

CHƯƠNG 1

PHÂN TÍCH VÀ THIẾT KẾ GIẢI THUẬT

Chương 1 trình bày các khái niệm về giải thuật và phương pháp tính chỉnh từng bước chương trình được thể hiện qua ngôn ngữ diễn đạt giải thuật. Chương này cũng nêu phương pháp phân tích và đánh giá một thuật toán, các khái niệm liên quan đến việc tính toán thời gian thực hiện chương trình.

Trong mỗi phần đều có các minh họa cụ thể. Phần đầu đưa ra ví dụ về bài toán nút giao thông và phương pháp giải quyết bài toán từ phân tích vấn đề cho đến thiết kế giải thuật, tính chỉnh từng bước cho tới mức cụ thể hơn. Phần 2 đưa ra một ví dụ về phân tích và tính toán thời gian thực hiện giải thuật sắp xếp nổi bọt.

Để học tốt chương này, sinh viên cần nắm vững phần lý thuyết và tìm các ví dụ tương tự để thực hành phân tích, thiết kế, và đánh giá giải thuật.

1.1 GIẢI THUẬT VÀ NGÔN NGỮ DIỄN ĐẠT GIẢI THUẬT

1.1.1 Giải thuật

Trong thực tế, khi gặp phải một vấn đề cần phải giải quyết, ta cần phải đưa ra 1 phương pháp để giải quyết vấn đề đó. Khi muốn giải quyết vấn đề bằng cách sử dụng máy tính, ta cần phải đưa ra 1 giải pháp phù hợp với việc thực thi bằng các chương trình máy tính. Thuật ngữ “thuật toán” được dùng để chỉ các giải pháp như vậy.

Thuật toán có thể được định nghĩa như sau:

Thuật toán là một chuỗi hữu hạn các lệnh. Mỗi lệnh có một ngữ nghĩa rõ ràng và có thể được thực hiện với một lượng hữu hạn tài nguyên trong một khoảng hữu hạn thời gian.

Chẳng hạn lệnh $x = y + z$ là một lệnh có các tính chất trên.

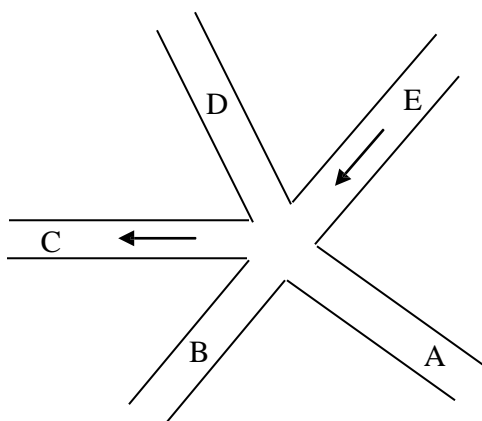
Trong một thuật toán, một lệnh có thể lặp đi lặp lại nhiều lần, tuy nhiên đối với bất kỳ bộ dữ liệu đầu vào nào, thuật toán phải kết thúc sau khi thực thi một số hữu hạn lệnh.

Như đã nói ở trên, mỗi lệnh trong thuật toán phải có ngữ nghĩa rõ ràng và có thể được thực thi trong một khoảng thời gian hữu hạn. Tuy nhiên, đôi khi một lệnh có ngữ nghĩa rõ ràng đối với người này nhưng lại không rõ ràng đối với người khác. Ngoài ra, thường rất khó để chứng minh một lệnh có thể được thực hiện trong 1 khoảng hữu hạn thời gian. Thậm chí, kể cả khi biết rõ ngữ nghĩa của các lệnh, cũng khó để có thể chứng minh là với bất kỳ bộ dữ liệu đầu vào nào, thuật toán sẽ dừng.

Tiếp theo, chúng ta sẽ xem xét một ví dụ về xây dựng thuật toán cho bài toán đèn giao thông:

Giả sử người ta cần thiết kế một hệ thống đèn cho một nút giao thông có nhiều đường giao nhau phức tạp. Để xây dựng tập các trạng thái của các đèn giao thông, ta cần phải xây dựng một chương trình có đầu vào là tập các ngã rẽ được phép tại nút giao thông (lối đi thẳng cũng được xem như là 1 ngã rẽ) và chia tập này thành 1 số ít nhất các nhóm, sao cho tất cả các ngã rẽ trong nhóm có thể được đi cùng lúc mà không xảy ra tranh chấp. Sau đó, chúng ta sẽ gán trạng thái của các đèn giao thông với mỗi nhóm vừa được phân chia. Với cách phân chia có số nhóm ít nhất, ta sẽ xây dựng được 1 hệ thống đèn giao thông có ít trạng thái nhất.

