- Problema A - As Dicas de Ali Babá

  Apesar de poucas pessoas terem resolvido esse problema durante a aula, ele é bem simples. Note que as duas operações feitas sobre a permutação sempre inserem cópias dos elementos da permutação à frente dos originais. Então, para restaurar a permutação inicial, basta ler sequencialmente cada número da entrada e imprimi-lo caso seja a primeira vez que ele aparece.

- Problema B - Integral

  Ajuste a imagem de $f$ aplicada aos pontos inteiros de $[0, n] \setminus S$ para o máximo valor possível que obedeça às restrições do enunciado e calcule $\int_0^n f(x)dx$, notando que essa integral nada mais é do que a soma da área de vários trapézios. Se esse valor for menor que $y$, não existe solução, porque agora só podemos diminuir o resultado da integral.

  Agora, itere sobre todos os pontos inteiros de $[0, n]\setminus S$, em ordem crescente. Para cada ponto $x$ processado, ajuste $f(x)$ para o mínimo possível e cheque se a integral passa a ser menor ou igual a $y$ e, caso passe, encontre o ponto que a faz ser exatamente igual a $y$ e essa será sua resposta. Caso contrário, deixe $f(x)$ assumindo o menor valor possível e passe para o próximo ponto inteiro de $[0, n] \setminus S$. Chegando ao final da lista dos pontos ajustáveis, se a integral ainda for maior que $y$, não existe solução.

- Problema C - War

  Use a estrutura de union-find para unir em um mesmo conjunto as pessoas que são amigas e mantenha um vetor relacionando cada pessoa a um representante do seu conjunto de inimigos. Preste atenção para lembrar de incluir todas as propriedades das relações de amizade e inimizade na implementação.

- Problema D - Babelfish

  Use buscas binárias para encontrar as strings e suas traduções em um vetor ou, de forma mais simples, use as árvores binárias de busca já implementadas na sua linguagem de programação (`map` em C++ e `TreeMap` em Java, por exemplo).

- Problema E - Homem, Elefante e Rato

  Esse problema é uma aplicação direta de árvore de segmentos, não tem nenhum segredo, as consultas e atualizações que devem ser feitas são exatamente aquelas descritas no enunciado. A única dificuldade é que o código apresentado nos slides da aula não tem uma função para atualizar um intervalo arbitrário da árvore de $N$ folhas em $\mathcal{O}(\log N)$ operações, então, você deveria extrapolar as ideias apresentadas para construir essa função ou consultar de outra fonte.

De fato, a função de atualização em intervalo segue as mesmas ideias das outras operações na árvore de segmentos. O único truque aqui é que você não precisa propagar as mudanças da raiz da árvore até as folhas, você pode parar em um nó da árvore que esteja inteiramente dentro do intervalo atualizado e apenas pôr uma *flag* avisando que aquele intervalo deve ser atualizado. Assim, da próxima vez que você precisar fazer uma consulta usando esse nó, você de fato atualiza o seu valor e empurra a *flag* de atualização para os nós filhos. Para uma explicação detalhada, incluindo código fonte, sobre atualização em intervalos, eu recomendo a seguinte página: `https://www.hackerearth.com/practice/notes/segment-tree-and-lazy-propagation/`

- Problema F - Ataque Fulminante

  Esse problema pode ser um pouco trabalhoso porque exige algumas manipulações matemáticas para encontrar as intersecções das retas com a circuferência e fazer o cálculo das áreas, mas não tem nenhum segredo.

- Problema G - Graph Connectivity

  Há outras alternativas para resolver esse problema, mas a mais direta é usando um union-find. No começo, você tem tantas componentes conexas quanto vértices no grafo e, então, para cada aresta adicionada entre vértices pertencentes a conjuntos diferentes, o número de componentes conexas diminui em um.

- Problema H - Liga da Justiça

  Construa um vetor $g = (g_1, g_2, ..., g_H)$ em que $g_i$ é o número de heróis com que o herói $i$ tem uma relação. Ordene o vetor $g$ em ordem decrescente. Agora, encontre o maior índice $k$ tal que $d_k \geq k-1$ e então você sabe que, se existe uma Liga da Justiça, ela tem que ser formada exatamente pelos heróis com os números de relações descritos por $d_1, d_2, ..., d_k$, porque todos os heróis da Liga têm que ter uma relação entre eles. Uma vez definido o único conjunto que é candidato a Liga da Justiça, só resta checar se existe alguma relação entre os que ficaram de fora desse conjunto.

- Problema I - Balé

  O problema pede para que, dada uma permutação $P = (P_1, P_2, ..., P_N)$, contemos o número de inversões nessa permutação, isto é, quantos pares $(i, j)$ distintos existem tais que $i < j$ e $P_i > P_j$. Note que, como $N$ pode assumir valores até $10^5$, algoritmos que façam da ordem de $N^2$ operações não serão rápidos o suficiente.

  Uma forma eficiente de resolver o problema é construindo uma árvore de segmentos que consulta a soma em um intervalo arbitrário dos elementos

de um vetor $V = (V_1, V_2, ..., V_N)$. Inicialize $V$ com zeros em todas as entradas. Leia sequencialmente os elementos da permutação e, para cada $P_i$ lido, consulte a soma dos elementos de $V_{P_i+1}$ a $V_N$, some esse resultado à sua resposta e então atualize o valor de $V_{P_i}$ para 1. A ideia é que, assim, quando você lê um valor, você imediatamente conta todos os números maiores que ele que já apareceram antes na permutação e, fazendo isso para todos, você conta todas as inversões.

Existem também outras maneiras de resolver esse exercício, como por exemplo usando uma implementação adaptada do *mergesort* ou com uma estrutura chamada *Binary Indexed Tree*.

- Problema J - Triângulos

  O que você tem que perceber aqui é que é possível calcular o perímetro $P$ da circunferência (basta somar todos os comprimentos de arco dados) e que todos os triângulos equiláteros devem ter seus vértices separados por um arco de comprimento $P/3$.

  Então, meça o comprimento de arco da origem até cada um dos pontos dados, em sentido horário (ou anti-horário, você escolhe), e armazene essas medidas na sua estrura de dados preferida que permita fazer buscas eficientes por um elemento específico. Então, para cada ponto a uma distância $x$ da origem, verifique se também existem pontos nas posições $x + P/3$ e $x + 2P/3$, para que formem um triângulo equilátero.