28.11.2024

Trabalho 2 - Implementação de uma Árvore B+

Júlia Tadeu - 2312392

Theo Canuto - 2311293

Professor Luiz Fernando Seibel

INF1010 - 3WA

2024.2

1. **Objetivo**

O objetivo deste trabalho é implementar uma árvore B+ (2-3) com, no máximo, duas chaves e três ponteiros por nó, permitindo operações de inclusão, exclusão e busca de chaves. A inserção deve ser realizada com cisão dos nós quando necessário, enquanto a exclusão deve tratar cenários de concatenação e redistribuição. Ao final, espera-se que a árvore seja capaz de armazenar dados de forma eficiente e balanceada, representando corretamente a estrutura solicitada e facilitando a navegação e a busca de informações.

2. **Estrutura do programa**

O programa é composto por uma estrutura de dados principal e treze funções essenciais, além da função main. A estrutura de dados é usada para representar os nós da Árvore B+, enquanto as funções desempenham papeis fundamentais para criar, inserir, excluir, buscar e imprimir a árvore. A seguir, são descritas a estrutura de dados e as funções:

Estrutura de Dados:

```
typedef struct node {
    int key1;
    int key2;
    int overflow;
    int isLeaf;
    struct node* pointers[MAX_CHILDREN];
    struct node* parent;
    struct node* pOverflow;
} Node;
```

A estrutura *Node* define os nós da Árvore B+, contendo:

- *key1:* Uma chave inteira que armazena o primeiro valor no nó.
- *key2:* Uma chave inteira que armazena o segundo valor no nó
- *overflow:* Uma chave extra, utilizada em caso de overflow do nó.
- *isLeaf:* Um inteiro que indica se o nó é folha (1) ou interno (0).
- *pointers[3]:* Um array de ponteiros para filhos.
- *parent*: Um ponteiro para o nó pai.
- *pOverflow:* Um ponteiro para o próximo nó folha, conectando as folhas em sequência.

Função *main(void):*

- **Descrição:** A função main executa uma série de testes para verificar o comportamento da árvore B+ implementada. Cada caso representa uma operação diferente sobre a árvore, como inserção, busca e remoção de nós. Para cada operação, a árvore é impressa para verificar sua estrutura.

- **Parâmetros:** Não recebe parâmetros.
- **Retorno:** Não retorna valor.

Função *createNode(int isLeaf):*

- **Descrição:** Cria um novo nó para a árvore B+, inicializando suas chaves e ponteiros com valores padrões. O nó é configurado como folha ou interno, dependendo do valor do parâmetro *isLeaf*.
- **Parâmetros:** *int isLeaf:* Define se o nó será uma folha (1) ou um nó interno (0).
- **Retorno:** Retorna um ponteiro para o novo nó criado.

Função *insertLeaf(Node leaf, int n):*

- **Descrição:** Insere um valor n no nó folha. Se o nó ainda tiver espaço, o valor é inserido nas chaves *key1* ou *key2*. Caso contrário, o valor é colocado no campo *overflow*. A função mantém a ordem das chaves, reorganizando quando necessário.
- **Parâmetros:** *Node leaf*:* O ponteiro para o nó folha onde o valor será inserido; *int n:* O valor que será inserido no nó folha.
- **Retorno:** Retorna o ponteiro para o nó folha após a inserção do valor.

Função *splitLeaf(Node childNode, Node parentNode):*

- **Descrição:** Divide um nó folha que está cheio em dois nós e insere a chave do meio no nó pai. Se o nó pai também estiver cheio, a função pode causar recursão para lidar com cisões adicionais.
- **Parâmetros:** *Node childNode*:* O nó folha que será dividido. Contém as chaves e filhos a serem redistribuídos; *Node parentNode*:* O nó pai que receberá a chave do meio após a divisão. Se o nó pai for *NULL*, ele é criado.
- **Retorno:** Retorna o nó pai após a cisão, que pode ter sido modificado ou criado, dependendo da situação.

Função *insert(Node bPlusTree, int n)*:

- **Descrição:** Insere um valor n na árvore B+. A função percorre a árvore até encontrar uma folha onde o valor pode ser inserido. Se a folha estiver cheia, ela será dividida (com a chave do meio sendo promovida ao nó pai). Caso o nó pai também esteja cheio, a função recursivamente faz a divisão dos nós até a raiz, garantindo que a árvore se mantenha balanceada.
- **Parâmetros:** *Node bPlusTree*:* A raiz da árvore B+ onde o valor será inserido; *int n:* O valor a ser inserido na árvore.
- **Retorno:** Retorna o nó raiz da árvore B+ após a inserção e possíveis cisões.

Função *checkParent(Node parentNode)*:

- **Descrição:** Essa função é responsável por verificar e ajustar o ponteiro para o nó pai de todos os nós filhos do nó *parentNode*. A função percorre recursivamente a árvore, garantindo que todos os filhos diretos (*pointers*) de um nó pai tenham o ponteiro *parent* corretamente atualizado. A função também é chamada recursivamente para os

filhos, garantindo que todos os nós da árvore estejam com seus ponteiros *parent* ajustados.
- **Parâmetros:** *Node parentNode*:* O nó pai cujos filhos precisam ter seus ponteiros *parent* ajustados.
- **Retorno:** Não há retorno.

