T(v) của T. Một cây T(r) có thể không có cây con bên trái hay bên phải.

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

Thí dụ 5:



6.4.2. Các thuật toán duyệt cây nhị phân:

1) Thuật toán tiền thứ tự:

1. Thăm gốc r.

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

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

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

1. Thăm a
2. Duyệt T(b)
    - 2.1. Thăm b
    - 2.2. Duyệt T(d)
        - 2.2.1. Thăm d
        - 2.2.2. Duyệt T(g)
            - 2.2.2.1. Thăm g
            - 2.2.2.3. Duyệt T(*l*): Thăm *l*
        - 2.2.3. Duyệt T(h): Thăm h
    - 2.3. Duyệt T(e)
        - 2.3.1. Thăm e
        - 2.3.2. Duyệt T(i)
            - 2.3.2.1. Thăm i
            - 2.3.2.2. Duyệt T(m): Thăm m
            - 2.3.2.3. Duyệt T(n): Thăm n
3. Duyệt T(c)

3.1. Thăm c

3.3. Duyệt T(f)

3.3.1.Thăm f

3.3.2. Duyệt T(j)

3.3.2.1. Thăm j

3.3.2.2. Duyệt T(o): Thăm o

3.3.2.3. Duyệt T(p): Thăm p

3.3.3. Duyệt T(k)

3.3.3.1. Thăm k

3.3.3.2. Duyệt T(q): Thăm q

3.3.3.3. Duyệt T(s): Thăm s

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

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

2) Thuật toán trung thứ tự:

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

2. Thăm gốc r.

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

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

1. Duyệt T(b)

1.1. Duyệt T(d)

1.1.1. Duyệt T(g)

1.1.1.2. Thăm g

1.1.1.3. Duyệt T(*l*): thăm *l*

1.1.2. Thăm d

1.1.3. Duyệt T(h): Thăm h

1.2. Thăm b

1.3. Duyệt T(e)

1.3.1. Duyệt T(i)

1.3.1.1. Duyệt T(m): Thăm m

1.3.1.2. Thăm i

1.3.1.3. Duyệt T(n): Thăm n

1.3.2. Thăm e

2. Thăm a

3. Duyệt T(c)

3.2. Thăm c

3.3. Duyệt T(f)

3.3.1. Duyệt T(j)

3.3.1.1. Duyệt T(o): Thăm o
3.3.1.2. Thăm j
3.3.1.3. Duyệt T(p): Thăm p
3.3.2. Thăm f
3.3.3. Duyệt T(k)
3.3.3.1. Duyệt T(q): Thăm q
3.3.3.2. Thăm k
3.3.3.3. Duyệt T(s): Thăm s

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

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

3) Thuật toán hậu thứ tự:

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

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

3. Thăm gốc r.

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

1. Duyệt T(b)
1.1. Duyệt T(d)
1.1.1. Duyệt T(g)
1.1.1.2. Duyệt T(*l*): thăm *l*
1.1.1.3. Thăm g
1.1.2. Duyệt T(h): thăm h
1.1.3. Thăm d
1.2. Duyệt T(e)
1.2.1. Duyệt T(i)
1.2.1.1. Duyệt T(m): Thăm m
1.2.1.2. Duyệt T(n): Thăm n
1.2.1.3. Thăm i
1.2.3. Thăm e
1.3. Thăm b
2. Duyệt T(c)
2.2. Duyệt T(f)
2.2.1. Duyệt T(j)
2.2.1.1. Duyệt T(o): Thăm o
2.2.1.2. Duyệt T(p): Thăm p
2.2.1.3. Thăm j
2.2.2. Duyệt T(k)
2.2.2.1. Duyệt T(q): Thăm q

2.2.2.2. Duyệt T(s): Thăm s

2.2.2.3. Thăm k

2.2.3. Thăm f

2.3. Thăm c

3. Thăm a

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

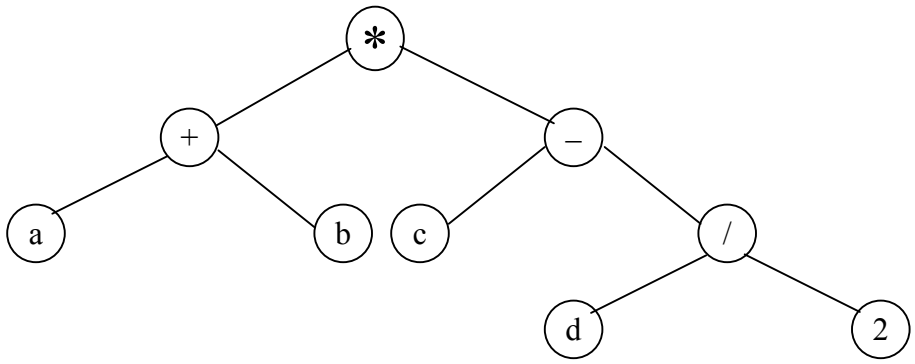
l, g, h, d, m, n, i, e, b, o, p, j, q, s, k, f, c, a.

6.4.3. Ký pháp Ba Lan:

Xét biểu thức đại số sau đây:

$$(a+b)(c-\frac{d}{2}) \qquad (1)$$

Ta vẽ một cây nhị phân như hình dưới đây, trong đó mỗi đỉnh trong mang dấu của một phép tính trong (1), gốc của cây mang phép tính sau cùng trong (1), ở đây là dấu nhân, ký hiệu là $*$, mỗi lá mang một số hoặc một chữ đại diện cho số.



Duyệt cây nhị phân trong hình trên theo trung thứ tự là:

$$a + b * c - d / 2 \qquad (2)$$

và đây là biểu thức (1) đã bỏ đi các dấu ngoặc.

Ta nói rằng biểu thức (1) được biểu diễn bằng cây nhị phân T($*$) trong hình trên, hay cây nhị phân T($*$) này tương ứng với biểu thức (1). Ta cũng nói: cách viết (ký pháp) quen thuộc trong đại số học như cách viết biểu thức (1) là ký pháp trung thứ tự kèm theo các dấu ngoặc.

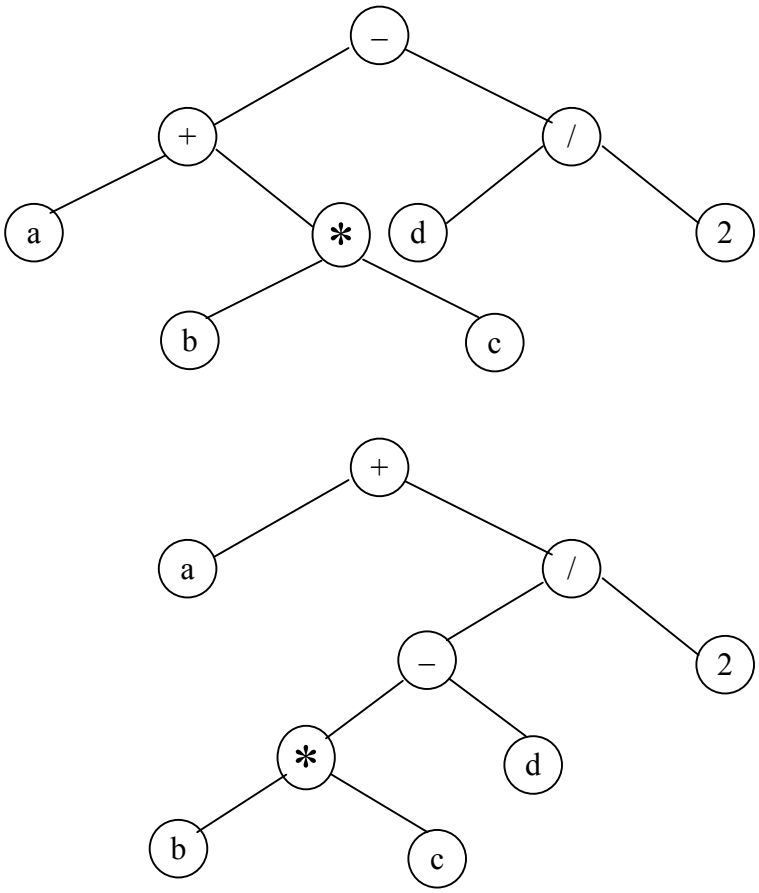
Ta biết rằng các dấu ngoặc trong (1) là rất cần thiết, vì (2) có thể hiểu theo nhiều cách khác (1), chẳng hạn là

$$(a + b * c) - d / 2 \qquad (3)$$

hoặc là $a + (b * c - d) / 2 \qquad (4)$

Các biểu thức (3) và (4) có thể biểu diễn bằng cây nhị phân trong các hình sau. Hai cây nhị phân tương ứng là khác nhau, nhưng đều được duyệt theo trung thứ tự là (2).

www.VIETMATHS.com



Đối với cây trong hình thứ nhất, nếu duyệt theo tiền thứ tự, ta có

$$* + a\ b - c / d\ 2 \qquad (5)$$

và nếu duyệt theo hậu thứ tự, ta có:

$$a\ b + c\ d\ 2 / - * \qquad (6)$$

Có thể chứng minh được rằng (5) hoặc (6) xác định duy nhất cây nhị phân trong hình thứ nhất, do đó xác định duy nhất biểu thức (1) mà không cần dấu ngoặc. Chẳng hạn cây nhị phân trên hình thứ hai được duyệt theo tiền thứ tự là

$$- + a * b\ c / d\ 2$$ khác với (5).

và được duyệt theo hậu thứ tự là

$$a\ b\ c * + d\ 2 / -$$ khác với (6).

Vì vậy, nếu ta viết các biểu thức trong đại số, trong lôgic bằng cách duyệt cây tương ứng theo tiền thứ tự hoặc hậu thứ tự thì ta không cần dùng các dấu ngoặc mà không sợ hiểu nhầm.

Người ta gọi cách viết biểu thức theo tiền thứ tự là ký pháp Ba Lan, còn cách viết theo hậu thứ tự là ký pháp Ba Lan đảo, để ghi nhớ đóng góp của nhà toán học và lôgic học Ba Lan Lukasiewicz (1878-1956) trong vấn đề này.

www.VIETMATHS.com

Việc chuyển một biểu thức viết theo ký pháp quen thuộc (có dấu ngoặc) sang dạng ký pháp Ba Lan hay ký pháp Ba Lan đảo hoặc ngược lại, có thể thực hiện bằng cách vẽ cây nhị phân tương ứng, như đã làm đối với biểu thức (1). Nhưng thay vì vẽ cây nhị phân, ta có thể xem xét để xác định dần các công thức bộ phận của công thức đã cho. Chẳng hạn cho biểu thức viết theo ký pháp Ba Lan là

$$- * \uparrow / - - a\ b * 5\ c\ 2\ 3 \uparrow - c\ d\ 2 * - - a\ c\ d / \uparrow - b * 3\ d\ 3\ 5$$

Trước hết, chú ý rằng các phép toán +, –, *, /, ↑ đều là các phép toán hai ngôi, vì vậy trong cây nhị phân tương ứng, các đỉnh mang dấu các phép toán đều là đỉnh trong và có hai con. Các chữ và số đều đặt ở lá. Theo ký pháp Ba Lan (t.ư. Ba Lan đảo) thì T a b (t.ư. a b T) có nghĩa là a T b, với T là một trong các phép toán +, –, *, /, ↑.

$$- * \uparrow / - \underbrace{- a\ b}_{a-b} * \underbrace{5\ c}_{5c}\ 2\ 3 \uparrow \underbrace{- c\ d}_{c-d}\ 2 * - \underbrace{- a\ c}_{a-c}\ d / \uparrow - b * \underbrace{3\ d}_{3d}\ 3\ 5$$

$$- * \uparrow / \underbrace{- (a-b)\ 5c}_{a-b-5c}\ 2\ 3 \underbrace{\uparrow (c-d)\ 2}_{(c-d)^2} * \underbrace{- (a-c)\ d}_{a-c-d} / \uparrow \underbrace{- b\ (3d)}_{b-3d}\ 3\ 5$$

$$- * \uparrow \underbrace{/ (a-b-5c)\ 2}_{\frac{a-b-5c}{2}}\ 3\ (c-d)^2 * (a-c-d) / \underbrace{\uparrow (b-3d)\ 3}_{(b-3d)^3}\ 5$$

$$- * \underbrace{\uparrow \frac{a-b-5c}{2}\ 3}_{\left(\frac{a-b-5c}{2}\right)^3}\ (c-d)^2 * (a-c-d) \underbrace{/ (b-3d)^3\ 5}_{\frac{(b-3d)^3}{5}}$$

$$\underbrace{- * \left(\frac{a-b-5c}{2}\right)^3 (c-d)^2}_{\left(\frac{a-b-5c}{2}\right)^3 (c-d)^2} * \underbrace{(a-c-d)\ \frac{(b-3d)^3}{5}}_{(a-c-d)\frac{(b-3d)^3}{5}}$$

$$\left(\frac{a-b-5c}{2}\right)^3 (c-d)^2 - (a-c-d)\frac{(b-3d)^3}{5}$$

BÀI TẬP CHƯƠNG VI:

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

a) 4 đỉnh **b)** 5 đỉnh **c)** 6 đỉnh

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

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

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

a) Mọi đỉnh đều có bậc 1.

b) Mọi đỉnh đều có bậc 2.

c) Có 6 đỉnh bậc 2 và 2 đỉnh bậc 1.

d) Có đỉnh bậc 7 và 7 đỉnh bậc 1.

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

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

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

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

a) Đội B vô địch, đội D nhì.

b) Đội B nhì, đội C ba.

c) Đọi A nhì, đội C tư.

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

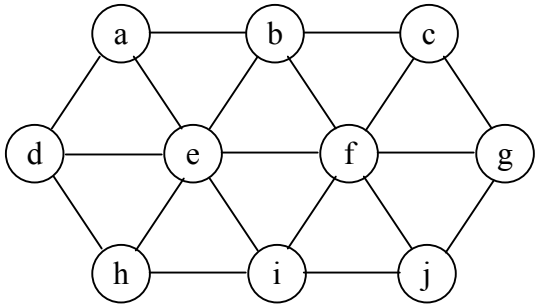
7. Cây *Fibonacci* có gốc T_n được dịnh nghĩa bằng hồi quy như sau. T_1 và T_2 đều là cây có gốc chỉ gồm một đỉnh và với n=3,4, … cây có gốc T_n được xây dựng từ gốc với T_{n-1} như là cây con bên trái và T_{n-2} như là cây con bên phải.

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

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

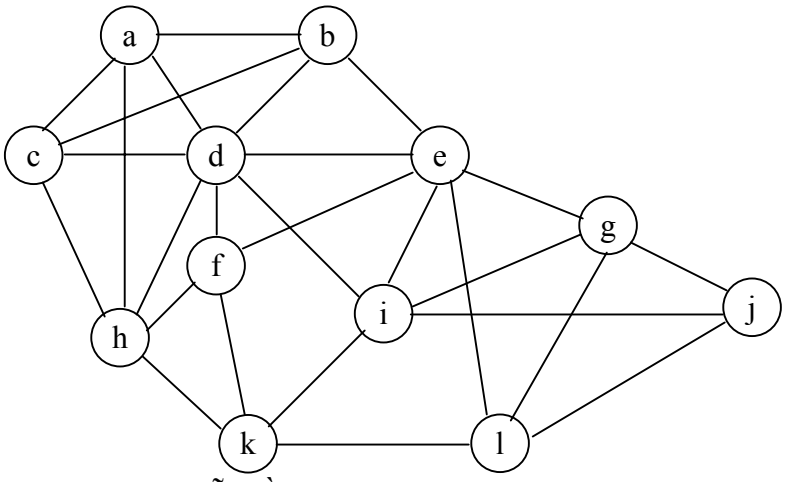
8. Hãy tìm cây khung của đồ thị sau bằng cách xoá đi các cạnh trong các chu trình đơn.

a)



www.VIETMATHS.com

b)



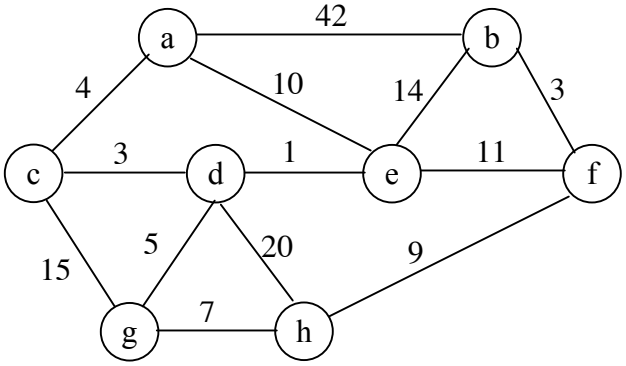
9. Hãy tìm cây khung cho mỗi đồ thị sau.

a) K_5 **b)** $K_{4,4}$ **c)** $K_{1,6}$

d) Q_3 **e)** C_5 **f)** W_5.

10. Đồ thị K_n với n=3, 4, 5 có bao nhiêu cây khung không đẳng cấu?

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



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

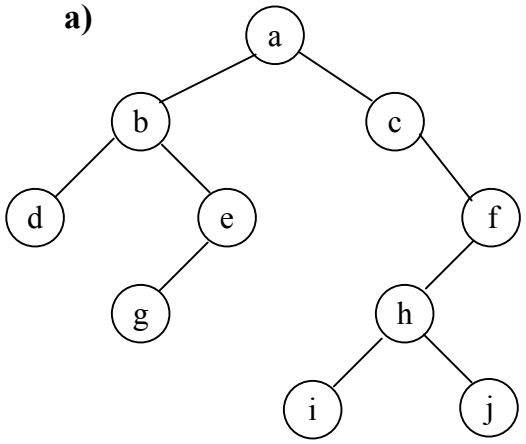
$$\begin{array}{c} \\ A \\ B \\ C \\ D \\ E \\ F \\ G \\ H \end{array} \begin{array}{c} \begin{array}{cccccccc} A & B & C & D & E & F & G & H \end{array} \\ \begin{pmatrix} \infty & 16 & 15 & 23 & 19 & 18 & 32 & 20 \\ 16 & \infty & 13 & 33 & 24 & 20 & 19 & 11 \\ 15 & 13 & \infty & 13 & 29 & 21 & 20 & 19 \\ 23 & 33 & 13 & \infty & 22 & 30 & 21 & 12 \\ 19 & 24 & 29 & 22 & \infty & 34 & 23 & 21 \\ 18 & 20 & 21 & 30 & 34 & \infty & 17 & 14 \\ 32 & 19 & 20 & 21 & 23 & 17 & \infty & 18 \\ 20 & 11 & 19 & 12 & 21 & 14 & 18 & \infty \end{pmatrix} \end{array}.$$

Yêu cầu viết các kết quả trung gian trong từng bước lặp, kết quả cuối cùng cần đưa ra tập cạnh và độ dài của cây khung nhỏ nhất.

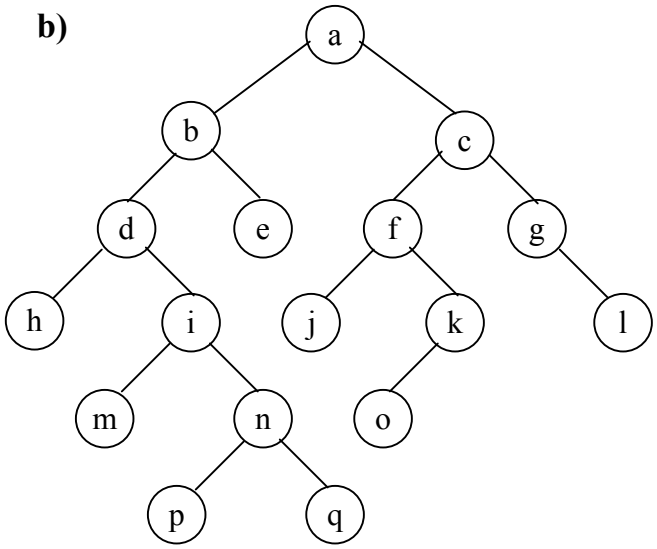
www.VIETMATHS.com

13. Duyệt các cây sau đây lần lượt bằng các thuật toán tiền thứ tự, trung thứ tự và hậu thứ tự.

a)



b)



14. Viết các biểu thức sau đây theo ký pháp Ba Lan và ký pháp Ba Lan đảo.

a) $\dfrac{(A+B)(C+D)}{(A-B)C+D}+\dfrac{A^2+BD}{C^2-BD}$.

b) $\left[(a-b)^4-\dfrac{c}{3}-5d\right]^2+\left(\dfrac{a-d}{3}\right)^4\dfrac{(3a+4b-2d)^3}{5}$.

15. Viết các biểu thức sau đây theo ký pháp quen thuộc.

a) x y + 2 ↑ x y − 2 ↑ − x y * /.

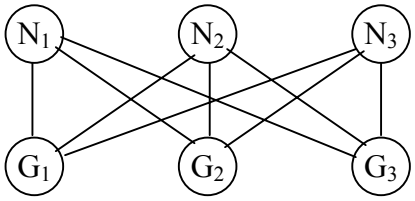
b) − * ↑ / − − a b * 3 c 2 4 ↑ − c d 5 * − − a c d / ↑ − b * 2 d 4 3.

CHƯƠNG VII

ĐỒ THỊ PHẲNG VÀ TÔ MÀU ĐỒ THỊ

Từ xa xưa đã lưu truyền một bài toán cổ "Ba nhà, ba giếng": Có ba nhà ở gần ba cái giếng, nhưng không có đường nối thẳng các nhà với nhau cũng như không có đường nối thẳng các giếng với nhau.

Có lần bất hoà với nhau, họ tìm cách làm các đường khác đến giếng sao cho các đường này đôi một không giao nhau. Họ có thực hiện được ý định đó không?



Bài toán này có thể được mô hình bằng đồ thị phân đôi đầy đủ $K_{3,3}$. Câu hỏi ban đầu có thể diễn đạt như sau: Có thể vẽ $K_{3,3}$ trên một mặt phẳng sao cho không có hai cạnh nào cắt nhau? Trong chương này chúng ta sẽ nghiên cứu bài toán: có thể vẽ một đồ thị trên một mặt phẳng không có các cạnh nào cắt nhau không. Đặc biệt chúng ta sẽ trả lời bài toán ba nhà ba giếng. Thường có nhiều cách biểu diễn đồ thị. Khi nào có thể tìm được ít nhất một cách biểu diễn đồ thị không có cạnh cắt nhau?

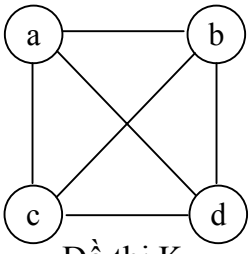
7.1. ĐỒ THỊ PHẲNG.

7.1.1. Định nghĩa: Một đồ thị được gọi là phẳng nếu nó có thể vẽ được trên một mặt phẳng mà không có các cạnh nào cắt nhau (ở một điểm không phải là điểm mút của các cạnh). Hình vẽ như thế gọi là một biểu diễn phẳng của đồ thị.

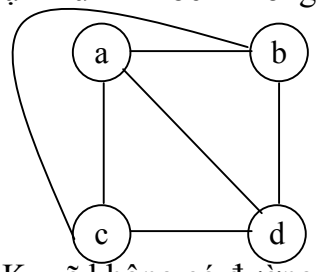
Một đồ thị có thể là phẳng ngay cả khi nó thường được vẽ với những cạnh cắt nhau, vì có thể vẽ nó bằng cách khác không có các cạnh cắt nhau.

Thí dụ 1: 1) Một cây, một chu trình đơn là một đồ thị phẳng.

2) K_4 là đồ thị phẳng bởi vì có thể vẽ lại như hình bên không có đường cắt nhau

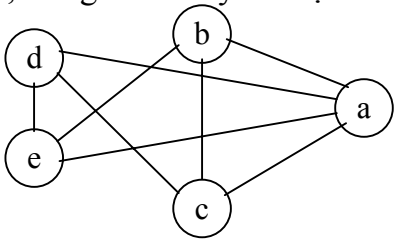


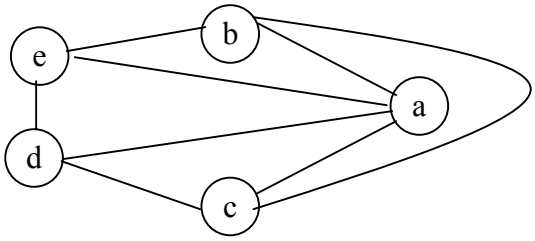
Đồ thị K_4



K_4 vẽ không có đường cắt nhau

3) Xét đồ thị G như trong hình a dưới đây. Có thể biểu diễn G một cách khác như trong hình b, trong đó bất kỳ hai cạnh nào cũng không cắt nhau.



