**Khái niệm:**

**Quy hoạch động là gì ?**

QHĐ là một kỹ thuật giải quyết một vấn đề phức tạp nhất định bằng cách chia nó thành các bài toán con và sử dụng lời gi ải của các bài toán con để tìm lời giải cho các bài toán ban đầu.

Giải thuật **Qui hoạch động** giống như giải thuật **chia để trị** trong việc chia nhỏ bài toán thành các bài toán con nhỏ hơn và sau đó thành các bài toán con nhỏ hơn nữa có thể.

Nhưng không giống chia để trị, các bài toán con này không được giải một cách độc lập. Thay vào đó, kết quả của các bài toán con này được **lưu vào bộ nhớ** (thường là một mảng), và sau đó lấy lời giải của bài toán con ở trong mảng đã tính trước để giải bài toán lớn. Việc lưu lại lời giải vào bộ nhớ khiến cho ta không phải tính lại lời giải của các bài toán con mỗi khi cần, do đó, tiết kiệm được thời gian tính toán.

Để các bạn dễ hình dung về **qui hoạch động** và thấy rõ hơn về sự khác biệt so với **Chia để trị** chúng ta quan sát cách thức hoạt động của 2 thuật toán này.

Ví dụ : Bài toán kinh điển **Fibonaci**. Tính số fibonaci thứ n, F(n).

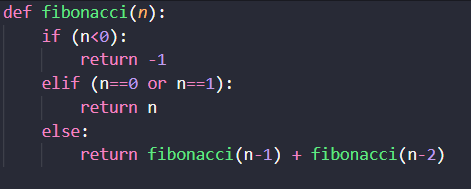
F(0) = 0 F(1) = 1

F(n) = F(n-2) + F(n-1) với n >1

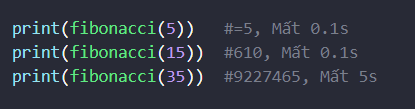
F(2) = 1, F(3) = 2, F(4) = 3, F(5) = 5, F(6) = 8 …

\*Chia để trị:

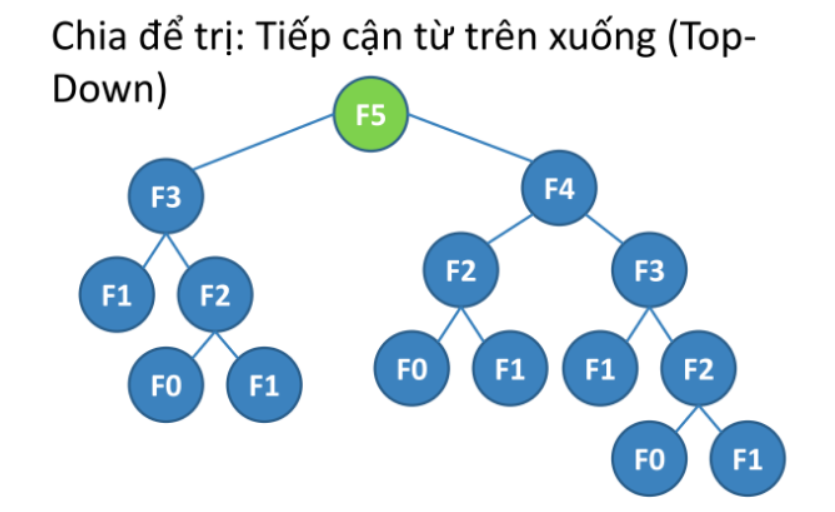
Code:



