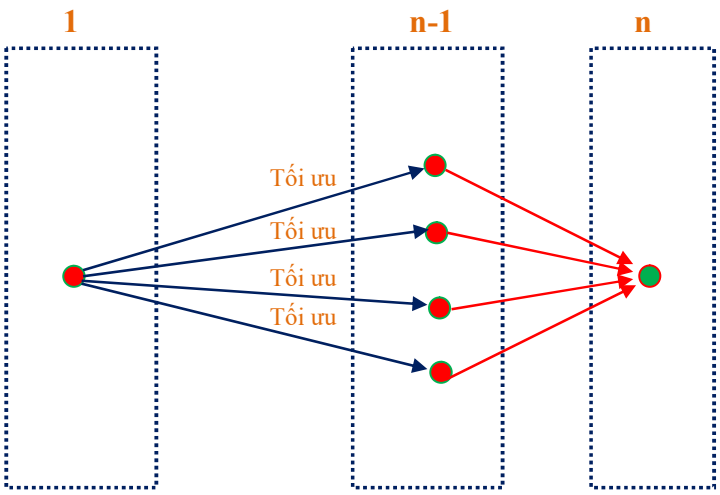
NGUYỄN KHẮC NHO

LÝ THUYẾT QUY HOẠCH ĐỘNG VÀ ÁP DỤNG



NĂM 2012

Lời nói đầu

Trong các kỳ thi học sinh giỏi tin học như: Olympic 30/4; Học sinh giỏi quốc gia; Olympic tin học quốc tế…Thì các lớp bài toán về tối ưu hóa luôn được ưu tiên lựa chọn trong các đề thi, vì tính ứng dụng vào thực tiễn cao.

Có rất nhiều phương pháp để giải quyết lớp các bài toán tối ưu như: Phương pháp *nhánh cận*, phương pháp *tham lam*, phương pháp *quy hoạch động* (QHĐ)…Tùy từng bài toán cụ thể mà ta chọn 1 phương pháp để áp dụng nhằm đạt được hiệu quả (hiệu quả về phép tính toán (tốc độ), hiệu quả về bộ nhớ) tốt nhất. Trong đó phương pháp QHĐ luôn được ưu tiên lựa chọn vì tính hiệu quả của chúng cao hơn các phương pháp khác trong đại đa số các bài toán về tối ưu hóa.

Phương pháp QHĐ là một phương pháp khó bởi vì: Mỗi bài toán tối ưu có một đặc thù riêng, cách giải hoàn toàn khác nhau, cho nên cách áp dụng phương pháp QHĐ cho các bài toán cũng hoàn toàn khác nhau, không có khuân mẫu chung cho tất cả các bài.

Phương pháp QHĐ là phương pháp giải quyết tốt các bài toán về tối ưu hóa nó cũng còn được áp dụng giải quyết một số bài toán không phải tối ưu và cũng đem lại hiệu quả cao. Việc xác định những bài toán như thế nào thì có thể áp dụng được phương pháp QHĐ vẫn còn rất khó khăn cho rất nhiều người.

Cho nên để giải quyết được nhiều bài toán khác nhau bằng PP QHĐ thì đòi hỏi người lập trình phải nắm vững bản chất của PP QHĐ. Với tài liệu này hi vọng sẽ giúp bạn đọc làm chủ được PP QHĐ một cách tự nhiên nhất.

Trong tài liệu này có sử dụng một số tài liệu tham khảo trên internet, một số cuốn sách chuyên tin, một số tài liệu trong và ngoài nước…

Trong quá trình biên soạn sẽ không tránh khỏi những sai sót mong bạn đọc đóng góp gởi góp ý cũng như thắc mắc về địa chỉ email: quyhoachdong@gmail.com.

