UNIVERSIDADE FEDERAL DO MATO GROSSO *CAMPUS* ARAGUAIA

ANTHONY MUNIZ PRADO DE OLIVEIRA, VINICIUS SPANHOL FERRARI

TRABALHO - TABELA HASH

BARRA DO GARÇAS - MT

2022

1 INTRODUÇÃO

Uma tabela hash, tabela de dispersão ou ainda tabela de espalhamento, é uma estrutura de dados utilizada para tornar o processo de busca mais eficiente. Assim, esta estrutura de dados é muito utilizada não para inserções ou remoções, mas quando há a necessidade de realizar muitas buscas e com rápido tempo de resposta.

O objetivo deste trabalho é exercitar a implementação e manipulação de tabelas Hashes com a linguagem de programação C. Também foi trabalhado a manipulação de arquivos para inserir dados nas tabelas hashes em questão.

Foi utilizado o editor de textos Visual Studio Code para a programação, rodando o Ubuntu 20.04 no WSL 2 via Windows.

2 DESENVOLVIMENTO

O desenvolvimento consiste de 5 etapas onde 3 delas serão tratadas neste trabalho:

Etapa 1:

Elabore um algoritmo em linguagem C/C + + que gere um arquivo com um conjunto de valores e suas respectivas chaves, para serem inseridos em uma tabela Hash. Este algoritmo recebe como parâmetro o número de elementos a ser gerado, o tipo das entradas e o nome do arquivo resultante. Exemplo:

> ./algoritmo 1000 1 saida.txt

Onde algoritmo é o nome do programa, 1000 é o número de elementos a ser gerado, 1 é o tipo de arquivo (explicado a seguir) e saida.txt o nome do arquivo resultante.

Como resultado, no arquivo resultado, cada linha deve conter uma chave inteira com valor entre 0 e 1023, e um valor composto por 3 letras. Estes valores serão definidos aleatoriamente, mas garanta que não existem chaves com valores repetidos.

São dois tipos de arquivos de entrada possíveis:

1. Todas as chaves geradas têm valor par;
2. Sem restrição.

Nesta etapa houveram problemas para se gerar os arquivos (foi resolvido), também em gerar a função rand de forma que os valores não se repetissem(foi resolvido).

Etapa 2:

Implemente um algoritmo que receba um arquivo resultante da etapa 1 a aplique em uma tabela Hash de tamanho 100 e que trate as colisões com encadeamento externo.

Para o arquivo o resultante da etapa 1 teste duas 2 funções Hash distintas de tratamento de colisões a seu critério.

Desenvolva todas as funções necessárias para manipulação da estrutura (inserção, busca, etc).

Nesta etapa escolhemos como funções hashes a de divisão (d mod m) e multiplicação (piso(m * (d * A mod 1))) onde d=chave, m=tamanho da tabela e A é a constante 0.618034. Porém, entretanto, houve um problema onde mod não "aceita" tipos float, então precisamos criar um algoritmo equivalente:

```
double n = m * ((chave * A) - ((int)(chave * A)));
return floor(n);
```

Que funciona da seguinte forma: para pegar apenas a parte fracionária do número, nós pegamos o número e subtraímos sua parte inteira sobrando apenas a parte fracionária.

Etapa 3:

Avaliando as funções hash.

Com o algoritmo da etapa 01:

- Gere 5 arquivos do tipo 1 com 50 números gerados, com 100 números gerados e 150 números gerados. Num total de 15 arquivo do tipo 1;
- Gere 5 arquivos do tipo 2 com 50 números gerados, com 100 números gerados e 150 números gerados. Num total de 15 arquivo do tipo 2;

Desta forma, ao final desta etapa teremos 30 arquivos.

Execute os 30 arquivos no algoritmo da etapa 2 e para cada execução, contabilize o número de colisões que ocorreram em cada uma das funções hash.

Arquivos gerados com sucesso, ficam armazenados na pasta "arquivos".

3 RESULTADOS

Conseguimos executar o trabalho e obtivemos os seguintes resultados:

Arquivos tipo 1

Colisões com Hash 1
Com 50 valores foram contadas: 98 colisões
Com 100 valores foram contadas: 278 colisões
Com 150 valores foram contadas: 508 colisões

Colisões com Hash 2
Com 50 valores foram contadas: 66 colisões
Com 100 valores foram contadas: 169 colisões
Com 150 valores foram contadas: 336 colisões