Função *removeNode(Node* bPlusTree, int n)*:
- **Descrição:** Remove a chave n de uma árvore B+. Se a chave for encontrada, ela é removida e a árvore é reestruturada para garantir a sua integridade e balanceamento. Se a chave não for encontrada, a árvore permanece inalterada.
- **Parâmetros:** *Node bPlusTree**: Ponteiro para a raiz da árvore B+; *int n:* A chave que deve ser removida da árvore.
- **Retorno:** Retorna a árvore B+ atualizada após a remoção da chave n. Se a chave não for encontrada, a árvore não é alterada.

Função *fixNodes(Node tree)*:
- **Descrição:** Corrige a estrutura da árvore B+ após a remoção de uma chave. A função redistribui as chaves entre os nós filhos ou realiza a concatenação de nós quando necessário.
- **Parâmetros:** *Node tree:** O nó da árvore que precisa ser corrigido após uma remoção ou redistribuição de chaves.
- **Retorno:** Retorna o nó corrigido, ou a árvore reestruturada, após as operações de redistribuição ou concatenação.

Função *getFromLeft(Node tree, int pIndex)*:
- **Descrição:** Redistribui uma chave do nó à esquerda (no pai da árvore) para o nó atual (*tree*). Se o nó à esquerda estiver com mais de uma chave, uma chave será movida para o nó atual, e a chave do pai será promovida.
- **Parâmetros:** *Node tree:** O nó que precisa ser corrigido, recebendo a chave do nó à esquerda; *int pIndex:* O índice que indica a posição do nó atual no seu nó pai.
- **Retorno:** Retorna o nó corrigido após a redistribuição de chaves.

Função *getFromRight(Node tree, int pIndex)*:
- **Descrição:** Redistribui uma chave do nó à direita (no pai da árvore) para o nó atual (*tree*). Se o nó à direita tiver mais de uma chave, uma chave será movida para o nó atual, e a chave do pai será promovida.
- **Parâmetros:** *Node tree:* O nó que precisa ser corrigido, recebendo a chave do nó à direita; int pIndex: O índice que indica a posição do nó atual no seu nó pai.*
- **Retorno:** Retorna o nó corrigido após a redistribuição de chaves.

Função *concRight(Node* tree):
- **Descrição:** Concatena o nó atual (*tree*) com o nó à sua direita (dentro do nó pai). Se o nó à direita contiver chaves, essas chaves são movidas para o nó atual, e a chave do nó pai é promovida. A concatenação é realizada para garantir que a árvore B+

continue balanceada após remoções ou cisões. Esse processo é feito atualizando os ponteiros e as chaves do nó atual, do nó à direita e do nó pai.
- **Parâmetros:** *Node tree:** O nó que precisa ser corrigido (o nó que será concatenado com o nó à sua direita).
- **Retorno:** Retorna o nó atualizado após a concatenação.

Função *search(Node bPlusTree, int n)*:
- **Descrição:** Realiza a busca de um valor n na árvore B+. Ela percorre os nós internos até encontrar a folha correspondente. A busca segue a estrutura da árvore, verificando as chaves em cada nó para decidir qual ponteiro seguir até encontrar a folha. Uma vez na folha, o valor n é comparado com as chaves do nó. Se encontrado, o nó contendo o valor é retornado, caso contrário, *NULL* é retornado, indicando que o valor não está presente na árvore.
- **Parâmetros:** *Node bPlusTree:** A raiz ou o nó atual da árvore B+ onde a busca será iniciada; *int n:* O valor que está sendo buscado na árvore.
- **Retorno:** Retorna o ponteiro para o nó da folha que contém o valor n, ou NULL caso o valor não seja encontrado na árvore.

Função *printNode(Node node)*:
- **Descrição:** Imprime as informações de um nó da árvore B+, exibindo suas chaves (*key1* e *key2*), bem como os ponteiros para seus filhos (*pointers[0]*, *pointers[1]*, *pointers[2]*). **Parâmetros:** *Node node:** O nó da árvore B+ que será impresso.
- **Retorno:** Não possui retorno.

Função *printTree(Node bPlusTree):*
- **Descrição:** Imprime a estrutura de uma árvore B+ de forma recursiva. Começando pela raiz, ela exibe o nó atual utilizando a função *printNode*. Se o nó não for uma folha, a função percorre os filhos (ponteiros para outros nós), imprimindo-os recursivamente.
- **Parâmetros:** *Node bPlusTree:** A raiz da árvore B+ que será impressa.
- **Retorno:** Não possui retorno.

