

2.8. Thuật toán qui hoạch động(Dynamic Programming)

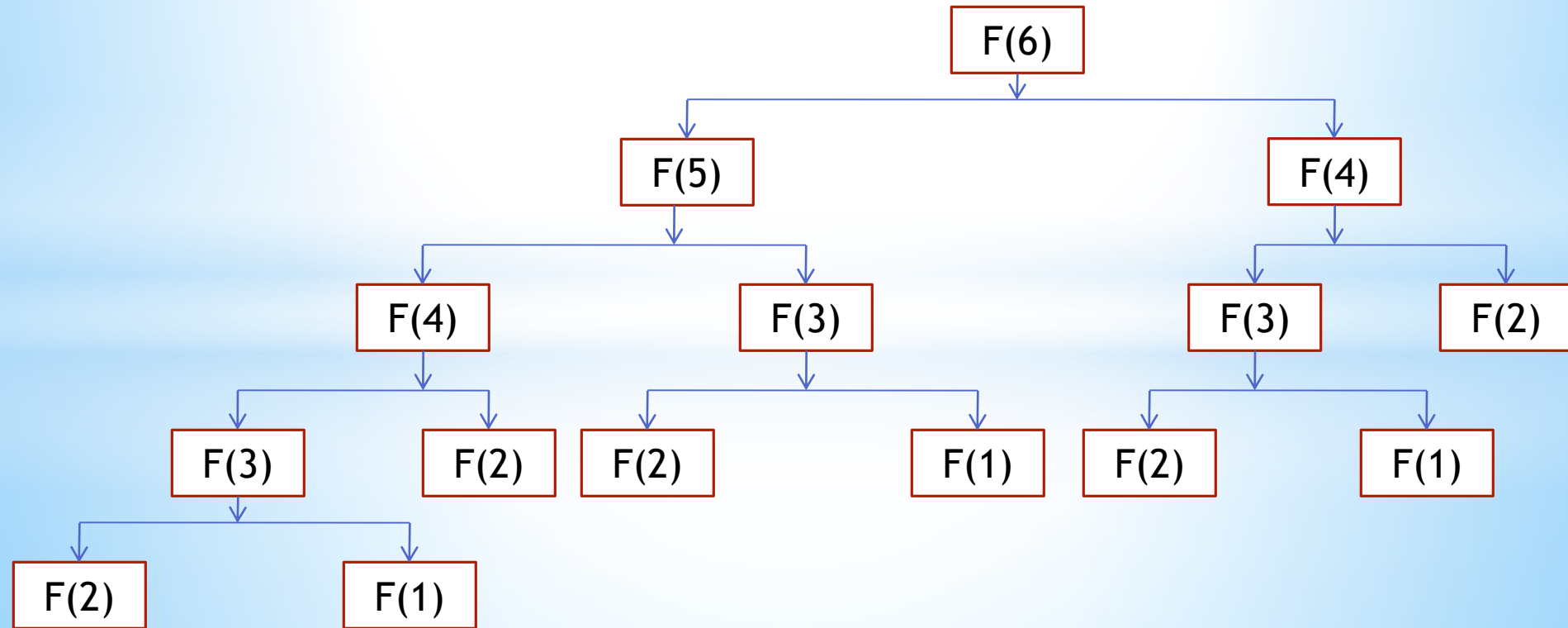
Phương pháp qui hoạch động dùng để giải lớp các bài toán thỏa mãn những điều kiện sau:

- Bài toán lớn cần giải có thể phân rã được thành nhiều bài toán con. Trong đó, sự phối hợp lời giải của các bài toán con cho ta lời giải của bài toán lớn. Bài toán con có lời giải đơn giản được gọi là cơ sở của qui hoạch động. Công thức phối hợp nghiệm của các bài toán con để có nghiệm của bài toán lớn được gọi là công thức truy hồi của qui hoạch động.
- Phải có đủ không gian vật lý lưu trữ lời giải các bài toán con (Bảng phương án của qui hoạch động). Vì qui hoạch động đi giải quyết tất cả các bài toán con, do vậy nếu ta không lưu trữ được lời giải các bài toán con thì không thể phối hợp được lời giải giữa các bài toán con.
- Quá trình giải quyết từ bài toán cơ sở (bài toán con) để tìm ra lời giải bài toán lớn phải được thực hiện sau hữu hạn bước dựa trên bảng phương án của qui hoạch động.

