UNIVERSIDADE FEDERAL DE SÃO JOÃO DEL-REI
CAMPUS TANCREDO NEVES
DEPARTAMENTO DE CIÊNCIAS DA COMPUTAÇÃO

JOÃO VITOR ALVES MORAIS
LUAN LIDUÁRIO SILVA

TRABALHO PRÁTICO 1

SÃO JOÃO DEL-REI

2019

JOÃO VITOR ALVES MORAIS
LUAN LIDUÁRIO SILVA

TRABALHO PRÁTICO 1

Trabalho apresentado ao curso de Ciências da Computação da Universidade Federal de São João del-Rei, como parte dos requisitos necessários à obtenção de créditos na disciplina de Algoritimos e Estruturas de Dados III..

Professor: Leonardo Chaves Dutra da Rocha
Disciplina: Algoritmos e Estruturas de Dados III

SÃO JOÃO DEL-REI

2019

SUMÁRIO

1 INTRODUÇÃO

O objetivo do presente trabalho é criar um programa que tenha como saída o número de inteiros compostos por dois ou mais números primos diferentes (os inteiros "Xulambs", nome dado pelo professor Leonardo Rocha) provenientes da fatoração de um inteiro n, cujos valores estão entre 1 e 10^{12}.

Diante disso, espera-se que as primitivas básicas da linguagem C sejam revistas e exercitadas. Além disso, o trabalho busca trazer à tona discussão sobre problemas complexos e suas soluções.

O programa deverá ler um arquivo de texto contendo os números inteiros a serem analisados em cada linha. Ele deve gerar um arquivo de saída contendo o número de inteiros Xulambs presentes na fatoração dos inteiros da entrada.

2 REFERENCIAL TEÓRICO

Buscando melhorar o entendimento sobre tópicos que fizeram parte do trabalho, o referencial teórico apresenta os assuntos: "Tempo real, de usuário e sistema"e "Crivo de Eratóstenes".

2.1 Tempo real, de usuário e sistema

Tempo real diz respeito ao tempo que o programa gasto visto aos olhos humanos do início ao fim do programa. Já os tempo de usuário e sistema se referem ao tempo que o programa gasta em modo-usuário e modo-Kernel, respectivamente. O programa entra no modo-Kernel quando é necessário realizar funções mais privilegiadas e de baixo nível, como, por exemplo, acessar o hardware ou alocar memória. Enquanto esse modo é usado o tempo de sistema é aumentado. As funções de alto nível do programa são realizadas em modo-usuário e, nesse caso, o tempo de usuário que é contado. (MULONDA, 2018)

2.2 Crivo de Eratóstenes

Segundo Maier (2005), o método “Crivo de Eratóstenes” tem como função determinar os números primos menores que um número natural N. Nesse método tira-se todos os números menores que N que são múltiplos dos números primos menores que a raiz de N. Ainda segundo o autor, se não houver nenhum primo menor que a raiz de N que seja divisor dele, significa que N é primo. Além disso, esse método garante que não existe mais de um número primo maior que a raiz de N que seja seu divisor.

3 LISTAGEM DE FUNÇÕES

Descrição das funções criadas no desenvolvimento do trabalho e função *main*.

3.1 void lerArquivo (char *nomeArq, char *nomeSaida)

A função lerArquivo recebe os nomes dos aquivos de entrada e de saída. Após abrir o arquivo, depois cria um vetor de números primos preenchido pela função crivoEratostenes, e é alocado espaço na memória para receber o primeiro resultado. O laço de repetição, enquanto não encontra o final do arquivo, lê linha por linha do arquivo de texto. Cada linha corresponde a um inteiro N, o qual é submetido a uma conversão de tipo, de *string* para inteiro, então N recebe o resultado da função fatorar(N), em seguida N recebe calcXulambs (N), O resultado de N depois de passar por todos esses processos é salvo em um vetor. Em cada repetição é alocado espaço na memória para o próximo resultado. Os processos realizados em N é descrito no fluxograma da figura 1.

Quando é encontrado o fim do arquivo, é chamada a função escreverArquivo() para gravar o vetor resultante em um arquivo.

