Criação de um *software* para geração de índices remissivos

Arthur Alexsander Martins Teodoro[1]

[1]Instituto Federal de Minas Gerais - Campus Formiga

arthurmteodoro@gmail.com

Introdução

O trabalho proposto era a criação de um *software* para a criação de índices remissivos. Um índice remissivo é basicamente um arquivo que serve de referência de um texto. O índice remissivo é uma lista em ordem alfabética que lista onde cada palavra chave previamente definida se encontra no texto. Para a geração de tal, cada palavra lida no texto deve ser procurada na base de palavras chave, caso existindo, acrescentando a página(no caso deste trabalho, linha), onde tal palavra está. Porém, como visto, é necessário realizar várias buscas, o que acarreta em uma demora no processamento do *software* caso não seja usada uma estrutura de dados especializada em busca.

O sistema implementado para tal problema foi o uso da estrutura de dados tabela *hash* ou tabela de dispersão. Tabela *hash* nada mais é do que um vetor indexado por uma função *hash* que utiliza como parâmetro uma chave de busca. Isso faz com que caso queira realizar a busca da palavra, não é necessário realizar uma busca linear, e sim utilizar a função *hash* para descobrir em que índice do vetor tal dado se encontra. Porém, pode ocorrer colisões, isto é, quando mais de um dado possui o mesmo retorno da função *hash*. Para tal fato, existe tratamentos de colisões que serão apresentados neste trabalho.

Implementação

Nesta seção deste documento será mostrada as escolhas de implementação, demonstração das estruturas de dados e demais questões de implementação.

Descrição sobre as decisões de projeto e implementação

Foi usada a linguagem de programação C para a implementação do trabalho. O trabalho foi codificado usando o editor de texto Sublime Text 3[1], foi usado o compilador GCC versão 5.4.0, sistema operacional GNU/Linux Ubuntu 16.04, e controle de versão usando método Git.

Foi criado uma TAD de lista encadeada(*lista.c, lista.h*) para o índice remissivo, uma tabela *hash* com tratamento de colisão *open Hash*(*openHash.c, openHash.h*) e Encadeamento Externo(*hashEncadeada.c, hashEncadeada.h*). Além das TAD's, foi criado um arquivo que utiliza a TAD criada para criar o índice remissivo(*indice.c, indice.h*), sendo realizada neste arquivo a manipulação de *strings*, leitura e gravação de arquivos de saída. Foi criado um arquivo que contém a função *main() (main.c)* que utiliza as função criadas no arquivo *indice.c*, não sendo necessário o usuário conhecer as minúcias do processo de criação do índice remissivo.

Para a criação do índice, é lido um arquivo de palavras chaves seguindo um padrão especificado pelo professor, logo as palavras que estão no arquivo de palavras chave são

[1]https://www.sublimetext.com/

as usadas no processo de busca, logo qualquer palavra que não possua a mesma sequência de caracteres que as palavras do arquivo de palavras chave são consideradas diferentes, exceto pelo uso de letras maiúsculas, que são tratados no código. O arquivo de palavras chave é copiada exatamente igual para a estrutura de dados, logo caso exista uma palavra repetida, a mesma será contabilizada duas vezes.

Compilação e Execução do projeto

Foi criado um arquivo de *Makefile* usado para a compilação do sistema, porém este deve ser alterado dependendo da estrutura que deseja ser criada, além de qual função *hash* usar e seu método de tratamento de colisão. Algumas alterações também devem acontecer nos arquivos fonte para a compilação. Quando é desejado o uso da lista encadeada, o usuário deve alterar nos arquivos *indice.c*, e no arquivo *main.c* o arquivo que será incluído no código(include). Além disso, existe uma linha específica no arquivo *main.c* que deve ser comentada quando a estrutura de dados deve possuir o mesmo tamanho que a quantidade de palavras chave(*openHash* e lista encadeada).