Ví dụ. Tìm số Fibonacci thứ n . Cho $F_1 = F_2 = 1$; $F_n = F_{n-1} + F_{n-2}$ với $n \geq 3$. Hãy tìm $F_6 = ?$.

```
#include <iostream.h>
int F( int i ) {
    if ( i < 3 ) return (1);
    else return (F(i-1) + F(i-2))
}
void main(void) {
    cout<<"\n F[6] "<<F(6);
}
```

```
#include <iostream.h>
int main()
{
    int F[100]; F[1]=1; F[2] =1;
    for (int i=3; i<=6; i++)
        F[i] = F[i-1] + F[i-2];
    cout<<"\n F[6]="<<F[6];
}
```



MỘT VÀI NHẬN XÉT

- 1. Cách giải thứ nhất:** Sử dụng lời gọi đệ qui đến $F(6)$. Để tính được $F(6)$ cách giải thứ nhất cần phải tính 1 lần $F(5)$, 2 lần $F(4)$, 3 lần $F(3)$, 5 lần $F(2)$, 3 lần $F(1)$. Với cách làm này, để tính $F(60)$ ta cần đến 10h.
- 2. Cách giải thứ hai (Qui hoạch động):** về bản chất cũng là đệ qui. Tuy nhiên phương pháp thực hiện theo các bước sau:
 - **Bước cơ sở** (*thường rất đơn giản*): Tính $F[1] = 1$, $F[2] = 1$.
 - **Công thức truy hồi** (*thường không đơn giản*): Lưu lời giải các bài toán con biết trước vào bảng phương án (*ở đây là mảng một chiều $F[100]$*). Sử dụng lời giải của bài toán con trước để tìm lời giải của bài toán con tiếp theo. Trong bài toán này là $F[i] = F[i-1] + F[i-2]$ với $n \geq 3$. Bằng cách lưu trữ vào bảng phương án, ta chỉ cần giải mỗi bài toán con một lần. Tuy vậy, cái giá phải trả là liệu ta có đủ không gian nhớ để lưu trữ lời giải các bài toán con hay không?
 - **Truy vết** (*thường đơn giản*): Đưa ra nghiệm của bài toán bằng việc truy lại dấu vết phối hợp lời giải các bài toán con để có được nghiệm của bài toán lớn. Trong bài toán này, vết của ta chính là $F[6]$ do vậy ta không cần phải làm gì thêm nữa.

Ví dụ. Tìm số các cách chia số tự nhiên n thành tổng các số tự nhiên nhỏ hơn n . Các cách chia là hoán vị của nhau chỉ được tính là một cách.

Lời giải. Gọi $F[m, v]$ là số cách phân tích số v thành tổng các số nguyên dương nhỏ hơn hoặc bằng m . Khi đó, bài toán trên tương ứng với bài toán tìm $F[n, n]$ (số các cách chia số n thành tổng các số nguyên dương nhỏ hơn hoặc bằng n).

Ta nhận thấy, số các cách chia số v thành tổng các số nhỏ hơn hoặc bằng m được chia thành hai loại:

- **Loại 1:** Số các cách phân tích số v thành tổng các số nguyên dương không chứa số m . Điều này có nghĩa số các cách phân tích Loại 1 là số các cách chia số v thành tổng các số nguyên dương nhỏ hơn hoặc bằng $m-1$. Như vậy số Loại 1 chính là $F[m-1, v]$.
- **Loại 2:** . Số các cách phân tích số v thành tổng các số nguyên dương chứa ít nhất một số m . Nếu ta xem sự xuất hiện của một hay nhiều số m trong phép phân tích v chỉ là 1, thì theo nguyên lý nhân số lượng các cách phân tích loại này chính là số cách phân tích số v thành tổng các số nhỏ hơn $m-v$ hay $F[m, v-m]$.

