# **Bài 1:**

Cho dãy số nguyên . Tìm một đoạn con có đúng k phần tử liên tiếp sao cho ước chung lớn nhất của các phần tử này là lớn nhất.

***Dữ liệu vào:***

* Dòng đầu:

Dòng tiếp theo: .

***Kết quả:***

* Giá trị lớn nhất của ước chung lớn nhất của phần tử liên tiếp..

***Ví dụ:***

|  |  |
| --- | --- |
| **INPUT** | **OUTPUT** |
| **10 3**  **2 6 4 3 18 12 24 8 7 5** | **6** |

GCD(a,b) dùng thuật toán Euclide O(log(max(a,b))

Naïve: Duyệt từng đoạn dài k từ phần tử I, tìm gcd đoạn đó, so sánh với gcd max. O(kn)

Hạn chế thuật toán trên là ở đoạn k tiếp theo có cặp số đã tìm GCD ở trước nhưng không sử dụng lại dc kết quả đó.

# **Bài 2:**

Gọi là số lượng các cặp khác nhau sao cho và là nguyên tố. Hai cặp được gọi là khác nhau nếu 1 trong 2 số tham gia vào cặp này và không tham gia vào cặp còn lại. Cho số nguyên . Hãy tính giá trị biểu thức: .

**Dữ liệu vào**:

* Chứa số nguyên .

**Kết quả**:

* Gồm một số nguyên duy nhất là giá trị ..

**Ví dụ:**

|  |  |
| --- | --- |
| **INP**UT | **OUT**PUT |
| **9** | **12** |

Naive: Xây dựng sẵn 1 mảng prime từ 2 đén 2\*n để đếm. Vì không có số trùng trong dãy prime nên ta ko cần check trường hợp cặp này khác cặp kia.

Để tính f(n) ta đệ quy: f(n) = f(n-1) + g(n)

g(n) thì đếm trong mảng xây dựng sẵn

* Dùng 2 vòng for O(n^2)
* Dùng binary search O(nlogn)
* Dùng two pointer O(n)

Độ phức tạp O(nO(g))

# **Bài 3:**

Cho bảng số nguyên kích thước . Các dòng và cột của bảng được đánh thứ tự bắt đầu từ . Phần tử ở dòng , cột có giá trị . Sắp xếp các phần tử của bảng theo thứ tự tăng dần. Hãy tìm phần tử thứ .

**Dữ liệu vào:**

* Gồm một dòng chứa 3 số nguyên

**Kết quả:**

* Chứa một số nguyên duy nhất là giá trị phần tử thứ

**Ví dụ:**

|  |  |
| --- | --- |
| **INP**UT | **OUT**PUT |
| **5 3 10** | **18** |

Comment:

1. Nếu n \*m <=10^6 thì for rồi lưu vào mảng sort lại in ra phần tử thứ k O(n\*m\*log(nm)
2. Nếu n, m <= 10^6 thì chặt nhị phân giá trị phần tử thứ K (vì giá trị càng lớn thì ở vị trí càng lớn -> tính đơn điệu -> có thể chặt), với giá trị X kiểm tra vị trí của X ở thứ tự bao nhiêu, vị trí của X sẽ bằng số lượng số <= nó. Giá trị X cần tìm kiếm sẽ nằm trong đoạn 1^2 + 1^2 đến n^2 + m^2
   1. Ta cần đếm I^2 + j^2 <= X tương đương j <= sqrt(X-I^2)
   2. For I = 1 -> n sau đó cnt += sqrt(X - I^2) để
   3. Sau cùng thì cnt chính là vị trí của X
   4. Chặt tìm giá trị X nhỏ nhất thỏa mãn điều kiện là được

=> O(log2(n^2 + m^2) \* n)

# **Bài 4:**

Cho dãy số gồm số nguyên dương và một số nguyên . Người ta định nghĩa hai hàm số:

* số lượng cặp chỉ số sao cho
* .

Yêu cầu: Cho số nguyên Hãy tính .

Input

* Dòng đầu gồm số .
* Dòng tiếp theo gồm số cách nhau bởi ít nhất 1 dấu cách.

Output: Ghi ra một số nguyên duy nhất là giá trị của ..

***Ví dụ:***

|  |  |
| --- | --- |
| **INP**UT | **OUT**PUT |
| **5 6**  **1 2 3 4 5** | **6** |

Comment: for 2 vòng p[a[I] + a[j]]++ với 1 <= I < j <= n mảng p[10^6 \* 2]

Cuối cùng tính tổng trên mảng p hoy for I = 1 -> 2\*10^6 : ans += p[i]

\*\*\* Bài này có thể hiểu là tìm các cặp sao cho :

Phân tích :

* số lượng cặp chỉ số sao cho
* số lượng cặp chỉ số sao cho
* số lượng cặp chỉ số sao cho

Tổng quát hóa lại ta được số lượng cặp chỉ số sao cho:

hay .

Tính số lượng cặp thỏa yêu cầu đề bài:

* + Vét cạn: Với mỗi phần tử tìm số lượng phần tử sao cho .
  + 2 con trỏ:

Ý tưởng: Ví dụ ta cần tìm các cặp sao cho . Giả sử mảng được sắp xếp, khi đó nếu thì các cặp đều thỏa tình chất .

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Index | 0 | 1 | 2 | 3 | 4 | 5 |
| Value | 1 | 3 | 7 | 9 | 15 | 20 |

Cài đặt: Sort mảng a, với . Nếu có cặp thỏa, ++. Nếu không --. Lắp lặp tới khi .

Độ phức tạp: vét cạn , 2 con trỏ (nếu không tính chi phí sort).

# **Bài 5:**

Cho đa đồ thị có hướng đỉnh, cạnh (giữa 2 đỉnh có thể tồn tại nhiều cạnh nối) và số nguyên . Xét truy vấn, mỗi truy vấn có dạng một cặp đỉnh yêu cầu cho biết số đường đi khác nhau xuất phát từ đỉnh , kết thúc tại đỉnh và đi qua đúng cạnh. Hai đường đi được gọi là khác nhau nếu tồn tại ít nhất một cạnh thuộc đường đi này nhưng không thuộc đường đi còn lại.

**Dữ liệu vào:**

* Dòng đầu tiên: .
* Dòng thứ trong dòng tiếp theo:
* Dòng tiếp theo:
* Dòng thứ trong dòng tiếp theo:

**Kết quả:**

* Gồm dòng, mỗi dòng là câu trả lời cho từng truy vấn số đường đi khác nhau từ đến qua đúng cạnh tương ứng trong input. Kết quả khá lớn nên chỉ lấy phần dư khi chia cho

**Ví dụ:**

|  |  |
| --- | --- |
| **INPUT** | **OUT**PUT |
| **3 4 4 // n m k**  **1 2 // ui,vi**  **2 1**  **2 3**  **3 2**  **// Em nghĩ là thêm như này**  **1 // 1 truy vấn**  **1 3**  **// bao nhiêu đường đi từ 1->3?** | **2** |

- Xây dựng ma trận kề graph[I, j] += 1 nếu có cạnh (I, j) (Đa đồ thị)

Đọc lần lượt các dữ liệu vào ma trận. Truy vấn vào map<int,int>.

Text

Description automatically generated

Tính lũy thừa a^k rồi, in ra lần lượt a[u, v] là số đường đi từ u->v. Lưu ý: số lớn nên lấy phần dư khi chia cho

Độ phức tạp: O(n4) \*hic (Cái này giới thiệu code kiểu naive thôi nghe, đừng chạy vì lỗi đó)

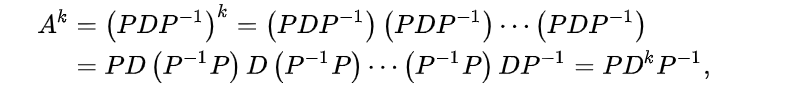
**Cải tiến:** Chéo hoá tính luỹ thừa ma trận.

<https://vi.wikipedia.org/wiki/Ma_tr%E1%BA%ADn_ch%C3%A9o_h%C3%B3a_%C4%91%C6%B0%E1%BB%A3c>

Graphical user interface, text, application, email

Description automatically generated

Tìm ma trận chuyển cơ sở P => D = P-1AP



D là ma trận đường chéo nên Ak tính dễ dàng bằng cách lấy mỗi phần tử trong đường chéo ^k

Cải tiến: Dùng python

Cài sympy: pip3 install sympy

**Complexity**: <http://man.hubwiz.com/docset/SymPy.docset/Contents/Resources/Documents/_modules/sympy/matrices/matrices.html>

Hàm diagonalize() nó gọi thêm hstack, diag gì nên cũng k biet sao. Nhưng chắc O(n)

Tổng cộng : O(n2) do tốn 2 vòng for khởi tạo graph full 0

# **Bài 6:**

Cho đoạn giá trị phủ tất cả giá trị từ đến . Hãy loại bỏ đoạn trong số đoạn sao cho đoạn còn lại cũng phủ được tất cả giá trị từ đến và đoạn loại bỏ có độ dài lớn nhất.

