Tutorial: Academia

Enzo Niho

A questão é bem simples, basta ler os três inteiros e imprimir a soma deles.

Tutorial: Bola Quadrada

Daniel Saad Nogueira Nunes

Enumerar o espaço de busca de maneira eficiente faz com que o problema possa ser resolvido no tempo limite.

Para cada célula vazia:

- Inserimos um número no intervalo de $[1, 9]$, desde que ele seja compatível.
- Chamamos o procedimento recursivamente para a próxima célula vazia.
- Na volta da recursão, atribuímos à célula o valor 0 para que o *backtracking* possa ser realizado.

Um número em uma célula é dito **compatível** desde que ele seja diferente de todos os outros em sua linha ou coluna. Testar pela compatibilidade do número antes de realizar a chamada recursiva reduz significativamente o espaço de busca.

Tutorial: Caleb, Chefe Competente

Alberto Neto

Solução:

Monte a árvore de hierarquias e escreva uma função recursiva $f(u)$ que retorna um booleano: 1 se algum funcionário na subárvore de u foi chamado inicialmente para a reunião, ou 0 caso contrário. Diremos que v é filho direto de u se u é chefe direto de v.

Seja u um funcionário. Então u será chamado para a reunião se, e somente se:

- u foi um dos convidados iniciais para a reunião, ou
- existem filhos distintos v_j, v_l de u tais que $f(v_j) = f(v_j) = 1$. Em outras palavras, existem pelo menos dois filhos de u tais que alguém foi convidado na subárvore de ambos.

A própria função recursiva f pode também calcular a resposta:

```
int f(int u) {
    int filhos_com_convidado = 0;
    for(auto v : g[u]) {
        if(f(v)) filhos_com_convidado++;
    }
    if(filhos_com_convidado >= 2 or inicial[u] == 1) resposta.push_back(u);
    filhos_com_convidado += pega[u]; // considera para cima se o proprio u foi convidado ou não

    return filhos_com_convidado >= 1;
}
```

Tutorial: Daniel Triste

Guilherme Ramos

Basta ler a mensagem e, havendo uma ocorrência de “Daniel Sad” (independente da caixa), dobrar a letra ‘a’ (ou ‘A’) do sobrenome.

Tutorial: Evidências de um Merge

Daniel Porto

Fazer o merge de forma eficiente, aproveitando o fato de que os arrays estão ordenados. Basta usar o mesmo procedimento de junção do Mergesort, mas desconsiderando elementos repetidos. Evite estruturas que resultem em tempo acima de $O(n + m)$.

Tutorial: Fernando, Francisco e Frações

Maxwell Oliveira

Para resolver este problema podemos, primeiro, quebrar a string em tokens. Ou seja, se o input for “1/2 + 2/5 * 3/7” podemos dividi-la em [“1/2”, “+”, “2/5”, “*”, “3/7”]. Com isso, agora podemos processar uma operação por vez (respeitando sua ordem). Aplicando primeiramente a multiplicação, nossos tokens se tornam [“1/2”, “+”, “6/35”] e, após aplicar a soma, [“47/70”].

Importante: lembre-se de, no final, simplificar sua fração ao máximo (dividindo numerador e denominador pelo seu máximo divisor comum) e de garantir que o denominador seja positivo.

Caso implemente em Python, as funções split” e math.gcd” podem ser bem úteis.

Tutorial: Galinhas Globais

Daniel Saad Nogueira Nunes

Basta fazer várias estruturas condicionais comparando o nome lido com os possíveis países e imprimir a pronúncia adequada. Deve-se ter cuidado na hora de imprimir a pronúncia de Israel sem acento: `tarnegolet`.

Tutorial: Hanimeitor

Jeremias Moreira Gomes

Sabendo-se que o efeito e o relógico começam sincronizados, e chamando-se de T_0 e T_f os tempos de início e término da competição em milissegundos; a quantidde Q de vezes que ambos irão sincronizar é dada por:

$$Q = \left\lceil \frac{T_f - T_0}{LCM(S * 1000, P)} \right\rceil$$

onde LCM é o Mínimo Múltiplo Comum.

Tutorial: Incríveis Permutações

Felipe Louza

Análise do Problema

Queremos encontrar uma permutação $P = (p_1, p_2, \ldots, p_N)$ dos números de 1 a N tal que $|p_i - p_{i+1}| \neq 1$ para todo i.

Vamos analisar:

- Para $N = 1$: trivial, resposta é 1.
- Para $N = 2$ ou $N = 3$: impossível.
- Para $N \geq 4$: é possível construir uma solução.

Ideia da Solução

A solução é simples: separar os números pares e ímpares. Se imprimirmos todos os pares primeiro, seguidos dos ímpares (ou vice-versa), garantimos que entre quaisquer dois números consecutivos haverá diferença de pelo menos 2.

Exemplo para $N = 5$:

Pares: 2 4 Ímpares: 1 3 5

Uma resposta possível: 2 4 1 3 5

Nota: A ordem dentro dos pares e ímpares pode variar, desde que a separação mantenha a diferença entre consecutivos maior que 1.

