

## Relatório - Aula 2

Giovanna Naves Ribeiro  
2022043647

Começamos o processo criando as pastas e escrevendo os códigos para Fibonacci iterativo, Fibonacci recursivo, fatorial iterativo e fatorial recursivo nos arquivos fib.c e fat.c. O arquivo fatop.c executa as operações recursiva e iterativa de fatorial e o fibop.c executa as operações recursiva e iterativa de Fibonacci.

Após organizar os códigos, testamos cada um deles na faixa de valores de 1 a 55 e imprimimos o tempo de execução de usuário, de sistema e de relógio (respectivamente) para cada um dos valores da faixa testada.

### Linhas de comando:

- make fat
- make fib

```
1 CC = cc
2 LIBS = -lm
3 SRC = src
4 OBJ = obj
5 INC = include
6 BIN = bin
7 OBJSFAT = $(OBJ)/fatop.o $(OBJ)/fat.o
8 HDRSFAT = $(INC)/fat.h
9 OBJSFIB = $(OBJ)/fibop.o $(OBJ)/fib.o
10 HDRSFIB = $(INC)/fib.h
11 CFLAGS = -Wall -c -pg -ftest-coverage -I$(INC)
12
13 EXEFAT = $(BIN)/fatop
14 EXEFIB = $(BIN)/fibop
15
16 fat: $(EXEFAT)
17     $(EXEFAT)
18
19 fib: $(EXEFIB)
20     $(EXEFIB)
21
22
23 $(BIN)/fatop: $(OBJSFAT)
24     $(CC) -pg -o $(BIN)/fatop $(OBJSFAT) $(LIBS) -pg
25
26 $(OBJ)/fatop.o: $(HDRSFAT) $(SRC)/fatop.c
27     $(CC) $(CFLAGS) -o $(OBJ)/fatop.o $(SRC)/fatop.c
28
29 $(OBJ)/fat.o: $(HDRSFAT) $(SRC)/fat.c
30     $(CC) $(CFLAGS) -o $(OBJ)/fat.o $(SRC)/fat.c
31
32
33 $(BIN)/fibop: $(OBJSFIB)
34     $(CC) -pg -o $(BIN)/fibop $(OBJSFIB) $(LIBS) -pg
35
36 $(OBJ)/fibop.o: $(HDRSFIB) $(SRC)/fibop.c
37     $(CC) $(CFLAGS) -o $(OBJ)/fibop.o $(SRC)/fibop.c
38
39 $(OBJ)/fib.o: $(HDRSFIB) $(SRC)/fib.c
40     $(CC) $(CFLAGS) -o $(OBJ)/fib.o $(SRC)/fib.c
41
42
43 cleanfat:
44     rm -f $(EXE) $(OBJSFAT) gmon.out
45
46 cleanfib:
47     rm -f $(EXE) $(OBJSFIB) gmon.out
```

Segue o resultado para alguns valores dos testes:

Fatorial iterativo e recursivo de 1 a 10:

```
Fatorial iterativo de 1: executou em 5.000000s de usuario, em 0.000000s de sistema e 0.000000 de relógio.
Fatorial recursivo de 1: executou em 0.000000s de usuario, em 0.000000s de sistema e 0.000000 de relógio.
Fatorial iterativo de 2: executou em 0.000000s de usuario, em 0.000000s de sistema e 0.000000 de relógio.
Fatorial recursivo de 2: executou em 0.000000s de usuario, em 0.000000s de sistema e 0.000000 de relógio.
Fatorial iterativo de 3: executou em 0.000000s de usuario, em 0.000000s de sistema e 0.000000 de relógio.
Fatorial recursivo de 3: executou em 0.000000s de usuario, em 0.000000s de sistema e 0.000000 de relógio.
Fatorial iterativo de 4: executou em 1.000000s de usuario, em 0.000000s de sistema e 0.000000 de relógio.
Fatorial recursivo de 4: executou em 0.000000s de usuario, em 0.000000s de sistema e 0.000000 de relógio.
Fatorial iterativo de 5: executou em 0.000000s de usuario, em 0.000000s de sistema e 0.000000 de relógio.
Fatorial recursivo de 5: executou em 0.000000s de usuario, em 0.000000s de sistema e 0.000000 de relógio.
Fatorial iterativo de 6: executou em 1.000000s de usuario, em 0.000000s de sistema e 0.000000 de relógio.
Fatorial recursivo de 6: executou em 0.000000s de usuario, em 0.000000s de sistema e 0.000000 de relógio.
Fatorial iterativo de 7: executou em 0.000000s de usuario, em 0.000000s de sistema e 0.000000 de relógio.
Fatorial recursivo de 7: executou em 0.000000s de usuario, em 0.000000s de sistema e 0.000000 de relógio.
Fatorial iterativo de 8: executou em 1.000000s de usuario, em 0.000000s de sistema e 0.000000 de relógio.
Fatorial recursivo de 8: executou em 0.000000s de usuario, em 0.000000s de sistema e 0.000000 de relógio.
Fatorial iterativo de 9: executou em 1.000000s de usuario, em 0.000000s de sistema e 0.000000 de relógio.
Fatorial recursivo de 9: executou em 0.000000s de usuario, em 0.000000s de sistema e 0.000000 de relógio.
Fatorial iterativo de 10: executou em 0.000000s de usuario, em 0.000000s de sistema e 0.000000 de relógio.
Fatorial recursivo de 10: executou em 0.000000s de usuario, em 0.000000s de sistema e 0.000000 de relógio.
```

