**Matrix-chain Multiplication**

**Using Dynamic Programming**

**Tạ Đình Sơn Tùng - HE160762**

1. Đặt vấn đề :

* Chúng ta đã rất quen thuộc với phép nhân 2 ma trận. Khi nhân ma trận A có chiều a\*b với ma trận B có chiều b\*c, ta sẽ được ma trận mới C có chiều là a\*c và số phép nhân cần để hoàn thành phép tính là a\*b\*c.
* Tuy nhiên khi nhân chuỗi nhiều ma trận với nhau, ta không chỉ có 1 mà có nhiều cách để thực hiện phép nhân chuỗi ma trận.
* Ví dụ: A\*B\*C\*D có nhiều cách để tính như (A\*B)\*(C\*D), A\*(B\*C)\*D,(A\*(B\*C))\*D,...
* Tại sao ta không nhân bình thường theo thứ tự mà phải tìm cách khác để nhân? Do mỗi cách tính toán sẽ cần số phép nhân khác nhau để hoàn thành.
* Ví dụ: Ta có 4 ma trận A(20\*50), B(50\*100), C(100\*5) , D(5\*10)

+ Nếu tính theo thứ tự bình thường, ta sẽ cần :

20\*50\*100 + 20\*100\*5 + 20\*5\*10 = 111.000 phép nhân để hoàn thành bài toán

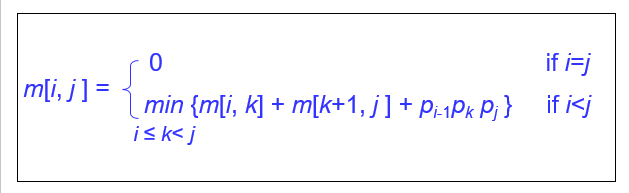
+ Nếu tính theo thứ tự A\*(B\*C)\*D, ta chỉ cần:

50\*100\*5 + 20\*50\*5 + 20\*5\*10 = 31.000 phép nhân để hoàn thành bài toán

* Như đã thấy, ta có thể giảm được khối lượng phép tính nếu có thể sắp xếp thứ tự nhân các chuỗi ma trận 1 cách hợp lý
* Vì vậy, em đã quyết định viết 1 chương trình có thể nhận đầu vào là chiều của các chuỗi ma trận, sau đó chương trình sẽ tính toán và trả về cho ta thứ tự tính toán các chuỗi để đạt đuọc số phép nhân cần phải thực hiện ít nhất có thể.

1. Phương pháp thực hiện:

* Bài toán sẽ được giải quyết bằng cách sử dụng dynamic programming.Để tìm được giá trị tối ưu cho bài toán, ta sẽ chia nhỏ bài toàn thành các bài toán con rồi đi tìm giá trị tối ưu cho các bài toán con đó, sau khi tổng hợp các giá trị tối ưu của các bài toán con, ta sẽ tìm được lời giải cho bài toán lớn.
* Cụ thể hơn, để tính giá trị tối ưu cho chuỗi ma trận A(1) đến A(n), ta có nhiều cách khác nhau : (giá trị tối ưu chuỗi A(1) đến A(k)) cộng với (giá trị tối ưu chuỗi A(k+1) đến A(n)) cộng với giá trị tối ưu khi nhân 2 ma trận A(1)...A(k) với A(k+1)...A(n). Bằng cách tính toán từng giá trị k, ta sẽ tìm được giá trị k cần để đạt được giá trị tối ưu, từ đó chia bài toán thành 2 bài toán nhỏ hơn rồi tiếp tục thực hiện tính k.
* Sau khi tính toán tất cả a của bài toán nhân chuỗi ma trận:



* Dynamic programming sẽ có 2 cách tiếp cận, đó là:

+ Top-down

+ Bottom-up

* Bottom-up:Ta đi từ dưới lên trên, bắt đầu từ tối ưu các bài toán căn bản, sau nó gộp chúng lại thành các bài toán lớn hơn, cứ làm như vậy cho đến khi giải quyết được bài toán được đề ra.
* Top-down:Ta sẽ đi từ trên xuống dưới, tách bài toán lớn thành các phần nhỏ hơn rồi tách phần nhỏ thành các phần nhỏ hơn nữa cho đến khi thành bài toán nhỏ nhất.

