Tutorial: Atunda Batata

Guilherme Ramos

Enquanto houver entradas, basta ignorar a primeira linha (é sempre o mesmo valor), mapear cada trecho dado na segunda linha ao respectivo autor e apresentá-lo.

Tutorial: Bombons

Edson Alves

Uma solução que responda cada uma das M perguntas em $O(N)$ terá complexidade $O(NM)$ e receberá veredito TLE.

Para resolver o problema com menor complexidade, primeiramente é preciso manter um histograma `hist` dos pacotes disponíveis, onde `hist[i] = j` indica que o pacote j contém i bombons. Se houverem dois ou mais pacotes com o mesmo número de bombons, basta registrar qualquer um deles no histograma. Se não há nenhum pacote com i bombons, faça `hist[i] = -1`.

A segunda parte da solução consiste em usar uma versão modificada do crivo de Erastótenes para identificar, para cada inteiro k, todos os seus múltiplos no histograma e registrar como solução para k o índice registrado no maior múltiplo m de k tal que `hist[m] != -1`. Esta etapa tem complexidade $O(B \log B)$, onde $B = \max(b_1, b_2, \ldots, b_N)$.

Após este pré-processamento, cada pergunta pode ser respondida em $O(1)$.

Tutorial: Cinuca Olímpica

Jeremias Moreira Gomes

Uma das soluções para o problema é feita em duas partes, para cada jogador, separadamente. Para resolver um jogador, iniclamente realiza-se uma busca binária, para localizar o índice onde G deveria ser inserido na lista de bolas do jogador. A partir dessa posição, olha-se as extremidades de B elementos à esquerda e B elementos à direita, verificando qual dos dois está mais próximo de G. O elemento mais próximo faz parte da solução e, na extremidade que está mais distante, deve-se iterar uma posição para mais próximo do índice central, até que todos os elementos sejam identificados.

Tutorial: Dupla de dois

Daniel Saad Nogueira Nunes

Uma forma simples de resolver esse problema é ordenar todas as duplas em ordem crescente de nota e, em seguida, parear:

- O aluno de maior nota com o de menor nota.
- O aluno de segunda maior nota com o de segunda menor nota.
- E assim por diante.

Isso pode ser feito em tempo $\Theta(n \lg n)$. Como as notas estão em uma faixa de $[0, 100]$, pode-se ainda aplicar métodos pseudolineares de ordenação, como Countingsort ou Bucketsort e obter uma complexidade $\Theta(n)$.

Tutorial: Encontros

Jeremias Moreira Gomes

A solução consiste em fracionar uma volta no relógio em $H-1$ iterações, pois como o horário começa a partir de `00:00`, a volta completa iria repetir um horário ao final do dia. Dessa forma, basta ir calculando a proporção a partir do quanto cada fatia da hora irá consumir em minutos e segundos. Considerando h, m, s como a proporção da divisão do relógio por horas, dá o seguinte:

$h = 1$

$m = \frac{M}{H-1}$

$s = \frac{S*M}{H-1}$

Por último, pode-se considerar as operações modulares acrescentando o resto da divisão da quantidade de segundos por fatia nas iterações seguintes.

Tutorial: Flores do Jardim

Caleb Martim

Vamos identificar cada componente conexo diferente por algum número inteiro único. Comece um contador c que representa o número de componentes contados até o momento. c inicialmente é 0. Defina também dois vetores $qtdVertices$ e $qtdArestas$ de modo que $qtdVertices[j]$ irá representar a quantidade de vértices contidos no componente j e $qtdArestas[j]$ irá representar a quantidade de arestas contidas no componente j. Todos os valores em $qtdVertices$ e $qtdArestas$ são inicialmente 0.

Itere por todo inteiro i entre 1 a N, caso o vértice i já não tenha sido visitado, aumente c por 1 e inicie uma busca em largura ou em profundidade a partir dele. O componente conexo que contém o vértice i será representado pelo valor atual de c. Para cada vértice u visitado na busca atual, aumente em 1 o valor contido em $qtdVertices[c]$ e para cada vértice v adjacente a u, aumente em 1 o valor contido em $qtdArestas[c]$. Após todos os vértices entre 1 a N terem sido visitados, podemos contar a quantidade de componentes em que sua quantidade de arestas é igual a quantidade de seus vértices subtraída por 1. Isto é, contar a quantidade de valores k entre 1 e C em que $qtdArestas[k]$ é igual a $qtdVertices[k] - 1$. Porém, como toda aresta em um grafo contém dois vértices, note que este algoritmo fará com que cada aresta do componente k seja contada duas vezes em $qtdArestas[k]$. Para corrigirmos isso, podemos simplesmente dividir esse número por 2 antes de nossa verificação. Isto é, $qtdArestas[k] := \frac{qtdArestas[k]}{2}$. E assim finaliza-se nossa solução.

Complexidade de tempo: $O(N + M)$ caso o grafo tenha sido implementado com listas de adjacência ou $O(N^2)$ caso tenha sido implementado com uma matriz de adjacência.

Complexidade de espaço: $O(N + M)$ caso o grafo tenha sido implementado com listas de adjacência ou $O(N^2)$ caso tenha sido implementado com uma matriz de adjacência.