Para a execução do *software*, deve ser passado por argumentos na linha de comando o nome de três arquivos, nesta ordem: a) Palavras Chave: arquivo que possui as palavras que devem ser indexadas, b) Texto: arquivo com o texto que se deseja criar o índice remissivo e c) Saída: arquivo com o índice remissivo criado com sucesso.

Após o término da execução será mostrado o tempo gasto na geração do índice remissivo e a quantidade de colisões que ocorreu, dados usados para a confecção deste documento.

Estruturas Criadas

Para tal trabalho foi criada duas estruturas de dados: uma *openHash*, onde colisões são tratadas na própria *hash* e a *hash* com encadeamento externo, onde colisão são tratadas em uma lista de colisão.

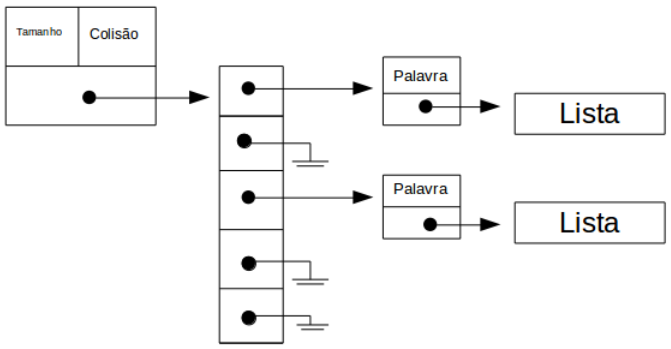

Figura 1. Diagrama da estrutura *openHash* implementada

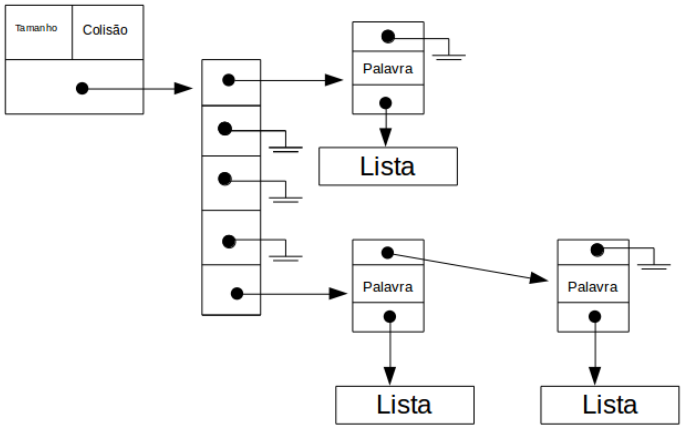

Figura 2. Diagrama da estrutura *Hash* com encadeamento externo implementada

Funções *hash* implementadas

Foi implementada três funções *hash*, sendo respectivamente: uma função que realiza a multiplicação do valor ASCII por um valor de um vetor randômico, uma função que multiplica o ASCII pelo índice da string que o caractere se encontra, e uma função que multiplicao o ASCII pelo índice e soma por um vetor randômico.

Já as funções *hash* implementada é respectivamente: uma função que, caso exista dado na posição, caminha uma posição à frente, outra função que em caso de colisão, verifica se três posições para a frente já esta ocupada, e caso esteja, verifica a próxima posição, ou seja, a quarta posição à frente da inicial. Já a outra função de colisão caminha uma posição à trás.

Testes Realizados

Para a realização dos testes, foi usado o teste de número 5 disponibilizado pelo professor. Tal teste possui um número relativamente grande de palavras chave e um texto extenso, por isso foi o escolhido para a bateria de testes. Foi realizado com a TAD *openHash* testes para descobrir de quais das funções *hash* implementadas seria a melhor além do tratamento de colisão. Já no caso da *hash* com encadeamento externo, foi realizado testes para realizar a comparação entre a *openHash* e verificar qual função *hash* mostra melhor resultado além do tamanho recomendado da mesma. O teste com lista encadeada serviu apenas de comparação com os métodos *hash*.

