HƯỚNG DẪN GIẢI ĐỀ THI CODE TOUR ONLINE CODE CHALLENGE #2

A. GAME OF STONE

Solution:

C++	https://ideone.com/426ObO
Java	https://ideone.com/YZDNpg
Python	https://ideone.com/3eUzcz

Tóm tắt đề:

Có N viên bi trong hộp. A và B lần lượt lấy ra trong hộp x viên bi với x là một số chính phương (bình phương của một số nguyên dương). Tại lượt của mình, nếu người đó không thể thực hiện được nước đi trên thì sẽ là người thua cuộc. Chúng ta cần tìm ra người có chiến thuật chơi chắc chắn đem lại chiến thắng với giả sử rằng cả hai người chơi đều chơi một cách tối ưu nhất và A là người đi trước.

Input

- Dòng đầu tiên chứa số nguyên dương $T(1 \leq T \leq 100)$là số bộ test.
- Tdòng tiếp theo, mỗi dòng chứa một số nguyên dương $N(1 \leq N \leq 10^5)$

Output

In ra T dòng, với mỗi dòng là đáp án của bộ test thứ $i(1 \leq i \leq T)$. Với mỗi bộ test, in ra "A" nếu A là người chiến thắng, ngược lại in ra "B".

1 4	A

1 2	B

Giải thích:

- Với N = 2, A buộc phải lấy ra 1 viên trong hộp. Đến lượt của mình, B chỉ cần lấy ra thêm 1 viên nữa và trở thành người chiến thắng.
- Với N = 4, A chỉ cần lấy hết tất cả 4 viên và trở thành người chiến thắng.

Hướng dẫn giải:

Gọi K là tập hợp những giá trị mà nếu trong hộp có $k(k \in K)$ viên bi thì người đang có lượt chơi sẽ chắc chắn bị thua. Giá trị k đơn giản nhất mà chúng ta có được chính là k = 0 (đến lượt người nào mà trong hộp không còn viên bi nào thì người đó chắc chắn sẽ thua). Như vậy, chiến thuật tối ưu sẽ là luôn cố gắng lấy bi sau cho cho số bi còn lại trong hộp sẽ bằng một trong những giá trị k như vậy. Vì A là người đi trước, nên để A chiến thắng thì A chỉ cần kiểm tra trong số những số chính phương bé hơn số bi hiện có trong hộp - số bi mà A có thể lấy ra, có số nào mà sau khi lấy trong hộp sẽ còn lại $k(k \in K)$viên bi hay không. Nếu có thì A sẽ là người chiến thắng. Ngoài trường hợp này ra, B sẽ là người chiến thắng.

Đây là dạng bài **kinh điển trong DP** bạn có thể tìm thấy bài này nhiều dạng khác nhau, tuy nhiên phải cẩn thận vì có thể bài của bạn sẽ tăng độ phức tạp nếu chạy T test không phù hợp. Bạn cần áp dụng DP để lưu trữ lại đáp án đã có trước đó.

Như vậy ta có mã giả như sau:

```
# Khởi tạo mảng dp có N+1 với dp[i] = true khi A sẽ thắng nếu trong
hộp có i viên bi.

# Tìm những giá trị mà A có thể thắng

for i=1...N+1
      // Nếu A có thể đưa hộp về i viên bi thì A sẽ thắng
      if dp[i] = false:
            j = 1
            while (i + j*j <= N):
                  dp[i + j*j] = true
                  j += 1
ans = dp[N]
```

Chúng ta có thể tính toán trước tất cả giá trị có thể và lưu trong mảng dp để tiết kiệm chi phí tính toán cho nhiều testcase khác nhau.

Độ phức tạp: $O(MAXN\sqrt{(MAXN)} + T)$ với T là số bộ test và MAXN là giá trị lớn nhất có thể của N.

B. COUNT ARRAYS

Solution:

C++	https://ideone.com/cHF365
Java	https://ideone.com/yNtRE0
Python	https://ideone.com/qGytHT

Tóm tắt đề:

Cho 2 số nguyên N và K. Đếm số lượng dãy số nguyên A kích thước 2N sao cho có đúng N số 1, N số -1, và giá trị đúng bằng K.

