Trabalho Prático 1
Logaritmo Discreto - Álgebra A

Lucas André dos Santos 2022093032

Rodrigo Luiz Macedo Ferreira 2020007007

Vinicius Trindade Dias Abel 2020007112

Universidade Federal de Minas Gerais (UFMG)
Belo Horizonte - MG - Brasil

1. Introdução

A presente documentação é responsável por relatar o desenvolvimento do trabalho prático da matéria de Álgebra A ministrada pelo professor Fabio Enrique Brochero Martinez. Este trabalho tem como objetivo desenvolver um algoritmo para encontrar um primo gigante, procurar um elemento gerador do grupo multiplicativo do número primo e determinar o logaritmo discreto de um elemento no grupo para valores numéricos gigantes.

Para solucionar as funcionalidades propostas foi escolhido desenvolver o algoritmo utilizando a linguagem Python por ela possuir a precisão aritmética de tamanho arbitrário de forma nativa, dividindo o desenvolvimento em diferentes módulos para a elaboração e estruturação do algoritmo buscando utilizar algoritmos otimizados para ser possível encontrar o logaritmo discreto dos números de tamanho arbitrário em tempo razoável.

2. Módulos e interdependência

O problema foi desenvolvido em diferentes módulos, permitindo a organização das funcionalidades em diferentes agrupamentos para resolver cada cálculo necessário na resolução dos problemas. Os módulos são divididos para calcular o número primo, fatoração, gerador do grupo multiplicativo e logaritmo discreto.

2.1. Primitividade

O cálculo do próximo primo é feito através da utilização do algoritmo de Miller-Rabin repetidas vezes para cada valor ímpar maior que o valor inicial que não é divisível pelos fatores pequenos possíveis. Para o cálculo foi armazenada uma lista dos 20 primeiros primos para verificação da divisão pelos fatores primos mínimos e para utilização nos testes de Miller-Rabin.

Em relação a interdependência, a busca pelo próximo primo é o primeiro módulo a ser executado no programa, assim não possuindo dependência de nenhum módulo externo.

2.2. Fatoração

O cálculo da fatoração é feito através da combinação do algoritmo da divisão por primos pré-computados, divisão por tentativa e Pollard 's rho utilizando o algoritmo busca-ciclos de Floyd. Para aplicar essa combinação o algoritmo verifica os primeiros 20 primos pré-computados, utiliza a divisão por tentativa para testar todos os primos ímpares não divisíveis por 3 e 5 até 1.000.000 e utiliza o Pollard 's rho para tentar buscar o resto dos fatores. Antes de cada execução do Pollard' s rho é utilizado o algoritmo de primitividade de Miller-Rabin para verificar se o valor restante é primo. Como o algoritmo pode não encontrar os fatores em tempo razoável, foi definido um limite de 5 minutos para tentar encontrar todos os fatores primos, assim retornando o fator restante que não foi possível fatorar.

Em relação a interdependência, a fatoração em fatores primos recebe o primo gigante gerado pelo módulo de busca pelo primo e utiliza o algoritmo de Miller-Rabin para reduzir o número de execuções necessárias do algoritmo de Pollard 's Rho.

2.3. Gerador do grupo multiplicativo

O elemento gerador é encontrado buscando valores no grupo multiplicativo do número primo p com ordem igual a cada potência primal dos fatores de p-1, assim multiplicando todos os valores encontrados para resultar no gerador do grupo. Como a fatoração pode não encontrar os fatores em tempo razoável, busca-se um valor com ordem igual ao produtório de todas as potências primais, assim multiplicando esse valor pelos elementos com ordem igual a cada potência primal dos fatores, resultando em um elemento ordem alta e estimativa de ordem mínima igual ao produtório das potências primais.

Em relação a interdependência, o gerador do grupo multiplicativo recebe os fatores e o fator remanescente após as divisões sucessivas pelos fatores descobertos do módulo da fatoração e o primo gigante gerado pelo módulo de busca pelo primo.

2.4. Logaritmo Discreto

O cálculo do logaritmo discreto pode ser feito através da escolha de utilização dos algoritmos de Força Bruta, Baby-step Giant-Step e Pohlig-Hellmann para tentar calcular o logaritmo discreto em tempo e espaço razoável. Para aplicar essa escolha o algoritmo define que valores primos menores que 1000 são calculados utilizando o Força Bruta, valores primos menores que 1.000.000.000 são calculos utilizando o Baby-Step Giant-Step e para o restante dos casos utiliza-se o algoritmo de Pohlig-Hellmann. No algoritmo de Pohlig Hellmann utiliza-se a fatoração da ordem do valor primo calculada previamente para dividir o problema de cálculo do logaritmo discreto em diversos cálculos de logaritmo discreto para cada fator primal, assim o algoritmo calcula o logaritmo discreto de fator primo utilizando os algoritmos de Força Bruta e Baby-Step Giant-Step com a mesma lógica em relação ao tamanho do valor primo.