**Dữ liệu vào:**

* Dòng đầu tiên .
* Các dòng sau

**Kết quả:**

* Chứa một số nguyên duy nhất là : độ dài của đoạn bị loại bỏ.

|  |  |
| --- | --- |
| **INPUT** | **OUT**PUT |
| **4 1 10**  **1 5**  **3 8**  **5 8**  **8 12** | **5** |

Comment: n <= 10^3 nên có thể 2 for nè. For thử bỏ từng đoạn trong N đoạn đề cho, sau đó check xem N – 1 đoạn còn lại nó phủ được không trong O(n). Check bằng cách sort các đoạn theo X, rồi for lưu lại Y = vị trí xa nhất được phủ, nếu (x[I], y[I]) giao với (X, Y) thì merge 2 đoạn này lại, nếu không giao thì tạch, trong lúc merge cũng check xem (X, Y) có phủ được (L, R) chưa.

Cách cộng dồn mảng:

Mảng C[MAX\_V], D[MAX\_V], E[MAX\_V]

Lặp I từ 1 đến n:

Gọi (left, right) là đoạn I

C[left]++;

C[right+1]--;

Mảng D là mảng cộng dồn của mảng C --> D[I] là số đoạn phủ lên vị trí I

Tới đây, xác đinh đoạn nào có thể bị xóa nữa là xong

Gọi đoạn [left, right] có thể bị xóa --> Không có D[t] nào bằng 1 với t thuộc [left, right]

Nếu ở đây đoạn nào cũng lặp t [left, right] thì quá chậm

Mảng E được xử lý 2 lần:

Lần 1: E[I]=0 nếu d[I]=1 và e[I]=1 nếu d[I]>0

Lần 2: cộng dồn mảng E lên

Lặp I từ 1 đến n:

Gọi (left, right) là đoạn thứ I

Nếu right<L || left>R || E[min(right, r)]-E[max(left, l)-1]==min(right, r)-max(left, l)+1:

Đoạn (left, right) có thể xóa được

ans=max(ans, right-left)

Sau lần xử lý 2, E[b]-E[a-1]==b-a+1 nghĩa là trong đoạn [a, b] không có chỗ t nào là E[t]!=1 hết--> Trong đoạn [a, b] không có chỗ t nào mà D[t]==1

Việc lấy min(right, r) và max(left, l) là vì ta chỉ quan tâm đến đoạn [l, r] mà đề bài yêu cầu phủ kín nên chỉ cần tìm trong phần giao giữa đoạn [left, right] ta đang xét và đoạn [l, r] đề cho xem có tồn tại vị trí t nào mà D[t]==1 không.

Ví dụ: đề bài yêu cầu phải phủ kín đoạn [6, 8], để xét coi có bỏ được đoạn [1, 7] hay không ta phải xét tất cả ô trong phần giao của 2 tụi nó là đoạn [6,7] coi có tồn tại t để D[t]==1 không

Ví dụ:

1 2 3 4 5 6 7 8 9 10 11 12 13

C 1 0 1 0 1 -1 0 1 -2 0 0 0 -1

D 1 1 2 2 3 2 2 3 1 1 1 1 0

E(1) 0 0 1 1 1 1 1 1 0 0 0 0 0

E(2) 0 0 1 2 3 4 5 6 6 6 6 6 6

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
| C | 1 | 0 | 1 | 0 | 1 | -1 | 0 | 1 | -2 | 0 | 0 | 0 | -1 |
| D | 1 | 1 | 2 | 2 | 3 | 2 | 2 | 3 | 1 | 1 | 1 | 1 | 0 |
| E(1) | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| E(2) | 0 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 6 | 6 | 6 | 6 | 6 |

Xét đoạn (1, 5):

E[5]-E[0]=3 != (5-1+1) à không xóa được

Xét đoạn (3, 8):

E[8]-E[2]=6 == (8-3+1) à Xóa được à ans=5

Xét đoạn (5, 8):

E[8]-E[4]=4 == (8-5+1) à Xóa được

Xét đoạn (8, 12):

E[10]-E[7]=6-5+1 != (10-7+1) à Không xóa được

# **Bài 7:**

Cho 2 dãy số và là các hoán vị của các số từ đến . Thực hiện nối các cặp giá trị trên 2 dãy số thỏa sao cho các cặp nối không được chéo nhau, nghĩa là với 2 cặp và nếu thì (hình minh họa)

2

3

1

5

6

4

3

2

5

6

1

4

**Yêu cầu:** Tìm cách nối thỏa điều kiện để số cặp nối với nhau là nhiều nhất.

**Dữ liệu vào:**

* Dòng 1:
* Dòng 2:
* Dòng 3:

**Kết quả:**

* Chứa một số nguyên duy nhất là số cặp nối được nhiều nhất.

**Ví dụ:**

|  |  |
| --- | --- |
| **INP**UT | **OUT**PUT |
| **6**  **2 3 1 5 6 4**  **3 2 5 6 1 4** | **4** |

Giải: dùng phương pháp LCS

gọi F[i][j] = độ dài dãy con chung dài nhất xét khi xét i phần tử đầu tiên của dãy a và j phần tử đầu tiên của dãy b

A screenshot of a computer

Description automatically generated with low confidence

# **Bài 8:**

Cho dãy số nguyên và số nguyên . Dãy số được thu gọn về còn phần tử bằng cách thực hiện thao tác sau:

* Loại 2 phần tử nằm cạnh nhau ra khỏi dãy và đặt vào chỗ trống phần tử mới có giá trị .
* Chi phí của thao tác là .

**Yêu cầu:** Tính chi phí nhỏ nhất sau khi thực hiện tất cả thao tác.

**Dữ liệu vào:**

* Dòng đầu tiên:
* Dòng tiếp theo:

**Kết quả:**

* Gồm một số nguyên duy nhất là chi phí thu gọn dãy nhỏ nhất.

**Ví dụ:**

|  |  |
| --- | --- |
| **INPUT** | **OUTPUT** |
| **3 100**  **40 60 20** | **2400** |

Comment: đây là bài toán quy hoạch động. ([SPOJ.com - Problem MIXTURES)](https://vn.spoj.com/problems/MIXTURES/)

Gọi f(I, j) = chi phí nhỏ nhất để thao tác trên đoạn [I..j]

For I for j for k mình sẽ merge đoạn f(I, k) với f(k + 1, j) vào f(I, j)

F(I, j) = (sum(I, k) \* sum(k + 1,j) % M + f(I, k) + f(k + 1, j)

Vì khi merge được đoạn (I, k) rồi thì giá trị của phần tử sau khi merge chính là sum(I, k)

Do chỉ có thể thao tác với các phần tử nằm cạnh nhau, nên ta giả sử với đoạn [i,j]

Sau khi thực hiện j-i-1 thao tác ta còn mảng 2 phần tử là sum(I,k)%M và sum(k+1,j)%M

Tiếp tục merge 2 phần tử này ta được mảng 1 phần tử sum(I,j)%M với chi phí sum(I,k)\* sum(k+1,j)

Graphical user interface, application, table, Excel

Description automatically generated

Gọi dp[I,j] là chi phí nhỏ nhất để thu gọn mảng từ i->j

Suy ra

Dp[I,j] = min(dp[I,j], dp[i, k] + dp[k + 1, j] + sum(i, k) \* sum(k + 1, j)) với I <= k <= j;

Kết quả bài toán là dp[0,n-1]

# **Bài 9:**

Một số nguyên được gọi là nguyên tố bậc 3 nếu tất cả các số tạo thành từ 3 chữ số liên tiếp tính từ trái qua phải đều là số nguyên tố. Ví dụ số là số nguyên tố bậc 3 vì các số và đều là các số nguyên tố.

Yêu cầu: Cho số nguyên . Đếm số lượng số nguyên tố bậc 3 có đúng chữ số.

Input: Số nguyên dương

Output: Số lượng số nguyên tố bậc 3 có đúng chữ số. Kết quả khá lớn nên chỉ in ra phần dư khi chia cho .

|  |  |
| --- | --- |
| **INP**UT | **OUT**PUT |
| **4** | **204** |

Comment: quy hoạch động (dùng đệ quy có nhớ cho dễ)

F(I, value) là số lượng số khi xét đến chữ số thứ I và 3 số cuối có giá trị là value

Chuyển trạng thái:

f(I, value) -> f(I + 1, (value % 100) \* 10 + x) trong đó value % 100 là lấy 2 chữ số cuối sau đó chèn chữ số x = [0..9] vào

int F(int I, int value) {

if dp[I][value] đã được tính thì return dp[I][value]

ret = 0;

for (int x = 0; x <= 9; ++x)

ret += F(I + 1, (value % 100) \* 10 + x);

Dp[I][value] = ret;

Return ret;

}

**Cách đệ quy có nhớ:**

F(l, value): số lượng số nguyên tố bậc 3 có độ dài l và 2 số đầu là value

F(l, value) =

* Nếu value<10, 0
* Nếu l==3, tất cả các số nguyên tố <1000 và >=100 có 2 số đầu là value
* Ngược lại:

Lặp t từ 0->9:

P=value\*10+t

Nếu p là số nguyên tố, ans=ans+F(l-1, p%100)

F(l, value)=ans

Để tìm số nguyên tố bậc ba có độ dài length, và bắt đầu bởi 2 số là b

VD: tìm số nguyên tố bậc 3 dài 4, bắt đầu là 13, F(4, 13)

Lặp từ 0->9

P=130, 131, 132, …

Kiểm tra nếu P nguyên tố thì tìm số nguyên tố bậc 3 độ dài length-1 và bắt đầu bởi 2 chữ số cuối của P (lấy P%100)

VD thấy 131 là nguyên tố thì tìm F(3, 31)

# **Bài 10:**

Cho đoạn giá trị trên trục số. Các đoạn được tô màu theo quy tắc: 2 đoạn có phần giao nhau phải được tô bằng 2 màu màu khác nhau, ngược lại có thể tô cùng màu nhau.

