Seletiva para Maratona de Programação de 2014

Comentários sobre os problemas

Patrocínio:

Departamento de Ciência da Computação IME-USP

Problema A: Torneio de Yusuf II

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

Um torneio é um grafo completo com arestas orientadas. Sejam T um torneio e u e v vértices de $V(T)$. Dizemos que u vence v se a aresta uv está orientada de u para v. Um vértice v é chamado de rei se todos os outros vértices são vencidos por v ou são vencidos por algum vértice que é vencido por v. Em outras palavras, um vértice é rei se ele alcança os outros vértices usando até duas arestas.

O problema pede para encontrar uma ordenação dos vértices, $v_1, v_2, \ldots, v_n$, tal que, para todo $1 \leq i < n$, v_i vence v_{i+1} e v_i é rei no subtorneio T_i, obtido ao remover os vértices $v_1, \ldots, v_{i-1}$.

Solução do juiz

Dado um torneio T, denotaremos por $d^-(v)$ o grau de saída de um vértice $v \in V(T)$, ou seja, o número de arestas orientadas saindo de v.

Uma primeira coisa a se notar é que todo vértice com grau máximo de saída é um rei. Para verificar isso, sejam v um vértice com grau de saída máximo, $N^-(v)$ o subconjunto de vértices que são alcançados por v por uma aresta e $N^+(v)$ o subconjunto de vértices que alcançam v por uma aresta. Note que $N^+(v) \cup \{v\} \cup N^-(v) = V(T)$. Então, basta verificar que para todo vértice $v^+ \in N^+(v)$ existe $v^- \in N^-(v)$ tal que v^-v^+ é uma aresta orientada em T, caso contrário v^+ teria grau de saída maior que de v. Absurdo!

Então, podemos tomar v_1 como um vértice com grau de saída máximo em T, v_2 como um vértice com grau de saída máximo em T_2, e assim por diante. Assim, v_1,...,v_n será uma sequência onde cada v_i é um rei em T_i. Isso não é suficiente, pois poderia existir um v_j que não vence v_{j+1}. Neste caso, note que podemos trocá-los de ordem na sequência sem afetar o fato de serem reis nos subtorneios correspondentes. Logo, podemos fazer um procedimento no estilo *bubble sort* para ordenar a sequência de forma que, para todo $1 \leq i < n$, v_i vence v_{i+1} e v_i.

Solução da equipe Pesadelo Maximo

Em qualquer torneio T, a observação feita na seção anterior de que $N^+(v) \cup \{v\} \cup N^-(v) = V(T)$ na verdade vale para qualquer vértice v.

Seja v um vértice qualquer de T. Seja $i = |N^+(v)|$. Claramente, v é rei no subtorneio $T - N^+(v)$, cujo conjunto de vértices é $N^-(v) \cup \{v\}$. Logo, podemos tomar $v_{i+1} = v$ e repetir o procedimento recursivamente para $N^+(v)$ e $N^-(v)$. Note que todo vértice de $N^+(v)$ alcança qualquer vértice de $N^-(v)$ com duas arestas através de v. Portanto, esse procedimento no estilo *quicksort* também funciona.

Problema B: Desmascarando o empregado do sultão

Autor do problema: Gabriel Peixoto
Análise: Gabriel Peixoto

A informação do i-ésimo relatório pode ser representada pela equação:

$$A_{i,1}d_1 + A_{i,2}d_2 + \ldots A_{i,N}d_N = T_i - S_i + 1 \qquad (\text{mod P}),$$

onde d_j representa a duração (em horas) da tarefa j. Dessa forma temos um sistema de N equações e N incógnitas que queremos resolver.

A observação chave desse problema é que, quando P é primo, o conjunto $\{0, 1, \ldots, P-1\}$, munido das operações de soma e multiplicações mod P, é um corpo[1]. Isso é, é um conjunto que possui operações de soma e adição comutativas e associativas, com um elemento neutro e inverso para soma e multiplicação.