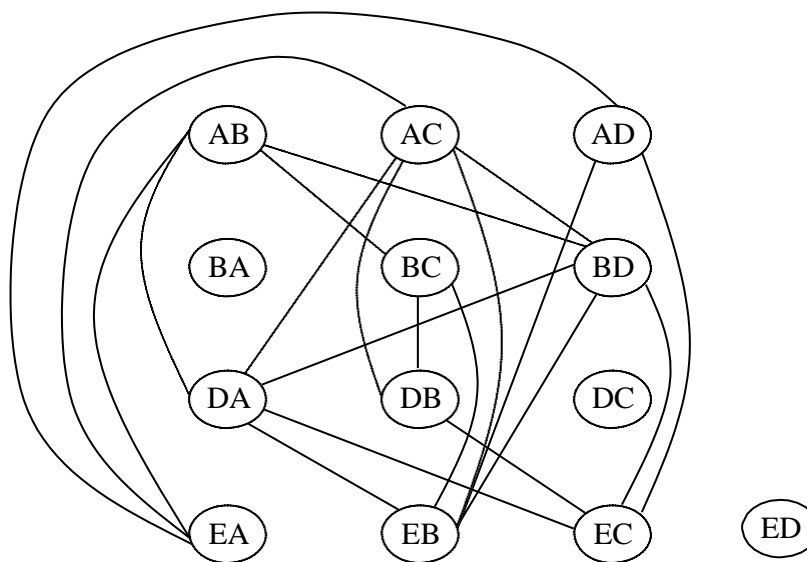
Chẳng hạn, ta xem xét bài toán trên với nút giao thông được cho như trong hình 1.1 ở dưới. Trong nút giao thông trên, C và E là các đường 1 chiều, các đường còn lại là 2 chiều. Có tất cả 13 ngã rẽ tại nút giao thông này. Một số ngã rẽ có thể được đi đồng thời, chẳng hạn các ngã rẽ AB (từ A rẽ sang B) và EC. Một số ngã rẽ thì không được đi đồng thời (gây ra các tuyến giao thông xung đột nhau), chẳng hạn AD và EB. Hệ thống đèn tại nút giao thông phải hoạt động sao cho các ngã rẽ xung đột (chẳng hạn AD và EB) không được cho phép đi tại cùng một thời điểm, trong khi các ngã rẽ không xung đột thì có thể được đi tại cùng 1 thời điểm.



Hình 1.1 Nút giao thông

Chúng ta có thể mô hình hóa vấn đề này bằng một cấu trúc toán học gọi là đồ thị (sẽ được trình bày chi tiết ở chương 5). Đồ thị là một cấu trúc bao gồm 1 tập các điểm gọi là đỉnh và một tập các đường nối các điểm, gọi là các cạnh. Vấn đề nút giao

thông có thể được mô hình hóa bằng một đồ thị, trong đó các ngã rẽ là các đỉnh, và có một cạnh nối 2 đỉnh biểu thị rằng 2 ngã rẽ đó không thể đi đồng thời. Khi đó, đồ thị của nút giao thông ở hình 1.1 có thể được biểu diễn như ở hình 1.2.



Hình 1.2 Đồ thị ngã rẽ

Ta có thể sử dụng đồ thị trên để giải quyết vấn đề thiết kế hệ thống đèn cho nút giao thông như đã nói.

Việc tô màu một đồ thị là việc gán cho mỗi đỉnh của đồ thị một màu sao cho không có hai đỉnh được nối bởi 1 cạnh nào đó lại có cùng một màu. Để thấy rằng vấn đề nút giao thông có thể được chuyển thành bài toán tô màu đồ thị các ngã rẽ ở trên sao cho phải sử dụng số màu ít nhất.

Bài toán tô màu đồ thị là bài toán đã xuất hiện và được nghiên cứu từ rất lâu. Tuy nhiên, để tô màu một đồ thị bất kỳ với số màu ít nhất là bài toán rất phức tạp. Để giải bài toán này, người ta thường sử dụng phương pháp “vét cạn” để thử tất cả các khả năng có thể. Có nghĩa, đầu tiên thử tiến hành tô màu đồ thị bằng 1 màu, tiếp theo dùng 2 màu, 3 màu, v.v. cho tới khi tìm ra phương pháp tô màu thỏa mãn yêu cầu.

Phương pháp vét cạn có vẻ thích hợp với các đồ thị nhỏ, tuy nhiên đối với các đồ thị phức tạp thì sẽ tiêu tốn rất nhiều thời gian thực hiện cũng như tài nguyên hệ thống. Ta có thể tiếp cận vấn đề theo hướng cố gắng tìm ra một giải pháp đủ tốt, không nhất thiết phải là giải pháp tối ưu. Chẳng hạn, ta sẽ cố gắng tìm một giải pháp tô màu cho đồ thị ngã rẽ ở trên với một số màu khá ít, gần với số màu ít nhất, và thời gian thực hiện việc tìm giải pháp là khá nhanh. Giải thuật tìm các giải pháp đủ tốt nhưng chưa phải tối ưu như vậy gọi là các giải thuật tìm theo “cảm tính”.

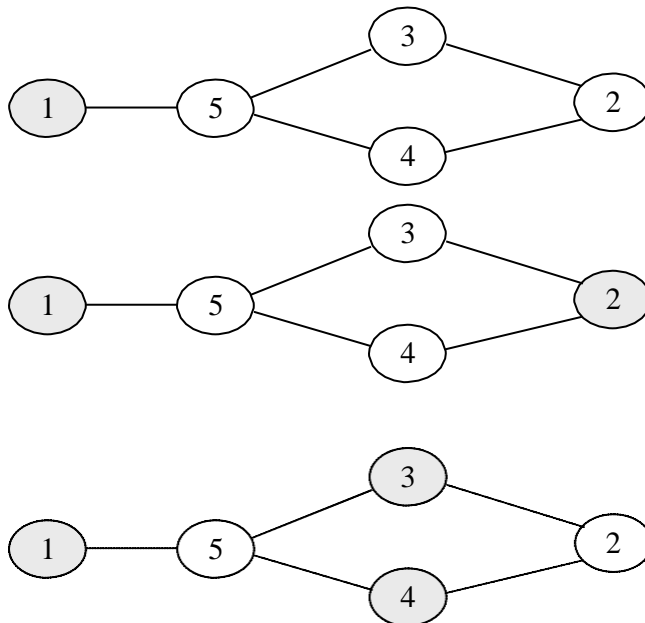
Đối với bài toán tô màu đồ thị, một thuật toán cảm tính thường được sử dụng là thuật toán “tham ăn” (greedy). Theo thuật toán này, đầu tiên ta sử dụng một màu để tô nhiều nhất số đỉnh có thể, thỏa mãn yêu cầu bài toán. Tiếp theo, sử dụng màu thứ 2 để tô các đỉnh chưa được tô trong bước 1, rồi sử dụng đến màu thứ 3 để tô các đỉnh chưa được tô trong bước 2, v.v.

