Trabalho da M2 de Estruturas de Dados: Tabela de Dispersão

Allan Nathan Baron de Souza, Israel Efraim de Oliveira, José Carlos Zancanaro

Centro de Ciências Tecnológicas da Terra e do Mar
Universidade do Vale do Itajaí (UNIVALI)
Itajaí – SC – Brasil

allanbaron98@edu.univali.br

israel.oliveira@edu.univali.br

jose.zancanaro@edu.univali.br

Resumo: *Este artigo abordará duas implementações de tratamento de colisão em tabelas de dispersão com foco na comparação e estudo de comportamento de seu desenvolvimento.*

1. Introdução

Neste artigo, introduzimos uma análise e comparação de duas implementações de tabelas de dispersão (tabela *hash*) sugeridas na disciplina de Estrutura de Dados no curso de bacharelado em Ciência da Computação. As implementações consistem em dois diferentes métodos de tratamento de colisão de índices de dispersão: o endereçamento aberto e o encadeamento de chaves.

O objetivo central é estudar o comportamento de uma tabela hash e seus diferentes tipos de implementação, analisando principalmente a complexidade e variação nas operações de inserção e consulta.

Para o desenvolvimento do trabalho, focamos em definir os princípios do tipo abstrato de dados da tabela de dispersão e demonstrar as etapas necessárias de cada operação realizada por esta. Ademais, evidenciaremos o código fonte sempre que for necessário para o auxílio da compreensão das operações.

2. Definições do Trabalho

Nesta seção são apresentadas as operações utilizadas para a Tabela Hash desenvolvidas para o Trabalho da M2 da disciplina de Estruturas de Dados.

2.1. Estrutura e Operações do Endereçamento Aberto

No método de endereçamento aberto, quando uma chave colide com outra, esta colisão é resolvida encontrando-se uma entrada da tabela hash disponível diferente da posição da qual esta chave foi dispersada. Se uma posição k dada pela função de dispersão h estiver ocupada, serão realizadas outras combinações (geralmente lineares) em uma sequência de sondagem até que uma célula disponível seja encontrada, considerando que a tabela não tenha alcançado o limite de dados.

2.1.1. Estrutura

Na estrutura de uma tabela de dispersão com resolução por endereçamento aberto, utiliza-se um vetor de N posições composto por elementos capazes de armazenar uma chave do tipo inteiro e um dado. Opcionalmente, podemos armazenar um contador de inserções e a função de hash da tabela (qual função será chamada para dispersar os elementos na tabela).

```
template <typename T>
struct TElementoIndexado
{
    int chave;
    T dado;
    bool ocupado;
};

template <typename T, int MAX>
struct TabelaHashAberta
{
    TElementoIndexado<T> elementos[MAX];
    int quantidade;
    std::function<int(int,int)> hash;
};
```

Para a tabela acima, definimos também um estado de ocupação para cada elemento na tabela para que seja possível identificar facilmente entradas não utilizadas. Outra aproximação para este problema seria transformar o vetor da tabela em um vetor de ponteiros do tipo *TelementoIndexado*, permitindo-nos identificar uma entrada disponível quando o endereço apontado for nulo.

2.1.2. Inicialização

No procedimento de inicialização de uma tabela hash de endereçamento aberto, devemos apenas definir os valores iniciais dos componentes da tabela, isto significa inicializar a contagem de inserções no valor numérico 0, definir todos os elementos como não ocupados e associar a função de dispersão à tabela.

```
template <typename T, int MAX>
void inicializarTabelaHash(TabelaHashAberta<T, MAX> &tabela,
                           std::function<int(int,int)> funcao = hashModular)
{
    tabela.quantidade = 0;
    tabela.hash = funcao;
    for (int i = 0; i < MAX; i++)
        tabela.elementos[i].ocupado = false;
}
```

Não precisamos redefinir os valores de chaves e dados dos elementos da tabela, porque a partir da *flag* de ocupação de cada um, conseguimos identificar se o valor presente num determinado espaço de memória é apenas resto de outra aplicação anterior ou um valor definido pelo usuário.

2.1.3. Inserção

Nesta função, realizamos a inserção de um elemento na tabela através da busca pela primeira posição disponível indicado pelo índice de dispersão. Como comentado anteriormente, encontramos o índice inicial de pesquisa através da função de dispersão da tabela e realizamos uma sondagem para encontrar uma entrada disponível.