**Yêu cầu:** Tính số màu ít nhất cần sử dụng.

**Dữ liệu vào**

* Dòng đầu tiên:
* Dòng thứ trong dòng tiếp theo:

**Kết quả:**

* Dòng đầu tiên chứa số màu ít nhất cần sử dụng.
* Dòng thứ hai thể hiện kết quả một cách tô theo màu.

|  |  |
| --- | --- |
| INPUT | OUTPUT |
| **3**  **1 2**  **2 4**  **4 4** | **2**  **1 2 1** |

Để ví dụ minh họa cho bài toán em một cách rõ hơn, em sẽ dùng các input sau:

4

2 5

1 3

4 7

5 9

A picture containing text, sky, whiteboard, flock

Description automatically generated

Dòng Color phía dưới là thể hiện số lượng các đoạn giao nhau tại các điểm tương ứng. Có thể thấy tại 5 có giá trị 3 là lớn nhất và đó cũng chính là số màu ít nhất cần sử dụng cho bài toán của chúng ta.

Vậy từ bài toán tìm số màu ít nhất cần sử dụng ta chuyển về bài toàn tìm số đoạn giao nhau nhiều nhất từ tất cả các điểm.

1. Đầu tiên cần tìm số đoạn giao nhau nhiều nhất

* Cách tiếp cận 1: Nhìn nhanh thì với từng đoạn input [a,b] thì ta cho từng điểm trong đó thành 1 hết, xong rồi cộng dồn tất cả từng vị trí đó trong các đoạn.

Table

Description automatically generated

Tiếp theo chọn ra max trong Tổng đó, nhìn ở trên thì là 3 tại vị trí 5 -> đó cũng chính là số màu ít nhất cần tô.

Nhưng nếu làm theo cách này thì ta cần tới 2 vòng lặp, 1 vòng lặp để duyệt từng đoạn và 1 vòng lặp duyệt từng vị trí để kiểm tra +1 vào. Sau đó tìm max trong danh sách màu các điểm đó -> độ phức tạp thời gian: O(n^2) + O(n) = O(n^2)

* Cách tiếp cận 2: giảm độ phức tạp thời gian cho cách tiếp cận 1:

+ Thay vì đánh 1 ở tất cả vị trí trong [a,b], ta chỉ cần đánh 1 vào a và -1 vào b + 1

+ Tiếp theo, ta dùng 1 vòng lặp để tính lại giá trị từng điểm bằng cách lấy giá trị tại điểm đó cộng với giá trị điểm trước nó -> nó sẽ quay lại kết quả ở cách tiếp cận 1

A screenshot of a computer

Description automatically generated with medium confidence

Với cách này ta chỉ cần 1 vòng for để duyệt các đoạn để đánh dấu a = 1 và b + 1 = -1 và sau đó tìm max -> độ phức tạp giảm xuống O(n) + O(n) = O(n)

1. Tiếp theo in màu sử dụng cho từng đoạn trong input theo thứ tự nhập

* Trước khi làm cần phải sort lại mảng segment trước theo điều kiện sau: đưa các đoạn có phần đầu a nhỏ hơn về phía trước (lý do cần sort sẽ đề cập sau)
* Xây dựng 1 priority queue theo điều kiện sau, các phần tử có dạng (vị trí cuối của đoạn đưa vào, mã màu)

(x.a < y.a) || (x.a == y.a && x.b < y.b)

* Cách làm là ta pop lần lượt trong PQ ra, sau đó cập nhật mã màu đó cho đoạn đầu tiên chưa được, sau đó push vào lại PQ value là (vị trí cuối đoạn vừa được tô màu đó, mã màu). Làm lần lượt như vậy đến khi duyệt hết các đoạn. Ta được kết quả cần tìm

Table

Description automatically generated

Giải thích lý do sort: ví dụ thứ tự input lần lượt là [4,5] [6,8] [3,7]. Nếu ta không sort mà làm như cách trên thì [4,5] có mã màu 1, [6,8] có mã màu 2. Lúc này tới đoạn [3,7] sẽ được đánh mã màu 1 nhưng thực tế thì sai vì [3,7] có giao với [4,5] mà cùng mã màu 1. Vậy nên cần sort các đoạn trước theo phần tử đầu a nhỏ hơn về phía trước để tránh tình trạng này

Đánh giá:

* Độ phức tạp về mặt thời gian: O(nlogn)
* Độ phức tạp về mặt không gian: O(n)

# **Bài 11:**

Xét cụm dữ liệu, mỗi cụm dữ liệu được chia thành 2 nhóm: nhóm loại 1 và nhóm loại 2. Cụm dữ liệu thứ có trọng số các nhóm tương ứng . Cần chọn ra nhóm loại 1 và nhóm loại 2 trong các cụm dữ liệu để làm mẫu thử với nguyên tắc mỗi cụm dữ liệu chỉ chọn 1 nhóm, do đó .

**Yêu cầu:** Tìm cách chọn các nhóm dữ liệu trong mỗi cụm sao cho tổng trọng số của tất cả các nhóm được chọn là lớn nhất.

**Dữ liệu vào:**

* Dòng đầu tiên:
* Dòng thứ trong dòng tiếp theo:

**Kết quả:**

* Gồm một số nguyên duy nhất là tổng trọng số lớn nhất.

|  |  |
| --- | --- |
| **INPUT** | **OUTPUT** |
| **4 2 1**  **4 9**  **3 5**  **7 2**  **5 5** | **21** |

Cách 1: Dùng backtracking – cách ngây thơ

Cơ chế backtrack sẽ duyệt qua tất cả các trường hợp khả thi, nếu thành công thì lưu vết lại để tính ra kết quả, nếu không thì đánh dấu bỏ trường hợp đó và tiếp tục xét đến cách khác (về mục đích thì giống như vét cạn).

Mỗi lần duyệt qua thì tính giá trị total mới, và tìm max của ans và total đó để cập nhật max mới cho ans. Sau khi backtrack xong hết thì ans chính là kết quả cuối cùng

Đánh giá:

* Độ phức tạp về mặt thời gian: O(k^(m+n) \* (m+n)^n)
* Độ phức tạp về mặt không gian: O(n)

Cách 2: Ta xây dựng 2 cluster\_a và cluster\_b là 1 mảng các dòng từ input vào, sau đó sort lại tương ứng cho từng cluster. Ở mỗi cluster sẽ có cách thức sort khác nhau. Quy luật sort như sau: Giả sử ta xét cluster\_a đại diện cho nhóm 1, thì ta cần sắp xếp các đoạn (a,b) theo cách sau:

+ Đưa các dòng có a lớn hơn về trước

+ Nếu giá trị a ở 2 đoạn bằng nhau, ta cân nhắc chọn dòng nào có b nhỏ hơn. Vì khi đó b đó có thể dùng lại bên cluster\_b

Tương tự cho TH cluster\_b

* Dùng 2 con trỏ i,j để duyệt và so sánh cluster\_a[i].a và cluster\_b[j].b, nếu:

+ n=0: tức bên nhóm 1 đã đầy, ta chỉ cần thêm liên tục bên nhóm 2 cho tới khi m về 0

+ m=0: tức bên nhóm 1 đã đầy, ta chỉ cần thêm liên tục bên nhóm 2 cho tới khi n về 0

+ Phía cluster\_a: cluster\_a[i].a > cluster\_b[j].b hoặc cluster\_a[i].a = cluster\_b[j].b && cluster\_a[i].b < cluster\_b[j].a, ta thêm bên nhóm 1 giá trị a đó và bỏ đi đánh dấu check tại dòng đó ở cả 2 cluster sau này không dùng lại

+ Ngược lại phía cluster\_b

Lấy ví dụ input:

