ESTRUTURA DE DADOS

25/08/2017

APLICAÇÕES DE ÁRVORES AVL E TABELAS HASH EM REDES DE COMPUTADORES

Este relatório trata-se de uma simulação de uma rede de computadores através do uso de duas estruturas de dados: Árvores AVL e Tabelas Hash (ou tabelas de dispersão). O projeto consistirá da simulação duas infraestruturas computacionais: a internet e a intranet. Para a intranet foi implementado uma árvore AVL, onde cada nó da árvore corresponde a um dispositivo em rede. No caso da internet foi implementado uma Tabela Hash com encadeamento, onde cada posição do vetor corresponde a um ou vários dispositivos. A implementação foi feita na linguagem C++ com uso da IDE Microsoft Visual Studio Community 2017.

UNIVERSIDADE FEDERAL DE OURO PRETO

ICEB – DEPARTAMENTO DE COMPUTAÇÃO

AUTORES: PEDRO DA COSTA LOPES

RAYNER JOEL MEDEIROS DE SÁ

ESTRUTURA DE DADOS

APLICAÇÕES DE ÁRVORES AVL E TABELAS HASH EM REDES DE COMPUTADORES

1. INTRODUÇÃO

A estrutura geral de uma árvore AVL (já discutida no relatório anterior) é resumida em raízes, sub-árvores, alturas e fator de balanceamento. A árvore é balanceada a cada mudança feita que deixe a desbalanceada, e o métodos internos continuam o mesmo de uma BST. Portanto, não falaremos sobre a estrutura geral das árvores AVL no escopo deste relatório e nos limitaremos a mostrar apenas sua estrutura aplicada a uma simulação de redes de computadores. Como as tabelas Hash ainda não foram citadas, faremos uma breve explicação sobre estas.

Este relatório mostrará o funcionamento destas duas estruturas através de sua aplicação a redes de computadores através de demonstrações e exemplos. As árvores AVL foram implementadas unicamente com o propósito de simular uma intranet neste sistema, ou seja, uma rede de computadores interna e privada, que por sua vez possui uma quantidade pequena de dispositivos. As tabelas Hash foram implementas para a simulação da internet, ou seja, a rede global de computadores, que por sua vez possui uma quantidade muito maior de dispositivos conectados.

Portanto, veremos detalhes da implementação destas estruturas para esta simulação, tempos de execução, analise de complexidade, melhorias e possíveis trabalhos futuros.

2. ESTRUTURAS DE DADOS

2.1 Árvores AVL

Como citado anteriormente, não citaremos a estrutura de uma árvore AVL genérica, pois já foi citada em outro relatório.

2.2 Árvores AVL vs. Tabelas Hash

A grande melhoria das Tabelas Hash em comparação às árvores AVL é a melhoria no tempo de execução assintótico. Vimos que as árvores AVL possuem tempo de execução $O(\log_2 n)$ no pior caso para os métodos internos (busca, inserção, remoção, etc.). Já as tabelas Hash possuem tempo de execução $O(1)$ para busca de uma chave, que é uma grande melhoria. Veremos com detalhes como estas melhorias são feitas

2.3 Tabelas Hash e suas propriedades

2.3.1 INTRODUÇÃO

Tabelas Hash possuem um funcionamento completamente diferente de estruturas tipo árvore. O seu funcionamento é parecido com o funcionamento de uma estrutura de endereçamento direto (um vetor ou matriz alocado na memória), porém, no caso de vetores e matrizes, as chaves de valor k são diretamente alocadas a um índice k do vetor. Nas tabelas Hash, as chaves de valor k são armazenadas em índices de valor $h(k)$, onde h é chamada de **função Hash**, que mapeia as posições as chaves k nas posições $h(k)$ em um vetor chamado de **tabela Hash**.

É notório que, quando o universo de chaves é maior do que a tabela Hash, chaves que passarem pela função $h(k)$, a função Hash, poderão possuir o mesmo índice na tabela. Esta situação é conhecida como **colisão**, e existem técnicas eficientes de contornar esta situação. A escolha de uma boa função Hash é essencial para evitar estas colisões, e normalmente estas funções possuem algoritmos aleatorizados para este fim.

2.3.2 ENCADEAMENTO

No encadeamento, colisões são tratadas de forma que as chaves que possuem índice iguais na tabela Hash vão para a mesma posição que aponta para uma lista encadeada. Ou seja, todas as posições da tabela Hash possuem um apontador para NULL, quando não há colisões para aquela posição, e possui um ponteiro para o primeiro elemento da lista encadeada quando há colisões para aquela posição.

