UNIVERSIDADE DO ESTADO DE SANTA CATARINA - UDESC
CENTRO DE CIÊNCIAS TECNOLÓGICAS

Estudo de Caso

Árvores AVL, Rubro-Negra e B

DEBORA LAWALL LANGNER
LAVÍNIA RAFAELA DE MARCO
LUIS EDUARDO BET
MARIA EDUARDA CAMPOS
MIRUNA MARIA ONOFREI

Universidade do Estado de Santa Catarina - Campus CCT
Rua Paulo Malschitzki, 200, Zona Industrial Norte, Joinville/SC, CEP: 89.219-710

1. INTRODUÇÃO:

No âmbito da ciência da computação, a análise de algoritmos desempenha um papel fundamental na compreensão do desempenho e eficiência de estruturas de dados. Este trabalho propõe uma investigação sobre a complexidade algorítmica associada às operações de adição e remoção de nós em três tipos distintos de árvores: AVL, rubro-negra e B. O elemento diferenciador para as árvores B é a consideração do parâmetro de ordem, sendo exploradas ordens iguais a 1, 5 e 10.

A complexidade algorítmica destas operações será avaliada em função do tamanho do conjunto de dados, representado pelo número de chaves, variando de 1 a 10000. O caso médio será reproduzido através da geração de chaves aleatórias. A robustez da análise será assegurada pela repetição deste processo em pelo menos 10 conjuntos de amostras, permitindo a obtenção de resultados médios.

Por fim, o estudo das operações em árvores AVL, rubro-negra e B, considerando variações significativas nos parâmetros, contribuirá para uma visão mais ampla do comportamento dessas estruturas de dados em diferentes quadros.

2. METODOLOGIA:

De modo geral, para o desenvolvimento do trabalho, foi utilizada a linguagem de programação C para implementar os três tipos de árvores propostas, bem como suas lógicas de funcionamento disponíveis nos códigos deste trabalho. Além disso, a fim de gerar os gráficos de linha de adição e remoção das estruturas implementadas, foi empregada a linguagem *Python*, em que se obteve as imagens onde o eixo X representa o tamanho dos conjuntos de dados (1 a 10000) e o eixo Y representa o esforço computacional das operações considerando os eventuais balanceamentos.

Com isso em mente, abaixo serão descritas algumas considerações específicas para o desenvolvimento do programa de cada código das respectivas árvores AVL, Rubro-Negra e B.

2.1 Árvore AVL:

Para realizar a inserção na árvore AVL a função ‘inserir’ realiza-se uma inserção igualmente encontrada em uma árvore de busca binária, algo já visto durante as aulas, e insere-se o nó na posição adequada. Após a inserção, as alturas dos nós são atualizadas e consequentemente os fatores de balanceamento são calculados para verificar as possíveis rotações que podem ser feitas caso a árvore esteja desbalanceada.

A árvore AVL possui um esforço maior comparado às outras árvores analisadas neste estudo de caso devido ao grande número de rotações que ela realiza, pois ela sempre busca uma árvore balanceada e com altura mínima.

Para a remoção de um nó são analisados três casos dentro da função ‘deletar’:

- Caso 1: O nó a ser removido é um nó folha
- Caso 2: O nó a ser removido possui 1 filho
- Caso 3: O nó a ser removido possui 2 filhos

A lógica da remoção gira em torno desses três casos. Há também a possibilidade do nó que foi solicitada a remoção não ser encontrado na estrutura da árvore, assim não acontece nada. Vale lembrar que após a remoção de um nó são atualizadas as alturas, calculados os fatores de balanceamento e, se a árvore estiver desbalanceada, as rotações.

2.2 Árvore Rubro-Negra:

Na inserção em uma árvore rubro-negra, a função ‘adicionar’ adiciona um novo nó, inicialmente colorido de vermelho na árvore, mantendo as propriedades da busca binária. Caso a árvore já tenha o valor a ser inserido, a quantidade (qtd) do nó existente é incrementada. Após a inserção, a função ‘balancear’ é chamada para ajustar as cores e realizar rotações conforme necessário, assegurando que a árvore permaneça balanceada e atenda às propriedades rubro-negras.

Na remoção, a função ‘deleteNo’ procura e remove um nó específico na árvore, substituindo-o por seu sucessor, se necessário. A função ‘transplant’ facilita essa substituição. Em seguida, a função ‘deleteFixup’ é chamada para corrigir eventuais violações das propriedades rubro-negras, realizando rotações e ajustes nas cores para manter o equilíbrio da árvore após a remoção. Essas operações são cruciais para garantir que a árvore rubro-negra preserve seu balanceamento durante as inserções e remoções, mantendo a eficiência nas operações de busca.