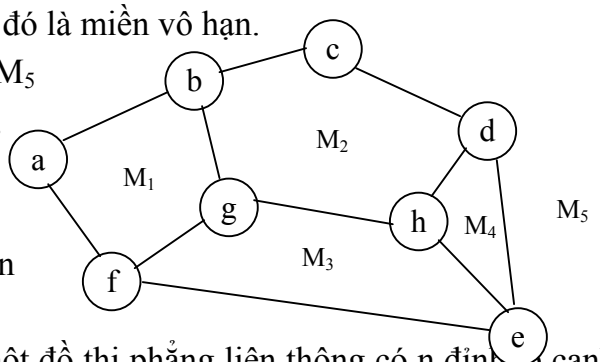


4) Đồ thị đầy đủ K_5 là một thí dụ về đồ thị không phẳng (xem Định lý 7.2.2).

7.1.2. Định nghĩa: Cho G là một đồ thị phẳng. Mỗi phần mặt phẳng giới hạn bởi một chu trình đơn không chứa bên trong nó một chu trình đơn khác, gọi là một miền (hữu hạn) của đồ thị G. Chu trình giới hạn miền là biên của miền. Mỗi đồ thị phẳng liên thông có một miền vô hạn duy nhất (là phần mặt phẳng bên ngoài tất cả các miền hữu hạn). Số cạnh ít nhất tạo thành biên gọi là đai của G; trường hợp nếu G không có chu trình thì đai chính là số cạnh của G.

Thí dụ 2: 1) Một cây chỉ có một miền, đó là miền vô hạn.

2) Đồ thị phẳng ở hình bên có 5 miền, M_5 là miền vô hạn, miền M_1 có biên abgfa, miền M_2 có biên là bcdhgb, … Chu trình đơn abcdhgfa không giới hạn một miền vì chứa bên trong nó chu trình đơn khác là abgfa.



7.1.3. Định lý (Euler, 1752): Nếu một đồ thị phẳng liên thông có n đỉnh, p cạnh và d miền thì ta có hệ thức:

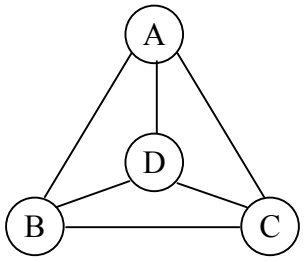
$$n - p + d = 2.$$

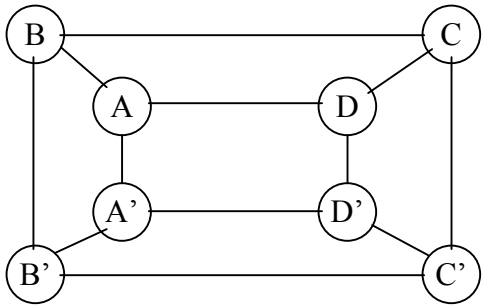
Chứng minh: Cho G là đồ thị phẳng liên thông có n đỉnh, p cạnh và d miền.

Ta bỏ một số cạnh của G để được một cây khung của G. Mỗi lần ta bỏ một cạnh (p giảm 1) thì số miền của G cũng giảm 1 (d giảm 1), còn số đỉnh của G không thay đổi (n không đổi). Như vậy, giá trị của biểu thức n – p + d không thay đổi trong suốt quá trình ta bỏ bớt cạnh của G để được một cây. Cây này có n đỉnh, do đó có n – 1 cạnh và cây chỉ có một miền, vì vậy:

$$n - p + d = n - (n - 1) + 1 = 2.$$

Hệ thức n – p + d = 2 thường gọi là "hệ thức Euler cho hình đa diện", vì được Euler chứng minh đầu tiên cho hình đa diện có n đỉnh, p cạnh và d mặt. Mỗi hình đa diện có thể coi là một đồ thị phẳng. Chẳng hạn hình tứ diện ABCD và hình hộp ABCDA'B'C'D' có thể biểu diễn bằng các đồ thị dưới đây.





7.1.4. Hệ quả: Trong một đồ thị phẳng liên thông tuỳ ý, luôn tồn tại ít nhất một đỉnh có bậc không vượt quá 5.

Chứng minh: Trong đồ thị phẳng mỗi miền được bao bằng ít nhất 3 cạnh. Mặt khác, mỗi cạnh có thể nằm trên biên của tối đa hai miền, nên ta có $3d \le 2p$.

Nếu trong đồ thị phẳng mà tất cả các đỉnh đều có bậc không nhỏ hơn 6 thì do mỗi đỉnh của đồ thị phải là đầu mút của ít nhất 6 cạnh mà mỗi cạnh lại có hai đầu mút nên ta có $6n \le 2p$ hay $3n \le p$. Từ đó suy ra $3d+3n \le 2p+p$ hay $d+n \le p$, trái với hệ thức Euler $d+n=p+2$.

7.2. ĐỒ THỊ KHÔNG PHẲNG.

7.2.1. Định lý: Đồ thị phân đôi đầy đủ $K_{3,3}$ là một đồ thị không phẳng.

Chứng minh: Giả sử $K_{3,3}$ là đồ thị phẳng. Khi đó ta có một đồ thị phẳng với 6 đỉnh ($n=6$) và 9 cạnh ($p=9$), nên theo Định lý Euler đồ thị có số miền là $d=p-n+2=5$.

Ở đây, mõi cạnh chung cho hai miền, mà mỗi miền có ít nhất 4 cạnh. Do đó $4d \le 2p$, tức là $4x5 \le 2x9$, vô lý.

Như vậy định lý này cho ta lời giải của bài toán "Ba nhà ba giếng", nghĩa là không thể thực hiện được việc làm các đường khác đến giếng sao cho các đường này đôi một không giao nhau.

7.2.2. Định lý: Đồ thị đầy đủ K_5 là một đồ thị không phẳng.

Chứng minh: Giả sử K_5 là đồ thị phẳng. Khi đó ta có một đồ thị phẳng với 5 đỉnh ($n=5$) và 10 cạnh ($p=10$), nên theo Định lý Euler đồ thị có số miền là $d=p-n+2=7$.

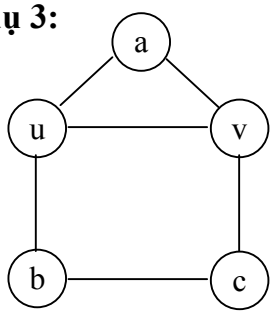
Trong K_5, mỗi miền có ít nhất 3cạnh, mỗi cạnh chung cho hai miền, vì vậy $3d \le 2n$, tức là $3x7 \le 2x10$, vô lý.

7.2.3. Chú ý: Ta đã thấy $K_{3,3}$ và K_5 là không phẳng. Rõ ràng, một đồ thị là không phẳng nếu nó chứa một trong hai đồ thị này như là đồ thị con. Hơn nữa, tất cả các đồ thị không phẳng cần phải chứa đồ thị con nhận được từ $K_{3,3}$ hoặc K_5 bằng một số phép toán cho phép nào đó.

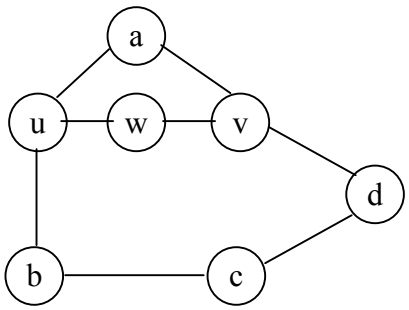
Cho đồ thị G, có cạnh (u,v). Nếu ta xoá cạnh (u,v), rồi thêm đỉnh w cùng với hai cạnh (u,w) và (w,v) thì ta nói rằng ta đã thêm đỉnh mới w (bậc 2) đặt trên cạnh (u,v) của G.

Đồ thị G' được gọi là đồng phôi với đồ thị G nếu G' có được từ G bằng cách thêm các đỉnh mới (bậc 2) đặt trên các cạnh của G.

Thí dụ 3:



G



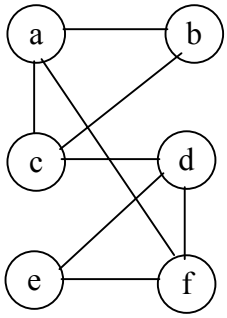
G'

Đồ thị G là đồng phôi với đồ thị G'.

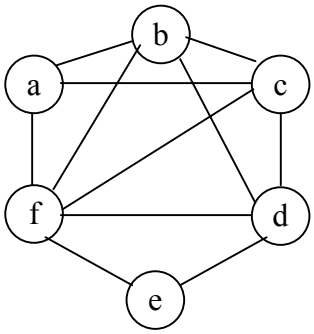
Nhà toán học Ba Lan, Kuratowski, đã thiết lập định lý sau đây vào năm 1930. Định lý này đã biểu thị đặc điểm của các đồ thị phẳng nhờ khái niệm đồ thị đồng phôi.

7.2.4. Định lý (Kuratowski): Đồ thị là không phẳng khi và chỉ khi nó chứa một đồ thị con đồng phôi với $K_{3,3}$ hoặc K_5.

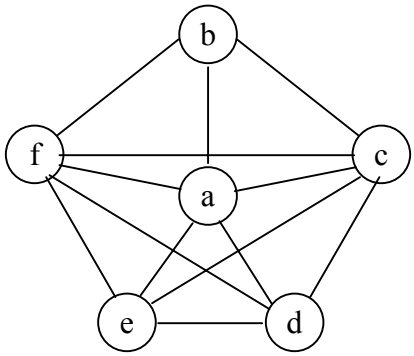
Thí dụ 4:



Hình 1



Hình 2



Hình 3

Đồ thị trong hình 1 và 2 là đồ thị phẳng. Các đồ thị này có 6 đỉnh, nhưng không chứa đồ thị con $K_{3,3}$ được vì có đỉnh bậc 2, trong khi tất cả các đỉnh của $K_{3,3}$ đều có bậc 3; cũng không thể chứa đồ thị con K_5 được vì có những đỉnh bậc nhỏ hơn 4, trong khi tất cả các đỉnh của K_5 đều có bậc 4.

Đồ thị trong hình 3 là đồ thị không phẳng vì nếu xoá đỉnh b cùng các cạnh (b,a), (b,c), (b,f) ta được đồ thị con là K_5.

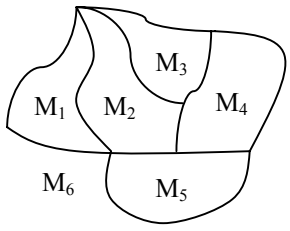
7.3. TÔ MÀU ĐỒ THỊ.

7.3.1. Tô màu bản đồ:

Mỗi bản đồ có thể coi là một đồ thị phẳng. Trong một bản đồ, ta coi hai miền có chung nhau một đường biên là hai miền kề nhau (hai miền chỉ có chung nhau một điểm biên không được coi là kề nhau). Một bản đồ thường được tô màu, sao cho hai miền kề nhau được tô hai màu khác nhau. Ta gọi một cách tô màu bản đồ như vậy là một cách tô màu đúng.

Để đảm bảo chắc chắn hai miền kề nhau không bao giờ có màu trùng nhau, chúng ta tô mỗi miền bằng một màu khác nhau. Tuy nhiên việc làm đó nói chung là không hợp lý. Nếu bản đồ có nhiều miền thì sẽ rất khó phân biệt những màu gần giống nhau. Do vậy người ta chỉ dùng một số màu cần thiết để tô bản đồ. Một bài toán được đặt ra là: xác định số màu tối thiểu cần có để tô màu đúng một bản đồ.

Thí dụ 5: Bản đồ trong hình bên có 6 miền, nhưng chỉ cần có 3 màu (vàng, đỏ, xanh) để tô đúng bản đồ này. Chẳng hạn, màu vàng được tô cho M_1 và M_4, màu đỏ được tô cho M_2 và M_6, màu xanh được tô cho M_3 và M_5.

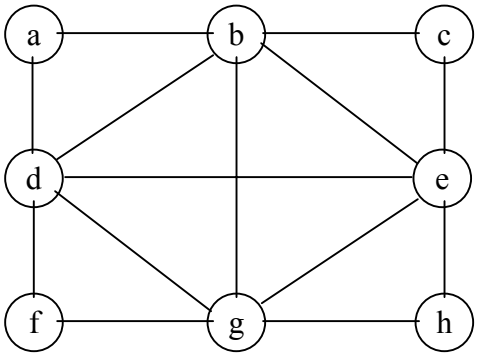


7.3.2. Tô màu đồ thị:

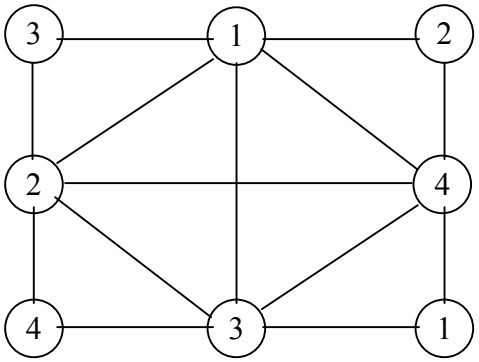
Mỗi bản đồ trên mặt phẳng có thể biểu diễn bằng một đồ thị, trong đó mỗi miền của bản đồ được biểu diễn bằng một đỉnh; các cạnh nối hai đỉnh, nếu các miền được biểu diễn bằng hai đỉnh này là kề nhau. Đồ thị nhận được bằng cách này gọi là đồ thị đối ngẫu của bản đồ đang xét. Rõ ràng mọi bản đồ trên mặt phẳng đều có đồ thị đối ngẫu phẳng. Bài toán tô màu các miền của bản đồ là tương đương với bài toán tô màu các đỉnh của đồ thị đối ngẫu sao cho không có hai đỉnh liền kề nhau có cùng một màu, mà ta gọi là tô màu đúng các đỉnh của đồ thị.

Số màu ít nhất cần dùng để tô màu đúng đồ thị G được gọi là sắc số của đồ thị G và ký hiệu là $\chi(G)$.

Thí dụ 6:



Ta thấy rằng 4 đỉnh b, d, g, e đôi một kề nhau nên phải được tô bằng 4 màu khác nhau. Do đó $\chi(G) \geq 4$. Ngoài ra, có thể dùng 4 màu đánh số 1, 2, 3, 4 để tô màu G như sau:



Như vậy $\chi(G) = 4$.

7.3.3. Mệnh đề: Nếu đồ thị G chứa một đồ thị con đồng phôi với đồ thị đầy đủ K_n thì $\chi(G) \geq n$.

Chứng minh: Gọi H là đồ thị con của G đồng phôi với K_n thì $\chi(H) \geq n$. Do đó $\chi(G) \geq n$.

7.3.4. Mệnh đề: Nếu đơn đồ thị G không chứa chu trình độ dài lẻ thì $\chi(G) = 2$.

Chứng minh: Không mất tính chất tổng quát có thể giả sử G liên thông. Cố định đỉnh u của G và tô nó bằng màu 0 trong hai màu 0 và 1. Với mỗi đỉnh v của G, tồn tại một đường đi từ u đến v, nếu đường này có độ dài chẵn thì tô màu 0 cho v, nếu đường này có độ dài lẻ thì tô màu 1 cho v. Nếu có hai đường đi mang tính chẵn lẻ khác nhau cùng nối

u với v thì dễ thấy rằng G phải chứa ít nhất một chu trình độ dài lẻ. Điều mâu thuẫn này cho biết hai màu 0 và 1 tô đúng đồ thị G.

7.3.5. Mệnh đề: Với mỗi số nguyên dương n, tồn tại một đồ thị không chứa K_3 và có sắc số bằng n.

Chứng minh: Ta chứng minh mệnh đề bằng quy nạp theo n.

Trường hợp n=1 là hiển nhiên.

Giả sử ta có đồ thị G_n với k_n đỉnh, không chứa K_3 và có sắc số là n. Ta xây dựng đồ thị G_{n+1} gồm n bản sao của G_n và thêm k_n^n đỉnh mới theo cách sau: mỗi bộ thứ tự $(v_1, v_2, \ldots, v_n)$, với v_i thuộc bản sao G_n thứ i, sẽ tương ứng với một đỉnh mới, đỉnh mới này được nối bằng n cạnh mới đến các đỉnh $v_1, v_2, \ldots, v_n$. Dễ thấy rằng G_{n+1} không chứa K_3 và có sắc số là n+1.

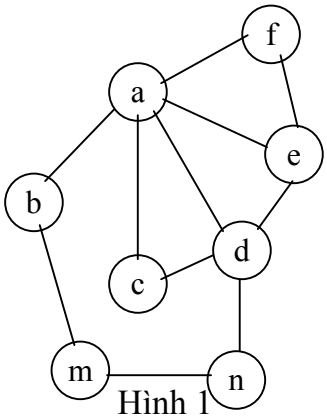
7.3.6. Định lý (Định lý 5 màu của Kempe-Heawood): Mọi đồ thị phẳng đều có thể tô đúng bằng 5 màu.

Chứng minh: Cho G là một đồ thị phẳng. Không mất tính chất tổng quát có thể xem G là liên thông và có số đỉnh $n \geq 5$. Ta chứng minh G được tô đúng bởi 5 màu bằng quy nạp theo n.

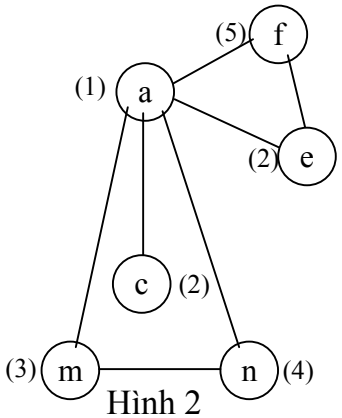
Trường hợp n=5 là hiển nhiên. Giả sử định lý đúng cho tất cả các đồ thị phẳng có số đỉnh nhỏ hơn n. Xét G là đồ thị phẳng liên thông có n đỉnh.

Theo Hệ quả 7.1.4, trong G tồn tại đỉnh a với $\deg(a) \leq 5$. Xoá đỉnh a và các cạnh liên thuộc với nó, ta nhận được đồ thị phẳng G' có n−1 đỉnh. Theo giả thiết quy nạp, có thể tô đúng các đỉnh của G' bằng 5 màu. Sau khi tô đúng G' rồi, ta tìm cách tô đỉnh a bằng một màu khác với màu của các đỉnh kề nó, nhưng vẫn là một trong 5 màu đã dùng. Điều này luôn thực hiện được khi $\deg(a) < 5$ hoặc khi $\deg(a)=5$ nhưng 5 đỉnh kề a đã được tô bằng 4 màu trở xuống.

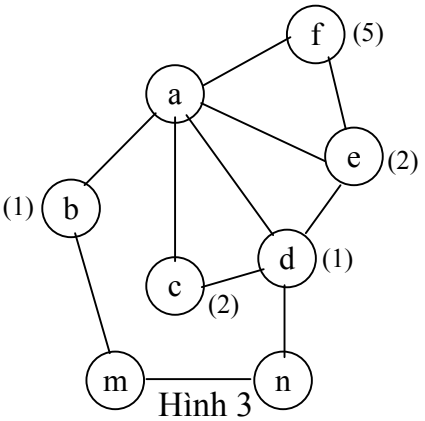
Chỉ còn phải xét trường hợp $\deg(a)=5$ mà 5 đỉnh kề a là b, c, d, e ,f đã được tô bằng 5 màu rồi. Khi đó trong 5 đỉnh b, c, d, e ,f phải có 2 đỉnh không kề nhau, vì nếu 5 đỉnh đó đôi một kề nhau thì b c d e f là đồ thị đầy đủ K_5 và đây là một đồ thị không phẳng, do đó G không phẳng, trái với giả thiết. Giả sử b và d không kề nhau (Hình 1).



Hình 1



Hình 2



Hình 3

Xoá 2 đỉnh b và d và cho kề a những đỉnh trước đó kề b hoặc kề d mà không kề a (Hình 2), ta được đồ thị mới G'' có n−2 đỉnh. Theo giả thiết quy nạp, ta có thể tô đúng G'' bằng 5 màu. Sau khi các đỉnh của G'' được tô đúng rồi (Hình 2), ta dựng lại 2 đỉnh b và d, rồi tô b và d bằng màu đã tô cho a (màu 1, Hình 3), còn a thì được tô lại bằng màu khác với màu của b, c, d, e, f. Vì b và d không kề nhau đã được tô bằng cùng màu 1, nên với 5 đỉnh này chỉ mới dùng hết nhiều lắm 4 màu.. Do đó G được tô đúng bằng 5 màu.

7.3.7. Định lý (Định lý 4 màu của Appel-Haken): Mọi đồ thị phẳng đều có thể tô đúng bằng 4 màu.

Định lý Bốn màu đầu tiên được đưa ra như một phỏng đoán vào năm 1850 bởi một sinh viên người Anh tên là F. Guthrie và cuối cùng đã được hai nhà toán học Mỹ là Kenneth Appel và Wolfgang Haken chứng minh vào năm 1976. Trước năm 1976 cũng đã có nhiều chứng minh sai, mà thông thường rất khó tìm thấy chỗ sai, đã được công bố. Hơn thế nữa đã có nhiều cố gắng một cách vô ích để tìm phản thí dụ bằng cách cố vẽ bản đồ cần hơn bốn màu để tô nó.

Có lẽ một trong những chứng minh sai nổi tiếng nhất trong toán học là chứng minh sai "bài toán bốn màu" được công bố năm 1879 bởi luật sư, nhà toán học nghiệp dư Luân Đôn tên là Alfred Kempe. Nhờ công bố lời giải của "bài toán bốn màu", Kempe được công nhận là hội viên Hội Khoa học Hoàng gia Anh. Các nhà toán học chấp nhận cách chứng minh của ông ta cho tới 1890, khi Percy Heawood phát hiện ra sai lầm trong chứng minh của Kempe. Mặt khác, dùng phương pháp của Kempe, Heawood đã chứng minh được "bài toán năm màu" (tức là mọi bản đồ có thể tô đúng bằng 5 màu).

Như vậy, Heawood mới giải được "bài toán năm màu", còn "bài toán bốn màu" vẫn còn đó và là một thách đố đối với các nhà toán học trong suốt gần một thế kỷ. Việc tìm lời giải của "bài toán bốn màu" đã ảnh hưởng đến sự phát triển theo chiều hướng khác nhau của lý thuyết đồ thị.

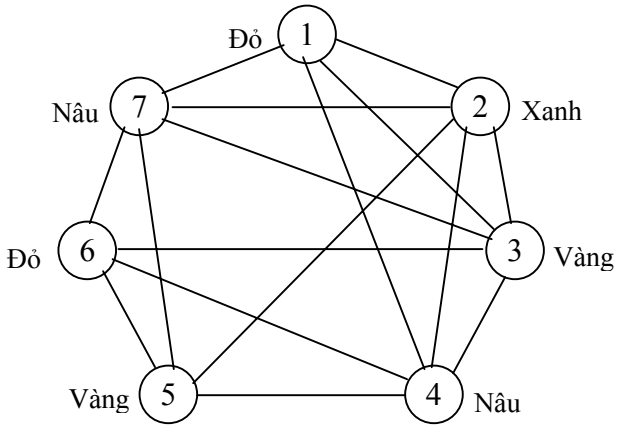
Mãi đến năm 1976, khai thác phương pháp của Kempe và nhờ công cụ máy tính điện tử, Appel và Haken đã tìm ra lời giải của "bài toán bốn màu". Chứng minh của họ dựa trên sự phân tích từng trường hợp một cách cẩn thận nhờ máy tính. Họ đã chỉ ra rằng nếu "bài toán bốn màu" là sai thì sẽ có một phản thí dụ thuộc một trong gần 2000 loại khác nhau và đã chỉ ra không có loại nào dẫn tới phản thí dụ cả. Trong chứng minh của mình họ đã dùng hơn 1000 giờ máy. Cách chứng minh này đã gây ra nhiều cuộc tranh cãi vì máy tính đã đóng vai trò quan trọng biết bao. Chẳng hạn, liệu có thể có sai lầm trong chương trình và điều đó dẫn tới kết quả sai không? Lý luận của họ có thực sự là một chứng minh hay không, nếu nó phụ thuộc vào thông tin ra từ một máy tính không đáng tin cậy?

