Universidade do Estado do Rio de Janeiro
Campus Regional - Instituto Politécnico

Algoritmos e Estrutura de Dados

Trabalho e Custo Computacional
Instrutor: Ângelo Mondaini Calvão.
https://github.com/oangelo

Por Navar Nunes da Silva.
capunche@hotmail.com

Entregue: 17 de Janeiro de 2019.

Introdução

Independente da linguagem de programação utilizada, um problema possui inúmeras aplicações, mesmo em sua forma “pura”. Mesmo com alguns problemas específicos não possuírem uma solução completamente eficiente, dependendo da extensão do problema, a complexidade da sua solução é de grande importância para uma velocidade maior de resolução.

Complexidade

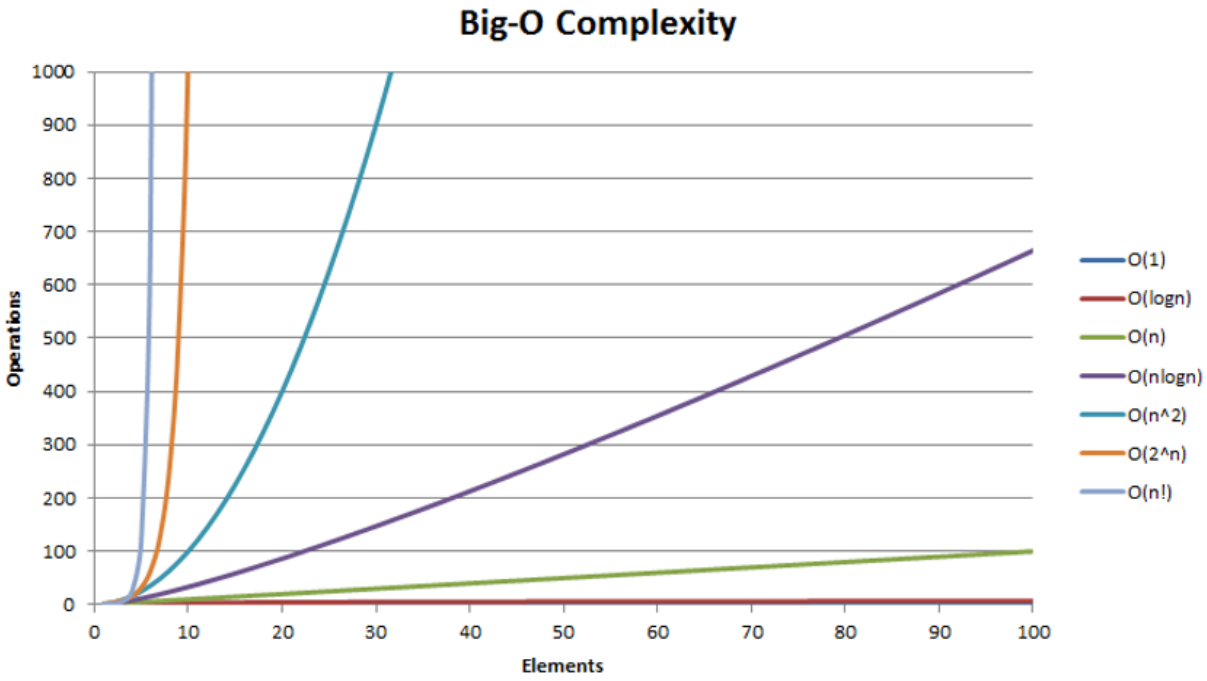
Na forma matemática, a complexidade de um algoritmo é representado por quantas operações um computador pode realizar por segundo. Assumindo que computadores comuns podem realizar mais de 1 milhão de operações por segundo, algoritmos simples são executados em milésimos. Porém, quando tal algoritmo possui muitos loops e laços de procura exaustiva, esse tempo cresce de forma exponencial, resultando em um acúmulo de tempo enorme, como pode ser representado por essa tabela:

