



Eduardo Figueredo Pacheco

Relatório de informações sobre o Exercício Programa de Algoritmos e Estruturas de
Dados 1

São Paulo
2022

● Conceitos Matemáticos

Neste trabalho foi utilizado uma melhoria que só é garantida nesse caso específico em que são testados os números, em sequência, de um intervalo decidido pelo usuário. Então, a melhoria que foi adicionada faz com que, durante a sequência de loops, por cada número, caso o número atual esteja no intervalo dado até o momento anterior, sabe-se que já foi calculada a quantidade de vezes que esse elemento precisou para chegar até o número 1, então basta somar esse número com a quantidade armazenada no vetor na posição de tamanho (número atual) menos (número inicial).

exemplo:

4 6 // input pelo usuário do intervalo. Vão ser analisados os números {4, 5, 6}.

4 2 **1** // Como 4 é um número par, temos pela regra que $4/2 = 2$ e, da mesma forma, $2/2 = 1$

5 16 8 4 // Como 5 é ímpar e 16 é par seguiu a sequência. Contudo, pode-se observar que não houve continuidade depois do 4 porque o programa já havia salvo esse dado anteriormente num vetor de tamanho ideal para esse cálculo. Assim, foi somado +2 para a quantidade de vezes rodadas para chegar ao 1 pelo número 5.

6 3 10 5 // Seguiu a mesma ideia anteriormente citada, contudo, nesse exemplo chegou ao número 5, cujo resultado já havia sido calculado.

output do programa:

2 // número 4

5 // número 5

8 // número 6

o programa rodou em 0.000115 segundos

● Observações interessantes

É importante pontuar que, em intervalos pequenos, talvez o programa não se mostre tão otimizado tendo em vista a baixa quantidade de dados armazenados no vetor e, conseqüentemente, a baixa efetividade do código, em relação ao tempo de execução, pois esse terá que testar todos os casos, quase por completo. Contudo, em casos maiores, como os que serão mostrados posteriormente, temos que o tempo de execução diminui significativamente.

Outra coisa interessante é que, sendo o começo do intervalo de números pequenos, como 1 ou 2, 100% dos casos (exceto os próprios 1 e 2) se beneficiarão do método da otimização para acelerar o programa. Agora, ao testar para casos que começam com números maiores o resultado varia bastante, dependendo principalmente do tamanho do intervalo, à grandeza do número de início dado pelo usuário. Para fim de teste, foram testados alguns casos iniciando do número 500.

Então, para os intervalos:

[500,600] → 47% dos números testados neste intervalo utilizaram o vetor para otimização.

[500,700] → 67% dos números testados neste intervalo utilizaram o vetor para a otimização.

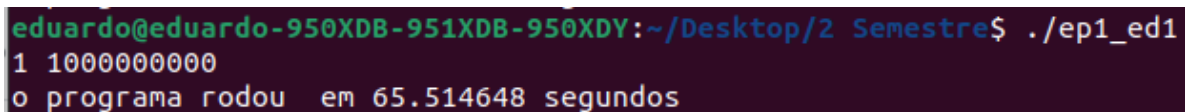
[500,1000] → 81% dos números testados neste intervalo utilizaram o vetor para a otimização.

● Dados e análise da otimização

A princípio, é importante pontuar que o tempo observado no output de cada execução do programa se trata do tempo que o programa leva para alocar a memória e realizar todos os cálculos do loop até chegar em “1” para cada número da sequência do intervalo indicado pelo usuário.

Para que fosse possível analisar até onde o programa poderia chegar, neste caso específico, foi retirado o output da quantidade de repetições por números, pois números muito grandes iriam demorar muito. Então apenas foi contado o tempo de execução; contudo, no programa apresentado ainda terão tais outputs.

