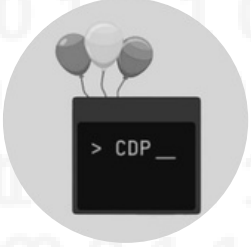
Maratona de Programação

2026.1

Caderno de soluções: Beginner

Organização:

Maratona
</>PPCi



Clube de Programação
(UTFPR Curitiba)

Problema A. Ajude N

Tempo limite: 1000 ms
Memória limite: 256 MiB
Autor: Rafael Tomazini Marani

Solução

O problema se resume a encontrar o tamanho do complemento da componente conexa que contém P. Primeiramente construa o grafo não-direcionado com Q vértices e L arestas. Em seguida execute um algoritmo de BFS (*Breadth-first search*) ou DFS (*Depth First Search*), a partir do vértice P, marcando todos os vértices alcançáveis. A resposta é $Q - |\text{componente de } P|$.

Referência para a BFS: https://cp-algorithms.com/graph/breadth-first-search.html.

Referência para a DFS: https://cp-algorithms.com/graph/depth-first-search.html.

Complexidade: $O(Q + L)$.

Problema B. Bandeirinhas

Tempo limite: 1000 ms
Memória limite: 256 MiB
Autor: Caio Welter Bogo

Solução

O problema consiste em verificar se existe um arranjo intercalado de três tipos de elementos. A condição necessária e suficiente para que uma sequência válida exista é que nenhuma cor possua mais bandeirinhas do que a soma das outras duas mais um, ou seja:

$$\max(A, V, Z) \leq \lfloor (A + V + Z + 1)/2 \rfloor$$

Solução gulosa:

1. Verifique a condição acima. Se não for satisfeita, imprima `F`.
2. Caso contrário, construa a sequência usando uma fila de prioridade (max-heap): em cada passo, insira a bandeirinha da cor mais frequente, desde que ela seja diferente da última inserida. Caso a cor mais frequente seja igual à última, insira a segunda mais frequente.
3. Repita até inserir todas as bandeirinhas.

Complexidade: $O(A + V + Z)$.

Problema C. Contornar Multidão

Tempo limite: 1000 ms
Memória limite: 256 MiB
Autor: Gabriel Muller

Solução

A solução consiste em executar um algoritmo de BFS (*Breadth-First Search*) na Grid $N \times M$ a partir do ponto de inicio e retornar a menor distância para o estádio, que é igual a quantidade mínima de tempo.

Referência para a BFS: https://cp-algorithms.com/graph/breadth-first-search.html.

Complexidade: $O(N \times M)$

Problema D. Duplicatas

Tempo limite: 1000 ms
Memória limite: 256 MiB
Autor: Luiz A Scheeren

Solução

Como os IDs das figurinhas estão limitados entre 1 e 1000, a solução ideal utiliza Vetores de Frequência.

Contabilizamos a quantidade de cada figurinha que Ricardo e Daniel possuem. Uma figurinha i é considerada uma opção de troca válida se:

- **Para Ricardo:** ricardo[i] > 1 e daniel[i] $== 0$
- **Para Daniel:** daniel[i] > 1 e ricardo[i] $== 0$

Como cada troca necessita de uma figurinha de cada lado, o número máximo de trocas é determinado pelo limitante mínimo entre as opções de ambos:

$$\text{Total de Trocas} = \min(\text{opcoes\_ricardo}, \text{opcoes\_daniel})$$

Complexidade: $\mathcal{O}(N + M + K)$

Onde K é o tamanho do vetor de frequência (1000).

Problema E. Encontre o 'vencedor'

Tempo limite: 1000 ms
Memória limite: 256 MiB
Autor: Felipe Alberto

Solução

A principal dificuldade do problema é manter as estatísticas de cada time de forma eficiente.

Para cada time, precisamos armazenar seu nome, sua quantidade gols e sua quantidade de vitórias, que pode ser feito usando uma struct ou um vetor de tuplas por exemplo.

Como as partidas referenciam os times pelos seus nomes, uma busca linear para encontrar as estatísticas de cada equipe tornaria a solução muito lenta. Portanto, utilizamos uma estrutura de busca eficiente, como `map`, associando o nome do time às suas estatísticas.

Inicialmente, lemos os N nomes e criamos uma entrada para cada time. Em seguida, processamos as $N - 1$ partidas. Para cada partida:

1. adicionamos os gols marcados por cada equipe ao seu total;
2. incrementamos o número de vitórias do vencedor.

