Tutorial: Ambição a cafeína

José Leite

Modelagem inicial

Podemos modelar uma rede onde o fluxo máximo é a resposta do nosso problema.

Criamos um vértice s, com uma aresta para todo estande j com capacidade b_j. Criamos uma aresta entre todo estande j e todo amigo i com capacidade c_j. Por fim, criamos arestas de todo amigo i para um vértice t com capacidade a_i. O fluxo máximo nessa rede é a resposta do nosso problema.

O fluxo máximo de s a t é o mesmo que o menor corte de s a t no grafo. Podemos analisar um corte qualquer e fazer observações sobre ele.

Seja S o conjunto de vértices conectados com s num corte, T o conjunto de vértices conectados com t, E o conjunto de estandes, e A o conjunto de amigos. O custo desse corte será $\sum_{j \in E \cap T} b_j + \sum_{j \in E \cap S} \sum_{j \in A \cap T} c_j + \sum_{i \in A \cap S} a_i$. Seja $k = |A \cap T|$, temos $\sum_{j \in E \cap T} b_j + \sum_{j \in E \cap S} k \cdot c_j + \sum_{i \in A \cap S} a_i$.

Então, para um k fixo, qualquer estande j vai aparecer ou no primeiro somário $\sum_{j \in E \cap T} b_j$ ou no segundo $\sum_{j \in E \cap S} k \cdot c_j$, como queremos minimizar o corte vamos escolher o menor valor possível dentre os dois. A resposta, é então $\sum_{j \in E} \min(b_j, k \cdot c_j) + \sum_{i \in A \cap S} a_i$.

Por fim, se fixarmos $k = |A \cap T|$, então a soma $\sum_{j \in A \cap S} a_i$ será a soma de $N - k$ elementos de a. Como queremos minimizar o corte, escolhemos os $N - k$ menores deles. Vamos denotar por p_k a soma dos $N - k$ menores elementos de a.

Podemos guardar o corte para um k fixo $cut_k = p_k + \sum_{j \in E} \min(b_j, k \cdot c_j)$ para todo k de 0 a n. cut_k será o menor corte tal que $k = |A \cap T|$ e o menor de todos será o menor corte global.

Com isso podemos resolver cada update em $O(N)$ para uma complexidade total de $O(Q * N)$.

Otimizando atualizações

A cada atualização podemos retirar a contribuição do estande j, atualizar

os valores e adicionar a nova contribuição para cada cut_k. Vamos descrever como adicionar um estande.

Temos que somar $k \cdot c_j$ em cut_k para todo $k \leq \left\lfloor \frac{b_j}{c_j} \right\rfloor$ e somar b_j para todo $k > \left\lfloor \frac{b_j}{c_j} \right\rfloor$.

Vamos usar decomposição em raiz quadrada para dividir o array cut em blocos de tamanho B.

Dentro de cada bloco mantemos retas $y = k \cdot x + cut_k$ — comumente conhecido como convex hull trick, a menor reta no ponto $x = 0$ nos dá a melhor resposta dentro do bloco. Se quisermos somar $k \cdot c_i$ em todo o bloco, então a menor reta no ponto $x = c_i$ passa a nos dar a melhor resposta dentro do bloco.

Podemos, então, resolver o update da seguinte forma: caso um bloco esteja totalmente dentro do update, somamos numa variável lazy do bloco; caso o bloco esteja parcialmente dentro do update, atualizamos um a um o que precisar e reconstruimos o bloco em tempo linear. Portanto, um tempo total de $O(\frac{N}{B} + B)$.

Para o segundo update de somar b num intervalo, podemos a mesma estratégia com outra váriável lazy no bloco em $O(\frac{N}{B} + B)$.

Por fim, para recuperar o mínimo global, podemos iterar por todos os blocos e recuperar a melhor resposta de cada um. Neste passo, é necessário uma busca binária dentro de cada bloco para uma complexidade de $O(\frac{N}{B} \log(B))$.

A complexidade final é $O(Q * (B + \frac{N}{B} \cdot \log(N)))$, para $B = \sqrt{N \cdot \log(N)}$ temos $O(Q * \sqrt{N * \log(N)})$.

