



Maratona Nordestina de Programação

Caderno de Soluções

2026

Problema A. Arretadas Remoções

Tempo limite: 1000 ms
Memória limite: 1024 MiB
Autor: João Ayalla

Solução

Uma retirada arretada é válida quando todos os números removidos possuem algum divisor primo em comum. Essa é a observação que transforma a condição do gcd em algo mais fácil. Em vez de pensar diretamente no valor do gcd, podemos pensar em qual primo é comum a todos os elementos naquela remoção.

Como as bandeiras restantes se juntam depois de cada retirada, é importante perceber que qualquer subsequência do array original pode ser removida em uma mesma operação, desde que os elementos entre eles tenham sido removidos previamente. Considerando isso, podemos pensar em uma programação dinâmica em intervalos.

Vamos olhar para um intervalo $[L, R]$ do vetor original e pensar no que acontece com a primeira bandeira dele, A_L . Na retirada em que A_L for removida, existe algum primo p que divide todos os elementos removidos junto com ela. Além disso, se outro elemento A_M também for removido nessa mesma retirada, então todos os elementos entre L e M precisam ter sido removidos antes, para que A_L e A_M sejam adjacentes no momento da retirada deles.

Definimos então $dp[L][R][P]$, no qual:

- se $P = 0$, ainda não escolhemos qual primo será usado para remover A_L , esse primo tem que ser um divisor de A_L ;
- se $P \neq 0$, já decidimos que A_L será removido em uma retirada cujo gcd é divisível por esse primo P . note que nesse estado P deve ser um número primo válido.

Quando $P = 0$, basta testar cada primo que divide A_L e escolher a melhor resposta:

$$dp[L][R][0] = \min_{P|A_L} dp[L][R][P].$$

Agora suponha que $P \neq 0$. Temos duas possibilidades. A primeira é remover A_L sem juntá-lo com nenhum outro elemento usando esse primo. Isso custa uma retirada, e depois resolvemos o restante do intervalo:

$$1 + dp[L+1][R][0].$$

A segunda possibilidade é escolher algum índice $M > L$ tal que P divide A_M . Podemos remover primeiro tudo que está entre L e M ; depois disso, A_L e A_M ficam lado a lado, e podem participar da mesma retirada sustentada pelo primo P . Nesse caso, o custo fica:

$$dp[L+1][M-1][0] + dp[M][R][P].$$

Repare que a retirada que remove A_L ainda não foi contada na primeira parte: ela será contada dentro de $dp[M][R][P]$, quando decidirmos finalmente remover esse grupo.

Assim, para $P \neq 0$, a transição é:

$$dp[L][R][P] = \min \left(1 + dp[L+1][R][0], \min_{\substack{M>L \\ P|A_M}} dp[L+1][M-1][0] + dp[M][R][P] \right).$$

Os casos base são simples: um intervalo vazio custa 0 retiradas, e um intervalo com apenas um elemento custa 1, pois todo $A_i \geq 2$.

A resposta final é $dp[1][N][0]$.

Para implementar, fatoramos todos os A_i e guardamos apenas os primos distintos que aparecem na entrada. Como $A_i \leq 10^6$, cada número possui no máximo 7 fatores primos distintos. Seja F essa quantidade máxima de primos distintos, há $O(N^2 \cdot F)$ estados e $O(N)$ transições a partir de cada estado, resultando em uma complexidade total de $O(N^3 \cdot F)$.

Problema B. Bons Refletores

Tempo limite: 1000 ms
Memória limite: 1024 MiB
Autor: Paulo Correia

Solução

A solução principal consiste em pegar a sequência do sample dada no PDF e rotacioná-la para que a configuração X_0 apareça na primeira posição. Como o sample já é um ciclo válido que passa por todas as configurações de 3 bits, qualquer rotação dele continua sendo um ciclo válido. Depois disso, basta imprimir a sequência rotacionada e repetir X_0 no final.

Essa ideia funciona porque a saída aceita qualquer resposta válida. A sequência do sample já tem exatamente as propriedades pedidas: todas as configurações aparecem uma vez entre as primeiras 8 linhas, duas linhas consecutivas diferem em um único bit, e a última configuração também difere da primeira em um único bit. Rotacionar um ciclo não muda nenhuma dessas relações de vizinhança, apenas escolhe outro ponto de partida.

Outra forma clássica de construir uma sequência desse tipo é usando Gray code. Para cada i de 0 a $2^3 - 1$, considere

$$G_i = i \oplus \left\lfloor \frac{i}{2} \right\rfloor.$$

A sequência $G_0, G_1, \dots, G_{2^3-1}$ visita todos os números de 3 bits exatamente uma vez, e elementos consecutivos diferem em um bit. Além disso, o último elemento também difere do primeiro em um bit, formando um ciclo.

Também é possível enxergar o problema como procurar um ciclo hamiltoniano no hipercubo de dimensão 3. Cada configuração é um vértice, e duas configurações são ligadas por uma aresta quando diferem em exatamente um bit. Como há apenas 8 vértices, uma DP de caixeiro viajante com bitmasks também resolve: o estado guarda o conjunto de configurações já visitadas e a configuração atual, e no fim exigimos que a última configuração esteja ligada a X_0 .

Usando a sequência fixa do sample, basta encontrar a posição de X_0 e imprimir a rotação do ciclo. A complexidade de tempo é $O(8 \cdot 3)$, que neste problema é constante.

