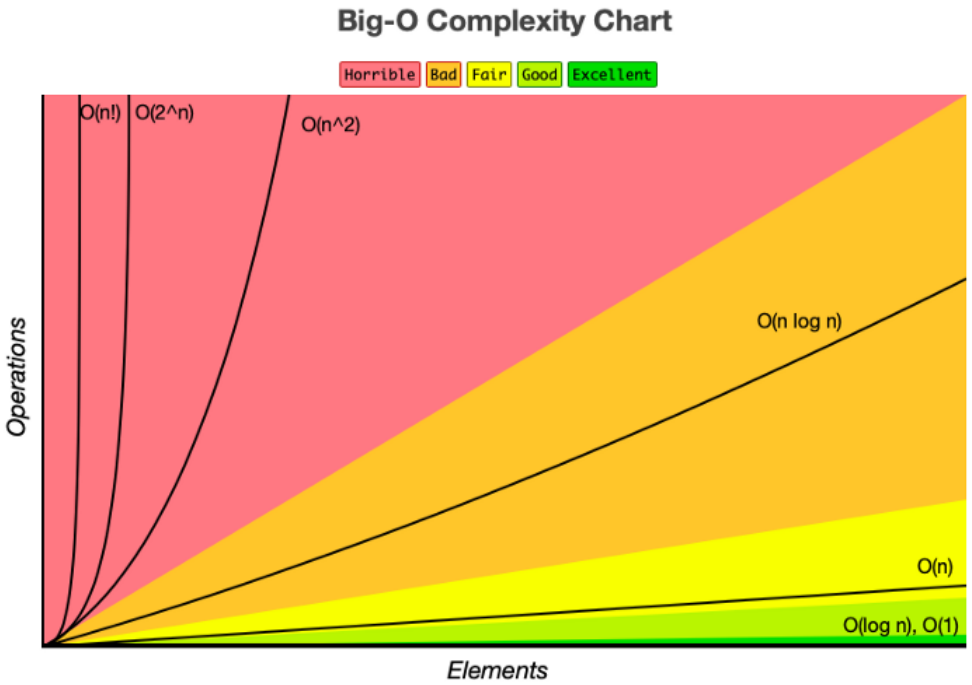
Trabalho prático 2 – Estrutura de Dados
Aluna: Raissa Gonçalves Diniz

Relatório de experimentos referentes ao desempenho de códigos recursivos e não recursivos

Informações iniciais: o seguinte relatório traz informações sobre os testes realizados por mim usando 3 valores (5, 20 e 40), para que uma análise mais aprofundada pudesse ser feita. Contudo, como especificado no Moodle, essa prática roda através do comando ``make run", compilando o executável e executando cada versão de cada algoritmo apenas com o parâmetro 5**.**

Introdução

Ao adentrarmos nos estudos das Estruturas de Dados, é crucial compreender a relação intrínseca entre as estruturas e algoritmos que moldam a eficiência e o desempenho de nossos programas. Assim como o exemplo do gráfico apresentado, a complexidade algorítmica desempenha um papel fundamental na determinação do tempo de resposta de um sistema. Assim, neste relatório, exploraremos como diferentes tamanhos de entrada para os mesmos cálculos, apenas com algoritmos diferentes, podem gerar discrepâncias no tempo de execução.



Para este experimento foram utilizadas quatro funções, sendo elas:
- O cálculo de Fibonacci de forma recursiva e iterativa;
- O cálculo de Fatorial de forma recursiva e iterativa.

Além disso, foram utilizados, respectivamente, os valores (5, 20, 40) para a análise de desempenhos de ambas as funções (tanto iterativa, quanto recursivamente).

Resultados gerais

	Fibonacci iterativo			Fibonacci recursivo		
	5	20	40	5	20	40
System time	0.000000000 s	0.000000000 s	0.000000000 s	0.000000000 s	0.000228167 s	3.396507978 s
User time	0.000001000 s	0.000001000 s	0.000001000 s	0.000001000 s	0.000228000 s	3.230289000 s
Clock time	0.000001400 s	0.000001500 s	0.000001700 s	0.000003100 s	0.000456800 s	6.627008300 s

	Fatorial iterativo			Fatorial recursivo		
	5	20	40	5	20	40
System time	0.000000000 s	0.000000000 s	0.000000000 s	0.000000000 s	0.000000000 s	0.000000954 s
User time	0.000001000 s	0.000001000 s	0.000001000 s	0.000001000 s	0.000001000 s	0.000001000 s
Clock time	0.000001500 s	0.000002500 s	0.000001600 s	0.000002500 s	000001900 s	0.000002300 s

Análise do Tempo de relógio

Primeiramente, a tabela acima traz o tempo de relógio, sistema e usuário, para ambas funções de Fibonacci (iterativa e recursiva), onde as colunas indicam as entradas na função e os valores de cada lacuna indicam o tempo em segundos que obtive como resposta. Informações pertinentes que auxiliam na análise desses dados:

- O algoritmo recursivo tem uma complexidade exponencial, O(2^n)
- O algoritmo iterativo tem uma complexidade de tempo linear, O(n)

É visível que, quando o número de entradas (n) aumenta, a função Fibonacci recursiva tende a demorar mais do que a função Fibonacci iterativa. Isso ocorre devido à natureza exponencial do crescimento do tempo de execução da função recursiva em relação a n.

