

## MỤC LỤC

<b>I.</b>	<b>CƠ SỞ LÝ THUYẾT VỀ QUY HOẠCH ĐỘNG .....</b>	<b>2</b>
1.1.	<b>1.1. Một số khái niệm:.....</b>	<b>2</b>
1.1.1.	<b>Công thức truy hồi (Hệ thức truy hồi): .....</b>	<b>2</b>
1.2.	<b>1.2. Phương pháp quy hoạch động: .....</b>	<b>3</b>
1.3.	<b>1.3. Áp dụng quy hoạch động.....</b>	<b>5</b>
1.4.	<b>1.4. Các bước giải bài toán tối ưu bằng quy hoạch động .....</b>	<b>6</b>
1.5.	<b>1.5. Ưu điểm và hạn chế của phương pháp quy hoạch động .....</b>	<b>7</b>
<b>II.</b>	<b>MỘT SỐ BÀI TOÁN QUY HOẠCH ĐỘNG CƠ BẢN DẠY</b>	
	<b>HỌC SINH CHUYÊN TIN TRUNG HỌC PHỔ THÔNG .....</b>	<b>9</b>
1.6.	<b>2.1. Một số bài toán đơn giản .....</b>	<b>9</b>
1.7.	<b>2.2. Một số bài toán điển hình .....</b>	<b>21</b>
1.8.	<b>2.3. Giới thiệu một số bài toán mở rộng.....</b>	<b>28</b>
<b>C.</b>	<b>KẾT LUẬN .....</b>	<b>34</b>
	<b>TÀI LIỆU THAM KHẢO .....</b>	<b>35</b>

## I. CƠ SỞ LÝ THUYẾT VỀ QUY HOẠCH ĐỘNG

### 1.1. Một số khái niệm:

#### 1.1.1. Bài toán tối ưu:

##### *a. Khái niệm:*

Bài toán tối ưu gồm có 1 hàm  $f$  gọi là hàm mục tiêu hay hàm đánh giá; các hàm  $g_1, g_2, \dots, g_n$  cho giá trị logic gọi là hàm ràng buộc. Yêu cầu của bài toán là tìm một cấu hình  $x$  thỏa mãn tất cả các ràng buộc  $g_1, g_2, \dots, g_n: g_i(x) = \text{TRUE} \ (\forall i: 1 \leq i \leq n)$  và  $x$  là tốt nhất, theo nghĩa không tồn tại một cấu hình  $y$  nào khác thỏa mãn các hàm ràng buộc mà  $f(y)$  tốt hơn  $f(x)$ .

Bài toán tối ưu là bài toán thường có nhiều nghiệm chấp nhận được và mỗi nghiệm có một giá trị đánh giá. Mục tiêu đặt ra là tìm nghiệm tối ưu, đó là nghiệm có giá trị đánh giá lớn nhất hoặc nhỏ nhất (tối ưu).

##### *b. Một số ví dụ về bài toán tối ưu:*

###### *Ví dụ 1.1:*

Trong mặt phẳng tọa độ Oxy tìm tọa độ  $(x, y)$  để tổng  $x + y$  đạt giá trị lớn nhất mà  $x^2 + y^2 \leq 1$ .

Ở bài toán trên ta thấy:

Hàm mục tiêu :  $x + y \rightarrow \max$

Hàm ràng buộc :  $x^2 + y^2 \leq 1$ .

###### *Ví dụ 1.2: Bài toán xếp Ba lô*

Có một ba lô có thể chứa tối đa trọng lượng  $M$  và có  $n$  đồ vật ( $n \leq 100$ ), mỗi đồ vật có trọng lượng  $w_i$  và giá trị  $b_i$ ;  $M, w_i, b_i$  là các số nguyên. Hãy chọn và xếp các đồ vật vào ba lô để tổng giá trị của ba lô là lớn nhất.

Với bài toán trên ta thấy:

Hàm mục tiêu:  $\sum b_i \rightarrow \max, i = 1, 2, \dots, n$

Hàm ràng buộc:  $\sum w_i \rightarrow M, i = 1, 2, \dots, n$

*Tóm lại, bài toán tối ưu rất phong phú, đa dạng, được ứng dụng nhiều trong thực tế nhưng chúng ta cũng cần biết rằng vẫn tồn tại một số các bài toán tối ưu là không giải được hoặc chưa giải được.*

#### 1.1.1. Công thức truy hồi (Hệ thức truy hồi):

*Khái niệm:* Công thức truy hồi là công thức thể hiện quan hệ giữa các bước trong một bài toán và kết quả của bước sau thường dựa vào kết quả của các

bước trước đó. Kết quả của bước cuối cùng là kết quả của bài toán.

## **1.2. Phương pháp quy hoạch động:**

### **1.2.1. Phương pháp chia để trị:**

“Chia để trị” là việc tách bài toán ban đầu thành các bài toán con độc lập, sau đó giải các bài toán con này rồi tổ hợp dần lời giải từ bài toán con nhỏ nhất đến bài toán ban đầu.

Phương pháp chia để trị là phương pháp thông dụng nhất trong Tin học.

Phương pháp chia để trị thường được áp dụng cho những bài toán có bản chất đệ quy (bài toán P có bản chất đệ quy thì bài toán P có thể được giải bằng lời giải của bài toán P' có dạng giống như P. Tuy nhiên, chúng ta cần lưu ý rằng: P' tuy có dạng giống như P nhưng theo một nghĩa nào đó P' phải nhỏ hơn P, để giải hơn P và việc giải nó không cần dùng đến P).

Giải thuật dùng để giải bài toán có bản chất đệ quy gọi là giải thuật đệ quy.

### **1.2.2. Khái niệm về phương pháp quy hoạch động:**

#### **a. Khái niệm:**

Phương pháp quy hoạch động (Dynamic Programming) là một kỹ thuật nhằm đơn giản hóa việc tính toán các công thức truy hồi bằng cách lưu toàn bộ hay một phần kết quả tính toán tại mỗi bước trước đó với mục đích sử dụng lại.

Như vậy, Quy hoạch động = Chia để trị + Mảng (lưu lại kết quả).

Phương pháp quy hoạch động do nhà toán học người Mỹ Richard Bellman (1920-1984) phát minh năm 1953. Phương pháp này dùng để giải các bài toán tối ưu có bản chất đệ quy, tức là 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.

Điểm khác nhau cơ bản giữa quy hoạch động và phương pháp đệ quy là :

Phương pháp đệ quy giải quyết bài toán theo hướng top-down, nghĩa là để giải bài toán ban đầu, ta phải đi giải tất cả các bài toán con của nó. Đây là một phương pháp hay, tuy nhiên phương pháp này sẽ gặp hạn chế về mặt thời gian, tốc độ do phải tính đi tính lại nhiều lần một số bài toán con giống nhau nào đó.

Phương pháp quy hoạch động sử dụng nguyên lý bottom-up, nghĩa là "đi từ dưới lên". Đầu tiên, ta sẽ phải giải các bài toán con đơn giản nhất, có thể tìm ngay ra nghiệm. Sau đó kết hợp các bài toán con này lại để tìm lời giải cho bài toán lớn hơn và cứ như thế cho đến khi giải được bài toán yêu cầu. Với phương

pháp này, mỗi bài toán con sau khi giải xong đều được lưu trữ lại và đem ra sử dụng nếu cần. Do đó tiết kiệm bộ nhớ và cải thiện được tốc độ.

### ***b. Đặc điểm chung của quy hoạch động:***

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).

Quy hoạch động cần phải có bảng phương án

Ý tưởng cơ bản của phương pháp quy hoạch động là tránh tính toán lại các bài toán con đã xét, nói cách khác phương pháp quy hoạch động đã thể hiện sức mạnh của nguyên lý chia để trị đến cao độ.

*Tóm lại:*

- Quy hoạch động dùng để giải quyết bài toán tối ưu theo nguyên lý “chia để trị” nhưng thực chất là một phương pháp cải tiến hơn của phương pháp giải quyết bài toán theo hướng đệ quy.

- Quy hoạch động làm giảm độ phức tạp, giảm thời gian giải quyết bài

- Quy hoạch động thường **tiếp cận theo hướng từ dưới lên (Bottom – up)**

### **1.2.3. Các cách thực hiện phương pháp quy hoạch động**

Quy hoạch động thường dùng một trong 2 cách tiếp cận sau:

- Tiếp cận từ dưới lên (bottom up)

- Tiếp cận từ trên xuống (top down)

Cách tiếp cận từ dưới lên hiệu quả hơn nên cách tiếp cận từ dưới lên (bottom up) thường được sử dụng nhiều hơn.

### **1.2.4. Các yêu cầu của một bài toán tối ưu sử dụng được phương pháp quy hoạch động**

Một bài toán tối ưu muốn giải được bằng phương pháp quy hoạch động khi bài toán tối ưu đó có các đặc điểm dưới đây:

- 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 kết quả (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.

### 1.3. Áp dụng quy hoạch động

#### 1.3.1. Nguyên lý tối ưu Bellman:

Ta biết rằng quy hoạch động thường dùng để giải bài toán tối ưu- bài toán yêu cầu tìm một giải pháp tốt nhất trong các giải pháp có thể tìm được. Cơ sở của quy hoạch động trong bài toán tối ưu là nguyên lý tối ưu Bellman.

Nguyên lý tối ưu Bellman được phát biểu như sau: *“Dãy tối ưu các quyết định trong một quá trình quyết định nhiều giai đoạn có thuộc tính là dù trạng thái và các quyết định ban đầu bất kể như thế nào, những quyết định còn lại phải tạo thành một cách giải quyết tối ưu không phụ thuộc vào trạng thái được sinh ra từ những quyết định ban đầu”*.

Hay nói cách khác: *“Giải pháp tối ưu cho bài toán  $P$  cần chứa giải pháp tối ưu cho các bài toán con của  $P$ ”*.

Do vậy khi giải bài toán theo quy hoạch động nếu dùng một phương pháp gồm nhiều bước tiến hành thì điều kiện cần để giải pháp này tối ưu là nó được xây dựng từ nghiệm tối ưu của những bước trước.

#### 1.3.2. Hàm mục tiêu

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.

Như vậy, nguyên lý Bellman có thể phát biểu cơ bản lại 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”.

Ví dụ, ta có thể giải thích ý này qua bài toán sau: Cho một dãy  $N$  số nguyên  $A_1, A_2, \dots, A_N$ . Hãy tìm cách xóa 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.

Trong một số trường hợp, khi giải một bài toán A, trước hết ta tìm cho 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 cho 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.

#### **1.4. Các bước giải bài toán tối ưu bằng quy hoạch động**

##### ***Bước 1: Lập công thức truy hồi***

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).

*Diễn hình 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  $i+1$  ( $i=1,2,\dots,n-1$ ). 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  $F_k$  và giá trị tối ưu của bài toán tính riêng ở giai đoạn  $k$  là  $G_k$  thì ta có :

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

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

### **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 để 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:

$$F_1(t(1)) = \max_{\forall d(1)} \{G_1(t(1), d(1)) + F_0(t(0))\}$$

### **Bước 3: Truy vết, tìm nghiệm của bài toán dựa vào bảng phương án**

- 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 4: Làm tốt thuật toán**

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.

## **1.5. Ưu điểm và hạn chế của phương pháp quy hoạch động**

### **1.5.1. Ưu điểm:**

Tiết kiệm được thời gian thực hiện vì không cần phải tính đi tính lại nhiều lần một số bài toán con giống nhau.

### 1.5.2. Hạn chế

- Việc tìm công thức truy hồi 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 đến hàng trăm.

- Có những bài toán tối ưu không thể giải được bằng quy hoạch động

Như vậy, quy hoạch động là một phương pháp hay và hiệu quả, nó có thể giải được hầu hết các bài toán tối ưu. Tuy nhiên, khi giải bài toán theo hướng quy hoạch động, ta cần phải tìm công thức truy hồi thật chính xác và chứng minh độ chính xác tin cậy của nó.

Tuy nhiên, không phải lúc nào việc kết hợp các bài toán con cũng cho ta kết quả của bài toán lớn hơn. Hay nói cách khác là việc tìm kiếm "công thức truy hồi" rất khó khăn. Ngoài ra, số lượng các bài toán con cần lưu trữ có thể rất lớn, không chấp nhận được vì dữ liệu và bộ nhớ máy tính không cho phép. Để biết được bài toán có thể giải bằng phương pháp quy hoạch động hay không, ta có thể tự đặt câu hỏi như: "Một nghiệm tối ưu của bài toán lớn có phải là phối hợp các nghiệm tối ưu của các bài toán con hay không?" và "Liệu có thể nào lưu trữ được nghiệm các bài toán con dưới một hình thức nào đó để phối hợp tìm nghiệm của bài toán lớn?"



## II. MỘT SỐ BÀI TOÁN QUY HOẠCH ĐỘNG CƠ BẢN DẠY HỌC SINH CHUYÊN TIN TRUNG HỌC PHỔ THÔNG

Quy hoạch động (DP) thường là kỹ thuật giải quyết vấn đề đòi hỏi cần có kiến thức nền và kỹ năng giải quyết một số bài toán liên quan, đó là các thuật toán tìm kiếm, lặp, đệ quy, chia để trị và tham lam. Do đó cần đảm bảo đã nắm vững các thuật toán trên, đặc biệt là lặp và đệ quy.

Như đã đề cập trong chương I, các kỹ năng quan trọng cần phải phát triển để thành thạo DP là khả năng xác định trạng thái của vấn đề và xác định mối quan hệ hoặc sự chuyển tiếp giữa các vấn đề hiện tại và các vấn đề phụ của chúng, tương tự như việc theo dõi đệ quy. Trên thực tế, các vấn đề DP với các ràng buộc về kích thước đầu vào nhỏ có thể đã được giải quyết bằng phương pháp brute force ngược đệ quy. DP chủ yếu được sử dụng để giải quyết khá hiệu quả các bài toán tối ưu hóa và các bài toán đếm. Nếu một bài toán có nội dung yêu cầu xác định giá trị “tối đa” hoặc “tối thiểu” hoặc “đếm các cách để thực hiện điều đó”, thì khả năng cao đó là vấn đề DP. Phần lớn các bài toán chỉ yêu cầu giá trị tìm tối ưu / tổng chứ không phải bản thân giải pháp tối ưu, điều này thường làm cho vấn đề dễ giải hơn bằng cách loại bỏ nhu cầu chạy lại và đưa ra giải pháp. Tuy nhiên, một số bài toán DP khó hơn cũng yêu cầu trả về giải pháp tối ưu.

Trong chương này, chúng tôi xin giới thiệu một số bài toán quy hoạch động cơ bản, điển hình, nâng cao sử dụng giảng dạy cho học sinh chuyên tin khối THPT.

### 2.1. Một số bài toán đơn giản

#### 2.1.1. Bài 1. Dãy Fibonacci:

Đề bài: Dãy Fibonacci là một dãy số nguyên 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, ... Theo toán học, dãy Fibonacci  $F_n$  được định nghĩa bắt đầu với  $F_0=1$ ,  $F_1=1$  và bởi sự liên quan giữa các phần tử liên tiếp của dãy số:  $F_i = F_{i-1} + F_{i-2}$ , với  $i > 1$ . Cho số  $n$ , hãy in ra màn hình số thứ  $n$  của dãy Fibonacci.

*Ví dụ 1:*

Input :  $n = 2$

Output : 1

*Ví dụ 2:*

Input :  $n = 9$

Output : 34

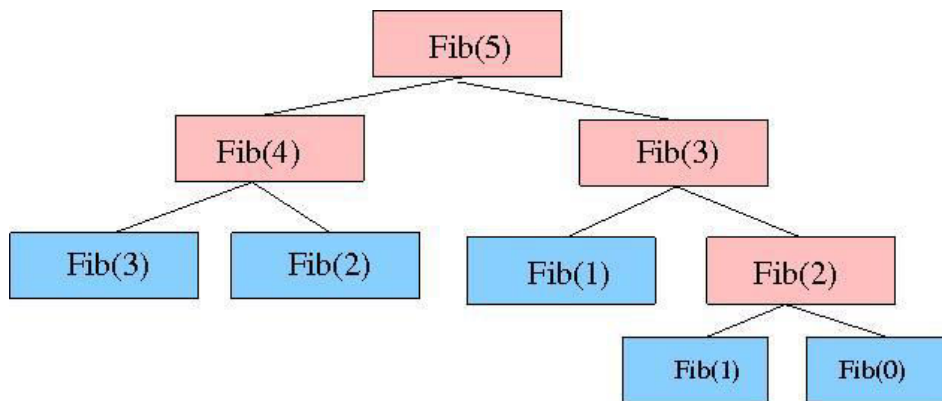
Hướng dẫn giải thuật:

Cách 1: Sử dụng đệ quy. Độ phức tạp hàm mũ  $O(2^n)$

Code tham khảo

```
int fib(int n)
{
    if (n <= 1)
        return n;
    return fib(n-1) + fib(n-2);
}
```

Cách 2: Sử dụng phương pháp quy hoạch động. Độ phức tạp  $O(n)$ . Độ lớn không gian lưu trữ  $O(n)$ .



Ảnh minh họa 1. Sơ đồ cây tính giá trị trong chuỗi Fibonacci.

Bước 1: Tìm công thức truy hồi biểu diễn nghiệm tối ưu của bài toán lớn thông qua nghiệm tối ưu của các bài toán con.

$$F_n = \begin{cases} n & \text{với } n \leq 1 \\ F_{n-1} + F_{n-2} & \text{với } n > 1 \end{cases}$$

Bước 2: Tổ chức dữ liệu và chương trình

- Tạo bảng để lưu các giá trị  $F_n$  cần sử dụng lại của chuỗi Fibonacci
- Tính nghiệm của bài toán trong trường hợp riêng đơn giản nhất.

$$F_0 = 0; F_1 = 1$$

n	0	1	2	3	4	5	6	7	...
$F_n$	1	1							

Bước 3: Truy vết, tìm nghiệm của bài toán dựa vào bảng phương án

$$F_n = F_{n-1} + F_{n-2}$$

Code tham khảo

```
int fib(int n)
{
    int f[n + 2];
    int i;
    // gán giá trị cho 2 phần tử đầu tiên của dãy fibonacci
    f[0] = 0;
```

Bước 4: Tối ưu hóa không gian lưu trữ, chỉ sử dụng biến nguyên để lưu trữ 2 giá trị gần nhất cần tính. Độ lớn không gian lưu trữ  $O(1)$ .

Code tham khảo

```
int fib(int n)
{
    int a = 0, b = 1, c, i;
    if( n == 0)
        return a;
    for(i = 2; i <= n; i++)
    {
        c = a + b;
        a = b;
        b = c;
    }
    return b;
}
```

### 2.1.2. Bài 2. Tính tổ hợp chập k của n phần tử

Đề bài: Cho k và n. Tính tổ hợp chập k của n phần tử biết:

$$C_n^0 = C_n^1 = 1$$
$$C_n^k = C_{n-1}^{k-1} + C_{n-1}^k$$

*Ví dụ 1:*

Input : k=0; n = 10

Output : 1

*Ví dụ 2:*

Input : k=4; n = 10

Output : 210

Hướng dẫn giải thuật:

Bước 1: Tìm công thức truy hồi biểu diễn nghiệm tối ưu của bài toán lớn

thông qua nghiệm tối ưu của các bài toán con.

$$C_n^k = \begin{cases} 1 & \text{với } k = 0; k = n \\ C_{n-1}^{k-1} + C_{n-1}^k & \text{với } 1 \leq k \leq n-1 \end{cases}$$

Bước 2: Tổ chức dữ liệu và chương trình

- Tạo bảng để lưu các giá trị  $C_n^k$  cần sử dụng để tính nghiệm tối ưu của bài toán

- Tính nghiệm của bài toán trong trường hợp riêng đơn giản nhất.

$$C_n^0 = 1; C_n^1 = 1$$

```
for(i = 1; i <= n; i++)
{
    C[0, i] := 1;
    C[i, i] := 1;
}
```

Bước 3: Truy vết, tìm nghiệm của bài toán dựa vào bảng phương án

```
for(i= 2; i <= n; i++)
{
    for(j = 1; j <= i-1; j++)
    {
        C[i, j] := C[i-1, j-1] + C[i-1, j];
    }
}
```

n	k	0	1	2	3	4	5
1		1	1				
2		1	2	1			
3		1	3	3	1		
4		1	4	6	4	1	
5		1	5	10	10	5	1

Độ phức tạp về thời gian là  $O(nk)$ .

Độ phức tạp về mặt không gian là  $O(nk)$ .

Bước 4: Tối ưu hóa không gian lưu trữ: Có thể cải tiến dùng 2 mảng một chiều thay cho 1 mảng hai chiều.

### 2.1.3. Bài 3. Dãy con đơn điệu tăng dài nhất

Đề bài: 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.

Dữ liệu vào: Chứa trong tệp “DAYCON.INP”.

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

Dữ liệu ra: Chứa trong tệp “DAYCON.OUT”.

- Dòng 1 là độ dài của dãy con tăng dài nhất.
- Dòng 2 là các phần tử của dãy con tăng dài nhất, mỗi phần tử cách nhau 1 khoảng trắng. Ví dụ:

DAYCON.INP	DAYCON.OUT
6	4
1 2 5 4 6 2	1 2 4 6

Hướng dẫn giải thuật:

Bước 1: Tìm công thức truy hồi biểu diễn nghiệm tối ưu của bài toán lớn thông qua nghiệm tối ưu của các bài toán con.

- Hàm mục tiêu:  $f$  = độ dài dãy con.
- Vì độ dài dãy con chỉ phụ thuộc vào một yếu tố là dãy ban đầu nên bảng phương án là bảng một chiều. Gọi  $L_i$  là độ dài dãy con tăng dài nhất, các phần tử lấy trong miền từ  $A_1$  đến  $A_i$  và phần tử cuối cùng là  $A_i$ .
- Nhận xét với cách làm này ta đã chia 1 bài toán lớn (dãy con tối ưu của n số) thành các bài toán tối ưu con cùng kiểu có kích thước nhỏ hơn (dãy con của dãy i số).

- Ta có công thức QHĐ để tính  $L_i$  như sau:

$$L_i = \begin{cases} 1 & \text{với } i = 1 \\ 1 + \max(L_j) & \text{với } 0 < j < i \text{ và } A_j < A_i \end{cases}$$

Bước 2: Tổ chức dữ liệu và chương trình

- Tạo bảng để lưu các giá trị  $L_i$  cần sử dụng để tính nghiệm tối ưu của bài toán

- Tính nghiệm của bài toán trong trường hợp riêng đơn giản nhất.

Bước 3: Truy vết, tìm nghiệm của bài toán dựa vào bảng phương án

- Tính  $L_i$ : phần tử đang được xét là  $A_i$ . Ta tìm đến phần tử  $A_j < A_i$  có  $L_j$  lớn nhất. Khi đó nếu bổ sung  $A_i$  vào sau dãy con  $\dots A_j$  ta sẽ được dãy con tăng dần dài nhất xét từ  $A_1 \dots A_i$

- Truy vết:  $T[i]=j$  và xuất kết quả.

- Độ phức tạp thuật toán:  $O(n^2)$ .

Code tham khảo:

```

#include <iostream>
#include <fstream>
#include <memory.h>

using namespace std;
int n,a[100],l[100],t[100],i,j;int
max(int x[],int n)
{
    int tmp=x[0];
    for (i=1;i<n;i++)
        if (tmp<x[i]) tmp=x[i];
    return tmp;
}

int main()
{
    ifstream fi("DAYCON.INP");
    ofstream fo("DAYCON.OUT");
    fi>>n;
    for (i=0;i<n;i++) fi>>a[i];
    memset(t,-1,sizeof(t));
    for (i=0;i<n;i++)
    {
        l[i]=1;
        for (j=0;j<i;j++)
            if ((a[j]<=a[i]) && (l[i]<l[j]+1))
            {
                l[i]=l[j]+1;
                t[i]=j;
            }
    }
    int res=max(l,n);
    fo<<a[res]; res=max(t,n);

    while (res!=-1)
    {
        fo<<a[res]<<" "; res=t[res];
    }

    fi.close();
    fo.close(); return 0;
}

```

#### 2.1.4. Bài 4. Dãy con chung dài nhất

Cho hai dãy ký hiệu X và Y, dãy con chung dài nhất của X và Y là dãy các ký hiệu nhận được từ X bằng cách xóa đi một số các phần tử và cũng nhận được từ Y bằng cách xóa đi một số phần tử.

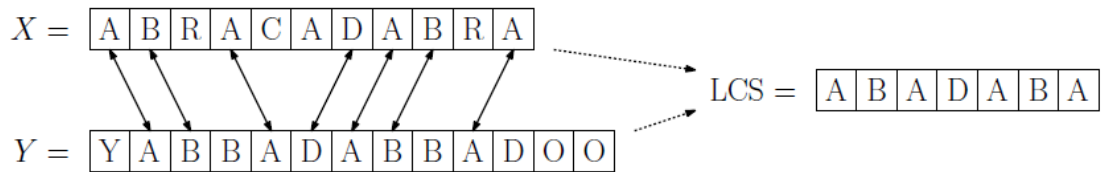
Ví dụ:

Input: X = ABCDCAE; Y = DACDBA

Output: Dãy con chung dài nhất: ACDA.

LCS.INP	LCS.OUT
ABRACADABRA	ABADABA
YABBADABBADOO	

Hướng dẫn giải thuật:



Ảnh minh họa 2. Tìm chuỗi con chung dài nhất

Bước 1: Tìm công thức truy hồi

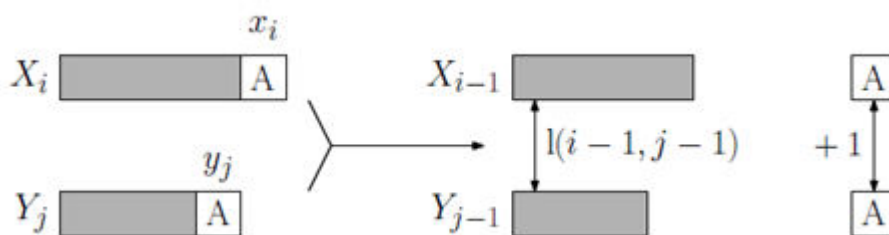
Gọi  $L(i,j)$  là độ dài xâu con chung dài nhất của xâu  $X(i)$  gồm  $i$  ký tự phần đầu của X ( $X(i) = X[1..i]$ ) và xâu  $Y(j)$  gồm  $j$  ký tự phần đầu của Y ( $Y(j) = Y[1..j]$ ).

