Trabalho Prático 2

Crivo Quadrático

Gabriel Campos Prudente - 2022069425

Leonardo Romano Andrade - 2023075151

Vinicius Trindade Dias Abel - 2020007112

Vitor Costa - 2022043418

Departamento de Ciência da Computação - Universidade Federal de Minas Gerais (UFMG)
Belo Horizonte, MG, Brasil

1. Descrição do trabalho

Este trabalho tem o objetivo de aplicar e entender o algoritmo do Crivo Quadrático (CQ) para a fatoração de números grandes. O Crivo Quadrático foi inventado por Carl Pomerance em 1981 e foi o mais rápido algoritmo de fatoração conhecido até que o Crivo de Campo de Número fosse descoberto em 1993. Ainda assim, o CQ continua mais rápido que seu concorrente para números de até 110 dígitos.

O algoritmo, que usa um raciocínio semelhante ao do algoritmo de Fermat, tenta encontrar inteiros x e y tais que $x \not\equiv \pm y \pmod{n}$ e $x^2 \equiv y^2 \,(mod\, n)$. Isso implicaria que $(x - y)(x + y) \equiv 0 \,(mod\, n)$, e simplesmente calcularíamos $(x - y, n)$ usando o Algoritmo Euclidiano para verificar se um número é divisor não-trivial.

2. Descrição dos módulos e interdependência

O código implementado para a fatoração de números grandes utilizando o Crivo Quadrático foi dividido em módulos, cada um responsável por uma parte específica do processo. A seguir, apresentamos uma descrição detalhada dos principais módulos e como eles interagem entre si.

2.1 Módulo calcular_B

Este módulo é responsável por calcular o limite superior B para os primos que serão utilizados no Crivo Quadrático. O valor de B é crucial, pois determina o conjunto de números primos que serão considerados na fatoração de f(x).

- **Função Principal**: calcular_B(N)
  - **Entrada**: O número N a ser fatorado.
  - **Saída**: O valor de B, calculado como $B = e^{0,5*\sqrt{log(N)*log(log(N))}}$
  - **Interdependência**: Este módulo é utilizado diretamente pelo módulo quadratic_sieve para determinar os primos a serem utilizados no processo de fatoração.

2.2 Módulo quadratic_sieve

Este é o módulo principal do programa, responsável por implementar o algoritmo do Crivo Quadrático para a fatoração do número N. Ele integra as funcionalidades dos outros módulos para realizar a fatoração de forma eficiente.

- **Funções Principais**:
  - **Passo 1: Cálculo do Limite Superior B**
    - Utiliza o módulo calcular_B para obter o valor de B e, em seguida, gera a lista de primos até B.
  - **Passo 2: Cálculo de f(x)**
    - Calcula os valores da função $f(x) = (x + \sqrt{N})^2 - N$ para M valores de x, onde M é um parâmetro configurável.
  - **Passo 3: Fatoração de f(x)**
    - Fatora cada valor de f(x) na base dos primos gerados e armazena a fatoração em uma matriz de fatores.
  - **Passo 4: Resolução do Sistema Linear**
    - Resolve o sistema linear mod 2 usando Decomposição em Valores Singulares (SVD) para encontrar uma combinação não trivial de fatores.
  - **Passo 5: Determinação dos Fatores de N**
    - Utiliza o MDC para calcular os fatores de N a partir dos produtos x e y.
  - **Interdependência**:
    - O módulo quadratic_sieve depende do módulo calcular_B para obter o limite superior B.
    - Ele também faz uso de funções da biblioteca gmpy2 para operações aritméticas de precisão arbitrária e numpy para operações matriciais.

2.3 Interdependência Geral

Os módulos calcular_B e quadratic_sieve estão intimamente interligados, com o primeiro fornecendo os parâmetros necessários para que o segundo execute o algoritmo de Crivo Quadrático. A função quadratic_sieve integra todo o processo, desde o cálculo dos parâmetros iniciais até a obtenção dos fatores de N.

3. Formato de entrada dos dados

O programa utiliza a biblioteca "gmpy" de python para operar com números grandes. Assim, utilizamos o formato "mpz", da biblioteca citada acima, para declarar e operar esses valores muito grandes.

4. Formato de saída dos dados

O programa gera 5 saídas, sendo elas:

1) **Limite superior** *(int)* - indica até onde os primos são considerados na base de fatoração. Ele é calculado usando uma estimativa baseada nos logaritmos de N e serve como uma referência para escolher os números primos que serão utilizados para a fatoração.
2) **Número de primos** *(int)* - indica quantos números primos foram selecionados na base de fatoração até o limite superior
3) **Tamanho do vetor de índices 'j'** *(int)*- Este valor representa quantos valores da função polinomial foram calculados. Isso corresponde ao número de linhas na matriz de fatoração, onde cada linha corresponde à fatoração de um valor da função polinomial.
4) **x e y** *(mpz)* - São os valores calculados a partir das combinações dos valores da função polinomial que formam quadrados perfeitos (ou se aproximam disso). x é o produto das soluções de x e y é a raiz quadrada do produto dos valores da função polinomial.
5) **mdc(x-y , N) e mdc(x+y , N)** *(mpz)* - são os fatores de N obtidos através do cálculo do MDC entre (x − y) e N, e entre (x + y) e N, respectivamente.

