ALGORITMO E ESTRUTURA DE DADOS II

Tabelas Hash – LAB1

Aluno: Gabriel Nobrega de Lima
E-mail: gabriel.nobrega.lima@gmail.com
Professor: David Correa Martins Jr

Índice

Desenvolvimento do Projeto

O projeto desenvolvido teve como objetivo a criação de tabelas hash com encadeamento externo. Os esforços iniciais se concentraram no desenvolvimento de uma pequena biblioteca que implementasse as estruturas de dados necessárias. Um simples gerenciador de menus também foi desenvolvido. Ele modulariza as funções de cada menu em classes distinta e facilita a manutenção do software.

Como um dos requisitos do exercício é a programação de boa qualidade, foram adotadas as práticas e padrões de projetos contidas no livro "Use a cabeça – Padrões de Projeto". Em linhas gerais procurou-se separar módulos variáveis dos módulos estáticos, programar orientado a interfaces(classes puramente abstratas) e o uso de padrões de projeto, como o Strategy.

Lista Encadeada

A classe que implementa as listas encadeadas foi nomeada LinkedList, e pode ser localizada em .../src/hashlib/LinkedList.cpp e .../src/hashlib/LinkedList.h. Esta estrutura utiliza para cada nó a classe Node, localizada no mesmo diretório.

Tabela Hash

A classe que implementa a Tabela Hash foi nomeada HashTable, e pode ser localizada em .../src/hashlib/HashTable.cpp e .../src/hashlib/HashTable.h. A classe possui um vetor de ponteiros para a estrutura LinkedList, sendo cada um dos indices um endereço base da tabela.

O problema da refatoração da função de hash

Os requisitos iniciais do projeto previam apenas um tipo de função de hash. Contudo, os requisitos foram alterados e o projeto passou a suportar dois tipos de funções de hash. No desenvolvimento de software a mudança de requisitos estáticos para variáveis geralmente é problemática e devem ser solucionados separando imediatamente as características que se tornaram variáveis das que continuam estáticas. Pensando nisto foi utilizado o padrão de projeto Strategy, de maneira que se possa incluir infinitas funções de hash sem a necessidade da alteração do código da biblioteca.

Por exemplo, as funções de hash podem ser intercambiadas atualmente através de uma função set:

```
hashTable->setHashFunction(new WordSumKeyCalc());

hashTable->setHashFunction(new WordMultKeyCalc());
```

Caso seja necessário incluir o suporte de outras funções de hash, será necessário apenas a adição da classe que descreve a função de hash. Por exemplo, suponha que tenha sido incluido a necessidade de implementar uma função de hash

WordModuleKeyCalc previamente não prevista. Ela estaria disponível para uso apenas informando a biblioteca:

hashTable->setHashFunction(new WordModuleKeyCalc());

Esta padrão de projeto permite separar o que é variável do que é estático, fazendo com que o projeto seja totalmente expansivel.

Tabelas Hash I/O

As operações de escrita e leitura de Tabelas Hash em arquivos de texto foram implementadas separadamente na classe HashFile que extende a classe HashTable. Esta decisão foi tomada baseando-se no fato de que não existem padrões de arquivos globais para leitura e escrita de Tabelas Hash em disco. Caso futuramente deseje utilizar arquivos de Tabelas Hash em outros formatos não será necessário modificar a estrutura principal, apenas extende-la. Isto minimiza problemas na manutenção de software. A Classe pode ser encontrada em .../src/hashlib/HashFile.cpp e .../src/hashlib/HashFile.h .

Histograma

A classe Histogram implementa a politica de um Histograma e pode ser localizada em .../src/hashlib/Histogram.cpp e .../src/hashlib/Histogram.h .

Histogramas de Tabelas Hash

A classe HashHistogram implementa especificamente histogramas para Tabelas Hash considerando o formato de arquivo definido no enunciado do trabalho. Ela extende Histogram e realiza uma composição com HashTable. Tal decisão foi realizada pois futuramente poderia ser interessante gravar histogramas de Tabela Hash com formatos diferentes. Separando o módulo de escrita da estrutura não será necessário alterar a estrutura de dados principal de Histogram. Isto minimiza problemas na manutenção de software. A composição com HashTable permite por sua vez que uma Tabela Hash possa ser passada como argumento e o histograma gerado transparentemente para o usuário.

A classe HashHistogram está localizada em ../src/hashlib/HashHistogram.cpp e .../src/hashlib/HashHistogram.h .

Testes

O desenvolvimento das estruturas de dados supracitadas foi feita na ordem descrita. A conclusão de cada módulo foi seguida pela execução de testes unitários (quando necessários). Estes testes unitários utilizaram como ferramenta funções Assert. Elas indicam em que linha de código houve um resultado que não esperado, isto facilita a correção de erros.

