UFRR
UNIVERSIDADE FEDERAL DE RORAIMA
DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO
ANÁLISE DE ALGORITMOS - DCC606

ALGORITMO

FIBONACCI

MARCIA GABRIELLE BONIFÁCIO DE OLIVEIRA - 2020011319
PAULO FERREIRA DA SILVA JÚNIOR - 2019034400

SEQUÊNCIA DE FIBONACCI

CONCEITO E DEFINIÇÃO

Ela recebe seu nome do matemático italiano Leonardo de Pisa, mais conhecido como Fibonacci, que a introduziu no mundo ocidental em seu livro Liber Abaci no início do século XIII.

Conceito:
A sequência de Fibonacci é uma série de números onde cada número é a soma dos dois anteriores. Inicia-se com 0 e 1, e é definida pela fórmula:

$$F(n)=F(n-1)+F(n-2)$$

para $n>1$, com $F(0)=0$ e $F(1)=1$.

C fibonacci.c

```c
#include <stdio.h>
#include <gmp.h>
#include <time.h>

int main() {
    FILE *fp = fopen("fibonacci_times.csv", "w");  // Arquivo para salvar os resultados
    fprintf(fp, "Terms,ExecutionTime\n");

    int terms[] = {100000, 500000, 1000000, 1500000, 2000000};  // Tamanhos de entrada específicos
    int numTests = 5;  // Número de entradas

    for (int test = 0; test < numTests; test++) {
        int n = terms[test];
        mpz_t t1, t2, nextTerm, sum;
        mpz_init(t1);
        mpz_init(t2);
        mpz_init(nextTerm);
        mpz_init(sum);
        mpz_set_ui(t1, 0);
        mpz_set_ui(t2, 1);
        mpz_add(nextTerm, t1, t2);
        mpz_add(sum, t1, t2);

        clock_t start, end;
        double cpu_time_used;

        start = clock();

        for (int i = 3; i <= n; ++i) {
            mpz_add(sum, sum, nextTerm);
            mpz_set(t1, t2);
            mpz_set(t2, nextTerm);
            mpz_add(nextTerm, t1, t2);
        }

        end = clock();
        cpu_time_used = ((double)(end - start)) / CLOCKS_PER_SEC;

        fprintf(fp, "%d,%f\n", n, cpu_time_used);  // Salva os resultados no arquivo

        // Limpeza das variáveis GMP
        mpz_clear(t1);
        mpz_clear(t2);
        mpz_clear(nextTerm);
        mpz_clear(sum);
    }

    fclose(fp);
    return 0;
}
```

ALGORITMO ITERATIVO:

Calcula a sequência de maneira linear, utilizando um loop para acumular os valores dos termos anteriores até o termo desejado.