6 3 2

4 9

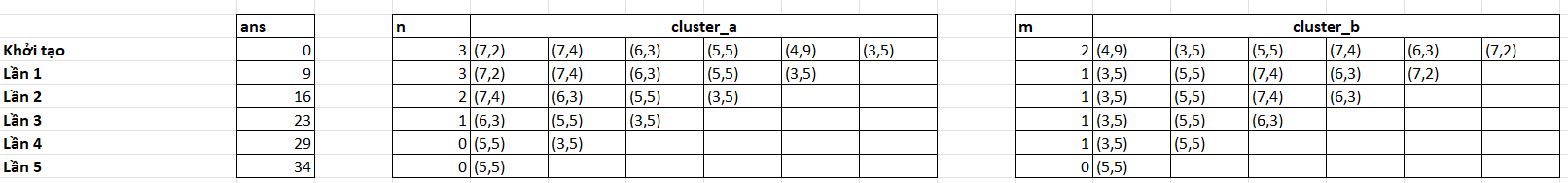
3 5

7 2

5 5

7 4

6 3



Đánh giá:

* Độ phức tạp về mặt thời gian: O(nlogn)
* Độ phức tạp về mặt không gian: O(n)

Quy hoạch động

Thuật simple:

f(I, j, k) là xét tới cụm thứ I đã chọn j thằng vô nhóm 1 và k thằng vô nhóm 2, vậy công thức là (này cài theo cách for không đệ quy)

* + - * Không chọn thằng I = f(I – 1, j, k)
      * Chọn vô nhóm 1 = f(I – 1, j – 1, k) + a[I]
      * Chọn vô nhóm 2 = f(I – 1, j, k – 1) + b[I]

Lấy max 2 cách.

Tối ưu: vì n <= 10^3 nên không thể chứa mảng 3 chiều dc,ý tưởng giảm xuống 2, và nhận thấy rừng mình sẽ chọn max bên a hoặc max bên b để lợi, nên sort giảm dần theo a hay b gì đó.

# **Bài 12:**

Hãy chỉ ra cách xây dựng một hình vuông có diện tích là số nguyên dương cho trước với các tọa độ nguyên trong phạm vi từ đến .

Input: Số nguyên dương .

Output: Tọa độ 4 đỉnh của hình vuông được liệt kê cùng hoặc ngược chiều kim đồng hồ. Trường hợp không có lời giải thì thông báo Impossible.

|  |  |
| --- | --- |
| **INPUT** | **OUTPUT** |
| **5** | **1 2**  **2 4**  **3 1**  **4 3** |
| **3** | **Impossible** |

Bài toán đưa về kiểm tra số nguyên thể hiện thanh tổng 2 bình phương của a, b

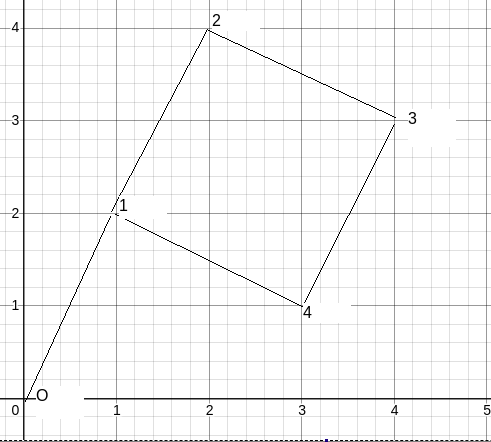
và tạo ra tọa độ hình vuông từ 2 số a, b

Output khi possible chỉ cần in ra 1 trường hợp khả thi

// Để kiểm tra 1 số có là tổng 2 bình phương dùng Fermat Theorem

Tuy nhiên ta cần biết a và b nên có các cách sau:

* Duyệt 2 vòng for từ 0 đến n để tìm O(n^2)
* Dùng 2 con trỏ O(n)
* Duyệt 1 vòng for từ 0 đến n, mỗi lần duyệt thì thêm bình phương vào set, và tìm n^2-i^2 trong set O(sqrt(n))

Với bài toán 2 ta chọn 1 điẻm làm gốc, và đi theo hướng (i, j), (i, j), (j, -1\*i), (-1\*i, -1\*j) 

Cải tiến để hình vuông nằm đều cả 4 góc phần tư, chọn gốc là điểm có tọa độ = - (a+b)/2

# **Bài 13:**

Trong mặt phẳng cho điểm có tọa độ nguyên sao cho không có 2 điểm nào trùng nhau và không có 3 điểm nào thẳng hàng ( là một số chẵn).

**Yêu cầu:** Hãy chọn ra 2 điểm trong số điểm để đường thẳng qua 2 điểm này chia tập điểm thành 2 phần có số lượng điểm bằng nhau.

**Dữ liệu vào:**

* Dòng đầu tiên:
* Dòng thứ trong dòng tiếp theo: – tọa độ của điểm thứ .

**Kết quả:**

* Gồm 2 số là số thứ tự của 2 điểm được chọn.

**Ví dụ:**

|  |  |
| --- | --- |
| **INPUT** | **OUTPUT** |
| **6**  **3 5**  **1 3**  **3 1**  **6 1**  **8 3**  **6 5** | **2 5** |

Tương tự bài dưới nhưng thay vì tính diện tích, ta đếm số lượng điểm nằm một bên (để đếm điểm nằm một bên thì có thể dùng phương trình đường thẳng, hoặc gọn lẹ thì tích có hướng)

1. naive() O(n^3)

3 vòng for i,j,k

xét điểm k nằm trái / phải so với đường thẳng nối 2 điểm i,j => đếm, nếu = return

1. better O(n^2)

chọn 1 điểm bất kỳ trong tập (chọn điểm giữa) gọi là điểm i

2 vòng for j,k

xét điểm k nằm trái / phải so với đường thẳng nối 2 điểm i,j => đếm, nếu = return

1. best O(n)

chọn điểm có hoành độ min gọi là O

sort theo trọng số góc (tưởng tượng giống theo chiều kim đồng hồ) => sau khi sort chọn điểm chính giữa I

kq bài toán là điểm O và I

Chart, scatter chart

Description automatically generated

# **Bài 14:**

Cho đa giác lồi đỉnh có tọa độ nguyên . Các đỉnh được liệt kê ngược chiều kim đồng hồ và không có 3 đỉnh nào thẳng hàng. Chỉ ra một cách kẻ 1 đường chéo đi qua 2 đỉnh và chia đa giác thành 2 phần có diện tích chênh lệch nhau ít nhất.

**Dữ liệu vào**

* Dòng đầu tiên:
* Dòng thứ trong dòng sau: .

**Kết quả:**

* Gồm 2 số nguyên là số thứ tự 2 đỉnh được chọn..

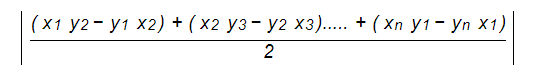
**Ví dụ:**

|  |  |
| --- | --- |
| **INPUT** | **OUTPUT** |
| **4**  **0 2**  **0 0**  **2 0**  **3 3** | **2 4** |

Simple thì n^3 for chọn 2 đỉnh của đoạn thẳng sau đó tính diện tích hai bên

Giảm đpt xuống thì  
Lần lượt cố định điểm bắt đầu, sau đó for tăng dần j đi theo vòng tròn (vì đây là đa giác lồi nên khi điểm đi càng xa thì chênh lệch diện tích sẽ càng giảm đến một mức nào đó thì nó sẽ tăng dần). -> Chặt nhị phân (hoặc tam phân) tìm ra cái điểm ở đỉnh đó. DPT O(n^2 log(n))

Ý tưởng đơn giản: xét từng cặp đỉnh, rồi tính chênh lệch 2 đa giác tạo ra, cái nào nhỏ nhất thì trả về cặp đỉnh đó

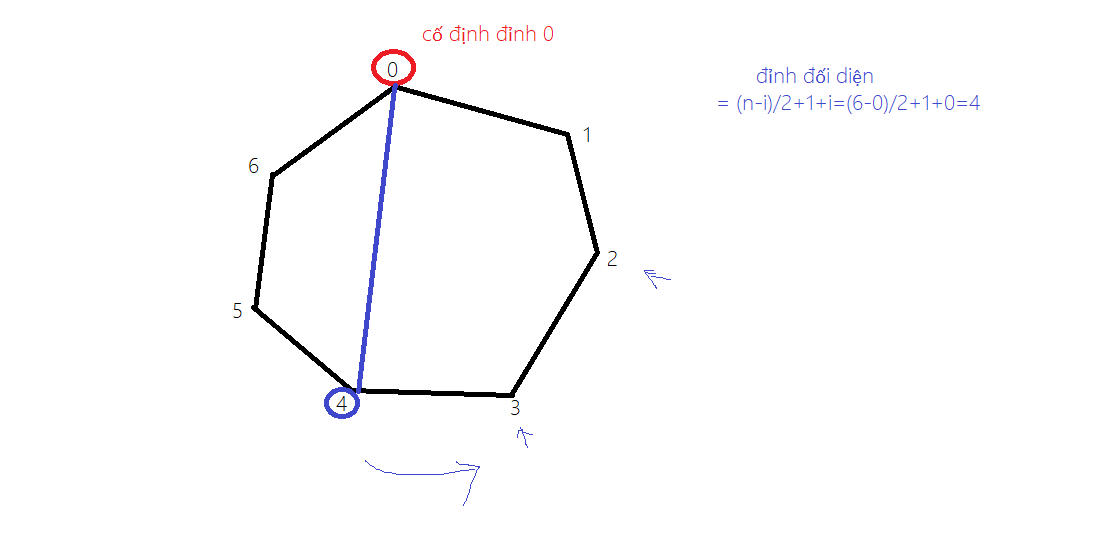
công thức tính diện tích đa giác lồi theo tọa độ đỉnh là:  


