# Introdução

No presente trabalho são apresentadas cinco funções hash que podem vir a ser utilizadas subsequentemente para o projeto em questão. Estas funções são para o cálculo hash com objetivo de indexação de dados.

Descrevemos neste documento metodologia empregada, o funcionamento de cada função hash, os dados de entrada e a comparação das funções via experimento.

# Metodologia

O estudo foi dividido em duas partes: revisão bibliográfica e experimentos.

Feita revisão bibliográfica é relatado o funcionamento interno de cada função de hash. Definido os dados de entrada para os ensaios experimentais são feitos os cálculos hash para cada entrada, os gráficos em histograma que mostram as repetições de valores são mostrados nos gráficos.

# Revisão Bibliográfica

## A função hash de Robert Sedgwicks (Algorithms in C book)

Essa função hash consiste em somar cada caractere da chave ao produto valor hash calculado anteriormente por uma variável que é resultado do produtório de uma constante inicial por uma constante permanete. O mecanismo em questão torna o resultado da função hash dependente de todos os caracteres da chave, o custo de computação é linear dependendo do comprimento da chave tendo comportamento assintótico O(n) em todos os casos para o cálculo.

## A função hash de Justin Sobel

Essa função utiliza o deslocamento de bits (rotação), para cada caractere da chave é somado ao valor hash já calculado deslocado uma constante a esquerda e ao valor hash deslocado uma constante à direita, o resultado passa por uma operação bit a bit XOR com o valor hash já calculado anteriormente. O valor hash de saída é dependente de todos os caracteres da chave de entrada e o custo de computação é linear dependendo apenas do comprimento da chave, ou seja, tem comportamento assintótica O(n) em todos os casos para o cálculo.

## A função hash de Brian Kernighan and Dennis Ritchie's (The C Programming Language)

Essa função se baseia em somar o caractere da chave com o produto do hash anteriormente calculado por uma constante (semente) que por característica fundamental é uma permutação dos dígitos 1 e 3 (ex.: 31, 131, 1313, 13131, …). O valor hash de saída da função depende de todos os caracteres de entrada o custo de computação é linear dependendo apenas do comprimento da chave, ou

seja, tem comportamento assintótica O(n) em todos os casos para o cálculo. Essa função se destaca pela simplicidade do cálculo para cada caractere que se resume a duas operações elementares, um produto e uma soma.

### A função hash de Professor Daniel J. Bernstein

Está entre as funções hash mais eficientes já publicadas, consiste em somar a cada caractere da chave a soma do hash anteriormente calculado pelo mesmo hash deslocado uma constante a esquerda.O valor hash de saída da função depende de todos os caracteres de entrada o custo de computação é linear dependendo apenas do comprimento da chave, ou seja, tem comportamento assintótica O(n) em todos os casos para o cálculo. O diferencial da eficiência dessa função está na realização de duas somas e um deslocamento por caractere no cálculo da função de hash, sendo que as operações de soma e deslocamento estão entre as operações mais baratas para o processador.

### A função hash ELF

É uma variação da função hash de Peter J. Weinberger da AT&T Bell Labs, essa variante é muito utilizada em sistemas UNIX. Esse algoritmo consiste em somar cada caractere da chave ao hash anteriormente calculado cujos bits são deslocados uma constante (calculada conforme arquitetura da máquina, tamanho da palavra) à esquerda, então os bits de mais alta ordem (os mais à esquerda) são deslocados uma constante (proporcional ao tamanho da palavra) à direita e operam bit a bit XOR com o hash calculado. O valor hash de saída da função depende de todos os caracteres de entrada o custo de computação é linear dependendo apenas do comprimento da chave, ou seja, tem comportamento assintótica O(n) em todos os casos para o cálculo, algumas de suas operações por caractere como o deslocamento dos bits de mais alta ordem ocorre de maneira condicional, reduzindo o custo de computação para chaves pequenas ou o custo no início da computação de qualquer chave.

