# Tutorial: Athleta Alemão

Guilherme Ramos

Primeiramente, leia a meta para comparativo. Na sequência, leia o tempo de volta de cada linha enquanto forem dadas. Para cada tempo, compare com a meta e, se for maior que 105% apresente "Athelera, fera!", se for inferior a 95% apresente "Calma que ainda tem o returno.". Caso contrário, mostre "Boa! Merece cafe e pao de queijo."

# Tutorial: Balões de Festa

Arthur Botelho

A melhor opção é todas as turmas ficarem com a mesma quantidade de balões que a turma que trouxe menos. Dessa forma, a quantidade total de balões descartados é, sendo $M$ o menor entre os valores $B_i$: soma de $B_i - M$ para todo $i$.

# Tutorial: Cafezim

Daniel Porto

Dadas as 3 variáveis de entrada: $n$ = número de professores (ou xícaras disponíveis), $k$ = número alvo, e $v$ = volume total de café disponível, podemos tentar algumas opções.

Tentar a força bruta testando todas as soluções (seja da menor para a maior ou o contrário) não é a melhor estratégia. Uma opção mais eficiente é ir tentando "adivinhar" um valor para a nossa resposta.

O processo de adivinhação se dará por tentativa e erro. O primeiro valor que testaremos é a metade do volume total disponível. Caso seja possível colocar $\frac{v}{2}$mL de café, podemos tentar colocar mais. Caso contrário, tentaremos colocar menos. Para os próximos valores a tentar "adivinhar", podemos tentar a metade entre $\frac{v}{2}$ e $v$, caso tenha sido possível colocar $\frac{v}{2}$ml de café; caso contrário, podemos tentar colocar a metade entre 0 e $\frac{v}{2}$. A ideia é ir tentando "adivinhar" a quantidade máxima que é possível colocar usando essa estratégia de **busca binária**. Assim, chegaremos na resposta com uma complexidade $O(n \log n)$, devido ao custo $O(n)$ de verificar se um certo volume é possível de ser colocado nas xícaras.

De forma alternativa, podemos pensar na forma da resposta $x$. É certo que essa resposta sempre é no máximo $v/k$ (arredondando para o menor inteiro), pois todas as xícaras devem ter a mesma quantidade. Além disso, ao escolher $k$ xícaras para servir, a melhor opção é sempre pegar as de maior capacidade, pois queremos servir o maior volume possível. Dentre as $k$ maiores capacidades, seja $m$ a menor delas. É certo também que a resposta $x$ é no máximo $m$, pois as xícaras não podem transbordar. A resposta, portanto, pode ser expressa por $min(v/k, m)$, de forma que conseguimos servir as $k$ maiores xícaras com o máximo possível de café.

O que limita a complexidade desta última solução é achar as $k$ maiores xícaras. Isso pode ser feito por meio de um algoritmo de ordenação eficiente ($O(n \log n)$ ou próximo disso).

# Tutorial: Dias de Viagem

Eduardo Freire

Representaremos a quantidade de passeios entre cada par de cidades no $d$-ésimo dia por uma matriz $3 \times 3$ chamada $M_d$. Convencionando que as cidades $R$, $Q$ e $I$ serão representadas pelos números 0, 1 e 2, respectivamente, construímos a matriz de tal forma que sua entrada na $i$-ésima linha e $j$-ésima coluna seja a quantidade de passeios da cidade $i$ para a cidade $j$. Note que, como sempre é possível não realizar um passeio em um dado dia, a diagonal principal dessa matriz consiste apenas do número 1. Além disso, essa matriz é simétrica.

Para responder à ação do tipo 1, basta multiplicar as matrizes no intervalo $[L, R]$. A matriz resultante $M$ terá na sua entrada $M[i][j]$ a quantidade de maneiras distintas de realizar uma sequência de passeios que parte da cidade $i$ e termina na cidade $j$. Para realizar a ação do tipo 2, basta atualizar a matriz do $i$-ésimo dia da maneira como foi especificado.

