Trabalho Prático de Algoritmos e Estrutura de dados II
Hashbin

1 Introdução

Em grandes empresas e ambientes de pesquisa que lidam com um grande volume de dados, a implementação de estrutura de pesquisa que tenha um rápido retorno de informações com o menor custo possível é essencial para otimizar o ambiente de trabalho.

O objetivo principal deste trabalho é modelar e implementar uma estrutura de dados capaz de criar uma estrutura de pesquisas eficientes usando como base o hash com o tratamentos de colisão utilizando árvores binárias de pesquisa. A proposta é dar ao usuário a opção de escolher um tamanho M_1 da tabela hash e logo em seguida a opção de escolher o número de elementos N_1 a serem inseridos na tabela hash. Logo em seguida o usuário deve digitar os N_1 elementos a serem inseridos. É também dado ao usuário a opção de escolher um índice M_1 da tabela hash para ser impresso na tela usando o caminhamento pré-ordem na tela. O objetivo secundário é realizar testes de complexidade de tempo e de espaço, avaliando a eficácia desses algoritmos em relação ao hashing com o método de resolução de colisão com a utilização de uma lista encadeada.

2 Solução do Problema

A estrutura utilizada para a implementação foi a do hashing com o método de resolução de colisão com a utilização de árvores binárias de pesquisa. Na implementação do hashin foi a função $f(x) = x \bmod M_1$, onde x e o elemento a ser inserido e M_1 e número de índices da tabela.

A avaliação de complexidade de tempo foi feita em termos assintóticos utilizando a notação O. Já a avaliação de complexidade de espaço foi considerada apenas a alocação e/ou movimentação de memória.

Complexidade de Tempo:

A complexidade de tempo da operação de inserção em uma árvore binária e $\log(N_1)$, onde N_1 e o número de elementos na árvore. Logo assumindo que a probabilidade de um elemento ser inserido em cada índice M_1 da tabela ser igual, temos que a complexidade de tempo da operação e de $\log(N_1/M_1)$ ou $O(\log(N_1/M_1))$. Já a complexidade de tempo da hash é sempre 1 ou O(1).

Utilizando a definição soma O(f(n)) + O(g(n)) = O(max(f(n), g(n))) é selecionado o termo de maior crescimento para definir o comportamento assintótico da função chegando a complexidade de $O(\log(N_1/M_1))$:

$$\mathbf{O(\log(N_1/M_1)) + O(1) = O(\max(\log(N_1/M_1), 1)) = O(\log(N_1/M_1))}$$

Já a operação de impressão dos elementos e de $O(N_1)$, sendo que há a necessidade de passar em todos os nós da árvore. Logo também assumindo que a probabilidade de um elemento ser inserido em cada índice M_1 da tabela ser igual, temos que a complexidade de tempo da operação e de N_1/M_1 ou $O(N_1/M_1)$.

Com a utilização da definição de soma assintótica e novamente levando em consideração a complexidade de tempo da hash sendo sempre 1 ou O(1). temos a complexidade de $O(N_1/M_1)$:

$$\mathbf{O(N_1/M_1) + O(1) = O(\max(N_1/M_1, 1)) = O(N_1/M_1)}$$

Complexidade de espaço:

Na implementação do rash e utilizada a tad árvore com os seguinte atributos:

```
struct arvore{
        No *raiz;
};
```

```
struct no{
        int chave;
        struct no *esq;
        struct no *dir;
};
```

Legenda: A = arvore, pA= ponteiro para arvore, M_1 = tamanho da tabela hash, N_1 = número de elementos, K = no.

1. Primeiramente é criado um array com M_1 ponteiros para a árvore.
   M_1*pA
2. Logo após são criadas M_1 árvores.
   M_1*A
3. E depois são inseridos N_1 elementos na tabela hash.
   N_1*K

A soma final de todas as alocações de memória da tabela hash temos M_1*pA + M_1*A + N_1*K, ou seja, temos uma complexidade de espaço de:

M_1*ponteiro pra arvore + M_1*arvore + N_1*no

3 Análise Experimental

Para os testes a seguir foram utilizados os seguintes vetores com tamanho Mil, 10 Mil, 100 Mil e 1 Milhão, sendo eles aleatórios sem elementos repetidos, e foi utilizado o número de índices da tabela hash com os seguintes valores: 10, 20, 100.

No primeiro teste é apresentado os resultados do hashing com o método de resolução de colisão com a utilização de árvores binárias de pesquisa binária.

Elementos Indices	Mil	10 Mil	100 Mil	1 Milhão
10	0.0196	0.0827	1.0012	11.7429
20	0.0068	0.0773	0.9504	12.0918
100	0.0058	0.0687	0.8640	11.0521

Tabela 1(T^1)

No segundo teste é apresentado os resultados do hashing com o método de resolução de colisão com a utilização de uma lista encadeada.

Elementos Indices	Mil	10 Mil	100 Mil	1 Milhão
10	0.0132	0.0348	0.3754	3.8185
20	0.0037	0.0355	0.3664	3.7599
100	0.0041	0.0352	0.3678	3.7837

Tabela 2(T^2)

*As análises a seguir foram feitas levando em consideração a implementação da lista encadeada feita em aula síncrona e adaptada para a aplicação com os resultados apresentados na $\mathbf{T^2}$. Com a complexidade assintótica de tempo das operações de inserção e impressão sendo respectivamente O(1) e $O(N_1/M_1)$, e com a complexidade de espaço sendo iguais com a ressalva de trocar as variáveis A e pA para L = Lista e pL = ponteiro para lista.