Trong trường hợp $m > v$ thì $F[m, v-m] = 0$ nên ta chỉ có các cách phân tích loại 1. Trong trường hợp $m \leq v$ thì ta có cả hai cách phân tích loại 1 và loại 2. Vì vậy:

- $F[m, v] = F[m-1, v]$ nếu $m > v$.
- $F[m, v] = F[m-1, v] + F[m, v-m]$ nếu $m \leq v$.

Bảng phương án: Sử dụng công thức truy hồi cho phép ta tính toán $F[m, v]$ thông qua $F[m-1, v]$ và $F[m, v-m]$. Tập các giá trị tính toán theo phương pháp lặp của hệ thức truy hồi được gọi là bảng phương án của bài toán. Ví dụ với $n=5$ ta sẽ có bảng phương án như sau:

- Dòng 0 ghi lại $F[0, v]$. Trong đó, $F[0,0] = 1$; $F(0,v) = 0$, với $v > 1$.
- Dựa vào dòng 0, các dòng tiếp theo trong bảng được tính toán như bảng dưới đây:
 - $F[m, v] = F[m-1, v]$ nếu $m > v$;
 - $F[m, v] = F[m-1, v] + F[m, v-m]$ nếu $m \leq v$.
 - Từ đó ta có $F[5, 5] = F[4,5] + F[5,0] = 7$ (cách).

	0	1	2	3	4	5
0	1	0	0	0	0	0
1	1	1	1	1	1	1
2	1	1	2	2	3	3
3	1	1	2	3	4	5
4	1	1	2	3	5	6
5	1	1	2	3	5	7

Bài tập 1. Hãy cho biết kết quả thực hiện chương trình dưới đây?

```
#include <iostream.h>

void main (void ) {
    int F[100][100], n, m, v;
    cout<<"Nhập n="; cin>>n; //Nhập n=5
    for ( int j=0; j <=n; j++) F[0][j] =0; //Thiết lập dòng 0 là 0.
    F[0][0] = 1; //Duy chỉ F[0][0] thiết lập là 1.
    for (m =1; m<=n; m++) {
        for (v= 0; v<=n; v++){
            if ( m > v ) F[m][v] = F[m-1][v];
            else F[m][v] = F[m-1][v] + F [m][v-m];
        }
    }
    cout<<" Kết quả:"<<F[n][n]<<endl;
}
```


Bài tập 4. Hãy cho biết kết quả thực hiện chương trình dưới đây?

```
#include <iostream.h>
int n, dem=0;
int F(int m, int v){
    if (m==0) {
        if (v==0) return(1);
        else return(0);
    }
    else {
        if (m>v ) return (F(m-1, v));
        else return(F(m-1,v)+F(m,v-m));
    }
}
int main (void ) {
    cout<<"\n Nhap n=";<<cin>>n;
    cout<<"\n Ket qua:"<<F(n,n)<<endl;
    system("PAUSE");return 0;
}
```

Ví dụ 2.16. Giải bài toán cái túi (0-1 Knapsack Problem). Một nhà thám hiểm cần đem theo một cái túi trọng lượng không quá w . Có n đồ vật cần đem theo. Đồ vật i có trọng lượng a_i có giá trị sử dụng c_i . Hãy tìm cách đưa đồ vật vào túi cho nhà thám hiểm sao cho tổng giá trị sử dụng các đồ vật trong túi là lớn nhất.

Bài toán cái túi được chia thành 2 loại: nếu số lượng mỗi đồ vật được đưa vào túi không nhất thiết là một số nguyên được gọi là bài toán Fraction Knapsack. Trong trường hợp này ta đã giải quyết bằng thuật toán tham. Trong trường hợp bài toán mỗi đồ vật được đưa vào túi chỉ là một số nguyên 0, 1 được gọi là bài toán 0-1 Knapsack.

