Tên bài giảng
Tổng quan

Môn học:	**Thuật toán và ứng dụng**
Chương:	1
Hệ:	Đại học
Giảng viên:	TS. Phạm Đình Phong
Email:	phongpd@utc.edu.vn
SĐT:	0972481813

Nội dung bài học

1. **Giới thiệu môn học**
2. **Một số mô hình lập trình**
3. **Phân tích thuật toán**

Giới thiệu môn học

- Mục tiêu
  - Kiến thức
    - **Giải thích** và **áp dụng** một số mô hình lập trình cơ bản, các phương pháp phân tích thuật toán để phân tích và lựa chọn thuật toán
    - **Giải thích** và **áp dụng** được các cấu trúc dữ liệu nâng cao như cấu trúc dữ liệu tập hợp, cấu trúc dữ liệu từ điển, hàng đợi ưu tiên, cấu trúc dữ liệu đống, bảng băm, cấu trúc cây
    - **Hiểu** và **áp dụng** hiệu quả các thuật toán sắp xếp, tìm kiếm, các thuật toán trên đồ thị, xử lý xâu ký tự vào phát triển các ứng dụng
    - **Hiểu** và **áp dụng** các thuật toán thực tế trong một số lĩnh vực như đồ thị, xử lý văn bản, xử lý ảnh, …

Giới thiệu môn học

- Mục tiêu
  - Kỹ năng
    - **Phân tích**, **khảo sát** bài toán và áp dụng các cấu trúc dữ liệu phù hợp vào các thuật toán, lập trình giải các bài toán thực tế. Kiểm tra tính đúng đắn và hiệu quả của các thuật toán được áp dụng.
    - **Áp dụng** kỹ năng làm việc nhóm trong thảo luận các nội dung môn học và giải bài tập trên lớp. Thuyết trình nội dung bài tập, bài tập lớn và đóng góp ý kiến cho nội dung thuyết trình của các nhóm và các bạn trong lớp.
    - **Lập kế hoạch** hoàn thành các bài tập theo từng nội dung bài học có sự trao đổi với các bạn trong lớp.

Giới thiệu môn học

- Mục tiêu
  - Thái độ
    - Ngày càng yêu thích say mê với thuật toán
    - Học tập tích cực, sáng tạo
    - Có ý thức vận dụng các kiến thức được học vào giải quyết các bài toán thực tế
  - Nhận thức
    - Nhận thức được vai trò của các cấu trúc dữ liệu và thuật toán trong giải quyết các bài toán thực tế

Giới thiệu môn học

- Nội dung
  - Hướng người học áp dụng những thuật toán vào giải các bài toán thực tế sử dụng
    - Các cấu trúc dữ liệu phức tạp
    - Các thuật toán phức hợp
  - Đi sâu giải quyết các bài toán trên đồ thị (Chương 4) hoặc các bài toán về xâu ký tự (Chương 5)

Giới thiệu môn học

- Các học phần tiên quyết
  - Cấu trúc dữ liệu và giải thuật
  - Phân tích thiết kế thuật toán

Giới thiệu môn học

- Học liệu
  - Slide bài giảng điện tử
  - Tài liệu tham khảo
    - Algorithms - Fourth edition, Robert Sedgewick, Kenvin Wayne, Princeton University
    - Introduction to Algorithms – Third edition, Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein, The MIT Press

Giới thiệu môn học

- Phân bổ giờ tín chỉ

Các hoạt động	Số giờ
Lý thuyết	30
Bài tập	15
Thực hành	15
Tự học	90

Giới thiệu môn học

- Đánh giá kết quả học tập
  - Kiểm tra thường xuyên: 10%
    - Đi học đầy đủ, đúng giờ: 5%
    - Tích cực trên lớp: 5%
  - Đánh giá định kỳ: 40%
    - Kiểm tra giữa kỳ: 10%
    - Thảo luận, thực hành: 30%
  - Thi kết thúc học phần
    - Thi viết
    - Tỷ trọng: 50%

Một số mô hình lập trình

- Mô hình lập trình là gì?
  - Không có định nghĩa chính thức
  - Thuật ngữ này được sử dụng để nói đến một phong cách lập trình (style of programming)
    - Không dùng cho một ngôn ngữ lập trình cụ thể
    - Đề cập đến cách bạn lập trình, cách bạn giải quyết vấn đề
    - Ví dụ: Có nhiều ngôn ngữ lập trình như Basic, C, C++, Java, C#, F#, Python, JavaScript, … nhưng tất cả chúng khi xây dựng đều phải chọn lựa cho mình một hoặc vài chiến lược và chiến lược đó là mô hình lập trình
    - Mô hình lập trình là cách phân loại các ngôn ngữ lập trình (NNLT) dựa vào đặc điểm của chúng. Một NTTLT có thể được phân loại vào nhiều mô hình lập trình

Một số mô hình lập trình

- Có bao nhiêu mô hình lập trình?
  - Gồm hai nhóm chính sau:

Lập trình mệnh lệnh (Imperative programming) Mô hình lập trình sử dụng các câu lệnh để thay đổi trạng thái của chương trình	**Lập trình khai báo (Declarative programming)** Khai báo các thuộc tính của kết quả mong muốn nhưng không thực hiện tính toán nó
1. Lập trình hướng thủ tục/cấu trúc	1. Lập trình hàm (functional programming)
2. Lập trình hướng đối tượng	2. Lập trình logic
3. Lập trình song song	3. Lập trình hướng dữ liệu

Một số mô hình lập trình

- Mô hình lập trình mệnh lệnh
  - Imperative Programming: telling the “machine” how to do something, and as a result what you want to happen will happen
  - Lập trình hướng thủ tục liên quan đến việc viết ra một danh sách các chỉ dẫn để nói với máy tính những gì cần làm theo từng bước để hoàn thành nhiệm vụ
  - Nói nôm na: Lập trình mệnh lệnh là việc bảo máy tính hãy làm việc A, B, C ... X, Y, Z theo thứ tự nhất định thì máy sẽ làm đúng như thế và chỉ kiểm tra lại kết quả xem có đúng không

Một số mô hình lập trình

- Lập trình hướng thủ tục/cấu trúc
  - Thường là mô hình lập trình đầu tiên mà người lập trình viên mới sẽ học
  - Mã chương trình trực tiếp chỉ dẫn thực hiện hoàn thành một công việc theo các bước hợp lý
  - Sử dụng cách tiếp cận tuyến tính từ trên xuống, coi dữ liệu và thủ tục là hai thực thể khác nhau
  - Chia chương trình thành các chương trình con được gọi là các thủ tục và hàm → chỉ đơn giản chứa một loạt các bước sẽ được thực hiện

Một số mô hình lập trình

- Lập trình hướng thủ tục/cấu trúc
  - Một số khái niệm
    - **Kiểu dữ liệu:** byte, integer, real, string, …
    - **Biến cục bộ**: chỉ có hiệu lực trong phạm vi chương trình con, thủ tục và hàm
    - **Biến toàn cục**: các thủ tục/hàm có thể sử dụng
    - **Biểu thức**: tính toán → thứ tự ưu tiên các toán tử
    - **Truyền tham số**: theo tham trị và tham biến
    - **Câu lệnh**: gán, điều khiển, …
    - **Mô-đun**: chia chương trình thành nhiều mô-đun

