Estrutura de Dados II

Universidade de Vila Velha

Comprometida com a excelência no ensino, pesquisa e inovação.

wanderson.santana@uvv.br

1 Princípios da análise de algoritmos

Resumo

Nesta unidade, faremos uma introdução à análise de algoritmos. Apresentaremos os conceitos de problema computacional, instância e tamanho da entrada. Também discutiremos a noção de desempenho (eficiência) de algoritmos, abordando os conceitos de melhor caso, pior caso e caso médio, bem como a análise assintótica. A análise de algoritmos busca estimar a relação entre o tempo de execução (ou espaço) de um algoritmo e o tamanho de sua entrada, sem depender de implementação específica ou plataforma [1].

"Sou muito grato às dificuldades que se apresentaram na minha vida. Elas me ensinaram a ter tolerância, simpatia, autocontrole, perseverança e outras virtudes que, sem essas dificuldades, eu jamais conheceria."

Introdução

Um problema computacional é uma tarefa bem definida que se deseja resolver utilizando computadores. Ele é descrito por uma entrada, que representa os dados ou parâmetros, e uma saída, que corresponde à solução esperada. Por exemplo, o problema da multiplicação de dois inteiros envolve como entrada dois números a e b, e como saída o produto $a \times b$ [2].

Uma instância de um problema é uma atribuição concreta de valores à entrada. Cada instância possui uma solução específica.

Definição: Instância

Uma instância é um conjunto de valores específicos atribuídos aos parâmetros de entrada de um problema computacional.

Exemplo: Multiplicação de inteiros

Dado o problema de multiplicar dois números naturais u e v, uma instância seria $u = 102$, $v = 452$, cuja solução é $u \times v = 46104$.

Para resolver um problema computacional, desenvolve-se um algoritmo — uma sequência finita de passos bem definidos que, fornecidos os dados de entrada, produz uma saída correta em tempo finito [3].

Definição: Algoritmo

Um algoritmo é uma sequência finita de instruções não ambíguas, executáveis, que, a partir de uma entrada, produzem uma saída em tempo finito.

Exemplo: Equação do segundo grau

Dado o problema de encontrar soluções inteiras da equação $ax^2+bx+c = 0$, uma instância com $a = 1$, $b = 2$ e $c = 3$ não possui solução no conjunto dos inteiros.

A análise de algoritmos busca avaliar algoritmos quanto à corretude, eficiência (tempo e espaço) e adequação. Frequentemente, existe mais de um algoritmo correto para um problema, e a análise permite escolher o mais adequado a uma aplicação específica [1].

Exemplo: Ordenação de vetor

Dado um vetor $A[1..n]$ de inteiros, deseja-se ordená-lo em ordem crescente. Para $A = (876, 145, 323, 112, 221)$, a saída esperada é $(112, 145, 221, 323, 876)$.

Tamanho de uma Instância

O tamanho de uma instância representa a quantidade de informação necessária para descrever completamente a entrada do problema. Em geral, usa-se um número natural para expressar esse tamanho [2].

- No problema de multiplicação de inteiros, uma medida possível de tamanho é o número total de bits ou dígitos para representar ambos os números.
- No problema de ordenação, o tamanho da instância geralmente é n, o número de elementos do vetor.
- No problema do ciclo máximo em um dígrafo, o tamanho da entrada pode ser descrito por (n, m), número de vértices e arcos, respectivamente.

Algoritmos para problemas

A solução de uma instância de um dado problema pode ser um número, um vetor, um valor booleano, etc., dependendo da natureza do problema. Já a solução de um problema é sempre um algoritmo.

Definição: Algoritmo

Dizemos que um algoritmo resolve um dado problema se, ao receber a descrição de qualquer instância do problema, devolve uma solução da instância ou informa que a instância não tem solução.

Essa exigência é consideravelmente complexa, pois obriga o algoritmo a resolver não só as instâncias que aparecem em aplicações práticas como também aquelas instâncias patológicas que não parecem ser razoáveis [3], ou seja, **existe uma dificuldade natural de se projetar algoritmos eficientes e corretos em todos os casos possíveis**.

Desempenho de um algoritmo

Acontece que o comportamento e desempenho de um algoritmo envolve o uso de recursos computacionais como memória, largura de banda e, principalmente, tempo. Para descrever o uso desses recursos, levamos em consideração o tamanho da entrada e contamos a quantidade de passos básicos que são feitos pelo algoritmo. O tamanho da entrada depende muito do problema que está sendo estudado: em vários problemas, como o de ordenação descrito acima, o tamanho é dado pelo número de elementos na entrada; em outros, como o problema de somar dois números, o tamanho é dado pelo número total de *bits* necessários para representar esses números em notação binária. Com relação a passos básicos, consideraremos operações simples que podem ser feitas pelos processadores comuns atuais, como por exemplo somar, subtrair, multiplicar ou dividir dois números, atribuir um valor a uma variável, ou comparar dois números[1] [1].