Về nguyên tắc ta có thể sử dụng một phương án nhánh cận để hạn chế các phương án duyệt của bài toán, tuy nhiên độ phức tạp trong trường hợp xấu nhất của phương pháp vẫn là hàm mũ. Dưới đây là một phương pháp giải bài toán bằng qui hoạch động.

Input:

- N, W tương ứng với số lượng đồ vật và trọng lượng túi.
- $A[] = (a_1, \dots, a_n)$: vector trọng lượng các đồ vật
- $C[] = (c_1, \dots, c_n)$: vector giá trị sử dụng các đồ vật

Output:

- Tìm $\max(\sum_{i=1}^N c_i * x_i) : \sum_{i=1}^N a_i * x_i \leq W \ (x_i \in \{0,1\})$.

Lời giải:

Bước cơ sở: $F[0, w] = 0$ vì giá trị lớn nhất khi chọn 0 đồ vật với giới hạn trọng lượng w là 0.

Bước truy hồi: với mọi $i > 0$ và trọng lượng w , khi đó việc chọn tối các đồ vật từ $\{1, 2, \dots, i\}$ sẽ có hai khả năng: chọn được đồ vật i và không chọn được đồ vật i .

$$F[i, W] = \begin{cases} F[i - 1, W] & \text{nếu ta không chọn được đồ vật } i \\ c_i + F[i - 1, W - a_i] & \text{nếu ta chọn được đồ vật } i \end{cases}$$

Xây dựng bảng phương án: Bảng phương án gồm $n+1$ dòng và $m+1$ cột ($m = W$ là giới hạn trọng lượng túi). Dòng đầu tiên ghi toàn bộ số 0 đó là lời giải bài toán cơ sở. Sử dụng công thức truy hồi tính giá trị dòng 1 từ dòng 0, tính giá trị dòng 2 từ dòng 0 và 1 và làm cho đến khi nhận được đầy đủ $n + 1$ dòng.

Bước 3 (Truy vết): $F[n, m]$ trong bảng phương án chính là giá trị lớn nhất chọn n đồ vật với giới hạn trọng lượng m . Nếu $F[n, m] = F[n-1, m]$ thì phương án tối ưu không chọn đồ vật n và ta truy tiếp $F[n-1, m]$. Nếu $F[n, m] \neq F[n-1, m]$ thì phương án tối ưu có đồ vật n và ta truy tiếp $F[n-1, m - a_n]$. Truy tiếp đến hàng thứ 0 của bảng phương án ta nhận được phương án tối ưu.

Thuật toán. Giải bài toán cái túi bằng qui hoạch động.

```
void Optimization(void) {//thuật toán qui hoạch động
    int i, j;
    for (i = 1; i <= n; i++){
        for(j=0; j<=B; j++){
            F[i][j] = F[i-1][j];
            If (j >= A[i] && F[i][j] < F[i-1][j-A[i]] + C[i])
                F[i][j] = F[i-1][j-A[i]] + C[i];
        }
    }
}

void Trace(void){ //lần vết
    cout << "\n Giá trị tối ưu:" << F[n][B];
    cout << "\n Phương án tối ưu:";
    while(n != 0) {
        if(F[n][B] != F[n-1][B]) {
            cout << n << " ";
            B = B - A[n];
        }
        n--;
    }
}
```

BÀI 1. XÂU CON CHUNG DÀI NHẤT

Xâu ký tự X được gọi là xâu con của xâu ký tự Y nếu ta có thể xoá đi một số ký tự trong xâu Y để được xâu X.

Cho hai xâu ký tự A và B dài không quá 1000 ký tự (chữ cái viết thường hoặc chữ số), hãy tìm xâu ký tự C có độ dài lớn nhất và là con của cả A và B.

