Algoritmos e Estrutura de Dados 2
Lab 1 - Hash Table

Jurandir de Mattos Jardim Neto
Davidson Oliveira Rodrigues
Rafael Zambianco

Índice

Estrutura do código

O programa basicamente se divide em três elementos (main, classe de tabela hash e uma classe de lista encadeada simples). Na classe main temos todo o código necessário por iniciar, gerenciar e finalizar o programa através de um menu interativo. Na classe de tabela hash temos todos os métodos relacionados aos processos de inserção, exclusão e impressão dos elementos que usamos para gerar os histogramas. A classe de lista encadeada simples nada mais é do que uma célula que aceita um valor string e contém um ponteiro para o próximo elemento da lista. Segue abaixo a declaração dos métodos e comentários sobre a classe de tabela hash:

HashTable(int size);
- Construtor da classe que recebe como parâmetro o tamanho da lista de endereços da tabela.

int m;
- Variável que armazena o valor do tamanho da tabela hash.

double c;
- Variável que guarda o valor da razão áurea.

Les **les;
- Um ponteiro para ponteiro, que referencia a lista de endereços na memória.

int *qtdPosicao;
- Um array para poder guardar a quantidade de elementos em uma dada posição, é utilizado para poder gerar o histograma.

void addWord(string);
- Uma das funções principais que adiciona uma string em uma tabela hash convertendo seu valor de ASCII através da função hash implementada, nessa função será usada a abordagem da soma dos caracteres ASCII.

void addWord_2(string);
- O mesmo de addWord, porém utiliza a abordagem da multiplicação dos valores dos caracteres ASCII.

void imprime();
- Usado para testes, imprime a tabela hash e seus elementos na tela.

void imprimeTxt(string fileName);
- Usado para imprimir em um arquivo txt a tabela hash e seus elementos, recebe como parâmetro o nome do arquivo a ser gerado.

void imprimeHistoGrama(string fileName);
- Irá gerar um arquivo com valores que representam o histograma, nele os valores serão do tipo:
Elementos na celula, quantidade de endereços

Com isso podemos descobrir quantos endereços possuem a quantidade de elementos denotados pelo valor antes da virgula, ex:
0,300
1,200
Uma saída com esses valores indica que na tabela 300 endereços tem 0 elementos e 200 endereços tem 1 elemento.

void imprimePositionValue(string);
- Imprime a posição de um dado elemento

void removeWord(string word);
- Remove uma palavra da lista quando encontrada

int qtdLinha(Les *celula);
- Calcula a quantidade de elemento dentro da Lista encadeada passada como parâmetro.
int getIndiceValue(string);
- Retorna o indice na qual a palavra procurada deveria se encontrar.

int calcValueWord(string);
- Calcula o valor da string para ser usado na função hash, utiliza a abordagem de soma.

int calcValueWord_2(string);
- Calcula o valor da string para ser usado na função hash, utiliza a abordagem de multiplicação.