Problema C. Categorias de Base

Tempo limite: 2000 ms
Memória limite: 1024 MiB
Autor: Jackson Souza

Solução

Primeiro, há dois casos em que a resposta é imediatamente zero. Se $K > N$, não há profissionais suficientes para deixar todas as comissões não vazias. Se alguma função aparece mais de K vezes, esses profissionais não podem ser distribuídos entre as comissões sem que alguma delas repita essa função.

Fora desses casos, considere inicialmente uma versão em que todos os profissionais têm funções distintas. Nela, a restrição do problema nunca é violada. Portanto, bastaria particionar N elementos distintos em exatamente K grupos não vazios: a resposta seria o número de Stirling do segundo tipo $S(N, K)$.

Uma fórmula fechada para esse número é

$$S(N, K) = \frac{1}{K!} \sum_{j=0}^K (-1)^{K-j} \binom{K}{j} j^N.$$

Para entendê-la, primeiro rotulamos as K comissões. Se apenas j delas estão disponíveis, cada um dos N profissionais pode escolher qualquer uma dessas j comissões, totalizando j^N distribuições. O fator $\binom{K}{j}$ escolhe as comissões disponíveis, e a soma alternada usa inclusão-exclusão para garantir que todas as K comissões sejam usadas. Por fim, dividimos por $K!$, pois no problema as comissões não são rotuladas.

Podemos adaptar exatamente essa fórmula para incorporar as funções. Seja f_a a quantidade de profissionais cuja função é a . Sem a restrição, os profissionais dessa função contribuiriam com j^{f_a} para o termo j^N . Com a restrição, eles precisam ir para comissões distintas. Como os profissionais são distintos, a quantidade de atribuições passa a ser

$$j(j-1)\cdots(j-f_a+1) = f_a! \binom{j}{f_a} = \frac{j!}{(j-f_a)!}.$$

Se $f_a > j$, essa quantidade é zero. Daqui em diante, sempre que a razão $\frac{j!}{(j-f_a)!}$ aparecer com $f_a > j$, ela deve ser interpretada como zero; equivalentemente, podemos ignorar todos os valores de j menores que a maior frequência.

Portanto, a adaptação central da fórmula de Stirling é

$$j^N = \prod_{a: f_a > 0} j^{f_a} \longrightarrow \prod_{a: f_a > 0} \frac{j!}{(j-f_a)!}.$$

Substituindo essa nova contribuição na mesma inclusão-exclusão, obtemos

$$\boxed{\frac{1}{K!} \sum_{j=0}^K (-1)^{K-j} \binom{K}{j} \prod_{a: f_a > 0} \frac{j!}{(j-f_a)!}} \pmod{998244353}.$$

Se implementássemos a fórmula exatamente como ela está escrita, para cada valor de j precisaríamos percorrer todas as funções que aparecem na entrada. Como podem existir N funções diferentes e existem K valores possíveis de j , isso resultaria em *Time Limit Exceeded*.

A melhoria vem de observar que o produto depende apenas da quantidade de vezes que essa função aparece. Então agrupamos as funções pela sua frequência. Seja $\text{cnt}[x]$ a quantidade de funções que aparecem exatamente x vezes. Para um j fixo, uma função com frequência x contribui com $\frac{j!}{(j-x)!}$; portanto, todas as funções com essa mesma frequência contribuem juntas com

$$\left(\frac{j!}{(j-x)!} \right)^{\text{cnt}[x]}.$$

Com isso, o produto sobre todas as funções pode ser reescrito como

$$\prod_x \left(\frac{j!}{(j-x)!} \right)^{\text{cnt}[x]}.$$

Assim, em vez de iterar sobre todas as funções, iteramos apenas sobre as frequências distintas que aparecem.

Seja D a quantidade de frequências positivas distintas. Podemos mostrar que D é pequeno. Se existem D frequências positivas distintas, a menor soma possível dessas frequências acontece quando elas são $1, 2, \dots, D$. Mesmo nesse melhor caso, precisamos ter

$$1 + 2 + \dots + D = \frac{D(D+1)}{2} \leq N.$$

Logo, $D = O(\sqrt{N})$. Assim, devemos computar a seguinte expressão:

$$\frac{1}{K!} \sum_{j=0}^K (-1)^{K-j} \binom{K}{j} \prod_{\substack{x \geq 1 \\ \text{cnt}[x] > 0}} \left(\frac{j!}{(j-x)!} \right)^{\text{cnt}[x]} \pmod{998244353}.$$

Na implementação, pré-calculamos os fatoriais $\text{fat}[i]$ e os fatoriais inversos $\text{invfat}[i]$. Assim, tanto $\frac{j!}{(j-x)!}$ quanto $\binom{K}{j}$ podem ser obtidos em $O(1)$.

Para um j fixo, uma exponenciação rápida com expoente $c > 0$ executa $1 + \lfloor \log_2 c \rfloor$ iterações. Portanto, o custo de todas as exponenciações é proporcional a

$$T = \sum_{x: \text{cnt}[x] > 0} (1 + \lfloor \log_2 \text{cnt}[x] \rfloor).$$

Podemos calcular por programação dinâmica o maior valor possível de T . Para uma frequência f , obter uma contribuição r custa, no mínimo, $f \cdot 2^{r-1}$ profissionais, pois 2^{r-1} é o menor valor de $\text{cnt}[f]$ que faz a exponenciação executar r iterações. Portanto, basta considerar $\text{cnt}[f] \in \{0, 1, 2, 4, 8, \dots\}$. Para $N = 10^5$, há no máximo 446 frequências distintas. Além disso, substituir um conjunto de D frequências por $1, 2, \dots, D$ apenas reduz o número de profissionais usados, sem alterar a contribuição. Portanto, alguma solução ótima usa apenas $f < 450$.