2.3.3 ANALISE DE COMPLEXIDADE

Em uma função de Hash $h(k)$ que não gera nenhuma colisão, a tabela de Hash possuirá índices distintos para toda chave que passa por $h(k)$. Logo, a tabela Hash ficará com a estrutura parecida ao de um vetor comum, porém a procura será feita também através de $h(k)$, o que demanda tempo constante. Portanto, no pior caso, a procura por uma chave em uma tabela Hash seria $O(1)$. Esta função de Hash que não gera nenhuma colisão é conhecida como **Hashing Perfeito.**

Na maioria dos casos, as funções Hash não gerarão um Hashing Perfeito, logo adotaremos a estratégia de encadeamento para contornar estas colisões. Para um Hash com encadeamento, o pior caso de execução seria quando todas as chaves colidam no mesmo índice da tabela Hash, o que demandaria tempo $O(n)$ para a mesma busca.

3. SIMULAÇÃO DE UM SISTEMA DE INTRANET ATRAVÉS DE ÁRVORES AVL

3.1 Introdução

Será mostrado nesta seção a estrutura da árvore AVL para simulação de uma intranet. A rede de intranet implementada se restringe ao simples envio de pacotes em rede através de buscas binárias pelo IPv4 do *host* e do *receiver*, logo não será implementado nenhum tipo de camada de rede, dispositivos de segurança, protocolos de rede, etc.

3.2 Estrutura do dispositivo e do pacote dentro da árvore

A estrutura geral de um dispositivo nesta simulação limita-se aos campos: *IPv4_ADDRESS,* que corresponde ao endereço de IP da máquina na rede; *DEFAULT_GATEWAY*, que corresponde ao gateway padrão onde o dispositivo está conectado; *MAC_ADDRESS*, que corresponde ao endereço físico do dispositivo; e *NAME*, que corresponde ao nome do dispositivo. Os campos de valor *Package** na estrutura do dispositivo correspondem respectivamente ao último pacote recebido e ao último pacote enviado dentro da rede.

A estrutura de um pacote que será enviado dentro da rede é formada pelos seguintes campos: *MESSAGE*, correspondente à mensagem que um *host* envia ao *receiver*; e *TIME*, que corresponde ao tempo que o pacote custou para chegar ao seu destino. A figura 1 ilustra e implementação desta estrutura.

```
struct Device {
    string IPv4_ADDRESS;
    string DEFAULT_GATEWAY;
    string MAC_ADDRESS;
    string NAME;
    Package * last_received_package;
    Package * last_sent_package;
};

struct Package {
    string message;
    long time;
};
```

Figura 1. Estrutura de um dispositivo e de um pacote

3.3 MÉTODO DE ENVIO DE PACOTES NA REDE

O método de envio de pacotes em redes consiste de quatro etapas condicionais que definem para onde o pacote será enviado. No caso de um pacote ser enviado por um *host* dentro de uma rede para um *receiver* nesta mesma rede, o que é checado pelo campo *DEFAULT_GATEWAY*. Caso o IP do *receiver* possua o mesmo gateway padrão do *host*, o pacote é enviado através de uma busca binária pelo *host* e outra pelo *receiver*, que se encontram em uma árvore AVL. No caso do *receiver* se encontrar fora da rede interna do *host*, é feito uma busca em uma tabela Hash.

No caso dos dois endereços se encontrarem na mesma rede, então os dois dispositivos estão na mesma rede interna e a estrutura de dados utilizada é uma estrutura de árvore AVL. A busca sempre é feita em $O(\lg n)$ pois a árvore é balanceada.

No caso dos dois endereços se encontrarem em redes distintas, então a estrutura de dados adotada é uma tabela Hash com encadeamento, que sempre realiza uma busca em $O(1)$ no caso médio, e $O(1+\beta)$ no pior caso, tal que β é o tamanho da lista encadeada em cada posição da tabela Hash.

A árvore AVL implementada é uma BST simples com métodos de balanceamento implementados internamente. Já a tabela Hash implementada, possui como função hash $h(x) = (\sum_i^m key[i] \,.\, pesos[i \; mod \; p]) \; mod \; m$ para a determinação do índice correspondente na tabela Hash. Temos que:

m: número de algarismos em um dado IP.

key: valor do inteiro IPv4

$pesos$: vetor de pesos gerado aleatoriamente

p: tamanho do vetor de pesos

Na criação da tabela Hash são criados dois vetores: um corresponde a tabela Hash em si, onde cada posição é uma lista encadeada que possui primeiro elemento o valor NULL; e outro vetor de pesos

