**Cấu trúc dữ liệu và giải thuật**

**(Data Structure and Algorithms)**

Với các sinh viên chuyên nghành tin học thì cụm từ Cấu trúc dữ liệu (Data Structure) không còn là xa lạ. Đây là một môn học bắt buộc và sẽ là thực sự khó cho bất kỳ sinh viên nào nếu không có sự chuẩn bị kỹ lưỡng và dành cách tiếp cận tích cực cho môn học này. Vậy Cấu trúc dữ liệu là gì?

**Cấu trúc dữ liệu** là cách lưu trữ, tổ chức dữ liệu có thứ tự, có hệ thống để dữ liệu có thể được sử dụng một cách hiệu quả.

**Cấu trúc dữ liệu (Data Structure) là gì ?**

Cấu trúc dữ liệu là cách lưu trữ, tổ chức dữ liệu có thứ tự, có hệ thống để dữ liệu có thể được sử dụng một cách hiệu quả.

Dưới đây là hai khái niệm nền tảng hình thành nên một cấu trúc dữ liệu:

* Interface: Mỗi cấu trúc dữ liệu có một Interface. Interface biểu diễn một tập hợp các phép tính mà một cấu trúc dữ liệu hỗ trợ. Một Interface chỉ cung cấp danh sách các phép tính được hỗ trợ, các loại tham số mà chúng có thể chấp nhận và kiểu trả về của các phép tính này.
* Implementation (có thể hiểu là sự triển khai): Cung cấp sự biểu diễn nội bộ của một cấu trúc dữ liệu. Implementation cũng cung cấp phần định nghĩa của giải thuật được sử dụng trong các phép tính của cấu trúc dữ liệu.

Đặc điểm của một Cấu trúc dữ liệu

* Chính xác: Sự triển khai của Cấu trúc dữ liệu nên triển khai Interface của nó một cách chính xác.
* Độ phức tạp về thời gian (Time Complexity): Thời gian chạy hoặc thời gian thực thi của các phép tính của cấu trúc dữ liệu phải là nhỏ nhất có thể.
* Độ phức tạp về bộ nhớ (Space Complexity): Sự sử dụng bộ nhớ của mỗi phép tính của cấu trúc dữ liệu nên là nhỏ nhất có thể.

Tại sao Cấu trúc dữ liệu là cần thiết ?

Ngày nay, các ứng dụng ngày càng phức tạp và lượng dữ liệu ngày càng lớn với nhiều kiểu đa dạng. Việc này làm xuất hiện 3 vấn đề lớn mà mỗi lập trình viên phải đối mặt:

* Tìm kiếm dữ liệu: Giả sử có 1 triệu hàng hóa được lưu giữ vào trong kho hàng hóa. Và giả sử có một ứng dụng cần để tìm kiếm một hàng hóa. Thì mỗi khi thực hiện tìm kiếm, ứng dụng này sẽ phải tìm kiếm 1 hàng hóa trong 1 triệu hàng hóa. Khi dữ liệu tăng lên thì việc tìm kiếm sẽ càng trở lên chậm và tốn kém hơn.
* Tốc độ bộ vi xử lý: Mặc dù bộ vi xử lý có tốc độ rất cao, tuy nhiên nó cũng có giới hạn và khi lượng dữ liệu lên tới hàng tỉ bản ghi thì tốc độ xử lý cũng sẽ không còn được nhanh nữa.
* Đa yêu cầu: Khi hàng nghìn người dùng cùng thực hiện một phép tính tìm kiếm trên một Web Server thì cho dù Web Server đó có nhanh đến mấy thì việc phải xử lý hàng nghìn phép tính cùng một lúc là thực sự rất khó.

Để xử lý các vấn đề trên, các cấu trúc dữ liệu là một giải pháp tuyệt vời. Dữ liệu có thể được tổ chức trong cấu trúc dữ liệu theo một cách để khi thực hiện tìm kiếm một phần tử nào đó thì dữ liệu yêu cầu sẽ được tìm thấy ngay lập tức.