2.3 Árvore B:

Para a inserção de dados com chaves iguais na árvore B, foi criado um vetor de inteiros que mantém o controle da quantidade de vezes que cada chave foi adicionada. Com essa abordagem, ao inserir um valor já existente, evita-se criar um novo nó na árvore. Em vez disso, é incrementado o campo "qdd" no nó correspondente à chave em uma unidade. Isso faz com que rotações desnecessárias sejam evitadas, bem como haja otimização de espaço.

Ademais, para a remoção de chaves nessa estrutura, foi levado em consideração os três casos típicos: Remoção de chave em um nó folha, sem causar *underflow*, que elimina a chave da árvore e rearranja as chaves remanescentes para suprir o espaço liberado; remoção de uma chave em um nó, causando *underflow*, em que procura um nó irmão adjacente que possua mais chaves que o mínimo, e se encontrar, será redistribuído as chaves do nó, modificando o conteúdo do nó pai; Remoção de uma chave em um nó não folha, que remove chaves somente nas folhas, ou seja, a chave a ser removida é trocada pela sua chave sucessora imediata, e então, é removido a chave diretamente do nó folha.

3. RESULTADOS E DISCUSSÕES:

Posterior à implementação dos códigos, levando em consideração as pontuações descritas anteriormente, foram incorporados contadores às estruturas para estimar o número de iterações durante a inserção dos dados. Nesse processo, se levou em conta a geração de um conjunto de dados (chaves) com o tamanho variando entre 1 e 1000. Os resultados obtidos estão dispostos nas Tabelas AVL, Rubro Negra, Árvore B - Ordem 1, Árvore B - Ordem 5 e Árvore B - Ordem 10, respectivamente anexadas no tópico 5.

A partir dos resultados apresentados nas tabelas (Seção 5), foi possível plotar as curvas dos dados, tanto de inserção (Figura 1), como de remoção de nós (Figura 2). Desse modo, observa-se que o eixo X do gráfico corresponde ao tamanho do conjunto de dados, e o eixo Y diz respeito ao esforço computacional das operações realizadas.

Figura 1 - Gráfico para a Inserção de nós.
Comparativo Esforço x Tamanho de Dados