7.3.8. Những ứng dụng của bài toán tô màu đồ thị:

1) Lập lịch thi: Hãy lập lịch thi trong trường đại học sao cho không có sinh viên nào có hai môn thi cùng một lúc.

Có thể giải bài toán lập lịch thi bằng mô hình đồ thị, với các đỉnh là các môn thi, có một cạnh nối hai đỉnh nếu có sinh viên phải thi cả hai môn được biểu diễn bằng hai đỉnh này. Thời gian thi của mỗi môn được biểu thị bằng các màu khác nhau. Như vậy việc lập lịch thi sẽ tương ứng với việc tô màu đồ thị này.

Chẳng hạn, có 7 môn thi cần xếp lịch. Giả sử các môn học đuợc đánh số từ 1 tới 7 và các cặp môn thi sau có chung sinh viên: 1 và 2, 1 và 3, 1 và 4, 1 và 7, 2 và 3, 2 và 4, 2 và 5, 2 và 7, 3 và 4, 3 và 6, 3 và 7, 4 và 5, 4 và 6, 5 và 6, 5 và 7, 6 và 7. Hình dưới đây biểu diễn đồ thị tương ứng. Việc lập lịch thi chính là việc tô màu đồ thị này. Vì số màu của đồ thị này là 4 nên cần có 4 đợt thi.



2) Phân chia tần số: Các kênh truyền hình từ số 1 tới số 12 được phân chia cho các đài truyền hình sao cho không có đài phát nào cách nhau không quá 240 km lại dùng cùng một kênh. Có thể chia kênh truyền hình như thế nào bằng mô hình tô màu đồ thị.

Ta xây dựng đồ thị bằng cách coi mỗi đài phát là một đỉnh. Hai đỉnh được nối với nhau bằng một cạnh nếu chúng ở cách nhau không quá 240 km. Việc phân chia kênh tương ứng với việc tô màu đồ thị, trong đó mỗi màu biểu thị một kênh.

3) Các thanh ghi chỉ số: Trong các bộ dịch hiệu quả cao việc thực hiện các vòng lặp được tăng tốc khi các biến dùng thường xuyên được lưu tạm thời trong các thanh ghi chỉ số của bộ xử lý trung tâm (CPU) mà không phải ở trong bộ nhớ thông thường. Với một vòng lặp cho trước cần bao nhiêu thanh ghi chỉ số? Bài toán này có thể giải bằng mô hình tô màu đồ thị. Để xây dựng mô hình ta coi mỗi đỉnh của đồ thị là một biến trong vòng lặp. Giữa hai đỉnh có một cạnh nếu các biến biểu thị bằng các đỉnh này phải được lưu trong các thanh ghi chỉ số tại cùng thời điểm khi thực hiện vòng lặp. Như vậy số màu của đồ thị chính là số thanh ghi cần có vì những thanh ghi khác nhau được phân cho các biến khi các đỉnh biểu thị các biến này là liền kề trong đồ thị.

BÀI TẬP CHƯƠNG VII:

1. Cho G là một đơn đồ thị phẳng liên thông có 10 mặt, tất cả các đỉnh đều có bậc 4. Tìm số đỉnh của đồ thị G.

2. Cho G là một đơn đồ thị phẳng liên thông có 9 đỉnh, bậc các đỉnh là 2, 2, 2, 3, 3, 3, 4, 4, 5. Tìm số cạnh và số mặt của G.

3. Tìm số đỉnh, số cạnh và đai của:

a) K_n; **b)** $K_{m,n}$.

4. Chứng minh rằng:

a) K_n là phẳng khi và chỉ khi $n \le 4$.

b) $K_{m,n}$ là phẳng khi và chỉ khi $m \le 2$ hay $n \le 2$.

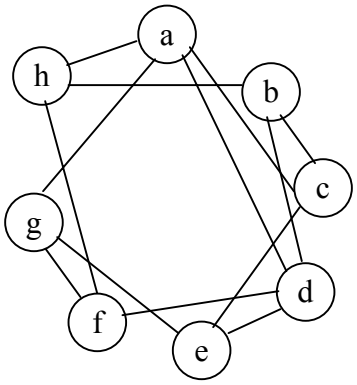
5. Đồ thị nào trong các đồ thị không phẳng sau đây có tính chất: Bỏ một đỉnh bất kỳ và các cạnh liên thuộc của nó tạo ra một đồ thị phẳng.

a) K_5; **b)** K_6; **c)** $K_{3,3}$.

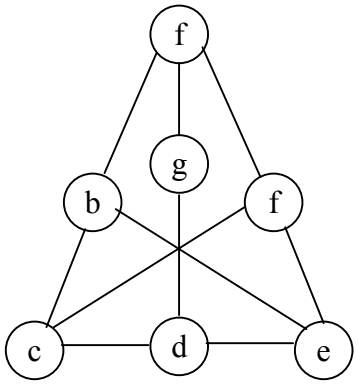
6. Cho G là một đơn đồ thị phẳng liên thông có n đỉnh và m cạnh, trong đó $n \ge 3$. Chứng minh rằng:

$$m \le 3n - 6.$$

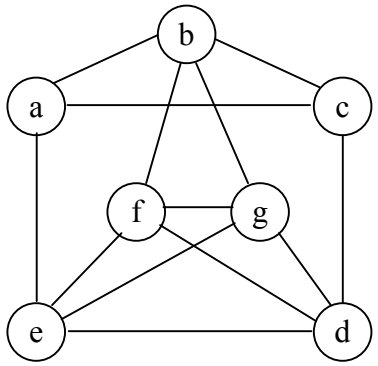
7. Trong các đồ thị ở hình dưới đây, đồ thị nào là phẳng, đồ thị nào không phẳng? Nếu đồ thị là phẳng thì có thể kẻ thêm ít nhất là bao nhiêu cạnh để được đồ thị không phẳng?



G_1



G_2



G_3

8. Chứng minh rằng đồ thị Peterson (đồ thị trong Bài tập 8, Chương IV) là đồ thị không phẳng.

9. Cho G là một đồ thị phẳng liên thông có n đỉnh, m cạnh và đai là g, với $g \ge 3$. Chứng minh rằng:

$$m \le \frac{g}{g-2}(n-2).$$

10. Đa diện lồi có d mặt ($d \ge 5$), mà từ mỗi đỉnh có đúng 3 cạnh. Hai người chơi trò chơi như sau: mỗi người lần lượt tô đỏ một mặt trong các mặt còn lại. Người thắng là

người tô được 3 mặt có chung một đỉnh. Chứng minh rằng tồn tại cách chơi mà người được tô trước luôn luôn thắng.

11. Chứng minh rằng:

a) Một đồ thị phẳng có thể tô đúng các đỉnh bằng hai màu khi và chỉ khi đó là đồ thị phân đôi.

b) Một đồ thị phẳng có thể tô đúng các miền bằng hai màu khi và chỉ khi đó là đồ thị Euler.

12. Tìm sắc số của các đồ thị cho trong Bài tập 7.

13. Tìm sắc số của các đồ thị K_n, $K_{m,n}$, C_n, và W_n.

14. Khoa Toán có 6 hội đồng họp mỗi tháng một lần. Cần có bao nhiêu thời điểm họp khác nhau để đảm bảo rằng không ai bị xếp lịch họp hai hội đồng cùng một lúc, nếu các hội đồng là:

$$H_1 = \{H, L, P\}, H_2 = \{L, M, T\}, H_3 = \{H, T, P\}.$$

15. Một vườn bách thú muốn xây dựng chuồng tự nhiên để trưng bày các con thú. Không may, một số loại thú sẽ ăn thịt các con thú khác nếu có cơ hội. Có thể dùng mô hình đồ thị và tô màu đồ thị như thế nào để xác định số chuồng khác nhau cần có và cách nhốt các con thú vào các chuồng thú tự nhiên này?

16. Chứng minh rằng một đơn đồ thị phẳng có 8 đỉnh và 13 cạnh không thể được tô đúng bằng hai màu.

17. Chứng minh rằng nếu G là một đơn đồ thị phẳng có ít hơn 12 đỉnh thì tồn tại trong G một đỉnh có bậc ≤ 4. Từ đó hãy suy ra rằng đồ thị G có thể tô đúng bằng 4 màu.

CHƯƠNG VIII
ĐẠI SỐ BOOLE

Các mạch điện trong máy tính và các dụng cụ điện tử khác đều có các đầu vào, mỗi đầu vào là số 0 hoặc số 1, và tạo ra các đầu ra cũng là các số 0 và 1. Các mạch điện đó đều có thể được xây dựng bằng cách dùng bất kỳ một phần tử cơ bản nào có hai trạng thái khác nhau. Chúng bao gồm các chuyển mạch có thể ở hai vị trí mở hoặc đóng và các dụng cụ quang học có thể là sáng hoặc tối. Năm 1938 Claude Shannon chứng tỏ rằng có thể dùng các quy tắc cơ bản của lôgic do George Boole đưa ra vào năm 1854 trong cuốn "Các quy luật của tư duy" của ông để thiết kế các mạch điện. Các quy tắc này đã tạo nên cơ sở của đại số Boole. Sự hoạt động của một mạch điện được xác định bởi một hàm Boole chỉ rõ giá trị của đầu ra đối với mỗi tập đầu vào. Bước đầu tiên trong việc xây dựng một mạch điện là biểu diễn hàm Boole của nó bằng một biểu thức được lập bằng cách dùng các phép toán cơ bản của đại số Boole. Biểu thức mà ta sẽ nhận được có thể chứa nhiều phép toán hơn mức cần thiết để biểu diễn hàm đó. Ở cuối chương này, ta sẽ có các phương pháp tìm một biểu thức với số tối thiểu các phép tổng và tích được dùng để biểu diễn một hàm Boole. Các thủ tục được mô tả là bản đồ Karnaugh và phương pháp Quine-McCluskey, chúng đóng vai trò quan trọng trong việc thiết kế các mạch điện có hiệu quả cao.

8.1. KHÁI NIỆM ĐẠI SỐ BOOLE.

8.1.1. Định nghĩa: Tập hợp khác rỗng S cùng với các phép toán ký hiệu nhân (**.**), cộng (+), lấy bù (') được gọi là một đại số Boole nếu các tiên đề sau đây được thoả mãn với mọi a, b, c $\in$ S.

1. Tính giao hoán: **a)** a**.**b = b**.**a,
b) a+b = b+a.

2. Tính kết hợp: **a)** (a**.**b)**.**c = a**.**(b**.**c),
b) (a+b)+c = a+(b+c).

3. Tính phân phối: **a)** a**.**(b+c) = (a**.**b)+(a**.**c),
b) a+(b**.**c) = (a+b)**.**(a+c).

4. Tồn tại phần tử trung hoà: Tồn tại hai phần tử khác nhau của S, ký hiệu là 1 và 0 sao cho: **a)** a**.**1 = 1**.**a = a,
b) a+0 = 0+a = a.

1 gọi là phần tử trung hoà của phép **.** và 0 gọi là phần tử trung hoà của phép +.

5. Tồn tại phần tử bù: Với mọi a $\in$ S, tồn tại duy nhất phần tử a' $\in$ S sao cho:
a) a**.**a' = a'**.**a = 0,
b) a+a' = a'+a = 1.

a’ gọi là phần tử bù của a.

Thí dụ 1:

1) Đại số lôgic là một đại số Boole, trong đó S là tập hợp các mệnh đề, các phép toán $\wedge$ (hội), $\vee$ (tuyển), − (phủ định) tương ứng với **.** , +, **’**, các hằng đ (đúng), s (sai) tương ứng với các phần tử trung hoà 1, 0.

2) Đại số tập hợp là một đại số Boole, trong đó S là tập hợp $\mathcal{P}(X)$ gồm các tập con của tập khác rỗng X, các phép toán $\cap$ (giao), $\cup$ (hợp), − (bù) tương ứng với **.** , +, **’**, các tập X, Ø tương ứng với các phần tử trung hoà 1, 0.

3) Cho B = {0,1}, các phép toán **.** , +, **’** trên B được định nghĩa như sau:

$$\begin{array}{lll} 1.1 = 1, & 1+1 = 1, & 1' = 0, \\ 1.0 = 0, & 1+0 = 1, & 0' = 1. \\ 0.1 = 0, & 0+1 = 1, & \\ 0.0 = 0, & 0+0 = 0, & \end{array} \qquad (1)$$

Khi đó B là một đại số Boole. Đây cũng chính là đại số lôgic, trong đó 1, 0 tương ứng với đ (đúng), s (sai). Mỗi phần tử 0,1 của B gọi là một bit. Ta thường viết $\bar{x}$ thay cho *x’*.

Tổng quát, gọi B^n là tập hợp các xâu n bit (xâu nhị phân độ dài n). Ta định nghĩa tích, tổng của hai chuỗi và bù của một chuỗi theo từng bit một như trong Bảng 1, mà thường được gọi là các phép toán AND-bit, OR-bit, NOT-bit. B^n với các phép toán này tạo thành một đại số Boole.

4) Cho M là tập hợp các số thực có cận trên p, cận dưới q và tâm đối xứng O. Các phép toán **.** , +, **’** trên M được định nghĩa như sau:

$a.b = \min(a, b)$, $a+b = \max(a, b)$, a’ là điểm đối xứng của a qua O.

Khi đó M là một đại số Boole, trong đó q, p tương ứng với các phần tử trung hoà 1, 0.

8.1.2. Chú ý: Trước hết cần lưu ý điều quan trọng sau đây: các tiên đề của đại số Boole được xếp theo từng cặp a) và b). Từ mỗi tiên đề a), nếu ta thay **.** bởi +, thay + bởi **.**, thay 1 bởi 0 và thay 0 bởi 1 thì ta được tiên đề b) tương ứng.

Ta gọi cặp tiên đề a), b) là đối ngẫu của nhau. Do đó nếu ta chứng minh được một định lý trong đại số Boole thì ta có ngay một định lý khác, đối ngẫu của nó, bằng cách thay **.** và 1 tương ứng bởi + và 0 (và ngược lại). Ta có:

Quy tắc đối ngẫu: Đối ngẫu của một định lý là một định lý.

8.1.3. Định lý:

6. (Tính nuốt)

a) a.0 = 0,

b) a+1 = 1

7. (Tính luỹ đẳng)

a) a.a = a,

b) a+a = a.

8. (Hệ thức De Morgan)

a) (a.b)’ = a’+b’,

b) (a+b)’ = a’.b’.

9. (Hệ thức bù kép)

(a’)’ = a.

10. **a)** 1’ = 0,

b) 0’ = 1.

11. (Tính hút)

a) a.(a+b) = a,

b) a+(a.b) = a.

Chứng minh:

6. 0 = a.a (tiên đề 5a))

= a.(a’+0) (tiên đề 4b))

= (a.a’)+(a.0) (tiên đề 3a))

= 0+(a.0) (tiên đề 5a))

= a.0 (tiên đề 4b)).

7. a = a.1 (tiên đề 4a))

= a.(a+a’) (tiên đề 5b))

= (a.a)+(a.a’) (tiên đề 3a))

= (a.a)+0 (tiên đề 5a))

= a.a (tiên đề 4b))

8. Ta chứng minh rằng a’+b’ là bù của a.b bằng cách chứng minh rằng:

(a.b).(a’+b’) = 0 (theo 5a)) và (a.b)+(a’+b’) = 1 (theo 5b)).

Thật vậy, (a.b).(a’+b’) = (a.b.a’)+(a.b.b’) = (a.a’.b)+(a.b.b’) = (0.b)+(a.0) = 0+0 = 0,

(a.b)+(a’+b’) = (a’+b’)+(a.b) = (a’+b’+a).(a’+b’+b) = (1+b’).(a’+1) = 1.1 = 1.

Vì a.b chỉ có một phần tử bù duy nhất nên (a.b)’ = a’+b’.

9. Có ngay từ tiên đề 5.

10. Có từ các hệ thức 1.0 = 0 và 1+0 = 1.

11. a.(a+b) = (a+0).(a+b) = a+(0.b) = a+0 = a.

8.1.4. Chú ý: Hệ tiên đề của đại số Boole nêu ra ở đây không phải là một hệ tối thiểu. Chẳng hạn, các tiên đề về tính kết hợp có thể suy ra từ các tiên đề khác. Thật vậy, với A=(a.b).c và B=a.(b.c), ta có: a+A = a+((a.b).c) = (a+(a.b)).(a+c) = a.(a+c) = a, a+B = a+(a.(b.c)) = (a+a).(a+(b.c)) = a.(a+(b.c)) = a, a’+A = a’+((a.b).c) = (a’+(a.b)).(a’+c) = ((a’+a).(a’+b)).(a’+c) = (1.(a’+b)).(a’+c) = (a’+b).(a’+c) = a’+(b.c), a’+B = a’+(a.(b.c)) = (a’+a).(a’+(b.c)) = 1.(a’+(b.c)) = a’+(b.c).

Do đó a+A = a+B và a’+A = a’+B. Từ đó suy ra rằng:

A = A+0 = A+(a.a') = (A+a).(A+a') = (a+A).(a'+A) = (a+B).(a'+B)=(a.a')+B=0+B= B hay ta có 2a) và đối ngẫu ta có 2b). Ngoài ra, tính duy nhất của phần tử bù cũng được suy ra từ các tiên đề khác.

Tương tự trong đại số lôgic, trong đại số Boole ta cũng xét các công thức, được thành lập từ các biến a, b, c, … nhờ các phép toán . , +, '. Trong công thức, ta quy ước thực hiện các phép toán theo thứ tự: ', ., +; a.b được viết là ab, gọi là tích của a và b còn a+b gọi là tổng của a và b. Ta có thể biến đổi công thức, rút gọn công thức tương tự trong đại số lôgic. Ta cũng xét các tích sơ cấp và tổng sơ cấp tương tự "hội sơ cấp" và "tuyển sơ cấp". Mọi công thức đều có thể đưa về dạng tích chuẩn tắc hoàn toàn hoặc về dạng tổng chuẩn tắc hoàn toàn tương tự dạng "hội và tuyển chuẩn tắc hoàn toàn". Mỗi công thức trong đại số Boole cũng được gọi là biểu diễn một hàm Boole.

8.2. HÀM BOOLE.

8.2.1. Định nghĩa: Ký hiệu B = {0, 1} và $B^n = \{(x_1, x_2, \ldots, x_n) \mid x_i \in B, 1 \le i \le n\}$, ở đây B và B^n là các đại số Boole (xem 2) và 3) của Thí dụ 1). Biến x được gọi là một biến Boole nếu nó nhận các giá trị chỉ từ B. Một hàm từ B^n vào B được gọi là một hàm Boole (hay hàm đại số lôgic) bậc n.

Các hàm Boole cũng có thể được biểu diễn bằng cách dùng các biểu thức được tạo bởi các biến và các phép toán Boole (xem Bảng 1 trong Thí dụ 1). Các biểu thức Boole với các biến $x_1, x_2, \ldots, x_n$ được định nghĩa bằng đệ quy như sau:

- 0, 1, $x_1, x_2, \ldots, x_n$ là các biểu thức Boole.
- Nếu P và Q là các biểu thức Boole thì $\overline{P}$, PQ và $P+Q$ cũng là các biểu thức Boole.

Mỗi một biểu thức Boole biểu diễn một hàm Boole. Các giá trị của hàm này nhận được bằng cách thay 0 và 1 cho các biến trong biểu thức đó.

Hai hàm n biến F và G được gọi là bằng nhau nếu $F(a_1, a_2, \ldots, a_n)=G(a_1, a_2, \ldots,a_n)$ với mọi $a_1, a_2, \ldots, a_n \in B$. Hai biểu thức Boole khác nhau biểu diễn cùng một hàm Boole được gọi là tương đương. Phần bù của hàm Boole F là hàm $\overline{F}$ với $\overline{F}(x_1, x_2, \ldots, x_n) = \overline{F(x_1, x_2, \ldots, x_n)}$. Giả sử F và G là các hàm Boole bậc n. Tổng Boole F+G và tích Boole FG được định nghĩa bởi:

$(F+G)(x_1, x_2, \ldots, x_n) = F(x_1, x_2, \ldots, x_n)+G(x_1, x_2, \ldots, x_n)$,

$(FG)(x_1, x_2, \ldots, x_n) = F(x_1, x_2, \ldots, x_n)G(x_1, x_2, \ldots, x_n)$.

Thí dụ 2:

Theo quy tắc nhân của phép đếm ta suy ra rằng có 2^n bộ n phần tử khác nhau gồm các số 0 và 1. Vì hàm Boole là việc gán 0 hoặc 1 cho mỗi bộ trong số 2^n bộ n phần tử đó, nên lại theo quy tắc nhân sẽ có 2^{2^n} các hàm Boole khác nhau.

Bậc	Số các hàm Boole
1	4
2	16
3	256
4	65.536
5	4.294.967.296
6	18.446.744.073.709.551.616

Bảng sau cho giá trị của 16 hàm Boole bậc 2 phân biệt:

x	y	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}	F_{15}	F_{16}
0	0	0	1	0	0	0	1	1	1	1	1	0	0	1	0	1	0
0	1	0	1	0	1	1	1	0	0	1	0	0	1	0	0	1	1
1	0	0	1	0	1	1	0	0	0	1	1	1	0	1	1	0	0
1	1	0	1	1	1	0	1	1	0	0	1	1	1	0	0	0	0

trong đó có một số hàm thông dụng như sau:

- Hàm F_1 là hàm hằng 0,
- Hàm F_2 là hàm hằng 1,
- Hàm F_3 là hàm hội, $F_3(x,y)$ được viết là xy (hay $x \wedge y$),
- Hàm F_4 là hàm tuyển, $F_4(x,y)$ được viết là x+y (hay $x \vee y$),
- Hàm F_5 là hàm tuyển loại, $F_5(x,y)$ được viết là $x \oplus y$,
- Hàm F_6 là hàm kéo theo, $F_6(x,y)$ được viết là $x \Rightarrow y$,
- Hàm F_7 là hàm tương đương, $F_7(x,y)$ được viết là $x \Leftrightarrow y$,
- Hàm F_8 là hàm Vebb, $F_8(x,y)$ được viết là $x \downarrow y$,
- Hàm F_9 là hàm Sheffer, $F_9(x,y)$ được viết là $x \uparrow y$.

Thí dụ 3: Các giá trị của hàm Boole bậc 3 $F(x, y, z) = xy + \bar{z}$ được cho bởi bảng sau:

x	y	z	xy	$\bar{z}$	$F(x, y, z) = xy + \bar{z}$
0	0	0	0	1	1
0	0	1	0	0	0
0	1	0	0	1	1
0	1	1	0	0	0
1	0	0	0	1	1
1	0	1	0	0	0
1	1	0	1	1	1
1	1	1	1	0	1

8.2.2. Định nghĩa: Cho x là một biến Boole và $\sigma \in B$. Ký hiệu:

$$x^{\sigma} = \begin{cases} x \; khi \; \sigma = 1, \\ \bar{x} \; khi \; \sigma = 0. \end{cases}$$

Dễ thấy rằng $x^{\sigma} = 1 \Leftrightarrow x = \sigma$. Với mỗi hàm Boole F bậc n, ký hiệu:

$$T_F = \{(x_1, x_2, \ldots, x_n) \in B^n \mid F(x_1, x_2, \ldots, x_n) = 1\}$$

Và gọi nó là tập đặc trưng của hàm F. Khi đó ta có:

$$T_{\bar{F}} = \overline{T_F}, \; T_{F+G} = T_F \cup T_G, \; T_{FG} = T_F \cap T_G.$$

Cho n biến Boole $x_1, x_2, \ldots, x_n$. Một biểu thức dạng:

$$x_{i_1}^{\sigma_1} x_{i_2}^{\sigma_2} \ldots x_{i_k}^{\sigma_k}$$

trong đó $\sigma_1,\sigma_2,\ldots,\sigma_k \in B$, $1 \le i_1 < i_2 < \cdots < i_k \le n$ được gọi là một hội sơ cấp của n biến $x_1, x_2, \ldots, x_n$. Số các biến xuất hiện trong một hội sơ cấp đươc gọi là hạng của của hội sơ cấp đó.