Độ phức tạp thời gian thực thi trong cấu trúc dữ liệu và giải thuật

Có 3 trường hợp thường được sử dụng để so sánh thời gian thực thi của các cấu trúc dữ liệu khác nhau:

* Trường hợp xấu nhất (Worst Case): là tình huống mà một phép tính của cấu trúc dữ liệu nào đó tốn thời gian tối đa (thời gian dài nhất). Ví dụ với ba số 1, 2, 3 thì nếu sắp xếp theo thứ tự giảm dần thì thời gian thực thi sẽ là dài nhất (và đây là trường hợp xấu nhất); còn nếu sắp xếp theo thứ tự tăng dần thì thời gian thực thi sẽ là ngắn nhất (và đây là trường hợp tốt nhất).
* Trường hợp trung bình (Average Case): miêu tả thời gian thực thi trung bình một phép tính của một cấu trúc dữ liệu.
* Trường hợp tốt nhất (Best Case): là tình huống mà thời gian thực thi một phép tính của một cấu trúc dữ liệu là ít nhất. Ví dụ như trên.

Thuật ngữ cơ bản trong Cấu trúc dữ liệu

* Dữ liệu: Dữ liệu là các giá trị hoặc là tập hợp các giá trị.
* Phần tử dữ liệu: Phần tử dữ liệu là một đơn vị đơn lẻ của giá trị.
* Các phần tử nhóm: Phần tử dữ liệu mà được chia thành các phần tử con thì được gọi là các phần tử nhóm.
* Các phần tử cơ bản: Phần tử dữ liệu mà không thể bị chia nhỏ thành các phần tử con thì gọi là các phần tử cơ bản.
* Thuộc tính và Thực thể: Một thực thể là cái mà chứa một vài thuộc tính nào đó, và các thuộc tính này có thể được gán các giá trị.
* Tập hợp thực thể: Các thực thể mà có các thuộc tính tương tự nhau thì cấu thành một tập hợp thực thể.
* Trường: Trường là một đơn vị thông tin cơ bản biểu diễn một thuộc tính của một thực thể.
* Bản ghi: Bản ghi là một tập hợp các giá trị trường của một thực thể đã cho.
* File: Là một tập hợp các bản ghi của các thực thể trong một tập hợp thực thể đã cho.

**Giải thuật là gì ?**

Giải thuật (hay còn gọi là thuật toán - tiếng Anh là Algorithms) là một tập hợp hữu hạn các chỉ thị để được thực thi theo một thứ tự nào đó để thu được kết quả mong muốn. Nói chung thì giải thuật là độc lập với các ngôn ngữ lập trình, tức là một giải thuật có thể được triển khai trong nhiều ngôn ngữ lập trình khác nhau.

Xuất phát từ quan điểm của cấu trúc dữ liệu, dưới đây là một số giải thuật quan trọng:

* Giải thuật Tìm kiếm: Giải thuật để tìm kiếm một phần tử trong một cấu trúc dữ liệu.
* Giải thuật Sắp xếp: Giải thuật để sắp xếp các phần tử theo thứ tự nào đó.
* Giải thuật Chèn: Giải thuật để chèn phần từ vào trong một cấu trúc dữ liệu.
* Giải thuật Cập nhật: Giải thuật để cập nhật (hay update) một phần tử đã tồn tại trong một cấu trúc dữ liệu.
* Giải thuật Xóa: Giải thuật để xóa một phần tử đang tồn tại từ một cấu trúc dữ liệu.

Đặc điểm của giải thuật

Không phải tất cả các thủ tục có thể được gọi là một giải thuật. Một giải thuật nên có các đặc điểm sau:

* Tính xác định: Giải thuật nên rõ ràng và không mơ hồ. Mỗi một giai đoạn (hay mỗi bước) nên rõ ràng và chỉ mang một mục đích nhất định.
* Dữ liệu đầu vào xác định: Một giải thuật nên có 0 hoặc nhiều hơn dữ liệu đầu vào đã xác định.
* Kết quả đầu ra: Một giải thuật nên có một hoặc nhiều dữ liệu đầu ra đã xác định, và nên kết nối với kiểu kết quả bạn mong muốn.
* Tính dừng: Các giải thuật phải kết thúc sau một số hữu hạn các bước.
* Tính hiệu quả: Một giải thuật nên là có thể thi hành được với các nguồn có sẵn, tức là có khả năng giải quyết hiệu quả vấn đề trong điều kiện thời gian và tài nguyên cho phép.
* Tính phổ biến: Một giải thuật có tính phổ biến nếu giải thuật này có thể giải quyết được một lớp các vấn đề tương tự.
* Độc lập: Một giải thuật nên có các chỉ thị độc lập với bất kỳ phần code lập trình nào.

