Universidade Federal de Juiz de Fora

Instituto de Ciências Exatas

Departamento de Ciência da Computação

DCC012
ESTRUTURAS DE DADOS II

Trabalho 2 - Busca de Tweet

Felipe Barra Knop - 201565553C

Lohan Rodrigues N. Ferreira - 201565082AC

Lucas Carvalho Ribeiro - 201565554AC

Professora Vânia de Oliveira Neves

Juiz de Fora - MG
13 de novembro de 2017

Sumário

Lista de Figuras

Lista de Tabelas

1 Introdução

Este relatório será dividido em duas partes principais:

- Parte 1 - Análise de diferentes tipos de Árvores de Busca
- Parte 2 - Implementção de uma estrutura de dados eficiente em armazenamento e busca de um arquivo de tweets

Na primeira parte faremos análises quanto ao desempenho de diferentes estruturas de dados, também chamadas de árvores, que tem como função armazenar de forma organizada um conjunto de dados de modo a facilitar funções como busca, inserção e remoção. Faremos uma comparação entre os custos de cada árvore para a realização de cada função, utilizando de Tweets como elemento chave, ao comparar os seguintes critérios: tempo médio de processamento, número de cópias de chaves e número de comparações.

Na segunda parte será mostrada a estrutura de dados, bem como ideias de sua implementação, que facilite funções de busca e inserção num cenário onde quer se armazenar uma grande massa de dados na forma de Tweets.

1.1 Considerações iniciais

- Ambiente de desenvolvimento do código fonte: IntelliJ IDEA Community Edition.
- Linguagem utilizada: Java 8.
- Ambiente de desenvolvimento da documentação: ShareLaTeX - Editor online de LaTeX.

1.2 Dados utilizados

Todos os testes documentados neste relatório foram realizados com um conjunto real de Tweets públicos. A lista utilizada aqui possui 1.000.000 registros de Tweets diferentes e pode ser obtida no seguinte endereço:
`https://www.dropbox.com/s/07zflza2hj6njzj/tweets.txt?dl=0`

1.3 Testes

Todos os testes documentados neste relatório foram realizados da seguinte forma: para cada valor de N (1.000, 5.000, 10.000, 50.000, 100.000, 500.000 e 1.000.000) foram gerados 5 conjuntos de N elementos aleatórios sorteados com 5 sementes diferentes. Os resultados inclusos nas tabelas são formados pela média dos resultados de cada conjunto gerado para aquele número N de elementos.

Os resultados documentados de cada teste são como descritos na seção 1.

1.4 Realização das Atividades

Todos os membros do grupo contribuíram de alguma forma em todas as partes do trabalho, mas a divisão de tarefas foi, principalmente, a seguinte:

- Felipe: Parte 1
- Lohan: Relatório
- Lucas: Parte 2

2 Análise de Árvores

Nesta seção analisaremos por partes o desempenho de 4 Estruturas de Dados, também chamada de arvores nas diferentes funções comuns a elas que são inserção, remoção e busca. As árvores a serem comparadas serão: Árvore Vermelho e Preto, Árvore B (Ordem 5), Árvore AVL e Árvore Splay.

2.1 Avaliando a inserção

Analisamos aqui o funcionamento da inserção em cada uma das árvores bem como seus custos, lembrando novamente que as chaves inseridas nas árvores serão Tweets que serão identificados organizados por seus TweetID.

A metodologia utilizada para os testes foi como descrita na seção 1.3.