Để tô màu các đỉnh với màu mới, chúng ta thực hiện các bước:

- Lựa chọn 1 đỉnh chưa được tô màu và tô nó bằng màu mới.
- Duyệt qua các đỉnh chưa được tô màu. Với mỗi đỉnh dạng này, kiểm tra xem có cạnh nào nối nó với một đỉnh vừa được tô bởi màu mới hay không. Nếu không có cạnh nào thì ta tô màu đỉnh này bằng màu mới.

Thuật toán này được gọi là “tham ăn” vì tại mỗi bước nó tô màu tất cả các đỉnh có thể mà không cần phải xem xét xem việc tô màu đó có để lại những điểm bất lợi cho các bước sau hay không. Trong nhiều trường hợp, chúng ta có thể tô màu được nhiều đỉnh hơn bằng 1 màu nếu chúng ta bớt “tham ăn” và bỏ qua một số đỉnh có thể tô màu được trong bước trước.

Ví dụ, xem xét đồ thị ở hình 1.3, trong đó đỉnh 1 đã được tô màu đỏ. Ta thấy rằng hoàn toàn có thể tô cả 2 đỉnh 3 và 4 là màu đỏ, với điều kiện ta không tô đỉnh số 2 màu đỏ. Tuy nhiên, nếu ta áp dụng thuật toán tham ăn theo thứ tự các đỉnh lớn dần thì đỉnh 1 và đỉnh 2 sẽ là màu đỏ, và khi đó đỉnh 3, 4 sẽ không được tô màu đỏ.



a) Đồ thị ban đầu

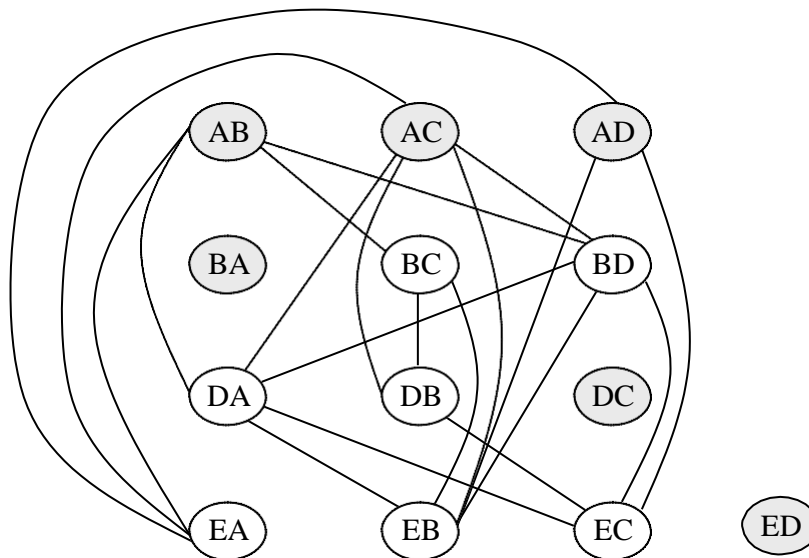
b) Tô màu theo thuật toán tham ăn

c) Một cách tô màu tốt hơn

Hình 1.3 Đồ thị

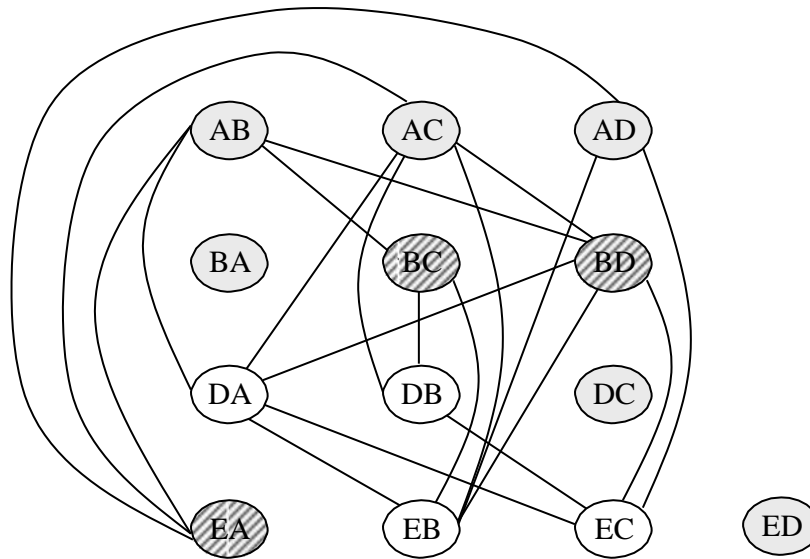
Bây giờ ta sẽ xem xét thuật toán tham ăn được áp dụng trên đồ thị các ngã rẽ ở hình 1.2 như thế nào. Giả sử ta bắt đầu từ đỉnh AB và tô cho đỉnh này màu xanh. Khi đó,

