30.09.2024

Tarefa 4 - Árvores Binárias, Binárias de Busca e de Busca Balanceada

Júlia Tadeu - 2312392

Theo Canuto - 2311293

Professor Luiz Fernando Seibel

INF1010 - 3WA

1. **Objetivo**

O objetivo deste trabalho é implementar e explorar as funcionalidades de diferentes tipos de árvores binárias, incluindo árvores binárias comuns, árvores binárias de busca (BST) e árvores AVL, através de três tarefas específicas:

- **Inserção de chaves em uma árvore binária:** Inserir as chaves fornecidas em uma árvore binária, onde o número de nós de suas subárvores será calculado e armazenado no nó correspondente. O programa deve exibir a árvore inicial (com zeros representando o número de nós das subárvores) e a árvore resultante, com os valores calculados.

- **Verificação de árvore binária de busca:** Verificar se uma dada árvore é uma árvore binária de busca, confirmando se a árvore atende às propriedades de uma BST, onde para cada nó, todos os nós à esquerda são menores e os à direita são maiores que o nó.

- **Verificação de árvore AVL:** Avaliar se a árvore inserida é uma árvore AVL, verificando se, além de ser BST, ela mantém a condição de balanceamento, ou seja, se a diferença de altura entre as subárvores esquerda e direita de cada nó é no máximo 1.

Essas tarefas proporcionarão uma compreensão mais profunda sobre a construção, balanceamento e análise de árvores binárias, com foco nas estruturas mais avançadas como BSTs e árvores AVL.

2. **Estrutura do programa**

O programa é composto por uma estrutura de dados para nós de uma árvore binária e várias funções que implementam as funcionalidades requeridas: inserção de nós, cálculo do número de nós nas subárvores, verificação se a árvore é uma árvore binária de busca (BST), e verificação se a árvore é uma árvore AVL. Cada uma dessas funcionalidades é descrita detalhadamente abaixo.

Estrutura de Dados:

```
typedef struct Node {
    int key;
    int num_subtrees;
    struct Node* left;
    struct Node* right;
} Node;
```

A estrutura de dados Node é um nó da árvore binária, que contém:

- *key*: Um número inteiro que representa a chave armazenada no nó.

- *num_subtrees*: Um número inteiro que armazena o número de nós nas subárvores do nó.
- *left*: Um ponteiro para o nó filho à esquerda.
- *right*: Um ponteiro para o nó filho à direita.

Função *Node* create_node(int key)*:

- **Descrição:** Cria um novo nó na árvore binária com a chave fornecida. Os campos *left*, *right* e *num_subtrees* são inicializados com valores padrão (*NULL* e 0, respectivamente).
- **Parâmetros:** *int key:* A chave a ser armazenada no nó.
- **Retorno:** Retorna um ponteiro para o novo nó criado.

Função *Node* insert(Node* root, int key):*

- **Descrição:** Insere uma chave na árvore binária, seguindo a propriedade de ordem (BST). Se a chave é menor que a chave do nó atual, ela é inserida à esquerda, se maior, à direita.
- **Parâmetros:** *Node* root*: Ponteiro para o nó raiz da árvore. *int key*: A chave a ser inserida.
- **Retorno:** Retorna o ponteiro para o nó raiz, com o nó inserido na posição correta.

Função *int calculate_subtrees(Node* root):*

- **Descrição:** Calcula recursivamente o número de nós nas subárvores de cada nó. A função percorre a árvore e para cada nó, conta o número de nós em suas subárvores esquerda e direita, e armazena esse valor no campo *num_subtrees* do nó.
- **Parâmetros:** *Node* root:* Ponteiro para o nó raiz da árvore.
- **Retorno:** Retorna o número total de nós na árvore (incluindo o nó raiz).

Função *void print_tree(Node* root)*:

- **Descrição:** Imprime os nós da árvore binária em ordem, exibindo a chave de cada nó, o número de nós nas suas subárvores e os endereços de memória dos nós filhos (esquerdo e direito).
- **Parâmetros:** *Node* root:* Ponteiro para o nó raiz da árvore.
- **Retorno:** A função não retorna valores.

Função *int is_bst(Node root, Node min_node, Node* max_node)*:

- **Descrição:** Verifica se a árvore binária é uma árvore binária de busca (BST), validando se as chaves de cada nó seguem a propriedade de BST, onde o valor da chave do nó deve ser maior que todos os valores da subárvore esquerda e menor que todos os valores da subárvore direita.

