Maratona de Programação

2026.1

Caderno de soluções: Standard

Organização:

Maratona
</>PPCi

Clube de Programação
(UTFPR Curitiba)

Problema A. Anceloti e Endrick

Tempo limite: 1000 ms
Memória limite: 256 MiB
Autor: Henrique Farias

Solução

A ideia é armazenar apenas os intervalos de atividades que Endrick ainda gosta. Inicialmente existe apenas um intervalo:

$$[1, n].$$

Além disso, mantemos uma variável `ans`, que representa a quantidade total de atividades de que Endrick gosta. Inicialmente,

$$\texttt{ans} = n.$$

Para cada operação sobre o intervalo $[l, r]$, percorremos todos os intervalos que intersectam esse trecho.

Se uma parte do intervalo permanecer fora da atualização, ela continua sendo um intervalo válido. Já a parte que pertence ao intervalo atualizado é removida, fazendo com que `ans` seja decrementado pela quantidade de atividades removidas.

Caso a operação seja do tipo $k = 2$, após remover todas as interseções basta adicionar o intervalo $[l, r]$ novamente, incrementando `ans` pelo tamanho desse intervalo.

Como sempre mantemos exatamente os intervalos das atividades de que Endrick gosta, o valor de `ans` corresponde à resposta após cada operação.

Complexidade: $O(Q \log Q)$

Problema B. Bola Passada

Tempo limite: 500 ms
Memória limite: 256 MiB
Autor: Eduardo Marca

Solução

Primeiro, uma observação importante. Durante toda a execução, os jogadores que já receberam a bola sempre formam um intervalo contínuo do círculo. Isso ocorre porque um novo jogador só pode receber a bola através de um de seus dois vizinhos. Assim, sempre que um novo jogador é descoberto, ele necessariamente é adjacente a uma das extremidades do intervalo já visitado.

Para resolver o problema, vamos provar que a probabilidade de que o último jogador a receber a bola seja um jogador específico A é a mesma para todos os jogadores A diferentes de K. Para o jogador K, a probabilidade é zero, pois ele já começou com a bola.

Para isso, vamos chamar a probabilidade de que o último jogador (diferente de K) a receber a bola seja A de p_A. Para que um jogador A seja o último a receber a bola, precisamos que primeiro que a bola chegue a um dos vizinhos de A (esquerdo ou direito), e então de a volta por todos os outros jogadores até que A seja o último a receber a bola.

Assim podemos escrever a seguinte equação para p_A:

$$p_A = p_e * p_v + p_d * p_v = (p_e + p_d) * p_v$$

onde p_e é a probabilidade de que a bola chegue primeiro ao vizinho esquerdo de A, p_d é a probabilidade de que a bola chegue primeiro ao vizinho direito de A, e p_v é a probabilidade de que, uma vez que a bola chegou a um dos vizinhos, ela percorra todos os outros jogadores antes de chegar em A. Perceba que p_v é a mesma independente de qual vizinho a bola chegou primeiro.

Além disso, $p_e + p_d = 1$, pois a bola necessariamente chegará primeiro a um dos vizinhos de A, antes de chegar em A. Assim

$$p_A = p_v$$

Porém p_v é a mesma para todos os jogadores A diferentes de K, pois a bola precisa percorrer todos os outros jogadores antes de chegar em A, e isso independe de qual jogador A estamos considerando.

Portando, p_A é a mesma para todos os jogadores A diferentes de K:

$$p_A = p_B \quad \forall A, B \in \{1, 2, \ldots, N\} \setminus \{K\}$$

Como a soma das probabilidades de que cada jogador diferente de K seja o último a receber a bola é 1, temos que

$$\sum_{A \in \{1,2,\ldots,N\} \setminus \{K\}} p_A = 1$$

Como existem $N - 1$ jogadores diferentes de K, temos que

$$(N-1) * p_A = 1 \implies p_A = \frac{1}{N-1}$$

Sabendo que $p_A = \frac{1}{(N-1)}$ para toda posição A diferente de K, a probabilidade de que o último jogador a receber a bola pertença ao intervalo S é simplesmente o número de jogadores diferentes de K no intervalo multiplicado por p_A.