+ Áp dụng vào bài toán này, ta sẽ bắt đầu tách tìm giá trị tối ưu để nhân n ma trận thành tìm giá trị tối ưu cho n-1 ma trận rồi cứ thế tách cho đến khi vấn đề trở thành tìm giá trị tối ưu khi nhân 2 ma trận(đơn giản nhất)

Recursion tree for the above approach

+ Tuy nhiên khi sử dụng phương pháp này,ta thường phải tính toán lại 1 giá trị nhiều lần.Ví dụ như hình trên, các giá trị trong ô màu xanh là các giá trị phải tính toán lại nhiều lần (ví dụ như giá trị 2.2 đã được tính đi tính lại đến 4 lần ).Điều này khiến cho việc tính toán trở nên tốn thời gian hơn rất nhiều.

+ Để khắc phục vấn đề này, ta sẽ sử dụng phương pháp Memoization, nghĩa là khi đã tính toán được 1 giá trị bất kỳ, ta sẽ lưu lại kết quả của tính toán đó để sau này nếu phải tính lại ta sẽ lấy lại từ bộ nhớ chứ không cần phải tính toán lại từ đầu. Điều này giúp giảm độ phức tạp thuật toán của ta từ O(2n) xuống còn O(n3).

* Cách áp dụng chi tiết của mỗi thuật toán vào bài toán này sẽ được nói chi tiết hơn ở phần demo code.

1. Các hàm có trong chương trình:

* Các hàm trong chương trình sẽ được phân thành 2 loại chính:

+ Các hàm liên quan đến thuật toán

+ Các hàm chức năng và giao diện của chương trình

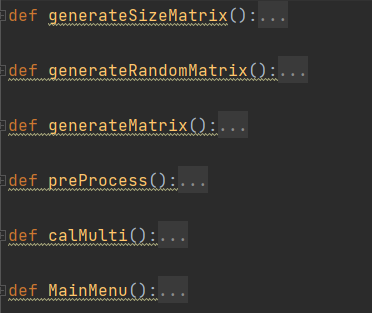
* Các hàm liên quan đến thuật toán:

+ Các hàm topDown, memoization,CalUsingTopDown để giải quyết bài toán theo hướng top-down. Các hàm bottomUp,CalUsingBottomUp để giải quyết bài toán theo hướng bottom-up

+ Hàm MatrixNormalMulti dùng để tính số phép nhân cần dùng để giải quyết bài toán nhân chuỗi ma trận theo thứ tự bình thường từ trái sang phải, ta sẽ sử dụng hàm này để có thể so sánh giá trị bình thường với giá trị đã được tối ưu bởi chương trình

+ Hàm parenthesis dùng để vẽ các dấu ngoặc đơn để biểu thị thứ tự nhân các chuỗi ma trận để được kết quả tối ưu.

* Các hàm chức năng và giao diện của chương trình:



+ Hàm MainMenu sẽ cung cấp giao diện đơn giản nhất cho người dùng.Người dùng sẽ được lựa chọn 2 phương án là tạo mảng input hoặc tính toán giá trị tối ưu của mảng input

+ Hàm generateMatrix sẽ giúp người dùng tạo mảng input theo 2 phương án , người dùng có thể lựa chọn tạo mảng random bằng hàm generateRandomMatrix hoặc có thể tạo mảng có giá trị do mình lựa chọn bằng hàm generateSizeMatrix

+Hàm calMulti sẽ cho người dùng 2 lựa chọn , tính toán theo phương pháp bottom-up hoặc top-down

+ Hàm preProcess sẽ xử lý 1 vài dữ liệu đầu vào để thuật toán có thể chạy 1 cách trơn tru nhất

* Dưới đây là ví dụ về input và output tương ứng:

