JOÃO VITOR CAVERSAN DOS PASSOS

FUNCIONAMENTO DE UMA MERKLE TREE COM PERMANÊNCIA DE DADOS PARA A DISCIPLINA DE ESTRUTURA DE DADOS II

Brasil

2021

JOÃO VITOR CAVERSAN DOS PASSOS

FUNCIONAMENTO DE UMA MERKLE TREE COM PERMANÊNCIA DE DADOS PARA A DISCIPLINA DE ESTRUTURA DE DADOS II

Relatório de projeto final da disciplina Estrutura de Dados II do curso Engenharia da Computação da Universidade Tecnológica Federal do Paraná, como requisito parcial para obtenção da média final.

UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

DAINF

Estrutura de Dados II

Brasil

2021

Introdução

O presente trabalho tem como seu objetivo demonstrar o funcionamento de uma *Merkle Tree*, também conhecida por *Hash Tree*, por meio de terminal, assim como fazendo a permanência de dados do que for inserido em tal estrutura de dados.

A interface gráfica é feita por meio de um menu em terminal. Neste o usuário é provido de diversas opções as quais proporcionam uma experiência completa do funcionamento da estrutura.

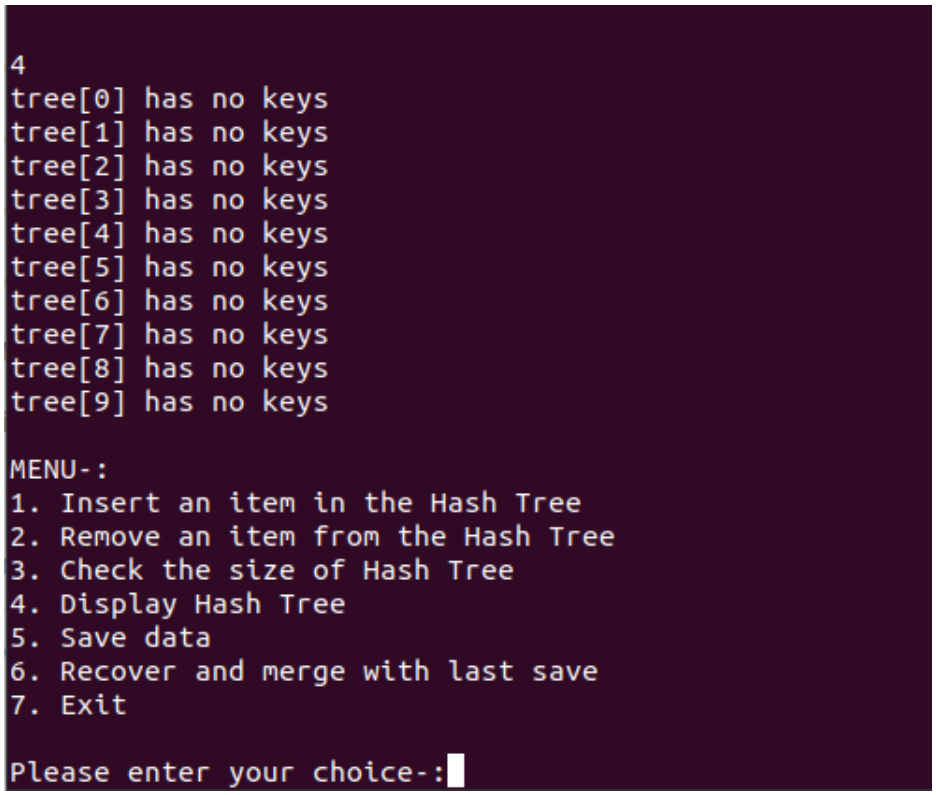
O programa foi implementado em linguagem C, com o intuito de se aproximar ao máximo do que fora aprendido em aula. Todas as estruturas novas são feitas por meio de *structs* e funções para manejo de tais.

1 Descrição do aplicativo

1.1 Interface do usuário

Para o propósito de demonstração direta do funcionamento de uma estrutura do tipo *Hash* juntamente com o conceito de árvore binária de busca, foi-se implementado um menu simples, porém intuitivo, que proporciona todo tipo de modificação que pode ser feito na estrutura, assim como visto na Figura 1

Figura 1 – Interface



Assim como será discutido mais adiante, todo valor a ser adicionado na estrutura, deve ser seguido por uma chave. Esta, por sua vez, é utilizada para cálculos matemáticos para a inclusão exata do valor.

Para a retirada pede-se apenas a chave, ou seja, o elemento responsável pela localização do valor.

Também pode-se ver na figura 1 que uma lista está presente acima do menu. Este é o resultado do comando de demonstração da árvore.

Ademais se tem o retorno da quantidade de elementos na estrutura, as opções relacionadas com a permanência de dados, e, por fim, a opção de saída do programa.

2 Aplicação de Estrutura de Dados

Esse capítulo relaciona o projeto com o aprendido na Disciplina de Estrutura de Dados 2. Aqui seguem dados mais técnicos e com relação a métodos de código.

2.1 Artifícios da matéria utilizados

Duas lógicas aprendidas na matéria foram utilizadas no código, elas sendo: Estrutura de *Hash* juntamente com uma árvore binária de busca.

Para a permanência de dados foi utilizada uma estrutura simples de gravação de dados em arquivo de texto. A seguir será discutido o funcionamento de cada uma.

2.1.1 Estrutura de Hash

A estrutura de *Hash* é um artifício principalmente utilizado para minimizar o custo de busca de valores (em linguagem assintótica, se aproximando ao máximo de O(1)). Seu funcionamento é inicialmente dado por uma simples conta matemática com base em uma chave (número qualquer) provida pelo usuário. Essa conta retorna um valor equivalente a uma posição de um vetor no qual os dados serão inseridos (denominado de *Hashcode*).

Os obstáculos se iniciam no momento em que o *Hashcode* para duas chaves diferentes, é igual, o que causa uma colisão. Para isso deve-se contornar o problema de inserção de alguma maneira. Caso se utilize apenas um vetor, a conta matemática deve tratar de retornar algum valor diferente para interações sequenciais. Porém há soluções mais convenientes para quando se maneja dados excessivos, e uma delas, a que foi utilizada no código em questão, é a de disponibilizar uma Árvore Binária de Busca (ABB) para cada posição deste vetor inicial, e inserir dados nela caso ocorra uma colisão.

Devido à essa necessária associação de um valor com uma chave, e a utilização de *Hashcodes* para cada chave, essa estrutura é amplamente utilizada na área de criptografia, uma vez que a única maneira de ter ciência da localização de um dado, é caso se saiba a chave de tal e a conta matemática a qual foi responsável pelo posicionamento deste no vetor.

No código, ao ser requisitada a inserção de um valor, o programa solicita uma chave juntamente, e a utiliza para fins de alocação na estrutura, assim como clarificado acima. A conta utilizada para o *Hashcode* foi a de:

$$H(k) = kmod(M)$$

Onde H(k) é o *Hashcode*, "k"é a chave, mod() é a função que retorna o resto de divisão euclidiana de "k"pelo valor "M", e "M"o tamanho do vetor inicial.

