

MỘT SỐ BÀI TOÁN QHD TIÊU BIỂU

1. Khái niệm về phương pháp quy hoạch động:

Phương pháp quy hoạch động dùng để giải bài toán tối ưu có bản chất đệ quy, tức là việc tìm phương án tối ưu cho bài toán đó có thể đưa về tìm phương án tối ưu của một số hữu hạn các bài toán con.

Đối với một số bài toán đệ quy, nguyên lý chia để trị (divide and conquer) thường đóng vai trò chủ đạo trong việc thiết kế thuật toán. Để giải quyết một bài toán lớn, ta chia nó thành nhiều bài toán con cùng dạng với nó để có thể giải quyết độc lập.

Trong phương án quy hoạch động, nguyên lý chia để trị càng được thể hiện rõ: Khi không biết phải giải quyết những bài toán con nào, ta sẽ đi giải quyết toàn bộ các bài toán con và lưu trữ những lời giải hay đáp số của chúng với mục đích sử dụng lại theo một sự phối hợp nào đó để giải quyết những bài toán tổng quát hơn.

Đó chính là điểm khác nhau giữa Quy hoạch động và phép phân giải đệ quy và cũng là nội dung phương pháp quy hoạch động:

- Phép phân giải đệ quy bắt đầu từ bài toán lớn phân ra thành nhiều bài toán con và đi giải từng bài toán con đó. Việc giải từng bài toán con lại đưa về phép phân ra tiếp thành nhiều bài toán nhỏ hơn và lại đi giải các bài toán nhỏ hơn đó bất kể nó đã được giải hay chưa.

- Quy hoạch động bắt đầu từ việc giải tất cả các bài toán nhỏ nhất (bài toán cơ sở) để từ đó từng bước giải quyết những bài toán lớn hơn, cho tới khi giải được bài toán lớn nhất (bài toán ban đầu).

Bài toán giải theo phương pháp quy hoạch động gọi là bài toán quy hoạch động.

Công thức phối hợp nghiệm của các bài toán con để có nghiệm của bài toán lớn gọi là công thức truy hồi của quy hoạch động.

Tập các bài toán có ngay lời giải để từ đó giải quyết các bài toán lớn hơn gọi là cơ sở quy hoạch động.

Không gian lưu trữ lời giải các bài toán con để tìm cách phối hợp chúng gọi là bảng phương án của quy hoạch động.

Trước khi áp dụng phương pháp quy hoạch động ta phải xét xem phương pháp đó có thỏa mãn những yêu cầu dưới đây không:

- Bài toán lớn phải phân rã được thành nhiều bài toán con, mà sự phối hợp lời giải của các bài toán con đó cho ta lời giải của bài toán lớn.

- Vì quy hoạch động là đi giải tất cả các bài toán con, nên nếu không đủ không gian vật lý lưu trữ lời giải (bộ nhớ, đĩa, ...) để phối hợp chúng thì phương pháp quy hoạch động cũng không thể thực hiện được.

- Quá trình từ bài toán cơ sở tìm ra lời giải bài toán ban đầu phải qua hữu hạn bước. Các bước cài đặt một chương trình sử dụng quy hoạch động:

- Giải tất cả các bài toán cơ sở (thông thường rất dễ), lưu các lời giải vào bảng phương án.

- Dùng công thức truy hồi phối hợp những lời giải của các bài toán nhỏ đã lưu trong bảng phương án để tìm lời giải của các bài toán lớn hơn rồi lưu chúng vào bảng phương án. Cho tới khi bài toán ban đầu tìm được lời giải.

- Dựa vào bảng phương án, truy vết tìm ra nghiệm tối ưu.







Cho tới nay, vẫn chưa có một định lý nào cho biết một cách chính xác những bài toán nào có thể giải quyết hiệu quả bằng quy hoạch động. Tuy nhiên để biết được bài toán có thể giải bằng quy hoạch động hay không, ta có thể đặt câu hỏi:

1. “Một nghiệm tối ưu của bài toán lớn có phải là sự phối hợp các nghiệm tối ưu của các bài toán con hay không?”

2. “Liệu có thể nào lưu trữ được nghiệm các bài toán con dưới một hình thức nào đó để phối hợp tìm được nghiệm bài toán lớn?”.