3. **Soluções**

A solução para a construção e manipulação da árvore B+ foi implementada utilizando nós encadeados, onde cada nó contém até duas chaves e pode ter até três ponteiros para seus filhos (esquerdo, meio e direito). A árvore é montada com base na inserção de chaves e reestruturada automaticamente através de cisões quando o número máximo de chaves é atingido em um nó. Abaixo estão os passos detalhados:

a. Criação de Nós:

A função *createNode* é responsável por criar e inicializar um novo nó na árvore. Cada nó é criado com 2 chaves inicializadas com -1, 3 ponteiros-filho, um ponteiro-pai e um

ponteiro para sinalizar *overflow* todos inicializados como *NULL*, além de uma *flag isLeaf* recebida como parâmetro.

b. Inserção de nós folha:

A inserção dos nós folha na árvore B+ é feita de maneira controlada, garantindo que cada nó folha armazene, no máximo, duas chaves e tenha, no máximo, três filhos. A função *insertLeaf* gerencia a inserção das chaves em cada nó folha, assegurando que os dados sejam inseridos em ordem crescente. Quando um nó folha tem espaço disponível (ou seja, quando um dos seus campos de chave está vazio), a chave é simplesmente colocada nesse espaço.

Se ambos os campos de chave estiverem preenchidos, o processo de inserção decide a posição das novas chaves comparando-as com as chaves já presentes. Se a chave a ser inserida for maior que a segunda chave (*key2*), ela será colocada no campo de overflow. Se for maior que a primeira chave (*key1*) mas menor que a segunda, a chave é deslocada para o segundo campo, e a chave anterior é movida para o campo de *overflow*. Finalmente, se a chave for menor do que a primeira, todas as chaves são deslocadas, e a nova chave é colocada no primeiro campo.

Quando não há mais espaço suficiente para inserir a chave diretamente no nó folha, um *overflow* ocorre, e o valor excedente é movido para um nó subsequente ou novo nó folha. Esse processo mantém a integridade da árvore B+, com suas propriedades de balanceamento e ordenação.

c. Cisão (*Split*)

A divisão de um nó folha na árvore B+ ocorre quando um nó folha atinge seu limite de capacidade, ou seja, quando ele já contém duas chaves e um novo valor precisa ser inserido. A função *splitLeaf* é responsável por dividir o nó folha em dois, movendo uma chave para o nó pai e criando um novo nó folha para armazenar a chave excedente.

O processo começa verificando se o nó pai existe. Caso não, um novo nó pai é criado. Se o nó pai ainda tiver espaço, a chave excedente do nó folha (que é a segunda chave, *key2*) é movida para o nó pai. Em seguida, um novo nó folha é criado para armazenar a chave excedente (ou *overflow*), juntamente com qualquer valor adicional que não couber no nó original. Dependendo da estrutura do nó (se for folha ou não), as chaves e ponteiros são reorganizados de acordo com as regras da árvore B+.

Quando o nó pai já está cheio, a chave do nó folha é comparada com as chaves presentes no nó pai, e a inserção é feita de forma a manter a ordenação. Se necessário, o nó pai pode ser dividido também, propagando a divisão para níveis superiores.

Se o nó pai já tiver a capacidade máxima de chaves (ou seja, quando *key1*, *key2* e *overflow* estão ocupados), o valor excedente é movido para o nó pai e a função realiza uma divisão recursiva, propagando a necessidade de divisão até que o nó pai tenha espaço suficiente ou um novo nó pai seja criado.

Esse processo assegura que a árvore B+ mantenha sua estrutura balanceada e suas propriedades de ordenação, garantindo uma busca eficiente e uma distribuição balanceada dos dados.

d. Inserção de nós:

A inserção de um nó na árvore B+ é realizada de forma iterativa, começando pela raiz da árvore e percorrendo os nós até chegar a um nó folha. A função *insert* é responsável por localizar o nó folha apropriado para a inserção de um novo valor e, caso necessário, realizar a divisão de nós para garantir que a estrutura da árvore seja mantida.

A inserção começa na raiz da árvore B+, com a variável *currentNode* apontando para o nó raiz. A função percorre a árvore descendo de nó em nó, seguindo os ponteiros para o próximo nó de acordo com a comparação das chaves. Se o valor a ser inserido (n) for menor que a primeira chave (*key1*) do nó atual, o ponteiro para o próximo nó a ser seguido é o *pointers[0]*. Se for maior que *key1* mas menor que a segunda chave (*key2*) ou se a segunda chave não estiver preenchida, o ponteiro *pointers[1]* é seguido. Caso contrário, o ponteiro *pointers[2]* é seguido, movendo o processo para o próximo nível da árvore.

Ao chegar a um nó folha, a função *insertLeaf* é chamada para inserir a chave no nó folha encontrado. Se a inserção não causar *overflow* (ou seja, o nó folha não ultrapassar a quantidade máxima de chaves), o processo é concluído e a árvore B+ não precisa ser modificada.

Entretanto, se o nó folha sofrer um *overflow* (isto é, se o nó folha contiver mais de duas chaves após a inserção), o nó folha é dividido. O valor excedente é movido para um novo nó folha, e a divisão é propagada para o nó pai, caso necessário, por meio da função *splitLeaf*. Esse processo de divisão pode ser recursivo, propagando-se para os nós pais até que um nó pai tenha espaço suficiente ou que uma nova raiz seja criada.