No código abaixo, $\text{dp}[i]$ guarda a maior contribuição alcançada em um estado com i profissionais ainda disponíveis. Cada transição escolhe uma das multiplicidades possíveis para a frequência atual:

```
int dp[N + 1] = {};
for (int f = 1; f < 450; ++f) {
    for (int i = 0; i <= N; ++i) {
        int value = 1;
        for (long long cost = f; cost <= i; cost *= 2) {
            dp[i - cost] = max(dp[i - cost], dp[i] + value);
            ++value;
        }
    }
}
```

```

    }
  }
}
```

O resultado é 769. Logo, para os limites do problema, cada valor de j faz no máximo 769 iterações de exponenciação rápida. A complexidade é $O(N + K \cdot T)$. Como 769 pode ser visto como um valor próximo de $\sqrt{N}$ para o limite de N , podemos expressar a complexidade assintótica como $O(N + K\sqrt{N})$.

Problema D. Decodificador Maluco

Tempo limite: 1000 ms
Memória limite: 1024 MiB
Autor: Ian Gabriel

Solução

Vamos entender primeiro o caso $N = 1$.

Note que multiplicar X por 2^K é a mesma coisa que deslocar todos os bits de X para a esquerda, colocando zeros no começo. Logo, chamando de x_i o i -ésimo bit menos significativo de X e s_i o i -ésimo bit menos significativo de S (0-indexado), temos a seguinte configuração:

...	x_{3K-1}	...	x_{2K}	x_{2K-1}	...	x_K	x_{K-1}	...	x_0	X
...	x_{2K-1}	...	x_K	x_{K-1}	...	x_0	0	...	0	$X \cdot 2^K$
...	s_{3K-1}	...	s_{2K}	s_{2K-1}	...	s_K	s_{K-1}	...	s_0	S

Como $b \oplus 0 = b$, para qualquer b , então temos $x_i = s_i$ para $i < K$.

Para $i \geq k$, sabemos pelo enunciado que $s_i = x_{i-k} \oplus x_i$. Logo, pelas propriedades do XOR, podemos chegar que $x_i = x_{i-k} \oplus s_i$. Se iterarmos pelos bits do menos para o mais significativo, os valores de s_i e x_{i-k} já serão conhecidos. Portanto, também conhecemos o valor de x_i .

Desta forma, conseguimos reconstruir o X bit por bit e desfazer um passo da criptografia em $O(\log S)$.

Para $N > 1$, basta repetir o processo acima N vezes. A complexidade total fica $O(N \log S)$, o que é suficiente para o limite de tempo.

É preciso tomar cuidado para construir apenas os 32 primeiros bits de X .

Problema E. Enigma da Balança

Tempo limite: 1000 ms
Memória limite: 1024 MiB
Autor: Lua Guimarães

Solução

Cada pesagem tem três resultados possíveis: prato esquerdo mais pesado, prato direito mais pesado, ou empate. Portanto, com k pesagens é possível distinguir no máximo 3^k possibilidades. Como existem N escolhas possíveis para a camisa mais pesada, é necessário que $3^k \geq N$. Assim, o menor valor possível é $\lceil \log_3 N \rceil$.

Esse limite também é atingível. Mantenha o conjunto de camisas que ainda podem ser a mais pesada. Enquanto houver mais de uma candidata, divida esse conjunto em três grupos, de modo que nenhum deles tenha mais do que $\lceil m/3 \rceil$ camisas, onde m é a quantidade atual de candidatas. Coloque o primeiro grupo no prato esquerdo e o segundo no prato direito.

Se o prato esquerdo ficar mais pesado, a camisa procurada está no primeiro grupo. Se o prato direito ficar mais pesado, ela está no segundo. Se houver empate, ela está no terceiro grupo, formado pelas camisas que ficaram fora da balança.

A cada pesagem, o número de candidatas cai para no máximo um terço arredondado para cima do valor anterior. Depois de $\lceil \log_3 N \rceil$ pesagens, restará apenas uma candidata, que deve ser informada como resposta final. Como $N \leq 2000 < 3^7$, são necessárias no máximo 7 pesagens, respeitando o limite de 8.

Assim, a complexidade em número de pesagens é $O(\log N)$, mais precisamente $\lceil \log_3 N \rceil$. O trabalho local para manter e dividir o conjunto de candidatas ao longo da interação é $O(N)$ no total.

Problema F. Fugindo do Sol

Tempo limite: 1000 ms
Memória limite: 1024 MiB
Autor: Paulo Correia

Solução

Pense em cada sombra como uma região onde Samuell pode andar de graça, isto é, sem aumentar a distância percorrida sob o sol. O que custa é apenas o trecho descoberto necessário para sair de uma região sombreada e alcançar outra.

Com essa visão, o problema vira um menor caminho. Criamos um vértice para cada guarda-sol, além de dois vértices especiais: a linha do mar e o calçadão. O peso entre dois vértices é a menor distância que precisa ser percorrida sob o sol para ir de uma dessas regiões até a outra.