Độ phức tạp:

duyệt cặp cạnh: n^2 / 2 vòng lặp ~ O(n^2)

tính diện tích : O(n)

* O(n^3)



***Ý TƯỞNG:***  
Ban đầu gán số chênh lệch 2 đa giác min= VÔ CỰC

Xét lần lượt từng đỉnh từ (0-> đỉnh thứ n)

Tính đỉnh đối diện nó bằng công thức = (n-I)/2+1+I

Tính độ chênh lệch 2 đa giác mới tạo ra (diff)

Nếu diff<min, cập nhật lại min, lưu 2 đỉnh

Mở rộng đỉnh đối diện qua bên phải, tính độ chênh lệch diff 2 đa giác mới tạo ra và so sánh với min

Nếu diff<min, cập nhật min, lưu 2 đỉnh, mở rộng tiếp qua bên phải

Nếu diff>min, dừng hướng này

Mở rộng đỉnh đối diện qua bên trái thực hiện tương tự

Cuối cùng ta tìm được độ chênh lệch min và 2 đỉnh đã lưu

***TỐI ƯU***:

Ý tưởng là khi 1 cặp đỉnh cắt tạo ra 2 đa giác mới, bên nào diện tích lớn hơn thì mở rộng thì phía đó. Cái này hay nên t nghĩ sẽ chừa lại để có gì thầy hỏi mình có cái nói .

***ĐỘ PHỨC TẠP:***

Duyệt lần lượt cố định từng đỉnh: O(n)

Mỗi lần duyệt mở rộng ra 1 phía: trường hợp xấu nhất là duyệt tới n/2 lần

Mỗi lần duyệt tính độ chênh lệch diện tích: 0(2n)~O(n)

=O(n^2 \* n/2)

# **Bài 15:**

Trên mặt phẳng cho 2 hình tròn có cùng bán kính và tọa độ tâm tương ứng và . Tính diện tích phần hợp của 2 hình tròn trên mặt phẳng. Tọa độ tâm và bán kính hình tròn đều là các số nguyên.

**Dữ liệu vào:**

* Gồm các số nguyên

**Kết quả:**

* Chứa một số thực duy nhất là diện tích phần hợp lấy chính xác 3 chữ số thập phân.

**Ví dụ:**

|  |  |
| --- | --- |
| **INPUT** | **OUTPUT** |
| **1 2 3 4 2** | **22.850** |

<https://vfo.vn/r/bai-toan-tinh-dien-tinh-mat-phang-giao-nhau-cua-2-duong-tron.56388/>

S phần hợp: tổng S 2 đường tròn – phần giao

R bằng nhau => hình thoi => 2 đường chéo vuông

Chart, radar chart

Description automatically generated

Diagram

Description automatically generatedText, letter

Description automatically generatedA picture containing pie chart

Description automatically generated



Giải thích: Cách làm tổng quát như trên. Áp dụng bài toán:

+ Tam giác vuông (hai đường chéo vuông góc) nên tính dễ dàng = (gọi là )

* cung tròn BCD – S tam giác BCD => S phần tô màu vàng

Tương tự tính được S phần tô màu đỏ

(tính theo công thức S\_chung với R1=R2,

**Tối ưu**: tính toán chính xác cho số thực. Nhất là khi sử dụng sqrt

Độ phức tạp: O(logn) với n\_max = 100^2 + 100^2

# **Bài 16:**

Cho một cây đỉnh. Khoảng cách giữa 2 đỉnh là số cạnh trên đường đi từ đến . Độ rộng của cây là tổng khoảng cách giữa tất cả cặp đỉnh . Hãy xác định độ rộng của cây.

**Dữ liệu vào**

* Dòng đầu tiên: .
* dòng tiếp theo: – một cạnh của cây.

**Kết quả:**

* Gồm một số nguyên duy nhất là độ rộng của cây.

**Ví dụ:**

|  |  |  |
| --- | --- | --- |
| **TREE.INP** | **TREE.OUT** |  |
| **5**  **1 2**  **1 3**  **3 4**  **3 5** | **18** |

1 – 2

1 – 3

1 – 3 – 4

1 – 3 – 5

2 – 1 – 3

2 – 1 – 3 – 4

2 – 1 – 3 – 5

3 – 4

3 – 5

4 – 3 – 5

Comment: lần lượt cố định từng đỉnh làm root, sau đó BFs/DFs cũng được đi tính khoảng cách từ các đỉnh đến root. Bài này giải O(n) được nhưng 10^3 thì không cần lắm

**Cách ngây thơ:**

Sử dụng BFS từ mọi đỉnh, có biến g có ý nghĩa là quãng đường đã đi được từ đỉnh bắt đầu tới đỉnh hiện tại.

Quá trình BFS cho 1 đỉnh s như sau:

* Push đỉnh s vào queue, đánh dấu nó đã đi rồi, cho g lúc đầu là 1 (xem nó như g của đỉnh s)
* Pop đỉnh s ra và expand nó
* Expand như vậy, xét từng hàng xóm v chưa được đi qua, kiểm tra xem khoảng cách từ **s (từ s, chứ không phải từ đỉnh được pop ra hiện tại)** tới v đã được thêm vào kết quả chưa (có 1 mảng 2 chiều đánh dấu), nếu chưa thì cộng g vào kết quả và đánh dấu đã thêm vào kết quả. Push các v vào queue, đánh dấu những v đó đã đi rồi.
* Nếu có hàng xóm mới thì tăng g lên 1, không thì không tăng.

**Độ phức tạp:** *O(nV+nE)* *với V là số đỉnh, E là số cạnh, độ phức tạp BFS là O(V+E), BFS n lần*

**Cách Lowest common ancestor:**

[**https://www.geeksforgeeks.org/find-distance-between-two-nodes-of-a-binary-tree/**](https://www.geeksforgeeks.org/find-distance-between-two-nodes-of-a-binary-tree/)

Cách này mình sẽ tính khoảng cách của từng đôi node n1, n2 bằng cách tìm LCA (lowest common ancestor) giữa chúng, và tìm level của n1, n2 so với LCA đó, và cộng 2 level lại sẽ ra khoảng cách.

**Cách tìm LCA:** ta có một cây có đỉnh gốc là root, và ta muốn tìm LCA của 2 node n1, n2, thì ta đệ quy tìm với các trường hợp sau đây:

* Khi cây rỗng thì không có LCA
* Khi một trong 2 node là root, thì suy ra LCA của 2 node là root đó luôn
* Khi 2 node nằm ở 2 bên cây con trái phải, thì LCA của 2 node sẽ là root luôn
* Khi 2 node nằm lệch sang 1 bên cây con trái (phải), thì ta đệ quy tìm tiếp LCA với root là cây con trái (phải)
* Khi 2 node không nằm ở cây con nào, thì không có LCA

**Cách tìm level:** với 1 cây có gốc là root, tìm level của node u, thì ta bắt đầu từ root và đệ quy xuống 2 cây con, mỗi lần đệ quy thì tăng biến level lên 1, khi tìm thấy u rồi thì biến level hiện tại sẽ là level của u luôn. Vì hàm này sẽ được dùng nhiều lần nên ta sẽ lưu những level tìm được vào 1 bảng truy hồi (dp top down), bảng truy hồi levels[u, v] có nghĩa là trong 1 cây có root là u thì node v có level là levels[u, v]

Vì đề cho input là **danh sách kề**, để thực hiện những thuật toán trên ta phải chuyển danh sách kề về dạng cấu trúc **Node\*,** để làm vậy ta chỉ bắt đầu từ 1 đỉnh bất kỳ, rồi dùng BFS loang ra, mỗi lần loang thì gán left right cho từng node đang đi qua (nếu left chưa có thì gán hàng xóm hiện tại vào left, nếu có rồi thì gán vào right)

Độ phức tạp: *O(n3), LCA là O(n), chạy qua mọi cặp đỉnh là O(n2)*

# **Bài 17:**

Cho số nguyên dương . Hãy tìm số nguyên dương nhỏ nhất có không quá 9 chữ số thỏa chỉ chứa các chữ số {0,1} và là bội của .

