Tutorial: A Caminho!

Isabela Souza

O objetivo do problema é ler um número inteiro (o tempo do GPS) e imprimir esse mesmo número.
Para isso, basta seguir três passos simples:
1 - Definir uma variável para receber o tempo.
2 - Ler a entrada e guardar o valor nessa variável.
3 - Imprimir esse valor exatamente como recebeu, apenas o número puro.
Exemplo de código em python:

```
t = int(intput())
print(t)
```

Tutorial: Biscoitos

Samuel Silveira

Todos os biscoitos faltantes foram comidos pela cachorrinha. Assim, a resposta é dada pela diferença entre o número de biscoitos na cozinha antes e depois de atender à porta, ou seja, $X - Y$.

Tutorial: Classificadas?

João Vítor Fonseca Pimenta

Para saber se o time se classificou, basta calcular quantos problemas ele resolveu ao todo.

Somamos os valores A, B e C, que representam a quantidade de problemas resolvidos por cada integrante da equipe.

Em seguida, comparamos essa soma com X, que é a quantidade mínima necessária para avançar para a Final Nacional.

- Se $A + B + C \geq X$, imprimimos `S`.
- Caso contrário, imprimimos `N`.

Tutorial: Dessa vez a maratona não é de programação

Luisa Oliveira

Esse problema pode ser resolvido calculando o resto da divisão (módulo). A pista é circular, então após cada volta completa Adrielly retorna ao quilômetro 0. A posição final depende apenas da distância que sobra após completar todas as voltas possíveis. Portanto, basta calcular o resto da divisão entre a distância percorrida e o comprimento da pista. A resposta é dada por N % K.

Tutorial: Escalonamento de Candidatas

Eduardo Freire

Qualquer ordenação válida das candidatas as deixaria ordenadas em ordem estritatmente crescente segundo o primeiro critério. Note que uma tal ordenação, se existe, é única, ou seja, não existem duas ordenações distintas das candidatas que as deixam ordenadas de forma estritamente crescente segundo o primeiro critério.

Assim, para resolver o problema, basta primeiramente ordenar as candidatas segundo o primeiro critério e em seguida verificar se essa ordenação é válida, ou seja, verificar se ela ordena as candidatas de forma estritamente crescente segundo todos os critérios. Se sim, a resposta é `SIM`, claramente. Se não, segue da unicidade que a resposta é `NAO`.

Tutorial: Floricultura

Emerson Luiz Cruz Junior

Cada posição da matriz representa uma flor. Uma flor só pode ser considerada se estiver disponível, ou seja, se a posição correspondente no mapa de disponibilidade possuir valor 1.

Entre todas as flores disponíveis, Isa deseja escolher um **valor de beleza** que apareça o maior número de vezes. Em outras palavras, devemos descobrir qual valor de beleza ocorre com maior frequência entre as flores disponíveis.

Por exemplo, se as flores disponíveis possuem valores:

$$20,\ 10,\ 10,\ 25,\ 10$$

o valor 10 aparece três vezes, enquanto os demais aparecem apenas uma vez. Portanto, a resposta seria 10.

Além disso, caso existam vários valores com a mesma frequência máxima, Isa prefere o menor valor de beleza, pois valores menores representam flores mais bonitas.

Também é importante observar que, se nenhuma flor estiver disponível, a resposta deve ser -1.

Com isso podemos descrever a seguinte solução: O mapa de disponibilidade informa quais flores podem ser compradas.

Ao ler a matriz de beleza, podemos verificar a disponibilidade correspondente:

- se a flor não está disponível (0), ignoramos seu valor;
- se a flor está disponível (1), incrementamos a frequência daquele valor de beleza.

Assim, ao final da leitura teremos, para cada valor de beleza, quantas flores disponíveis possuem aquele valor.

Depois disso, basta percorrer todas as frequências e encontrar o valor de beleza que aparece mais vezes.

A implementação fornecida utiliza um vetor `freqs`, onde

`freqs[x]`

armazena quantas flores disponíveis possuem beleza x.

Como os valores de beleza satisfazem

$$1 \leq V_{i,j} \leq 10^5,$$

é possível armazenar todas as frequências diretamente em um vetor de tamanho $10^5 + 1$ ou em um dict/map.

Tutorial: Encontro Performático

Arthur Bispo

A condição para escolher um número X no intervalo é equivalente ao máximo divisor comum (mdc) dele com C ou L ser maior do que 1. Assim, podemos dividir os números válidos do intervalo em três grupos distintos:

- **Os que apenas Cibelly pode escolher:** $\text{mdc}(X, C) > 1$ e $\text{mdc}(X, L) = 1$.
- **Os que apenas Luisa pode escolher:** $\text{mdc}(X, L) > 1$ e $\text{mdc}(X, C) = 1$.
- **Os que ambas podem escolher:** $\text{mdc}(X, C) > 1$ e $\text{mdc}(X, L) > 1$.

Como o jogo termina somente quando nenhuma das jogadoras puder realizar mais jogadas, os números exclusivos de cada uma estão garantidos e não podem ser "roubados" pela adversária. Portanto, a disputa real ocorre nos números que ambas podem escolher.

Sendo assim, a estratégia ótima para maximizar a pontuação é jogar de forma gulosa nos números compartilhados: na sua vez, a jogadora deve sempre escolher o **maior** número disponível que ambas podem escolher. Quando esses números forem esgotados, basta somar à pontuação final de cada jogadora todos os números que eram exclusivos dela.