2. Các bước cơ bản để giải một bài toán quy hoạch động:

Với mỗi bài toán ứng dụng giải thuật quy hoạch động, ta vẫn phải trả lời rõ ràng, chính xác 6 câu hỏi:

-  Tên và ý nghĩa các biến phục vụ sơ đồ lặp,
-  Cách khai báo các biến đó,
-  Sơ đồ (công thức) lặp chuyển từ một bước sang bước tiếp theo,
-  Giá trị đầu của các biến tham gia tính lặp,
-  Tham số điều khiển lặp: thay đổi từ đâu đến đâu,
-  Kết quả: ở đâu và làm thế nào để dẫn xuất ra.

Các cách trả lời khác nhau sẽ dẫn đến những giải thuật khác nhau cả về cách thực hiện lẫn độ phức tạp.

3. Ví dụ về các bài toán có thể giải bằng phương pháp quy hoạch động

3.1. Bài toán Tính N!

GT.PAS

Ta có định nghĩa như sau: $n! = \begin{cases} 1 & \text{nếu } n=0 \\ n*(n-1)! & \text{nếu } n>0 \end{cases}$

Cho một số nguyên dương n ($0 \leq n \leq 13$).

Yêu cầu: Hãy tính $n!$ bằng phương pháp quy hoạch động (lập bảng phương án).

Dữ liệu vào: Ghi trong file văn bản GT.INP có cấu trúc như sau:

- Dòng 1: Ghi số nguyên dương n .

Dữ liệu ra: Ghi ra file văn bản GT.OUT theo cấu trúc như sau:

- Dòng 1: Ghi giá trị tính được của $n!$

Ví dụ:

GT.INP	GT.OUT
3	6

Thuật toán:

Gọi GT[i] là giá trị của $i!$ ($0 \leq i \leq 13$)

Ta có công thức quy hoạch động như sau:

GT[i] := GT[i-1]*i;

Như vậy, việc tính $n!$ sẽ được thực hiện bằng vòng lặp:

GT[0] :=1;

For i:=1 to n do

GT[i] := GT[i-1]*i;

Kết quả: giá trị của $n!$ nằm trong phần tử GT[n].

3.2. Bài toán Tính dãy Fibonacci

FIBO.PAS

Ta có định nghĩa như sau: $F(n) = \begin{cases} 1 & \text{nếu } n=0 \text{ or } n=1 \\ F(n-1)+F(n-2) & \text{nếu } n>1 \end{cases}$

Cho một số nguyên dương n ($0 \leq n \leq 50$).

Yêu cầu: Hãy tính $F(n)$ bằng phương pháp quy hoạch động (lập bảng phương án).

Dữ liệu vào: Ghi trong file văn bản FIBO.INP có cấu trúc như sau:

- Dòng 1: Ghi số nguyên dương n .

Dữ liệu ra: Ghi ra file văn bản FIBO.OUT theo cấu trúc như sau:

- Dòng 1: Ghi giá trị tính được của $F(n)$.

Ví dụ:

FIBO.INP	FIBO.OUT
----------	----------

5	8
---	---

Thuật toán:

Gọi $F[i]$ là giá trị Fibonacci của f_i ($0 \leq i \leq 50$).

Ta có công thức quy hoạch động như sau:

$$F[i] := F[i-1] + F[i-2];$$

Như vậy, việc tính f_n được thực hiện bằng vòng lặp:

$$F[0] := 0;$$

$$F[1] := 1;$$

For $i := 2$ to n do

$$F[i] := F[i-1] + F[i-2];$$

Kết quả: giá trị f_n nằm trong $F[n]$.

3.3. Bài toán Tính tổng của dãy số

SUM.PAS

Cho dãy số nguyên gồm n phần tử a_1, a_2, \dots, a_n ($1 \leq n \leq 10^5$) và hai số nguyên dương p và q ($1 \leq p \leq q \leq n$).

Yêu cầu: Hãy tính tổng của các phần tử liên tiếp từ $a_p \dots a_q$ bằng phương pháp quy hoạch động (lập bảng phương án).