Entre dois guarda-sóis i e j , essa distância é a distância entre os centros menos os dois raios:

$$\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} - r_i - r_j.$$

Se as sombras se tocam ou se sobrepõem, esse valor é negativo ou zero, e a distância sob o sol deve ser 0. Portanto usamos

$$\max\left(0, \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} - r_i - r_j\right).$$

Para ligar a linha do mar a um guarda-sol, basta medir quanto falta para a sombra alcançar a reta $y = 0$. O custo é

$$\max(0, y_i - r_i).$$

De modo parecido, o custo para sair de um guarda-sol até o calçadão, na reta $y = H$, é

$$\max(0, H - y_i - r_i).$$

Também adicionamos a aresta direta entre mar e calçadão com custo H , que representa atravessar toda a faixa sem usar nenhuma sombra.

Agora qualquer caminho nesse grafo representa uma estratégia válida: Samuell anda sob o sol apenas nos pesos das arestas, e dentro das sombras o custo é zero. Por outro lado, qualquer rota real pode ser descrita pela sequência de sombras visitadas, pagando exatamente as lacunas entre elas. Então o menor caminho do vértice do mar até o vértice do calçadão dá a resposta.

Vamos deixar essa corretude um pouco mais explícita. Dentro de uma sombra, ou ao longo da linha do mar, o custo é zero, então podemos contrair cada uma dessas regiões para um vértice. Sempre que uma rota sai de uma região protegida e vai até outra, o menor trecho descoberto possível é exatamente a distância entre as duas regiões. Entre dois círculos, essa distância é a distância entre os centros menos os raios; entre um círculo e uma reta horizontal, é a distância vertical que ainda falta para a sombra alcançar a reta. Se as regiões se tocam, essa distância é zero.

Portanto, toda rota real gera um caminho no grafo com custo menor ou igual ao custo da rota. No outro sentido, qualquer caminho no grafo pode ser realizado na praia: basta andar de graça dentro das sombras escolhidas e atravessar em linha reta cada lacuna representada por uma aresta. Assim, o menor caminho no grafo e a melhor rota real têm exatamente o mesmo valor.

Como há $N + 2$ vértices e o grafo é completo, calculamos todos os pesos em $O(N^2)$ e rodamos Dijkstra. Usando a implementação direta com matriz de adjacência, não é necessário usar fila de prioridade: a cada passo escolhemos o vértice não visitado de menor distância por varredura e relaxamos todos os seus vizinhos. A complexidade de tempo total continua sendo $O(N^2)$.

Problema G. Gaiatice na Fila

Tempo limite: 3000 ms
Memória limite: 1024 MiB
Autor: Luis Enilton

Solução

Uma resenha é exatamente uma subsequência **AB**: escolhemos um torcedor **A** e, depois dele na fila, um torcedor **B**. Então não precisamos guardar todas as subsequências. Basta manter a quantidade atual de letras **A**, a quantidade atual de letras **B**, e a resposta atual.

Também mantemos a fila em uma **deque**, pois as operações inserem e removem nas duas pontas. A **deque** serve apenas para saber qual caractere foi removido quando ocorre uma operação de tipo 3 ou 4.

Ao inserir no final da fila, só surgem novas resenhas se o novo caractere for **B**. Nesse caso, ele forma uma subsequência **AB** com cada **A** que já estava na fila, então somamos $cntA$ à resposta. Se o novo caractere for **A**, ele não possui ninguém depois dele ainda, então não cria nenhuma nova resenha.

Ao inserir no início da fila, a situação é simétrica. Só surgem novas resenhas se o novo caractere for **A**. Ele fica antes de todos os **B** já presentes, então somamos $cntB$ à resposta. Se o novo caractere for **B**, ele não pode ser o segundo caractere de uma resenha nova, pois não há nada antes dele.

Para remover, fazemos o processo inverso. Se removemos um **B** do final, desaparecem as resenhas formadas por esse **B** com todos os **A** anteriores, então subtraímos $cntA$. Se removemos um **A** do início, desaparecem as resenhas formadas por ele com todos os **B** posteriores, então subtraímos $cntB$. Remover um **A** do final ou um **B** do início não remove nenhuma resenha em que esse caractere participe.

Depois de atualizar a resposta, ajustamos $cntA$ ou $cntB$ e imprimimos o valor atual. A resposta pode ser grande, então deve ser guardada em inteiro de 64 bits. Cada operação é processada em $O(1)$, então a complexidade de tempo total é $O(Q)$.

Problema H. Histórias de Cordel

Tempo limite: 1000 ms
Memória limite: 1024 MiB
Autor: Vinícius Silva de Jesus

Solução

Em resumo, o problema consiste em contar o número de tries distintas que podem ser formadas a partir de uma árvore com raiz fixa.

Primeiro, vamos pensar em como resolver o problema para uma subárvore do roteiro e depois combinar essas respostas parciais até chegar ao capítulo 1.

Defina $sub[u]$ como a quantidade de tries distintas que podem ser obtidas usando apenas a subárvore enraizada em u .

Se u for folha, então existe apenas uma possibilidade, logo:

$$sub[u] = 1.$$

Agora suponha que u não seja folha, denote por X a quantidade de continuções imediatas de u . Também denotaremos por $A(n, k)$ a função arranjo:

$$A(n, k) = \frac{n!}{(n - k)!}.$$

Se $X > 26$, então a resposta é 0, pois não é possível atribuir letras distintas a todas as saídas de u .

Caso contrário, vamos separar a transição em dois casos.

Caso 1: todas as subárvores de u são não isomorfas entre si.

Nesse caso, podemos atribuir letras distintas às X saídas de u de $A(26, X)$ maneiras. Além disso, qualquer escolha válida de trie em cada subárvore gera um conjunto final de palavras diferente, já que as letras escolhidas no primeiro passo já distinguem essas ramificações. Portanto:

$$sub[u] = A(26, X) \cdot \prod_{v \text{ continuação de } u} sub[v].$$

Caso 2: algumas subárvores de u são isomorfas entre si.

