# Busca gulosa

[**Busca gulosa**](#_wcz6een69gr0)

**1**

[Troco Mínimo: Dado um conjunto de moedas de diferentes valores e um valor total, encontre a menor quantidade de moedas necessárias para dar o troco.](#_3hkzcpa98ocb) 1

[**Escalonamento de Tarefas: Dado um conjunto de tarefas com tempos de início e término, encontre o número máximo de tarefas que podem ser executadas simultaneamente.**](#_2gqq5ia2ywoz) 2

[Agendamento de Intervalos: Dado um conjunto de intervalos de tempo com início e término, encontre o menor número de salas de aula necessárias para acomodar todas as aulas.](#_68afunfg0wjg) 3

[**Seleção de Moedas: Dado um valor a ser alcançado e um conjunto de moedas disponíveis, selecione as moedas de forma a obter o valor com o menor número de moedas possível.**](#_myzlbyqio9a4) 4

## Troco Mínimo: Dado um conjunto de moedas de diferentes valores e um valor total, encontre a menor quantidade de moedas necessárias para dar o troco.

Digite os valores das moedas disponíveis separados por espaço: 10 20 5 1

Digite o valor total para dar o troco: 30

A menor quantidade de moedas necessárias é: 2

def min\_coins(coins, total\_amount):

coins.sort(reverse=True) # Ordena as moedas em ordem decrescente

num\_coins = 0

i = 0

while total\_amount > 0:

if i < len(coins) and coins[i] <= total\_amount:

num\_coins += 1

total\_amount -= coins[i]

else:

i += 1

if i >= len(coins):

break

if total\_amount == 0:

return num\_coins

else:

return -1 # Não é possível dar o troco com as moedas disponíveis

# Solicita a entrada do usuário para as moedas disponíveis e o valor total

coins = list(map(int, input("Digite os valores das moedas disponíveis separados por espaço: ").split()))

total\_amount = int(input("Digite o valor total para dar o troco: "))

# Calcula a menor quantidade de moedas necessárias para dar o troco

result = min\_coins(coins, total\_amount)

if result != -1:

print("A menor quantidade de moedas necessárias é:", result)

else:

print("Não é possível dar o troco com as moedas disponíveis.")

## Escalonamento de Tarefas: Dado um conjunto de tarefas com tempos de início e término, encontre o número máximo de tarefas que podem ser executadas simultaneamente.

Número máximo de tarefas simultâneas: 3

def max\_concurrent\_tasks(tasks):

events = [] # Lista de eventos (inicio ou fim de tarefa)

for start, end in tasks:

# Usamos tuplas (tempo, tipo de evento, tarefa) para ordenar

events.append((start, "start", None))

events.append((end, "end", None))

events.sort() # Ordena os eventos por tempo

max\_concurrent = 0

concurrent = 0

for time, event, \_ in events:

if event == "start":

concurrent += 1

max\_concurrent = max(max\_concurrent, concurrent)

else:

concurrent -= 1

return max\_concurrent

# Exemplo de uso

tasks = [(1, 5), (2, 7), (4, 8), (6, 9)]

print("Número máximo de tarefas simultâneas:", max\_concurrent\_tasks(tasks)) # Saída: 3

## Agendamento de Intervalos: Dado um conjunto de intervalos de tempo com início e término, encontre o menor número de salas de aula necessárias para acomodar todas as aulas.

Digite o número de intervalos: 3

Digite o tempo de início e término do intervalo (formato: início término): 1 9

Digite o tempo de início e término do intervalo (formato: início término): 2 8

Digite o tempo de início e término do intervalo (formato: início término): 3 4

O menor número de salas de aula necessárias é: 3

def min\_classrooms(intervals):

if not intervals:

return 0

intervals.sort(key=lambda x: x[0]) # Ordena os intervalos por tempo de início

classrooms = [intervals[0]]

for i in range(1, len(intervals)):

allocated = False

for j in range(len(classrooms)):

if intervals[i][0] >= classrooms[j][1]:

classrooms[j] = intervals[i]

allocated = True

break

if not allocated:

classrooms.append(intervals[i])

return len(classrooms)

# Solicita a entrada do usuário para o número de intervalos

num\_intervals = int(input("Digite o número de intervalos: "))

intervals = []

# Solicita a entrada do usuário para os tempos de início e término de cada intervalo

for \_ in range(num\_intervals):

start, end = map(int, input("Digite o tempo de início e término do intervalo (formato: início término): ").split())

intervals.append((start, end))

# Calcula o menor número de salas de aula necessárias

result = min\_classrooms(intervals)

print("O menor número de salas de aula necessárias é:", result)

## Seleção de Moedas: Dado um valor a ser alcançado e um conjunto de moedas disponíveis, selecione as moedas de forma a obter o valor com o menor número de moedas possível.

Digite o número de moedas disponíveis: 5

Digite os valores das moedas disponíveis separados por espaço: 1 20 10 50 2

Digite o valor a ser alcançado: 52

O menor número de moedas necessário é: 2

def min\_coins(coins, target\_amount):

dp = [float('inf')] \* (target\_amount + 1)

dp[0] = 0

for i in range(1, target\_amount + 1):

for coin in coins:

if i - coin >= 0:

dp[i] = min(dp[i], dp[i - coin] + 1)

return dp[target\_amount]

# Solicita a entrada do usuário para o número de moedas

num\_coins = int(input("Digite o número de moedas disponíveis: "))

coins = list(map(int, input("Digite os valores das moedas disponíveis separados por espaço: ").split()))

# Solicita a entrada do usuário para o valor a ser alcançado

target\_amount = int(input("Digite o valor a ser alcançado: "))

# Calcula o menor número de moedas necessário para alcançar o valor

result = min\_coins(coins, target\_amount)

print("O menor número de moedas necessário é:", result)