Dữ liệu vào: Ghi trong file văn bản SUM.INP có cấu trúc như sau:

- Dòng 1: Ghi số nguyên dương n và k , hai số được ghi cách nhau một dấu cách.
- Dòng 2: Ghi n số nguyên a_1, a_2, \dots, a_n , các số được ghi cách nhau ít nhất một dấu cách ($-32000 \leq a_i \leq 32000$).
- Dòng thứ i trong k dòng tiếp theo: Mỗi dòng ghi hai số nguyên dương p_i và q_i , hai số được ghi cách nhau một dấu cách ($1 \leq p_i \leq q_i \leq n$).

Dữ liệu ra: Ghi ra file văn bản SUM.OUT theo cấu trúc như sau:

- Dữ liệu được ghi trên k dòng: Dòng thứ i ghi một số nguyên là tổng giá trị của các phần tử trong đoạn $a_{p_i} \dots a_{q_i}$.

Ví dụ:

SUM.INP	SUM.OUT
5 3	21
2 9 -3 5 8	6
1 5	5
2 3	
4 4	

Thuật toán:

Gọi $A[i]$ là giá trị của phần tử thứ i trong dãy số a_1, a_2, \dots, a_n .

Gọi $T[i]$ là tổng giá trị các phần tử a_1, a_2, \dots, a_i ($1 \leq i \leq n$).

Ta có công thức quy hoạch động để tính $T[i]$ như sau:

$$T[i] := T[i - 1] + A[i];$$

Như vậy, việc tính $T[n]$ được thực hiện bằng vòng lặp:

$$T[0] := 0;$$

For $i:=1$ to n do

$$T[i] := T[i - 1] + A[i];$$

Kết quả: Tổng các phần tử liên tiếp từ a_p đến a_q được tính theo công thức:

$$\text{Sum} := A[q] - A[p-1];$$

3.4. TRIANGLE PASCAL (Tam giác Pascal)

TRIANPAS.PAS

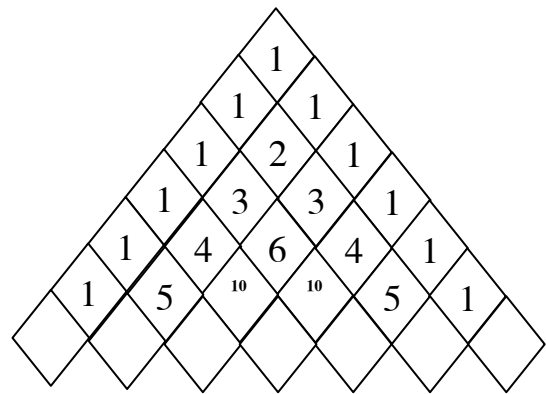
Tam giác Pascal là một mô hình dùng để đưa ra các hệ số của khai triển nhị thức Newton bậc N $(x+1)^N$.

Ví dụ: trong khai triển $(x+1)^2 = x^2 + 2x + 1$

có các hệ số là 1 2 1

Trong khai triển $(x+1)^3 = x^3 + 3x^2 + 3x + 1$

có các hệ số là 1 3 3 1



Yêu cầu: Hãy tìm các hệ số trong khai triển nhị thức Newton $(x + 1)^N$.

Dữ liệu vào: Cho trong file văn bản TRIANPAS.INP có cấu trúc như sau:

Dòng 1: Ghi số nguyên dương N ($1 \leq N \leq 100$).

Dữ liệu ra: Ghi ra file văn bản TRIANNUM.OUT theo cấu trúc:

Dòng 1: Ghi ra các số nguyên dương lần lượt là các hệ số trong khai triển nhị thức Newton $(x + 1)^N$, các số được ghi cách nhau một dấu cách.

Ví dụ:

TRIANPAS.INP	TRIANPAS.OUT
5	1 5 10 10 5 1

Thuật toán:

+ Ta xây dựng mảng hai chiều có kích thước $[0..100, 0..101]$

+ Sử dụng phương pháp quy hoạch động với công thức như sau:

Dòng thứ i được tính thông qua dòng $i-1$

$$L[i, j] = L[i-1, j-1] + L[i-1, j]$$

+ Thuật toán cụ thể như sau:

$L[0,1] = 1; L[1,1] = 1; L[1,2] = 1;$

For $i := 2$ to N do

 Begin

$L[i, 1] := 1;$

 For $j := 2$ to $i+1$ do

$L[i, j] = L[i-1, j-1] + L[i-1, j];$

 End;