Como a operação de multiplicação de matrizes é associativa, podemos implementar essas duas ações utilizando uma árvore de segmentos. Já que a multiplicação de matrizes feita de maneira usual necessita de $\Theta(D^3)$ operações em que $D$ é a dimensão da matriz (no nosso caso, $D = 3$), a complexidade dessa solução é $O(D^3(N + Q\log(N)))$.

# Tutorial: Espera

Jeremias Moreira Gomes

Seja $m$ o número de caracteres únicos da string, e $F$ o cálculo da frequência com que cada um desses caracteres $v$ aparece na string. A contribuição $X$, ou o número de ocorrências de um caractere fora da sua posição, é dada por:

$$X_i = \mathrm{F}(v_i) \cdot (\mathrm{F}(v_i) - 1)$$

E a resposta $R$ é a soma dessas contribuições:

$$R = \sum_{i=1}^{m} \mathrm{X}_i$$

# Tutorial: Fura-Cão

Guilherme Ramos

Basta ler a entrada e verificar se existem as palavras "athletico" ou "furacao" para identificar que a mensagem é favorável. Caso contrário, a mensagem é sobre o rival.

# Tutorial: Guardadim

Edson Alves

Este problema pode ser convertido em um problema de se determinar a submatriz de soma máxima. Para isso, o primeiro passo é transformar a matriz $C$ dada na entrada em uma matriz $M_{H\times W}$ tal que $m_{ij} = 1$ se $A \leq c_{ij} \leq B$ e $m_{ij} = 0$, caso contrário.

Agora, para cada par $(i, j)$, com $i \leq j$, mantemos um vetor de somas acumuladas $p$ tal que

$$p_k = \sum_{s=i}^{j} m_{ks},$$

ou seja, o vetor $p$ contém as somas acumuladas das linhas entre as colunas $i$ e $j$. Se, para cada coluna $i$, forem processadas as colunas $j$ de $i$ a $W$, nesta ordem, é possível computar computar $p$ em $O(HW^2)$.

Para cada par de colunas $(i, j)$, podemos computar um vetor $q$ tal que $q_k = j - i + 1$, se $p_k = j - i + 1$, e $q_k = 0$, caso contrário. Aplicando uma variante do algoritmo de Kadane (na qual a soma parcial é zerada se encontramos um valor igual a zero no vetor), ou usando dois ponteiros, é possível encontrar o intervalo $(a, b)$ de soma máxima em $q$. O retângulo cujo vértice superior esquerdo é $(i, a)$ e vértice inferior direito $(j, b)$ é um candidato a solução, e a solução será, dentre os candidatos, a de maior área.

Esta solução tem complexidade $O(HW^2)$ e memória $O(HW)$.

# Tutorial: Ironman

Daniel Porto

**Solução 1**: podemos abordar esse problema testando, para todo $i$, qual seria o $j > i$ que maximizaria o valor de $F_j - F_i$. É óbvio que, para um valor fixo de $i$, este $j$ é simplesmente o maior valor de $F_j$ para $j > i$. Dessa forma, podemos iterar a lista $F$ pelas posições de $N$ até 1 e ir mantendo o maior valor até cada momento da iteração (o maior $F_j$). Fazendo isso, podemos testar para todo $i$ o maior valor possível de $F_j - F_i$, e para a resposta final imprimimos o maior dos valores, ou 0 caso todos sejam negativos.

**Solução 2**: a abordagem de divisão e conquista também pode ser utilizada. Podemos iniciar com a lista $F$ toda e calcular recursivamente, para cada uma de suas metades , a maior diferença encontrada e tanto o menor quanto o maior de $F_i$ encontrados nesta metade. O caso base, quando a metade chega a ter apenas uma posição, é apenas 0 como a menor diferença e o próprio valor da posição como menor e maior. Tendo o cálculo dessas informações para as duas metades, pode-se uní-las:

- A maior diferença é o máximo entre as maiores diferenças das metades da esquerda e direita e a diferença entre o maior valor da metade da direita e o menor valor da metade da esquerda.
- Os menores e maiores valores são simplesmente o mínimo e máximo entre os valores calculados nas metades

Ao final, teremos, para toda a lista $F$, a maior diferença, o menor valor e o maior valor. Imprimimos apenas a maior diferença, ou 0 caso seja negativa.

