Seletiva para Maratona de Programação de 2015

Comentários sobre os problemas

Patrocínio:

Departamento de Ciência da Computação IME-USP

Bridge A: Bridge

Autor do problema: Stefano Tommasini
Análise: Stefano Tommasini

Nesse problema bastava rodar um backtracking testando todas as possibilidades, podemos ver que cada jogador tem 4! possibilidades de ordem de cartas. Logo a complexidade fica $O((4!)^4)$.

Problema B: Renzo e a decoração capicuânica

Autor do problema: Marcio T. I. Oshiro / Carlinhos
Análise: Marcio T. I. Oshiro

Este problema pode ser divido em duas partes: encontrar a representação de N em uma base b; e verificar se essa representação é capicua (palíndromo).

A representação de N em uma base $b > 0$ é $(a_m a_{m-1} \dots a_1 a_0)_b$ tal que $a_i \in \{0, 1, \dots, b-1\}$, para todo $i \in \{0, 1, \dots, m\}$, $a_m \neq 0$ e

$$N = a_m b^m + a_{m-1} b^{m-1} + \cdots + a_1 b + a_0.$$

Note que $N = b(a_m b^{m-1} + a_{m-1} b^{m-2} + \cdots + a_1) + a_0$. Logo a_0 é o resto da divisão de N por b. De forma análoga, temos que a_1 é o resto da divisão de $(N - a_0)/b$ [1] por b. Repetindo esse raciocínio, podemos facilmente encontrar o valor de a_i, para todo $i \in \{0, 1, \dots, m\}$.

Dada a sequência $(a_m, a_{m-1}, \dots, a_0)$, basta verificar se $a_{m-i} = a_i$, para $i \in \{0, 1, \dots, \lfloor m/2 \rfloor\}$. Se todos os pares forem iguais, a representação de N na base b é capicua. Caso contrário, não é.

Como $N < 2^{31}$, vale que $m \leq 31$, pois quanto menor a base, mais dígitos são necessários para representar o mesmo número.

Problema C: Competição de Robótica

Autor do problema: Antonio Roberto C. Jr.
Análise: Marcio T. I. Oshiro

A tarefa neste problema é basicamente simular as instruções que o robô recebe e determinar a posição final dele. No entanto, a instrução deve ser repetida $n \leq 10^9$ vezes. Logo, para valores grandes de n, calcular n vezes a translação e a rotação vai demorar.

Se a instrução $\mathrm{I}(\alpha, \ell)$ fosse uma transformação linear, poderíamos escrevê-la como uma matriz M e calcular M^n para encontrar a posição final do robô após as n execuções da instrução $\mathrm{I}(\alpha, \ell)$.

Translação não é uma transformação linear no sistema de coordenadas cartesiano. Mas ela é uma transformação linear para *coordenadas homogêneas*[2]. Portanto, a instrução $\mathrm{I}(\alpha, \ell)$, com rotação seguida de translação, pode ser descrita pela matriz

$$M = \begin{bmatrix} \cos(\alpha') & \sin(\alpha') & 0 \\ -\sin(\alpha') & \cos(\alpha') & 0 \\ \ell & 0 & 1 \end{bmatrix},$$

onde $\alpha' = \alpha\pi/180$ é o valor de α em radianos.

Para calcular M^n, é preciso utilizar a técnica de exponenciação com $O(\log n)$ multiplicação de matrizes, caso contrário o algoritmo será tão eficiente quanto a simulação direta.

[1] Note que $(N - a_0)/b$ corresponde à divisão inteira de N por b.
[2] `https://en.wikipedia.org/wiki/Homogeneous_coordinates`

Passeios D: Passeios aleatórios pela Tailândia

Autor do problema: Stefano Tommasini
Análise: Stefano Tommasini

Primeira observação é que se usarmos um barco em algum momento nunca mais vamos usar aviões pois era melhor ter usado o avião diretamente. Então vamos usar avião por algumas vezes e depois pegar o barco usando o menor caminho até o destino. É fácil provar que se usamos um avião e formos parar numa cidade mais longe do destino da que já estávamos é ótimo usar o avião novamente. Então basta definirmos se estivermos a uma distância menor ou igual a algum D usamos barco caso contrário. Seja então o vetor $dist$ a distância de cada nó até o destino, para todo i testamos $D = dist[i]$.

