Diener Dornelas
Franciane Pereira Gonçalves
2019/2

1° TRABALHO DE AEDS III
DIVISORES XULAMBS

São João Del Rei-MG
UFSJ

Sumário

1. Introdução

Um divisor Xulambs de um número é obtido através dos divisores primos deste número e todas as comparações possíveis entre eles.

O objetivo deste trabalho prático é encontrar a quantidade de divisores Xulambs de cada entrada inserida no arquivo de texto, e as respectivas quantidades são colocadas no arquivo de saída.

O algoritmo funciona da seguinte maneira: primeiro as entradas do arquivo são lidas,é feita a fatoração de cada um deles, excluindo os divisores repetidos. Os divisores primos são armazenados no vetor junto com os primos entre eles até o último divisor primo. Depois que esses valores são armazenados no vetor, são feitas as comparações entre eles, primeiro dois a dois, três a três e assim sucessivamente enquanto houverem divisores. Ao fim desse processo, serão obtidos os divisores Xulambs de cada entrada.

As comparações foram baseadas nas fórmulas de arranjo $An,p = n!/(n-p)!$ e combinação $C(n,p) = n!/(p!(n-p)!)$.

O código foi dividido em três partes: tp1_fc.c, tp1_main.c, e header.h, para uma melhor organização e entendimento do programa, além de dois arquivos de texto, em que um armazena as entradas e o outro as saídas respectivamente.

Para a contagem de tempo de usuário e sistema, foi utilizada a função **getrusage** para obter o resultado total de tempo após executar todas as entradas do arquivo. Após a observação desses tempos foi possível representar graficamente todo o comportamento do programa.

2. Proposta de Trabalho

Este trabalho tem por objetivo exercitar primitivas básicas da Linguagem C e iniciar a discussão sobre problemas complexos e a sua solução.

Conforme veremos ao final dessa disciplina, os principais e mais utilizados algoritmos de criptografia são baseado na teoria dos números primos. Os números inteiros podem ser representados como um produto de potência de números primos. Por exemplo, 126 = 21∗32∗7. Resumidamente, algoritmos como RSA utilizam o conceito de chaves públicas e privadas. Essas, por sua vez, são compostas, em sua essência, da multiplicação de números primos muito grandes. A decomposição desses números levaria muitos e muitos séculos de processamento intenso para que pudesse ser realizada. Mas fique tranquilo, por ora, você não precisa se preocupar exatamente como funciona o processo de criptografia.

Nesse trabalho vamos trabalhar com o conceito de inteiros Xulambs! Um inteiro é Xulambs se pode ser escrito como um produto de dois ou mais primos distintos, sem repetição. Por exemplo, 15 = 3∗5, 14 = 2∗7 e 21 = 3∗7 são inteiros Xulambs. Porém 4 = 22, 5, 7, 8 = 23 e 18 = 2∗32 não são inteiros Xulambs.

Considerando como entrada um arquivo contendo um inteiro N ($1 \leq N \leq 10^{12}$) em cada linha. Você deve fazer um programa que produza um arquivo de saída contendo um inteiro por linha que representa o número de divisores Xulambs de cada valor N passado no arquivo de entrada.

3. Desenvolvimento

A intenção deste trabalho prático foi encontrar a quantidade de divisores Xulambs dos números no arquivo de entrada.

No processo para encontrar esses divisores, antes de qualquer coisa, o programa aloca um vetor dinamicamente dentro da função de fatoração que inicialmente terá 50 espaços na memória, depois é feita uma verificação para inferir se o tamanho já é adequado ou não. A verificação é feita da seguinte forma: se o valor de entrada for maior que 1000000 o vetor será realocado com tamanho 1000001; Caso seja menor, o vetor será realocado com o tamanho limitep que é igual ao tamanho do número de entrada em si. Após a alocação do vetor o algoritmo encontra os divisores primos de 2 até o último divisor primo do número de entrada e armazena esses primos dentro deste vetor.

