UNIVERSIDADE FEDERAL DE MINAS GERAIS (UFMG)
DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO

NICOLAS VON DOLINGER MOREIRA ROCHA (2022035571)

TRABALHO PRÁTICO 0
GERAÇÃO DA SEQUÊNCIA DE FIBONACCI

BELO HORIZONTE
2024

TRABALHO PRÁTICO 0: GERAÇÃO DA SEQUÊNCIA DE FIBONACCI

Nicolas von Dolinger Moreira Rocha[1]

1. Introdução

Este código aborda a questão do cálculo do n-ésimo número da sequência de Fibonacci, um problema clássico em ciência da computação e matemática. A sequência de Fibonacci é uma sequência de números em que cada número é a soma dos dois números anteriores, começando com 0 e 1.

O objetivo é implementar e comparar duas abordagens diferentes para calcular o n-ésimo número de Fibonacci: uma recursiva e outra iterativa. A abordagem recursiva é implementada na função “recursiveFib”, enquanto a abordagem iterativa é implementada na função “iterativeFib”.

Para medir o desempenho de ambas as abordagens, o código utiliza uma estratégia de limitação de tempo. Ele executa repetidamente os cálculos até que um limite de tempo pré-definido seja atingido. Em seguida, compara o número de termos calculados e o valor do último número de Fibonacci alcançado usando cada método dentro desse intervalo de tempo.

Ao comparar as duas abordagens, analisaremos não apenas a precisão dos resultados, mas também o tempo de execução necessário para calcular os números de Fibonacci até um determinado ponto. Isso nos permitirá entender melhor as vantagens e desvantagens de cada método e sua eficiência em termos de tempo de execução.

2. Metodologia

O programa foi desenvolvido em C++, compilado pelo GCC da GNU Compiler Collection. A máquina utilizada tem as configurações descritas abaixo.

2.1. Configurações da Máquina

Sistema operacional: Ubuntu 23.04
Compilador: G++/GNU Compiler Collection

---

[1] Graduando em Sistemas de Informação pela Universidade Federal de Minas Gerais (UFMG). Product Owner na iJunior, empresa júnior do Departamento de Ciência da Computação da UFMG. Monitor de Programação e Desenvolvimento de Software II.

Processador: AMD Ryzen 5-5500U
Memória RAM: 16 GB

2.2. Abordagem

Para implementar o cálculo do n-ésimo número de Fibonacci, foram adotadas duas abordagens diferentes: uma recursiva e outra iterativa. Abaixo estão os detalhes de cada método:

2.2.1. Abordagem Recursiva (Função recursiveFib):

Esta abordagem utiliza recursão para calcular o n-ésimo número de Fibonacci. O algoritmo baseia-se na definição recursiva da sequência de Fibonacci, onde cada número é a soma dos dois números anteriores.

O critério de parada é quando o índice n é menor ou igual a 1, nesse caso, o valor de n é retornado diretamente. Caso contrário, a função chama a si mesma recursivamente para calcular os números de Fibonacci para n-1 e n-2, e retorna a soma desses valores.

Essa abordagem é simples de implementar, mas pode ser ineficiente devido à quantidade de repetições de cálculos. Sua complexidade de tempo é exponencial, O(2^n), devido às chamadas recursivas repetidas.

2.2.2. Abordagem Iterativa (Função iterativeFib):

Nesta implementação iterativa, os números de Fibonacci são calculados de maneira eficiente, evitando a sobrecarga de chamadas de função típica da abordagem recursiva. Aqui está o que acontece:

O código começa declarando duas variáveis, f1 e f2, para representar os dois primeiros números de Fibonacci. Um loop é então iniciado em uma thread separada (iterative), que atualiza esses valores repetidamente até que a variável valid se torne falsa.

Dentro desse loop, o algoritmo calcula o próximo número de Fibonacci somando f1 e f2 e armazenando o resultado em f2. Antes disso, o valor de f1 é atribuído à variável aux para que ele possa ser usado na atualização de f1. Isso garante que os valores sejam atualizados

corretamente, pois primeiro f1 é substituído pelo valor anterior de f2, e depois f2 é atualizado para o próximo número de Fibonacci.

Enquanto isso, o loop principal no main() imprime o tempo decorrido após aguardar um certo número de segundos especificado no vetor seconds, que contém os intervalos de tempo para os quais os cálculos serão medidos. Após cada intervalo de tempo, o número atual de Fibonacci (representado pela variável n) é impresso.

Finalmente, quando todos os cálculos necessários são concluídos, a variável valid é definida como false para encerrar a thread iterative.

3. Testes

Para verificar a corretude e eficiência dos algoritmos de cálculo do n-ésimo número de Fibonacci, foram realizados os seguintes testes:

3.1. Corretude dos Resultados

Foram realizados testes simples para garantir que os algoritmos retornam os resultados corretos para valores pequenos de n (por exemplo, n = 0, n = 1, n = 2, etc.). Os resultados obtidos foram comparados com os valores conhecidos da sequência de Fibonacci para confirmar a precisão dos algoritmos.

3.2. Análise de Desempenho

Para avaliar a eficiência dos algoritmos, foram executados testes de desempenho com diferentes valores de n. Os tempos de execução de cada algoritmo foram medidos usando um conjunto de dados de entrada que varia de pequenos valores de n até valores maiores. O desempenho foi avaliado comparando o tempo de execução dos algoritmos para diferentes valores de n. O código utiliza uma estratégia de limitação de tempo para medir o desempenho, executando os cálculos dentro de um limite de tempo pré-definido e comparando a quantidade de termos calculados e o valor do último número de Fibonacci alcançado em cada método dentro desse intervalo de tempo.