Toda a teoria de álgebra linear pode ser feita sobre corpos, em vez de números reais. Em particular você pode aplicar o método da eliminação gaussiana[2] (também conhecido como escalonamento de matrizes) para resolver esse sistema de equações em tempo $\mathcal{O}(N^3)$.

A única diferença notável é que sempre que formos dividir algo por um número b, nós devemos multiplicar pelo inverso multiplicativo de b. Como $P < 24$, todos os inversos multiplicativos podem ser calculados por força bruta $\mathcal{O}(P^2)$ no início da execução e armazenados.

[1] `http://en.wikipedia.org/wiki/Field_(mathematics)`
[2] `http://en.wikipedia.org/wiki/Gaussian_elimination`

Problema C: O verdadeiro valor dos tapetes árabes

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

Dado um grafo orientado com pesos nas arestas, $G = (V, A, c)$, queremos encontrar o valor mínimo de um circuito (orientado) de G. Seja $\mathcal{C}(G)$ o conjunto de todos os circuitos de G. O valor de $C \in \mathcal{C}(G)$ é dado por

$$\text{valor}(C) = \frac{\sum_{uv \in A(C)} \text{c}(uv)}{|A(C)|}.$$

Note que somando ε ao peso de todos os arcos, o novo valor de C passa a ser

$$\frac{\sum_{uv \in A(C)} \text{c}(uv) + \varepsilon}{|A(C)|} = \frac{|A(C)|\varepsilon + \sum_{uv \in A(C)} \text{c}(uv)}{|A(C)|} = \text{valor}(C) + \varepsilon.$$

Logo, somando $\varepsilon = -\min_{C \in \mathcal{C}(G)} \text{valor}(C)$ ao peso de todos os arcos, temos que o menor valor de um circuito será zero. Ademais, tal circuito de valor zero também tem valor mínimo no grafo original.

Sejam $\varepsilon \geq 0$ e $H = (V, A, c - \varepsilon)$ obtido ao subtrair ε ao peso de todos os arcos em A. Sabemos que G não possui circuitos negativos, então, se H possui circuito negativo, significa que $\min_{C \in \mathcal{C}(H)} \text{valor}(C) < 0$ e $\varepsilon > \min_{C \in \mathcal{C}(G)} \text{valor}(C)$. Se H não possui circuito negativo e tem circuito de valor zero, então $\varepsilon = \min_{C \in \mathcal{C}(G)} \text{valor}(C)$. Finalmente, se H não possui circuito negativo e o valor mínimo de um circuito em H é positivo, então $\varepsilon < \min_{C \in \mathcal{C}(G)} \text{valor}(C)$. Logo, o problema pode ser resolvido fazendo uma busca binária em ε[3].

Para determinar se um grafo tem circuitos negativos, pode-se utilizar o algoritmo de Bellman-Ford[4]. Por esse algoritmos também podemos descobrir se um grafo tem ou não circuitos e qual o menor peso de um circuito no grafo.

Uma outra solução seria calcular $d_k(v)$, o peso mínimo de um passeio com exatamente k arcos e terminando em v, para todo vértice v e $k \leq n = |V|$.

$$d_0(v) = 0 \quad \text{e} \quad d_{k+1}(v) = \min\{d_k(u) + c(uv) \mid uv \in A\}$$

O valor mínimo de um circuito é dado por

$$\min_{C \in \mathcal{C}(G)} \text{valor}(C) = \min_{v \in V} \max_{0 \leq k \leq n-1} \frac{d_n(v) - d_k(v)}{n - k}.$$

A prova disso pode ser encontrada no livro *Combinatorial Optimization*, de Alexander Schrijver.

A solução por busca binária pode ser implementada com complexidade $\mathcal{O}(NM \log K)$, onde K é o peso máximo de um circuito. A segunda solução pode ser implementada com complexidade $\mathcal{O}(NM)$. Ambas eram aceitas. Infelizmente, por falta de atenção minha, os limites do problema ficaram altos ($N \leq 1000$) e alguns times deixaram de tentar este problema por achar que precisava de um algoritmo com complexidade melhor.