Tutorial: Guloso da Cidade

Bruno Vargas

Para resolver esse problema, uma abordagem eficiente é utilizar $DeltaEncoding$ junto com $PrefixSum(Psum)$, o que permite calcular a "gostosura" acumulada em $O(MAXR)$. Neste método, incrementamos o valor de "gostosura" no início do intervalo L e decrementamos em $R + 1$, depois acumulamos os valores ao longo do bolo com uma soma prefixada. Outra abordagem viável é o uso de $SweepLine$, onde processamos os eventos de início e fim das coberturas (pontos L e $R+1$) em ordem, mantendo as coberturas ativas em uma estrutura de dados ordenada, para gerenciar e contar as coberturas presentes em cada ponto de corte. A complexidade dessa abordagem é $NlogN$, dada a necessidade de ordenar os eventos e manipular a estrutura de dados para maximizar a "gostosura" respeitando o limite K.

Tutorial: Homiranho

Guilherme Ramos

Leia duas linhas, ignore os conteúdos, e apresente as mensagens como indicadas nos exemplos.

Tutorial: Inversões Mágicas de Botelho

Alberto Neto

Represente cada intervalo (l_i, r_i) como uma string binária t de tamanho n tal que $t_j = 1$ se $l_i \leq j \leq r_i$, e $t_j = 0$ caso contrário. O valor 1 em alguma posição da string t significa que aplicar a operação correspondente inverte o valor naquela posição, e 0 significa que a operação não muda o valor.

Ao considerarmos duas operações representadas pelas strings t e s, o XOR (ou exclusivo) $x = t^s$ delas representa as inversões após aplicar as duas operações consecutivamente. Com isso, podemos reduzir o problema para: dadas q strings binárias de tamanho n, quantas strings diferentes podemos formar fazendo o XOR de 0 ou mais delas?

Considerando uma string binária de tamanho n como um vetor de tamanho n sobre o corpo de dois elementos, o conjunto de strings binárias com operação de XOR forma um espaço vetorial. Assim, a resposta é 2^b, onde b é o tamanho do maior subconjunto do conjunto de operações que é linearmente indepentente. O tamanho desse conjunto pode ser calculado usando o algoritmo de Gauss.

Tutorial: Jogos Universitários

Vinicius Borges

Para resolver esse problema, devemos fazer uma busca completa para determinar a quantidade mínima de competições necessárias para obter exatamente N medalhas de ouro respeitando-se as restrições do problema. Entretanto, o uso de uma abordagem por busca em profundidade clássica leva ao TLE por conta dos grandes valores de N, M e l_i.

Para passar uma solução no limite de tempo de 1 segundo, pode-se utilizar uma abordagem baseada em programação dinâmica, em que podemos utilizar uma tabela para armazenar resultados intermediários para evitar o cálculo repetido de subproblemas. A tabela é modelada para armazenar a quantidade mínima de modalidades necessárias para calcular uma determinada quantidade de medalhas de ouro, de zero até N. Isso significa que cada valor i na tabela representa a menor quantidade de modalidades que podemos usar para conseguir i medalhas de ouro.

O algoritmo é fornecido a seguir:

1. INFINITO $= 2 \cdot 10^9$
2. $dp[i] =$ INFINITO para cada $i = 1, \ldots, 1001$
3. Inicialize $dp[0] = 0$ (zero competições resultam em zero medalhas)
4. Para cada $i = 0, \ldots, M - 1$
   - Para cada $j = N, \ldots, 0$
     (a) Para cada $k = 1, \ldots, l[i]$ e respeitando-se $k \cdot o[i] + j \leq N$
       - Se $dp[i] \neq$ INFINITO, faça $dp[j + k \cdot o[i]] = \min(dp[j + k \cdot o[i]], dp[j] + k)$;

Ao final, imprima $dp[N]$ se $dp[N] \neq$ INFINITO. Caso contrário, imprima -1.

A complexidade desse algoritmo no tempo é, no pior caso, $O(N \cdot M \cdot L)$, em que $L = \max\{l[i] | 0 \leq i \leq M - 1\}$.

Tutorial: Kátia e os fatoriais

Edson Alves

Seja n um inteiro positivo e p um número primo. A função

$$E_p(n) = \left\lfloor \frac{n}{p} \right\rfloor + \left\lfloor \frac{n}{p^2} \right\rfloor + \left\lfloor \frac{n}{p^3} \right\rfloor + \ldots + \left\lfloor \frac{n}{p^r} \right\rfloor,$$

onde $\frac{n}{p^{r+1}} = 0$, retorna a maior potência p^k de p que divide $n!$. Como $E_2(n) \geq E_5(n)$, para todo n inteiro positivo, e $10 = 2 \times 5$, o número de zeros à direita na representação de $n!$ é dada por $E_5(n)$.

Como $E_p(n)$ é não-decrescente, a equação

$$E_5(n) = M$$

pode ser resolvida por meio de uma busca binária na resposta, de modo que a solução tem complexidade é logarítimica no tamanho do intervalo de busca, cujo valor máximo, para a entrada do problema, é menor do que 5×10^9.

Tutorial: Lago Oeste

Daniel Saad Nogueira Nunes

Esse problema pode ser resolvido com uma técnica chamada *flood-fill* que nada mais é que uma busca em profundidade e é similar ao algoritmo de pintar uma região no programa `paint`. Ela consiste em percorrer a matriz e, se a célula não foi visitada, aplicar uma busca em profundidade olhando para as células que não foram visitadas ainda, marcando-as como visitada, desde que elas possuam o mesmo símbolo da célula onde se iniciou a busca. Nesse percurso da matriz, sempre que encontramos um símbolo `F`, incrementamos um contador. A complexidade dessa solução é $\Theta(N \cdot M)$.

Tutorial: Mediação

Jeremias Moreira Gomes

Uma solução para o problema consiste em calcular o vetor do prefix-sum do carisma dos candidatos nas duas direções (do início para o fim, e do fim para o início). Se em algum índice o valor do prefix sum for igual, está é a resposta válida para o problema.