O tipo de sondagem mais simples é a linear, onde a posição na qual uma chave pode ser armazenada é encontrada procurando-se sequencialmente todas as posições a partir do índice indicado pela função hash até que uma posição vazia seja encontrada. Se o limite da tabela é atingido, a busca é reiniciada no começo da tabela e tem como limite a posição onde a busca começou anteriormente.

Podemos codificar o contexto supracitado da seguinte forma:

```
template <typename T, int MAX>
bool inserirEmTabelaHash(TabelaHashAberta<T, MAX> &tabela,
                         int chave, T dado)
{
    if (tabela.quantidade < MAX) {
        int posicao = tabela.hash(chave, MAX), limiteBusca = 0;
        for (; tabela.elementos[posicao].ocupado && limiteBusca < MAX;
               posicao = tabela.hash(posicao + 1, MAX)) {
            if (tabela.elementos[posicao].chave == chave)
                return false;
            limiteBusca++;
        }
        tabela.elementos[posicao].chave = chave;
        tabela.elementos[posicao].dado = dado;
        tabela.elementos[posicao].ocupado = true;
        tabela.quantidade++;
        return true;
    }
    return false;
}
```

No caso de uma chave repetida, não faria sentido inseri-la na tabela, pois a pesquisa, que explicamos a seguir no artigo, retornará a partir do primeiro elemento encontrado que tenha a chave correspondente à busca, portanto, mesmo que a armazenamos, nunca seria acessada a menos que seja realizada uma exibição total da tabela.

2.1.4. Consulta

A operação de consulta numa tabela de endereçamento aberto funciona da mesma forma que a inserção, a diferença é que ao invés de procurar uma posição disponível a partir do índice de dispersão, procuramos o elemento com a chave indicada até encontrar um espaço vazio, que indica a inexistência da chave na tabela.

Desta forma, temos a seguinte implementação:

```
template <typename T, int MAX>
T* consultarChaveEmTabelaHash(TabelaHashAberta<T, MAX> &tabela,
                              int chave)
{
    if (tabela.quantidade != 0) {
        int posicao = tabela.hash(chave, MAX), limiteBusca = 0;
        for (; tabela.elementos[posicao].ocupado && limiteBusca < MAX;
               posicao = tabela.hash(posicao + 1, MAX)) {
            if (tabela.elementos[posicao].chave == chave) {
                return &tabela.elementos[posicao].dado;
            }
            limiteBusca++;
        }
        return nullptr;
    }
    return nullptr;
}
```

Como é possível notar, utiliza-se do mesmo tipo de sondagem que a inserção para obter um resultado otimizado, caso contrário, é provável que a pesquisa tenha uma performance não desejada.

Utilizamos o conceito de retornar um ponteiro para o dado em vez do dado propriamente, pois possibilita que o usuário saiba se o valor foi encontrato ou não e permite-nos trabalhar com tipos variantes na função (templates). Como não sabemos que tipo de dado será utilizado com a tabela, não há uma forma de definir um valor nulo para todo tipo de dado senão utilizar um ponteiro para o dado e retornar um ponteiro nulo.

2.2. Estrutura e Operações do Lista Encadeada

Neste método de tratamento de colisão, utiliza-se do conceito de listas encadeadas para melhorar a dispersão dos elementos na tabela. Cada posição do vetor está associado com uma lista encadeada (cadeia de estruturas), cujo dado pode ser consultado através de um campo específico de cada membro.

Não se faz necessário utilizar meios de sondagem neste método, tratamos todas as inserções através do posicionamento no final da lista encadeada correspondente ao índice de dispersão de uma determinada chave. A diferença, para todos os casos, é a quantidade de membros a ser percorrido para chegar ao final da lista, dependendo claramente do tamanho de cada lista.

2.2.1. Estrutura

Assim como no endereçamento aberto, precisamos de um tipo de dado com capacidade de armazenar uma chave e um dado, mas, neste caso, adicionaremos também um ponteiro para um próximo elemento do mesmo tipo, para criar a estrutura de encadeamento necessária para este método de tratamento de colisão.