onde são gerados inteiros aleatório em um intervalo de 1 a 10000. A função $h(x)$ mapeia as chaves de IPv4 que retorna um índice para a tabela Hash. Na figura 2 é mostra a implementação da função de busca de pacotes e nas figuras 3 e 4 a implementação das funções de Hash e busca em uma tabela Hash, respectivamente.

```
bool sendPackage(Hash *tabela, struct Node* root, string message, string hostIPv4, string receiverIPv4) {
    clock_t Ticks[2];
    Ticks[0] = clock();
    NoH *hostHT = NULL;
    NoH*receiverHT = NULL;
    Node* hostA = NULL;
    Node* receiverA = NULL;

    Package *pacote = criarPackage(message);

        if(tryIP(hostIPv4)){
            hostA = AVLsearch(root, ipSplit(hostIPv4));
                if(hostA == NULL){
                    return false;
                }
            setPackEnv(getDevice(hostA), pacote);
        }else{
            Device *dadosPC = criarDevice(hostIPv4, "", "", "");
            hostHT = criarNo(dadosPC);
            bool aux = searchHashItem (tabela, hostHT);
                if(!aux){
                    return false;
                }
            (getNoDevice(hostHT)->last_sent_package = pacote);
        }
            if(tryIP(receiverIPv4)){
                receiverA = AVLsearch(root, ipSplit(receiverIPv4));
                    if(receiverA == NULL){
                        cout << "Sem Conexao" << endl;
                        return false;
                    }
                setPackReceb(getDevice(receiverA), pacote);
            }else{
                Device *dadosPC = criarDevice(receiverIPv4, "", "", "");
                receiverHT = criarNo(dadosPC);
                bool aux = searchHashItem(tabela, receiverHT);
                    if(!aux){
                        cout << "Sem Conexao" << endl;
                        return false;
                    }
                (getNoDevice(receiverHT)->last_received_package = pacote);
            }

    Ticks[1] = clock();
    pacote->time = (Ticks[1] - Ticks[0]) * 1000.0 / CLOCKS_PER_SEC;
    cout << "TIME_ELAPSED " << pacote->time << "ms" << endl;
    return true;
}
```

Figura 2. Função de envio de pacotes

```
int indiceHash (Hash *dadosHash, NoH *dadosNo){ // Retorna o Indice da Chave na Tabela Hash
    int i;
    unsigned int soma = 0;
    string chave = changeInttoString(returnIP(getNoDevice(dadosNo)));
    int comp =  chave.length(); // retorna o tamanho da string chave.
        for (i = 0; i < comp; i++){
            soma += (unsigned int) chave[i] * dadosHash-> pesos[i % dadosHash-> sizePesos];
        }
    return (soma % dadosHash-> sizeListas);
}
```

Figura 3. Função *h(x)* para retorno de um índice na tabela Hash

```
NoH* searchHashNo(Hash *dadosHash, NoH *dadosNo){ // Retorna um ponteiro para o No anterior ao procurado.
    int i = indiceHash(dadosHash, dadosNo);

        if(ehLiVazia(dadosHash->vetor[i])){// é vazia
            return NULL;
        }

    NoH *anterior = buscarNoLista(dadosHash->vetor[i], getDeviceIp(getNoDevice(dadosNo)));

    return anterior;
}

bool searchHashItem (Hash *dadosHash, NoH *dadosNo){ // Se o item existir retorna o No por referencia, com todos os dados
    NoH *anterior = searchHashNo(dadosHash, dadosNo);  // retorna o no anterior do valor que busco

        if(anterior == NULL){
            return false;
        }
            if(getNoProx(anterior) != NULL){
                dadosNo = getNoProx(anterior);
            }else{
                dadosNo = anterior;
            }
    return true;
}
```

Figura 4. Função de busca em uma tabela Hash

CONCLUSÃO E TRABALHOS FUTUROS

Foi implementado neste projeto duas estruturas de dados fundamentais no estudo de computação para a simulação de uma rede de computadores hipotética. Nesta simulação não foram implementados nenhum tipo de protocolo de redes, segurança, estabelecimento de conexão ou qualquer tipo de recurso que uma rede de computadores ideal possua. O envio de pacotes é feito através de n buscas feitas em uma árvore AVL, no caso de intranet, ou em uma tabela Hash se o IP estiver na internet.

Para futuros trabalhos poderia ter sido implementado novos recursos para esta de rede de computadores, como protocolos, verificações de segurança e talvez mais importante o estabelecimento de uma nova conexão com algum dispositivo (3-way-handshake). Estes recursos também poderiam ser recursos hipotéticos, e assim poderíamos comparar estes novos recursos implementados com recursos das redes de computadores que são utilizadas atualmente.

No fim, o trabalho foi uma boa prática para a implementação destas duas estruturas de dados, que possuem papel importante em diversas ferramentas da computação, o que torna o trabalho em torno do projeto agradável do ponto de vista do aprendizado.

REFERENCIAS

[1] CORMEN, Thomas H.; LEISERSON, Charles E.; RIVEST, Ronald L.; STEIN, Clifford; Algoritmos Teoria e Prática 3ª ed.

[2] http://www.geeksforgeeks.org/hashing-data-structure/, *acessado em 24/08/2017*

[3] https://visualgo.net/en/hashtable, *acessado em 25/08/2017*

[4] http://www.geeksforgeeks.org/avl-tree-set-1-insertion/, *acessado em 23/08/2017*

[5] JAGADISH, H. V.; OOI, Beng Chin; RINARD, Martin; VU, Quang Hieu; BATON: A balanced tree structure for peer-to-peer networks