Instituto Tecnológico de Aeronáutica
CES-12 - Algoritmos e Estruturas de dados II
Aluno: Bernardo Hoffmann da Silva

Relatório do Laboratório 4: Paradigmas da programação

A fim de realizar comparações entre os algoritmos, pode-se plotar gráficos que auxiliem na análise. Esses estão presentes nas figuras 1, 2, 3, 4, 5 e 6. Comparou-se o tempo de execução de cada algoritmo e o número de moedas retornadas, com a finalidade de verificar se o algoritmo é ótimo.

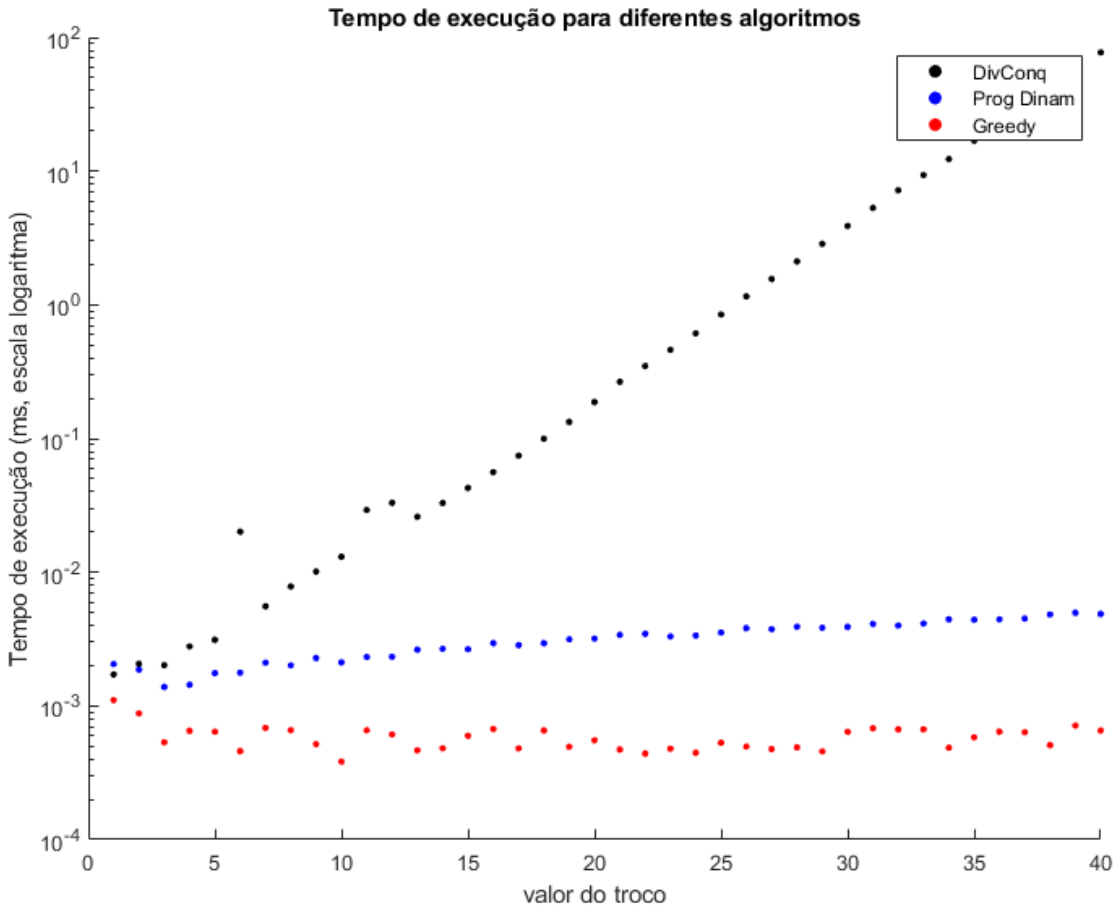

Figura 1: Tempo de execução para cada valor de troco, para os diferentes algoritmos.