Giá trị của một dãy số A được định nghĩa: gọi dãy B là tổng tiền số của A (B[0] = 0), giá trị của dãy A được định nghĩa là số lượng chỉ số i sao cho B[i] < 0 hoặc B[i+1] < 0.

Input

- Dòng đầu tiên chứa số nguyên dương T là số lượng test (T ≤ 1000)
- T dòng tiếp theo, mỗi dòng chứa hai số nguyên dương N và K (1 ≤ N ≤ 100000, 0 ≤ K ≤ 2N)

Output

In ra T dòng, với mỗi dòng là đáp án của bộ test thứ $i (1 \leq i \leq T)$, là số lượng dãy thỏa mãn yêu cầu khi lấy dư cho 1000000007.

2 1 1 2 2	0 2

Giải thích:

- Với N = 1, ta không có dãy nào có giá trị bằng 1.
- Với N = 2, ta có hai dãy có giá trị bằng 2. Đó là [1, -1, -1, 1] và [-1, 1, 1, -1]

Hướng dẫn giải:

Ta sẽ biến đổi bài toán này như sau.

- Ta sẽ bắt đầu từ điểm (0, 0) trên mặt phẳng tọa độ. Xét một dãy A hợp lệ bất kì, xét từng phần tử một của dãy A. Với mỗi i theo thứ tự từ 0 đến 2N - 1, nếu A[i] = +1, ta sẽ di chuyển từ (x, y) sang phải (x+1, y). Nếu A[i] = -1, ta sẽ di chuyển lên trên từ (x, y) lên (x, y+1).

Do dãy A có đúng N số -1 và N số 1, nên ta sẽ di chuyển N bước đi lên, và N bước sang phải, và cuối cùng ta sẽ đến ô (N, N).

Nhận xét: Giá trị của dãy A đúng bằng số lượng bước đi nằm phía trên đường chéo x = y.

Thật vậy, nếu tại một thời điểm nào đó, ta đang ở một tọa độ có y > x, có nghĩa là lúc đó ta đã thực hiện số lượng bước đi lên trên nhiều hơn số lượng bước đi sang phải. Như vậy tổng tiền tố lúc đó của dãy A phải âm, và sẽ đóng góp vào giá trị của dãy.

Với K = 0, thì đường đi từ (0, 0) đến (N, N) không bao giờ vượt lên trên đường chéo chính. Số lượng đường đi thỏa mãn chính là số Catalan thứ N.

Với K tổng quát, thì theo https://en.wikipedia.org/wiki/Catalan_number#Third_proof, ta có cách để song ánh bất kì một đường đi nào có giá trị từ K sang giá trị K+2. Như vậy số lượng đường đi có giá trị bằng K đúng bằng số lượng đường đi có giá trị K+2. Điều này cũng được thể hiện qua công thức của số Catalan: có tổng cộng C(2N, N) đường đi có giá trị bất kì. Tất cả các đường đi này được chia đều cho N+1 giá trị có thể của đường đi (tất cả các K chẵn từ 0 đến 2N). Như vậy có công thức Catalan(N) = C(2N, N)/(N+1). Đồng thời cũng chứng minh được nếu K lẻ, thì không có dãy nào thỏa.

Như vậy ta có lời giải: Nếu K chẵn, in ra số Catalan thứ N. Nếu K lẻ, in ra 0.

Việc còn lại là tính đáp án modulo 1000000007. Ta có thể tính trước tất cả các giai thừa từ 0 đến 2N, và sử dụng thuật toán tìm nghịch đảo modulo để tính được giá trị cần.

Độ phức tạp: O(MAXN + T) với T là số bộ test và MAXN là giá trị lớn nhất có thể của N. Độ phức tạp có thể có thêm nhân tố log(MOD) với MOD = 1000000007, tùy vào cách của người giải.