Assim a respota final é $\frac{|S \setminus K|}{N-1}$

Cada caso de teste é resolvido em tempo constante a complexidade é $O(1)$ utilizando apenas memória constante.

Problema C. Campeões da Copa

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

Solução

Ideia central (truque dos fantasmas). Quando dois torcedores colidem e invertem o sentido, o efeito sobre o *conjunto* de posições ocupadas é idêntico ao de dois "fantasmas" que simplesmente atravessam um ao outro. Ou seja, o multiconjunto de trajetórias é o mesmo de n pontos andando em linha reta, sem colisões. O mesmo vale para as saídas pelos portões: o conjunto de eventos de saída é preservado.

Assim, para cada torcedor i calcule a posição final do fantasma correspondente:

$$g_i = x_i + d_i \cdot T, \quad d_i = +1 \text{ se } \texttt{D},\ -1 \text{ se } \texttt{E}.$$

Se $g_i \le 0$, esse fantasma saiu pela **esquerda**; se $g_i \ge M$, saiu pela **direita**; caso contrário ele está na posição g_i.

Reatribuição pela ordem. Como os torcedores reais nunca se cruzam (eles quicam um no outro), a ordem relativa entre eles é **constante**. Logo, ordenando os torcedores pela posição inicial, o k-ésimo torcedor da esquerda para a direita recebe o k-ésimo menor desfecho: sejam ℓ a quantidade de fantasmas com $g_i \le 0$ e r a quantidade com $g_i \ge M$. Os ℓ torcedores mais à esquerda saem pela `ESQUERDA`, os r mais à direita saem pela `DIREITA`, e os do meio recebem, em ordem, as posições internas ordenadas crescentemente.

Complexidade: $O(n \log n)$ por causa das ordenações.

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. El Classico

Tempo limite: 1000 ms
Memória limite: 256 MiB
Autor: Henrique Farias

Solução

Um conjunto de pontos pode ser coberto por um círculo de raio R se o menor círculo que contém todos esses pontos (*Minimum Enclosing Circle*) possuir raio menor ou igual a R. Assim, percorremos todos os subconjuntos possíveis de jogadores e verificamos quais deles podem ser cobertos por um único círculo de raio R. Como $N \leq 18$, podemos utilizar bitmask para representar subconjuntos de pontos.

Depois disso, pode-se utilizar programação dinâmica para calcular a menor quantidade de círculos necessária para cobrir todos os jogadores. Para cada subconjunto de jogadores, tentamos escolher um grupo válido que possa ser coberto por um único círculo e adicionamos 1 à resposta do restante dos jogadores. A resposta final será o menor número de círculos necessário para cobrir todos os pontos (jogadores) do plano.

Referência para o MEC: https://cp-algorithms.com/geometry/enclosing-circle.html

Complexidade: $O(3^N)$

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. Grande 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 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. Jornada até a Taça

Tempo limite: 2000 ms
Memória limite: 256 MiB
Autor: Joao Vitor Tanaka

Solução

A rota que minimiza o maior caos entre dois vértices é um *caminho de gargalo mínimo*. Vale o teorema: o gargalo mínimo entre s e t é igual à **maior aresta no caminho entre s e t dentro da árvore geradora mínima (MST)** do grafo — toda MST é também uma árvore de gargalos mínimos (argumento de troca pela propriedade do corte).

O algoritmo tem três passos: (1) construir a MST com Kruskal usando *union-find*, ordenando as rodovias por caos crescente; (2) enraizar a MST com uma BFS, transformando o problema em "maior aresta no caminho entre dois nós de uma árvore"; (3) responder cada consulta com **binary lifting**, guardando, junto de cada ancestral 2^k, o **maior peso de aresta** naquele salto — o máximo no caminho s–t sai combinando os saltos até o LCA.

Se s e t caem em componentes diferentes (árvores distintas da floresta), a resposta é -1.

Complexidade: $O((N + M + Q) \log N)$.

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. Lado ou Bola

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

Solução

Note que o grafo dado é direcionado e acíclico (um DAG), e que as moedas são independentes entre si. Sob essas condições, este problema pode ser resolvido com a clássica teoria dos jogos dos *Grundy numbers*.