Ví dụ trong trường hợp trên ta có  $X_5 = \{ABRAC\}$  and  $Y_6 = \{YABBAD\}$ . Chuỗi con chung dài nhất lúc này là  $\{ABA\}$ . Do đó,  $L(5,6) = 3$ .

Xét trạng thái  $L(i,j)$  có thể xảy ra 3 trường hợp:

- Nếu một trong 2 chuỗi so sánh là *chuỗi rỗng*, thì giá trị chiều dài chuỗi con chung là 0, hay  $L(0,j) = L(i,0) = 0$ .
- Nếu tại giá trị tại 2 vị trí của 2 chuỗi đang xét *bằng nhau* hay  $X[i] = Y[j]$ , thì tăng độ dài chuỗi con chung lên 1 đơn vị:

$$L(i,j) = L(i-1,j-1) + 1$$

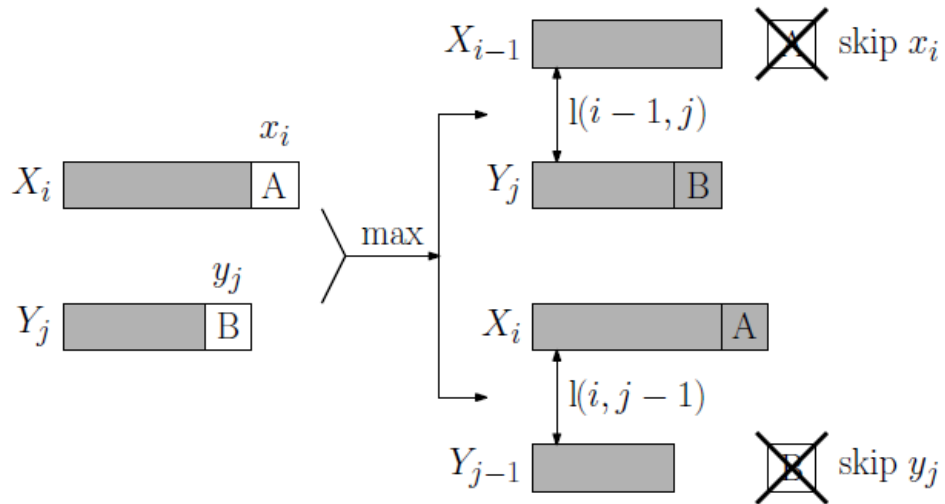


Ảnh minh họa 3. Khi  $X[i] = Y[j]$  thì tăng độ dài chuỗi con chung lên 1 đơn vị



- Nếu tại giá trị tại 2 vị trí của 2 chuỗi đang xét *khác nhau* hay  $X[i] \neq Y[j]$ , thì so sánh các chuỗi con và lấy độ dài chuỗi con lớn nhất:

$$L(i, j) = \max(L(i-1, j), L(i, j-1))$$



Ảnh minh họa 4. Khi  $X[i] \neq Y[j]$  thì so sánh lấy độ dài chuỗi con lớn nhất

Như vậy, ta có công thức quy hoạch động như sau:

$$L(i, j) = \begin{cases} 0 & \text{nếu } i = 0 \text{ hoặc } j = 0 \\ L(i-1, j-1) + 1 & \text{nếu } i, j > 0 \text{ và } X[i] = Y[j] \\ \max(L(i-1, j), L(i, j-1)) & \text{nếu } i, j > 0 \text{ và } X[i] \neq Y[j] \end{cases}$$

Bước 2: Tổ chức dữ liệu và chương trình

Tạo bảng phương án là một mảng 2 chiều  $L[0..m, 0..n]$  để lưu giá trị độ dài của các dãy con chung dài nhất của các cặp tiền tố.

Ta gọi  $L[i, j]$  là độ dài dãy con chung dài nhất của  $X_i$  và  $Y_j$ . Khi đó độ dài dãy con chung dài nhất của  $X$  và  $Y$  sẽ là  $c[n, m]$

- Trường hợp đơn giản nhất: độ dài dãy con chung dài nhất của của một dãy rỗng và một dãy bất kỳ luôn bằng 0 do đó  $c[i, 0] = 0$  và  $c[0, j] = 0$  với mọi  $i, j$ .

Bước 3: Cài đặt thuật toán tìm nghiệm

**for** ( $i=0; i < m; i++$ )  $L[i, 0] := 0;$

**for** ( $j=0; j < n; j++$ )  $L[0, j] := 0;$

**for** ( $i=0; i < m; i++$ )

{

**for** ( $j=0; j < n; j++$ )

if ( $X[i] = Y[j]$ )

{

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

```

    }
    else
    {
        L[i,j]:=max(L[i-1,j],L[i,j-1]);
    }
}

```

Như vậy chi phí không gian của bài toán là  $O(n^2)$ , chi phí thời gian là  $O(n^2)$ .

#### Bước 4: Làm tốt thuật toán

Gợi ý tham khảo: Có một phương pháp cài đặt tốt hơn, chỉ với chi phí không gian  $O(n)$  dựa trên nhận xét sau: để tính ô  $L[i,j]$  của bảng phương án, ta chỉ cần 3 ô  $L[i-1,j-1]$ ,  $L[i-1,j]$  và  $L[i,j-1]$ . Tức là để tính dòng  $L[i]$  thì chỉ cần dòng  $L[i-1]$ . Do đó ta chỉ cần 2 mảng 1 chiều để lưu dòng vừa tính và dòng đang tính ( $L$ ) mà thôi.

Code tham khảo:

```

int lcs(char* X, char* Y, int m, int n,
        vector<vector<int> >& dp)
{
    if (m == 0 || n == 0)
        return 0;
    if (X[m - 1] == Y[n - 1])
        return dp[m][n] = 1 + lcs(X, Y, m - 1, n - 1, dp);

    if (dp[m][n] != -1) {
        return dp[m][n];
    }
    return dp[m][n] = max(lcs(X, Y, m, n - 1, dp),
                          lcs(X, Y, m - 1, n, dp));
}

```

### 2.1.5. Bài toán xếp ba lô

**Đề bài:** Có một ba lô có thể chứa tối đa trọng lượng  $M$  và có  $n$  đồ vật, mỗi đồ vật có trọng lượng  $w_i$  và giá trị  $b_i$ .  $M$ ,  $w_i$ ,  $b_i$  là các số nguyên. Hãy chọn và xếp các đồ vật vào ba lô để tổng giá trị của ba lô là lớn nhất.

Hướng dẫn giải thuật:

Bước 1: Tìm công thức truy hồi

- Sử dụng mảng  $v[0..n, 0..M]$  để lưu trữ lại các giải pháp của các bài toán con.
- Gọi  $v[i, j]$  là tổng giá trị lớn nhất của ba lô mà trọng lượng không vượt

quá  $j$  khi chỉ sử dụng các đồ vật  $\{1, 2, \dots, i\}$ . Khi đó giá trị lớn nhất khi được chọn trong số  $n$  gói với giới hạn trọng lượng  $M$  chính là  $v[n, M]$ .

- Với giới hạn trọng lượng  $j$ , việc chọn tối ưu trong số các gói  $\{1, 2, \dots, i-1, i\}$  để có giá trị lớn nhất sẽ có hai khả năng:

- Nếu không chọn gói thứ  $i$  thì  $v[i, j]$  là giá trị lớn nhất có thể bằng cách chọn trong số các gói  $\{1, 2, \dots, i-1\}$  với giới hạn trọng lượng là  $j$  tức là  $v[i, j] = v[i-1, j]$
- Nếu có chọn gói thứ  $i$  (tất nhiên chỉ xét tới trường hợp này khi mà  $w_i \leq j$ ) thì  $v[i, j] = v_i + v[i-1, j-w_i]$

Vì theo cách xây dựng  $v[i, j]$  là giá trị lớn nhất có thể nên  $v[i, j]$  sẽ là max trong hai giá trị thu được ở trên tức là :

$$v[i, j] = \max\{v[i-1, j], v_i + v[i-1, j-w_i]\}$$

Bước 2: Tổ chức dữ liệu và chương trình

- Khởi tạo các giá trị ban đầu:

$v[0, j] = 0$  với mọi  $j$ .

$v[i, j] = 0$  với mọi  $i$ .

- Tính  $v[i, j]$  theo  $v[i-1, j]$  hoặc  $v[i-1, j-w_i]$ .

Bước 3: Cài đặt thuật toán tìm nghiệm

ước

4:

Cải

tiến

thu

ật

toá

n

(

ode

tha

m

khả

o:

```

for (i=0; i<n; i++) v[i,0]=0;
B for (j=0; j<m; j++) v[0,j]=0;
  for (i=1; i<n; i++)
  {
    for (j=1; j<m; j++)
    {
      if (w_i <= j) // có thể sử dụng đồ vật i
      {
        int i, w;
        if (b_i + v[i-1, j-w_i] > v[i-1, j])
          vector<vector<int>>> K(n+1, vector<int>(W+1));
          v[i,j] = b_i + v[i-1, j-w_i]; // sử dụng đồ vật i
      }
      // Xây dựng bảng K[][]
      for (i = 0; i <= n; i++)
      {
        for (w = 0; w <= W; w++)
        {
          v[i, j] = v[i-1, j]; // không sử dụng đồ vật i
          if (i <= 0) // trường hợp 0) // Khởi tạo basic
            K[i][w] = 0;
          else if (wt[i] >= j) // không sử dụng đồ vật i
            K[i][w] = max(val[i-1] +
                          K[i-1][w-wt[i-1]],
                          K[i-1][w]);
          else
            K[i][w] = K[i-1][w];
        }
      }
    }
  }
  return K[n][W];
}

```

## 2.2. Một số bài toán điển hình

### 2.2.1. Phân tích số

**Đề bài:** Cho số tự nhiên  $n \leq 100$ . Hãy cho biết có bao nhiêu các phân tích số  $n$  thành tổng của dãy các số nguyên dương, các cách phân tích là hoán vị của nhau chỉ tính là một cách.

Ví dụ, với  $n = 5$ , ta có 7 cách phân tích:

$$5 = 1 + 1 + 1 + 1 + 1$$

$$5 = 1 + 1 + 1 + 2$$

$$5 = 1 + 1 + 1 + 3$$

$$5 = 1 + 2 + 2$$

$$5 = 1 + 4$$

$$5 = 2 + 3$$

$$5 = 5$$

- Dữ liệu vào: Chứa trong tệp “PHANTICH.INP”.

Gồm 1 dòng duy nhất là số  $n$ .

- Dữ liệu ra: Chứa trong tệp “PHANTICH.OUT”.

Gồm 1 dòng duy nhất là số các phân tích số  $n$  thành tổng của dãy các số nguyên dương. Ví dụ:

PHANTICH.INP   PHANTICH.OUT

5                      7

#### **Hướng dẫn thuật toán:**

- Gọi  $F[m,v]$  là số cách phân tích số  $v$  thành tổng của các số nguyên dương  $\leq m$ . Như vậy có chứa ít nhất 1 số  $m$  trong phép phân tích. Khi đó nếu trong cách phân tích ta bỏ đi số  $m$  thì ta sẽ được các cách phân tích số  $v - m$  thành tổng các số nguyên dương  $\leq m$ .

- Ta có công thức quy hoạch động sau:

$$F[0,k]=1 ; \forall k: 1 \leq k \leq n.$$

$$\forall i: 1 \leq i \leq n:$$

$$F[i,k] = F[i,k-1] + F[i-k,k] ; \forall k: 1 \leq k \leq i.$$

$$F[i,k] = F[i,k-1] ; \forall k: i+1 \leq k \leq n.$$

- Kết quả là:  $F[n,n]$ .
- Độ phức tạp thuật toán:  $O(n^2)$ .

Code tham khảo:

```
#include <iostream>#include <fstream>
using namespace std; int n,f[100][100],i,j,k;
int main()
{
    ifstream fi("PHANTICH.INP"); ofstream
    fo("PHANTICH.OUT");fi>>n;
    for (k=1;k<=n;k++) f[0][k]=1;
    for (i=1;i<=n;i++)
    {
        for (k=1;k<=i;k++) f[i][k]=f[i][k-1]+f[i-k][k];
        for (k=i+1;k<=n;k++) f[i][k]=f[i][k-1];
    }
    fo<<f[n][n];
    fi.close();
    fo.close();
}
```

### 2.2.2. Tìm hình vuông lớn nhất

**Đề bài:** Cho một bảng  $m \times n$  ( $0 < m, n \leq 1000$ ) các ô nhỏ chứa các số 0 và 1.

Hãy tìm hình vuông chứa toàn số 1 và có diện tích lớn nhất.

- Dữ liệu vào: chứa trong tệp “HINHVUONG.INP”.  
Dòng đầu tiên chứa 2 số  $m$  và  $n$ .  
 $m$  dòng tiếp theo chứa các số 0 và 1 là biểu diễn của bảng.

- Dữ liệu ra: chứa trong tệp “HINHVUONG.OUT”.

Một dòng chứa diện tích lớn nhất của hình vuông tìm được.

Ví dụ:

HINHVUONG.INP	HINHVUONG.OUT
4 5 0 1 1 0 1 0 1 1 1 1 0 1 1 1 1 0 1 1 1 1	9

***Hướng dẫn thuật toán:***

- Gọi  $B[m,n]$  là bảng chứa cạnh của hình vuông lớn nhất tìm được. Từ đó, cạnh hình vuông lớn nhất là  $\text{Max}(B[m,n])$ . Ta có công thức quy hoạch động sau:

$$B[0, i] = A[0, i] \quad , \forall i \leq m$$

$$B[i, 0] = A[i, 0] \quad , \forall i \leq n$$

$$B[i, j] = \begin{cases} \text{Min}(B[i - 1, j - 1], B[i, j - 1], B[i - 1, i]) + 1, & \forall i \leq n, 2 \leq j \leq n, A[i, j] = 1 \\ 0, & \forall A[i, j] = 0 \end{cases}$$

- Kết quả là bình phương của  $\text{Max}(B[m,n])$ .
- Độ phức tạp thuật toán:  $O(n*m)$ .

```

#include <iostream>
#include <fstream>

using namespace std;

int a[1000][1000],b[1000][1000];
int m,n,i,j,tmp;

int min(int a,int b)
{
    return (a<b) ? a:b;
}

int main()
{
    ifstream fi("HINHVUONG.INP");
    ofstream
    fo("HINHVUONG.OUT"
    );fi>>m>>n;
    for (i=0;i<m;i++)
        for (j=0;j<n;j++)
            fi>>a[i][j];for
            (i=0;i<m;i++)
                b[0][i]=a[0][i];
            for (i=0;i<n;i++)
                b[i][0]=a[i][0];for
                (i=1;i<n;i++)
                    for (j=1;j<m;j++)
                        b[i][j]=min(min(b[i-1][j-1],b[i][j-1]),b[i-1][i])+1;
            tmp=b[0][0];
            for (i=0;i<m;i++)
                for (j=0;j<n;j++)
                    if (tmp < b[i][j])
                        tmp=b[i][j];fo<<tmp*tmp;
    fi.close();
    fo.close();
    return 0;
}

```

***Code tham khảo:***





### 2.2.3. Đi tìm kho báu

Cho một bảng gồm  $n$  dòng,  $m$  cột, trong mỗi ô chứa số lượng các hạt kim cương khác nhau. Người chơi xuất phát từ ô bên trái phía dưới của bảng và cần phải di chuyển đến ô phía bên phải trên cùng của bảng. Mỗi bước người chơi chỉ có thể di chuyển sang ô kề bên phải hoặc phía trên, người chơi dừng lại ở ô nào thì sẽ thu được số kim cương ở ô đó. Hãy tìm con đường người chơi cần di chuyển để đạt được tổng số viên kim cương là lớn nhất.

- Dữ liệu vào: Chứa trong tệp “TIMKHOB AU.INP”.
  - Dòng đầu là 2 số  $n$  và  $m$  ( $0 < m, n \leq 1000$ ).
  - $n$  dòng tiếp theo là biểu diễn của bảng.
- Dữ liệu ra: Chứa trong tệp “TIMKHOB AU.OUT”.
  - Dòng đầu là tổng lớn nhất tìm được.
  - Dòng thứ 2 là các bước di chuyển của con rùa.

**Ví dụ:**

TIMKHOB AU. INP	TIMKHOB AU.OUT
4 4 9 8 6 2 10 11 13 11 3 7 12 8 5 9 13 9	65 (4,1) => (4,2) => (4,3) => (3,3) => (2,3) => (2,4) => (1,4)

#### **Hướng dẫn thuật toán:**

- Gọi bảng ban đầu là  $A$ , bảng kết quả là  $B$ . Ta có công thức quy hoạch động sau:
  - $B[n,1] = A[n,1]$ .
  - $B[i,1] = B[i+1,1] + A[i,1]$ ,  $\forall i: n-1 \geq i \geq 1$ .
  - $B[n,j] = B[n,j-1] + A[n,j]$ ,  $\forall j: 2 \leq j \leq m$ .
  - $B[i,j] = \max(B[i+1,j], B[i,j-1]) + A[i,j]$ ,  $\forall i,j: n-1 \geq i \geq 1; 2 \leq j \leq m$ .
- Tổng lớn nhất là:  $B[1,m]$ .
- Tiến hành truy vết để in ra đường đi.
- Độ phức tạp thuật toán:  $O(n*m)$ .

#### 2.2.4. Palindrom

**Đề bài:** Một chuỗi gọi là chuỗi đối xứng (palindrom) nếu chuỗi đó đọc từ trái sang phải hay từ phải sang trái đều như nhau. Cho một chuỗi S, hãy tìm số kí tự ít nhất cần thêm vào S để S trở thành chuỗi đối xứng.

**Hướng dẫn thuật toán:**

Gọi  $L(i,j)$  là số kí tự ít nhất cần thêm vào chuỗi con  $S[i..j]$  của S để chuỗi đó trở thành đối xứng.

Đáp số của bài toán sẽ là  $L(1,n)$  với  $n$  là số kí tự của S. Ta có công thức sau để tính  $L(i,j)$ :

- $L(i,i)=0$ .
- $L(i,j)=L(i+1,j-1)$  nếu  $S[i]=S[j]$
- $L(i,j)=\max(L(i+1,j), L(i,j-1))$  nếu  $S[i] \neq S[j]$

Gợi ý cài đặt: Gọi P là chuỗi đảo của S và T là chuỗi con chung dài nhất của S và P. Khi đó các kí tự của S không thuộc T cũng là các kí tự cần thêm vào để S trở thành đối xứng. Đáp số của bài toán sẽ là  $n-k$ , với  $k$  là độ dài của T.

Ví dụ:  $S=edbabcd$ , chuỗi đảo của S là  $P=dcbabde$ . Chuỗi con chung dài nhất của S và P là  $T=dbabd$ . Như vậy cần thêm 2 kí tự là e và c vào để S trở thành chuỗi đối xứng.

#### 2.2.5. Chọn ô

**Đề bài:** Cho một bảng hình chữ nhật kích thước  $4 \times n$  ô vuông. Các dòng được đánh số từ 1 đến 4, từ trên xuống dưới, các cột được đánh số từ 1 đến  $n$  từ trái qua phải.

Ô nằm trên giao của dòng  $i$  và cột  $j$  được gọi là ô  $(i,j)$ . Trên mỗi ô  $(i,j)$  có ghi một số nguyên  $a_{ij}$ ,  $i=1, 2, 3, 4$ ;  $j=1, 2, \dots, n$ . Một cách chọn ô là việc xác định một tập con khác rỗng S của tập tất cả các ô của bảng sao cho không có hai ô nào trong S có chung cạnh. Các ô trong tập S được gọi là ô được chọn, tổng các số trong các ô được chọn được gọi là trọng lượng của cách chọn. Tìm cách chọn sao cho trọng lượng là lớn nhất.

Ví dụ: Xét bảng với  $n=3$  trong hình vẽ dưới đây:

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

Cách chọn cần tìm là tập các ô  $S = \{(3,1), (1,2), (4,2), (3,3)\}$  với trọng lượng 32.

Input

- Dòng đầu tiên chứa số nguyên dương  $n$  là số cột của bảng.
- Cột thứ  $j$  trong số  $n$  cột tiếp theo chứa 4 số nguyên  $a_{1j}, a_{2j}, a_{3j}, a_{4j}$ , hai số liên tiếp cách nhau ít nhất một dấu cách, là 4 số trên cột  $j$  của bảng.

Output

- Gồm 1 dòng duy nhất là trọng lượng của cách chọn tìm được.

Example

Input

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

Output

```
32
```

Hạn chế: Trong tất cả các test:  $n \leq 10000$ ,  $|a_{ij}| \leq 30000$ . Có 50% số lượng test với  $n \leq 1000$ .

***Hướng dẫn thuật giải:***

- Theo đề bài thì bảng có 4 dòng và n cột;
- Gọi S là trạng thái chọn các ô ở cột thứ j, ta có thể biểu diễn S bằng 4 bit (các bit được đánh số từ phải sang bắt đầu bằng 0) với ý nghĩa:
  - +  $S[i-1] = 0$ : dòng thứ i của cột j không được chọn;
  - +  $S[i-1] = 1$ : dòng thứ i của cột j được chọn.
- Với 4 bit, S có thể biểu diễn 16 trạng thái từ {0000} đến {1111} (từ 0 đến 15), tuy nhiên ta nhận thấy chỉ có 8 trạng thái sau là thỏa yêu cầu của bài toán: {0000}, {0001}, {0010}, {0100}, {1000}, {1001}, {0101}, {1010} (tương ứng với các giá trị 0, 1, 2, 4, 5, 8, 9, 10).
- Gọi  $T[S, j]$  là trọng lượng lớn nhất khi chọn các ô đến cột thứ j với trạng thái chọn là S, ta có công thức quy hoạch động như sau:
 
$$T[S, j] = \max(T[P, j-1] + \text{value}(S))$$
 với P là trạng thái của cột liền trước của S sao cho P và S không có 2 bit 1 đồng thời ở cùng vị trí, còn  $\text{value}(S)$  là giá trị cách chọn cột j với trạng thái S.
- Khi cài đặt, với bài toán này, ta chỉ cần xây dựng hàm `getBit` để giải mã trạng thái S.

## 2.3. Giới thiệu một số bài toán mở rộng

### 2.3.1. Trò chơi làm nổ bóng:

**Đề bài:** Cho N quả bóng được xếp theo hàng, mỗi quả bóng được đánh số đồng xu kèm theo. Khi làm nổ một quả bóng thứ i, số đồng xu giành được sẽ là tích 3 số xu của quả bóng thứ i và 2 quả bóng liền kề trước và sau, hay tương đương với  $A[i-1] * A[i] * A[i+1]$ . Hãy tìm số đồng xu tối đa có thể nhận được sau khi làm nổ tất cả các quả bóng. Giả sử thêm 1 ở ranh giới 2 đầu dãy bóng.

Input : 1 dòng gồm giá trị xu của từng quả bóng

Output : Số đồng xu tối đa khi làm nổ tất cả các quả bóng

Ràng buộc:

$$1 \leq n \leq 300$$

$$0 < A[i] \leq 100$$

*Ví dụ 1:*

Input : [5, 10]

Output : 60

Giải thích: Thêm 1 vào đầu và cuối dãy xu, dãy số sẽ là: [1, 5, 10, 1]

Có 2 lần nổ cho 2 quả bóng:

- Lần 1 cho nổ quả bóng trị số 5 xu, số đồng xu nhận =  $1*5*10=50$  xu

- Lần 2 cho nổ quả bóng trị số 10 xu, số đồng xu nhận =  $1*10*1=10$  xu

Tổng cộng:  $50+10=60$  xu

*Ví dụ 2:*

Input: [3,1,5,8]

Output: 167

Giải thích:

Số bóng = [3,1,5,8] --> [3,5,8] --> [3,8] --> [8] --> []

Số xu =  $3*1*5 + 3*5*8 + 1*3*8 + 1*8*1 = 167$

*Ví dụ 3:*

Input : [1, 2, 3, 4, 5]

Output : 110

***Hướng dẫn thuật giải:***

Đầu tiên, xét một mảng con từ các chỉ số [Left, Right]. Giả sử quả bóng tại chỉ số Last sẽ là quả bóng cuối cùng chọn nổ trong mảng con này, thì số đồng xu nhận được sẽ là  $x=A[left-1]*A[last]*A[right+1]$

Gọi  $dp[i][j]$  là giá trị số xu lớn nhất có được để làm nổ tất cả bóng trong đoạn con [i,j]

Như vậy, tổng số xu nhận được sẽ là tổng của x và số xu nhận được khi làm nổ 2 đoạn còn lại trong dãy bóng, chính bằng  $= x+ dp[left][last - 1] + dp[last + 1][right]$

Do đó, đối với mỗi đoạn [Left,Right], chúng ta cần tìm và chọn một giá trị là Last với số tiền tối đa đạt được, và cập nhật mảng dp. Câu trả lời là giá trị tại dp[1] [N].

***Code tham khảo:***

```

#include <bits/stdc++.h>
#include <iostream>
using namespace std;

int getMax(int A[], int N)
{
    // Thêm 2 giá trị biên cho dãy bóng
    int B[N + 2];
    B[0] = 1;
    B[N + 1] = 1;
    for (int i = 1; i <= N; i++)
        B[i] = A[i - 1];
    // Tạo mảng dp
    int dp[N + 2][N + 2];
    memset(dp, 0, sizeof(dp));

    for (int length = 1; length < N + 1; length++)
    {
        for (int left = 1; left < N - length + 2; left++)
        {
            int right = left + length - 1;
            // Lặp tìm với mỗi mảng con [left, right]
            for (int last = left; last < right + 1; last++)
            {
                dp[left][right] = max(dp[left][right], dp[left][last - 1] +
                                         B[left - 1] * B[last] * B[right + 1] +
                                         dp[last + 1][right]);
            }
        }
    }
    return dp[1][N];
}

// Driver code
int main()
{
    int A[] = { 1, 2, 3, 4, 5 };

    // Lấy kích thước mảng
    int N = sizeof(A) / sizeof(A[0]);

    // gọi hàm
    cout << getMax(A, N) << endl;
}

```

### 2.3.2. Bài toán Quy hoạch nông trại

**Đề bài:** Người chủ nông trại có  $N$  mảnh đất và  $M$  dải đất. Các mảnh đất có thể coi là một tứ giác và các dải đất thì coi như một đường thẳng. Dọc theo các dải đất ông ta trồng các cây táo, dải đất thứ  $i$  có  $A_i$  cây táo. Ông ta cũng trồng các cây táo trên viền của các mảnh đất, mảnh đất thứ  $j$  có  $B_j$  cây táo. Cả ở trên các mảnh đất và dải đất, xen giữa 2 cây táo ông ta trồng một cây chà là làm quà cho con trai. Ông ta cho con trai được chọn các mảnh đất và dải đất tùy ý với điều kiện tổng số cây táo không vượt quá  $Q$ . Người con trai phải chọn như thế nào để có được nhiều cây chà là nhất.