Quantidade	100	1,000	10,000	100,000	1,000,000
N	< 1 seg	< 1 seg	< 1 seg	< 1 seg	1 seg
N^2	< 1 seg	1 seg	2 min	3 horas	12 dias
N^3	1 seg	18 min	12 dias	32 anos	31,710 anos
2^N	10^{17} anos	muito tempo	muito tempo	muito tempo	muito tempo
N!	muito tempo	muito tempo	muito tempo	muito tempo	muito tempo

muito tempo > 10^{25} anos

Operações como soma, divisão, declaração de variáveis e outras aumentam minimamente os valores de complexidade. Porém, loops de busca, além de terem uma duração imprevisível (podendo gerar o melhores e piores casos, onde o loop executa apenas uma ou centenas de vezes, respectivamente), aumentam a complexidade de forma exponencial. E nesses casos exponenciais, as operações menores podem ser desconsideradas, devido a seu impacto mínimo na complexidade final.

Função	Nome	Exemplos
1	constante	Somar dois números
log n	logarítmica	Pesquisa binária, inserir um número em uma heap
n	linear	1 ciclo para buscar o valor máximo em um vetor
n log n	linearítimica	Ordenação (merge sort, heap sort)
n^2	quadrática	2 ciclos (bubble sort, selection sort)
n^3	cúbica	3 ciclos (Floyd-Warshall)
2^n	exponencial	Pesquisa exaustiva (subconjuntos)
n!	fatorial	Todas as permutações



Sendo assim, para otimizar o código e reduzir o máximo possível a complexidade do mesmo, é necessário reduzir a quantidade de loops e buscas exaustivas, com o objetivo de produzir um código com complexidade (NlogN) ou no máximo N^2.

Código da Tarefa

Os dois códigos apresentados possuem estruturas parecidas, agindo como pequenos compiladores, procurando palavras reservadas específicas contidas na entrada, atráves de comparações com uma tabela pré-definida de palavras reservadas, e apresenta as palavras achadas e sua posição na tabela para o usuário:

O primeiro é uma simples busca de char por char, onde a cada leitura, é feita uma comparação com as palavras presentes na tabela. Caso não encontre, o loop é refeito, até que a entrada acabe. A complexidade estaria próxima de N^2 no pior caso.

O segundo é mais complexo na escrita, porém sua complexidade de execução é menor: ele ainda lê um char de cada vez, mas dessa vez apenas faz a comparação quando a palavra que está sendo lida termina. O código também é capaz de diferenciar char de números, e os apresenta de forma diferente na saída.

