

Resumo do projeto:

A execução do projeto utiliza todos os conceitos de *Árvore Rubro Negra* e *Árvores Dois Três*, 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 vantagem e desvantagem da entre ambas (*Árvore Binária* e *AVL*)

Introdução

- **Árvore Rubro-Negra**
- **Árvore Dois Três**
- **Manipulação de arquivos**
- **Ponteiros**
- **Estruturas**
- **Uso de Listas Encadeadas**
- **Medição de tempo de inserção**

Árvores 2-3 tendem a manter a árvore totalmente balanceada após a inserção de elementos, com a profundidade menor que a Rubro-Negra. O custo de inserção é mais lento que a árvore rubro negra. Já a rubro negra (variação da AVL) tende a se preocupar menos com o balanceamento, fazendo assim, a mesma ser mais rápida na inserção dos dados. A profundidade dos elementos na rubro negra também tendem a ser muito mais profundos e desbalanceados.

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 Rubro-Negra* contém as seguintes funções:

- ***RBTNode* rbtnodeCreate(char *palavra);**
- ***bool* rbtnodeIsLeaf(RBTNode node);**
- ***void* rbtnodePrint(RBTNode node);**
- ***RBT* rbtnodeStart();**
- ***RBTNode* rbtnodeRotateLeft(RBTNode node);**

- ***RBTNode rotateToRight(RBTNode node);***
- ***void changeColor(RBTNode node);***
- ***int getColor(RBTNode node);***
- ***RBTNode move2LeftRED(RBTNode node);***
- ***RBTNode move2RightRED(RBTNode node);***
- ***RBTNode searchMin(RBTNode node);***
- ***RBTNode balanceNode(RBTNode node);***
- ***RBTNode removeMin(RBTNode node);***
- ***RBTNode fixInsertion(RBTNode node);***
- ***RBTNode rbttInsert(RBTNode node, char *palavra);***
- ***RBTNode rbttNodeSearch(RBTNode root, char *palavra);***
- ***RBTNode rbttNodeRemove(RBTNode node, char *palavra);***
- ***bool rbttRemove(RBTNode *node, char *palavra);***
- ***void rbttPrint(RBTNode root);***

2 - A biblioteca da Árvore 2_3 contém as seguintes funções:

- ***DTNode dttCreateNode(char *palavra, DTNode left, DTNode cen);***
- ***bool isLeaf(DTNode node);***
- ***void dtNodePrint(DTNode node);***
- ***DTT dttStart();***
- ***void dttPrint(DTNode root);***
- ***void getTotalNodes(DTNode root, int *count);***
- ***DTNode dttSeach(DTNode node, char *palavra);***
- ***DTNode insertNode(DTNode root, char *palavra, DTNode new);***
- ***DTNode explodeNode(DTNode *root, DTNode new_node, char *palavra, char *palavra_cen);***
- ***DTNode ddtlInsert(DTNode parent, DTNode *root, char *palavra, char *palavra_cen);***
- ***int contains(DTNode root, char *palavra);***
- ***int dttDelete(DTNode *parent, DTNode *root, char *palavra);***

Resultados da execução

- exercício 1 relacionado ao conteúdo de *Árvore Rubro Negra*:

```
Windows PowerShell
PS C:\Users\joaoc\Documents\UFPI\Estruturas de dados

tempo de inserção: 15 ns.

1 - imprimir todas as palavras do texto
2 - buscar uma palavra
3 - acrescentar uma palavra
4 - excluir uma palavra
>> 1
litro: 1 8
di: 4
abertis: 1 8
Mussum: 1 8
Forevis: 9
Casamentiss: 2 13
Aenean: 6
Copo: 10
Ipsum: 1 8
In: 5
Mauris: 3 5
Mais: 2
Sapien: 11
Praesent: 3 12
Não: 8
Nam: 13
Paisis: 1 9
Quem: 4
Todo: 6
Suco: 11
```

tempo de inserção médio foi de 15 ns em 10 rodadas de execução

Foi utilizado uma função (função **processFile()**) para pegar cada linha do arquivo. Em seguida, com a linha do arquivo obtida (frase) é necessário separar em palavras (função **toWord()**). Ao obter a palavra, a mesma é inserida (função **rbtInsert()**) na árvore Rubro-Negra. É realizado a busca (função **rbtNodeSeach()**) pela nó da última palavra inserida após a inserção e em seguida (função **addLines()**), é realizada a busca pelas linhas do texto (função **contains()**) em que a palavra existe. Ao encontrar, é adicionado à

lista encadeada daquele nó a linha encontrada, verificando se já não existe (função **ListSearch()**) a linha para evitar duplicação de linhas.