Após processar todas as partidas, copiamos as estatísticas para um vetor e o ordenamos utilizando os critérios especificados no enunciado. Por fim, imprimimos os nomes dos times na ordem obtida.

Se utilizarmos um `map`, cada acesso possui custo $O(\log N)$.

Como processamos $N - 1$ partidas e realizamos apenas um número constante de operações por partida, o custo total dessa etapa é: $O(N \log N)$.

A ordenação final de N elementos também possui complexidade: $O(N \log N)$

Portanto, a complexidade total da solução é: $O(N \log N)$

Problema F. Folga Frustrada

Tempo limite: 1000 ms
Memória limite: 256 MiB
Autor: Caio Welter Bogo

Solução

Este é o clássico Problema de Seleção de Atividades (*Interval Scheduling*).

A abordagem correta e ótima utiliza um Algoritmo Guloso (*Greedy*). A solução consiste em:

1. Armazenar as requisições em pares de $(Fim, Inicio)$.

2. Ordenar todos os intervalos de forma crescente baseando-se estritamente no **horário de término**.

3. Selecionar o primeiro intervalo da lista ordenada e iterar pelos próximos. Sempre que o horário de início do intervalo atual for maior ou igual ao horário de término do último intervalo selecionado, ele é adicionado à resposta e o limite de término é atualizado.

Referência para o problema *Interval Scheduling*:

https://www.geeksforgeeks.org/dsa/scheduling-in-greedy-algorithms/.

Complexidade: $O(N \log N)$

Problema G. Gols do Galvão

Tempo limite: 1000 ms
Memória limite: 256 MiB
Autor: Erick Rosim

Solução

Apenas some $A + B$

Complexidade: $O(1)$

Problema H. Habilidades Futebolísticas

Tempo limite: 1000 ms
Memória limite: 256 MiB
Autor: Eduardo Vinicius Marca

Solução

Como cada habilidade tem no máximo um pré-requisito e não há contradições entre os pré-requisitos (não há ciclos), podemos modelar o problema como uma floresta de árvores, onde cada habilidade é um nó e cada pré-requisito é uma aresta direcionada do nó pré-requisito para o nó da habilidade. Assim cada raiz da árvore é uma habilidade que não possui pré-requisitos.

Além disso, podemos criar um novo nodo que chamaremos de 0, e ligar esse nodo a todas as raízes da floresta, formando uma única árvore. Dessa forma, o problema se reduz a contar o número de maneiras de percorrer essa árvore respeitando a ordem das dependências. Isso é equivalente ao problema original, apesar de não ser necessário para a solução.

A solução envolve calcular o número de maneiras de percorrer a árvore a partir de um nodo a partir do número de maneiras de percorrer cada subárvore desse nodo.

Para isso podermos calcular para cada nodo v o tamanho da subárvore a partir daquele nodo, que chamaremos de $size[v]$, e o número de maneiras de percorrer a subárvore a partir daquele nodo, que chamaremos de $dp[v]$.

Podemos calcular $size[v]$ de forma recursiva, somando 1 (o próprio nodo) com o tamanho das subárvores de cada filho de v. É possível usar uma DFS, por exemplo.

Para calcular $dp[v]$, podemos utilizar um coeficiente multinomial para contar as formas de intercalar as subárvores do nodo v. Podemos usar a seguinte fórmula:

$$dp[v] = \prod_{u \in filhos(v)} dp[u] \times \binom{sz[v]-1}{sz[u_1], sz[u_2], \ldots, sz[u_k]}$$

Complexidade

Cada nodo é visitado uma vez, e para cada nodo visitamos todos os seus filhos para cada vetor de dp, portanto a complexidade é $O(N)$.

A complexidade de espaço é $O(N)$, para armazenar a árvore e os vetores $size$ e dp.

Problema I. Igualdade de Figurinhas

Tempo limite: 1000 ms
Memória limite: 256 MiB
Autor: Gabriel Müller

Solução

Seja S a soma dos *scores* de todas as figurinhas. Para que seja possível dividir as figurinhas em dois grupos com a mesma soma, cada grupo deve possuir soma igual a $S/2$.

O primeiro caso a ser tratado é quando S é ímpar. Nesse cenário, não existe maneira de dividir as figurinhas em dois grupos de mesma soma, portanto a resposta é imediatamente Nao.