Para padronizar os testes foi criada a classe puramente abstrata(interface) Test localizada em ../src/hashlib/Test.h que força todo e qualquer teste a implementar o método test(). Foi padronizado que toda classe de teste para um dado módulo teria uma classe com nome “Nome do módulo” e sufixo Test. As classes de teste podem ser visualizadas em .../src/hashlib/test/. Todas elas implementam a interface Test, e assim podem ser executadas polimorficamente.

A execução dos testes foi incluida na função “main” do projeto e podem ser ativadas com a simples definição da flag HASH_LIB_TEST.

```
#ifdef HASH_LIB_TEST

	// Sessão de testes
	Test *test;

	test = new LinkedListTest();
	test->execute();
	delete test;

	test = new HashTableTest();
	test->execute();
	delete test;

	test = new HashHistogramTest();
	test->execute();
	delete test;

	test = new HashFileTest();
	test->execute();
	delete test;
#endif
```

Os testes sobre as estruturas de dados também foram importantes para a correção de problemas de vazamento de memória, o qual serão comentados mais a frente.

Interface de usuário

Para o projeto foi criada uma interface baseada em console. O usuário visualiza as opções na tela e as escolhe com auxilio do teclado. Este tipo de interface é frequentemente implementada de maneira imperativa, dificultando a leitura e manutenção de código. Tentando evitar este problema foi desenvolvido um pequeno gerenciador de menus orientado a objetos seguindo o padrão de projeto *Strategy*.

O gerenciador controla a execução dos menus de maneira automática e permite que cada um seja inteiramente codificado em sua classe correspondente. O gerenciamento utiliza como base uma política de pilha de menus. O menu no topo é aquele sendo visualizado pelo usuário. Conforme o usuário acessa os menus, estes são empilhados. Quando o usuário deseja realizar um retrocesso ocorre um desempilhamento, fazendo com que o menu atual seja removido da memória e o anterior assuma o topo da pilha, tornando-se visível.

A arquitetura baseada em pilhas e no padrão de projeto *Strategy* facilita o desenvolvimento e manutenção de código. Cada menu pode ser executado e testado separadamente, não importando a ordem em que ele aparece no software final. Qualquer erro encontrado nos menus pode ser rapidamente modificado através de seu módulo correspondente. A ordem dos menus pode ser alterada com pouco esforço.

Para utilizar a arquitetura deve ser criado para cada menu uma classe que implementa a interface "Menu". Esta interface força o desenvolvedor implementar dois métodos de callback:

```
void show(Screen *parent, int code);
Menu* onOptionSelected(int userInput);
```

O método "show" é chamado pelo gerenciador quando o menu deve ser mostrado na tela. O argumento "parent" indica qual menu fez a requisição e "code" ajuda diferenciar chamadas em que um mesmo menu chama outro através de múltiplas opções. O método "onOptionSelected" é executando pelo gerenciador sempre que um usuário entrar com uma opção de menu. Desta maneira temos em "show" a modelagem da visão e em "onOptionSelected" o controle. O retorno do método "onOptionSelected" indica qual deve ser o próximo menu a ser mostrada pelo gerenciador.

O gerenciador pode ser utilizando extendendo-se a classe "Application" e definindo o menu principal. Uma simples chamada para o método "start" faz com que os menus sejam mostrados de acordo com definido pelo retorno do método "onOptionSelected" de cada menu.

***Vale ressaltar que a pilha utilizada no gerenciador de menus é uma implementação da biblioteca STL. A mesma foi utilizada por não envolver estruturas de dados relevantes ao trabalho, minimizando o tempo de desenvolvimento.**

Valgrind

Após obter um aplicativo estável foi utilizado o Valgrind para encontrar possíveis problemas de vazamento de memória. Com uma exaustiva série de correções foi possivel tornar a aplicação isenta de vazamento de memória. Todas as etapas utilizadas para produzir as respostas dos exercícios foram realizadas com a supervisão do Valgrind. Após execução das tarefas A), B), C), D) e E) da seção "Resposta dos Exercícios" o Valgring gerou em seu log o seguinte fragmento:

```
==3289==
==3289== HEAP SUMMARY:
==3289== in use at exit: 0 bytes in 0 blocks
==3289== total heap usage: 188,969 allocs, 188,970 frees, 3,321,928 bytes allocated
==3289==
```

==3289== All heap blocks were freed -- no leaks are possible

Documentação

As estruturas de dados e o gerenciador de menus foram separados do restante do projeto como se fossem bibliotecas. Elas permitem a resolução dos exercícios propostos em sala, mas também podem ser reaproveitadas em outros projetos. Pensando no reaproveitamento de código, todos as classes pertencentes a biblioteca foram documentadas com o apoio do doxygen. Esta documentação pode ser acessada através do diretório .../doc/index.html.