**Cách viết một giải thuật ?**

Bạn đừng tìm, bởi vì sẽ không có bất kỳ tiêu chuẩn nào cho trước để viết các giải thuật. Như bạn đã biết, các ngôn ngữ lập trình đều có các vòng lặp (do, for, while) và các lệnh điều khiển luồng (if-else), … Bạn có thể sử dụng những lệnh này để viết một giải thuật.

Chúng ta viết các giải thuật theo cách thức là theo từng bước một. Viết giải thuật là một tiến trình và được thực thi sau khi bạn đã định vị rõ ràng vấn đề cần giải quyết. Từ việc định vị vấn đề, chúng ta sẽ thiết kế ra giải pháp để giải quyết vấn đề đó và sau đó là viết giải thuật.

Ví dụ viết giải thuật

Bạn theo dõi ví dụ minh họa dưới đây để thấy rõ các bước và cách viết một giải thuật. Tất nhiên là ví dụ dưới đây là khá đơn giản vì đây chỉ là ví dụ minh họa mở đầu cho cách viết giải thuật thôi, nên mình nghĩ càng đơn giản sẽ càng tốt.

Bài toán: Thiết kế một giải thuật để cộng hai số và hiển thị kết quả.

<b>Bước 1</b>: Bắt đầu

<b>Bước 2</b>: Khai báo ba số <b>a</b>, <b>b</b> & <b>c</b>

<b>Bước 3</b>: Định nghĩa các giá trị của <b>a</b> & <b>b</b>

<b>Bước 4</b>: Cộng các giá trị của <b>a</b> & <b>b</b>

<b>Bước 5</b>: Lưu trữ kết quả của <u>Bước 4</u> vào biến <b>c</b>

<b>Bước 6</b>: In biến <b>c</b>

<b>Bước 7</b>: Kết thúc

Các giải thuật nói cho lập trình viên cách để viết code. Ngoài ra, bạn cũng có thể viết một giải thuật cho bài toán trên như sau:

<b>Bước 1</b>: Bắt đầu

<b>Bước 2</b>: Lấy giá trị của <b>a</b> & <b>b</b>

<b>Bước 3</b>: c ← a + b

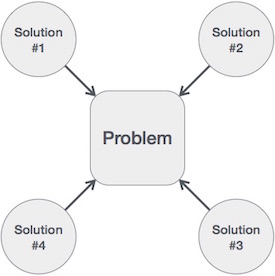
<b>Bước 4</b>: Hiển thị c

<b>Bước 5</b>: Kết thúc

Trong khi thiết kế và phân tích các giải thuật, thường thì phương thức thứ hai được sử dụng để miêu tả một giải thuật. Cách thứ hai này giúp dễ dàng phân tích giải thuật khi đã bỏ qua các phần định nghĩa không cần thiết. Nhìn vào cách thứ hai này, người ta có thể biết các phép tính nào đang được sử dụng và cách tiến trình thực thi.

Tất nhiên, việc viết tên các bước là tùy ý.

Chúng ta viết một giải thuật để tìm giải pháp để xử lý một bài toán nào đó. Một bài toán có thể được giải theo nhiều cách khác nhau.



Do đó, một bài toán có thể sẽ có nhiều lời giải. Vậy lời giải nào sẽ là thích hợp nhất cho bài toán đó. Mời bạn tiếp tục theo dõi.

Phân tích giải thuật

Hiệu quả của một giải thuật có thể được phân tích dựa trên 2 góc độ: trước khi triển khai và sau khi triển khai:

* Phân tích lý thuyết: Có thể coi đây là phân tích chỉ dựa trên lý thuyết. Hiệu quả của giải thuật được đánh giá bằng việc giả sử rằng tất cả các yếu tố khác (ví dụ: tốc độ vi xử lý, …) là hằng số và không ảnh hưởng tới sự triển khai giải thuật.
* Phân tích tiệm cận: Việc phân tích giải thuật này được tiến hành sau khi đã tiến hành trên một ngôn ngữ lập trình nào đó. Sau khi chạy và kiểm tra đo lường các thông số liên quan thì hiệu quả của giải thuật dựa trên các thông số như thời gian chạy, thời gian thực thi, lượng bộ nhớ cần dùng, …

Chương này chúng ta sẽ tìm hiểu phân tích lý thuyết. Còn phân tích tiệm cận chúng ta sẽ cùng tìm hiểu ở chương tiếp theo.