Os resultados podem ser vistos nas tabelas 1, 2, 3:

	Tempo(ms)			
N	AVL	Vermelho e Preto	Splay	Arvore B
1000	1	0	0	0
5000	1	2	2	3
10000	3	4	4	9
50000	29	28	35	65
100000	66	70	102	157
500000	562	604	691	1207
1000000	1312	1376	1633	2536

Tabela 1: Tempo médio gasto nas inserções

Analisando o tempo podemos notar que mesmo para grandes valores, o desempenho das árvores em relação à inserção é aproximadamente o mesmo onde provavelmente a diferença de tempo encontrada entre elas está relacionada ao fato de cada árvore possuir características adicionais e portanto requerer funções adicionais. Por exemplo o fato de um nó da arvore B ocupar mais memória que um nó AVL e Splay pode ocasionar demoras em seu processamento, assim como a utilização da função splay na árvore Splay que pode tornar a inserção demorada visto que ela busca uma organização a cada inserção, o que nem sempre acontece nas outras árvores.

	Comparações			
N	AVL	Vermelho e Preto	Splay	Arvore B
1000	11234	22688	11393	13154
5000	72012	143405	73717	86674
10000	157599	312206	162130	196079
50000	945853	1854268	980025	1122001
100000	2026234	3961569	2105212	2459224
500000	11706607	22703896	12211329	13463492
1000000	24784482	47901072	25873980	28835413

Tabela 2: Número de comparações na inserção

O número de comparações aumenta bastante conforme o número de chaves aumenta, o que era esperado, afinal conforme a árvore cresce, mais comparações precisam ser feitas até chegar numa folha onde o próximo nó será inserido. Analisando estes dados podemos ver a árvore Splay se sobressaindo, isso acontece pelo fato de a cada inserção esta árvore se reorganizar levando um nó inserido ao topo facilitando portanto a entrada de outros dados caso eles sejam bem próximos. A árvore B não fica atrás, como seus nós possuem capacidade de muitas chaves, a caminhada pela árvore não dura muito tempo e portanto poucas comparações são feitas até descobrir o local da inserção, porém um número relevante ainda é gasto para que uma inserção ordenada seja feita no devido nó. Vale lembrar que os dados nas tabelas sao referentes ao somatório após as N inserções em cada um dos 5 conjuntos testados , não o resultado de elemento por elemento, e depois uma media é feita do resultado dos conjuntos.

	Cópias			
N	AVL	Vermelho e Preto	Splay	Arvore B
1000	491	584	8418	730
5000	2794	2926	54698	3683
10000	5523	5823	120480	7424
50000	28207	29162	728887	37034
100000	55992	58237	1566635	74323
500000	282131	291921	9093291	380787
1000000	555569	587837	19269842	823741

Tabela 3: Número de cópias nas inserções

Entedemos por cópias sempre que ocorre troca de valores entre os nós, e portanto sempre que rotações, por exemplo, começam a ocorrer. Olhando por esse ponto de vista é esperado que a árvore Splay seja a com maior número de cópias visto que a cada nova inserção o nó deve ser levado até a raiz através de rotações e portanto um grande número de rotações é feita a cada inserção, acumulando assim um grande número de cópias, o que pode ou não comprometer seu tempo médio de funcionamento. Nas outras árvores os dados são relativamente iguais sendo a árvore B um pouco diferente pois sua contagem de copias é diferente, visto que não possui rotações, e portanto é baseado nas vezes em que ocorre splits e reorganizações.

2.2 Avaliando a busca

Nesta subseção analisamos e comparamos, com testes semelhantes aos do item anterior, o funcionamento de busca de uma chave nas árvores.

Os dados destes testes podem ser vistos nas tabelas 4, 5:

	Tempo(ms)			
N	AVL	Vermelho e Preto	Splay	Arvore B
1000	0	0	0	0
5000	0	1	2	1
10000	2	2	4	4
50000	21	19	41	41
100000	58	49	107	99
500000	475	413	813	605
1000000	1129	941	1906	1226

Tabela 4: Tempo médio gasto nas buscas

	Comparações			
N	AVL	Vermelho e Preto	Splay	Arvore B
1000	19219	14430	21693	24290
5000	119641	90291	133838	146124
10000	259530	194795	289398	312351
50000	1530432	1147679	1693788	1760113
100000	3254657	2432827	3589212	3783182
500000	18323890	13660094	19860143	19474088
1000000	38028891	28321507	40654476	38992424

Tabela 5: Número de comparações nas buscas

Não faz sentido falar em cópias aqui visto que as funções de busca em geral só caminham pela árvore e retornam ou não o nó conforme ele seja encontrado ou não. É importante lembrar que existem sim cópias na busca da árvore Splay visto que ela é a unica que realiza rotações mesmo na busca e portanto cópias são feitas no processo e podemos notar a diferença que isso causa na busca ao ver o tempo gasto na tabela 4 onde o tempo da árvore Splay chega a ser quase o dobro do tempo gasto nas outras árvores.

