

Resumo do projeto:

A execução do projeto utiliza todos os conceitos de *Árvore Binária* e *AVL*, tais como inserção de elementos, remoção, busca e também a contabilização do tempo de inserção e busca em ambas a fim de avaliar a diferença de tempo, além das vantagens e desvantagens de ambas (*Árvore Binária* e *AVL*). Espera-se que a inserção na *Árvore Binária* seja mais rápida que na *Árvore AVL*, ao contrário das buscas serem demoradas devido ao desbalanceamento que a Binária proporciona. Já na *Árvore AVL*, o tempo de inserção é mais lento que a Binária devido sempre ocorrer o balanceamento ao inserir e remover elementos; mais com vantagem das buscas serem muito mais rápidas, resultado de uma profundidade menor de nós.

Introdução

- **Árvores Binárias;**
- **Árvores AVL;**
- **Medições do tempo de execução;**
- **Manipulação de arquivos;**
- **TADs;**
- **Ponteiros;**

O projeto está organizado em arquivos separados. O mesmo contém arquivos de cabeçalho e de implementação para uma melhor organização de responsabilidades dos algoritmos aplicados: Todo código relacionado a Árvores Binárias e AVL estão em uma pasta com seus arquivos *header* e de execução, bastando o *import* indicar o caminho de onde se encontram, como por exemplo, a partir do arquivo *main*, a directiva de *include* é “./BST/bst.h” ou “./AVL/avl.h”. A partir dos *imports*, todas as estruturas e funções relacionadas podem ser acessadas.

Seções específicas

Todas as funções estão relacionadas a um tipo específico para manipulação das estruturas de dados (descrição apenas das relacionadas ao conteúdo programático da disciplina).

1 - A “biblioteca” da *Árvore Binária* contém as seguintes funções:

- *BST bstStart();*
 - *BSTNode bstInsert(BSTNode root_node, int key);*
 - *BSTNode bstSeach(BSTNode root_node, int key);*
 - *BSTNode bstSeach(BSTNode root_node, int key);*
 - *int bstCountNodes(BSTNode root_node);*
 - *void bstPrint(BSTNode root_node);*
 - *BSTNode bstSeachDad(BSTNode node, int key, BSTNode * dad);*
 - *BSTNode bstDelete(BSTNode root_node, int key);*
 - *int bstCountSheets(BSTNode node);*
 - *int bstHeight(BSTNode root_node);*
 - *int bstDephNode(BSTNode root_node, int key);*
 - *BSTNode bstDestroy(BSTNode root_node);*
 - *void bstMinDepth(BSTNode root_node, int count, int *less_depth_result).*
-
- *BSTNode bstNodeCreate(int key) ;*
 - *BSTNode bstNodeDestroy(BSTNode node) ;*
 - *void bstNodePrint(BSTNode node) ;*
 - *bool bstNodeIsLeaf(BSTNode node);*

2 - A “biblioteca” da *Árvore AVL* contém as seguintes funções:

- *AVL avlStart();*
- *int avlHeight(AVLNode node);*
- *int max(int x, int y);*
- *AVLNode avlRotateToRight(AVLNode node);*
- *AVLNode avlRotateToLeft(AVLNode node);*
- *AVLNode avlRotateLeftToRight(AVLNode node);*
- *AVLNode avlRotateRightToLeft(AVLNode node);*
- *AVLNode avlInsert(AVLNode root_node, int key);*
- *AVLNode avlSeach(AVLNode root_node, int key);*
- *int avlCountNodes(AVLNode root_node);*
- *void avlPrint(AVLNode root_node);*
- *AVLNode avlDelete(AVLNode root_node, int key);*
- *int avlCountSheets(AVLNode node);*
- *AVLNode avlDestroy(AVLNode root_node);*
- *void bstMinDepth(AVLNode root_node, int count, int *less_depth_result).*

Resultados da execução

- **exercício 1:**
 - *Árvore Binária*

- Teste de execução de desempenho (médio) entre o tempo de inserção e tempo busca por em 30 árvores diferentes, suas diferenças e quantidade de repetições entre a maior altura e menor altura.

Rodada	Tempo de inserção	Tempo de busca	Maior profundidade	Menor profundidade
1 ^a	1320 ns	1270 ns	21	7
2 ^a	651 ns	718 ns	21	4
3 ^a	1185 ns	1242 ns	20	3
4 ^a	578 ns	511 ns	20	4
5 ^a	715 ns	765 ns	20	4
6 ^a	1356 ns	1290 ns	20	6
7 ^a	1345 ns	1308 ns	19	4
8 ^a	1316 ns	1318 ns	23	5
9 ^a	493 ns	546 ns	20	4
10 ^a	422 ns	484 ns	21	4

```

Windows PowerShell

Tempo medio de procura: 690 ns.

Total de elementos inseridos com sucesso e sem repeticoes: 1007

Tempo medio das insercoes: 627 ns

diferenca de profundidade (15) que ocorreram repetiu 5 vezes
diferenca de profundidade (18) que ocorreram repetiu 7 vezes
diferenca de profundidade (14) que ocorreram repetiu 1 vezes
diferenca de profundidade (16) que ocorreram repetiu 7 vezes
diferenca de profundidade (19) que ocorreram repetiu 3 vezes
diferenca de profundidade (17) que ocorreram repetiu 3 vezes
diferenca de profundidade (13) que ocorreram repetiu 1 vezes
diferenca de profundidade (20) que ocorreram repetiu 1 vezes
diferenca de profundidade (21) que ocorreram repetiu 2 vezes

Media das MAIORES profundidades: 21
Media das MENORES profundidades: 4

total de arvores [30]. Numero de elementos [1060]. Range [10600]

0 - sair
1 - Alterar parametros pre definidos
2 - rodar simulacao
>> |