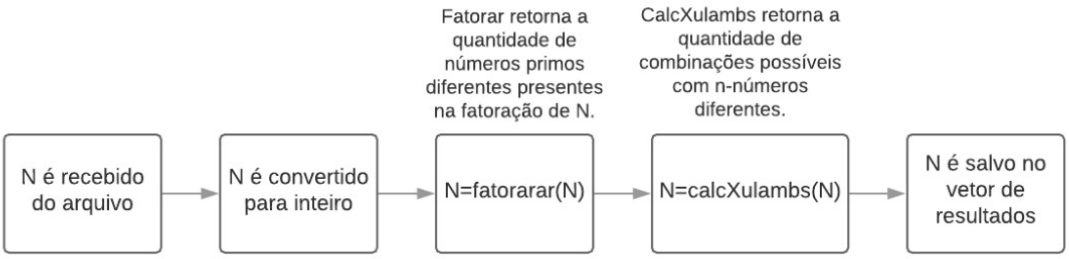

FIGURA 1 – **Fluxograma dos processos realizados nos números recebidos**

3.2 void escreverArquivo (char *nomeSaida, long long int *listaR, int i)

A função escreverArquivo recebe o nome do arquivo que deseja escrever os resultados obtidos, o vetor com os resultados e a quantidade de elementos que serão escritos. O laço de repetição, *for*, percorre o vetor recebido, cada posição contém um inteiro que é convertido em *string*, e é escrito em uma linha do arquivo.

3.3 double tempo()

Função que retorna o tempo desde 1 de janeiro de 1970, em microssegundos.

3.4 **long long int* crivoEratostenes()**

A função é baseada no conceito do crivo de Eratóstenes. Primeiro ela cria um vetor somente com números primos, para isso a função aloca um vetor (usando a função *calloc*) com um milhão de posições que são preenchidas com 0. Em seguida um laço de repetição faz de 2 a 1000 repetições, sempre somando 1 em uma variável I, que inicialmente é igual a 2 (usando a lógica do crivo, percorremos até a raiz do número total de posições do vetor, procurando os números primos). Se o elemento na posição I for 0, quer dizer que ele é primo, então um segundo laço de repetição atribui 1 nas posições do vetor que são múltiplos de I. Ao final deste processo a função retorna um vetor com todos os primos até um milhão marcados com valor 0 e o restante com valor 1.

3.5 **long long int qtdPrimos(long long int n,long long int* lista)**

Essa função recebe como parâmetro uma long long int que será fatorado (n) e a lista com todos os primos até um milhão. Ela retorna o número total de primos diferentes estão presentes na fatoração do número recebido.

Primeiramente, uma variável é criada para receber o número de primos (nPrimos), outra para auxiliar (j) e uma outra para receber o valor a ser fatorado (x) e dividi-lo. Depois é criado um laço com variável para percorrer o vetor (i) podendo ir de 2 até a raiz de n e algumas condições são conferidas: caso o valor da posição i da lista for 0 (significando que o número de seu índice é primo) e o resto da divisão de n por i for igual a zero, a variável x recebe a divisão de x por i. Dentro dessa condição, é conferido se a variável auxiliar é diferente do atual índice e caso seja, nPrimos é incrementado com mais um. Caso não entre na condição, o i é incrementado para que o próximo índice seja analisado. Ao final desse processo, se o x não chegou ao valor 1 temos dois casos: ou o número a ser fatorado é primo ou o último fator de n é maior que a raiz, nos casos basta somar mais um no contador de números primos.

3.6 **long long int fatorial (long long int n)**

Essa função recebe um número e retorna o seu fatorial. No início da função, a variável que será retornada recebe um. Depois disso é criado um laço que vai do valor recebido até 2 onde esse valor recebido é decrementado a cada nova repetição. Dentro do laço a variável de retorno recebe ela mesma multiplicada pelo valor atual da variável usada para percorrer o laço (i).