2.1.2 Árvore Binária de Busca

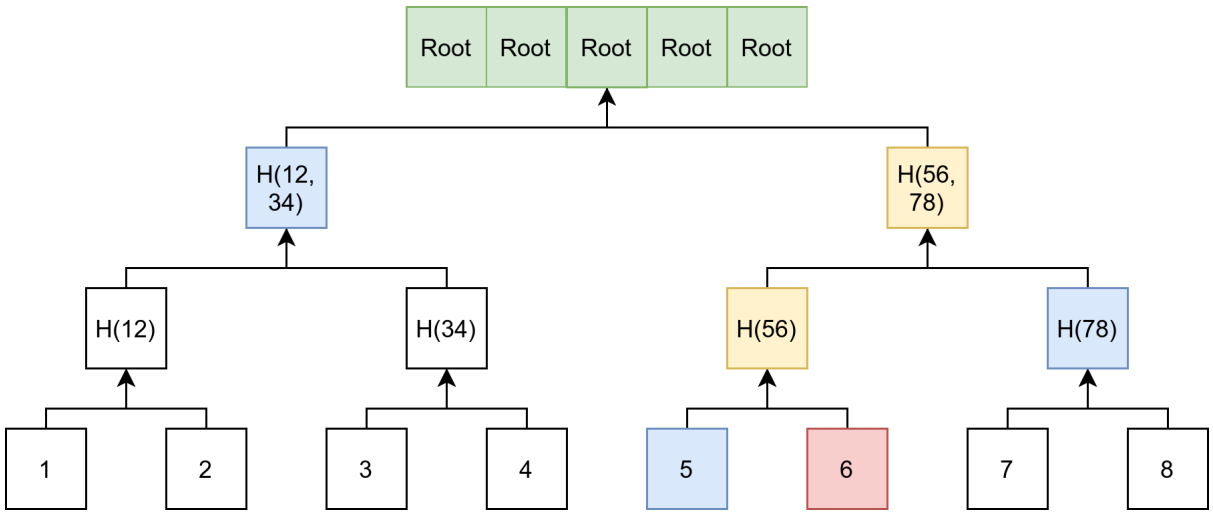
Como solução para colisões no vetor, foi-se utilizada a implementação de uma estrutura de Árvore Binária de Busca (ABB) para cada posição de tal.

Uma ABB utiliza o conceito básico de árvore computacional: todo nó tem 2 nós que derivam de si, sendo o nó inicial chamado de nó pai e os secundários chamados de filhos. Porém com uma pequena diferença: todos os nós à direita de um devem conter chaves maiores que o do pai, e todos os nós à esquerda devem conter chaves menores.

Com isso em mente, o tratamento de colisões foi feito da seguinte maneira. Caso o *Hashcode* de uma chave coincida com o de outra chave, esse pacote de dados (chave com valor) é adicionado em uma árvore de posição igual ao *Hashcode*. Caso a chave em si for maior que a chave pai, ela é mandada para a direita da árvore, e caso seja menor, é mandada para a esquerda.

A estrutura como conjunto fica visualmente similar ao demonstrado na Figura 2.

Figura 2 – Exemplo de Merkle Tree



2.1.3 Permanência de dados

Para este quesito, foi utilizado as funções file já presentes na biblioteca padrão da linguagem C. Apesar de não condizer com o conceito de armazenamento e busca de dados veloz que a *Merkle Tree* oferece, esse método era o único conhecido no momento de produção do código.

Para o armazenamento de dados, primeiro se abre um arquivo no modo de armazenamento, depois o vetor é percorrido linearmente (O(n)), e em cada posição de tal, a árvore equivalente é percorrida (O(logN)) e as chaves, assim como os valores, são armazenados linha à linha por meio da função *fprintf()*. No final, um arquivo do tipo .txt (*Text*) é criado, e todas as informações em sequência podem ser vistas caso o usuário decida abrir tal arquivo. No geral, a operação é de custo médio O(NlogN).

Para a recuperação de dados e inserção na estrutura em questão, o arquivo mencionado acima é aberto em modo de leitura, e caso ele exista (não necessariamente deve conter informações nele), seu conteúdo é percorrido de linha em linha por meio da função *fscanf()*, e as chaves, assim como os valores, são adicionados as suas respectivas árvores em posição correta. Para isso, o arquivo de texto deve ser percorrido linearmente, resultando em um custo de O(n).

Uma vez que a operação é concluída, o usuário pode continuar a inserir dados na estrutura, assim como armazená-los, à vontade. A função armazenamento de dados reescreve todo o arquivo de texto sempre que é requisitada. Por este motivo, o usuário é provido apenas de um arquivo para gravação.