```

exercício 2:

- **Árvore AVL**
- Teste de execução de desempenho entre o tempo de inserção e tempo busca por em 30 árvores diferentes, suas diferenças e quantidade de

repetições entre a maior altura e menor altura.

Rodada	Tempo de inserção	Tempo de busca	Maior profundidade	Menor profundidade
1 ^a	1078 ns	1020 ns	11	7
2 ^a	390 ns	390 ns	11	8
3 ^a	483 ns	386 ns	11	7
4 ^a	365 ns	421 ns	11	7
5 ^a	328 ns	328 ns	11	8
6 ^a	723 ns	781 ns	10	7
7 ^a	1615 ns	1550 ns	11	7
8 ^a	296 ns	239 ns	11	8
9 ^a	234 ns	231 ns	11	8
10 ^a	218 ns	218 ns	11	7

```
Windows PowerShell
[ 0] - tempo de procura: 950 ns. Nao encontrado
[ 9] - tempo de procura: 950 ns. Nao encontrado
[ 99] - tempo de procura: 950 ns. Nao encontrado
[ 999] - tempo de procura: 951 ns. Nao encontrado
[ 1999] - tempo de procura: 951 ns. Nao encontrado
[ 29000] - tempo de procura: 952 ns. Nao encontrado

Tempo medio de procura: 951 ns.

Total de elementos inseridos com sucesso e sem repeticoes: 1007

Tempo medio das insercoes: 884 ns

[4] repetiu 28 vezes
[5] repetiu 1 vezes
[3] repetiu 1 vezes

Media das MAIORES profundidades: 11
Media das MENORES profundidades: 7

total de arvores [30]. Numero de elementos [1060]. Range [10600]

0 - sair
1 - Alterar parametros pre definidos
2 - rodar simulacao
>>
```

● Exercício 3

Foi utilizado uma *Árvore binária* para armazenamento de palavras em inglês contendo cada nó, uma *lista encadeada* de cada palavra em inglês. Primeiramente foi utilizado uma função que processa um arquivo de texto. Ao ler cada linha, o algoritmo trata a frase obtida e em seguida é distribuída: se é um capítulo, uma nova árvore é criada; se é uma palavra em inglês, um novo nó é adicionado à árvore criada anteriormente; se é uma

palavra em português, é adicionada a lista do nó atual da árvore.

- **Exercício 4:**

Foi utilizado uma *Árvore AVL* para armazenamento de palavras em inglês contendo cada nó, uma *lista encadeada* de cada palavra em inglês. Primeiramente foi utilizado uma função que processa um arquivo de texto. Ao ler cada linha, o algoritmo trata a frase obtida e em seguida é distribuída: se é um capítulo, uma nova árvore é criada; se é uma palavra em inglês, um novo nó é adicionado à árvore criada anteriormente; se é uma palavra em português, é adicionada a lista do nó atual da árvore.

Conclusão

A execução de todos os exercícios levantou as vantagens e desvantagens entre uma *Árvore binária* e *Árvore AVL* para avaliação de desempenho entre ambas. Percebeu-se logo de cara uma diferença grande entre as profundidades (maior e menor) entre cada uma, sendo a binária a que possui uma variação grande de menor profundidade, onde variou de 7 a 4. Já na AVL, tendeu-se a ter uma maior uniformidade entre a menor profundidade, basicamente não variando e mantendo-se num valor constante de 7 a 8. Já a maior profundidade na AVL, observou-se que é cerca de 50% menor em basicamente todos os casos em relação à binária, mantendo-se entre 11 a 10 (maior profundidade). Já a binária manteve-se entre 19 a 21 (maior profundidade).

Os testes em relação a tempo de execução, não são tão conclusivos devido a cada execução ter um valor quase aleatório, mais foi notado que a inserção na AVL é mais demorado devido a ser necessários o balanceamento, onde se tem a vantagem caso ocorra inserção ordenada de elementos, sempre se manter balanceada, melhorando assim, a busca por elementos nesse caso específico. Já na binária, caso ocorra inserção ordenada, a cada elemento vai ficando mais lento devido sempre percorrer até o final para a inclusão. A busca também é afetada sendo necessário percorrer basicamente a profundidade da árvore (checagem de elemento a elemento) que no pior caso, deve ser toda a árvore.