C. MELODY

Solution:

C++	https://ideone.com/xC9TQN

Tóm tắt đề:

Một bài hát sẽ bao gồm nhiều hợp âm và mỗi hợp âm sẽ gồm một hay nhiều nốt nhạc. Chỉ xét 7 loại nốt nhạc cơ bản là Đô, Rê, Mi, Fa, Sol, La, Si. Quy ước độ cao của nốt Đô trầm nhất là 1, độ cao của các nốt kề nhau sẽ chênh nhau là 1 đơn vị.

Bài nhạc của An sáng tác gồm có N hợp âm, hợp âm thứ i có Ni nốt nhạc. Kiểm tra liệu bài nhạc của An có thỏa mãn: với mọi hợp âm thứ i ($2 \le i \le N$) có ít nhất $N_{i-1} / 2$ nốt cùng loại với hợp âm thứ $i - 1$.

(Lưu ý: Nếu $N_{i-1} / 2$ không phải là số nguyên, lấy con số nguyên nhỏ hơn gần $N_{i-1} / 2$ nhất).

Input

- Dòng đầu tiên là N – số hợp âm có trong bài hát ($2 \le N \le 120000$).
- N dòng tiếp theo:
  - Số đầu tiên sẽ là N_i – số nốt có trong hợp âm thứ i ($1 \le Ni \le 7, 1 \le i \le N$). N_i số tiếp theo lần lượt là độ cao của các nốt có trong hợp âm thứ i. (Độ cao của các nốt luôn là số nguyên dương bé hơn 10^9).

Output

"YES" nếu bài nhạc của An thỏa mãn điều kiện, ngược lại in "NO"

Ví dụ

3 2 1 10 3 5 2 8 3 12 1 6	YES

Giải thích:

- Bài nhạc của An gồm N = 3 hợp âm:

    - Hợp âm đầu tiên gồm N1 = 2 nốt tương ứng với 2 độ cao: 1 10
    - Hợp âm thứ hai gồm N2 = 3 nốt tương ứng với 3 độ cao: 5 2 8
    - Hợp âm thứ ba gồm N3 = 3 nốt tương ứng với 3 độ cao: 12 1 6
- Hợp âm đầu tiên gồm 2 loại nốt là Đô và Mi; hợp âm thứ hai gồm 3 loại nốt Sol, Rê, Đô; hợp âm thứ ba gồm 3 loại nốt là Sol, Đô, La. Vậy hợp âm thứ hai có 1 nốt cùng loại với hợp âm đầu tiên là Đô; hợp âm thứ ba có 2 nốt cùng loại với hợp âm thứ hai là Sol và Đô.

➔ Đây là một bài nhạc thỏa mãn điều kiện.

Hướng dẫn giải:

Ban đầu, ta cho $\text{isGood} = \text{True}$.

Với mỗi hợp âm thứ curChordPos ($2 \leq \text{curChordPos} \leq \text{N}$), ta sẽ dùng 2 mảng $\text{cnt\_pre}[]$ và $\text{cnt\_cur}[]$ gồm 7 phần tử tương ứng với 7 loại nốt. Phần tử $\text{cnt\_pre}[\text{j}] = \text{true}$ tức là hợp âm thứ $\text{currentChordPos} - 1$ có chứa loại nốt j, tương tự $\text{cnt\_cur}[\text{j}] = \text{true}$ tức là hợp âm thứ curChordPos có chứa loại nốt j. Dùng thêm 2 biến n_pre và n_cur để lưu số nốt có trong hợp âm thứ $(\text{curChordPos} - 1)$ và hợp âm curChordPos. Sau đó lần lượt duyệt qua từng loại nốt, nếu hợp âm thứ curChordPos không có đủ $\text{n\_pre} / 2$ loại nốt giống với hợp âm trước đó, ta sẽ cho $\text{isGood} = \text{False}$ và ngừng kiểm tra.

Ngoài ra, ta có 1 nhận xét: Nếu hợp âm hiện tại có số nốt ít hơn một nửa số nốt của hợp âm trước đó thì điều kiện cũng không thể nào thỏa mãn. Vì vậy, ta có thể cho ngay $\text{isGood} = \text{false}$ và ngừng kiểm tra.