Para um D fixo temos

$$R = \left(\sum_{i|dist[i]>D} \left(\frac{K+R}{n} \right) \right) + \left(\sum_{i|dist[i]\leq D} \left(\frac{K+dist[i]}{n} \right) \right),$$

onde R é a resposta. Fixamos aqui que vamos usar o avião inicialmente, se voltarmos para uma cidade mais longe que D voltamos a mesma situacão onde a resposta continua sendo R. Essa recorrência pode ser resolvida isolando o R em O(1) usando somas acumuladas.

Problema E: Fuga de Ayutthaya

Autor do problema: Renzo Gonzalo Gómez Diaz
Análise: Renzo Gonzalo Gómez Diaz

Se considerarmos a grade como um grafo onde cada posição é um vértice e dois vértices distintos são adjacentes se podemos mover-nos de entre essas posições em uma unidade de tempo, então o problema se reduz a decidir se a distância da saída à posição inicial da pessoa é menor do que a distância da saída à algum fogo. Isso pode ser resolvido usando uma busca em largura (BFS).

Rajesi F: Lutando contra os Rajasi

Autor do problema: Stefano Tommasini
Análise: Stefano Tommasini/Arthur Nascimento

Para os rajasis com $y_i \geq x_i$, um guloso sempre funciona, então podemos assumir que $x_i > y_i$. Nesse caso, dá pra provar por um argumento de "swap" que, numa ordem ótima, os y's são decrescentes. Podemos então ordenar os rajasis por y decrescente e fazer $dp[m][k]$ = qual o mínimo HP inicial que precisamos ter para matar os últimos m rajasis, sendo que podemos usar k feitiços?, e ver se $dp[n][k] \leq X$.

Problema G: Loteria tailandesa

Autor do problema: Arthur Nascimento
Análise: Arthur Nascimento

A resposta é sim (`Y`) se e somente se cada um dos fatores primos de N aparece como fator de alguma das faces f_i. Segue a demonstração disso.

Vamos olhar para todos os *outcomes* (sequências de lançamentos e resultados) que fazem uma determinada pessoa (por exemplo, a 1ª pessoa) vencer. A probabilidade de cada *outcome* é da forma $1/(f_{a_1} f_{a_2} \cdots f_{a_x})$, então temos algo do tipo:

$$\frac{1}{f_{a_1} f_{a_2} \cdots f_{a_x}} + \frac{1}{f_{b_1} f_{b_2} \cdots f_{b_y}} + \cdots + \frac{1}{f_{c_1} f_{c_2} \cdots f_{c_z}} = \frac{1}{N}.$$

Simplificando a expressão, obtemos que $f_{a_1} f_{a_2} \cdots f_{a_x} f_{b_1} f_{b_2} \cdots f_{b_y} \cdots f_{c_1} f_{c_2} \cdots f_{c_z}$ é múltiplo de N, ou seja, todos os primos de N aparecem nesse produto.

Por outro lado, se todos os primos aparecem nos f_i's, então existe um "*submultiset*" deles que é múltiplo de N e a resposta é sim.

Podemos usar o mdc para achar rápido os primos em comum entre N e os f_i, resultando numa complexidade $O(M+\log(N))$. Notem que as duas primeiras regras não mudam em nada o problema, e são apenas para confundir as pessoas.

Problema H: Resguardando os Templos

Autor do problema: Renzo Gonzalo Gómez Diaz
Análise: Marcio T. I. Oshiro

Considere um grafo onde cada templo corresponde a um vértice e cada rua corresponde a uma aresta. O problema passa a ser encontrar uma cobertura mínima por arestas, isto é, encotrar um subconjunto C de arestas do grafo com cardinalidade mínima e tal que cada vértice do grafo é extremo de alguma aresta em C.