[3] Note que o valor de um circuito não é necessariamente inteiro, logo é preciso tomar cuidados para a busca binária terminar. Mais detalhes podem ser encontrados no livro *Combinatorial Optimization: Networks and Matroids*, de Eugene Lawler

[4] `http://en.wikipedia.org/wiki/Bellman-Ford_algorithm`

Problema D: Picos do Átlas

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

Dada uma matriz $N \times M$, o problema pede para imprimir a posição dos seus picos. Um pico é uma posição da matriz com valor **maior** que de seus vizinhos. Cada posição da matriz tem 8 vizinhos, exceto pelas posições na borda da matriz, que tem menos.

Tudo que precisava ser feito era percorrer a matriz na ordem indicada no enunciado e imprimir as posições dos picos encontrados. Se a matriz não tem picos, deveria ser impresso o valor -1.

Note que se $N = M = 1$, a matriz só tem a posição $(1, 1)$. Tal posição tem zero vizinhos, logo ela tem valor maior que de todos os seus vizinhos.

Problema E: O gato do zelador do armazém

Autor do problema: Gabriel Peixoto
Análise: Gabriel Peixoto

Para resolver esse problema, é interessante que o competidor conheça conceitos e algoritmos fundamentais de grafos e tenha familiaridade com a manipulação de máscaras de bits. Como referência recomendo a leitura dos seguintes artigos no topcoder:

- Introduction to Graphs and Their Data Structures - Section 1[5]
- Introduction to Graphs and Their Data Structures - Section 2[6]
- Introduction to Graphs and Their Data Structures - Section 3[7]
- A bit of fun: fun with bits[8]

Numere as portas arbitrariamente de 0 até $p-1$, onde p é o número de portas no tabuleiro. Cada possível estado do tabuleiro pode ser representado por 5 inteiros. Dois deles representam a posição do gato, outros dois a posição do bloco e o último é uma máscara de bits que representa quais portas estão abertas.

Dessa forma temos no máximo $N^2 \times M^2 \times 2^p$ estados possíveis, o que totaliza $12,5$ milhões no pior caso para os limites desse problema.

Podemos pensar que cada um desses estados é um vértice de um grafo dirigido, e um vértice alcança o outro se ele pode ser alcançado por exatamente um movimento do jogador, seja esse movimento andar para uma posição livre (possivelmente abrindo uma porta) ou empurrar o bloco em uma direção.

O número de arestas pode ser limitado por 4 vezes o número de vértices, dessa forma uma busca em largura nele rodaria em tempo linear no número de vértices, e seria rápida o suficiente.

Os iniciantes devem notar que não precisamos armazenar o grafo explicitamente, apenas a distância para os estados visitados. Também um cuidado deve ser tomado porque queremos minimizar o número de portas abertas primeiro e depois o número de movimentos.

[5] http://help.topcoder.com/data-science/competing-in-algorithm-challenges/algorithm-tutorials/introduction-to-graphs-and-their-data-structures-section-1/

[6] http://help.topcoder.com/data-science/competing-in-algorithm-challenges/algorithm-tutorials/introduction-to-graphs-and-their-data-structures-section-2/

[7] http://help.topcoder.com/data-science/competing-in-algorithm-challenges/algorithm-tutorials/introduction-to-graphs-and-their-data-structures-section-3/

[8] http://help.topcoder.com/data-science/competing-in-algorithm-challenges/algorithm-tutorials/a-bit-of-fun-fun-with-bits/

Problema F: Partição do rebanho

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

Dado um multiconjunto X com N inteiros, queremos encontrar o valor mínimo de

$$S(A,B) = \sum_{a_1,a_2 \in A} \mid a_1 - a_2 \mid + \sum_{b_1,b_2 \in B} \mid b_1 - b_2 \mid,$$

onde $\{A, B\}$ é uma partição de X, isto é, $A \cup B = X$ e $|A| + |B| = N$.