Cho F là một hàm Boole bậc n. Nếu F được biểu diễn dưới dạng tổng (tuyển) của một số hội sơ cấp khác nhau của n biến thì biểu diễn đó được gọi là dạng tổng (tuyển) chuẩn tắc của F. Dạng tổng (tuyển) chuẩn tắc hoàn toàn là dạng chuẩn tắc duy nhất của F mà trong đó các hội sơ cấp đều có hạng n.

Thí dụ 4: $\bar{x}y + x\bar{y}$ là một dạng tổng chuẩn tắc của hàm $x \oplus y$.

$\bar{x} + \bar{y}$ và $\bar{x}\bar{y} + \bar{x}y + x\bar{y}$ là các dạng tổng chuẩn tắc của hàm Sheffer $x \uparrow y$.

8.2.3. Mệnh đề: Mọi hàm Boole F bậc n đều có thể biểu diễn dưới dạng:

$$F(x_1,x_2,\ldots,x_n) = \sum_{(\sigma_1,\ldots,\sigma_n)\in B^i} x_1^{\sigma_1}\ldots x_i^{\sigma_i} F(\sigma_1,\ldots,\sigma_i,x_{i+1},\ldots,x_n) \quad (1),$$

trong đó i là số tự nhiên bất kỳ, $1 \le i \le n$.

Chứng minh: Gọi G là hàm Boole ở vế phải của (1). Cho $(x_1, x_2, \ldots, x_n) \in T_F$. Khi đó số hạng ứng với bộ giá trị $\sigma_1 = x_1, \ldots, \sigma_i = x_i$ trong tổng ở vế phải của (1) bằng 1, do đó $(x_1, x_2, \ldots, x_n) \in T_G$. Đảo lại, nếu $(x_1, x_2, \ldots, x_n) \in T_G$ tức là vế phải bằng 1 thì phải xảy ra bằng 1 tại một số hạng nào đó, chẳng hạn tại số hạng ứng với bộ giá trị $(\sigma_1, \ldots, \sigma_i)$, khi đó $x_1=\sigma_1, \ldots, x_i=\sigma_i$ và $f(\sigma_1,\ldots,\sigma_i, x_{i+1},\ldots, x_n)=1$ hay $(x_1, x_2, \ldots, x_n) \in T_F$. Vậy $T_F=T_G$ hay $F=G$.

Cho i=1 trong mệnh đề trên và nhận xét rằng vai trò của các biến x_i là như nhau, ta được hệ quả sau.

8.2.4. Hệ quả: Mọi hàm Boole F bậc n đều có thể được khai triển theo một biến x_i:

$$F(x_1,\ldots,x_n) = \bar{x_i}F(x_1,\ldots,x_{i-1},0,x_{i+1},\ldots,x_n) + x_iF(x_1,\ldots,x_{i-1},1,x_{i+1},\ldots,x_n).$$

Cho i=n trong mệnh đề trên và bỏ đi các nhân tử bằng 1 trong tích, các số hạng bằng 0 trong tổng, ta được hệ quả sau.

8.2.5. Hệ quả: Mọi hàm Boole F bậc n đều có thể được khai triển dưới dạng:

$$F(x_1,\ldots,x_n) = \sum_{(\sigma_1,\ldots,\sigma_n)\in T_F} x_1^{\sigma_1}\ldots x_n^{\sigma_n}.$$

8.2.6. Chú ý: Từ Hệ quả 8.2.5, ta suy ra rằng mọi hàm Boole đều có thể biểu diễn dưới dạng tổng (tuyển) chuẩn tắc hoàn toàn. Như vậy mọi hàm Boole đều có thể biểu diễn bằng một biểu thức Boole chỉ chứa ba phép tích (hội), tổng (tuyển), bù (phủ định). Ta nói rằng hệ {tích, tổng, bù} là đầy đủ.

Bằng đối ngẫu, ta có thể chứng minh một kết quả tương tự bằng việc thay tích bởi tổng và ngược lại, từ đó dẫn tới việc biểu diễn F qua một tích các tổng. Biểu diễn này được gọi là dạng tích (hội) chuẩn tắc hoàn toàn của F:

$$F(x_1,\ldots,x_n) = \prod_{(\sigma_1,\ldots,\sigma_n)\in\overline{T_F}} (x_1^{\overline{\sigma_1}} + \ldots + x_n^{\overline{\sigma_n}})$$

Thí dụ 5: Dạng tổng chuẩn tắc hoàn toàn của hàm F cho trong Thí dụ 3 là:

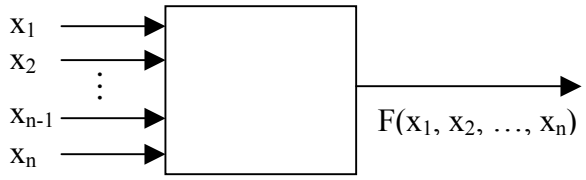
$$F(x,y,z) = \bar{x}\bar{y}\bar{z} + \bar{x}y\bar{z} + x\bar{y}\bar{z} + xy\bar{z} + xyz,$$

và dạng tích chuẩn tắc hoàn toàn của nó là:

$$F(x,y,z) = (x+y+\bar{z})(x+\bar{y}+\bar{z})(\bar{x}+y+\bar{z}).$$

8.3. MẠCH LÔGIC.

8.3.1. Cổng lôgic:



Xét một thiết bị như hình trên, có một số đường vào (dẫn tín hiệu vào) và chỉ có một đường ra (phát tín hiệu ra). Giả sử các tín hiệu vào x_1, x_2, …, x_n (ta gọi là đầu vào hay input) cũng như tín hiệu ra F (đầu ra hay output) đều chỉ có hai trạng thái khác nhau, tức là mang một bit thông tin, mà ta ký hiệu là 0 và 1.

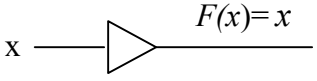
Ta gọi một thiết bị với các đầu vào và đầu ra mang giá trị 0, 1 như vậy là một mạch lôgic.

Đầu ra của một mạch lôgic là một hàm Boole F của các đầu vào x_1, x_2, …, x_n. Ta nói mạch lôgic trong hình trên thực hiện hàm F.

Các mạch lôgic được tạo thành từ một số mạch cơ sở, gọi là cổng lôgic. Các cổng lôgic sau đây thực hiện các hàm phủ định, hội và tuyển.

1. Cổng NOT: Cổng NOT thực hiện hàm phủ định. Cổng chỉ có một đầu vào. Đầu ra *F(x)* là phủ định của đầu vào *x*.

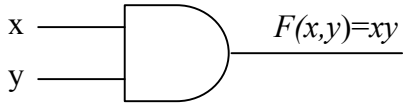
$$F(x) = \bar{x} = \begin{cases} 0 \; khi = 1, \\ 1 \; khi \; x = 0. \end{cases}$$

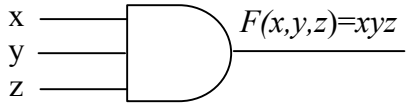


Chẳng hạn, xâu bit 100101011 qua cổng NOT cho xâu bit 011010100.

2. Cổng AND: Cổng AND thực hiện hàm hội. Đầu ra *F(x,y)* là hội (tích) của các đầu vào.

$$F(x,y) = xy = \begin{cases} 1 \; khi \; x = y = 1, \\ 0 \;\; trong \; các \; trường \; hợp \; khác. \end{cases}$$

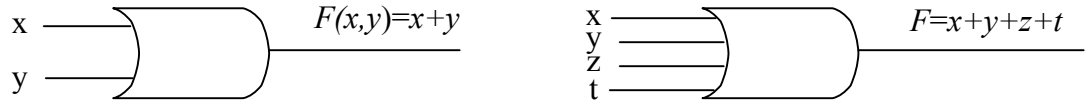




Chẳng hạn, hai xâu bit 101001101 và 111010110 qua cổng AND cho 101000100.

3. Cổng OR: Cổng OR thực hiện hàm tuyển (tổng). Đầu ra *F(x,y)* là tuyển (tổng) của các đầu vào.

$$F(x,y) = x + y = \begin{cases} 1 \; khi \; x = 1 \; hay \; y = 1, \\ 0 \; khi \; x = y = 0. \end{cases}$$



Chẳng hạn, hai xâu bit 101001101 và 111010100 qua cổng OR cho 111011101.

8.3.2. Mạch lôgic:

1. Tổ hợp các cổng: Các cổng lôgic có thể lắp ghép để được những mạch lôgic thực hiện các hàm Boole phức tạp hơn. Như ta đã biết rằng một hàm Boole bất kỳ có thể biểu diễn bằng một biểu thức chỉ chứa các phép −, **.**, +. Từ đó suy ra rằng có thể lắp ghép thích hợp các cổng NOT, AND, OR để được một mạch lôgic thực hiện một hàm Boole bất kỳ.

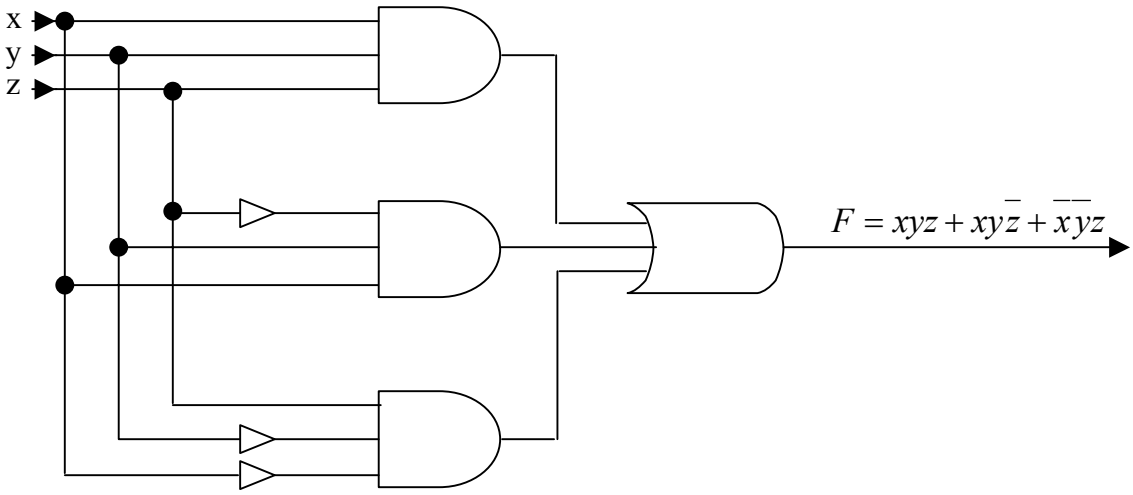
Thí dụ 6: Xây dựng một mạch lôgic thực hiện hàm Boole cho bởi bảng sau.

x	y	z	F(x,y,z)
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Theo bảng này, hàm F có dạng tổng (tuyển) chuẩn tắc hoàn toàn là:

$$F(x,y,z) = xyz + xy\bar{z} + \bar{x}\bar{y}z \; .$$

Hình dưới đây vẽ mạch lôgic thực hiện hàm F đã cho.

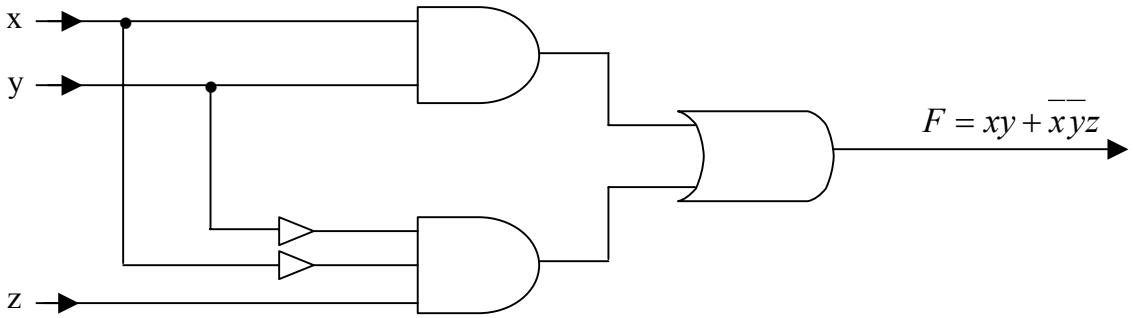


www.VIETMATHS.com

Biểu thức của *F(x, y, z)* có thể rút gọn:

$xyz + xy\bar{z} + \bar{x}\bar{\bar{y}}z = xy(z + \bar{z}) + \bar{x}\bar{\bar{y}}z = xy + \bar{x}\bar{\bar{y}}z$.

Hình dưới đây cho ta mạch lôgic thực hiện hàm $xy + \bar{x}\bar{\bar{y}}z$.



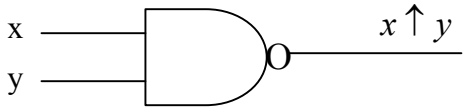
Hai mạch lôgic trong hai hình trên thực hiện cùng một hàm Boole, ta nói đó là hai mạch lôgic tương đương, nhưng mạch lôgic thứ hai đơn giản hơn.

Vấn đề tìm mạch lôgic đơn giản thực hiện một hàm Boole F cho trước gắn liền với vấn đề tìm biểu thức đơn giản nhất biểu diễn hàm ấy. Đây là vấn đề khó và lý thú, tuy ý nghĩa thực tiễn của nó không còn như mấy chục năm về trước.

Ta vừa xét việc thực hiện một hàm Boole bất kỳ bằng một mạch lôgic chỉ gồm các cổng NOT, AND, OR.

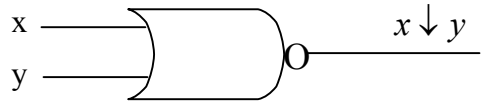
Dựa vào đẳng thức $x + y = \bar{\bar{x}}.\bar{\bar{y}}$ cũng như $xy = \overline{\overline{x} + \overline{y}}$, cho ta biết hệ {**.**, −} và hệ {+, −} cũng là các hệ đầy đủ. Do đó có thể thực hiện một hàm Boole bất kỳ bằng một mạch lôgic chỉ gồm có các cổng NOT, AND hoặc NOT, OR.

Xét hàm Sheffer $F(x,y) = x \uparrow y = \begin{cases} 0 \; khi \; x = y = 1, \\ 1 \; khi \; x = 0 \; hay \; y = 0. \end{cases}$ Mạch lôgic thực hiện hàm $\uparrow$ gọi là cổng NAND, được vẽ như hình dưới đây.



Dựa vào các đẳng thức $\bar{x} = x \uparrow x, xy = (x \uparrow y) \uparrow (x \uparrow y), x + y = (x \uparrow x) \uparrow (y \uparrow y)$, cho ta biết hệ $\{\uparrow\}$ là đầy đủ, nên bất kỳ một hàm Boole nào cũng có thể thực hiện được bằng một mạch lôgic chỉ gồm có cổng NAND.

Xét hàm Vebb $F(x,y) = x \downarrow y = \begin{cases} 0 \; khi \; x = 1 \; hay \; y = 1, \\ 1 \; khi \; x = y = 0. \end{cases}$ Mạch lôgic thực hiện hàm $\downarrow$ gọi là cổng NOR, được vẽ như hình dưới đây.



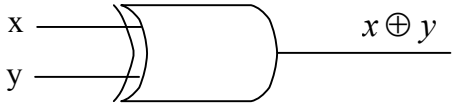
Tương tự hệ $\{\downarrow\}$ là đầy đủ nên bất kỳ hàm Boole nào cũng có thể thực hiện được bằng một mạch lôgic chỉ gồm có cổng NOR.

Một phép toán lôgic quan trọng khác là phép tuyển loại:

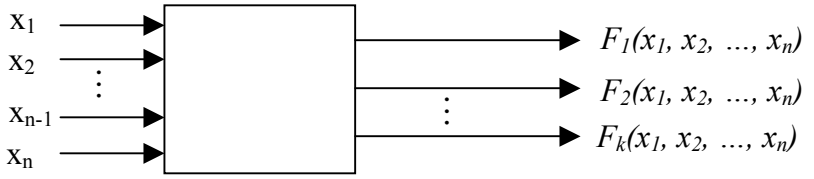
www.VIETMATHS.com

$$F(x,y) = x \oplus y = \begin{cases} 0 \; khi \; x = y, \\ 1 \; khi \; x \neq y. \end{cases}$$

Mạch lôgic này là một cổng lôgic, gọi là cổng XOR, được vẽ như hình dưới đây.



2. Mạch cộng: Nhiều bài toán đòi hỏi phải xây dựng những mạch lôgic có nhiều đường ra, cho các đầu ra $F_1, F_2, ..., F_k$ là các hàm Boole của các đầu vào $x_1, x_2, \ldots, x_n$.

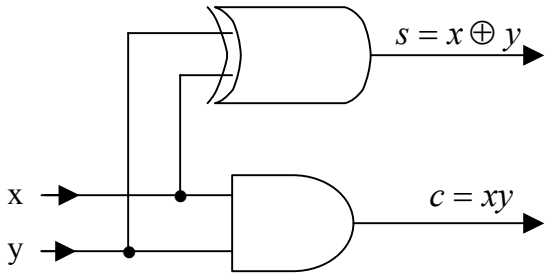


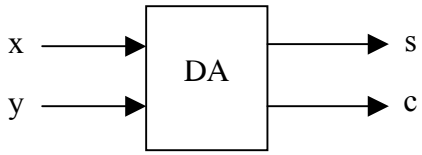
Chẳng hạn, ta xét phép cộng hai số tự nhiên từ các khai triển nhị phân của chúng. Trước hết, ta sẽ xây dựng một mạch có thể dược dùng để tìm x+y với x, y là hai số 1-bit. Đầu vào mạch này sẽ là x và y. Đầu ra sẽ là một số 2-bit $\overline{cs}$, trong đó s là bit tổng và c là bit nhớ.

0+0 = 00
0+1 = 01
1+0 = 01
1+1 = 10

x	y	c	s
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Từ bảng trên, ta thấy ngay $s = x \oplus y, c = xy$. Ta vẽ được mạch thực hiện hai hàm $s = x \oplus y$ và $c = xy$ như hình dưới đây. Mạch này gọi là mạch cộng hai số 1-bit hay mạch cộng bán phần, ký hiệu là DA.





Xét phép cộng hai số 2-bit $\overline{a_2 a_1}$ và $\overline{b_2 b_1}$,

$$\begin{array}{r} a_2 \; a_1 \\ \underline{b_2 \; b_1} \end{array}$$

Thực hiện phép cộng theo từng cột, ở cột thứ nhất (từ phải sang trái) ta tính $a_1 + b_1$ được bit tổng s_1 và bit nhớ c_1; ở cột thứ hai, ta tính $a_2 + b_2 + c_1$, tức là phải cộng ba số 1-bit.

Cho *x, y, z* là ba số 1-bit. Tổng *x+y+z* là một số 2-bit $\overline{cs}$, trong đó *s* là bit tổng của *x+y+z* và *c* là bit nhớ của *x+y+z*. Các hàm Boole *s* và *c* theo các biến *x, y, z* được xác định bằng bảng sau:

x	y	z	c	s
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Từ bảng này, dễ dàng thấy rằng:

$$s = x \oplus y \oplus z .$$

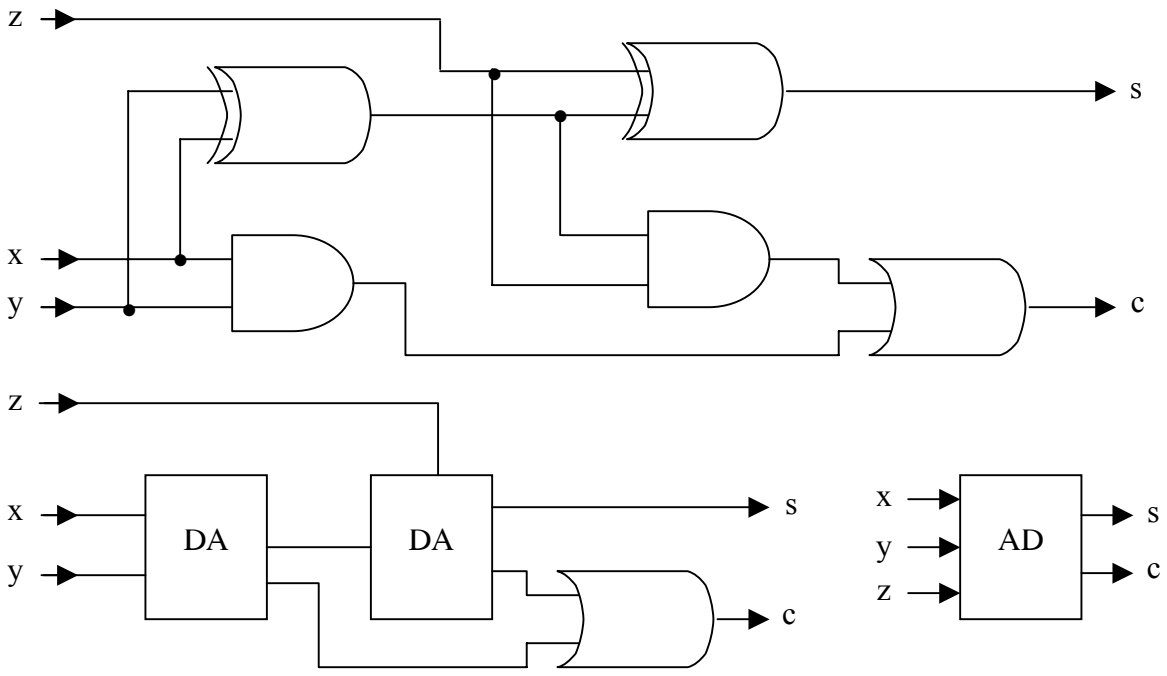
Hàm c có thể viết dưới dạng tổng chuẩn tắc hoàn toàn là:

$$c = \bar{x}yz + x\bar{y}z + xy\bar{z} + xyz .$$

Công thức của *c* có thể rút gọn:

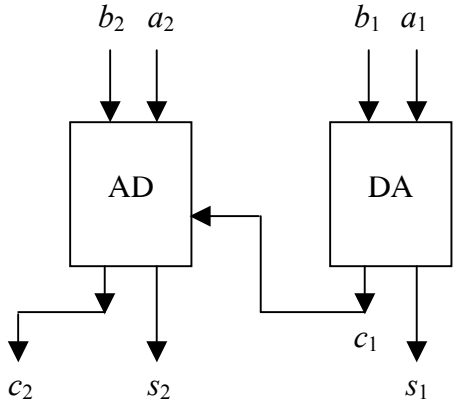
$$c = z(\bar{x}y + x\bar{y}) + xy(\bar{z} + z) = z(x \oplus y) + xy .$$

Ta vẽ được mạch thực hiện hai hàm Boole $s = x \oplus y \oplus z$ và $c = z(x \oplus y) + xy$ như hình dưới đây, mạch này là ghép nối của hai mạch cộng bán phần (DA) và một cổng OR. Đây là mạch cộng ba số 1-bit hay mạch cộng toàn phần, ký hiệu là AD.