Một số mô hình lập trình

- Lập trình hướng thủ tục/cấu trúc
  - Sử dụng ba cấu trúc chính tạo chương trình
    - **Tuần tự**: Các câu lệnh được thực hiện theo trình tự nhất định (thường là từ trên xuống)
    - **Quyết định**: Chương trình chạy phụ thuộc vào điều kiện nhất định nào đó (điều khiển bởi các từ khóa **if**, **switch**, …)
    - **Lặp**: Các mệnh lệnh được lặp đi lặp lại khi điều kiện nào đó vẫn được thoả mãn (**for**, **while**, …)

Một số mô hình lập trình

- Lập trình hướng thủ tục/cấu trúc
  - Phương pháp phân tích từ trên xuống thường được sử dụng để xây dựng chương trình

    → Chia chương trình thành các thủ tục và hàm chính, rồi lại chia nhỏ ra thành các thủ tục và hàm nhỏ hơn đến mức mà chúng chỉ thực hiện các công việc cụ thể nhất
  - Một số ngôn ngữ lập trình hướng cấu trúc phổ biến như Basic, C, Pascal, …

Một số mô hình lập trình

- Ưu điểm của lập trình hướng thủ tục
  - Tuyệt vời cho lập trình đa mục đích
  - Dễ dàng thực hiện các trình biên dịch và trình thông dịch
  - Các thuật toán theo hướng thủ tục giúp cho việc học dễ dàng hơn
  - Mã chương trình có tính khả chuyển
  - Có thể sử dụng lại mã nguồn
  - Yêu cầu bộ nhớ ít hơn
  - Dễ dàng theo dõi luồng của chương trình

Một số mô hình lập trình

- Nhược điểm của lập trình hướng thủ tục
  - Mã của chương trình lớn khó viết, khó bảo trì, nhiều lỗi hơn
  - Mã của thủ tục khó sử dụng lại → viết lại mã nếu cần sử dụng trong ứng dụng khác
  - Khó mô tả với các đối tượng trong thế giới thực
  - Chương trình hướng thủ tục tập trung vào thao tác thay vì dữ liệu → có thể gây ra sự cố trong một số trường hợp nhạy cảm với dữ liệu
  - Dữ liệu được hiển thị cho toàn bộ chương trình → việc bảo mật dữ liệu không được tốt lắm

Một số mô hình lập trình

- Lập trình hướng hướng đối tượng (OOP)
  - Là một kỹ thuật lập trình cho phép lập trình viên tạo ra các đối tượng trong mã chương trình trừu tượng hóa các đối tượng trong thực tế cuộc sống
  - Đặc điểm
    - Tập trung vào dữ liệu thay cho các phương thức
    - Chương trình được chia thành các đối tượng độc lập
    - Cấu trúc dữ liệu được thiết kế sao cho đặc tả được các đối tượng
    - Dữ liệu được che giấu và bao bọc
    - Các đối tượng trao đổi thông qua các phương thức

Một số mô hình lập trình

- Lập trình hướng hướng đối tượng (OOP)
  - **Đối tượng** (object) có thể là con người, đồ vật (ô tô, điện thoại, máy tính, ...) và bao gồm hai thành phần chính
    - **Thuộc tính** (Attribute): là những thông tin, đặc điểm của đối tượng
    - **Phương thức** (Method): là những hành động mà đối tượng có thể thực hiện

Một số mô hình lập trình

- Lập trình hướng hướng đối tượng (OOP)
  - Khi các đối tượng có những đặc tính như nhau sẽ được gom lại thành một **lớp đối tượng** (class)
  - **Lớp** (class) cũng có thể được dùng để định nghĩa một kiểu dữ liệu mới
  - Ví dụ, máy vi tính ta hiểu **lớp** máy vi tính gồm có:
    - Các **thuộc tính**: màu sắc, kích thước, bộ nhớ, ...
    - Các **phương thức**: quét virus, tắt máy, khởi động máy, …
    - **Đối tượng** (object) có thể là các dòng máy như Asus, Acer, Lenovo, Thinkpad, ... đều mang đặc tính của lớp máy vi tính

      → Chia thành các đối tượng để xử lý

Một số mô hình lập trình

- Lập trình hướng hướng đối tượng (OOP)
  - Các nguyên lý cơ bản của OOP
    - Tính đóng gói (Encapsulation)
      - Các dữ liệu và phương thức có liên quan với nhau được đóng gói thành các lớp để tiện cho việc quản lý và sử dụng
      - Để che giấu một số thông tin và chi tiết cài đặt nội bộ để bên ngoài không thể nhìn thấy
    - Tính kế thừa (Inheritance)
      - Cho phép xây dựng một lớp mới dựa trên các định nghĩa của lớp đã có → lớp cha có thể chia sẽ dữ liệu và phương thức cho các lớp con
      - Mở rộng các thành phần kế thừa và bổ sung thêm các thành phần mới

Một số mô hình lập trình

- Lập trình hướng hướng đối tượng (OOP)
  - Các nguyên lý cơ bản của OOP
    - Tính đa hình (Polymorphism)
      - Một hành động có thể được thực hiện bằng nhiều cách khác nhau → hai hoặc nhiều lớp có những phương thức giống nhau nhưng có thể thực thi theo những cách thức khác nhau
    - Tính trừu tượng (Abstraction)
      - Trừu tượng có nghĩ là tổng quát hóa một cái gì đó lên, không cần chú ý chi tiết bên trong
      - Tính trừu tượng nghĩa là chọn ra các thuộc tính, phương thức của đối tượng cần cho việc giải quyết bài toán đang lập trình

Một số mô hình lập trình

- Lập trình hướng hướng đối tượng (OOP)
  - Ưu điểm
    - Do tính mô-đun và đóng gói → dễ dàng quản lý mã nguồn khi có sự thay đổi chương trình
    - Nguyên lí đóng gói và che giấu thông tin → có tính bảo mật dữ liệu cao
    - Bắt chước các đối tượng trong thế giới thực → giúp dễ hiểu chương trình hơn
    - Truyền thông và trao đổi thông tin với các đối tượng giúp mô tả giao diện đơn giản
    - Dễ mở rộng dự án → Những hệ thống đối tượng ngày càng được mở rộng và nâng cấp thành những hệ thống lớn
    - Có thể sử dụng lại mã nguồn trong các chương trình khác

Một số mô hình lập trình

- Lập trình hướng hướng đối tượng (OOP)
  - Nhược điểm
    - Các chương trình hướng đối tượng có xu hướng chậm hơn và sử dụng nhiều bộ nhớ
    - Quá khái quát
    - Các chương trình được xây dựng theo mô hình OOP có thể mất nhiều thời gian hơn