Input:

- Dòng đầu tiên bao gồm 3 số nguyên: Đầu tiên là số  $Q$  ( $0 \leq Q \leq 150000$ ) là số cây táo mà người con được chọn; sau đó là số nguyên  $M$  là số cánh đồng; tiếp theo là số nguyên  $K$  là số dải đất.

- Dòng thứ hai chứa  $M$  số nguyên  $N_1, N_2, \dots, N_m$  ( $3 \leq N_1, N_2, \dots, N_m \leq 150$ ) là số cây bách trên cánh đồng

- Dòng thứ 3 chứa  $K$  số nguyên  $R_1, R_2, \dots, R_k$  ( $2 \leq R_1, R_2, \dots, R_k \leq 150$ ) là số cây bách trên dải đất.

Output: Gồm 1 số nguyên duy nhất thể hiện số lượng cây Chà là lớn nhất mà người con có thể thừa hưởng.

Ví dụ:

Input	Output
17 3 3 13 4 8 4 8 6	17

#### **Hướng dẫn thuật toán**

Theo đề ta thấy mảnh đất thứ  $i$  có  $A_i$  cây chà là và dải đất thứ  $j$  có  $B_j - 1$  cây chà là.



Coi các mảnh đất và dải đất là các “đồ vật”, đồ vật thứ  $k$  có khối lượng  $W_k$  và giá trị  $V_k$  (nếu  $k$  là mảnh đất  $i$  thì  $W_k=V_k=A_i$ , nếu  $k$  là dải đất  $j$  thì  $W_k=B_j$ ,  $V_k=B_j-1$ ). Ta cần chọn các “đồ vật”, sao cho tổng “khối lượng” của chúng không vượt  $Q$  và tổng “giá trị” là lớn nhất. Đến đây chúng ta quy bài toán về dạng bài toán xếp balô đã được trình bày phần các bài toán cơ bản.

### 2.3.3. Giới thiệu các bài toán quy hoạch động mở rộng

Ngoài ra, có thể tham khảo thêm một số bài tập mới trên một số website lập trình như:

- GeeksforGeeks: <https://www.geeksforgeeks.org/dynamic-programming/>
- LeetCode: <https://leetcode.com/tag/dynamic-programming/>
- <https://github.com/skm2000/Dynamic-Programming>
- VN Online Judge, VnAtcoder Educational DP Contest, [https://oj.vnoi.info/contest/atcoder\\_dp?fbclid=IwAR1m0XnDF4nGROftlpFgHsLTBnCeUYR3Ilu8FCGq\\_OClckc4q2f3zT2mmBw](https://oj.vnoi.info/contest/atcoder_dp?fbclid=IwAR1m0XnDF4nGROftlpFgHsLTBnCeUYR3Ilu8FCGq_OClckc4q2f3zT2mmBw)
- <https://www.codechef.com/tags/problems/dynamic-programming>



## TÀI LIỆU THAM KHẢO

1. Đào Thị Thảo Sương, Phương pháp Quy hoạch động, Đà Nẵng, 2012
2. Lê Minh Hoàng, Giải thuật và lập trình, Đại học Sư phạm Hà Nội, 2006.
3. Trần Quang Quá, Phương pháp quy hoạch động, Trường Chuyên Tế Vinh, Biên Hòa, 2002.
4. Antti Laaksonen, Competitive Programmer's Handbook, Helsinki, 2018.
5. Steven Halim, Felix Halim, Competitive Programming 3, Handbook for ACM ICPC and IOI contestants, 2013.
6. <https://www.geeksforgeeks.org/dynamic-programming/>
7. <https://leetcode.com/tag/dynamic-programming/>
8. <https://github.com/skm2000/Dynamic-Programming>
9. VN Online Judge, VnAtcoder Educational DP Contest, [https://oj.vnoi.info/contest/atcoder\\_dp?fbclid=IwAR1m0XnDF4nGROftlpFgHsLTBnCeUYR3IIu8FCGq\\_OClckc4q2f3zT2mmBw](https://oj.vnoi.info/contest/atcoder_dp?fbclid=IwAR1m0XnDF4nGROftlpFgHsLTBnCeUYR3IIu8FCGq_OClckc4q2f3zT2mmBw)
10. TopCoder, Nhập môn Quy hoạch động, [https://vnoi.info/wiki/translate/topcoder/dynamic-programming.md?fbclid=IwAR3v9yDgUJ4-AS6yLfBwXwRerfYEtNIJpJb3dSxBcmVUeCAIQer3tj5I\\_GU](https://vnoi.info/wiki/translate/topcoder/dynamic-programming.md?fbclid=IwAR3v9yDgUJ4-AS6yLfBwXwRerfYEtNIJpJb3dSxBcmVUeCAIQer3tj5I_GU)

### Bài toán 1: Xây cầu

" Trên hai đường thẳng song song L1 và L2, Người ta đánh dấu trên mỗi đường N điểm, Các điểm trên đường thẳng L1 Được đánh số từ 1 đến N, từ trái qua phải, còn các điểm trên đường thẳng L2 được đánh số bởi  $P[1], P[2], \dots, P[N]$  cũng từ trái qua phải, trong đó  $P[1], P[2], \dots, P[N]$  là một hoán vị của các số  $1, 2, \dots, N$ . Ta gọi các số gán cho các điểm là số hiệu của chúng. Cho phép nối hai điểm trên 2 đường thẳng có cùng số hiệu.

**Yêu Cầu :** Tìm cách nối được nhiều cặp điểm nhất với điều kiện các đoạn nối không được cắt nhau.

**Dữ Liệu :** Vào từ File BaiToan1.Inp:

Dòng đầu tiên chứa số nguyên dương  $N (N \leq 1000)$

Dòng thứ hai chứa các số  $P[1], P[2], \dots, P[N]$

**Kết Quả Ghi Ra File :** BaiToan1.Out

Dòng Đầu tiên chứa K là số lượng đoạn nối tìm được

Dòng tiếp theo chứa K số hiệu của các đầu mút của các đoạn nối được ghi theo thứ tự tăng dần.

**Ví Dụ :**

Baitoan1.INP

9

2 5 3 8 7 4 6 9 1

baitoan1.OUT

5

2 3 4 6 9

### Bài Toán 2: Dạo Chơi Bằng Xe Buýt

Trên một tuyến đường ở thành phố du lịch nổi tiếng X có ô tô Buýt công cộng phục vụ việc đi lại của du khách. Bến xe buýt có ở từng Km của tuyến đường. Mỗi lần đi qua bến xe đều đỗ cho du khách lên xuống. Mỗi bến đều có xe xuất phát từ nó, nhưng mỗi xe chỉ chạy không quá B km kể từ bến xuất phát của nó. Hành khách khi đi xe sẽ phải trả tiền cho độ dài đoạn đường mà họ ngồi trên xe. Cước phí cần trả để đi đoạn đường độ dài i là  $C_i (=1, 2, \dots, B)$ . Một du khách xuất phát từ một bến nào đó muốn đi dạo L km trên tuyến đường nói trên. Hỏi ông ta phải lên xuống xe như thế nào để tổng số tiền phải trả cho chuyến dạo chơi bằng xe buýt là nhỏ nhất.

**Dữ Liệu :** Vào Từ File : Bus.Inp

Dòng đầu tiên chứa 2 số nguyên dương B, L ( $B \leq 100, L \leq 10000$ )

Dòng thứ hai chứa B số Nguyên dương  $C_1, C_2, \dots, C_n$  , được ghi cách nhau bởi dấu trắng.

**Kết Quả :** Ghi ra File Văn bản : Bus.Out

Dòng đầu tiên ghi chi phí tìm được, và số lần xuống xe K .

Dòng tiếp theo ghi K số là độ dài của các đoạn đường của K lần ngồi xe.

**Ví Dụ:**

BUS.INP

10 15

12 21 31 40 49 58 69 79 90 101

BUS.OUT

147 3

3 6 6

### Bài Toán 3: Dãy Con Tăng Cực Đại

" Cho một dãy số nguyên dương  $A_1, A_2, \dots, A_n$ . Hãy tìm bớt một số ít nhất các phần tử của dãy số nguyên đó và giữ nguyên thứ tự các phần tử còn lại sao cho dãy số còn lại là một dãy tăng dần. Ta gọi dãy số nguyên tăng dần còn lại sau khi đã tìm bớt một số phần tử là dãy con của dãy đã cho.

Dữ Liệu : Vào từ File BaiToan3.Inp :

Dòng đầu tiên ghi số  $N$  là số phần tử ( $N \leq 10000$ )

Dòng tiếp theo ghi  $N$  số là các số nguyên của dãy

Kết Quả : Ghi Ra File : BaiToan3.Out

Dòng đầu tiên ghi số phần tử của dãy con lớn nhất đó

Dòng thứ hai ghi các số của dãy cần tìm .

### Bài toán 6: Vòng Quanh Thế Giới

(Đề Thi Học Sinh Giỏi Quốc Gia 2000-2001 - Bảng A )

Trên tuyến đường của xe chở khách du lịch vòng quanh thế giới xuất phát từ bến  $X$  có  $N$  khách sạn đánh số từ 1 đến  $N$  theo thứ tự xuất hiện trên tuyến đường, trong đó khách sạn  $N$  là địa điểm cuối cùng của tuyến đường mà tại đó xe bắt buộc phải dừng.

Khách sạn  $I$  cách địa điểm xuất phát  $A_i$  Km ( $i=1,2,\dots,N$ );  $A_1 < A_2 < \dots < A_N$  . Để đảm bảo sức khỏe cho khách hàng, theo tính toán của các nhà chuyên môn, sau khi đã chạy được  $P$  (Km) xe nên dừng lại cho khách nghỉ ở khách sạn. Vì thế, nếu xe dừng lại cho khách nghỉ ở khách sạn sau khi đã đi được  $Q$ (Km) thì lái xe phải trả một lượng phạt là :  $(Q-P)^2$ .

Ví Dụ :

Với  $N=4$  ,  $P=300$ ,  $A_1=250$ ,  $A_2=310$ ,  $A_3=550$  ,  $A_4=590$  . Xe bắt buộc phải dừng lại ở khách sạn 4 là địa điểm cuối cùng của hành trình. Nếu trên đường đi lái xe chỉ dừng lại tại khách sạn thứ 2 thì lượng phạt phải trả là :

$$(310-300)^2 + ((590-310)-300)^2 = 500$$

Yêu Cầu : Hãy xác định xem trên tuyến đường đến khách sạn  $N$ , xe cần dừng lại nghỉ ở những khách sạn nào để tổng lượng phạt mà lái xe phải trả là nhỏ nhất.

Dữ Liệu : Vào từ File văn bản có tên Bai4.Inp :

Dòng đầu tiên chứa số nguyên dương  $N$  ( $N \leq 10000$ );

Dòng thứ hai chứa số nguyên dương  $P$  ( $P \leq 500$ );

Dòng thứ ba chứa các số nguyên dương  $A_1, A_2, A_3, \dots, A_n$  (hai số liên tiếp cách nhau ít nhất bởi 1 dấu cách) ( $A_i \leq 2000000$  ,  $i=1,2,\dots,N$ ).

Kết Quả : Ghi ra File Văn Bản Bai4.Out:

Dòng đầu tiên ghi Z là lượng phạt mà lái xe phải trả ;  
Dòng thứ hai ghi K là tổng số khách sạn mà lái xe cần dừng lại cho khách nghỉ;

Dòng thứ ba chỉ chứa chỉ số của K khách sạn mà xe dừng lại cho khách nghỉ  
(Trong đó nhất thiết phải có khách sạn thứ N).

Ví Dụ:

BAI4.INP	BAI4.OUT
4	50
300	2
250 310 550 590	2 4

Cho đồ thị vô hướng  $G$  gồm  $n$  đỉnh được đánh số từ 1 tới  $n$  và  $m$  cạnh được đánh số từ 1 tới  $m$ . Cạnh thứ  $i$  nối giữa hai đỉnh  $u_i$  và  $v_i$ .

**Yêu cầu:** Hãy tìm tất cả các cạnh cầu của đồ thị đã cho.

**Dữ liệu:** Vào từ file văn bản **CANHCAU.INP** gồm:

- Dòng đầu tiên ghi 2 số nguyên dương  $n, m$  ( $n, m \leq 1000$ );
- $m$  dòng tiếp theo, dòng thứ  $i$  ghi 2 số  $u_i$  và  $v_i$

Các số trên cùng dòng viết cách nhau ít nhất một dấu cách.

**Kết quả:** Ghi ra file văn bản **CANHCAU.OUT** một số duy nhất là số lượng cạnh cầu của đồ thị.

CANHCAU.INP	CANHCAU.OUT
6 6 1 2 1 3 2 3 3 4 4 5 4 6 5 6	1

## Bài 6. (7 điểm) Nâng cấp mạng

Một hệ thống gồm  $n$  máy tính đánh số từ 1 tới  $n$  được kết nối thành một mạng bởi  $m$  đoạn cáp mạng đánh số từ 1 tới  $m$ . Đoạn cáp mạng thứ  $i$  có thông lượng  $w_i$  kết nối hai máy  $u_i, v_i$  cho phép truyền dữ liệu theo cả hai chiều giữa hai máy này.

Một dãy các máy  $x_1, x_2, \dots, x_p$ , trong đó giữa hai máy  $x_j$  và  $x_{j+1}$  ( $j = 1, 2, \dots, p - 1$ ) có đoạn cáp nối, được gọi là một đường truyền tin từ máy  $x_1$  tới máy  $x_p$ . Thông lượng của đường truyền tin được xác định như là thông lượng nhỏ nhất trong số các thông lượng của các đoạn cáp mạng trên đường

truyền. Giả thiết là mạng được kết nối sao cho có đường truyền tin giữa hai máy bất kỳ và giữa hai máy có không quá một đoạn cáp mạng nối chúng.

Người ta muốn nâng cấp mạng bằng cách tăng thông lượng của một số đoạn cáp nối trong mạng. Để tăng thông lượng của mỗi đoạn cáp mạng thêm một lượng  $\Delta$  ( $\Delta > 0$ ) ta phải trả một chi phí đúng bằng  $\Delta$ . Việc nâng cấp mạng phải đảm bảo là sau khi hoàn tất, thông lượng của mỗi đoạn cáp  $i$  đều bằng thông lượng của đường truyền tin có thông lượng lớn nhất từ máy  $u_i$  tới máy  $v_i$ .

**Yêu cầu:** Tìm phương án nâng cấp các đoạn cáp mạng sao cho tổng chi phí nâng cấp là nhỏ nhất.

**Dữ liệu:** Vào từ file văn bản UPGRANET.INP

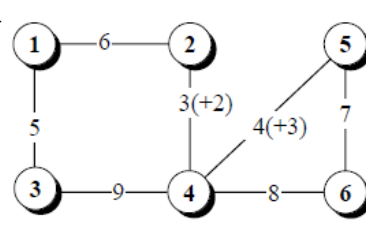
- Dòng thứ nhất chứa hai số nguyên dương  $n, m$  ( $n, m \leq 10^5$ );
- Dòng thứ  $i$  trong số  $m$  dòng tiếp theo chứa ba số nguyên dương  $u_i, v_i, w_i$  ( $w_i \leq 10^6$ ),  $i = 1, 2, \dots, m$ .

Các số trên cùng một dòng được ghi cách nhau ít nhất một dấu cách.

**Kết quả:** Ghi ra file văn bản UPGRANET.OUT một số nguyên duy nhất là tổng chi phí nâng cấp theo phương án tìm được.

**Ví dụ:**

UPGRANET . INP	UPGRANET . OUT
6 7 1 2 6 1 3 5 2 4 3 3 4 9 4 5 4 4 6 8 5 6 7	5



**Ràng buộc:** 50% số tests ứng với 50% số điểm của bài có  $n \leq 100$ .



Có  $N$  con bò ( $1 \leq N \leq 10^5$ ), để thuận tiện ta đánh số từ  $1 \rightarrow N$ , đang ăn cỏ trên  $N$  đồng cỏ, để thuận tiện ta cũng đánh số các đồng cỏ từ  $1 \rightarrow N$ . Biết rằng con bò  $i$  đang ăn cỏ trên đồng cỏ  $i$ .

Một vài cặp đồng cỏ được nối với nhau bởi 1 trong  $N - 1$  con đường 2 chiều mà các con bò có thể đi qua. Con đường  $i$  nối 2 đồng cỏ  $A_i$  và  $B_i$  ( $1 \leq A_i, B_i \leq N$ ) và có độ dài  $L_i$  ( $1 \leq L_i \leq 10^4$ ).

Các con đường được thiết kế sao cho với 2 đồng cỏ bất kỳ đều có duy nhất 1 đường đi giữa chúng. Như vậy các con đường này đã hình thành 1 cấu trúc cây.

Các chú bò rất có tinh thần tập thể và muốn được thăm thường xuyên. Vì vậy lũ bò muốn bạn giúp chúng tính toán độ dài đường đi giữa  $Q$  ( $1 \leq Q \leq 1000$ ) cặp đồng cỏ (mỗi cặp được mô tả là 2 số nguyên  $u, v$  ( $1 \leq u, v \leq N$ )).

**Dữ liệu vào:** Nhập vào từ tệp PWALK.inp

Dòng đầu ghi 2 số nguyên cách nhau bởi dấu cách:  $N$  và  $Q$

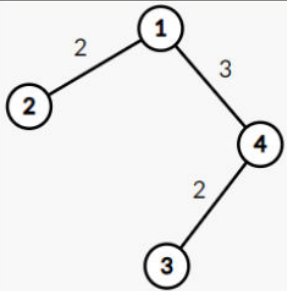
$N - 1$  dòng tiếp theo: Mỗi dòng chứa 3 số nguyên cách nhau bởi dấu cách:  $A_i, B_i$  và  $L_i$ , mô tả có đường đi trực tiếp giữa  $A_i, B_i$  và độ dài của nó là  $L_i$

$Q$  dòng tiếp theo: Mỗi dòng chứa 2 số nguyên ( $u, v$ ) khác nhau yêu cầu tính toán độ dài 2 đồng cỏ mà lũ bò muốn đi thăm qua lại.

**Kết quả ra:** Ghi ra tệp PWALK.out

Ghi  $Q$  số là kết quả của từng yêu cầu theo thứ tự, mỗi số viết trên một dòng.

**Ví dụ:**

PWALK.inp	PWALK.out	Giải thích	
4 2 2 1 2 4 3 2 1 4 3 1 2 3 2	2 7		Đường đi từ $1 \rightarrow 2$ độ dài 2. Đường đi từ 3 đến 2 độ dài 7.

### Bài 3. ROAD

Bác VA có N đồng cỏ, được kết nối với nhau bởi N-1 đoạn đường 2 chiều (có cấu trúc giống 1 cây). Lần này, vì quá già yếu, bác muốn chia N-1 đoạn đường thành nhiều con đường 2 chiều dài hơn bằng cách ghép một số đoạn đường 2 chiều lại với nhau và giao cho những người làm thuê của mình quản lí. Để tìm người quản lí xứng đáng thay thế mình, bác quyết định mỗi con đường phải có cùng độ dài. Bác đang tự hỏi độ dài mỗi con đường bằng bao nhiêu.

Nhưng bác VA lại nghĩ ra 1 vấn đề khác, đó là với mỗi độ dài K thỏa mãn

$1 \leq K \leq N-1$ , liệu có thể chia N-1 đoạn đường thành các con đường độ dài K hay không. Biết mỗi đoạn đường có độ dài là 1.

**Dữ liệu:** Vào từ file ROAD.INP gồm

+ Dòng đầu tiên là số N ( $2 \leq N \leq 100000$ )

+ N-1 dòng tiếp theo gồm 2 số u và v (u khác v) cho biết tồn tại đoạn đường từ u đến v. ( $1 \leq u, v \leq N$ ).

**Kết quả:** Ghi ra file ROAD.OUT gồm N-1 số  $a[1], a[2], \dots, a[n-1]$  một cách liên tiếp. Trong đó  $a[i]=1$  nếu tồn tại cách chia N-1 đoạn đường thành các con đường lớn có độ dài là i,  $a[i]=0$  nếu không chia được.

**Ví dụ :**

ROAD.INP	ROAD.OUT
13	111000000000
1 2	
2 3	
2 4	
4 5	
2 6	
6 7	
6 8	
8 9	
9 10	
8 11	
11 12	
12 13	

**Giải thích:**

Thỏa mãn để chia thành các con đường độ dài 1,2 hoặc 3.

Với độ dài  $K=3$  thì có thể chia thành các con đường gồm các đoạn đường nhỏ sau :

+ (13-12-11-8)

+(10-9-8-6)

+(7-6-2-3)

+(5-4-2-1)

***Ràng buộc:***

Subtask1 : 30% số điểm có  $N \leq 1000$

Subtask2: 70% số điểm không có ràng buộc gì thêm

Cho một cây có trọng số gồm  $N$  đỉnh,  $N-1$  cạnh, đỉnh gốc là đỉnh 1. Có  $Q$  truy vấn, mỗi truy vấn cho dưới dạng  $(u, v, x)$  hỏi đường đi có trọng số lớn nhất giữa cặp  $(u, v)$  trên cây nếu được phép nối một đỉnh thuộc cây con gốc  $u$  với một đỉnh thuộc cây con gốc  $v$  (đảm bảo cây con gốc  $u$ , và cây con gốc  $v$  là khác nhau) bởi một cạnh có trọng số là  $x$  ( $x \geq 0$ ) bằng bao nhiêu?

**Dữ liệu vào:** Từ tệp TREEEDGE.inp

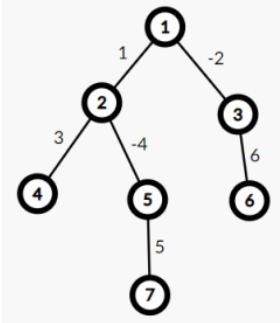
Dòng đầu ghi số  $N, Q$  ( $1 \leq N, Q \leq 2 * 10^5$ ) là số đỉnh của cây, số truy vấn.

$N - 1$  dòng tiếp ghi bộ số  $u, v, w$  ( $1 \leq u, v \leq N, |w| \leq 10^9$ ) mô tả các cạnh của cây, đỉnh đầu  $u$ , đỉnh cuối  $v$ , trọng số  $w$  của cạnh.

$Q$  dòng tiếp theo ghi bộ số  $u, v, x$  thể hiện truy vấn tìm trọng số đường đi từ  $u$  đến  $v$  lớn nhất khi được thêm một cạnh trọng số  $x$  ( $0 \leq x \leq 10^9$ ) nối một đỉnh thuộc cây con gốc  $u$  với 1 đỉnh thuộc cây con gốc  $v$ .

**Kết quả ra:** Ghi ra tệp TREEEDGE.out

**Ví dụ:**

TREEDGE.inp	TREEDGE.out	Giải thích
7 3	10	<p>Với truy vấn đầu tiên (2,3,1): Tối ưu khi thêm cạnh nối giữa 4 và 6. Tạo ra đường đi <math>2 \rightarrow 4 \rightarrow 6 \rightarrow 3</math> với tổng trọng số: <math>3+1+6=10</math>.</p> 
1 2 1	7	
1 3 -2	5	
2 4 3		
2 5 -4		
5 7 5		
3 6 6		
2 3 1		
5 4 2		
5 6 0		

## MARARHON

Thầy giáo dạy giáo dục thể chất tại trường chuyên XYZ đang cần tổ chức cuộc thi chạy cho học sinh, biết địa bàn thành phố là đồ thị vô hướng dạng cây gồm  $N$  đỉnh và  $N-1$  cạnh. Do cần giám sát, đảm bảo an toàn, giáo sư đã nhờ một chuyên gia khoa học máy tính thiết kế một camera để giám sát trên đoạn đường chạy. Chuyên gia đã đưa cho thầy giáo  $Q$  phương án. Mỗi phương án là bộ 3 số  $u, v, w$  trong đó  $u, v$  là điểm đầu, cuối của đoạn đường chạy,  $w$  là vị trí đặt camera. Bạn hãy giúp xem chuyên gia đã thực hiện đúng yêu cầu của thầy giáo đặt ra chưa.

**Dữ liệu vào:** Từ tệp Marathon.inp

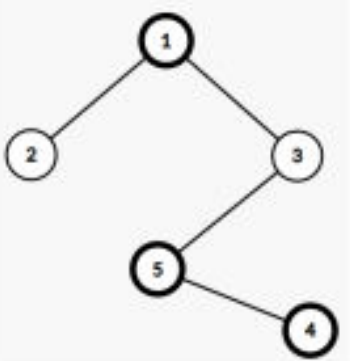
- Dòng đầu số đỉnh của đồ thị  $N$ , và số phương án chọn đường chạy  $Q$ .
- Các dòng tiếp theo thể hiện cạnh của đồ thị.
- Các dòng sau đó là  $Q$  phương án

**Kết quả ra:** Ghi ra tệp Marathon.out

Gồm  $Q$  dòng, nếu phương án đảm bảo việc lắp camera trên đường chạy thì in ra 1, ngược lại in ra 0.

**Ràng buộc:**  $1 \leq N, Q \leq 10^5$ .

**Ví dụ:**

Marathon.inp	Marathon.out	Giải thích
5 3 1 2 1 3 3 5 4 5 2 3 1 5 4 5 2 3 4	1 1 0	

## TOURIST

Tại thành phố cây, có  $N$  điểm du lịch hấp dẫn được đánh số từ 1 đến  $N$ . Thành phố có  $N-1$  con đường 2 chiều để nối các điểm du lịch. Thị trưởng thành phố phát hiện ra là việc tổ chức các tour đi từ địa điểm  $u$  đến các địa điểm được đánh số là bội của nó sẽ rất thú vị, các tour như vậy thì du khách sẽ được thăm tất cả các địa điểm trên đường đi đơn giữa 2 địa điểm này. Hỏi với tất cả cách tổ chức tour như vậy thì tổng số địa điểm được thăm là bao nhiêu?