FUNÇÃO DE CUSTO

$T(n)=4n-4$

COMPLEXIDADE

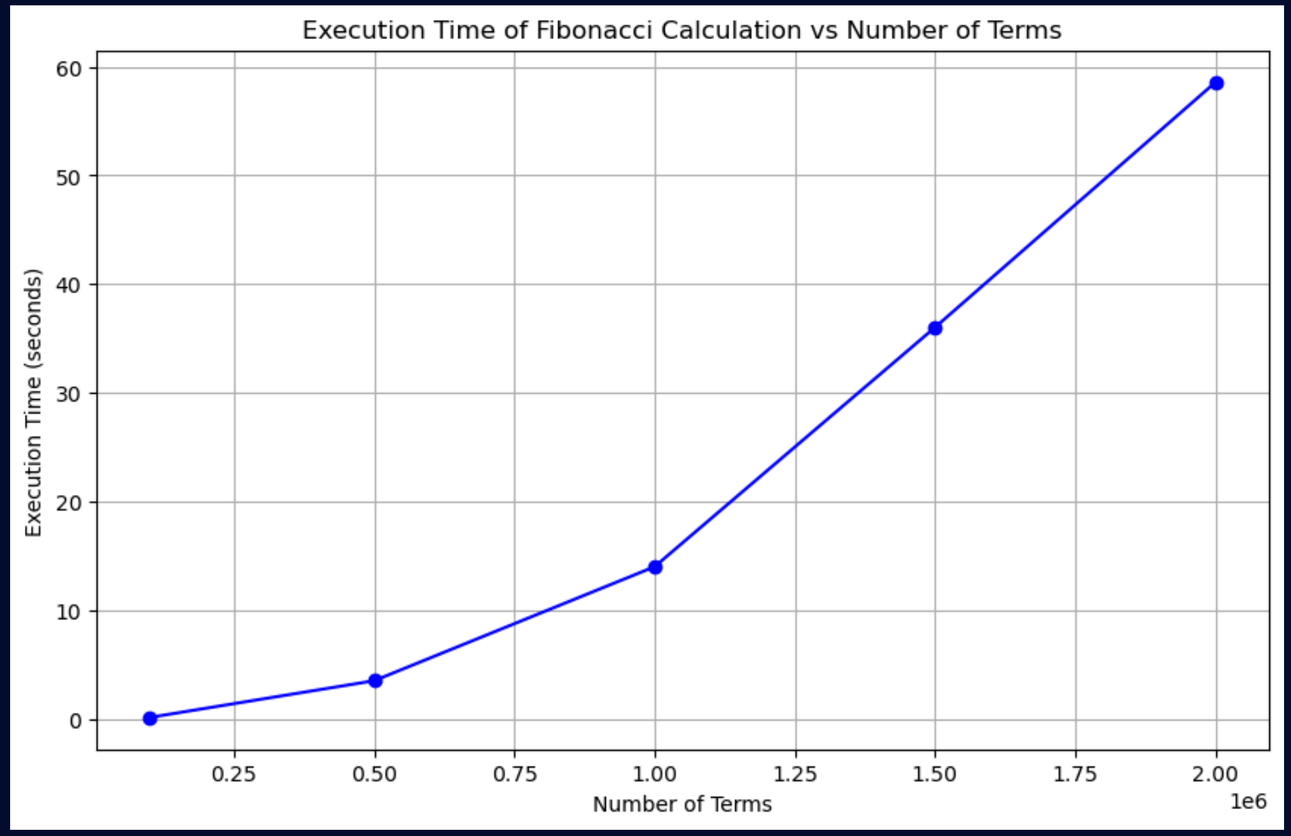
Tem complexidade de tempo O(n), tornando-o mais eficiente que o recursivo.

EXECUÇÃO DO ALGORITMO INTERATIVO

Utilizamos 5 diferentes entradas

fibonacci_times_interativo.csv > data

```
1   Terms,ExecutionTime
2   100000,0.172000
3   500000,3.546000
4   1000000,14.032000
5   1500000,36.042000
6   2000000,58.583000
7
```



EXPONENCIAÇÃO DE MATRIZES:

Uma abordagem mais avançada que utiliza multiplicação de matrizes para calcular a sequência em O(logn) tempo. Esta técnica é ideal para grandes valores de n, pois reduz significativamente o número de operações necessárias.

FUNÇÃO DE CUSTO

$T(n)=c1+c2 \cdot \log(n)$

c1 = Representa o custo das operações de inicialização, que é constante e independente de n.

c2 = Representa o custo de cada multiplicação de matrizes, que é O(1), multiplicado pelo número de multiplicações de matrizes necessárias, que é proporcional a log(n) devido à exponenciação rápida.

COMPLEXIDADE

O(logn)