Fibonacci iterativo e recursivo de 30 a 38:

```
Fibonacci iterativo de 30: executou em 0.000000s de usuario, em 0.000000s de sistema e 0.000000 de relógio.
Fibonacci recursivo de 30: executou em 0.000340s de usuario, em 0.000000s de sistema e 0.000341 de relógio.
Fibonacci iterativo de 31: executou em 2.000000s de usuario, em 0.000000s de sistema e 0.000000 de relógio.
Fibonacci recursivo de 31: executou em 0.000562s de usuario, em 0.000017s de sistema e 0.000579 de relógio.
Fibonacci iterativo de 32: executou em 1.000000s de usuario, em 0.000000s de sistema e 0.000000 de relógio.
Fibonacci recursivo de 32: executou em 0.000830s de usuario, em 0.000000s de sistema e 0.000832 de relógio.
Fibonacci iterativo de 33: executou em 1.000000s de usuario, em 0.000000s de sistema e 0.000000 de relógio.
Fibonacci recursivo de 33: executou em 0.001486s de usuario, em 0.000000s de sistema e 0.001496 de relógio.
Fibonacci iterativo de 34: executou em 1.000000s de usuario, em 0.000000s de sistema e 0.000000 de relógio.
Fibonacci recursivo de 34: executou em 0.002228s de usuario, em 0.000000s de sistema e 0.002229 de relógio.
Fibonacci iterativo de 35: executou em 1.000000s de usuario, em 0.000000s de sistema e 0.000000 de relógio.
Fibonacci recursivo de 35: executou em 0.003800s de usuario, em 0.000009s de sistema e 0.003810 de relógio.
Fibonacci iterativo de 36: executou em 1.000000s de usuario, em 0.000000s de sistema e 0.000000 de relógio.
Fibonacci recursivo de 36: executou em 0.005884s de usuario, em 0.000019s de sistema e 0.005903 de relógio.
Fibonacci iterativo de 37: executou em 1.000000s de usuario, em 0.000000s de sistema e 0.000000 de relógio.
Fibonacci recursivo de 37: executou em 0.009218s de usuario, em 0.000027s de sistema e 0.009245 de relógio.
Fibonacci iterativo de 38: executou em 1.000000s de usuario, em 0.000000s de sistema e 0.000000 de relógio.
Fibonacci recursivo de 38: executou em 0.014918s de usuario, em 0.000007s de sistema e 0.014925 de relógio.
```

Observação 1:

Podemos perceber, pelo tempo de execução dos algoritmos de Fibonacci recursivo e iterativo, que a partir do número 42, o algoritmo recursivo se mostra menos eficiente em tempo que o iterativo para encontrar o resultado. Como é possível analisar abaixo, se assumirmos TI e TR como os tempos de usuário para os algoritmos iterativo e recursivo respectivamente,  $TI > TR$  para números menores que 42. Caso o número seja maior ou igual a 42,  $TI < TR$ .

```
Fibonacci iterativo de 40: executou em 1.000000s de usuario, em 0.000000s de sistema e 0.000000 de relógio.
Fibonacci recursivo de 40: executou em 0.938818s de usuario, em 0.000003s de sistema e 0.938821 de relógio.
Fibonacci iterativo de 41: executou em 1.000000s de usuario, em 0.000000s de sistema e 0.000000 de relógio.
Fibonacci recursivo de 41: executou em 0.066592s de usuario, em 0.000007s de sistema e 0.966600 de relógio.
Fibonacci iterativo de 42: executou em 1.000000s de usuario, em 0.000000s de sistema e 0.000000 de relógio.
Fibonacci recursivo de 42: executou em 1.004058s de usuario, em 0.000001s de sistema e 1.004061 de relógio.
Fibonacci iterativo de 43: executou em 1.000000s de usuario, em 0.000000s de sistema e 0.000000 de relógio.
Fibonacci recursivo de 43: executou em 1.965630s de usuario, em 0.000000s de sistema e 1.965630 de relógio.
Fibonacci iterativo de 44: executou em 1.000000s de usuario, em 0.000000s de sistema e 0.000000 de relógio.
Fibonacci recursivo de 44: executou em 3.027581s de usuario, em 0.000001s de sistema e 3.027591 de relógio.
```

### Observação 2:

Apesar de os tempos de sistema e de relógio do algoritmo de Fibonacci tenderem a zero para os primeiros números dos testes, a partir do número 17, esses tempos se mostram maiores para o algoritmo recursivo, se tornando significativamente maiores nos últimos números de teste.