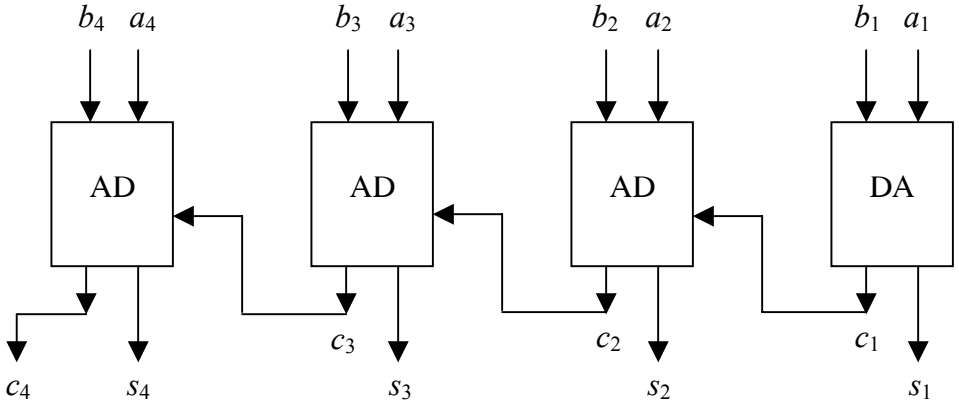


Trở lại phép cộng hai số 2-bit $\overline{a_2a_1}$ và $\overline{b_2b_1}$. Tổng $\overline{a_2a_1}+\overline{b_2b_1}$ là một số 3-bit $\overline{c_2s_2s_1}$, trong đó s_1 là bit tổng của a_1+b_1: $s_1 = a_1 \oplus b_1$, s_2 là bit tổng của $a_2+b_2+c_1$, với c_1 là bit nhớ của a_1+b_1: $s_2 = a_2 \oplus b_2 \oplus c_1$ và c_2 là bit nhớ của $a_2+b_2+c_1$.

Ta có được mạch thực hiện ba hàm Boole s_1, s_2, c_2 như hình dưới đây.



Dễ dàng suy ra mạch cộng hai số n-bit, với n là một số nguyên dương bất kỳ. Hình sau cho một mạch cộng hai số 4-bit.



8.4. CỰC TIỂU HOÁ CÁC MẠCH LÔGIC.

Hiệu quả của một mạch tổ hợp phụ thuộc vào số các cổng và sự bố trí các cổng đó. Quá trình thiết kế một mạch tổ hợp được bắt đầu bằng một bảng chỉ rõ các giá trị đầu ra đối với mỗi một tổ hợp các giá trị đầu vào. Ta luôn luôn có thể sử dụng khai triển tổng các tích của mạch để tìm tập các cổng lôgic thực hiện mạch đó. Tuy nhiên,khai triển tổng các tích có thể chứa các số hạng nhiều hơn mức cần thiết. Các số hạng trong khai triển tổng các tích chỉ khác nhau ở một biến, sao cho trong số hạng này xuất hiện biến đó và trong số hạng kia xuất hiện phần bù của nó, đều có thể được tổ hợp lại. Chẳng hạn, xét mạch có đầu ra bằng 1 khi và chỉ khi $x = y = z = 1$ hoặc $x = z = 1$ và $y = 0$. Khai triển tổng các tích của mạch này là $xyz + x\bar{y}z$. Hai tích trong khai triển này chỉ khác nhau ở một biến, đó là biến y. Ta có thể tổ hợp lại như sau:

$$xyz + x\bar{y}z = (y + \bar{y})xz = 1xz = xz .$$

Do đó *xz* là biểu thức với ít phép toán hơn biểu diễn mạch đã cho. Mạch thứ hai chỉ dùng một cổng, trong khi mạch thứ nhất phải dùng ba cổng và một bộ đảo (cổng NOT).

8.4.1. Bản đồ Karnaugh:

Để làm giảm số các số hạng trong một biểu thức Boole biểu diễn một mạch, ta cần phải tìm các số hạng để tổ hợp lại. Có một phương pháp đồ thị, gọi là bản đồ Karnaugh, được dùng để tìm các số hạng tổ hợp được đối với các hàm Boole có số biến tương đối nhỏ. Phương pháp mà ta mô tả dưới đây đã được Maurice Karnaugh đưa ra vào năm 1953. Phương pháp này dựa trên một công trình trước đó của E.W. Veitch. Các bản đồ Karnaugh cho ta một phương pháp trực quan để rút gọn các khai triển tổng các tích, nhưng chúng không thích hợp với việc cơ khí hoá quá trình này. Trước hết, ta sẽ minh hoạ cách dùng các bản đồ Karnaugh để rút gọn biểu thức của các hàm Boole hai biến.

Có bốn hội sơ cấp khác nhau trong khai triển tổng các tích của một hàm Boole có hai biến x và y. Một bản đồ Karnaugh đối với một hàm Boole hai biến này gồm bốn ô vuông, trong đó hình vuông biểu diễn hội sơ cấp có mặt trong khai triển được ghi số 1. Các hình ô được gọi là kề nhau nếu các hội sơ cấp mà chúng biểu diễn chỉ khác nhau một biến.

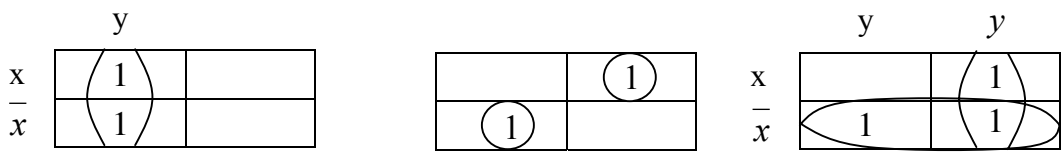
	y	$\bar{y}$
x	xy	$x\bar{y}$
$\bar{x}$	$\bar{x}y$	$\bar{x}\bar{y}$

Thí dụ 7: Tìm các bản đồ Karnaugh cho các biểu thức:

a) $xy + \bar{x}y$ b) $x\bar{y} + \bar{x}y$ c) $x\bar{y} + \bar{x}y + \bar{x}\bar{y}$

và rút gọn chúng.

Ta ghi số 1 vào ô vuông khi hội sơ cấp được biểu diễn bởi ô đó có mặt trong khai triển tổng các tích. Ba bản đồ Karnaugh được cho trên hình sau.



Việc nhóm các hội sơ cấp được chỉ ra trong hình trên bằng cách sử dụng bản đồ Karnaugh cho các khai triển đó. Khai triển cực tiểu của tổng các tích này tương ứng là:

a) y, b) $x\bar{y} + \bar{x}y$, c) $\bar{x} + \bar{y}$.

Bản đồ Karnaugh ba biến là một hình chữ nhật được chia thành tám ô. Các ô đó biểu diễn tám hội sơ cấp có được. Hai ô được gọi là kề nhau nếu các hội sơ cấp mà chúng biểu diễn chỉ khác nhau một biến. Một trong các cách để lập bản đồ Karnaugh ba biến được cho trong hình bên.

	yz	$y\bar{z}$	$\bar{y}\bar{z}$	$\bar{y}z$
x	xyz	$xy\bar{z}$	$x\bar{y}\bar{z}$	$x\bar{y}z$
$\bar{x}$	$\bar{x}yz$	$\bar{x}y\bar{z}$	$\bar{x}\bar{y}\bar{z}$	$\bar{x}\bar{y}z$

www.VIETMATHS.com

Để rút gọn khai triển tổng các tích ba biến, ta sẽ dùng bản đồ Karnaugh để nhận dạng các hội sơ cấp có thể tổ hợp lại. Các khối gồm hai ô kề nhau biểu diễn cặp các hội sơ cấp có thể được tổ hợp lại thành một tích của hai biến; các khối 2 x 2 và 4 x 1 biểu diễn các hội sơ cấp có thể tổ hợp lại thành một biến duy nhất; còn khối gồm tất cả tám ô biểu diễn một tích không có một biến nào, cụ thể đây là biểu thức 1.

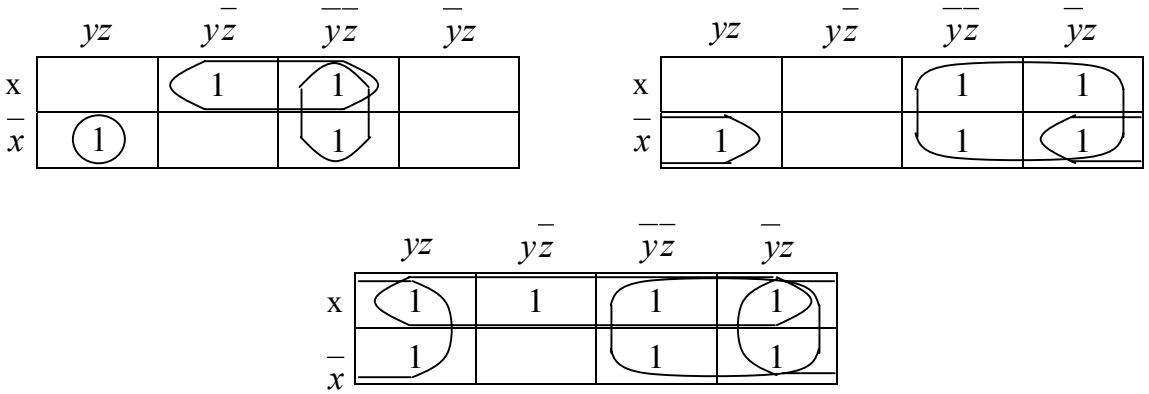
Thí dụ 8: Dùng các bản đồ Karnaugh ba biến để rút gọn các khai triển tổng các tích sau:

a) $x\bar{y}\bar{z} + x\bar{\bar{y}}\bar{z} + \bar{x}yz + \bar{x}\bar{y}\bar{z}$,

b) $x\bar{y}z + x\bar{\bar{y}}\bar{z} + \bar{x}yz + \bar{x}\bar{\bar{y}}z + \bar{x}\bar{y}\bar{z}$,

c) $xyz + xy\bar{z} + x\bar{y}z + x\bar{\bar{y}}\bar{z} + \bar{x}yz + \bar{x}\bar{y}z + \bar{x}\bar{y}\bar{z}$.

Bản đồ Karnaugh cho những khai triển tổng các tích này được cho trong hình sau:



Việc nhóm thành các khối cho thấy rằng các khai triển cực tiểu thành các tổng Boole của các tích Boole là:

a) $x\bar{z} + \bar{\bar{y}}z + \bar{x}yz$, b) $\bar{y} + \bar{x}z$, c) $x + \bar{y} + z$.

Bản đồ Karnaugh bốn biến là một hình vuông được chia làm 16 ô. Các ô này biểu diễn 16 hội sơ cấp có được. Một trong những cách lập bản đồ Karnaugh bốn biến được cho trong hình dưới đây.

	yz	$y\bar{z}$	$\bar{\bar{y}}z$	$\bar{y}z$
wx	$wxyz$	$wxy\bar{z}$	$wx\bar{\bar{y}}z$	$wx\bar{y}z$
$w\bar{x}$	$w\bar{x}yz$	$w\bar{x}y\bar{z}$	$w\bar{x}\bar{\bar{y}}z$	$w\bar{x}\bar{y}z$
$\bar{\bar{w}}x$	$\bar{\bar{w}}xyz$	$\bar{\bar{w}}xy\bar{z}$	$\bar{\bar{w}}x\bar{\bar{y}}z$	$\bar{\bar{w}}x\bar{y}z$
$\bar{w}x$	$\bar{w}xyz$	$\bar{w}xy\bar{z}$	$\bar{w}x\bar{\bar{y}}z$	$\bar{w}x\bar{y}z$

Hai ô được gọi là kề nhau nếu các hội sơ cấp mà chúng biểu diễn chỉ khác nhau một biến. Do đó, mỗi một ô kề với bốn ô khác. Sự rút gọn một khai triển tổng các tích bốn biến được thực hiện bằng cách nhận dạng các khối gồm 2, 4, 8 hoặc 16 ô biểu diễn các hội sơ cấp có thể tổ hợp lại được. Mỗi ô biểu diễn một hội sơ cấp hoặc được dùng để lập một tích có ít biến hơn hoặc được đưa vào trong khai triển. Cũng như trong trường

www.VIETMATHS.com

hợp bản đồ Karnaugh hai và ba biến, mục tiêu là cần phải nhận dạng các khối lớn nhất có chứa các số 1 bằng cách dùng một số ít nhất các khối, mà trước hết là các khối lớn nhất.

8.4.2. Phương pháp Quine-McCluskey:

8.4.2.1. Mở đầu: Ta đã thấy rằng các bản đồ Karnaugh có thể được dùng để tạo biểu thức cực tiểu của các hàm Boole như tổng của các tích Boole. Tuy nhiên, các bản đồ Karnaugh sẽ rất khó dùng khi số biến lớn hơn bốn. Hơn nữa, việc dùng các bản đồ Karnaugh lại dựa trên việc rà soát trực quan để nhận dạng các số hạng cần được nhóm lại. Vì những nguyên nhân đó, cần phải có một thủ tục rút gọn những khai triển tổng các tích có thể cơ khí hoá được. Phương pháp Quine-McCluskey là một thủ tục như vậy. Nó có thể được dùng cho các hàm Boole có số biến bất kỳ. Phương pháp này được W.V. Quine và E.J. McCluskey phát triển vào những năm 1950. Về cơ bản, phương pháp Quine-McCluskey có hai phần. Phần đầu là tìm các số hạng là ứng viên để đưa vào khai triển cực tiểu như một tổng các tích Boole mà ta gọi là các nguyên nhân nguyên tố. Phần thứ hai là xác định xem trong số các ứng viên đó, các số hạng nào là thực sự dùng được.

8.4.2.2. Định nghĩa: Cho hai hàm Boole F và G bậc n. Ta nói G là một nguyên nhân của F nếu $T_G \subset T_F$, nghĩa là $G \Rightarrow F$ là một hằng đúng.

Dễ thấy rằng mỗi hội sơ cấp trong một dạng tổng chuẩn tắc của F là một nguyên nhân của F. Hội sơ cấp A của F được gọi là một nguyên nhân nguyên tố của F nếu trong A xoá đi một biến thì hội nhận được không còn là nguyên nhân của F.

Nếu F_1, ..., F_k là các nguyên nhân của F thì $T_{F_i} \subset T_F$, $1 \le i \le k$. Khi đó $T_{\sum_{i=1}^{k} F_i} = \bigcup_{i=1}^{k} T_{F_i} \subset T_F$. Do đó $\sum_{i=1}^{k} F_i$ là một nguyên nhân của F.

Cho S là một hệ các nguyên nhân của F. Ta nói rằng hệ S là đầy đủ đối với F nếu $F = \sum_{G \in S} G$, nghĩa là $T_F = \bigcup_{G \in S} T_G$.

8.4.2.3. Mệnh đề: Hệ các nguyên nhân nguyên tố của hàm F là một hệ đầy đủ.

Chứng minh: Gọi S là hệ các nguyên nhân nguyên tố của F. Ta có $T_G \subset T_F$, $\forall g \in S$, Nên $T_{\sum_{G \in S} G} = \bigcup_{G \in S} T_G \subset T_F$. Giả sử dạng tổng chuẩn tắc hoàn toàn của F là $F = \sum_{G' \in S'} G'$ nên $T_F = \bigcup_{G' \in S'} T_{G'}$.

Xét $G' \in S'$, nếu G' không phải là nguyên nhân nguyên tố của F thì bằng cách xoá bớt một số biến trong G' ta thu được nguyên nhân nguyên tố G của F. Khi đó $T_{G'} \subset T_G$ và $\bigcup_{G' \in S'} T_{G'} \subset \bigcup_{G \in S} T_G$ hay $T_F \subset \bigcup_{G \in S} T_G$. Vì vậy $T_F = \bigcup_{G \in S} T_G$ hay $F = \sum_{G \in S} G$.

Dạng tổng chuẩn tắc $F = \sum_{G \in S} G$ được gọi là dạng tổng chuẩn tắc thu gọn của F.

8.4.2.4. Phương pháp Quine-McCluskey tìm dạng tổng chuẩn tắc thu gọn:

Giả sử F là một hàm Boole n biến $x_1, x_2, \ldots, x_n$. Mỗi hội sơ cấp của n biến đó được biểu diễn bằng một dãy n ký hiệu trong bảng $\{0, 1, -\}$ theo quy ước: ký tự thứ i là 1 hay 0 nếu x_i có mặt trong hội sơ cấp là bình thường hay với dấu phủ định, còn nếu x_i không có mặt thì ký tự này là −. Chẳng hạn, hội sơ cấp của 6 biến $x_1, \ldots, x_6$ là $\overline{x_1} x_3 x_4 \overline{x_6}$ được biểu diễn bởi 0−11−0. Hai hội sơ cấp được gọi là kề nhau nếu các biểu diễn nói trên của chúng chỉ khác nhau ở một vị trí 0, 1. Rõ ràng các hội sơ cấp chỉ có thể dán được với nhau bằng phép dán $Ax + A\overline{x} = A$ nếu chúng là kề nhau.

Thuật toán được tiến hành như sau: Lập một bảng gồm nhiều cột để ghi các kết quả dán. Sau đó lần lượt thực hiện các bước sau:

Bước 1: Viết vào cột thứ nhất các biểu diễn của các nguyên nhân hạng n của hàm Boole F. Các biểu diễn được chia thành từng nhóm, các biểu diễn trong mỗi nhóm có số các ký hiệu 1 bằng nhau và các nhóm xếp theo thứ tự số các ký hiệu 1 tăng dần.

Bước 2: Lần lượt thực hiện tất cả các phép dán các biểu diễn trong nhóm i với các biểu diễn trong nhóm i+1 (i=1, 2, …). Biểu diễn nào tham gia ít nhất một phép dán sẽ được ghi nhận một dấu * bên cạnh. Kết quả dán được ghi vào cột tiếp theo.

Bước 3: Lặp lại Bước 2 cho cột kế tiếp cho đến khi không thu thêm được cột nào mới. Khi đó tất cả các biểu diễn không có dấu * sẽ cho ta tất cả các nguyên nhân nguyên tố của F.

Thí dụ 9: Tìm dạng tổng chuẩn tắc thu gọn của các hàm Boole:

$$F_1 = \overline{w}\overline{x}\overline{y}z + \overline{w}\overline{x}y\overline{z} + \overline{w}x\overline{y}z + w\overline{x}\overline{y}z + w\overline{x}yz + w\overline{x}y\overline{z} + wxyz,$$
$$F_2 = \overline{w}\overline{x}yz + \overline{w}x\overline{y}z + \overline{w}xy\overline{z} + w\overline{x}\overline{y}z + w\overline{x}y\overline{z} + wx\overline{y}\overline{z} + wxyz.$$

0 0 0 1 *	0 − 0 1 *	0 − − 1
0 1 0 1 *	0 0 − 1 *	− 0 − 1
0 0 1 1 *	− 0 0 1 *	− − 1 1
1 0 0 1 *	− 0 1 1 *	
1 0 1 1 *	1 0 − 1 *	
0 1 1 1 *	0 1 − 1 *	
1 1 1 1 *	0 − 1 1 *	
	1 − 1 1 *	
	− 1 1 1 *	

0 0 1 0 *	0 0 1 −	1 1 − −
0 0 1 1 *	− 0 1 1	
1 1 0 0 *	1 1 0 − *	
1 0 1 1 *	1 1 − 0 *	
1 1 0 1 *	1 − 1 1	
1 1 1 0 *	1 1 − 1 *	
1 1 1 1 *	1 1 1 − *	

Từ các bảng trên ta có dạng tổng chuẩn tắc thu gọn của F_1 và F_2 là:

$$F_1 = \overline{w}z + \overline{x}z + yz,$$
$$F_2 = \overline{w}\overline{x}y + \overline{x}yz + wyz + wx.$$

8.4.2.5. Phương pháp Quine-McCluskey tìm dạng tổng chuẩn tắc tối thiểu:

Sau khi tìm được dạng tổng chuẩn tắc thu gọn của hàm Boole F, nghĩa là tìm được tất cả các nguyên nhân nguyên tố của nó, ta tiếp tục phương pháp Quine-McCluskey tìm dạng tổng chuẩn tắc tối thiểu (cực tiểu) của F như sau.

Lập một bảng chữ nhật, mỗi cột ứng với một cấu tạo đơn vị của F (mỗi cấu tạo đơn vị là một hội sơ cấp hạng n trong dạng tổng chuẩn tắc hoàn toàn của F) và mỗi dòng ứng với một nguyên nhân nguyên tố của F. Tại ô (i, j), ta đánh dấu cộng (+) nếu nguyên nhân nguyên tố ở dòng i là một phần con của cấu tạo đơn vị ở cột j. Ta cũng nói rằng khi đó nguyên nhân nguyên tố i là phủ cấu tạo đơn vị j. Một hệ S các nguyên nhân nguyên tố của F được gọi là phủ hàm F nếu mọi cấu tạo đơn vị của F đều được phủ ít nhất bởi một thành viên của hệ. Dễ thấy rằng nếu hệ S là phủ hàm F thì nó là đầy đủ, nghĩa là tổng của các thành viên trong S là bằng F.

Một nguyên nhân nguyên tố được gọi là cốt yếu nếu thiếu nó thì một hệ các nguyên nhân nguyên tố không thể phủ hàm F. Các nguyên nhân nguyên tố cốt yếu được tìm như sau: tại những cột chỉ có duy nhất một dấu +, xem dấu + đó thuộc dòng nào thì dòng đó ứng với một nguyên nhân nguyên tố cốt yếu.

Việc lựa chọn các nguyên nhân nguyên tố trên bảng đã đánh dấu, để được một dạng tổng chuẩn tắc tối thiểu, có thể tiến hành theo các bước sau.

Bước 1: Phát hiện tất cả các nguyên nhân nguyên tố cốt yếu.

Bước 2: Xoá tất cả các cột được phủ bởi các nguyên nhân nguyên tố cốt yếu.

Bước 3: Trong bảng còn lại, xoá nốt những dòng không còn dấu + và sau đó nếu có hai cột giống nhau thì xoá bớt một cột.

Bước 4: Sau các bước trên, tìm một hệ S các nguyên nhân nguyên tố với số biến ít nhất phủ các cột còn lại.

Tổng của các nguyên nhân nguyên tố cốt yếu và các nguyên nhân nguyên tố trong hệ S sẽ là dạng tổng chuẩn tắc tối thiểu của hàm F.

Các bước 1, 2, 3 có tác dụng rút gọn bảng trước khi lựa chọn. Độ phức tạp chủ yếu nằm ở Bước 4. Tình huống tốt nhất là mọi nguyên nhân nguyên tố đều là cốt yếu. Trường hợp này không phải lựa chọn gì và hàm F có duy nhất một dạng tổng chuẩn tắc tối thiểu cũng chính là dạng tổng chuẩn tắc thu gọn. Tình huống xấu nhất là không có nguyên nhân nguyên tố nào là cốt yếu. Trường hợp này ta phải lựa chọn toàn bộ bảng.

Thí dụ 10: Tìm dạng tổng chuẩn tắc tối thiểu của các hàm Boole cho trong Thí dụ 9.

	$\bar{w}\bar{x}\bar{y}z$	$\bar{w}\bar{x}y\bar{z}$	$\bar{w}\bar{x}yz$	$\bar{w}x\bar{y}\bar{z}$	$\bar{w}x\bar{y}z$	$\bar{w}xyz$	$wxyz$
$\bar{w}z$	+	+	+				
$\bar{x}z$	+		+	+	+		
yz			+		+	+	+

www.VIETMATHS.com

Các nguyên nhân nguyên tố đều là cốt yếu nên dạng tổng chuẩn tắc tối thiểu của F_1 là:

$$F_1 = \overline{w}z + \overline{x}z + yz$$

	$\overline{w}\overline{x}y\overline{z}$	$\overline{w}\overline{x}yz$	$\overline{w}xyz$	$wx\overline{y}\overline{z}$	$wx\overline{y}z$	$wxy\overline{z}$	$wxyz$
wx				+	+	+	+
$\overline{w}\overline{x}y$	+	+					
$\overline{x}yz$		+	+				
wyz			+				+

Các nguyên nhân nguyên tố cốt yếu nằm ở dòng 1 và 2. Sau khi rút gọn, bảng còn dòng 3, 4 và một cột 3. Việc chọn S khá đơn giản: có thể chọn một trong hai nguyên nhân nguyên tố còn lại. Vì vậy ta được hai dạng tổng chuẩn tắc tối thiểu là:

$$F_2 = wx + \overline{w}\overline{x}y + \overline{x}yz,$$

$$F_2 = wx + \overline{w}\overline{x}y + wyz.$$

www.VIETMATHS.com

BÀI TẬP CHƯƠNG VIII:

1. Cho S là tập hợp các ước nguyên dương của 70, với các phép toán •, + và ’ được định nghĩa trên S như sau:

$$a \bullet b = UCLN(a, b),\ a + b = BCNN(a, b),\ a' = 70/a.$$

Chứng tỏ rằng S cùng với các phép toán •, + và ’ lập thành một đại số Boole.