Xin chân thành cảm ơn

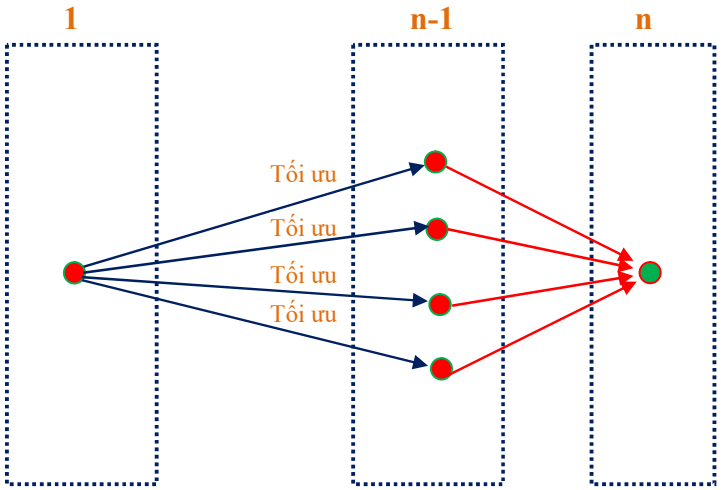
Mục Lục

CHƯƠNG 1: PHƯƠNG PHÁP QUY HOẠCH ĐỘNG

I. Khái niệm về phương pháp quy hoạch động (QHĐ)

Phương pháp quy hoạch động cùng nguyên lý tối ưu được nhà toán học Mỹ R.Bellman đề xuất vào những năm 50 của thế kỷ 20. Phương pháp này đã được áp dụng để giải hàng loạt bài toán thực tế trong các quá trình kỹ thuật cộng nghệ, tổ chức sản xuất, kế hoạch hoá kinh tế… Tuy nhiên cần lưu ý rằng có một số bài toán mà cách giải bằng quy hoạch động tỏ ra không thích hợp.

Nguyên lý tối ưu của R.Bellmam được phát biểu như sau: "*tối ưu bước thứ n bằng cách tối ưu tất cả con đường tiến đến bước n-1 và chọn con đường có tổng chi phí từ bước 1 đến bước n-1 và từ n-1 đến n là thấp nhất (nhiều nhất)*.



Chú ý rằng nguyên lý này được thừa nhận mà không chứng minh.

Trong thực tế, ta thường gặp một số bài toán tối ưu loại sau: Có một đại lượng *f* hình thành trong một quá trình gồm nhiều giai đoạn và ta chỉ quan tâm đến kết quả cuối cùng là giá trị của *f* phải lớn nhất hoặc nhỏ nhất, ta gọi chung là giá trị tối ưu của *f*. Giá trị của *f* phụ thuộc vào những đại lượng xuất hiện trong bài toán mà mỗi bộ giá trị của chúng được gọi là một *trạng thái* của hệ thống và phụ thuộc vào cách thức đạt được giá trị *f* trong từng giai đoạn mà mỗi cách tổ chức được gọi là một *điều khiển*. Đại lượng *f* thường được gọi là *hàm mục tiêu* và quá trình đạt được giá trị tối ưu của *f* được gọi là *quá trình điều khiển tối ưu*.

Nguyên lý tối ưu Bellman (cũng gọi là *nguyên lý Bellman*) có thể diễn giải theo một cách khác như sau: "Với mỗi quá trình điều khiển tối ưu, đối với trạng thái bắt đầu A_0, với trạng thái *A* trong quá trình đó, phần quá trình kể từ trạng thái *A* xem như trạng thái bắt đầu cũng là tối ưu".

Phương pháp tìm điều khiển tối ưu theo nguyên lý Bellman thường được gọi là *quy hoạch động*. Thuật ngữ này nói lên thực chất của quá trình điều khiển là động: Có thể trong một số bước đầu tiên lựa chọn điều khiển tối ưu dường như không tốt nhưng tựu chung cả quá trình lại là tốt nhất.

Ta có thể giải thích ý này qua bài toán sau: Cho một dãy N số nguyên A_1, A_2,…,A_N. Hãy tìm cách xoá đi một số ít nhất số hạng để dãy còn lại là đơn điệu hay nói cách khác hãy chọn một số nhiều nhất các số hạng sao cho dãy B gồm các số hạng đó theo trình tự xuất hiện trong dãy A là đơn điệu.

Quá trình chọn B được điều khiển qua N giai đoạn để đạt được mục tiêu là số lượng số hạng của dãy B là nhiều nhất, điều khiển ở giai đoạn i thể hiện việc chọn hay không chọn A_i vào dãy B.

Giả sử dãy đã cho là 1 8 10 2 4 6 7. Nếu ta chọn lần lượt 1, 8, 10 thì chỉ chọn được 3 số hạng nhưng nếu bỏ qua 8 và 10 thì ta chọn được 5 số hạng 1, 2, 4, 6, 7.

Khi giải một bài toán bằng cách "chia để trị" chuyển việc giải bài toán kích thước lớn về việc giải nhiều bài toán cùng kiểu có kích thước nhỏ hơn thì thuật toán này thường được thể hiện bằng các chương trình con đệ quy. Khi đó, trên thực tế, nhiều kết quả trung gian phải tính nhiều lần.

Vậy ý tưởng cơ bản của quy hoạch động thật đơn giản: Tránh tính toán lại mọi thứ hai lần, mà lưu giữ kết quả đã tìm kiếm được vào một bảng làm giả thiết cho việc tìm kiếm những kết quả của trường hợp sau. Chúng ta sẽ làm đầy dần giá trị của bảng này bởi các kết quả của những trường hợp trước đã được giải. Kết quả cuối cùng chính là kết quả của bài toán cần giải. Nói cách khác phương pháp quy hoạch động đã thể hiện sức mạnh của nguyên lý chia để trị đến cao độ.

