Universidade Federal
de São João del-Rei

UNIVERSIDADE FEDERAL DE SÃO JOÃO DEL-REI

Conceitos de linguagem de programação

Implementação da sequencia de Fibonacci em tipos diferentes de linguagem

Prof.: Dr. DARLINGTON B. F. CARVALHO

Matheus N. Silva

São João del-Rei, 25 de outubro de 2022

Sumário

1 INTRODUÇÃO

1.1 Objetivo

O objetivo deste trabalho é desenvolver soluções para o cálculo do enésimo termo desta série utilizando 3 (três) linguagens de paradigmas de programação diferentes, sendo Python uma destas linguagem. Os resultados deverão ser apresentados na forma de um relatório.

2 MÉTODOS

Para implementar a sequencia de Fibonacci cada algoritmo foi pensado de forma recursiva, pois a criação do código se mostrou mais simples tanto de desenvolvimento quanto de entendimento. abaixo serão apresentados cada um dos códigos utilizados.

Nota: *Tanto no algoritmo em C quanto no em Python foram utilizados métodos de aferição de tempo de execução fornecidas pela linguagem*

2.1 Linguagens utilizadas

Foi proposto a utilização de três linguagens diferentes, sendo uma delas obrigatória (Python), alem de Python foi deixado como escolha livre as outras duas, portanto as outras linguagens escolhidas para a elaboração desse desafio foram C e Prolog.

2.1.1 Linguagem C

Abaixo podemos notar a implementação do método para o calculo da sequencia de Fibonacci em C. O método recebe um inteiro i (que o numero a ser calculado). Após é comparado com a condição minima de retorno e caso essa condição seja aceita o método retorna 1 para ser somado às outras chamadas (caso existam), caso contrario o método chama a si mesmo duas vezes passando como parâmetro em uma i-1 e i-2 na outra.

```c
#include "fibonacci.h"

int fibonacci(int i){
    if(i == 1 || i == 2){
        return 1;
    } else{
        return fibonacci(i-1) + fibonacci(i-2);
    }
}
```

2.1.2 Python

Assim como em C o método recebe um inteiro i para o calculo da sequencia de Fibonacci, sendo comparado com a condição minima de parada, caso essa condição seja aceita o método retorna 1 para ser somado às outras chamadas (caso existam), caso contrario o método chama a si mesmo, novamente duas vezes, passando como parâmetro i-1 em uma e i-2 na outra.

```python
def fibonacci(i):
    if i == 1 or i == 2:
        return 1
    else:
        return fibonacci(i-1) + fibonacci(i-2)
```

2.1.3 Prolog

Na implementação no prolog foi utilizado o método de recursão em cauda a fim de aumentar a velocidade de execução. Nota-se que o algoritmo recebe uma lista como predicado e realiza a chamada do método em que o calculo será executado.

```
fib(0,[0]).                    % <- base case 1
fib(1,[0,1]).                  % <- base case 2
fib(N,Seq) :-
   N > 1,
   fib(N,SeqR,1,[1,0]),       % <- actual relation (all other cases)
   reverse(SeqR,Seq).              % <- reverse/2 from library(lists)

fib(N,Seq,N,Seq).
fib(N,Seq,N0,[B,A|Fs]) :-
   N > N0,
   N1 is N0+1,
   C is A+B,
   fib(N,Seq,N1,[C,B,A|Fs]). % <- recursão em calda
```

3 ANÁLISE

Após alguns testes foi possivel notar que quanto maior a entrada, nos algoritmos desenvolvidos em C e em Python, maior o tempo para a execução das operações, tornando-os inviáveis para entradas relativamente grandes.

O algoritmo em Python foi o que demandou maior tempo de execução entre todos os algoritmos testados, sendo o que obteve o pior resultado na comparação

Ja o algoritmo desenvolvido em Prolog tende a ter um crescimento de tempo de execução menor quando comparado com os outros algoritmos acima mencionados.

3.1 Apresntação dos resultados

Entrada	C	Pyton	Prolog
10 elementos	0.000	0.001	0.000
30 elementos	0.055	3.380	0.000
50 elementos	131.949	NPC	0.000
50000 elementos	NPC	NPC	0.301

Tabela 1: comparação entre algoritmos

* TEMPOS EM SEGUNDOS

* NPC - NÃO FOI POSSIVEL CALCULAR

4 CONCLUSÃO

Após uma bateria de testes, que não se limitaram aos presentes na tabela 1, foi possivel notar que a forma em que a linguagem é implementada se torna bastante importante para determinadas tarefas.

o algoritmo em Python, embora utilizasse uma linguagem atual, se mostrou o mais custoso em termos de tempo de processamento à medida que a numero de elementos na entrada aumentava.

O C por sua vez ficou no meio termo, por mais que consumisse mais tempo de processamento que o algoritmo desenvolvido em prolog ele se mostrou mais eficiente que o algoritmo em Python.

Já o prolog por sua vez demonstrou ser o mais eficiente, tanto para entradas menores quanto para entradas relativamente maiores.