Thuật toán giải cụ thể như sau:

- **Bước 1:** Khởi tạo
    - $\text{isGood} = \text{true}$.
    - Mảng cnt_cur và cnt_pre gồm 7 phần tử với giá trị 0.

- **Bước 2:** Nhập dữ liệu của hợp âm đầu tiên

  Ta sẽ đọc vào các độ cao x của các nốt trong hợp âm đầu tiên và tăng $\text{cnt\_pre}[\text{x} \% 7]$ lên 1.

- **Bước 3:** Lần lượt duyệt i từ 2 đến N:
    - Nhập vào n_cur, độ cao x của các nốt trong hợp âm thứ i và tăng $\text{cnt\_cur}[\text{x} \% 7]$ lên 1.

- Nếu $\text{n\_cur} < \text{n\_pre} / 2$:
    - Gán isGood = false.
    - Ngừng vòng lặp.
- Khởi tạo $\text{count} = 0$.
- Duyệt j từ 0 đến 6 (duyệt qua từng loại nốt):
    - Nếu $\text{cnt\_pre}[j] > 0$ và $\text{cnt\_cur}[j] > 0$:
        - Tăng count lên 1.
- Nếu $\text{count} < \text{n\_pre} / 2$:
    - Gán $\text{isGood} = \text{false}$.
    - Dừng vòng lặp.
- Cập nhật lại n_pre và cnt_pre: $\text{n\_pre} = \text{n\_cur}$, $\text{cnt\_pre} = \text{cnt\_cur}$. Khởi tạo lại mảng cnt_cur với 7 phần tử với giá trị 0.

- **Bước 4:** Nếu $\text{isGood} = \text{true}$, xuất "YES". Ngược lại, xuất "NO".

Độ phức tạp thời gian: $\boldsymbol{O(N)}$.

D.MAGIC OF LOVE

Solution:

C++	https://ideone.com/w9ETqa
Python	https://ideone.com/vk2DGs
Java	https://ideone.com/yNRDCw

Tóm tắt đề:

Tính tổng dương lớn nhất trong ma trận con mà tất cả các phần tử trong ma trận đều là số chẵn.

Input

Mỗi bài gồm có T test cases $(1 \leq \mathrm{T} \leq 10)$. Mỗi test case gồm có:
Dòng đầu tiên chứa 2 số M, N với M, N là số dòng và số cột của ma trận $(0 \leq M, N \leq 100)$.
M dòng tiếp theo, mỗi dòng là N số nguyên dương $A[i][j]$ $(|A[i][j]| \leq 10^9$ với $0 \leq i < M$, $0 \leq j < N)$.

Output

Đầu ra gồm có T dòng:

Dòng thứ i $(1 \leq i \leq T)$: In ra "Test i: "
(không bao gồm dấu ngoặc kép) và tổng lớn nhất của ma trận con.
Nếu không tìm được ma trận thỏa yêu cầu của đề bài thì in ra "$impossible$"
(không bao gồm dấu ngoặc kép).

Ví dụ

1 4 5 1 3 3 4 7 1 -2 2 4 8 7 9 6 4 6 1 -12 3 3 1	30

Giải thích:

Ở đây, ta thấy được, ma trận con thỏa yêu cầu là ma trận

[2, 4, 8]

[6, 4, 6]

Hướng dẫn giải:

Đây là bài toán kinh điển **Maximum sum rectangle in a 2D matrix**, nhưng có chút khác biệt là bạn cần phải thực hiện với số chẵn.

Trước khi tìm tổng, ta cần xử lí các phần tử trong mảng đầu vào bằng cách gán âm vô cực cho những phần tử lẻ trong mảng vì theo yêu cầu của đề bài, ta cần tìm ma trận con có tổng dương, nên khi gán âm vô cực cho các phần tử lẻ, tổng ma trận con được tạo ra từ phần tử ở vị trí đó sẽ luôn có giá trị nhỏ hơn 0. Vì vậy, ma trận con đó sẽ không thỏa đề bài và ta đã xử lí được yêu cầu không có phần tử lẻ trong ma trận con lớn nhất đó. Cách này giúp ta tiết kiệm bộ nhớ hơn vì không phải tạo một mảng 2 chiều check để kiểm tra xem tổng ma trận con có tồn tại phần tử lẻ hay không.