Note que, se o grafo tem um emparelhamento perfeito, então tal emparelhamento corresponde a uma cobertura mínima por arestas. De fato, cada vértice do grafo é coberto por exatamente uma aresta do emparelhamento perfeito, logo tal emparelhamento é uma cobertura por arestas e não é difícil notar que não é possível cobrir os N vértices com menos do que $N/2$ arestas.

Se o grafo não tem emparelhamento perfeito, podemos encontrar um emparelhamento máximo M e adicionar uma aresta para cada vértice não coberto por M. Isso corresponde a uma cobertura mínima, pois para utilizar menos arestas seria necessário incluir uma aresta que cobre dois vértices não cobertos por M. Mas se tal aresta existisse, então poderíamos incluí-la em M e obter um emparelhamento maior, contradizendo a maximalidade de M.

O enunciado garante que o grafo sempre é bipartido. Isso facilita muito a tarefa de encontrar um emparelhamento máximo, que pode ser realizada através de algoritmos de fluxo em redes.

Problema I: As vias férreas Kunming-Cingapura

Autor do problema: Arthur Nascimento
Análise: Arthur Nascimento/Stefano Tommasini

Traduzindo o problema para a linguagem de grafos, o custo mínimo de manutenção para manter o sistema viário conectado equivale ao custo de uma árvore geradora mínima (MST) no grafo das vias. Considere que cada uma das Q vias construídas é uma *query* e precisamos responder a cada uma delas.

Para cada aresta, vamos pensar em quais momentos, a partir do momento que ela é inserida, ela estará na MST (a aresta é "útil" no momento) e quais não estará (ela é "inútil"). É fácil de ver que, se uma aresta se torna inútil em algum momento, ela nunca voltará a ser útil. Também é fácil ver que a cada instante, existem $N-1$ arestas úteis, e a cada operação, uma aresta é inutilizada (ou a aresta que acabou de ser adicionada é inútil, ou ela será útil e uma das $N-1$ que até então eram úteis vira inútil). Vamos usar do fato que o conjunto de arestas úteis muda pouco com o tempo (no máximo em 1 elemento por vez) para responder várias *queries* de uma vez. Considere o algoritmo:

Particionar as *queries* em blocos de tamanho $\sim \sqrt{Q}$.

Para cada bloco, calculamos a MST da primeira e da última *query*, usando Kruskal ou Prim. Sabemos que essas duas MST's têm no mínimo $N - \sqrt{Q}$ arestas em comum, e sabemos que essas arestas são úteis em todas as *queries* do bloco. Considere o grafo que possui apenas essas arestas: Esse grafo é uma floresta com no máximo $\sqrt{Q}$ componentes. Podemos contrair esse grafo (cada componente vira um vértice). Logo, para cada *query* no bloco, reduzimos o problema para encontrar a MST do grafo contraído, que possui no máximo $2\sqrt{Q}$ arestas (as que eram úteis no começo do

bloco e inúteis no final, e as que foram adicionadas no meio do bloco até o momento da *query*). Isso pode ser feito em $\sqrt{Q}\log(\sqrt{Q})$.

A complexidade total fica $O((N+Q)\sqrt{Q}\log(Q))$. Observe que é possivel fazer essas ordenacoes em tempo linear dentro de cada bloco basta ir dando swap de tras pra frente nas arestas. Para esse problema somente era aceita a complexidade $Q\sqrt{Q}$.

Problema J: Os chedis de Kamphaeng Phet

Autor do problema: Stefano Tommasini
Análise: Marcio T. I. Oshiro

Dadas duas *strings* A e B de comprimentos N e M, respectivamente, o problema pede para calcular o valor mínimo esperado da "diferença" entre A e B. Considere que $A = a_1a_2\cdots a_N$ e $B = b_1b_2\cdots b_M$.