A árvore B+ é então ajustada para manter a sua estrutura balanceada, com cada nó possuindo no máximo duas chaves e três ponteiros, respeitando a ordenação das chaves em cada nível. Quando necessário, um novo nó pai pode ser criado para manter a árvore balanceada. Por fim, a função *checkParent* é chamada para verificar e ajustar qualquer inconsistência nos nós pais, garantindo que a árvore B+ permaneça correta após a inserção e possíveis divisões.

e. Verificação/Atualização dos Pais

A função *checkParent* é responsável por garantir que os ponteiros de *parent* de cada nó na árvore B+ sejam corretamente atribuídos após a inserção ou divisão de nós. Ela percorre a árvore de maneira recursiva, ajustando o ponteiro para o nó pai de todos os nós filhos, começando pelo nó fornecido como argumento (*parentNode*).

O processo começa verificando se o nó fornecido (*parentNode*) é nulo. Caso seja nulo, a função retorna, pois não há nada a ser ajustado. Se o nó não for nulo, a função prossegue para ajustar os ponteiros dos filhos desse nó.

O nó pai (*parentNode*) possui até três ponteiros para filhos: *pointers[0]*, *pointers[1]* e *pointers[2]*. Para cada ponteiro, a função verifica se o filho correspondente não é nulo. Se o filho não for nulo, a função atribui o nó pai ao ponteiro *parent* do nó filho e, em seguida, chama recursivamente a função *checkParent* para garantir que todos os filhos do nó também tenham o ponteiro *parent* atualizado. Isso é feito para cada um dos filhos, incluindo os filhos à esquerda (*LeftNode*), ao meio (*midNode*) e à direita (*rightNode*).

Esse processo de atualização recursiva assegura que, após qualquer alteração na árvore, como uma divisão de nó ou uma inserção, todos os ponteiros de parent sejam

consistentes em toda a árvore. A árvore B+ mantém sua integridade, com cada nó apontando corretamente para seu respectivo pai, o que facilita a navegação e manutenção da estrutura.

f. Remover Nó da Árvore B+:

A função *removeNode* tem como objetivo remover um nó da árvore B+, garantindo que as propriedades da árvore sejam mantidas após a remoção. A função começa procurando pela chave n nos nós da árvore até encontrar o nó onde a chave se encontra. Uma vez encontrado o nó, o valor n é removido e a árvore é reestruturada, se necessário, para garantir que continue balanceada.

Quando o nó a ser removido for uma folha e contiver apenas uma chave, a chave é removida e o nó pode ser fundido com um nó vizinho (à esquerda ou à direita), ou a chave é redistribuída do nó pai para manter o equilíbrio da árvore. Quando o nó a ser removido é uma folha com mais de uma chave, o valor n é simplesmente removido e a árvore é reorganizada de acordo com a sua estrutura.

A função também lida com a possibilidade de não encontrar a chave no nó atual, e nesse caso, ela continua a busca recursivamente nos filhos até encontrar o nó correspondente. Durante o processo de remoção, a árvore pode passar por uma reestruturação, realizando operações como redistribuição de chaves ou concatenação de nós, dependendo da necessidade.

Se a chave não for encontrada no nó da árvore, a função apenas retorna a árvore sem alterações. Caso contrário, após a remoção, a árvore é reorganizada e a função retorna o nó raiz da árvore B+ com as modificações realizadas.

g. Correção de nós:

A função *fixNodes* é responsável por corrigir a estrutura de nós na árvore B+ quando uma alteração, como a remoção de uma chave ou a fusão de nós, ocorre e afeta a consistência da árvore. Ela realiza a reorganização dos nós para garantir que a árvore mantenha suas propriedades de balanceamento e ordenação.

O processo começa verificando se o nó em questão é a raiz da árvore. Se for, e se o nó raiz tiver apenas um filho, a função realiza uma simplificação da árvore, promovendo o filho para a raiz e liberando o nó antigo. Nesse caso, as chaves e os ponteiros são ajustados de forma que o nó pai da árvore passe a apontar diretamente para o filho, e a árvore é então corrigida.

Se o nó não for a raiz ou se ele ainda tiver mais de um filho, a função prossegue verificando a posição do nó em relação ao seu pai e seus irmãos. A árvore é analisada para decidir se os ponteiros precisam ser reorganizados, se há a necessidade de pegar uma chave do irmão à esquerda ou à direita, ou mesmo se é necessário concatenar os nós.

A função utiliza um vetor *childArray* para armazenar os filhos do nó pai e determinar em qual posição o nó atual se encontra em relação aos seus irmãos. Dependendo dessa posição e da disponibilidade de chaves nos nós irmãos, a função tenta corrigir a estrutura da árvore através de operações como "pegar uma chave do irmão à esquerda" (função *getFromLeft*) ou "pegar uma chave do irmão à direita" (função *getFromRight*), ou até mesmo concatenando os nós (com a função *concRight*).

Esse processo de correção assegura que a árvore B+ continue balanceada e eficiente, mantendo suas propriedades, como a distribuição adequada de chaves e a manutenção da altura da árvore. A função também é recursiva, corrigindo os nós de forma ascendente, ajustando cada nível da árvore conforme necessário.

h. Mover uma chave do nó da esquerda para o nó atual:

A função *getFromLeft* é responsável por reorganizar os nós de uma árvore B+ durante o processo de remoção, quando um nó precisa pegar uma chave do seu irmão à esquerda (ou seja, um nó adjacente na árvore).

