Projeto e Análise de Algoritmos - Lista 3

FGV - EMAp

Outubro de 2025

Para entregar os exercícios desta lista, você deve alterar o arquivo `lista_3_paa.py` adicionando a solução no corpo da função correspondente.

Para cada questão abaixo existe uma função definida no script que deve ser completada. Caso você queira definir uma classe ou função auxiliar, faça isso dentro da função do problema que estiver resolvendo.

Atenção:

- Não altere o nome do arquivo Python.
- Não altere a assinatura das funções. Você deve receber e retornar os tipos definidos.
- Não importe bibliotecas extras no seu script.
- Cuidado com o uso de memória, o foco dessa lista não é eficiência de mémoria, mas você não deve alocar listas com mais de 10^8 inteiros.
- Remova ou comente prints/debugs/testes que você fez no seu código antes de enviar. O ideal é fazer os testes em outro arquivo, importando as funções (a lista será corrigida dessa forma).

1. Sisi e a Sorveteria: Parte 2

Sisi ganhou um cupom que permite pegar uma quantidade "ilimitada" de sorvete, desde que siga algumas regras. Na sorveteria existem n sabores de sorvete, e para cada sabor i (de 1 a n) há um estoque a_i.

Sisi definirá uma quantidade q_i para pegar de cada sabor i, seguindo as seguintes regras:

- **Estoque:** A quantidade de cada sabor não pode exceder o estoque disponível: $0 \leq q_i \leq a_i$.
- **Regra da Sequência:** Ao decidir as quantidades $q_1, q_2, \ldots, q_n$, Sisi deve obedecer a uma regra crescente. Para qualquer sabor i (de 1 a $n-1$):
  - Se $q_i = 0$, a quantidade q_{i+1} pode ser qualquer valor (desde que $0 \leq q_{i+1} \leq a_{i+1}$).
  - Se $q_i > 0$, a quantidade q_{i+1} deve ser estritamente maior que q_i (ou seja, $q_{i+1} > q_i$).

Em outras palavras, a sequência de quantidades q deve ser não-decrescente, e assim que incluir um número positivo, ela deve se tornar estritamente crescente a partir dali (ex: $[0, 0, 2, 5, 8]$ é válido, mas $[0, 0, 2, 5, 5]$ não é).

Ajude Sisi a maximizar a quantidade total de sorvete $Q = \sum_{i=1}^{n} q_i$. Desenvolva um algoritmo com complexidade de tempo $O(n)$ que resolva este problema.

Entrada: A entrada consiste em uma lista de n inteiros $A = [a_1, a_2, \ldots, a_n]$, onde a_i é o estoque do i-ésimo sorvete.
Saída: Retorne um único inteiro Q, a quantidade máxima total de sorvete que Sisi pode obter.

Restrições:

- $1 \leq n \leq 2 \cdot 10^5$.
- $1 \leq a_i \leq 10^9$ para todo $1 \leq i \leq n$.

Exemplos:

Entrada	**Saída**
$n = 5,\ A = [1, 2, 1, 3, 6]$	10
$n = 5,\ A = [3, 2, 5, 4, 10]$	20
$n = 4,\ A = [2, 2, 2, 2]$	3

No primeiro exemplo $q = [0, 0, 1, 3, 6]$.
No segundo exemplo $q = [1, 2, 3, 4, 10]$.
No terceiro exemplo $q = [0, 0, 0, 1, 2]$.

2. Minimizando Custos de Reparo

A empresa DpC (Divisão para Consertar) precisa otimizar seus custos no serviço de reparo de buracos em estradas. A empresa sempre opera em estradas contínuas cujo comprimento L é uma potência de 2 (ou seja, $L = 2^n$ para algum $n \geq 0$). O processo de reparo é definido por uma estratégia recursiva. A equipe começa analisando a estrada inteira (um segmento de 1 a L).

