**<a>Recursion – Phần 2</a>**

<p>Quay lại câu chuyện đọc sách, việc đọc sách đơn giản là việc kêu nhân viên thư viện mang ra cuốn sách cần đọc và đọc hết cuốn đó, sau đó tiếp tục kêu họ mang ra cuốn khác và lại ngồi đọc cho hết, quá trình cứ tiếp diễn như vậy cho đến khi bạn đọc xong quyển cuối cùng. Cách làm trên có thể hiểu giống như phương thức cuốn gói, đọc cuốn nào xong cuốn đó. Nhưng giả sử trong trường hợp nhân viên thư viện không đủ kiên nhẫn để đợi bạn đọc xong một cuốn sách rồi mới mang tiếp cuốn khác ra đặt trên bàn cho bạn đọc, để cho tiện họ muốn lấy từng cuốn sách mang ra bàn cho bạn đến khi bàn của bạn có đủ 5 cuốn sách thì bạn mới bắt đầu việc đọc. Đây chính là cách hoạt động của Recursion bạn sẽ gọi, gọi và gọi… cho đến khi có đủ 5 cuốn sách trên bàn thì bắt đầu đọc. Ở đây có 2 yếu tố cần quan tâm, hoạt động gọi là giống nhau nhưng nội dung mỗi lần gọi (input) là khác nhau giống như mỗi lần bạn gọi nhân viên mang một cuốn sách với nhan đề khác nhau, yếu tố quan trọng thứ hai phải quan tâm là khi nào thì ngưng gọi, nếu không có điều kiện ngưng gọi thì bạn sẽ kêu nhân viên thư viện mang hết tất cả sách trong thư viện cho bạn đọc điều đó là không khả thi, điều kiện ngưng gọi này trong recursion được đặt tên là Base case</p>

<p>Quay lại ví dụ tính giai thừa của số 5, lần này mình sẽ dùng phương thức recrusion để giải quyết bài toán</p>

<p>function **factorial(num)** {</p>

<p> if (num == 1) return 1; </p>

<p> return num \* **factorial(num-1)** </p>

<p>}</p>

<p>Hàm factorial(num) được gọi lại chính nó, cứ sau mỗi lần gọi đầu vào của nó sẽ giảm đi một và khi đầu vào bằng 1 thì hàm trả về bằng 1, giả sử cho đầu vào num = 5, kết quả trả về sẽ là</p>

<p>Lần gọi 1: Return 5 \* factorial(5-1) </p>

<p>Lần gọi 2: Return 5 \* 4 \* factorial(4-1) </p>

<p>Lần gọi 3: Return 5 \* 4 \* 3 \* factorial(3-1) </p>

<p>Lần gọi 4: Return 5 \* 4 \* 3 \* 2 \* factorial(2-1) = Return 5 \* 4 \* 3 \* 2 \* 1 = 120</p>

<p>Ở lần gọi cuối cùng factorial(2-1) chạm đến base case sẽ trả giả trị là 1</p>

<p>Hãy xem recursion hoạt động ra sao trên call stack</p>

<p>Đầu tiên Call stack sẽ thực hiện việc gọi hàm factorial(5) để thực hiện tính giai thừa cho 5, phần Call stack bên dưới phía phải bước hình hiển thị Call stack dòng 6</p>

**<div class="a">**

**<img src="pics/1.png " alt="Trulli" width="600" height="300">**

**</div>**

<p>Để thực hiện Stack dòng 6 cần thực hiện stack dòng 2 trước tiên bằng việc kiểm tra num == 1. Lúc này stack dòng 2 sẽ chồng lên stack dòng 6 ở phần Call stack (hiển thị bên phải phía dưới hình) </p>

**<div class="a">**

**<img src="pics/2.png " alt="Trulli" width="600" height="300">**

**</div>**

<p>Xong stack dòng 2 tiến hành xóa bỏ ra hỏi Call stack, và chuyển qua stack dòng 3</p>