Uma primeira ideia seria um algoritmo guloso que ordena os inteiros e escolhe um ponto para separar os inteiros a esquerda e a direita desse ponto. Isso não funciona. Tome a instância com os seguintes inteiros: 0, 3, 3, 3 e 6. A única partição ótima é $\{0, 6\}$ e $\{3, 3, 3\}$ com valor 6.

Seja X um multiconjunto, denotamos por Y o conjunto de valores distintos em X e denotamos por $\mu(y)$ o número de cópias de y em X. Vamos supor que $|Y| \geq 2$, caso contrário a resposta trivialmente seria 0.

Considere que $Y = \{y_1, y_2, \ldots, y_m\}$ com $y_1 < y_2 < \cdots < y_m$. Para $1 \leq i \leq m$, definimos um multiconjunto X_i tal que $Y_i = \{y_1, y_2, \ldots, y_i\}$, $\mu_i(y_i) = \sum_{j=i}^{m} \mu(y_j)$ e $\mu_i(y_j) = \mu(y_j)$, para $j < i$. Basicamente, X_i é uma nova instância onde toda cópia de y_j, para $j > i$, é transformada em y_i. Por exemplo, para $X = \{0, 3, 3, 3, 6, 7, 7\}$ teríamos $X_3 = \{0, 3, 3, 3, 6, 6, 6\}$.

Para $0 \leq p \leq \sum_{j=1}^{i-1} \mu(y_j)$ e $0 \leq r \leq \mu_i(y_i)$, seja $S(i, p, r)$ o valor de $\min_{A_i,B_i} S(A_i, B_i)$, onde $\{A_i, B_i\}$ particiona X_i colocando em A_i exatamente p inteiros com valores em $\{y_1, y_2, \ldots, y_{i-1}\}$ e r cópias de y_i. Então, tomando $q = \sum_{j=1}^{i-1} \mu(y_j) - p$ e $t = \mu_i(y_i) - r$,

$$S(i,p,r) = \begin{cases} 0, & \text{se } i = 1 \\ (y_i - y_{i-1})(pr + qt) + \min\limits_{0 \leq r' \leq \mu(y_{i-1})} \big(S(i-1, p-r', r+r')\big) & \text{c.c.} \end{cases}$$

A ideia da recorrência acima é calcular $S(i, p, r)$ transformando toda cópia de y_i em y_{i-1}. Note que independentemente de como é a partição ótima na instância transformada, a diferença entre o valor dessa partição e o valor da partição correspondente na instância original é sempre

$$(y_i - y_{i-1})(pr + qt),$$

que corresponde a contribuição de cada cópia de y_i que é perdida com a transformação.

Portanto, o problema pode ser resolvido usando programação dinâmica para calcular

$$\min_{p,r} S(m, p, r)$$

com complexidade de tempo de $\mathcal{O}(N^3|Y|)$.

Problema G: Hiperprimos

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

Um número inteiro n é um hiperprimo se o número de divisores de n é primo. Sabemos que qualquer inteiro n pode ser escrito na forma

$$n = p_1^{a_1} p_2^{a_2} \cdots p_k^{a_k},$$

onde p_i é um primo e $a_i > 0$ para $1 \leq i \leq k$. Logo, o número de divisores de n é

$$(a_1 + 1)(a_2 + 1) \cdots (a_k + 1).$$

Assim é fácil ver que, se $k \geq 2$, então n não pode ser um hiperprimo. Portanto, n é um hiperprimo se e somente se $n = p^a$ para algum p primo e algum $a + 1$ primo.

Dessa forma, bastaria gerar todos os hiperprimos menores que 10^6 e guardar, na posição i de um vetor, a quantidade de hiperprimos no intervalo $[2, i]$. Assim, para cada valor da entrada, bastaria imprimir o valor da posição correspondente desse vetor.

