Estruturas de Dados II
Segundo projeto

Adriana Marcolino de Souza, 10295047 Caio Oliveira da Silva, 9390301
Caroline Santos Corrêia, 9874735

31 de julho de 2021

1 Introdução

Como proposto, implementamos sete algoritmos de busca: busca sequencial simples; busca sequencial com realocação *mover-para-frente*; busca sequencial com realocação por *transposicação*; busca sequencial com índice primário; busca com espalhamento estático e fechado usando *overflow progressivo* como *rehash*; busca com espalhamento estático e fechado usando *hash duplo* como *rehash*; e busca com espalhamento estático e aberto usando listas encadeadas. Pretendemos analisar tais algoritmos do ponto de vista empírico, isto é, observando quanto tempo foi gasto para inserir e buscar os itens desejados. As análises são feitas a partir de gráficos e tabelas.

2 Comentários sobre a implementação dos algoritmos

Tendo em vista a melhor medição de tempo de modo a não tornar o processo muito demorado, cada algoritmo de busca foi executado cinco vezes para obter um desvio padrão e uma média mais acurada de tempo. Vale a pena mencionar também que o algoritmo foi executado após o computador ter sido reiniciado e nenhum outro programa foi usado pelos membros do grupo durante o processo todo. Para o desenvolvimento do projeto foi utilizado o editor de texto Emacs, mas o algoritmo em si foi compilado e executado em um terminal para eliminar a influência que o editor causaria na medição de tempo. Os gráficos foram feitos usando o programa LibreOffice Calc.

Vale a pena mencionar que poucas alterações foram feitas no *template* fornecido além das funções implementadas para a realização dos métodos de busca e inserção em si. Uma função adicionada nos algoritmos de busca por espalhamento foi a `desalocar_strings`, pois as strings de inserção e busca não estavam sendo desalocadas em nenhum momento no *template*. E, nos algoritmos de busca sequencial, também liberamos a memória dos vetores de inserção e busca. Além dessas duas alterações, acrescentamos o cálculo de tempo de cada inserção e cada busca isoladamente. No entanto, como a variação dessas buscas e inserções foram extremamente pequenas, decidimos analisar apenas as buscas e inserções de todos os elementos. Por fim, na implementação do algoritmo de espalhamento usando listas encadeadas, transferimos algumas declarações para o `.h` do nosso TAD.

3 Buscas sequenciais

Em todas as buscas sequenciais foram encontrados 35557 itens ao todo.

3.1 Busca sequencial simples

Como comentado anteriormente, todos os algoritmos foram executados cinco vezes e os resultados obtidos para a busca sequencial simples foram:

Execução	Busca de cada item (em s)	Busca de todos os itens (em s)
1	0.000128	6.428736
2	0.000128	6.428786
3	0.000128	6.432853
4	0.000128	6.428955
5	0.000128	6.428596

Como comentado na seção anterior, a busca por cada item sofreu variações extremamente pequenas e, portanto, calculamos apenas a média e o desvio padrão de todas as 50000 buscas feitas. Obtivemos o seguinte resultado:

Média da busca (em s)	Desvio padrão da busca
6.4295852	0.0016379

3.2 Busca sequencial com realocação *mover-para-frente*

Também realizamos cinco medições independentes para a busca com realocação *mover-para-frente* e o resultado obtido foi:

Execução	Busca de cada item (em s)	Busca de todos os itens (em s)
1	0.000128	6.423273
2	0.000128	6.423453
3	0.000128	6.423507
4	0.000128	6.423299
5	0.000128	6.423492

A média e o desvio padrão de todas as buscas foram:

Média da busca (em s)	Desvio padrão da busca
6.4234048	0.0000989

3.3 Busca sequencial com realocação por *interpolação*

O resultado obtido para as cinco medições do método de busca com realocação por *interpolação* foi:

Execução	Busca de cada item (em s)	Busca de todos os itens (em s)
1	0.000128	6.429020
2	0.000128	6.429268
3	0.000128	6.429413
4	0.000128	6.429205
5	0.000128	6.429157

A média e o desvio padrão de todas as buscas foram:

Média da busca (em s)	Desvio padrão da busca
6.4292126	0.0001292

3.4 Busca sequencial com índice primário

O resultado obtido nas medições da busca sequencial com índice primário foi:

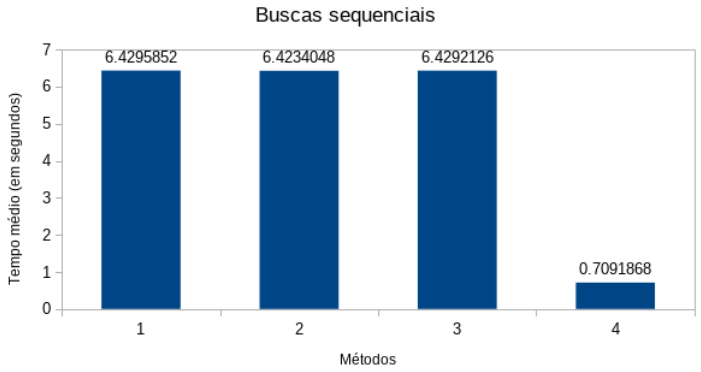
Execução	Busca de cada item (em s)	Busca de todos os itens (em s)
1	0.0000141	0.709216
2	0.0000141	0.709256
3	0.0000141	0.709171
4	0.0000141	0.709176
5	0.0000141	0.709115

