Seminário de ACA

2023\1

Problema do Troco

Proposta de resolução com as técnicas de
Programação Dinâmica e Algoritmo Guloso

Bernardo T. e Camila B.

Conteúdo

01

Problema

O que é o Problema do Troco?

Problema do Troco (Coin Change)

Especialização do **problema da mochila** sem limites

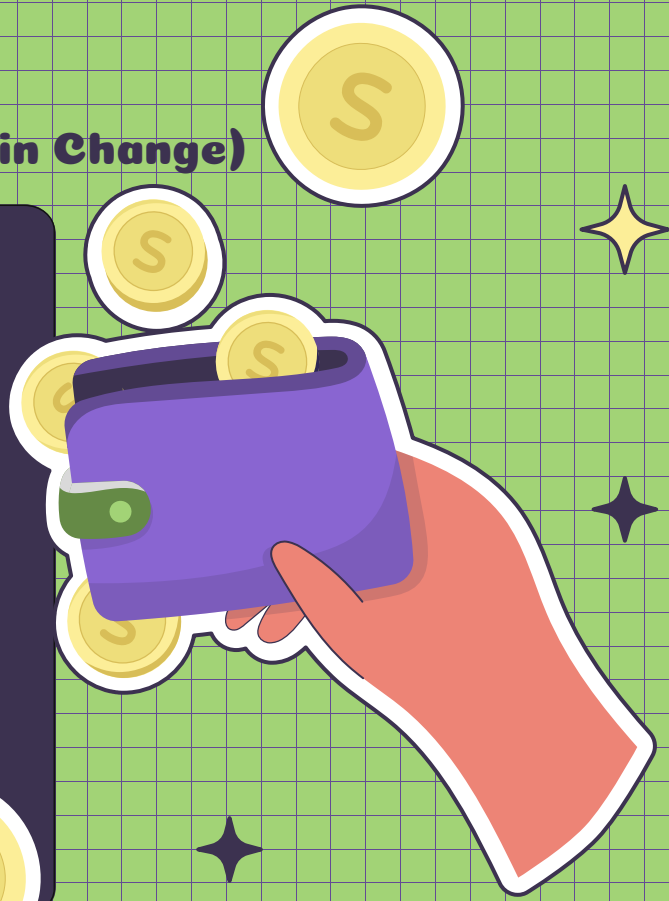
- Adição de regras: obter o menor valor possível e ocupar a capacidade total da mochila

Consiste em determinar a **menor quantidade de moedas** possíveis para fornecer o **troco** de um valor

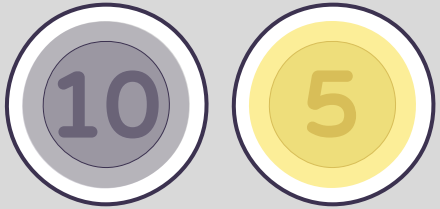
- Qual a combinação de moedas é a mais eficiente para um determinado troco?

Descrição formal

Dado um conjunto de **moedas A** e um **troco W**, qual é a menor quantia de moedas presentes em A que seus valores somados resultam em W?



Exemplo de troco → 15c



=

2 ✓

=

3 ✗

=

6 ✗

Resoluções Possíveis

Programação Dinâmica

- Complexidade O(n . W)
- Utiliza tabela para armazenar os valores mínimos de moedas para cada valor de troco possível
- Percorre 2 *loops* aninhados e verifica se a moeda é menor ou igual ao valor do troco e se a quantidade de moedas é a mínima
- Mais lento, mas funciona independente do sistema de moedas

Algoritmo Guloso

- Complexidade O(n)
- Seleciona a maior moeda disponível que seja menor ou igual ao valor do troco e repete o processo
- O conjunto de moedas deve estar ordenado de forma decrescente
- Mais rápido, mas exige sistema de moedas canônico

02

P. Dinâmica

O que é Programação Dinâmica

Programação Dinâmica

- A programação dinâmica é uma **abordagem algorítmica sofisticada e altamente eficiente** amplamente empregada na **resolução de problemas computacionalmente intensivos**, especialmente aqueles relacionados à otimização
- Essa estratégia se baseia na **decomposição de um problema complexo** em uma hierarquia de **subproblemas menores e mais gerenciáveis**
- A divisão permite que **instâncias maiores e mais complexas sejam solucionadas em um tempo razoável**, resultando em uma redução drástica da complexidade do tempo de execução

Programação Dinâmica

- Uma das ideias-chave por trás da programação dinâmica é a **reutilização de soluções ótimas para subproblemas previamente resolvidos**, isso é feito por meio da **memorização de soluções intermediárias**
- Ao guardar essas soluções, **evita-se a repetição desnecessária de cálculos**
- Esse aspecto da programação dinâmica permite uma **evolução progressiva**, enquanto as **soluções são atualizadas e aprimoradas**