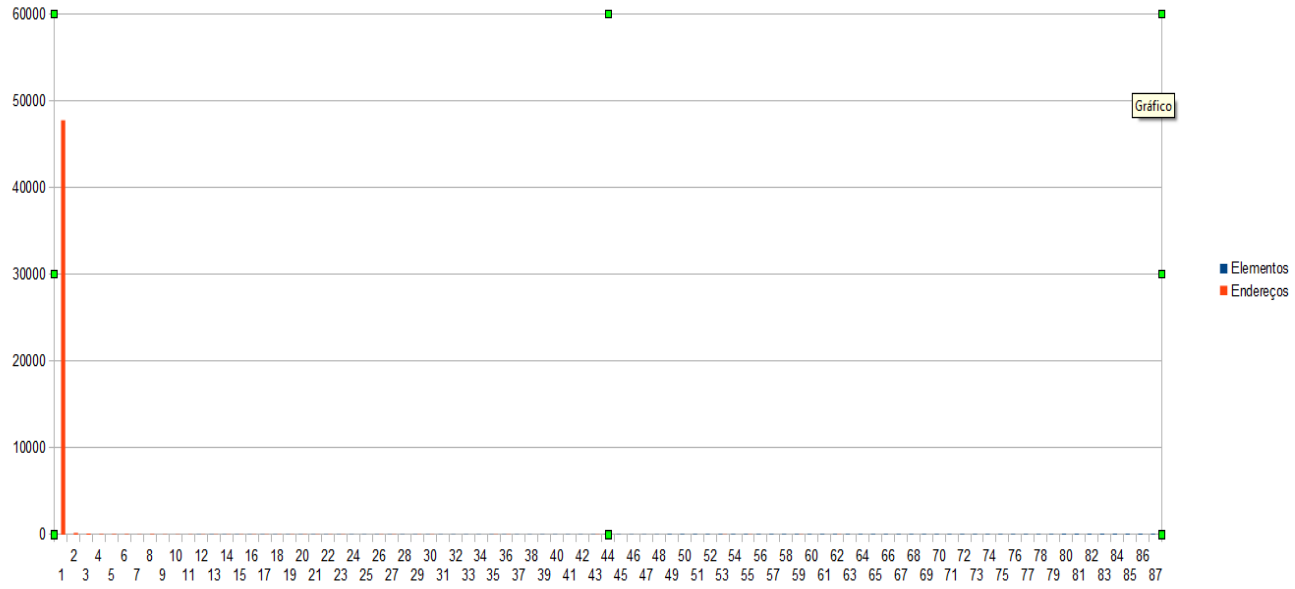
int functionHash(int);
- Aplica a função de hash utilizando a abordagem de soma.

unsigned long functionHash_2(int);
- Aplica a função de hash utilizando a abordagem de multiplicação.

Análise de histogramas

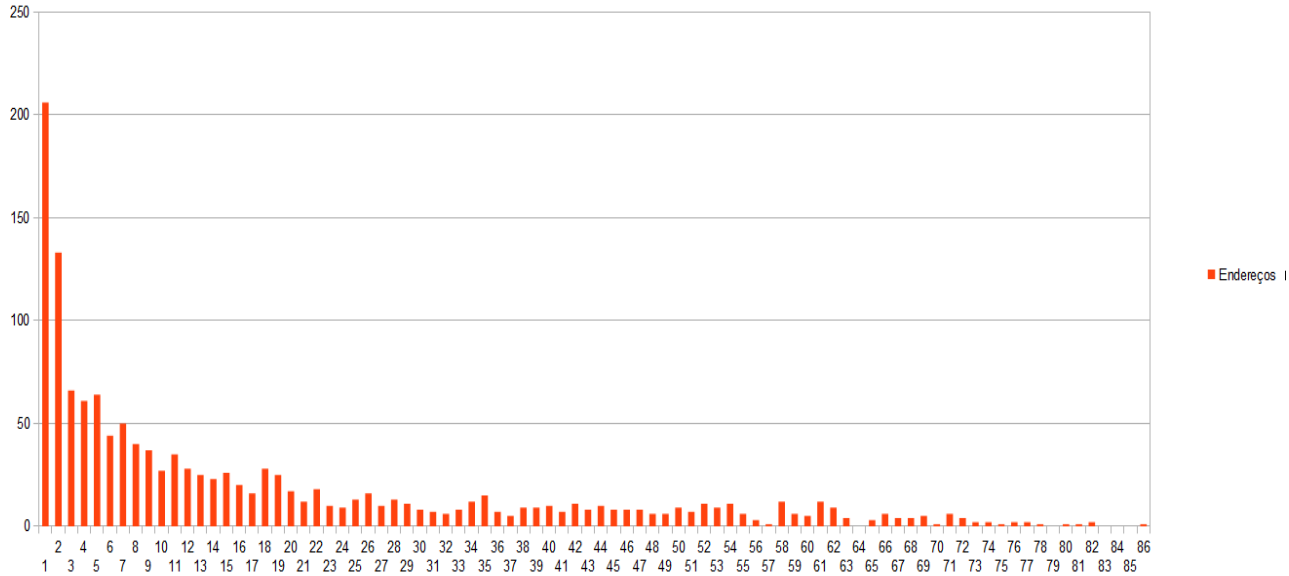
Utilizando a função dada nos slides, foi gerado um histograma com o intuito de mostrar como ficou a distribuição dos elementos nos endereços da tabela hash, nos primeiros teste foi notado que muitos endereços ficaram vazios, tanto para o item A quanto para o item B.