Fora isso, como a estrutura das outras árvores realizam buscas de forma semelhante, não existe grande diferenças em seu tempo de funcionamento e a tabela 4 mostra que os valores não se distanciam muito.

O número de comparações para as buscas dos N elementos são relativamente grandes, mas vale lembrar que para cada busca realizada espera-se que uma média de *log(n)* comparações no pior caso seja feita. Podemos notar novamente na tabela 5 que a árvore Splay lidera no consumo pelo modo como funciona, fazendo

comparações tanto na busca propriamente dita quanto na reorganização da árvore, lembrando que isso acontece porque os dados buscados são aleatórios e possuem pouca relação entre si, caso fossem mais relacionados a estrutura da árvore splay se tornaria extremamente eficiente na busca levando de uma vez para perto da raiz todos os nós semelhantes. A árvore B possui também um grande consumo de comparações, o que se dá provavelmente pelo fato de muitas comparações serem feitas dentro de cada nó até chegar na chave desejada.

2.3 Avaliando a remoção

Nesta subseção analisamos e comparamos, com testes semelhantes aos do item anterior, o funcionamento de remoção de chaves nas árvores.

	Tempo(ms)			
N	AVL	Vermelho e Preto	Splay	Arvore B
1000	0	0	0	0
5000	1	1	1	1
10000	3	2	3	4
50000	24	16	29	37
100000	59	41	72	89
500000	555	375	627	658
1000000	1386	923	1544	1662

Tabela 6: Tempo médio gasto nas remoções

	Comparações			
N	AVL	Vermelho e Preto	Splay	Arvore B
1000	20260	16043	20151	18102
5000	126168	98217	125484	111432
10000	272249	210623	272746	243302
50000	1600409	1230033	1616708	1453016
100000	3401158	2605399	3450209	3180727
500000	19346491	14796988	19743753	18761235
1000000	40679344	31152736	41535454	41924557

Tabela 7: Número de comparações na remoção

	Cópias			
N	AVL	Vermelho e Preto	Splay	Arvore B
1000	3411	3182	9031	1054
5000	17204	15953	57619	5418
10000	34435	31926	126114	10831
50000	172601	160105	756456	54519
100000	345845	321008	1620086	87355
500000	1760628	1645216	9317565	350048
1000000	3596920	3387035	19644674	776309

Tabela 8: Número de cópias nas remoções

Podemos notar através da tabela 8 que apesar de a árvore B ter sido a mais lenta em seu processamento foi a que menos teve número de cópias, o que se deve ao fato de suas remoções na maior parte dos casos ser simples e não exigir operações mais complexas, diferente das outras árvores que precisam se reorganizar na maior parte do tempo para garantir que regras como fator de balanceamento ou cores sejam satisfeitas a todo momento. Ainda assim teve um grande número de comparações e ainda é a arvore com maior gasto de memórias o que justifica seu lento processamento se comparado às outras arvores.

3 Estruturas de Dados utilizadas

Nesta seção, mostraremos as estruturas de dados utilizadas para armazenamento dos Tweets tanto para busca em memória interna quanto memória externa.