- **exercício 2 relacionado ao conteúdo de *Árvore 2_3*:**

```
Windows PowerShell
PS C:\Users\joaoc\Documents\UFPI\Estruturas de dados

tempo de inserção: 15 ns.

1 - imprimir todas as palavras do texto
2 - buscar uma palavra
3 - acrescentar uma palavra
4 - excluir uma palavra
>> 1
cacilds [ 1 8 ] cacilds [ 1 8 ]
Mussum [ 1 8 ] cacilds [ 1 8 ]
Mussum [ 1 8 ]
In [ 5 ] Mussum [ 1 8 ]
In [ 5 ]
Casamentiss [ 2 13 ] Forevis [ 9 ]
Copo [ 10 ]
Mais [ 2 ]
Ipsum [ 1 8 ] Casamentiss [ 2 13 ]
Mauris [ 3 5 ]
cacilds [ 1 8 ]
abertis [ 1 8 ]
Paisis [ 1 9 ]
Quem [ 4 ]
Não [ 8 ] Mussum [ 1 8 ]
alcoholatra [ 3 ]
santis [ 1 10 ]
Mussum [ 1 8 ]
bebadis [ 2 10 ]
aliquam [ 6 ]
```

O tempo de inserção médio foi de 15 nós em 10 rodadas de execução.

Foi utilizado uma função (função **processFile()**) para pegar cada linha do arquivo. Em seguida, com a linha do arquivo obtida (frase) é necessário separar em palavras (função **toWord()**). Ao obter a palavra, a mesma é inserida (função **rbtInsert()**) na árvore Rubro-Negra. Após a inserção de todas as palavras do texto, a função **selectNode()** é chamada. Ela pega cada nó da árvore individualmente e manda para a função **insertLines()**, onde as linhas do texto são inseridas no nó (palavra da esquerda ou palavra da direita) verificando em qual lista as linhas vão.

segue a parte do código:

```
for (i = 0; i < array_count; i++)
{
    if (str_contains(node->palavra_left, array_str[i]))
    {
        Data linha = dataCreate();
        linha->linha = i+1;

        Data search = ListSeach(node->linhasPalavraLeft, i+1); //checa pra evitar duplicação de
        if (search == NULL)
            listInsertOrdered(node->linhasPalavraLeft, linha);
    }

    if (strcmp(node->palavra_right, "") != 0 && str_contains(node->palavra_right, array_str[i]))
    {
        Data linha = dataCreate();
        linha->linha = i+1;

        Data search = ListSeach(node->linhasPalavraRight, i+1); //checa pra evitar duplicação de
        if (search == NULL)
            listInsertOrdered(node->linhasPalavraRight, linha);
    }
}
```

Conclusão

Árvores rubros-negras se propõe a melhorar o tempo de inserção de dados a custo de tem uma maior profundidade para um dos lados da árvore (direita ou esquerda, preferencialmente direita), logo o tempo de busca pode variar muito dependendo para que lado a informação está contida.

Árvores dois-três se propõe a melhorar a profundidade da árvore, consequentemente melhorando o tempo de busca, mas perdendo desempenho nas inserções e exclusão de dados, pois sempre a cada nova inserção, a árvore inteira deve ser balanceada.

referências:

[Árvore rubro-negra – Wikipédia, a enciclopédia livre \(wikipedia.org\)](#)

[Árvore 2-3 – Wikipédia, a enciclopédia livre \(wikipedia.org\)](#)

[Mussum Ipsum - O melhor lorem ipsum do mundis](#)

códigos de teste (feito por mim) para inserção e exclusão de dados numéricos a fim de demonstrar com mais clareza todo o processo:

[joaocarlos-losfe/Red-Black-Tree: implementação de uma árvore binaria em C \(github.com\)](#)

[joaocarlos-losfe/2_3-Tree: árvore 2-3 \(github.com\)](#)