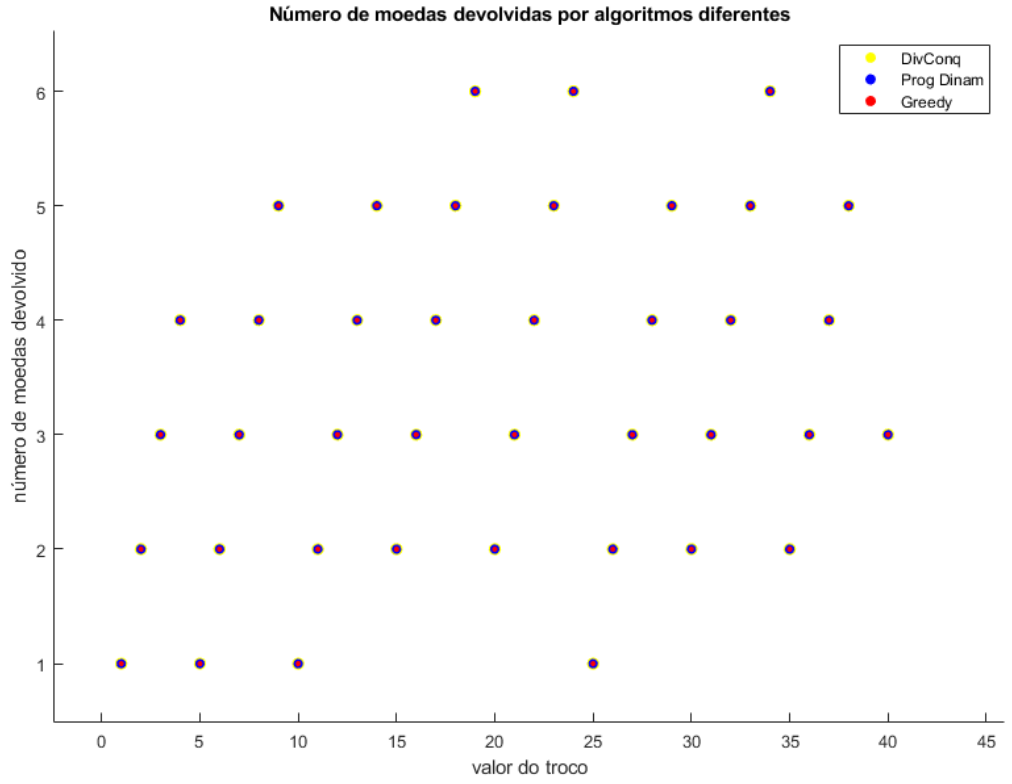

Figura 2: Número de moedas retornadas para cada valor de troco, em denominação de centavos brasileira