Quanto à tabela, é composta por um vetor de ponteiros, onde cada posição armazena o endereço do início de uma cadeia de estruturas, correspondente ao índice de dispersão. Além disto, armazenamos a quantidade de elementos já inseridos e a função hash da tabela.

```
template <typename T>

struct TNodoIndexado
{
    int chave;
    T dado;
    TNodoIndexado<T> *proximo;
};

template <typename T, int MAX>

struct TabelaHashEncadeada
{
    TNodoIndexado<T> *elementos[MAX];
    int quantidade;
    std::function<int(int,int)> hash;
};
```

Os nodos encadeados poderiam opcionalmente ter as informações de chave e dado separadas em outra estrutura, contudo, utilizamos esta notação para simplificar as implementações.

2.2.2. Inicialização

Antes de poder utilizar a tabela, há a necessidade de inicializar todos os componentes desta, definindo sua quantidade como 0 e todos os elementos como nulo. Além disto, associa-se a função de dispersão da tabela.

```
template <typename T, int MAX>
void inicializarTabelaHash(TabelaHashEncadeada<T, MAX> &tabela,
         std::function<int(int,int)> funcao = hashModular)
{
    tabela.quantidade = 0;
    tabela.hash = funcao;
    for (int i = 0; i < MAX; i++)
        tabela.elementos[i] = nullptr;
}
```

2.2.3. Inserção

Para inserir um novo elemento, devemos apenas buscar o índice de dispersão dado pela chave e pela função hash da tabela e, em seguida, encontrar o fim da lista na posição determinado. Caso a lista no índice de dispersão estiver nula, é necessário definir o novo elemento como o início da lista deste mesmo índice.

Implementa-se o contexto acima da seguinte forma:

```
    template <typename T, int MAX>
    bool inserirEmTabelaHash(TabelaHashEncadeada<T, MAX> &tabela, int chave,
                             T dado)
    {
        TNodoIndexado<T> *novo = novoTNodoIndexado(chave, dado);
        if (novo != nullptr) {
            int indiceInsercao = tabela.hash(chave, MAX);
            TNodoIndexado<T>    *nav    =    tabela.elementos[indiceInsercao],
*inserirEm = nullptr;
            if (nav == nullptr) tabela.elementos[indiceInsercao] = novo;
            else {
                while (nav != nullptr) {
                    inserirEm = nav;
                    if (nav->chave == chave) {
                        delete novo;
                        return false;
                    }
                    nav = nav->proximo;
                }
                inserirEm->proximo = novo;
            }
            tabela.quantidade++;
            return true;
        }
        return false;
    }
```

Assim como no endereçamento aberto, não permitimos chaves idênticas porque não seriam recuperáveis se buscadas individualmente.

2.2.4. Consulta

Para pesquisar um dado pela chave, basta encontrar o índice de dispersão da chave e procurar a chave até encontrar o fim da lista. Se chegarmos no fim e a chave ainda não tiver sido encontrada, significa que ela não existe na tabela.

```
    template <typename T, int MAX>
    T*  consultarChaveEmTabelaHash(TabelaHashEncadeada<T,  MAX>  &tabela,  int
chave, int &instrucoes)
    {
        int indiceInsercao = tabela.hash(chave, MAX);
        TNodoIndexado<T> *procurar = tabela.elementos[indiceInsercao];
        while (procurar != nullptr) {
            if (procurar->chave == chave) return &procurar->dado;
            procurar = procurar->proximo;
        }
        return nullptr;
    }
```

3. Comparativo

Nesta seção, apresentamos testes comparativos realizados com ambas as implementações de tabelas de dispersão. O teste concentra-se na inserção e consulta sequenciais de números aleatórios de 1 à 32000 nas tabelas delimitadas com 1000 entradas de dados.

Presumimos que os testes serão capazes de demonstrar as variações de comportamento e evidenciar as diferenças entre as implementações.

Ressaltamos que os testes, de caráter aleatório, podem não concretizar o pior caso de execução de ambas as tabelas, pois, para o método de encadeamento, este seria categorizado por inserções de elementos com índices de dispersão iguais, já para o método de endereçamento aberto, podemos facilmente demonstrá-lo através de inserções quando a tabela estiver próximo do limite de dados.

Na tabela 1, realizamos um comparativo entre o pior caso de tempo de execução das duas implementações da tabela de dispersão.

Na inicialização das tabelas, ambas precisam percorrer todos os elementos do vetor e, portanto, para a quantidade de elementos *n*, o procedimento deverá conter *n* instruções. Para inserir um elemento, o caso médio levaria tempo constante dependendo da função de dispersão, contudo, no pior caso, ambas sofrem *n* iterações, já que não há jeito de encontrar a posição de inserção sem realizar uma busca linear.