**Độ phức tạp giải thuật (Algorithm Complexity)**

Về bản chất, độ phức tạp giải thuật là một hàm ước lượng (có thể không chính xác) số phép tính mà giải thuật cần thực hiện (từ đó dễ dàng suy ra thời gian thực hiện của giải thuật) đối với bộ dữ liệu đầu vào (Input) có kích thước n. Trong đó, n có thể là số phần tử của mảng trong trường hợp bài toán sắp xếp hoặc tìm kiếm, hoặc có thể là độ lớn của số trong bài toán kiểm tra số nguyên tố, …

Giả sử X là một giải thuật và n là kích cỡ của dữ liệu đầu vào. Thời gian và lượng bộ nhớ được sử dụng bởi giải thuật X là hai nhân tố chính quyết định hiệu quả của giải thuật X:

* Nhân tố thời gian: Thời gian được đánh giá bằng việc tính số phép tính chính (chẳng hạn như các phép so sánh trong thuật toán sắp xếp).
* Nhân tố bộ nhớ: Lượng bộ nhớ được đánh giá bằng việc tính lượng bộ nhớ tối đa mà giải thuật cần sử dụng.

Độ phức tạp của một giải thuật (một hàm f(n)) cung cấp mối quan hệ giữa thời gian chạy và/hoặc lượng bộ nhớ cần được sử dụng bởi giải thuật.

Độ phức tạp bộ nhớ (Space complexity) trong phân tích giải thuật

Nhân tố bộ nhớ của một giải thuật biểu diễn lượng bộ nhớ mà một giải thuật cần dùng trong vòng đời của giải thuật. Lượng bộ nhớ (giả sử gọi là S(P)) mà một giải thuật cần sử dụng là tổng của hai thành phần sau:

* Phần cố định (giả sử gọi là C) là lượng bộ nhớ cần thiết để lưu giữ dữ liệu và các biến nào đó (phần này độc lập với kích cỡ của vấn đề). Ví dụ: các biến và các hằng đơn giản, …
* Phần biến đổi (giả sử gọi là SP(I)) là lượng bộ nhớ cần thiết bởi các biến, có kích cỡ phụ thuộc vào kích cỡ của vấn đề. Ví dụ: cấp phát bộ nhớ động, cấp phát bộ nhớ đệ qui, …

Từ trên, ta sẽ có <b>S(P) = C + SP(I)</b>. Bạn theo dõi ví dụ đơn giản sau:

Giải thuật: tìm tổng hai số A, B

Step 1 - Bắt đầu

Step 2 - C ← A + B + 10

Step 3 - Kết thúc

Ở đây chúng ta có ba biến A, B và C và một hằng số. Do đó: <b>S(P) = 1+3</b>.

Bây giờ, lượng bộ nhớ sẽ phụ thuộc vào kiểu dữ liệu của các biến và hằng đã cho và sẽ bằng tích của tổng trên với bộ nhớ cho kiểu dữ liệu tương ứng.

**Độ phức tạp thời gian (Time Complexity) trong phân tích giải thuật**

Nhân tố thời gian của một giải thuật biểu diễn lượng thời gian chạy cần thiết từ khi bắt đầu cho đến khi kết thúc một giải thuật. Thời gian yêu cầu có thể được biểu diễn bởi một hàm T(n), trong đó T(n) có thể được đánh giá như là số các bước.

Ví dụ, phép cộng hai số nguyên n-bit sẽ có n bước. Do đó, tổng thời gian tính toán sẽ là T(n) = c\*n, trong đó c là thời gian để thực hiện phép cộng hai bit. Ở đây, chúng ta xem xét hàm T(n) tăng tuyến tính khi kích cỡ dữ liệu đầu vào tăng lên.