Để làm bài toán này, ta có thể sử dụng 6 vòng for để duyệt các phần tử ở 4 đầu mút và tính tổng của các phần tử của ma trận con được tạo ra từ bốn phần tử trên. Như vậy, đây là một cách không tối ưu vì nó có độ phức tạp là $O(N^3 * M^3)$.

Ta có thể tối ưu hóa bài toán hơn bằng cách tạo ra một mảng sum với $sum[i][j]$ là tổng ma trận con với đầu mút trái trên là $(0, 0)$ và đầu mút phải dưới là (i, j). Với cách làm này, ta có thể tính tổng của một ma trận con bằng cách chạy 2 dầu mút là (i, j) và (k, t) với $k > i, t > j$ theo công thức:

$$sumMatrix = sum[k][t] - sum[i][t] - sum[k][j] + sum[i][j]$$

Với cách làm này ta, có thể giảm độ phức tạp xuống còn $O(N^2 * M^2)$.

Nhưng ở đây, ta còn một cách tối ưu hơn nữa là sử dụng thuật toán Kadane để tìm được tổng lớn nhất. Sau đó, ta có thể sử dụng Kadane.

Thuật toán giải cụ thể như sau:

- **Bước 1:** Khởi tạo

- $INF = 10^9 * 100 * 100$ vì một mảng có $M * N$ phần tử với M và N tối đa là 100, với mỗi phần tử, giá trị tối đa có thể có của nó là 10^9 vậy nên chỉ cần gán INF với giá trị trên thì nó sẽ là tổng lớn nhất có thể có cũng như là giá trị vô cực của mảng.
- $arr[i][j]$: Cho biết ô (i, j) có giá trị nào.
- Gán $arr[i][j] = -INF$ khi phần tử đó là một số lẻ.

- **Bước 2:** Áp dụng thuật toán Kadane

- Gán $maxSum = -INF$

- Duyệt theo cột từ trái qua phải với hai biến $left$ và $right$ là đầu mút trái và đầu mút phải của ma trận con đang xét

- Với mỗi lần duyệt, tạo mảng 1 chiều $temp$ với $temp[i]$ là tổng các phần tử tính từ vị trí $(i, left)$ đến $(i, right)$ đang xét. Với mỗi lần như vậy, ta có thể tính được sum lớn nhất từ một dãy con của temp thông qua $Kadane(temp, M)$ và lấy $maxSum = max(sum, maxSum)$

- Với $maxSum$, nếu $maxSum > 0$ thì in ra kết quả, nếu không, ta in ra "$impossible$"

Thuật toán Kanade:

- Ta có mảng arr có n phần tử.

- Ban đầu, ta khởi gán $Sum = 0, maxSum = -INF$ và $finish = -1$.
- Duyệt từng phần tử $arr[i]$, với mỗi lần duyệt:
  - Ta lấy $Sum = Sum + arr[i]$
  - Nếu $Sum < 0$, ta gán $Sum = 0$
  - Ngược lại, nếu $Sum >= maxSum$, ta cập nhật $maxSum = Sum$
- Nếu $finish != -1$ tức là tồn tại $maxSum$ thỏa yêu cầu, ta trả về kết quả đó
- Nếu không, ta chọn phần tử lớn nhất trong mảng để trả về.

Đánh giá độ phức tạp thời gian:

Để tạo mảng Sum, ta cần tốn chi phí thời gian là $O(M * N)$

Sau đó, ta xác định chiều rộng cho ma trận qua biến left và biến right. Việc xác định này tốn hết $O(\frac{1}{2} * N * (N - 2))$

Với mỗi chiều rộng như vậy, ta lại duyệt từng dòng để tính tổng từ $left$ đến $right$ cho mỗi tầng của ma trận, số lần duyệt này tốn một lượng chi phí là $O(M)$.

Độ phức tạp thời gian: $\boldsymbol{O(M * N + \frac{1}{2} * N * (N - 2) * M) \approx O(N^2 * M)}$