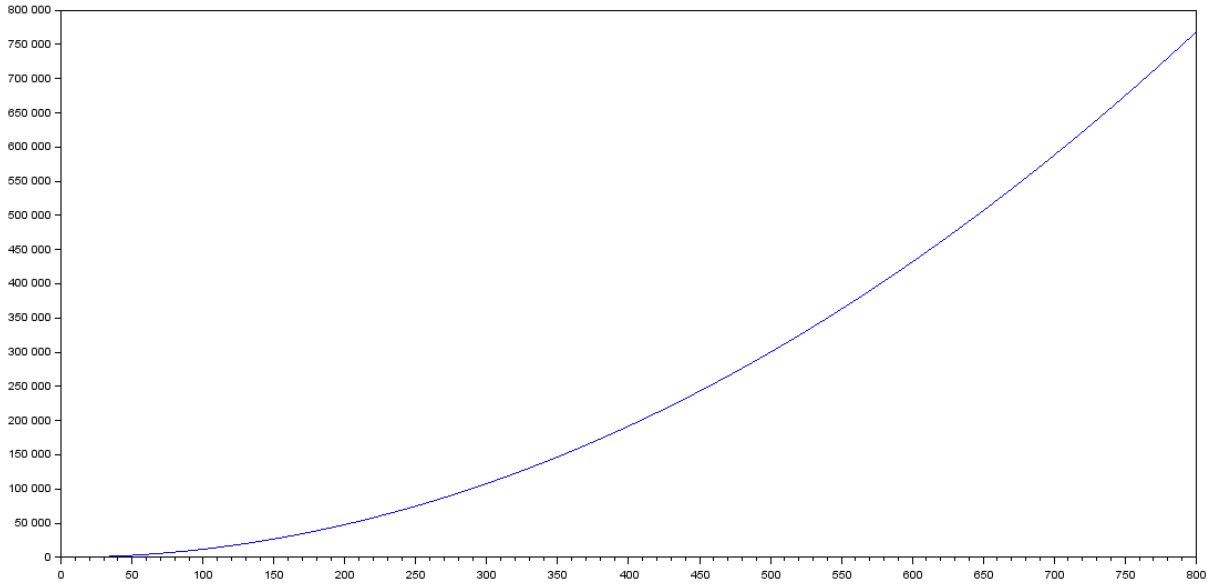
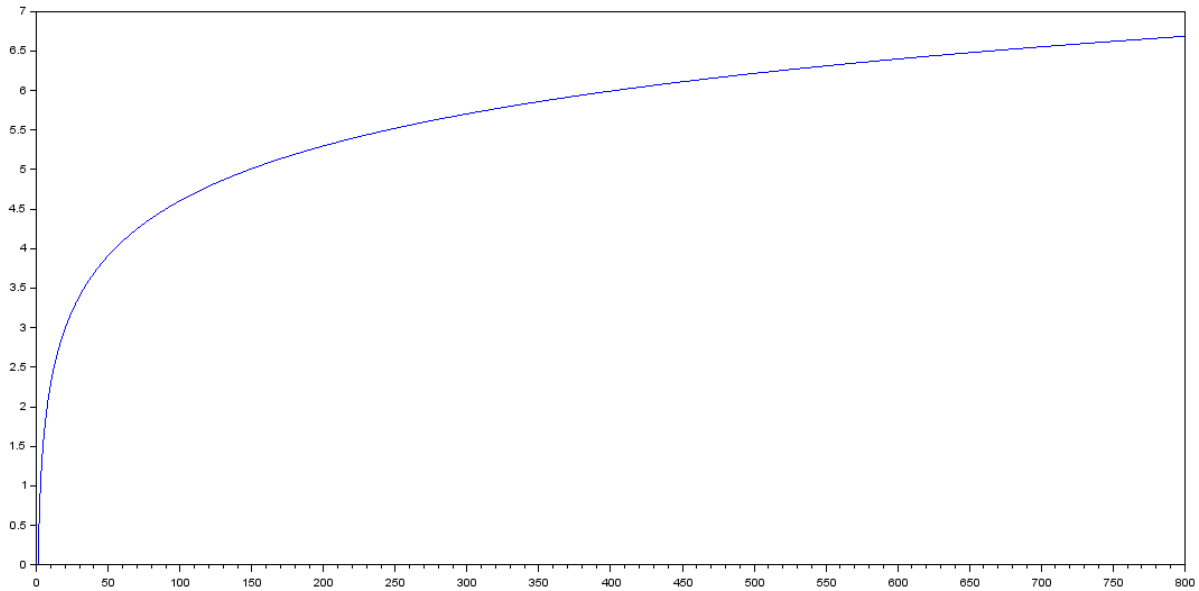


Gráfico de complexidade do primeiro código



Complexidade do segundo código

Como se pode observar, mesmo que o segundo código tenha uma complexidade maior em casos de entradas curtas e pequenas, a longo prazo é a melhor opção, por manter uma complexidade pouco variável em entradas grandes, diferente do primeiro código, com um aumento exponencial.

Conclusão

Pode-se concluir que para se evitar casos onde a execução dos códigos demore mais do que o necessário, deve-se otimizar o quanto for possível esse mesmo código, reduzindo a quantidade de loops e pesquisas exaustivas. Porém, em casos onde as entradas tem a garantia de serem simples e a execução das mesmas é consistentemente curta, códigos de “força bruta” ou “ingênuos” podem ser usados pela facilidade de implementação

Bibliografia

http://www.inf.puc-rio.br/~elima/paa/ (Aula 01 – Complexidade de Algoritmos, por Erdilei Soares de Lima)