# Tutorial: João Grilo

Edson Alves

Os valores investidos por Borges a cada mês formam a sequência $x_i \equiv K^i \pmod{B}$ e a solução do problema é o menor inteiro $j$ tal que $x_j \equiv 1 \pmod{B}$. Esta última equação terá solução se, e somente se, $K$ é invertível módulo $B$, ou seja, se $mdc(K, B) = 1$.

Os elementos invertíveis módulo $B$ formam um grupo multiplicativo cuja ordem é igual a $M = \varphi(B)$. As potências de $K$ formam um subgrupo de $B$ de ordem $m$. Assim, vale que $m$ divide $M$ e esta ordem corresponde ao inteiro $j$ descrito anteriormente.

Uma solução possível consiste em avaliar $K^d \pmod{B}$ para todos os divisores $d$ de $M$. Esta solução tem complexidade $O(\sqrt{B} + \log M \times \sqrt{M})$.

# Tutorial: Mineirinho

Daniel Saad Nogueira Nunes

Esse problema pode ser resolvido através do paradigma de divisão e conquista, de maneira similar ao do problema de encontrar a menor distância entre os dois pontos. Seja $P_x$ o conjunto de pontos ordenados pela coordenada $x$. Separamos $P_x$ em duas partições, separadas conforme a mediana de $P_x$: $P'_x$, e $P''_x$. Então, nós resolvemos o problema recursivamente para cada partição, obtendo o menor perímetro de qualquer triângulo que possa ser formado por pontos de cada partição. Seja $p$ o menor perímetro dentre as duas soluções encontradas, precisamos verificar se existe algum triângulo com perímetro menor que $p$ que tenha pelo menos um ponto de cada partição, visto que queremos contemplar triângulos que não foram formados ainda. Para isso, consideramos os pontos que estão próximos à linha de separação entre as duas partições, isto é, filtramos de $P_x$ todos os pontos que estão a uma distância, no máximo $p/2$ da mediana. Em outras palavras, construímos o seguinte conjunto:

$$P_y = \{p \in P_x \mid |p_x - m| \leq p/2\}$$

Caso $P_y$ esteja ordenado por $y$, para cada ponto $a \in P_y$, verificamos todos os pontos $b \in P_y$ e $c$ que estão a uma distância vertical de no máximo $p/2$ de $a$, visto que, se a distância vertical for maior, não há como o perímetro ser menor que $p$. Calculamos o perímetro formado pelos pontos $(a, b, c)$, e se a resposta por menor que $p$, atualizamos $p$. O detalhe é que existe apenas um número constante de pontos $b$ e $c$ que está a uma distância vertical de no máximo $p/2$ de $a$.

Isso garante a seguinte relação de recorrência de pior caso:

$$T(n) = \begin{cases} \Theta(1), & n < 3 \\ 2T(n/2) + \Theta(f(n) + n), & \text{caso contrário} \end{cases}$$

Em que $f(n)$ é o tempo gasto para ordenar $P_y$ por $y$. Se retornarmos das chamadas recursivas, o menor perímetro **E** os pontos da partição ordenados por $y$, conseguimos formar $P_y$ em tempo linear fazendo o *merge* dos dois conjuntos ordenados por $y$, formando assim, a relação de recorrência

$$T(n) = \begin{cases} \Theta(1), & n < 3 \\ 2T(n/2) + \Theta(n), & \text{caso contrário} \end{cases}$$

Que possui tempo de pior caso $\Theta(n \lg n)$. Abaixo segue o pseudocódigo do algoritmo. Aqui estamos assumindo que $P$, o conjunto de pontos, está ordenado por $x$.