Como a fatoração pode não ter sido possível calcular em tempo razoável utiliza-se o elemento com ordem alta para tentar calcular o logaritmo discreto utilizando unicamente o Força Bruta, já que o Pohlig-Hellmann necessita da fatoração total para funcionar e o Baby-step Giant-step teria problemas de espaço por o valor primo ser muito grande. Além disso, como o algoritmo pode não encontrar o logaritmo discreto em tempo e espaço razoável foi definido um limite de 10 minutos e a aplicação do algoritmo de Baby-step Giant-step apenas para valores primos menores que 1.000.000.000, assim não retornando um logaritmo caso não seja possível calcular.

Em relação a interdependência, o cálculo do logaritmo discreto recebe o primo gigante gerado pelo módulo de busca pelo primo, os fatores encontrados da fatoração da ordem do primo e o gerador do grupo multiplicativo.

3. Utilização do programa

Para a execução do algoritmo é necessário a inserção de dois valores inteiros utilizados para calcular os resultados estabelecidos no objetivo, assim recebendo pela entrada de dados dois valores inteiros de tamanho arbitrário. Esses dados são inseridos no programa através do armazenamento deles em um documentos de texto, assim inserindo na primeira linha o número inteiro arbitrário para que seja encontrado o próximo número primo maior que esse valor e na segunda linha inserindo o número para calcular o seu logaritmo discreto com base no grupo multiplicativo.

Dado a inserção dos dados o algoritmo retorna diversas informações acerca do processo até encontrar o logaritmo discreto, sendo eles o próximo primo maior que o valor inserido, número de testes de Miller-Rabin necessários, gerador do grupo multiplicativo ou elemento de ordem alta com estimativa da ordem mínima, logaritmo discreto do valor inserido dado o gerador encontrado e o tempo de execução total do programa em segundos ou a informação de que não foi possível calcular o logaritmo discreto em tempo razoável. A saída dos dados no programa é feita com legendas para identificação dos dados de duas formas, gerando os valores no prompt de comando e em um documento de texto chamado ‘resultado.txt'.

Com o arquivo de texto de inserção dos dados criado o programa é iniciado utilizando Python3 para executar o arquivo ‘main.py’, sendo recebido na linha de comando o nome do documento de texto. Para ser possível executar o algoritmo é necessário que o Python 3 esteja instalado no computador. Desse modo, com o Python3 instalado o algoritmo é executado com o comando ‘python3 main.py “exemplo.txt”’, substituindo o “exemplo.txt” pelo nome do arquivo de texto de inserção criado.

4. Análise de complexidade

Analisando os diferentes módulos do algoritmo é possível calcular a complexidade do algoritmo. Desse modo analisando a complexidade de cada módulo chega-se aos resultados a seguir:

- Primitividade: O algoritmo executa diversos testes de Miller-Rabin para verificar a primitividade de cada possível primo utilizando uma lista de primeiros primos para

utilização no teste. Desse modo, o módulo possui complexidade de tempo O($klog_2 n$) e espaço O(k), sendo ‘k’ o número de testes de Miller-Rabin realizados (definido como no máximo 20 para cada primo provável) e ‘n’ o valor que vai ser testada a primitividade.