Compilando e executando o projeto

O projeto foi desenvolvido em linguagem C++, sendo assim será necessário a utilização do compilador g++. Tente executar o g++ pelo terminal, caso este não seja localizado será necessário instala-lo. Nas distribuições derivadas do Debian (por ex:Ubuntu), a instalação pode ser realizada através do comando:

sudo apt-get install g++

Com o compilador instalado o projeto pode ser compilado com o comando make dentro do diretório src do projeto.

.../HashApp/src$ make

O comando irá gerar o binário hashapp dentro do diretório src. Para executa-lo utilize o comando:

./hashapp

O menu principal irá ser apresentado, como mostra abaixo:

<<HashApp>>
----
1)Criar tabela hash
0)Sair
----
*->**1***

Dicas para o estudo do código fonte

Para o estudo do código fonte é aconselhavel que o desenvolvedor se concentre inicialmente nas classes nomeadas com sufixo Menu contidas no diretório .../src. Cada um destes menus representa uma tela apresentada no software e realiza chamadas simples às bibliotecas de estrutura de dados. Elas irão passar a noção geral do software, enquanto que o aprofundamento deverá ser realizado com o estudos dos

códigos contidos na biblioteca de estrutura de dados no diretório .../src/hashlib . Você pode utilizar a documentação gerada através do doxygen para orientar sua análise.

Resposta dos Exercícios

Item A) Para criar uma tabela hash a partir do arquivo de palavras "word.txt" contido no diretório …/src siga os passos descritos abaixo:

./hashapp

<<HashApp>>
----
1)Criar tabela hash
0)Sair
----
*->**1***

*Numero de chaves:**49157***

Tabela hash criada...

<<Funcao de Hash>>
----
1)Soma de caracteres
2)Multiplicacao de caracteres
----
*->**1***

Funcao de hash considerando a soma de caracteres ativada...

<<HashApp>>
----
1)Criar tabela hash
2)Carregar tabela hash a partir de um arquivo
3)Salvar tabela hash
4)Gerar histograma
5)Localizar palavras
6)Editar
0)Sair
----
*->**2***
*Caminho do arquivo:**words.txt***

words.txt carregado com sucesso...
Numero de enderecos:49157

Numero de palavras:25143

Para escrever a tabela hash no arquivo "tabelaHash.txt" no diretório .../src sigas os passos a seguir:

<<HashApp>>
----
1)Criar tabela hash
2)Carregar tabela hash a partir de um arquivo
3)Salvar tabela hash
4)Gerar histograma
5)Localizar palavras
6)Editar
0)Sair
----
*->**3***
*Caminho do arquivo:**tabelaHash.txt***

Tabela hash com 25143 palavras salvo com sucesso...

Análise dos resultados

Os requisitos para o item foram implementados. O arquivo "words.txt" contêm um total de 25143 palavras . A tabela hash de encadeamento externo foi criada com 49157 endereços base. O fator de carga fc=(25143/49157) = 0,5115, ou seja, caso a função seja uniforme espera-se que em média existam 0,51 elementos em cada endereço base. Isto mostra que se a função de hash for uniforme deverão existir um grande número de endereços base com apenas 1 elemento e um número ainda maior endereços base vazios. Tal análise será melhor realizada após a obtenção do histograma, no item B.

Observações

A sintaxe da tabela hash incluiu em cada linha a string inicial "Linha n:", onde n é o número da linha(ou endereço base). Esta inclusão é util pois em alguns editores de texto existe uma limitação do número de caracteres por linha. Quando o número de caracteres ultrapassa o limite, o editor realiza uma quebra de linha automática. Isto pode dificultar a visualização da real linha que cada registro assumiu.

O arquivo "words.txt" foi gerado com o sistema operacional windows, e por isso a quebra de linha é definida pela marcação \r\n e não somente \n. Isto é problemático no sentido de que um software para linux geralmente so irá receber arquivos de texto produzidos no próprio sistema operacional. A função getline da implementação linux encontra o final de linha ao achar o caractere '\n', enquanto no windows \r\n. Sendo assim, todas as palavras obtidas com getline do arquivo "words.txt" carregariam o caractere '\r' no seu final. O ideal seria converter o arquivo "words.txt" para o padrão

linux, porém, como isto iria ferir o enunciado do exercício foram incluidas verificações no método de carregamento para que o caractere '\r' fosse removido caso encontrado.

Item B) Para gerar o histograma no arquivo "histograma49157.txt" no diretório …/src continue a execução do programa no mesmo ponto onde o item A abordou. Siga os passos descritos abaixo:

<<HashApp>>
----
1)Criar tabela hash
2)Carregar tabela hash a partir de um arquivo
3)Salvar tabela hash
4)Gerar histograma

5)Localizar palavras
6)Editar
0)Sair
----
*->**4***

*Caminho do arquivo:**histograma49157.txt***

Histograma gravado com sucesso...