fibonacci_melhorado.c

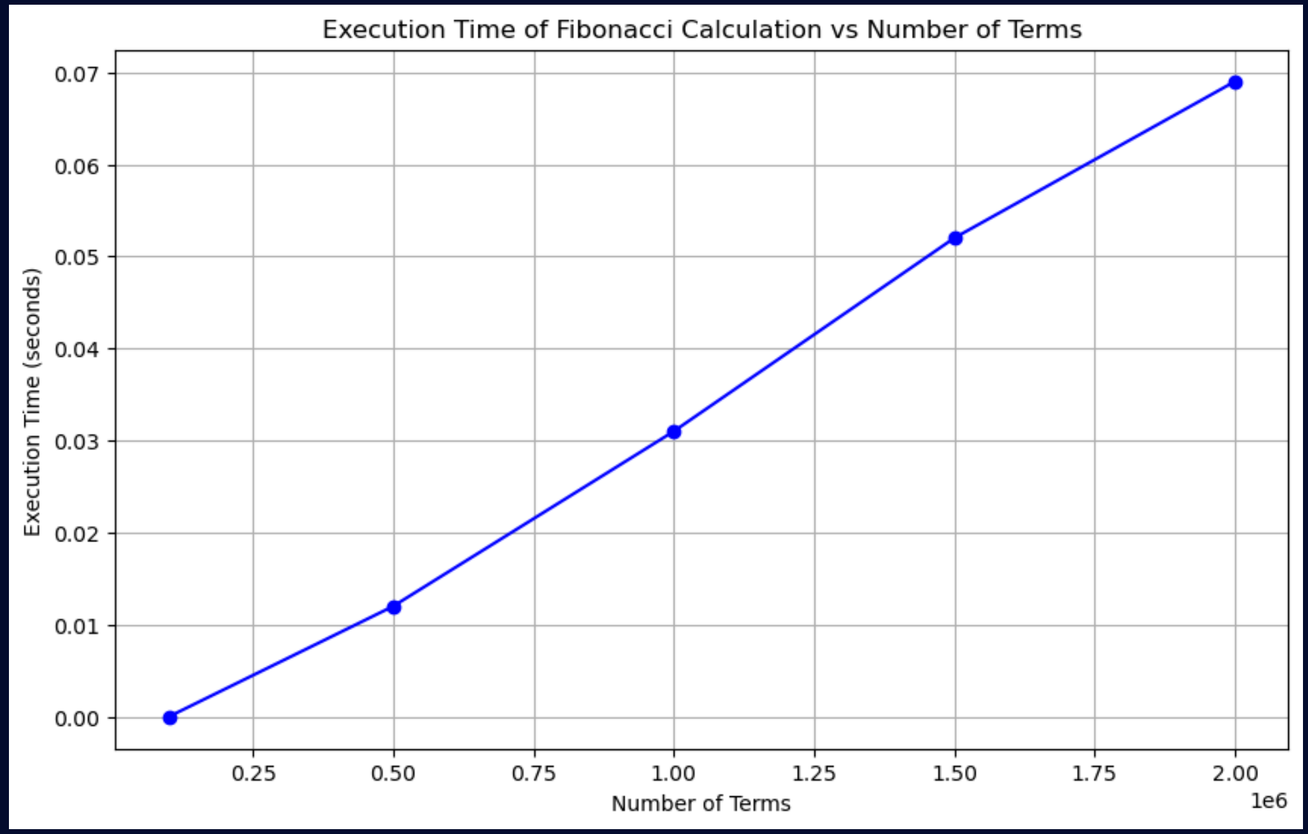
```c
#include <stdio.h>
#include <gmp.h>
#include <time.h>

void multiply_matrices(mpz_t F[2][2], mpz_t M[2][2]) {
    mpz_t a, b, c, d;
    mpz_init(a); mpz_init(b); mpz_init(c); mpz_init(d);

    mpz_mul(a, F[0][0], M[0][0]);
    mpz_addmul(a, F[0][1], M[1][0]);

    mpz_mul(b, F[0][0], M[0][1]);
    mpz_addmul(b, F[0][1], M[1][1]);

    mpz_mul(c, F[1][0], M[0][0]);
    mpz_addmul(c, F[1][1], M[1][0]);

    mpz_mul(d, F[1][0], M[0][1]);
    mpz_addmul(d, F[1][1], M[1][1]);

    mpz_set(F[0][0], a);
    mpz_set(F[0][1], b);
    mpz_set(F[1][0], c);
    mpz_set(F[1][1], d);

    mpz_clear(a); mpz_clear(b); mpz_clear(c); mpz_clear(d);
}

void power_matrix(mpz_t F[2][2], int n) {
    if (n == 0 || n == 1)
        return;

    mpz_t M[2][2];
    mpz_init(M[0][0]); mpz_init(M[0][1]);
    mpz_init(M[1][0]); mpz_init(M[1][1]);

    mpz_set_ui(M[0][0], 1); mpz_set_ui(M[0][1], 1);
    mpz_set_ui(M[1][0], 1); mpz_set_ui(M[1][1], 0);

    power_matrix(F, n / 2);
    multiply_matrices(F, F);

    if (n % 2 != 0)
        multiply_matrices(F, M);

    mpz_clear(M[0][0]); mpz_clear(M[0][1]);
    mpz_clear(M[1][0]); mpz_clear(M[1][1]);
}
```