**Dữ liệu vào:** Từ tệp Tourist.inp

Dòng đầu tiên là số địa điểm du lịch  $N$  của thành phố

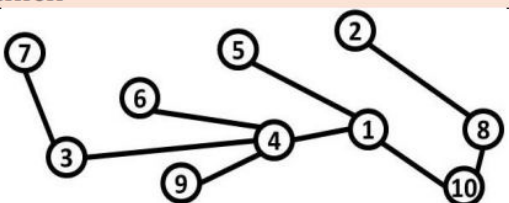
$N-1$  dòng tiếp theo thể hiện đường nối giữa các thành phố

**Kết quả ra:** Ghi ra tệp Tourist.out

Ghi tổng số địa điểm du lịch được thăm với tất cả các tour được xây dựng.

**Ràng buộc:**  $1 \leq N \leq 10^5$ .

**Ví dụ:**

Tourist.inp	Tourist.out	Giải thích
10 3 4 3 7 1 4 4 6 1 10 8 10 2 8 1 5 4 9	55	 <p>Chúng ta có tất cả các con đường và số địa điểm có thể thăm được như sau: <math>1 \rightarrow 2=4</math> ; <math>1 \rightarrow 3=3</math>; <math>1 \rightarrow 4=2</math>; <math>1 \rightarrow 5=2</math>; <math>1 \rightarrow 6=3</math>; <math>1 \rightarrow 7=4</math>; <math>1 \rightarrow 8=3</math>; <math>1 \rightarrow 9=3</math>; <math>1 \rightarrow 10=2</math>; <math>2 \rightarrow 4=5</math>; <math>2 \rightarrow 6=6</math>; <math>2 \rightarrow 8=2</math>; <math>2 \rightarrow 10=3</math>; <math>3 \rightarrow 6=3</math>; <math>3 \rightarrow 9=3</math> ; <math>4 \rightarrow 8=4</math> ; <math>5 \rightarrow 10=3</math>.</p> <p>Do đó tổng số địa điểm du lịch được thăm sẽ là: 55.</p>



## VOTREE

Cho cây gồm  $N$  đỉnh ( $N \leq 70000$ ), có gốc là đỉnh 1. Bạn cần trả lời  $Q$  truy vấn, mỗi truy vấn gồm 2 số  $u, v$ . Bạn cần tìm đỉnh xa gốc nhất, mà là tổ tiên của tất cả các đỉnh  $u, u + 1, \dots, v$ .

**Dữ liệu vào:** Từ tệp VOTREE.inp

Dòng đầu ghi 2 số nguyên dương  $N$  và  $Q$  ( $1 \leq N, Q \leq 70000$ ).

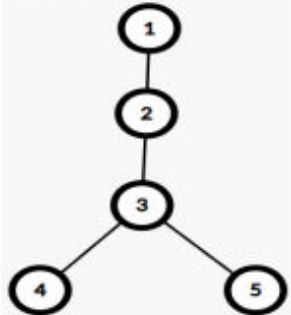
$N - 1$  dòng tiếp theo, mỗi dòng chứa 2 số nguyên dương  $u$  và  $v$ , thể hiện có 1 cạnh nối giữa 2 đỉnh  $u$  và  $v$ . ( $u \neq v; 1 \leq u, v \leq N$ ).

$Q$  dòng tiếp theo, mỗi dòng gồm 2 số nguyên dương  $u$  và  $v$  ( $1 \leq u \leq v \leq N$ ), thể hiện 1 truy vấn.

**Kết quả ra:** Ghi ra tệp VOTREE.out

Với mỗi truy vấn, in ra 1 dòng duy nhất là đáp số của truy vấn.

**Ví dụ:**

VOTREE.inp	VOTREE.out	Hình vẽ
5 3 1 2 2 3 3 4 3 5 2 5 1 3 4 5	2 1 3	

### Nối mạng

Cho  $N$  máy tính được đánh số từ 1 đến  $N$ . Chi phí nối mạng giữa máy  $i$  với máy  $j$  là  $C_{ij}$ . Ban đầu đã có sẵn  $K$  cặp máy được nối mạng.

**Yêu cầu:** Hãy tìm cách nối thêm dây cáp sao cho các máy tính trong mạng là liên thông và chi phí nối mạng là nhỏ nhất (chú ý có thể sử dụng lại hệ thống mạng cũ).

**Dữ liệu:** vào từ tập tin văn bản **NOIMANG.INP** có nội dung:

- Dòng đầu gồm hai số  $N, K$  ( $1 \leq N \leq 100000$ ,  $0 \leq K \leq 300000$ )
- $K$  dòng tiếp theo, mỗi dòng là ba số  $i, j, C_{ij}$  là thông tin về cặp máy đã nối trước đó.
- Các dòng còn lại, mỗi dòng là ba số  $i, j, C_{ij}$  là thông tin về cặp máy có thể nối với nhau trong mạng.

Trên cùng một dòng, các số viết cách nhau 1 dấu cách.

**Kết quả:** ghi vào tập tin văn bản **NOIMANG.OUT** gồm một số duy nhất là tổng chi phí nối mạng nhỏ nhất (không kể phần sử dụng lại mạng cũ).

NOIMANG.INP	NOIMANG.OUT
6 2 1 3 17 2 3 18 1 2 33 2 4 20 3 4 16 3 5 4 4 5 9 4 6 8 5 6 14	21



8 0	37
1 2 5	
1 3 7	
1 4 10	
3 5 6	
3 6 4	
4 6 8	
5 7 2	
6 7 7	
6 8 5	

## Bài 1. Phân phát

Kon Tum là một thành phố lớn gồm  $N$  điểm du lịch và  $N - 1$  con đường nối từ điểm du lịch này sang điểm du lịch khác sao cho từ một điểm du lịch bất kì có thể đi được tới tất cả các điểm còn lại.

Josuke được ngài thị trưởng giao trách nhiệm đặt các trạm thu phí trên mỗi con đường sao cho chi phí của mỗi con đường là một số nguyên dương và số lượng con đường có chi phí là 1 phải nhỏ nhất có thể. Josuke rất thích số  $K$  nên tích tất cả các chi phí trên mỗi con đường phải bằng  $K$ . Vì  $K$  là một số rất lớn nên  $K$  sẽ được biểu diễn sang tích  $M$  thừa số nguyên tố  $p_1, p_2, \dots, p_M$ . ( $p_1 \cdot p_2 \cdot \dots \cdot p_M = K$ ). Chi phí của đường đi từ điểm du lịch  $u$  đến điểm du lịch  $v$  là tổng tất cả các chi phí trên đường đi từ  $u$  đến  $v$ . Gọi  $d(i, j)$  là chi phí để đi từ  $i$  đến  $j$ . Hãy giúp Josuke phân phối các chi phí sao cho tổng chi phí để di chuyển giữa các điểm du lịch phải lớn nhất có thể. Hay nói cách khác :  $\sum_i^{n-1} \sum_j^n d(i, j)$  phải lớn nhất có thể.

**Dữ liệu:** Vào từ file PHANPHAT.INP gồm:

- Dòng đầu tiên gồm số  $N$  ( $N \leq 10^5$ )
- $N - 1$  dòng sau, dòng thứ  $i$  gồm 2 số  $u_i$  và  $v_i$  ( $1 \leq u_i, v_i \leq n$ ), tức là có đường đi từ điểm du lịch  $u_i$  đến điểm du lịch  $v_i$
- Dòng tiếp theo gồm số  $M$  ( $M \leq 6 \cdot 10^4$ )
- Dòng tiếp theo gồm  $M$  số  $p_1, p_2, \dots, p_M$  ( $2 \leq p_i \leq 6 \cdot 10^4$ )

**Kết quả:** Ghi ra file PHANPHAT.OUT gồm 1 dòng là kết quả tìm được. Vì kết quả rất lớn nên hãy lấy kết quả mod 1000000007.

**Ví dụ:**

PHANPHAT.INP	PHANPHAT.OUT
4 1 2 2 3 3 4 2 2 2	17
7 6 1 2 3 4 6 7 3	286

5 1 3 6 4 7 5 13 3	
-----------------------------	--

## Bài toán 1: Xây cầu



baitoan1.cpp

" Trên hai đường thẳng song song L1 và L2, Người ta đánh dấu trên mỗi đường N điểm, Các điểm trên đường thẳng L1 Được đánh số từ 1 đến N, từ trái qua phải, còn các điểm trên đường thẳng L2 được đánh số bởi  $P[1], P[2], \dots, P[N]$  cũng từ trái qua phải, trong đó  $P[1], P[2], \dots, P[N]$  là một hoán vị của các số  $1, 2, \dots, N$ . Ta gọi các số gán cho các điểm là số hiệu của chúng. Cho phép nối hai điểm trên 2 đường thẳng có cùng số hiệu.

Yêu Cầu : Tìm cách nối được nhiều cặp điểm nhất với điều kiện các đoạn nối không được cắt nhau.

Dữ Liệu : Vào từ File BaiToan1.Inp:

Dòng đầu tiên chứa số nguyên dương  $N (N \leq 1000)$

Dòng thứ hai chứa các số  $P[1], P[2], \dots, P[N]$

Kết Quả Ghi Ra File : BaiToan1.Out

Dòng Đầu tiên chứa K là số lượng đoạn nối tìm được

Dòng tiếp theo chứa K số hiệu của các đầu mút của các đoạn nối được ghi theo thứ tự tăng dần.

Ví Dụ :

Baitoan1.INP

9

2 5 3 8 7 4 6 9 1

baitoan1.OUT

5

2 3 4 6 9

## Bài Toán 2: Dạo Chơi Bằng Xe Buýt



bus.cpp

Trên một tuyến đường ở thành phố du lịch nổi tiếng X có ô tô Buýt công cộng phục vụ việc đi lại của du khách. Bến xe buýt có ở từng Km của tuyến đường. Mỗi lần đi qua bến xe đều đỗ cho du khách lên xuống. Mỗi bến đều có xe xuất phát từ nó, nhưng mỗi xe chỉ chạy không quá B km kể từ bến xuất phát của nó. Hành khách khi đi xe sẽ phải trả tiền cho độ dài đoạn đường mà họ ngồi trên xe. Cước phí cần trả để đi đoạn đường độ dài i là  $C_i (i=1, 2, \dots, B)$ . Một du khách xuất phát từ một bến nào đó muốn đi dạo L km trên tuyến đường nói trên. Hỏi ông ta phải lên xuống xe như thế nào để tổng số tiền phải trả cho chuyến dạo chơi bằng xe buýt là nhỏ nhất.

Dữ Liệu : Vào Từ File : Bus.Inp

Dòng đầu tiên chứa 2 số nguyên dương B,L ( $B \leq 100, L \leq 10000$ )  
Dòng thứ hai chứa B số Nguyên dương  $C_1, C_2, \dots, C_n$  , được ghi cách nhau bởi dấu trắng.

Kết Quả : Ghi ra File Văn bản : Bus.Out

Dòng đầu tiên ghi chi phí tìm được, và số lần xuống xe K .

Dòng tiếp theo ghi K số là độ dài của các đoạn đường của K lần ngồi xe.

Ví Dụ:

BUS.INP	BUS.OUT
10 15	147 3z3
12 21 31 40 49 58 69 79 90 101	3 6 6

### Bài Toán 3: Dãy Con Tăng Cực Đại



baitoan3.cpp

" Cho một dãy số nguyên dương  $A_1, A_2, \dots, A_n$ . Hãy tìm bớt một số ít nhất các phần tử của dãy số nguyên đó và giữ nguyên thứ tự các phần tử còn lại sao cho dãy số còn lại là một dãy tăng dần. Ta gọi dãy số nguyên tăng dần còn lại sau khi đã tìm bớt một số phần tử là dãy con của dãy đã cho.

Dữ Liệu : Vào từ File BaiToan3.Inp :

Dòng đầu tiên ghi số N là số phần tử ( $N \leq 10000$ )

Dòng tiếp theo ghi N số là các số nguyên của dãy

Kết Quả : Ghi Ra File : BaiToan3.Out

Dòng đầu tiên ghi số phần tử của dãy con lớn nhất đó

Dòng thứ hai ghi các số của dãy cần tìm .

### Bài toán 6: Vòng Quanh Thế Giới



bai4.cpp

(Đề Thi Học Sinh Giỏi Quốc Gia 2000-2001 - Bảng A )

Trên tuyến đường của xe chở khách du lịch vòng quanh thế giới xuất phát từ bến X có N khách sạn đánh số từ 1 đến N theo thứ tự xuất hiện trên tuyến đường, trong đó khách sạn N là địa điểm cuối cùng của tuyến đường mà tại đó xe bắt buộc phải dừng.

Khách sạn I cách địa điểm xuất phát  $A_i$  Km ( $i=1,2,\dots,N$ );  $A_1 < A_2 < \dots < A_N$  . Để đảm bảo sức khỏe cho khách hàng, theo tính toán của các nhà chuyên môn, sau khi đã chạy được P (Km) xe nên dừng lại cho khách nghỉ ở khách sạn. Vì thế, nếu xe dừng lại cho khách nghỉ ở khách sạn sau khi đã đi được Q(Km) thì lái xe phải trả một lượng phạt là :  $(Q-P)^2$ .

Ví Dụ :

Với  $N=4$  , $P=300$ , $A_1=250$ ,  $A_2=310$ ,  $A_3=550$  , $A_4=590$  . Xe bắt buộc phải dừng lại ở khách sạn 4 là địa điểm cuối cùng của hành trình. Nếu trên đường đi lái xe chỉ dừng lại tại khách sạn thứ 2 thì lượng phạt phải trả là :

$$(310-300)^2 + ((590-310)-300)^2 = 500$$

Yêu Cầu : Hãy xác định xem trên tuyến đường đến khách sạn N, xe cần dừng lại nghỉ ở những khách sạn nào để tổng lượng phạt mà lái xe phải trả là nhỏ nhất.

Dữ Liệu : Vào từ File văn bản có tên Bai4.Inp :

Dòng đầu tiên chứa số nguyên dương N( $N \leq 10000$ );

Dòng thứ hai chứa số nguyên dương P ( $P \leq 500$ );

Dòng thứ ba chứa các số nguyên dương  $A_1, A_2, A_3, \dots, A_n$  (hai số liên tiếp cách nhau ít nhất bởi 1 dấu cách) ( $A_i \leq 2000000$  , $i=1,2,\dots,N$ ).

Kết Quả : Ghi ra File Văn Bản Bai4.Out:

Dòng đầu tiên ghi Z là lượng phạt mà lái xe phải trả ;

Dòng thứ hai ghi K là tổng số khách sạn mà lái xe cần dừng lại cho khách nghỉ;

Dòng thứ ba chỉ chứa chỉ số của K khách sạn mà xe dừng lại cho khách nghỉ (Trong đó nhất thiết phải có khách sạn thứ N).

Ví Dụ:

BAI4.INP	BAI4.OUT
4	50
300	2
250 310 550 590	2 4

CHUYÊN ĐỀ

CẤU TRÚC DỮ LIỆU NÂNG CAO  
DISJOINT – SET – UNION

## PHẦN I: MỞ ĐẦU

Hệ thống các tập không giao nhau (disjoint-set-union – DSU) là một cách tổ chức dữ liệu đủ đơn giản nhưng rất hiệu quả để giải quyết nhiều loại bài toán khác nhau. Trong các năm gần đây các đề thi học sinh quốc gia, thi chuyên hải ... bài tập về đồ thị luôn chiếm một tỉ lệ lớn. Các bài tập ngày càng nâng cao về độ khó và đòi hỏi sử dụng tốt một số cấu trúc dữ liệu để giảm độ phức tạp. DSU là một cấu trúc rất hữu dụng và là nền tảng cho một số thuật toán như thuật toán Kruskal. Sau khi tìm hiểu tôi nhận thấy DSU là một cấu trúc dữ liệu hay và quan trọng. Do đó tôi quyết định chọn chuyên đề cấu trúc dữ liệu DSU.

Chuyên đề tổng hợp kiến thức về DSU đặc biệt tôi cũng đưa ra luôn hai kỹ thuật cải tiến cho phép thực hiện một số phép xử lý với độ phức tạp xấp xỉ  $O(1)$ . Trong khuôn khổ chuyên đề “Cấu trúc dữ liệu nâng cao disjoint-set-union” tôi chỉ xin trao đổi với các bạn đồng nghiệp một số bài tập có thể ứng dụng cấu trúc dữ liệu DSU này. Rất mong chuyên đề sẽ cung cấp cho các bạn đồng nghiệp và các em học sinh một phần kiến thức bổ ích.



## PHẦN II: NỘI DUNG

### I. LÝ THUYẾT

#### 1. Định nghĩa

Trong khoa học máy tính, Cấu trúc dữ liệu Disjoint – Set - Union (hợp của các tập hợp không giao nhau) là một cấu trúc dữ liệu quản lý các phần tử đã được phân chia thành các tập con không giao nhau. DSU cung cấp các phép toán: thêm các tập con mới, kết hợp các tập con lại với nhau, xác định các phần tử có trong cùng một tập hợp hay không.

DSU còn có các tên gọi khác như Disjoint-Set Data Structure, Union-Find.

#### 2. Các phép toán cơ bản

Ban đầu mỗi phần tử của dữ liệu thuộc một tập riêng. Các phép xử lý cơ sở trên cấu trúc là:

- **Make\_set(v):** Tạo ra một tập hợp mới để chứa phần tử mới v.
- **Union(a, b):** Hợp nhất hai tập hợp (tập hợp chứa phần tử a và tập hợp chứa phần tử b).
- **Find\_set(v):** Trả về phần tử đại diện của tập hợp mà chứa phần tử v. Phần tử đại diện này được lựa chọn cho mỗi tập hợp và phần tử này có thể thay đổi và được chọn lại sau phép toán Union\_set. Phần tử đại diện này được sử dụng để kiểm tra hai phần tử có cùng một tập hợp hay không.

#### • Xét bài toán

Cho đồ thị gồm  $n$  đỉnh được đánh số :  $0, 1, \dots, n - 1$  và không có cạnh nào. Lần lượt thêm  $m$  cạnh vô hướng vào đồ thị. Cho biết số miền liên thông sau mỗi bước thêm cạnh

#### Thuật toán

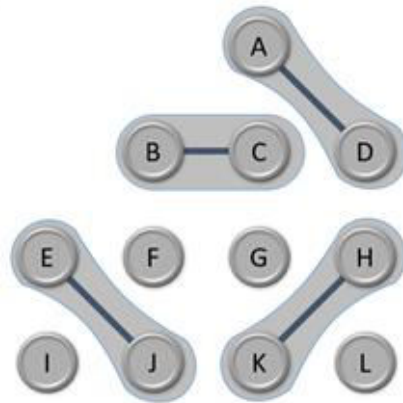
- Khởi tạo:  $n$  tập hợp, mỗi tập gồm 1 đỉnh, số miền liên thông  $cc = n =$  số tập
- Thêm cạnh  $(u, v)$ :

Trường hợp 1: Nếu  $u, v$  thuộc cùng 1 tập  $\rightarrow$  bỏ qua

Trường hợp 2: Nếu  $u, v$  thuộc hai tập khác nhau  $\rightarrow$  hợp nhất tập chứa  $u$  và tập chứa  $v$ ;  $--cc$ .

Giải mã

```
for ( $\forall u \in V$ ) MakeSet( $u$ );
cc = n;
for ( $\forall (u, v) \in E$  thêm vào)
{
    r = FindSet( $u$ ); s = FindSet( $v$ );
    if ( $r \neq s$ )
    {
        Union( $r, s$ );
        --cc;
    }
    Output  $\leftarrow$  cc;
}
```

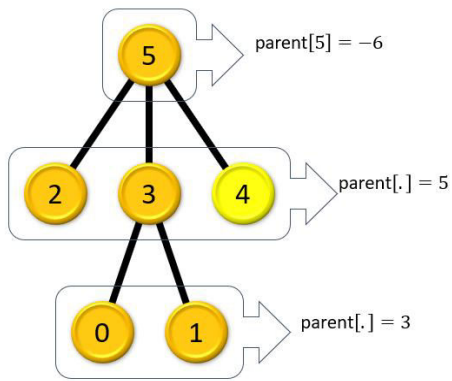


**DSU được cài đặt hiệu quả với cấu trúc cây**

Ban đầu chúng ta có rừng các tập rời nhau

- Đỉnh:  $0 \dots n - 1$
- Tập hợp đỉnh  $\equiv$  cây
  - Mỗi nút chứa một đỉnh
  - Nút  $\equiv$  đỉnh chứa trong nút
- Biểu diễn cây: sử dụng mảng  $parent[0 \dots n - 1]$  trong đó
  - $v$  không phải gốc:  $parent[v] =$  nút cha của  $v$
  - $v$  là gốc:  $parent[v] = -$ Số nút trong cây gốc  $v$
  - Khởi tạo:  $parent[0 \dots n - 1] = -1$

Ví dụ



$\{0,1,2,3,4,5\}$

//Dựng cây ứng với tập  $\{u\}$

```
void MakeSet(int u)
```

```
{
```

```
    parent[u] = -1;
```

```
}
```

```
void solve()
```

```
{
```

```
for ( $\forall u \in V$ ) MakeSet(u);
```

```
cc = n;
```

```
for ( $\forall (u, v) \in E$  thêm vào)
```

```
{
```

```
    r = FindSet(u); s = FindSet(v);
```

```
    if (r  $\neq$  s)
```

```
    {
```

```
        Union(r, s);
```

```
        --cc;
```

```
    }
```

```
    Output  $\leftarrow$  cc;
```

```
}
```

```
}
```

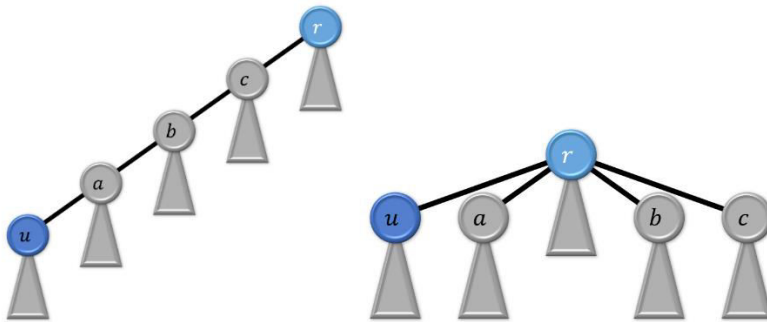
a) **Viết hàm FindSet**

Sử dụng phương pháp nén đường

**FindSet(u)**: Tìm gốc cây chứa u

Cách làm: Đi từ u lên gốc, kết quả trả về gốc. Dọc đường đi, đặt các nút đi qua làm con trực tiếp của gốc

```
int FindSet(int u)
{
    return parent[u] < 0 ? u : parent[u] = FindSet(parent[u]);
}
```



ĐPT: phương pháp nén đường mang lại hiệu quả trung bình  $O(\log n)$  cho mỗi truy vấn.

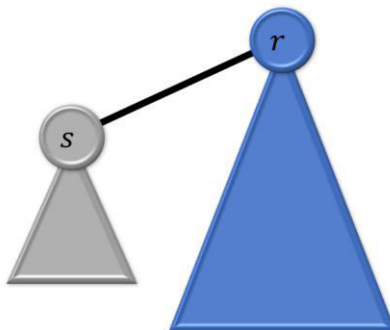
## b) Viết hàm Union

Unio(r,s): Hợp nhất cây gốc r và cây gốc s

Sử dụng phương pháp Hợp theo hạng(Union – by – Rank)

Cách làm: lấy gốc cây có lượng đỉnh bé hơn làm con của gốc cây có số lượng đỉnh to hơn

```
void Union(int r, int s)
{
    if (parent[s] < parent[r])
        swap(r, s);
    parent[r] += parent[s];
    parent[s] = r;
}
```



**Thời gian thực hiện hai phép xử lý trên:**

- Việc viết hai hàm FindSet và Union như trên sẽ làm cho độ phức tạp xử lý mỗi truy vấn trở thành một hằng. Chứng minh điều này khá phức tạp và đã nêu trong nhiều tài liệu khác nhau, ví dụ Tarjan năm 1975, Kurt Mehlhorn và Peter Sanders năm 2008.
- Người ta đã chứng minh được rằng độ phức tạp của mỗi truy vấn khi viết các phép xử lý trên là  $O(\alpha(n))$  trong đó  $\alpha(n)$  là hàm nghịch đảo Ackerman có độ tăng rất chậm, đến mức trong phạm vi  $n \leq 10^{600}$   $\alpha(n)$  có giá trị không quá 4. Vì vậy có thể nói hệ thống các tập không giao nhau hoạt động với độ phức tạp gần như một hằng.

## II. Bài tập vận dụng

Bài 1. <https://oj.vnoi.info/problem/ioibin>

### *Bin – Các thùng nước*

Có N thùng nước được đánh số từ 1 đến N, giữa 2 thùng bất kỳ đều có một ống nối có một van có thể khóa hoặc mở. Ở trạng thái ban đầu tất cả các van đều đóng.

Bạn được cho một số yêu cầu, trong đó mỗi yêu cầu có 2 dạng:

Dạng X Y 1 có ý nghĩa là bạn cần mở van nối giữa 2 thùng X và Y.

Dạng X Y 2 có ý nghĩa là bạn cần cho biết với trạng thái các van đang mở / khóa như hiện tại thì 2 thùng X và Y có thuộc cùng một nhóm bình thông nhau hay không? Hai thùng được coi là thuộc cùng một nhóm bình thông nhau nếu nước từ bình này có thể chảy đến được bình kia qua một số ống có van đang mở.

**Input:** BIN.INP. Dòng đầu tiên ghi một số nguyên dương P là số yêu cầu. Trong P dòng tiếp theo, mỗi dòng ghi ba số nguyên dương X, Y, Z với ý nghĩa có yêu cầu loại Z với 2 thùng X và Y.

