Técnicas de Programação

Vinicius A. Matias

May 30, 2021

1 Introdução

Este relatório passa pela definição e análise de complexidade para algoritmos que seguem três técnicas diferentes. O estudo começa pela Divisão e Conquista, seguido de Tentativa e Erro e terminando com Algoritmos Gulosos.

2 Divisão e Conquista

Divisão e Conquista é uma técnica de programação que segue o princípio de indução forte. Nessa abordagem um problema é decomposto em problemas menores (divisão) que conseguem ser resolvidos. A resolução dos subproblemas é feita recursivamente e também é chamada de conquista. O problema final solucionado vem da combinação das conquistas. Um algoritmo conhecido de divisão e conquista e que foi discutido no relatório 2 (Recursão) é o da busca binária, consistindo de dividir o problema no meio (irmos para o lado esquerdo ou direito) e a conquista é a resolução recursiva desses problemas (comparação entre o arranjo e o valor), para na combinação dos resultados retornar a resposta correta.

Um algoritmo de divisão e conquista segue uma equação de recorrência como:

$$T(n) = \begin{cases} \Theta(1), n \leq c \\ aT(\frac{n}{b}) + D(n) + C(n), n > c \end{cases}$$

Onde $aT(\frac{n}{b})$ é o custo da conquista. A conquista é formada por a chamadas recursivas, e b é o tamanho da divisão (se dividirmos por 2, $b = 2$);

$D(n)$ é o custo da divisão e $C(n)$ é o custo da combinação. Note que D e C não necessariamente englobarão a operação de interesse.

Uma equação de recorrência para algoritmos de divisão e conquista que dividem o problema inicial em parcelas de tamanhos iguais também pode ser identificada como:

$T(n) = aT(\frac{n}{b}) + f(n)$

Onde $f(n)$ é o custo da divisão mais a combinação.

E a imensa maioria dos algoritmos que seguem essa última equação de recorrência podem ter a complexidade assintótica identificada por meio do Teorema Mestre.

2.1 Teorema Mestre

A definição à seguir do teorema mestre provém do livro Algoritmos: Teoria e Prática (Cormen et al., 2012):

Sejam $a \geq 1$ e $b > 1$ constantes. Seja $f(n)$ uma função, e seja $T(n)$ definida no domínio dos números inteiros não negativos pela recorrência

$T(n) = aT(n/b) + f(n)$.

Então, $T(n)$ tem os seguintes limites assintóticos:

1. Se $f(n) \in \mathcal{O}(n^{\log_b a - \epsilon})$ para alguma constante $\epsilon > 0$, então $T(n) \in \Theta(n^{log_b a})$.

2. Se $f(n) \in \Theta(n^{log_b a})$, então $T(n) \in \Theta(n^{\log_b a} \log n)$.

3. Se $f(n) \in \Omega(n^{logba+\epsilon})$ para alguma constante $\epsilon > 0$, e se $af(n/b) \leq cf(n)$ para alguma constante $c < 1$ e todos os n suficientemente grandes, então $T(n) \in \Theta(f(n))$.

2.2 Exemplos de Aplicação do Teorema Mestre

I - $T(n) = 9T(n/3) + n$

Descobrir a complexidade assintótica da equação de recorrência $T(n) = 9T(n/3) + n$.

Para utilizar o teorema mestre neste problema, definimos:

$a = 9$,

$b = 3$

$f(n) = n$

Utilizando o teorema mestre, começaremos verificando se $f(n) \in \Theta(n^{\log_b a})$ - cláusula 2.

Veja que $\Theta(n^{\log_b a}) = \Theta(n^{\log_3 9}) = \Theta(n^2)$
E $\Theta(n^2)$ não cresce com a mesma velocidade que $f(n)$, ou seja, $f(n) \notin \Theta(n^2)$

Testando então com a cláusula 1 (notar que $f(n)$ cresce menos que $\Theta(n^2)$ ajuda a escolher esta opção):
Escolhendo um $\epsilon = 6$, percebemos que $\mathcal{O}(n^{\log_b a - \epsilon}) = \mathcal{O}(n^{\log_3 9 - 6}) = \mathcal{O}(n^{\log_3 3}) = \mathcal{O}(n)$
E $f(n) \in \mathcal{O}(n)$, logo:
$T(n) \in \Theta(n^2)$

II - $T(n) = 4T(n/2) + n^3$

$a = 4$; $b = 2$; $f(n) = n^3$
Verificando a cláusula 2 do Teorema Mestre:
$\Theta(n^{\log_2 4}) = \Theta(n^2)$
E $f(n) = n^3 \notin \Theta(n^2)$ (falha da cláusula 2)
Como $f(n)$ cresce mais rápido que $\Theta(n^2)$, buscaremos na cláusula 3 verificar se $f(n)$ obedece o crescimento mínimo para um ϵ
A primeira verificação da terceira cláusula do teorema mestre que verificaremos é se $af(n/b) \leq cf(n)$, para alguma constante $c < 1$:

$4 * (\frac{n}{2})^3 \leq cn^3$
$= 4 * \frac{n^3}{8} \leq cn^3$
$= n^3/2 \leq cn^3$
$= 1/2 \leq c$
Ou seja, a inequação é verdadeira para algum $c < 1$, como exemplo $c = 1/2$
Agora verificaremos se $f(n) \in \Omega(n^{\log_b a + \epsilon})$:
$\Omega(n^{\log_2 4 + \epsilon})$
$\Omega(n^{\log_2 4 + 4})$, com $\epsilon = 4$
$\Omega(n^{\log_2 8})$
$\Omega(n^3)$
E $f(n) \in \Omega(n^3)$
Logo, $T(n) \in \Theta(f(n)) = T(n) \in \Theta(n^3)$ para n suficientemente grande.

3 Tentativa e Erro

Essa técnica de programação consiste em buscar exaustivamente as possibilidades que um problema permite, gerando uma árvore de estados à se chegar mediante uma ação, chegando à solução ótima em algum momento. O tempo para chegar à melhor solução, contudo, não costuma ser viável.

Esses algoritmos também dependem de uma condição de parada, que costuma ser chegar na solução ótima. Cada passo de um algoritmo de backtracking (tentativa e erro) costuma ser feita por recursão, pois em uma falha encontrada, é simples de se chegar no estado anterior e tentar outra possibilidade (daí o termo backtracking). É muito comum que as soluções de tentativa e erro tenham complexidade exponencial.

4 Algoritmos Gulosos

Essa classe de algoritmos seguem uma ideia parecida aos algoritmos de tentativa e erro, mas quando se deparam com uma escolha de ações à serem feitas, algoritmos gulosos seguem uma estratégia para escolher a opção mais promissora para resolver o problema. A heurística que permite diferir entre uma opção e outra é o critério guloso (uma decisão localmente ótima).

5 Referências

Cormen, T.H.; Leiserson, C.E.; Rivest, R.L.; Stein, C. **Algoritmos: Teoria e Prática**. Tradução da 3a edição americana. Elsevier, 2012.