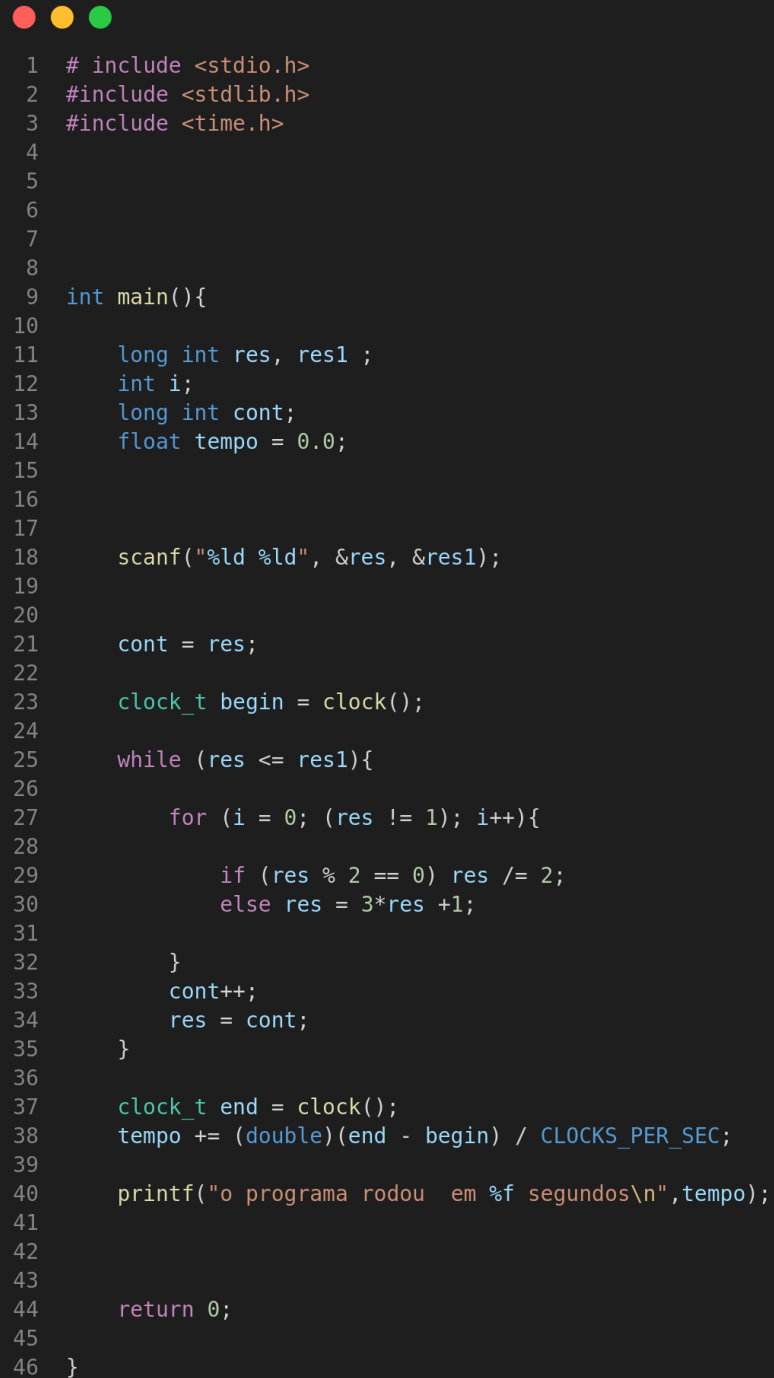
O maior caso de teste aconteceu do intervalo[1 , 1.000.000.000], ou seja :10⁹. No programa otimizado, sabe-se que o tempo foi de aproximadamente 1 minuto e 5 segundos, assim como pode ser visto na imagem 1.

A terminal window with a dark background and light-colored text. The prompt is 'eduardo@eduardo-950XDB-951XDB-950XDY:~/Desktop/2 Semestre\$'. The command './ep1_ed1' has been executed. The output shows '1 1000000000' on the first line and 'o programa rodou em 65.514648 segundos' on the second line.

```
eduardo@eduardo-950XDB-951XDB-950XDY:~/Desktop/2 Semestre$ ./ep1_ed1
1 1000000000
o programa rodou em 65.514648 segundos
```

imagem 1

Agora, para fins de comparação, foi feito um programa de exemplo (imagem 2) que não foi otimizado, mas que tinha a mesma função do anterior, contudo, já era esperado que o tempo de execução fosse maior.



```
1  # include <stdio.h>
2  #include <stdlib.h>
3  #include <time.h>
4
5
6
7
8
9  int main(){
10
11     long int res, res1 ;
12     int i;
13     long int cont;
14     float tempo = 0.0;
15
16
17
18     scanf("%ld %ld", &res, &res1);
19
20
21     cont = res;
22
23     clock_t begin = clock();
24
25     while (res <= res1){
26
27         for (i = 0; (res != 1); i++){
28
29             if (res % 2 == 0) res /= 2;
30             else res = 3*res +1;
31
32         }
33         cont++;
34         res = cont;
35     }
36
37     clock_t end = clock();
38     tempo += (double)(end - begin) / CLOCKS_PER_SEC;
39
40     printf("o programa rodou em %f segundos\n",tempo);
41
42
43
44     return 0;
45
46 }
```

Imagem 2

```
eduardo@eduardo-950XDB-951XDB-950XDY:~/Desktop/2 Semestre$ ./ex
1 1000000000
o programa rodou em 1107.194214 segundos
```

Imagem 3

Analisando o resultado do programa de exemplo, é notório que, aproximadamente 17 vezes mais rápido, o programa otimizado garantiu o tempo de execução de 1 minuto e 5 segundos, enquanto o não-otimizado obteve o resultado em 18 minutos e 30 segundos. Lembrando que para ambos os programas, não foi calculado o tempo necessário para que os resultados sejam mostrados na tela, pois isso elevaria o tempo para uma escala difícil de ser mostrada em relatório.

Ao ser testado o programa otimizado em 10^{10} , ou seja, no intervalo [1, 10.000.000.000], o programa não chega a rodar.