Resolução de Problemas

Técnicas de Programação (Guloso, Busca Binária)

Exercicio A - Binary String Minimizing
Resolução e Python 3

```
"""
Problema: Binary String Minimizing (Codeforces 1256D)

Voce recebe uma string binaria de tamanho n, contendo apenas os caracteres '0' e '1'.
Voce tambem recebe um numero k que representa a quantidade maxima de trocas que voce pode fazer.

Cada troca consiste em escolher dois caracteres adjacentes e trocar suas posicoes.

Objetivo:
Fazer no maximo k trocas para transformar a string na menor string possivel em ordem lexicografica
(ou seja, colocar o maior numero possivel de '0' no inicio da string).

Observacoes:
- Nao e necessario usar todas as k trocas.
- A cada troca voce pode apenas inverter dois caracteres consecutivos (i e i+1).
- Voce deve responder q casos de teste independentes.

Saida:
Para cada caso, imprima a menor string possivel (em ordem lexicografica) apos no maximo k trocas.
"""
```

```
# Funcao que resolve todos os casos de teste
def minimize_binary_string(q, test_cases):
    results = []

    # Para cada caso de teste
    for n, k, s in test_cases:
        s = list(s)  # transforma a string em lista de caracteres para facilitar a troca
        pos = 0      # posicao mais a esquerda disponivel para colocar um '0'

        for i in range(n):
            if s[i] == '0':
                # quantas trocas sao necessarias para mover esse '0' para a posicao 'pos'
                moves = min(k, i - pos)
                # se for possivel mover, faz a troca com a posicao correta
                s[i], s[i - moves] = s[i - moves], s[i]
                k -= moves  # atualiza o numero de trocas restantes
                pos += 1    # o proximo '0' pode ir mais para a direita

        results.append("".join(s))  # junta os caracteres de volta em uma string

    return results

# ========== PROGRAMA PRINCIPAL ==========
q = int(input())  # numero de casos de teste
test_cases = []
for _ in range(q):
    n, k = map(int, input().split())  # le n e k
    s = input().strip()               # le a string binaria
    test_cases.append((n, k, s))      # adiciona o caso a lista

# resolve todos os casos
answers = minimize_binary_string(q, test_cases)

# imprime os resultados
for ans in answers:
    print(ans)
```

Exercicio B - Find and Replace

Resolução em Python 3

```
"""
Problema: Alternating Binary String

Dado uma string contendo apenas letras minusculas, voce pode substituir
todas as ocorrencias de uma letra por '0' ou por '1'.
Seu objetivo e saber se e possivel fazer essas substituicoes de modo
que a string final seja uma string binaria alternada (isto e, sem dois bits iguais adjacentes).

Exemplo de string alternada: 010101, 1010
Exemplo de string nao alternada: 0011, 0110

Para isso, voce pode testar se ha alguma forma de mapear os caracteres para '0' e '1'
de forma coerente com o padrao alternado.
"""

# Funcao para verificar se e possivel transformar a string 's' em uma string binaria alternada
def is_alternating_possible(s):
    # Funcao auxiliar que tenta fazer o mapeamento comecando com o bit 'start_bit' (0 ou 1)
    def check(start_bit):
        mapping = {}  # Dicionario para armazenar qual caractere sera mapeado para 0 ou 1
        for i, ch in enumerate(s):
            expected = str((start_bit + i) % 2)  # Calcula o bit esperado nessa posicao (alternando 0 e 1)

            if ch in mapping:
                # Se o caractere ja foi mapeado antes, verifica se bate com o valor esperado
                if mapping[ch] != expected:
                    return False  # Conflito encontrado, esse padrao nao e possivel
            else:
                # Mapeia o caractere para o valor esperado
                mapping[ch] = expected
        return True  # Se passou por toda a string sem conflitos, esse padrao e possivel

    # Testa os dois padroes possiveis: comecando com 0 ou com 1
    return check(0) or check(1)

# ========== PROGRAMA PRINCIPAL ==========
t = int(input())  # Le o numero de casos de teste
for _ in range(t):
    n = int(input())      # Le o tamanho da string (nao e necessario diretamente)
    s = input().strip()   # Le a string de entrada
    # Imprime "YES" se for possivel, "NO" caso contrario (com quebra de linha)
    print("YES\n" if is_alternating_possible(s) else "NO\n")
```