Se duas ou mais subárvores são isomorfas, permutar entre elas as letras escolhidas não cria um novo conjunto de palavras. Por exemplo, se dois blocos idênticos recebem as letras (A, B) ou (B, A) , o resultado final é o mesmo.

Isso é análogo a contar anagramas de uma palavra com letras repetidas: se todos os elementos fossem distintos, contaríamos todas as permutações; quando alguns se repetem, precisamos dividir pelos fatoriais das frequências.

Então, se agruparmos as continuções de u por assinatura isomórfica e denotarmos por F_i a frequência de cada assinatura distinta, basta corrigir a fórmula anterior dividindo por todas essas permutações redundantes:

$$sub[u] = \frac{A(26, X) \cdot \prod_{v \text{ continuação de } u} sub[v]}{\prod_i F_i!}.$$

Essa fórmula, inclusive, cobre também o primeiro caso, já que quando todas as subárvores são distintas temos $F_i = 1$ para todo i .

Assim, a resposta do problema é $sub[1]$.

Para identificar quais subárvores são isomorfas, podemos usar o algoritmo de Aho-Hopcroft-Ullman (AHU) para isomorfismo de árvores enraizadas. A ideia é processar os vértices de baixo para cima e definir a assinatura de cada vértice como o multiconjunto ordenado das assinaturas de suas continuções imediatas. Duas subárvores são isomorfas se, e somente se, recebem a mesma assinatura.

Na implementação, ordenamos as assinaturas dos filhos de cada vértice e atribuímos um identificador para cada multiconjunto distinto. A soma dos graus dos vértices é $O(N)$, e o custo das ordenações ao longo da árvore fica $O(N \log N)$ no pior caso. Portanto, a complexidade de tempo é $O(N \log N)$.

Problema I. Incrível XORlândia

Tempo limite: 2000 ms
Memória limite: 1024 MiB
Autor: João Ayalla

Solução

Vamos primeiro esquecer a alteração de uma aresta e pensar em como reconhecer se um grafo já é incrível.

Se existe um ciclo com XOR diferente de zero no grafo, podemos sair de 1, ir até algum vértice desse ciclo, dar uma volta nele e depois seguir até N . Como o enunciado permite passar por uma mesma rua mais de uma vez, isso cria um caminho de 1 até N cujo XOR muda pelo valor desse ciclo. Portanto, para que todos os caminhos entre 1 e N tenham XOR zero, é necessário que todo ciclo do grafo tenha XOR zero.

Agora vamos supor que todo ciclo do grafo tenha XOR zero. Considere dois caminhos quaisquer P e Q entre os vértices 1 e N . Ao percorrer o caminho P de 1 até N e, em seguida, o caminho Q no sentido contrário, de N até 1, obtemos um ciclo, que não necessariamente é simples. Como todos os ciclos do grafo tem XOR 0, então não pode haver diferença entre o XOR de P e o XOR de Q . Portanto, todos os caminhos entre 1 e N possuem o mesmo XOR, e basta considerar qualquer um deles.

Logo, um grafo é incrível se, e somente se:

- todo ciclo do grafo (não necessariamente simples) tem XOR zero;
- existe algum caminho de 1 até N com XOR zero.

Agora vem a observação principal: não precisamos considerar todos os ciclos do grafo e verificar individualmente se cada um possui XOR zero. Afirmamos que todos os ciclos de um grafo têm XOR zero se, e somente se, todos os seus ciclos fundamentais têm XOR zero.

Fixando uma árvore geradora do grafo, um ciclo é chamado de fundamental quando é formado por uma única aresta que não pertence à árvore e pelo caminho, composto apenas por arestas da árvore, que conecta os extremos dessa aresta.

Vamos provar isso da seguinte forma: pegue uma árvore geradora qualquer do grafo e enraíze-a no vértice 1. Para cada vértice V , seja X_V o XOR dos pesos no caminho da árvore entre 1 e V .

Para cada aresta de fora da árvore, ligando U e V com peso W , ela forma um ciclo fundamental junto com o caminho da árvore entre U e V e o XOR dele é:

$$C = X_U \oplus X_V \oplus W.$$

Se todos esses ciclos fundamentais têm XOR zero, então para toda aresta fora da árvore vale:

$$W = X_U \oplus X_V.$$

Para uma aresta da própria árvore, essa igualdade também vale pela definição dos valores X . Portanto, para toda aresta (U, V) do grafo, seu peso é $X_U \oplus X_V$.

Considere então qualquer ciclo (não necessariamente simples) com vértices $V_0, V_1, \dots, V_K$, no qual $V_K = V_0$. O XOR desse ciclo é:

$$(X_{V_0} \oplus X_{V_1}) \oplus (X_{V_1} \oplus X_{V_2}) \oplus \dots \oplus (X_{V_{K-1}} \oplus X_{V_K}).$$

Seja cnt_v a quantidade de vezes em que o vértice v aparece no ciclo. Podemos ver que cada vértice v vai aparecer exatamente $2 \cdot cnt_v$ vezes nessa expressão, então tudo se cancela e o XOR do ciclo é zero. Assim, checar os ciclos fundamentais é suficiente para garantir que todos os ciclos do grafo têm XOR zero.