Quando um nó perde uma chave e não possui chaves suficientes para manter a estrutura da árvore, ele pode pegar uma chave do nó irmão à esquerda, promovendo essa chave para o nó pai. O índice *pIndex* é usado para identificar a posição do nó atual em relação ao seu nó pai, e com isso, determinar qual nó irmão à esquerda deve ser utilizado para realizar a redistribuição de chaves.

A função começa verificando quantas chaves estão presentes no nó atual (*tree*) e no nó irmão à esquerda (*left*). Caso o nó esquerdo tenha mais de uma chave, a chave do nó pai que se encontra no índice *pIndex* - 1 é promovida para o nó atual, e a chave mais à esquerda do nó irmão é movida para o nó pai.

O nó irmão à esquerda passa a "perder" uma chave, que é transferida para o nó atual, e o nó atual pode receber o ponteiro do nó irmão para a posição correta. A operação de redistribuição garante que as chaves nos nós da árvore permaneçam balanceadas, sem que a árvore perca a propriedade de árvore B+.

Se o nó à esquerda não tiver chaves suficientes para redistribuir, o nó atual pode precisar se fundir com o irmão à esquerda ou à direita. A função lida com a reorganização de ponteiros e chaves, realizando as modificações necessárias para preservar a integridade da árvore B+.

Após a operação de redistribuição, a função retorna o nó atual com as chaves reorganizadas e os ponteiros ajustados, garantindo que a árvore permaneça balanceada e suas propriedades preservadas.

i. Mover uma chave do nó da direita para o nó atual:

A função *getFromRight* é responsável por reorganizar os nós de uma árvore B+ durante o processo de remoção, quando um nó precisa pegar uma chave do seu irmão à direita (ou seja, um nó adjacente na árvore).

Quando um nó perde uma chave e não consegue manter a estrutura da árvore devido à falta de chaves, ele pode pegar uma chave do nó irmão à direita, promovendo essa chave para o nó pai. O índice pIndex é utilizado para identificar a posição do nó atual em relação ao seu nó pai, e assim, determinar qual nó irmão à direita deve fornecer uma chave.

Essa operação é essencial para garantir que a árvore B+ mantenha o balanceamento e as propriedades necessárias após a remoção de uma chave.

j. Concatenação de Nó com o Nó da Direita:

A função *concRight* é responsável por concatenar um nó com o nó à sua direita, quando a redistribuição de chaves entre nós não é possível (por exemplo, quando os dois nós

filhos de um nó interno não podem ser mesclados). Ela move a chave do nó pai para o nó atual e pega as chaves e ponteiros do nó à direita, colocando-os no nó atual. O nó da direita é então descartado após a concatenação.

k. Busca de um Valor na Árvore B+:

A função *search* é responsável por buscar um valor n na árvore B+. Ela começa na raiz da árvore e percorre os nós internos até encontrar a folha onde o valor pode estar armazenado. Caso o valor seja encontrado, a função retorna o nó que contém a chave. Caso contrário, retorna *NULL*, indicando que o valor não está presente na árvore.

l. Impressão de um Nó:

A função *printNode* é responsável por exibir as informações de um único nó da árvore B+. Ela imprime o endereço do nó, as duas chaves armazenadas no nó (*key1* e *key2*), e os ponteiros para os filhos (*pointers[0]*, *pointers[1]*, e *pointers[2]*). Essa função é chamada pela *printTree* para exibir os detalhes de cada nó durante a impressão recursiva da árvore.

m. Impressão da Árvore:

A função *printTree* é responsável por percorrer e exibir a estrutura da árvore B+ de forma recursiva. Ela imprime as informações do nó atual, incluindo suas chaves e ponteiros, e continua a impressão para os filhos do nó caso o nó não seja folha. A função chama *printNode* para exibir o conteúdo de cada nó, e segue para os filhos (ponteiros) do nó, caso o nó seja interno, ou simplesmente termina caso seja folha.

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:

- **Cenário 1:** Inserção de vários nós

Entrada: {15, 20, 5, 30, 25, 10, 35, 1, 40, 50}

Esperado: A árvore deve ser impressa por nível com os nós organizados de forma balanceada, respeitando a ordem de inserção e propriedades da B+.