Para qualquer segmento de estrada sendo analisado, a equipe de engenheiros pode tomar uma de duas decisões:

- Opção 1: Dividir
  Se o comprimento l do segmento atual for 2 ou mais, a equipe pode dividi-lo em duas metades exatas (de comprimento $l/2$ cada).
- Opção 2: Consertar
  A equipe pode consertar o segmento atual de comprimento l de uma só vez. O custo desta ação depende da quantidade de buracos (N_b) neste segmento específico:
  - Se $N_b = 0$ (sem buracos), o custo é C_1.
  - Se $N_b > 0$ (com buracos), o custo é $C_2 \cdot N_b \cdot l$.

Você foi contratado pela DpC para minimizar os custos da operação, desenvolva um algoritmo que encontre o custo mínimo para reparar a estrada inteira. O algoritmo deve ter uma complexidade de tempo $O(k \cdot \log k)$.

Entrada:

- $1 \leq n \leq 30$: O expoente do comprimento da estrada ($L = 2^n$).
- $1 \leq k \leq 10^5$: O número de buracos.
- $1 \leq C_1, C_2 \leq 10^9$: As constante de custo por unidade de comprimento para reparar um trecho sem buracos, e com buracos, respectivamente.
- $A = [a_1, a_2, \ldots, a_k]$, com $1 \leq a_i \leq 2^n$: Uma lista com as k posições dos buracos.

Saída:

- Retorne um único inteiro: o custo mínimo total para reparar toda a estrada.

Exemplos:

Entrada	**Saída**
$n = 2$; $k = 2$; $C_1 = 1$; $C_2 = 2$; $A = [1, 3]$	6
$n = 3$; $k = 2$; $C_1 = 1$; $C_2 = 2$; $A = [1, 7]$	8

Explicação do primeiro exemplo:
Começamos com o intervalo de pista (1, 4) e podemos consertá-lo com custo $2 \cdot 2 \cdot 4 = 16$.

Ou podemos dividi-lo em (1, 2) e (3, 4).

Para o intervalo (1, 2) podemos consertar com custo $2 \cdot 1 \cdot 2 = 4$, ou dividir em (1, 1) e (2, 2).

O intervalo (1, 1) tem custo 2 e o intervalo (2, 2) custo 1. Então o custo total fica $2 + 1 = 3$, que é menor que o custo de consertar o intervalo (1, 2) sem dividi-lo.

Analogamente, o custo mínimo do intervalo (3, 4) é 3. Dessa forma, o custo mínimo total do conserto é $3 + 3 = 6$.

3. Subsequências *radicais*

Pedrinho adora brincar com sequências de números inteiros. Recentemente, ele está com seu foco voltado para as sequências *radicais*: Uma sequência $A = [a_1, a_2, \ldots, a_k]$ de k

inteiros é dita radical se, para todo índice i (de 1 a k), o i-ésimo elemento a_i é divisível por i.
Ele gosta tanto dessas sequências que criou um jogo:

- Primeiro, ele escreve uma sequência $A = [a_1, a_2, \ldots, a_n]$ em seu quadro.
- Depois, ele mentalmente lista todas as subsequências de A. (Uma subsequência é formada removendo zero ou mais elementos de A, mantendo a ordem relativa dos demais).
- Finalmente, ele conta quantas dessas subsequências são *radicais*.

Por exemplo, se $A = [1, 2, 2]$, as subsequências *radicais* são:

- $[1]$, pois $1|1$.
- $[2]$, utilizando o primeiro 2 de A, pois $1|2$.
- $[2]$, utilizando o segundo 2 de A, pois $1|2$.
- $[1, 2]$, utilizando o primeiro 2 de A, pois $1|1$ e $2|2$.
- $[1, 2]$, utilizando o segundo 2 de A, pois $1|1$ e $2|2$.
- $[2, 2]$, utilizando o primeiro e o segundo 2, pois $1|2$ e $2|2$.

A subsequência [1, 2, 2] não é radical, pois o terceiro elemento (2) não é divisível por 3. Observe que Pedrinho não considera sequência vazia radical. O total, neste caso, é 6.