Com isso, podemos voltar para o problema original. Vamos achar qualquer árvore geradora do grafo da entrada, podendo ser encontrado usando um DFS ou um BFS. Em seguida devemos calcular os valores X_V , e analisar apenas os ciclos fundamentais das arestas fora da árvore. Chamaremos um ciclo fundamental de **ruim** se seu XOR é diferente de zero, e de **bom** caso contrário.

Agora considere que vamos alterar uma aresta E . Se seu peso antigo é W e seu novo peso é W' , então todos os ciclos e caminhos que usam essa aresta têm seu XOR alterado por:

$$T = W \oplus W'.$$

Como o novo peso deve ser diferente do antigo, obrigatoriamente $T \neq 0$.

Para que todos os ciclos fundamentais fiquem com XOR zero após a mudança, a aresta escolhida precisa satisfazer:

- todo ciclo fundamental que for ruim deve passar por ela, pois ciclos ruins que não passam por ela continuariam ruins;
- nenhum ciclo fundamental bom pode passar por ela, pois ciclos bons que passam por ela passariam a ter XOR $T \neq 0$;
- todos os ciclos fundamentais ruins devem ter o mesmo XOR, pois a mesma mudança T precisa zerar todos eles.

Se os ciclos ruins têm mais de um valor distinto de XOR, então é impossível modificar apenas uma aresta para fazer o grafo ser incrível.

Caso contrário, seja B a quantidade de ciclos ruins. Para cada aresta E , queremos saber se ela aparece em todos os B ciclos ruins e em nenhum ciclo bom. Para isso, definimos:

$D_E = (\text{quantidade de ciclos ruins que passam por } E) - (\text{quantidade de ciclos bons que passam por } E)$

Assim, se uma aresta tem $D_E = B$, podemos dizer que é possível modificar o valor dela para fazer o grafo ser considerado incrível se todos os ciclos fundamentais que forem ruins tiverem o mesmo XOR.

Por fim, resta explicar como calcular os valores de D_E . Para uma aresta fora da árvore, seu ciclo fundamental é exatamente o ciclo formado ao adicioná-la à árvore. Assim, definimos $D_E = 1$ se esse ciclo for ruim e $D_E = -1$ se ele for bom.

Para as arestas da árvore, inicialmente definimos $D_E = 0$. Em seguida, consideramos cada aresta (U, V) que não pertence à árvore. De acordo com o ciclo fundamental dessa aresta, devemos adicionar $+1$ ou -1 ao valor de todas as arestas pertencentes ao caminho entre U e V na árvore.

Essas atualizações podem ser realizadas de forma online utilizando Heavy-Light Decomposition (HLD), ou de forma offline por meio de um DFS. A complexidade total é $O(N + M)$ quando os valores de D_E são calculados offline com um DFS, ou $O(M \log^2 N)$ quando utilizamos HLD.

Problema J. Jogo do Brega

Tempo limite: 1000 ms
Memória limite: 1024 MiB
Autor: João Ayalla

Solução

O primeiro ponto importante é perceber que, em cada intervalo, só existe uma carta que realmente decide o vencedor: a carta de maior valor.

Como todos os valores da sequência são distintos, quem remover a maior carta do intervalo necessariamente terá pontuação maior que a do outro cantor. Portanto, o jogo inteiro pode ser visto como uma disputa para ver quem consegue pegar essa carta de maior valor.

Se a carta máxima já está em uma das extremidades do intervalo, então Adilson vence imediatamente: como ele joga primeiro, basta remover essa carta. Bem como se o tamanho do intervalo for igual a 1.

Agora considere o caso em que a carta de maior não está em uma das extremidades. Podemos afirmar o seguinte:

- se o tamanho do intervalo é par, o jogador da vez vence;
- se o tamanho do intervalo é ímpar, o jogador da vez perde.

Para entender a estratégia, podemos definir dois valores, l_{dist} e r_{dist} . Seja mx o índice do maior elemento no intervalo $[L_i, R_i]$. Definimos $l_{dist} = mx - L_i$ e $r_{dist} = R_i - mx$.

Quando um jogador escolhe uma das pontas do intervalo e remove o elemento mais à esquerda ou mais à direita, o valor de l_{dist} ou de r_{dist} diminui em uma unidade, respectivamente.

Observe que não é vantajoso fazer com que l_{dist} ou r_{dist} se torne zero, pois isso deixa o maior elemento em uma das pontas do intervalo e garante automaticamente a vitória do outro jogador. Portanto, um jogador somente fará esse tipo de jogada quando não houver nenhuma outra opção, ou seja, quando ambos os valores de l_{dist} e r_{dist} forem iguais a 1.

Com isso, para cada consulta $[L_i, R_i]$, basta determinar a posição do maior valor do intervalo:

- se o maior valor estiver em L_i ou em R_i , Adilson vence;
- caso contrário, se o tamanho do intervalo for par, Adilson vence;
- nos demais casos, Reginaldo vence.

Na implementação, podemos utilizar uma estrutura de dados capaz de responder consultas de *Range Maximum Query*, como uma *segment tree* ou uma *sparse table*. Após encontrar o maior valor presente no intervalo, verificamos se P_{L_i} ou P_{R_i} é igual a esse máximo. Em caso afirmativo, sabemos que Adilson vence. Caso contrário, o vencedor é determinado pela paridade de $R_i - L_i + 1$.

Com uma *sparse table*, o pré-processamento custa $O(N \log N)$ e cada consulta é respondida em $O(1)$, totalizando $O(N \log N + Q)$. Com uma *segment tree*, a complexidade de tempo fica $O((N + Q) \log N)$.

