**1.Bài toán “ Tính số Fibonacci thứ n ”**

* Để tính Fn (dãy số Fibonacci)

F0 = F1 = 1

Fn = Fn-1 + Fn-2 (n ≥ 2)

Ta có thể giải theo phương pháp top-down như sau:

int F(int n)

{

    if (n == 0 || n == 1)

        return 1;

   else

       return F(n-1) + F(n-2);

}

* Phương pháp này hạn chế khi n lớn, có thể chạy chậm rất chậm. Thậm chí có thể bị tràn stack khi n đủ lớn. Do tính trùng lại rất nhiều lần của các bài toán con nên phương pháp này khá chậm.

**ví dụ:**

F4

/        \

F3            F2

/      \          /    \

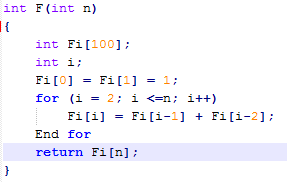
F2      **F1**     **F1**    **F0**

/    \

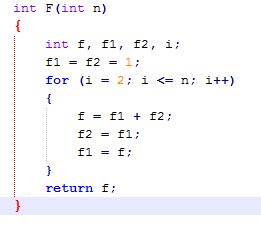
**F1**    **F0**

    Trong ví dụ trên, ta thấy ta phải tính 2 lần F2, 3 lần F1 và 2 lần F0.

    Bài toán trên được giải theo phương pháp quy hoạch động như sau:



    Ta nhận thấy, có thể tiết kiệm không gian lưu trữ các kết quả trung gian trong quá trình tính toán bằng cách chỉ lưu lại 2 giá trị fi-1 và fi-2.



**2.Bài toán “Thời gian qua cầu”**

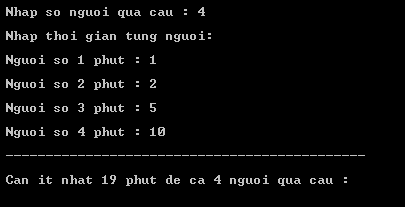
1. **Đề bài :**

An, Bình, Chung, Dung phải qua cầu khỉ vào buổi tối. Chỉ có một cái đèn pin. Tối đa là hai người đi cùng một lúc, và phải có đèn mới qua/lại được. Họ đi với tốc độ khác nhau. Nếu hai người cùng đi thì cả hai phải đi bằng tốc độ người đi chậm hơn.  
An cần 1 phút.  
Bình cần 2 phút.  
Chung cần 5 phút.  
Dung cần 10 phút.  
Dĩ nhiên là phải có người cầm đèn đi ngược lại để đón người kế qua nếu cần!  
Hỏi: ít nhất họ cần bao nhiêu phút để cả 4 qua được cầu?

1. **Phân tích**

2 người đi chung mà phụ thuộc vào tốc độ người đi chậm thì ta chọn ra người đi nhanh nhất để khi cầm đèn trở về là tối ưu nhất.  
Vì thế lượt đi sẽ là:  
An + Dung -> đi 10p và An về 1p;  
An + Chung -> đi 5p và An về 1p;  
An + Bình -> đi 2p.  
==> Tổng là 19p.  
Ta thấy số lần đi của An là 2 lần = n - 2 và những người khác mỗi người đi một lần nên Tổng là : Dung + Chung + Bình + An \* 2.

1. **VD**



**3.Bài toán “Dãy con tổng bằng S”**

1. Đề bài:

Cho 1 tập N số dương

Tìm 1 cách để biểu diễn số S dương bằng tổng của 1 số phần tử trong tập đã cho (nếu có)

1. **Ví dụ:**

Cho tập {1, 1, 5, 1}

S = 3

Phương án duy nhất là 1+1+1=3

* Input/Output:

nput:

N – số phần tử trong tập

A[1]…A[N]

S

Output:

Có biểu diễn được không?

Nếu có thì hiển thị 1 cách biểu diễn

1. Phân tích:

Nếu có thể biểu diễn S qua tập {A[1]…A[i]} thì đương nhiên cũng biểu diễn S qua tập {A[1]…A[i+1]..A[r]}

Nếu có thể biểu diễn S qua tập {A[1]…A[i]} và sử dụng A[i] thì phải biểu diễn được S-A[i] qua tập {A[1]…A[i-1]}

Bài toán này theo ta thấy thì không có yêu cầu tối ưu nào cả Tuy nhiên nó vẫn có thể giải theo phương pháp quy hoạch động. Tránh nhầm lẫn rằng bài toán không tối ưu thì không giải theo phương pháp quy hoạch động được (Ví dụ điển hình là bài toán Fibonacci cũng có thể giải theo phương pháp quy hoạch động).

Bài toán này không yêu cầu tối ưu, do đó ý nghĩa của bảng phương án cũng không phải là kết quả tối ưu mà chỉ là có hoặc không thể biểu diễn 1 số thành tổng 1 số phần tử.

* + **Công thức truy hồi:**

Gọi C[i][j] = khả năng biểu diễn số j thành tổng của 1 số phần tử trong tập {A[1]…A[i]}. Chú ý khác với các bài toán khác, C[i][j] nhận giá trị TRUE/FALSE.

Theo như phân tích, nếu C[i-1][j] = TRUE thì C[i][j] = TRUE hoặc C[i-1][j-A[i]ư = TRUE.

Các trường hợp còn lại, C[i][j] = FALSE.

* + **Cơ sở quy hoạch động:**

C[0][j] = khả năng biểu diễn số j thành tổng của 1 số phần tử trong tập rỗng = FALSE (trừ trường hợp j=0)

C[i][0] = khả năng biểu diễn số 0 = TRUE (bao gồm cả trường hợp i=0)

* + Tìm kết quả tối ưu

C[N][S] là khả năng biểu diễn số S theo tập {A[1]…A[N]}

Nếu C[N][S] = TRUE thì mới tiến hành truy vết

Nếu không thì không có cách biểu diễn nào.

Lưu ý là nếu đã có C[i][S] = TRUE thì không nhất thiết phải thực hiện hết toàn bộ quá trình xây dựng bảng mà có thể đánh dấu có thể biểu diễn được tổng S và thoát ngay khỏi quá trình đó.

* + **Truy vết**

Bắt đầu tại ô C[N][S] hoặc từ bất kỳ C[i][S] nào có giá trị = TRUE.

Kết thúc khi không còn tổng phải biểu diễn nữa tức j=0

1. Thiết kế:

