Seletiva individual 2019: Segunda prova

Nathan Benedetto Proença

27 de maio de 2019

Rope

Os pontos já são dados em sentido horário ou anti-horário. Podemos então percorrer os pontos calculando o perímetro. Para contabilizar as curvas dos pregos, basta então somar o perímetro de um círculo de raio R na resposta final.

Moodular Arithmetic

Aparentemente errei a mão nesse problema, e acabei complicando a prova para vocês. Desculpe por isso. Existem duas soluções para o problema, uma $O(p \lg^* p)$ e outra $O(\sqrt{p} \lg p)$. Eu imaginei que a primeira fosse simples de chegar, e pretendia explicar a segunda como uma curiosidade.

Queremos contabilizar as funções $f\colon \mathbb{Z}_p \to \mathbb{Z}_p$ tais que, para dado $k \in \mathbb{Z}_p$,

$$f(kx) = kf(x)$$

para todo x. Note que, ao escolhermos um valor para $f(x)$, estamos escolhendo um valor para $f(kx)$ — e vice e versa. Em outras palavras, ao escolhermos o valor de x, estamos escolhendo o valor de kx, de k^2x, de k^3x, ... etc.

Você precisa pensar sobre como tratar os casos $k = 0$ e $k = 1$. Se $k > 1$, podemos então concluir que $f(0) = 0$ (por que?). Basta então contar as escolhas para os demais vértices. Podemos montar um grafo cujo os vértices são $\mathbb{Z}_p^*$ (os elementos não nulos de $\mathbb{Z}_p$) e que conecta a e b se e somente se $b = ka$ ou $a = kb$. Da observação acima, para cada uma dessas componentes, ao escolher um dos valores você escolhe todos os demais. Mais ainda, se você escolher um carinha diferente, a componente toda fica diferente. Segue então que, nesse caso, existem p^c soluções — com c sendo a quantidade de componentes conexas desse grafo. Basta então fazer um union-find neste grafo para encontrar c.

A outra solução segue de trabalhar mais um pouco nessa ideia. Sejam $x, y \in \mathbb{Z}_p^*$. Podemos olhar para a *órbita* de $x \in \mathbb{Z}_p^*$, que é o conjunto

$$\left\{x, kx, k^2x, k^3x, ...\right\}$$

Note que queremos contar quantas órbitas diferentes existem em $\mathbb{Z}_p^*$. Observe então que a função $a \mapsto ax^{-1}y$ é uma função da órbita de x para a órbita de y,

e que tem como inversa a função $a \mapsto ay^{-1}x$. Segue então que todas as órbitas tem o mesmo tamanho.

Logo, as órbitas dos elementos particionam $\mathbb{Z}_p^*$ em conjuntos de mesmo tamanho. Seja t o tamanho das órbitas. Segue então que $t|p-1$, e que temos $(p-1)/t$ órbitas distintas — o que quer dizer que nossa resposta é $p^{t/(p-1)}$. Para calcular t, podemos então calcular o tamanho da órbita que contém 1. Precisamos apenas testar os divisores d de $p-1$, e achar o menor tal que

$$k^d = 1 \mod p,$$

que pode ser feito em $\lg p$ com exponenciação rápida.

Para quem gosta dos nomes, esse argumento das órbitas é o teorema de Lagrange aplicado ao grupo gerado por k em $\mathbb{Z}_p^*$. O argumento acima é uma pequena modificação da prova do teorema.

Titanic

Notei que vários resolveram não embarcar na viagem de navio, com medo do iceberg, ou da geometria, imagino.

A ideia desse problema é construir os pontos na esfera, e calcular a distância na esfera entre eles a partir do produto interno. Eu diria que vale a pena fazer o desenho da fatia de pizza e pra ver como a distância na borda é determinada pelo ângulo. Este ângulo pode ser calculado como o arco-cosseno do produto interno entre os vértices que representam a posição do iceberg e do navio.

Precisamos então saber criar os pontos na esfera a partir da latitude e da longitude (além de ler a entrada, que é meio pentelha). A ideia é que podemos começar do vetor $(R, 0, 0)$ e aplicar uma rotação que coloque ele na latitude correta, e outra que o coloque na longitude correta. Ambas as rotações fixam um dos três eixos do espaço tridimensional.

Reduzimos então o problema à aplicar rotações em torno da origem em vetores 3D. Pra quem se interessar, vale a pena ver na wikipédia a explicação de como rotações arbitrárias em torno da origem podem ser decompostas em rotações de ângulo θ que fixem um dos eixos. Já essas, podemos calcular de maneira simples utilizando a biblioteca complex do C++.

Vamos lá. Multiplicar por números complexos corresponde a fazer rotações e dilatações do plano. Podemos usar a função `polar(1.0, t)` para termos o número complexo que corresponde a rotacionar o plano t graus no sentido anti-horário — ou seja, o número $\cos t + i \sin t$. Com isso, fica trivial de implementar as três rotações. Irei denotar por $R_x(\theta)$, $R_y(\theta)$ e $R_z(\theta)$ as rotações que fixam, respectivamente, o eixo x, y e z, e rotacionam o restante do espaço por θ no sentido dado pela "regra da mão direita".