ta có thể tô cho đỉnh AC màu xanh vì không có cạnh nối đỉnh này với AB. AD cũng có thể tô màu xanh vì không có cạnh nối AD với AB, AC. Đỉnh BA không có cạnh nối tới AB, AC, AD nên cũng có thể được tô màu xanh. Tuy nhiên, đỉnh BC không tô được màu xanh vì tồn tại cạnh nối BC và AB. Tương tự như vậy, BD, DA, DB không thể tô màu xanh vì tồn tại cạnh nối chúng tới một trong các đỉnh đã tô màu xanh. Cạnh DC thì có thể tô màu xanh. Cuối cùng, cạnh EA, EB, EC cũng không thể tô màu xanh trong khi ED có thể được tô màu xanh.



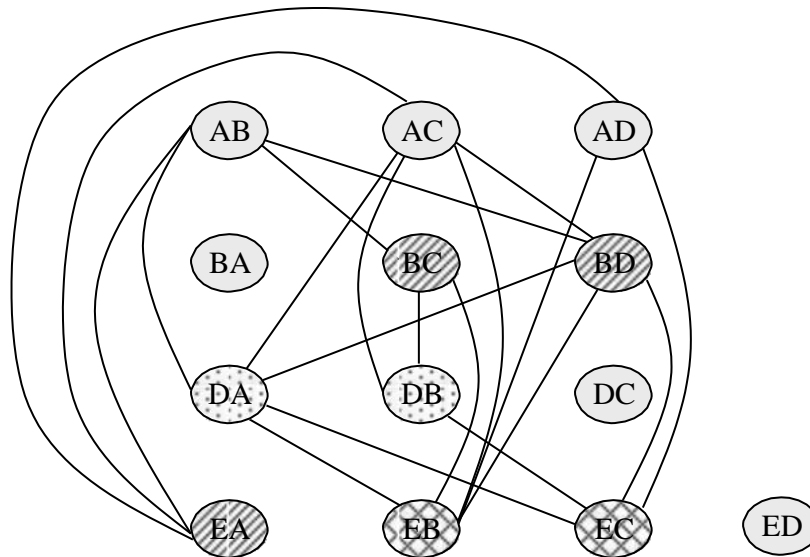
Hình 1.4 Tô màu xanh cho các đỉnh của đồ thị ngã rẽ

Tiếp theo, ta sử dụng màu đỏ để tô các đỉnh chưa được tô màu ở bước trước. Đầu tiên là BC. BD cũng có thể tô màu đỏ, tuy nhiên do tồn tại cạnh nối DA với BD nên DA không được tô màu đỏ. Tương tự như vậy, DB không tô được màu đỏ còn EA có thể tô màu đỏ. Các đỉnh chưa được tô màu còn lại đều có cạnh nối tới các đỉnh đã tô màu đỏ nên cũng không được tô màu.







Hình 1.5 Tô màu đỏ trong bước 2

Bước 3, các đỉnh chưa được tô màu còn lại là DA, DB, EB, EC. Nếu ta tô màu đỉnh DA là màu lục thì DB cũng có thể tô màu lục. Khi đó, EB, EC không thể tô màu lục và ta chọn 1 màu thứ tư là màu vàng cho 2 đỉnh này.



Hình 1.6 Tô màu lục và màu vàng cho các đỉnh còn lại

Như vậy, ta có thể dùng 4 màu xanh, đỏ, lục, vàng để tô màu cho đồ thị ngã rẽ ở hình 1.2 theo yêu cầu như đã nói ở trên. Bảng tổng hợp màu được mô tả như sau:

Màu	Ngã rẽ
Xanh 	AB, AC, AD, BA, DC,
Đỏ 	BC, BD, EA
Lục 	DA, DB
Vàng 	EB, EC

Bảng 1.2 Bảng tổng hợp màu

Thuật toán tham ăn không đảm bảo cho ra kết quả tối ưu là số màu ít nhất được dùng. Tuy nhiên, người ta có thể dùng một số tính chất của đồ thị để đánh giá kết quả thu được.

Trở lại với vấn đề nút giao thông, từ kết quả tô màu trên, ta có thể thiết kế hệ thống đèn giao thông theo bảng tổng hợp màu trên, trong đó mỗi trạng thái của hệ thống đèn tương ứng với 1 màu. Tại mỗi trạng thái, các ngã rẽ nằm tại hàng tương ứng với màu đó được cho phép đi, các ngã rẽ còn lại bị cấm.

1.1.2 Ngôn ngữ diễn đạt giải thuật và kỹ thuật tinh chỉnh từng bước

Sau khi đã xây dựng được mô hình toán học cho vấn đề cần giải quyết, tiếp theo, ta có thể hình thành một thuật toán cho mô hình đó. Phiên bản đầu tiên của thuật toán thường được diễn tả dưới dạng các phát biểu tương đối tổng quát, và sau đó sẽ được tinh chỉnh dần từng bước thành chuỗi các lệnh cụ thể, rõ ràng hơn. Ví dụ trong thuật toán tham ăn ở trên, ta mô tả bước thực hiện ở mức tổng quát là “Lựa chọn 1 đỉnh chưa được tô màu”. Với phát biểu như vậy, ta hy vọng rằng người đọc có thể nắm được ý tưởng thực hiện thao tác. Tuy nhiên, để chuyển các phát biểu đó thành chương trình máy tính, cần phải qua 1 số bước tinh chỉnh cho tới khi đạt đến mức các phát biểu đều có thể được chuyển đổi trực tiếp sang các lệnh của ngôn ngữ lập trình.