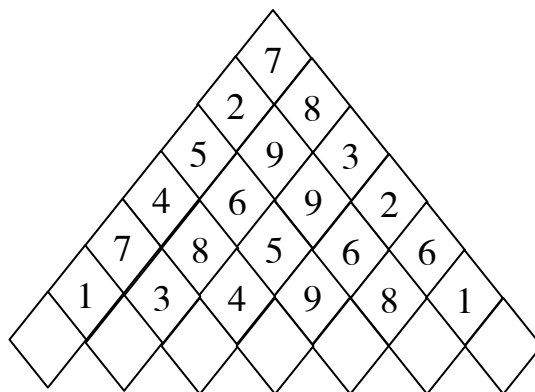
+ Kết quả được lưu trữ ở dòng N , cụ thể:

$L[N, 1], L[N, 2], L[N, 3], \dots, L[N, N], L[N, N+1]$

3.5. TRIANGLE NUMBER (Tam giác số)

TRIANNUM.PAS

Cho tam giác số như hình vẽ. Ta định nghĩa một đường đi trong tam giác số là đường đi xuất phát từ hình thoi ở đỉnh tam giác và đi đến được các hình thoi có chung cạnh với nó, đường đi kết thúc khi gặp một hình thoi ở đáy tam giác.



Yêu cầu: Hãy tìm một đường đi trong tam

giác số sao cho tổng giá trị của các ô trong đường đi có giá trị lớn nhất.

Dữ liệu vào: Cho trong file văn bản TRIANNUM.INP có cấu trúc như sau:

Dòng 1: Ghi số nguyên dương N là số hàng của tam giác ($1 \leq N \leq 100$).

Dòng thứ i trong N dòng tiếp theo: Ghi i số nguyên dương lần lượt là giá trị của các ô trên dòng thứ i tương ứng trong tam giác (Các số có giá trị không quá 32000). Các số được ghi cách nhau một dấu cách.

Dữ liệu ra: Ghi ra file văn bản TRIANNUM.OUT theo cấu trúc:

Dòng 1: Ghi ra số nguyên dương S là tổng giá trị của đường đi tìm được.

Ví dụ:

TRIANNUM.INP	TRIANNUM.OUT
6 7 2 8	48

5 9 3	
4 6 9 2	
7 8 5 6 6	
1 3 4 9 8 1	

Thuật toán:

- + Ta xây dựng mảng hai chiều có kích thước $[1..100, 0..101]$
- + Sử dụng phương pháp quy hoạch động với công thức như sau:

Dòng thứ i được tính thông qua dòng $i-1$

$$L[i, j] = \text{Max}(L[i-1, j-1] + L[i-1, j]) + A[i, j]$$

- + Thuật toán cụ thể như sau:

$$L[1, 1] = A[1, 1];$$

For $i := 2$ to N do

Begin

For $j := 1$ to i do

$$L[i, j] = \text{Max}(L[i-1, j-1] + L[i-1, j]) + A[i, j];$$

End;

- + Kết quả được lưu trữ ở dòng N , cụ thể:

$$\text{Tong} := L[N, 1]$$

For $j := 2$ to N do

if $(L[N, j] > \text{Tong})$ then

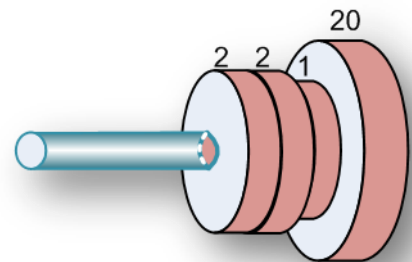
$$\text{Tong} := L[N, j];$$

4. Các bài tập nâng cao áp dụng phương pháp quy hoạch động để giải

4.1. Bài toán Cử tạ

DUMBBELL.???