2.2 Problemas Enfrentados

Durante a implementação das estruturas e técnicas de gravação, houve alguns problemas transpassados pelas as versões do programa. Com o intuito de relatar tais problemas, segue a seguinte seção.

2.2.1 Estrutura de Hash

Felizmente, durante a implementação de tal não houve problema algum.

2.2.2 Árvore Binária de Busca

Inicialização dos nós direita e esquerda: o único problema enfrentado com a implementação da árvore. Após o fim do código e testes, um problema muito específico ocorria: caso, após resgatar uma gravação anterior, se tentasse inserir um valor, no momento de imprimir o conteúdo das árvores o sistema acessava memória indevida e resultava em uma falha de segmentação (*segmentation fault*). O problema era a inicialização incorreta das informações de esquerda e direita de cada nó. Após a atribuição correta de *NULL* para os atributos devidos, o problema foi solucionado. O porquê do erro acontecer apenas nesse caso relacionado ao armazenamento continua um mistério.

2.2.3 Permanência de Dados

Erro de escolha da função: na primeira implementação houve um erro na escolha das funções. As utilizadas no armazenamento imprimiam letras sequenciais e não de linha em linha, o que impedia a leitura correta posteriormente. Em segunda tentativa, utilizou-se as funções *fscanf()* e *fprintf()*. Com isso os problemas foram solucionados.

Erro na escolha de variável para leitura do arquivo: de primeira mão, sabendo que a função de leitura retorna um ponteiro do tipo especificado (no caso variável *int*), foi feito o teste atribuindo o retorno à uma variável do tipo ponteiro de *int*. No teste com poucos valores o código funcionou perfeitamente. Ao implementar a ideia para adicionar os valores nas árvores, erros de segmentação ocorriam. Após diversos testes, ao mudar a variável de ponteiro para simples inteiro, e atribuir o retorno da função de leitura para seu endereço, assim como demonstrado abaixo, o problema se resolveu.

int* i, fscanf(arq, "%d", i) $\rightarrow inti, fscanf(arq, "\%d", \&i)$

3 Controle de versões

3.1 Versão 1.0

Na versão primária do aplicativo, era possível apenas as operações simples de inserção, remoção e demonstração dos elementos na árvore. Uma vez que o programa fosse fechado, todo o progresso seria perdido, funcionando apenas como demonstração de funcionamento dessa estrutura. Todas as estruturas de dados relacionadas à matéria já haviam sido implementadas.

3.2 Versão 1.1

Na segunda e última versão foi implementado o sistema de armazenamento de dados. Após a resolução dos problemas enfrentados, o programa estava completo, oferecendo todas as opções passadas assim como as de gravação e recuperação de dados.

4 Conclusão

O trabalho final da disciplina de estrutura de dados II é de suma importância para conclusão da matéria, é através dele que se consegue alcançar maior entendimento para aplicabilidade da cadeira de estrutura e ordenação de dados.

Por conta da brevidade da duração dos assuntos e das atividades, realmente não há tempo de correlacionar conteúdos. O trabalho proporciona esta relação não só de tópicos da própria disciplina como a oportunidade de empregar a interdisciplinariedade, consolidando a base prática da disciplina.

O experimento como um todo foi realizado individualmente, o que exigiu grande esforço e dedicação, mas que por fim se mostrou devidamente gratificante. Pode-se afirmar que todo o aprendizado foi absorvido, e sua aplicação foi assentada.

Referências

Merkle Tree. Wikipédia, 2021. Disponível em: https://en.wikipedia.org/wiki/Merkle_tree Acesso em: 05, agosto de 2021

H. Arquivos em C. Blog. Disponível em: http://linguagemc.com.br/arquivos-em-c-categoria-usando-arquivos/ Acesso em: 13, agosto 2021.

Moodle Estrutura de Dados 2 - BSI - Minetto. Disponível em: https://moodle.dainf.ct.utfpr.edu.br/course/view.php?id=125 Acesso em dias diversos.