Tutorial: A Canção dos Pássaros

Daniel Porto

Para resolver o problema é necessário somente ordenar a string de entrada.

Uma possível solução é usar as funções de ordenamento das próprias linguagens. Neste caso teríamos uma complexidade de O(n log n).

Outra opção mais eficiente é ordenar com o counting sort. Neste caso, teríamos na prática uma complexidade O(n).

Tutorial: Baru++

Guilherme Ramos

A solução é simples, basta ler a pergunta, repeti-la em uma linha e acrescentar outra linha com a mensagem “`Essa questao eh baru++!`”.

Tutorial: C-errado

Guilherme Ramos

A solução é simples, basta ler as sentenças enquanto fornecidas e, caso haja um "C" imediatamente após uma letra e antes de algo que não seja letra (como o fim da sentença, substitua-o pelo string "-se".

Tutorial: Dia Nacional do Cerrado

Edson Alves

Este problema é uma variação do clássico FizzBuzz. Primeiramente, seja n um inteiro positivo que não é múltiplo de 3 e nem de 5 e $f(n)$ a posição que n ocupa na sequência de placas verdes. Observe que, pelo Princípio da Inclusão/Exclusão, que

$$f(n) = n - \left\lfloor \frac{n}{3} \right\rfloor - \left\lfloor \frac{n}{5} \right\rfloor + \left\lfloor \frac{n}{15} \right\rfloor$$

Como a função $f(n)$ é crescente, é possível usar uma busca binária para localizar o valor m tal que $f(m) = p$. É preciso tratar os múltiplos de 3 ou de 5 na busca binária ou expandir f para incluir também tais números: neste caso, f será não-decrescente e a resposta m pode precisar de um ajuste, sendo alterado para o maior inteiro menor que m que não é múltiplo de 3 e nem de 5.

Para os limites da busca binária, atente que o menor valor possível é 1 e que, a cada conjunto de 15 números consecutivos, apenas 8 deles serão placas verdes, de modo que 2×18^{18} é um limite superior. Esta solução tem complexidade $O(\log p)$.

Tutorial: Fatorial

Edson Alves

A primeira parte do problema consiste em determinar quantos são os zeros à direita de $n!$ em sua representação decimal. Seja $E_p(n)$ a maior potência de um primo p que divide $n!$. Este valor pode ser computado em $O(\log n)$, uma vez que

$$E_p(n) = \left\lfloor \frac{n}{p} \right\rfloor + \left\lfloor \frac{n}{p^2} \right\rfloor + \ldots + \left\lfloor \frac{n}{p^k} \right\rfloor,$$

onde $\lfloor \frac{n}{p^{k+1}} \rfloor = 0$. Como $10 = 2 \times 5$ e $E_2(n) \geq E_5(n)$, o número de zeros à direita de $n!$ é igual a $E_5(p)$.

Seja $k = E_5(p)$. O problema consiste em determinar o resto da divisão de $n!/10^k$ por 10. Isto pode ser feito em três etapas.

1. determinar o resto da divisão de $n!/10^k$ por 2;
2. determinar o resto da divisão de $n!/10^k$ por 5; e
3. determinar o resto da divisão de $n!/10^k$ por 10 por meio do Teorema Chinês dos Restos.

A primeira etapa pode ser realizada em $O(1)$: se $E_2(n) > E_5(n)$, o número $n!/10^k$ será par, e portanto o resto será igual a zero. O único caso onde $E_2(n)$ é igual a $E_5(n)$ ocorre quando $n = 1$, e aqui o resto da divisão de 1! por 2 será igual a 1.

A segunda etapa é a mais trabalhosa, e pode ser realizada em $O(\log n)$. O primeiro passo é determinar o resto da divisão de 2^k por 5. Como $(2, 5) = 1$ e 5 é primo, vale o Pequeno Teorema de Fermat, ou seja,

$$a^{p-1} \equiv 1 \pmod{p},$$

onde p é primo e $(a, p) = 1$. No caso particular de $a = 2$ e $p = 5$, temos que

$$2^4 \equiv 1 \pmod{5}$$

Neste cenário, o resto da divisão a de 2^k por 5 é igual ao resto de 2^r por 5, onde $k = 4q+r$, com $0 \leq r < 5$.