**Algorithm 1:** SMALLEST-PERIMETER(P)

$n \leftarrow |P|$<br>
**if**( $n < 3$ )<br>
    $P_y \leftarrow P$ ordenado por $y$<br>
    **return** $(\infty, P_y)$;<br>
$med \leftarrow P[\lfloor n/2 \rfloor]$<br>
$(p_l, P'_y) \leftarrow$ SMALLEST-PERIMETER$(P[0, \lfloor n/2 \rfloor])$<br>
$(p_r, P''_y) \leftarrow$ SMALLEST-PERIMETER$(P[\lfloor n/2 \rfloor + 1, n-1])$<br>
$p \leftarrow \min(p_l, p_r)$<br>
$D_y \leftarrow$ MERGE$(P'_y, P''_y)$<br>
$P_y \leftarrow \{p \in D_y \mid |p.x - med.x| \leq p/2\}$<br>
**for**( $i \leftarrow 0$ **to** $|P_y| - 1$ )<br>
    **for**( $j \leftarrow i + 1$ **to** $|P_y| - 1$ )<br>
        **if**( $P_y[j].y - P_y[i].y > p/2$ ) **break**<br>
        **for**( $k \leftarrow j + 1$ **to** $|P_y| - 1$ )<br>
            **if**( $P_y[k].y - P_y[i].y > p/2$ ) **break**<br>
            $p \leftarrow \min(p,$ PERIMETER$(P_y[i], P_y[j], P_y[k]))$<br>
**return** $(p, P_y)$

# Tutorial: Netzwerk

Arthur Botelho

Esse problema pode ser resolvido com fluxo. Podemos criar um grafo com as cidades e as viagens de forma levemente modificada: para cada cidade, criaremos um vértice que recebe todas as suas arestas de chegada e outro do qual saem todas as suas arestas de partida, de forma que esses dois vértices estão ligados. A capacidade das arestas das viagens é definida como infinita, e o que importa para nós é a capacidade dessa aresta que liga o vértice de entrada ao vértice de saída — vamos chamar essas arestas de arestas especiais, e vamos modificar a sua capacidade ao longo da solução. Faremos também outra modificação no grafo: ligar todas as cidades com cargas a um vértice especial *source* e todas as cidades de destino a um vértice especial *sink* por arestas de capacidade 1.

Esse grafo representa que todas as cidades com cargas valiosas iniciam com 1 carga e todas as cidades de destino recebem apenas 1 carga. Se o fluxo desse grafo resultar em $K$, significa que todas as cargas foram transportadas com sucesso. Podemos iniciar com as arestas especiais tendo capacidade infinita para verificar se a resposta é $-1$. Agora, deve-se perceber que o fluxo que passa por uma aresta especial é exatamente o risco local definido no enunciado. Então, para minimizar o risco da operação inteira, podemos testar diferentes valores de capacidade para arestas especiais, verificando se ao final o fluxo resultante ainda é $K$.

Isso pode ser feito com busca binária (testando os valores de 1 a $K$). Porém, também podemos ir testando sequencialmente os valores de 1 até $k$ se aproveitarmos a configuração do fluxo resultante até o momento. Ou seja, inicialmente fazemos o fluxo com o valor 1 (passando uma unidade de fluxo), e depois do algoritmo terminar aumentamos a capacidade das arestas especiais em 1 e tentamos passar mais uma unidade de fluxo, e continuamos o processo até quando não seja possível passar mais fluxo.

Para este problema, é importante perceber que o custo de passar uma unidade de fluxo em quase todos os algoritmos é o mesmo de fazer uma travessia no grafo, de forma que uma das limitações da complexidade dos algoritmos é $F \cdot (V + V)$, onde $F$ é o fluxo máximo, $V$ é a quantidade de vértices e $E$ a de arestas. Portanto, utilizando busca binária a complexidade seria $K \cdot (N + M) \cdot \log K$, e a segunda abordagem teria complexidade $K \cdot (N + M)$.