Trở lại ví dụ về bài toán tô màu đồ thị bằng thuật toán tham ăn. Ta sẽ xem xét việc mô tả thuật toán từ mức tổng quát cho tới một số mức cụ thể hơn. Tại bước nào đó, giả sử ta có đồ thị G có 1 số đỉnh đã được tô màu theo quy tắc đã nói ở trên. Thủ tục *Tham_an* dưới đây sẽ xác định 1 tập các đỉnh chưa được tô màu thuộc G mà có thể cùng được tô bởi 1 màu mới. Thủ tục này sẽ được gọi đi gọi lại nhiều lần cho tới khi tất cả các đỉnh của G đã được tô màu. Ở mức tổng quát, thủ tục được mô tả như sau:

```

void Tham_an (GRAPH: G, SET: Mau_moi)
{
    Mau_moi = Tập rỗng;
    For mỗi đỉnh v chưa được tô màu thuộc G
        If v không được nối tới đỉnh nào trong tập
        Mau_moi
        {
            Tô màu mới cho
            đỉnh v; Đưa v vào
            tập Mau_moi;
        }
}

```

Trong thủ tục trên, ta sử dụng một ngôn ngữ diễn đạt giải thuật tựa như ngôn ngữ lập trình C. Trong ngôn ngữ này, các lệnh được mô tả dưới dạng ngôn ngữ tự nhiên nhưng vẫn tuân theo cú pháp của ngôn ngữ lập trình.

Ta nhận thấy rằng các phát biểu trong thủ tục trên còn rất tổng quát, và chưa tương ứng với các lệnh trong ngôn ngữ lập trình, chẳng hạn các điều kiện kiểm tra trong câu lệnh For và If ở mức mô tả hiện tại là không thực hiện được trong C. Để thủ tục có thể thực thi được, ta cần phải tinh chỉnh một số bước để có thể chuyển đổi về chương trình trong ngôn ngữ lập trình C thông thường.

Đầu tiên, ta xem xét lệnh If ở trên. Để kiểm tra xem đỉnh v có nối tới một đỉnh nào đó trong tập Mau_moi hay không, ta xem xét từng đỉnh w trong Mau_moi và sử dụng đồ thị G để kiểm tra xem có tồn tại cạnh nối v và w không. Để lưu giữ kết quả kiểm tra, ta sử dụng một biến ton_tai. Khi đó, thủ tục được tinh chỉnh như sau:

```

void Tham_an (GRAPH: G, SET: Mau_moi)
{
    int ton_tai;
    Mau_moi = Tập
    rỗng;
    For mỗi đỉnh v chưa được tô màu thuộc G
    {
        ton_tai = 0;
        For mỗi đỉnh w thuộc Mau_moi
            If tồn tại cạnh nối v và w
            trong G ton_tai = 1;
    }
}

```



```

        If ton_tai == 1
        {
            Tô màu mới cho đỉnh
            v; Đưa v vào tập
            Mau_moi;
        }
    }
}

```

Như vậy, ta có thể thấy rằng điều kiện kiểm tra trong phát biểu If đã được mô tả cụ thể hơn bằng các phát nhỏ hơn, và các phát biểu này có thể dễ dàng chuyển thành các lệnh cụ thể trong C. Tiếp theo, ta sẽ tinh chỉnh các vòng lặp For để duyệt qua các đỉnh thuộc G và thuộc Mau_moi. Để làm điều này, tốt nhất là ta thay For bằng một vòng lặp While, biến v ban đầu được gán là phần tử đầu tiên chưa tô màu trong tập G, và tại mỗi bước lặp, biến v sẽ được thay bằng phần tử chưa tô màu tiếp theo trong G. Vòng lặp F bên trong có thể thực hiện tương tự.

```

Void Tham_an (GRAPH: G, SET: Mau_moi)
{
    int
    ton_tai;
    int v, w
    Mau_moi = Tập rỗng;
    v = đỉnh chưa tô màu đầu tiên
    trong G ; While v != NULL
    {
        ton_tai = 0;
        w = đỉnh đầu tiên trong
        Mau_moi; While w != NULL{
            If tồn tại cạnh nối v và w
            trong G ton_tai = 1;
            w = đỉnh tiếp theo trong Mau_moi ;
        }
        If ton_tai == 1
        {
            Tô màu mới cho đỉnh
            v; Đưa v vào tập
            Mau_moi;
        }
    }
}

```

```

    }
    v = đỉnh chưa tô màu tiếp theo trong G;
  }
}

```

Như vậy, ta thấy các phát biểu trong thủ tục đã khá cụ thể, tuy nhiên, để chuyển đổi thành chương trình trong ngôn ngữ C thì cần tới vài bước tinh chỉnh nữa. Bạn đọc hãy xem như đó là bài tập và tự giải để hiểu rõ về ngôn ngữ diễn đạt giải thuật cũng như kỹ thuật tinh chỉnh từng bước.

1.2 PHÂN TÍCH THUẬT TOÁN

Với mỗi vấn đề cần giải quyết, ta có thể tìm ra nhiều thuật toán khác nhau. Có những thuật toán thiết kế đơn giản, dễ hiểu, dễ lập trình và sửa lỗi, tuy nhiên thời gian thực hiện lớn và tiêu tốn nhiều tài nguyên máy tính. Ngược lại, có những thuật toán thiết kế và lập trình rất phức tạp, nhưng cho thời gian chạy nhanh hơn, sử dụng tài nguyên máy tính hiệu quả hơn. Khi đó, câu hỏi đặt ra là ta nên lựa chọn giải thuật nào để thực hiện?