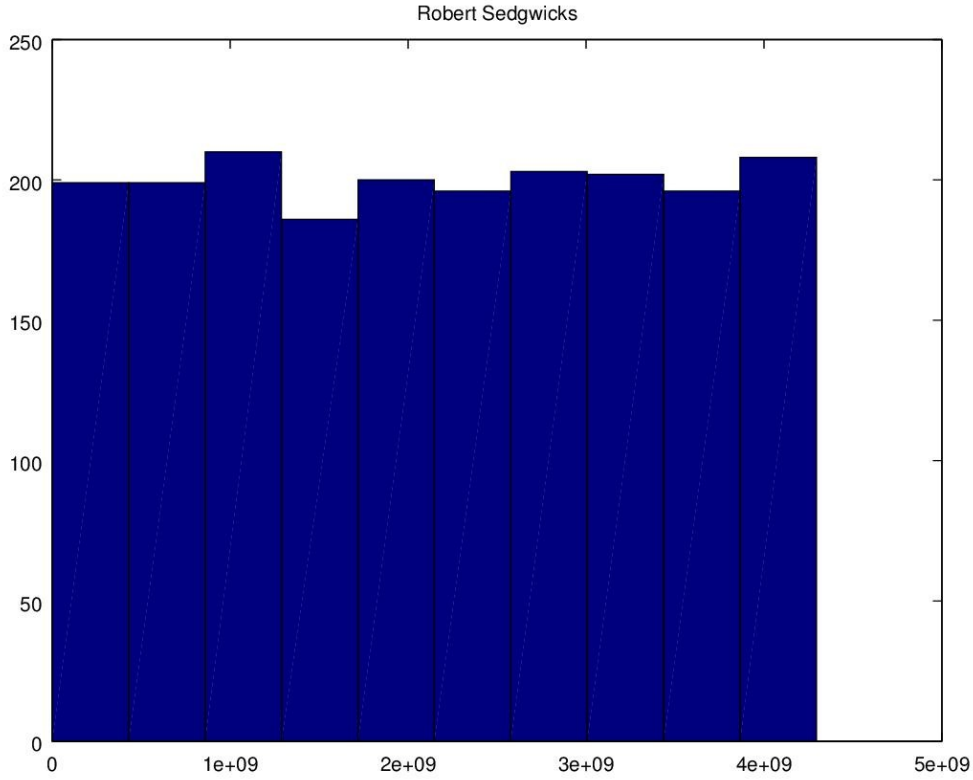
## Experimentos

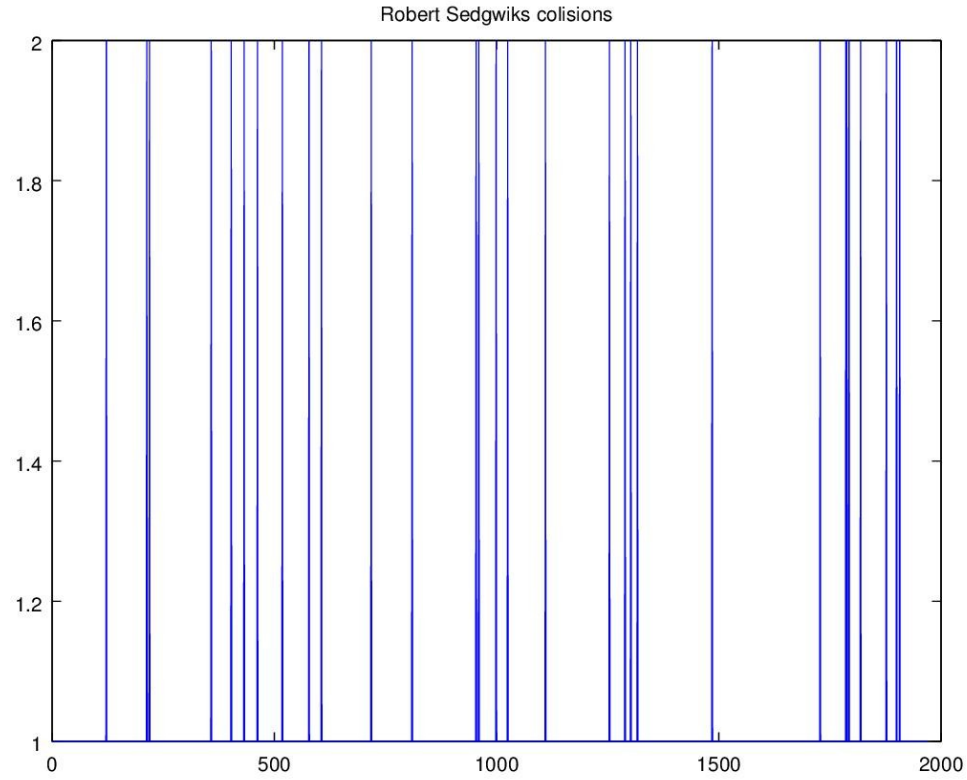
### Testes realizados

As entradas são cadeias de 20 caracteres aleatórios, para 2000 entradas são realizados os cálculos hash em todas as funções, posteriormente são gerados gráficos de distribuição (histograma) por valor e por colisão.