2. Chứng minh trực tiếp các định lý 6b, 7b, 8b (không dùng đối ngẫu để suy ra từ 6a, 7a, 8a).

3. Chứng minh rằng:

a) (a+b)**.**(a+b’) = a;

b) (a**.**b)+(a’**.**c) = (a+c)**.**(a’+b).

4. Cho các hàm Boole F_1, F_2, F_3 xác định bởi bảng sau:

x	y	z	F_1	F_2	F_3
0	0	0	1	1	0
0	0	1	1	0	1
0	1	0	0	1	1
0	1	1	1	1	0
1	0	0	1	0	1
1	0	1	0	0	1
1	1	0	0	1	1
1	1	1	1	1	1

Vẽ mạch thực hiện các hàm Boole này.

5. Hãy dùng các cổng NAND để xây dựng các mạch với các đầu ra như sau:

a) $\bar{x}$ **b)** xy **c)** $x+y$ **d)** $x \oplus y$.

6. Hãy dùng các cổng NOR để xây dựng các mạch với các đầu ra được cho trong Bài tập **5**.

7. Hãy dùng các cổng NAND để dựng mạch cộng bán phần.

8. Hãy dùng các cổng NOR để dựng mạch cộng bán phần.

9. Dùng các bản đồ Karnaugh, tìm dạng tổng chuẩn tắc tối thiểu (khai triển cực tiểu) của các hàm Boole ba biến sau:

a) $F = \bar{x}yz + \bar{x}\bar{y}z$.

b) $F = xyz + xy\bar{z} + +\bar{x}yz + \bar{x}y\bar{z}$.

c) $F = xy\bar{z} + +x\bar{y}z + x\bar{y}\bar{z} + \bar{x}yz + \bar{x}\bar{y}z$.

d) $F = xyz + x\bar{y}z + x\bar{y}\bar{z} + \bar{x}yz + \bar{x}y\bar{z} + \bar{x}\bar{y}\bar{z}$.

10. Dùng các bản đồ Karnaugh, tìm dạng tổng chuẩn tắc tối thiểu của các hàm Boole bốn biến sau:

a) $F = wxyz + wx\bar{y}z + wx\bar{y}\bar{z} + w\bar{x}\bar{y}z + w\bar{x}\bar{y}\bar{z}$.

b) $F = wxy\bar{z} + wx\bar{y}z + w\bar{x}yz + w\bar{x}y\bar{z} + w\bar{x}\bar{y}z + \bar{w}x\bar{y}z$.

c) $F = wxyz + wxy\bar{z} + wx\bar{y}z + w\bar{x}\bar{y}z + w\bar{x}y\bar{z} + \bar{w}x\bar{y}z + \bar{w}\bar{x}yz + \bar{w}x\bar{y}\bar{z}$.

d) $F = wxyz + wxy\bar{z} + wx\bar{y}z + w\bar{x}yz + w\bar{x}y\bar{z} + \bar{w}xyz + \bar{w}\bar{x}yz + \bar{w}x\bar{y}\bar{z} + \bar{w}xy\bar{z}$.

11. Dùng phương pháp Quine-McCluskey, tìm dạng tổng chuẩn tắc tối thiểu của các hàm Boole ba biến cho trong Bài tập 9 và hãy vẽ mạch thực hiện các dạng tối thiểu tìm được.

12. Dùng phương pháp Quine-McCluskey, tìm dạng tổng chuẩn tắc tối thiểu của các hàm Boole bốn biến cho trong Bài tập 9 và hãy vẽ mạch thực hiện các dạng tối thiểu tìm được.

13. Hãy giải thích làm thế nào có thể dùng các bản đồ Karnaugh để rút gọn dạng tích chuẩn tắc (tích các tổng) hoàn toàn của một hàm Boole ba biến. (Gợi ý: Đánh dấu bằng số 0 tất cả các tuyển sơ cấp trong biểu diễn và tổ hợp các khối của các tuyển sơ cấp.)

14. Dùng phương pháp ở Bài tập 13, hãy rút gọn dạng tích chuẩn tắc hoàn toàn:

$$F = (x + y + z)(x + y + \bar{z})(x + \bar{y} + z)(\bar{x} + y + z).$$

 www.VIETMATHS.com

TÀI LIỆU THAM KHẢO

[1] *Nguyễn Cam-Chu Đức Khánh*, Lý thuyết đồ thị, NXB Thành phố Hồ Chí Minh, 1999.
[2] *Hoàng Chúng*, Đại cương về toán học hữu hạn, NXB Giáo dục, 1997.
[3] *Phan Đình Diệu*, Lý thuyết Ô-tô-mat và thuật toán, NXB Đại học và THCN, 1977.
[4] *Đỗ Đức Giáo*, Toán rời rạc, NXB Đại học Quốc Gia Hà Nội, 2000.
[5] *Nguyễn Xuân Quỳnh*, Cơ sở toán rời rạc và ứng dụng, NXB Giáo dục, 1995.
[6] *Đặng Huy Ruận*, Lý thuyết đồ thị và ứng dụng, NXB Khoa học và Kỹ thuật, 2000.
[7] *Nguyễn Tô Thành-Nguyễn Đức Nghĩa*, Toán rời rạc, NXB Giáo dục, 1997.
[8] *Claude Berge*, Théorie des graphes et ses applications, Dunod, Paris 1963.
[9] *Richard Johnsonbaugh*, Discrete Mathematics, Macmillan Publishing Company, New york 1992.

PHẦN PHỤ LỤC

Phụ lục 1

Unit chứa khai báo các cấu trúc dữ liệu cho đồ thị và cài đặt thủ tục tìm đường đi ngắn nhất theo thuật toán

unit Func_DoThi;

interface

```
type
    TypeToaDo=record
        x,y:integer;
        end;
    TypeChiPhi=record
        VoCung:boolean;//Neu VoCung=True thi co nghia la chi phi bang Vo Cung,
nguoc lai thi chi phi bang Gia
        Gia:real;
        end;
    TypeDinh=record
        Ten:String;
        ToaDo:TypeToaDo;
        MucKichHoat:Byte;
        end;
    TypeDanhSachDinh=array of TypeDinh;
    TypeCanh=record
        DinhDau,DinhCuoi:Integer;//Tham chieu trong danh sach Dinh
        TrongSo:TypeChiphi;
        end;
    TypeDanhSachCanh=Array of TypeCanh;
    TypeDoThi=Record
        SoDinh:Integer;
        DSDinh:TypeDanhSachDinh;
        SoCanh:Integer;
        DSCanh:TypeDanhSachCanh;
        end;

    TypeCost=Array of Array of TypeChiPhi;
    TypeDist=Array of TypeChiPhi;
    TypeDuongDi=Array of Integer;

Function DuongDiNganNhat(G:TypeDoThi;X,Y:Integer;Var
DuongDiTuXdenY:TypeDuongDi;Var ChiPhi:real):Boolean;
Procedure DeleteGraph(VAR G:TypeDoThi);
var G:TypeDoThi;
```

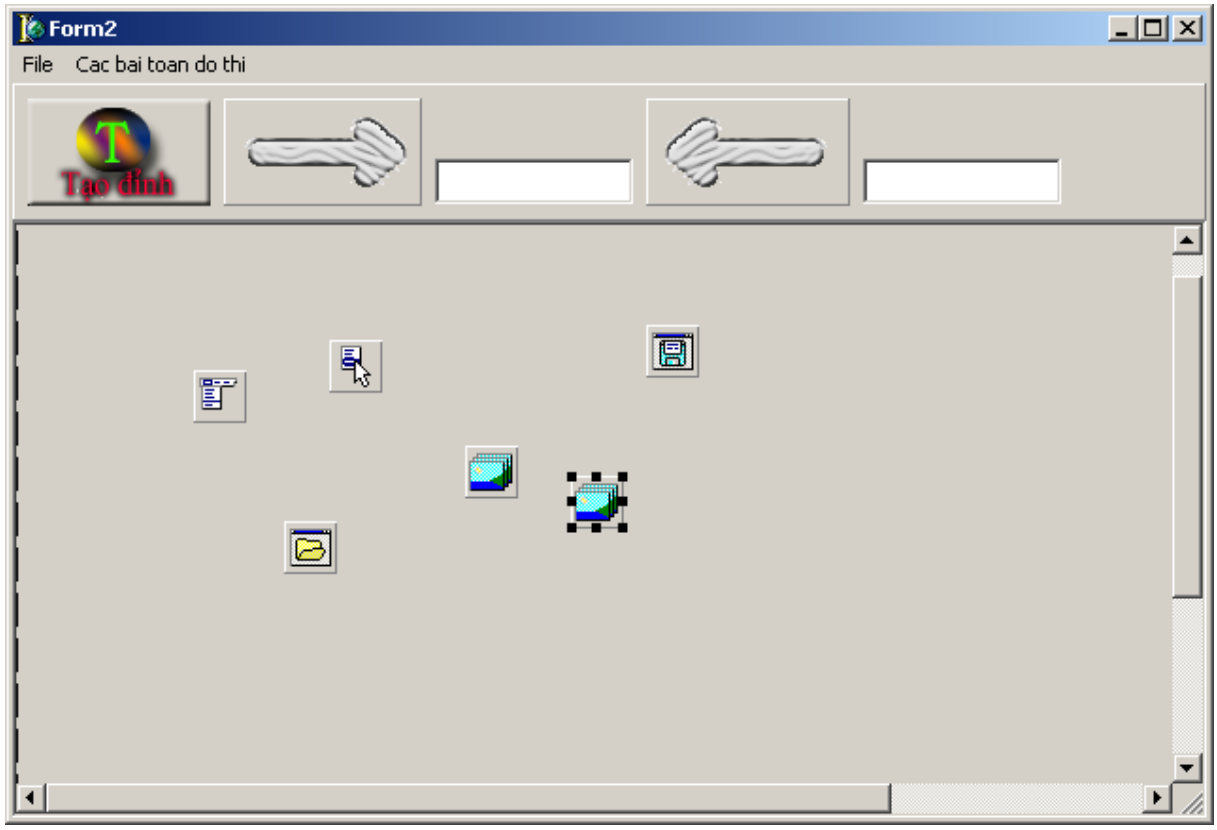
implementation

```
Function DuongDiNganNhat(G:TypeDoThi;X,Y:Integer;Var
DuongDiTuXdenY:TypeDuongDi;var ChiPhi:real):Boolean;
Var s:Array of byte;{S[i]=0 hoac S[i]=1}
Cost:TypeCost;Dist:TypeDist;MocXich:Array of Integer;
M,i,j,K,u,w:Integer;
Min:TypeChiPhi;
begin
M:=G.SoDinh; {Thuc ra M=N, ma tran vuong kich thuoc MxM}

Setlength(Cost,M,M);
Setlength(Dist,M);
Setlength(MocXich,M);
Setlength(S,M);
for i:=0 to M-1 do
   for j:=0 to M-1 do
      Cost[i,j].VoCung:=True;

for k:=0 to G.SoCanh-1 do
   begin
      i:=G.DSCanh[K].DinhDau;j:=G.DSCanh[K].DinhCuoi;
      Cost[i,j]:=G.DSCanh[K].TrongSo;
   end;

for i:=0 to M-1 do
   begin S[i]:=0;Dist[i]:=Cost[X,i];MocXich[i]:=X;end;
S[X]:=1;Dist[X].VoCung:=False;Dist[X].Gia:=0;K:=2; {Dua X vao S}
while k<M do {Xac dinh M-1 duong di}
   begin
      u:=0;
      While S[u]<>0 do u:=u+1;
      Min:=Dist[u];i:=u+1;
      While i<M do
         begin
         If S[i]=0 then
           If ((Min.VoCung)and(not Dist[i].VoCung))or
             ((Not min.VoCung)and((not Dist[i].VoCung)and(min.Gia>Dist[i].Gia)))
then
             begin Min:=Dist[i];u:=i;end;
           i:=i+1;
         end;
      S[u]:=1;k:=k+1;{Dua u vao tap S}
      For w:=0 to M-1 do
         if S[w]=0 then
            begin
              If (not Dist[u].VoCung)and(not Cost[u,w].VoCung)and
                ((Dist[w].VoCung)or(Dist[w].Gia>(Dist[u].Gia+Cost[u,w].Gia)))
```

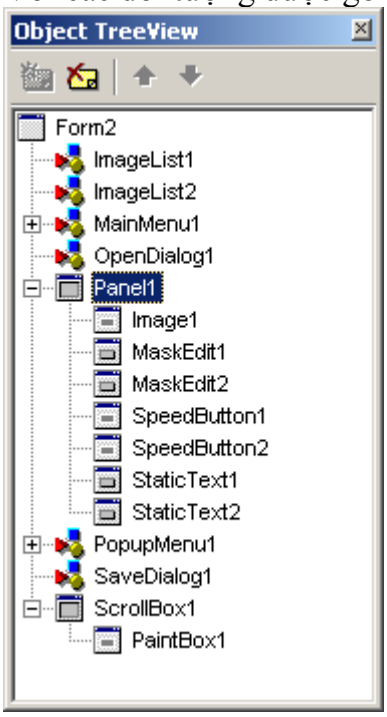
```
                then
                  begin
                    Dist[w].VoCung:=false;
                    Dist[w].Gia:=Dist[u].Gia+Cost[u,w].Gia;
                    MocXich[w]:=u;{Duong di ngan nhat den W thi phai di qua U}
                  end;
              end;
    end;
{Tim duong di tu X den Y}
Setlength(DuongDiTuXdenY,M);
If not Dist[Y].VoCung then
    begin
      DuongDiNganNhat:=true;
      ChiPhi:=Dist[Y].gia;
      {Xac dinh cac dinh phai di qua (theo day chuyen nguoc)}
      {k:=0;DuongDiTuXdenY[k]:=Y;k:=k+1;
      i:=MocXich[Y];DuongDiTuXdenY[k]:=i;}
      K:=0;i:=Y;DuongDiTuXdenY[k]:=i;
      while i<>X do
        begin
          i:=MocXich[i];k:=k+1;DuongDiTuXdenY[k]:=i;
        end;
      {Vi chuoi chua trong DuongDiTuXdenY la mot chuoi nguoc nen ta se dao lai}
      for i:=0 to (k div 2) do
        begin
          j:=DuongDiTuXdenY[i];
          DuongDiTuXdenY[i]:=DuongDiTuXdenY[K-i];
          DuongDiTuXdenY[K-i]:=j;
        end;
      {Dat lai kich thuoc cua mang DuongDiTuXdenY bang so dinh phai di qua}
      Setlength(DuongDiTuXdenY,K+1);
    end
else   DuongDiNganNhat:=false;

Setlength(Cost,0,0);
Setlength(Dist,0);
Setlength(MocXich,0);
Setlength(S,0);
end;
Procedure DeleteGraph(VAR G:TypeDoThi);
begin
G.SoDinh:=0;
G.SoCanh:=0;
Setlength(G.DSDinh,0);
Setlength(G.DSCanh,0);
end;
BEGIN
G.SoDinh :=0;G.SoCanh:=0;
```

www.VIETMATHS.com
END.

Thiết kế giao diện cho chương trình (Form 2)



Với các đối tượng được gồm:



Các khai báo và cài đặt cho chương form2:

unit Unit2;

interface

```
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, Mask, Buttons, ExtCtrls,Func_Dothi,Func_Graph,
Menus,IdGlobal,  ImgList,Jpeg;
const BanKinh=20;
RMuiTen=10;

type
  TForm2 = class(TForm)
    Panel1: TPanel;
    MaskEdit1: TMaskEdit;
    MaskEdit2: TMaskEdit;
    StaticText1: TStaticText;
    StaticText2: TStaticText;
    MainMenu1: TMainMenu;
    imduongdingannhat1: TMenuItem;
    imduongdingannhat2: TMenuItem;
    Caykhungbenhat1: TMenuItem;
    Image1: TImage;
    PopupMenu1: TPopupMenu;
    Rename1: TMenuItem;
    Delete1: TMenuItem;
    N1: TMenuItem;
    N2: TMenuItem;
    ImageList1: TImageList;
    File1: TMenuItem;
    New1: TMenuItem;
    Open1: TMenuItem;
    Save1: TMenuItem;
    N3: TMenuItem;
    Exit1: TMenuItem;
    ScrollBox1: TScrollBox;
    PaintBox1: TPaintBox;
    Save2: TMenuItem;
    N6: TMenuItem;
    ExportPicturefile1: TMenuItem;
    DeleteAll1: TMenuItem;
    SaveDialog1: TSaveDialog;
    OpenDialog1: TOpenDialog;
    ImageList2: TImageList;
    SpeedButton1: TSpeedButton;
    SpeedButton2: TSpeedButton;
    ExportPicturefile2: TMenuItem;
```

```
    N4: TMenuItem;
    procedure PaintBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
    procedure PaintBox1DragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    Procedure DrawPaint(PaintBox:TPaintBox;Bitmap:TBitmap);
    procedure FormResize(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    function DownDinh(x,y:integer;G:TypeDothi):integer;
    procedure PaintBox1MouseDown(Sender: TObject; Button: TMouseButton;
      Shift: TShiftState; X, Y: Integer);
    procedure PaintBox1MouseMove(Sender: TObject; Shift: TShiftState; X,
      Y: Integer);
    procedure PaintBox1MouseUp(Sender: TObject; Button: TMouseButton;
      Shift: TShiftState; X, Y: Integer);
    procedure HienThamSoCung(G:TypeDoThi);
    procedure MaskEdit1Change(Sender: TObject);
    procedure MaskEdit2Change(Sender: TObject);
    procedure PaintBox1Paint(Sender: TObject);
    procedure imduongdingannhat2Click(Sender: TObject);
    procedure FormCloseQuery(Sender: TObject; var CanClose: Boolean);
    procedure FormDestroy(Sender: TObject);
    procedure Rename1Click(Sender: TObject);
    procedure Exit1Click(Sender: TObject);
    procedure Delete1Click(Sender: TObject);
    procedure DeleteAll1Click(Sender: TObject);
    procedure Save1Click(Sender: TObject);
    procedure Open1Click(Sender: TObject);
    procedure SpeedButton1Click(Sender: TObject);
    procedure SpeedButton2Click(Sender: TObject);
    procedure New1Click(Sender: TObject);
    procedure ExportPicturefile2Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form2: TForm2;
  Pic:Tbitmap;
  Mouse_Down:Boolean;
  Dx,Dy,DinhDown:Integer;
  TextSizeTrongSo:Integer=10;
  Filename:String; FileChanged:Boolean;

procedure
Vecung(Pic:Tbitmap;T1,T2:TypeToaDo;Gia:Real;Line:Boolean;LineColor,TextColor:
Tcolor);
```

```
Procedure VeDoThi(G:TypeDothi;Pic:Tbitmap;Imagelist:Timagelist);
Function Delen(x,y,Width,Height:integer;DinhDown:integer):boolean;
Procedure
Veline(T1,T2:TypeToaDo;Gia:real;Pic:Tbitmap;LineColor:Tcolor;TimeDelay:TdateTi
me);
```