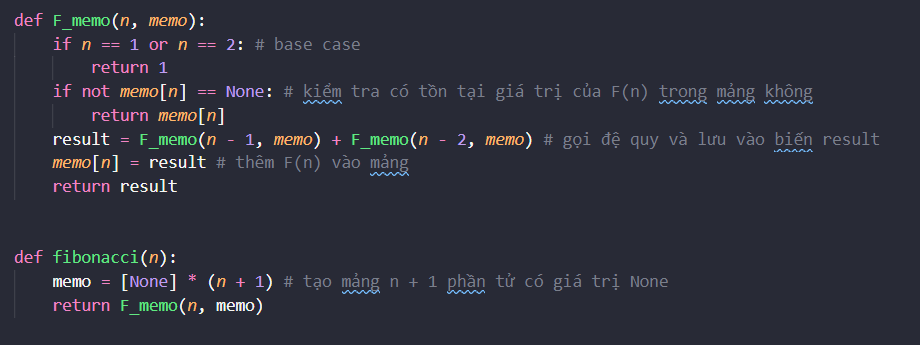
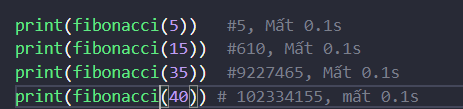
Kết quả:



Đồ thị đệ quy của F(5):

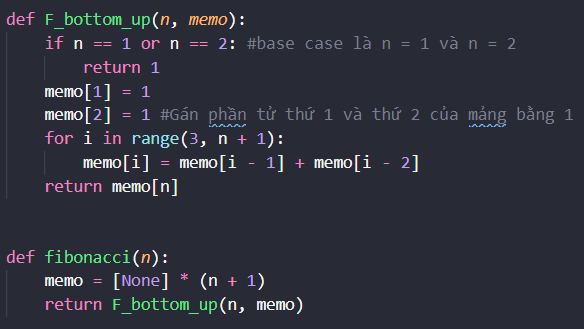


**-> Ta thấy được ở giải thuật chia để trị, khi gọi đệ quy ta phải tính lại các bài toán con rất nhiều lần dẫn đến mất rất nhiều thời gian**

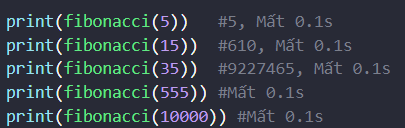
\*Quy hoạch động:  
+Top-down:  
Code:   
  
kết quả:  
  
Thời gian thực thi nhanh hơn chia để trị. Tuy nhiên với giá trị lớn (vd n>1000) thì ct sẽ báo lỗi:  
  
-> lỗi này sẽ xảy ra khi thực hiện đệ quy quá sâu.   
-> Để tối ưu hơn cho đoạn chương trình thì chúng ta có phương pháp Bottom-up.  
  
+Bottom-up:

Code:

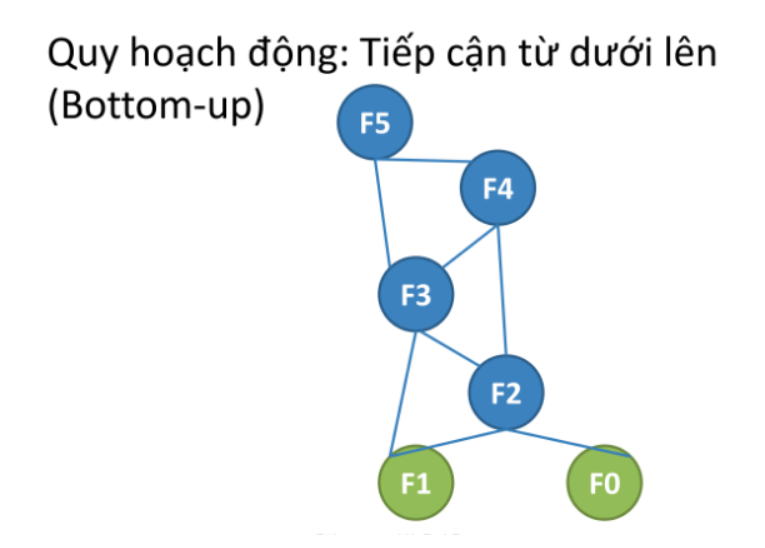
dùng một mảng lưu lại các giá trị của hàm F(n). Base case là n = 1 và n = 2, ta sẽ gán phần tử thứ nhất và hai của mảng bằng 1. Tiếp theo, chúng ta cho một biến chạy từ 3 đến n, phần tử thứ i của mảng sẽ bằng hai phần tử trước đó cộng lại, tức là phần tử i – 1 và i – 2. Kết quả cuối cùng, phần tử thứ n chính là kết quả F(n) cần tìm