A média e o desvio padrão de todas as buscas foram:

Média da busca (em s)	Desvio padrão da busca
0.7091868	0.0000472

3.5 Conclusão busca sequencial

Como pode ser visto no gráfico a seguir, a busca por índice primário foi, de longe, o melhor método de busca dentre todos os métodos sequenciais:



No gráfico acima o eixo x representa os métodos, com o número 1 sendo a busca sequencial simples, o número 2 sendo a busca sequencial com realocação *mover-para-frente*, o número 3 sendo a busca sequencial com realocação por *interpolação*, e o número 4 sendo a busca sequencial com índice primário.

4 Busca por espalhamento

Em todas as buscas por espalhamento foram encontrados 38344 itens ao todo.

4.1 Busca por espalhamento com *overflow progressivo*

O resultado obtido nas medições da busca por espalhamento com *overflow progressivo* usando a função de divisão como *rehash* foi:

Execução	Busca de cada item (em s)	Busca de todos os itens(em s)	Inserção de cada item (em s)	Inserção de todos os itens(em s)
1	0.0000018	0.129436	0.0000012	0.062418
2	0.0000018	0.129285	0.0000012	0.062337
3	0.0000018	0.129819	0.0000012	0.062395
4	0.0000018	0.129446	0.0000012	0.062667
5	0.0000018	0.129270	0.0000012	0.062272

A média e o desvio padrão de todas as buscas e todas as inserções foram:

Média da busca (em s)	Desvio padrão da busca	Média da inserção (em s)	Desvio padrão da inserção
0.1294512	0.0001979	0.0624178	0.0001344

O resultado obtido nas medições da busca por espalhamento com *overflow progressivo* usando a função de multiplicação como *rehash* foi:

Execução	Busca por item (em s)	Busca total (em s)	Inserção de cada item (em s)	Inserção de todos os itens (em s)
1	0.0000090163	0.633698	0.0000057533	0.288440
2	0.0000091794	0.630591	0.0000057555	0.287918
3	0.0000090480	0.634586	0.0000058628	0.287950
4	0.0000090056	0.630730	0.0000057473	0.287339
5	0.0000090055	0.633404	0.0000057610	0.286887

A média e o desvio padrão de todas as buscas e todas as inserções foram:

Média da busca (em s)	Desvio padrão da busca	Média da inserção (em s)	Desvio padrão da inserção
0.6326018	0.0016327	0.2877067	0.0005382

E, por fim, o número de colisões obtido em cada método foi:

Colisões com divisão	Colisões com multiplicação
1575740	2825968

4.2 Busca por espalhamento com *rehash duplo*

O resultado obtido nas medições da busca por espalhamento com *rehash duplo* foi:

Execução	Busca por item (em s)	Busca total (em s)	Inserção de cada item (em s)	Inserção de todos os itens (em s)
1	0.0000046	0.325696	0.0000031	0.158653
2	0.0000046	0.333060	0.0000031	0.159839
3	0.0000046	0.337063	0.0000031	0.159238
4	0.0000046	0.327233	0.0000031	0.158988
5	0.0000046	0.333491	0.0000031	0.159110

A média e o desvio padrão de todas as buscas e todas as inserções foram:

Média da busca (em s)	Desvio padrão da busca	Média da inserção (em s)	Desvio padrão da inserção
0.3313086	0.0042203	0.1591656	0.0003888

E, por fim, o número de colisões obtido em cada método foi:

Colisões
939779

4.3 Busca por espalhamento com *listas encadeadas*

O resultado obtido nas medições da busca por espalhamento usando listas encadeadas e com a função de divisão como *hash* foi:

Execução	Busca por item (em s)	Busca total (em s)	Inserção de cada item (em s)	Inserção de todos os itens (em s)
1	0.00000084	0.059410	0.00000065	0.032019
2	0.00000084	0.059654	0.00000065	0.032043
3	0.00000084	0.059840	0.00000065	0.032126
4	0.00000084	0.059822	0.00000065	0.032269
5	0.00000084	0.058714	0.00000065	0.031989

A média e o desvio padrão de todas as buscas e todas as inserções foram:

Média da busca (em s)	Desvio padrão da busca	Média da inserção (em s)	Desvio padrão da inserção
0.059488	0.0004166	0.0320892	0.0001007

O resultado obtido nas medições da busca por espalhamento usando listas encadeadas e com a função de multiplicação como *hash* foi:

Execução	Busca por item (em s)	Busca total (em s)	Inserção de cada item (em s)	Inserção de todos os itens (em s)
1	0.0000025	0.182911	0.0000017	0.087233
2	0.0000025	0.184613	0.0000017	0.087275
3	0.0000025	0.184302	0.0000017	0.087303
4	0.0000025	0.186566	0.0000017	0.087809
5	0.0000025	0.181726	0.0000017	0.087081