Muitas submissões receberam a resposta *Time limit exceeded*. Entre os principais motivos estão:

- não guardar a resposta em um vetor e ficar verificando para toda entrada quais números no intervalo $[2, N]$ são hiperprimos;
- para todo número no intervalo $[2, N]$, encontrar sua fatoração em primos e calcular seu número de divisores;
- gerar os números primos verificando para cada número entre 2 e 10^6, se ele possuia ou não um divisor menor que sua raiz.

O método citado no último item pode ser bom para verificar se um único inteiro é primo, mas ele é lento se quisermos encontrar todos os número primos menores que um $N_{\max}$. Um algoritmo mais rápido e também bem simples é o crivo de Erastóstenes[9].

[9] `http://pt.wikipedia.org/wiki/Crivo_de_Eratóstenes`

Problema H: Canais de qanat

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

Este problema mistura geometria e busca de padrões. No caso, o canal aberto (Q) é o padrão procurado e o canal fechado (P) é o "texto" onde procuramos o padrão. O algoritmo ingênuo que translada Q até um vértice de P e o rotaciona para tentar "encaixá-lo" e repete o processo até conseguir, não resolve o problema dentro do limite de tempo.

Para passar o problema era preciso utilizar um algoritmo mais eficiente, como o KMP[10]. Não explicarei o KMP aqui, mas mostrarei como aplicá-los a este problema.

Na entrada, P e Q são dados pelas coordenadas de seus vértices. Essa não é uma boa descrição desses objetos para este problema, pois, após aplicações de operações de translação e rotação, a descrição muda apesar do objeto em si não mudar. Uma descrição de P e Q que seja invariante às operações de translação e rotação é dada por uma sequência $a_1, \alpha_1, a_2, \alpha_2, \ldots$, onde cada a_i é o comprimento da i-ésima aresta e α_i o angulo interno entre a_i e a próxima aresta.

Aplicar o KMP nessa sequência resolveria o problema. No entanto, como os ângulos não são necessariamente inteiros, é preciso trabalhar com ponto flutuante e, nesse caso, deve-se tomar cuidado com precisão numérica.

Uma outra forma de descrever P e Q, mas que utiliza apenas números inteiros é a seguinte. Para cada vértice v mantemos os vetores $\vec{e}_v = (e_x, e_y)$ e $\vec{s}_v = (s_x, s_y)$ correspondendo, respectivamente, à aresta que entra entra em v e a aresta que sai de v (ver figura 1). No caso de Q, o primeiro vértice não possui $\vec{e}$ e o último não possui $\vec{s}$. Em vez de armazenar o ângulo α_v, armazenamos $\vec{e}_v \cdot \vec{s}_v$ e

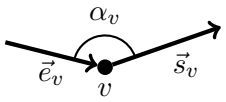

Figura 1: Componentes da descrição de um vértice.

$\vec{e}_v \times \vec{s}_v$, respectivamente, os produtos escalar e vetorial de $\vec{e}_v$ e $\vec{s}_v$. No caso do produto vetorial, calculamos apenas o valor da terceira coordenada, pois as outras duas valem 0 para vetores no $\mathbb{R}^2$. Note que

$$\vec{e}_v \cdot \vec{s}_v = e_x s_x + e_y s_y = ||\vec{e}_v||\,||\vec{s}_v|| \cos(\alpha_v) \text{ e}$$
$$\vec{e}_v \times \vec{s}_v = (0, 0, e_x s_y - e_y s_x) = ||\vec{e}_v||\,||\vec{s}_v|| \sin(\alpha_v)\,(0, 0, 1).$$

Logo, para vértices v e v', podemos verificar se α_v é igual a $\alpha_{v'}$ comparando os valores desse produtos escalar e vetorial. Então, consideramos que um vértice v de P é igual a um vértice v' de Q se

$$||\vec{e}_v|| = ||\vec{e}_{v'}||, \quad ||\vec{s}_v|| = ||\vec{s}_{v'}||, \quad \vec{e}_v \cdot \vec{s}_v = \vec{e}_{v'} \cdot \vec{s}_{v'} \quad \text{and} \quad \vec{e}_v \times \vec{s}_v = \vec{e}_{v'} \times \vec{s}_{v'}.$$

Como o primeiro e o último vértices de Q não possuem um dos vetores, $\vec{e}$ ou $\vec{s}$, podemos ignorá-los e comparar apenas os outros vértices de Q.