Teste	Colisão 1	Colisão 2	Colisão 3
1	0,073055	0,160575	0,09225
2	0,072901	0,1295	0,118828
3	0,078485	0,161455	0,111155
4	0,072819	0,148069	0,087041
5	0,072819	0,136204	0,085408
6	0,086206	0,129329	0,095506
7	0,109226	0,131329	0,084515
8	0,081717	0,123671	0,081119
9	0,095341	0,176807	0,100793
10	0,086417	0,142206	0,112044
Média	0,0828986	0,1439145	0,0968659

Figura 3. Tabela de Tempo da Função 1

Teste	Colisão 1	Colisão 2	Colisão 3
1	664	333	1920
2	912	427	798
3	884	213	670
4	1068	418	1046
5	722	424	798
6	808	268	599
7	801	226	573
8	767	331	792
9	722	375	690
10	851	309	795
Média	819,9	332,4	868,1

Figura 4. Tabela de Colisão da Função 1

Teste	Colisão 1	Colisão 2	Colisão 3
1	0,11821	0,122832	0,089356
2	0,126621	0,129617	0,10892
3	0,073396	0,121911	0,081006
4	0,099556	0,14801	0,081576
5	0,078688	0,127585	0,094074
6	0,119078	0,12755	0,10713
7	0,091545	0,1311554	0,080769
8	0,099929	0,12934	0,086086
9	0,102637	0,125919	0,081521
10	0,073095	0,124295	0,103557
Média	0,0982755	0,12882144	0,0913995

Figura 5. Tabela de Tempo da Função 2

Teste	Colisão 1	Colisão 2	Colisão 3
1	711	492	765
2	711	492	765
3	711	492	765
4	711	492	765
5	711	492	765
6	711	492	765
7	711	492	765
8	711	492	765
9	711	492	765
10	711	492	765
Média	711	492	765

Figura 6. Tabela de Colisão da Função 2

Teste	Colisão 1	Colisão 2	Colisão 3
1	0,083588	0,144188	0,081448
2	0,079486	0,165292	0,102736
3	0,081606	0,134684	0,099646
4	0,074652	0,131286	0,08605
5	0,128082	0,128468	0,085285
6	0,080445	0,143932	0,085064
7	0,074895	0,125911	0,85169
8	0,106146	0,127221	0,081502
9	0,074556	0,16463	0,093777
10	0,08783	0,126705	0,105628
Média	0,0871286	0,1392317	0,1672826

Figura 7. Tabela de Tempo da Função 3

Teste	Colisão 1	Colisão 2	Colisão 3
1	629	278	734
2	732	376	633
3	556	423	923
4	557	356	619
5	885	340	733
6	612	479	597
7	555	272	725
8	808	319	1017
9	686	458	989
10	1050	456	915
Média	707	375,7	788,5

Figura 8. Tabela de Colisão da Função 3

Para cada Função *Hash* da *openHash*, foi feito 3 testes, cada um com um método de colisão. Para cada teste, foi feito 10 testes, e no final, calculada a média de tempo e colisão de tal teste.

Teste	Função 1	Função 2	Função 3
1	0,029512	0,044398	0,033484
2	0,032194	0,033762	0,032523
3	0,038896	0,029963	0,022357
4	0,01645	0,031891	0,023053
5	0,015755	0,022409	0,017422
6	0,034671	0,03574	0,033823
7	0,040505	0,016975	0,032073
8	0,043492	0,037166	0,016153
9	0,020219	0,021619	0,029195
10	0,034698	0,016326	0,040976
Média	0,0306392	0,0290249	0,0281059

Figura 9. Tempo da Hash com Encadeamento Externo com tamanho de 25%

Teste	Função 1	Função 2	Função 3
1	93	94	94
2	93	94	93
3	93	94	94
4	95	94	93
5	94	94	93
6	94	94	93
7	94	94	94
8	93	94	95
9	93	94	93
10	94	94	93
Média	93,6	94	93,5