A função Fibonacci recursiva calcula os números de Fibonacci chamando-se a si mesma duas vezes para calcular o próximo número na sequência. Isso resulta em uma árvore de chamadas recursivas, na qual muitos cálculos intermediários são repetidos várias vezes. Conforme n aumenta, o número de chamadas recursivas e os cálculos intermediários crescem exponencialmente, levando a um aumento significativo no tempo de execução. Esse crescimento exponencial é conhecido como "complexidade de tempo exponencial" e torna a função Fibonacci recursiva ineficiente para valores grandes de n. Percebe-se que quando n = 40, para a o Fibonacci recursivo, o tempo total gasto é milhares de vezes maior do que para n = 20.

Por outro lado, a função Fibonacci iterativa calcula os números de Fibonacci em um loop, armazenando os valores intermediários e atualizando-os em cada iteração. Essa abordagem não envolve chamadas recursivas e, portanto, não sofre com o mesmo crescimento exponencial no tempo de execução. O tempo de execução da função Fibonacci iterativa aumenta linearmente com n, tornando-a muito mais eficiente para valores grandes de n.

Analisando a mesma tabela, mas agora com valores de tempo para as funções Fatorial, percebemos apenas uma menor variação no tempo de relógio marcado. Estudando a complexidade desses algoritmos fica claro o motivo pelo qual isso acontece: a complexidade dos algoritmos de Fatorial, tanto iterativo quanto recursivo, é linear.

A função fatorial iterativa calcula o fatorial de um número usando um loop. A complexidade de tempo dessa função é linear, o que significa que o tempo de execução aumenta linearmente com o número de entradas (n). Para cada valor de n, a função executa um loop que multiplica os números de 1 a n juntos.

Já a função fatorial recursiva calcula o fatorial de um número chamando-se a si mesma com um valor menor até atingir o caso base. A complexidade de tempo dessa função também é linear, pois cada chamada recursiva contribui para o tempo de execução, mas a relação é linear.

A discrepância menor entre as versões iterativa e recursiva da função fatorial em comparação com a função de Fibonacci ocorre porque ambas têm uma complexidade de tempo linear (O(n)). Nas funções fatoriais, não há sobreposição significativa de chamadas recursivas, e o tempo de execução aumenta de forma linear e proporcional ao valor de n, independentemente da abordagem usada (iterativa ou recursiva).

Já que a função de Fibonacci apresenta um crescimento exponencial do tempo de execução (complexidade O(2^n)) à medida que n aumenta. Isso cria uma diferença mais acentuada entre as versões iterativa e recursiva da função de Fibonacci, em comparação a Fatorial.

Tempo de usuário e de sistema

Agora, iremos analisar os valores relacionados ao tempo de usuário (user Time) e de sistema (system Time) para as mesmas funções já citadas.

O tempo de usuário representa o período em que o processo está ativamente executando seu próprio código, ou seja, o tempo que ele gasta realizando as operações específicas que foram programadas como parte de seu funcionamento. Por outro lado, o tempo de sistema se refere ao intervalo durante o qual o sistema operacional assume o controle das operações do processo, realizando tarefas em nome dele, como gerenciamento de recursos. Juntos, o tempo de usuário e o tempo de sistema somam o tempo total de CPU que o processo consome para realizar suas operações.

O Fibonacci iterativo mantém uma execução constante e eficiente, independentemente do tamanho da entrada, tanto para o tempo de usuário quanto o de sistema. Por outro lado, a função Fibonacci recursiva revela um notável aumento no tempo de usuário à medida que a entrada cresce, evidenciando o impacto significativo da complexidade recursiva no tempo de execução.

De maneira semelhante, função fatorial iterativa também exibe uma performance estável e rápida, independentemente do tamanho da entrada. Já a função fatorial recursiva revela um leve aumento no tempo de usuário à medida que a entrada cresce, com um pequeno incremento no tempo de sistema para a entrada 40.

Em resumo, funções iterativas geralmente mantêm tempos de execução consistentes e rápidos, independentemente do tamanho da entrada. Em contrapartida, funções recursivas, especialmente a Fibonacci, exibem um notável aumento no tempo de usuário à medida que a entrada aumenta, devido à sua natureza que envolve múltiplas chamadas de função. No entanto, mesmo em funções recursivas, o aumento no tempo de usuário pode ser gerenciável para entradas menores, se tornando substancial para entradas muito grandes, como no caso em que n = 40.

Relatório do GPROF

A partir daqui incluiremos no relatório dados relativos ao desempenho das funções perante a ferramenta GPROF. O GNU profiler (gprof), faz parte do conjunto de ferramentas GNU Binary Utilities (binutils), e tem como principal funcionalidade apontar quanto tempo está sendo gasto em cada parte do seu programa.

% time	cumulative seconds	self seconds	calls	self ms/call	total ms/call	name
95.06		0.89	0.89	6	148.93	148.93	recursiveFibonacci
5.34	0.94	0.05	6	8.37	8.37	iterativeFibonacci
0.00	0.94	0.00	24	0.00	0.00	getSystemTime
0.00	0.94	0.00	24	0.00	0.00	getUserTime
0.00	0.94	0.00	6	0.00	0.00	iterativeFactorial
0.00	0.94	0.00	6	0.00	0.00	recursiveFactorial

O relatório do gprof mostra que a função recursiveFibonacci é responsável pela maior parte do tempo de execução do programa, consumindo 95,06% do tempo total. A função iterativeFibonacci também consome algum tempo, mas em menor quantidade (5,34%). As funções de medição de tempo, getSystemTime e getUserTime, não contribuem significativamente para o tempo total de execução. As funções iterativeFactorial e recursiveFactorial têm um impacto mínimo no tempo de execução.