É definido que o primeiro elemento do vetor sempre será 2, a partir disso o valor de entrada passa pela fatoração que funciona da seguinte forma: o valor é dividido pelo item da primeira posição do vetor, caso o resto desta divisão seja 0, então encontrei o primeiro número primo divisor do valor, mais uma vez o programa dividirá o valor resultante da primeira divisão por este mesmo número do primeiro índice, se o resto desta divisão for zero isso não será somado como mais um divisor primo, e se o resto da divisão for diferente de zero o programa procura o próximo número primo, insere no vetor, verifica se é divisor do mesmo modo que o primeiro e assim sucessivamente até encontrar o último divisor primo do valor de entrada.

Depois de encontrar todos os divisores primos do valor, ainda é preciso todo um processo para encontrar a quantidade de divisores xulambs. Para isso foi utilizado equações matemáticas que auxiliassem a solucionar o problema, foram elas arranjo e combinação: $An,p = n!/(n-p)!$ e $C(n,p) = n!/(p!(n-p)!)$, respectivamente, para encontrar o total de combinações possíveis entre todos os divisores primos, que é a quantidade de divisores Xulambs.

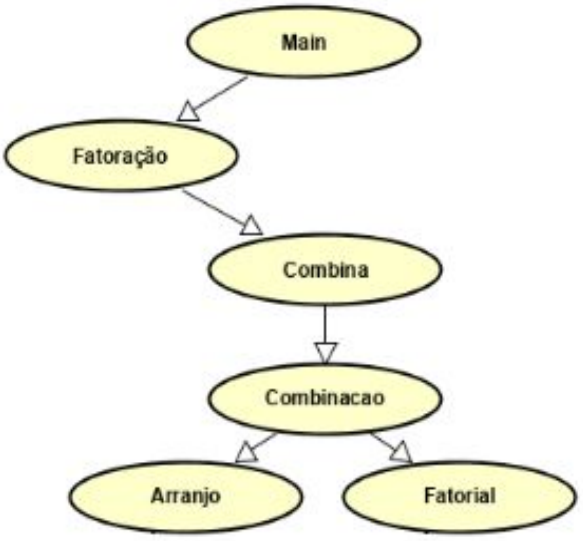
A busca dos divisores Xulambs começa a partir do momento em que o programa já possui a quantidade de divisores primos do valor inicial, essa quantidade é passada para a função *combina* junto com o arquivo de saída. Então a função *combina* inicializa uma variável x=0, que armazenará a soma da quantidade de Xulambs que será atribuída na função *combinação,* que é responsável por dividir o arranjo da quantidade de divisores com a quantidade de números a serem comparados pelo fatorial da quantidade de números a serem comparados $arranjo(qtdD,p)/fatorial(p)$, e então é possível obter a quantidade de divisores Xulambs que a função *combina* imprime no arquivo de saída.

O tipo utilizado nas variáveis é long int, para que comporte o tamanho das entradas que podem ser até de 10^{12}.

Para a contagem do tempo total que o programa gasta para diferentes tamanhos e quantidades dos números foi utilizada a função **getrusage** para essa contagem e tornar possível a representação gráfica dos casos.

2.1 As funções:

Cada função possui uma chamada à outra que passa os parâmetros necessários para sua execução. O fluxograma abaixo representa a ordem em que o algoritmo funciona.



Fatorial: É a função responsável por fatorar a quantidade de números a serem comparados.

```
fatorial (p) {
    resultado =1
enquanto (p for maior 1 ){
 resultado = fatorial de p
}
retorna resultado
}
```

Arranjo: É uma função que atribui arranjo da quantidade de divisores primos da entrada e o número de valores a serem comparados, baseando-se na fórmula $An,p = n!/(n-p)!$.

```
arranjo ( qtdD, p){
    arr=1
    enquanto (p maior ou igual a 1){
        arr = arr * quantidade de divisores do valor de entrada
        decrementa quantidade de divisores
        decrementa p
    }
        retorna o arranjo
}
```

Combinação: A função efetua a fórmula de combinações, utilizando os resultados dos fatoriais e dos arranjos. Esta função efetua a divisão da combinação das quantidades de números primos divisores do valor de entrada, com o número de comparações, pelo fatorial do número de comparações. A fórmula matemática em que foi baseada é $C(n,p) = n!/(p!(n-p)!)$.

```
combinacao(qtdD,p){
    retorna a divisão do arranjo de qtdD e p, pelo fatorial de p
}
```