Para criar uma nova tabela hash com 12289 endereços base, basta seguir a mesma sequência de passos iniciando o programa do zero ou continuando após a geração do "histograma49157.txt". Basta escolher a opção criar tabela hash. Ela irá perguntar o número de endereços bases e a função de hash de sua escolha (escolha a função de hash considerando a soma). Vale ressaltar que quando uma nova tabela hash é criada, a tabela hash prévia tem sua memória totalmente liberada.

Análise dos resultados

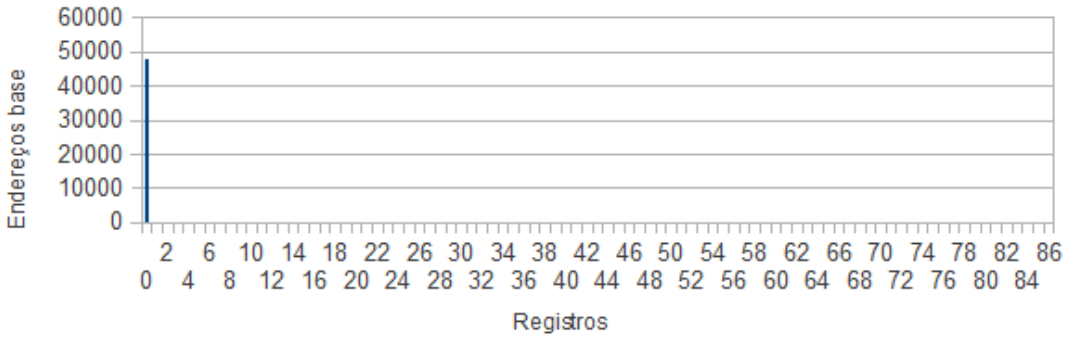
Gerando o histograma para a tabela hash com 49157 elementos base é possivel verificar que a função está longe de se aproximar de uma distribuição uniforme. Considerando o fator de carga calculado no item A), fc=0,5115 , o esperado era que o número de endereços sem qualquer registro fosse aproximadamente 29000 e o número de endereços com apenas um registro 15080. O histograma mostra uma realidade bem diferente, a grande maioria dos endereços base estão vazios, mais precisamente 47752 endereços não possuem qualquer registro armazenado. O número de endereços base com apenas 1 elemento é de apenas 207. Percebe-se então o grande número de colisões que a função de hash está gerando, visto que o número de registros totaliza 25143 .

O histograma obtido com a aplicação para o caso onde a tabela possui 49157 endereços pode ser visto na Tabela 1 e no Gráfico 1.

Endereco com 0 registros: 47753	Endereco com 44 registros: 10
Endereco com 1 registros: 206	Endereco com 45 registros: 8
Endereco com 2 registros: 133	Endereco com 46 registros: 8
Endereco com 3 registros: 66	Endereco com 47 registros: 8
Endereco com 4 registros: 61	Endereco com 48 registros: 6
Endereco com 5 registros: 64	Endereco com 49 registros: 6
Endereco com 6 registros: 44	Endereco com 50 registros: 9
Endereco com 7 registros: 50	Endereco com 51 registros: 7
Endereco com 8 registros: 40	Endereco com 52 registros: 11
Endereco com 9 registros: 37	Endereco com 53 registros: 9
Endereco com 10 registros: 27	Endereco com 54 registros: 11
Endereco com 11 registros: 35	Endereco com 55 registros: 6
Endereco com 12 registros: 28	Endereco com 56 registros: 3
Endereco com 13 registros: 25	Endereco com 57 registros: 1
Endereco com 14 registros: 23	Endereco com 58 registros: 12
Endereco com 15 registros: 26	Endereco com 59 registros: 6
Endereco com 16 registros: 20	Endereco com 60 registros: 5
Endereco com 17 registros: 16	Endereco com 61 registros: 12
Endereco com 18 registros: 28	Endereco com 62 registros: 9
Endereco com 19 registros: 26	Endereco com 63 registros: 4
Endereco com 20 registros: 16	Endereco com 64 registros: 0
Endereco com 21 registros: 12	Endereco com 65 registros: 3
Endereco com 22 registros: 18	Endereco com 66 registros: 6
Endereco com 23 registros: 10	Endereco com 67 registros: 4
Endereco com 24 registros: 9	Endereco com 68 registros: 4
Endereco com 25 registros: 13	Endereco com 69 registros: 5
Endereco com 26 registros: 16	Endereco com 70 registros: 1
Endereco com 27 registros: 10	Endereco com 71 registros: 6
Endereco com 28 registros: 13	Endereco com 72 registros: 4
Endereco com 29 registros: 11	Endereco com 73 registros: 2
Endereco com 30 registros: 8	Endereco com 74 registros: 2
Endereco com 31 registros: 7	Endereco com 75 registros: 1
Endereco com 32 registros: 6	Endereco com 76 registros: 2
Endereco com 33 registros: 8	Endereco com 77 registros: 2
Endereco com 34 registros: 12	Endereco com 78 registros: 1
Endereco com 35 registros: 15	Endereco com 79 registros: 0
Endereco com 36 registros: 7	Endereco com 80 registros: 1
Endereco com 37 registros: 5	Endereco com 81 registros: 1
Endereco com 38 registros: 9	Endereco com 82 registros: 2
Endereco com 39 registros: 9	Endereco com 83 registros: 0
Endereco com 40 registros: 10	Endereco com 84 registros: 0
Endereco com 41 registros: 7	Endereco com 85 registros: 0
Endereco com 42 registros: 11	Endereco com 86 registros: 1
Endereco com 43 registros: 8	**TOTAL: *25143 palavras***