**Output:** BIN.OUT. Với mỗi yêu cầu dạng X Y 2 (với Z = 2) bạn cần ghi ra số 0 hoặc 1 trên 1 dòng tùy thuộc 2 thùng X và Y không thuộc hoặc thuộc cùng một nhóm bình. Giới hạn:  $1 \leq N \leq 10000$ ,  $1 \leq P \leq 50000$ . Thời gian: 1 s/test. Bộ nhớ: 1 MB

BIN.INP	BIN.OUT
9	0
1 2 2	0
1 2 1	1
3 7 2	0
2 3 1	1
1 3 2	0
2 4 2	
1 4 1	
3 4 2	
1 7 2	

Ví dụ:

*Gợi ý.* Đây là bài toán tìm vùng liên thông. Chương trình sau dùng phương pháp hợp nhất cây.

```
#include <bits/stdc++.h>
#define fop(i,a,b) for(int i = a; i <= b; i++)
#define fom(i,a,b) for(int i = a; i >= b; i--)
#define inp() freopen("BIN.inp","r",stdin)
#define out() freopen("BIN.out","w",stdout)
using namespace std;
typedef long long ll;
int P,X,Y,Z,r1,r2;
int parent[100005];
int find_root(int u)
{
    return parent[u] < 0 ? u : parent[u] =
find_root(parent[u]);
}
void union_(int r, int s)
{
    if (parent[s] < parent[r])
        swap(r, s);
    parent[r] += parent[s];
    parent[s] = r;
}
int main()
{
    inp();
    out();
    cin >> P;
    memset(parent,-1,sizeof(parent));
    while(P)
    {
        cin >> X >> Y >> Z;
        if(Z==1)
        {
            r1 = find_root(X);
            r2 = find_root(Y);
```

```

        if(r1 != r2) union_(r1,r2);
    }
    else
    {
        r1 = find_root(X);
        r2 = find_root(Y);
        if(r1==r2) cout << 1<<'\n';
        else cout << 0 << '\n';
    }
    P--;
}
return 0;
}

```

**Cảm nhận:** Bài này có thể làm bằng nhiều cách như duyệt DFS để liệt kê các TPLT nhưng nếu áp dụng thuật toán hợp nhất cây mới hiệu quả và giải quyết bài toán một cách triệt để.

## Bài 2 - Thay thế ký tự

Cho hai xâu ký tự  $s$  và  $t$  đều có  $n$  ký tự là các chữ cái tiếng Anh in thường. Người ta muốn thay thế các ký tự trong hai xâu để chúng giống hệt nhau. Với một phép biến đổi, ta có thể thay đổi một số chữ cái trên 2 xâu. Bạn hãy tính toán số phép biến đổi tối thiểu để hoàn thành việc này.

Chính xác là, Bạn sử dụng các phép biến đổi dạng  $R(c1, c2)$  (trong đó  $c1$  và  $c2$  là các chữ cái). Bạn có thể thực hiện một phép biến đổi nào đó với số lần tùy ý để biến đổi một chữ cái  $c1$  thành một chữ cái  $c2$  và ngược lại trên cả hai hai xâu  $s$  và  $t$ . Bạn cần tìm số phép biến đổi tối thiểu để cho  $s$  và  $t$  giống hệt nhau. Thêm nữa, bạn cần in ra chi tiết về các phép biến đổi đó. Xem ví dụ để rõ hơn.

### *Dữ liệu*

- Dòng đầu chứa số nguyên  $n$  ( $1 \leq n \leq 10^5$ ) là độ dài các xâu ký tự.
- Dòng thứ hai chứa  $n$  chữ cái tiếng Anh in thường, mô tả xâu  $s$ .
- Dòng thứ ba chứa  $n$  chữ cái tiếng Anh in thường, mô tả xâu  $t$ .

### *Kết quả*

- Dòng đầu in ra số nguyên  $k$  là tổng số phép biến đổi tối thiểu cần thực hiện

- Trong  $k$  dòng tiếp theo, mỗi dòng in ra một cặp ký tự  $c1, c2$  cách nhau một dấu cách để mô tả về một phép biến đổi. Các cặp này có thể in theo trật tự bất kỳ. Chú ý, các phép biến đổi có thể không phải duy nhất.

*Ví dụ*

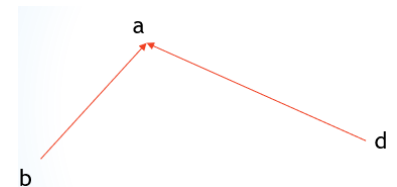
Replace.inp	Replace.out	Replace.inp	Replace.out
3	2	8	7
abb	a d	drpepper	l e
dad	b a	cocacola	e d
			d c
			c p
			p o
			o r
			r a

### Thuật toán:

- Với mỗi cặp ký tự của 2 xâu ta kiểm tra xem có cách biến đổi nào đưa về giống nhau không tức là kiểm tra xem có đường đi từ  $a$  đến  $b$  ko. Nếu coi mỗi phép thay thế là 1 cạnh trên đồ thị. Mỗi ký tự là 1 đỉnh.

Input1	Output1
3	2
abb	a d
dad	b a

- Ví dụ
- Ta có thể dùng 2 phép biến đổi:  $(a, d)$  và  $(b, a)$ . Như vậy các chữ cái đầu sẽ trùng khớp khi ta sẽ thay thế chữ  $a$  bằng  $d$ . Các chữ cái thứ hai sẽ trùng khớp khi ta thay  $b$  bằng  $a$ . Các ký tự thứ ba sẽ trùng khớp khi ta thay thế  $b$  bằng  $a$  và  $a$  bằng  $d$ .
- Đọc ký tự đầu tiên của mỗi xâu:  $a$  và  $d$ , thuộc 2 cây khác nhau, hợp nhất cây chứa 2 đỉnh, **res++**
- Đọc ký tự thứ hai của mỗi xâu:  $b$  và  $a$ , thuộc 2 cây khác nhau, hợp nhất cây chứa 2 đỉnh. **ress++**
- Đọc ký tự thứ hai của mỗi xâu:  $b$  và  $d$ , thuộc cùng một cây, bỏ qua.
- Xuất: **res**
- Xuất: **v** và **parent[v]**



### Code

```
include<bits/stdc++.h>
```



```

using namespace std;
const int MAXN = 256;
string s, t;
int par[MAXN];
int F(int x){
    return par[x]==x ? x : F(par[x]);
}
int main(){
    ios_base::sync_with_stdio(0);
    cin.tie();
    cout.tie();
    freopen("replace.inp", "r", stdin);
    freopen("replace.out", "w", stdout);
    int n;
    cin >> n >> s >> t;
    for(int i = 0; i < MAXN; ++i) par[i]=i;
    int res = 0;
    for(int i = 0; i < n; ++i){
        int x = F(s[i]), y = F(t[i]);
        if(x != y){
            par[x] = y;
            ++res;
        }
    }
    cout << res << endl;
    for(int i = 0; i < MAXN; ++i){
        if(par[i] != i){
            cout <<(char) i <<' ' <<(char)par[i] << endl;
        }
    }
    return 0;
}

```

**Cảm nhận:** Mô hình hoá bài toán trên đồ thị bài toán đưa về kiểm tra mỗi cặp kí tự của 2 xâu ta kiểm tra xem có đường đi từ a đến b ko. Ta sẽ hợp nhất dần các cạnh (thực hiện phép biến đổi  $R(c1, c2)$  để đồ thị liên thông).

**Bài 3. Moocast (Nguồn: USACO 2016)**

Nông dân John có  $N$  con bò ( $1 \leq N \leq 1000$ ) muốn tổ chức một hệ thống "moocast" khẩn cấp để phát các thông điệp quan trọng giữa chúng.

Thay vì la ó với nhau trong khoảng cách dài, những con bò quyết định trang bị cho mình bộ đàm, mỗi con một bộ đàm. Mỗi bộ đàm này đều có bán kính truyền giới hạn, nhưng các con bò có thể chuyển tiếp thông điệp cho nhau theo một con đường bao gồm một số bước nhảy, vì vậy không nhất thiết mỗi con bò đều có thể truyền trực tiếp cho mọi con bò khác.

Những con bò cần quyết định số tiền sẽ chi cho bộ đàm của chúng. Nếu chúng chi  $X$  đô la, mỗi con bò sẽ nhận được một bộ đàm có khả năng truyền đi một khoảng cách là  $\sqrt{X}$ . Tức là, khoảng cách bình phương giữa hai con bò phải nhiều nhất là  $X$  để chúng có thể giao tiếp.

Vui lòng giúp những con bò xác định giá trị nguyên tối thiểu của  $X$  sao cho cuối cùng một chương trình phát sóng từ bất kỳ con bò nào cũng có thể đến được với mọi con bò khác.

**DLV: (tệp `moocast.inp`):**

- Dòng đầu tiên của đầu vào chứa  $N$ .
- $N$  dòng tiếp theo, mỗi dòng chứa tọa độ  $x$  và  $y$  của một con bò duy nhất. Cả hai là các số nguyên trong phạm vi  $0 \dots 25.000$ .

**DLR: (tệp `moocast.out`):**

- Viết một dòng kết quả duy nhất chứa số nguyên  $X$  cho biết số tiền tối thiểu mà những con bò phải chi cho bộ đàm.

<code>moocast.inp</code>	<code>moocast.out</code>
4 1 3 5 4 7 2 6 1	17

Ví dụ

**Thuật toán:**

- Trong bài toán này, chúng ta có  $N$  con bò nằm rải rác trên mặt phẳng và muốn tính khoảng cách  $D$  tối thiểu sao cho tất cả các con bò có thể giao tiếp với nhau (thông qua một số con bò trung gian), vì hai con bò chỉ có thể giao tiếp trực tiếp nếu khoảng cách giữa chúng nhỏ hơn hoặc bằng  $D$ .

- Chúng ta có thể mô hình bài toán này như một bài toán đồ thị trong đó mỗi con bò là một đỉnh nối với mọi con bò khác với một cạnh với trọng lượng là khoảng cách giữa hai con bò. Do đó, chúng ta muốn tính trọng số cạnh tối thiểu sao cho đồ thị được liên thông.
- Chúng ta có thể giải quyết vấn đề này bằng cách sử dụng cấu trúc dữ liệu union-find. Một cấu trúc dữ liệu hỗ trợ hai hoạt động:
  - Find(x) - trả về một định danh duy nhất cho đỉnh x  $\text{Find}(x) = \text{Find}(y)$  khi và chỉ khi x và y nằm trong cùng một thành phần liên thông.
  - Merge(x, y) - nối x và y với một cạnh.
- Với N đỉnh ban đầu đều bị ngắt kết nối giữa các cặp, chúng ta có thể lặp lại các cạnh theo thứ tự không giảm theo trọng số và thêm một cạnh giữa hai đỉnh nếu chúng chưa nằm trong cùng một TPLT. Sau khi chúng ta thêm N - 1 cạnh, đồ thị đã được kết nối và cạnh cuối cùng cho chúng ta câu trả lời.

## Code

```
#include <bits/stdc++.h>
#define ll long long
#define DD double
#define LD long double
#define PII pair<int, int>
#define F first
#define S second
#define MP make_pair
#define VI vector<int>
#define PB push_back
#define Hm(I) (1<<(I))
#define Gr(I) ((I)&(-(I)))
#define getbit(I, X) (((X)>>(I))&1)
#define MS(A, X) memset(A, X, sizeof(A))
#define ffu(I, CUOI) for(int I=1; I<=CUOI; I++)
#define ffd(I, DAU) for(int I=DAU; I>=1; I--)
#define fou(I, DAU, CUOI) for(int I=DAU; I<=CUOI; I++)
#define fod(I, DAU, CUOI) for(int I=DAU; I>=CUOI; I--)
#define _bp(X) ((X)*(X))
```

```

#define _kc2(X, Y, XX, YY) (_bp((X)-(XX))+_bp((Y)-(YY)))
using namespace std;
const int NMX=4320000, oo=0x3C3C3C3C, Bs=10000000007;
int n, m=0, parent[NMX], sz[NMX], nt=0, kq;
PII a[NMX];
pair<PII, int> e[NMX];
bool cmp(pair<PII, int> x, pair<PII, int> y)
{
    return x.S<y.S;
}
int DVy(int x)
{
    return x==parent[x] ? x : parent[x]=DVy(parent[x]);
}
void TNg(int x, int y)
{
    x=DVy(x);
    y=DVy(y);
    if (x!=y)
    {
        if (sz[x]<sz[y]) swap(x, y);
        parent[y]=x;
        sz[x]+=sz[y];
    }
}
int main()
{
    freopen("moocast.INP", "r", stdin);
    freopen("moocast.OUT", "w", stdout);
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);
    cout.tie(NULL);
    cin >> n;
    ffu(i, n) cin >> a[i].F >> a[i].S;
}

```

```

        ffu(i, n-1)   fou(j, i+1, n)   e[++m]=MP(MP(i, j),
_kc2(a[i].F, a[i].S, a[j].F, a[j].S));
    sort(e+1, e+1+m, cmp);
    ffu(i, n)
    {
        parent[i]=i;
        sz[i]=1;
    }
    ffu(i, m)
    {
        if (nt==n-1) break;
        if (DVy(e[i].F.F)==DVy(e[i].F.S)) continue;
        nt++;
        TNg(e[i].F.F, e[i].F.S);
        kq=e[i].S;
    }
    cout << kq;
    return 0;
}

```

**Cảm nhận:** Bài này áp dụng đúng cách làm của DSU. Đầu tiên các cạnh của đồ thị chưa được kết nối. Sau đó kết nạp dần các cạnh, khi kết nạp đủ  $n-1$  cạnh đồ thị liên thông và cạnh cuối cùng chính là kết quả của bài toán. Chương trình bài này ban đầu gán  $\text{parent}[i]=i$ ; ( $i=1 \rightarrow n$ ).

#### Bài 4. GALAKSIJA (Nguồn COCI 2015)

Cách đây rất lâu trong một thiên hà xa có  $N$  hành tinh. Ngoài ra còn có  $N - 1$  con đường kết nối tất cả các hành tinh (trực tiếp hoặc gián tiếp). Nói cách khác, mạng lưới các hành tinh và những con đường tạo thành một cây. Ngoài ra, mỗi con đường dẫn liệt kê với một số nguyên biểu thị sự tò mò của con đường.

Một cặp hành tinh  $A, B$  thật nhàm chán nếu những điều sau đây là:

- $A$  và  $B$  là các hành tinh khác nhau
- Đường đi giữa hành tinh  $A$  và  $B$  có thể sử dụng một hoặc nhiều đường
- XOR nhị phân của sự tò mò của tất cả các con đường đó bằng 0

Than ôi, thời thế đã thay đổi và một hoàng đế độc ác đang cai trị thiên hà. Ông ta quyết định sử dụng Thần lực để phá hủy tất cả các con đường nối các hành tinh theo một thứ tự nhất định.

Bạn hãy xác định số lượng cặp hành tinh nhằm chẵn trước khi hoàng đế bắt đầu sự hủy diệt và sau khi mỗi lần phá hủy.

**DLV: GALAKSIJA.INP**

- Dòng đầu tiên của đầu vào chứa số nguyên  $N$  ( $1 \leq N \leq 100\,000$ ).
- Mỗi dòng trong số  $N - 1$  dòng sau chứa ba số nguyên  $A_i, B_i, Z_i$  ( $1 \leq A_i, B_i \leq N, 0 \leq Z_i \leq 1\,000\,000\,000$ ) biểu thị rằng hành tinh  $A_i$  và  $B_i$  được kết nối trực tiếp với một con đường tò mò  $Z_i$ .
- Dòng sau chứa hoán vị của  $N - 1$  số nguyên đầu tiên biểu thị thứ tự trong đó hoàng đế đang phá hủy các con đường. Nếu phần tử của hoán vị là  $j$ , thì hoàng đế đã phá hủy con đường giữa hai hành tinh  $A_j$  và  $B_j$  trong cùng một bước.

**DLR: GALAKSIJA.OUT**

- Đầu ra phải chứa  $N$  dòng, dòng thứ  $k$  chứa số cặp hành tinh buồn chán  $A, B$  từ nhiệm vụ sau khi hoàng đế phá hủy đúng  $k - 1$  con đường.

*Ràng buộc:*

- 20% tổng số điểm, nó sẽ giữ  $N \leq 1\,000$ .
- 30% tổng số điểm, sự tò mò của mọi con đường sẽ bằng 0

Ví dụ

<b>GALAKSIJA.INP</b>	<b>GALAKSIJA.OUT</b>
<b>2</b> <b>1 2 0</b> <b>1</b>	<b>1</b> <b>0</b>
<b>3</b> <b>1 2 4</b> <b>2 3 4</b> <b>1 2</b>	<b>1</b> <b>0</b> <b>0</b>
<b>4</b>	<b>6</b>

<b>1 2 0</b>	<b>3</b>
<b>2 3 0</b>	<b>1</b>
<b>2 4 0</b>	<b>0</b>
<b>3 1 2</b>	

- Giải thích ví dụ đầu tiên: Trước khi bị hủy diệt, con đường giữa hành tinh 1 và 2 rất nhàm chán. Sau hủy diệt, con đường giữa chúng không tồn tại nữa.
- Giải thích ví dụ thứ hai: Trước khi bị hủy diệt, một cặp hành tinh (1, 3) là nhàm chán. Con đường giữa 1 và 3 không còn có thể xảy ra sau lần phá hủy đầu tiên và sau lần hủy diệt thứ hai, và không có cặp nào trong số các cặp còn lại của hành tinh là nhàm chán.
- Giải thích ví dụ thứ ba: Lưu ý rằng trong ví dụ này, mỗi cặp hành tinh có một đường đi có thể giữa chúng là nhàm chán bởi vì tất cả các con đường có sự tò mò 0.

### Thuật toán

- Chúng ta chọn một nút tùy ý  $r$  làm gốc của một số cây. Ký hiệu  $P_x$  là tổng XOR của tất cả các điểm tò mò trên đường dẫn từ nút  $r$  đến nút  $x$ . Dễ dàng nhận thấy rằng một cặp hành tinh  $x, y$  là nhàm chán nếu và chỉ nếu  $P_x \text{ XOR } P_y = 0 \Leftrightarrow P_x = P_y$ .
- Ta có thể "đảo ngược" dữ liệu đầu vào, thay vì phá hủy đường dẫn, chúng ta có thể thêm chúng.
- Để kết nối các thành phần, chúng tôi sẽ sử dụng thuật toán DSU, đối với mỗi thành phần, ta nhớ gốc, kích thước của nó và ngoài ra, một map ghi nhớ số lần  $P_x$  xuất hiện trong thành phần.
- Khi chúng ta cần kết nối hai thành phần, thành phần mới được tạo sẽ có gốc là gốc của thành phần lớn hơn. Bây giờ chúng ta phải duyệt toàn bộ thành phần nhỏ hơn (tức là sử dụng thuật toán DFS) và sửa các giá trị trong các giá trị lớn hơn map của thành phần.
- Độ phức tạp thời gian của thuật toán này là  $O(N \lg^2 N)$

### Code

```
#include <cstdio>
#include <cassert>
```

```

#include <map>
#include <algorithm>
#include <vector>
using namespace std;
const int MAXN = 100005;
typedef pair <int, int> pii;
typedef long long llint;
llint curr;
llint ans[MAXN];
int n;
int a[MAXN], b[MAXN], z[MAXN];
int p[MAXN];
int path[MAXN];
int dad[MAXN];
int sz[MAXN];
map <int, vector<int>> M[MAXN];
void join (int a, int b, int c) {
    if (sz[dad[a]] < sz[dad[b]]) swap(a, b);
    int da = dad[a];
    int db = dad[b];
    for (auto it: M[db])
        for (auto x: it.second)
            curr += M[da][path[a] ^ c ^ path[b] ^
path[x]].size();
    int old = path[b];
    for (auto it: M[db]) {
        for (auto x: it.second) {
            path[x] = path[x] ^ old ^ c ^ path[a];
            dad[x] = da;
            M[da][path[x]].push_back(x);
        }
    }
    sz[da] += sz[db];
    M[db].clear();
    sz[db] = 0;
}

```



```

int main (void) {
    scanf("%d", &n);
    for (int i = 0; i < n-1; ++i) {
        scanf("%d%d%d", &a[i], &b[i], &z[i]);
        --a[i]; --b[i];
    }
    for (int i = 0; i < n-1; ++i) scanf("%d", &p[i]);
    for (int i = 0; i < n-1; ++i) --p[i];
    for (int i = 0; i < n; ++i) {
        M[i][0].push_back(i);
        dad[i] = i;
        sz[i] = 1;
    }
    for (int i = n-2; i >= 0; --i) {
        join(a[p[i]], b[p[i]], z[p[i]]);
        ans[i] = curr;
    }
    ans[n-1] = 0;
    for (int i = 0; i < n; ++i)
        printf("%lld\n", ans[i]);
    return 0;
}

```

**Cảm nhận:** Các bài toán áp dụng thuật toán DSU hầu như là ta phải tư duy bài toán theo chiều ngược lại. Giả sử các cặp đỉnh chưa được kết nối với nhau sau đó ta áp dụng DSU để gộp các thành phần vào. Đây là dạng bài tương đối khó với học sinh.

### **Bài 5. Sjekira (Nguồn COCI 2020)**

Mirko cảm thấy mệt mỏi với công việc hàng ngày của mình vì vậy anh quyết định sống một cuộc sống đơn giản và chuyển đến một vùng nông thôn. Tuy nhiên, mùa đông ở ngôi làng xa xôi anh ấy mới chuyển đến rất khắc nghiệt, vì vậy anh ta quyết định tự mình đi chặt củi.

Hôm nay, anh ấy sẽ chặt chiếc cây đầu tiên của mình. Trước khi cắt, anh ta dán nhãn các bộ phận của thân cây đủ nhỏ để vừa với lò sưởi và đo độ cứng của

chúng. Mirko là một lập trình viên, vì vậy anh ấy nhận thấy rằng các bộ phận và mối liên hệ giữa chúng tạo thành biểu đồ cây.

Thiệt hại trên chiếc rìu của anh ta do cắt một kết nối trên thân cây bằng **tổng của độ cứng tối đa trong hai thành phần được kết nối được tạo thành bằng cách cắt kết nối**. Mirko chỉ có một chiếc rìu, vì vậy anh ấy muốn tổng sát thương càng nhỏ càng tốt. Anh ấy muốn biết Tổng thiệt hại tối thiểu đối với chiếc rìu, nếu anh ta cắt toàn bộ thân cây thành các bộ phận nhỏ vừa với lò sưởi.

**DLV: Sjekira.inp**

- Dòng đầu tiên chứa số nguyên  $n$ , số phần. Các phần được gắn nhãn bởi các số nguyên từ 1 đến  $n$ .
- Dòng thứ hai chứa  $n$  số nguyên  $t_i$  ( $1 \leq t_i \leq 10^9$ ). Số  $t_i$  là độ cứng của phần có nhãn  $i$ .
- Mỗi dòng trong số  $n - 1$  dòng sau chứa hai số nguyên  $x$  và  $y$  ( $1 \leq x, y \leq n$ ) - nhãn của các bộ phận là kết nối trực tiếp.

**DLR: Sjekira.out**

- Tạo ra tổng thiệt hại tối thiểu sau  $n - 1$  lần cắt.

Ràng buộc:

- Sub1: 5% test với  $1 \leq n \leq 10$
- Sub2: 5% test với thành phần  $i$  và  $i+1$  được nối trực tiếp
- Sub3: 30% test  $n \leq 1000$
- Sub4: 60% test với  $n \leq 10^5$

- **Giải thích ví dụ 1:** Có hai cách để cắt thân cây này. Đầu tiên anh ta có thể cắt kết nối (1, 2), gây ra  $1 + 3 = 4$  thiệt hại, và sau đó cắt kết nối (2, 3), gây ra thiệt hại  $2 + 3 = 5$ . Tổng thiệt hại là 9 trong này trường hợp. Nếu không, trước tiên anh ta có thể cắt (2, 3), và sau đó (1, 2). Tổng thiệt hại trong trường hợp đó  $(2 + 3) + (1 + 2) = 8$ .

Thuật toán:

Sjekira.inp	Sjekira.out
3 1 2 3 1 2 2 3	8
4 2 2 3 2 1 3 3 2 4 3	15

**Sub1:**  $n \leq 10$ , có thể thử tất cả các hoán vị của việc cắt kết nối và xác định từng giá hoán vị bằng cách sử dụng dfs hoặc cấu trúc DSU.

*Nhận xét:* nó sẽ luôn là tối ưu để cắt các kết nối xung quanh nút độ cứng lớn nhất trong cây đầu tiên. Chứng minh: trên đường đi giữa nút cứng nhất có giá trị  $M$  và nút cứng thứ hai của giá trị  $m$ , chúng ta phải cắt một số kết nối tại một số điểm và trả  $m + M$ . Nếu kết nối gần  $M$  hơn, việc cắt giảm chi phí hiệu quả hơn vì sẽ có nhiều nút hơn trong cây con có giá trị lớn nhất là  $m \leq M$ .

Vì vậy, sẽ luôn có một giải pháp tìm nút có độ cứng tối đa trong cây, cắt đứt tất cả các kết nối xung quanh nó, ta lại tìm nút cực đại trong các cây mới được hình thành, và đi xuống một cách đệ quy vào chúng. Bài toán đưa về tìm nút có độ cứng lớn nhất một cách hiệu quả.

**Sub2:** sử dụng cây phân đoạn (Segment Tree), cho phép bạn nhanh chóng tìm kiếm giá trị tối đa tại một khoảng (hoặc cây con) theo thời gian độ phức tạp  $O(n \log n)$ .

**Sub3:** ( $n \leq 1000$ ), chỉ cần tìm kiếm giá trị lớn nhất bằng cách sử dụng dfs là đủ, với độ phức tạp về thời gian  $O(n^2)$ .

**Sub4:** . Giải pháp tương đương với

$$\sum_{i=1}^n t_i - \max_{1 \leq i \leq n} t_i + \sum_{i=1}^{n-1} \max(t_{x_i}, t_{y_i}).$$

Chứng minh: phụ thuộc vào  $n$ . Trường hợp cơ sở  $n = 1$  là đúng. Khi chúng ta cắt kết nối  $a - b_j$  xung quanh mức tối đa đỉnh  $a$ , ta cộng thêm  $t_a + t_{a_j}$ , trong đó  $a_j$  là đỉnh lớn nhất trong cây con của  $b_j$ . Bây giờ thêm vào công thức cho các cây con kết thúc bước quy nạp.