Tutorial: Baguete

Guilherme Ramos

A solução é simples, basta apresentar “TMJ!” caso a entrada fornecida seja “au-au”. Caso contrário, a entrada necessariamente será “rrrrr” e basta apresentar “Esta repreendida!”.

Tutorial: Coffee-Break Fit

Vinicius Borges

Uma estratégia por busca binária na resposta é capaz de resolver o problema. Para isso, deve-se ordenar um novo vetor contendo apenas os competidores que estão com lanche incompleto, considerando o valor excedente $a_i - b_i$ multiplicado pelo custo c_i. Suponha então que sejam K competidores com lanches incompletos.

Em seguida, devemos chutar o valor X e calcular o novo custo para cada competidor considerando as taxas impostas pela fornecedora. Para isso, faça $l = 1$ e $r = K$, e calcule $mid = (l + r)/2$. Sabendo que esse mid pode ser uma solução para o problema, verifique se o custo total de complementar o lanche fit excede o valor do orçamento M. Caso sim, guarde esse valor mid como uma resposta válida e tente aumentar o tamanho do chute X fazendo-se $l = mid + 1$. Caso contrário, deve-se diminuir o tamanho do chute para $r = mid - 1$ e tentar novamente.

A solução completa fica complexidade no tempo $O(N \log N)$ devido à operação de ordenação (a busca binária analisada separadamente possui complexidade $O(\log N)$.

Tutorial: dogsay

Guilherme Ramos

O primeiro passo é organizar a "imagem" do cachorro, copiando o formato de um dos exemplos. Apenas as 3 primeiras linhas precisam ser ajustadas com o texto fornecido (`msg`). Sendo L a quantidade de caracteres da mensagem, a primeira linha tem $2 + L$ caracteres '_' (com um espaço os antecedendo). A segunda linha apresenta a própria mensagem entre espaços e os sinais '`<`' e '`>`'. Por fim, a terceira linha se assemelha à primeira, apenas troca-se o sinal por '`=`'.

Tutorial: Estruturando o coffee-break

Daniel Saad Nogueira Nunes

Esse problema pode ser resolvido através do algoritmo de Duval, que obtém a fatoração **Lyndon** em $\Theta(n)$. Mas antes precisamos de duas definições:

Uma palavra **Lyndon** é aquela que é menor que todos os seus sufixos próprios. Por exemplo, `ababb` é uma palavra Lyndon, pois é menor do que:

- `babb`
- `abb`
- `bb`
- `b`

O período de uma string S, dado por X, é o menor prefixo de S tal que $S = X^k X'$, isto é, concatenações de k cópias de X com um prefixo X' (possivelmente vazio) de X. Por exemplo, se S é `abcab`, então seu período é `abc`.

Para realizar a fatoração Lyndon, que é o que se pede no problema, dividimos a string de entrada S em três partes, $S = S_1 S_2 S_3$. S_1 é o que já foi fatorizado. S_2 é o que está sendo fatorizado e S_3 é o que será fatorizado.

Gulosamente, tentamos estender S_2 o máximo possível, ao manter três índices, i, j e k. i é o início de S_2, j é o caractere que estamos avaliando se fará parte de S_2 ou não, e k sinaliza o período de S_2 de tal forma que $j - k$ é o tamanho do período.

Temos três situações:

- Se $S_j > S_k$, então podemos incluir S_j em S_2, isto é, estendemos S_2 e o período de S_2 é a própria string, portanto k passa a valer i.
- Se $S_j = S_k$, o período de S_2 aumenta de uma unidade, portanto incrementa-se k.
- Se $S_j < S_k$, não é possível estender S_2.

No terceiro caso, $S_2 = X^k X'$ para uma string X de período $j-k$. Assim, fatoramos S_2 como $|X|X|\ldots|X|$, deixando X' para a próxima iteração.

O algoritmo de Duval pode ser visualizado no seguinte link `https://cp-algorithms.com/string/lyndon_factorization.html`

Tutorial: Fenótipos

Guilherme Ramos

Para resolver este problema, basta ler as siglas dos três animais e verificar, para cada característica do filhote, se ela existe em qualquer um dos pais. Caso todas existam, o cachorrinho as herdou. Caso uma ou mais não existam, a ascendência não é certa.