Problema K. Karamelos no São João

Tempo limite: 1000 ms
Memória limite: 1024 MiB
Autor: Ian Gabriel

Solução

João quer que, ao dividir os $X \cdot Y$ doces entre os A amigos, sobre exatamente um doce. Portanto, precisamos que

$$X \cdot Y \equiv 1 \pmod{A}.$$

Isso só é possível quando $\gcd(X, A) = 1$. Caso contrário, todo múltiplo de X também será divisível por $\gcd(X, A)$ e nunca poderá deixar resto 1 módulo A .

Quando $\gcd(X, A) = 1$, existe um inverso de X módulo A . Ele poderia ser encontrado usando o algoritmo de Euclides estendido, mas, como $A \leq 10^4$, basta testar os valores de Y até encontrar o menor que satisfaz a congruência.

Além disso, é necessário que haja doces suficientes para todos os amigos, ou seja, $X \cdot Y \geq A + 1$. Assim, testamos os valores de Y em ordem crescente até que ambas as condições sejam satisfeitas.

A complexidade é $O(A)$.

Problema L. Lampions League

Tempo limite: 1000 ms
Memória limite: 1024 MiB
Autor: João Ayalla

Solução

Chamemos de T a quantidade total de vezes que **Tengo** foi gritado, e de L a quantidade total de vezes que **Lengo** foi gritado. O problema pede apenas a paridade de $|T - L|$.

Para paridade, o sinal não faz diferença. Além disso, subtrair ou somar um número tem o mesmo efeito módulo 2. Portanto:

$$|T - L| \equiv T - L \equiv T + L \pmod{2}.$$

Mas $T + L$ é simplesmente o total de palavras gritadas em todos os lances:

$$T + L = 1 + 2 + 3 + \dots + N = \frac{N(N+1)}{2}.$$

Então só precisamos descobrir se essa soma é par ou ímpar. A complexidade é $O(1)$.

Problema M. Maior São João do Mundo

Tempo limite: 1000 ms
Memória limite: 1024 MiB
Autor: Jackson Souza

Solução

Para uma votação qualquer, defina

$$A = \#(\text{pessoas que preferem G}) - \#(\text{pessoas que preferem R})$$

e

$$B = \#(\text{votos computados como G}) - \#(\text{votos computados como R}).$$

A pontuação dessa votação é exatamente $A \cdot B$: quem prefere G contribui com B , enquanto quem prefere R contribui com $-B$.

Agora classifique cada pessoa pelo par formado por opinião verdadeira e voto computado. Existem quatro tipos:

$$GG, GR, RG, RR.$$

Em uma votação, se as quantidades desses tipos forem a, b, c, d , respectivamente, então

$$A = a + b - c - d$$

e

$$B = a - b + c - d.$$

Multiplicando,

$$A \cdot B = (a - d)^2 - (b - c)^2.$$

Essa fórmula mostra a estrutura do problema. Os tipos GG e RR contribuem para a parte positiva, enquanto os tipos GR e RG aparecem na parte que subtrai.

Para maximizar a parte positiva, queremos separar os GG dos RR. Se colocarmos todos os GG em uma votação e todos os RR na outra, essa contribuição vira

$$cnt_{GG}^2 + cnt_{RR}^2.$$

Misturar os dois tipos em uma mesma votação só diminui o valor de $(a - d)^2$, pois eles têm sinais opostos nessa diferença.

Resta lidar com os tipos conflitantes GR e RG. Eles aparecem como $-(b - c)^2$ em cada votação, então queremos minimizar

$$(b_1 - c_1)^2 + (b_2 - c_2)^2.$$

O valor total de $(b - c)$ nas duas votações é fixo:

$$cnt_{GR} - cnt_{RG}.$$

A soma de dois quadrados com soma fixa é mínima quando os dois valores ficam o mais equilibrados possível. Portanto, se

$$D = |cnt_{GR} - cnt_{RG}|,$$

a menor perda possível é

$$\left\lfloor \frac{D}{2} \right\rfloor^2 + \left\lceil \frac{D}{2} \right\rceil^2.$$

Assim, a resposta é

$$cnt_{GG}^2 + cnt_{RR}^2 - \left\lfloor \frac{D}{2} \right\rfloor^2 - \left\lceil \frac{D}{2} \right\rceil^2.$$

Basta contar os quatro tipos e aplicar a fórmula. A complexidade é $O(N)$ e a memória é $O(1)$.

Como observação, também existe uma solução alternativa que modela diretamente quantas pessoas de cada um dos quatro tipos vão para a primeira votação; o restante vai para a segunda. Como são apenas quatro variáveis, dá para otimizar a função objetivo com quatro buscas binárias ou ternárias discretas aninhadas. Depois da contagem inicial dos tipos, essa alternativa faz $O(\log^4 N)$ avaliações, então sua complexidade de tempo fica $O(N + \log^4 N)$. A fórmula acima é a forma mais simples de implementar a mesma ideia.

Problema N. Não visse meu álbum?

Tempo limite: 1000 ms
Memória limite: 1024 MiB
Autor: Ícaro Nunes

Solução

Defina $dp[i][j]$ como a quantidade esperada de dias restantes para completar o álbum, sabendo que o álbum possui atualmente j jogadores e Xand já tem figurinhas de i deles. Se $i = j$, o álbum já está completo, então

$$dp[i][j] = 0.$$

Agora considere um estado com $i < j$. No começo do dia, pode entrar um novo jogador no álbum. Seja $p = P/100$. Se $j = 2026$, não há mais como adicionar jogador novo, então usamos probabilidade 0 para esse evento. Caso contrário, a probabilidade é p .