Combina: É uma das principais funções do algoritmo, pois é ela que dá início ao processo de combinações para encontrar a quantidade de divisores Xulambs, armazena essa quantidade e retorna para o arquivo de saída.

```
combina(qtdD, arquivo de saída){
    x=0
    for(p inicializa em 2 e é sempre menor ou igual à quantidade de divisores primos, incrementa p){
        x recebe a soma de combinação todas as vezes, é a quantidade de divisores Xulambs
    }
    imprime a quantidade de Xulambs no arquivo de saída
}
```

Fatoração: Essa função é o ponto de partida do algoritmo, pois ela receberá os valores do arquivo de entrada, fará a fatoração deles para encontrar os divisores, fará o teste para comprovar a primalidade de cada um deles e só então chamará a função combina para encontrar a quantidade dos divisores Xulambs, objetivo do algoritmo. No final da função ela retorna para o arquivo de saída a quantidade procurada, encerra os arquivos de texto e limpa o vetor de divisores que foi alocado dinamicamente.

```
fatoracao(){ na primeira chamada aloco dinamicamente o vetor onde
    guardarei os numeros primos de 2 ate a Raiz Quadrada  do maior numero
    que eu tiver como entrada

    enquanto (num diferente de 1){ Verifico se o numero que recebi do
        arquivo de texto(num), é divisivel por 2 que é o item na posição 0  do
        meu vetor de numeros primos(VetPrimos), caso seja a divisão é
        repetida, mas se o resto da divisão  for diferente de 0 a proxima
        etapa é procurar o número primo sucessor ao que foi adicionado no
        vetor  na posição (NumPrimos-1). Então retorno a tentativa de divisão
        até encontrar o divisor de num que o  resultado é 1 e quebrar a
        condição que mantém dentro do laço. Se após verificar todos os números
        até a Raiz Quadrada de num e não tiver achado nenhum divisor que
        quebre a condição do laço, o programa assume que o numero de entrada é
        primo e termina o processo atual.
    }
}
```

2.2 Organização do código

O programa foi dividido em três partes, sendo elas :

tp1_main.c: que é a onde está função principal do programa, onde os arquivos de entrada e saída são abertos, a contagem do tempo de execução do sistema tem início e fim e a função fatoração é chamada.
tp1_fc.c: todas as funções do programa, exceto a main estão neste documento. No fim de toda a execução os arquivos de texto são fechados e o vetor que armazena os divisores primos das entradas é limpo.
header.h: neste arquivo estão as bibliotecas utilizadas pelo programa e todas as assinaturas das funções.

Além disso o programa possui dois arquivos de texto, um possui as entradas que serão processadas e no outro serão impressas as saídas encontradas pelo algoritmo.

Para medir o tempo de execução do sistema e do usuário, foi utilizado a função **getrusage**, com as variáveis Systemp e Usertemp, uma para armazenar o tempo do sistema e a outra o tempo do usuário, respectivamente. Os tempos armazenados são os tempos totais de execução, de forma em que é pego o tempo inicial e após obter o tempo final, o inicial é subtraído deste. O tempo do sistema computacional é quase imperceptível, visto que em existem casos em que o tempo final é zero ou bem próximo dele, enquanto o tempo do usuário leva alguns segundos a mais .

4. Complexidade

Fatorial: O(p) onde p é o número de pares para as combinações.

Arranjo: O(p) onde é o número de quantos números eu tenho para gerar uma combinação.

Combinação: O(p²) pois é onde eu tenho um for que chamar *fatorial e arranjo* que são O(n) para gerar as combinações de quantos Xulambs eu tenho para cada número recebido.

Fatoração: $O(n\sqrt{numMax})$ onde n é o número de linhas que tenho no arquivo de entrada e numMax é o maior número do arquivo de entrada.

Main: $O(n\sqrt{numMax}) + O(p^2)$ pois tenho a chamada para a função *Fatoração* que por consequência chama a função *Combinação* que fica responsável de gerar quantos Xulambs tenho e imprimir no arquivo de saída.