Input: Dòng 1: chứa xâu A. Dòng 2: chứa xâu B

Output: Chỉ gồm một dòng ghi độ dài xâu C tìm được

Ví dụ:

Input	Output
abc1def2ghi3 abcdefghi123	10

Lời giải: gọi độ dài xâu X là m, độ dài xâu y là n và $LCS(X,Y,m,n)$ là độ dài xâu con dài nhất của X và Y. Khi đó:

Bước cơ sở: với $n=0$ hoặc $m=0$ thì $LCS(X,Y,m,n)=0$.

Bước truy hồi : với mọi m, $n \geq 1$ khi đó:

$$LCS(X,Y,m,n)=\begin{cases} 1 + LCS(X,Y,m-1,n-1) & \text{nếu} \\ & X[m-1] = Y[n-1] \\ \max(LCS(X,Y,m,n-1), LCS(X,Y,m-1,n)) & \\ & \text{nếu } X[m-1] \neq Y[n-1] \end{cases}$$

Bước truy vết:

Trả lại giá trị $LSC(X,Y,m,n)$.

Ví dụ $X = \text{“AGGTAB”}$, $Y = \text{“GXTXAYB”}$, ta có bảng phương án của bài toán được thể hiện như sau:

L=

	G	X	T	X	A	Y	B
A	0	0	0	0	0	0	0
G	1	1	1	1	1	1	1
G	1	1	1	1	1	1	1
T	1	1	2	2	2	2	2
A	1	1	2	2	3	3	3
B	1	1	2	2	3	3	4

BÀI 2. DÃY CON TĂNG DÀI NHẤT

Cho một dãy số nguyên gồm N phần tử $A[1], A[2], \dots, A[N]$.

Biết rằng dãy con tăng đơn điệu là 1 dãy $A[i_1], \dots, A[i_k]$

thỏa mãn $i_1 < i_2 < \dots < i_k$ và $A[i_1] < A[i_2] < \dots < A[i_k]$.

Hãy cho biết dãy con tăng đơn điệu dài nhất của dãy này có bao nhiêu phần tử?

Input: Dòng 1 gồm 1 số nguyên là số N ($1 \leq N \leq 1000$). Dòng thứ 2 ghi N số nguyên $A[1], A[2], \dots, A[N]$ ($1 \leq A[i] \leq 10000$).

Output: Ghi ra độ dài của dãy con tăng đơn điệu dài nhất.

Ví dụ:

Input	Output
6 1 2 5 4 6 2	4

Lời giải: gọi $L[i]$ là độ dài dãy con tăng dài nhất từ $A[0], A[1], \dots, A[i]$. Khi đó:

- **Bước cơ sở:** $L[0] = 1$ nếu $n = 1$.
- **Bước truy hồi:**

$$L[i] = \begin{cases} 1 + \max(L[j]) & \text{nếu } 0 < j < i \text{ và } A[j] < A[i] \\ 1 & \text{trong các trường hợp khác} \end{cases}$$

- **Bước truy vết:** trả lại $\max(L[0], L[1], \dots, L[n-1])$

BÀI 3. DẪY CON CÓ TỔNG BẰNG S

Cho N số nguyên dương tạo thành dãy $A = \{A_1, A_2, \dots, A_N\}$. Tìm ra một dãy con của dãy A (không nhất thiết là các phần tử liên tiếp trong dãy) có tổng bằng S cho trước.

Input: Dòng đầu tiên ghi hai số nguyên dương N và S ($0 < N \leq 200$) và S ($0 < S \leq 40000$). Các dòng tiếp theo lần lượt ghi N số hạng của dãy A là các số A_1, A_2, \dots, A_N ($0 < A_i \leq 200$).

