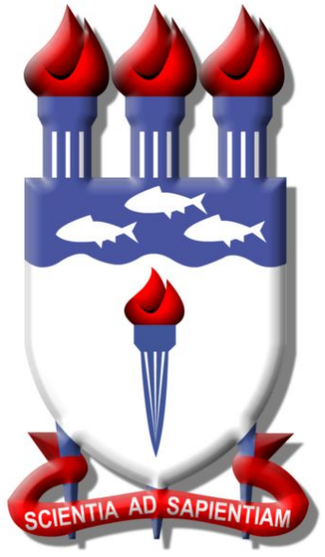


Arvore 2-3-4

Adrian Vieira
Allan Douglas
Aristoteles Peixoto
João da Silva Muniz
Mateus Monteiro

https://github.com/AriBarros/ProjetoP2

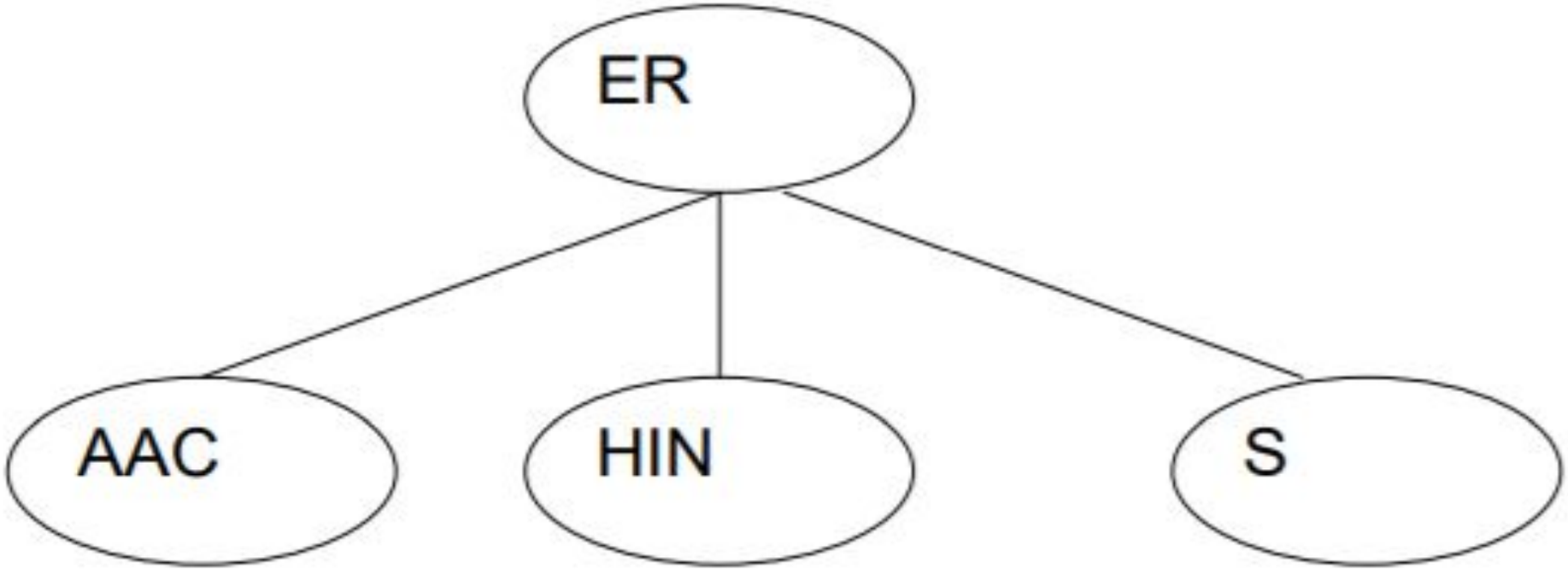
Motivação

- **Motivação:** Buscar em algum dicionário determinada palavra de forma mais rápida e eficiente? Ou uma forma de auto completar a palavra?

- **Estrutura:** Árvore 2-3-4

Introdução a árvore 2-3-4

Está estrutura trata-se de uma árvore balanceada e ordenada, apresentando cada nó máximo de 4 filhos. Como na árvore 2-3, ela se mantém com altura invariante (“balanço perfeito”).