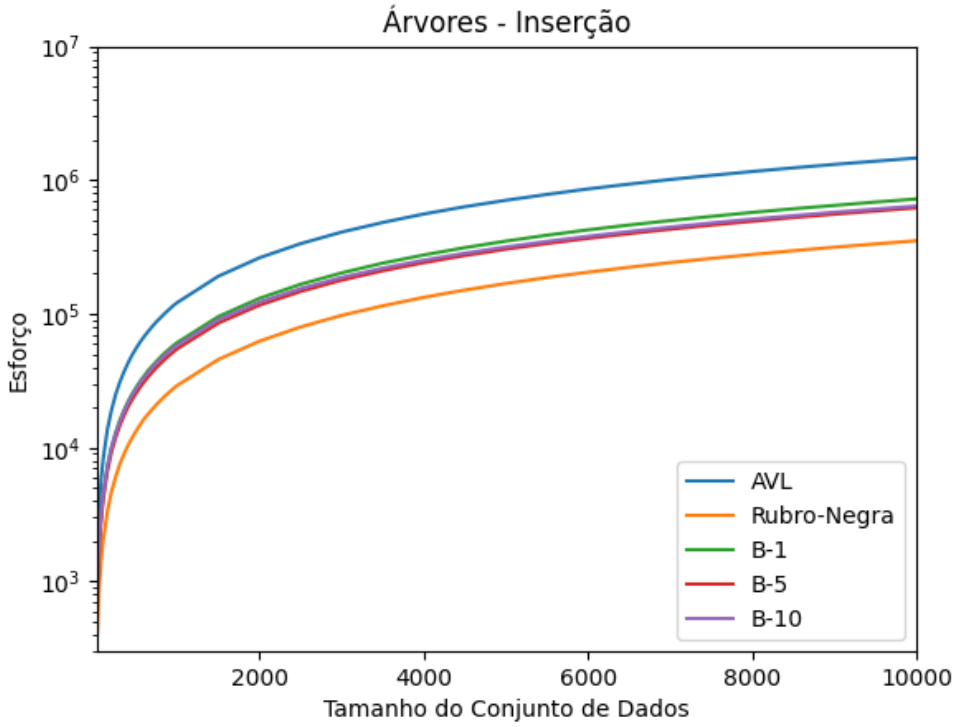
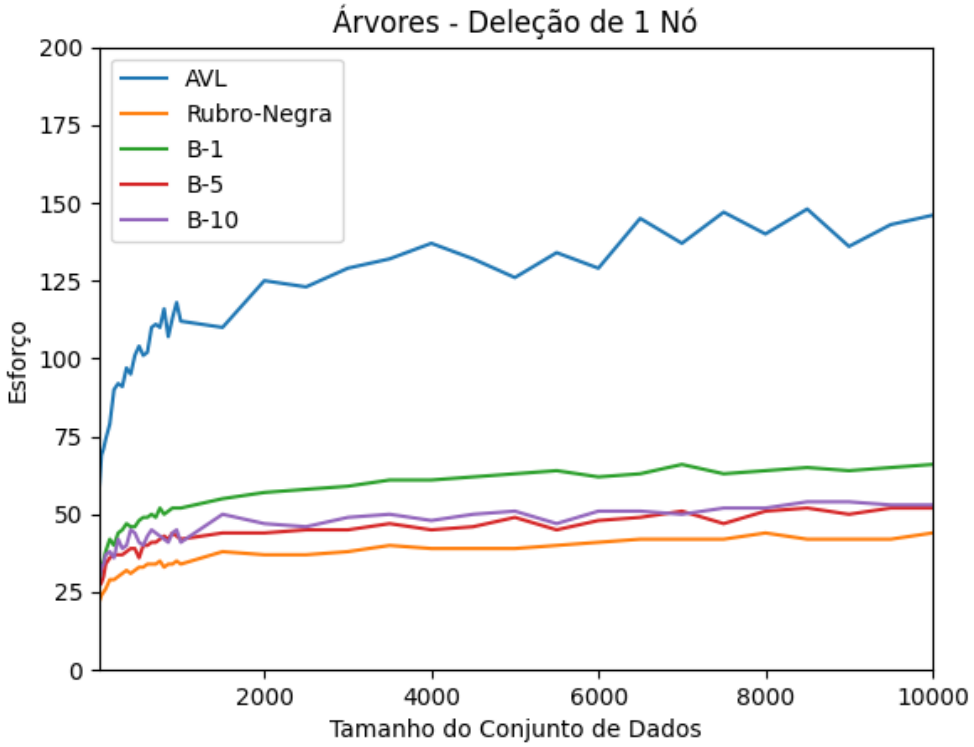


Figura 2 - Gráfico para a Remoção de nós.
Comparativo Esforço x Tamanho de Dados



Embora na teoria a complexidade algorítmica de inserção e remoção seja supostamente igual para todas as estruturas, no caso, log(n), os resultados reais das imagens destacam as disparidades entre as árvores. Essas discrepâncias ressaltam o custo computacional mais elevado associado à árvore AVL. A explicação para tal diferença pode residir no balanceamento minucioso desta árvore, o que aumenta consideravelmente a complexidade das inserções. Isso ocorre porque, a cada novo nó inserido, é necessário reequilibrar toda a árvore.

Por outro lado, no contexto da árvore Rubro-Negra, observa-se que em cada inserção ocorrem, no máximo, duas rotações. Esse comportamento mais eficiente contrasta com a árvore AVL, no qual o processo de balanceamento é mais intensivo. Assim, esse desempenho demonstra a influência da estratégia de balanceamento na eficiência operacional das árvores.

Além disso, ao analisar as curvas, a árvore B tanto na ordem 1, 5 ou 10 se mostram muito próximas uma das outras, ainda que seja visualmente perceptível que a ordem 5 foi a de melhor desempenho. Isso pode representar um equilíbrio entre a altura da árvore e o número de chaves por nó, levando a uma combinação eficiente nas operações realizadas e minimização do custo adicional de memória.

Nos gráficos, também se observa que, quando comparado a árvore AVL, a árvore B demonstra maior proximidade com a árvore Rubro-Negra no número de interações durante a inserção de dados. No entanto, é evidente que, para este conjunto específico de dados, a árvore Rubro-Negra possui um menor número de iterações, resultando em um custo computacional inferior, se mostrando mais vantajosa em relação à árvore B.