Para computar o resto da divisão de $n!/5^k$ é preciso observar que, para um múltiplo m de 5, o resto da divisão dos quatro inteiros consecutivos entre m e $m + 5$ é igual a

$$(m + 1)(m + 2)(m + 3)(m + 4) \equiv 1 \times 2 \times 3 \times 4 \equiv 4 \equiv -1 \pmod{5}$$

Assim, o seguinte algoritmo computa o resto da divisão de $n!/5^k$ por 5:

1. Faça $b \leftarrow 0$
2. Enquanto $n > 0$ faça:
   (a) Se n não é múltiplo de 5, faça $b \leftarrow b \times n \pmod{5}$ e $n \leftarrow n - 1$;
   (b) Caso contrário, faça $t \leftarrow n$ div 5, $n \leftarrow t$ e $b \leftarrow b \times (-1)^t \pmod{5}$.

No algoritmo, t corresponde ao número de múltiplos de 5 entre 1 e n. Este número corresponde também ao número de quatro números consecutivos entre dois múltiplos de 5 neste intervalo, e a divisão por 5 de todos estes termos resulta em $t!$.

Por fim, o resto da divisão c de $n!$ por 10^k será dado por $c = a^{-1} \times b \pmod 5$, onde a^{-1} é o inverso multiplicativo de a módulo 5.

A terceira e última etapa consiste em usar o Teorema Chinês dos restos para resolver o sistema de equações de congruência

$$\begin{cases} x \equiv a \pmod 2 \\ x \equiv c \pmod 5 \end{cases}$$

O teorema nos diz que $x \equiv 6a + 5c \pmod{10}$. Assim esta etapa pode ser realizada em $O(1)$, usando-se esta expressão e os resultados das duas etapas anteriores. A solução tem complexidade $O(\log n)$.

Tutorial: Guarás Anônimos

Daniel Porto

A solução é ir construindo uma fila de prioridades, considerando como chave primária a FOME e como secundária a ordem inversa de chegada, e atualizando essa fila ou seus itens de acordo com a sequência das ações.

Tutorial: Hora Extra

Edson Alves

Seja y_k a sequência original ordenada, m a posição ocupada pela mediana ($m = N/2$ se N é par, $m = (N+1)/2$ se N é ímpar) e a e b os índices da primeira e da última ocorrência de M em y_k. Há três cenários possíveis:

1. se $m \in [a, b]$, então a resposta é zero;
2. se $m < a$, então devem ser removidos elementos da esquerda para a direita (primeiro y_1, depois y_2, etc);
3. se $m > b$, então devem ser removidos elementos da direita para a esquerda.

Definido o sentido das remoções, basta remover um elemento por vez e reavaliar a mediana após cada remoção. Para recuperar os índices dos elementos a serem removidos, basta construir um vetor cujos elementos são pares (x_i, i) e ordená-lo: este vetor fará o papel de y_k e trará os índices dos elementos a serem removidos.

Esta solução tem complexidade $O(N \log N)$.

Tutorial: Karaoke

Maxwell Oliveira

Podemos visualizar o problema usando grafos. Suponha um grafo completo de N vértices e retire as arestas dadas no input. Após isso, o problema se resume a encontrar e mostrar um caminho hamiltoniano do grafo. Note que, em geral, esse problema seria NP completo, mas aqui temos uma restrição que nos ajuda: Cada vértice possui muitas arestas!

Solução 1: Guloso

O problema pode ser resolvido com um algoritmo guloso que é dividido em dois passos:

i) Escolha um vertice inicial e de forma gulosa, adicione qualquer vértice válido à sua sequencia até que restem apenas 10 vértices não utilizados.

ii) Neste momento há poucos vértices sobrando e começa a ficar dificil decidir qual colocar, pois uma escolha errada pode nos levar a um beco sem saída. Sendo assim, podemos aproveitar o fato de que restam poucos vértices e testarmos todas as permutações deles possíveis. Para cada permutação, basta concatenar com a sequencia criada no passo anterior e testar se ela continua válida.