No método descrito no enunciado para calcular uma diferença d, temos dois índices, $i \in [1, N+1]$ e $j \in [1, M+1]$, correspondendo aos caracteres de A e B. Considere que $a_{N+1} = b_{M+1}$ é o caracter vazio. Inicialmente $i = j = 1$, $d = 0$ e a cada passo podemos aumentar o valor de i ou de j da seguinte forma:

- somar 1 em i e d; ou
- somar 1 em j e d; ou
- somar x em i, y em j e K em d, onde x é um valor aleatório em $[1, N-i+1]$ e y é um valor aleatório em $[1, M-j+1]$; ou
- somar 1 em i e j sem alterar o valor de d, apenas se $a_i = b_j$.

Note que se $i = N+1$, no terceiro caso, o valor x deve ser zero. Analogamente, se $j = M+1$, o valor de y deve ser zero.

Seja $d(i, j)$ a diferença mínima esperada entre as *strings* $a_ia_{i+1}\cdots a_N$ e $b_jb_{j+1}\cdots b_M$. A resposta para o problema é $d(1, 1)$. De acordo com as possíveis escolhas em cada passo do método, podemos escrever a seguinte recorrência.

$$d(i,j) = \begin{cases} 0, & \text{se } i = N+1 \text{ e } j = M+1 \\ \min(1 + d(i+1, j), K + \mathbb{E}[X_i]), & \text{se } i \leq N \text{ e } j = M+1 \\ \min(1 + d(i, j+1), K + \mathbb{E}[Y_j]), & \text{se } i = N+1 \text{ e } j \leq M \\ \min(1 + \min(d(i+1, j), d(i, j+1)), K + \mathbb{E}[Z_{ij}]), & \text{se } i \leq N,\ j \leq M \text{ e } a_i \neq b_j \\ d(i+1, j+1), & \text{se } i \leq N,\ j \leq M \text{ e } a_i = b_j \end{cases} \quad (1)$$

As variáveis aleatórias X_i, Y_j e Z_{ij} correspondem aos valores de $d(i+x, j)$, $d(i, j+y)$ e $d(i+x, j+y)$, respectivamente. Como x e y são escolhidos com probabilidade uniforme, quando não são zero, temos que

$$\mathbb{E}[X_i] = \sum_{x=1}^{N-i+1} \frac{d(i+x, j)}{N-i}, \quad \mathbb{E}[Y_j] = \sum_{y=1}^{M-j+1} \frac{d(i, j+y)}{M-j}, \quad \mathbb{E}[Z_{ij}] = \sum_{x=1}^{N-i+1} \sum_{y=1}^{M-j+1} \frac{d(i+x, j+y)}{(N-i)(M-j)}.$$

A recorrência (1) pode facilmente ser calculada por programação dinâmica. A complexidade de tempo esperada é $O(NM)$. Logo, as esperanças devem ser calculadas em tempo $O(1)$. Uma forma simples de fazer isso é mantendo uma matriz de somas acumuladas.

Problema K: Treinando com as larvas de Phuket

Autor do problema: Arthur Nascimento
Análise: Marcio T. I. Oshiro

Dados N comprimentos de arestas, o problema pede para encontrarmos três deles que formam um triângulo de área mínima, se isso for possível. Sejam $a \leq b \leq c$ três dos comprimentos dados. Eles podem formar um triângulo se satisfazem a desigualdade triângular:

$$a + b \geq c.$$

Suponha que a desigualdade triângular é satisfeita para a, b e c. A área A do triângulo formado por esses comprimentos pode ser calculada pela fórmula de Herão. Seja $s = (a + b + c)/2$,

$$A = \sqrt{s(s-a)(s-b)(s-c)}.$$

Podemos reescrever a área do triângulo como

$$A = \frac{b(c\sin(\alpha))}{2},$$

onde α é o ângulo interno oposto ao lado a. Supondo que os valores de b e de c estão fixos, temos que A será mínima se $\sin(\alpha)$ for mínimo. Como a é um lado de menor comprimento do triângulo, sabemos que $\alpha \leq 60°$ é um menor ângulo interno. A função seno é crescente no intervalo $(0°, 60°]$. Logo, a será o menor dos comprimentos dados que forma um triângulo com os comprimentos b e c.