Quy hoạch động là kỹ thuật thiết kế bottom-up (từ dưới lên). Nó được bắt đầu với những trường hợp con nhỏ nhất (thường là đơn giải nhất và giải được ngay). Bằng cách tổ hợp các kết quả đã có (*không phải tính lại*) của các trường hợp con, sẽ đạt đạt tới kết quả của trường hợp có kích thước lớn dần lên và tổng quát hơn, cho đến khi cuối cùng đạt tới lời giải của trường hợp tổng quát nhất.

Trong một số trường hợp, khi giải một bài toán A, trước hết ta tìm họ bài toán $A(p)$ phụ thuộc tham số p (có thể p là một véc tơ) mà $A(p_0)=A$ với p_0 là trạng thái ban đầu của bài toán A. Sau đó tìm cách giải họ bài toán $A(p)$ với tham số p bằng cách áp dụng nguyên lý tối ưu của Bellman. Cuối cùng cho $p=p_0$ sẽ nhận được kết quả của bài toán A ban đầu.

II. Các bước thược hiện quy hoạch động

***Bước* 1**: **Lập hệ thức**

Dựa vào nguyên lý tối ưu tìm cách chia quá trình giải bài toán thành từng giai đoạn, sau đó tìm hệ thức biểu diễn tương quan quyết định của bước đang xử lý với các bước đã xử lý trước đó. Hoặc tìm cách phân rã bài toán thành các "bài toán con" tương tự có kích thước nhỏ hơn, tìm hệ thức nêu quan hệ giữa kết quả bài toán kích thước đã cho với kết quả của các "bài toán con" cùng kiểu có kích thước nhỏ hơn của nó nhằm xây dựng phương trình truy toán (dạng hàm hoặc thủ tục đệ quy).

Về một cách xây dựng phương trình truy toán:

Ta chia việc giải bài toán thành n giai đoạn. Mỗi giai đoạn i có trạng thái ban đầu là t(i) và chịu tác động điều khiển d(i) sẽ biến thành trạng thái tiếp theo t(i+1) của giai đoạn i+1 (i=1,2,…,n-1). Theo nguyên lý tối ưu của Bellman thì việc tối ưu giai đoạn cuối cùng không làm ảnh hưởng đến kết quả toàn bài toán. Với trạng thái ban đầu là t(n) sau khi làm

giai đoạn n tốt nhất ta có trạng thái ban đầu của giai đoạn n-1 là t(n-1) và tác động điều khiển của giai đoạn n-1 là d(n-1), có thể tiếp tục xét đến giai đoạn n-1. Sau khi tối ưu giai đoạn n-1 ta lại có t(n-2) và d(n-2) và lại có thể tối ưu giai đoạn n-2 … cho đến khi các giai đoạn từ n giảm đến 1 được tối ưu thì coi như hoàn thành bài toán. Gọi giá trị tối ưu của bài toán tính đến giai đoạn k là F_k giá trị tối ưu của bài toán tính riêng ở giai đoạn k là G_k thì

$F_k = F_{k-1} + G_k$

Hay là: $F_1(t(k)) = \max_{\forall d(k)} \{G_k(t(k), d(k)) + F_{k-1}(t(k-1))\}$ (*)

***Bước* 2: Tổ chức dữ liệu và chương trình**

Tổ chức dữ liệu sao cho đạt các yêu cầu sau:

- Dữ liệu được tính toán dần theo các bước.
- Dữ liệu được lưu trữ để giảm lượng tính toán lặp lại.
- Kích thước miền nhớ dành cho lưu trữ dữ liệu càng nhỏ càng tốt, kiểu dữ liệu được chọn phù hợp, nên chọn đơn giản dễ truy cập.

Cụ thể

- Các giá trị của F_k thường được lưu trữ trong một bảng (mảng một chiều hoặc hai, ba, v.v… chiều).
- Cần lưu ý khởi trị các giá trị ban đầu của bảng cho thích hợp, đó là các kết quả của các bài toán con có kích cỡ nhỏ nhất của bài toán đang giải: $F_1(t(1)) = \max_{\forall d(1)} \{G_1(t(1), d(1)) + F_0(t(0))\}$
- Dựa vào công thức, phương trình truy toán (*) và các giá trị đã có trong bảng để tìm dần các giá trị còn lại của bảng.
- Ngoài ra còn cần mảng lưu trữ nghiệm tương ứng với các giá trị tối ưu trong từng gian đoạn.
- Dựa vào bảng lưu trữ nghiệm và bảng giá trị tối ưu trong từng giai đoạn đã xây dựng, tìm ra kết quả bài toán.