3.1 Nós das árvores

Para todas as árvores binárias (AVL, Vermelho e Preto e Splay) foram utilizados nós com as seguintes propriedades:

- *key*, de um tipo T que deve ser comparável, representando a chave de comparação entre os nós da árvore
- *value*, de um tipo E qualquer, representando um valor a ser guardado naquele nó
- *parent*, um outro nó, representando o pai do nó em questão
- *leftChild*, um outro nó, representando o filho à esquerda, com chave menor que o do nó em questão
- *rightChild*, um outro nó, representando o filho à direita, com chave maior que o do nó em questão

Para as árvores binárias, a fim de fazer o armazenamento dos Tweets em memória interna, foram utilizados como tipos T e E, *Long* e *Tweet*, respectivamente, pois cada Tweet possui um *TWEETID* comparável que serve como chave.

3.1.1 AVL e Splay

Para as árvores AVL e Splay foram utilizados nós semelhantes, exatamente como descrito acima, sem nenhuma propriedade adicional.

3.1.2 Vermelho e Preto

Para a árvore Vermelho e Preto, uma única mudança foi feita na estrutura: a adição de uma propriedade do tipo *boolean* que representa a cor do nó: vermelho ou preto.

3.1.3 Árvore B

Para a árvore B, a fim de guardar tanto a chave quanto o valor e manter a consistência entre eles na lista de chaves que cada nó possui, foi criada uma estrutura auxiliar, chamada de *Entry*, que possui a seguintes propriedades:

- *key*, de um tipo T que deve ser comparável, representando a chave de comparação entre os nós da árvore
- *value*, de um tipo E qualquer, representando um valor a ser guardado naquele nó

Portanto, a estrutura do nó ficou da seguinte forma:

- *entries*, uma lista de objetos do tipo *Entry*, representando as entradas contidas naquele nó
- *children*, uma lista de outros nós, representando os filhos do nó em questão
- *parent*, um outro nó, representando o pai do nó em questão

Esta árvore foi usada para os dois propósitos: busca de Tweets em memória interna e memória externa, portanto foi usada de duas formas diferentes.

Para o caso de utilização de memória interna, foram utilizados como tipos T e E, *Long* e *Tweet* respectivamente, assim como nas outras árvores.

Para utilização de memória externa, foram utilizados como tipos T e E, *Long* e *Long*, sendo que o primeiro Long representa o *TWEETID*, e o segundo guarda a linha do arquivo em que o Tweet se encontra. Dessa forma, não se mantém o Tweet todo em memória, apenas a linha em que o Tweet que possui aquele *TWEETID* se encontra.

4 Conclusão

Analisando parte por parte das operações básicas realizadas pelas diferentes árvores podemos chegar a algumas conclusões sobre suas aplicações para nosso problema em questão que é o armazenamento de um arquivo de Tweets.

Primeiro, como os arquivos são relativamente esparsos a árvore Splay não foi capaz de realizar sua maior função, que é facilitar operações entre dados correlacionados levando-os à raiz, e podemos notar nas tabelas que seu desempenho foi de longe o pior em todos os testes.

A árvore B , apesar de sua premissa que nos fazia esperar que fosse essa a de melhor desempenho para os testes acabou por não possuir também bons resultados, o que provavelmente ocorreu por conta da origem dos dados que deve ter feito a arvore se degenerar em alguns momentos e portanto não realizando bem sua função.

As árvores AVL e Vermelho e Preto foram as que apresentaram os melhores resultados na maioria das tabelas, sendo estes resultados bem semelhantes.O que nos leva a concluir que para essa primeira parte as árvores que melhor resolvem nosso problema de armazenamento de tweets sorteados são estas árvores, o que deve acontecer por conta de suas organizações serem mais independentes da origem dos dados para garantir suas propriedades.

Ainda dentro da comparação entre a AVL e a Vermelho e Preto, é possível perceber que a Vermelho e Preto sacrifica um pouco de tempo a mais na inserção, mas no lugar disso garante tempos muito melhores que a AVL na busca e na remoção, o que pode ser melhor no contexto de busca de Tweets, pois uma vez que são inseridos, apesar de demorar mais, poderão ser buscados de forma muito mais rápida que na AVL.