Portanto, o problema pode ser resolvido testando, para todos os possíveis comprimentos b e c, se existe o comprimento a que forma um triângulo e, se existir, calculando a área desse triângulo. O comprimento a pode ser encontrado, ou verificado que não existe, por uma busca binária. Lembrando que é preciso ordenar os comprimentos antes de executar a busca binária. O algoritmo resultante tem complexidade de tempo $O(N^2 \log N)$.

Problema L: Emplacando os *tuk-tuks*

Autor do problema: Marcio T. I. Oshiro / Carlinhos
Análise: Marcio T. I. Oshiro

Este problema é bem direto, basta contar a quantidade máxima de placas com C consoantes e D dígitos, lembrando que as consoantes sempre aparecem antes dos dígitos.

Existem 26 escolhas de consoantes e 10 escolhas de dígitos. Logo, a resposta é

$$26^C \times 10^D.$$

O enunciado garante que a resposta sempre é menor que 2^{31}, logo todo cálculo pode ser feito com variáveis do tipo `int` (inteiro de 32-bits).

Um caso que deveria ser tratado separadamente é o caso $C = D = 0$. Tal caso está dentro das restrições do enunciado e sua resposta é zero, pois uma placa não pode ser vazia.

Problema M: Removendo moedas no Kem Kradãn

Autor do problema: Marcos Kawakami
Análise: Marcos Kawakami

A condição de existência de solução para o jogo das moedas é surpreendentemente simples: Existe solução se e somente se o número de moedas com a face dourada inicialmente voltada para cima (chamaremos estas simplesmente de moedas douradas) for ímpar. Vamos dividir a prova deste fato em duas partes. Primeiramente provaremos que, se o número de moedas douradas for inicialmente ímpar, então existe solução. Depois, provaremos que não é possível remover todas as moedas se o número inicial de moedas douradas for par.

Provaremos a primeira parte por indução. Não é difícil perceber que, se o número de moedas douradas for inicialmente 1, é possível remover todas as moedas do jogo. Suponha agora que o número de moedas douradas seja $K > 1$ e ímpar. Considere a remoção da moeda dourada mais à esquerda. À esquerda desta moeda retirada, ou teremos um segmento contendo uma única moeda dourada, ou não teremos segmento algum. À direita desta moeda, dependendo da cor da face da moeda imediatamente à direita, podemos ter um segmento contendo K (se a face for branca) ou $K - 2$ (se a face for dourada) moedas douradas. Note que, caso o segmento tenha K moedas douradas, podemos repetir o passo para este segmento e, como o número de moedas brancas é limitado, eventualmente obteremos um segmento com $K - 2$ moedas douradas. Teremos, após este processo, um segmento com $K - 2$ moedas douradas e zero ou mais segmentos com uma única moeda dourada. Como cada segmento pode ser resolvido de forma independente, a configuração inicial admite solução.

A prova de que uma configuração com número par de moedas douradas não tem solução é bastante similar ao caso ímpar. Claramente um segmento sem moedas douradas não tem solução, pois nenhuma moeda pode ser retirada neste caso. Se o número de moedas douradas for par maior que zero, percebe-se que, independentemente da escolha da moeda dourada a ser retirada, sempre teremos como resultado ao menos um segmento não vazio também com número par de moedas. Dessa forma, qualquer sequencia de retiradas levará ao surgimento de um segmento sem moedas douradas, e portanto não há solução para este caso.

A primeira parte da prova nos dá um algoritmo linear para encontrar uma sequência válida de remoções: Retirar sempre a moeda dourada mais à esquerda. Mas é claro que esta solução não é sempre única. De fato, se enumerarmos somente as moedas douradas de 1 a K, podemos retirar qualquer moeda de índice ímpar pois os segmentos resultantes sempre terão um número ímpar de moedas douradas.