4. CONCLUSÃO:

O estudo comparativo das árvores AVL, rubro-negra e B revelou disparidades significativas em suas complexidades algorítmicas durante operações de adição e remoção. A árvore AVL, devido ao seu rigoroso balanceamento, mostrou-se computacionalmente mais custosa, enquanto a árvore rubro-negra destacou-se pela eficiência ao limitar rotações. A árvore B, com variações nas ordens 1, 5 e 10, apresentou desempenho consistente, com a ordem 5 sendo a mais eficiente. Os resultados consolidaram a superioridade da árvore rubro-negra em termos de eficiência operacional para o conjunto de dados analisados, fornecendo importantes *insights* para aplicações práticas em ciência da computação.

5. ANEXOS:

AVL		
Nós	**Inserção**	**Remoção**
25	1550	59
50	3665	69
75	5993	71
100	8342	74
150	13694	79
200	19067	90
250	24957	92
300	30685	91
350	36666	97
400	42787	95
450	49156	101
500	55366	104
550	61715	101
600	68138	102
650	74383	110
700	81045	111
750	87787	110
800	94092	116
850	101064	107
900	107769	113
950	115056	118
1000	121552	112
1500	191154	110
2000	262547	125
2500	334727	123
3000	408573	129
3500	482018	132
4000	557903	137
4500	632586	132
5000	706541	126
5500	781517	134
6000	858926	129
6500	933245	145
7000	1010079	137

7500	1084023	147
8000	1159244	140
8500	1234844	148
9000	1310671	136
9500	1384714	143
10000	1462684	146

Rubro Negra		
Nós	**Inserção**	**Remoção**
25	382	22
50	898	24
75	1463	25
100	2065	26
150	3336	29
200	4693	29
250	6040	30
300	7510	31
350	8906	32
400	10384	31
450	11817	32
500	13362	33
550	14884	33
600	16541	34
650	17965	34
700	19488	34
750	21103	35
800	22650	33
850	24312	34
900	25886	34
950	27589	35
1000	29183	34
1500	45654	38
2000	62514	37
2500	79801	37
3000	97493	38

3500	114858	40
4000	132922	39
4500	150797	39
5000	168786	39
5500	187089	40
6000	205037	41
6500	223190	42
7000	241922	42
7500	259722	42
8000	278107	44
8500	296654	42
9000	315286	42
9500	333455	42
10000	352324	44

Árvore B - Ordem 1		
Nós	**Inserção**	**Remoção**
25	852	28
50	1986	33
75	3229	35
100	4499	38
150	7256	42
200	10090	40
250	12914	44
300	15928	45
350	18948	47
400	22012	46
450	25157	46
500	28288	48
550	31449	49
600	34612	49
650	38013	50
700	41059	49
750	44443	52
800	47814	50
850	51250	51
900	54399	52

950	57899	52
1000	61086	52
1500	95450	55
2000	130692	57
2500	166457	58
3000	202545	59
3500	239547	61
4000	275701	61
4500	312736	62
5000	349981	63
5500	387427	64
6000	424886	62
6500	461739	63
7000	498193	66
7500	535434	63
8000	573615	64
8500	610800	65
9000	646894	64
9500	685582	65
10000	723055	66

Árvore B - Ordem 5		
Nós	**Inserção**	**Remoção**
25	813	27
50	1864	28
75	2966	30
100	4172	34
150	6659	36
200	9291	37
250	11900	37
300	14557	37
350	17259	38
400	20052	39
450	22802	39
500	25594	36
550	28370	40
600	31188	40

650	34054	41
700	36917	41
750	39875	42
800	42768	43
850	45798	42
900	48801	44
950	51746	43
1000	54847	42
1500	85342	44
2000	116205	44
2500	147127	45
3000	178711	45
3500	210075	47
4000	241537	45
4500	273016	46
5000	304445	49
5500	335775	45
6000	366777	48
6500	398158	49
7000	429241	51
7500	460888	47
8000	492303	51
8500	525024	52
9000	556406	50
9500	587768	52
10000	620056	52

Árvore B - Ordem 10		
Nós	**Inserção**	**Remoção**
25	854	29
50	2037	33
75	3271	34
100	4556	37
150	7187	38
200	9954	36
250	12682	42
300	15535	39

350	18476	40
400	21439	45
450	24434	44
500	27479	41
550	30575	40
600	33615	43
650	36750	45
700	39853	44
750	43011	43
800	46038	42
850	49179	41
900	52377	44
950	55462	45
1000	58498	41
1500	90465	50
2000	122512	47
2500	154532	46
3000	187181	49
3500	219598	50
4000	251756	48
4500	284052	50
5000	316219	51
5500	347955	47
6000	380150	51
6500	412420	51
7000	445220	50
7500	478155	52
8000	511009	52
8500	542557	54
9000	575098	54
9500	607268	53
10000	639780	53