Histograma Item A (considerando endereços vazios)



A imagem acima mostra que a distribuição não foi muito igual, pois a maioria dos endereços ficaram sem elementos, além disso, ela esconde os resultados de todos os demais endereços. No total foram 47753 índices sem nenhuma palavra inserida. Se considerarmos que o número de palavras era supeior a 25000 então podemos dizer que tivemos um número enorme de colisões. Na figura abaixo apresentamos o mesmo gráfico, pórem desconsiderando os endeços vazios, assim é possível ver o resultado dos endereços que estavam escondidos.

Histograma Item A (desconsiderando endereços vazios)

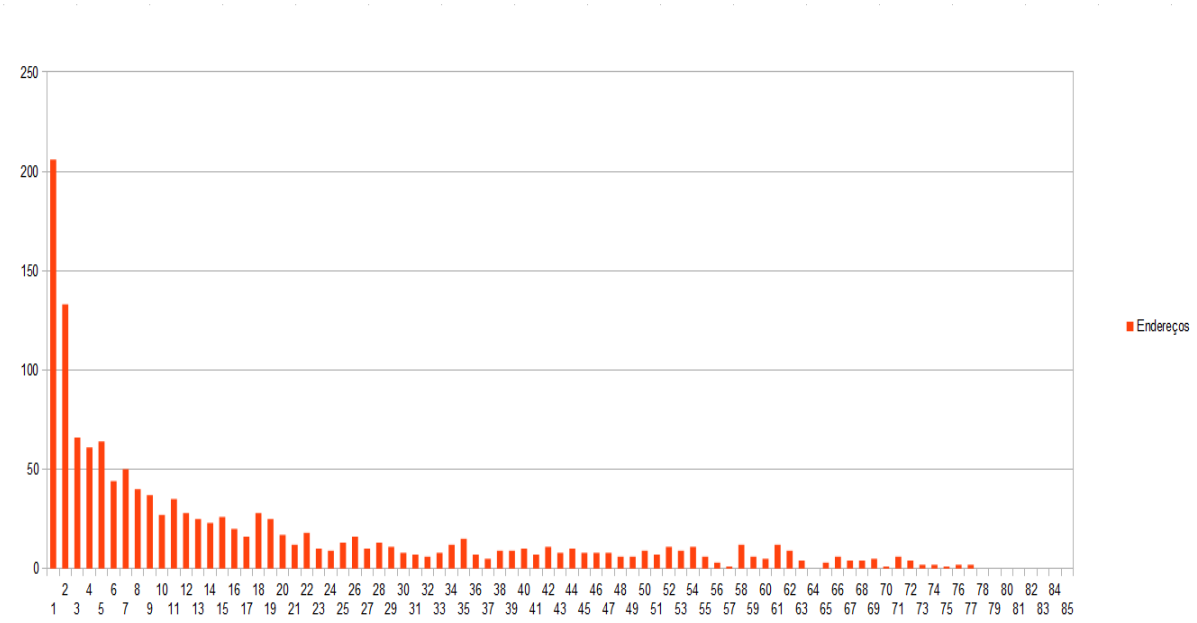


No gráfico acima é possível ver como ficou a distribuição com os outros endereços, podemos notar que ao ignorar os endereços vazios a distruição teve mais endereços com 1 elemento, seguido por uma cadeia quase que decrescente de enderços com 2 e mais elementos. De qualquer maneira não podemos considerar esses resultados, pois foram excluidos os enderços vazios. Esta é apenas uma visualização melhor das coisas.