***Bước* 3: Làm tốt**

Làm tốt thuật toán bằng cách thu gọn hệ thức (*) và giảm kích thước miền nhớ. Thường tìm cách dùng mảng một chiều thay cho mảng hai chiều nếu giá trị một dòng (hoặc cột) của mảng hai chiều chỉ phụ thuộc một dòng (hoặc cột) kề trước.

Trong một số trường hợp có thể thay mảng hai chiều với các giá trị phần tử chỉ nhận giá trị 0, 1 bởi mảng hai chiều mới bằng cách dùng kỹ thuật quản lý bit.

III. Các thao tác tổng quát của phương pháp QHĐ

1. Xây dựng hàm QHĐ

2. Lập bảng lưu lại giá trị của hàm

3. Tính các giá trị ban đầu của bảng

4. Tính các giá trị còn lại theo kích thước tăng dần của bảng cho đến khi đạt được giá trị tối ưu cần tìm

5. Dùng bảng lưu để truy xuất lời giải tối ưu.

IV. Hạn chế của phương pháp QHĐ

Việc tìm công thức, phương trình truy toán hoặc tìm cách phân rã bài toán nhiều khi đòi hỏi sự phân tích tổng hợp rất công phu, dễ sai sót, khó nhận ra như thế nào là thích hợp, đòi hỏi nhiều thời gian suy nghĩ. Đồng thời không phải lúc nào kết hợp lời giải của các bài toán con cũng cho kết quả của bài toán lớn hơn.

Khi bảng lưu trữ đòi hỏi mảng hai, ba chiều … thì khó có thể xử lý dữ liệu với kích cỡ mỗi chiều lớn hàng trăm.

Có những bài toán không thể giải được bằng quy hoạch động.

CHƯƠNG 2: NHẬN DIỆN CÁC BÀI TOÁN CÓ THỂ GIẢI ĐƯỢC BẰNG PP QHĐ

I. Các bài toán không phải là bài toán tối ưu hóa.

Các bài toán có thể áp dụng được phương pháp QHĐ thì phải có tính chất: "***các bài toán con phủ chồng***".

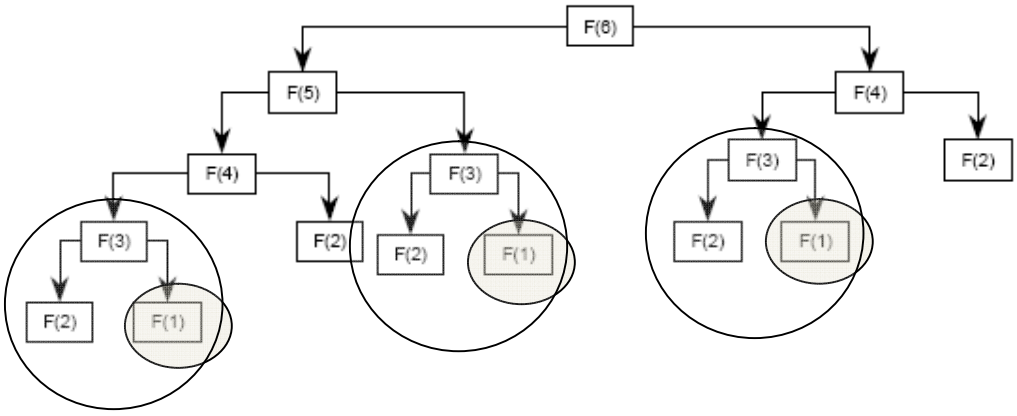
Có nghĩa là một bài toán có thể áp dụng phương pháp QHĐ thì một thuật toán đệ quy cho bài toán sẽ giải quyết lặp lại các bài toán con tương tự, thay vì luôn phát sinh bài toán con mới. Khi một thuật toán đệ quy ghé thăm hoài cùng một bài toán con, ta nói rằng bài toán có "các bài toán con phủ chồng". Ngược lại bài toán thích hợp với cách tiếp cận chia để trị thường phát sinh các bài toán con hoàn toàn mới tại mỗi bước đệ quy. Các thuật toán lập trình động thường vận dụng các bài toán phủ chồng bằng cách giải quyết từng bài toán con một rồi lưu trữ kết quả trong một bảng ở đó nó có thể được tra cứu khi cần.