3.7 **long long int calcXulambs (long long int n)**

A função calcXulambs é baseada na fórmula de análise combinatória que consiste em $C_{n,p}= \frac{n!}{p!(n-p)!}$. Ela recebe como parâmetro um número que representa o número de primos encontrados na fatoração de um número (n). A princípio, duas condições são executadas: caso o valor recebido for menor que 2, sabemos que não há nenhum Xulambs na sua fatoração, então o valor 0 é retornado; caso seja igual a 2, sabemos que há somente um número Xulambs nessa fatoração, então o valor 1 é retornado. Caso não entre nessas condições, a variável de retorno recebe 1 (combinação dos n números de n em n, ou seja, todos eles juntos) e o um laço é criado. No laço a variável usada para percorrer o laço (i) vai diminuindo de n-1 até 2 e a variável de retorno recebe a soma dela mesma com o valor obtido da fórmula citada, sendo o n da fórmula o valor recebido como parâmetro e o p da fórmula a atual variável i no laço.

3.8 **int main (int argc, char **argv)**

A main recebe o nome dos arquivos de entrada e saída, usando a função getopt,. Logo depois é chamada a função tempo que é atribuído a um variável chamada t1. Então é chamada a função lerArquivo que é responsável por ler as entradas e realizar as operações necessárias para produzir o arquivo de saída. Após a função lerArquivo é chamada a função tempo novamente que atribui a outra variável, t2. Então é impresso na tela o tempo gasto pelo algoritmo, ou seja a diferença entre t2 e t1, e os tempos de sistema e de usuário.

4 ANÁLISE DE COMPLEXIDADE

Descrição da análise de complexidade das funções citadas.

4.1 lerArquivo

A função chama a função crivoEratostenes que tem complexidade de O(1), depois o laço de repetição percorre o arquivo todo, sendo n a quantidade de elementos do arquivo. Neste laço a cada repetição é chamada outras duas funções: calcXulambs() e qtdPrimos(), que tem complexidade respectivamente O(n^2) e O($\sqrt{n}$). Então O=(O(1)+n(O(n^2)+ O($\sqrt{n}$)) = O(n^3).
Complexidade: O(n^3)

4.2 escreverArquivo

É uma função simples que escreve n elementos em um arquivo utilizando um laço de repetição, ou seja, realiza n operações. Além disso, existi um condicional *if* que verifica se arquivo foi aberto corretamente, sendo assim O(n) + O(1) = O(n).
Complexidade: O(n)

4.3 double tempo

É uma função que apenas retorna o tempo em microssegundos, ou seja, realiza apenas uma operação.
Complexidade: O(1)

4.4 crivoEratostenes

A função é O(1), porque independente da entrada irá realizar o mesmo número de operações.
Complexidade: O(1)

4.5 qtdPrimos

A função realiza no máximo o número da raiz quadrada de sua entrada de operações. Caso a entrada seja um número primo o laço de repetição irá até a sua raiz sem passar pela condição do primeiro *if*. Após o laço de repetição existe outro *if*, ou seja a complexidade é O($\sqrt{n}$) + O(1) = O($\sqrt{n}$).
Complexidade: O($\sqrt{n}$)

4.6 **fatorial**

Considerando que existe um laço de repetição que realiza atribuição a uma variável, a função realiza n operações.
Complexidade: O(n)

4.7 **calcXulambs**

Em seu pior caso a função realiza n operações, sendo que a cada interação, chama a função fatorial 3 vezes que tem complexidade O(n), ou seja, O((n-2)(3n))=O(n^2).
Complexidade: O(n^2)

4.8 **main**

A função main chama as funções tempo e lerArquivo, que tem complexidade, respectivamente, O(1) e O(n^3), logo a complexidade da main é O(1) + O(n^3) = O(n^3). Ou seja o algoritmo tem complexidade de O(n^3).
Complexidade: O(n^3)

5 TESTES

Foram criados três algoritmos para solucionar o problema dado. A diferença entre eles está nas funções usadas para encontrar a quantidade de números primos diferentes divisores de determinado inteiro. O conceito de crivo de Eratóstenes, explicado anteriormente, foi utilizado em dois destes testes e o outro fazia uma fatoração do inteiro recebido.