Tutorial: Pão de Metro

Daniel Saad Nogueira Nunes

Podemos resolver esse problema usando a técnica de busca-binária na resposta. Iremos tentar adivinhar um valor x e verificar se é possível obter, ao menos, k pedaços de pão de tamanho x. Se for possível, podemos aumentar a nossa estimativa, caso contrário, precisamos diminuir. Então, x é ajustado conforme a busca binária. Dada a estimativa x, o número de pedaços de tamanho x obtenível é exatamente:

$$\sum_{i=1}^{n} \left\lfloor \frac{p_i}{x} \right\rfloor$$

O pseudocódigo abaixo ilustra essa ideia:

Algoritmo 1: Pseudocódigo da solução

```
1  l ← 0
2  r ← max{p_i | 1 ≤ i ≤ n}
3  for( it ← 1 to 100 )
4  |  x ← (l + r)/2
5  |  count ← 0
6  |  for( i ← 1 to n )
7  |  |  count ← count + ⌊p_i / x⌋
8  |  if( count ≥ k )
9  |  |  l ← x
10 |  else
11 |  |  r ← x
12 return l
```

Precisamos fazer um número suficiente de iterações de busca-binária para garantir a precisão exigida. No pseudocódigo, utilizou-se 100 iterações.

Análise: A complexidade dessa solução é $\Theta(n \cdot m)$, onde m é o número de iterações da busca binária. Na prática, com poucas iterações, conseguimos a precisão exigida, então o valor $m = 100$ é muito mais do que suficiente.

Tutorial: Montanha

João Carlos Gonçalves de Oliveira

Dado uma árvore com N vértices enraizado no vértice 1, onde cada vértice i (para $i \geq 2$) possui um único pai p_i tal que $p_i < i$, queremos responder a seguinte pergunta para cada vértice i: Dentre todas as $\binom{N}{K}$ maneiras possíveis de escolher exatamente K vértices iniciais, em quantas delas o Menor Ancestral Comum (LCA) de todos os K vértices escolhidos será especificamente o vértice i?

Para que um vértice u seja o LCA exato de um conjunto de K vértices escolhidos, duas condições obrigatórias devem ser atendidas simultaneamente:

1. Todos os K vértices escolhidos devem pertencer à subárvore enraizada em u.
2. Os K vértices não podem ficar todos dentro da subárvore de um único filho de u. Se todos os K elementos fossem escolhidos dentro da subárvore de um filho v, o LCA real desse conjunto seria o próprio v (ou algum descendente dele), e não u.

A partir disso, é possível calcular a resposta de cada vértice de maneira direta:

$$\text{ans}[u] = \binom{\text{sub}[u]}{K} - \sum_{v \in \text{filhos}(u)} \binom{\text{sub}[v]}{K}$$

Em que $\text{sub}[x]$ representa o tamanho total da subárvore enraizada no vértice x.

Tutorial: Sopa de Dígitos

João Carlos Gonçalves de Oliveira

Uma solução direta seria iterar para cada número x no intervalo $[L, R]$, simular o preenchimento da matriz 4×4 adicionando os zeros à esquerda, percorrer o trajeto gerado pelos comandos para calcular a soma S e verificar a divisibilidade por 3.

Essa abordagem não é rápida o suficiente, uma vez que o intervalo $[L, R]$ pode ter até 10^{15} números.

Uma técnica clássica para problemas de contagem em intervalos é transformar a busca no intervalo $[L, R]$ em duas buscas. Seja $f(x)$ uma função que retorna a quantidade de números especiais e a soma deles no intervalo $[0, x]$. A resposta para o problema original pode ser obtida por:

$$f(R) - f(L-1)$$

Para computar $f(x)$ de forma eficiente, construímos os números de 16 dígitos da esquerda para a direita (do dígito mais significativo para o menos significativo) usando programação dinâmica.

O preenchimento da matriz 4×4 e as posições dos dígitos que serão somados são fixos para um determinado caso de teste. Portanto, podemos pré-calcular um vetor booleano collect$[i]$, onde collect$[i] = true$ indica que o dígito na posição i da string de 16 dígitos (equivalente a uma célula visitada na matriz) deve fazer parte da soma S.

Definimos uma DP que guardará um par de valores (cnt, sum):

$$\text{DP}[i][\text{rem}][\text{tight}]$$

Em que:

- i: O índice do dígito atual que estamos decidindo.
- rem: O resto da soma acumulada dos dígitos coletados até o momento módulo 3.
- tight: Um estado booleano clássico em DP de Dígitos. Se tight = true, significa que os dígitos colocados até agora são exatamente iguais aos prefixos de x, logo, o dígito atual não pode passar de x$[i]$. Se tight = false, estamos livres para escolher qualquer dígito de 0 a 9.

Para cada estado, iteramos por todos os dígitos válidos d (de 0 até o limite definido por tight).

Sejam cnt e sum, respectivamente, a quantidade de sufixos válidos e a soma acumulada desses sufixos a partir do estado $(i+1)$, colocando o dígito d. A transição a partir de i é definida por:

- $\text{DP}[i][j][k].\text{first} = \sum(\text{cnt}) \pmod{\text{MOD}}$
- $\text{DP}[i][j][k].\text{second} = \sum \left[\left((d \times 10^{15-i} \pmod{\text{MOD}}) \times \text{cnt}\right) + \text{sum}\right] \pmod{\text{MOD}}$

Ao final, a resposta para um limite x será obtida em dp[0][0][true].

O número de estados da nossa DP é extremamente pequeno: 16 (dígitos) $\times$ 3 (restos) $\times$ 2 (tight) = 96 estados. Para cada estado, iteramos no máximo 10 dígitos (de 0 a 9). Essa abordagem evita recalcular caminhos repetidos e é rápida o suficiente para passar no tempo limite.