Một số mô hình lập trình

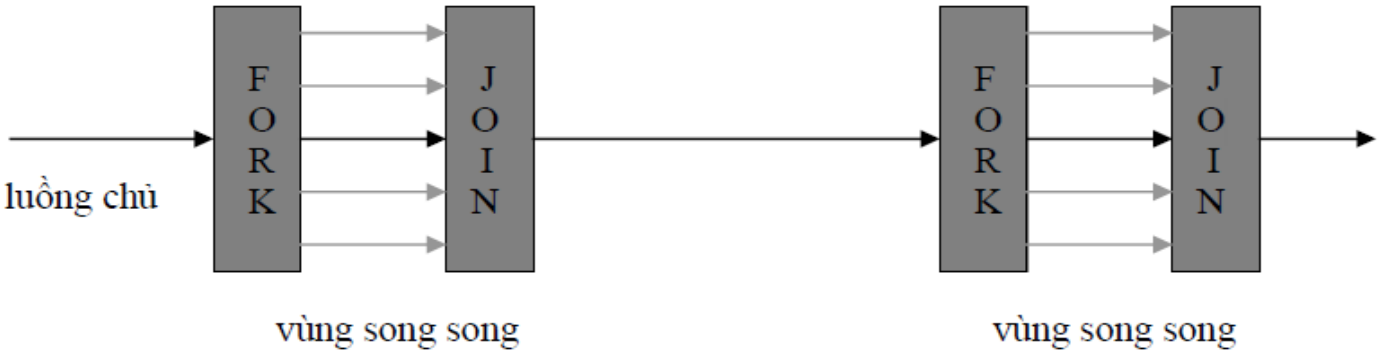
- Lập trình song song
  - Chia nhỏ bài toán để có thể xử lý song song trên nhiều bộ xử lý
  - Các mô hình thông dụng bao gồm
    - **Mô hình chia sẻ bộ nhớ chung**: các tác vụ cùng chia sẻ một không gian địa chỉ chung theo cơ chế không đồng bộ
    - **Mô hình luồng**: luồng chương trình chính được chia thành các tác vụ có dữ liệu riêng và chia sẻ bộ nhớ toàn cục. Mỗi tác vụ được thực hiện bởi các luồng một cách đồng thời
    - **Mô hình truyền thông điệp**: một chương trình song song được chia thành các tác vụ được thực hiện tuần tự trên các bộ xử lý khác nhau và chúng giao tiếp với nhau bằng các thông điệp

Một số mô hình lập trình

- Lập trình song song
  - **Mô hình song song dữ liệu**: song song hóa dựa trên một tập dữ liệu lớn. Mỗi tác vụ thực hiện các thao tác giống nhau trên từng phân vùng khác nhau của dữ liệu

Một số mô hình lập trình

- Lập trình song song
  - **Mô hình Fork-Join**: chương trình bắt đầu bởi một tiến trình đơn gọi là luồng chính và thực hiện tuần tự cho đến khi gặp chỉ thị khai báo vùng cần song song hóa



  - **Fork**: tạo luồng song song khi gặp chỉ thị song song hóa
  - **Join**: khi các luồng đã thực hiện xong → đồng bộ hóa, ngắt luồng và chỉ để lại một luồng chính duy nhất

Một số mô hình lập trình

- Lập trình song song
  - Ý nghĩa ứng dụng:
  - Với xu hướng phát triển như vũ bão của các lĩnh vực như trí tuệ nhân tạo (AI), học máy (Machine learning), dữ liệu lớn (Big data) hiện nay mô hình lập trình song song cũng đang có sự phát triển rất mạnh mẽ
  - Một số công nghệ hỗ trợ lập trình song song như OpenCL framework, CUDA platform (Nvidia), … giúp xây dựng các ứng dụng theo mô hình lập trình song song, tận dụng tài nguyên tính toán của hàng trăm đến hàng nghìn core trên GPU, FPGA
  - Hầu hết các nền tảng xử lý dữ liệu lớn, học máy, … như Apache Hadoop, Apache Spark, Tensor flow, Pandas, … đều xử lý dữ liệu song song

Một số mô hình lập trình

- Mô hình lập trình khai báo
  - Declarative programming: telling the “machine” what you would like to happen, and let the computer figure out how to do it
  - Mô hình lập trình khai báo được hiểu là tôi sẽ nói cho máy tính biết rằng tôi muốn cái gì và sau đó máy tính sẽ tìm cách thực hiện nó cho tôi

Một số mô hình lập trình

- Phân biệt lập trình khai báo và mệnh lệnh
  - Lập trình khai báo → nói những gì bạn muốn mà không cần phải nói làm thế nào để làm điều đó
  - Lập trình mệnh lệnh → phải xác định các bước chính xác để có kết quả

Một số mô hình lập trình

- Phân biệt lập trình khai báo và mệnh lệnh
  - Ví dụ: Tôi cần một tách trà
  - Lập trình mệnh lệnh:
    1. Vào bếp
    2. Lấy đường, sữa và trà,
    3. Trộn chúng và đun nóng trên lửa cho đến khi nó sôi
    4. Đặt nó vào một cái cốc và mang nó đến cho tôi
  - Lập trình khai báo:
    - Lấy cho tôi một tách trà

Một số mô hình lập trình

- Mô hình lập trình khai báo
  - **Ví dụ 1**: một câu lệnh SQL thường dùng để truy vấn cơ sở dữ liệu

```
SELECT CustomerName FROM Customers
WHERE Country='Korean';
```

    - Với cú pháp đơn giản trên là bảo với máy tính rằng tôi muốn lấy ra tên những khách hàng có quốc tịch là Hàn Quốc

      → Tôi không cần quan tâm máy dùng lệnh **for** hay **while**, **if** hay **switch**, DBMS sẽ thực hiện các câu lệnh phù hợp để đưa ra danh sách tên các khách hàng một cách chính xác và nhanh chóng nhất

Một số mô hình lập trình

- Mô hình lập trình khai báo
  - Như ví dụ trên thì SQL thực hiện mô hình lập trình khai báo
    - Mỗi câu lệnh chỉ mô tả hành động được yêu cầu và tùy thuộc vào DBMS để quyết định cách triển khai nó, tức là lên kế hoạch cho các hoạt động cơ bản cần thiết để thực hiện hành động và thực hiện chúng
    - Cần phải hiểu cách DBMS phân tích từng câu lệnh như thế nào để có thể sử dụng SQL một cách hiệu quả

Một số mô hình lập trình

- **Ví dụ 2**: so sánh lập trình mệnh lệnh và lập trình khai báo với ngôn ngữ C#
  - **Bài toán**: cho một tập hợp, tìm các số không chia hết cho 2
    - **Với Lập trình mệnh lệnh** → sử dụng các câu lệnh để duyệt qua danh sách các phần tử

```
List collection = new List { 1, 2, 3, 4, 5 };
List results = new List();
Foreach (var num in collection) {
    if (num % 2 != 0)  results.Add(num);
}
```

    - **Với lập trình khai báo**

```
List collection = new List { 1, 2, 3, 4, 5 };
var results = collection.Where (num => num % 2 != 0);
```

Một số mô hình lập trình

- Mô hình lập trình hàm (functional programming)
  - Mô hình lập trình hàm *coi các cấu trúc và tính toán của một chương trình là việc áp dụng các hàm toán học* và tránh thay đổi các trạng thái của dữ liệu hoặc trạng thái chương trình → Coi các hàm là các giá trị
    - → Về lý tưởng thì chỉ có hàm và chuỗi các hàm, không lệnh gán, không cần tới các biến, không lệnh điều khiển, không lưu giữ trạng thái toàn cục
    - → Cần phối hợp các hàm qua lại, nhận vào hàm, trả ra hàm, lồng ghép, xâu chuỗi, biến hóa chúng theo mọi cách để giải quyết được bài toán hiệu quả nhất