Caso S seja par, o problema passa a ser descobrir se existe algum subconjunto de figurinhas cuja soma seja exatamente $S/2$. Se tal subconjunto existir, as figurinhas restantes também terão soma $S/2$, tornando a divisão possível.

Para isso, utilizamos programação dinâmica. Defina um vetor `dp`, onde `dp[x]` indica se é possível obter uma soma igual a x utilizando algumas das figurinhas já processadas.

Inicialmente, apenas a soma zero é possível, pois basta não escolher nenhuma figurinha:

$$dp[0] = \texttt{true}.$$

Em seguida, para cada figurinha de valor v_i, atualizamos o vetor de trás para frente. Se antes era possível formar a soma $j - v_i$, então também passa a ser possível formar a soma j utilizando essa figurinha.

Percorrer o vetor de trás para frente é essencial para garantir que cada figurinha seja utilizada no máximo uma vez.

Ao final do processamento, basta verificar o valor de `dp[S/2]`.

- Se `dp[S/2]` for verdadeiro, existe um subconjunto cuja soma é $S/2$, logo a resposta é Sim.
- Caso contrário, a resposta é Nao.

Complexidade:$O(N \cdot S)$, onde S é a soma dos scores das figurinhas.

Problema J. Juiz Carrasco

Tempo limite: 1000 ms
Memória limite: 256 MiB
Autor: Ricardo Oliveira

Solução

Há 22 jogadores no início do jogo (11 para cada time). Durante o jogo, saíram de campo A de um time e B do outro. Logo, a resposta é $22 - A - B$.

Complexidade: $O(1)$

Problema K. Kurt e o maldito VAR

Tempo limite: 1000 ms
Memória limite: 256 MiB
Autor: Erick Rosim

Solução

Primeiramente, ordenamos todos os minutos em que houve paralisação. Além disso, adicionamos um ponto fictício na posição $N + 1$, que representa o final da partida. Dessa forma, qualquer intervalo livre entre duas paralisações consecutivas pode ser calculado facilmente.

Percorremos as paralisações ordenadas da direita para a esquerda. Seja x_i a paralisação atual e x_{i+1} a próxima paralisação à direita. O maior intervalo contínuo criado pela remoção de x_i possui tamanho

$$x_{i+1} - x_i - 1,$$

pois os minutos correspondentes às próprias paralisações não fazem parte do intervalo jogado.

Durante esse percurso, mantemos a variável `maior`, que armazena o maior intervalo encontrado até o momento. Em cada passo fazemos

$$\texttt{maior} = \max(\texttt{maior},\, x_{i+1} - x_i - 1),$$

e atribuímos esse valor como resposta da paralisação correspondente.

Como as paralisações foram ordenadas apenas para facilitar o processamento, armazenamos junto de cada minuto sua posição original na entrada. Assim, após calcular todas as respostas, basta imprimi-las na ordem original.

Complexidade: $O(Q \log Q)$.

Problema L. Logistica da Escalação

Tempo limite: 1000 ms
Memória limite: 256 MiB
Autor: João Tanaka

Solução

O único caso impossível é $R = C = 1$, pois o MDC da única linha e o MDC da única coluna são iguais ao mesmo elemento.

Se $R = 1$ e $C > 1$, podemos imprimir a linha

$$2, 3, 4, \ldots, C + 1.$$

O MDC da linha será 1, enquanto cada coluna terá assinatura igual ao seu único elemento.

O caso $C = 1$ e $R > 1$ é análogo: imprimimos os valores $2, 3, \ldots, R + 1$, um por linha.

Para $R > 1$ e $C > 1$, escolha

$$L_i = i + 2$$

para as linhas, com i começando em 0, e

$$K_j = R + j + 2$$

para as colunas. Então imprima

$$a_{i,j} = L_i \cdot K_j.$$

Em uma linha fixa i, todos os valores têm fator L_i, e os fatores restantes são inteiros consecutivos. Como há pelo menos dois deles, seu MDC é 1, então o MDC da linha é exatamente L_i. O mesmo argumento vale para as colunas.

Os valores L_i pertencem a $2, 3, \ldots, R{+}1$, enquanto os valores K_j pertencem a $R{+}2, R{+}3, \ldots, R{+}C{+}1$. Logo, todas as assinaturas são distintas.

O maior valor impresso pela construção é $(R + 1)(R + C + 1) \leq 501 \cdot 1001$, dentro do limite de 10^6.

Complexidade: $O(RC)$.