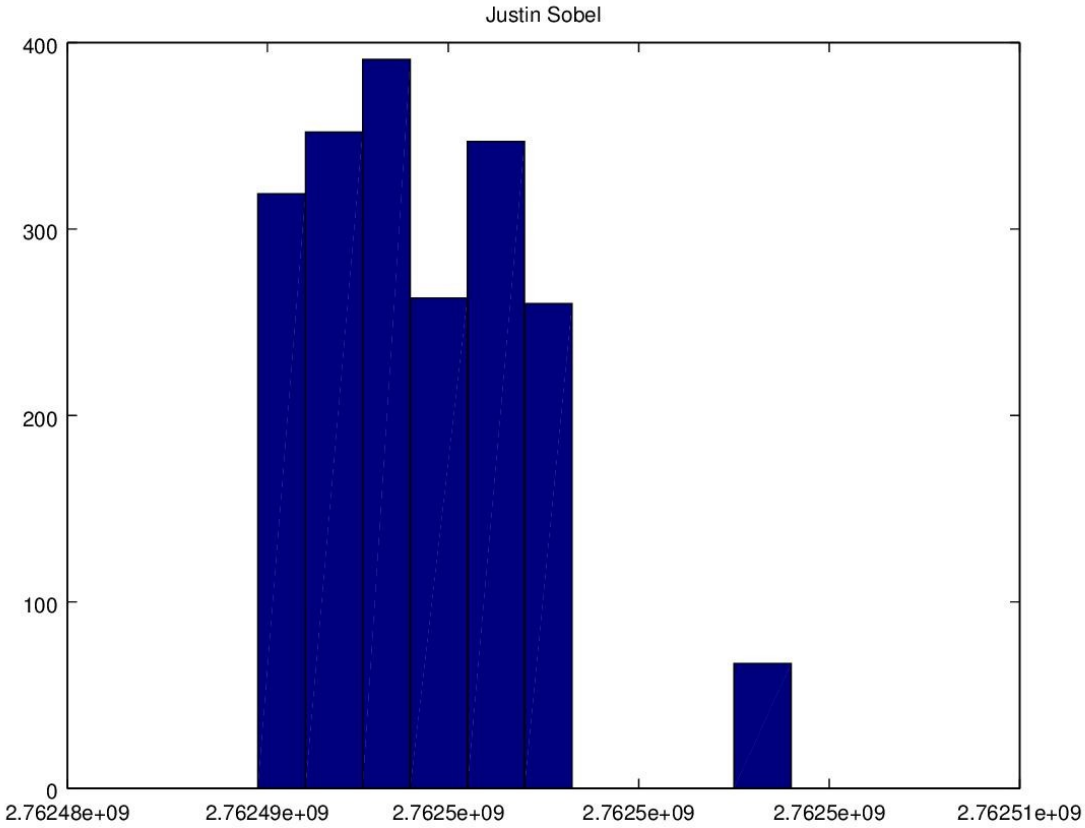
## Resultados

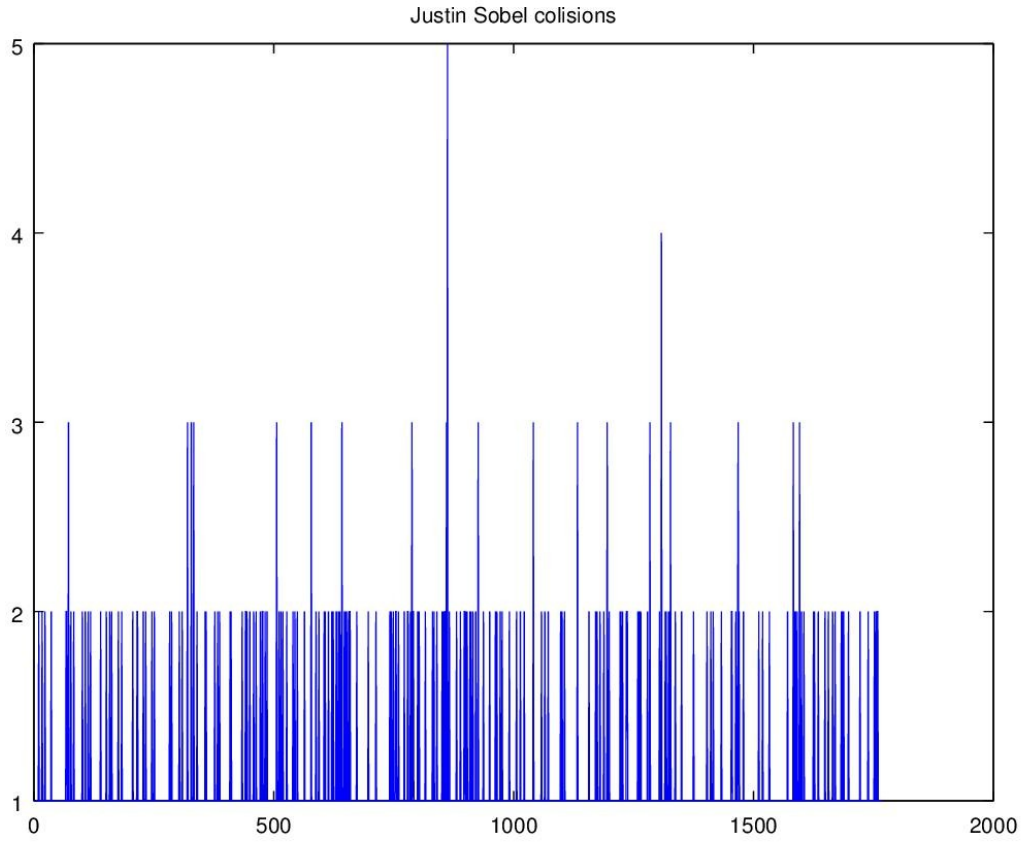
## A função hash de Robert Sedgwicks (Algorithms in C book)