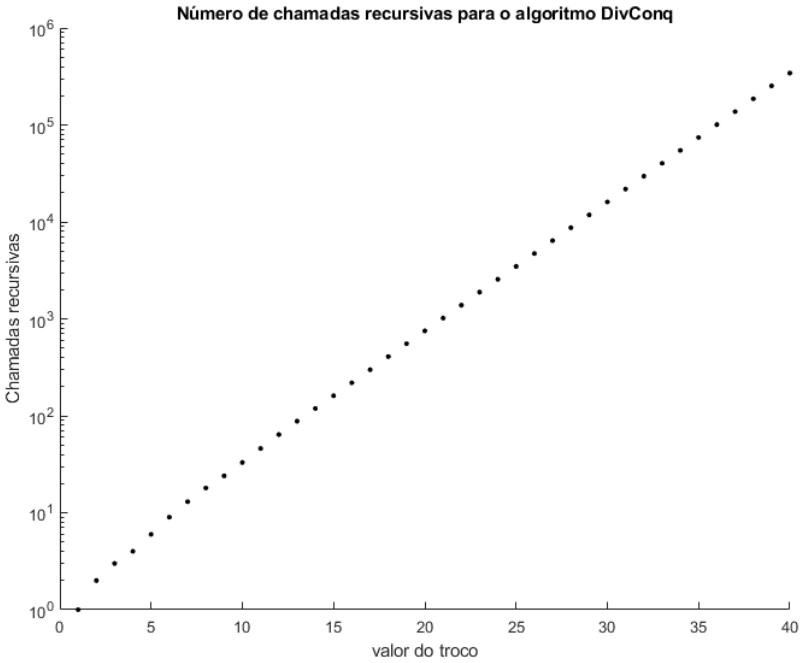

Figura 3: Chamadas recursivas para valores de troco de até 40, no divisão e conquista.

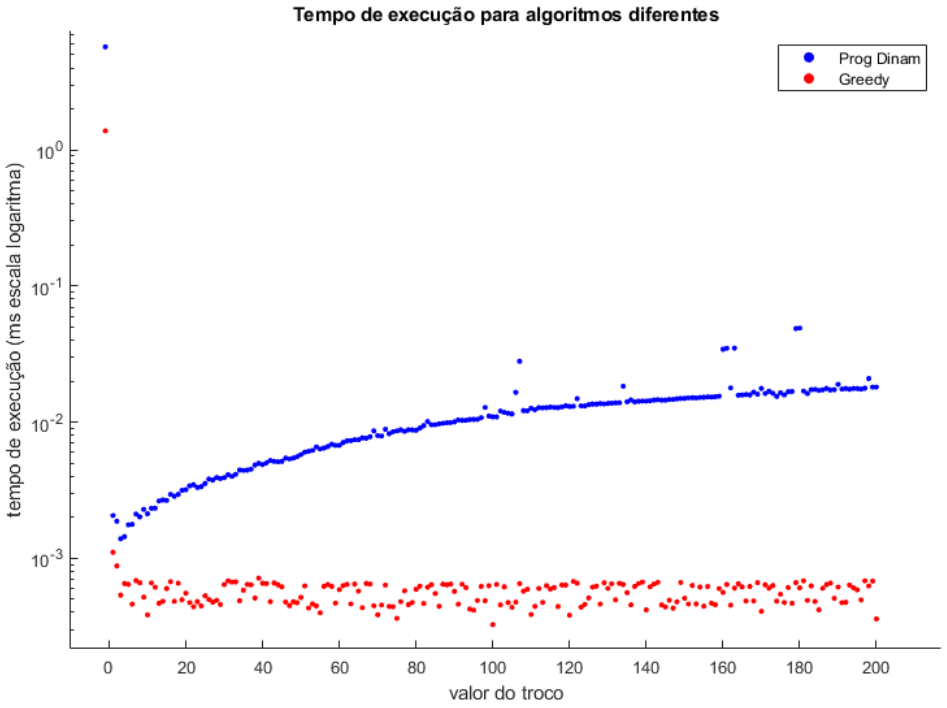

Figura 4: Tempo de execução para valores de troco de até 200 centavos, em sistema de centavos brasileiro.