Programação Dinâmica

- Para aplicar a programação dinâmica de maneira eficaz, é fundamental identificar **dois atributos** nos problemas: sobreposição de subproblemas e propriedade de subestrutura ótima
  - **A sobreposição de subproblemas** ocorre quando um **problema maior pode ser dividido em subproblemas menores** e independentes entre si
  - **A propriedade de subestrutura ótima** indica que uma **solução ótima para o problema global contém soluções ótimas para os subproblemas locais**

Programação Dinâmica

Divisão e Conquista

+

Armazenamento

+

Dependência de soluções

=

Programação dinâmica

A. Guloso

O que é Algoritmo Guloso

Algoritmo Guloso

- Algoritmos Gulosos são **abordagens heurísticas** que adotam escolhas localmente ótimas em cada etapa para **buscar uma solução globalmente ótima**
- Esses algoritmos **não revisam decisões tomadas anteriormente**, o que pode resultar em **soluções subótimas em alguns casos**
- Eles **selecionam a melhor opção disponível no momento**, sem considerar o impacto das escolhas futuras

Algoritmo Guloso

- **Algoritmo de Dijkstra** é um exemplo clássico de algoritmo guloso, usado para **encontrar o caminho mais curto** em um grafo ponderado
- Outro exemplo é o **problema do pen drive**, em que é necessário **selecionar arquivos para maximizar a quantidade de dados armazenados** em um espaço limitado
- Algoritmos gulosos geralmente têm **menor complexidade** em comparação com outras técnicas, o que os torna eficientes em muitos problemas
- No entanto, em cenários específicos, **os algoritmos gulosos podem não alcançar a solução ótima**

04

Implementação

Implementação e resultados obtidos

Implementação em **Python**

- Compatibilidade com todos os SOs
- Facilidade e praticidade

Simula **3 cenários** diferentes

- Sistema canônico
- Sistema não-canônico
- Teste de velocidade com valores altos

Adaptado para o contexto de **notas em Reais**

- As moedas simbolizam notas de R$1 a R$200

Programação Dinâmica

Cria uma lista de tamanho **valor do troco + 1** para armazenar os valores mínimos de moedas necessárias para cada valor de troco possível

Inicializa a lista com **infinito** e define **0 para a posição 0**, pois não são necessárias moedas para atingir um troco de zero

Para cada valor **de 1 até o valor do troco** desejado, segue os seguintes passos:

- Percorre todas as moedas disponíveis
- Se o valor da moeda for menor ou igual ao valor do troco atual e a soma do valor da moeda com o valor mínimo de moedas necessárias para alcançar o troco atual for menor que o valor mínimo atual na lista, atualiza o valor mínimo na lista

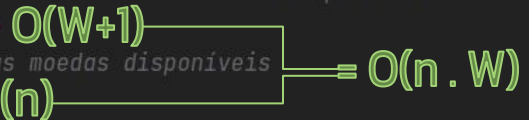
O **valor mínimo na última posição** da lista será a **resposta final**, ou seja, o **número mínimo de moedas** necessário para fornecer o troco desejado

```
Moedas disponíveis: [1, 5, 10]
Digite o valor do troco desejado: R$15
Abordagem com Programação Dinâmica
Inicialização: [inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf]
Definição do troco 0: [0, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf]
[0, 1, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf]
[0, 1, 2, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf]
[0, 1, 2, 3, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf]
[0, 1, 2, 3, 4, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf]
[0, 1, 2, 3, 4, 5, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf]
[0, 1, 2, 3, 4, 1, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf]
[0, 1, 2, 3, 4, 1, 2, inf, inf, inf, inf, inf, inf, inf, inf, inf]
[0, 1, 2, 3, 4, 1, 2, 3, inf, inf, inf, inf, inf, inf, inf, inf]
[0, 1, 2, 3, 4, 1, 2, 3, 4, inf, inf, inf, inf, inf, inf, inf]
[0, 1, 2, 3, 4, 1, 2, 3, 4, 5, inf, inf, inf, inf, inf, inf]
[0, 1, 2, 3, 4, 1, 2, 3, 4, 5, 6, inf, inf, inf, inf, inf]
[0, 1, 2, 3, 4, 1, 2, 3, 4, 5, 2, inf, inf, inf, inf, inf]
[0, 1, 2, 3, 4, 1, 2, 3, 4, 5, 1, inf, inf, inf, inf, inf]
[0, 1, 2, 3, 4, 1, 2, 3, 4, 5, 1, 2, inf, inf, inf, inf]
[0, 1, 2, 3, 4, 1, 2, 3, 4, 5, 1, 2, 3, inf, inf, inf]
[0, 1, 2, 3, 4, 1, 2, 3, 4, 5, 1, 2, 3, 4, inf, inf]
[0, 1, 2, 3, 4, 1, 2, 3, 4, 5, 1, 2, 3, 4, 5, inf]
[0, 1, 2, 3, 4, 1, 2, 3, 4, 5, 1, 2, 3, 4, 5, 6]
[0, 1, 2, 3, 4, 1, 2, 3, 4, 5, 1, 2, 3, 4, 5, 2]
Finalização [0, 1, 2, 3, 4, 1, 2, 3, 4, 5, 1, 2, 3, 4, 5, 2]
 Mínimo de moedas necessárias para obter o troco de R$15: 2
 Moedas utilizadas: [10, 5]
```