Com a análise das tabelas $\mathbf{T^1}$ e $\mathbf{T^2}$, tem-se a comprovação das análises assintóticas dos algoritmos utilizados, em todos os testes eles apresentam o crescimento esperado, variando apenas quanto a o aumento do número N_1 de elementos e número M_1 de índices da tabela hash.

A $\mathbf{T^2}$ apresenta melhores comportamentos em relação a $\mathbf{T^1}$ provavelmente devido a complexidade assintótica da operação de inserção de elementos que apresenta uma complexidade constante O(1) contra uma complexidade logarítmica $O(\log(N_1/M_1))$ respectivamente.

No $\mathbf{G^1}$ é apresentado o tempo gasto para cada interação em relação ao número de elementos N_1 e o número de índices M_1 utilizados. O número N_1 de elementos vetores utilizados foi de Mil, 10 Mil e 100 Mil com o número M_1 de índices de 10, 20 e 100. Também é utilizado o tempo de ambas as implementações, sendo elas o hashing com o método de resolução de colisão com a utilização de árvores binárias de pesquisa binária e o hashing com o método de resolução de colisão com a utilização de uma lista encadeada, representados respectivamente por A e B.

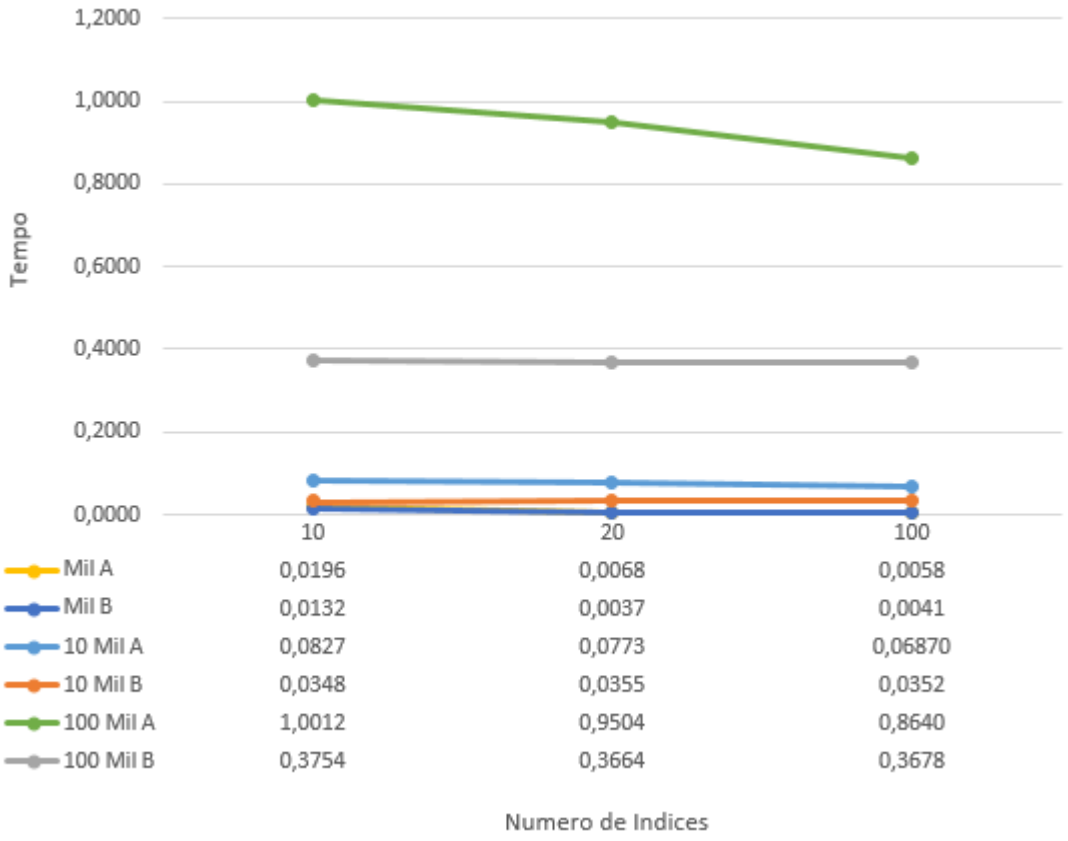

Gráfico 1(G^1)

4 Conclusão

Em todos os casos testes os algoritmos apresentaram o comportamento esperado. Para o problema apresentado e as soluções implementadas, conclui-se que a implementação do hashing com o método de resolução de colisão com a utilização de uma lista encadeada apresenta o melhor comportamento em todos os casos. Caso no futuro a aplicação venha a ter a funcionalidade de pesquisa com pouca ou nenhuma alteração nos dados, a implementação do hashing com o método de resolução de colisão com a utilização de árvores binárias de pesquisa binária se tornará a melhor opção com a análise de sua complexidade assintótica, sendo ela $O(\log(N^1/M^1))$ contra $O(N^1/M^1)$ da implementação com lista encadeada.

Após todos os apresentados, conclui-se que o trabalho apresenta uma aplicação funcional e um comportamento estável com uma implementação compacta e de fácil entendimento.

5 Bibliografia

1. Slides da disciplina de algoritmo e estrutura de dados II.
2. CORMEN, Thomas H. Algoritmos teoria e prática. 3 edição. São Paulo GEN LTC 2012.
3. CORMEN, Thomas H. Desmistificando algoritmos. Rio de Janeiro GEN LTC 2013.