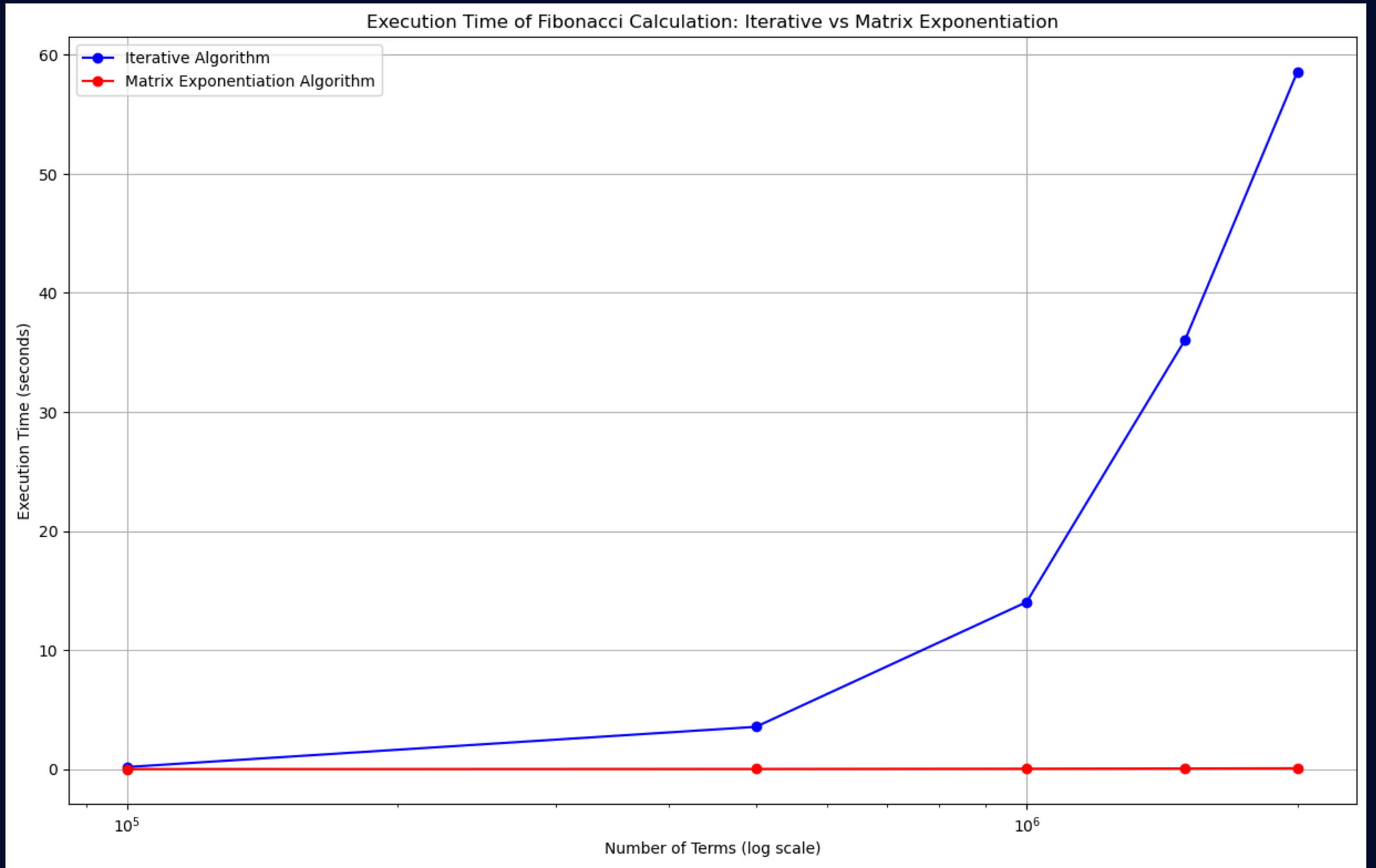
EXECUÇÃO DO ALGORITMO COM MATRIZES

Utilizamos 5 diferentes entradas

fibonacci_times_matrix.csv > data

```
1    Terms,ExecutionTime
2    100000,0.000000
3    500000,0.012000
4    1000000,0.031000
5    1500000,0.052000
6    2000000,0.069000
7
```




Execution Time of Fibonacci Calculation: Iterative vs Matrix Exponentiation
Iterative Algorithm
Matrix Exponentiation Algorithm
Execution Time (seconds)
0
10
20
30
40
50
60
10^5
10^6
Number of Terms (log scale)

OBSERVAÇÕES

- Os arquivos desse projeto está em um repositório do Github: https://github.com/juniorrkcm/Paulo_Marcia_AA_RR_2024

- Os arquivos csv de tempo de execução e os valores Fibonacci de cada entrada, são gerados a partir da execução do código.

OBRIGADO(A)!