Esse processo é muito demorado, e Pedrinho deseja fazer a contagem de maneira mais rápida. Ajude Pedrinho: sua tarefa é desenvolver um algoritmo que receba uma sequência $A = [a_1, a_2, \ldots, a_n]$ (com $a_i \leq n$) e retorne a quantidade total de subsequências *radicais* que ela possui. A solução deve ter uma complexidade de tempo $O(n\sqrt{n})$.

OBS: Como o número total pode ser muito grande, retorne o resultado módulo 999999937 (um número *descolado*, mas isso é outro problema).

Restrições:

- $1 \leq n \leq 10^5$.
- $1 \leq a_i \leq n$ para todo $1 \leq i \leq n$.

Entrada	**Saída**
$n = 3$, $A = [1, 2, 2]$	6
$n = 1$, $A = [1]$	1
$n = 2$, $A = [1, 2]$	3
$n = 5$, $A = [2, 2, 1, 5, 3]$	8

4. Cavalo

Dado um tabuleiro de xadrez $n \times n$, você deve calcular, para cada posição do tabuleiro, o número mínimo de movimentos que um cavalo leva para partir desta posição e chegar à posição superior esquerda (primeira linha e primeira coluna).

O algoritmo deve ter uma complexidade de tempo de $O(n^2)$ e retornar uma matriz A, onde $A_{i,j}$:= "número mínimo de movimentos para um cavalo ir da posição (i, j) para posição (1, 1)".

Restrições:

- $3 \leq n \leq 10^3$.

Exemplos:

Entrada	**Saída**
$n = 8$	$A = \begin{bmatrix} 0 & 3 & 2 & 3 & 2 & 3 & 4 & 5 \\ 3 & 4 & 1 & 2 & 3 & 4 & 3 & 4 \\ 2 & 1 & 4 & 3 & 2 & 3 & 4 & 5 \\ 3 & 2 & 3 & 2 & 3 & 4 & 3 & 4 \\ 2 & 3 & 2 & 3 & 4 & 3 & 4 & 5 \\ 3 & 4 & 3 & 4 & 3 & 4 & 5 & 4 \\ 4 & 3 & 4 & 3 & 4 & 5 & 4 & 5 \\ 5 & 4 & 5 & 4 & 5 & 4 & 5 & 6 \end{bmatrix}$

5. Escape se for possível

Você está em uma caverna, representada por um grid $n \times m$. Quando ocorre um desmoronamento, a caverna começa a inundar a partir de uma ou mais posições. Seu objetivo é encontrar um caminho para fora da caverna antes que a água chegue até você.

Legenda do Grid:

- 'V': Sua posição inicial.
- 'A': Posição inicial da água.
- '.' (ponto): Espaço livre (pode ser percorrido por você e pela água).
- '#': Parede (bloqueia você e a água).

Exemplo de Caverna:

$$\begin{bmatrix} \# & \# & \# & \# & \# & \# & \# & \# \\ \# & A & . & . & V & . & . & \# \\ \# & . & \# & . & A & \# & . & \# \\ \# & A & \# & . & . & \# & . & . \\ \# & . & \# & \# & \# & \# & \# & \# \end{bmatrix}$$

Neste exemplo, é possível escapar em 5 segundos. O único caminho minimal possível é mover-se duas vezes para a direita, duas vezes para baixo, uma vez para direita, chegando na posição (4, 8), que é uma saída.

Regras:

- No instante inicial ($t = 0$), você está na posição 'V' e a água está em todas as posições A.
- A cada instante de tempo ($t = 1, 2, 3, \ldots$):

(a) A água se propaga simultaneamente de todas as suas posições atuais para todas as posições adjacentes (cima, baixo, esquerda, direita) que não sejam paredes (#).

(b) Você se move para uma posição adjacente (cima, baixo, esquerda, direita) que não seja uma parede (#).

- Você não pode se mover para uma célula que já esteja inundada. Além disso, você não pode se mover para uma célula no instante t se a água também for alcançá-la no mesmo instante t. Você deve sempre ser mais rápido que a água.
- Para escapar, você deve alcançar qualquer célula '.' que esteja na borda do grid (primeira ou última linha, primeira ou última coluna).