181091225252	11
252	4
11	0
97	0
18	1
6	1
6270	26
4181490	26
8	0
8371885830	502

FIGURA 2 – **Exemplo de entradas e saídas do programa**

5.1 Crivo de Eratóstenes 1

Essa função se mostrou muito ineficiente para muitas entradas ou com valores grandes. Para cada inteiro N, era preenchido um vetor de tamanho de 2 a N, depois era retirado todos os números primos do vetor utilizando o conceito de crivo de Eratóstenes, e por ultimo verificava se quais dos primos encontrados era um divisor de N, para cada primo encontrado se somava 1 em um contador, então ao final do processo a função retornava o contador. Após avaliar o código, constatou se que o algoritmo gastaria muito tempo criando vários vetores e os percorrendo muitas vezes.

5.2 Fatoração

A função apresentou melhor desempenho do que a descrita no tópico anterior. Nela foi utilizado o conceito de fatoração, que consiste decompor um número utilizando fatores primos. Para cada inteiro N, atribuía-se N a uma variável X, verificava-se era divisível por 2 e, enquanto fosse, X recebia $\frac{x}{2}$, caso não fosse se somava 1 ao contador. Esse processo era repetido por 3, 4, 5 até X ser igual a 1 ou se o contador ultrapassar

$\frac{N}{2}$+1, pois nenhum número é divisível por mais de sua metade. Para cada vez que X era dividido por algum fator, esse fator era salvo em outra variável, quando era X dividido novamente se verificava se o fator era diferente salvo na divisão anterior, caso fosse se somava 1 em um contador nPrimos, então ao final do processo a função retornava o contador nPrimos. Essa função se apresentou funcional, ou seja retornava os valores corretos, mas quando N era primo ou fosse maior que 10^4, o algoritmo apresentava certa demora como mostrado na tabela da Figura 3.

5.3 **Crivo de Eratóstenes 2**

Esse método foi o que se mostrou o mais eficiente entre algoritmos testados. Nele também é usado o conceito do crivo de Eratóstenes, mas de uma forma um pouco diferente. Pela definição desse conceito, sabe-se que não é possível que haja mais de um número primo maior que a raiz do número a ser fatorado, que esteja nessa fatoração. Devido a isso, um vetor de um milhão de posições é criado em uma função, já preenchido totalmente com zeros logo na sua criação. O vetor possui esse tamanho porque um milhão elevado ao quadrado é o maior número possível que será usado no trabalho, um trilhão, ou seja, nunca será possível que mais de um número primo maior que um milhão esteja presente na fatoração. Então esse vetor é percorrido até a posição 1000 (raiz de um milhão), se na atual posição o valor for zero todas as posições múltiplas desse número têm seu valor alterado para 1. Então quando o programa encontrar posições com valor 1 elas são ignoradas e a próxima posição é analisada. Ao final essa lista é retornada, consequentemente com valor zero nas posições com índices que são números primos e 1 no resto.

A segunda etapa desse método é fatorar o número recebido. Para isso uma função é criada para receber o número a ser fatorado e a lista de números primos. A lista de primos é então percorrida até a raiz do número recebido (usando mais uma vez a lógica do crivo). O programa vai conferindo se o número é divisível pelos índices das posições que possuem zero como valor: caso seja divisível, a divisão é feita e o processo se repete com esse mesmo índice até que não seja mais divisível; caso não seja divisível o programa analisa o próximo índice válido. Nesse processo existe uma variável que serve para contar o número de números primos diferentes presentes na fatoração e uma outra para saber se é a primeira vez que o índice é analisado. Então, quando é a primeira vez que o índice é analisado e ele é um divisor na fatoração a variável do número de primos aumenta em 1. A lógica do crivo nos garante que se chegamos na raiz do número a ser fatorado e sua fatoração não chegou ao final, pode significar duas coisas: que esse número é primo ou só existe mais um número nessa fatoração. Então no cenário do trabalho só é necessário adicionar 1 na variável do número de primos, uma vez que se o número for primo essa variável estará com valor 0 e, se não só está faltando um número na fatoração. Ao final essa variável é retornada.

5.4 **Comparações**

Para verificar qual solução tem melhor desempenho fizemos alguns testes. Para realizar os testes foram utilizados cinco elementos iguais para os três algoritmos, ou seja com o limite inferior 0 e limite superior 100, foram utilizados 18,78,97,11 e 57. Segue a tabela com os resultados obtidos (Figura 3). O tempo foi obtido com a função *gettimeofday* e é contado o tempo de entrada e saída, ou seja, o tempo de ler e escrever no arquivo também foram considerados, o limite inferior dos elementos gerados aleatoriamente é o limite superior anterior.