Tutorial: Hurricane!

Daniel Porto

Pode-se usar uma pilha para armazenar monstros de cada jogador. Ao jogar uma carta *Hurricane!*, basta desempilhar (se possível) a carta da pilha do oponente.

Tutorial: Jornada à Fortaleza

José Leite

Este é um problema onde tentar generalizar um pouco pode ser útil para chegar em uma solução. Tentar pensar em k-caminhos, ao invés de 2, pode ajudar.

Cada aresta tem um custo e também uma capacidade, todo vértice também tem uma capacidade atrelada — dois para vértices 1 e N ou um para o resto. Esses são temas comuns em problemas de fluxo.

Neste problema queremos passar duas unidades de fluxo do vértice 1 ao vértice N com o menor custo possível. Para isso, podemos criar um vértice s, com uma nova aresta de s para 1, e um vértice t, com uma nova aresta de N para t. Ambas as arestas com custo 0 e capacidade 2. O min-cost max-flow de s para t será a resposta.

Para limitar o número de vezes que passamos por um vértice, podemos usar um truque bem comum em problemas de fluxo. Dividimos todo vértice u em dois vértices u_{in} e u_{out}. Toda aresta que chegava em u, agora vai chegar em u_{in}. Toda aresta que saía de u, agora saí de u_{out}. Adicionalmente, criamos um aresta entre u_{in} e u_{out} com custo 0 e capacidade desejada — dois ou um.

Para calcular o min cost flow, podemos encontrar caminhos usando dijkstra se mantermos potenciais de Johnson. Neste problema, os custos da rede inicial são todos não-negativo, então podemos inicializar todos os potenciais como zero. Ao final, só precisamos recuperar ambos os caminhos utilizados.

Complexidade $O(M * \log(N))$ para executar dijkstra com potenciais duas vezes.

Tutorial: K-ésimo Kara Kickado

Alberto Neto

Este problema é uma variação do problema de Josephus.

Escreva os números na base $(m + 1)$. Ao passar uma vez pela lista, vamos estar excluindo todos os números com dígito menos significativo diferente de m. Note que, ao passar pela segunda vez na lista, o primeiro ciclo de exclusão possivelmente será menor por termos excluido no final da lista. Seja q o número de exclusões deste ciclo antes de dar a volta. Agora, vamos excluir todos os números com segundo dígito menos significativo diferente de $(m - q)$.

Este algoritmo pode ser simulado com uma função recursiva, que recebe como argumento a quantidade n de pessoas ainda na lista, o parâmetro m, quantos ainda devem ser excluidos e o parâmetro q. É possível calcular em O(1) quantas pessoas serão excluidas nesta iteração da lista, e os próximos parâmetros. A complexidade final fica $O(log(n))$, pois sempre dividimos o n por $(m + 1)$ e, quando $n = 1$, a recursão para.

Tutorial: Letra aleatória

Daniel Saad Nogueira Nunes

Uma forma de resolver esse problema é contar o número de ocorrências de cada símbolo de S_1 com uma tabela T.

Em seguida, para cada símbolo c em S_2, o número de ocorrências de $T[c]$ é diminuído de 1. A resposta é o caractere c tal que $T[c] = -1$.

A complexidade desta solução é $\Theta(|S_2|)$.

Outra forma de resolver é ordenando as duas strings em ordem crescente. Em seguida, comparamos cada caractere $S_1[i]$ com $S_2[i]$ e, caso sejam diferentes, a resposta é $S_2[i]$. Se chegarmos ao final de S_1 ,então a resposta é o último caractere de S_2. Esta solução tem complexidade $\Theta(|S_2| \lg(|S_2|))$.

Tutorial: Nürburgring

Daniel Porto

Para resolver o problema, deve-se criar uma função recursiva que calculará o tempo de cada volta n. Para não estourar o tempo de execução, deve-se registrar em uma hash os tempos já calculados para evitar repetir contas já realizadas.