Rèn luyện thể lực bằng cách tập nâng tạ thu hút được sự chú ý của rất nhiều bạn trẻ. Tạ là một thanh trục có gắn ở hai đầu các đĩa tạ. Bộ đĩa tạ trong phòng tập bao gồm các loại 1kg, 2kg, 5kg, 10kg, 15kg và 20kg với số lượng mỗi loại là đủ nhiều. Các đĩa tạ ở hai đầu thanh được gắn đối xứng để đảm bảo thanh tạ được cân. Mỗi người, tùy theo



thể lực của mình, lắp các đĩa tạ để có trọng lượng phù hợp. Để điều chỉnh trọng lượng, người ta tháo các đĩa ngoài cùng, lắp các đĩa mới vào. Do tính đối xứng của thanh tạ, ta chỉ xét các thao tác điều chỉnh ở một đầu.

Hiện tại ở một đầu đang có n đĩa tạ gắn vào trục ($1 \leq n \leq 10$), tính từ trong ra ngoài đĩa thứ i có trọng lượng p_i . Bạn cần có thanh tạ với trọng lượng một đầu là w ($0 \leq w \leq 100$).

Ví dụ, hiện tại $n = 4$ và các đĩa tạ là (2, 2, 1, 20), bạn cần điều chỉnh trọng lượng thành 14kg. Bạn sẽ phải thực hiện 3 thao tác tháo lắp: tháo đĩa 20kg, tháo đĩa 1kg và lắp đĩa 10kg.

Yêu cầu: Cho $n, p_i, i = 1 \div n, w$. Hãy xác định số thao tác ít nhất cần thực hiện.

Dữ liệu vào: Cho trong file văn bản DUMBBELL.INP có cấu trúc như sau

- Dòng 1: Ghi số nguyên n .
- Dòng 2: Ghi n số nguyên p_1, p_2, \dots, p_n , các số được ghi cách nhau ít nhất một dấu cách.
- Dòng 3: Ghi số nguyên w .

Dữ liệu ra: Ghi ra file văn bản DUMBBELL.OUT theo cấu trúc như sau:

- Dòng 1: Ghi số thao tác cần thực hiện.

Ví dụ:

DUMBBELL.INP	DUMBBELL.OUT
4	3
2 2 1 20	
14	

Thuật toán:

Ta tạo ra bảng phương án B, B_i xác định số lần lắp đĩa tối thiểu để làm tăng trọng lượng lên i kg.

Var b:array[0..100] of byte;

Việc xác định B_i ($i = 0 \div 100$) khá đơn giản:

```

t := i div 20;      j := i mod 20;
t := t + j div 15;  j := j mod 15;
t := t + j div 10;  j := j mod 10;
t := t + j div 5;   j := j mod 5;
t := t + j div 2;   j := j mod 2;

```


$B[i] := t + j$;

Nếu sử dụng mảng hằng C với C_i là số lần lắp đĩa tối thiểu để làm tăng trọng lượng lên i kg ($i = 0 \div 19$).

$C : \text{array}[0..19] \text{ of byte}$;

$= (0, 1, 1, 2, 2, 1, 2, 2, 3, 3, 1, 2, 2, 3, 3, 1, 2, 2, 3, 3)$;

Việc tính B_i ($i := 0 \div 100$) lúc này chỉ cần sử dụng vòng lặp:

For $i := 0$ to 19 do $B[i] := C[i]$;

For $i := 20$ to 100 do $B[i] := i \text{ div } 20 + B[i \text{ mod } 20]$;

Lời giải của bài toán có thể nhận được bằng cách duyệt tất cả các cách tháo lần lượt đĩa tại $n, n-1, n-2, \dots, 2, 1$.

Bảng phương án còn là công cụ sắc bén với các loại bài toán liên quan tới phát hiện, nhận dạng chu trình. Nó giúp ta đạt được hiệu quả $O(n)$ và trong nhiều trường hợp – $O(1)$!

4.2. Bài toán Thỏ nhặt cà rốt

Các con thú nuôi trong chuồng ở vườn bách thú thường ít có điều kiện vận động. Điều này vừa có hại cho sức khỏe của thú nuôi, vừa làm giảm hứng thú của khách tham quan. Để khắc phục điều đó, Ban giám đốc cho đặt một cái thang có n bậc trong chuồng thỏ. Đến giờ cho ăn người ta đặt cà rốt - thứ khoái khẩu

nhất của thỏ, lên bậc trên cùng của thang. Thỏ phải nhảy theo các bậc thang để lấy cà rốt. Mỗi bước nhảy thỏ có thể vượt được k bậc ($1 \leq k \leq n \leq 300$). Có thể có nhiều cách nhảy để lấy cà rốt. Hai cách nhảy gọi là khác nhau nếu tồn tại một bậc thỏ tới được ở một cách nhảy và bị bỏ qua ở cách kia. Ví dụ, với $n = 4$ và $k = 3$ có tất cả 7 cách lấy cà rốt khác nhau: $1+1+1+1, 1+1+2, 1+2+1, 2+1+1, 2+2, 1+3, 3+1$.