Suponha que A é um algoritmo para um certo problema. Seja $t(I)$ a quantidade de tempo que A consome para processar uma instância I do problema. Desejamos estudar o comportamento de t em função do **tamanho** das instâncias. É necessário lembrar que, via de regra, um problema tem muitas instâncias diferentes de um mesmo tamanho e A consome em tempo diferente para cada uma. Assim, para cada n, sejam:

$$T_*(n) \text{ e } T^*(n) \tag{1.1}$$

o mínimo e o máximo, respectivamente, de $t(I)$ para todas as instâncias I que têm tamanho n. Assim, $T_*(n) \leqslant t(I) \leqslant T^*(n)$ para toda instância I de tamanho n. Diremos que as funções T_* e T^* medem o consumo de tempo do algoritmo A no **melhor caso** - **(***best case***)** e no **pior caso** - **(***worst case***)**, respectivamente. O melhor caso corresponde às instâncias mais "fáceis" e o pior caso corresponde às instâncias mais "diféceis" [2].

Por exemplo, o consumo de tempo do algoritmo pode ser $200n$ no melhor caso e $3n^2$ no pior caso. Em geral, o consumo de tempo é proporcional ao **número de operações elementares** que o algoritmo executa ao processar a instância. Tipicamente, uma *operação elementar* é uma operação aritmética entre variáveis do algoritmo, ou uma comparação entre duas variáveis, ou uma atribuição de valor a uma variável. Não é necessário contar todas as operações elementares: basta escolher uma operação específica de modo que o consumo de tempo do algoritmo seja proporcional ao número de execuções dessa operação. No problema da ordenação que vimos no exemplo anterior, por exemplo, parece óbvio que a operação relevante é a comparação enter elementos do vetor. Já no problema da equação inteira do segundo grau, deve ficar claro que todas as operações aritméticas que envolvem a, b e c são relevantes.

Exemplo: Avaliação do tamanho de uma instância

Suponha que n mede o tamanho das instâncias de um certo problema. A documentação de um algoritmo para o problema diz que ele consome $10^3 n + 10^6$

[1] Estamos falando aqui de números que possam ser representados por uma quantidade constante de bits, como por exemplo 32 ou 64.

unidades de tempo no melhor caso e $2n^2/10$ no pior caso. Isso faz sentido?

Vamos analisar a documentação do algoritmo e verificar se os tempos de execução fornecidos fazem sentido, tanto para o melhor caso quanto para o pior caso.

Tempo no melhor caso:
O tempo de execução no melhor caso é dado por $10^3n + 10^6$ unidades de tempo. Aqui, n é o tamanho da instância.

- O termo 10^3n é linear em n, ou seja, aumenta proporcionalmente ao tamanho da instância.
- O termo 10^6 é constante, um custo fixo que não depende do tamanho da entrada.

Assim, para valores grandes de n, o termo 10^3n dominará o tempo de execução, e o termo constante poderá ser ignorado em análises assintóticas. Portanto, o tempo no melhor caso é aproximadamente linear.

Tempo no pior caso:
O tempo de execução no pior caso é dado por $\frac{2n^2}{10} = 0,2n^2$ unidades de tempo, que é uma função quadrática. Isso indica que o algoritmo pode ser muito mais lento para instâncias grandes no pior caso.

Conclusão: Sim, os tempos fornecidos fazem sentido e são plausíveis para um algoritmo que pode se comportar de maneira linear no melhor caso e quadrática no pior caso [1].

Análise Assintótica

A análise assintótica é uma ferramenta matemática que permite descrever o comportamento do consumo de recursos de algoritmos para entradas de tamanho muito grande. Essa análise utiliza notações como O, Ω e Θ para comparar funções, abstraindo detalhes constantes e termos de menor ordem [3].

A ideia é considerar a função que descreve o tempo de execução do algoritmo e observar como ela cresce quando o tamanho da entrada tende ao infinito. Por exemplo, uma função que cresce como n^2 é considerada mais cara do que uma que cresce como $n \log n$ para valores grandes de n.

Definição: Notação O (Grande-O)

Uma função $f(n)$ é $O(g(n))$ se existem constantes positivas c e n_0 tais que, para todo $n \geqslant n_0$, temos $f(n) \leqslant c \cdot g(n)$.

Definição: Notação Ω (Grande-Omega)

Uma função $f(n)$ é $\Omega(g(n))$ se existem constantes positivas c e n_0 tais que, para todo $n \geqslant n_0$, temos $f(n) \geqslant c \cdot g(n)$.

Definição: Notação Θ (Grande-Theta)

Uma função $f(n)$ é $\Theta(g(n))$ se é simultaneamente $O(g(n))$ e $\Omega(g(n))$.

Referências Bibliográficas

[1] CORMEN, T. H. et al. *Algoritmos: teoria e prática*. 3. ed.. ed. Rio de Janeiro: Elsevier, 2009.

[2] SEDGEWICK, R.; WAYNE, K. *Algoritmos*. 4. ed.. ed. São Paulo: Pearson Addison Wesley, 2011.

[3] KNUTH, D. E. *The Art of Computer Programming*. vol. 1–3, terceira edição. Reading, MA, USA: Addison-Wesley, 1997.