Depois disso, Xand compra uma figurinha. Existem quatro possibilidades:

- não entra jogador novo e a figurinha é repetida, com probabilidade $(1 - p)\frac{i}{j}$; o estado continua sendo (i, j) ;
- entra jogador novo e a figurinha é repetida, com probabilidade $p\frac{i}{j+1}$; o estado vira $(i, j + 1)$;
- não entra jogador novo e a figurinha é nova, com probabilidade $(1 - p)\left(1 - \frac{i}{j}\right)$; o estado vira $(i + 1, j)$;
- entra jogador novo e a figurinha é nova, com probabilidade $p\left(1 - \frac{i}{j+1}\right)$; o estado vira $(i + 1, j + 1)$.

Todo caso consome exatamente um dia. A única dificuldade é que, no primeiro caso, voltamos para o mesmo estado. Por isso escrevemos a equação da esperança e isolamos esse termo.

Chamando as quatro probabilidades acima de w, x, y, z , temos

$$dp[i][j] = 1 + w \cdot dp[i][j] + x \cdot dp[i][j + 1] + y \cdot dp[i + 1][j] + z \cdot dp[i + 1][j + 1].$$

Logo,

$$dp[i][j] = \frac{1 + x \cdot dp[i][j + 1] + y \cdot dp[i + 1][j] + z \cdot dp[i + 1][j + 1]}{1 - w}.$$

Preenchemos a tabela de trás para frente, pois as transições usam estados com i maior, j maior, ou ambos. A resposta final é $dp[0][N]$, já que Xand começa sem nenhuma figurinha e o álbum começa com N jogadores.

Como $j \leq 2026$, a tabela tem $O(2026^2)$ estados. Cada estado é calculado em $O(1)$, então a complexidade de tempo é $O(2026^2)$.

Problema O. Operação Mandacaru

Tempo limite: 5000 ms
Memória limite: 1024 MiB
Autor: Vinícius Silva de Jesus

Solução

Primeiro, esqueça por um momento os valores dos pedidos e pense apenas em como os passageiros podem ocupar os assentos. Em um mesmo assento, duas viagens não podem compartilhar um trecho, mas uma pode começar exatamente na cidade em que a outra termina. Portanto, toda escolha válida de pedidos pode ser vista como até C sequências de viagens compatíveis, uma para cada assento.

Agora acompanhe um único assento desde a cidade 1 até a cidade M . Se ele está vazio entre duas cidades consecutivas, apenas avançamos pela rota. Se ele é usado pelo passageiro i , avançamos diretamente de L_i até R_i e recebemos P_i . Assim, o histórico de um assento se comporta como um caminho em um grafo ordenado pelas cidades.

Precisamos escolher C desses caminhos simultaneamente, sem permitir que dois deles usem o mesmo pedido. Vários caminhos com restrições de uso compartilhadas podem ser representados por unidades de fluxo: cada unidade corresponde a um assento, e a capacidade de uma aresta determina quantas unidades podem utilizá-la. Essa observação conduz à modelagem por fluxo de custo mínimo.

Crie um vértice para cada cidade, mantendo a ordem da rota. Para cada j de 1 a $M - 1$, adicione uma aresta da cidade j para a cidade $j + 1$, com capacidade C e custo 0.

Agora, para cada pedido i , adicione uma aresta de L_i para R_i , com capacidade 1 e custo $-P_i$. Por fim, envie exatamente C unidades de fluxo da cidade 1 até a cidade M .

Cada unidade de fluxo representa um dos assentos do ônibus. Quando essa unidade usa uma aresta $(j, j + 1)$ de custo zero, o assento permanece desocupado nesse trecho. Quando usa a aresta de um pedido i , o assento é ocupado pelo passageiro desde L_i até R_i , e o custo do fluxo diminui em P_i .

Uma aresta de pedido tem capacidade 1, portanto um pedido nunca é escolhido mais de uma vez. Além disso, como apenas C unidades de fluxo percorrem o grafo, no máximo C passageiros ocupam o ônibus em qualquer trecho.

Por outro lado, qualquer conjunto válido de pedidos pode ser distribuído entre os C assentos. Basta processar as viagens pela cidade de embarque e atribuir cada passageiro a um assento que já esteja livre. Se nenhum assento estivesse livre, haveria mais de C passageiros em algum trecho, contradizendo a validade do conjunto. Para cada assento, suas viagens formam um caminho da cidade 1 até a cidade M , usando arestas de custo zero entre uma viagem e outra.

Logo, há uma correspondência entre soluções válidas e fluxos de valor C . Como cada pedido escolhido contribui com custo $-P_i$, minimizar o custo total equivale a maximizar a arrecadação. A resposta é, portanto, o oposto do custo do fluxo mínimo.

O grafo possui $O(M)$ vértices e $O(N + M)$ arestas. Inicialmente, todas as arestas com capacidade positiva avançam pela rota, formando um DAG; assim, os potenciais iniciais podem ser calculados em $O(N + M)$. Depois disso, usamos Dijkstra com custos reduzidos para cada aumento de fluxo. Como o valor total do fluxo é C , são necessários no máximo C aumentos.

A complexidade de tempo é

$$O(C(N + M) \log M),$$

e a complexidade de memória é

$$O(N + M).$$

Essa família de problemas é conhecida como *weighted scheduling problem*, mais especificamente como *weighted interval scheduling with capacity*.