Sua tarefa é desenvolver um algoritmo com complexidade de tempo $O(n^2)$ que retorne o menor tempo (número de movimentos) necessário para escapar da caverna. Se não for possível escapar, seu algoritmo deve retornar -1.

Restrições:

- $1 \leq n, m \leq 10^3$.

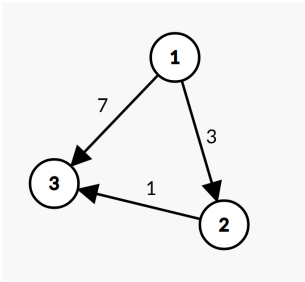
Exemplos:

Entrada	**Saída**
$n = 5$, $m = 8$, grid $= \begin{bmatrix} \# & \# & \# & \# & \# & \# & \# & \# \\ \# & A & . & . & V & . & . & \# \\ \# & . & \# & . & A & \# & . & \# \\ \# & A & \# & . & . & \# & . & . \\ \# & . & \# & \# & \# & \# & \# & \# \end{bmatrix}$	5
$n = 5$, $m = 8$, grid $= \begin{bmatrix} \# & \# & \# & \# & \# & \# & \# & \# \\ \# & . & . & . & V & . & . & \# \\ \# & . & \# & . & . & \# & . & \# \\ \# & . & A & . & . & A & . & . \\ \# & . & \# & \# & \# & \# & \# & \# \end{bmatrix}$	-1
$n = 5$, $m = 8$, grid $= \begin{bmatrix} \# & \# & \# & \# & \# & \# & \# & \# \\ \# & A & . & . & \# & . & . & \# \\ \# & . & \# & . & A & \# & . & \# \\ \# & A & \# & . & . & \# & . & V \\ \# & . & \# & \# & \# & \# & \# & \# \end{bmatrix}$	0

6. Viagem Intergalática

Você está em um futuro utópico no qual a humanidade colonizou n planetas. O meio de transporte mais comum são os ônibus espaciais da União Intergaláctica, que opera m rotas.

Cada rota parte de um planeta a e vai para um planeta b com um custo c. Você possui um cupom que permite que o custo c de **exatamente uma** rota do seu percurso seja reduzido para $\lfloor \frac{c}{2} \rfloor$.



Primeiro Exemplo: Nesse caso o menor custo é atingido utilizando o cupom para ir do planeta 1 ao planeta 2 pagando $\lfloor \frac{3}{2} \rfloor = 1$ e depois pagando 1 para ir ao planeta 3

.

Você está na Terra (planeta 1) e deseja chegar ao planeta '51 Pegasi b' (planeta n). Sua tarefa é encontrar o menor custo total para fazer essa viagem.

Desenvolva um algoritmo com complexidade de tempo $O(m \log n)$ que, dado o número de planetas n, o número de rotas m, e a lista de m rotas (cada uma com origem a, destino b e custo x), encontre o menor custo possível.

Restrições:

- $1 \leq n \leq 10^5$.
- $1 \leq m \leq 2 \cdot 10^5$.
- $1 \leq a_i, b_i \leq n$ para todo $1 \leq i \leq m$.
- $1 \leq c_i \leq 10^9$ para todo $1 \leq i \leq m$.
- Observe que $m_{\max} << n^2_{\max}$.

Exemplos:

Entrada	**Saída**
$n = 3$; $m = 3$; $A = [(1, 2, 3), (2, 3, 1), (1, 3, 7)]$	2
$n = 3$; $m = 3$; $A = [(1, 2, 3), (2, 3, 3), (1, 3, 7)]$	3

7. Reparo das Estradas

Em um estado existem n cidades e m estradas bidirecionais. Infelizmente, devido ao desgaste, a rede de estradas está fragmentada, e podem não existir caminhos entre todos os pares de cidades. Cada estrada conecta duas cidades distintas, a e b, e possui um custo de reparação $c \geq 0$. Cada par de cidades possui no máximo uma estrada conectando-as diretamente.