Histograma Item B (considerando endereços vazios)

O mesmo acontece para o item B que considera um M de 12289. Novamente não é possível visualizar os outros endeços, pois a carga resultou em muitos endereços com 0 elementos. Na figura abaixo é realizado o mesmo procedimento anterior (retiramos os resultados de endereços vazios), para conseguirmos uma visão das mudanças.

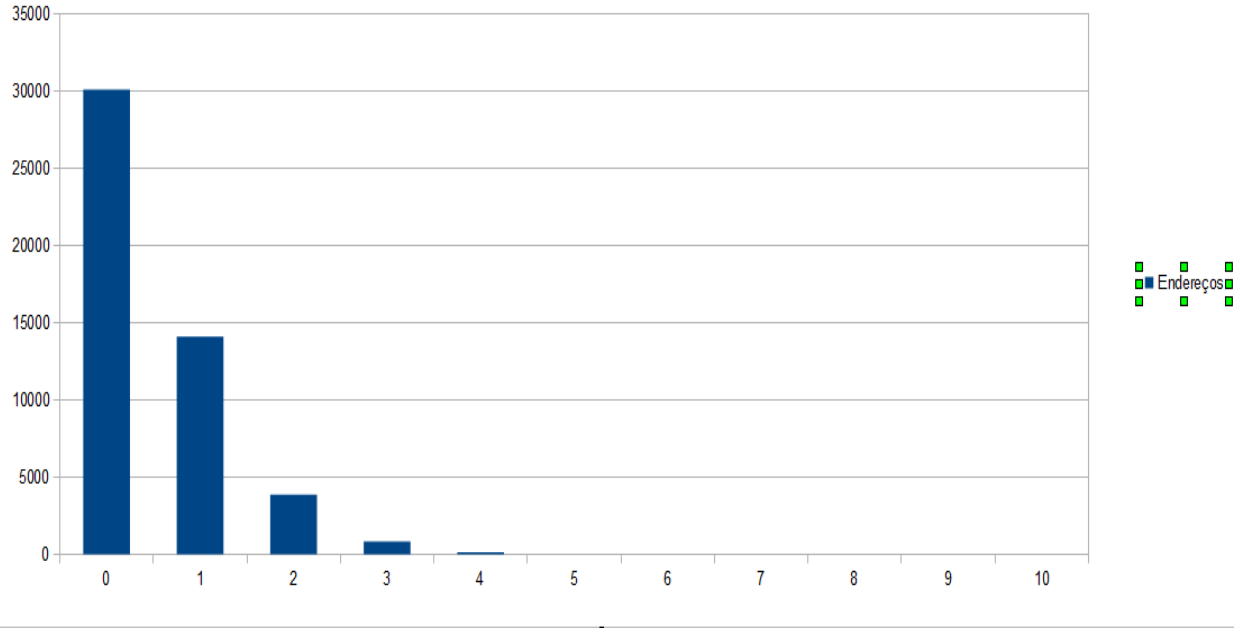
Histograma Item B (desconsiderando endereços vazios)



Não houve muita diferença com relação ao item anterior, apenas um leve aumento dos endereços com elementos. Esse tipo de problema está associado com a função que gera o valor correspondente a palavra lida, essa função calcula os valores individuais das letras e os soma seus caracteres ASCII. A seguir veremos uma outra função que diferente da primeira multiplica o valor dos caracteres. O problema da abordagem da soma deve-se ao valor pequeno que resultava da soma dos caracteres se comparados com os mesmo sendo multiplicados, logo o alcance da segunda abordagem é muito maior gerando um domínio de valores mais diversificado em comparação a primeira abordagem.