Figura 10. Colisões da Hash com Encadeamento Externo com tamanho de 25%

Teste	Função 1	Função 2	Função 3
1	0,016748	0,016899	0,02082
2	0,022521	0,016769	0,016368
3	0,017013	0,017214	0,018454
4	0,016781	0,016729	0,017855
5	0,017501	0,016864	0,017372
6	0,017291	0,017167	0,039103
7	0,033759	0,016266	0,037936
8	0,02124	0,018281	0,02206
9	0,034339	0,017266	0,016294
10	0,021482	0,017039	0,033307
Média	0,0218675	0,0170494	0,0239569

Figura 11. Tempo da Hash com Encadeamento Externo com tamanho de 50%

Teste	Função 1	Função 2	Função 3
1	73	69	70
2	71	69	72
3	69	69	77
4	71	69	68
5	68	69	77
6	69	69	68
7	69	69	73
8	74	69	73
9	73	69	73
10	70	69	73
Média	70,7	69	72,4

Figura 12. Colisão da Hash com Encadeamento Externo com tamanho de 50%

Teste	Função 1	Função 2	Função 3
1	0,036308	0,014993	0,02142
2	0,033595	0,038147	0,038995
3	0,037085	0,036872	0,040915
4	0,049314	0,032815	0,038297
5	0,036792	0,020575	0,01536
6	0,035511	0,032381	0,030634
7	0,042441	0,01663	0,029556
8	0,041213	0,022273	0,031264
9	0,021357	0,021525	0,039691
10	0,031222	0,040329	0,022892
Média	0,0364838	0,027654	0,0309024

Figura 13. Tempo da Hash com Encadeamento Externo com tamanho de 75%

Teste	Função 1	Função 2	Função 3
1	53	56	81
2	59	56	84
3	58	56	79
4	55	56	79
5	59	56	82
6	53	56	80
7	53	56	85
8	58	56	79
9	58	56	83
10	55	56	80
Média	56,1	56	81,2

Figura 14. Tempo da Hash com Encadeamento Externo com tamanho de 75%

Também foi realizado 3 testes, cada um para verificar qual é o melhor tamanho da *hash* além de qual melhor função.

Estudo de complexidade

As funções de inserção quanto busca tanto dos dois métodos de *hash* possui complexidade média de *O(1)*, somente no caso da lista encadeada que a busca terá caso médio de *O(n)*. As funções de manipulação de string, na maioria das vezes, percorre toda a string, fazendo com que sua complexidade seja *O(n)*. A função de leitura do arquivo de palavras chave possui complexidade de *O(n)*, uma vez que lê o arquivo de palavras até o fim. Já a função de geração do índice remissivo é *O(n)*, uma vez que cada linha é lida e tratada. Já o

arquivo de saída também possui complexidade de *O(n*log n)*, uma vez que é necessário retornar uma lista além da ordenação do vetor para gerar o índice em ordem alfabética.

O código em suma possui complexidade de *O(n*log n)*, uma vez que o *quicksort* possui tal complexidade.

Análise dos Resultados

Segundo os dados demonstrados na seção Testes Realizados, foi possível perceber que, a melhor função *hash* é a segunda, ou seja, a função que multiplica pelo índice do vetor. Tal função possui o diferencial que, não dependendo do tamanho continua gastando pouco tempo, uma vez que não é necessário randomizar um vetor nesta função. Já a melhor função de colisão é a *re-hash*, que acaba gerando uma quantidade melhor de colisões, porém gastando mais tempo.

Entre as estruturas criadas, a que se melhor destaca é a *hash* com encadeamento externo, gerando menos tempo e um número menor de colisões. O melhor tamanho de *hash* com encadeamento externo de acordo com os resultados colhidos é entre 50% a 75% do tamanho de dados.

Já com dados maiores, foi mostrado que a *openHash* deve desempenho duas vezes pior que a *hash* com encadeamento externo, e a lista encadeada se mostrou entre a *openHash* e *hash* com encadeamento externo.

Conclusão

Foi concluído que, a estrutura de dados *hash* é melhor para o problema de construção de índice remissivo, uma vez que tal problema necessita de muitas buscas, sendo a busca linear não recomendada para tal. Também foi aumentado o aprendizado da escolha de estruturas de dados para cada um tipo de problema e implementação de tipos abstratos de dados.