- Fatoração: Como a fatoração baseia-se na junção de diversos algoritmos é necessário a análise da complexidade de tempo para diferentes situações dependendo das propriedades do valor a ser fatorado, enquanto a complexidade de espaço se mantém para todos os casos como O(F), sendo ‘F’ o número de fatores primos do número.
    - Quando todos os fatores são menores que 1.000.000 utiliza-se apenas o algoritmo de divisão por tentativa e de primos pré-computados que calculam os valores independente da entrada gerada. Desse modo, possui complexidade de tempo O(1)
    - Quando há apenas um fator maior que 1.000.000 utiliza-se uma execução do Miller-Rabin para identificar a primitividade desse fato. Desse modo, possui complexidade de tempo O($klog_2 n$), sendo ‘n’ o fator remanescente após as divisões sucessivas pelos fatores descobertos anteriormente e ‘k’ o número de testes de Miller-Rabin realizados.
    - Quando possui mais de um fator maior que 1.000.00 é necessário executar o algoritmo de Miller-Rabin seguido do algoritmo de Pollard’ rho para cada fator restante a ser identificado, diminuindo o valor testado a cada fator encontrado através da divisão do valor anterior pelo fator encontrado. Desse modo, possui complexidade de tempo O($\sqrt[4]{n}$), sendo ‘n’ o fator remanescente após as divisões sucessivas pelos fatores descobertos anteriormente.
- Gerador do grupo multiplicativo: O cálculo do gerador do grupo multiplicativo baseia-se na busca de um elemento com ordem igual a potência primal de cada fator resultante da fatoração. Desse modo, possui complexidade de tempo e espaço O(f), sendo ‘f’ o número total de fatores primos.
- Logaritmo discreto: Como o logaritmo discreto baseia-se na escolha entre diferentes algoritmos é necessário a análise da complexidade de tempo e espaço para as diferentes situações dependendo do tamanho do valor primo utilizado:

- Quando o valor primo é menor que 1000 ou a fatoração é parcial utiliza-se apenas o algoritmo de Força Bruta que calcula os valores testando cada valor presente no presente no grupo multiplicativo. Desse modo, possui complexidade de tempo O(p) e espaço O(1), sendo ‘p’ o valor primo
- Quando o valor primo é maior que 1000 e menor que 1000000000 utiliza-se o algoritmo de Baby-Step Giant-step de tamanho $\sqrt{p}$ que utiliza o armazenamento e cálculo de duas tabelas de baby e giant steps para buscar correspondência entre as duas tabelas. Desse modo, possui complexidade de tempo O($\sqrt{p}$) e espaço O($\sqrt{p}$), sendo ‘p’ o valor primo
- Quando o valor primo é maior que 1000000000 é necessário utilizar o algoritmo de Pohlig Hellmann dividindo o cálculo do logaritmo discreto de p no cálculo do logaritmo discreto de cada fator primo da fatoração da ordem do valor primo. Desse modo, possui complexidade de tempo O(pi) e espaço O(1) quando o maior fator primo for menor que 1000 e O($\sqrt{pi}$) e espaço O($\sqrt{p}$i) para o resto dos casos, sendo ‘pi’ o maior fator primo da fatoração da ordem de p

Portanto, o programa tem complexidade de tempo e espaço que varia dependendo do tamanho do valor de inserção e do tamanho dos fatores da ordem do primo, mas possui como piores caso de complexidade de tempo O($\sqrt{p}$), O($\sqrt{pi}$) ou O($\sqrt[4]{n}$), já que nesses casos os valores de ‘p’,’pi’ ou ‘n’ podem ser muito grandes, assim tornando o cálculo complexo, enquanto a pior complexidade de espaço possível é O($\sqrt{p}$i), sendo inviável para valores ‘pi’ muito grandes. Com isso, percebe-se que o algoritmo tem complexidade alta pelos valores serem muito grandes, assim não conseguindo executar em tempo razoável para determinados valores de entrada. Com isso, percebe-se a necessidade do controle de tempo de execução para valores que a resposta não é executável em tempo razoável, mesmo utilizando algoritmos otimizados para tentar diminuir esse tempo de cálculo.

5. Conclusão

Portanto, podemos concluir que o trabalho lidou com o problema da aplicação de algoritmos optimizados de cálculo de Álgebra buscando encontrar os valores buscados em tempo razoável mesmo utilizando valores de tamanho arbitrário, assim possibilitando observar valores primos gigantes no qual a aplicação do El Gamal seria possível aplicá-los devido a dificuldade no cálculo do logaritmo discreto

Desse modo, através do desenvolvimento do trabalho foi possível ampliar o aprendizado sobre a vantagem da utilização do logaritmo discreto na criptografia devido a dificuldade em seu cálculo, percebendo como o desempenho dos algoritmos em tempo razoável é impossível para valores primos específicos.

6. Referências

ALGORITHMS FOR COMPETITIVE PROGRAMMING. **Integer factorization**. 2024. Disponível em: <https://cp-algorithms.com/algebra/factorization.html>. Acesso em: 30 mai. 2024

ALGORITHMS FOR COMPETITIVE PROGRAMMING. **Primality tests**. 2024. Disponível em: <https://cp-algorithms.com/algebra/primality_tests.html>. Acesso em: 30 mai. 2024

CRYPTOLOGIE. **Pohlig-Hellman Algorithm**. Disponível em: <https://www.cryptologie.net/article/196/pohlig-hellman-algorithm/>. Acesso em: 4 jun. 2024.