### Code

```
#include <cstdio>
#include <algorithm>
#include <vector>
using namespace std;
const int MAXN = 100005;
typedef pair<int,int> ii;
```

```

int n;
int t[MAXN];
vector<ii> e;
int uf[MAXN], mv[MAXN];
bool cmp(const ii& a, const ii& b) {
    int x = max(t[a.first], t[a.second]);
    int y = max(t[b.first], t[b.second]);
    return x<y;
}
int fnd(int x){
    if(x==uf[x]) return x;
    else return uf[x]=fnd(uf[x]);
}
long long sol = 0;
void un(int x, int y){
    x = fnd(x);
    y = fnd(y);
    if(x != y){
        sol += mv[x] + mv[y];
        mv[y] = max(mv[y], mv[x]);
        uf[x] = y;
    }
}
int main(){
    scanf("%d", &n);
    for(int i=0;i<n;i++){
        scanf("%d", &t[i]);
        uf[i] = i;
        mv[i] = t[i];
    }
    for(int i=1;i<n;i++){
        int u,v;
        scanf("%d%d", &u, &v);
        e.push_back(ii(u-1, v-1));
    }
    sort(e.begin(), e.end(), cmp);

```

```

        for(auto i:e){
            un(i.first, i.second);
        }
        printf("%lld\n", sol);
    }
    void dfs(int w)
    {
        visited[w] = true;
        ancestor[w] = w; //Đặt w vào một tập không giao nhau riêng
        for(int u : adj[w])
        {
            if(!visited[u])
            {
                dfs(u);
                union_set(w, u); //hợp các đỉnh trong nhánh w
                ancestor[find_set(u)] = w; //vì trong quá trình
                hợp nhất đại diện có thể bị thay đổi
            }
        }
        ///tra loi cac truy van
        for(int u : queries[w]) //Duyệt mọi truy vấn (w,u)
        {
            if(visited[u])
            cout << "lca(" << w << ", " << u
            << ") = " << ancestor[find_set(u)] << "\n";
        }
    }
}

```

## Một số bài tập khác

### Bài 6. Chiến binh (Nguồn Thầy Hiếu)

Một phú ông giàu có tại một vùng nọ, một hôm cảm thấy chán nản vì không có gì để chơi, liền nghĩ ra một trò vô cùng hấp dẫn. Phú ông thuê  $n$  chiến binh đánh số họ từ  $1 \dots n$ . Sau đó phú ông cho các chiến binh chiến đấu với nhau trong  $m$  cuộc chiến, trong mỗi cuộc chiến phú ông sẽ chọn ra các chiến binh có chỉ số từ  $l$  đến  $r$  và cho họ chiến đấu với nhau, người thua sẽ bị loại ra. Chiến binh cuối cùng sót lại sau cuộc chiến sẽ là chiến binh thắng tất cả các chiến binh

trong cuộc chiến đó, đương nhiên một chiến binh đã bị loại trong một cuộc chiến thì không thể tham gia các cuộc chiến sau đó nữa.

Cho  $n$  chiến binh và  $m$  cuộc chiến, hãy xác định chiến binh thứ  $i$  thua chiến binh nào.

Input

- Dòng đầu tiên chứa  $n$  và  $m$  ( $2 \leq n \leq 3 \times 10^5$ ,  $1 \leq m \leq 3 \times 10^5$ )

- $m$  dòng tiếp theo mỗi dòng chứa  $l_i$ ,  $r_i$  và  $x_i$  mô tả cuộc chiến thứ  $i$ , các chiến binh được chọn có chỉ số từ  $l_i$  đến  $r_i$  và chiến binh  $x_i$  là chiến binh thắng cuộc trong cuộc chiến đó.

Output

- Gồm một dòng duy nhất chứa  $n$  số  $x_i$  với  $x_i$  là chiến binh thắng chiến binh  $i$ ,  $x_i = 0$  nếu chiến binh  $i$  chưa chết sau  $m$  cuộc chiến.

Knight.inp	Knight.out
8 4	0 8 4 6 4 8 6 1
3 5 4	
3 7 6	
2 8 8	
1 8 1	

Ví dụ

### Thuật toán

Gọi  $\text{next}[u]$  là người bên phải còn sống gần nhất. Ban đầu  $\text{next}[u]=u$ .

Viết hàm  $\text{get\_next}(u)$ : phần tử đại diện của tập chứa  $u$ .

```
while (true) //L<=R
{
    int u=get_next(l);
    if (u>r) break;
    if (u<k)
    {
        res[u]=k;
        nextt[u]=k;
    }
    else if (u>k)
    {
```

```

res[u]=k;
nextt[u]=des;
}
l=u+1;
}
} //Xuất mảng res

```

## Code

```

# include <bits/stdc++.h>
# define N 30000005
using namespace std;
int n,m;
int nextt[N],res[N];
int get_next(int x)
{
if (x==nextt[x]) return x;
else return nextt[x]=get_next(nextt[x]);
}
int main()
{
cin>>n>>m;
for (int i=1;i<=n+1;i++) {nextt[i]=i;res[i]=0;}
for (int i=1;i<=m;i++)
{
int l,r,k;
cin>>l>>r>>k;
int des=get_next(r+1);
while (true)
{
int u=get_next(l);
if (u>r) break;
if (u<k)
{
res[u]=k;

```

```

nextt[u]=k;
}
else if (u>k)
{
res[u]=k;
nextt[u]=des;
}
l=u+1;
}
}
for (int i=1;i<=n;i++) cout<<res[i]<<" ";
}

```

### Bài 7 CFARM - Closing the Farm (Nguồn USACO2016)

Nông dân John và những con bò của anh ta dự định rời thị trấn để đi nghỉ dài ngày, vì vậy FJ muốn tạm thời đóng cửa trang trại của mình để tiết kiệm tiền trong thời gian chờ đợi.

Trang trại bao gồm  $N$  chuồng trại kết nối với  $M$  đường dẫn hai chiều giữa một số cặp chuồng ( $1 \leq N, M \leq 3000$ ). Để đóng cửa trang trại, FJ có kế hoạch đóng cửa từng chuồng một. Khi một chuồng trại đóng cửa, tất cả các lối đi liền kề với chuồng trại đó cũng đóng lại và không thể sử dụng được nữa.

FJ quan tâm đến việc biết từng thời điểm (ban đầu và sau mỗi lần đóng cửa) liệu trang trại của anh ấy có liên thông hay không - nghĩa là có thể đi từ bất kỳ chuồng đang mở nào đến bất kỳ chuồng đang mở khác dọc theo một chuỗi đường thích hợp. Vì trang trại của FJ ban đầu ở trong tình trạng hư hỏng, nó thậm chí có thể không được kết nối hoàn toàn.

#### **INPUT:** Dữ liệu vào từ file **CFARM.INP**

- Dòng đầu tiên chứa  $N$  và  $M$ .
- Tiếp theo  $M$  mỗi dòng mô tả một con đường về cặp chuồng mà nó kết nối (các chuồng được đánh số thuận tiện  $1 \dots N$ ).



- N dòng cuối là một hoán vị của  $1 \dots N$  mô tả thứ tự đóng cửa các chuồng trại.

**OUTPUT:** Ghi ra file **CFARM.OUT** gồm N dòng, mỗi dòng chứa "YES" hoặc "NO". Dòng đầu tiên cho biết trang trại ban đầu đã được kết nối đầy đủ hay chưa và dòng  $i + 1$  cho biết liệu trang trại đã được liên thông sau khi nông trại thứ  $i$  đóng cửa.

### VÍ DỤ

CFARM.INP	CFARM.OUT
4 3	YES
1 2	NO
2 3	YES
3 4	YES
3	
4	
1	
2	

### Thuật toán

- Ta mô tả bài toán dưới dạng đồ thị: chuồng: đỉnh (có  $n$  đỉnh), con đường: cạnh (có  $m$  cạnh). Ban đầu trang trại được kết nối đầy đủ nếu tất cả các đỉnh thuộc thành phần liên thông
- Giải pháp đơn giản nhất là làm theo đúng quá trình. Sau khi đóng mỗi chuồng, làm lại ds kề đồ thị và đếm số TPLT. Cụ thể ta duyệt DFS bắt đầu từ chuồng mở và kết thúc khi đã thăm hết chuồng thì đồ thị liên thông. Việc làm lại đồ thị và tìm kiếm mất  $O(m+n)$ . Vì vậy tổng số lần đóng chuồng ta ĐPT:  $O(n^2 + mn)$
- Chú ý nếu  $(u, v)$  là một cạnh trong đồ thị ban đầu, thì  $u$  và  $v$  vẫn được kết nối cho đến khi  $u$  hoặc  $v$  bị loại bỏ. Do đó, chúng ta muốn một cấu trúc dữ liệu có thể theo dõi thành phần được kết nối mà một đỉnh nằm trong đó

và cũng hỗ trợ hoạt động ngắt kết nối hai đỉnh. Ta có một cấu trúc dữ liệu được gọi là disjoint-set (DSU) hỗ trợ hiệu quả hai hoạt động - theo dõi thành phần được kết nối mà một đỉnh nằm trong đó và kết nối hai đỉnh.

- Nếu chúng ta muốn sử dụng DSU, chúng ta cần phải kết nối các đỉnh với nhau, vì vậy hãy tưởng tượng quá trình đang diễn ra ngược lại. Chúng ta bắt đầu với một trạng thái trống và giới thiệu lại từng chuồng một, thêm đường từ chuồng mới đến các chuồng hiện có nếu chúng là các cạnh trong đồ thị ban đầu. Đối với mỗi con đường mà chúng ta thêm vào, hãy sử dụng thao tác tìm DSU để kiểm tra xem các chuồng trại ở các điểm cuối có nằm trong các thành phần được kết nối khác nhau hay không. Nếu vậy, hãy sử dụng thao tác hợp nhất DSU để nối hai thành phần được kết nối. Điều này cho chúng ta ĐPT  $O(m \log n)$
- Ta lưu ý: giả sử ta có  $(u,v)$  là một cạnh. Nếu đỉnh  $u$  có truy vấn thực hiện sau  $v$  thì ta cho  $u$  nối vào  $v$ . Tức là đỉnh  $v$  có đỉnh kề là  $u$  nhưng  $u$  không có đỉnh kề là  $v$ .

Ví dụ:  $\text{Order}[1]=3, \text{Order}[2]=4, \text{Order}[3]=1, \text{Order}[4]=2,$

$\text{Place}[3]=1, \text{Place}[4]=2, \text{Place}[1]=3, \text{Place}[2]=4$

cạnh  $(1,2)$  ta có:  $\text{adj}[1]=2$ , do  $\text{place}[1] < \text{place}[2]$

cạnh  $(2,3)$  ta có:  $\text{adj}[3]=2$ , do  $\text{place}[3] < \text{place}[2]$

cạnh  $(3,4)$  ta có:  $\text{adj}[3]=4$ , do  $\text{place}[3] < \text{place}[4]$

Đọc truy vấn các đỉnh theo thứ tự ngược

## Code

```
#include <bits/stdc++.h>
#define ffu(I, CUOI) for(int I=1; I<=CUOI; I++)
#define ffd(I, DAU) for(int I=DAU; I>=1; I--)
#define fou(I, DAU, CUOI) for(int I=DAU; I<=CUOI; I++)
using namespace std;
const int NMX=432000;
int n, m, parent[NMX], sz[NMX], u[NMX], v[NMX], a[NMX],
b[NMX], slt=0;
vector<vector<int>> > g;
```

```

bool kq[NMX];
void KDg(int x)
{
    parent[x]=x;
    sz[x]=1;
}
int DVy(int x)
{
    if (x==parent[x]) return x;
    return parent[x]=DVy(parent[x]);
}
void TNg(int x, int y)
{
    x=DVy(x);
    y=DVy(y);
    if (x!=y)
    {
        if (sz[x]<sz[y]) swap(x, y);
        parent[y]=x;
        sz[x]+=sz[y];
        slt--;
    }
}
int main()
{
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);
    cout.tie(NULL);
    freopen("CFARM.INP", "r", stdin);
    freopen("CFARM.OUT", "w", stdout);
    cin >> n >> m;
    ffu(i, n) KDg(i);
    ffu(i, m) cin >> u[i] >> v[i];
    ffu(i, n)
    {
        cin >> a[i];
    }
}

```

```

        b[a[i]]=i;
    }
    g.resize(n+1);
    ffu(i, m) if (b[u[i]]>b[v[i]])
g[v[i]].push_back(u[i]);
    else g[u[i]].push_back(v[i]);
    ffd(i, n)
    {
        int u=a[i];
        slt++;
        for (int j=0; j<g[u].size(); j++)
        {
            int v=g[u][j];
            TNg(u, v);
        }
        kq[i]=(slt<=1);
    }
    ffu(i, n) if (kq[i]) cout << "YES\n";
    else cout << "NO\n";
    return 0;
}

```

## Bài 8. MOOTUBE (Nguồn USACO 2018)

Trong thời gian rảnh rỗi, Farmer John đã tạo ra một dịch vụ chia sẻ video mới, anh đặt tên là MooTube. Trên MooTube, những con bò của Farmer John có thể ghi lại, chia sẻ và khám phá nhiều video thú vị. Bò của anh ấy đã đăng  $N$  video ( $1 \leq N \leq 100000$ ), được đánh số thuận tiện  $1 \dots N$ . Tuy nhiên, FJ không thể tìm ra cách giúp những con bò của mình tìm thấy những video mới mà chúng có thể thích.

FJ muốn tạo danh sách "video đề xuất" cho mọi video trên MooTube. Bằng cách này, những con bò sẽ được giới thiệu những video phù hợp nhất với những video mà chúng đã xem.

FJ đặt ra một chỉ số về "mức độ liên quan", xác định mức độ liên quan của hai video với nhau. Anh ấy chọn  $N-1$  các cặp video và tính toán mức độ liên quan theo từng cặp theo cách thủ công. Sau đó, FJ hình dung các video của mình như một mạng lưới, nơi mỗi video là một nút và  $N-1$  các cặp video mà anh ấy coi là được kết nối theo cách thủ công. Một cách thuận tiện, FJ đã chọn  $N-1$  ghép nối để có thể truy cập bất kỳ video nào từ bất kỳ video nào khác theo đường dẫn kết nối theo đúng một cách. FJ quyết định rằng mức độ liên quan của bất kỳ cặp

video nào phải được xác định là mức độ liên quan tối thiểu của bất kỳ kết nối nào dọc theo đường đi này.

Farmer John muốn chọn một giá trị  $K$  để bên cạnh bất kỳ video nào trên MooTube, tất cả các video khác có mức độ liên quan ít nhất  $K$  video đó sẽ được đề xuất. Tuy nhiên, FJ lo lắng rằng quá nhiều video sẽ được gợi ý cho những con bò của anh ấy, điều này có thể khiến chúng mất tập trung vào việc sản xuất sữa! Do đó, anh ấy muốn cẩn thận đặt một giá trị thích hợp của  $K$ . Farmer John muốn bạn giúp trả lời một số câu hỏi về các video được đề xuất cho một số giá trị nhất định của  $K$ .

**INPUT:** Dữ liệu vào từ file **MOOTUBE.INP** gồm

- Dòng đầu tiên chứa  $N$  và  $Q$  ( $1 \leq Q \leq 100000$ ).
- Tiếp theo  $N-1$  dòng mô tả một cặp video mà FJ so sánh thủ công. Mỗi dòng bao gồm ba số nguyên  $p_i$ ,  $q_i$  và  $r_i$  ( $1 \leq p_i, q_i \leq N$ ,  $1 \leq r_i \leq 10^9$ ), cho biết rằng video  $p_i$  và  $q_i$  được kết nối với mức độ liên quan  $r_i$ .
- Tiếp theo  $Q$  dòng, mô tả có  $Q$  các câu hỏi. Mỗi câu hỏi chứa hai số nguyên  $k_i$  và  $v_i$  ( $1 \leq k_i \leq 10^9$ ,  $1 \leq v_i \leq N$ ), với câu hỏi thứ  $i$ , có bao nhiêu video sẽ được gợi ý cho người xem video  $v_i$  nếu  $K=k_i$ .

**OUTPUT:** Với mỗi câu hỏi trong, ghi ra tệp **MOOTUBE.OUT** câu trả lời có bao nhiêu video liên quan đến video  $v_i$ .

MOOTUBE.INP	MOOTUBE.OUT
4 3	3
1 2 3	0
2 3 2	2
2 4 4	
1 2	
4 1	
3 1	

### GIẢI THÍCH

Farmer John nhận thấy rằng video một và hai có mức liên quan ba, video hai và ba có mức liên quan hai và video hai và bốn có mức liên quan bốn. Dựa trên điều này, video một và ba có mức liên quan tối thiểu  $(3, 2) = 2$ , video một và

bốn có mức liên quan tối thiểu  $(3, 4) = 3$  và video ba và bốn có mức liên quan tối thiểu  $(2, 4) = 2$ .

Farmer John muốn biết có bao nhiêu video sẽ được đề xuất từ video hai nếu  $K=1$ , từ video một nếu  $K=3$  và từ video một nếu  $K=4$ . Chúng tôi thấy điều đó với  $K=1$ , video 1, 3 và 4 sẽ được đề xuất trên video hai. Với  $K=4$ , không có video nào sẽ được đề xuất từ video một. Với  $K=3$ . Tuy nhiên, video 2 và 4 sẽ được gợi ý cho video một.

### Thuật toán

- Chúng ta có một cây có trọng số vô hướng. Giả sử  $f(v, w)$  là trọng số nhỏ nhất trên tất cả các cạnh trên đường đi từ  $v$  đến  $w$ . Chúng tôi muốn trả lời một số truy vấn cho một đỉnh  $v$  và  $k$  đã cho có dạng - có bao nhiêu đỉnh  $w$  tồn tại trong đó  $f(v, w) \geq k$ ? Để trả lời truy vấn này cho một đỉnh nhất định, chúng ta có thể bắt đầu bằng cách thực hiện một BFS từ  $v$ . Chúng ta không bao giờ đi qua một cạnh có trọng lượng cạnh nhỏ hơn  $k$ , vì vậy chúng ta bỏ qua các cạnh đó. Sau đó chúng ta có thể đếm xem chúng ta đã đến thăm bao nhiêu đỉnh khác.
- Chúng ta có thể đọc toàn bộ đồ thị và các truy vấn, sau đó sắp xếp các truy vấn theo thứ tự giảm dần của ngưỡng liên quan. Việc sắp xếp chúng theo thứ tự giảm dần giúp chúng ta như thế nào? Nếu chúng ta bắt đầu với một đồ thị trống và các truy vấn xử lý, chúng ta có thể sử dụng tất cả các cạnh từ truy vấn trước đó và thêm vào bất kỳ cạnh mới nào hiện có ít nhất lớn bằng ngưỡng mà chúng ta đang truy vấn. Tuy nhiên, lưu ý rằng bây giờ chúng ta chỉ đơn giản là đếm số đỉnh trong một thành phần được kết nối. Chúng ta có thể sử dụng cấu trúc dữ liệu union-find để duy trì kích thước của mọi thành phần được kết nối và hợp nhất hai thành phần bất cứ khi nào một cạnh trở nên hợp lệ.
- TT: ĐT đọc vào danh sách cạnh
- -Sắp xếp các truy vấn giảm dần theo mức độ liên quan
- Sắp xếp cạnh theo mức độ liên quan
- Duyệt các truy vấn theo thứ tự trên: Mỗi truy vấn duyệt các cạnh có mức độ liên quan  $\geq$  mức độ liên quan của truy vấn đó. Hợp nhất cây chứa 2

đỉnh của cạnh. Xuất ra số đỉnh của vùng liên thông chứa đỉnh của truy vấn cần tìm

### Code

```
# include<bits/stdc++.h>
using namespace std;
int n,q;
int root[1000005],sz[1000005];
int res[1000005];
struct data {
    int u,v;
    long long c;
}p[1000005];
struct dataz{
    long long k;
    int x,i;
}ord[1000005];
bool cmp(data x, data y)
{
    return x.c>y.c;
}
bool cmpz (dataz x, dataz y)
{
    return x.k>y.k;
}
int find_set(int x)
{
    if (x==root[x]) return x;
    else return root[x]=find_set(root[x]);
}
int main()
{
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);
    cout.tie(NULL);
    freopen("MOOTUE.INP","r",stdin);
```

```

freopen("MOOTUE.OUT","w",stdout);
cin>>n>>q;
for (int i=1;i<n;i++)
{
    cin>>p[i].u>>p[i].v>>p[i].c;
}
sort(p+1,p+n,cmp);
for (int i=1;i<=q;i++)
{
    cin>>ord[i].k>>ord[i].x;
    ord[i].i=i;
}
sort(ord+1,ord+q+1,cmpz);
for (int i=1;i<=n;i++)
{
    root[i]=i;
    res[i]=0;
    sz[i]=1;
}
int idx=1;
for (int i=1;i<=q;i++)
{
    while (idx<n&& p[idx].c>=ord[i].k)
    {
        int a=find_set(p[idx].u);
        int b=find_set(p[idx].v);
        idx++;
        if (a==b) continue;
        if (sz[a]<sz[b]) swap(a,b);
        root[b]=a;
        sz[a]+=sz[b];
    }
    res[ord[i].i]=sz[find_set(ord[i].x)]-1;
}
for (int i=1;i<=q;i++) cout<<res[i]<<endl;
}

```



### Bài 9 Favorite Colors (Nguồn USACO 2020)

Mỗi con bò trong  $N$  con bò của Nông dân John ( $1 \leq N \leq 2 \cdot 10^5$ ) có một màu yêu thích. Những con bò được dán nhãn  $1 \dots N$  và mỗi màu có thể được biểu thị bằng một số nguyên trong phạm vi  $1 \dots N$ .

Có  $M$  cặp bò  $(a,b)$  mô tả cho con bò  $b$  ngưỡng mộ con bò  $a$  ( $1 \leq M \leq 2 \cdot 10^5$ ). Có thể là  $a = b$ , trong trường hợp đó con bò tự ngưỡng mộ chính mình. Đối với một màu  $c$  bất kì, nếu có 2 con bò  $x$  và  $y$  cùng ngưỡng mộ con bò  $a$  có màu sắc yêu thích  $c$ , thì con bò  $x$  và  $y$  cũng yêu thích màu  $c$ .

Với thông tin này, hãy xác định việc gán màu sắc ưa thích của các con bò sao cho số lượng màu sắc ưa thích riêng biệt giữa tất cả các con bò là tối đa. Vì có nhiều cách gán đáp ứng yêu cầu này, hãy xuất ra đáp án nhỏ nhất về mặt từ vựng (nghĩa là bạn nên gán màu cho các con bò theo thứ tự từ  $1 \dots N$ ).

**INPUT:** Dữ liệu vào từ file **FCOLOR.INP**

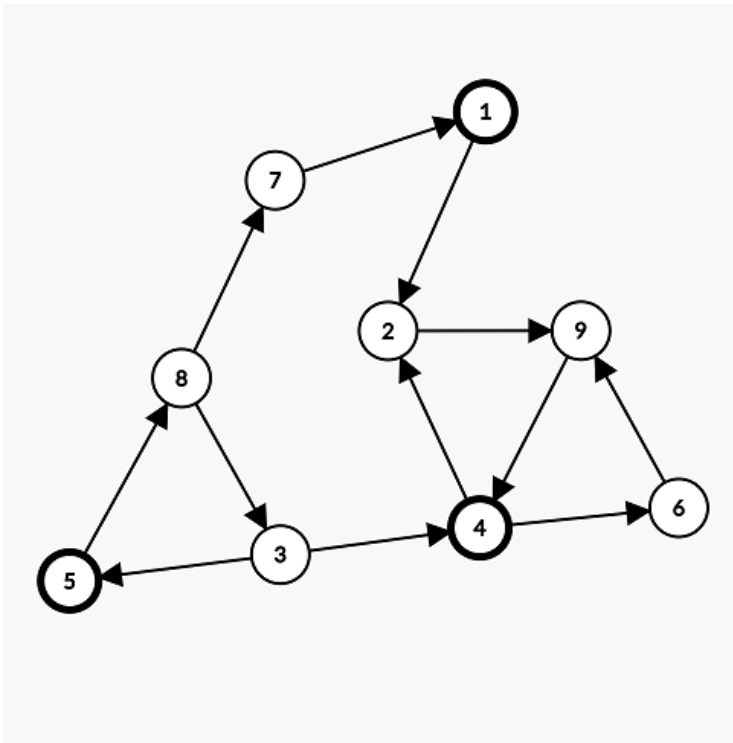
- Dòng đầu tiên chứa  $N$  và  $M$ .
- Tiếp theo  $M$  dòng, mỗi dòng chứa hai số nguyên được phân tách bằng dấu cách  $a$  và  $b$  ( $1 \leq a, b \leq N$ ), biểu thị con bò  $b$  ngưỡng mộ con bò  $a$ . Cùng một cặp có thể xuất hiện nhiều lần trong đầu vào.

**OUTPUT:**

Với mỗi con bò thứ  $i$  trong các con bò từ  $1 \dots N$ , ghi ra file **FCOLOR.OUT** màu yêu thích của con bò thứ  $i$  tương ứng.

FCOLOR.INP	FCOLOR.OUT
9 12	1
1 2	2
4 2	3
5 8	1
4 6	1
6 9	2
2 9	3
8 7	2
8 3	3
7 1	
9 4	
3 5	
3 4	

Trong hình dưới đây, các vòng tròn có đường viền in đậm đại diện cho những con bò có màu yêu thích 1.



### Sol

- Nếu cả hai con bò b và c ngưỡng mộ con bò a thì cả hai con bò b và c phải có cùng màu sắc. Từ nay, chúng ta có thể coi cả b và c như một con bò; thay đổi tất cả các lần xuất hiện của c thành b và hợp nhất danh sách liên kề của c thành danh sách của b.
- Lặp lại quá trình đó khi mà vẫn có ít nhất hai con bò khác biệt chiêm ngưỡng cùng một con. Khi chúng ta đạt được cấu hình trong đó một con bò được ngưỡng mộ bởi nhiều nhất một con bò, quá trình này sẽ kết thúc; chúng ta có thể chỉ định mỗi con bò một màu riêng biệt.
- Nếu chúng ta luôn hợp nhất danh sách kề nhỏ hơn của hai con bò thành danh sách lớn hơn thì giải pháp của chúng ta sẽ chạy trong thời gian  $O((N + M) \log N)$ .