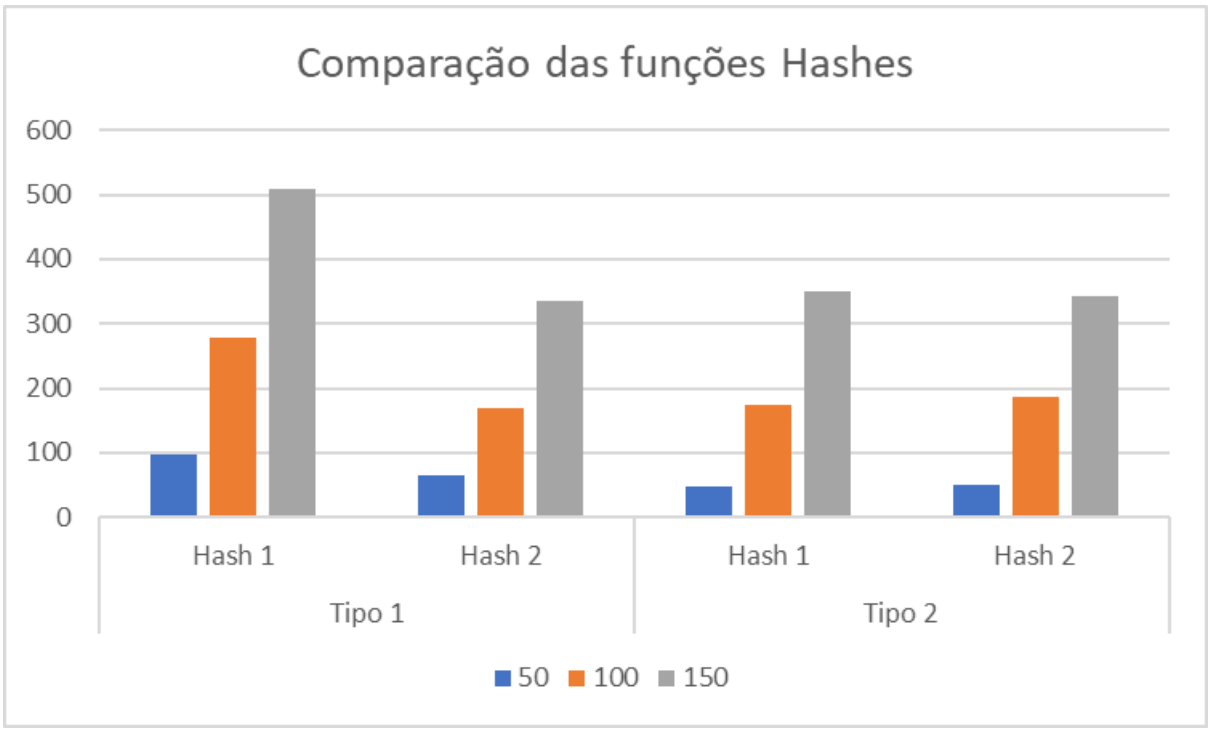
==========

Arquivos tipo 2
Colisões com Hash 1
Com 50 valores foram contadas: 48 colisões
Com 100 valores foram contadas: 173 colisões
Com 150 valores foram contadas: 350 colisões

Colisões com Hash 2
Com 50 valores foram contadas: 49 colisões
Com 100 valores foram contadas: 186 colisões
Com 150 valores foram contadas: 342 colisões

Ao total foram contadas:
Colisões: 2603 colisões

	Tipo 1		Tipo 2	
Arquivos	**Hash 1**	**Hash 2**	**Hash 1**	**Hash 2**
50	98	66	48	49
100	278	169	173	186
150	508	336	350	342



4 CONSIDERAÇÕES FINAIS

Ao final da análise, e com os resultados obtidos do desenvolvimento do trabalho, concluímos que os diferentes algoritmos hash tanto o 1º que utiliza módulo realizando sucessivas divisões quanto o hash que utiliza o 2º método de multiplicação como foi denominado, têm certas diferenças tratando os diferentes tipos de arquivos. Foi observado que o 2º Hash foi o mais bem sucedido em ambos os casos, com uma atenção maior a os arquivos do tipo 1 que o 1º Hash não conseguiu lidar muito bem o que foi o completo oposto para o outro (Hash 2) que conseguiu uma ótima performance com apenas 336 colisões nos testes com a maior quantidade de números enquanto o seu concorrente realizou 508 colisões.

Porém apesar do caso dos arquivos tipo 1 que eram composto por números pares apesar deste método de cálculo do 2º ter sido melhor, o 1º teve sucesso com valores aleatórios do tipo 2 de arquivos, conseguindo uma performance um pouco melhor para os testes com ambos as quantidades de valores, a não ser pelo final, em que sua eficiência em controlar as colisões caiu um pouco dando ao segundo Hash mais uma vez a vitória para uma maior eficiência no tratamento do problema para grandes quantidades de número, ressaltando novamente que neste tipo de arquivo (tipo 2: todos os valores são aleatórios de 0 a 1023), ele teve uma performance muito parecida com seu adversário, ganhando neste último teste.

Por fim definitivamente concluímos que o 2º método de Hash era o melhor em ambos os tipos de arquivos e em tratar uma quantidade de valores maior, mesmo ele não tendo o controle das colisões mais preciso tratando o tipo 1 no início com poucos valores, ele não se distanciou tanto dos resultados do outro método e até consegui superá lo à medida que a quantidade dos valores do teste aumentava.