Saída

```
Case 1:

Inserting 15 into the tree...

Tree currently:
00D28568 -> node: 15, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
--------------------------------------------------------------------------------

Inserting 20 into the tree...

Tree currently:
00D28568 -> node: 15, 20 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
--------------------------------------------------------------------------------

Inserting 5 into the tree...

Tree currently:
00D29F48 -> node: 15, -1 | ptr1 = 00D28568, ptr2 = 00D29F98, ptr3 = 00000000
00D28568 -> node: 5, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00D29F98 -> node: 15, 20 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
--------------------------------------------------------------------------------

Inserting 30 into the tree...

Tree currently:
00D29F48 -> node: 15, 20 | ptr1 = 00D28568, ptr2 = 00D29F98, ptr3 = 00D2E7E0
00D28568 -> node: 5, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00D29F98 -> node: 15, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00D2E7E0 -> node: 20, 30 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
--------------------------------------------------------------------------------

Inserting 25 into the tree...

Tree currently:
00D2E8D0 -> node: 20, -1 | ptr1 = 00D29F48, ptr2 = 00D2E920, ptr3 = 00000000
00D29F48 -> node: 15, -1 | ptr1 = 00D28568, ptr2 = 00D29F98, ptr3 = 00000000
00D28568 -> node: 5, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00D29F98 -> node: 15, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00D2E920 -> node: 25, -1 | ptr1 = 00D2E7E0, ptr2 = 00D2EB00, ptr3 = 00000000
00D2E7E0 -> node: 20, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00D2EB00 -> node: 25, 30 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
--------------------------------------------------------------------------------
```

```
Inserting 10 into the tree...

Tree currently:
00D2E8D0 -> node: 20, -1 | ptr1 = 00D29F48, ptr2 = 00D2E920, ptr3 = 00000000
00D29F48 -> node: 15, -1 | ptr1 = 00D28568, ptr2 = 00D29F98, ptr3 = 00000000
00D28568 -> node: 5, 10 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00D29F98 -> node: 15, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00D2E920 -> node: 25, -1 | ptr1 = 00D2E7E0, ptr2 = 00D2EB00, ptr3 = 00000000
00D2E7E0 -> node: 20, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00D2EB00 -> node: 25, 30 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
--------------------------------------------------------------------------------

Inserting 35 into the tree...

Tree currently:
00D2E8D0 -> node: 20, -1 | ptr1 = 00D29F48, ptr2 = 00D2E920, ptr3 = 00000000
00D29F48 -> node: 15, -1 | ptr1 = 00D28568, ptr2 = 00D29F98, ptr3 = 00000000
00D28568 -> node: 5, 10 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00D29F98 -> node: 15, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00D2E920 -> node: 25, 30 | ptr1 = 00D2E7E0, ptr2 = 00D2EB00, ptr3 = 00D2E970
00D2E7E0 -> node: 20, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00D2EB00 -> node: 25, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00D2E970 -> node: 30, 35 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
--------------------------------------------------------------------------------

Inserting 1 into the tree...

Tree currently:
00D2E8D0 -> node: 20, -1 | ptr1 = 00D29F48, ptr2 = 00D2E920, ptr3 = 00000000
00D29F48 -> node: 5, 15 | ptr1 = 00D28568, ptr2 = 00D2EB50, ptr3 = 00D29F98
00D28568 -> node: 1, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00D2EB50 -> node: 5, 10 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00D29F98 -> node: 15, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00D2E920 -> node: 25, 30 | ptr1 = 00D2E7E0, ptr2 = 00D2EB00, ptr3 = 00D2E970
00D2E7E0 -> node: 20, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00D2EB00 -> node: 25, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00D2E970 -> node: 30, 35 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
--------------------------------------------------------------------------------
```

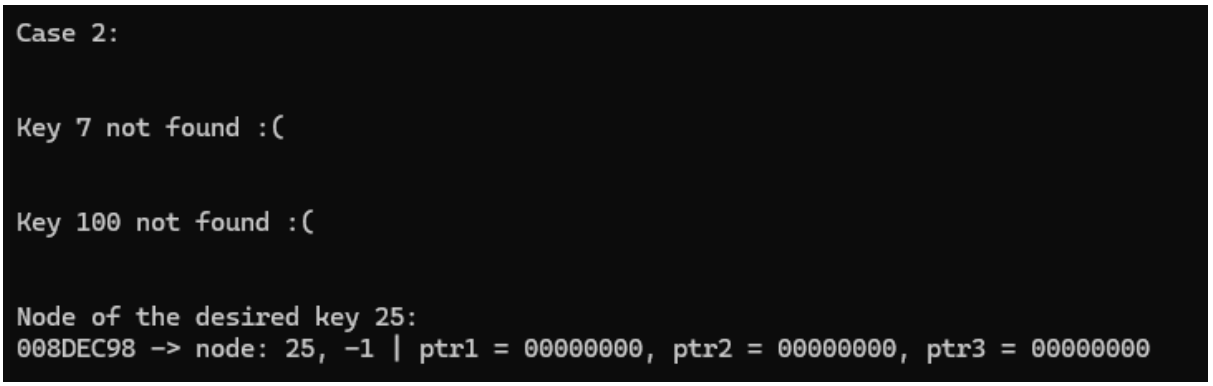
```
Inserting 40 into the tree...

Tree currently:
00D2E8D0 -> node: 20, 30 | ptr1 = 00D29F48, ptr2 = 00D2E920, ptr3 = 00D2E6F0
00D29F48 -> node: 5, 15 | ptr1 = 00D28568, ptr2 = 00D2EB50, ptr3 = 00D29F98
00D28568 -> node: 1, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00D2EB50 -> node: 5, 10 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00D29F98 -> node: 15, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00D2E920 -> node: 25, -1 | ptr1 = 00D2E7E0, ptr2 = 00D2EB00, ptr3 = 00000000
00D2E7E0 -> node: 20, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00D2EB00 -> node: 25, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00D2E6F0 -> node: 35, -1 | ptr1 = 00D2E970, ptr2 = 00D2E9C0, ptr3 = 00000000
00D2E970 -> node: 30, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00D2E9C0 -> node: 35, 40 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
--------------------------------------------------------------------------------

Inserting 50 into the tree...

Tree currently:
00D2E8D0 -> node: 20, 30 | ptr1 = 00D29F48, ptr2 = 00D2E920, ptr3 = 00D2E6F0
00D29F48 -> node: 5, 15 | ptr1 = 00D28568, ptr2 = 00D2EB50, ptr3 = 00D29F98
00D28568 -> node: 1, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00D2EB50 -> node: 5, 10 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00D29F98 -> node: 15, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00D2E920 -> node: 25, -1 | ptr1 = 00D2E7E0, ptr2 = 00D2EB00, ptr3 = 00000000
00D2E7E0 -> node: 20, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00D2EB00 -> node: 25, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00D2E6F0 -> node: 35, 40 | ptr1 = 00D2E970, ptr2 = 00D2E9C0, ptr3 = 00D2E830
00D2E970 -> node: 30, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00D2E9C0 -> node: 35, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00D2E830 -> node: 40, 50 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
--------------------------------------------------------------------------------

Final Tree after insertions:
00D2E8D0 -> node: 20, 30 | ptr1 = 00D29F48, ptr2 = 00D2E920, ptr3 = 00D2E6F0
00D29F48 -> node: 5, 15 | ptr1 = 00D28568, ptr2 = 00D2EB50, ptr3 = 00D29F98
00D28568 -> node: 1, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00D2EB50 -> node: 5, 10 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00D29F98 -> node: 15, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00D2E920 -> node: 25, -1 | ptr1 = 00D2E7E0, ptr2 = 00D2EB00, ptr3 = 00000000
00D2E7E0 -> node: 20, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00D2EB00 -> node: 25, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00D2E6F0 -> node: 35, 40 | ptr1 = 00D2E970, ptr2 = 00D2E9C0, ptr3 = 00D2E830
00D2E970 -> node: 30, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00D2E9C0 -> node: 35, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00D2E830 -> node: 40, 50 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
```