Na pesquisa, notamos o mesmo problema para ambas as implementações: na tabela aberta, a dispersão pode não ser muito bem respeitada devido ao método de endereçamento e uma chave pode ser deslocada para o pior local de sua recuperação, e portanto, teríamos de iterar a tabela inteira para achá-la; na tabela de encadeamento, teríamos de buscar através de *n* elementos de mesma chave no pior caso.

Tabela 1. Comparativo entre o pior caso

	Endereçamento Aberto	**Lista Encadeada**
Inicializa	n	n
Inserir	n	n
Pesquisar	n	n

Na Tabela 2, demonstramos a média de operações realizadas para inserir um elemento em determinados estados das tabelas.

No endereçamento aberto, a média de tempo de execução é geralmente constante, com exceção do momento em que a quantidade de elementos inseridos aproxima-se do limite da tabela, onde há um crescimento demasiado, devido a quantidade de elementos a serem iterados. Já no método de encadeamento, a dispersão por chaves é sempre respeitada e a inserção continua sempre próxima de um tempo constante.

Tabela 2. Média de operações na inserção por taxa de ocupação

	Endereçamento Aberto	Lista Encadeada
10%	1,4	1
25%	1,8	1,8
75%	4,4	1,7
100%	478,7	1,5
200%	1	1,7

Na Tabela 3, representamos a média de crescimento de operações realizadas com base nos estados da tabela.

Como era de se esperar, ambas as implementações expressam crescimento, principalmente a tabela de endereçamento aberto, pois em estados de ocupação maiores não reflete o dispersamento e a busca é sempre linear como em qualquer vetor; a tabela de encadeamento, por sua vez, demonstra menos crescimento no tempo de execução para o caso de muitas inserções, que não a afeta muito, já que seu pior caso é quando as chaves inseridas possuem um mesmo índice de dispersão.

Tabela 3. Média de operações na pesquisa por taxa de ocupação

	Endereçamento Aberto	Lista Encadeada
10%	1,2	1,1
25%	1,3	1,3
75%	3,7	1,7
100%	565,6	1,4
200%	1000,00	2,5

Na Figura 1, é possível visualizar um gráfico comparativo entre o crescimento de tempo de execução das tabelas de endereçamento aberto e encadeado. Assim como foi evidenciado acima, o método de encadeamento apresenta tempos similarmente constantes, enquanto que o endereçamento aberto demonstra crescimento em estados próximos do limite de quantidade da tabela (não se observa o mesmo comportamento quando a tabela está cheia, pois efetuamos uma verificação de estado para evitar trabalho desnecessário).

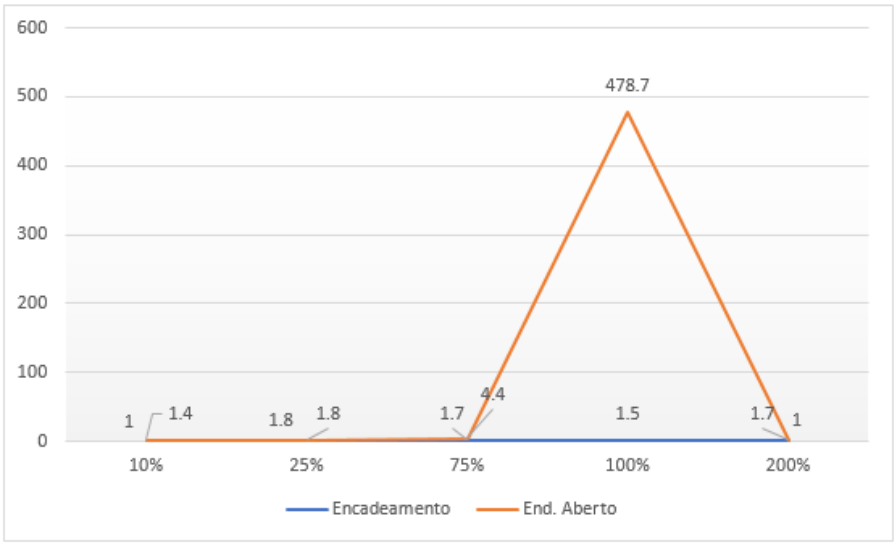

Figura 1. Gráfico da média operações na inserção por taxa de ocupação

Na Figura 2, observamos uma representação dos tempos de execução nas operações de pesquisa. É possível notar que não há muita variação para o método de encadeamento porque as chaves são distribuídas de acordo com sua função de dispersão, à medida que o endereçamento aberto permite que as chaves sejam distribuídas sem restrição de dispersão, tendo como consequência, tempos elevados para restauração.