Tutorial: O Teorema do Macaco infinito

José Leite

Existem várias formas de resolver este problema. Vamos detalhar duas.

Solução 1:

A primeira parte a considerar é que cadeias s de mesmo tamanho podem ter resultados diferentes. Isso acontece pois, ao digitar um novo caracterer "errado" podemos aproveitar um sufixo já digitado que é prefixo do objetivo. Por exemplo, com $s = ababac$, se já digitamos $ababa$ e depois digitarmos o caracterer b, podemos continuar com $abab$.

Podemos pensar em utilizar o automato do KMP para resolver o problema pois usa esta ideia de reaproveitar sufixos. Se tentarmos escrever uma programação dinâmica — como fazemos em vários problemas de probabilidade — temos a seguinte recorrência:

$f(u) = 1 + \sum_c p_c \cdot f(to[u][c])$, onde $f(u)$ é o tempo esperado para montar s partindo do estado u, ou seja já tendo feito um prefixo de tamanho u; p_c é a probabilidade de digitar o caractere c; $to[u][c]$ é o estado do automato que iremos após adicionar o caracterer c partindo de u.

Sabemos que $f(n) = 0$, sendo $n = |s|$ e a resposta será $f(0)$.

O problema desta recorrência é que $f(u)$ pode usar o próprio $f(u)$, em casos como $aaab$ e digitamos a depois de aaa, $f(v < u)$ em outros casos de falha, e $f(u + 1)$ em caso de o próximo caractere der match. Estes ciclos na recorrência impedem que façamos uma programação dinâmica.

Podemos, então, escrever o sistema de equações lineares e resolvê-lo em $O(n^2)$, uma vez que há poucos elementos acima da diagonal principal.

Podemos também usar alguns truques comuns para modelar a programação dinâmica. Primeiro vamos usar a equação de $f(u)$ para conseguir uma nova fórmula de $f(u + 1)$ que só usa respostas menores que $u + 1$. $f(u + 1) = \frac{f(u)-1-\sum_{c \neq s_i} p_c \cdot f(to[u][c])}{p_{s_i}}$, usando indexação em 0 em s.

O segundo problema desta modelagem é que precisamos calcular $f(0)$ enquanto sabemos o resultado de $f(n)$, mas agora estamos computando valores

usando respostas menores. Vamos modificar $f(u)$ para calcular seu valor com relação a $f(0)$. $f(u)$ passa a guardar um par (a, b) que representa $f(u) = a * f(0) + b$. Calculamos o valor de $f(n) = (a, b)$ e por sabermos que $f(n) = 0$ podemos recuperar o valor de $f(0) = \frac{-b}{a}$.

Complexidade final $O(n * \Sigma)$, onde $\Sigma = 26$, o tamanho do alfabeto.

Solução 2:

A segunda solução envolve uma familiaridade maior com probabilidade discreta.

Uma função geradora de probabilidade $G(x) = \sum_{k \geq 0} P(X = k)x^k$ é uma série de potências que descreve toda informação de uma variável aleatória X. O fato de todos os coeficientes terem soma 1 pode ser escrito por $G(1) = 1$.

Essas funções geradoras podem simplificar cálculo de valor esperado, uma vez que

$E(X) = \sum_{k \geq 0} P(X = k)k = \sum_{k \geq 0} P(X = k)k * x^{k-1}\Big|_{x=1} = G'(1)$

Ou seja, podemos calcular o valor esperado como o valor da derivada em $x = 1$.

Vamos tentar encontrar qual é a $G(x)$ em nosso problema. Os objetos que queremos contar são cadeias que terminam em s e este só ocorre uma vez. Como usual em funções geradoras, vamos considerar a "soma" desses objetos. Para um exemplo de $abaab$ temos: $A = abaab + aabaab + babaab + joseabaab + ...$, e se substituirmos cada caractere c por $p_c x$ nesta soma, onde p_c é a probabilidade de digitar o caractere c, obtemos $G(x)$.