5. Como utilizar o programa

O programa foi escrito em python utilizando a plataforma google colab. Para utilizar o programa, basta acessar a página do colab e executar a primeira célula (ctrl + enter na célula ou apertar o botão de "play" no canto superior esquerdo da célula), que irá instalar as bibliotecas necessárias para a utilização do código. Após isso, basta escolher o número a ser fatorado na penúltima linha de segunda célula (`n = '999999999000000001' # Exemplo de número a ser fatorado`)
que está como "999999999000000001" por padrão.

Após selecionar o número, basta executar a célula e, então, o programa irá fatorar o número e gerar as saídas especificadas anteriormente.

6. Complexidade dos algoritmos

6.1 Eliminação Gaussiana

Um passo crítico para o processo de fatoração é a eliminação Gaussiana, que usamos para encontrar o espaço coluna da solução. A matriz formada é grande e esparsa. Podemos usar técnicas padrões de álgebra linear para tentar reduzi-la e acelerar o algoritmo. Nesse sentido, uma observação trivial é que, se uma coluna possui apenas um único 1, podemos excluir a linha associada com ela. Não existe a possibilidade de que Q(x) seja um fator de um quadrado. Em nosso trabalho, usamos a eliminação Gaussiana convencional, cuja complexidade de tempo é $O(n^3)$, onde n é a dimensão da matriz. No que diz respeito à complexidade de espaço, precisamos alocar um espaço na memória da ordem de $O(n^2)$.

Uma possível melhoria para essa complexidade seria por meio do Algoritmo de Blocos de Wiedemann, que paraleliza a matriz em várias máquinas diferentes com o intuito de acelerar o processo de eliminação. Com ele, a complexidade de tempo passaria a ser aproximadamente

$$O(B(w + B\,ln(B)\,ln(ln(B))))$$

onde B é o número de fatores primos na base de fatores e w é o número aproximado de operações elementares requerido para multiplicar a matriz por um vetor. Como a matriz é esparsa, w seria bem pequeno. A complexidade de espaço também decairia para $2B^2 \in O(B^2)$ na memória.

6.2 Tempo de execução

Se o número de primos na nossa base de fatores fosse bem pequeno, não precisaríamos de muitas fatorações de Q(x) a fim de se obter um possível fator do número de entrada n. O problema em fazer isso é que encontrar cada uma das poucas fatorações completas levaria muito tempo, uma vez que uma proporção muito pequena de números são fatorados por um conjunto também pequeno de primos. Se fôssemos criar uma lista enorme de primos de maneira que tudo fosse fatorável pela base de fatores, nosso problema passaria a ser obter todos esses números para criar uma matriz grande o suficiente para reduzirmos. Sendo assim, o número de primos deve ser escolhido de maneira a otimizar a performance do algoritmo. Segundo Landquist (2001, p. 9), o valor ótimo para o tamanho da base de fatores primos é aproximadamente

$$B = (e^{\sqrt{ln(n)ln(ln(n))}})^{\frac{\sqrt{2}}{4}}$$

O intervalo de crivagem acaba sendo o cubo desse valor:

$$M = B^3 = (e^{\sqrt{ln(n)ln(ln(n))}})^{\frac{3\sqrt{2}}{4}}$$

Nesse sentido, o tempo de crivagem deve ser três vezes maior que o tempo de redução da matriz. Com B primos em nossa base de fatores, esse passo do algoritmo roda em tempo menor que

$O(B^3)$ e nos produz o tamanho do intervalo de crivagem. Juntando essas informações, a função de custo de tempo assintótico do crivo quadrático $T(n)$ é dada por

$$T(n) \in O(M) \approx (e^{\sqrt{1.125ln(n)ln(ln(n))}})$$

7. Listagem do programa fonte

O programa foi implementado em Python e interpretado pelo *CPython*. O código fonte pode ser encontrado na plataforma Google Colaboratory no endereço eletrônico <https://colab.research.google.com/drive/185apabqgBLnFCClxCaPutNb9LzDGfzmL?usp=sharing>.

8. Referências

LANDQUIST, E. The Quadratic Sieve Factoring Algorithm. *In:* **Cryptography Applications Bistro Seminar**, 2001.

AYODELE, A. Here's How Quadratic Sieve Factorization Works. Nerd for Tech: 6 fev. 2022. Disponível em: <https://medium.com/nerd-for-tech/heres-how-quadratic-sieve-factorization-works-1c878bc94f81>. Acesso em: 12 ago. 2024.