Relatório - Atividade Prática II

Aluno: Bruno Lima Soares - 2022055785

Introdução

Quando adentramos na disciplina de Estrutura de Dados, um dos pontos principais com o qual trabalhamos é a Análise de Desempenho de um algoritmo, que busca metrificar como uma implementação se sai, em diferentes situações experimentais. A ideia é ver como se dá o consumo de memória, de tempo, além do quanto o sistema teve que alocar desempenho para executar certas tarefas. Todos esses fatores são extremamente importantes no âmbito do *TradeOff.* que nada mais é, de forma geral, que uma decisão entre custos e benefícios.

Pensando no desenvolvimento de um sistema, e principalmente na execução de algoritmos o mais eficientes possível, essa análise é de suma importância, tendo em vista justamente o desempenho e harmonia do código construído para que desempenhe um bom papel. Ademais, a ideia em grandes sistemas e empreendimentos, é de que a demanda cresça de forma exponencial, e por isso, quanto mais complexo vai ficando o desenvolvimento, mais planejamento no sentido de análises de desempenho são necessárias.

Pensando assim em colocar em prática esses conceitos, a ideia nessa atividade é implementar duas funções, porém de jeitos diferentes, sendo uma delas recursiva e a outra iterativa, para testar como o sistema se sai sob essas condições.

- **Fatorial:** *Nada mais é do que a multiplicação de um número, por todos os seus antecessores maiores que zero. A função será implementada de forma recursiva e iterativa.*

  Seu protótipo é

  `unsigned long long fatorial ( int n );`
- **Sequência de Fibonacci:** *Sequência na qual um número será sempre o resultado da soma de seus dois antecessores imediatos, maiores que zero. A função também será implementada de forma recursiva e iterativa.*

  Seu protótipo é

  `unsigned long long fibonacci ( int n );`

Fatorial

O fatorial, como explicado acima, é utilizado para retornar a multiplicação de um número por todos os seus antecessores, até o 0 (Fatorial não é calculado para números negativos). Isso significa que 5! é igual a 5 x 4 x 3 x 2 x 1. E é nesse ponto que diferenciar o recursivo do iterativo se torna essencial. No caso recursivo, temos uma condição de execução, na qual a função realizará chamadas a si mesma, até que alcance uma condição de parada, a partir da qual poderá “desempilhar” todas as chamadas executadas anteriormente. É como se a tarefa fosse quebrada em tarefas menores, até chegar em um caso trivial, que é o fatorial de Zero ou Um. A chamada iterativa, contudo, traz uma abordagem diferente, sem quebrar tarefas em tarefas menores, mas sim multiplicando gradualmente os números e armazenando-os. Assim, no caso do fatorial, a chamada recursiva pode ser menos eficiente, dado que empilha vários valores até chegar a um caso mínimo, para depois desempilhar, o que torna a execução muito demorada para valores altos. Em contrapartida, a execução iterativa do método é eficiente no uso de memória, já que ao invés de empilhar chamadas, acumula os valores e vai só multiplicando pelos números que vêm em seguida.

Fibonacci

A sequência de Fibonacci, como já foi explicada, retorna, em seu n-ésimo termo, a soma dos seus dois termos imediatamente antecessores, isto é,

termo n = termo (n - 1) + termo (n - 2)

Isso significa que Fibonacci(0) é 0, Fibonacci(1) é 1, Fibonacci(2) é 1, e por ai vai na lógica descrita acima. O caso recursivo da chamada de Fibonacci acaba sendo ainda mais contundente que o caso do fatorial, porque como já descrevemos, a função fará chamadas a si mesma, mas dessa vez vai empilhar dois valores pendentes, para serem resolvidos após atingir o caso mínimo, isto porque dependerá do primeiro antecessor e do segundo antecessor do número a calcular. Assim, a chamada recursiva pode ser muito ineficiente conforme o número de chamadas for aumentando, implicando em uma complexidade exponencial.Já na implementação iterativa, é aplicado um loop para calcular o Fibonacci, começando pelos casos extremos, 0 e 1. Depois, a função basicamente faz uma soma e uma troca de variáveis a cada iteração do loop, sempre **armazenando valores** já

calculados, o que representa um ganho de tempo e redução de complexidade da função, em relação à sua versão recursiva, quando se tratam de números muito grandes.

Estrutura de Testes

Para executar a análise de desempenho das funções acima explicitadas, utilizei a seguinte faixa de valores: { 5, 15, 25, 35, 40 }. Eles foram testados em tempo de execução de recursos de sistema, de usuário, além de tempo de clock.