Exercicio C - Platforms Jumping
Resolução em Python3

```
"""
Problema: Platforms Jumping

Voce esta em uma margem de um rio (posicao 0) e deseja chegar ate a outra margem (posicao n+1).
Entre essas margens existem n celulas (de 1 a n) que representam o leito do rio.

Voce possui m plataformas de madeira com tamanhos c1, c2, ..., cm que voce pode posicionar livremente sobre o rio,
mas respeitando duas regras:
1. As plataformas nao podem se sobrepor.
2. A ordem original das plataformas deve ser mantida.

Depois de posicionar as plataformas, voce pode comecar a pular:
- Se estiver em uma posicao x, pode pular para qualquer posicao entre [x+1, x+d], mas **somente se essa posicao estiver sobre uma plataforma**.
- A celula 0 (inicio) e a celula n+1 (fim) sao consideradas como "seguras", ou seja, voce pode comecar ou terminar nelas.

Objetivo:
Verificar se e possivel posicionar as plataformas de modo que voce consiga chegar de 0 a n+1 seguindo as regras acima.
Se sim, imprima "YES" seguido de um vetor de tamanho n que representa cada celula do rio:
- 0 indica agua (vazia)
- 1 a m indicam qual plataforma ocupa a posicao (indice da plataforma)
Se nao for possivel, imprima "NO".
"""
```

```python
# Leitura da largura do rio (n), numero de plataformas (m) e distancia maxima de pulo (d)
n, m, d = map(int, input().split())

# Leitura dos tamanhos das plataformas
c = list(map(int, input().split()))

# Inicializa o rio com zeros (nenhuma plataforma posicionada ainda)
a = [0] * n

# Calcula quanto espaco vazio temos para distribuir entre as plataformas
remaining = n - sum(c)

# Comecamos a posicionar da esquerda para a direita
pos = 0        # posicao atual no rio
idx = 1        # indice da plataforma (comeca em 1 pois a resposta e 1-indexada)

# Para cada plataforma, posicionamos ela respeitando o limite de pulo
for length in c:
    # Espaco vazio antes da plataforma (maximo d-1, sem ultrapassar o espaco restante)
    gap = min(d - 1, remaining)
    pos += gap
    remaining -= gap

    # Posiciona a plataforma no rio
    for i in range(length):
        if pos >= n:
            break
        a[pos] = idx
        pos += 1

    idx += 1  # proxima plataforma

# Verifica onde termina a ultima plataforma (ultima celula ocupada)
last = n
while last > 0 and a[last - 1] == 0:
    last -= 1

# Verifica se da ultima plataforma conseguimos pular ate n+1
if n - last < d:
    print("YES")
    print(" ".join(map(str, a)))
else:
    print("NO")
```

Exercicio D – Bits
Resolução em Python3

```
"""
Problema:

Dado um intervalo [l, r], queremos encontrar um número x tal que:
1. l ≤ x ≤ r
2. O número x tenha o maior número possível de bits '1' em sua representação binária.
3. Caso haja mais de um número com essa quantidade máxima de bits '1', escolha o menor.

Para cada consulta (l, r), devemos retornar esse valor x.

Exemplo:
Se l = 10 e r = 15
- 10 = 1010 (2 bits 1)
- 11 = 1011 (3 bits 1)
- 12 = 1100 (2 bits 1)
- 13 = 1101 (3 bits 1)
- 14 = 1110 (3 bits 1)
- 15 = 1111 (4 bits 1) → resposta

Abordagem:
Construímos o número x a partir de l, tentando ligar os bits (1s) do mais significativo para o menos, desde que não ultrapasse r.
"""
```

```
def resolve(queries):
    results = []

    for l, r in queries:
        x = l
        for i in range(63, -1, -1):
            # Tentamos ligar o bit i
            temp = x | (1 << i)
            # Agora queremos saber: se ligarmos esse bit, e colocarmos todos os bits menores como 1 (preencher com 1s), ficamos dentro de r?
            mask = (1 << i) - 1
            max_possible = temp | mask

            if max_possible <= r:
                x = temp  # Podemos ligar esse bit e ainda tentar maximizar os menores também

        results.append(x)

    return results

# Leitura da entrada
n = int(input())
queries = [tuple(map(int, input().split())) for _ in range(n)]

# Executa as consultas
answers = resolve(queries)

# Imprime os resultados
for ans in answers:
    print(ans)
```