Programação Dinâmica

```python
def troco_programacao_dinamica(moedas, valor):
    '''
    Calcula o mínimo de moedas necessárias para dar o troco utilizando programação dinâmica

    Args:
        moedas (list): Lista das moedas disponíveis
        valor (int): Valor do troco desejado em centavos

    Returns:
        tuple: Tupla contendo o mínimo de moedas necessárias e a lista das moedas utilizadas
               Retorna (-1, []) se não for possível obter o troco
    '''

    # Lista de tamanho 'valor+1' para armazenar a quantidade mínima de moedas necessárias para
    # cada valor de 0 a 'valor' e inicializa em infinito 'float('inf')'
    min_moedas = [float('inf')] * (valor+1)

    # Definir o troco de 0 como 0, pois ele não tem troco
    min_moedas[0] = 0

    # Lista para armazenar as moedas utilizadas para cada valor de 0 a 'valor'
    moedas_utilizadas = [[] for _ in range(valor + 1)]
```

```python
# Loop para calcular a quantidade mínima de moedas para cada valor de 1 a 'valor'
for i in range(1, valor+1):
    # Loop para percorrer as moedas disponíveis
    for moeda in moedas:
        # Verifica se a moeda atual é menor ou igual ao valor atual (pode ser utilizada)
        # e se a moeda atual pode diminuir a atual menor quantidade de moedas (compara se a
        # quantidade mínima com a moeda é menor do que a quantidade mínima atual)
        if moeda <= i and min_moedas[i - moeda] + 1 < min_moedas[i]:
                # Atualiza a quantidade mínima de moedas para o valor atual
                min_moedas[i] = min(min_moedas[i], min_moedas[i-moeda] + 1)
                # Armazena as moedas utilizadas para o valor atual
                moedas_utilizadas[i] = moedas_utilizadas[i - moeda] + [moeda]

# Verificar se é possível dar o troco exato com as moedas disponíveis
if min_moedas[valor] == valor + 1:
    return -1, []

# Retorna a quantidade mínima de moedas e a lista de moedas utilizadas
return min_moedas[valor], moedas_utilizadas[valor]
```



Algoritmo Guloso

Recebe a lista de moedas em **ordem decrescente** e inicializa a contagem de moedas para o troco em 0

Para **cada moeda na lista de moedas**, verifica se é menor ou igual ao valor do troco

- Indica que pode ser utilizada para compor o troco
- **Divide** ela pelo valor do troco para saber **quantas pode utilizar**
- **Atualiza** a quantidade mínima de moedas com o valor obtido na divisão
- Subtrai do troco o valor total das moedas utilizadas para saber o valor de troco restante

Considera a **maior moeda disponível** e pode não levar sempre ao resultado ótimo, como em cenários em que a **moeda 1** não está disponível (sistema **não-canônico**)

Algoritmo Guloso

```python
def troco_algoritmo_guloso(moedas, valor):
    '''
    Calcula o mínimo de moedas necessárias para dar o troco utilizando algoritmo guloso

    Args:
        moedas (list): Lista das moedas disponíveis
        valor (float): Valor do troco desejado em centavos

    Returns:
        tuple: Tupla contendo o mínimo de moedas necessárias e a lista das moedas utilizadas
               Retorna (-1, []) se não for possível obter o troco
    '''

    # Inicializar a contagem de moedas mínimas
    min_moedas = 0

    # Criar lista para armazenar as moedas utilizadas
    moedas_utilizadas = []
```

```python
# Loop para iterar sobre as moedas disponivel
for moeda in moedas:
    # Verificar se a moeda atual pode ser utilizada para o troco
    if valor >= moeda:
        # Calcular a quantidade máxima de moedas daquele valor que pode ser utilizada
        quant_moedas = valor // moeda
        # Atualizar a contagem de moedas
        min_moedas += quant_moedas
        # Adiciona a moeda atual multiplicada pela quantidade de vezes
        # que ela é utilizada à lista de moedas utilizadas
        moedas_utilizadas.extend([moeda] * quant_moedas)
        # Atualizar o valor do troco restante
        valor -= quant_moedas * moeda

# Verificar se é possível dar o troco exato com as moedas disponíveis
if valor == 0:
    return min_moedas, moedas_utilizadas

return -1, []
```

O(n)

Bora testar!

Obrigado!

Referências:

COIN Change | DP-7. **Geeks for Geeks**, s. d. Disponível em: https://www.geeksforgeeks.org/coin-change-dp-7/. Acesso em: 15 jun. 2023.

CORMEN, Thomas H; RIVEST, Ronald L.; LEISERSON, Charles E.; STEIN, Clifford. **Algoritmos - Teoria e Prática** - 3ª Ed., 2012, São Paulo-SP, Elsevier.

MATTIOLI, Lucas V. **Comparativo entre a estratégia gulosa e a programação dinâmica para o problema do troco com sistemas de moedas canônicos.** Monografia (Graduação em Engenharia de Software) - Universidade de Brasília, Brasília, 2018.