Tabela 1 - Histograma para a função de hash considerando a soma de caracteres e tabela com 49157 endereços base.

Gráfico 1 – Histograma da tabela com 49157 endereços base utilizando função de hash de soma.



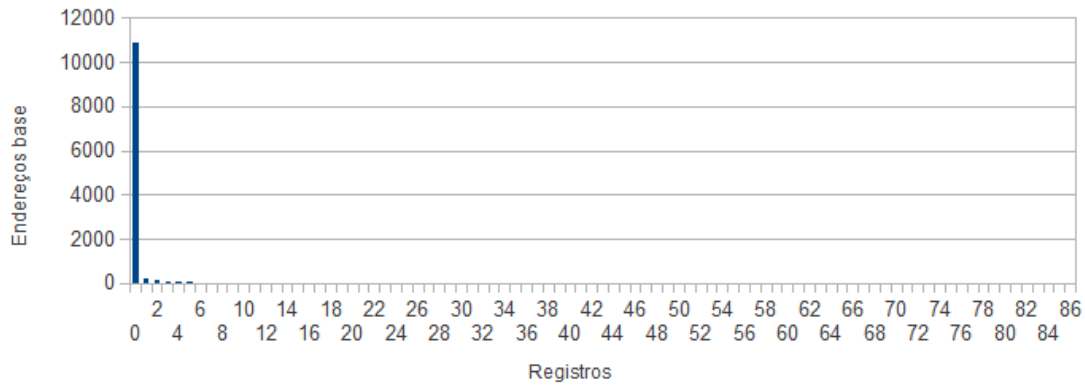
Considerando uma tabela hash com 12289 endereços base e um total de 25143 registros, temos um fator de carga fc=(25143/12289)=2,046. Percebemos no Gráfico 2 que a função de hash não possui um comportamento uniforme e com um fator de carga ainda maior, a tendencia é a ocorrência de um maior número de colisões. Considerando uma distribuição uniforme esperamos que o número de endereços vazios seja da ordem de 1588 e o número de endereços com apenas um registro da ordem de 3250.

Analisando o histograma na Tabela 2 e o Gráfico 2 pode-se perceber um resultado semelhante ao gerado na Tabela 1. Isto é um forte indicio de que a função hash utilizada não é adequada para o problema que estamos tratando. Ela não possui um comportamento que se aproxima do uniforme.

Endereco com 0 registros: 10885	Endereco com 44 registros: 10
Endereco com 1 registros: 206	Endereco com 45 registros: 8
Endereco com 2 registros: 133	Endereco com 46 registros: 8
Endereco com 3 registros: 66	Endereco com 47 registros: 8
Endereco com 4 registros: 61	Endereco com 48 registros: 6
Endereco com 5 registros: 64	Endereco com 49 registros: 6
Endereco com 6 registros: 44	Endereco com 50 registros: 9
Endereco com 7 registros: 50	Endereco com 51 registros: 7
Endereco com 8 registros: 40	Endereco com 52 registros: 11
Endereco com 9 registros: 37	Endereco com 53 registros: 9
Endereco com 10 registros: 27	Endereco com 54 registros: 11
Endereco com 11 registros: 35	Endereco com 55 registros: 6
Endereco com 12 registros: 28	Endereco com 56 registros: 3
Endereco com 13 registros: 25	Endereco com 57 registros: 1
Endereco com 14 registros: 23	Endereco com 58 registros: 12
Endereco com 15 registros: 26	Endereco com 59 registros: 6
Endereco com 16 registros: 20	Endereco com 60 registros: 5
Endereco com 17 registros: 16	Endereco com 61 registros: 12
Endereco com 18 registros: 28	Endereco com 62 registros: 9
Endereco com 19 registros: 26	Endereco com 63 registros: 4
Endereco com 20 registros: 16	Endereco com 64 registros: 0
Endereco com 21 registros: 12	Endereco com 65 registros: 3
Endereco com 22 registros: 18	Endereco com 66 registros: 6
Endereco com 23 registros: 10	Endereco com 67 registros: 4
Endereco com 24 registros: 9	Endereco com 68 registros: 4
Endereco com 25 registros: 13	Endereco com 69 registros: 5
Endereco com 26 registros: 16	Endereco com 70 registros: 1
Endereco com 27 registros: 10	Endereco com 71 registros: 6
Endereco com 28 registros: 13	Endereco com 72 registros: 4
Endereco com 29 registros: 11	Endereco com 73 registros: 2
Endereco com 30 registros: 8	Endereco com 74 registros: 2
Endereco com 31 registros: 7	Endereco com 75 registros: 1
Endereco com 32 registros: 6	Endereco com 76 registros: 2
Endereco com 33 registros: 8	Endereco com 77 registros: 2
Endereco com 34 registros: 12	Endereco com 78 registros: 1
Endereco com 35 registros: 15	Endereco com 79 registros: 0
Endereco com 36 registros: 7	Endereco com 80 registros: 1
Endereco com 37 registros: 5	Endereco com 81 registros: 1
Endereco com 38 registros: 9	Endereco com 82 registros: 2
Endereco com 39 registros: 9	Endereco com 83 registros: 0
Endereco com 40 registros: 10	Endereco com 84 registros: 0
Endereco com 41 registros: 7	Endereco com 85 registros: 0
Endereco com 42 registros: 11	Endereco com 86 registros: 1
Endereco com 43 registros: 8	**TOTAL: *25143 palavras***