Primeiramente, associe a cada vértice v um inteiro $g(v)$ (dito *grundy number* do vértice) da seguinte maneira:

- $g(v) = 0$ se não há conexões saindo de v;
- $g(v) = mex\{g(v_1), g(v_2), ..., g(v_{d(v)})\}$ onde v_1, v_2, ..., $v_{d(v)}$ são os vértices para os quais há conexão direta saindo de v (os "vizinhos" de v), e mex é a operação de *menor-excluido*, isto é, o menor inteiro não negativo que *não* aparece no conjunto dos *Grundy numbers* dos "vizinhos" de v.

Lembre de utilizar alguma forma de memorização, Programação Dinâmica ou ordenação topológica para computar os valores de $g(v)$ de maneira eficiente.

Agora, sejam m_1, m_2, ..., m_K os vértices nos quais as moedas estão inicialmente. Seja $X = g(m_1) \oplus g(m_2) \oplus ... \oplus g(m_K)$, isto é, X é o XOR (ou-exclusivo) *bit-a-bit* dos *grundy numbers* destes vértices. Você pode vencer se e somente se $X \neq 0$.

O enunciado garante, em outras palavras, que $X \neq 0$ inicialmente. Além disso, pela teoria dos *Grundy numbers*, é garantido que, sempre que $X \neq 0$, existe uma jogada possível que faz X se tornar 0 após ela (e, logo, coloca o adversário em uma configuração perdedora). É esta a jogada que você deve fazer em todas as rodadas do jogo para garantir sua vitória.

Seja i o índice do *bit* mais significativo de X que é igual a 1, e seja m_j um vértice contendo uma moeda tal que o $i-$ésimo *bit* de $g(m_j)$ é 1 (certamente existe sob as condições dadas). Agora, seja w um "vizinho" de m_j tal que $g(w) = X \oplus g(m_j)$ (certamente existe sob as condições dadas). Faça a jogada que move uma moeda de m_j para w.

Fazendo esta jogada, o valor de X se torna $X = X \oplus g(m_j) \oplus g(w) = X \oplus g(m_j) \oplus (X \oplus g(m_j))$ $= (X \oplus X) \oplus (g(m_j) \oplus g(m_j)) = 0 \oplus 0 = 0$, como desejado.

Não importa qual jogada o adversário irá fazer em seguida pois, após ela, X certamente voltará a ser diferente de 0. Assim, repita esta estratégia até o fim do jogo.

Para entender melhor sobre a teoria dos *Grundy Numbers*, você pode ler https://cp-algorithms.com/game_theory/sprague-grundy-nim.html .

A solução do autor tem complexidade $O((N + M) \lg N)$ para a computação dos *Grundy Numbers*, e $O(K) + O(d(m_j))$ para cada jogada. Note que, ao todo, cada vértice será escolhido como m_j no máximo K vezes, e $O(\sum_v K \times d(v)) = O(K \times \sum_v d(v)) = O(K \times M)$.

Problema M. Metades Iguais

Tempo limite: 2000 ms
Memória limite: 256 MiB
Autor: João Tanaka

Solução

Considere um grafo bipartido em que há um vértice para cada conjunto e um vértice para cada valor distinto. Para cada ocorrência de um valor v no conjunto i, adicione uma aresta entre o vértice do conjunto i e o vértice do valor v.

Cada vértice de conjunto tem grau par, pois todo conjunto tem tamanho par. Assim, uma distribuição existe se, e somente se, todo valor aparece um número par de vezes — ou seja, todo vértice de valor também tem grau par.

Quando todos os graus são pares, cada componente do grafo admite um circuito euleriano. Como o grafo é bipartido, esse circuito tem comprimento par; percorrendo-o e colorindo as arestas alternadamente com E e D, cada vértice recebe metade das arestas de cada cor. Isso satisfaz, ao mesmo tempo, a divisão de cada conjunto ao meio e a igualdade dos multiconjuntos E e D.

Complexidade $O(S \log S)$, onde S é a soma dos tamanhos dos conjuntos (o fator logarítmico vem da compressão de coordenadas dos valores).