Dessa forma, basta aplicar o KMP considerando que a sequência de vértices de P corresponde ao "texto" e a sequência de vértices de Q corresponde ao padrão procurado. Como P forma uma sequência circular, copiamos os primeiros $|Q|$ vértices de P ao final da sequência de P.

[10] `http://en.wikipedia.org/wiki/Knuth-Morris-Pratt_algorithm`

Problema I: *Pair-voting* no conselho de Gueliz

Autor do problema: Renzo Gomez
Análise: Marcio T. I. Oshiro

Lendo o enunciado do problema com cuidado, percebemos que trata-se de um problema de emparelhamento perfeito em grafos. Isto é, queremos particionar os vértices em pares tal que cada par corresponda a uma aresta do grafo e cada vértice participe de exatamente um par. Existem vários algoritmos conhecidos para resolver problemas de emparelhamento em grafos, no entanto, todos esses algoritmos são lentos demais para este problema.

Note que a maneira como as ruas são formadas indicam que o grafo em questão possui uma estrutura especial, ele é na verdade uma árvore (um grafo conexo e sem circuitos). Isso é evidenciado pelo comentário de que não existem quarteirões no bairro. Dado que o grafo é uma árvore, a tarefa de encotrar emparelhamentos é muito mais simples. Existem várias maneiras de resolver este problema, utilizando, por exemplo, busca em profundidade ou largura.

A principal coisa a se notar é que, se uma árvore T possui emparelhamento perfeito, então suas folhas (vértices com apenas uma aresta incidente) estão, por falta de opção, emparelhadas com os mesmos vértices em qualquer emparelhamento perfeito de T. Seja T' a subárvore T após removermos todas as folhas e os vértices adjacentes às folhas, com exceção de folhas adjacentes ao vértice 1 (ver figura 2). Note que T' é uma árvore e possui emparelhamento perfeito se e somente se T também tem. Ademais, sabemos com quem as folhas de T' devem estar emparalhadas em qualquer emparelhamento perfeito de T'. Repetindo esse raciocínio, podemos encontrar, caso exista, um emparelhamento perfeito de T.

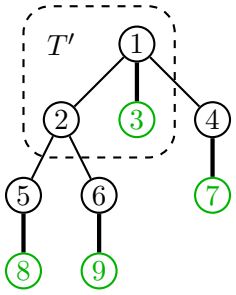

Figura 2: Vértices verdes são folhas. Arestas destacadas indicam vértices emparelhados.

Um algoritmo que procura um emparelhamento perfeito em árvores começando pelas folhas pode ser implementado em tempo linear no número de vértices (é basicamente o processo descrito no parágrafo anterior). Com tudo isso, é fácil mostrar que, se uma árvore tem emparelhamento perfeito, então ele é único. Logo, não precisamos nos preocupar com a parte do enunciado que diz para escolher o emparelhamento perfeito lexicograficamente menor, no caso de existir mais de um.

Tudo isso funciona muito bem quando N é par. Agora, quando N é ímpar, podemos utilizar o mesmo algoritmo, apenas tomando cuidado para que o vértice 1 seja o único sem par. Note que para o exemplo da figura 2, não existe um emparelhamento como pedido pelo problema, pois, o vértice que fica sem par é o 2.