Một số mô hình lập trình

- Mô hình lập trình hàm (functional programming)
  - Ví dụ trong toán học, chúng ta thường viết $y = f(x)$, tức y là một hàm của x
  - Nếu x là dữ liệu, f là tính toán thì y là kết quả của tính toán đó
  - Việc tính $f(x)$ không làm thay đổi x và với mỗi x thì chỉ có một y. Khi gọi hàm f nhiều lần trên cùng một dữ liệu x thì ta sẽ nhận được các kết quả giống nhau → giúp tránh được các hiệu ứng phụ (side-effects) vốn thường gặp trong các mô hình lập trình khác

Một số mô hình lập trình

- Mô hình lập trình hàm (functional programming)
  - Các ngôn ngữ thuần lập trình hàm có ảnh hưởng lớn trong giới học thuật hơn là dùng để phát triển các phần mềm thương mại
  - Một số ngôn ngữ lập trình hỗ trợ đồng thời nhiều phong cách lập trình, kết hợp cả lập trình mệnh lệnh, lập trình hướng đối tượng, lập trình hàm như Scala, Python và JavaScript
  - Các ngôn ngữ lập trình hàm hiện đại được sử dụng trong công nghiệp và thương mại như Erlang, Haskell, Lisp, Scheme, Clojure, F#, Nemerle, Scala, Python và JavaScript

Một số mô hình lập trình

- Mô hình lập trình hàm (functional programming)
  - **Ví dụ**: ngôn ngữ lập trình Scala
    - Scala là ngôn ngữ lập trình chạy trên nền Java, vừa hỗ trợ mô hình lập trình hướng đối tượng, vừa lập trình hàm, có cú pháp gọn hơn Java
    - Scala giúp dễ dàng học thêm phong cách hàm
  - **Bài toán**: tính tổng một mảng các số nguyên

```
val xs = Array(1, 3, 5, 7, 9)
val s = xs.sum
println(s)
```

Lưu ý: Scala không phải là ngôn ngữ thuần lập trình hàm

    - Tư duy của lập trình hàm trong bài toán này là coi *x* là một mảng, *f* là hàm tính tổng, và *y* là kết quả, *y* = *f*(*x*)

Một số mô hình lập trình

- Mô hình lập trình hàm (functional programming)
  - **Bài toán**: tính tổng $s = 1^2 + 3^2 + 5^2 + 7^2 + 9^2$
    - Về mặt toán học → cần sử dụng hai hàm, một hàm tính bình phương của một số và một hàm tính tổng của nhiều số
    - Sử dụng hàm map (ánh xạ) của Scala tác động lên mảng số để biến mỗi phần tử của mảng đó thành một số chính phương và sau đó dùng hàm sum để tính tổng

```
val r = xs.map(x => x*x).sum
println(r)
```

    - Ký hiệu mũi tên => chính là phép biến đổi hàm giúp chuyển mỗi số vào x thành số ra tương ứng là x^2
    - Trong toán học, đây là phép tổ hợp hàm hay dùng hàm hợp: y = g(f(x)) với f và g là các hàm

Một số mô hình lập trình

- Mô hình lập trình hàm (functional programming)
  - Ưu điểm
    - Cung cấp một môi trường được bảo vệ
    - Trong khi nhiều ngôn ngữ khác yêu cầu một lượng thông tin đáng kể để thực hiện các thao tác đúng cách, lập trình hàm loại bỏ một lượng lớn mã cần thiết để xác định trạng thái
    - Vì mô hình này chỉ phụ thuộc vào các đối số đầu vào, không có hiệu ứng phụ

Một số mô hình lập trình

- Mô hình lập trình hàm (functional programming)
  - Nhược điểm
    - Không nên sử dụng lập trình hàm trong phát triển phần mềm thương mại
    - Nó đòi hỏi một lượng lớn bộ nhớ và thời gian
    - Có thể chứng minh là nó kém hiệu quả hơn các mô hình lập trình khác

Một số mô hình lập trình

- Mô hình lập trình logic (luận lý)
  - Coi tính toán là suy luận logic tự động trên vốn hiểu biết đã có (cơ sở dữ liệu tri thức) được tạo ra từ các sự kiện và luật (quy tắc) → để chứng minh câu hỏi là một mệnh đề, là đúng hay là sai
  - Một chương trình gồm một tập hợp các câu ở dạng logic và mỗi câu diễn đạt các sự kiện và luật về một miền vấn đề

Một số mô hình lập trình

- Mô hình lập trình logic (luận lý)
  - Sự kiện (fact) và luật (rule)
    - Sự kiện là những tuyên bố đúng. Chẳng hạn, nói rằng: Hà Nội là thủ đô của Việt Nam
    - Các luật là các ràng buộc dẫn chúng ta đến kết luận về miền vấn đề → là những mệnh đề logic thể hiện sự kiện

Một số mô hình lập trình

- Mô hình lập trình logic (luận lý)
  - Ví dụ: một số mệnh đề Horn như sau
    1) Nếu một người già và khôn khoan thì hạnh phúc
    2) An là người hạnh phúc
    3) Nếu *X* là cha của *Y* và *Y* là cha của *Z* thì *X* là ông của *Z*
    4) Hùng là ông của Hoàng
    5) Tất cả mọi người đều chết
    6) Huy là người
  - Các mệnh đề 1, 3, 5 là các luật. Các mệnh đề còn lại là các sự kiện.
  - Chạy chương trình logic bằng cách đặt câu hỏi. Ví dụ: *Huy có chết không*? Câu trả lời là *Yes* vì thỏa *luật 5*.

Một số mô hình lập trình

- Mô hình lập trình logic (luận lý)
  - Ngôn ngữ lập trình logic
    - Prolog: PROgramming in LOGic
    - Datalog: dùng cho cơ sở dữ liệu suy diễn
    - Python

Một số mô hình lập trình

- Mô hình lập trình hướng dữ liệu
  - Dựa trên dữ liệu và luồng hoạt động của nó
  - Thay vì cung cấp các bước để thực hiện chương trình thì LTV cung cấp dữ liệu cần lấy và các xử lý cần thiết
  - Ví dụ: SQL mang tính khai báo vì các truy vấn không chỉ định các bước để tạo kết quả

Phân tích thuật toán

- Khái niệm thuật toán
  - Một dãy hữu hạn các thao tác và trình tự thực hiện các thao tác đó sao cho sau khi thực hiện dãy thao tác này theo trình tự đã chỉ ra, với đầu vào xác định ta thu được kết quả đầu ra mong muốn
  - Các đặc trưng của thuật toán
    - **Đầu vào**
    - **Đầu ra**
    - **Tính đúng đắn**: kết quả của thuật toán là chính xác
    - **Tính xác định**: các bước thực hiện có trình tự xác định
    - **Tính dừng**: phải cho ra kết quả sau một số hữu hạn bước
    - **Tính phổ dụng**: áp dụng cho các bài toán cùng dạng
    - **Tính khách quan**: cho kết quả như nhau khi chạy trên các máy tính khác nhau

Phân tích thuật toán

- Biểu diễn thuật toán
  - Cách 1: Ngôn ngữ tự nhiên
  - Cách 2: Ngôn ngữ lưu đồ (sơ đồ khối)
  - Cách 3: Mã giả
  - Cách 4: Các ngôn ngữ lập trình như Pascal, C/C++, Java, … nhưng không nhất thiết phải sử dụng đúng ký pháp của ngôn ngữ đó