- Se $c = 0$, a estrada está em boas condições e pode ser usada livremente.
- Se $c > 0$, a estrada precisa ser reparada a um custo c para poder ser utilizada.

O governador do estado deseja reconectar todas as cidades, garantindo que seja possível viajar entre quaisquer dois pontos, gastando a menor quantidade de dinheiro possível em reparos. Desenvolva um algoritmo com complexidade de tempo $O(m \log n)$ que, dado o número de cidades n, o número de estradas m, e a descrição de cada estrada (as duas

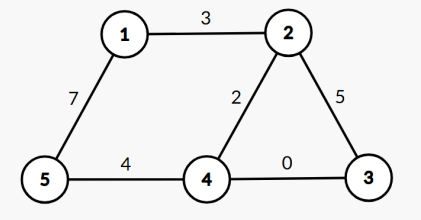
cidades que ela conecta e seu custo de reparo), retorne o custo mínimo total para garantir que todas as n cidades estejam conectadas.

Restrições:

- $1 \leq n \leq 10^5$.
- $1 \leq m \leq 2 \cdot 10^5$.
- $1 \leq a_i, b_i \leq n$ para todo $1 \leq i \leq m$.
- $0 \leq c_i \leq 10^9$ para todo $1 \leq i \leq m$.
- Observe que $m_{\max} << n^2_{\max}$.

Exemplos:

Entrada	**Saída**
$n = 5$; $m = 6$; $A = [(1, 2, 3), (2, 3, 5), (2, 4, 2), (3, 4, 8), (5, 1, 7), (5, 4, 4)]$	14
$n = 5$; $m = 6$; $A = [(1, 2, 3), (2, 3, 5), (2, 4, 2), (3, 4, 0), (5, 1, 7), (5, 4, 4)]$	9
$n = 2$; $m = 1$; $A = [(1, 2, 0)]$	0



Segundo Exemplo: Reparamos as estradas (1, 2), (2, 4) e (4, 5).

8. Video Game

Existe um jogo com n estados diferentes, numerados de 1 a n. A partir de cada estado você é obrigado a tomar uma decisão (entre um conjunto de possibilidades pré estabelecidas para esse estado) que levam a um estado diferente. O jogo foi criado de forma que não é possível retornar a estado já visitado.

Seu objetivo é ir do começo (estado 1) até o final do jogo (estado número n). De quantas formas isso pode ser feito?

Dada a descrição todas as transições de estado, uma lista de pares de inteiros (a, b) (onde cada par representa uma transição válida do estado a para o estado b), calcule a quantidade de formas de 'zerar' o jogo.

Desenvolva um algoritmo com complexidade $O(n + m)$.

Entrada:

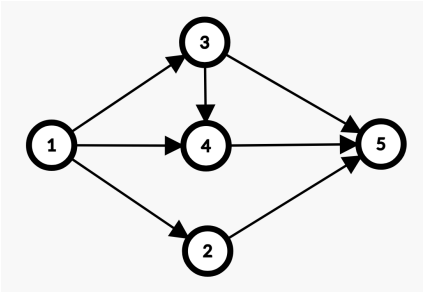
- $1 \leq n \leq 10^5$: O número de estados.
- $1 \leq m \leq 2 \cdot 10^5$: O número de transições.
- $[(a_1, b_1), \ldots, (a_m, b_m)]$, com $1 \leq a_i, b_i \leq n$: Uma lista com as m transições.

Saída:

- Retorne um único inteiro: o número de formas distintas de ir do estado 1 ao estado n.

Exemplos:

Entrada	**Saída**
$n = 3$; $m = 2$; $A = [(1, 2), (2, 3)]$	1
$n = 5$; $m = 7$; $A = [(1, 3), (3, 4), (1, 2), (2, 5), (1, 4), (4, 5), (3, 5)]$	4
$n = 3$; $m = 2$; $A = [(1, 2), (3, 2)]$	0



Segundo Exemplo: Os caminhos possíveis são (1, 2, 5), (1, 4, 5), (1, 3, 5) e (1, 3, 4, 5).