Fatorial Recursivo

	Tempo de Sistema	Tempo de Usuário	Tempo de Clock
5	0	0.000001	0.0063
15	0	0.000001	0.0057
25	0	0.000001	0.0074
35	0	0.000002	0.0096
40	0	0.000002	0.0074

Fatorial Iterativo

	Tempo de Sistema	Tempo de Usuário	Tempo de Clock
5	0	0	0.0054
15	0	0.000001	0.004
25	0	0.000001	0.0038
35	0	0	0.006
40	0	0.000001	0.0047

Fibonacci Recursivo

	Tempo de Sistema	Tempo de Usuário	Tempo de Clock
5	0	0.000001	0.0058
15	0	0.000035	0.0762
25	0	0.004836	9.0827
35	0.007277	0.495013	10.17
40	0	0.631495	11.14

Fibonacci Iterativo

	Tempo de Sistema	Tempo de Usuário	Tempo de Clock
5	0	0	0.0039
15	0	0.000001	0.007
25	0	0.000001	0.0075
35	0	0.000002	0.0061
40	0	0.000001	0.0054

Conclusão

Percebemos através dos dados levantados nesse experimento que a Sequência de Fibonacci, quando calculada de forma iterativa, é executada de maneira mais eficiente do que sua abordagem recursiva. Esse resultado ressalta a importância de pensar bem em um modelo de implementação para seguir que seja mais interessante, e não utilizar apenas a recursividade de maneira cega.

Já na implementação do algoritmo para cálculo de fatorial, percebemos que o viés recursivo lida bem com a sobrecarga da função até determinados valores, mas com o passar do tempo e o crescimento exponencial dos cálculos e dos valores

empilhados, a função perde muito rendimento e a iterativa passa a fazer ainda mais sentido.

Relatório obtido através do GPROF

```
You, 9 seconds ago | 1 author (You)
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  ms/call  ms/call  name
 93.57      1.69     1.69        2   846.85   846.85  fibonacciRecursivo(int)
  6.64      1.81     0.12        1   120.26   120.26  _GLOBAL__sub_I__Z17fatorialRecursivoi
  0.00      1.81     0.00        4     0.00     0.00  systemTime()
  0.00      1.81     0.00        4     0.00     0.00  userTime()
  0.00      1.81     0.00        2     0.00     0.00  fibonacciIterativo(int)
  0.00      1.81     0.00        2     0.00     0.00  clockTime(timespec, timespec)
  0.00      1.81     0.00        1     0.00     0.00  _GLOBAL__sub_I__Z18fibonacciRecursivoi
  0.00      1.81     0.00        1     0.00     0.00  _GLOBAL__sub_I__Z8userTimev
  0.00      1.81     0.00        1     0.00     0.00  _GLOBAL__sub_I_numeroEscolhido
  0.00      1.81     0.00        1     0.00     0.00  parse_args(int, char**)
  0.00      1.81     0.00        1     0.00     0.00  __static_initialization_and_destruction_0(int, int)
  0.00      1.81     0.00        1     0.00     0.00  __static_initialization_and_destruction_0(int, int)
  0.00      1.81     0.00        1     0.00     0.00  __static_initialization_and_destruction_0(int, int)
  0.00      1.81     0.00        1     0.00     0.00  __static_initialization_and_destruction_0(int, int)
```

```
granularity: each sample hit covers 2 byte(s) for 0.55% of 1.81 seconds

index % time    self  children    called     name
                                                 <spontaneous>
[1]     93.4    0.00    1.69                 main [1]
                1.69    0.00       2/2           fibonacciRecursivo(int) [2]      You, 14 seconds ago
                0.00    0.00       4/4           systemTime() [11]
                0.00    0.00       4/4           userTime() [12]
                0.00    0.00       2/2           clockTime(timespec, timespec) [14]
                0.00    0.00       2/2           fibonacciIterativo(int) [13]
                0.00    0.00       1/1           parse_args(int, char**) [18]
-----------------------------------------------
                                409336616           fibonacciRecursivo(int) [2]
                1.69    0.00       2/2           main [1]
[2]     93.4    1.69    0.00       2+409336616 fibonacciRecursivo(int) [2]
                                409336616           fibonacciRecursivo(int) [2]
-----------------------------------------------
```

A ferramenta gprof trouxe ao longo dessa pesquisa e teste de desempenho uma série de dados e insigths valiosos com relação às diferentes implementações das funções recursiva e iterativa de Fatorial e Fibonacci.

Por meio do relatório é possível perceber o quanto a função recursiva do Fibonacci consumiu de tempo e o quanto ela representou em desgaste para o sistema, com quase 94% de tempo de execução, isso para uma mesma faixa de valores em relação às outras funções. Isso representa o que falei acima, que a tendência para a recursividade, em Fibonacci, é ser cada vez mais complexa, já que o fato de empilhar duas chamadas por vez torna o aumento de complexidade da função exponencial. Ademais, esse cenário representa a importância de análises e do uso de ferramentas para testar o desempenho de algoritmos sempre que possível, já

que é inaceitável ter uma, ou até mais funções com esse nível de complexidade e uso de recursos, em um sistema de alta complexidade e escalabilidade, principalmente quando se fala em desempenho e rapidez, valores que o mercado tanto procura, determinantes para uma solução / algoritmo de impacto.