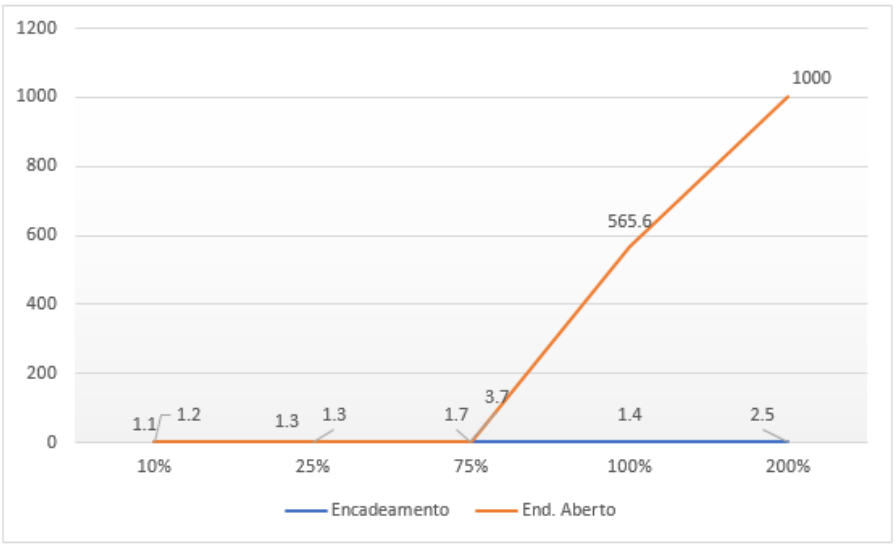

Figura 2. Gráfico da média de operações na pesquisa por taxa de ocupação

4. Conclusões

Com base nos respectivos testes e representações gráficas, podemos afirmar que a complexidade do pior caso para ambas as implementações de tabela de dispersão são iguais, contudo, são pouco aparentes pois os piores casos se diferem.

O método de endereçamento aberto tende a criar agrupamento de chaves que podem se tornar um problema para o bom desempenho da tabela, isto é, quando chaves com mesmo índice de dispersão entram em colisão, a sondagem linear encaixa-as na tabela em índices disponíveis, mas, acabam ocupando espaço de outros índices de dispersão, que, por sua vez, têm de ocupar mais espaços e causam agrupamentos (espaços vazios depois do agrupamento têm mais chances de serem preenchidos com chaves que não fazem parte do seu índice de dispersão), que configuram uma tabela que perde a capacidade de operar em tempo constante, pois os índices de dispersão já não são mais respeitados. Em suma, se um agrupamento é criado, ele tende a crescer e quanto maior ele se torna, maior a possibilidade de que ficará ainda maior e este fato diminui o desempenho da tabela de dispersão. Existem tipos de sondagem que reduzem este efeito, mas, quando há a possibilidade de uma grande quantidade de colisões, o endereçamento aberto pode não ser uma escolha ideal.

O método de endereçamento encadeado evita que agrupamentos sejam criados e por isto pode otimizar o desempenho da tabela, pois os elementos são encadeados no índice de dispersão, portanto, mesmo que índices possam colidir, o efeito não é propagado e o tempo de consulta continua relativamente rápido. O único problema é quando a fraqueza desta tabela é explorada, isto é, se muitos índices colidirem (porque provavelmente a quantidade de entradas é pequena) o encadeamento nas posições de dispersão ficarão maiores e, o desempenho pode diminuir porque a performance de varredura em listas encadeadas é menor que a de um simples vetor. Além disto, o encadeamento necessita de mais memória física para ocorrer, pois para cada dado inserido, necessitamos de um ponteiro para referencia-lo. Em consequência, para n chaves, são necessários n + *quantidade de entradas da tabela* ponteiros, que pode ser muito caso o n seja grande.

De forma geral, definimos o comportamento da implementação de um método de tratamento de colisão como volátil e dependente do conhecimento que se tem das chaves que serão inseridas quanto a previsibilidade dos índices de dispersão, pois dado um alto nível de colisão, é possível que haja grande perda de desempenho com a escolha incorreta de tratamento.