Vamos introduzir a soma $N = 1 + abaa, zz, ...$ das cadeias onde s ainda não apareceu e 1 é a cadeia vazia. Agora podemos encontrar duas equações para estas duas variáveis — A e N — para encontrar o valor de A.

Temos que $1 + N(a + b + ... + z) = N + A$. Do lado esquerdo todo termo vai terminar em s, e fazer parte de A, ou não, e fazer parte de N. Do lado direito, todo termo é vazio ou faz parte de $N(a + b + ... + z)$.

Temos, ainda, que $Ns = A(\sum_{i=0}^{m-1} s^{(i)}[s^{(m-i)} = s_{(m-i)}])$ — onde $s^{(i)}$ é a cadeia formada pelos últimos i caracteres, respectivamente $s_{(i)}$ é formada pelos primeiros i caracteres. $[true] = 1$ e $[false] = 0$. Do lado esquerdo da equação, toda cadeia de N vai formar uma ocorrência depois de adicionar um prefixo de tamanho $m - i$, para algum i, deixando i caracteres extra. Os primeiros $m - i$ caracteres só podem terminar uma ocorrência se forem

iguais ao últimos $m - i$ caracteres. Do lado direito, toda cadeia termina em s, uma vez que $s^{(m-i)} = s_{(m-i)}$, e ao remover os últimos m caracteres com certeza removeremos pelo menos um caractere da primeira ocorrência, portanto, pertencerá a N.

Com as duas equações: $\frac{1-A}{1-\sum c} \cdot s = A \sum_{i=0}^{m-1} s^{(i)}[s^{(m-i)} = s_{(m-i)}]$

Para obter $G(x)$, substituimos todo caractere c por $p_c x$:

$$\frac{1 - G(x)}{1 - \sum p_c \cdot x} \cdot \widetilde{s} \cdot x^n = G(x) \sum_{i=0}^{m-1} \widetilde{s^{(i)}} x^i [s^{(m-i)} = s_{(m-i)}]$$

onde $\widehat{w}$ é o resultado após substituir todo caractere c por p_c em w, para $w = abacaba$ $\widehat{w} = p_a p_b p_a p_c p_a p_b p_a = p_a^4 p_b^2 p_c$. Como $\sum p_c = 1$, conseguimos

$$\begin{aligned} G(x) &= \frac{\widetilde{s} x^n}{\widetilde{s} x^n + (1-x) \sum_{i=0}^{m-1} \widetilde{s^{(i)}} x^i [s^{(m-i)} = s_{(m-i)}]} \\ &= \frac{x^n}{x^n + (1-x) \sum_{i=0}^{m-1} \widehat{s_{(m-i)}} x^i [s^{(m-i)} = s_{(m-i)}]} \end{aligned}$$

Onde $\widehat{w}$ é $\widetilde{w}^{-1}$.

Pela regra do quociente, com $P(x) = x^n$ e $Q(x) = x^n + (1-x) \sum_{i=0}^{m-1} \widehat{s_{(m-i)}} x^i [s^{(m-i)} = s_{(m-i)}]$ temos

$$G'(1) = \frac{P'(1)Q(1) - P(1)Q'(1)}{Q(1)^2}$$

$$Q'(1) = n - \sum_{i=0}^{m-1} \widehat{s_{(m-i)}} [s^{(m-i)} = s_{(m-i)}]) = n - \sum_{i=1}^{m} \widehat{s_{(i)}} [s^{(i)} = s_{(i)}])$$

Como $Q(1) = 1$ e $P(1) = 1$

$$G'(1) = \frac{n - (n - \sum_{i=1}^{m} \widehat{s_{(i)}} [s^{(i)} = s_{(i)}])}{1^2} = \sum_{i=1}^{m} \widehat{s_{(i)}} [s^{(i)} = s_{(i)}]$$

Neste problema específico, todos os caracteres têm probabilidade uniforme, então podemos calcular a solução final com a fórmula $\sum_{i=1}^{m} 26^i [s^{(i)} = s_{(i)}]$.

Isso nos dá uma fórmula direta, só precisamos calcular as bordas da cadeia s que pode ser feito de forma direta em $O(N^2)$ ou usando KMP para uma complexidade total de $O(N)$.