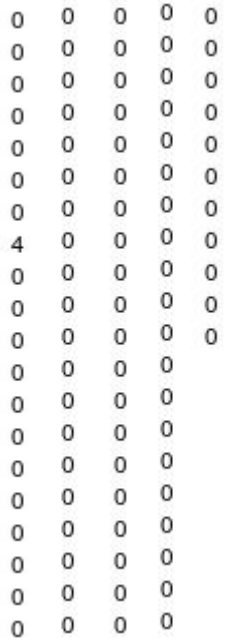
5. Resultados e testes

Entradas

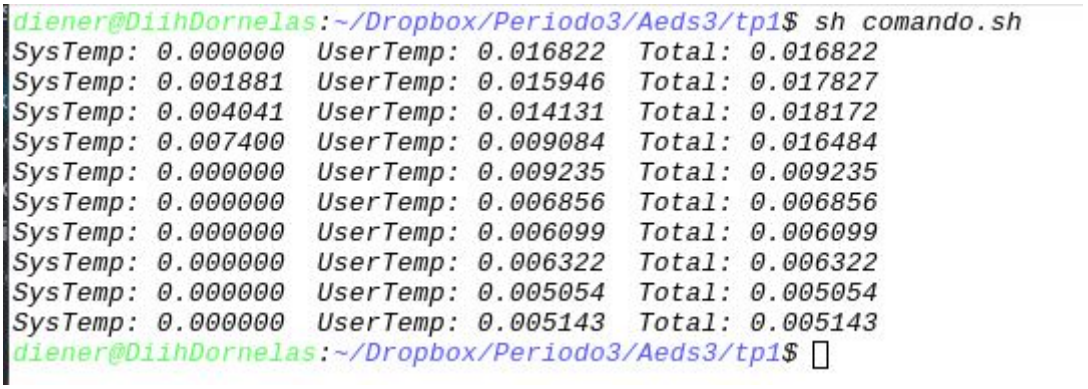
Foram utilizadas para este teste 171 números de entrada, obtidos através de um gerador online. Todos os números deste teste são primo,exceto o número 95761 que tem 3 divisores.

218921827	7995991	7996601	7996741	7987387	7987687	7988093	7996993
5300891	7996031	7996607	7996787	7987393	7987699	7988117	7996997
50273801	7996049	7996619	7996789	7987403	7987703	7988129	7997029
861589	7996061	7996631	7996823	7987423	7987717	7988131	7997047
16843	7996073	7996649	7996843	7987439	7987729	7988177	7997051
5451503	7996081	7996337	7996867	7987457	7987751	7988203	7997057
32957	7996097	7996363	7996871	7987459	7987757	7988219	7997071
95761	7996099	7996367	7987123	7987477	7987781	7988221	7997083
7995769	7996133	7996369	7987129	7987481	7987799	7996873	7997107
7995773	7996187	7996381	7987139	7987487	7987813	7996889	7997113
7995791	7996193	7996423	7987223	7987531	7987823	7996903	7997131
7995809	7996231	7996463	7987241	7987537	7987843	7996913	7997141
7995817	7996243	7996487	7987247	7987541	7987867	7996951	7997147
7995821	7996259	7996501	7987249	7987559	7987883	7996957	7997153
7995839	7996277	7996519	7987253	7987561	7987897	7996969	7997159
7995863	7996309	7996661	7987289	7987589	7987901	7996973	7997167
7995877	7996319	7996669	7987307	7987591	7987921	7996981	
7995901	7996333	7996687	7987313	7987607	7987957		
7995907	7996537	7996691	7987319	7987613	7988009		
7995917	7996559	7996697	7987327	7987627	7988033		
7995929	7996577	7996711	7987351	7987649	7988063		
7995931	7996589	7996727	7987363	7987663	7988081		
7995937	7996591	7996733	7987373	7987669	7988089		