Ví dụ 1: Bài toán tìm tính phần tử thứ n của dãy Fibonacci:

Dãy Fibonacci là dãy vô hạn các số tự nhiên bắt đầu bằng hai phần tử 0 và 1, các phần tử sau đó được thiết lập theo quy tắc *mỗi phần tử luôn bằng tổng hai phần tử trước nó*. Công thức truy hồi của dãy Fibonacci là:

$$F_n := F(n) := \begin{cases} 0, & \text{khi } n = 0; \\ 1, & \text{khi } n = 1; \\ F(n-1) + F(n-2) & \text{khi } n > 1. \end{cases}$$

Sơ đồ gọi đệ quy với bài toán tính F(6):



Ta thấy bài toán F(6) sẽ gọi bài toán F(5) và bài toán F(4). Cả bài toán F(5) và bài toán F(4) đều gọi bài toán F(3), hai bài toán con F(4) và F(3) lại cùng gọi bài toán con F(2)…ta gọi đó là tính chất "Bài toán con phủ chồng".

Một cài đặt đơn giản của một hàm tính phần tử thứ *n* của dãy Fibonacci, trực tiếp dựa theo định nghĩa toán học. Cài đặt này thực hiện rất nhiều tính toán thừa.

Int *fib(n)*

```
{
    if ((n = =0)||( n = 1)) return n;
    else return fib(n-1) + fib(n-2);
}
```

Lưu ý rằng nếu ta gọi, chẳng hạn: fib(5), ta sẽ tạo ra một cây các lời gọi hàm, trong đó các hàm của cùng một giá trị được gọi nhiều lần:

1. fib(5)
2. fib(4) + fib(3)
3. (fib(3) + fib(2)) + (fib(2) + fib(1))
4. ((fib(2) + fib(1)) + (fib(1) + fib(0))) + ((fib(1) + fib(0)) + fib(1))
5. (((fib(1) + fib(0)) + fib(1)) + (fib(1) + fib(0))) + ((fib(1) + fib(0)) + fib(1))

Áp dụng thuật toán QHĐ: Ta dùng một bảng để lưu trữ tất cả các bài toán con, như thế mỗi bài toán con chỉ phải tính một lần.

Cài đặt:

```
{
        fib[0]:=0;
        fib[1]:=1;
        for (i=2;i<=n;i++)  f[i]:=f[i-2] + f[i-1];
}
```

Ví dụ 2: Bài Toán: **Mê Cung** (đề chọn đội tuyển tin Olympic 10 trường THPT chuyên Quang Trung năm 2010- 2011).

Phát biểu bài toán: Trong một chuyến thám hiểm mạo hiểm, một đoàn thám hiểm không may lọt vào một mê cung với nhiều cạm bẫy. Trong mê cung đó chỉ có một lối ra duy nhất, lối ra bao gồm các ô hình vuông được xếp thành một hàng dài. Muốn đi được ra ngoài mọi người phải bước qua một hàng các ô hình vuông đó và phải bước theo quy tắc sau:

- Quy tắc 1: Mỗi bước chỉ có thể bước một ô hoặc hai ô hoặc ba ô.
- Quy tắc 2: Từ người thứ 2 trở đi bước theo quy tắc 1 và không được trùng với các cách bước của tất cả những người trước đó.

Hỏi đoàn thám hiểm đó còn lại tối thiểu bao nhiêu người không thể thoát ra khỏi mê cung đó được.

Input Data:

- Dòng 1 ghi một số nguyên m ($m \leq 10^{18}$) là số người trong đoàn thám hiểm.
- Dòng 2 ghi một số nguyên n ($n \leq 70$) là tổng số ô vuông.

Output Data: Gồm 1 số nguyên duy nhất là số người còn lại tối thiểu không thể thoát ra khỏi mê cung

Example:

Input.inp	Output.out
20 5	7

Thực chất của bài toán là tìm xem có bao nhiêu cách bước ra ngoài.

Công thức truy hồi cho bài toán này được tính như sau:

+ Để bước lên ô thứ n chúng ta có 3 cách bước:

Cách 1: Bước tới ô thứ n-3 rồi bước 3 bước nữa

Cách 2: Bước tới ô thứ n-2 rồi bước 2 bước nữa

Cách 3: Bước tới ô thứ n-1 rồi bước 1 bước nữa

+ Theo nguyên lý cộng chúng ta có tổng số cách bước tới ô thứ n: **F(n) = F(n-3)+F(n-2) + F(n-1)** trong đó: F(1)= 1; F(2)= 2; F3(=4).

Chúng ta thấy bài toán xuất hiện rất nhiều "bài toán con phủ chồng" như bài toán tính phần tử thứ n của dãy Fibonacci. Chúng ta có thể áp dụng phương pháp QHĐ cho bài toán này để đạt được hiệu quả cao.