Exercício E – 01 Tree

Resolução em Python3

```
Você tem uma árvore binária completa com n folhas. Em uma árvore binária completa,
cada vértice não-folha tem exatamente dois filhos (um esquerdo e um direito).

Cada aresta (ligação entre pai e filho) tem um peso: uma aresta tem peso 0 e a outra peso 1,
mas não se sabe qual lado (esquerda ou direita) recebe o peso 0.

Você conhece um vetor a de tamanho n, onde a[i] representa a distância do vértice raiz até a i-ésima
folha (na ordem do DFS da árvore).

Sua tarefa é dizer se **existe alguma árvore possível** com essa estrutura e essas distâncias
para as folhas, considerando que em cada vértice interno, um filho tem aresta de peso 0 e o outro de peso 1.

Entrada:
- t: número de casos de teste (1 ≤ t ≤ 10^4)
- Para cada caso:
    - n: número de folhas (2 ≤ n ≤ 2*10^5)
    - a: vetor de n inteiros representando as distâncias das folhas (0 ≤ ai ≤ n-1)
    - A soma total de n em todos os casos não ultrapassa 2*10^5.

Saída:
- Para cada caso de teste, imprima "YES" se existe uma árvore que satisfaça a condição, senão "NO".
```

```python
def read_int():
    # Lê um número inteiro da entrada
    return int(input())

def read_list():
    # Lê uma linha da entrada e a converte para uma lista de inteiros
    return list(map(int, input().split()))

def solve_single(n, a):
    # Conta quantas vezes o valor 0 aparece na lista (só pode haver exatamente uma folha com distância 0)
    zero = a.count(0)

    if zero != 1:
        return "NO"  # Se não houver exatamente uma raiz (distância 0), já é inválido

    # Vetor para registrar quais nós "satisfazem" a condição de estrutura da árvore
    satisfied = [False] * n
    stk = []  # Pilha para processar elementos em ordem crescente de profundidade

    # Primeira varredura da esquerda para a direita
    for i in range(n):
        while stk and stk[-1][0] > a[i]:  # Enquanto o topo da pilha for maior que o atual
            if stk[-1][0] == a[i] + 1:
                # Se o valor do topo for exatamente 1 a mais, ele "conecta" ao atual
                satisfied[stk[-1][1]] = True
            stk.pop()  # Remove o topo da pilha
        # Adiciona o elemento atual à pilha (profundidade, índice)
        stk.append((a[i], i))

    stk.clear()  # Limpa a pilha para a segunda varredura

    # Segunda varredura da direita para a esquerda
    for i in range(n - 1, -1, -1):
        while stk and stk[-1][0] > a[i]:  # Mesmo processo, agora invertido
            if stk[-1][0] == a[i] + 1:
                satisfied[stk[-1][1]] = True
            stk.pop()
        stk.append((a[i], i))

    # Conta quantos elementos não foram satisfeitos
    foo = satisfied.count(False)

    # Exatamente um nó (a raiz) pode não ser "satisfeito"
    return "YES" if foo == 1 else "NO"

def main():
    results = []  # Lista de resultados para cada caso de teste
    t = read_int()  # Número de casos de teste

    for _ in range(t):
        n = read_int()       # Número de folhas
        a = read_list()      # Lista de distâncias
        results.append(solve_single(n, a))  # Armazena o resultado

    # Imprime todos os resultados ao final
    print("\n".join(results))

# Executa o programa principal
if __name__ == "__main__":
    main()
```