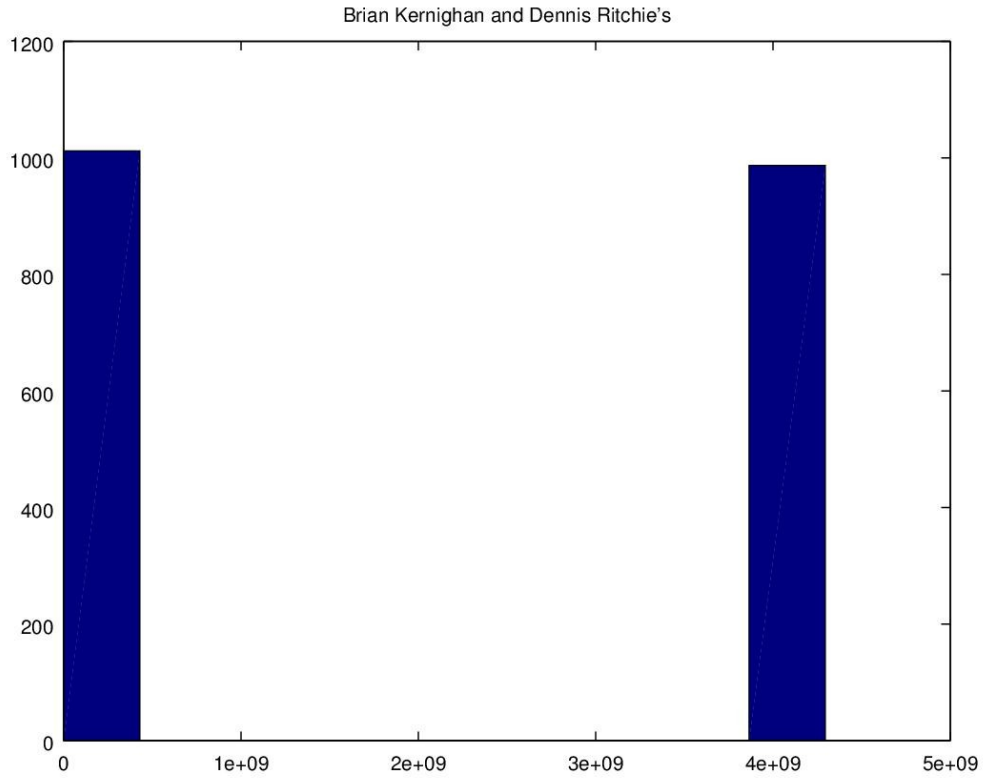



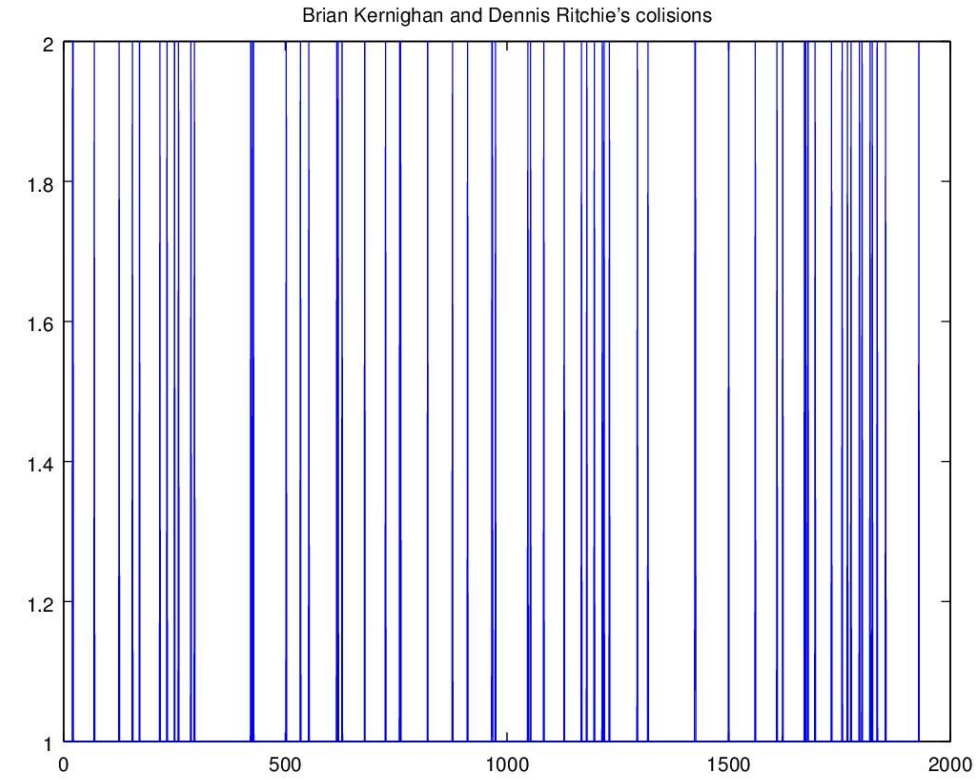