Đối với những chương trình chỉ được thực hiện một vài lần thì thời gian chạy không phải là tiêu chí quan trọng nhất. Đối với bài toán kiểu này, thời gian để lập trình viên xây dựng và hoàn thiện thuật toán đáng giá hơn thời gian chạy của chương trình và như vậy những giải thuật đơn giản về mặt thiết kế và xây dựng nên được lựa chọn.

Đối với những chương trình được thực hiện nhiều lần thì thời gian chạy của chương trình đáng giá hơn rất nhiều so với thời gian được sử dụng để thiết kế và xây dựng nó. Khi đó, lựa chọn một giải thuật có thời gian chạy nhanh hơn (cho dù việc thiết kế và xây dựng phức tạp hơn) là một lựa chọn cần thiết. Trong thực tế, trong giai đoạn đầu của việc giải quyết vấn đề, một giải thuật đơn giản thường được thực hiện trước như là 1 nguyên mẫu (prototype), sau đó nó sẽ được phân tích, đánh giá, và cải tiến thành các phiên bản tốt hơn.

1.2.1 Ước lượng thời gian thực hiện chương trình

Thời gian chạy của 1 chương trình phụ thuộc vào các yếu tố sau:

- Dữ liệu đầu vào
- Chất lượng của mã máy được tạo ra bởi chương trình dịch
- Tốc độ thực thi lệnh của máy
- Độ phức tạp về thời gian của thuật toán

Thông thường, thời gian chạy của chương trình không phụ thuộc vào giá trị dữ liệu đầu vào mà phụ thuộc vào kích thước của dữ liệu đầu vào. Do vậy thời gian chạy của chương trình nên được định nghĩa như là một hàm có tham số là kích thước dữ liệu đầu vào. Giả sử T là hàm ước lượng thời gian chạy của chương trình, khi đó với dữ liệu đầu

vào có kích thước n thì thời gian chạy của chương trình là $T(n)$. Ví dụ, đối với một số chương trình thì thời gian chạy là an hoặc an^2 , trong đó a là hằng số. Đơn vị của hàm $T(n)$ là không xác định, tuy nhiên ta có thể xem như $T(n)$ là tổng số lệnh được thực hiện trên 1 máy tính lý tưởng.

Trong nhiều chương trình, thời gian thực hiện không chỉ phụ thuộc vào kích thước dữ liệu vào mà còn phụ thuộc vào tính chất của nó. Khi tính chất dữ liệu vào thoả mãn một số đặc điểm nào đó thì thời gian thực hiện chương trình có thể là lớn nhất hoặc nhỏ nhất. Khi đó, ta định nghĩa thời gian thực hiện chương trình $T(n)$ trong trường hợp xấu nhất hoặc tốt nhất. Đó là thời gian thực hiện lớn nhất hoặc nhỏ nhất trong tất cả các bộ dữ liệu vào có kích thước n . Ta cũng định nghĩa thời gian thực hiện trung bình của chương trình trên mọi bộ dữ liệu vào kích thước n . Trong thực tế, ước lượng thời gian thực hiện trung bình khó hơn nhiều so với thời gian thực hiện trong trường hợp xấu nhất hoặc tốt nhất, bởi vì việc phân tích thuật toán trong trường hợp trung bình khó hơn về mặt toán học, đồng thời khái niệm “trung bình” không có ý nghĩa thực sự rõ ràng.

Yếu tố chất lượng của mã máy được tạo bởi chương trình dịch và tốc độ thực thi lệnh của máy cũng ảnh hưởng tới thời gian thực hiện chương trình cho thấy chúng ta không thể thể hiện thời gian thực hiện chương trình dưới đơn vị thời gian chuẩn, chẳng hạn phút hoặc giây. Thay vào đó, ta có thể phát biểu thời gian thực hiện chương trình tỷ lệ với n hoặc n^2 v.v. Hệ số của tỷ lệ là 1 hằng số chưa xác định, phụ thuộc vào máy tính, chương trình dịch, và các nhân tố khác.

Ký hiệu $O(n)$

Để biểu thị cấp độ tăng của hàm, ta sử dụng ký hiệu $O(n)$. Ví dụ, ta nói thời gian thực hiện $T(n)$ của chương trình là $O(n^2)$, có nghĩa là tồn tại các hằng số dương c và n_0 sao cho $T(n) \leq cn^2$ với $n \geq n_0$.

Ví dụ, xét hàm $T(n) = (n+1)^2$. Ta có thể thấy $T(n)$ là $O(n^2)$ với $n_0 = 1$ và $c = 4$, vì ta có $T(n) = (n+1)^2 < 4n^2$ với mọi $n \geq 1$. Trong ví dụ này, ta cũng có thể nói rằng $T(n)$ là $O(n^3)$, tuy nhiên, phát biểu này “yếu” hơn phát biểu $T(n)$ là $O(n^2)$.

Nhìn chung, ta nói $T(n)$ là $O(f(n))$ nếu tồn tại các hằng số dương c và n_0 sao cho $T(n) < c.f(n)$ với $n \geq n_0$. Một chương trình có thời gian thực hiện là $O(f(n))$ thì được xem là có cấp độ tăng $f(n)$.