Exercício F – Vabank
Resolução em Python3

```
'''
Gustaw é o gerente-chefe de um grande banco. Ele pode transferir qualquer valor para sua conta,
mas existe um sistema antifraude que detecta operações maiores que um certo limite M.

Ele não sabe qual é o valor exato de M. Se tentar transferir um valor X:
- Se X <= M: ele recebe o valor normalmente e ninguém percebe ("Lucky!").
- Se X > M: a fraude é detectada, a operação é cancelada e ele é multado em X euros ("Fraudster!").
- Se X > M e ele não tiver o dinheiro para pagar a multa: ele é demitido ("Fired!").

Ele começa com apenas 1 euro e quer descobrir M sem ser demitido.

Entrada: um número t (1 ≤ t ≤ 1000) indicando o número de casos de teste.
Para cada teste, o sistema responde interativamente às tentativas de Gustaw.

Objetivo: descobrir o valor exato de M usando no máximo 10⁵ operações por caso de teste.
'''
```

```python
import sys

def main():
    input = sys.stdin.readline  # leitura de linha rápida
    lim = 10**14                # limite máximo permitido para X

    def work():
        # cur: saldo atual na conta de Gustaw
        # cnt: contador de consultas realizadas
        cur = 1
        cnt = 0

        # função para realizar a consulta "? X"
        def qry(x):
            nonlocal cur, cnt
            # se exceder o limite global, retorna falso diretamente
            if x > lim:
                return False
            # incrementa contador e falha se ultrapassar 105 consultas
            cnt += 1
            if cnt > 105:
                sys.exit(1)
            # envia consulta ao interactor
            print(f"? {x}", flush=True)
            res = input().strip()
            # se X <= M, saldo aumenta e retornamos True
            if res.startswith("Lucky!"):
                cur += x
                return True
            # se X > M mas há saldo, saldo diminui e retornamos False
            if res.startswith("Fraudster!"):
                cur -= x
                return False
            # "Fired!": não há saldo para pagar a multa -> encerra
            sys.exit(0)

        # função para reportar a resposta final "! M"
        def repo(x):
            print(f"! {x}", flush=True)

        # ========== FASE 1: crescimento exponencial ===========
        # testar X=1 para ver se já é fraude
        if not qry(1):
            # se falhar em 1, então M=0
            repo(0)
            return
        # dobrar X até encontrar a primeira fraude
        x = 2
        while qry(x):
            x <<= 1
        # agora qry(x) foi Fraudster!, e qry(x/2) foi Lucky!
        L = x // 2  # menor valor ainda seguro
        R = x       # primeiro valor que falhou

        # ========== FASE 2: busca dentro do intervalo [L, R] ===========
        while L < R - 1:
            # recarregar saldo usando sempre L, que sabemos ≤ M
            # garantimos cur >= R antes de testar qualquer k
            while cur < R:
                if not qry(L):
                    # isso não deveria ocorrer pois L ≤ M
                    sys.exit(1)
            # calculamos o tamanho do intervalo
            gap = R - L
            # __lg(gap) equivalente a bit_length-1
            lg = gap.bit_length() - 1
            # e = cur / (L * lg)
            e = cur / (L * lg) if lg > 0 else float('inf')
            # definimos probabilidade p conforme o valor de e
            if e > 1:
                p = 0.5
            else:
                p = 0.3 + 0.2 * e
            # escolhemos k em [L+1, R)
            k = L + max(1, int(p * gap))
            # testamos k
            if qry(k):
                # se Lucky, movemos L para k
                L = k
            else:
                # se Fraudster, movemos R para k
                R = k
        # no fim, L e R são adjacentes; M = L
        repo(L)

    # Leitura do número de casos de teste
```