### Code

```

#include <bits/stdc++.h>
using namespace std;

void setIO(string s) {
    ios_base::sync_with_stdio(0); cin.tie(0);
    freopen((s+".in").c_str(), "r", stdin);
    freopen((s+".out").c_str(), "w", stdout);
}

```

```

const int MX = 2e5+5;

int N,M;

int par[MX],cnt[MX];
vector<int> adj[MX], rpar[MX];
queue<int> q;

void merge(int a, int b) {
    a = par[a], b = par[b];
    if (rpar[a].size() < rpar[b].size()) swap(a,b);
    for (int t: rpar[b]) { par[t] = a;
rpar[a].push_back(t); }
    adj[a].insert(end(adj[a]),begin(adj[b]),end(adj[b]))
;
    adj[b].clear();
    if (adj[a].size() > 1) q.push(a);
}

int main() {
    setIO("fcolor");
    cin >> N >> M;
    for (int i = 0; i < M; ++i) {
        int a,b; cin >> a >> b;
        adj[a].push_back(b);
    }
    for (int i = 1; i <= N; ++i) {
        par[i] = i; rpar[i].push_back(i);
        if (adj[i].size() > 1) q.push(i);
    }
    while (q.size()) {
        int x = q.front(); if (adj[x].size() <= 1) {
q.pop(); continue; }
        int a = adj[x].back(); adj[x].pop_back();
        if (par[a] == par[adj[x].back()]) continue;
        merge(a,adj[x].back());
    }
    int co = 0;
    for (int i = 1; i <= N; ++i) {
        if (!cnt[par[i]]) cnt[par[i]] = ++co;
        cout << cnt[par[i]] << "\n";
    }
}

```

**Bài 10. Suma(Nguồn COCI 2014)**

Mirko sống trong một khu rừng lớn đầy mê hoặc, nơi cây cối rất cao và phát triển rất nhanh. Khu rừng đó có thể được đại diện bởi ma trận  $N \cdot N$  trong đó mỗi ô chứa một cây.

Mirko rất thích những cái cây trong khu rừng đầy mê hoặc. Anh đã dành nhiều năm quan sát chúng và đo từng cây đã phát triển bao nhiêu mét trong một năm. Cây cối mọc liên tục. Nói cách khác, nếu cây cao 5 mét trong một năm, nó sẽ phát triển 2,5 mét trong nửa năm.

Hôm qua, anh tự hỏi kích thước lớn nhất của nhóm cây được **kết nối** với nhau mà tất cả đều có chiều cao bằng nhau sẽ là bao nhiêu nếu cây cối tiếp tục phát triển với cùng tốc độ mà chúng đang phát triển tại thời điểm đó. Mirko nhanh chóng đo chiều cao hiện tại của tất cả các cây trong rừng và yêu cầu bạn trả lời câu hỏi của anh ấy.

Hai cây **kề nhau** nếu các ô của chúng trong ma trận có chung một cạnh.

Hai cây được **kết nối** với nhau nếu có một dãy cây liên kề dẫn từ cây thứ nhất đến cây thứ hai.

Một nhóm cây được **kết nối** nếu mọi cặp cây trong nhóm được **kết nối**

DLV: file **SUMA.INP**

- Dòng đầu tiên của đầu vào chứa số nguyên  $N$  ( $1 \leq N \leq 700$ ).
- Sau dòng đầu tiên,  $N$  dòng tiếp theo, mỗi dòng chứa  $N$  số nguyên.
- Dòng thứ  $i$  chứa các số nguyên  $h_{ij}$  ( $1 \leq h_{ij} \leq 10^6$ ), chiều cao ban đầu của cây trong hàng thứ  $i$  và cột thứ  $j$ , được cho trong mét.
- Sau đó, thêm  $N$  dòng tiếp theo với  $N$  số nguyên.
- Dòng thứ  $i$  chứa các số nguyên  $v_{ij}$  ( $1 \leq v_{ij} \leq 10^6$ ), tốc độ phát triển của cây ở hàng thứ  $i$  và cột thứ  $j$ , đã cho tính bằng mét.

Suma.inp	Suma.out
2	3
3 1	
3 3	
2 5	
2 5	

DLR: SUMA.OUT

- Dòng đầu tiên và duy nhất của đầu ra phải chứa số yêu cầu từ nhiệm vụ.

ĐIỂM

30% tổng số điểm với  $1 \leq N \leq 70$

Ví dụ

Giải thích: sau 8 tháng (hai phần ba của năm), các cây ở (0, 0), (0, 1) và (1, 0) chiều cao sẽ là 13/ 3 mét

**Sol**

Kiểm tra hai ô liền kề. Các cây trong các ô đó có thể bằng chiều cao trong nhiều nhất một thời điểm. Khoảng khắc khi hai cây liền kề các ô có cùng độ cao sẽ được gọi là sự kiện.

Số sự kiện nhỏ hơn  $4N$ . Đối với mỗi sự kiện, chúng ta có thể sử dụng DFS (hoặc BFS) để duyệt từ ô nơi cây có cùng chiều cao và đếm cây trong một thành phần, hãy cẩn thận trong một thời điểm nhất định chúng ta làm không lặp lại trên một ô nhiều hơn một lần.

Thuật toán này có độ phức tạp là  $O(N^2)$  vì chúng tôi sẽ lặp lại mọi ô tối đa 4 lần (một lần cho mỗi sự kiện bao gồm ô đó). Một thành phần cây được kết nối có thể phát triển cùng nhau (nếu độ cao ban đầu và tốc độ phát triển của chúng bằng nhau). Nếu chúng ta áp dụng thuật toán được mô tả trước, chúng ta có độ phức tạp thời gian là  $O(N^4)$ . Đạt 30% tổng số điểm.

Ý tưởng tiếp theo là nén thành phần của các cây mọc cùng nhau thành một nút và xây dựng đồ thị trong đó hai nút được kết nối nếu các thành phần tương ứng của chúng nằm liền kề trong ma trận. Cách tiếp cận này không thay đổi độ phức tạp vì có thể tồn tại một thành phần có nhiều cạnh (độ lớn  $N^2$ ) và DFS cho mọi sự kiện, chúng tôi sẽ lặp lại trên tất cả các cạnh của nó. Cách tiếp cận này, đưa ra một số tối ưu hóa bổ sung là đủ cho 50% tổng điểm.

Giải pháp cho 100% tổng số điểm dựa trên giải pháp trên, nhưng ta sẽ không sử dụng BFS hoặc DFS mà sử dụng cấu trúc DSU để xử lý sự kiện.

DSU sẽ nhớ cha của mỗi nút và gốc của mỗi tập hợp được kết nối sẽ ghi nhớ kích thước của tập hợp tương ứng của nó.

Đầu tiên, chúng ta nén tất cả các cây được kết nối phát triển như nhau thành một nút duy nhất, sau đó tính toán tất cả các sự kiện cho hai nút liền kề. Các sự kiện được xử lý để chúng ta kết nối các nút nếu thời điểm đó chúng có độ cao bằng

nhau. Trong quá trình xử lý sự kiện, chúng ta nhớ tất cả những thay đổi mà chúng ta đã thực hiện để được có thể khôi phục cấu trúc về trạng thái trước khi quá trình xử lý sự kiện bắt đầu.

Sau khi xử lý sự kiện, chúng ta khôi phục cấu trúc union-find về tình trạng ban đầu. Giải pháp này có độ phức tạp  $O(N^2 \lg N)$ .

### Code

```
#include <algorithm>
#include <cassert>
#include <cstring>
#include <iostream>
using namespace std;
#define LOG(x) cerr << #x << " = " << (x) << "\n"
typedef long long llint;
typedef pair<int,int> pii;
const int MAXN = 710;
const int dx[] = { -1, 0, 0, 1 };
const int dy[] = { 0, -1, 1, 0 };
struct event { int y1, x1, y2, x2, k, l; };
struct change {
    pii B, old_dad, A;
    int old_size;
};
bool cmp(const event &A, const event &B) {
    return (llint)A.l * B.k < (llint)B.l * A.k;
}
int n;
int l[MAXN][MAXN];
int k[MAXN][MAXN];
vector<event> events;
struct union_find {
    pii dad[MAXN][MAXN];
    int sz[MAXN][MAXN];
    vector<change> changes;
    union_find() {
        for (int i = 0; i < MAXN; ++i)
```

```

        for (int j = 0; j < MAXN; ++j)
            dad[i][j] = {-1, -1}, sz[i][j] = 1;
    }
    pii find_dad(int y, int x, bool permanent=true) {
        if (dad[y][x].first == -1) return {y, x};
        pii new_dad = find_dad(dad[y][x].first,
dad[y][x].second);
        if (permanent == false)
            changes.push_back({pii(y, x), dad[y][x], {-1, -1}, -
1});
        return dad[y][x] = new_dad;
    }
    int merge_sets(int y1, int x1, int y2, int x2, bool
permanent) {
        pii A = find_dad(y1, x1, permanent);
        pii B = find_dad(y2, x2, permanent);
        if (A == B) return sz[A.first][A.second];
        if (sz[A.first][A.second] < sz[B.first][B.second])
            swap(A, B);
        if (permanent == false)
            changes.push_back({B, dad[B.first][B.second], A,
sz[A.first][A.second]});
        sz[A.first][A.second] += sz[B.first][B.second];
        dad[B.first][B.second] = A;
        return sz[A.first][A.second];
    }

    void restore() {
        reverse(changes.begin(), changes.end());
        for (const change &c : changes) {
            dad[c.B.first][c.B.second] = c.old_dad;
            if (c.A.first != -1)
                sz[c.A.first][c.A.second] = c.old_size;
        }
        changes.clear();
    }
}

```

```

} U;

void process_neighbours(int y1, int x1, int y2, int x2) {
    if (y2 < 0 || y2 >= n || x2 < 0 || x2 >= n) return;
    int k_diff = k[y2][x2] - k[y1][x1];
    int l_diff = l[y1][x1] - l[y2][x2];
    if (k_diff == 0) {
        if (l_diff == 0) U.merge_sets(y1, x1, y2, x2, true);
        return;
    }
    if (k_diff < 0) {
        l_diff = -l_diff;
        k_diff = -k_diff;
    }
    if (l_diff < 0) return;
    events.push_back({y1, x1, y2, x2, k_diff, l_diff});
}

int main() {
    scanf("%d", &n);
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < n; ++j)
            scanf("%d", l[i]+j);
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < n; ++j)
            scanf("%d", k[i]+j);
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < n; ++j)
            for (int d = 0; d < 2; ++d)
                process_neighbours(i, j, i+dy[d], j+dx[d]);
    int ans = 0;
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < n; ++j) {
            U.find_dad(i, j);
            ans = max(ans, U.sz[i][j]);
        }
    sort(events.begin(), events.end(), cmp);
}

```



```

    for (int i = 0; i < (int)events.size(); ++i) {
        const event &e = events[i];
        ans = max(ans, U.merge_sets(e.y1, e.x1, e.y2, e.x2,
false));
        if (i + 1 == (int)events.size() || cmp(e,
events[i+1]))
            U.restore();
    }
    printf("%d\n", ans);
    return 0;
}

```

### **PHẦN III: KẾT LUẬN**

Chuyên đề disjoint-set-union – DSU tôi đã trình bày một số phép xử lý cải tiến trên cấu trúc DSU và tôi cũng đưa ra một số bài tập áp dụng DSU theo trình tự từ dễ đến khó. Đây là một phần khá khó hiểu với học sinh vì nhiều bài toán các em phải tư duy theo chiều ngược lại. Rất mong chuyên đề trên sẽ giúp ích phần nào cho các thầy cô và em học sinh trong quá trình giảng dạy và học tập.

Do kinh nghiệm chưa nhiều, nên chắc chắn có thiếu sót. Tôi rất mong đồng nghiệp và bạn bè chia sẻ, góp ý thêm cho tôi.

## **TÀI LIỆU THAM KHẢO**

1. Tập huấn GV 2019, Thầy Lê Minh Hoàng
2. Bài tập tin học chọn lọc, Thầy Trần Đỗ Hùng
3. Bài tập trên các trang USACO, COCI...
4. Một số tài liệu của các đồng nghiệp.

## MỤC LỤC

PHẦN I MỞ ĐẦU .....	1
PHẦN II NỘI DUNG.....	2
I.    LÝ THUYẾT .....	2
1. Định nghĩa .....	2
2. Các phép toán .....	2
II.   BÀI TẬP VẬN DỤNG .....	5
BÀI 1 . BIN.....	5
BÀI 2 . THAY THẾ KÝ TỰ .....	9
BÀI 3 . MOOCAST.....	11
BÀI 4 . GALAKSIJA.....	15
BÀI 5 . SJEKIRA.....	19
BÀI 6 . CHIẾN BINH.....	24
BÀI 7 . CFARM.....	27
BÀI 8 . MOOTUBE.....	32
BÀI 9 . FAVORITE.....	36
BÀI 10 . SUMA.....	39
PHẦN III. KẾT LUẬN .....	46
TÀI LIỆU THAM KHẢO .....	47



## Cows in a Row

Nông dân John (FJ) có  $N$  con bò ( $1 \leq N \leq 1000$ ) đang xếp hàng thành một được thẳng. Mỗi con bò được định dạng bằng một số tự nhiên là “mã giống”, mã định dạng của con bò thứ  $i$  trong dãy là  $B(i)$ .

FJ suy nghĩ rằng dãy các con bò này sẽ càng ấn tượng hơn nếu như có một nhóm các con bò có cùng mã giống đứng cạnh nhau. Để thực hiện được điều này, FJ quyết định loại bỏ những con bò có mã giống được ông ta chỉ định. Hãy giúp FJ tìm ra độ dài lớn nhất của dãy các con bò có cùng mã giống mà ông ta có thể xếp được bằng cách bỏ các con bò có cùng mã giống mà ông ta chỉ định.

**Dữ liệu vào:** file **cowrow.inp** gồm:

Dòng 1: Số tự nhiên  $N$

Dòng 2.. $1+N$ : Dòng thứ  $i+1$  chứa số  $B(i)$  là mã giống của  $i$  nằm trong khoảng  $0..1,000,000$ .

**Kết quả ra:** file **cowrow.out** gồm:

Dòng 1: Độ dài lớn nhất của một nhóm các con bò có cùng mã giống mà FJ có thể tạo được.

**Ví dụ:**

cowrow.inp	cowrow.out
9 2 7 3 7 7 3 7 5 7	4

## Hệ cơ số

Khánh vừa mới học phương pháp chuyển đổi giữa các hệ cơ số khác nhau. Nhưng cậu ta thường hay sai sót khi ghi lại các dãy số vừa chuyển đổi.

Mỗi khi chuyển đổi một số từ hệ cơ số này sang hệ cơ số khác và ghi lại dãy kết quả, cậu ta luôn luôn viết sai một số ở một vị trí nào đó trong dãy kết quả. Ví dụ, khi chuyển đổi số 14 từ hệ cơ số 10 sang hệ cơ số 2, kết quả đúng phải là “1110”, nhưng Khánh có thể viết thành “0110” hoặc “1111” hoặc “1100” hoặc “1010”. Khánh không bao giờ thêm hoặc bớt bất kỳ số nào trong dãy kết quả, và vì thế nếu có chữ số “0” ở vị trí đầu tiên thì vị trí đó là vị trí mà cậu ta đã viết sai.

**Yêu cầu:** Cho biết dãy kết quả khi chuyển số  $N$  (ở hệ cơ số 10) thành hệ cơ số 2 và 3 (trong đó có đúng 1 vị trí sai). Hãy xác định giá trị chính xác của  $N$ , biết rằng số  $N$  lớn nhất có thể là  $10^9$ , và có duy nhất một số  $N$  trong một test.

**Dữ liệu vào:** cho trong file **digits.inp** gồm:

Dòng 1: Chứa dãy số ở hệ cơ số 2 của  $N$ , với đúng một chữ số được viết không chính xác.

Dòng 2: Chứa dãy số ở hệ cơ số 3 của  $N$ , với đúng một chữ số được viết không chính xác.

**Kết quả:** ghi vào file **digits.out** một số nguyên duy nhất là giá trị chính xác của số  $N$ .

Ví dụ:

digits.inp	digits.out	Giải thích
1010 212	14	Chỉ có thể là cặp số: “1110” ở hệ 2 và “112” ở hệ 3 là thỏa mãn với $N=14$ .

## Tặng quà

FJ muốn tặng quà cho  $N$  con bò của mình, nhưng ông chỉ có số tiền là  $B$  đồng.

Các con bò nghe nói tặng quà nên chúng rất vui mừng, nhưng chúng lại có những yêu cầu được tặng quà. Con bò  $i$  yêu cầu một món quà với giá là  $P[i]$  và chi phí vận chuyển là  $S[i]$  đồng (vì vậy tổng giá trị là  $P[i] + S[i]$  đồng cho món quà đó). FJ chỉ có một phiếu giảm giá khi mua quà, nếu sử dụng phiếu này thì giá của món quà được mua giảm còn một nửa. Nếu sử dụng phiếu giảm giá này cho con bò  $i$ , thì số tiền phải trả là  $P[i]/2 + S[i]$ .

**Yêu cầu:** Hãy giúp FJ tính số lượng tối đa các con bò có thể nhận quà.

**Dữ liệu vào:** file **gifts.inp** gồm:

Dòng 1: Chứa hai số nguyên  $N, B$  ( $1 \leq N \leq 1000, 1 \leq B \leq 10^9$ ).

$N$  dòng tiếp theo, dòng thứ  $i$  chứa hai số nguyên  $P[i]$  và  $S[i]$  là yêu cầu món quà của con bò thứ  $i$  ( $0 \leq P[i], S[i] \leq 10^9$ ; và  $P[i]$  là một số chẵn).

**Kết quả ra:** file **gifts.out** ghi một số nguyên quy nhất là số lượng tối đa các con bò có thể nhận quà.

**Ví dụ:**

gifts.inp	gifts.out
5 24 4 2 2 0 8 1 6 3 12 5	4

$$(4+2) + (2+0) + (4+1) + (6+3) = 22$$



## 1. Hay Bales

Những con bò lại là nó! Nông dân John (FJ) đã cẩn thận sắp xếp  $N$  ( $1 \leq N \leq 10000$ ) đồng kiện hàng hóa, mỗi đồng kiện có cùng chiều cao. Tuy nhiên, khi ông ta không trông chừng, những con bò di chuyển một số kiện hàng hóa giữa các đồng kiện hàng, vì vậy chiều cao của chúng không bằng nhau nữa.

Bạn được cho chiều cao mới của các đồng kiện hàng, hãy giúp FJ tính số lượng các kiện hàng hóa nhỏ nhất mà ông ta phải di chuyển để khôi phục lại các đồng kiện hàng hóa như ban đầu, có chiều cao bằng nhau.

**Dữ liệu vào:** file **haybales.inp** gồm:

- Dòng 1: Số lượng các đồng kiện hàng hóa là  $N$  ( $1 \leq N \leq 10,000$ ).
- Dòng 2..1+N: Mỗi dòng chứa số lượng kiện hàng hóa trong một đồng (là một số nguyên trong đoạn 1..10,000).

**Kết quả ra:** file **haybales.out**:

- Dòng 1: Một số nguyên là số kiện hàng cần phải di chuyển để khôi phục lại các đồng kiện hàng có chiều cao bằng nhau.

**Ví dụ:**

haybales.inp	haybales.out
4	7
2	
10	
7	
1	

## 2. Lập kế hoạch vắt sữa bò

Farmer John có  $N$  con bò được đánh số từ 1 đến  $N$ . Mỗi con bò  $i$  sẽ mất  $T(i)$  thời gian để hoàn thành việc vắt sữa cho nó. Do cách bố trí chuồng bò, nên một số con bò phải được vắt sữa trước những con bò khác. Nếu con bò  $A$  phải được vắt trước con bò  $B$ , thì FJ phải kết thúc việc vắt sữa cho con bò  $A$  rồi mới bắt đầu với con bò  $B$ .

Để vắt sữa bò của mình càng nhanh càng tốt, FJ đã thuê một số lượng lớn các công nhân – đủ để vắt tất cả các con bò cùng một lúc. Tuy nhiên, một số con bò rất bướng bỉnh, nó sẽ không cho họ vắt sữa nếu có những con bò mà nó muốn phải được vắt trước nó chưa hoàn thành việc vắt sữa.

**Yêu cầu:** Giúp FJ tính tổng thời gian tối thiểu để hoàn thành việc vắt sữa cho  $N$  con bò của mình.

**Dữ liệu vào:** cho trong file **msched.inp** gồm:

- Dòng đầu tiên chứa hai số nguyên  $N, M$  ( $1 \leq N \leq 10,000$  ;  $1 \leq M \leq 50,000$ ) tương ứng là số lượng bò và số lượng yêu cầu của các con bò.
- $N$  dòng tiếp theo: dòng thứ  $i$  chứa số nguyên  $T(i)$  là thời gian cần thiết để vắt sữa cho con bò thứ  $i$  ( $1 \leq T(i) \leq 100,000$ ).
- $M$  dòng tiếp theo, mỗi dòng chứa hai số nguyên  $A, B$  ( $1 \leq A, B \leq N$ ) cho biết con bò  $A$  phải hoàn thành trước khi con bò  $B$  bắt đầu.

*Dữ liệu luôn luôn đảm bảo là sẽ lập được kế hoạch vắt sữa.*

**Kết quả:** ghi vào file **msched.out** một số nguyên duy nhất là thời gian tối thiểu để hoàn thành vắt sữa cho tất cả các con bò.

**Ví dụ:**

msched.inp	msched.out
3 1 10 5 6 3 2	11

### 3. Bale Share

FJ vừa mới nhận được một lô hàng mới gồm  $N$  kiện phân bón, với kiện thứ  $i$  có trọng lượng là  $S(i)$ . Ông ta muốn chia làm ba phần để bón cho ba thửa ruộng sao cho chúng cân bằng nhất có thể.

Sau khi suy nghĩ cẩn thận, FJ quyết định một cách chia cân bằng là các kiện được chia sao cho phần lớn nhất càng nhỏ càng tốt. Cụ thể, nếu  $B1, B2, B3$  là tổng trọng lượng của 3 phần sau khi chia, và  $B1 \geq B2 = B3$  thì FJ muốn  $B1$  càng nhỏ càng tốt.

Ví dụ, có 8 kiện hàng với trọng lượng là: 2 4 5 8 9 14 15 20 thì sau khi chia ta được 3 phần có tổng trọng lượng  $B1, B2, B3$  tương ứng là 26, 26, 25  $\sim (2+9+15, 4+8+14, 5+20)$ .

**Yêu cầu:** Hãy giúp FJ xác định giá trị của  $B1$  cho phép chia cân bằng các kiện hàng.

**Dữ liệu vào:** file **baleshare.inp** gồm:

- Dòng 1: Chứa số nguyên  $N$  ( $1 \leq N \leq 20$ ).
- $N$  dòng tiếp theo, dòng thứ  $i$  chứa một số nguyên  $S(i)$  là trọng lượng của kiện hàng thứ  $i$  ( $1 \leq S(i) \leq 100$ ).

**Kết quả ra:** file **baleshare.out** ghi một số nguyên duy nhất là giá trị  $B1$  trong phép chia cân bằng.

**Ví dụ:**

<b>baleshare.inp</b>	<b>baleshare.out</b>
8 14 2 5 15 8 9 20 4	26

## Chụp hình

An là một nhiếp ảnh gia chuyên nghiệp. Có một đoàn N người nhờ cô chụp ảnh giúp, tuy nhiên đoàn người này quá nhiều khiến cô không thể chụp một bức ảnh toàn cảnh được. Nên cô đề xuất chụp nhiều tấm ảnh và đảm bảo rằng mỗi người trong đoàn đều xuất hiện trong ít nhất một tấm ảnh. Để thuận tiện, cô xếp N người thành một hàng và đánh số họ từ 1 đến N. Mỗi bức ảnh chụp một loạt liên tiếp một số người đứng cạnh nhau.

Thật không may cho cô, trong đoàn có đúng K cặp người không thích nhau và họ từ chối chụp cùng một bức ảnh.

**Yêu cầu:** Cho biết vị trí của K cặp người không thích nhau. Hãy giúp An xác định số lượng tối thiểu các bức hình cần phải chụp sao cho mỗi người đều xuất hiện trong ít nhất là một bức ảnh.

**Dữ liệu vào:** cho trong file **photo.inp** gồm:

- Dòng đầu tiên chứa hai số nguyên dương N và K. ( $2 \leq N \leq 10^9$ ;  $1 \leq K \leq 1000$ ).
- K dòng tiếp theo, dòng thứ i chứa hai số nguyên  $A_i$  và  $B_i$  thể hiện cặp người đứng ở vị trí  $A_i$  và  $B_i$  không thích nhau.

Kết quả: ghi vào file photo.out một số nguyên duy nhất là số bức ảnh tối thiểu mà An phải chụp thỏa mãn yêu cầu.

Ví dụ:

photo.inp	photo.out	Giải thích	photo.inp	photo.out	Giải thích
7 3 1 3 2 4 5 6	3	1: từ 1 đến 2 2: từ 3 đến 5 3: từ 6 đến 7	10 5 1 5 2 6 3 8 4 9 7 10	3	1: từ 1 đến 4 2: từ 5 đến 9 3: từ 10 đến 10

## CHỤP ẢNH

Nhân dịp giao lưu gặp gỡ giữa các dân tộc trên toàn quốc. Ban tổ chức đã thuê một nhiếp ảnh gia chuyên nghiệp để chụp một bức ảnh lưu niệm. Vì các đại biểu là đại diện cho các dân tộc khác nhau, và mỗi dân tộc có thể có nhiều hơn một đại biểu. Ban tổ chức muốn có một bức ảnh sao cho mỗi dân tộc có ít nhất một người xuất hiện trong bức ảnh.

Có  $N$  đại biểu và tất cả họ đứng ở các vị trí khác nhau dọc theo một con đường, mỗi một dân tộc được mô tả bởi một số nguyên đại diện cho ID của dân tộc đó. Chi phí cho bức ảnh chụp là kích thước tối đa của bức ảnh. Ban tổ chức muốn chụp bức ảnh có chi phí thấp nhất mà vẫn đảm bảo yêu cầu của họ.

**Yêu cầu:** Bạn được cho biết vị trí và ID của các đại biểu, hãy giúp ban tổ chức tính toán xem chi phí tối thiểu họ phải trả là bao nhiêu để có một bức ảnh như mong muốn.

**Dữ liệu vào:** file **CHUPANH.INP** gồm

- Dòng 1: chứa số nguyên  $N$  ( $1 \leq N \leq 50\,000$ ).
- $N$  dòng tiếp theo, dòng thứ  $i$  chứa hai số nguyên  $X$  và ID tương ứng là tọa độ và ID của đại biểu thứ  $i$  ( $1 \leq X, ID \leq 10^6$ ).