**Dữ liệu vào:**

* Gồm một số nguyên dương

**Kết quả:**

* Gồm một số nguyên duy nhất là

**Ví dụ:**

|  |  |
| --- | --- |
| **INPUT** | **OUTPUT** |
| **6** | **1110** |

Comment: N <= 100 rất nhỏ => Quy hoạch động

F(I, mod) là xây được tới chữ số thứ I và số đó % m = mod. Dưới là mã giả theo style đệ quy có nhớ

f(I, mod) -> f(I + 1, (mod \* 10 + x)% n) với x lần lượt 0 hoặc 1

tất nhiên mod = 0 thì đó là số chia hết cho n. Vấn đề là truy vết kết quả như thế nào 😊

Liệt kê hết tất cả các dãy nhị phân có độ dài là 9, for từ 000000001 lên, từng thằng chuyển thành số rồi lấy check xem có phải là bội của n không, nếu là bội thì dừng luôn

Việc liệt kê dãy nhị phân có thể làm theo cách sau:

Với mỗi chuỗi, để ra chuỗi tiếp theo, tìm thằng 0 đầu tiên từ phải qua, gán nó bằng 1, rồi gán hết những thằng 1 đằng sau thành 0 hết. Ví dụ: 10**0**11, số tiếp theo sẽ là 10**100**

Vì sao nó đúng: Nó cũng giống như cách mình “tìm” số tiếp theo ở cơ số 10, là mình tìm số đầu tiên khác 9, tăng nó lên 1, vì chuyển hết những số 9 đằng sau thành 0. Ví dụ: số tiếp theo của 53**4**99 là 53**500,** số tiếp theo của 12**3** là 12**4**

Ngoài ra cũng có thể dùng quay lui để liệt kê dãy nhị phân, nhưng tui nghĩ cách trên dễ hơn.

Cách này thử với n = 100 chạy cái ra ngay lập tức luôn, vì số dãy nhị phân có độ dài 9 là 2^9 = 512. Thậm chí tui cho chạy với tất cả n từ 1 tới 100 cùng lúc, nó vẫn ra ngay lập tức.

***Độ phức tạp****: O(2k) với k là số chữ số của bội nhỏ nhất thoả yêu cầu đề bài của n*

# **Bài 18:**

Cho dãy số là một hoán vị của các số từ đến . Có thể thực hiện thao tác sau trên dãy số: chọn vị trí và đảo ngược thứ tự các phần tử , các phần tử còn lại giữ nguyên.

**Yêu cầu:** Tìm số thao tác thực hiện ít nhất để dãy có thứ tự tăng dần.

**Dữ liệu vào:**

* Dòng đầu tiên:
* Dòng thứ hai:

**Kết quả:**

* Gồm một số duy nhất là số thao tác ít nhất.

**Ví dụ:**

|  |  |  |
| --- | --- | --- |
| **INP**UT | **OUT**PUT | **Giải thích** |
| 5  5 2 3 4 1 | 4 | 5 2 3 4 1  4 3 2 5 1  2 3 4 5 1  5 4 3 2 1  1 2 3 4 5 |

N quá nhỏ => Quay lui duyệt qua mọi trường hợp

\*\*\* Tìm kiếm : sử dụng thuật toán \* với hàm nếu .

Biểu diễn lại bài toán: Cho trạng trái bắt đầu và trạng thái kết thúc. Ta cần tìm số bước hành động ít nhất để đi từ trạng thái bắt đầu đến trạng thái kết thúc.

Ý tưởng: Sử dụng một giải thuật tìm kiểm có thông tin để giải quyết bài toán thuật toán tìm kiểm \*. Với hàm của một trạng thái: , trong đó : số bước hành động (lật mảng) và : hàm cho trạng thái, nếu .

Chứng minh kết quả của thuật toán \* là một kêt quả tối ưu (một kết quả đúng):

* <https://www.cs.cmu.edu/~anupamg/251-notes/pancakes.pdf>
* <https://www.researchgate.net/publication/303711066_Optimally_solving_permutation_sorting_problems_with_efficient_partial_expansion_bidirectional_heuristic_search>
* <https://www.researchgate.net/publication/220743657_Landmark_Heuristics_for_the_Pancake_Problem>

Qua 3 nghiên cứu trên, các tác giả chứng minh được rằng hàm nếu là một hàm tối ưu và đảm bảo thuật toán \* cho ra một kết quả tối ưu (kêt quả đúng).

Độ phức tạp:

* Thời gian: trong trường hợp tệ nhất ta phải đi đến tất cả hoán vị của mảng, với mỗi lần tạo ra một hoán vị ta tốn cho hành động lật mảng. Do đó độ phức tạp là
* Không gian: trong trường hợp tệ nhất ta phải đi đến tất cả hoán vị của mảng, đồng thời lưu lại mỗi hoán vị này thành 1 trạng thái của bài toán, với mỗi trạng thái có kích thước kích thước của mảng. Do đó tổng không gian lưu trữ cần dùng là

Do \* lưu trữ tất cả trạng thái được mở nên thường sẽ hết bộ nhớ cho phép trước khi giải xong, ta có thể sử dụng sử dụng \* để giảm độ phức tạp về bộ nhớ. Nhưng với giả thiết đề bài , ta có thể bỏ qua vẫn đề về bộ nhớ.

# **Bài 19:**

Cho đoạn liên tiếp, đoạn thứ có chiều cao. Chiều cao chênh lệch giữa 2 đoạn liên tiếp không vượt quá 1 đơn vị. Ta cần đào sâu xuống càng sâu càng tốt. Ta dự định chi đồng cho công việc này. Chi phí giảm chiều cao của một đoạn đi 1 đơn vị là 1 đồng.

***Yêu cầu:*** Cho biết các chiều cao và chi phí . Hãy xác định chiều cao thấp nhất (đào sâu nhất) mà ta có thể đào được với đồng mà vẫn đảm bảo chênh lệch giữa 2 đoạn liên tiếp không quá 1 đơn vị.