## A função hash de Justin Sobel



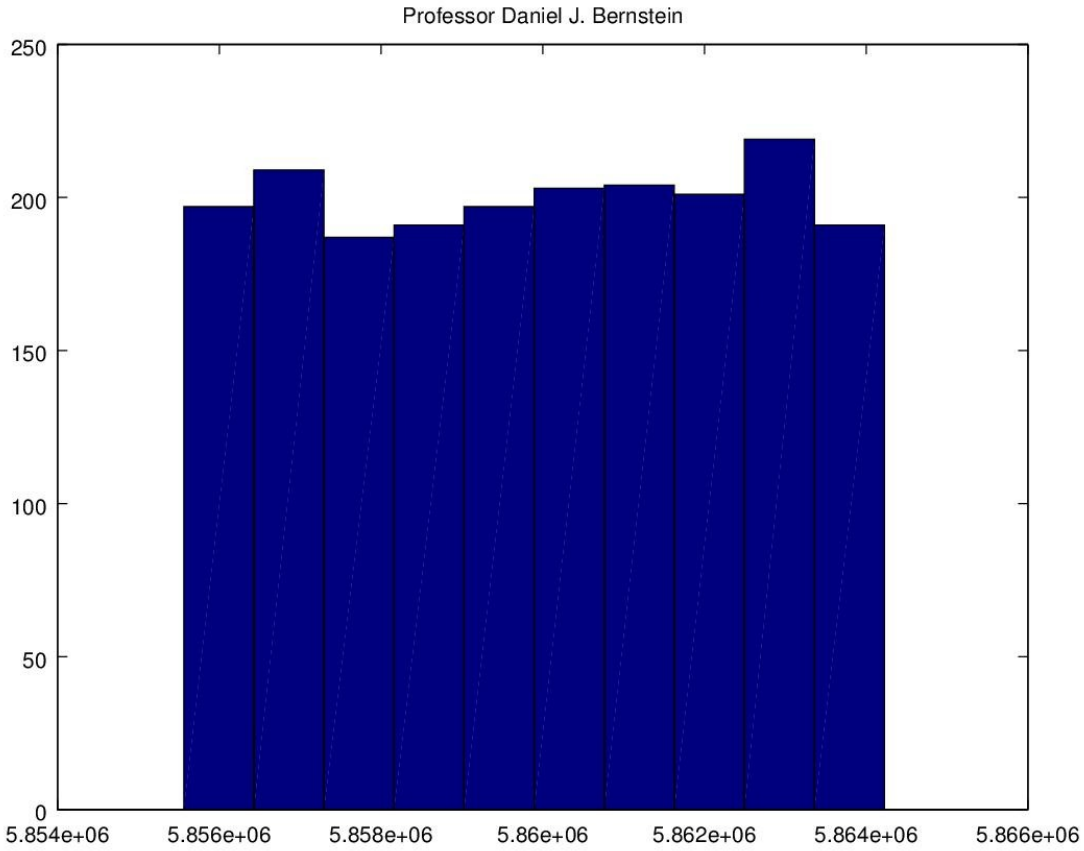



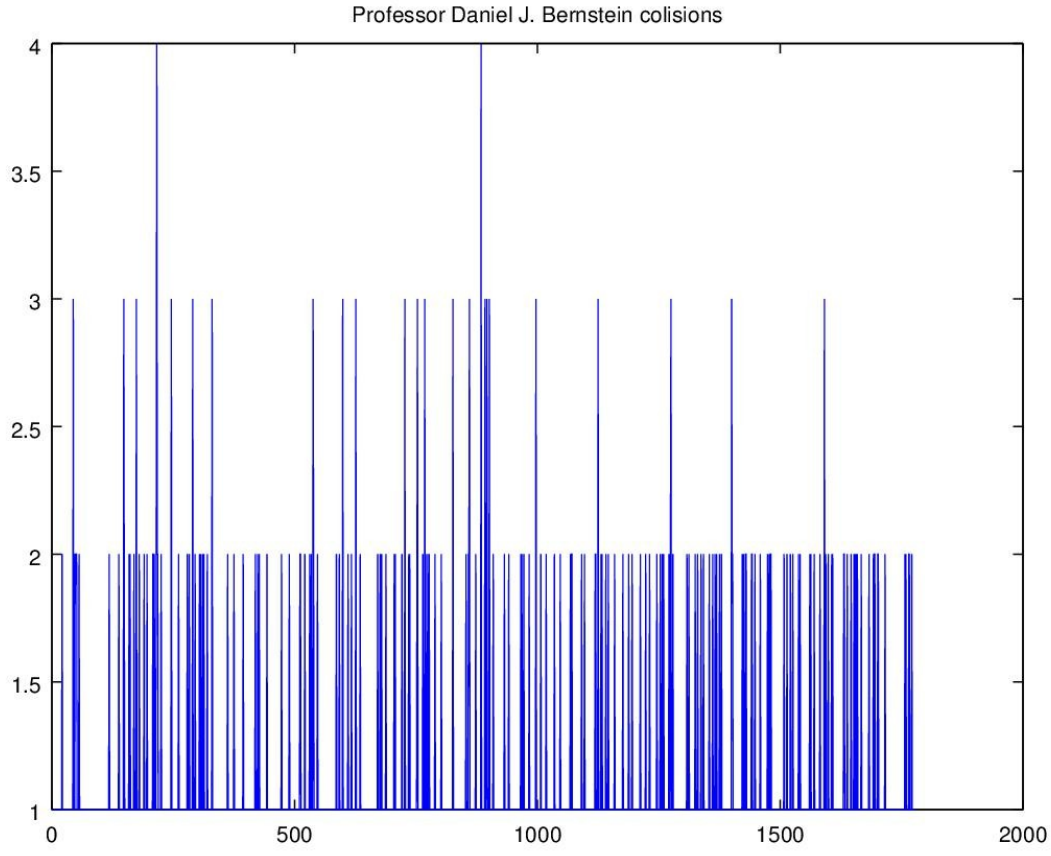
## A função hash de Brian Kernighan and Dennis Ritchie's (The C Programming Language)