Limites Superior	Fatoração	Crivo de Eratóstenes 1	Crivo de Eratóstenes 2
100	0.012729	0.024989	0.035989
1000	0.008865	0.018546	0.039665
10000	0.012307	0.010022	0.039722
100000	0.013999	0.022282	0.041663
1000000	0.047328	0.085717	0.040982
10000000	0.028930	0.708290	0.041226
100000000	0.265927	8.053.159	0.035720
1000000000	0.069972	51.550.277	0.035395
10000000000	0.064687	erro	0.037247
100000000000	20.405.056	erro	0.037260
1000000000000	893.462.398	erro	0.054657

FIGURA 3 – **Tabela de comparação entre os algoritmos, em relação a tempo de entrada e saída em microssegundos**

Pode se observar que em números com limite superior de 10^9 o crivo de Eratóstenes 1 travou o sistema operacional impossibilitando continuar com os testes. Enquanto o algoritmo de fatoração aumentou consideravelmente o tempo no limite superior 10^{10}. Em entradas com valores pequenos o crivo 2 apresentou pior desempenho, por sempre criar o vetor de primos antes de receber qualquer inteiro N, porém manteve o desempenho quando os valores das entradas aumentaram. Segue o gráfico que compara o desempenho dos algoritmos (Figura 4). Para melhor visualização foram considerados resultados inferiores a 1 micro segundo, nele pode se observar o notório aumento do tempo gasto do crivo1, e o pior desempenho do crivo 2 em entradas com valores pequenos.

5.5 **Testes Finais**

Com as comparações realizadas nos tópicos anteriores foi decidido optar pelo Crivo de Eratóstenes 2, por manter o desempenho independente do valor da entrada. A partir deste tópicos todos os testes foram realizados no algoritmo Crivo 2. Os testes foram realizados com um arquivo *excel* e um *word* aberto, e uma aba do navegador. O gráfico da Figura 5 demonstra o tempo gasto pelo algoritmo, os tempos foram obtidos

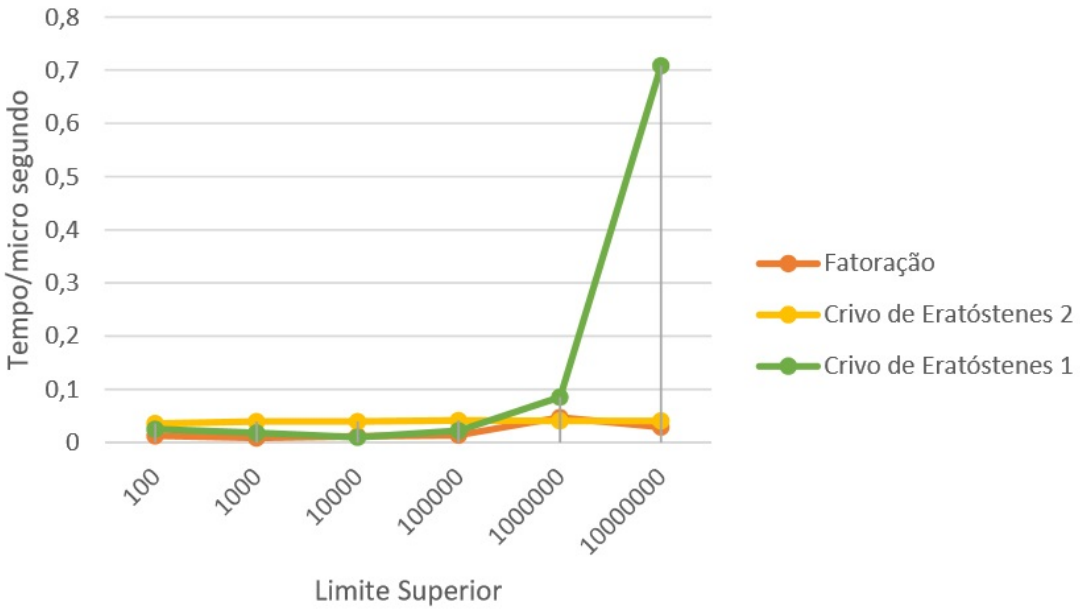

FIGURA 4 – **Gráfico de comparação entre os algoritmos, em relação a tempo de entrada e saída em microssegundos**

pelas funções *gettimeofday*, e função *getrusage*. Esse teste foi realizado com apenas 5 entradas.