Tabela 2 - Histograma para a função de hash considerando a soma de caracteres e tabela com 49157 endereços base.

Gráfico 2 - Histograma da tabela com 12289 endereços base utilizando função de hash de soma.



Os histogramas da Tabela 1 e 2 são semelhantes. O somatório dos caracteres das palavras contidas em “words.txt” não são suficientes para utilizar todos os endereços disponiveis em ambos os testes. Para que as tabelas fossem melhor aproveitadas seria interessante a utilização de palavras com maior número de caracteres. Outra opção seria modificar a função de hash de maneira que o somatório de caracteres fosse subtituído por uma expressão que gere valores maiores. Isto faria com que toda a tabela fosse realmente utilizada.

Conluindo, os histogramas gerados mostraram que a função de hash para a aplicação que estamos utilizando, armazenar palavras do arquivo “words.txt”, não é a mais adequada. A distribuição dos registros está bem distante de um comportamento uniforme.

Item C) A busca por palavras na tabela hash está implementada e pode ser acessada através da opção “5)Localizar palavras ” do menu principal. Os passos a seguir mostram como buscar uma lista de palavras listadas em um arquivo de texto “lista.txt” sobre a tabela hash com 49587 registros bases carregados com os registros do arquivo “words.txt”.

Conteúdo do arquivo lista.txt:
gabriel
abc
AAA
extracellular

<<HashApp>>
---
1)Criar tabela hash
2)Carregar tabela hash a partir de um arquivo
3)Salvar tabela hash
4)Gerar histograma
5)Localizar palavras

6)Editar
0)Sair
---
*->**5***

<<Busca>>
---
1)Buscar palavra
2)Buscar palavras do arquivo
0)Voltar
---
*->**2***

Caminho do arquivo:lista.txt

gabriel 34049 52 -1
abc 34507 4
AAA 25395 0
extracellular 12170 2
Total de palavras pesquisadas:4

Foram pesquisadas quatro palavras das quais três foram encontradas, “abc”, “AAA” e “extracellular”. A palavra “gabriel” não foi encontrada, mas seu endereço base e posição que deveriam estar foram listados. Palavras podem ser pesquisadas diretamente do terminal, selecionando a opção “1)Buscar palavra ”.

<<Busca>>
---
1)Buscar palavra
2)Buscar palavras do arquivo
0)Voltar
---
*->**1***
Palavra:gabriel
gabriel 34049 52 -1

Voltando ao menu principal poderiamos adicionar a palavra “gabriel” manualmente escolhendo a opção “6)Editar” e “1)Adicionar palavra”.

<<HashApp>>
---
1)Criar tabela hash
2)Carregar tabela hash a partir de um arquivo
3)Salvar tabela hash
4)Gerar histograma
5)Localizar palavras
6)Editar

0)Sair
----
*->**6***
<<Edicao>>
----
1)Adicionar palavra
2)Remover palavras
0)Voltar
----
*->**1***
Palavra:gabriel
Palavra "gabriel" adicionada...

Uma nova busca pela palavra “gabriel” poderia ser feita para verificar se a inclusão foi realizada com sucesso:

<<Busca>>
----
1)Buscar palavra
2)Buscar palavras do arquivo
0)Voltar
----
*->**2***
Caminho do arquivo:lista.txt

gabriel 34049 0
abc 34507 4
AAA 25395 0
extracellular 12170 2
Total de palavras pesquisadas:4

Perceba que a palavra “gabriel” foi incluída no inicio da lista encadeada. A busca realizada sem sucesso anteriormente não encontrou a palavra após ter varrido todos os elementos do endereço base 34049. Por isso anteriormente o programa sugeriu que a palavra “gabriel” deveria estar no registro 52 do endereço base 34049. A inclusão de elementos no inicio da lista é mais eficiente porque não necessita percorrer a lista encadeada do endereço base ou a manutenção de um ponteiro adicional.