Kết quả:



Đồ thị QHĐ (bottom-up) của F(5):

  
**->Ta sử dụng một mảng để lưu kết quả của bài toán trước, vì vậy việc tính toán trở nên đơn giản hơn rất nhiều**

**+ QHĐ là 1 giải thuật dùng để giải các bài toán bằng cách kết hợp các lời giải của các bài toán con của bài toán đang xét.**

**+ QHĐ làm giảm độ phức tạp, giảm thời gian để giải quyết bài toán.**

**+ QHĐ có 2 dạng tiếp cận:**

**\*Top- down: Bài toán được chia thành bài toán con, các bài toán con này được giải và lời giải được ghi nhớ để dùng lại chúng. Đệ quy và lưu trữ kết hợp với nhau.  
   
\* Bottom -up: Tất cá bài toán con có thể cần đến đều được giải trước, sau đó được dùng để xây dựng lời giải cho các bài toán lớn hơn.**

**Khi nào thì dùng?**

“Khi nào thì chúng ta cần đến quy hoạch động? Đó là một câu hỏi rất khó trả lời. Không có một công thức nào cho các bài toán như vậy.

Tuy nhiên, có một số tính chất của bài toán mà bạn có thể nghĩ đến quy hoạch động. Hai tính chất nổi bật nhất trong số chúng:

* Bài toán có các bài toán con gối nhau.
* Bài toán có cấu trúc con tối ưu.

Thường thì một bài toán có đủ cả hai tính chất này, chúng ta có thể dùng quy hoạch động được. “

**“cụ thể hơn về những tính chất này thì nhóm sau sẽ trình bày”**

**Dạng thuật toán phổ quát**

Có 3 bước:  
1. Chia bài toán thành các bài toán con nhỏ hơn.  
2. Giải các bài toán con một cách tối ưu.  
3. Sử dụng các kết quả tối ưu đó để xây dựng một lời giải tối ưu cho bài toán ban đầu.

Giải thích:

1/ Chia bài toán cần giải thành những bài toán con nhỏ hơn đến mức có thể giải trực tiếp được hay không? Nếu giải được chuyển sang bước giải bài toán con.

2/ Lưu trữ lời giải của các bài toán con vào một bảng (mảng 1 chiều hoặc 2 3 chiều) để sử dụng về sau.

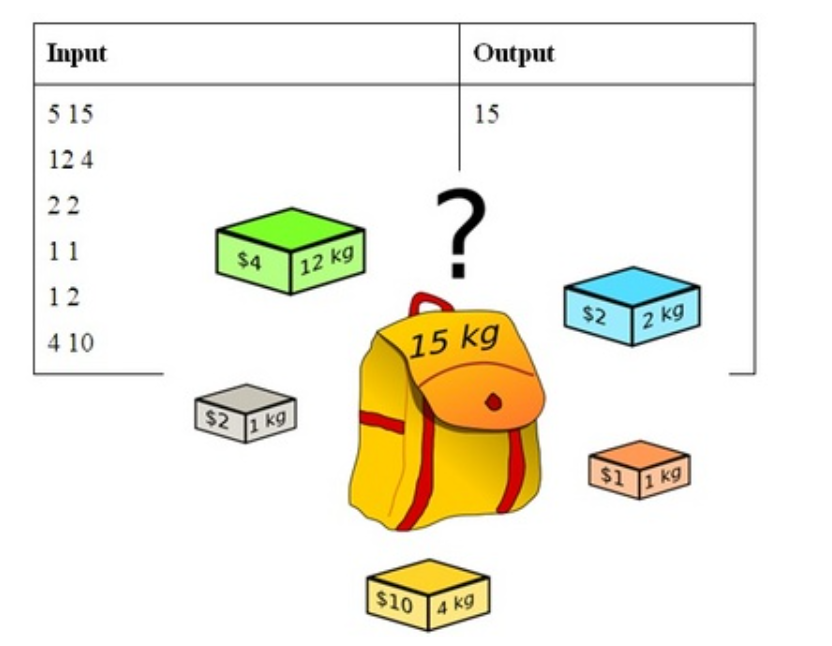
3/ Tổng hợp lời giải:   
 Tổng hợp lời giải các bài toán con kích thước nhỏ hơn thành lời giải bài toán lớn hơn.