Phân tích thuật toán

- Độ phức tạp của thuật toán
  - Độ phức tạp tính toán của thuật toán được xác định như là lượng tài nguyên các loại mà thuật toán đòi hỏi sử dụng
  - Có hai loại tài nguyên quan trọng đó là thời gian và bộ nhớ (không gian lưu trữ)
  - Việc tính chính xác được các loại tài nguyên mà thuật toán đòi hỏi là rất khó. Vì thế ta quan tâm đến việc đưa ra các đánh giá sát thực cho các đại lượng này
  - Trong môn học này ta đặc biệt quan tâm đến đánh giá thời gian cần thiết để thực hiện thuật toán mà ta sẽ gọi là ***thời gian tính*** của thuật toán

Phân tích thuật toán

- Độ phức tạp của thuật toán
  - Rõ ràng: Thời gian tính phụ thuộc vào dữ liệu vào

    Ví dụ: Việc nhân hai số nguyên có 3 chữ số đòi hỏi thời gian khác hẳn so với việc nhân hai số nguyên có $3*10^9$ chữ số!
  - Kích thước dữ liệu đầu vào (độ dài dữ liệu vào): là số bít cần thiết để biểu diễn nó
  - **Ví dụ**: Nếu *x, y* là đầu vào cho bài toán nhân hai số nguyên thì kích thước dữ liệu vào của bài toán là $n = \lceil \log |x| \rceil + \lceil \log |y| \rceil$

Phân tích thuật toán

- Phép toán cơ bản (phép tính cơ bản)
  - Đo thời gian tính bằng đơn vị đo nào?
  - **Định nghĩa**. Ta gọi ***phép toán cơ bản*** là phép toán có thể thực hiện với thời gian bị chặn bởi một hằng số không phụ thuộc vào kích thước dữ liệu đầu vào
  - Để tính toán thời gian tính của thuật toán ta sẽ đếm ***số phép toán cơ bản*** mà nó phải thực hiện
  - **Ví dụ**: trong phân tích thuật toán, ta thường coi các phép gán, cộng, trừ, nhân, chia, lũy thừa, … là các phép toán cơ bản

Phân tích thuật toán

- Đếm số phép toán cơ bản
  - Nhận diện các phép toán cơ bản trong thuật toán
  - Nếu có nhiều phép toán cơ bản thì xác định phép toán cơ bản T chiếm nhiều thời gian chạy nhất so với các phép toán cơ bản còn lại (ví dụ: i = i + 1 → coi phép cộng chạy lâu hơn phép gán)
  - Phép toán T thường xuất hiện trong các vòng lặp
  - Đếm số lần thực hiện phép toán T, sẽ thu được hàm thời gian chạy $f(n)$

Phân tích thuật toán

- Ví dụ đếm số phép toán cơ bản

Ví dụ 1: In các phần tử

```
for (i = 0; i < n; i++)
   cout << a[i] << endl;
```

Số lần in ra màn hình = n

Ví dụ 2:
Nhân ma trận tam giác dưới với véctơ (mã giả)

```
for (i = 0; i< n; i++)
    c[i] = 0
for (i = 0; i < n; i++)
    for (j = 0; j < i; j++)
        c[i] = c[i] + a[i][j] * b[j];
```

Số phép nhân: n(n+1)/2

Ví dụ 3: Kiểm tra tính sắp xếp

```
template <class T>
bool isSorted(T *a, int n)
{
    bool sorted = true;
    for (int i=1; i<n; i++)
        if (a[i-1] > a[i])
            sorted = false;
    return sorted;
}
```

Số phép so sánh là n-1

Phân tích thuật toán

- Các loại thời gian tính
  - Thời gian tối thiểu cần thiết để thực hiện thuật toán với mọi bộ dữ liệu đầu vào kích thước *n*. Thời gian như vậy sẽ được gọi là ***thời gian tính tốt nhất*** của thuật toán với đầu vào kích thước *n*
  - Thời gian nhiều nhất cần thiết để thực hiện thuật toán với mọi bộ dữ liệu đầu vào kích thước *n*. Thời gian như vậy sẽ được gọi là ***thời gian tính tồi nhất*** của thuật toán với đầu vào kích thước *n*
  - Thời gian trung bình cần thiết để thực hiện thuật toán trên tập hữu hạn các đầu vào kích thước *n*. Thời gian như vậy sẽ được gọi là ***thời gian tính trung bình*** của thuật toán

Phân tích thuật toán

- Ký hiệu tiệm cận
  - Các ký hiệu Θ, O, Ω
  - Được xác định đối với các hàm nhận giá trị nguyên không âm
  - Dùng để so sánh tốc độ tăng của các hàm
  - Được sử dụng để mô tả thời gian tính của thuật toán. Thay vì nói chính xác, ta có thể nói thời gian tính là, chẳng hạn, $\Theta(n^2)$

Phân tích thuật toán

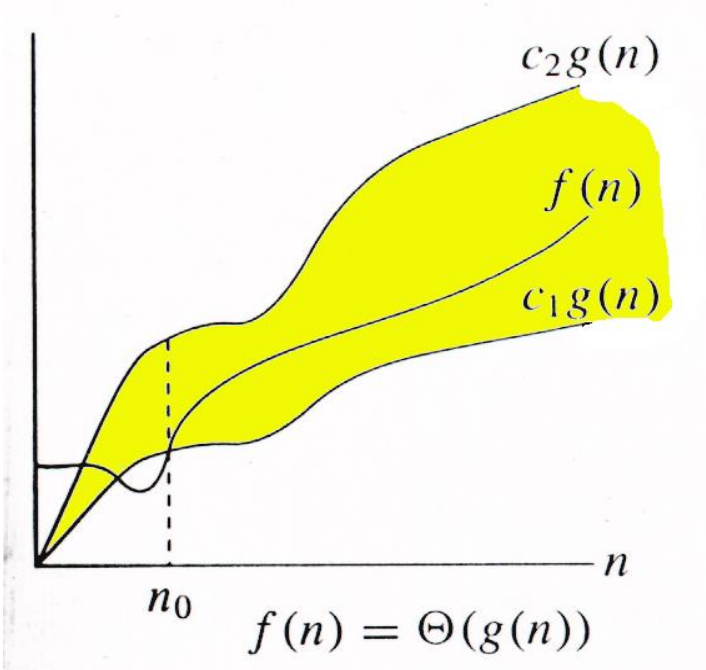
- Ký hiệu Θ

Đối với hàm $g(n)$ cho trước, ta ký hiệu $\Theta(g(n))$ là tập các hàm

$\Theta(g(n)) = \{f(n) \mid$ tồn tại các hằng số c_1, c_2 và n_0 sao cho

$0 \le c_1 g(n) \le f(n) \le c_2 g(n)$, với mọi $n \ge n_0\}$

Ta nói rằng $g(n)$ là ***đánh giá tiệm cận đúng*** cho $f(n)$



Phân tích thuật toán

- Ví dụ
  - $10n^2 - 3n = \Theta(n^2)$
  - Với giá trị nào của các hằng số n_0, c_1, và c_2 thì bất đẳng thức sau đây là đúng với $n \geq n_0$:

$$c_1 n^2 \leq 10n^2 - 3n \leq c_2 n^2$$

  - Ta có thể lấy c_1 bé hơn hệ số của số hạng với số mũ cao nhất, còn c_2 lấy lớn hơn hệ số này, chẳng hạn:

$$c_1 = 9 < 10 < c_2 = 11,\ n_0 = 10.$$

  - *Tổng quát, để so sánh tốc độ tăng của các đa thức, cần nhìn vào số hạng với số mũ cao nhất*