- **Cenário 2:** Busca de valores existentes e inexistentes

Entrada: Chaves para busca: {7, -5, 50}

Esperado: Para 7: O nó contendo o valor deve ser retornado.
Para 100: Deve retornar NULL, indicando que o valor não está na árvore.
Para 25: O nó correto contendo o valor deve ser retornado.

Saída:



- **Cenário 3:** Remoção de nós em ordem aleatória

Entrada: Chaves para remoção: {20, 30, 10, 1}

Esperado: Após cada remoção, a árvore deve ser reestruturada (redistribuição ou concatenação). Nenhuma inconsistência estrutural deve ocorrer.

Saída:

```
Case 3:

Tree before deletion:
00E9E748 -> node: 20, 30 | ptr1 = 00E99F48, ptr2 = 00E9E978, ptr3 = 00E9E9C8
00E99F48 -> node: 5, 15 | ptr1 = 00E98568, ptr2 = 00E9EC98, ptr3 = 00E99F98
00E98568 -> node: 1, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00E9EC98 -> node: 5, 10 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00E99F98 -> node: 15, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00E9E978 -> node: 25, -1 | ptr1 = 00E9E838, ptr2 = 00E9E888, ptr3 = 00000000
00E9E838 -> node: 20, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00E9E888 -> node: 25, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00E9E9C8 -> node: 35, 40 | ptr1 = 00E9EC48, ptr2 = 00E9E8D8, ptr3 = 00E9E7E8
00E9EC48 -> node: 30, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00E9E8D8 -> node: 35, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00E9E7E8 -> node: 40, 50 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000

Tree after deletion of key 20:
00E9E748 -> node: 15, 30 | ptr1 = 00E99F48, ptr2 = 00E9E978, ptr3 = 00E9E9C8
00E99F48 -> node: 5, -1 | ptr1 = 00E98568, ptr2 = 00E9EC98, ptr3 = 00000000
00E98568 -> node: 1, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00E9EC98 -> node: 5, 10 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00E9E978 -> node: 25, -1 | ptr1 = 00E99F98, ptr2 = 00E9E888, ptr3 = 00000000
00E99F98 -> node: 15, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00E9E888 -> node: 25, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00E9E9C8 -> node: 35, 40 | ptr1 = 00E9EC48, ptr2 = 00E9E8D8, ptr3 = 00E9E7E8
00E9EC48 -> node: 30, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00E9E8D8 -> node: 35, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00E9E7E8 -> node: 40, 50 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000

Tree after deletion of key 30:
00E9E748 -> node: 15, 35 | ptr1 = 00E99F48, ptr2 = 00E9E978, ptr3 = 00E9E9C8
00E99F48 -> node: 5, -1 | ptr1 = 00E98568, ptr2 = 00E9EC98, ptr3 = 00000000
00E98568 -> node: 1, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00E9EC98 -> node: 5, 10 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00E9E978 -> node: 25, -1 | ptr1 = 00E99F98, ptr2 = 00E9E888, ptr3 = 00000000
00E99F98 -> node: 15, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00E9E888 -> node: 25, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00E9E9C8 -> node: 40, -1 | ptr1 = 00E9E8D8, ptr2 = 00E9E7E8, ptr3 = 00000000
00E9E8D8 -> node: 35, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00E9E7E8 -> node: 40, 50 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
```

```
Tree after deletion of key 10:
00E9E748 -> node: 15, 35 | ptr1 = 00E99F48, ptr2 = 00E9E978, ptr3 = 00E9E9C8
00E99F48 -> node: 5, -1 | ptr1 = 00E98568, ptr2 = 00E9EC98, ptr3 = 00000000
00E98568 -> node: 1, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00E9EC98 -> node: 5, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00E9E978 -> node: 25, -1 | ptr1 = 00E99F98, ptr2 = 00E9E888, ptr3 = 00000000
00E99F98 -> node: 15, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00E9E888 -> node: 25, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00E9E9C8 -> node: 40, -1 | ptr1 = 00E9E8D8, ptr2 = 00E9E7E8, ptr3 = 00000000
00E9E8D8 -> node: 35, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00E9E7E8 -> node: 40, 50 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000

Tree after deletion of key 1:
00E9E748 -> node: 35, -1 | ptr1 = 00E99F48, ptr2 = 00E9E9C8, ptr3 = 00000000
00E99F48 -> node: 15, 25 | ptr1 = 00E9EC98, ptr2 = 00E99F98, ptr3 = 00E9E888
00E9EC98 -> node: 5, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00E99F98 -> node: 15, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00E9E888 -> node: 25, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00E9E9C8 -> node: 40, -1 | ptr1 = 00E9E8D8, ptr2 = 00E9E7E8, ptr3 = 00000000
00E9E8D8 -> node: 35, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00E9E7E8 -> node: 40, 50 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
```