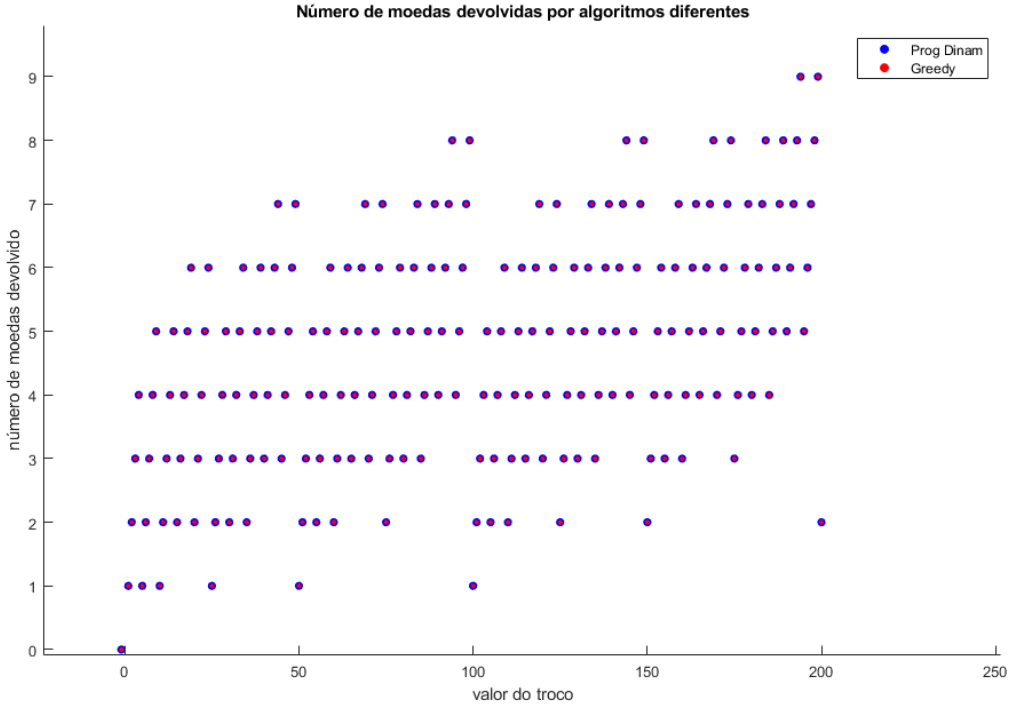

Figura 5: Número de moedas devolvidas para valores de troco de até 200 centavos brasileiros.