3.3. Comparação de Eficiência

Os tempos de execução dos algoritmos foram comparados para determinar qual abordagem é mais eficiente em termos de tempo para calcular números de Fibonacci até um determinado ponto. Foram examinados os padrões de desempenho em relação ao aumento do valor de n, identificando em qual ponto cada algoritmo começa a mostrar uma degradação significativa no desempenho.

Através desses testes, podemos validar tanto a corretude dos algoritmos quanto sua eficiência em diferentes cenários de entrada. Essas análises nos permitem compreender melhor as características de desempenho de cada abordagem e tomar decisões informadas sobre a escolha do método mais adequado para uma aplicação específica.

4. Discussão de Resultados

A análise comparativa dos resultados dos testes revela insights significativos sobre o desempenho dos algoritmos de cálculo do n-ésimo número de Fibonacci. Aqui estão algumas observações-chave:

4.1. Corretude dos Resultados

Ambos os algoritmos foram validados com sucesso em termos de corretude dos resultados para valores pequenos de n. Os resultados obtidos para os primeiros números de Fibonacci foram consistentes com os valores conhecidos da sequência, confirmando a precisão dos algoritmos.

4.2. Análise de Desempenho

```
--------- ITERATIVO ---------

Tempo decorrido: 15.0001 segundos
Quantidade de números calculados em 15 segundos: 4677698388
Tempo decorrido: 15.0001 segundos
Quantidade de números calculados em 30 segundos: 9271780219
Tempo decorrido: 15.0001 segundos
Quantidade de números calculados em 45 segundos: 13951360694
Tempo decorrido: 15.0001 segundos
Quantidade de números calculados em 60 segundos: 18648097781
Tempo decorrido: 15.0001 segundos
Quantidade de números calculados em 75 segundos: 23312757275
Tempo decorrido: 15.0001 segundos
Quantidade de números calculados em 90 segundos: 27974609042
Tempo decorrido: 15.0001 segundos
Quantidade de números calculados em 105 segundos: 32647014969
Tempo decorrido: 15.0001 segundos
Quantidade de números calculados em 120 segundos: 37160507818
```

```
--------- RECURSIVO ---------

Tempo decorrido: 15.0001 segundos
Quantidade de números calculados em 15 segundos: 42
Tempo decorrido: 15.0001 segundos
Quantidade de números calculados em 30 segundos: 43
Tempo decorrido: 15.0001 segundos
Quantidade de números calculados em 45 segundos: 44
Tempo decorrido: 15.0001 segundos
Quantidade de números calculados em 60 segundos: 45
Tempo decorrido: 15.0001 segundos
Quantidade de números calculados em 75 segundos: 45
Tempo decorrido: 15.0001 segundos
Quantidade de números calculados em 90 segundos: 46
Tempo decorrido: 15.0004 segundos
Quantidade de números calculados em 105 segundos: 46
Tempo decorrido: 15.0001 segundos
Quantidade de números calculados em 120 segundos: 46
```

Os testes de desempenho revelaram uma diferença significativa entre os tempos de execução dos algoritmos. A abordagem iterativa mostrou-se consistentemente mais eficiente em termos de tempo de execução em comparação com a abordagem recursiva. À medida que o valor de n aumenta, a diferença de desempenho entre os algoritmos se torna mais evidente. A abordagem recursiva exibe um crescimento exponencial no tempo de execução, enquanto a abordagem iterativa mantém um crescimento linear.

4.3. Impactos da Utilização

A escolha entre os algoritmos tem um impacto significativo na eficiência e na escalabilidade de um sistema. Para valores pequenos de n, a diferença de desempenho pode ser menos perceptível. No entanto, conforme n aumenta, a abordagem iterativa se torna cada vez mais vantajosa em termos de tempo de execução. Em situações onde o cálculo de números de Fibonacci é realizado repetidamente ou para valores grandes de n, a escolha da abordagem iterativa pode resultar em economia significativa de recursos computacionais e tempo de execução.

Em resumo, os resultados dos testes destacam a importância de selecionar o algoritmo adequado com base nos requisitos específicos de desempenho e escalabilidade de uma aplicação. Enquanto a abordagem recursiva pode ser mais simples de implementar, a abordagem iterativa oferece um desempenho superior, especialmente para problemas que exigem o cálculo de números de Fibonacci para valores grandes de n.

5. Conclusão

Este exercício proporcionou uma análise abrangente dos algoritmos de cálculo do n-ésimo número de Fibonacci, comparando uma abordagem recursiva com uma abordagem iterativa. As principais conclusões são as seguintes:

i. A abordagem iterativa mostrou-se significativamente mais eficiente em termos de tempo de execução em comparação com a abordagem recursiva. Enquanto a abordagem recursiva apresenta um crescimento exponencial no tempo de execução, a abordagem iterativa mantém um crescimento linear, tornando-se mais vantajosa para valores maiores de n.

ii. Este exercício proporcionou uma compreensão mais profunda das diferenças de desempenho entre algoritmos recursivos e iterativos. Além disso, destacou a importância de considerar o tempo de execução e a escalabilidade ao escolher um algoritmo para resolver um problema específico.

Em resumo, este exercício serviu como uma oportunidade valiosa para explorar e comparar diferentes abordagens para resolver um problema clássico de computação. As conclusões obtidas podem orientar a seleção de algoritmos adequados para aplicações específicas e destacam a importância da eficiência algorítmica na otimização de sistemas computacionais.