Mã nguồn:

```
#include<iostream>
#include<fstream>
#include<iostream>
long int i,n,m;
long int f[101];
using namespace std;
//-------------------------------------------------------/
void inputdata()
{
   ifstream fi("mecung.inp");
   fi>>m>>n;
   fi.close();
}
//-------------------------------------------------------/
void outputdata()
{
   ofstream fo("mecung.out");
   if(m>f[n]) fo<<m-f[n];
   else fo<<0;
   fo.close();
}
//-------------------------------------------------------/
void process()
```

```
{
   f[1]=1; f[2]=2; f[3]=4;
   for(i=4;i<=n;i++) f[i]=f[i-1]+f[i-2]+f[i-3];
}
//-------------------------------------------------------------/

int main()
{
   inputdata();
   process();
   outputdata();
}
```

II. Đối với các bài toán tối ưu:

Các bài toán tối ưu có thể áp dụng phương pháp QHĐ thì phải thỏa mãn 2 tính chất sau:

+ **Tính chất 1**: Các bài toán con phủ chồng

+ **Tính chất 2**: Cấu trúc con tối ưu: *Cấu trúc con tối ưu* có nghĩa là các lời giải tối ưu cho các bài toán con có thể được sử dụng để tìm các lời giải tối ưu cho bài toán toàn cục.

Ví dụ 1: Bài toán: Dãy con chung dài nhất:

Phát biểu bài toán: Xâu ký tự A được gọi là xâu con của xâu ký tự B nếu ta có thể xoá đi một số ký tự trong xâu B để được xâu A.

Cho biết hai xâu ký tự X và Y, hãy tìm xâu ký tự Z có độ dài lớn nhất và là con của cả X và Y.

Input

Dòng 1: chứa xâu X

Dòng 2: chứa xâu Y

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

Ví dụ:

Input.inp	Output.Out
ALGORITHM LOGARITHM	7

Ta đánh giá:

Tính chất 1: Ta có thể dễ thấy tính chất các bài toán con phủ chồng trong bài toán LCS (độ dài xâu con chung dài nhất). Để tìm một LCS của X và Y, có thể ta cần tìm các LCS của X và Y_{n-1} và của X_{m-1} và Y. Nhưng mỗi bài toán con này đều có bài toán cháu tìm LCS của X_{m-1} và Y_{n-1}. Nhiều bài toán con khác chia sẻ các bài toán cháu.

Tính chất 2: Ta có nhận xét như sau:

1. Nếu $x_m = y_n$, thì $z_k = x_m = y_n$ và Z_{k-1} là một LCS của X_{m-1} và Y_{n-1}.
2. Nếu $x_m \neq y_n$, thì $z_k \neq x_m$ hàm ý Z là một LCS của X_{m-1} và Y.
3. Nếu $x_m \neq y_n$, thì $z_k \neq y_n$ hàm ý Z là một LCS của X và Y_{n-1}.

Vậy ta hoàn toàn có thể tính được LCS của X và Y khi biết một LCS của X_{m-1} và Y_{n-1} hoặc LCS của X_{m-1} và Y hoặc LCS của X và Y_{n-1}.

Vậy bài toán thỏa mãn cả hai tính chất do vậy hoàn toàn có thể giải được bằng phương pháp QHĐ.

Hướng dẫn giải chi tiết:

Bước 1: Xác định đặc điểm của dãy con chung dài nhất (giải pháp tối ưu của bài toán)

Cho $X = <x_1, x_2,\ldots,x_m>$ và $Y = <y_1, y_2,\ldots,y_n>$ là các dãy, và $Z = <z_1, z_2,\ldots,z_k>$ là một dãy LCS bất kỳ của X và Y.

1. Nếu $x_m = y_n$, thì $z_k = x_m = y_n$ và Z_{k-1} là một LCS của X_{m-1} và Y_{n-1}.
2. Nếu $x_m \neq y_n$, thì $z_k \neq x_m$ hàm ý Z là một LCS của X_{m-1} và Y.
3. Nếu $x_m \neq y_n$, thì $z_k \neq y_n$ hàm ý Z là một LCS của X và Y_{n-1}.

Chứng minh:

1. Nếu $z_k \neq x_m$, thì ta có thể chắp $x_m = y_n$ vào Z để có được một dãy con chung của X và Y có chiều dài k+1, mâu thuẫn với giả thiết cho rằng Z là LCS của X và Y. Gả sử có một dãy con chung W của X_{m-1} và Y_{n-1} có chiều dài lớn hơn k-1. Như vậy việc chắp $x_m = y_n$ vào W sẽ tạo ra một dãy con chung của X và Y có chiều dài lớn hơn k, là một sự mâu thuẫn.
2. Nếu $z_k \neq x_m$, thì Z là một dãy con chung của X_{m-1} và Y. Nếu có một dãy con chung W của X_{m-1} và Y có chiều dài lớn hơn k, thì W cũng sẽ là sẽ là một dãy con chung của X và Y, mẫu thuẫn với giả thiết cho rằng Z là LCS của X và Y
3. Chứng minh đối xứng với (2)

Như vậy: Bài toán LCS có một tính chất cấu trúc con tối ưu (từ cấu trúc con tối ưu → dẫn đến bài toán tối ưu).