**<div class="a">**

**<img src="pics/3.png " alt="Trulli" width="600" height="300">**

**</div>**

<p>Stack dòng 3 sẽ gọi lại function factorial(num), function này sẽ bắt đầu kiểm tra giá trị num = 1 ở stack dòng 2 nên nó sẽ quay lại stack dòng 2</p>

**<div class="a">**

**<img src="pics/4.png " alt="Trulli" width="600" height="300">**

**</div>**

<p>Sau khi stack dòng 2 hoàn thành, nó sẽ được gỡ ra khỏi Call stack, tiếp tục chuyển qua stack dòng 3</p>

**<div class="a">**

**<img src="pics/5.png " alt="Trulli" width="600" height="300">**

**</div>**

<p>Stack dòng 3 sẽ gọi lại hàm factorial(num), function này sẽ bắt đầu kiểm tra giá trị num == 1 ở stack dòng 2 nên nó sẽ quay lại stack dòng 2</p>

**<div class="a">**

**<img src="pics/6.png " alt="Trulli" width="600" height="300">**

**</div>**

<p>Lặp lại quá trình trên cho đến khi giá trị đầu vào num = 1, lúc này trên phần Call stack phía bên phải phía dưới hình sẽ thấy stack dòng 3 đã được gọi đến 4 lần và đang chờ xử lý, stack 2 đang được xử lý tại thời điểm hiện tại thỏa mãn điều kiện num = 1 nó sẽ trả giá trị bằng 1</p>

**<div class="a">**

**<img src="pics/7.png " alt="Trulli" width="600" height="300">**

**</div>**

<p>Sau khi xử lý xong stack dòng 2 quay về xử lý tiếp stack dòng 3 đang chờ được xử lý</p>

**<div class="a">**

**<img src="pics/8.png " alt="Trulli" width="600" height="300">**

**</div>**

<p>Tổng cộng cần phải xử lý hết 4 stack dòng 3 đang chờ xử lý tương ứng với các giá trị là tích số của 2\*3\*4\*5, sau khi xử lý hết 4 stack dòng 3 này đồng nghĩa với nhiệm vụ tính giai thừa ở stack dòng 6 được hoàn thành, quá trình tính toán kết thúc</p>

<p>Qua việc debug để hiểu rõ hơn Recursion hoạt động như thế nào có thể thấy phần Call stack trong recursion sẽ được chồng lên nhau sau mỗi lần hàm được gọi lại, cụ thể quay lại hàm tính giai thừa phía trên nếu tính giai thừa của 5 sẽ có 4 stack chồng lên nhau để đợi xử lý, cụ thể 4 stack đó là nhiệm vụ nhân từng số trong mỗi stack với nhau 2\*3\*4\*5. Nếu tăng số tính giai thừa lên tới 1000 thì sẽ có 999 stack chồng lên nhau cần xử lý, đó là tích số từ 2 đến 1000</p>

<p>Qua phần phân tích trên có thể thấy việc sử dụng Recursion sau mỗi lần gọi hàm sẽ tạo các Stack chồng lên nhau chờ được xử lý, nếu quy mô đầu vào càng lớn thì sẽ có càng nhiều dòng stack trong Call stack chờ được xử lý, việc này sẽ ngốn nhiều ram của máy tính hơn. Giống như việc bạn nhờ nhân viên thư viện mang hết các quyển sách ra bàn đọc rồi mới bắt đầu đọc từng cuốn một, sẽ ra sao nếu bàn của bạn không đủ diện tích để đặt nhiều quyển sách lên? đó là một vấn đề cần phải cân nhắc. Trong thực tế, có những trường hợp mà recursion phát huy được thế mạnh tuyệt đối trong tốc độ xử lý so với các cách tiếp cận khác, ví dụ như việc sort một dãy số gồm rất nhiều con số, việc sử dụng recursion trong phương pháp Merge sort hay Pivot sort sẽ cho tốc độ xử lý nhanh hơn rất nhiều so với phương pháp Bubble sort</p>