Việc đánh giá các chương trình có thể được thực hiện qua việc đánh giá các hàm thời gian chạy của chương trình, bỏ qua các hằng số tỷ lệ. Với giả thiết này, một chương trình với thời gian thực hiện là $O(n^2)$ sẽ tốt hơn chương trình với thời gian chạy $O(n^3)$. Bên cạnh các yếu tố hằng số xuất phát từ chương trình dịch và máy, còn có thêm hằng số từ bản thân chương trình. Ví dụ, trên cùng một chương trình dịch và cùng 1 máy,

chương trình đầu tiên có thời gian thực hiện là $100n^2$, trong khi chương trình thứ 2 có thời gian thực hiện là $5n^3$. Với n nhỏ, có thể $5n^3$ nhỏ hơn $100n^2$, tuy nhiên với n đủ lớn thì $5n^3$ sẽ lớn hơn $100n^2$ đáng kể.

Một lý do nữa để xem xét cấp độ tăng về thời gian thực hiện của chương trình là nó cho phép ta xác định độ lớn của bài toán mà ta có thể giải quyết. Mặc dù máy tính có tốc độ ngày càng cao, tuy nhiên, với những chương trình có thời gian thực hiện có cấp độ tăng lớn (từ n^2 trở lên), thì việc tăng tốc độ của máy tính tạo ra sự khác biệt không đáng kể về kích thước bài toán có thể xử lý bởi máy tính trong một khoảng thời gian cố định.

1.2.2 Tính toán thời gian thực hiện chương trình

Để tính toán được thời gian thực hiện chương trình, ta cần chú ý một số nguyên tắc cộng và nhân cấp độ tăng của hàm như sau :

Giả sử $T1(n)$ và $T2(n)$ là thời gian chạy của 2 đoạn chương trình P1 và P2, trong đó $T1(n)$ là $O(f(n))$ và $T2(n)$ là $O(g(n))$. Khi đó, thời gian thực hiện của 2 đoạn chương trình nối tiếp P1, P2 là $O(\max(f(n), g(n)))$.

Nguyên tắc cộng trên có thể sử dụng để tính thời gian thực hiện của chương trình bao gồm 1 số tuần tự các bước, mỗi bước có thể là 1 đoạn chương trình bao gồm 1 số vòng lặp và rẽ nhánh. Ví dụ, giả sử ta có 3 bước thực hiện chương trình lần lượt có thời gian chạy là $O(n^2)$, $O(n^3)$, $O(n \log n)$. Khi đó, thời gian chạy của 2 đoạn chương trình đầu là $O(\max(n^2, n^3)) = O(n^3)$, còn thời gian chạy của cả 3 đoạn chương trình là $O(\max(n^3, n \log n)) = O(n^3)$.

Nguyên tắc nhân cấp độ tăng của hàm như sau: Với giả thiết về $T1(n)$ và $T2(n)$ như trên, nếu 2 đoạn chương trình P1 và P2 không được thực hiện tuần tự mà lồng nhau thì thời gian chạy tổng thể sẽ là $T1(n).T2(n) = O(f(n).(g(n)))$.

Để minh họa cho việc phân tích và tính toán thời gian thực hiện của 1 chương trình, ta sẽ xem xét một thuật toán đơn giản để sắp xếp các phần tử của một tập hợp, đó là thuật toán sắp xếp nổi bọt (bubble sort).

Thuật toán như sau :

```
void bubble (int a[n]){  
    int i, j, temp;
```

```

1.   for (i = 0; i < n-1; i++)
2.       for (j = n-1; j >= i+1; j--)
3.           if (a[j-1] > a[j]){
4.               // Đổi chỗ cho a[j] và a[j-1]
5.               temp = a[j-1];
6.               a[j-1] = a[j];
7.               a[j] = temp;
8.           }
9.   }

```

Trong thuật toán này, mỗi lần duyệt của vòng lặp trong (biến duyệt j) sẽ làm “nổi” lên trên phần tử nhỏ nhất trong các phần tử được duyệt.

Dễ thấy rằng kích thước dữ liệu vào chính là số phần tử được sắp, n. Mỗi lệnh gán sẽ có thời gian thực hiện cố định, không phụ thuộc vào n, do vậy, các lệnh 4, 5, 6 sẽ có thời gian thực hiện là $O(1)$, tức thời gian thực hiện là hằng số. Theo quy tắc cộng cấp độ tăng thì tổng thời gian thực hiện cả 3 lệnh là $O(\max(1, 1, 1)) = O(1)$.

Tiếp theo ta sẽ xem xét thời gian thực hiện của các lệnh lặp và rẽ nhánh. Lệnh If kiểm tra điều kiện để thực hiện nhóm lệnh gán 4, 5, 6. Việc kiểm tra điều kiện sẽ có thời gian thực hiện là $O(1)$. Ngoài ra, chúng ta chưa biết được là điều kiện có thỏa mãn hay không, tức là không biết được nhóm lệnh gán có được thực hiện hay không. Do vậy, ta giả thiết trường hợp xấu nhất là tất cả các lần kiểm tra điều kiện đều thỏa mãn, và các lệnh gán được thực hiện. Như vậy, toàn bộ lệnh If sẽ có thời gian thực hiện là $O(1)$.