Já nos algoritmos recursivo e iterativo de fatorial, os tempos de sistema e de relógio foram de 0 segundos para todos os testes.

### Observação 3:

O tempo de usuário se mostrou maior que o tempo de sistema e de relógio na grande maioria dos testes, com a única exceção dos testes de Fibonacci recursivo, em que os dois mostraram ter o mesmo valor.

Em seguida, usamos o comando `gprof bin/fatop gmon.out > analise.txt` para rodar o gprof e gerar um relatório de execução.

Entretanto, os tempos fornecidos pelo gprof foram 0 para todas as operações executadas.

```
% cumulative self self total
time seconds seconds calls Ts/call Ts/call name
0.00 0.00 0.00 1 0.00 0.00 __static_initialization_and_destruction_0(int, int)
```

  

```
index % time self children called name
0.00 0.00 1/1 _GLOBAL__sub_I_main [10]
[8] 0.0 0.00 0.00 1 __static_initialization_and_destruction_0(int, int) [8]
-----
```

Não cheguei à conclusão de se isso aconteceu por um problema de ajuste numérico do computador ou se por o gprof não estar funcionando da maneira adequada. Logo, não foi possível fazer uma comparação dos tempos como pedido no item 5 e nem se mostrou útil fazer a chamada recursiva de uma função que consuma recursos computacionais, já que não seria possível comparar os tempos de execução.

Em seguida, o próximo passo foi adequar o código para que o comando `make run` funcionasse como `make fat` e `make fib` ao mesmo tempo. Dessa maneira, o novo makefile fica como a imagem abaixo. Além disso, adaptamos o código para que ele executasse apenas o fatorial e o Fibonacci iterativos e recursivos de 5 e medisse seu valor.

```

CC = gcc
LIBS = -lm
SRC = src
OBJ = obj
INC = include
BIN = bin
OBJSFAT = $(OBJ)/fatop.o $(OBJ)/fat.o
HDRSFAT = $(INC)/fat.h
OBJSFIB = $(OBJ)/fibop.o $(OBJ)/fib.o
HDRSFIB = $(INC)/fib.h
CFLAGS = -Wall -c -pg -ftest-coverage -I$(INC)

EXE = $(BIN)/fatop $(BIN)/fibop

run: $(EXE)
    $(BIN)/fatop
    $(BIN)/Fibop

$(BIN)/fatop: $(OBJSFAT)
    $(CC) -pg -o $(BIN)/fatop $(OBJSFAT) $(LIBS) -pg

$(OBJ)/fatop.o: $(HDRSFAT) $(SRC)/fatop.c
    $(CC) $(CFLAGS) -o $(OBJ)/fatop.o $(SRC)/fatop.c

$(OBJ)/fat.o: $(HDRSFAT) $(SRC)/fat.c
    $(CC) $(CFLAGS) -o $(OBJ)/fat.o $(SRC)/fat.c

$(BIN)/fibop: $(OBJSFIB)
    $(CC) -pg -o $(BIN)/fibop $(OBJSFIB) $(LIBS) -pg

$(OBJ)/fibop.o: $(HDRSFIB) $(SRC)/fibop.c
    $(CC) $(CFLAGS) -o $(OBJ)/fibop.o $(SRC)/fibop.c

$(OBJ)/fib.o: $(HDRSFIB) $(SRC)/fib.c
    $(CC) $(CFLAGS) -o $(OBJ)/fib.o $(SRC)/fib.c

cleanfat:
    rm -f $(EXE) $(OBJSFAT) gmon.out

cleanfib:
    rm -f $(EXE) $(OBJSFIB) gmon.out

```

```

bin/fatop
Fatorial de 5: 120.
Fatorial iterativo de 5: executou em 7.000000s de usuario, em 0.000000s de sistema e 0.000000 de relógio.
Fatorial recursivo de 5: executou em 0.000000s de usuario, em 0.000000s de sistema e 0.000000 de relógio.
bin/fibop
Fibonacci de 5: 5.
Fibonacci iterativo de 5: executou em 5.000000s de usuario, em 0.000000s de sistema e 0.000000 de relógio.
Fibonacci recursivo de 5: executou em 0.000000s de usuario, em 0.000000s de sistema e 0.000000 de relógio.

```

Repare que, para números pequenos, os algoritmos recursivos são consideravelmente mais eficientes que os iterativos.