implementation

```
{$R *.dfm}

Function MidPoint(T1,T2:TypeToaDo;PhanTram:Integer):TypeToaDo;
Var Dx,Dy:integer;
begin
Dx:=T2.x -T1.x ;Dy:=T2.y -T1.y ;
MidPoint.x:=T1.x +Round(Dx*PhanTram/100);
MidPoint.y:=T1.y +Round(Dy*PhanTram/100);
end;
Procedure
Veline(T1,T2:TypeToaDo;Gia:real;Pic:Tbitmap;LineColor:Tcolor;TimeDelay:TdateTi
me);
var i:integer;T3:TypeToaDo;TimeNow:TDateTime;
TempPic:Tbitmap;
begin
      TempPic:=Tbitmap.Create;

      For i:=1 to 100 do
         begin
            TempPic.Assign(Pic);
            TimeNow:=Time;
            T3:=MidPoint(T1,T2,i);
            Vecung(TempPic,T1,T3,Gia,True,RGB(255,0,0),RGB(0,0,255));
            Form2.DrawPaint(Form2.PaintBox1,TempPic);
            repeat
               Application.ProcessMessages;
            until (TimeNow+TimeDelay)>Time;
         end;
      TempPic.Free;
end;

Procedure TForm2.DrawPaint(PaintBox:TPaintBox;Bitmap:TBitmap);
begin
      Paintbox.Canvas.Draw(0,0,Bitmap);
end;
procedure CatZeroThua(var St:string);
var i,P,L:integer;
begin
L:=length(st);
If St[L]=' ' then begin delete(st,1,L);L:=length(st);end;
P:=pos('.',st);i:=L;
```

```
If P=0 then exit;
while (i>P)and(st[i]='0') do i:=i-1;
If st[i]='.' then i:=i-1;
delete(St,i+1,L-i);
end;
Function Quay(P,Tam:TypeToaDo;Goc:Real):TypeToaDo;
Var Q:TypeToaDo;
begin
Goc:=Goc*Pi/180;
P.x:=P.x-Tam.x;
P.y:=P.y-Tam.y;
Q.x:=Round(P.x*Cos(goc)-P.y*Sin(goc));
Q.y:=Round(P.x*Sin(goc)+P.y*Cos(goc));
Q.x:=Q.x+Tam.x;
Q.y:=Q.y+Tam.y;
Quay:=Q;
end;
procedure
Vecung(Pic:Tbitmap;T1,T2:TypeToaDo;Gia:Real;Line:Boolean;LineColor,TextColor:
Tcolor);
var DX,DY,X,Y:Integer;P,Q1,Q2:TypeToaDo;L,TL:real;St:String;
begin

DX:=T2.x-T1.x;DY:=T2.y-T1.y;
L:=sqrt(DX*DX+DY*DY);
if L<=2*Bankinh then exit;
TL:=BanKinh/L;
Q1.X:=round(T1.x+DX*TL);
Q1.Y:=round(T1.y+DY*TL);
Q2.X:=round(T2.x-DX*TL);
Q2.Y:=round(T2.y-DY*TL);
T1:=Q1;T2:=Q2;
DX:=T2.x-T1.x;DY:=T2.y-T1.y;
L:=sqrt(DX*DX+DY*DY);
If L=0 then exit;
TL:=RMuiTen/L;
P.X:=round(T2.x-DX*TL);
P.Y:=round(T2.y-DY*TL);
Q1:=Quay(P,T2,-35);
Q2:=Quay(P,T2,35);
pic.Canvas.Brush.Style:=bsSolid;
pic.Canvas.Brush.Color:=LineColor;
pic.Canvas.Pen.Color:=LineColor;

If Line then
   begin pic.Canvas.MoveTo(T1.x,T1.y); pic.Canvas.LineTo(T2.x,T2.y) end;
```

```
Pic.Canvas.Polygon([point(T2.x,T2.y),point(Q1.x,Q1.y),point((T2.x+P.x) div
2,(T2.y+P.y) div 2),point(Q2.x,Q2.y)]);
str(Gia:0:10,st);CatZeroThua(st);
Pic.Canvas.Font.Color:=TextColor;
Pic.Canvas.Font.Size:=TextSizeTrongSo;
Pic.Canvas.Brush.Style:=bsclear;
Pic.Canvas.TextOut(T2.x-((T2.x-T1.x) div 3),T2.y -((T2.y-T1.y)div 3),St);
end;

Function Delen(x,y,Width,Height:integer;DinhDown:integer):boolean;
Var i,W,H:integer;
begin
for i:=0 to G.SoDinh-1 do
   begin
      If (i<>DinhDown)and((G.DSDinh[i].ToaDo.x-
Width<x)and(x<G.DSDinh[i].ToaDo.x+Width))
            and((G.DSDinh[i].ToaDo.y-
Height<y)and(y<G.DSDinh[i].ToaDo.y+Height)) then
      begin
         Delen:=true;exit;
      end;
   end;
Delen:=false;
end;
Procedure VeDoThi(G:TypeDothi;Pic:Tbitmap;Imagelist:Timagelist);
Var i,j:integer;R:Trect;W,H:Integer; T1,T2:TypeToaDo;LineColor,TextColor:Tcolor;
Bitmap:Tbitmap;
begin
Pic.Canvas.Brush.Style:=bsSolid;
Pic.Canvas.Pen.Style:=psSolid;
Pic.Canvas.Brush.Color:=rgb(255,255,255);
Pic.Canvas.Pen.Color:=rgb(255,255,255);
Pic.Canvas.FillRect(Rect(0,0,Pic.Width,Pic.Height));
Bitmap:=Tbitmap.Create;
Bitmap.PixelFormat:=Pf24bit;
For i:=0 to G.SoDinh-1 do
   with G.DSDinh[i]  do
      begin
         W:=Imagelist.Width;      H:=Imagelist.Height;
         Imagelist.GetBitmap(MucKichHoat,Bitmap);
         R:=Rect(Toado.x-(W div 2),ToaDo.y-(H div 2),Toado.x+(W div
2),ToaDo.y+(H div 2));
         //Pic.Canvas.Draw(Toado.x-(W div 2),ToaDo.y-(H div 2),Bitmap);
         Pic.Canvas.Brush.Style:=bsClear;
         Pic.Canvas.BrushCopy(R,Bitmap,Rect(0,0,Bitmap.Width-1,Bitmap.Height-
1),RGB(255,255,255));
         Bitmap.FreeImage;
```

```
            Pic.Canvas.Font.Color:=rgb(0,255,0);
            Pic.Canvas.Brush.Style:=bsClear;
            W:=Pic.Canvas.TextWidth(ten);
            H:=Pic.Canvas.TextHeight(ten);
            If W<Imagelist.Width then
               Pic.Canvas.TextRect(R,Toado.x-(W div 2),ToaDo.y-(H div 2),ten )
            else
               Pic.Canvas.TextRect(R,R.Left,ToaDo.y-(H div 2),ten );
         end;
Bitmap.Free;
LineColor:=RGB(0,0,255);
TextColor:=RGB(255,0,0);
for i:=0 to G.SoCanh -1 do
   with G.DSCanh[i] do
      begin
         T1:=G.DsDinh[DinhDau].ToaDo;
         T2:=G.DsDinh[DinhCuoi].ToaDo;
         Vecung(Pic,T1,T2,Trongso.Gia,true,LineColor,TextColor);
      end;
end;

procedure KhuKichHoatThua(Var G:TypeDothi);
var i,count:integer;
begin
count:=0;
for i:=0 to G.SoDinh-1 do
   begin
      if (G.DSDinh[i].MucKichHoat>0)and(count<2) then
         begin count:=count+1;
              If count=2 then break;
         end;
   end;
if count>0 then
   for i:=0 to G.SoDinh-1 do
      if G.DSDinh[i].MucKichHoat=1 then
         G.DSDinh[i].MucKichHoat:=2
         else
      if G.DSDinh[i].MucKichHoat=2 then
         if count=2 then G.DSDinh[i].MucKichHoat:=0
end;
Function TimCacDinhKichHoat(G:TypeDoThi;Var D1,D2:integer):Integer;
var i,count:integer;
begin
count:=0;   i:=0;
while i<=G.SoDinh -1 do
   begin
   if G.DSDinh[i].MucKichHoat>0 then
      begin
```

```
            count:=count+1;
            If G.DSDinh[i].MucKichHoat=1 then D1:=i else D2:=i;
            If count=2 then i:=G.SoDinh
        end;
        i:=i+1;
    end;
TimCacDinhKichHoat:=count;
end;
function TimCung(G:TypeDoThi;D1,D2:integer; var Chiso:integer):Boolean;
var i:integer;
begin
    Timcung:=false;
    for i:=0 to G.SoCanh -1 do
        If (G.DSCanh[i].DinhDau=D1)and(G.DSCanh[i].DinhCuoi=D2) then
            begin
                ChiSo:=i;
                TimCung:=true;
                exit;
            end;
end;
procedure Tform2.HienThamSoCung(G:TypeDoThi);
var i,D1,D2,count,loi:integer;St:string;
begin
maskedit1.Enabled:=False;maskedit1.Text:='';
maskedit2.Enabled:=False;maskedit2.Text:='';
statictext1.Caption:='';
statictext2.Caption:='';
If TimCacDinhKichHoat(G,D1,D2)=2 then
    begin count:=0;
        maskedit1.Enabled:=False;maskedit1.Text:='';
        maskedit2.Enabled:=False;maskedit2.Text:='';
        statictext1.Caption:='';
        statictext2.Caption:='';
        SpeedButton1.Down:=False;
        SpeedButton2.Down:=False;
        i:=0;
        while i<=(G.SoCanh-1) do
            begin
            if (G.DSCanh[i].DinhDau=D2)and(G.DSCanh[i].DinhCuoi=D1) then
                begin
                    statictext1.Caption:=G.DSDinh[D2].Ten + '--->' + G.DSDinh[D1].Ten;
                    str(G.DSCanh[i].TrongSo.Gia:0:10,st);
                    catzerothua(st);
                    maskedit1.Text:=(st);
                    maskedit1.Enabled:=true;
                    SpeedButton1.Down:=True;
                    Count:=count+1;
                    If count=2 then i:=G.SoCanh;
```

```
            end
          else
          if (G.DSCanh[i].DinhDau=D1)and(G.DSCanh[i].DinhCuoi=D2) then
            begin
              statictext2.Caption:=G.DSDinh[D2].Ten + '<---' + G.DSDinh[D1].Ten;
              str(G.DSCanh[i].TrongSo.Gia:0:0,st);
              catzerothua(st);
              maskedit2.Text:=st;
              maskedit2.Enabled:=true;
              SpeedButton2.Down:=True;
              Count:=count+1;
              If count=2 then i:=G.SoCanh;
            end;
          i:=i+1;
          end;
      //bitbtn2.Enabled:=True;
      //bitbtn3.Enabled:=True;
      SpeedButton1.Enabled:=True;
      SpeedButton2.Enabled:=True;
   end
else
   begin
      //bitbtn2.Enabled:=False;
      //bitbtn3.Enabled:=False;
      SpeedButton1.Enabled:=False;
      SpeedButton2.Enabled:=False;
   end;

end;
procedure TForm2.PaintBox1MouseDown(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
var i:integer;T:Tpoint;
begin
i:=DownDinh(x,y,G);

If (button=mbRight)and(i<>-1) then
   begin
      DinhDown:=i;
      T:=PaintBox1.ClientToScreen(Point(x,y));
      PopupMenu1.Popup(T.X,T.Y);
      exit;
   end;
If i<>-1 then
   begin
      Mouse_Down:=true;
      DinhDown:=i;
      if G.DSDinh[i].MucKichHoat=0 then
         begin
```

```
            KhuKichHoatThua(G);
            G.DSDinh[i].MucKichHoat:=1;
            Dx:=x-G.DSDinh[i].ToaDo.x;
            Dy:=y-G.DSDinh[i].ToaDo.y;
         end
      else
         G.DSDinh[i].MucKichHoat:=0;
      HienThamSoCung(G);
   end;
end;

procedure TForm2.PaintBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
Var H:Integer;
begin
if {(Sender is TListBox) and} (Source is Timage) then
   If Timage(Source).Name ='Image1' then
      begin
            G.SoDinh:=G.SoDinh+1;
            Setlength(G.DSDinh,G.SoDinh);
            G.DSDinh[G.SoDinh-1].ToaDo.X:=x;
            G.DSDinh[G.SoDinh-1].ToaDo.Y:=y;
            G.DSDinh[G.SoDinh-1].Ten:='T' + InttoStr(G.SoDinh);
            VeDoThi(G,Pic,imagelist1);
            DrawPaint(PaintBox1,Pic);
            FileChanged:=true;
      end;
end;

procedure TForm2.PaintBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
Var i:integer;
begin
Accept:=true;
i:=0;
While i<=(G.SoDinh-1) do
   if not Delen(x,y,imagelist1.Width,imagelist1.Height,i) then
      i:=i+1
   else
      begin
         Accept:=False;
         i:=G.SoDinh;
      end;
If Accept then
   begin
      VeDoThi(G,Pic,imagelist1);
      Pic.Canvas.Draw(x+20,y,Image1.Picture.Bitmap);
      DrawPaint(PaintBox1,Pic);
   end
```

```
else
   begin
      VeDoThi(G,Pic,imagelist1);
      DrawPaint(PaintBox1,Pic);
   end;
end;

procedure TForm2.FormResize(Sender: TObject);
begin
If (self.WindowState<>wsMinimized)and((pic is Tbitmap)) then
   begin
      Pic.Width:=Paintbox1.Width;
      Pic.Height:=Paintbox1.Height;
   end;
end;

procedure TForm2.FormCreate(Sender: TObject);
begin
Pic:=Tbitmap.Create;
Pic.PixelFormat:=Pf24bit;
Pic.Width:=Paintbox1.Width;
Pic.Height:=Paintbox1.Height;
FileChanged:=false;
Filename:='';
Self.Caption:='Graph Algorithm - New documents'
end;

function TForm2.DownDinh(x,y:integer;G:TypeDothi):integer;
var i:integer;
begin
   For i:=0 to G.Sodinh-1 do
      with G.DSDinh[i] do
         If Sqrt(sqr(Toado.x-x)+sqr(Toado.y-y))<20 then
            begin
               DownDinh:=i; exit;
            end;
   DownDinh:=-1;
end;

procedure TForm2.PaintBox1MouseMove(Sender: TObject; Shift: TShiftState; X,
  Y: Integer);
begin
If mouse_Down then
   begin
      if (not Delen(x,y,imagelist1.Width,imagelist1.Height,DinhDown))
         and((0<x)and(x<Pic.Width)and(0<y)and(y<Pic.Height)) then
         begin
            G.DSDinh[DinhDown].ToaDo.x:=x-Dx;
```

```
            G.DSDinh[DinhDown].ToaDo.y:=y-Dy;
            VeDoThi(G,Pic,imagelist1);
            DrawPaint(PaintBox1,Pic);
          end
   end
else
   begin
   end;
end;

procedure TForm2.PaintBox1MouseUp(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
begin
If mouse_Down then
   if (not Delen(x,y,imagelist1.Width,imagelist1.Height,DinhDown))
      and((0<x)and(x<Pic.Width)and(0<y)and(y<Pic.Height)) then
   begin
      G.DSDinh[DinhDown].ToaDo.x:=x-Dx;
      G.DSDinh[DinhDown].ToaDo.y:=y-Dy;
      mouse_Down:=false;
      VeDoThi(G,Pic,imagelist1);
      DrawPaint(PaintBox1,Pic);
      FileChanged:=True;
   end
   else
   begin
      mouse_Down:=false;
   end
end;

procedure TForm2.MaskEdit1Change(Sender: TObject);
var D1,D2,ChiSo,Loi:integer; X:real;
begin
if not maskedit1.Focused then exit;
val(maskedit1.Text,X,Loi);
If TimCacDinhKichHoat(G,D1,D2)=2 then
   if Timcung(G,D2,D1,ChiSo) then
      begin G.DSCanh[ChiSo].TrongSo.Gia:=X;
          VeDoThi(G,Pic,imagelist1);
          DrawPaint(PaintBox1,Pic);
      end;
end;

procedure TForm2.MaskEdit2Change(Sender: TObject);
var D1,D2,ChiSo,Loi:integer; X:real;
begin
if not maskedit2.Focused then exit;
val(maskedit2.Text,X,Loi);
```

```
If TimCacDinhKichHoat(G,D1,D2)=2 then
   if Timcung(G,D1,D2,ChiSo) then
      begin
         G.DSCanh[ChiSo].TrongSo.Gia:=X;
         VeDoThi(G,Pic,imagelist1);
         DrawPaint(PaintBox1,Pic);
      end;
end;

procedure TForm2.PaintBox1Paint(Sender: TObject);
begin
//VeDoThi(G,Pic,imagelist1);
DrawPaint(PaintBox1,Pic);
end;
Function TrongSo(DinhDau,DinhCuoi:Integer):TypeChiPhi;
Var i:integer;
begin
Trongso.VoCung:=true;
i:=0;
While (i<=(G.SoCanh-1)) do
    If (G.DSCanh[i].DinhDau=DinhDau)and(G.DSCanh[i].DinhCuoi=DinhCuoi) then
       begin
          TrongSo:=G.DSCanh[i].TrongSo;
          i:=G.SoCanh;
       end
    else i:=i+1;
end;
procedure TForm2.imduongdingannhat2Click(Sender: TObject);
Var D1,D2,i,x,y:integer;ChiPhi:real;DuongDi:TypeDuongDi;St,So:string;
TimeNow:TDateTime;

SubPic:Tbitmap;
begin
If TimCacDinhKichHoat(G,D1,D2)=2 then
   begin
      If DuongDiNganNhat(G,D2,D1,DuongDi,ChiPhi) then
         begin
            SubPic:=Tbitmap.Create;
            Imagelist2.GetBitmap(0,SubPic);

            x:=G.DSDinh[DuongDi[0]].ToaDo.x;
            y:=G.DSDinh[DuongDi[0]].ToaDo.y;
            Pic.Canvas.Brush.Style:=BSclear;
            Pic.Canvas.BrushCopy(rect(x,y-
SubPic.Height,x+Subpic.Width,y),SubPic,Rect(0,0,SubPic.Width-1,SubPic.Height-
1),RGB(255,255,255));
            for i:=0 to high(DuongDi)-1 do
               begin
```

```
Veline(G.DSDinh[DuongDi[i]].ToaDo,G.DSDinh[DuongDi[i+1]].ToaDo,

TrongSo(DuongDi[i],DuongDi[i+1]).Gia,Pic,RGB(255,0,0),100000);
                TimeNow:=Time;
                repeat
                      Application.ProcessMessages;
                until (TimeNow+100000)>Time;
              end;
           St:='Duong di Tu ' + G.DSDinh[D1].Ten + ' Den ' + G.DSDinh[D2].Ten +'
la:' + Cr + Lf;
           for i:=0 to high(DuongDi)-1 do
              begin
                st:=st+G.DsDinh[DuongDi[i]].Ten +' --> ';

Vecung(Pic,G.DSDinh[DuongDi[i]].ToaDo,G.DSDinh[DuongDi[i+1]].ToaDo,

TrongSo(DuongDi[i],DuongDi[i+1]).Gia,True,RGB(255,0,0),RGB(0,0,255))

//Veline(G.DSDinh[DuongDi[i]].ToaDo,G.DSDinh[DuongDi[i+1]].ToaDo,
                //
TrongSo(DuongDi[i],DuongDi[i+1]).Gia,Pic,RGB(255,0,0),10000)

              end;

           st:=st+G.DsDinh[DuongDi[high(DuongDi)]].Ten+ cr+lf;
           Str(ChiPhi:0:10,So);Catzerothua(So);
           St:=St+ 'Voi chi phi la: ' + So;

           Pic.Canvas.BrushCopy(rect(x,y-
SubPic.Height,x+Subpic.Width,y),SubPic,Rect(0,0,SubPic.Width-1,SubPic.Height-
1),RGB(255,255,255));
           x:=G.DSDinh[DuongDi[high(DuongDi)]].ToaDo.x;
           y:=G.DSDinh[DuongDi[high(DuongDi)]].ToaDo.y;
           Pic.Canvas.Brush.Style:=BSclear;
           Imagelist2.GetBitmap(1,SubPic);
           Pic.Canvas.BrushCopy(rect(x,y-
SubPic.Height,x+Subpic.Width,y),SubPic,Rect(0,0,SubPic.Width-1,SubPic.Height-
1),RGB(255,255,255));
           SubPic.Free;

           DrawPaint(PaintBox1,Pic);
           showmessage(st);
        end
     else
        begin
           Showmessage('Khong co duong di Tu ' + G.DSDinh[D1].Ten + ' Den ' +
G.DSDinh[D2].Ten);
```

```
            end;
    end;

end;

procedure TForm2.FormCloseQuery(Sender: TObject; var CanClose: Boolean);
var TraLoi:Word;
begin
If FileChanged then
   begin
      TraLoi:=MessageDlg('File changed. Do you want to save?',mtConfirmation
,[mbYes,mbNo,mbCancel],0);
      If TraLoi=mrYes then
         Form2.Save1Click(Sender)
      else
      If TraLoi=mrCancel then
         begin CanClose:=false; exit;end;
   end;

pic.FreeImage;
DeleteGraph(G);
end;

procedure TForm2.FormDestroy(Sender: TObject);
begin
pic.FreeImage;
end;

procedure TForm2.Rename1Click(Sender: TObject);
begin
G.DSDinh[DinhDown].Ten:=inputbox('Rename','Name:',G.DSDinh[DinhDown].Ten);
HienThamSoCung(G);
VeDoThi(G,Pic,imagelist1);
DrawPaint(PaintBox1,Pic);
FileChanged:=True;
end;

procedure TForm2.Exit1Click(Sender: TObject);
begin
close;
end;

procedure TForm2.Delete1Click(Sender: TObject);
Var i,N,Start:integer;
Index:Array of integer;
begin
For i:=DinhDown to G.SoDinh-2 do
   G.DSDinh[i]:=G.DSDinh[i+1];
```

```
G.SoDinh:=G.SoDinh-1;
Setlength(G.DSDinh,G.SoDinh);

Setlength(Index,G.SoCanh);
N:=0;Start:=-1;
For i:=0 to G.SoCanh-1 do
   If (G.DSCanh[i].DinhDau=DinhDown)or(G.DSCanh[i].DinhCuoi=DinhDown) then
      begin
         If Start=-1 then Start:=N;
      end
   else
      begin
         Index[N]:=i;
         N:=N+1;
      end;

If Start<>-1 then
   begin
      G.SoCanh:=N;
      For i:=Start to G.SoCanh-1 do
         G.DSCanh[i]:=G.DSCanh[Index[i]];
      For i:=0 to G.SoCanh-1 do
         With G.DSCanh[i] do
            begin
               If DinhDau>DinhDown then DinhDau:=DinhDau-1;
               If DinhCuoi>DinhDown then DinhCuoi:=DinhCuoi-1;
            end;
      Setlength(G.DSCanh,G.SoCanh);
   end;
Setlength(Index,0);
HienThamSoCung(G);
VeDoThi(G,Pic,imagelist1);
DrawPaint(PaintBox1,Pic);
FileChanged:=True;
end;

procedure TForm2.DeleteAll1Click(Sender: TObject);
begin
G.SoDinh:=0;G.SoCanh:=0;
Setlength(G.DSDinh,0);Setlength(G.DSCanh,0);
Pic.Canvas.Brush.Style:=bsSolid;
Pic.Canvas.Pen.Style:=psSolid;
Pic.Canvas.Brush.Color:=rgb(255,255,255);
Pic.Canvas.Pen.Color:=rgb(255,255,255);
Pic.Canvas.FillRect(Rect(0,0,Pic.Width,Pic.Height));
DrawPaint(PaintBox1,Pic);
FileChanged:=true;
end;
```

```
procedure TForm2.Save1Click(Sender: TObject);
var  F:textfile;
i:integer;
begin
SaveDialog1.DefaultExt:='*.GRD';
SaveDialog1.Filter:='Graph data file (*.GRD)|*.GRD';
If not SaveDialog1.Execute then exit;

AssignFile(F,SaveDialog1.FileName);
Rewrite(F);
Try
   Writeln(f,G.Sodinh,' ',G.Socanh);
   For i:=0 to G.SoDinh-1 do
      Writeln(F,G.DSDinh[i].ToaDo.x,' ',G.DSDinh[i].ToaDo.y,' ',G.DSDinh[i].Ten);

   For i:=0 to G.SoCanh-1 do
      Writeln(F,G.DSCanh[i].DinhDau,' ',G.DSCanh[i].DinhCuoi,'
',G.DSCanh[i].TrongSo.Gia);
except
   Showmessage('Writting error');
end;
CloseFile(F);
FileChanged:=false;
end;

procedure TForm2.Open1Click(Sender: TObject);
Var F:TextFile;
i:integer;
begin
OpenDialog1.DefaultExt:='*.GRD';
OpenDialog1.Filter:='Graph data file (*.GRD)|*.GRD';
If not OpenDialog1.Execute then exit;
AssignFile(F,OpenDialog1.FileName);

ReSet(F);
Try
   Readln(f,G.Sodinh,G.Socanh);
   Setlength(G.DSDinh,G.SoDinh);
   Setlength(G.DSCanh,G.SoCanh);

   For i:=0 to G.SoDinh-1 do
      begin
         Readln(F,G.DSDinh[i].ToaDo.x,G.DSDinh[i].ToaDo.y,G.DSDinh[i].Ten);
         G.DSDinh[i].Ten:=trimleft(G.DSDinh[i].Ten);
         G.DSDinh[i].MucKichHoat:=0;
      end;
```

```
    For i:=0 to G.SoCanh-1 do

Readln(F,G.DSCanh[i].DinhDau,G.DSCanh[i].DinhCuoi,G.DSCanh[i].TrongSo.Gia);

except
   DeleteGraph(G);
   showmessage('Error struct file');
   CloseFile(F);
   Self.Caption:='Graph Algorithm - New document';
   VeDoThi(G,Pic,imagelist1);
   DrawPaint(PaintBox1,Pic);
   exit;
end;
CloseFile(F);
VeDoThi(G,Pic,imagelist1);
DrawPaint(PaintBox1,Pic);
Filename:=OpenDialog1.FileName;
Self.Caption:='Graph Algorithm - ' + Filename;
FileChanged:=False;
end;

procedure TForm2.SpeedButton1Click(Sender: TObject);
var D1,D2,ChiSo,i:integer;
begin
TimCacDinhKichHoat(G,D1,D2);
If Not SpeedButton1.Down then
   begin
      Timcung(G,D2,D1,ChiSo);
      for i:=Chiso to G.SoCanh-2 do
         G.DSCanh[i]:=G.DSCanh[i+1];
      G.SoCanh:=G.SoCanh-1;
      Setlength(G.DSCanh,G.SoCanh);
   end
else
   begin
      G.SoCanh:=G.SoCanh+1;
      Setlength(G.DSCanh,G.SoCanh);
      With G.DSCanh[G.SoCanh-1] do
         begin
            DinhDau:=D2;
            DinhCuoi:=D1;
            TrongSo.VoCung:=false;
            TrongSo.Gia:=0;
         end;

   end;
HienThamSoCung(G);
VeDoThi(G,Pic,imagelist1);
```

```
DrawPaint(PaintBox1,Pic);
end;

procedure TForm2.SpeedButton2Click(Sender: TObject);
var D1,D2,ChiSo,i:integer;
begin
TimCacDinhKichHoat(G,D1,D2);

If not SpeedButton2.Down then
   begin
      Timcung(G,D1,D2,ChiSo);
      for i:=Chiso to G.SoCanh-2 do
         G.DSCanh[i]:=G.DSCanh[i+1];
      G.SoCanh:=G.SoCanh-1;
      Setlength(G.DSCanh,G.SoCanh);
   end
else
   begin
      G.SoCanh:=G.SoCanh+1;
      Setlength(G.DSCanh,G.SoCanh);
      With G.DSCanh[G.SoCanh-1] do
         begin
            DinhDau:=D1;
            DinhCuoi:=D2;
            TrongSo.VoCung:=false;
            TrongSo.Gia:=0;
         end;

   end;
HienThamSoCung(G);
VeDoThi(G,Pic,imagelist1);
DrawPaint(PaintBox1,Pic);

end;

procedure TForm2.New1Click(Sender: TObject);
begin
Filename:='';

FileChanged:=false;
DeleteGraph(G);

VeDoThi(G,Pic,imagelist1);
DrawPaint(PaintBox1,Pic);
end;

procedure TForm2.ExportPicturefile2Click(Sender: TObject);
Var T:TJpegimage;
```

www.VIETMATHS.com

```
begin
SaveDialog1.DefaultExt:='*.JPG';
SaveDialog1.Filter:='Bitmap image      (*.BMP)|*.BMP|Jpeg image
(*.JPG)|*.JPG';
SaveDialog1.FilterIndex:=2;
If not SaveDialog1.Execute then exit;
case SaveDialog1.FilterIndex of
1:{BMP}
      Pic.SaveToFile(SaveDialog1.FileName);
2:{Jpeg}
      begin
         T:=TJpegimage.Create;
         T.Assign(Pic);
         try
            T.SaveToFile(SaveDialog1.FileName);
         finally
            T.Free
         end;
      end;
end

end;

end.
```

Chương trình chính cài đặt như sau:

```
program Project1;

uses
  Forms,
  Func_DoThi in 'Func_DoThi.pas',
  Unit2 in 'Unit2.pas' {Form2},

{$R *.res}

begin
  Application.Initialize;
  Application.CreateForm(TForm2, Form2);
  Application.Run;
end.
```

 www.VIETMATHS.com

PHẦN PHỤ LỤC

Phụ lục 2

Bài toán luồng cực đại

Cho mạng G=(V,E). Hãy tìm luồng f* trong mạng với giá trị luồng val(f*) là lớn nhất. Luồng như vậy ta sẽ gọi là luồng cực đại trong mạng**.**

Bài toán như vậy có thể xuất hiện trong rất nhiều ứng dụng thực tế. Chẳng hạn khi cần xác định cường độ lớn nhất của dòng vận tải giữa hai nút của một bản đồ giao thông. Trong thí dụ này lời giải của bài toán luồng cực đại sẽ chỉ cho ta các đoạn đường xe đông nhất và chúng tạo thành chỗ hẹp tương ứng của dòng giao thông xét theo hai nút đã chọn. Một thí dụ khác là nếu xét đồ thị tương ứng với một hệ thống đường ống dẫn dầu, trong đó các ống tương ứng với các cung, điểm phát có thể coi là tàu chở dầu, điểm thu là bể chứa, còn các điểm nối giữa các ống là các nút của đồ thị, khả năng thông qua của các cung tương ứng với tiết diện các ống. Cần phải tìm luồng dầu lớn nhất có thể bơm dầu từ tàu chở dầu vào bể chứa.