Implementação em C++

Abaixo está uma solução eficiente em C++:

```cpp
#include <iostream>
using namespace std;

int main() {
    int n;
    cin >> n;

    if (n == 2 || n == 3) {
        cout << "NO SOLUTION" << endl;
        return 0;
    }

    // Imprime os pares primeiro
    for (int i = 2; i <= n; i += 2) {
        cout << i << " ";
    }
```

```
    // Depois os ímpares
    for (int i = 1; i <= n; i += 2) {
        cout << i << " ";
    }

    cout << endl;
    return 0;
}
```

Complexidade

- Tempo: $\mathcal{O}(N)$
- Espaço: $\mathcal{O}(1)$ (sem uso adicional de memória)

Conclusão

Este problema é uma boa introdução à ideia de construção de permutações com restrições simples. A chave está em identificar um padrão que evita os conflitos, neste caso, a separação entre pares e ímpares.

Tutorial: Jira Jira

Enzo Niho

É possível resolver a questão ao comparar os senos dos dois ângulos.

Tutorial: KMP

Edson Alves

A função $\rho(n)$ pode ser computada em $O(\sqrt{n})$. Computar esta função para $O(N)$ inteiros tem custo $O(N\sqrt{M})$, onde $M = \max(x_1, x_2, \ldots, x_N, x_S)$, o que leva ao veredito TLE.

É possível computar a função $\rho(n)$ para todos os inteiros do intervalo $[1, M]$ de forma eficiente, por meio de uma modificação do crivo de Erastótentes. Inicialize um vetor `r` de tamanho $M + 1$ com zeros. Agora, para cada $i = 2, 3, \ldots, M$, se `r[i] == 0`, então i é primo: para todos os múltiplos m de i (ele inclusive), incremente o valor de `r[m]`. Desta forma, teremos $\rho(n) =$ `r[n]` para todos $n \in [1, N]$, e esta rotina tem complexidade $O(M \log M)$.

Sejam a, b dois inteiros positivos e P_A, P_B os conjuntos dos primos que dividem a e b, respectivamente. Daí

$$\rho(ab) = |P_A| + |P_B| - |P_A \cap P_B|$$

Sabemos que $|P_A| = \rho(a)$ e que $|P_B| = \rho(B)$. Observe que, se q é um primo que divide a e também divide b, então q divide d, onde d é o máximo divisor comum de a e b. Assim, $|P_A \cap P_B| = \rho(d)$. Avaliando $\rho(x_S x_j)$ para $j = 1, 2, \ldots, N$, é possível determinar a resposta do problema em $O(N + M \log M)$.

Tutorial: Leila, a CabeLeila

Enzo Niho

Como o N é pequeno, basta fazer uma busca completa pela melhor resposta. Caso o N fosse maior, seria necessário uma abordagem diferente, como o uso de programação dinâmica.

Tutorial: Music Tour

Caleb Martim

1 Solução 2

É fácil observar que a ordem de visitação que minimiza o número de visitações para todo vértice na árvore ser visitado pelo menos uma vez segue a ordem de busca em profundidade (DFS) na árvore. É possível ver que toda aresta da árvore é utilizada exatamente duas vezes por Chuu, uma vez para descer e outra para subir. Então, nesta solução, temos que saber, primeiramente, o número de arestas da árvore, como ela é muito grande, não podemos gerar ela sem exceder os limites de tempo ou de memória. Observe que por se tratar de uma árvore, o número de arestas é determinado pelo número de vértices, se a árvore tem x vértices, ela terá $x - 1$ arestas; então, vamos, na verdade, calcular o número de vértices da árvore e subtrair 1 disso para obtermos o número de arestas.

Seguindo a definição de uma árvore binária perfeita, o primeiro nível terá apenas um vértice; daí, como cada vértice em um nível $L < N$ tem exatamente dois filhos, o número de vértices do nível $L+1$ é exatamente o dobro do número de vértices do nível L. Assim, é possível observar que o número de vértices de uma árvore binária perfeita de profundidade N é definida pela soma:

$$2^0 + 2^1 + 2^2 + 2^3 + ... + 2^{N-1}$$

É uma propriedade conhecida da matemática que

$$\sum_{i=0}^{k} 2^i = 2^{k+1} - 1$$

Portanto, o número de vértices da árvore é $2^N - 1$ e, dessa forma, podemos concluir que o número de arestas da árvore é $2^N - 2$. Como afirmado, cada aresta será passada exatamente duas vezes, então vamos multiplicar esse número obtido por dois. Por fim, é necessário somar esse número por 1, para considerarmos o momento que o vértice 1 é visitado pela primeira vez (O momento que Chuu chega no aeroporto). Dado tudo isso, podemos afirmar que a resposta do problema é igual a:

$$2 \cdot (2^N - 2) + 1 \mod 10^9 + 7$$

O que pode ser calculado em $O(N)$ ou em $O(\log(N))$ com exponenciação rápida.