Item D) A remoção de palavras da tabela hash pode ser realizada através do arquivo ou manualmente. Seguindo os passos anteriores, considerando a adição da palavra “gabriel” a tabela hash poderiamos utilizar o arquivo “lista.txt” para remover todas aquelas palavras préviamente pesquisadas. Para isso realize os seguintes passos:

<<HashApp>>
----
1)Criar tabela hash

2)Carregar tabela hash a partir de um arquivo
3)Salvar tabela hash
4)Gerar histograma
5)Localizar palavras
6)Editar
0)Sair
---
*->**6***

<<Edicao>>
---
1)Adicionar palavra
2)Remover palavras
0)Voltar
---
*->**2***

<<Remocao>>
---
1)Remover palavra
2)Remover palavras a patir do arquivo
0)Voltar
---
*->**2***
*Caminho do arquivo:**lista.txt***
Palavra "gabriel" removida com sucesso...
Palavra "abc" removida com sucesso...
Palavra "AAA" removida com sucesso...
Palavra "extracellular" removida com sucesso...

Todas as palavras foram removidas, uma nova tentativa de remoção das mesmas palavras gera o resultado:

Palavra "gabriel" nao encontrada...
Palavra "abc" nao encontrada...
Palavra "AAA" nao encontrada...
Palavra "extracellular" nao encontrada...

Item E) Para utilizar a função de hash que considera a multiplicação de caracteres, crie uma nova tabela hash considerando a opção “2)Multiplicacao de caracteres ”:

<<Funcao de Hash>>
----
1)Soma de caracteres
2)Multiplicacao de caracteres
----
*->**2***

Como feito anteriormente é possivel carregar o arquivo de palavras, gerar o histograma, buscar, remover e inserir registros.

Análise dos resultados

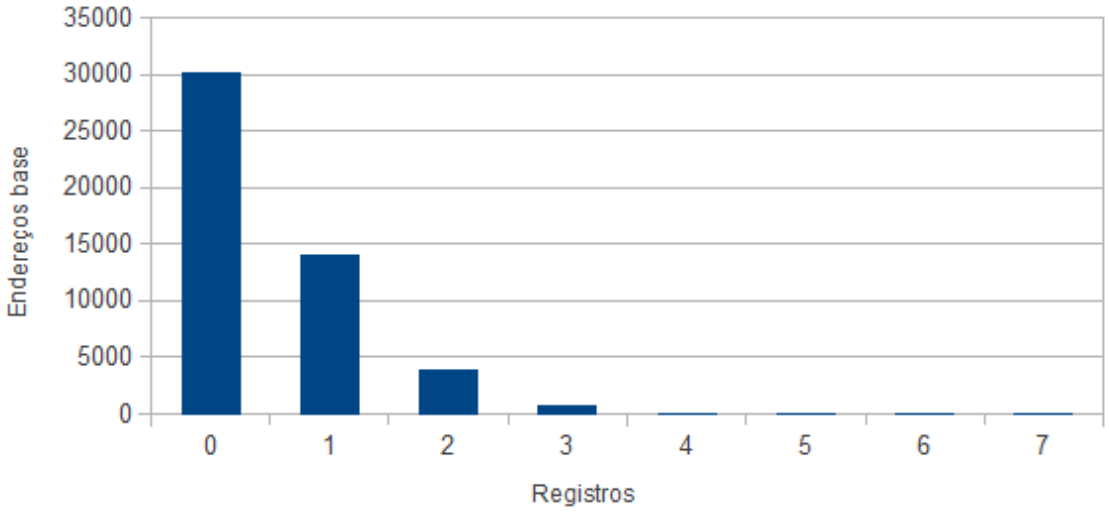
No item B) foi mostrado que a função de hash considerando a soma dos caracteres nas palavras não era adequada. O número de caracteres nas palavras não era suficiente para utilizar todo o endereçamento disponível uniformemente na tabela hash. Desta forma esta função de hash não possuia comportamento uniforme para tal aplicação.

No item E) foi implementada a modificação da função hash, considerando o produtório dos caracteres. A multiplicação dos caracteres passa a gerar números muito superiores, e provavelmente devem melhorar os resultados. A Tabela 3 e o Gráfico 3 mostram o histograma gerado pela função de hash que considera a multiplicação de caracteres em uma tabela hash com 49157 endereços bases.