Output: Nếu bài toán vô nghiệm thì in ra "NO". Nếu bài toán có nghiệm thì in ra "YES"

Ví dụ:

Input	Output
5 6 1 2 4 3 5	YES

Lời giải: Gọi $F(A[], n, S)$ là phép kiểm tra tập các tập con của $A[]$ có tổng bằng S hay không. Khi đó:

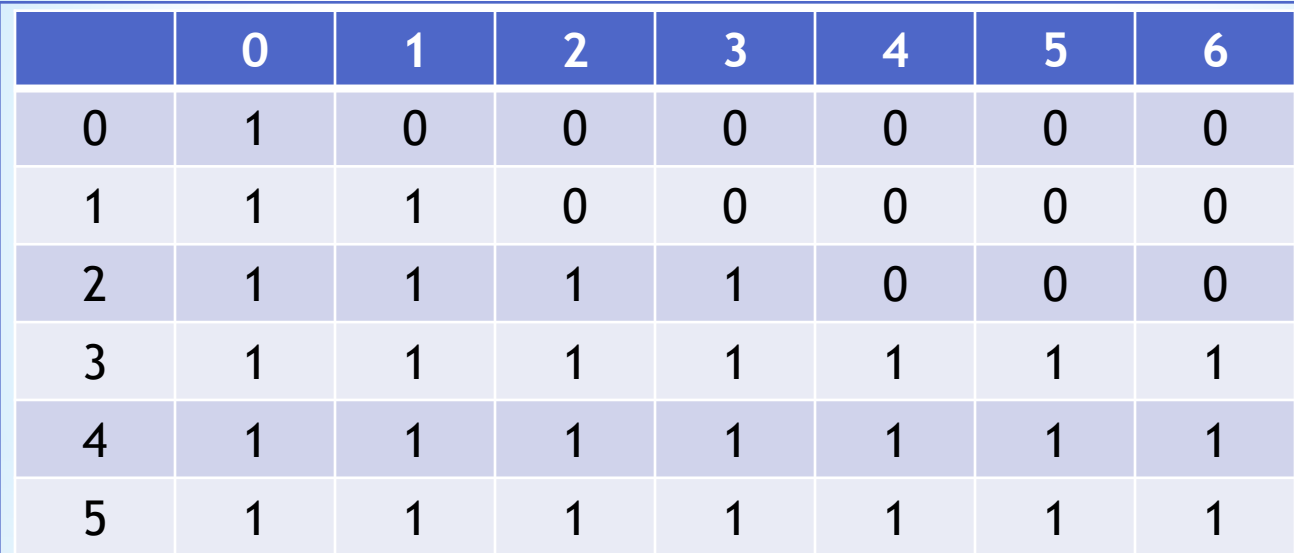
- **Bước cơ sở:**
 - Nếu $S=0$ thì $F(A, n, S) = \text{true}$ vì luôn tồn tại tập con rỗng.
 - Nếu $A \neq \emptyset$ và $S \neq 0$ thì $F(A, n, S) = \text{false}$

- **Bước truy hồi:**

$$F(A, n, S) = \begin{cases} \text{true} & \text{nếu } S = 0 \\ \text{false} & \text{nếu } n = 0 \text{ và } S \neq 0 \\ F(A, n - 1, S) & \text{nếu } A[n - 1] > S \\ F(A, n - 1, S) \text{ hoặc } F(A, n - 1, S - A[n - 1]) & \text{trường hợp còn lại} \end{cases}$$

- **Bước truy vết:** trả lại $F(A, n, S)$.

Ví dụ $A[] = 1, 2, 3, 4, 5$ và $S = 6$ ta có bảng phương án của bài toán được thể hiện như sau:



	0	1	2	3	4	5	6
0	1	0	0	0	0	0	0
1	1	1	0	0	0	0	0
2	1	1	1	1	0	0	0
3	1	1	1	1	1	1	1
4	1	1	1	1	1	1	1
5	1	1	1	1	1	1	1

BÀI 6. XÂU CON ĐỐI XỨNG DÀI NHẤT

Cho xâu S chỉ bao gồm các ký tự viết thường và dài không quá 5000 ký tự.

Hãy tìm xâu con đối xứng dài nhất của S.

Input:

Dòng đầu tiên là số lượng bộ test T ($T \leq 10$).

Mỗi test gồm một xâu S có độ dài không vượt quá 5000, chỉ gồm các kí tự thường.

Output:

Với mỗi test, in ra đáp án tìm được.

Ví dụ:

Input	Output
2	5
abcbadd	5
aaaaa	