- **Parâmetros:** *Node* root:* Ponteiro para o nó raiz da árvore. *Node* min_node*: Ponteiro para o nó com o menor valor aceitável para a subárvore. *Node* max_node*: Ponteiro para o nó com o maior valor aceitável para a subárvore.
- **Retorno:** Retorna 1 se a árvore é uma BST, caso contrário, retorna 0.

Função *int height(Node* node):*

- **Descrição:** Calcula a altura de um nó na árvore, que é definida como a maior distância entre o nó em questão e uma folha.
- **Parâmetros:** *Node* node*: Ponteiro para o nó cuja altura será calculada.
- **Retorno:** Retorna a altura do nó.

Função *int is_avl(Node* root):*

- **Descrição:** Verifica se a árvore é uma árvore AVL, garantindo que a diferença entre a altura das subárvores esquerda e direita de cada nó não seja maior que 1.
- **Parâmetros:** *Node* root*: Ponteiro para o nó raiz da árvore.
- **Retorno:** Retorna 1 se a árvore for AVL, caso contrário, retorna 0.

Função *int main():*

- **Descrição:** Ponto de entrada do programa. Cria a árvore binária, inserindo as chaves fornecidas, exibe a árvore inicial, calcula o número de nós nas subárvores de cada nó e exibe a árvore resultante com os valores atualizados. Em seguida, verifica se a árvore é uma árvore binária de busca (BST) e se a árvore é uma árvore AVL, printando esse diagnóstico.
- **Parâmetros:** Não possui parâmetros.
- **Retorno:** Retorna um inteiro (0), indicando a execução bem-sucedida do programa.

3. **Soluções**

A solução para a implementação das operações em árvores binárias, árvores binárias de busca (BST) e árvores AVL foi desenvolvida usando uma estrutura de nós para representar cada elemento da árvore e um conjunto de funções recursivas para realizar as operações necessárias. Abaixo estão os passos e as abordagens adotadas:

- Criação e Inserção de Nós:

A função *create_node* é responsável por criar um novo nó da árvore binária, inicializando o valor da chave, o número de nós nas subárvores como 0, e os ponteiros para os filhos esquerdo e direito como *NULL*. A função insert insere os nós na árvore de acordo com as regras de uma árvore binária de busca (BST), onde o nó é inserido à esquerda se sua chave for menor que a do nó atual, e à direita se for maior.

- Cálculo do Número de Nós nas Subárvores:

A função *calculate_subtrees* percorre a árvore de forma recursiva e calcula o número de nós em cada subárvore (esquerda e direita), armazenando esse valor no campo *num_subtrees* de cada nó. A soma das subárvores é feita utilizando uma abordagem pós-ordem, o que garante que a contagem esteja correta antes de retornar para o nó pai.

- Impressão da Árvore:

A função *print_tree* exibe a estrutura da árvore em ordem, imprimindo a chave do nó, o número de nós em suas subárvores, e os endereços de memória de seus filhos (esquerda e direita). A função é chamada de forma recursiva para garantir que a árvore seja percorrida e impressa corretamente.

- Verificação de Árvore Binária de Busca (BST):

A função *is_bst* verifica se a árvore binária segue as propriedades de uma árvore binária de busca. Para isso, ela compara cada nó com seus limites mínimo e máximo aceitáveis, garantindo que os nós da subárvore esquerda sejam menores que o nó atual e os da subárvore direita sejam maiores. O algoritmo é recursivo e utiliza uma abordagem *top-down* para checagem.

- Cálculo da Altura da Árvore:

A função *height* calcula a altura de cada nó, definida como a distância máxima entre o nó e uma folha. A altura de um nó é utilizada para verificar o balanceamento da árvore na etapa de verificação de AVL.

- Verificação de Árvore AVL:

A função *is_avl* utiliza a altura das subárvores esquerda e direita para verificar o balanceamento da árvore AVL. Para que uma árvore seja AVL, a diferença de altura entre as subárvores de cada nó não pode ser maior que 1. A verificação é feita de forma recursiva, percorrendo toda a árvore e garantindo que cada nó atenda à condição de balanceamento.

Testes e soluções:

Foram realizados testes utilizando diferentes conjuntos de chaves para garantir que o programa se comportasse corretamente. Abaixo estão alguns exemplos de cenários de teste:

- **Cenários 1:** BST mas não AVL

Entrada: {15, 17, 3, 5, 2, 20, 25, 13, 10, 16}

Esperado: uma árvore desbalanceada, onde a função *is_avl* corretamente identifica que a árvore não é AVL, mas a função *is_bst* confirma que a árvore é uma BST válida.