Bước 2: **Một giải pháp đệ quy cho các bài toán con**

Ta có thể dễ thấy tính chất các bài toán con phủ chồng trong bài toán LCS. Để tìm một LCS của X và Y, có thể ta cần tìm các LCS của X và Y_{n-1} và của X_{m-1} và Y. Nhưng mỗi bài toán con này đều có bài toán cháu tìm LCS của X_{m-1} và Y_{n-1}. Nhiều bài toán con khác chia sẻ các bài toán cháu.

Ta định nghĩa c[i,j] là chiều dài của một LCS của các dãy X_i và Y_j. Nếu i=0 hoặc j=0, một trong các dãy sẽ có chiều dài 0, do đó LCS có chiều dài 0. Cấu trúc con tối ưu của bài toán LCS cho công thức đệ quy:

$$c[i,j] = \begin{cases} 0 & if\ i = 0\ or\ j = 0 \\ c[i-1, j-1] + 1 & if\ i,j > 0\ and\ xi = yj \\ \max(c[i, j-1], c[i-1, j]) & if\ i,j > 0\ and\ xi \neq yi \end{cases}$$

Bước 3: **Tính toán chiều dài của một LCS**

Dựa trên công thức trên, ta có thể dễ dàng viết một thuật toán đệ quy thời gian mũ để tính toán chiều dài của một LCS cũa hai dãy. Tuy nhiên chỉ có O(mn) bài toán riêng biệt, nên ta có thể dùng lập trình động để tính toán các giải pháp từ dưới lên.

LCS – LENGTH (X, Y)

```
1.   m ← length (X)
2.   n ← length (Y)
3.   for i ← 1 to m
4.        do c[i,0] ← 0
5.   for j ← 1 to n
6.        do c[0,j] ← 0
7.   for i ← 1 to m
8.        do for j ← 1 to n
9.             do if xi =yj
10.            Then c[i,j] ← c[i-1, j-1]+1
11.            B[i,j] ← "↖"
12.       else if c[i-1,j] ≥ c[I,j-1]
13.       then c[i,i] ← c[i-1, j]
14.            B[i,j] ← "↑"
15.       else c[i,i] ← c[i, j-1]
16.            B[i,j] ← "←"
17. Return c và b
```

Thủ tục LCS – LENGTH tiếp nhận hai dãy X = <x_1, x_2,…,x_m> và Y = <y_1, y_2,…,y_n> làm nhập liệu. Nó lưu trữ c[i,j] giá trị trong một bảng c[0..m, 0..n] có các việc nhập được tính toán theo thứ tự chính là hàng (ghĩa là: Hàng đầu tiên của c được điền từ trái qua phải, sau đó đến hàng thứ 2…). Nó cũng duy trì bảng b[1..m, 1..n] để rút gọn việc kiến tạo một giải pháp tối ưu. Theo quan sát, b[i,j] trỏ đến việc nhập bảng tương ứng với giải pháp bài toán con tối ưu được chọn khi tính toán c[i,j]. Thủ tục trả về bảng b và c; c[m, n] chứa chiều dài của một LCS của X và Y.

Bước 4: **Kiến tạo một LCS**

		A	L	G	O	R	I	T	H	M
	0	0	0	0	0	0	0	0	0	0
L	0	0	1	1	1	1	1	1	1	1
O	0	0	1	1	2	2	2	2	2	2
G	0	0	1	2	2	2	2	2	2	2
A	0	1	1	2	2	2	2	2	2	2
R	0	1	1	2	2	3	3	3	3	3
I	0	1	1	2	2	3	4	4	4	4
T	0	1	1	2	2	3	4	5	5	5
H	0	1	1	2	2	3	4	5	6	6
M	0	1	1	2	2	3	4	5	6	7