Fonte: https://homepages.dcc.ufmg.br/~rprates/aedsII/TranspArvore234.pdf

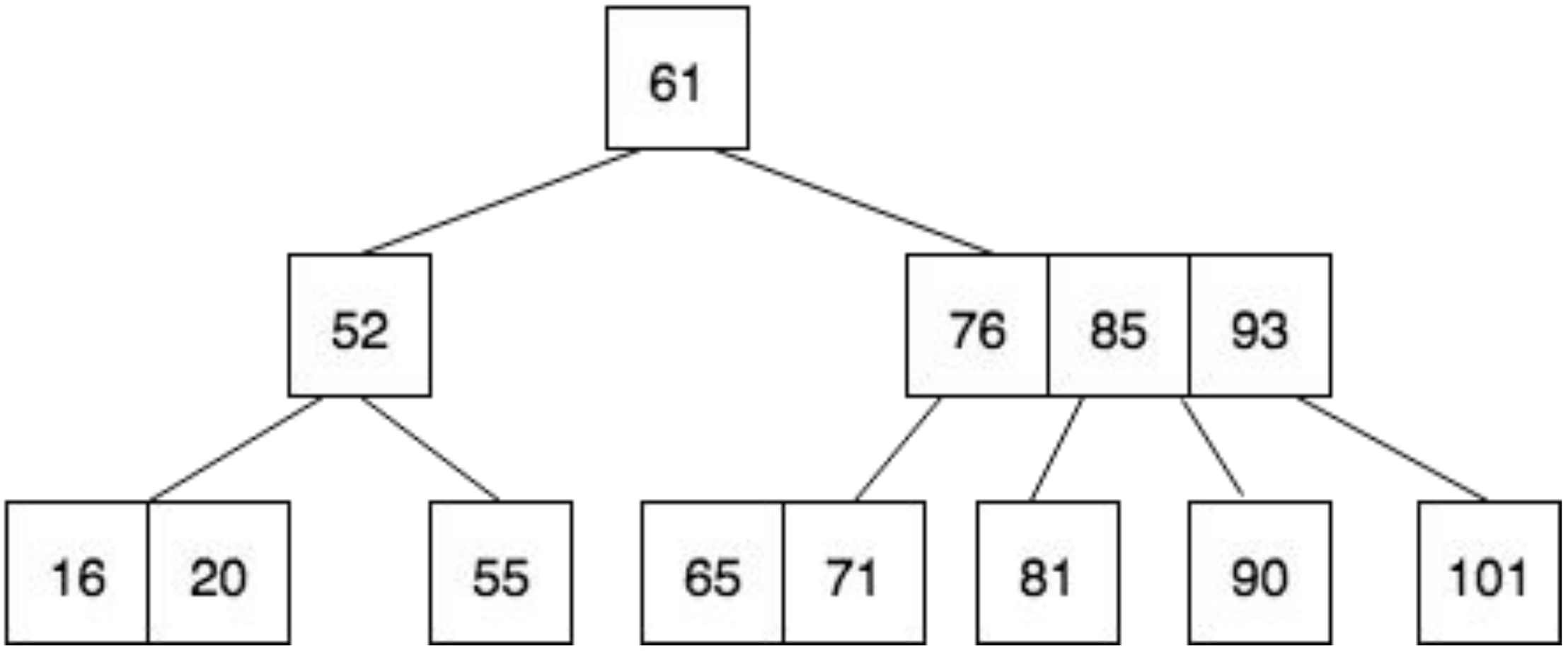
Introdução a árvore 2-3-4

Uma árvore 2-3-4 é uma estrutura de busca balanceada com três tipos de nós:

- O **2-nó** possui uma chave e dois nós filhos (assim como o nó da árvore de busca binária);
- O **3-nó** tem duas chaves e três nós filhos;
- O **4-nó** tem três chaves e quatro nós filhos.

Introdução a árvore 2-3-4

Se um nó tiver mais de uma chave (3 e 4 nós), as chaves deverão estar ordenadas. Isso garante que a travessia em ordem sempre produza as chaves ordenadas.



Fonte: https://algorithmtutor.com/Data-Structures/Tree/2-3-4-Trees/

Definição de árvore 2-3-4

Árvore 2-3-4 é um tipo de árvore de pesquisa múltipla. É uma árvore auto-equilibrada e está sempre perfeitamente balanceada.

Eficiência da árvore 2-3-4

*A busca de uma **árvore 2-3-4** é **O(log(n))** tanto para seu melhor quanto para seu pior caso. A distância entre a raiz e as folhas é a mesma sempre.*

Estrutura da árvore 2-3-4 em C

```c
#include <stdio.h>
#include <stdlib.h>
#define M 3

struct node {
        int n; /* n < M No. of keys in node will always less than order of B tree */
        int keys[M - 1]; /*array of keys*/
        struct node *p[M]; /* (n+1 pointers will be in use) */
}*root = NULL;

enum KeyStatus { Duplicate, SearchFailure, Success, InsertIt, LessKeys };
```