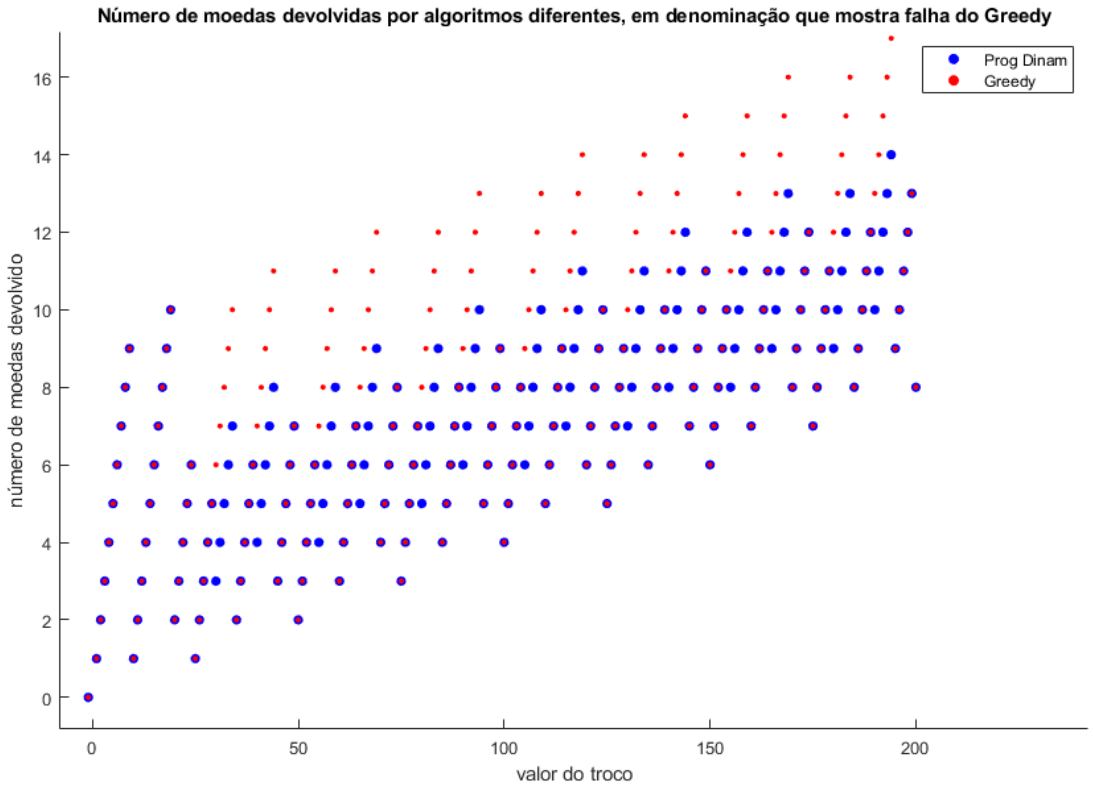

Figura 6: Número de moedas devolvidas para denominação em que o algoritmo Greedy falha.

Percebe-se que, para o sistema de moedas brasileiro, testado para um troco de até 40 centavos, os três algoritmos para o problema do troco retornam o número ótimo de moedas, isto é, o menor possível. Já para o tempo de execução, nota-se que o algoritmo de divisão e conquista apresenta um tempo muito maior que os outros dois. Esse crescimento exponencial do tempo de execução está relacionado ao número de chamadas recursivas, que acabam resolvendo subproblemas que já foram resolvidos em outras etapas do algoritmo. Logo, não é muito eficiente para essa situação. A figura 3 ilustra a quantidade de chamadas recursivas feitas conforme aumenta os valores de troco. Para o valor 35, nota-se que são feitas mais de 10^5 chamadas.

Em análise de tempo e de moedas para o Greedy (GR) e para a Programação Dinâmica (PD), percebe-se que no sistema de centavos brasileiro ambos entregam os resultados ótimos (figura 5) . Analisando-se o tempo, percebe-se que o troco em PD é um algoritmo que demora mais que o GR. Podemos atribuir isso às diversas indexações e buscas à valores realizadas em PD, o que gera uma complexidade relativamente maior para o algoritmo. Isso se traduz, entretanto, em otimalidade em todos os casos, independente do sistema de moedas utilizado.

Percebe-se, pela figura 6, como o algoritmo GR pode falhar dependendo da denominação de moedas escolhida. Isso ocorre pois o GR escolhe sempre primeiro a maior moeda possível para o troco, gerando casos em que o número ótimo de moedas não é retornado pelo algoritmo, pois moedas de valores menores seriam mais ideais. EX: (30 = 25 + 1 + 1+ 1 + 1 + 1 (GREEDY) ou 30 = 10 + 10 + 10 (ÓTIMO)).

Nesse caso, a programação dinâmica resolve o problema encontrado pela divisão e conquista, guardando valores calculados e evitando repetições e iterações desnecessárias. Isso aumenta a complexidade do algoritmo, mas garante otimalidade em um tempo relativamente pequeno. Por exemplo, para um subproblema já resolvido, a PD apenas coleta os valores já indexados no vetor.

O algoritmo de divisão e conquista, apesar de ser lento para o problema analisado, não deve ser descartado para outras situações, pois esse tempo grande não é devido a uma limitação inerente do paradigma. Um exemplo em que a divisão e conquista resolve um problema rapidamente é a busca binária. Nele, busca-se um elemento do vetor dividindo-se o vetor em outros dois e aplicando o mesmo algoritmo recursivo. O tempo para a busca binária é O(log(n)), o que é muito eficiente. Nesse caso, não há resolução de subproblemas já resolvidos, já que a subdivisão não possui interseções. Portanto, a divisão e conquista torna-se rápida. Para o problema do troco, há muitas interseções entre os subproblemas, já que calcula-se o troco de 1 centavo diversas vezes, por exemplo. Essa característica, quando em grande escala, aumenta muito o tempo do algoritmo. Conclui-se então, que é muito mais eficiente aplicar a divisão e conquista em algoritmos em que não há interseções entre os subproblemas resolvidos.