Định lý: Các mệnh đề dưới đây là tương đương:

(i) f là luồng cực đại trong mạng.

(ii) Không tìm được đường tăng luồng f.

(iii) Val(f)=c(X,X*) với một lát cắt (X,X*) nào đó.

(Ta gọi lát cắt (X,X*) là một cách phân hoạch tập đỉnh V của mạng ra thành hai tập X và X*=V\X, trong đó $s \in X$ và $t \in X^*$.)

Định lý trên là cơ sở để xây dựng thuật toán lặp sau đây để tìm luồng cực đại trong mạng: Bắt đầu từ luồng trên tất cả các cung bằng 0 (ta sẽ gọi luồng như vậy là luồng không), và lặp lại bước lặp sau đây cho đến khi thu được luồng mà đối với nó không còn đường tăng:

Bước lặp tăng luồng (Ford – Fulkerson): Tìm đường tăng P đối với luồng hiện có, tăng luồng dọc theo đường P.

Khi đã có luồng cực đại, lát cắt hẹp nhất có thể tìm theo thủ tục mô tả trong việc chứng minh định lý trên. Thuật toán Ford-Fulkerson được mô tả trong thủ tục sau đây:

```
Procedure Luongcucdai;
  Begin
    Stop := false;
    While not Stop do
        If < Tìm đường tăng luồng P> then
                < Tăng luồng dọc theo P>
        Else   Stop := true;
      End;
```


Để tìm đường tăng luồng trong G(f) có thể sử dụng thuật toán tìm kiếm theo chiều rộng (hay tìm kiếm theo chiều sâu), bắt đầu từ đỉnh s trong đó không cần xây dựng tường minh đồ thị G(f). Ford-Fulkerson đề nghị thuật toán gán nhãn chi tiết sau đây để giải bài toán luồng cực đại trong mạng. Thuật toán bắt đầu từ luồng chấp nhận được nào đó trong mạng (có thể bắt đầu từ luồng không) , sau đó ta sẽ tăng luồng bằng cách tìm các đường tăng luồng. Để tìm đường tăng luồng ta sẽ áp dụng phương pháp gán nhãn cho các đỉnh. Mỗi đỉnh trong quá trình thực hiện thuật toán sẽ ở một trong ba trạng thái: chưa có nhãn, có nhãn chưa xét, có nhãn đã xét. Nhãn của một đỉnh v gồm hai phần và có một trong hai dạng sau : $[+p(v), \varepsilon(v)]$ hoặc $[-p(v), \varepsilon(v)]$. Phần thứ nhất +p(v) (-p(v)) chỉ ra là cần tăng giảm luồng theo cung (p(v),v)(cung (v,p(v)) còn phần thứ hai $\varepsilon(v)$ chỉ ra lượng lớn nhất có thể tăng hoặc giảm luồng theo cung này. Đầu tiên chỉ có đỉnh s được khởi tạo nhãn và nhãn của nó là chưa xét, còn tất cả các đỉnh còn lại đều chưa có nhãn. Từ s ta gán nhãn cho tất cả các đỉnh kề với nó và nhãn của đỉnh s sẽ trở thành đã xét. Tiếp theo, từ một đỉnh v có nhãn chưa xét ta lại gán nhãn cho tất cả các đỉnh chưa có nhãn kề với nó và nhãn của đỉnh v trở thành đã xét. Quá trình sẽ được lặp lại cho đến khi hoặc là đỉnh t trở thành có nhãn hoặc là nhãn của tất cả các đỉnh có nhãn đầu là đã xét nhưng đỉnh t vẫn không có nhãn. Trong trường hợp thứ nhất ta tìm được đường tăng luồng, còn trong trường hợp thứ hai đối với luồng đang xét không tồn tại đường tăng luồng (tức là luồng đã cực đại). Mỗi khi tìm được đường tăng luồng, ta lại tăng luồng theo đường tìm được, sau đó xoá tất cả các nhãn và đối với luồng mới thu được lại sử dụng phép gán nhãn các đỉnh để tìm đường tăng luồng. Thuật toán sẽ kết thúc khi nào đối với luồng đang có trong mạng không tìm được đường tăng luồng.

Hai thủ tục tìm đường tăng luồng có thể mô tả như sau :

```
Procedure Find-path;
    {
      Thủ tục gán nhãn đường tăng luồng
      p[v], ∈ ε [v] là nhãn của đỉnh v;
      V_T là danh sách các đỉnh có nhãn chưa xét ;
      c[u,v] là khả năng thông qua của cung (u,v),u,v ∈ V;
      f[u,v] là luồng trên cung (u,v), (u,v ∈ V);
     }
    BEGIN
     p[s] := s ;
     ε [s] :=+∞ ;
     V_T := {s};
     Pathfound := true;
     While V_T<> {} do
```

```
            BEGIN
              u ⇐ V_T ;( * lấy u từ V_T *)
              For v∈V do
               If (v chưa có nhãn)  then
                Begin
                 If (c[u,v] >0) and  (f[u,v] < c[u,v] ) then
                  Begin
                   P[v] := u ;
                   ε[v] := min { ε[u],c[u,v]-f[u,v] };
                   V_T:=V_T ∪ {v};(* nạp v vào danh sách các đỉnh có nhãn *)
                   If v = t then exit;
                  End
                 Else
                 If (c[v,u] > 0) and (f[v,u] < 0) then
                  Begin
                   P[v] := u ;
                   ε[v] := min { ε[u] , f[u,v] };
                   V_T:=V_T ∪ {v};(* nạp v vào danh sách các đỉnh có nhãn *)
                   If v = t then exit;
                  End;
               End;
End;
 PathFound :=false;
End;
Procedure Inc_flow ;
   { thuật toán tăng luồng theo đường tăng }
 Begin
   v := t ;
   u := t ;
   tang := [t];
   while u <> s do
        begin
          v := p[u];
          if v > 0 then  f[v,u] := f[v,u] + tang
          else
               begin
                 v := -v;
                 f[u,v] :=f[u,v] –tang;
               end;
          u := v ;
```

```
end;

Procedure FF;
 { thủ tục thể hiện thuật toán Ford_fulkerson }
  Begin
   (* khởi tạo bắt đầu từ luồng với giá trị 0 *)
   For u ∈ V do
        For v ∈ V do f[u,v] :=0;
        Stop := false;
   While not Stop do
        begin
             find_path;
            If  pathfound  then
                  Inc_flow
            Else
                  Stop:=true;
        End;
< Luồng cực đại trong mạng là f[u,v], u,v ∈ V >
< Lát cắt hẹp nhất là (V_T , V\ V_T) >
   End;
```

Chương trình sau là chương trình phục vụ cho việc học tập và giảng dạy về bài toán tìm luồng cực đại trong mạng. Chương trình sau được xây dựng bằng công cụ lập trình Delphi.

Các chức năng của chương trình: Ta xây dựng chương trình bao gồm những chức năng sau:

* Tóm tắt thuật toán Ford – Fulkeson.
* Hiển thị các bước thực hiện ứng với từng ví dụ cụ thể.

Tóm tắt thuật toán Ford – Fulkerson :

Chức năng này có mục đích giúp cho người sử dụng nắm vững được thuật toán trước khi đi vào các thí dụ cụ thể.

Hiển thị các bước thực hiện của bài toán:

Do chương trình nhằm mục đích phục vụ cho việc dạy và học môn Toán rời rạc nên chức năng việc hiển thị chi tiết các bước giải bài toán ứng với tưng thí dụ cụ thể giúp cho người sử dụng hiểu rõ hơn về thuật toán.

Cấu trúc dữ liệu và cài đặt thuật toán:

Cấu trúc dữ liệu:

Đồ thị được lưu giữ dưới dạng tập đỉnh và tập cạnh. Mỗi đỉnh được lưu theo cấu trúc của một Record như sau:

```
L_TypeDinh = record
               Ten:String;
               ToaDo:L_TypeToaDo;
               MucKichHoat:Byte;
             end;
```

Trong đó:

- Biến Ten có kiểu String , lưu giữ tên đỉnh (mặt định là V_0,V_1,…)
- Biến ToaDo có kiểu L_TypeToaDo, lưu giữ toạ độ x, y của mỗi đỉnh có cấu trúc của một Record như sau :

```
L_TypeToaDo = record
                 x,y:integer;
               end;
```

Biến Muckichhoat có kiểu Byte lưu giữ mức độ kích hoạt của đỉnh (mỗi đỉnh có 4 mức kích hoạt khác nhau), biến này dùng để xác định đỉnh đầu, đỉnh cuối, đỉnh hẹp….

Tập cạnh của đồ thị cũng được lưu theo cấu trúc của Record, cấu trúc của mỗi cạnh được lưu trữ như sau:

```
L_TypeCanh = record
               DinhDau,DinhCuoi:Integer;
             TrongSo:L_TypeChiphi;
           end;
```

trong đó :

- Biến DinhDau có kiểu Integer, lưu giữ chỉ số đỉnh đầu của cạnh .
- Biến DinhCuoi có kiểu Integer, lưu giữ chỉ số đỉnh cuối của cạnh .
- Biến TrongSo có kiểu L_TypeChiPhi, lưu giữ giá và khả năng thông qua của cạnh đang xét. Kiểu L_TypeChiPhi là một Record có dạng như sau :

```
L_TypeChiPhi = record
                 Gia:real;
               kntq:real;
                 end;
```

Cài đặt thuật toán:

Như đã trình bày ở phần trên , thuật toán Ford –Fulkerson được cài đặt bằng cách kết hợp 2 thủ tục Find-Path (thủ tục gán nhãn tìm đường tăng luồng) và Inc-Flow (thủ tục tăng luồng theo đường tăng).

Đây là phần cài đặt chi tiết của thuật toán Ford – Fulkerson (viết theo ngôn ngữ lập trình Delphi):

```
procedure L_find_path(var L_G1:L_typedothi);
{
  thu tuc gan nhan tim duong tang luong:
L_p[v],L_nhan,L_e[v] la nhan cua dinh v;
   L_v la danh sach cac dinh co nhan nhung chua xet;
  }
VAR     x,y:integer;
        ok:boolean;
        a1,b1,k1,l1:real;
        t,t1,i:integer;
BEGIN
for i:=0 to L_G1.sodinh-1 do
  L_p1[i]:=-1;
L_p1[0]:=0;
L_nhan[0]:=true;
L_e[0]:=vocung;
L_v:=[0] ; L_v1:=[0];
L_pathfound:=true;
While L_v<>[] do
 Begin
  ok:=true;
  x:=0;
  While (x<=L_G1.sodinh-1) and (ok=true) do
   Begin
    If x in L_v then   ok:=false
    Else
      x:=x+1;
 End;
L_v:=L_v-[x];
For y:=0 to L_G1.sodinh-1 do
If L_p1[y]=-1 then
   Begin
      L_giatri(L_G1,x,y,t,a1,b1);   {a:=c[x,y],b:=f[x,y]}
      L_giatri(L_G1,y,x,t1,k1,l1);   {k:=c[y,x],l:=f[y,x]}
      If (a1>0) and (b1<a1) then
        Begin
          L_p1[y]:=x;
          L_nhan[y]:=true;
          L_e[y]:=L_min(L_e[x],a1-b1);
          L_v:=L_v+[y];
```

www.VIETMATHS.com

```
                    L_v1:=L_v1+[y];
                    If y=L_G1.sodinh-1 then
                     Begin

        exit;
                     End;
                  End
                Else
                If (k1>0) and (l1>0) then
                   Begin
                     L_p1[y]:=x;
                     L_nhan[y]:=false;
                     L_e[y]:=L_min(L_e[x],l1);
                     L_v:=L_v+[y];
                     L_v1:=L_v1+[y];
                     If y=L_G1.sodinh-1 then
                       Begin
                         exit;
                       End;
                   End;
              End;
            End;
   L_pathfound:=false;
end;
procedure L_Inc_flow(var L_G1:L_typedothi);
 {
  tang luong theo duong tang
 }
  var x,y,t,t1:integer;
     tang,a,k:real;
     s,s1,s2,s3,s4:string;
     ok:boolean;
 begin
    x:=L_G1.sodinh-1;
    y:=L_G1.sodinh-1;
   tang:=L_e[L_G1.sodinh-1];
   ok:=false;
   while x<>0 do
      begin
        y:=L_p1[x];
```


www.VIETMATHS.com

```
        L_giatri(L_G1,x,y,t,a,L_b);  {a:=c[x,y],b:=f[x,y]}
       L_giatri(L_G1,y,x,t1,k,L_l);  {k:=c[y,x],l:=f[y,x]}
       if L_nhan[x] then
         L_G1.dscanh[t1].trongso.gia:=L_G1.dscanh[t1].trongso.gia+tang
      else
        begin
         L_G1.dscanh[t].trongso.gia:=L_G1.dscanh[t].trongso.gia-tang;
         ok:=true;
       end;
    x:=y;
   end;
end;
 procedure L_luongcucdai(L_G:L_typedothi; var L_G1:L_typedothi;var gt:real);
 {
  thu tuc the hien thuat toan Ford_fulkerson
 }
 var x,y,z,t,i,j,t1,t2:integer;
    a1,b1,f:real;
    ok1,stop:boolean;
    s,s1,ch,ch1,a:string;
 begin
  L_G1.SoDinh:=L_G.SoDinh ;
  L_G1.socanh:=L_G.socanh;
  setlength(L_p1,L_G1.SoDinh);
  setlength(L_nhan,L_G1.SoDinh );
  setlength(L_e,L_G1.SoDinh );
  setlength(L_G1.DSdinh,L_G1.SoDinh );
  Setlength(L_G1.dscanh,L_G1.SoCanh );
   for j:=0 to L_G.SoDinh -1 do
      L_G1.DSDinh[j]:=L_G.DSDinh[j];
   for j:=0 to L_G.SoCanh -1 do
      L_G1.DSCanh[j]:=L_G.DSCanh[j];
   stop:=false;
       while not stop do
   begin
    L_find_path(L_G1);
    if L_pathfound then
      begin
       tam:=tam+1;
        if tam>1 then
```

```
            L_inc_flow(L_G1)
        else
            stop:=true;
      end;
   f:=0;
  for y:= 0 to L_G1.sodinh-1 do
    begin
      L_giatri(L_G1,0,y,t1,a1,b1);
      f:=f+b1;
   end;
 for y:=0 to L_G1.Socanh -1 do
     if L_G1.DSCanh[y].DinhCuoi =L_G1.SoDinh -1 then
        begin
             break;
         end;
   tam:=0;
  t2:=1;
    while (t2<=L_G1.sodinh-2) do
       begin
          if t2 in L_v1 then
             L_G1.dsdinh[t2].MucKichHoat :=3
          else
             L_G1.dsdinh[t2].MucKichHoat :=0;
       end;
          t2:=t2+1;
       end;
   L_G1.dsdinh[0].MucKichHoat :=3;
   L_G1.dsdinh[L_G1.SoDinh -1].MucKichHoat :=0;
  end;
```

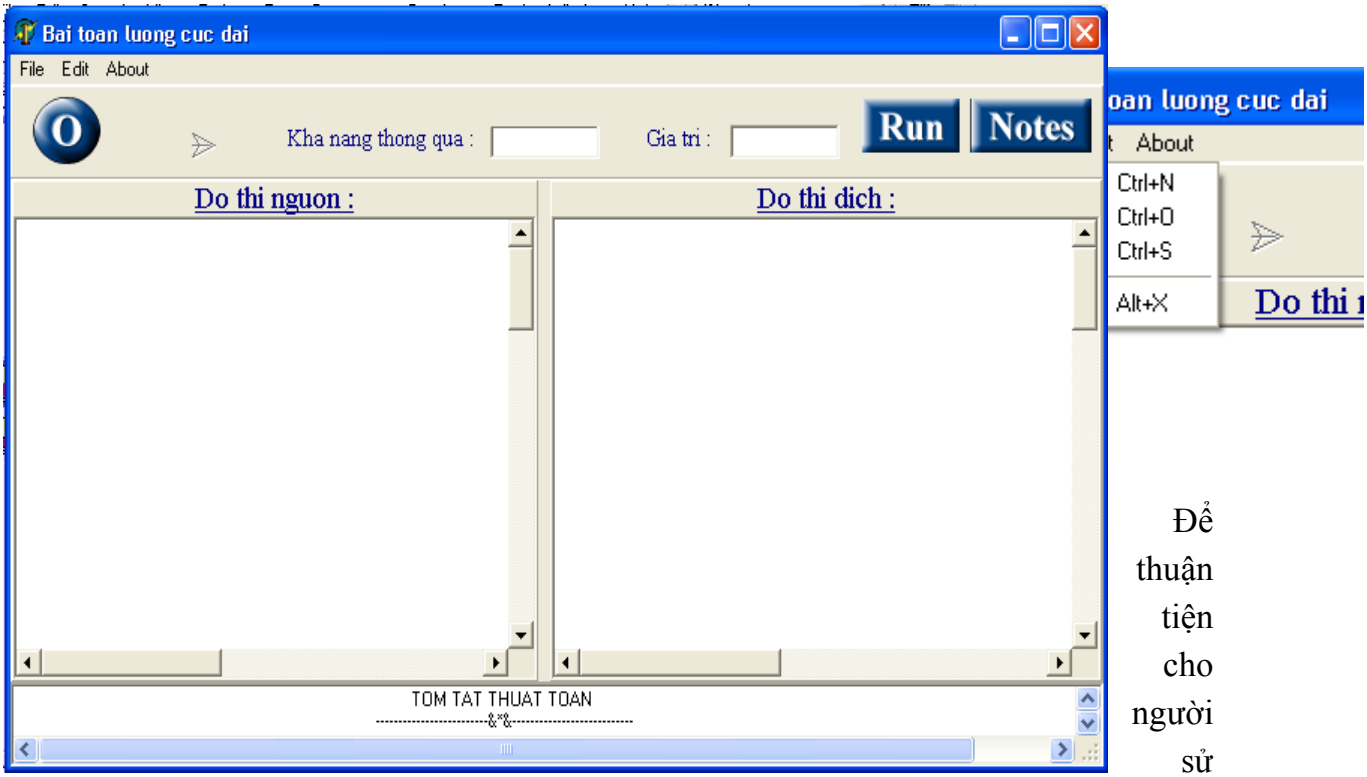
Giao diện chương trình :

Hình dưới đây là form chính của chương trình, người sử dụng có thể tự vẽ đồ thị để kiểm tra thuật toán (đồ thị được vẽ sẽ nằm ở phần đồ thị nguồn). Sau khi đã có đồ thị nguồn, muốn biết kết quả của bài toán thì ta nhấn nút Run trên thanh công cụ của form, ta sẽ được đồ thị kết quả (nằm ở phần đồ thị đích).

Các bước giải ứng với từng bài toán cụ thể được trình bày khi ta nhấn Notes. Đây là phần giúp cho người sử dụng hiểu rõ hơn về thuật toán, nó trình bày cách làm bài toán theo từng bước tương ứng với thuật toán đã nêu.

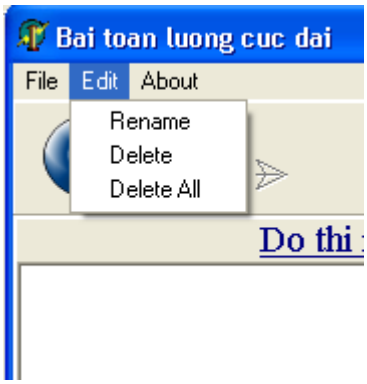
Ngoài ra, người sử dụng có thể xem lại thuật toán bằng cách click đôi vào phần dưới của form. Phần này giúp người sử dụng luôn nắm vững được thuật toán.

www.VIETMATHS.com



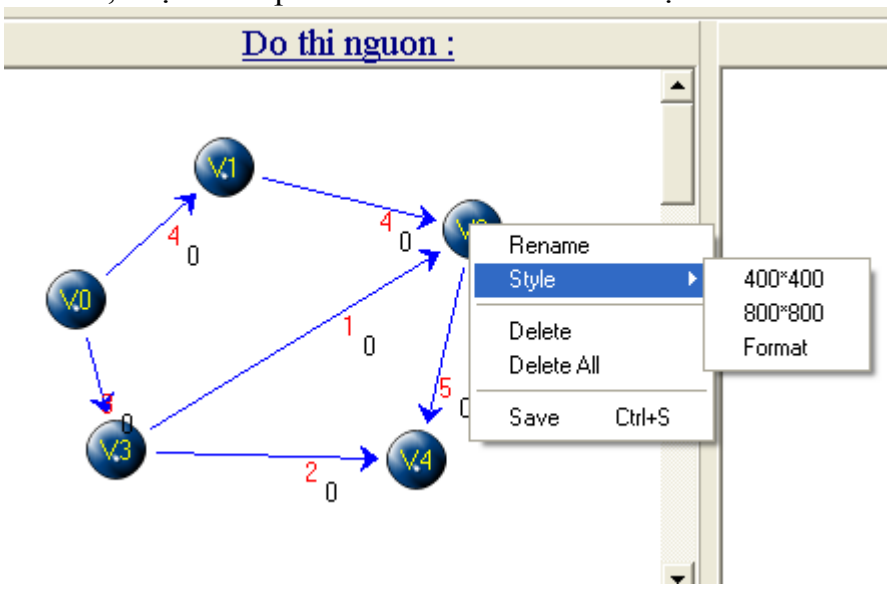
Để thuận tiện cho người sử dụng, chương trình này đã lưu sẵn một số thí dụ cụ thể để mô tả thuật toán, người sử dụng chỉ cần vào file →open, sau đó chọn một ví dụ cần xem.

Chương trình còn có chức năng giúp cho người sử dụng tạo ra các thí dụ mới và lưu lại các ví dụ vừa tạo.



Tên của các đỉnh đồ thị được mặt định là $V_0, V_1, \ldots$. Tuy nhiên chương trình có chức năng đổi tên cho đỉnh, người sử dụng có thể đổi tên đỉnh bằng cách vào Edit → rename sau đó đánh tên mới vào (xem hình bên).

Việc đổi tên đỉnh và xoá đỉnh có thể thực hiện theo hai cách, người sử dụng có thể chọn đỉnh rồi chọn Edit như cách trên, hoặc click phải vào đỉnh cần xét rồi chọn các



www.VIETMATHS.com