INSTITUTO FEDERAL
Rio Grande do Norte
Campus Natal-central

DIATINF – Diretoria Acadêmica de Gestão e Tecnologia da Informação
TADS – Tecnologia em Análise e Desenvolvimento de Sistemas

Algoritmos 2024.2
Avaliação 22/02/2025
Professor Jorgiano Marcio Bruno Vidal

Aluno _____

1. **Problema do troco**
Um sistema de moedas é uma tupla $S = (c_1, c_2, c_3, \ldots, c_{n-1}, c_n)$
onde c_1 representa o valor da primeira peça, c_2 o da segunda e assim sucessivamente.
Dado um sistem S e um inteiro positivo v, o problema do troco é um problema de **otimização combinatória** que consiste em encontrar uma tupla de inteiros $T = (x_1, x_2, \ldots, x_n)$ que minimize$\sum_{i=i}^{n} x_i$, com a restrição $\sum_{i=1}^{n} x_i c_i = v$
Dessa forma, v representa a soma, x_i a quantidade de moedas c_i a utilizar.
De forma simplificada, o problema deve encontrar a quantidade mínima de moedas no sistema S a usar para obter o valor v.
Exemplo:
Considere o sistema $S = (1, 2, 5, 10, 20, 50, 100, 200, 500)$ e o valor a ser obtido $v = 7$. Para esse caso vamos usar apenas a tupla $(1, 2, 5)$ (um, dois e cinco unidades monetárias), já que os valores a partir de 10 são inúteis para esse caso.
Uma forma de de se obter o valor 7 é através da tupla $T = (7, 0, 0)$, onde temos 7 moedas de 1, 0 moedas de 2 e 0 de 5. As outras tuplas são: $(5, 1, 0)$, $(3, 2, 0)$, $(1, 3, 0)$, $(2, 0, 1)$ e $(0, 1, 1)$.
A solução para o problema de moedas $(S, 7)$ é a tupla (x_1, x_2, x_3) que minimiza o número total $x_1 + x_2 + x_3$. Nesse caso a solução é a tupla $T = (0, 1, 1)$, ou seja, 1 moeda de 2 e uma moeda de 5. Temos então que $M_{(1,2,5)}(7) = 2$.
Sobre o problema da moeda:

   (a) Descreva uma solução usando o método guloso.

   (b) Descreva situações em que o método guloso deve ser aplicado e situações em que o método guloso não resolve o problema do troco.

   (c) Podemos definir uma função recursiva para resolver o problema do troco, chamada `troco(M, V)`, que retorna a quantidade mínima de moedas, usando as moedas do conjunto M para o valor V como sendo.

$$troco(M, V) = min(troco(C, V - c_i) + 1) \forall c_i \in C$$

   Considere um sistema com 3 moedas $C = \{1, 3, 5\}$, ao aplicarmos a função `troco()` ao valor 13 podemos dizer a menor quantidade de moedas é

$$troco(C, 13) = 1 + min(troco(C, 13 - 1), troco(C, 13 - 2), troco(C, 13 - 5))$$

   Com 2 casos de base, quando $V = 0$, retorna 0 e quanto $V < 0$ deve-se retornar um valor indicando a impossibilidade de gerar esse valor, ou seja, ∞.

   A função completa pode ser definida como:

$$troco(C, V) = \begin{cases} 0 & Se \quad V = 0 \\ \infty & Se \quad V < 0 \\ 1 + min(troco(C, V - c_i)) & Se \quad V > 0 \end{cases}$$

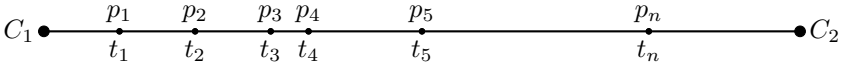
   Podemos definir a função `troco(v)` que retorna a quantidade mínima de moedas

   Escreva uma função em C++ que receba a lista de valores de moedas C, o valor a ser obtido V e retorne a quantidade mínima de moedas a serem usadas para obter o valor V.

   (d) Descreva uma solução para o problema do troco em moedas usando programação dinâmica.

   (e) Implemente, em C/C++, uma função recursiva com *memoization* para evitar a repetição de cálculos.

2. **Problema do abastecimento de combustível**
Um estrada entre 2 cidade C_1 e C_2 possui n postos de combustível para reabastecimento $P = \{p_1, p_2, p_3, \ldots, p_n\}$, onde P_i é a distância da cidade de origem C_1 ao posto i. Cada posto tem um tempo total de completar o tanque de combustível diferente t_i (para o posto i), devido ao tempo de espera e capacidade da bomba. O valor do combustível é o mesmo, então essa variável pode ser ignorada.
Seu carro pode percorrer uma distância fixa de 100km com o tanque cheio. Em cada posto que você decidir parar você deve encher o tanque por completo. Você quer chegar o mais rápido possível em C_2



(a) Descreva um algoritmo que determine o menor tempo possível que você estará nos postos reabastecendo.
DICA: Chegando em C_2, você deve ter parado em algum dos postos a menos de 100km da chegada.

(b) Escreva, em C/C++, uma função para retornar o menor tempo gasto nos postos.

(c) Escreva 3 casos de testes, explicando manualmente a solução ótima para cada um.

(d) Otimize o algoritmo usando a técnica de programação dinâmica. Descreva a tabela a ser gerada.

3. **Problema da loja de livros**
Considere uma loja com N livros a venda. Cada livro i possui um preço de venda p_i e uma quantidade de páginas q_i. Um cliente, com orçamento fixo X decide comprar a maior quantidade de páginas possível com esse valor.

(a) Descreva uma função, usando *backtracking*, que determine a quantidade de páginas máxima que ele consegue comprar.

DICA: Cada livro pode ou não estar na lista de livros a ser comprado.

(b) Implemente a solução com programação dinâmica.

(c) Descre a tabela e faça uma função não recursiva que preencha a tabela completa.

(d) Defina casos de teste e mostre o preenchimento da tabela.

DICA: Faça um função para imprimir a tabela

(e) Faça uma análise de desempenho comparativa sem a programação dinâmica e com a programação dinâmica.

4. **Problema das moedas**
Um sistema monetário é composto de C moedas, cada uma com um valor c_i. Dado um valor V, determine de quantas formas **distintas** e **ordenadas** é possível obter o valor V.
Exemplo: Um sistema com 3 moedas $C = \{2, 3, 5\}$ possui 3 formas diferentes de obter o valor 9.

- $\{2 + 2 + 5\}$
- $\{3 + 3 + 3\}$
- $\{2 + 2 + 2 + 3\}$

(a) Define a função de recorrência para identificar de quantoas formas diferentes, ordenadas, é possível obter o valor V.

(b) Escreva uma função recursiva que conte de quantas formas é possível obter o valor.

(c) Defina a tabela para a implementação com programação dinâmica.

(d) Implemente o preenchimento da tabela com programação dinâmica.

(e) Faça uma análise de desempenho comparativa sem a programação dinâmica e com a programação dinâmica.

Boa avaliação!
Natal, 22/02/2025