Phân tích thuật toán

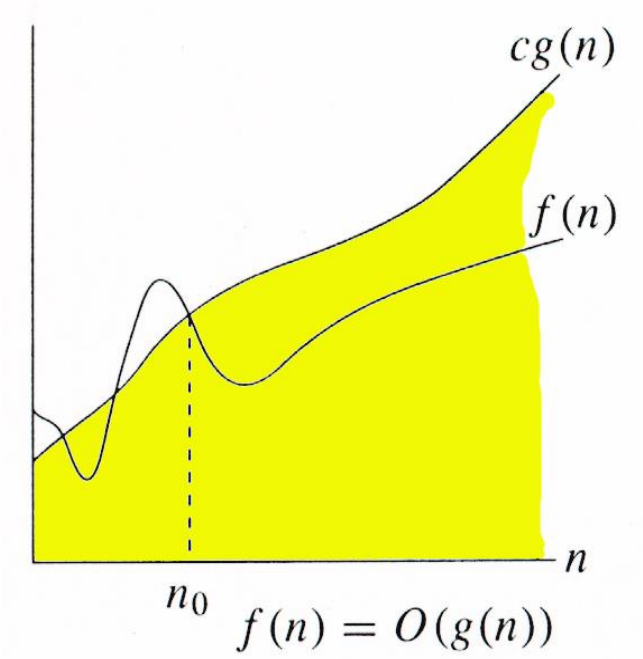
- Kí hiệu *O* lớn

Đối với hàm $g(n)$ cho trước, ta ký hiệu $O(g(n))$ là tập các hàm

$O(g(n)) = \{f(n)$ | tồn tại các hằng số dương c và n_0 sao cho:

$$f(n) \leq c \times g(n) \text{ với mọi } n \geq n_0 \}$$

Ta nói $g(n)$ là *cận trên tiệm cận* của $f(n)$



Phân tích thuật toán

- Ví dụ kí hiệu *O* lớn
  - **Chứng minh:** $f(n) = n^2 + 2n + 1$ là $O(n^2)$
    - Cần chỉ ra: $n^2 + 2n + 1 \leq c*n^2$

      với c là hằng số nào đó và khi $n \geq n_0$ nào đó
  - Ta có:

    $2n^2 \geq 2n$ khi $n \geq 1$

    và $n^2 \geq 1$ khi $n \geq 1$
  - Vì vậy

    $n^2 + 2n + 1 \leq 4*n^2$ với mọi $n \geq 1$
  - Như vậy hằng số $c = 4$, và $n_0 = 1$

Phân tích thuật toán

- Ví dụ kí hiệu *O* lớn
  - Ta thấy rằng: Nếu $f(n)$ là $O(n^2)$ thì nó cũng là $O(n^k)$ với $k > 2$.
  - **Chứng minh:** $f(n) = n^2 + 2n + 1 \notin O(n)$.
  - Phản chứng. Giả sử trái lại, khi đó phải tìm được hằng số c và số n_0 để cho:

    $$n^2 + 2n + 1 \le c*n \text{ khi } n \ge n_0$$

  - Suy ra

    $$n^2 < n^2 + 2n + 1 \le c*n \text{ với mọi } n \ge n_0$$

  - Từ đó ta thu được:

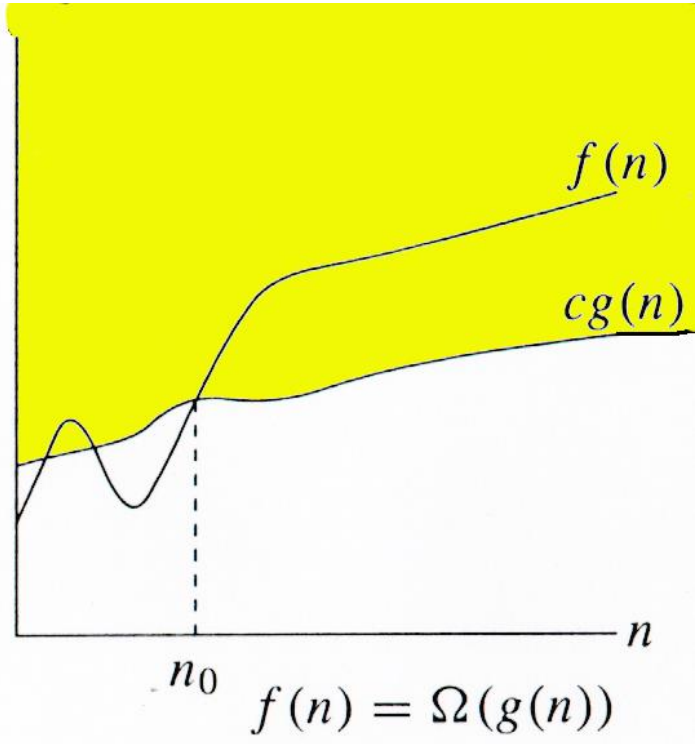
    $$n < c \text{ với mọi } n \ge n_0 \quad ?!$$

Phân tích thuật toán

- Kí hiệu Ω

Đối với hàm $g(n)$ cho trước, ta ký hiệu $\Omega(g(n))$ là tập các hàm:

$\Omega(g(n)) = \{f(n)|$ tồn tại các hằng số dương c và n_0 sao cho $cg(n) \leq f(n)$ với mọi $n \geq n_0\}$



Ta nói *g*(*n*) là *cận dưới tiệm cận* cho *f*(*n*)

Phân tích thuật toán

- Ví dụ kí hiệu Ω
  - **Chứng minh:** $f(n) = n^2 - 2000n$ là $\Omega(n^2)$
    - Cần chỉ ra: $n^2 - 2000n \geq c*n^2$

      với c là hằng số nào đó và khi $n \geq n_0$ nào đó
  - Ta có:

    $n^2 - 2000n \geq 0.5*n^2$ với mọi $n \geq 10000$

    (vì $n^2 - 2000n - 0.5*n^2 = 0.5*n^2 - 2000n$

    $= n(0.5*n - 2000) \geq 0$

    khi $n \geq 10000$)
  - Như vậy hằng số $c = 1$, và $n_0 = 10000$

Phân tích thuật toán

- Liên hệ giữa Θ, Ω, O
  - Đối với hai hàm bất kỳ $g(n)$ và $f(n)$,

$$f(n) = \Theta(g(n))$$

khi và chỉ khi

$$f(n) = O(g(n)) \text{ và } f(n) = \Omega(g(n)).$$

tức là

$$\Theta(g(n)) = O(g(n)) \cap \Omega(g(n))$$

Phân tích thuật toán

- Tính chất bắc cầu
  - Nếu $f(n) = O(g(n))$ và $g(n) = O(h(n))$

    → $f(n) = O(h(n))$
  - Nếu $f(n) = \Omega(g(n))$ và $g(n) = \Omega(h(n))$

    → $f(n) = \Omega(h(n))$
  - Nếu $f(n) = \Theta(g(n))$ và $g(n) = \Theta(h(n))$

    → $f(n) = \Theta(h(n))$

Phân tích thuật toán

- Chú ý
  - Giá trị của n_0 và c ***không phải là duy nhất*** trong chứng minh công thức tiệm cận
  - Chứng minh rằng $100n + 5 = O(n^2)$
    - $100n + 5 \leq 100n + n = 101n \leq 101n^2$ với mọi $n \geq 5$