Bảng b

Bảng b mà thủ tục LCS – LENGTH trả về có thể được dùng để nhanh chóng kiến tạo một LCS của X = <$x_1, x_2,\ldots,x_m$> và Y = <$y_1, y_2,\ldots,y_n$>. Ta đơn giản bắt đầu từ b[m,n] và rà qua bảng theo các mũi tên. Mỗi khi ta gặp một *"↖" trong việc nhập b[i,j], nó hàm ý rằng* $x_i = y_j$ *là một thành phần của LCS. Phương pháp này in ra một đề xuất đảo ngược. Thủ tục đệ quy sau đây in ra một LCS đúng đắn.*

```
PRINT – LCS (b,X,i, j)
1.    if i =0 or j=0
2.          Then return
3.    if b[i,j] =  "↖"
4.          Then Print – LCS (b,X,i-1,j-1);
5.          Print x_i
6.    else if b[i,j] = "↑"
7.          Then Print – LCS (b,X,i-1,j);
8.    else Print – LCS (b,X,i,j-1);
```

Thủ tục PRINT – LCS (b,X,i, j) mất một thời gian O(m+n), bởi ít nhất một trong số biến i, j được giảm lượng trong mỗi giai đoạn đệ quy.

Bước 5: **Cải thiện mã**

Sau khi phát triển một thuật toán, bạn thường thấy có thể cải thiện thời gian hoặc không gian mà nó sử dụng. Điều này hoàn toàn dễ dàng thực hiện đối với các thuật toán lập trình động không phức tạp. Vài thay đổi có thể rút gọn mã và cải thiện các thừa số bất biến, các thay đổi có thể mang lại các khoản tiết kiệm đáng kể về thời gian và không gian.

Ví dụ: Có thể loại bỏ bảng b. Mỗi việc nhập c[i,j] chỉ tùy thuộc vào ba việc nhập bảng c: c[i-1, j-1], c[i-1,j], c[i, j-1] Cho giá trị c[i,j], ta có thể xác định trong O(1) thời gian giá trị nào trong 3 giá trị này được dùng để tính toán c[i,j], mà không kiểm tra bảng b. Như vậy, ta có thể dùng một thủ tục O(m +n) thời gian tương tự như thủ tục PRINT – LCS (b,X,i, j). Tuy tiết kiệm O(mn) không gian bằng phương pháp này song yêu cầu không gian phụ để để tính một LCS không giảm theo tiệm cận, bởi ta vẫn cần O(mn) không gian cho bảng c.

Tuy nhiên ta có thể rút gọn các yêu cầu không gian tiệm cận LCS – LENGTH, bởi nó chỉ cần hai hàng của bảng c vào một thời điểm: Hàng đang được tính toán và hàng trước đó. Áp dụng việc cải thiện kiểu này nếu ta chỉ cần tính chiều dài của LCS. Nếu cần kiến tạo lại các thành phần của một LCS, bảng nhỏ hơn không lưu giữ đủ thông tin để rà lại các bước của chúng ta trong O(m+n) thời gian.

Ví dụ 2: **Bài toán cái túi:**

Phát biểu bài toán: Cho N đồ vật, vật i có khối lượng W[i] và giá trị là V[i]. Một cái túi có thể chịu được khối lượng tối đa là M, quá thì sẽ rách. Hãy tìm cách nhét 1 số đồ vật vào trong túi sao cho túi không bị rách và tổng giá trị của các đồ vật nhét vào là lớn nhất.

Input

Dòng đầu tiên là số nguyên T là số bộ test. ($1 \le T \le 40$)

Mỗi bộ test sẽ có định dạng như sau:

+ Dòng 1: 2 số nguyên dương N, M ($1 \le N \le 10000$, $1 \le M \le 1000$).
+ Dòng 2: Gồm N số nguyên là W[i] ($1 \le W[i] \le 1000$).
+ Dòng 3: Gồm N số nguyên là V[i] ($1 \le V[i] \le 10000$).

Output

Với mỗi bộ test:

+ Dòng đầu tiên ghi ra giá trị lớn nhất có thể đạt được và số K là số đồ vật lựa chọn.
+ Dòng thứ 2 ghi ra chỉ số của K đồ vật được chọn.

Input.inp	Output.Out
1 3 4 1 2 3 4 5 6	10 2 1 3

Ta đánh giá:

+ **Tính chất 1**: ***Bài toán con phủ chồng***: Khi ta xét một vật để xếp vào túi, ta phải so sánh giải pháp cho bài toán khi xếp vật được xếp vào túi với giải pháp cho bài toán con trước khi xếp vật đó vào cái túi. Bài toán được lập theo cách này gây ra nhiều bài toán con phủ chồng.

+ **Tính Chất 2**: ***Cấu trúc con tối ưu***: Ta hãy xét tải trọng có giá trị lớn nhất cân nặng tối đa W. Nếu gỡ bỏ mục j ra khỏi tải trọng này, tải trọng còn lại phải là tải trọng có giá trị nhất cân nặng tối đa $W - w_j$ mà ta có thể xếp từ n-1 vật ban đầu trừ vật j.