Tiếp tục xét từ trong ra ngoài, ta xét đến vòng lặp trong (biến duyệt j). Trong vòng lặp này, tại mỗi bước lặp ta cần thực hiện các thao tác như kiểm tra đã gặp điều kiện dừng chưa và tăng biến duyệt lên 1 nếu chưa dừng. Như vậy, mỗi bước lặp có thời gian thực hiện là $O(1)$. Số bước lặp là $n-i$, do đó theo quy tắc nhân cấp độ tăng thì tổng thời gian thực hiện của vòng lặp này là $O((n-i) \times 1) = O(n-i)$.

Cuối cùng, ta xét vòng lặp ngoài cùng (biến duyệt i). Vòng lặp này được thực hiện $(n-1)$ lần, do đó, tổng thời gian thực hiện của chương trình là:

$$\sum (n-i) = n(n-1)/2 = n^2/2 - n/2 = O(n^2)$$

Như vậy, thời gian thực hiện giải thuật sắp xếp nổi bọt là tỷ lệ với n^2 .

Một số quy tắc chung trong việc phân tích và tính toán thời gian thực hiện chương trình

- Thời gian thực hiện các lệnh gán, đọc, ghi .v.v, luôn là $O(1)$.
- Thời gian thực hiện chuỗi tuần tự các lệnh được xác định theo quy tắc cộng cấp độ tăng. Có nghĩa là thời gian thực hiện của cả nhóm lệnh tuần tự được

tính là thời gian thực hiện của lệnh lớn nhất.

- Thời gian thực hiện lệnh rẽ nhánh If được tính bằng thời gian thực hiện các lệnh khi điều kiện kiểm tra được thoả mãn và thời gian thực hiện việc kiểm tra điều kiện. Thời gian thực hiện việc kiểm tra điều kiện luôn là $O(1)$.
- Thời gian thực hiện 1 vòng lặp được tính là tổng thời gian thực hiện các lệnh ở thân vòng lặp qua tất cả các bước lặp và thời gian để kiểm tra điều kiện dừng (thường là $O(1)$). Thời gian thực hiện này thường được tính theo quy tắc nhân cấp độ tăng số lần thực hiện bước lặp và thời gian thực hiện các lệnh ở thân vòng lặp. Các vòng lặp phải được tính thời gian thực hiện một cách riêng rẽ.

1.3 TÓM TẮT CHƯƠNG 1

Các kiến thức cần nhớ trong chương 1:

- Thuật toán là một chuỗi hữu hạn các lệnh. Mỗi lệnh có một ngữ nghĩa rõ ràng và có thể được thực hiện với một lượng hữu hạn tài nguyên trong một khoảng hữu hạn thời gian.
- Thuật toán thường được mô tả bằng các ngôn ngữ diễn đạt giải thuật gần với ngôn ngữ tự nhiên. Các mô tả này sẽ được tinh chỉnh dần dần để đạt tới mức ngôn ngữ lập trình.
- Thời gian thực hiện thuật toán thường được coi như là 1 hàm của kích thước dữ liệu đầu vào.
- Thời gian thực hiện thuật toán thường được tính trong các trường hợp tốt nhất, xấu nhất, hoặc trung bình.
- Để biểu thị cấp độ tăng của hàm, ta sử dụng ký hiệu $O(n)$. Ví dụ, ta nói thời gian thực hiện $T(n)$ của chương trình là $O(n^2)$, có nghĩa là tồn tại các hằng số dương c và n_0 sao cho $T(n) \leq cn^2$ với $n \geq n_0$.
- Cấp độ tăng về thời gian thực hiện của chương trình cho phép ta xác định độ lớn của bài toán mà ta có thể giải quyết.
- Quy tắc cộng cấp độ tăng: Giả sử $T_1(n)$ và $T_2(n)$ là thời gian chạy của 2 đoạn chương trình P_1 và P_2 , trong đó $T_1(n)$ là $O(f(n))$ và $T_2(n)$ là $O(g(n))$. Khi đó, thời gian thực hiện của 2 đoạn chương trình nối tiếp P_1, P_2 là $O(\max(f(n), g(n)))$.
- Quy tắc nhân cấp độ tăng: Với giả thiết về $T_1(n)$ và $T_2(n)$ như trên, nếu 2 đoạn chương trình P_1 và P_2 không được thực hiện tuần tự mà lồng nhau thì thời gian chạy tổng thể sẽ là $T_1(n).T_2(n) = O(f(n).(g(n)))$.

1.4 CÂU HỎI VÀ BÀI TẬP

1. Trình bày khái niệm thuật toán? Các đặc điểm của thuật toán?

2. Trong một giải vô địch bóng đá có 6 đội bóng đá vòng tròn. Các đội là A, B, C, D, E, F. Đội A đã đá với B và C. Đội B đã đá với D và F. Đội E đã đá với C và F. Mỗi đội đá mỗi tuần 1 trận. Hãy lập 1 lịch cho các đội bóng sao cho tất cả các đội đều đá đủ số trận quy định trong 1 số tuần vừa phải. Thực hiện phân tích, thiết kế thuật toán. Biểu diễn thuật toán bằng ngôn ngữ diễn đạt giải thuật, sau đó tinh chỉnh về dạng ngôn ngữ lập trình C.
3. Thời gian thực hiện một chương trình thường phụ thuộc vào các yếu tố nào? Phân tích cụ thể từng yếu tố.
4. Nói thời gian thực hiện chương trình là $T(n) = O(f(n))$ có nghĩa là gì? Cho ví dụ minh họa.
5. Nêu quy tắc cộng và nhân cấp độ tăng của hàm. Có ví dụ minh họa.
6. Nêu các quy tắc chung trong việc phân tích và đánh giá thời gian thực hiện chương trình.