Prova do algoritmo

Para o primeiro passo, basta notar que cada vértice está limitado a, no máximo, outros 4. Como pretendemos deixar sobrar 10, sempre existe um vértice válido disponível.

Para o segundo passo, considere G o subgrafo gerado apenas por esses 10 vértices. Note que G é um grafo completo em que foram removidas as aretas do input. Assim, cada vértice de G possui grau de, pelo menos, $10 - 4 - 1 = 5$. Como $5 >= 10/2$, o teorema de Dirac nos diz que existe um ciclo hamiltoniano e, portanto, um caminho hamiltoniano também existirá. Neste caso, basta encontrar alguma ordem que satisfaça todos os requisitos.

* *Teorema de Dirac: Se um grafo G tem N vértices (com $N >= 3$) e cada vértice tem um grau de, pelo menos, $N/2$, então G tem um ciclo hamiltoniano..*

Solução 2: Random

Nesta solução, basta seguir os passos a seguir:

i) Para N pequeno ($N <= 12$) basta testar todas as permutações possíveis.

ii) Para N grande (N ¿ 12) podemos supor a ordenação inicial (1, 2, ..., N) e testar permutações aleatórias (random shuffle) até que uma funcione.

A intuição para esta solução é que, pelo mesmo argumento da outra solução, sabemos que para N ¿ 10 a solução existe. O total de soluções para um grafo completo é na ordem de O((N - 1)!)* e, nesse problema, temos um grafo quase completo, nos dando que essa é uma boa aproximação do numero real. Com tantas soluções existentes, a probabilidade do random encontrar uma é alta o suficiente, ainda mais que conseguimos testar mais de 1000 ordenações dentro do tempo limite.

A implementação é simples, mas ainda é necessário um certo cuidado para evitar que a complexidade tenha um log desnecessario, podendo causar um TLE.

* `https://math.stackexchange.com/questions/249817/how-many-hamiltonian-cycles-are-there-in-a-complete`

Tutorial: Mangabas

Edson Alves

Uma possível solução para o problema é composta de duas etapas. A primeira etapa é resolver o problema para um único balde. Seja $f_K(x, m)$ o número máximo de mangabas que podem ser ensacadas a partir de um único balde com x mangabas. São dois os casos-base:

1. o primeiro ocorre quando $m = 0$: não havendo mais tempo hábil, não é possível ensacar mangabas, logo $f_K(x, 0) = 0$.

2. O segundo caso-base ocorre quando $x \leq K$, ou seja, quando é possível ensacar todo o balde. Daí, $f_K(x, m) = x$ se $x \leq K$ e $m > 0$.

Se $x > K$, é preciso dividir x e distribuir o tempo restante. de modo que são m transições possíveis, dentre as quais devemos optar pela que maximiza o valor da função f_K:

$$f_K(x, m) = \max_{0 \leq i \leq m-1} f_K\left(\left\lfloor \frac{x}{2} \right\rfloor, i\right) + f_K\left(\left\lceil \frac{x}{2} \right\rceil, m - 1 - i\right)$$

Usando programação dinâmica, é possível computar f_k em $O(XM^2)$, pois são $O(XM)$ estados e cada transição é $O(M)$, onde X é o maior valor possível para a quantidade de mangabas em um balde.

A segunda parte da solução consiste em uma variante do problema da mochila: para cada balde i, deve-se optar por usar ou não este balde. Se o balde for utilizado, deve-se escolher quando tempo será dedicado ao balde e, usando a função $f_k(x_i, m_i)$, determinar qual é a melhor solução para o tempo m_i que será dedicado para processar as x_i mangabas. Esta segunda etapa também pode ser resolvida por programação dinâmica em $O(NM^2)$, pois são $O(NM)$ estados, com transição $O(M)$.

Resolvidas ambas etapas, resta apenas a reconstrução dos melhores caminhos em ambas etapas, o que pode ser feito por meio de tabelas auxiliares em $O(M)$.