Problema J: Montando sua própria cáfila

Autor do problema: Gabriel Peixoto
Análise: Gabriel Peixoto

Vamos denotar por $f(m_1, m_2)$ o número esperado de lotes que precisamos comprar para adquirir dromedários com m_1 características desejadas e camelos com m_2 características. A resposta do problema é dada por $f(M_1, M_2)$.

Denote por $g(m_1, m_2, a, b)$ a probabilidade de que em um lote tenhamos a características de dromedários desejadas (entre as m_1) e b características de camelo desejadas (entre as m_2). Queremos montar uma recorrência para f, para isso note o caso base $f(0,0) = 0$. Enquanto que para $(m_1, m_2) \neq (0,0)$ vale que:

$$\begin{aligned} f(m_1, m_2) &= \sum_{a=0}^{3} \sum_{b=0}^{3-a} \left[1 + f(m_1 - a, m_2 - b)\right] g(m_1, m_2, a, b) \\ &= 1 + \sum_{a=0}^{3} \sum_{b=0}^{3-a} f(m_1 - a, m_2 - b) g(m_1, m_2, a, b). \end{aligned}$$

Isolando o termo onde $(a, b) = (0, 0)$, passando-o para o outro lado da igualdade, podemos escrever:

$$f(m_1, m_2) = \frac{1}{1 - g(m_1, m_2, 0, 0)} \left[1 + \sum_{a=0}^{3} \sum_{\substack{b=0 \\ (a,b)\neq(0,0)}}^{3-a} f(m_1 - a, m_2 - b) g(m_1, m_2, a, b) \right]$$

Com essa relação podemos montar uma solução em programação dinâmica ou memorização para encontrar o valor de $f(M_1, M_2)$, desde que saibamos calcular os valores de g. Vamos atacar esse problema agora.

Primeiramente note que a probabilidade de termos i dromedários e $3 - i$ camelos, para $i \in \{0, 1, 2\}$, em um lote é dada por:

$$p(i) := \frac{\binom{3}{i} P_1^i P_2^{3-i}}{1 - P_1^3}.$$

Agora imagine que um lote tem i dromedários, temos m_1 dromedários com características desejáveis, e queremos saber a probabilidade de que obtenhamos a dessas características nesse lote. Vamos chamar essa probabilidade de $h_1(m_1, i, a)$. Ela pode ser calculada através da recorrência:

$$h_1(m_1, i, a) := \begin{cases} 0 & \text{se } a > i \text{ ou } a > m_1, \\ \left(\frac{N_1 - m_1}{N_1}\right)^i & \text{se } a = 0, \\ \frac{m_1}{N_1} h_1(m_1 - 1, i - 1, a - 1) + \frac{N_1 - m_1}{N_1} h_1(m_1, i - 1, a) & \text{caso contrário.} \end{cases}$$

Analogamente podemos calcular h_2. Com ambas escritas podemos calcular g. Comecemos com os casos base, $g(m_1, m_2, a, b) = 0$ sempre que $a > m_1$ ou $b > m_2$. Caso contrário g satisfaz a relação:

$$g(m_1, m_2, a, b) := \sum_{i=0}^{2} p(i) h_1(m_1, i, a) h_2(m_2, 3 - i, b).$$

Problema K: As dicas de Ali Babá

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

Este problema é bem fácil, mas uma lida rápida do enunciado pode acabar enganando o leitor, fazendo o problema parecer mais complicado do que ele realmente é.

A princípio pode parecer que a falta de informação sobre quais operações foram aplicadas dificulta a resolução do problema, mas na verdade essa informação é irrelevante. Basta notar uma propriedade que as operações "duplicação" e "espelhamento" têm em comum: toda cópia de um número é posicionado sempre a direita do original. Portanto, para obter a permutação original, bastaria percorrer a sequência, da esquerda para a direita, imprimindo a primeira ocorrência de cada número.

Este problema é uma adaptação do problema *Copier*[11] do IPSC deste ano.

[11] `http://ipsc.ksp.sk/2014/real/problems/c.html`