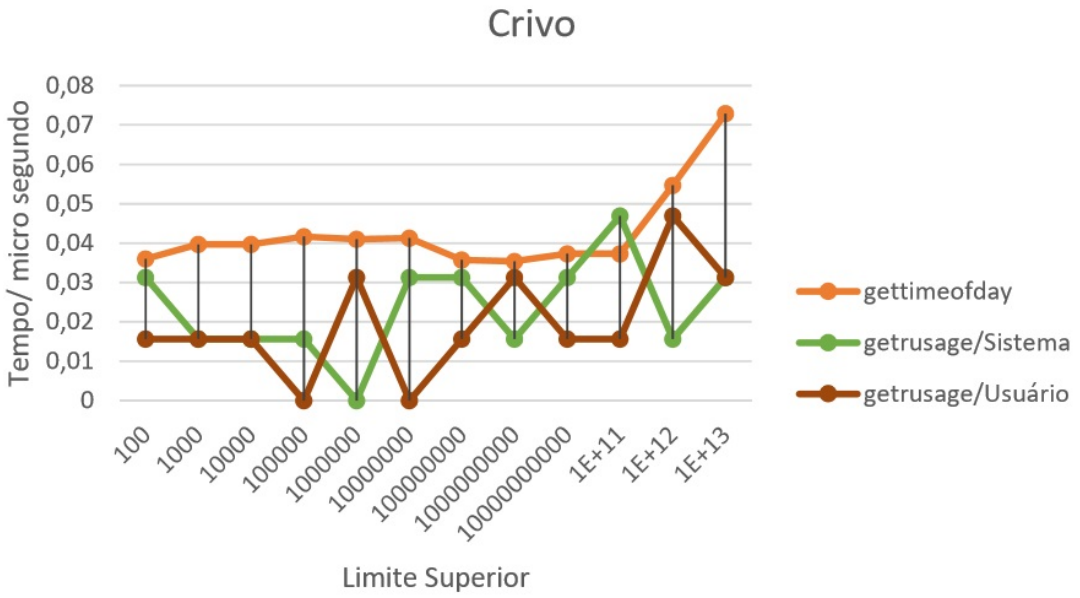

FIGURA 5 – **Gráficos com tempo real, tempo de sistema e tempo de usuário do algoritmo da versão final**

O gráfico da figura 6 demonstra o comportamento do algoritmo com aumento da quantidade de entrada, com valores aleatórios de 0 a 10^{12}.

FIGURA 6 – **Gráficos da variação do tempo de execução com diferentes tamanhos de entrada**

6 CONCLUSÃO

A implementação do trabalho apresentou algumas dificuldades no decorrer de sua execução. O primeiro protótipo do programa fazia uso, de forma ineficiente, do algoritmo do “crivo de Eratóstenes”. Por causa disso, o programa demorava muito em sua execução, causando o travamento do sistema em caso de números muito grandes. O segundo protótipo usava um algoritmo de fatoração que basicamente testava todos os números inferiores até sua metade. Esse processo não travava o sistema, mas demorava muitos minutos, tornando o algoritmo ineficiente. Então uma variação do algoritmo do crivo de Eratóstenes foi usada, solucionando o problema do programa, que passou a rodar em um tempo aceitável.

A dupla avalia os resultados do trabalho como positivos, uma vez que o programa foi capaz de realizar as funções solicitadas nos testes. Além disso, a dupla trabalhou bem em conjunto e exercitou os conceitos que foram pedidos.

REFERÊNCIAS

MAIER, Rudolf R. Teoria dos números. **Universidade de Brasılia-Departamento de Matematica-IE**, 2005. Citado 1 vez na página 5.

MULONDA, Yann. **How to get the execution time of a script**. Jul. 2018. Disponível em: <https://medium.com/coinmonks/kernel-space-vs-user-space-how-to-get-the-execution-time-of-a-script-11c56290d8f4>. Citado 1 vez na página 5.