A média e o desvio padrão de todas as buscas e todas as inserções foram:

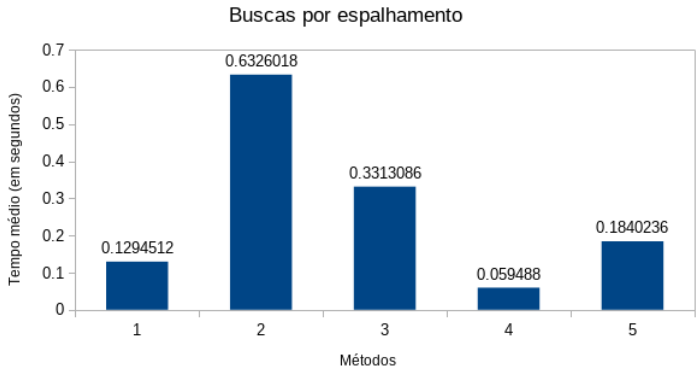
Média da busca (em s)	Desvio padrão da busca	Média da inserção (em s)	Desvio padrão da inserção
0.1840236	0.0016374	0.0873402	0.0002466

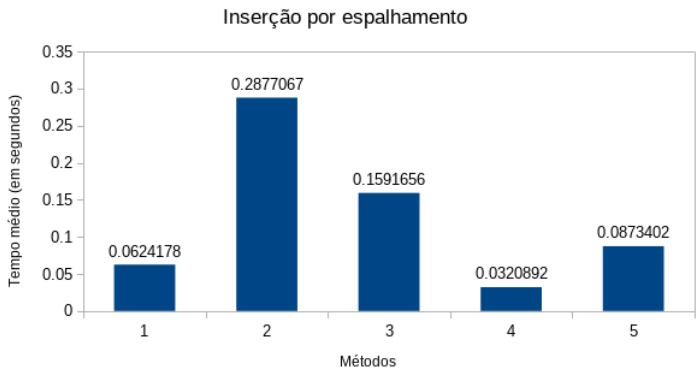
E, por fim, o número de colisões obtido em cada método foi:

Colisões com divisão	Colisões com multiplicação
39582	49495

4.4 Conclusão busca e inserção por espalhamento

Como pode ser visto nos gráficos a seguir, a busca por espalhamento usando listas encadeadas e função *hash* de divisão foi a vencedora em todos os três aspectos: busca, inserção e colisão.





Nos gráficos acima o eixo *x* representa os métodos, com o número 1 sendo a busca por espalhamento usando *overflow progressivo* com função *hash* dada pela divisão, o número 2 sendo a busca por espalhamento usando *overflow progressivo* com função *hash* dada pela multiplicação, o número 3 sendo a busca por espalhamento com *hash duplo*, o número 4 sendo a busca por espalhamento usando listas encadeadas com *hash* dado pela divisão, e o número 5 a busca por espalhamento usando listas encadeadas com *hash* dado pela multiplicação.

5 Conclusões

Como pudemos ver, o processo de busca é muito mais rápido que o de ordenação de vetores e listas, pois a busca mais ingênua tem complexidade $O(n)$ ao invés de $O(n^2)$. Dentre as buscas sequenciais, a que mais se destacou, de longe, foi a busca que utiliza índice primário, o que faz sentido, pois foi necessário realizar a busca em apenas $1/5$ do vetor de busca ao invés de no vetor todo. As buscas por interpolação não trouxeram uma grande melhoria com relação à busca sequencial simples. Uma das possíveis explicações é que tais buscas acabam se tornando muito eficiente quando buscamos muito um conjunto pequeno de valores, visto que tais valores ficarão sempre no começo do vetor. No entanto, o conjunto de buscas fornecido era muito grande e variado, o que provavelmente acabou fazendo com que não houvesse um padrão bom que tornasse tais buscas mais eficiente.

Embora uma análise mais precisa não tenha sido feita, foi possível ver empiricamente que todos os métodos de busca por espalhamento venceram o método de busca sequencial com índice primário, que havia sido o melhor método dentre os de busca sequencial. O destaque principal foi para a variação que utiliza listas encadeadas e função *hash* dada pela divisão. Em geral, a função de divisão obteve menos colisões que a função de multiplicação, mas provavelmente a função vencedora foi a do *hash duplo*, que mistura as duas funções. Ou seja, se tivéssemos usado a mesma função *hash* que foi utilizada no método de *hash duplo* dentro do método que utiliza listas encadeadas, provavelmente essa versão seria a vencedora em todos os três aspectos analisados: colisões, buscas e inserções. Vale a pena comentar também que o número de colisões no caso das listas encadeadas foi bem menor, pois apenas uma colisão é contabilizada por inserção, visto que logo que uma colisão ocorre o item já é inserido na lista. Nas outras variações nós contabilizamos todas as colisões obtidas antes de a *string* ter sido de fato armazenada no vetor. Por fim, na nossa implementação de lista encadeada demos preferência para a inserção, colocando a nova informação no início da lista ao invés de no final ou em alguma posição específica. No entanto, isso pode afetar o desempenho da busca, caso os primeiros valores a serem inseridos sejam mais buscados no futuro.