Endereco com 0 registros: 30135
Endereco com 1 registros: 14129
Endereco com 2 registros: 3870
Endereco com 3 registros: 847
Endereco com 4 registros: 153
Endereco com 5 registros: 18
Endereco com 6 registros: 4
Endereco com 7 registros: 1

Tabela 3 - Histograma para a função de hash considerando a multiplicação de caracteres e a tabela hash com 49157 endereços base.

Gráfico 3 – A função de hash com 49157 endereços base considerando a multiplicação dos caracteres permite uma maior proximidade de uma distribuição uniforme.



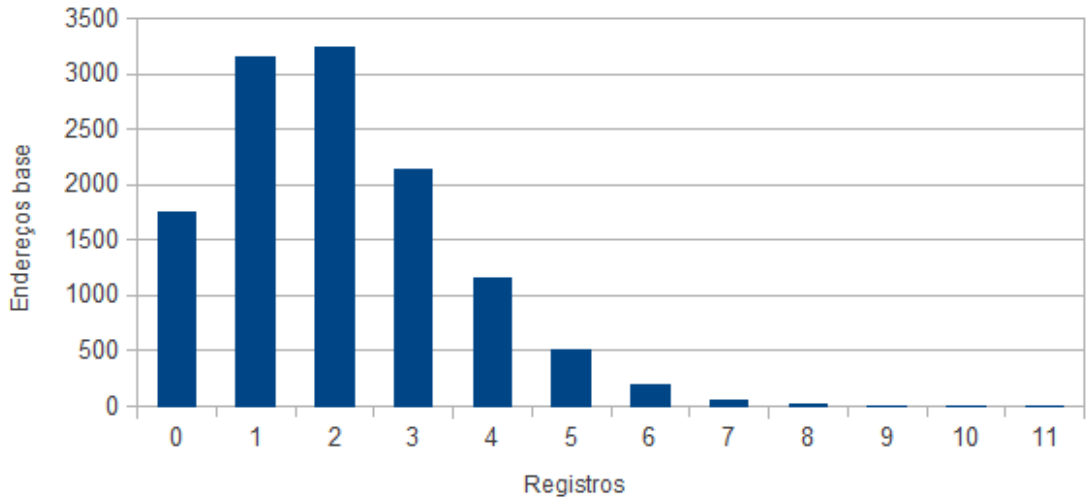
Analisando o histograma é possivel observar grande melhora na distribuição dos registros. O número de endereços base sem registros é de 30134 e o número de endereços base com apenas 1 registro é de 14130. No item B) foi mostrado que o número de registros base sem nenhum elemento considerando uma função de hash uniforme deveria ser de 29000 e o com apenas 1 registro 15080 (o cálculo considerou a distribuição de Poisson). Perceba que a alteração na função fez com que os resultados se aproximassem muito de uma função de hash uniforme. Este comportamento melhora abruptamente o desempenho da tabela, visto que o número de colisões são extremamente reduzidos.

O mesmo procedimento foi realizado para uma tabela hash com 12289 endereços base. O histograma obtido pode ser visto na Tabela 4 e Gráfico 4.

Endereco com 0 registros: 1765
Endereco com 1 registros: 3170
Endereco com 2 registros: 3257
Endereco com 3 registros: 2141
Endereco com 4 registros: 1164
Endereco com 5 registros: 510
Endereco com 6 registros: 194
Endereco com 7 registros: 55
Endereco com 8 registros: 21
Endereco com 9 registros: 8
Endereco com 10 registros: 3
Endereco com 11 registros: 1

Tabela 4 - Histograma para a função de hash considerando a múltiplicação de caracteres e a tabela hash com 12289 endereços base.

Gráfico 4 – A função de hash com 12289 endereços base considerando a multiplicação dos caracteres obteve uma maior proximidade de uma distribuição uniforme.



Novamente obtemos uma distribuição bem próxima da uniforme calculada no item B). Considerando uma distribuição uniforme esperavamos que o número de endereços vazios fossem da ordem de 1588 e o número de endereços com apenas um registro da ordem de 3250. Observando a Tabela 4, percebemos grande proximidade desta distribuição.

Conclusão

A implementação da aplicação foi de grande valia para compreender como a função de hash é importante no desempenho das tabelas hash. Ficou claro que a função de hash que considera a soma de caracters não satisfez os requisitos. Ela obteve um comportamento pouco uniforme, o que acarretava em um elevado número de colisões. Este comportamento ocorreu porque o somatório dos caracteres não acabava gerando números elevados o suficiente para utilizar todos os endereços disponíveis na tabela. Ela poderia obter melhores resultados caso armazenasse frases e não palavras. Por sua vez, a função de hash que considerava a multiplicação de caracteres obteve um resultado muito próximo de uma distribuição uniforme. Gerando um menor número de colisões e melhor desempenho na tabela hash. Isto ocorreu pois a múltiplicação passou a gerar valores suficientemente altos para cobrir todos os endereços base criados na tabela.