Saída:

```
Initial tree:

Key: 2
Number of nodes on subtrees: 0
Address: 015DEC50
Left Child: NULL
Right Child: NULL

Key: 3
Number of nodes on subtrees: 0
Address: 015DEBD0
Left Child: 2 (Address: 015DEC50)
Right Child: 5 (Address: 015DEC10)

Key: 5
Number of nodes on subtrees: 0
Address: 015DEC10
Left Child: NULL
Right Child: 13 (Address: 015DFCE0)

Key: 10
Number of nodes on subtrees: 0
Address: 015DFBE0
Left Child: NULL
Right Child: NULL

Key: 13
Number of nodes on subtrees: 0
Address: 015DFCE0
Left Child: 10 (Address: 015DFBE0)
Right Child: NULL

Key: 15
Number of nodes on subtrees: 0
Address: 015D8370
Left Child: 3 (Address: 015DEBD0)
Right Child: 17 (Address: 015D83B0)

Key: 16
Number of nodes on subtrees: 0
Address: 015DFA60
Left Child: NULL
Right Child: NULL

Key: 17
Number of nodes on subtrees: 0
Address: 015D83B0
Left Child: 16 (Address: 015DFA60)
Right Child: 20 (Address: 015DF538)
```

```
Key: 20
Number of nodes on subtrees: 0
Address: 015DF538
Left Child: NULL
Right Child: 25 (Address: 015DF578)

Key: 25
Number of nodes on subtrees: 0
Address: 015DF578
Left Child: NULL
Right Child: NULL


Calculating the number of nodes in the subtrees...

Resulting tree:
Key: 2
Number of nodes on subtrees: 0
Address: 015DEC50
Left Child: NULL
Right Child: NULL

Key: 3
Number of nodes on subtrees: 4
Address: 015DEBD0
Left Child: 2 (Address: 015DEC50)
Right Child: 5 (Address: 015DEC10)

Key: 5
Number of nodes on subtrees: 2
Address: 015DEC10
Left Child: NULL
Right Child: 13 (Address: 015DFCE0)

Key: 10
Number of nodes on subtrees: 0
Address: 015DFBE0
Left Child: NULL
Right Child: NULL

Key: 13
Number of nodes on subtrees: 1
Address: 015DFCE0
Left Child: 10 (Address: 015DFBE0)
Right Child: NULL
```

```
Key: 13
Number of nodes on subtrees: 1
Address: 015DFCE0
Left Child: 10 (Address: 015DFBE0)
Right Child: NULL

Key: 15
Number of nodes on subtrees: 9
Address: 015D8370
Left Child: 3 (Address: 015DEBD0)
Right Child: 17 (Address: 015D83B0)

Key: 16
Number of nodes on subtrees: 0
Address: 015DFA60
Left Child: NULL
Right Child: NULL

Key: 17
Number of nodes on subtrees: 3
Address: 015D83B0
Left Child: 16 (Address: 015DFA60)
Right Child: 20 (Address: 015DF538)

Key: 20
Number of nodes on subtrees: 1
Address: 015DF538
Left Child: NULL
Right Child: 25 (Address: 015DF578)

Key: 25
Number of nodes on subtrees: 0
Address: 015DF578
Left Child: NULL
Right Child: NULL


This tree is a BST.
The tree is NOT an AVL tree.
```

- **Cenário 2:** BST e AVL

Entrada: {15, 3, 20, 2, 10, 17, 25, 5, 13, 16}

Esperado: árvore balanceada, onde tanto a função *is_bst* quanto *is_avl* confirmam que a árvore é uma BST e uma AVL, respectivamente.