Yêu cầu: Cho k và n . Hãy xác định số cách khác nhau thỏ có thể thực hiện để lấy cà rốt.

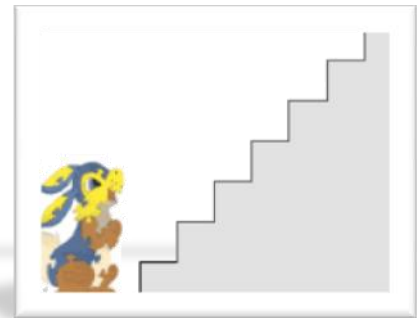
Dữ liệu vào: Ghi trong file văn bản CARROT.INP có cấu trúc như sau:

- Dòng 1: Ghi số nguyên dương t là số lượng cặp k và n ($1 \leq t \leq 50$).

- Dòng thứ i trong t dòng tiếp theo: Mỗi dòng ghi 2 số nguyên k và n .

Dữ liệu ra: Ghi ra file văn bản CARROT.OUT, kết quả mỗi test đưa ra trên một dòng dưới dạng số nguyên.

CARROT.???



Ví dụ:

CARROT.INP
3
1 3
2 7
3 10

CARROT.OUT
1
21
274

Thuật toán:

Đây là bài toán áp dụng sơ đồ tính lặp để tích lũy kết quả. Loại sơ đồ này có bản chất rất gần với giải thuật quy hoạch động nên nhiều khi người ta cũng gộp nó vào bài toán có thuật giải quy hoạch động.

Ta có thuật toán như sau:

a) Gọi f_i là số cách mà thỏ có thể nhảy tới bậc thứ i của thang,

b) Công thức lặp (cách tính f_i): $f_i = \sum_{j=i-k}^{i-1} f_j$

For $i:=1$ to n do

For $j:=i-k$ to $i-1$ do

$F[i] := F[i] + F[j];$

c) Giá trị đầu: Cần có k giá trị ban đầu để triển khai công thức lặp. Có thể chọn một trong hai cách:

1. Tính riêng f_i ($i = 1 \div k$) theo công thức $f_0=1, f_i = \sum_{j=1}^{i-1} f_j$

$F[0] := 1;$

For $i:=1$ to k do

For $j:=1$ to $i-1$ do

$F[i] := F[i] + F[j];$

Lúc này ta có thủ tục tính như sau:

Procedure Process;

Var i, j :Longint;

Begin

$F[0] := 1;$

For $i:=1$ to k do

For $j:=1$ to $i-1$ do

$F[i] := F[i] + F[j];$

```

For i:=k+1 to n do
    For j:= i - k to i - 1 do
        F[i] := F[i] + F[j];

```

End;

2. Cho $f_0 = 1, f_i = 0, i = -k+1 \div -1$,

d) Khai báo: Nếu áp dụng cách chuẩn bị giá trị đầu thứ 2 thì cần khai báo:

Var f:array[-300..300] of int64;

Procedure Process;

Var i, j:Longint;

Begin

F[0] := 1;

For i:=-k+1 to -1 do F[i] := 0;

For i:= 1 to n do

For j:= i - k to i - 1 do

F[i] := F[i] + F[j];

End;

e) Nếu sử dụng cách chuẩn bị giá trị đầu thứ 2 thì phải tính với $i = 1 \div n$,

f) Kết quả: giá trị f_n .

Lưu ý: - Bài toán này có thể áp dụng giải thuật tìm kiếm quay lui (Back Tracking).

- Kiểu dữ liệu ở đây chưa thật quan trọng, nó sẽ được xác định chính xác trong quá trình hiệu chỉnh chương trình, khi thử nghiệm với $k = n = 300$.

Trần Lương Vương