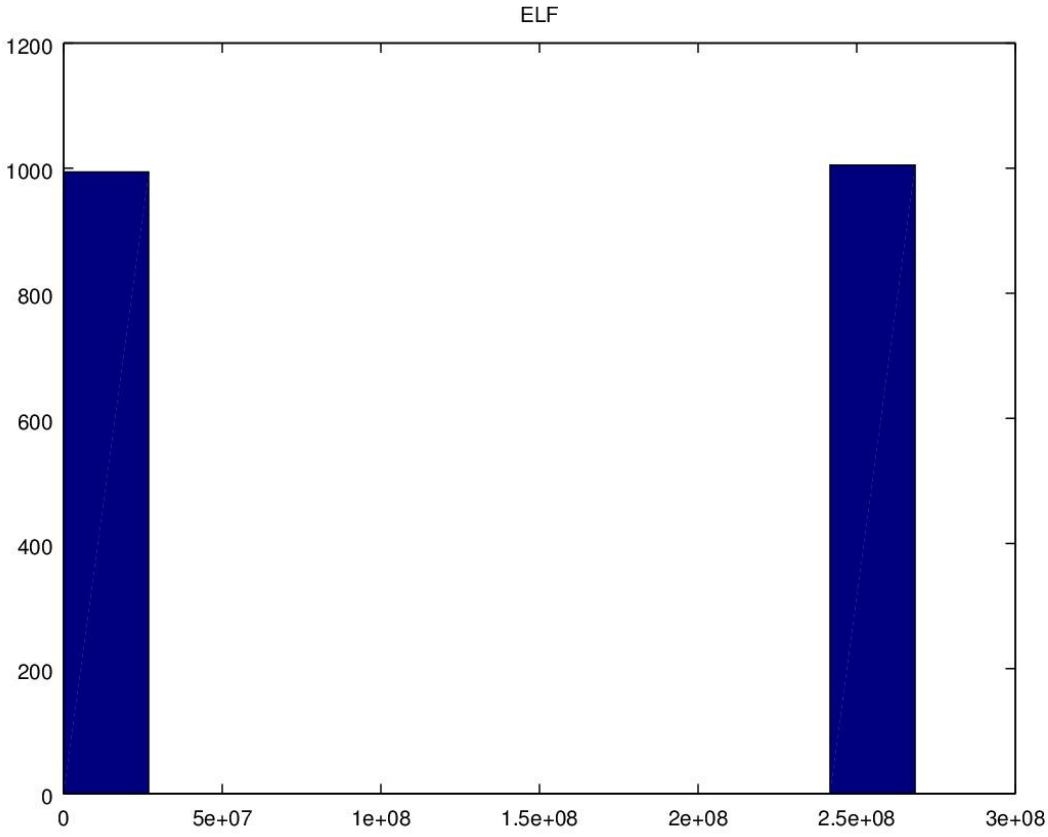



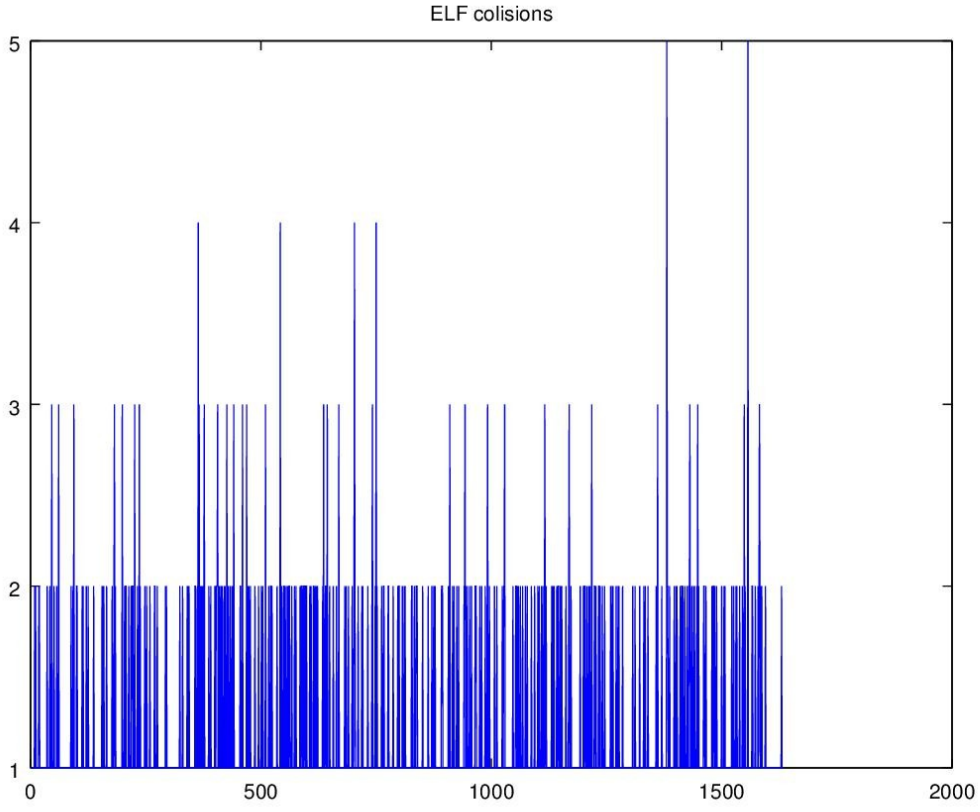
## A função hash de Professor Daniel J. Bernstein



## A função hash ELF



## Conclusão

As funções hash de Robert Sedgwicks e Professor Daniel J. Bernstein apresentam as melhores distribuições por valor. Algorítmos como ELF e  Brian Kernighan and Dennis Ritchie's geram valores hash em extremos opostos. Os menores números de colisões são encontrados nos algoritmos de  Robert Sedgwicks e  Brian Kernighan and Dennis Ritchie's. Apesar de não serem o que teve o menor número de colisões o ELF foi o algoritmo com o melhor espalhamento das entradas, o que o torna uma boa alternativa para tabelas de indexação esparsa, mas não para tabelas comuns dado espalhamento dos valores resultantes da função hash.

Referências Bibliográficas


- http://www.partow.net/programming/hashfunctions/#AvailableHashFunctions, Acessado em 31/10/2015
- Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein, *Algoritmos Teoria e PráticaI*, 2ª ed.
- Nivio Ziviane, *Projeto de Algoritmos com implementações em Java e C++*
- Robert Sedgewick and Kevin Wayne, *Algorithms*, 4th Edition
- Brian Kernighan and Dennis Ritchie's, *The C Programming Language*, 2ª ed