**Kết quả ra:** file **CHUPANH.OUT** ghi một số nguyên duy nhất là chi phí tối thiểu phải trả cho nhiếp ảnh gia.

**Ví dụ:**

CHUPANH.INP	CHUPANH.OUT
6 25 7 26 1 15 1 22 3 20 1 30 1	4

## Wifi Setup

$N$  ( $1 \leq N \leq 2000$ ) con bò của nông dân John (FJ) đang đứng ở các vị trí khác nhau trên một đường thẳng từ chuồng đến bãi cỏ như một đường thẳng. Khi những con bò muốn dùng email để liên lạc với nhau, FJ muốn lắp trạm phát Wifi ở các vị trí khác nhau để có thể phủ sóng khắp các vị trí của các con bò.

Sau khi đi mua sắm, FJ biết rằng giá của một trạm phát Wifi phụ thuộc vào khoảng cách mà nó có thể phủ sóng: một trạm phát sóng có độ phủ sóng  $r$  có giá  $A+B*r$ , trong đó  $A$  là số tiền để thiết kế một trạm phát wifi, và  $B$  là giá để phủ sóng trên mỗi đơn vị độ dài. Nếu FJ cài đặt một thiết bị ở vị trí  $x$  thì nó có thể phủ sóng cho các con bò trong đoạn  $x-r..x+r$ . Một thiết bị Wifi có thể có độ phủ sóng  $r=0$ , nhưng nó chỉ có thể cấp phát Wifi cho con bò ở cùng vị trí lắp đặt thiết bị đó.

**Yêu cầu:** Cho trước giá trị của  $A$ ,  $B$ , và các vị trí của các con bò của FJ, hãy tính toán một chi phí rẻ nhất để FJ có thể phủ sóng Wifi cho tất cả các con bò của ông ta.

**Dữ liệu vào:** cho trong file **wifi.inp** gồm:

\*Dòng 1: Gồm 3 số tự nhiên cách nhau:  $N$   $A$   $B$  ( $0 \leq A, B \leq 1000$ ).

\*Dòng 2.. $1+N$ : Mỗi dòng chứa một số tự nhiên trong khoảng  $0..1,000,000$  thể hiện vị trí của một con bò của FJ.

**Kết quả:** ghi vào file **wifi.out** ghi ra chi phí nhỏ nhất để phát Wifi cho mọi con bò.

Ví dụ:

wifi.inp	wifi.out
3 20 5	57.5
7	
0	
100	

## TỔNG QUAN

	Tên tệp chương trình	Tên tệp INPUT	Tên tệp OUTPUT	Điểm
<b>Câu 1</b>	<b>TRIANGLE.*</b>	<b>TRIANGLE.INP</b>	<b>TRIANGLE.OUT</b>	6
<b>Câu 2</b>	<b>SECRCD.*</b>	<b>SECRCD.INP</b>	<b>SECRCD.OUT</b>	7
<b>Câu 3</b>	<b>PLAN.*</b>	<b>PLAN.INP</b>	<b>PLAN.OUT</b>	7

### Câu 1. Tam giác.

Có  $N$  cái que, mỗi cái có một trong ba màu xanh dương, xanh lá cây và màu đỏ.

**Yêu cầu:** Cho biết màu và độ dài của mỗi cái que, hỏi có bao nhiêu tam giác được tạo ra từ  $N$  cái que đã cho mà mỗi tam giác có các cạnh đủ cả ba màu?

**Dữ liệu** vào từ tệp văn bản TRIANGLE.INP gồm:

- Dòng đầu tiên chứa số nguyên dương  $N$  ( $N \geq 3$ );
- $N$  dòng tiếp theo, mỗi dòng chứa hai giá trị, một chữ cái  $c$  và một số nguyên dương  $l$  là màu và độ dài của một cái que ghi cách nhau dấu cách ( $c \in \{ 'b', 'g', 'r' \}$ ,  $b$ -xanh dương,  $g$ -xanh lá cây,  $r$ -đỏ;  $1 \leq l \leq 10^5$ ).

**Kết quả** ghi ra tệp văn bản TRIANGLE.OUT gồm một dòng ghi một số nguyên dương là số tam giác có thể tạo được.

### Ràng buộc:

- 30% số tests tương ứng với 30% số điểm của bài có:  $N \leq 100$ ;
- 20% số tests khác tương ứng với 20% số điểm của bài có:  $N \leq 1500$ ;
- 30% số tests khác tương ứng với 30% số điểm của bài có:  $N \leq 7500$ ;
- 20% số tests còn lại tương ứng với 20% số điểm của bài có:  $N \leq 10000$ .

### Ví dụ:

TRIANGLE . INP	TRIANGLE . OUT
5 r 10 g 10 b 12 r 5 g 6	3

*Giải thích:* Ba tam giác:  $\{(r\ 10), (g\ 10), (b\ 12)\}$ ;  $\{(r\ 10), (g\ 6), (b\ 12)\}$ ;  $\{(r\ 5), (g\ 10), (b\ 12)\}$ .

### Câu 2. Mật mã

Sau khai giảng năm học mới, tập thể liên khối chuyên Tin năm học 2023 – 2024 tham gia các trò chơi tập thể ngoại khóa nhằm giúp các bạn trong liên khối gắn kết với nhau, hỗ trợ nhau trong học tập và rèn luyện. Mỗi lớp tổ chức một trò chơi chung cho cả liên khối. Bạn An đại diện lớp 11 Tin tổ chức trò chơi tìm mật mã như sau: Cho một số nguyên dương  $x$ , mật mã của  $x$  chính là số lượng ước của  $x$ . Ví dụ:  $x = 8$  có bốn ước 1, 2, 4, 8 nên mật mã của  $x$  là 4.

Trong quá trình tham gia trò chơi thấy bài toán bạn An đưa ra còn đơn giản quá nên bạn Sơn mở rộng bài toán như sau: Cho  $n$  số nguyên dương  $a_1, a_2, \dots, a_n$ . Gọi  $S = a_1 \times a_2 \times \dots \times$

$a_n$  yêu cầu tìm mật mã của  $S$ . Ví dụ: với dãy số  $\{2, 8, 4\}$ ,  $S = 64$  có bảy ước  $1, 2, 4, 8, 16, 32, 64$ . Vậy mật mã của dãy là 7.

**Yêu cầu:** Cho dãy số  $a_1, a_2, \dots, a_n$ , hãy tìm mật mã của dãy số đã cho.

**Dữ liệu** vào từ tệp văn bản SECRC.D.INP gồm:

- Dòng đầu tiên chứa số nguyên dương  $n$  ( $1 \leq n \leq 10^6$ ).
- Dòng thứ hai chứa  $n$  số nguyên  $a_1, a_2, \dots, a_n$  ( $2 \leq a_i \leq 10^6$ ) các số ghi cách nhau dấu cách.

**Kết quả** ghi ra tệp văn bản SECRC.D.OUT gồm một dòng ghi một số là mật mã tìm được, tương ứng với số lượng ước của dãy khi chia lấy phần dư cho  $10^9 + 7$ .

**Ràng buộc:**

- 30% số tests tương ứng với 30% số điểm của bài có:  $S \leq 10^{12}$ ;
- 20% số tests khác tương ứng với 20% số điểm của bài có:  $n \leq 1000$  và  $a_i$  là số nguyên tố;
- 50% số tests còn lại tương ứng với 50% số điểm của bài không có ràng buộc gì thêm.

**Ví dụ:**

SECRC.D.INP	SECRC.D.OUT
3 2 8 4	7

### Câu 3. Kế hoạch.

Cô giáo chủ nhiệm cần tính toán kế hoạch chi tiêu cho  $M$  học sinh trở về trường sau một chuyến tham quan dã ngoại. Các học sinh được đánh số từ 1 đến  $M$ . Trường và các địa điểm tham quan đều nằm trên một trục đường giao thông. Hiện tại, tính từ trường thì học sinh thứ  $i$  tham quan tại địa điểm cách trường  $x_i$  kilometers. Tất cả các  $x_i$  đều là số nguyên không âm và  $x_1 \leq x_2 \leq \dots \leq x_{M-1} \leq x_M$ . Mỗi học sinh phải đi thẳng về trường, họ có thể đi bộ hoặc đi bằng xe buýt. Khi đi bộ thì học sinh thứ  $i$  cần  $v_i$  đồng để uống nước hoặc dùng cho các chi phí cần thiết khác.

Có  $N$  xe buýt, được đánh số từ 1 đến  $N$ . Xe thứ  $j$  chờ khách tại địa điểm cách trường  $y_j$  kilometers và khi được thuê để về trường cần phải trả  $c_j$  đồng. Tất cả các  $y_j$  đều là số nguyên không âm và  $y_1 \leq y_2 \leq \dots \leq y_{N-1} \leq y_N$ . Có thể có nhiều học sinh cùng lên một chuyến xe nếu họ đang ở tại địa điểm chờ khách của xe đó. Mỗi chuyến xe được thuê, dù chỉ có một người đi hay nhiều người cùng đi thì tiền thuê xe cũng không thay đổi. Mỗi xe được thuê sẽ chạy thẳng từ địa điểm chờ về trường, không dừng lại đón các học sinh khác tại bất kỳ một địa điểm nào trên đường.

**Yêu cầu:** Hãy giúp cô giáo tính toán tổng số tiền ít nhất để đưa học sinh về trường, cụ thể hơn giúp cô tính tổng số tiền ít nhất để đưa một học sinh đầu tiên, hai học sinh đầu tiên, ..., cuối cùng là  $M$  học sinh về trường.

**Dữ liệu** vào từ tệp văn bản PLAN.INP gồm:

- Dòng đầu tiên chứa số nguyên dương  $N$  ( $N \geq 1$ );
- $N$  dòng tiếp theo, dòng thứ  $j$  chứa hai số nguyên dương  $y_j, c_j$  cách nhau dấu cách;
- Dòng tiếp theo chứa số nguyên dương  $M$  ( $M \geq 1$ );



- $M$  dòng tiếp theo, dòng thứ  $i$  chứa hai số nguyên dương  $x_i, v_i$  cách nhau dấu cách.  
 $(0 \leq x_i, y_j \leq 2^{30}; x_i \leq x_{i+1}, \forall i: 1 \leq i < N; y_j \leq y_{j+1}, \forall j: 1 \leq j < M; 1 \leq v_i \leq 2^{30}, 1 \leq c_j \leq 2^{40})$

**Kết quả** ghi ra tệp văn bản PLAN.OUT gồm một dòng ghi  $M$  số nguyên dương, các số cách nhau dấu cách, số thứ  $k$  là tổng số tiền ít nhất để đưa  $k$  học sinh đầu tiên về trường.

**Ràng buộc:**

- 10% số tests tương ứng với 10% số điểm của bài có:  $N \leq 10, M \leq 6$ ;
- 20% số tests khác tương ứng với 20% số điểm của bài có:  $N \leq 14, M \leq 10^2$ ;
- 40% số tests khác tương ứng với 40% số điểm của bài có:  $N \leq 10^3, M \leq 10^2$ ;
- 30% số tests còn lại tương ứng với 30% số điểm của bài có:  $N \leq 2 \times 10^4, M \leq 10^3$ .

**Ví dụ:**

PLAN . INP	PLAN . OUT	Giải thích
6 1 3 2 10 3 100 4 100 5 15 6 10 3 2 5 4 9 8 3	8 28 44	<p>Kế hoạch đi và chi phí nhỏ nhất:</p> <ul style="list-style-type: none"> <li>• Với học sinh đầu tiên, học sinh 1 đi bộ đến địa điểm chờ của xe số hiệu 1, sau đó đi xe số hiệu 1 về trường, chi phí: <math>(2 - 1) \times 5 + 3 = 8</math>.</li> <li>• Với hai học sinh đầu tiên, học sinh 2 đi bộ đến địa điểm xe số hiệu 2, sau đó hai người cùng đi trên một chuyến xe số hiệu 2 về trường, chi phí: <math>(2 - 2) \times 5 + (4 - 2) \times 9 + 10 = 28</math>.</li> <li>• Với ba học sinh, học sinh 2 đi bộ đến địa điểm xe số hiệu 2, sau đó cả hai người đi xe số hiệu 2 về trường, chi phí 28. Còn học sinh 3 đi bộ đến địa điểm xe số hiệu 6, sau đó đi xe số hiệu 6 về trường, chi phí: <math>(8 - 6) \times 3 + 10 = 16</math>. Chi phí cả ba học sinh là: <math>28 + 16 = 44</math>.</li> </ul>

----- **HẾT** -----

TỔNG QUAN

	Tên bài	File chương trình	File dữ liệu vào	File kết quả	Điểm	Thời gian
Bài 1	Đường truyền	DUONGTRUYEN.*	DUONGTRUYEN.INP	DUONGTRUYEN.OUT	7,0	1,0s
Bài 2	Du lịch	DULICH.*	DULICH.INP	DULICH.OUT	7,0	1,0s
Bài 3	Điều kiện lỏng	DKL.*	DKL.INP	DKL.OUT	6,0	1,0s

**Bài 1 (7 điểm): Đường truyền**

Cho một mạng gồm  $n$  máy tính được kết nối với nhau bởi  $n - 1$  đường truyền 2 chiều sao cho 2 máy tính bất kỳ nào cũng có thể truyền thông tin đến nhau. Theo thời gian, các đường truyền mới sẽ được thêm vào để phục vụ nhu cầu sử dụng và đường truyền nào cũng cần 1 chi phí để duy trì. Sau đó sẽ có lúc mạng máy tính này được bảo trì và trong lúc bảo trì chỉ có một vài đường truyền quan trọng vẫn được sử dụng. Nhiệm vụ của bạn là trong thời gian bảo trì, hãy chọn ra những đường truyền để sử dụng sao cho tổng chi phí sử dụng là ít nhất và 2 máy tính bất kỳ vẫn có thể truyền thông tin cho nhau.

**Input: DUONGTRUYEN.INP**

Dòng đầu tiên chứa 2 số nguyên  $n, m$  ( $n \leq 200$ ,  $m \leq 10^5$ ) trong đó  $n$  là số máy tính trong mạng và  $m$  là số sự kiện diễn ra theo thứ tự (thêm đường truyền hoặc bảo trì).

$N - 1$  dòng tiếp theo gồm 3 số nguyên  $u, v, w$  là các đường truyền có sẵn trong mạng máy tính nối 2 máy  $u, v$  có chi phí sử dụng là  $w$  ( $w \leq 10^5$ ).

$M$  dòng cuối là các sự kiện có dạng:

- $ADD\ u\ v\ w$  là thêm đường kết nối giữa  $u, v$  có chi phí là  $w$ ;
- $FIX$  là bảo trì hệ thống, bạn cần in ra chi phí nhỏ nhất để duy trì mạng máy tính kết nối với nhau;

**Ouput: DUONGTRUYEN.OUT**

Với mỗi lần bảo trì, bạn in ra chi phí để duy trì mạng máy tính.

**Subtask:**

- 60% số test có  $m \leq 1000$ ;
- 40% số test còn lại không có ràng buộc gì thêm;

**Ví dụ:**

DUONGTRUYEN.INP	DUONGTRUYEN.OUT
4 2 1 2 4 3 4 2 1 4 4 ADD 2 4 3	9

**Bài 2 (7 điểm): Du lịch**

Max luôn mơ ước được khám phá thế giới bên ngoài biên giới thị trấn của mình và trải nghiệm những cuộc phiêu lưu ly kỳ. Anh bị cuốn hút bởi những câu chuyện về những vùng đất xa xôi và cảm giác hồi hộp trên con đường rộng mở.

Một ngày nọ, khi Max quan sát dòng xe cộ nhộn nhịp đi qua thị trấn của mình, anh bị thu hút bởi hình ảnh những chiếc mô tô phóng vút qua. Về ngoài bóng bẩy và mạnh mẽ của chúng đã khơi dậy một tia lửa trong anh. Kể từ thời điểm đó, Max biết rằng anh muốn bắt đầu một cuộc hành trình của riêng mình, không phải theo cách thông thường mà là trên chính chiếc xe của mình.

Cho  $n$  thị trấn được kết nối bởi  $m$  con đường hai chiều và 2 thị trấn bất kỳ luôn đi được đến nhau. Biết rằng thị trấn của Max là thị trấn 1 và điểm đến của cậu ấy là thị trấn  $n$ . Ban đầu, Max có được chiếc xe với dung tích bình xăng là  $x$  lít và có đầy xăng, mỗi con đường nối 2 thị trấn đều sẽ tốn 1 lượng xăng cụ thể mới có thể đi qua được và Max ban đầu chỉ có thể đổ xăng tại các trạm xăng ở trong thị trấn.

Điều đặc biệt hơn ở đây là tại mỗi thị trấn sẽ có các khuyến mại, cụ thể rằng tại thị trấn  $i$  sẽ được tặng  $a[i]$  chiếc vé đổ xăng miễn phí có thể được dùng tại tất cả các trạm xăng. Sau khi nhận được khuyến mại thì cậu có thể đổ xăng bất cứ lúc nào chứ không cần phải trong thị trấn nữa (kể cả sau khi đã dùng hết vé). Nhưng để tránh thua lỗ, trong suốt hành trình Max sẽ chỉ được nhận khuyến mại ở tối đa 1 thị trấn. Vì lý do tài chính, Max muốn tìm ra cách đổ xăng ít nhất (tất nhiên dùng vé free thì không tính) để đi từ thị trấn 1 đến  $n$ .

**Input:**

- Dòng đầu là 3 số nguyên dương  $n, m, x$  ( $n, m, x \leq 10^5$ ).
- $M$  dòng sau đó, mỗi dòng gồm 3 số nguyên  $u, v, w$  thể hiện đường đi giữa thị trấn  $u$  và  $v$  cần  $w$  lít xăng để đi qua ( $w \leq 10^6$ ).
- Dòng cuối cùng là mảng  $a$  cho biết số vé đổ xăng tại  $n$  thị trấn ( $a[i] \leq 1000, \forall i$ ).

**Output:**

Số lần đổ xăng ít nhất để đi từ thị trấn 1 đến thị trấn  $n$ .

**Subtask:**

- 30% số test có  $n, m \leq 1000$  và  $x, w, a[i] \leq 10$ ;
- 30% số test tiếp theo có  $a[i] = a[j]$  với  $\forall i, j$ ;
- 40% số test cuối cùng không có ràng buộc gì thêm;

**Ví dụ:**

DULICH.INP	DULICH.OUT
6 7 3 1 2 3 2 4 3 4 5 6 3 6 3 4 6 9	1

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

### Bài 3 (6 điểm): Điều kiện lỏng

Cho 3 số nguyên  $N, K, M$  và  $M$  bộ ba số:  $(l_1, r_1, \text{value}_1), (l_2, r_2, \text{value}_2), \dots, (l_M, r_M, \text{value}_M)$ .

Đếm số lượng dãy  $a$  khác nhau, gồm  $N$  phần tử số nguyên thỏa mãn:

- $0 \leq a_i < 2^K$  với mọi  $1 \leq i \leq N$ .
- Với mọi  $1 \leq i \leq M$ ,  $a[l_i] \mid a[l_i + 1] \mid a[l_i + 2] \mid \dots \mid a[r_i] = \text{value}_i$ , trong đó  $\mid$  là toán tử OR.

Hai dãy  $a$  và  $b$  được coi là khác nhau nếu tồn tại một vị trí  $i$  ( $1 \leq i \leq N$ ) mà  $a_i \neq b_i$ .

Kết quả của bài toán có thể rất lớn nên hãy in ra phần dư khi chia cho **998244353**.

#### INPUT

Dòng đầu tiên gồm ba số nguyên  $N, K, M$  ( $1 \leq n \leq 5 \cdot 10^5$ ,  $1 \leq K \leq 30$ ,  $0 \leq M \leq 5 \cdot 10^5$ ).

$M$  dòng tiếp theo mỗi dòng gồm 3 số nguyên, dòng thứ  $i$  là bộ 3 số  $l_i, r_i$ , và  $\text{value}_i$  ( $1 \leq l_i \leq r_i \leq n$ ,  $0 \leq \text{value}_i \leq 2^k$ ).

#### OUTPUT

Gồm một dòng duy nhất là kết quả bài toán khi lấy phần dư khi chia cho **998244353**.

#### SUBTASK

- 30% số test đảm bảo với mọi cặp chỉ số  $i, j$  ( $1 \leq i \leq j \leq M$ ),  $r_i \leq l_j$ .
- 30% số test khác có  $1 \leq N \leq 1000$ ,  $1 \leq M \leq 1000$ .
- 40% số test không có điều kiện gì thêm.

#### VÍ DỤ

DKL.INP	DKL.OUT
4 3 2 1 3 4 3 4 1	3

\_\_\_\_\_HẾT\_\_\_\_\_

## Đề số 7

### Bài toán 1: Chi phí đường đi

Một Công ty vận tải đường bộ muốn xác định chi phí nhỏ nhất để di chuyển từ một thành phố này đến một thành phố khác. Công ty có danh sách những trạm xăng trên đoạn đường di chuyển giữa hai thành phố. Danh sách bao gồm vị trí của các trạm xăng và giá bán mỗi lít xăng ở các trạm.

Để đơn giản cho cách tính chi phí, Công ty sử dụng các quy tắc về hành vi của người lái xe:

Lái xe không dừng lại trạm xăng để nạp xăng cho xe chừng nào xăng trong bình xăng của xe vẫn còn nhiều hơn một nửa dung tích của bình và vẫn đủ để đạt trạm xăng tiếp theo.

Mỗi khi dừng xe để nạp xăng, lái xe luôn nạp đầy bình xăng.

Mỗi khi dừng xe nạp xăng ở trạm xăng lái xe luôn uống nước mát 2 USD. Ở điểm xuất phát bình xăng luôn được đổ đầy.

Yêu cầu: Tính chi phí tối thiểu để lái xe mua xăng và uống nước trên dọc đường đi.

Dữ liệu vào từ file **chiphi.inp**

Thông tin về đoạn đường cần đi bao gồm các dòng sau:

Dòng 1: Chứa một số thực là khoảng cách (km) giữa điểm xuất phát và đích.

Dòng 2: Gồm 3 số thực và một số nguyên:

V - dung tích của bình xăng (lít);

C - khoảng cách xe có thể đi khi sử dụng 1 lít xăng (km);

P - chi phí đổ đầy xăng ở điểm xuất phát (USD);

N - số lượng trạm xăng trên tuyến đường ( $N < 101$ ).

Mỗi dòng thứ i trong số N dòng tiếp theo chứa hai số thực.

di - khoảng cách từ điểm xuất phát đến trạm xăng thứ i

pi - giá một lít xăng (đơn vị tính: cent = 0,01 USD),  $i = 1, 2, \dots, N$

Giả thiết rằng các trạm xăng được sắp xếp theo thứ tự tăng dần của khoảng cách từ điểm xuất phát đến chúng và các trạm xăng được phân bố ở các vị trí thích hợp để xe có thể đạt đích mà không thiếu nhiên liệu.

Kết quả ra: file **chiphi.out** tổng chi phí (có tính cả chi phí đổ xăng ở điểm xuất phát). Quy ước làm tròn chi phí mỗi lần chỉ ở trạm xăng đến cent.

Ví dụ:

**chiphi.inp**

475,6

11,9 27,4 14,98 6

102.0 99.9

220.0 132.9

256.3 147.9

275.0 102.9

277.6 112.9

381.8 100.9

**chiphi.out**

### Bài toán 2 : Lắp ráp linh kiện

Trong dây chuyền lắp ráp, một rô bốt phải lắp ráp lần lượt  $N$  linh kiện theo thứ tự từ 1 đến  $N$ . Rô bốt có  $M$  công cụ để lắp ráp. Biết  $C(i,j)$  là thời gian rô bốt lắp linh kiện  $i$  bằng công cụ  $j$  ( $1 \leq i \leq N$  ;  $1 \leq j \leq M$ ), đồng thời nếu  $i, j$  là hai công cụ được dùng trong lắp ráp hai linh kiện liên tiếp nhau thì rô bốt phải mất thêm  $D(i,j)$  thời gian ( $1 \leq i,j \leq N$ ). Hãy chỉ ra cho rô bốt lịch trình thực hiện lắp ráp các linh kiện sao cho tổng thời gian là nhỏ nhất.

Dữ liệu : Cho từ file **laprap.inp**

Dòng đầu ghi hai số  $N, M$  ( $N, M \leq 100$ )

$N$  dòng tiếp theo mô tả ma trận  $C(i,j)$ .

$M$  dòng cuối, mỗi dòng  $M$  số mô tả ma trận  $D$  (Các số cho trong input đều là các số nguyên dương  $\leq 32000$ )

Kết quả : Ghi ra file **Laprap.out**

Dòng đầu ghi  $T$  là thời gian nhỏ nhất tìm được.

Dòng hai ghi  $N$  số, trong đó số thứ  $i$  là công cụ thực hiện lắp ráp linh kiện  $i$ .

### Bài toán 3 : Đường chéo đa giác lồi

Cho một đa giác lồi  $N$  đỉnh trên mặt phẳng, các đỉnh theo thứ tự vòng quanh có tên là 1, 2, ...,  $N$ ,  $N < 20$ .

1) Hãy chia đa giác này thành  $N-2$  tam giác bởi các đường chéo không cắt nhau tại các điểm khác đỉnh sao cho tổng độ dài các đường chéo dùng để chia nhỏ nhất.

2) Hãy chia đa giác này thành  $N-2$  tam giác bởi các đường chéo không cắt nhau tại các điểm khác đỉnh sao cho đường chéo dài nhất trong các đường chéo dùng để chia có độ dài nhỏ nhất.

Dữ liệu vào file **dgcheo.inp**

Dòng đầu là số  $N$ ,

$N$  dòng tiếp theo, mỗi dòng gồm 2 số  $A_i, B_i$  với  $1 \leq i \leq N$ , đỉnh  $i$  được cho bởi tọa độ (số thực) là các đỉnh liên tiếp của một đa giác lồi.

Kết quả: Ghi ra **dgcheo.out**, dòng thứ nhất ghi giá trị tổng các đường chéo, trong  $N-2$  dòng tiếp theo ghi mỗi dòng một đường chéo bằng cách ghi tên hai đỉnh đầu mút. Sau đó là dòng ghi độ dài đường chéo dài nhất, trong  $N-2$  dòng tiếp theo ghi mỗi dòng một đường chéo theo quy cách như câu 1. Hai số liên tiếp trên một dòng cách nhau ít nhất một dấu trống. Số thực viết với 3 số lẻ sau dấu phẩy.