1. $R_x(\theta)(a, b, c) = (a, \text{Real}(w), \text{Imag}(w))$, onde $w = (b + ic) \cdot (\cos\theta + i \sin\theta)$.
2. $R_y(\theta)(a, b, c) = (\text{Imag}(w), b, \text{Real}(w))$, onde $w = (c + ia) \cdot (\cos\theta + i \sin\theta)$.
3. $R_z(\theta)(a, b, c) = (\text{Real}(w), \text{Imag}(w), c)$, onde $w = (a + ib) \cdot (\cos\theta + i \sin\theta)$.

Isso pode ser feito em uma pequena função, notando que fixar z é rodar $(a + ib)$, fixar x é rodar $b + ic$ e fixar x é rodar $c + ia$. É algo simples, bonito, e que eu diria que vale a pena colocar no caderno de vocês.

Civilization

Este problema tem duas ideias interessantes. A primeira é que conseguimos calcular o maior caminho em umar árvore, e a segunda é que conseguimos combinar as componentes de maneira a minimizar o maior caminho.

O maior caminho em uma árvore pode ser encotrado com duas buscas em profundidade. A primeira começa de qualquer vértice e marca a distância para todos os demais. A segunda escolhe uma das folhas que esteja a distância máxima e faz a dfs a partir dela. Qualquer um dos vértices que estiver na distância máxima nessa segunda dfs maximiza a distância na árvore.

A linha geral dessa prova consiste em escolher um caminho máximo na árvore, e fixar um vértice qualquer no grafo, que seria a origem da primeira dfs, e concluir que, fazendo o procedimento acima, você vai chegar num caminho que é maior ou igual ao caminho fixado. Como o fixado é máximo, segue a que o procedimento descrito encontra um caminho máximo.

Isso resolve a primeira parte do problema. Precisamos então ser capaz de colocar uma aresta entre duas árvores de maneira que o caminho máximo seja o menor possível. Para fazer isso, vamos manter a distância de um caminho máximo em cada componente, e ir atualizando conforme as uniões forem acontecendo.

Vamos assumir então que queremos juntar duas componentes, uma cujo maior caminho tem tamanho a e outra cujo maior caminho tem tamanho b. Note que conseguimos criar um caminho de tamanho

$$\left\lceil \frac{a}{2} \right\rceil + \left\lceil \frac{b}{2} \right\rceil + 1$$

ligando os vértices "no meio" dos caminhos. A outra obersvação, e que deixo também para vocês provarem, é que, como a e b são os tamanhos dos caminhos máximos, para qualquer aresta que eu coloque, o maior caminho terá pelo menos esse tamanho. Segue então que esse valor minimiza, e que conseguimos combinar componentes em tempo constante. Basta então manter um union find das componentes e calcular o valor novo na hora do merge.

DZY loves colors

Escolhi esse problema por dois motivos: Um, por eu já ter escrito um editorial que o resolve quando cursei MAC0214. Você pode ler a solução aqui. O outro motivo é que ele é uma versão mais complicada do problema D da final brasileira de 2017.

Fat hobbits

Este problema segue a temática de utilizar a seletiva individual para ensinar teoremas. Depois de ensinar o teorema de Lagrange, para resolver esse problema é necessário utilizar dois teoremas: O teorema de Konig e o teorema de Dilworth.

O primeiro passo é reformular o problema. Pensando que temos um grafo entre os hobbits com uma aresta entre eles se o peso de um é maior que o peso do outro, caminhos nesse grafo direcionado representam a conclusão de que um hobbit é mais pesado do que o outro. Assim, queremos escolher a maior quantidade de vértices tais que não exista um caminho entre quaisquer dois escolhidos.

Podemos primeiro utilizar o algoritmo de Floyd-Warshall para calcular o *fecho transitivo* desse grafo, ou seja, o grafo que tem uma aresta entre todos os pares nos quais no grafo original há um caminho.

Dado um grafo dirigido, chamamos de uma *anticadeia* um conjunto de vértices tal que não há um caminho entre quaisquer dois vértices. Queremos, então, encontrar a maior anticadeia do grafo. Aqui nosso primeiro teorema entra em cena: O teorema de Dilworth garante que o tamanho da maior anticadeia é exatamente o tamanho da menor cobertura por caminhos do digrafo. Uma *cobertura por caminhos* é uma partição dos vértices de um digrafo em caminhos disjuntos tal que todo vértice esteja em exatamente um caminho. Chamamos de tamanho da cobertura a quantidade de subconjuntos na partição.

Esse problema eu ainda não sei resolver. Felizmente, a wikipédia tem uma prova construtiva da maior anticadeia. Você pode ler a seção Proof via Konig's theorem para entender por que o teorema é verdade, e como obter uma anticadeia a partir de uma cobertura mínima por vértices de um grafo bipartido. Nesse contexto, uma *cobertura por vértices* é um subconjunto de vértices tal que toda aresta é incidente a ele.

Assim, reduzimos o problma a encontrar uma cobertura por vértices mínima em um grafo bipartido. O teorema de Konig garante que, em grafos bipartidos, o tamanho da cobertura mínima é exatamente o tamanho do emparelhamento máximo. Novamente, você pode ler a prova na wikipédia desse fato, e, novamente, entender como construir essa cobertura a partir de um emparelhamento.

Dado tudo isso, também acho legal vocês colocarem no caderno não apenas os teoremas mencionados, como também as construções que evidenciam sua veracidade.

Com tudo isto em mãos, a ideia do problema é encontrar um emparelhamento máximo no grafo bipartido cuja construção é descrita na página do teorema de Dilworth, encontrar uma coberturar mínima com a construção descrita na página do teorema de Konig, e então construir a anticadeia que resolve o problema.

Bons estudos!