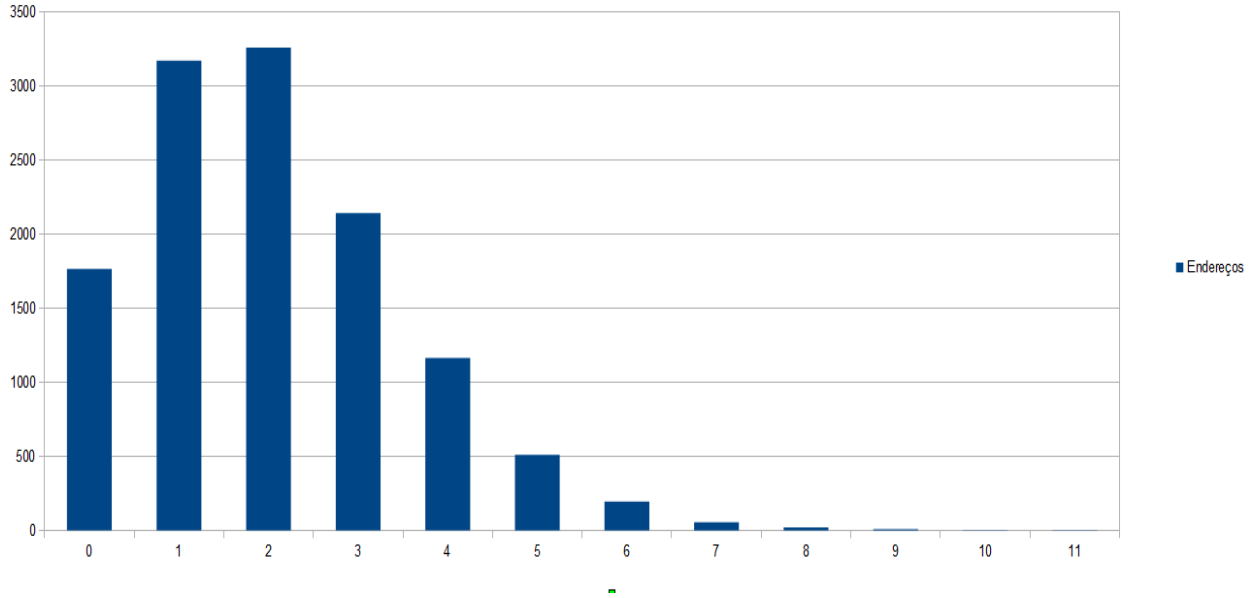
As diferenças entre as duas abordagens podem ser vistas nos graficos a seguir:

Histograma Item A (nova função de cálculo de palavra)



Uma grande mudança nos resultados do histograma ocorre quando é alterada a função de hash, pode-se notar que endereços com 0 e 1 elementos ficaram mais proximos seguidos pelos endereços de 2 e 3 elementos, implicando em uma melhor distribuição dos elementos e consequentemente menor numero de colisoes com poucos elementos, pois praticamente não houve quantidade de colisões maiores do que 5 ou 6 elementos.

Histograma Item B (nova função de cálculo de palavra)



Nesse caso (considerando m = 12289) houve um maior números de colisões envolvendo mais de 2 elementos, isso era esperado, pois se o tamanho da tabela foi reduzido para 12289. Por consequência, houve uma queda grande na quantidade de endereços vazios e mais colisões ocorreram.

Conclusão

O trabalho mostrou as vantagens de se trabalhar com o conceito de hashing table, paralelamente mostrando os prós e contras da técnica. Ficou evidente que a escolha da função de hashing e que uma simples mudança de operção em seu calculo pode resultar no sucesso ou não da performance da aplicação.

Foi mostrado também que, aliando-se técnicas de matemática com computação pode-se superar os limites da mesma, isto ficou evidente quando percebemos que o uso de funções matemáticas de dispersão são fortes candidatas a funções de hashing.