FORT & FORGE. **Survey of discrete log algorithms**. Disponível em: <https://fortenf.org/e/crypto/2017/12/03/survey-of-discrete-log-algos.html>. Acesso em: 4 jun. 2024.

GEEKSFORGEEKS. **Chinese Remainder Theorem in Python**. Disponível em: <https://www.geeksforgeeks.org/chinese-remainder-theorem-in-python/>. Acesso em: 4 jun. 2024.

GEEKSFORGEEKS. **Discrete Logarithm (Find integer k such that a^k is congruent to b mod n)**. Disponível em: <https://www.geeksforgeeks.org/discrete-logarithm-find-integer-k-ak-congruent-modulo-b/>. Acesso em: 4 jun. 2024.

GEEKSFORGEEKS. **Primality test| Set 3 (Miller-Rabin).**2022. Disponível em: <https://www.geeksforgeeks.org/primality-test-set-3-miller-rabin/>. Acesso em: 1 jun. 2024

GIT HUB. Gist: **Baby-step giant-step algorithm in Python**. Disponível em: <https://gist.github.com/0xTowel/b4e7233fc86d8bb49698e4f1318a5a73>. Acesso em: 4 jun. 2024.

GIT HUB. Crypton: **Baby-step giant-step algorithm implementation***. GitHub. Disponível em: <https://github.com/ashutosh1206/Crypton/blob/master/Discrete-Logarithm-Problem/Algo-Baby-Step-Giant-Step/bsgs.py>. Acesso em: 4 jun. 2024.

GIT HUB. Crypton: **README for Baby-step giant-step algorithm**. GitHub. Disponível em: <https://github.com/ashutosh1206/Crypton/blob/master/Discrete-Logarithm-Problem/Algo-Baby-Step-Giant-Step/README.md>. Acesso em: 4 jun. 2024.

GIT HUB. **Pohlig-Hellman algorithm**. GitHub. Disponível em: <https://github.com/mcerovic/PohligHellman> Acesso em: 4 jun. 2024.

GIT HUB. **Pohlig Hellman Python Implementation**.GitHub. Disponível em: <https://github.com/christiankrug/pohlig_hellman> Acesso em: 4 jun. 2024.

GIT HUB. **python_pohlig-hellman**. GitHub. Disponível em: <https://github.com/IssacZF/python_pohlig_hellman> Acesso em: 4 jun. 2024.

GIT HUB. **Fast prime factorization in python**. 2023. Disponível em: <https://github.com/nishanth17/factor>. Acesso em 1 jun. 2024

GIT HUB. **NTheory simpy implementation.** 2024. Disponível em: <https://github.com/sympy/sympy/blob/master/sympy/ntheory/residue_ntheory.py>. Acesso em 4 jun. 2024

IKENAGA, Bruce. **Cyclic groups.** 2019. Disponível em: <https://sites.millersville.edu/bikenaga/abstract-algebra-1/cyclic-groups/cyclic-groups.pdf>. Acesso em 1 jun. 2024

MARTINEZ, Fabio. **Calculo do gerador.** 2024. Notas de aula. Não paginado

MVP WORKSHOP. **Baby-step giant-step algorith**m. Disponível em: <https://mvpworkshop.co/baby-step-giant-step-algorithm/>. Acesso em: 4 jun. 2024.

RISENCRYPTO. **The Pohlig-Hellman Algorithm**. Disponível em: <https://risencrypto.github.io/PohligHellman/>. Acesso em: 4 jun. 2024.

WIKIPEDIA. **Pohlig-Hellman algorithm**. Disponível em: <https://en.wikipedia.org/wiki/Pohlig%E2%80%93Hellman_algorithm>. Acesso em: 4 jun. 2024.

WIKIPEDIA. **Baby-step giant-step**. Disponível em: <https://en.wikipedia.org/wiki/Baby-step_giant-step>. Acesso em: 4 jun. 2024.

YOUTUBE. **Solving the DLP: Baby Step/Giant Step Algorithm** .Disponível em: <https://www.youtube.com/watch?v=007MVsELvQw>. Acesso em: 4 jun. 2024.

YOUTUBE. **Chinese Remainder Problems 1**. Disponível em: <https://www.youtube.com/watch?v=oKMYNKbFHBE>. Acesso em: 4 jun. 2024.