      $n_0 = 5$ và $c = 101$ là các hằng số cần tìm
    - $100n + 5 \leq 100n + 5n = 105n \leq 105n^2$ với mọi $n \geq 1$

      $n_0 = 1$ và $c = 105$ cũng là các hằng số cần tìm
  - Chỉ cần tìm các hằng c và n_0 **nào đó** thoả mãn bất đẳng thức trong định nghĩa công thức tiệm cận

Phân tích thuật toán

- Cách nói về thời gian tính
  - Nói "Thời gian tính là $O(f(n))$" hiểu là: Đánh giá trong tình huống tồi nhất (worst case) là $O(f(n))$. Thường nói: "Đánh giá thời gian tính trong tình huống tồi nhất là $O(f(n))$"
    - Nghĩa là thời gian tính trong tình huống tồi nhất được xác định bởi một hàm nào đó $g(n) \in O(f(n))$
  - "Thời gian tính là $\Omega(f(n))$" hiểu là: Đánh giá trong tình huống tốt nhất (best case) là $\Omega(f(n))$. Thường nói: "Đánh giá thời gian tính trong tình huống tốt nhất là $\Omega(f(n))$"
    - *Nghĩa là thời gian tính trong tình huống tốt nhất được xác định bởi một hàm nào đó* $g(n) \in \Omega(f(n))$

Phân tích thuật toán

- Tên gọi của một số tốc độ tăng

Hàm	Tên gọi
c	Hằng
$\log n$	Logarit
$\log^2 n$	Logarit bình phương
n	tuyến tính
$n \log n$	n log n
n^2	bình phương
n^3	bậc 3
2^n	hàm mũ
n^k (k là hằng số dương)	đa thức

Phân tích thuật toán

- Hàm nào tăng chậm hơn?
  - $f(n) = n\log n$ và $g(n) = n^{1,5}$
  - Lời giải:
    - Chú ý rằng $g(n) = n^{1,5} = n * n^{0,5}$
    - Vì vậy, chỉ cần so sánh $\log n$ và $n^{0,5}$
    - Tương đương với so sánh $\log^2 n$ và n
    - Hàm $\log^2 n$ tăng chậm hơn n
    - Suy ra $f(n)$ tăng chậm hơn $g(n)$

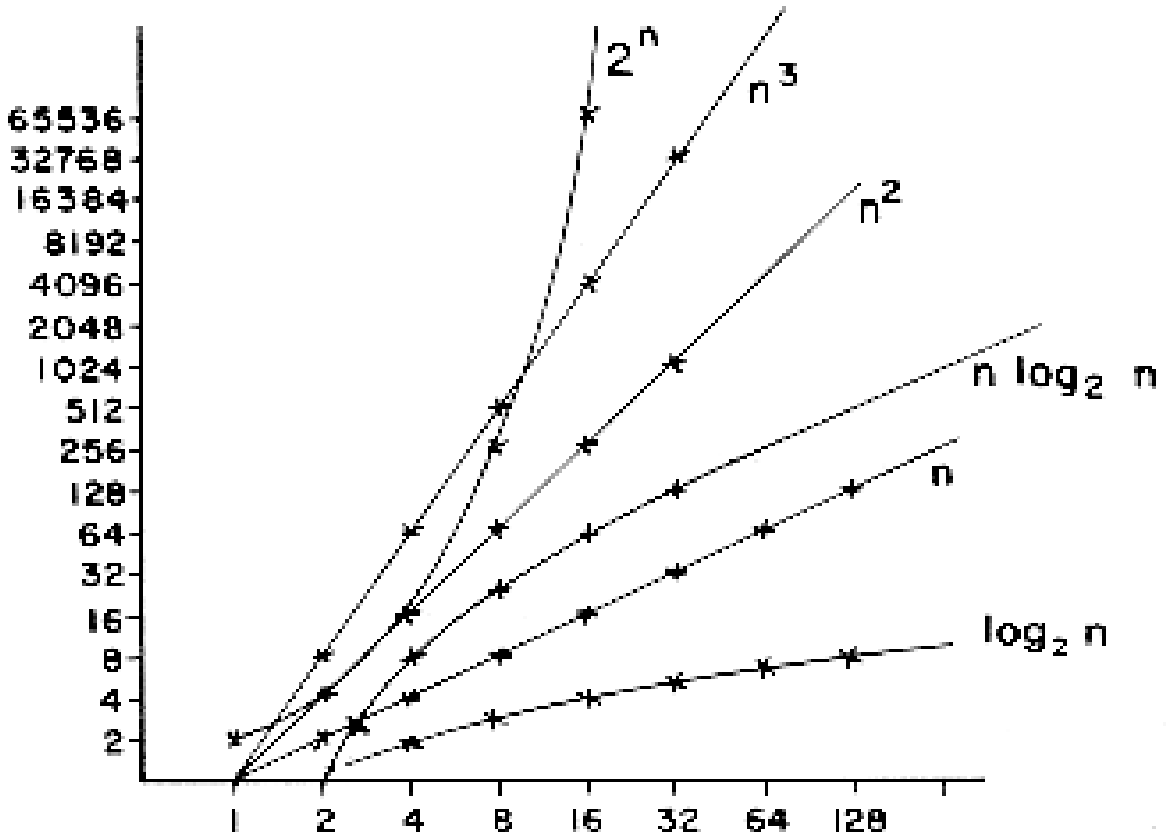
Phân tích thuật toán

- Hàm nào tăng chậm hơn?
  - Xét một chiếc máy tính thực hiện được 1.000.000 phép tính cơ bản trong một giây
  - Khi thời gian chạy vượt quá 10^{25} năm, ta viết "very long"

	n	$n \log_2 n$	n^2	n^3	1.5^n	2^n	$n!$
$n = 10$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	4 sec
$n = 30$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	18 min	10^{25} years
$n = 50$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	11 min	36 years	very long
$n = 100$	< 1 sec	< 1 sec	< 1 sec	1 sec	12,892 years	10^{17} years	very long
$n = 1{,}000$	< 1 sec	< 1 sec	1 sec	18 min	very long	very long	very long
$n = 10{,}000$	< 1 sec	< 1 sec	2 min	12 days	very long	very long	very long
$n = 100{,}000$	< 1 sec	2 sec	3 hours	32 years	very long	very long	very long
$n = 1{,}000{,}000$	1 sec	20 sec	12 days	31,710 years	very long	very long	very long

Phân tích thuật toán

- Đồ thị của một số hàm cơ bản



Phân tích thuật toán

- Ví dụ 1

```
1  for (i = 0; i < n; i++)
2  {
3      x = a[i] / 2;
4      a[i] = x + 1;
5  }
```

- Có 4 phép tính cơ bản ở các dòng 3 và 4 gồm 2 phép gán, 1 phép chia và 1 phép cộng (*bỏ qua 3 phép tính cơ bản điều khiển quá trình lặp ở dòng 1*)
- Cả 4 phép tính đó được lặp lại *n* lần
- Thời gian chạy: $T(n) = 4n = O(n)$

Quy tắc bỏ hằng số: Nếu đoạn chương trình P có thời gian thực hiện $T(n) = O(c1.f(n))$ với c1 là một hằng số dương thì có thể coi đoạn chương trình đó có độ phức tạp tính toán là $O(f(n))$

Phân tích thuật toán

- Ví dụ 2

Xét thuật toán tìm kiếm tuần tự để giải bài toán

Đầu vào: *n* và dãy số $s_1, s_2, \ldots, s_n$.