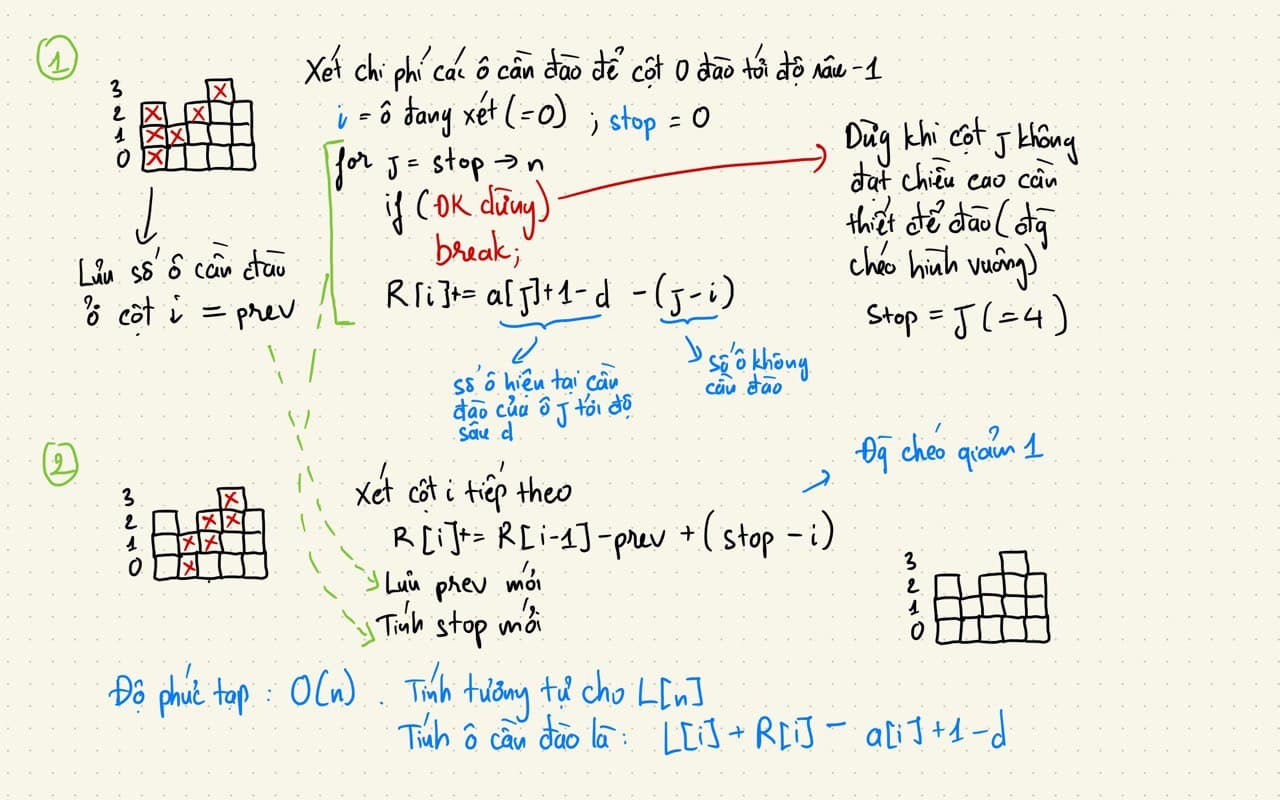
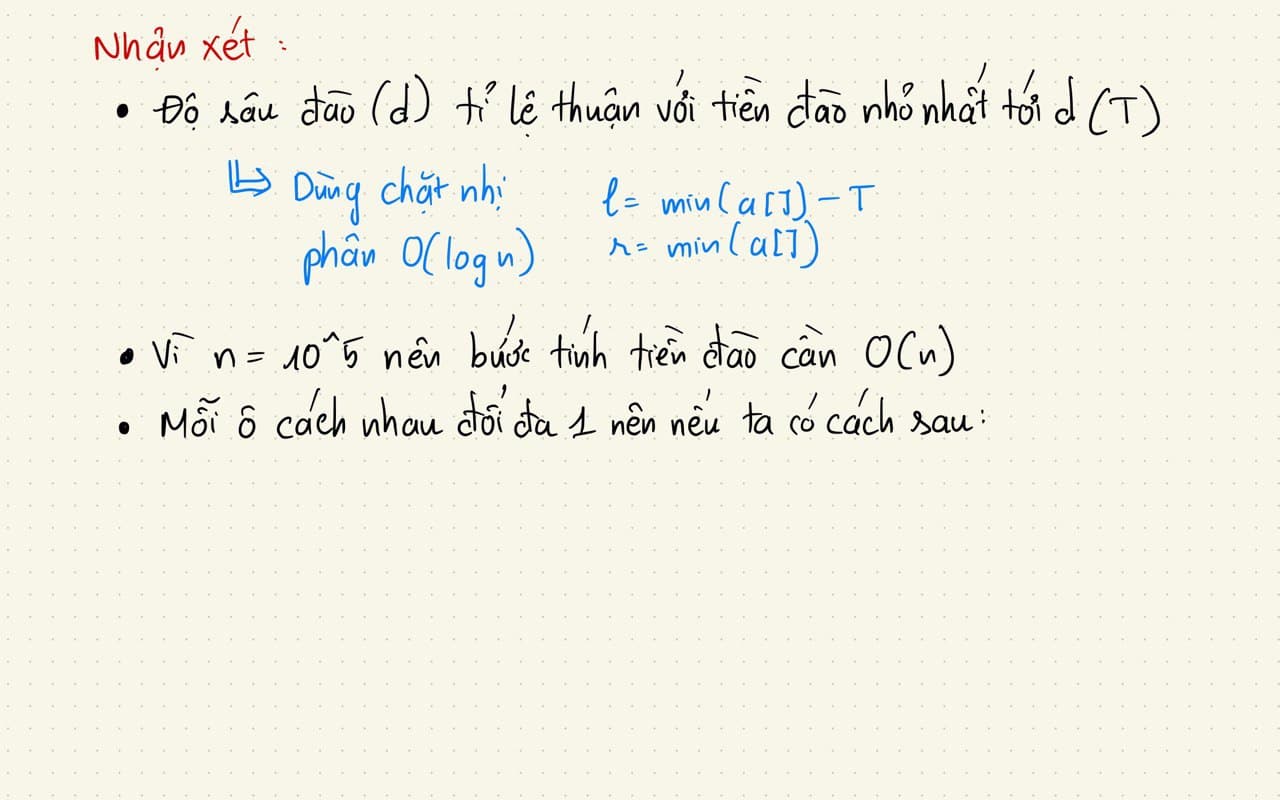
* Dòng đầu chứa 2 số nguyên dương
* Dòng thứ 2 chứa số nguyên không âm

***Kết quả***

Gồm một số nguyên duy nhất là độ cao thấp nhất mà công ty có thể đào được.

***Ví dụ:***

|  |  |  |
| --- | --- | --- |
| INPUT | OUTPUT | Giải thích |
| **4 3**  **1 1 1 1** | **-1** | |  |  |  |  |  | | --- | --- | --- | --- | --- | | **1** |  |  |  |  | | **0** |  |  |  |  | | **-1** |  |  |  |  | |
| **4 3**  **1 2 2 1** | **0** | |  |  |  |  |  | | --- | --- | --- | --- | --- | | **2** |  |  |  |  | | **1** |  |  |  |  | | **0** |  |  |  |  | |



Độ sâu cần đào (d) tỉ lệ thuận với tiền đào nhỏ nhất tới d (T)

Dùng chặt nhị phân O(logn).

l = min(a[]) – T.

r = min(a[])

Vì n = 10^5 nên bước tính tiền đào cần O(n).

Cách giải thích rõ:

Ta gọi số ô cần đào ở cột I sao cho đạt đến độ sâu d là d[i] (không cần thoả mãn ràng buộc). d[i] sẽ bằng a[j]+1-d

Với độ sâu d, gọi R[i] là số ô tổng cộng cần đào để cột I đạt tới độ sâu d mà vẫn thoả mãn điều kiện 2 cột kế bên nhau chênh lệch không quá 1 trong khi không quan tâm tới những cột bên trái i. Tương tự L[i], chỉ là chiều ngược lại.

Ta có tổng cộng số ô cần đào để cột I đạt tới độ sâu d mà vẫn thoả mãn điều kiện là L[i] + R[i] – d[i] (-d[i] để xoá đi phần bị trùng). Gọi nó là C[i]

Với mỗi độ sâu d, ta cần tìm C[i] nhỏ nhất với mọi cột, gọi C[i] nhỏ nhất đó là minDig(d)

Ta sẽ tìm ra d sao cho giá trị minDig(d) là lớn nhất mà chi phí vẫn <= T. Tìm bằng chặt nhị phân, nếu minDig hiện tại lớn hơn T thì xét phần bên phải (d cao hơn, đào ít hơn), minDig nhỏ hơn hoặc bằng T thì xét phần bên trái (d thấp hơn, đào sâu hơn)

Cách tính toán R[i]:

Khi ta muốn tính R[i], ta cho j chạy từ 0 tới n, mỗi lần lặp ta cộng thêm d[j] – (j+i) vào trong R[i]. Có nghĩa là ta cộng thêm số ô cần đào ở cột j (ko cần thoả mãn ràng buộc) trừ đi độ chênh lệch giữa cột j và I (để thoả ràng buộc, thành bậc thang).

Trong quá trình chạy nếu ta gặp 1 cột có độ cao thấp hơn mong đợi (j-i+1 > d[j]), thì ta dừng.

# **Bài 20:**

Xét tất cả các hoán vị của dãy số tự nhiên (1,2, … , 𝑛). Giả sử rằng các hoán vị được sắp xếp theo thứ tự từ điển và đánh số từ 1 tới 𝑛!

Ví dụ với 𝑛 = 3, có 6 hoán vị: (1,2,3); (1,3,2); (2,1,3); (2,3,1); (3,1,2); (3,2,1)

**Yêu cầu:** Cho trước một hoán vị () hãy cho biết số thứ tự 𝑥 của hoán vị đó và ngược lại: Cho trước một số thứ tự 𝑦 (1 ≤ 𝑦 ≤ 𝑛!) hãy tìm dãy hoán vị () mang số thứ tự 𝑦.

**Dữ liệu vào:**

* Dòng 1: Chứa 𝑛 số (𝑛 ≤ 20)
* Dòng 2: Chứa số 𝑦

**Kết quả:**

* Dòng 1: Ghi số 𝑥

Dòng 2: Ghi 𝑛 số

**Ví dụ:**

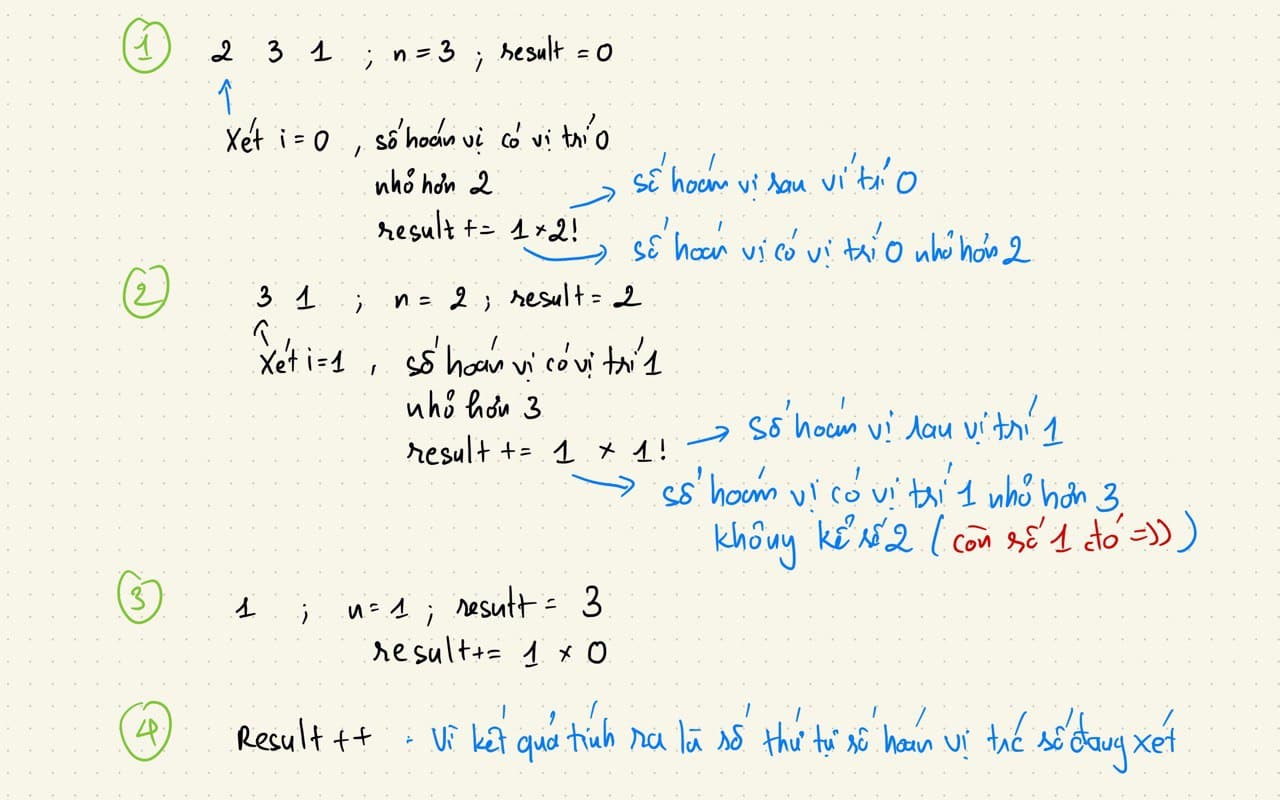
|  |  |
| --- | --- |
| INPUT | OUTPUT |
| **2 1 3**  **4** | **3**  **2 3 1** |