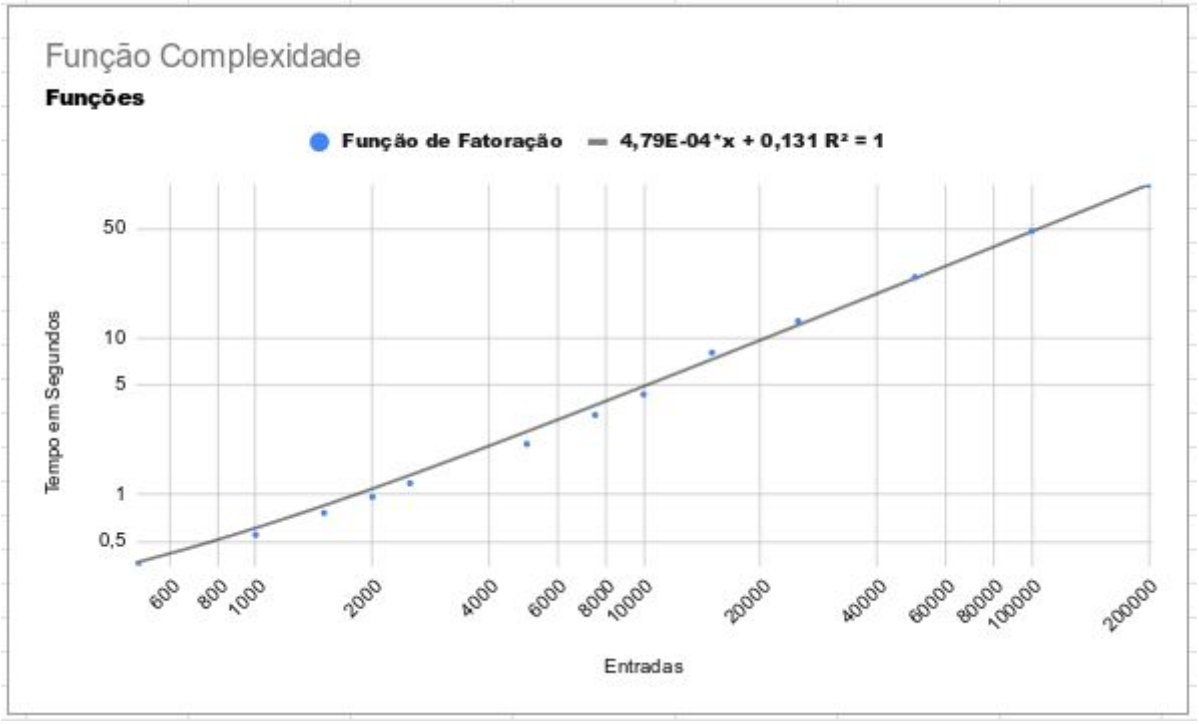
Estes são os resultados de divisores Xulambs das respectivas entradas da imagem anterior. Como são números primos não possuem nenhum divisor Xulambs exceto o inteiro 95761 que possui 4 divisores Xulambs.



Estes são os resultados dos tempos de sistema, usuário e o total obtido entre os dois tempos. Essa obtenção de tempo é feita através da função **getrusage** onde obtenho os tempo de quando começou e terminou o algoritmo, e imprimindo-os na tela separados e a soma dos dois.



6. Tempos de execução



(Gráfico com os tempos de execução em função da quantidade de números de entrada. A linha contínua representa a equação de regressão da função como uma estimativa do comportamento esperado e a linha pontilhada é o resultado real da função).

7. Conclusão

I Implementar um código sobre um assunto desconhecido pelos seus autores até então, tem seus desafios, pois as propostas e estratégias mudam ao longo da implementação para que o programa tenha uma melhor funcionalidade e atenda à todos os casos.

Quando tratamos de poucos e pequenos números, ter uma visão geral de como será o desenvolvimento do algoritmo é mais simples, mas quando é colocado para teste uma grande quantidade de números com muitos dígitos alguns erros podem surgir e, solucioná-los depois de que toda uma lógica foi montada para fazer o programa funcionar, é a maior dificuldade.

É possível observar que após a implementação destes códigos em C, uma maior afinidade com a linguagem é obtida, não só no quesito manipulação, mas também uma melhor organização e visualização do código.

Ao final da implementação do programa ele atendia à todas as expectativas da problemática de encontrar os divisores Xulambs e imprimir a quantidade deles no arquivo.

8. Referências

http://marmsx.msxall.com/cursos/c/c16.html

https://forum.scriptbrasil.com.br/topic/152202-programa-para-c%C3%A1lculo-de-n%C3%BAmeros-primos/