Saída:

```
Initial tree:

Key: 2
Number of nodes on subtrees: 0
Address: 01028318
Left Child: NULL
Right Child: NULL

Key: 3
Number of nodes on subtrees: 0
Address: 0102AF38
Left Child: 2 (Address: 01028318)
Right Child: 10 (Address: 0102E900)

Key: 5
Number of nodes on subtrees: 0
Address: 0102F498
Left Child: NULL
Right Child: NULL

Key: 10
Number of nodes on subtrees: 0
Address: 0102E900
Left Child: 5 (Address: 0102F498)
Right Child: 13 (Address: 0102F540)

Key: 13
Number of nodes on subtrees: 0
Address: 0102F540
Left Child: NULL
Right Child: NULL

Key: 15
Number of nodes on subtrees: 0
Address: 0102AEF8
Left Child: 3 (Address: 0102AF38)
Right Child: 20 (Address: 010282D8)

Key: 16
Number of nodes on subtrees: 0
Address: 0102FAC0
Left Child: NULL
Right Child: NULL

Key: 17
Number of nodes on subtrees: 0
Address: 0102E940
Left Child: 16 (Address: 0102FAC0)
Right Child: NULL
```

```
Key: 20
Number of nodes on subtrees: 0
Address: 010282D8
Left Child: 17 (Address: 0102E940)
Right Child: 25 (Address: 0102E980)

Key: 25
Number of nodes on subtrees: 0
Address: 0102E980
Left Child: NULL
Right Child: NULL


Calculating the number of nodes in the subtrees...

Resulting tree:
Key: 2
Number of nodes on subtrees: 0
Address: 01028318
Left Child: NULL
Right Child: NULL

Key: 3
Number of nodes on subtrees: 4
Address: 0102AF38
Left Child: 2 (Address: 01028318)
Right Child: 10 (Address: 0102E900)

Key: 5
Number of nodes on subtrees: 0
Address: 0102F498
Left Child: NULL
Right Child: NULL

Key: 10
Number of nodes on subtrees: 2
Address: 0102E900
Left Child: 5 (Address: 0102F498)
Right Child: 13 (Address: 0102F540)

Key: 13
Number of nodes on subtrees: 0
Address: 0102F540
Left Child: NULL
Right Child: NULL
```

```
Key: 15
Number of nodes on subtrees: 9
Address: 0102AEF8
Left Child: 3 (Address: 0102AF38)
Right Child: 20 (Address: 010282D8)

Key: 16
Number of nodes on subtrees: 0
Address: 0102FAC0
Left Child: NULL
Right Child: NULL

Key: 17
Number of nodes on subtrees: 1
Address: 0102E940
Left Child: 16 (Address: 0102FAC0)
Right Child: NULL

Key: 20
Number of nodes on subtrees: 3
Address: 010282D8
Left Child: 17 (Address: 0102E940)
Right Child: 25 (Address: 0102E980)

Key: 25
Number of nodes on subtrees: 0
Address: 0102E980
Left Child: NULL
Right Child: NULL


This tree is a BST.
The tree is an AVL tree.
```

4. **Observações e conclusões**

Durante a implementação das funcionalidades solicitadas para árvores binárias, árvores binárias de busca e árvores AVL, diversos pontos importantes foram identificados:

Facilidades:

- Criação e inserção de nós: A implementação da função de criação de nós e a inserção em uma árvore binária de busca (BST) foram realizadas de forma direta, aproveitando a simplicidade da lógica de comparação para inserir os nós nas subárvores corretas. A função *insert* seguiu a estrutura padrão de inserção em BSTs, garantindo que a propriedade da ordem fosse preservada.

- Cálculo do número de nós nas subárvores: A função *calculate_subtrees* mostrou-se eficiente ao percorrer a árvore de forma recursiva, somando corretamente o número de nós nas subárvores de cada nó. Essa abordagem garantiu que a contagem fosse precisa, mesmo para árvores mais complexas.

- Verificação de BST e AVL: A implementação da verificação de BST foi facilitada pela abordagem recursiva de comparação entre limites mínimos e máximos de nós. A verificação da propriedade AVL também foi direta, calculando a altura de cada subárvore e garantindo que a diferença de altura entre as subárvores não excedesse 1.

Dificuldades:

- De modo geral, apesar dos algoritmos prontos/próprios para isso, houve dificuldade em trabalhar com recursividade no código. A aplicação prática, na hora de programar, trouxe algumas dificuldades para a implementação do projeto, apesar do domínio teórico.

Conclusão:

A implementação das funcionalidades propostas foi bem sucedida, permitindo a construção de árvores binárias, a verificação de suas propriedades como BST e AVL, e o cálculo do número de nós nas subárvores. O programa demonstrou ser eficaz nos cenários de teste e funcionou conforme o esperado.

Para futuras melhorias, seria interessante implementar o balanceamento automático das árvores AVL, utilizando rotações para corrigir desbalanceamentos durante a inserção de nós. No geral, o programa atingiu os objetivos propostos e proporcionou uma boa compreensão dos conceitos relacionados a árvores binárias e suas variações.