=> Này dùng toán [SPOJ.com - Problem SHHV](https://vn.spoj.com/problems/SHHV/)

p1, p2, … pn

* có liên tiếp nhau
* có bắt đầu từ 1

để chỉ có 1 giá trị y



Chia các hoán vị thành các phần bằng nhau, mỗi phần các hoán vị có chung số ở vị trí 0

(1,2,3); (1,3,2); (2,1,3); (2,3,1); (3,1,2); (3,2,1)

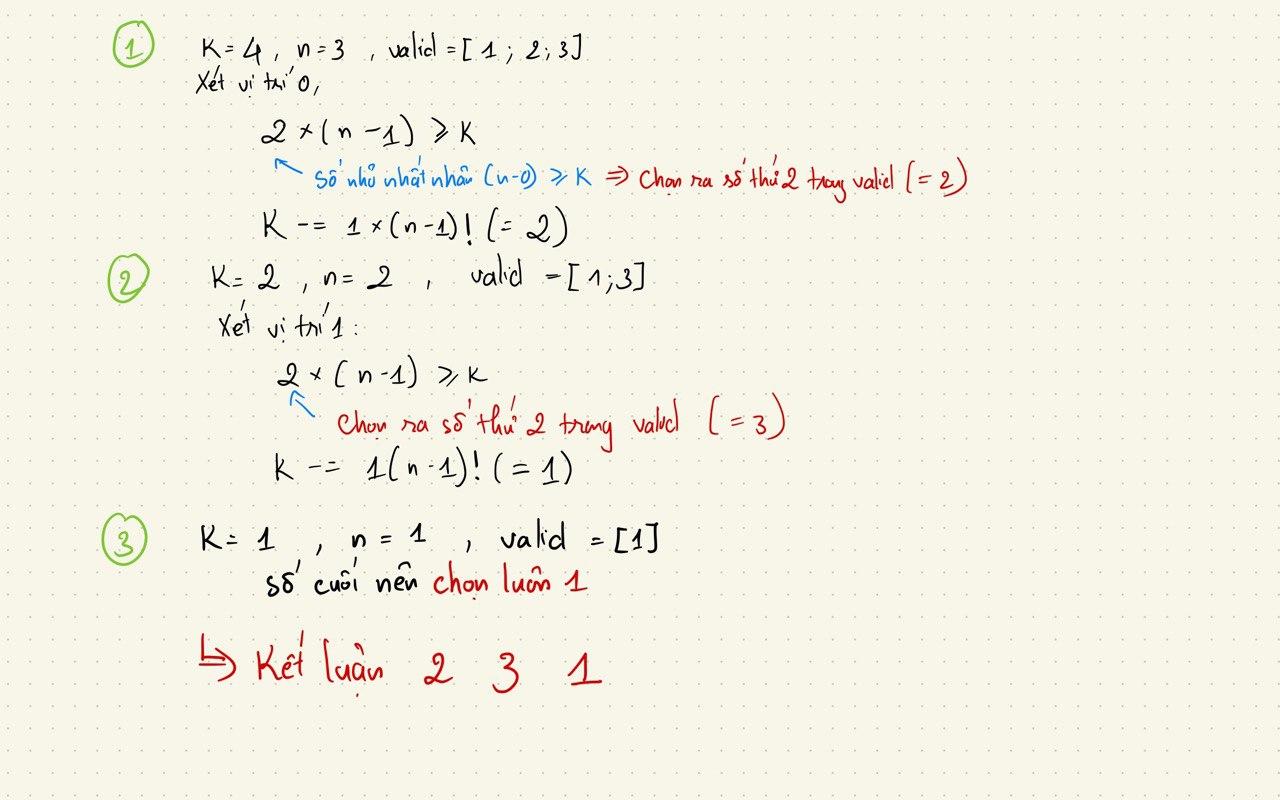
Tìm thứ tự của a = (2, 3, 1)

Ta bỏ đi các phần hoán vị nằm ở trước phần mà chứa (2, 3, 1), là bỏ đi phần (1,2,3); (1,3,2); Để biết các phần được bỏ đi có tổng cộng bao nhiêu hoán vị, là ta đang muốn biết số hoán vị có vị trí 0 nhỏ hơn 2 (a[0]). Thì ta sẽ lấy số phần nằm trước phần của (2, 3, 1) \* số hoán vị của những chữ số còn lại. Như trong trường hợp của (2, 3, 1) thì ta lấy 1 \* 2! = 2.

Khi đã xác định được vị trí phần theo số ở vị trí 0 (là 2), ta thu nhỏ bài toán thành tìm thứ tự của (3, 1) trong các hoán vị (1, 3); (3, 1). Ta lại tiếp tục chia các hoán vị này thành các phần bằng nhau mà mỗi phần có chung số ở vị trí 0, là ta sẽ chia thành (1; 3); (3, 1). Số hoán vị có vị trí 0 nhỏ hơn 3 là 1\*1! = 1.

Ta tiếp tục thu nhỏ bài toán, là tìm thứ tự của (1) trong các hoán vị (1). Số hoán vị có vị trí 0 nhỏ hơn 1 là 0 \* 0! = 0.

Để hoàn thành bài toán ta lấy 2+1+0=3, số 3 này là số hoán vị đằng trước hoán vị (2, 3, 1), nên ta chỉ cần lấy 3+1=4 là ra vị trí của (2, 3, 1)



Với bài toán này ta lại chia các hoán vị thành các phần bằng nhau mà mỗi phần các hoán vị có chung số ở vị trí đầu:

(1,2,3); (1,3,2); (2,1,3); (2,3,1); (3,1,2); (3,2,1)

Y = 4

Ta bắt đầu tìm phần của hoán vị có vị trí 4, ta thấy rằng mỗi phần bằng nhau như vậy sẽ có số hoán vị bằng số hoán vị của số các chữ số - 1 (vì chữ số đầu là cố định), là 2! = 2. Gọi số hoán vị của mỗi phần bằng nhau là k.

Ta sẽ tìm số a đầu tiên (nhỏ nhất) sao cho a \* k >= y (a sẽ là thứ tự của phần sẽ chứa hoán vị thứ tự y), thế số là a \* 2 >= 4. Ta thấy rằng a = 2, vậy suy ra hoán vị thứ tự y sẽ nằm trong phần thứ 2. Suy ra số ở vị trí 0 của hoán vị thứ tự 4 là 2.

Ta thu nhỏ bài toán, là tìm hoán vị có vị trí 4–(1\*2!) = 2 trong các hoán vị (1, 3); (3, 1). Chia thành các phần bằng nhau: (1; 3); (3, 1). Ta lại tìm số a đầu tiên sao cho a \* 1 >= 2, là a = 2. Vậy hoán vị có thứ tự 2 sẽ nằm trong phần thứ 2. Suy ra trong bài toán lớn, số ở vị trí 1 của hoán vị thứ tự 4 là 3.

Thu nhỏ bài toán, tìm hoán vị có vị trí 2-(1\*1!)=1 trong các hoán vị (1). Vì chỉ còn 1 hoán vị còn lại nên ta sẽ chọn luôn thứ tự 1. Suy ra trong bài toán lớn, số ở vị trí 2 của hoán vị thứ tự 4 là 1.

Suy ra hoán vị có thứ tự 4 là (2, 3, 1)