Đầu ra: Vị trí phần tử có giá trị *key* hoặc là *n* nếu không tìm thấy

```
void  Linear_Search(s, n, key)  {
    i = -1;
    do {
            i = i + 1;
    } while  (i < n) || (key = s[i]);
    return i;
}
```

Phân tích thuật toán

- Ví dụ 2
  - Cần đánh giá thời gian tính tốt nhất, tồi nhất, trung bình của thuật toán với độ dài đầu vào là *n.*
  - Thời gian tính của thuật toán có thể được đánh giá bởi số lần thực hiện câu lệnh $i = i + 1$ trong vòng lặp **do** (*thực tế là câu lệnh này có hai phép tính cơ bản là phép gán và phép cộng. Ta cũng bỏ qua hai phép so sánh của vòng lặp* ***do***)
  - Nếu $s_0 = key$ thì câu lệnh $i = i + 1$ trong thân vòng lặp **do** thực hiện 1 lần. Do đó thời gian tính tốt nhất của thuật toán là $\Omega(1)$
  - Nếu *key* không có mặt trong dãy đã cho, thì câu lệnh $i=i+1$ thực hiện n lần. Vì thế thời gian tính tồi nhất của thuật toán là $O(n)$

Phân tích thuật toán

- Ví dụ 2
  - Cuối cùng, ta tính thời gian tính trung bình của thuật toán
    - Nếu *key* tìm thấy ở vị trí thứ *i* của dãy (*key* = s_i) thì câu lệnh *i* = *i*+1 phải thực hiện *i* lần (*i* = 1, 2, ..., *n*),
    - Nếu *key* không có mặt trong dãy đã cho thì câu lệnh *i* = *i*+1 phải thực hiện *n* lần.
  - Từ đó suy ra số lần trung bình phải thực hiện câu lệnh *i* = *i*+1 là

    $$[(1 + 2 + \ldots + n) + n] /(n+1) = [n(n+1)/2 + n]/(n+1)$$
  - Ta có

    $$n/4 \leq [n(n+1)/2 + n]/(n+1) \leq n \text{ với mọi } n \geq 1$$
  - Vậy thời gian tính trung bình của thuật toán là $\Theta(n)$

Phân tích thuật toán

- Ví dụ 3: vòng lặp có câu lệnh break

```
1  for (i = 0; i < n; i++)
2  {
3      x = a[i] / 2;
4      a[i] = x + 1;
5      if (a[i] > 10) break;
6  }
```

- Có 5 phép tính cơ bản ở các dòng 3, 4, 5 gồm 2 phép gán, 1 phép chia, 1 phép cộng và 1 phép so sánh
- Không thể đếm chính xác số lần thực hiện 5 thao tác đó vì ta không biết khi nào điều kiện a[i]>10 xảy ra
- Trường hợp tồi nhất, điều kiện a[i]>10 xảy ra ở bước lặp cuối cùng hoặc không xảy ra, cả 5 thao tác cơ bản được lặp lại n lần
- Thời gian chạy trong trường hợp tồi nhất: $T(n) = 5n = O(n)$

Phân tích thuật toán

- Ví dụ 4: Các vòng lặp tuần tự

```
1   for (i = 0; i < n; i++)
2   {
3       // 3 phép tính cơ bản trong vòng lặp
4   }
5   for (i = 0; i < n; i++)
6   {
7       // 5 phép tính cơ bản trong vòng lặp
6   }
```

  - Chỉ cần cộng thời gian chạy của các vòng lặp
  - Thời gian chạy tổng thể: $T(n) = 3n + 5n = 8n = O(n)$

Quy tắc cộng – lấy max: Nếu T1(n) và T2(n) là thời gian thực hiện của hai đoạn chương trình P1 và P2; và T1(n)=O($f(n)$), T2(n)=$O(g(n))$ thì thời gian thực hiện của đoạn hai chương trình đó nối tiếp nhau là T(n)=O(max($f(n)$,$g(n)$))

Phân tích thuật toán

- Ví dụ 5: Các vòng lặp lồng nhau

```
1   for (i = 0; i < n; i++) {
2         // 3 phép tính cơ bản ở đây
3         for (j=0; j < n; j++) {
4               // 3 phép tính cơ bản trong vòng lặp
5         }
6    }
```

  - Vòng lặp bên trong thực hiện $3n$ phép tính cơ bản
  - Mỗi bước lặp của vòng lặp bên ngoài thực hiện $3n$ phép tính cơ bản
  - Thời gian chạy tổng thể: $T(n) = 3n*3n = 9n^2 = O(n^2)$

Quy tắc nhân: Nếu T1(n) và T2(n) là thời gian thực hiện của hai đoạn chương trình P1 và P2 và T1(n) = $O(f(n))$, T2(n) = $O(g(n))$ thì thời gian thực hiện của đoạn hai đoạn chương trình lồng nhau đó là T(n) = $O(f(n)\times g(n))$

Phân tích thuật toán

- Bài tập
  1. Sắp xếp danh sách các hàm sau đây theo thứ tự tăng dần của tốc độ tăng trưởng

$$f_1(n) = n^{2,5}$$

$$f_2(n) = \sqrt{2n}$$

$$f_3(n) = n + 10$$

$$f_4(n) = 10^n$$

$$f_5(n) = 100^n$$

$$f_6(n) = n^2 \log n$$

Phân tích thuật toán

- Bài tập
  2. Phân tích thời gian chạy của thuật toán tìm phần tử lớn nhất (dùng ký hiệu *O*)

```
max = a[0];
for (i = 1; i < n; i++)
    if (a[i] > max)
        max = a[i];
```

Phân tích thuật toán

- Bài tập

3. Phân tích thời gian chạy của thuật toán hợp nhất hai danh sách đã sắp xếp $A = a_1, a_2, \ldots, a_n$ và $B = b_1, b_2, \ldots, b_n$ thành một danh sách tổng thể C cũng được sắp xếp

```
i = 0; j = 0;
while (cả A và B không rỗng) {
  if (a_i ≤ b_j)
      Thêm a_i vào cuối C và tăng i một đơn vị;
  else
      Thêm b_j vào cuối C và tăng j một đơn vị
}
Thêm phần còn lại của danh sách không rỗng vào cuối C
```

Phân tích thuật toán

- Bài tập

4. Hãy phân tích thời gian chạy của thuật toán sau

```
for (i = 0; i < n; i++) {
    for (j = i + 1; j < n; j++) {
        Cộng các phần tử A[i] và A[j];
        Lưu trữ kết quả vào B[i,j];
    }
}
```

Phân tích thuật toán

- Bài tập

5. Phân tích thời gian chạy của các đoạn chương trình C++ sau đây

(a)
```
sum = 0;
for (i = 0; i < n; i++)
    sum++;
```

(b)
```
sum = 0;
for (i = 0; i < n; i++)
    for (j = 0; j < n; j++)
        sum++;
```

(c)
```
sum = 0;
for (i = 0; i < n; i++)
    for (j = 0; j < n*n; j++)
        sum++;
```

TỔNG KẾT

1. **Tổng quan**