Tiếp tục cho đến khi thu được lời giải của bài toán xuất phát (là bài toán con có kích thước lớn nhất)

**->Để làm rõ hơn chi tiết hơn những những bước này, thì chúng ta giải bài toán cái túi**

**Bài toán cái túi**

Một kẻ trộm đột nhập vào một cửa hiệu tìm thấy có ***n*** mặt hàng có trọng lượng và giá trị khác nhau nhưng hắn chỉ mang theo một cái túi có sức chứa có trọng lượng tối đa ***W.*** Bài toán cái túi nhằm tìm một tổ hợp các mặt hàng mà kẻ trộm nên bỏ vào cái túi để đạt một giá trị cao nhất với những món hàng hắn mang đi.

  
 Mày làm hình khác vs cái input output trên ytb á

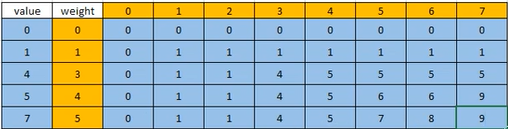
Cụ thể:

Có ***n***đồ vật, đồ vật i có trọng lượng ***weight*** và giá trị ***value***

Với i = 1, 2 , … , n

Tìm chất để đưa những đồ vật này vào cái túi có dung lượng ***W*** sao cho tổng trọng lượng của các đồ vật được bỏ vào túi không quá ***W***, đồng thời tổng giá trị của chúng là lớn nhất.

**Tìm lời giải:**

* Phân rã: Với các giá trị i(1..n) và w (0..W). Gọi K[i,w] là tổng giá trị lớn nhất có thể chọn trong i đồ vật (từ 1 đến n) với trọng lượng tối đa của túi là W. Khi đó K[n,W] là giá trị lớn nhất mang đi được.
* Giải bài toán con nhỏ nhất:  
  **K[0,w] = 0** với mọi w, và **K[i,0]** với mọi i
* Tổng hợp:  
  - Tại thời điểm đang xét vật i, để có giá trị lớn nhất sẽ có 2 khả năng:  
  + Nếu không chọn vật thứ i thì giá trị lớn nhất có thể chọn trong số các vật {1, 2, …, i-1}   
    **K[i,w] := K[i-1,w]**+ Nếu có chọn vật thứ i (phải thỏa điều kiện weigt[i] < w), thì **K[i,w]** bằng giá trị vật thứ i là **value[i]** cộng với giá trị lớn nhất có thể tìm được bằng cách chọn các vật {1, 2, …, i-1} với giới hạn trọng lượng **w – weight[i],** tức là:  
   **K[i,w] := value[i]** **+ K[i-1, w – weight[i]]  
  -** Chúng ta sẽ phải xem xét nếu chọn vật i hay không chọn vật i thì sẽ tốt hơn. Từ đó ta có công thức truy hồi:   
   **K[i,w] = max (K[i-1,w], value [i] + K[i-1, w – weight[i]])**
* Bảng mô tả:****
* Code   
  