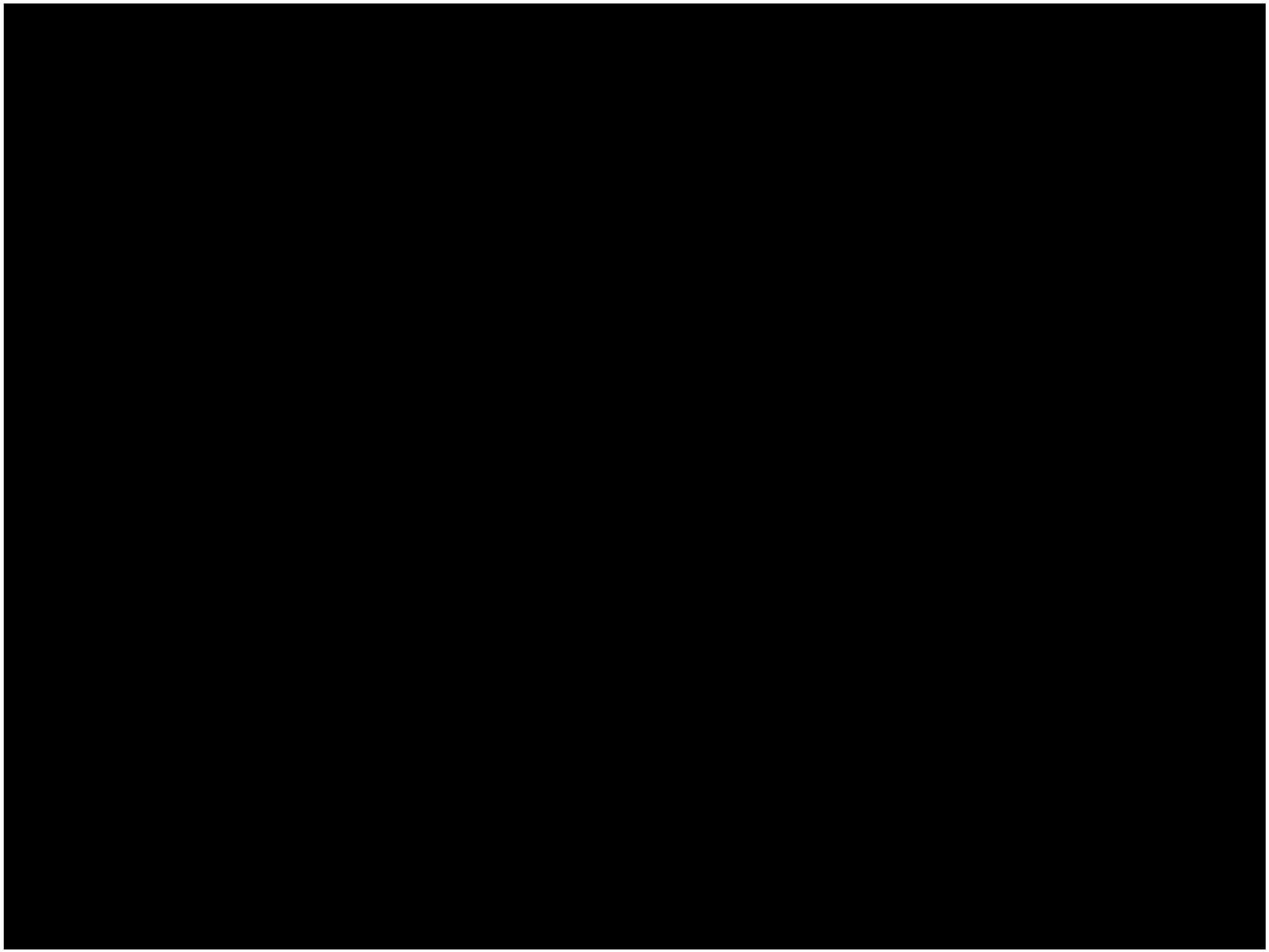
Fonte: https://github.com/YGAO008/2-3-4--Tree/blob/master/2-3-4%20Tree.c

Função: Inserir

```
void insert(int key)
{
        struct node *newnode;
        int upKey;
        enum KeyStatus value;
        value = ins(root, key, &upKey, &newnode);
        if (value == Duplicate)
                printf("Key already available\n");
        if (value == InsertIt)
        {
                struct node *uproot = root;
                root = malloc(sizeof(struct node));
                root->n = 1;
                root->keys[0] = upKey;
                root->p[0] = uproot;
                root->p[1] = newnode;
        }/*End of if */
}/*End of insert()*/
```

Fonte: https://github.com/YGAO008/2-3-4--Tree/blob/master/2-3-4%20Tree.c

Animação para o entendimento do código



Fonte: https://prezi.com/zcqwwn6msjq9/arvore-2-3-4/

Vantagens x Desvantagens

- Os tempos de busca são proporcionais a altura h dessa árvore;

- Alta facilidade de buscar, inserir e excluir elementos de uma árvore;

- Ela desperdiça muito espaço de memória alocado, já que alguns nós usam nem metade do que está disponível.

De volta à Motivação...

Podemos perceber que essa estrutura tem uma alta facilidade de buscar, inserir e excluir elementos de uma árvore. Mesmo tendo a desvantagem de ocupar memória desnecessária em alguns casos, no nosso exemplo do dicionário, ela deixa a busca por uma palavra muito rápida e simples por estar balanceada e por "quebrar" em vários casos menores, agilizando assim a busca.

Referências

https://algorithmtutor.com/Data-Structures/Tree/2-3-4-Trees/

https://www.educative.io/page/5689413791121408/80001

https://prezi.com/zcqwwn6msjq9/arvore-2-3-4/

https://github.com/YGAO008/2-3-4--Tree/blob/master/2-3-4%20Tree.c

https://homepages.dcc.ufmg.br/~rprates/aedsII/TranspArvore234.pdf