- **Cenário 4:** Remoção de chaves inexistentes

Entrada: Chaves para remoção: {100, -5}

Esperado: Nenhuma alteração na árvore. Mensagem indicando que as chaves não foram encontradas.

Saída:

```
Case 4:

Attempting to delete key 100 that is not in the tree:
Key not found :(

Attempting to delete key -5 that is not in the tree:
Key not found :(

Final Tree after deletions:
00ACE6A8 -> node: 35, -1 | ptr1 = 00AC9F48, ptr2 = 00ACEBF8, ptr3 = 00000000
00AC9F48 -> node: 15, 25 | ptr1 = 00ACEC48, ptr2 = 00AC9F98, ptr3 = 00ACE8D8
00ACEC48 -> node: 5, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00AC9F98 -> node: 15, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00ACE8D8 -> node: 25, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00ACEBF8 -> node: 40, -1 | ptr1 = 00ACED88, ptr2 = 00ACE888, ptr3 = 00000000
00ACED88 -> node: 35, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00ACE888 -> node: 40, 50 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
```

- **Cenário 5:** Inserção de nós que causam cisão de nó

Entrada: {15, 20, 5, 30, 25, 10, 35, 1, 40, 50, 60, 55}

Esperado: Durante a inserção, a árvore deve causar cisões de nó quando o nó da folha atingir o limite, fazendo a árvore se reorganizar e balancear conforme as propriedades de uma árvore B+. A saída deve mostrar a árvore sendo reorganizada após cada inserção.

Saída:

```
Case 5:
Inserting 15 into the tree...

Tree currently:
00ACEAB8 -> node: 15, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
--------------------------------------------------------------------------------
Inserting 20 into the tree...

Tree currently:
00ACEAB8 -> node: 15, 20 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
--------------------------------------------------------------------------------
Inserting 5 into the tree...

Tree currently:
00ACEA68 -> node: 15, -1 | ptr1 = 00ACEAB8, ptr2 = 00ACE608, ptr3 = 00000000
00ACEAB8 -> node: 5, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00ACE608 -> node: 15, 20 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
--------------------------------------------------------------------------------
Inserting 30 into the tree...

Tree currently:
00ACEA68 -> node: 15, 20 | ptr1 = 00ACEAB8, ptr2 = 00ACE608, ptr3 = 00ACEC98
00ACEAB8 -> node: 5, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00ACE608 -> node: 15, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00ACEC98 -> node: 20, 30 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
--------------------------------------------------------------------------------
Inserting 25 into the tree...

Tree currently:
00ACEB58 -> node: 20, -1 | ptr1 = 00ACEA68, ptr2 = 00ACECE8, ptr3 = 00000000
00ACEA68 -> node: 15, -1 | ptr1 = 00ACEAB8, ptr2 = 00ACE608, ptr3 = 00000000
00ACEAB8 -> node: 5, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00ACE608 -> node: 15, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00ACECE8 -> node: 25, -1 | ptr1 = 00ACEC98, ptr2 = 00ACE928, ptr3 = 00000000
00ACEC98 -> node: 20, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00ACE928 -> node: 25, 30 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
--------------------------------------------------------------------------------
Inserting 10 into the tree...

Tree currently:
00ACEB58 -> node: 20, -1 | ptr1 = 00ACEA68, ptr2 = 00ACECE8, ptr3 = 00000000
00ACEA68 -> node: 15, -1 | ptr1 = 00ACEAB8, ptr2 = 00ACE608, ptr3 = 00000000
00ACEAB8 -> node: 5, 10 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00ACE608 -> node: 15, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00ACECE8 -> node: 25, -1 | ptr1 = 00ACEC98, ptr2 = 00ACE928, ptr3 = 00000000
00ACEC98 -> node: 20, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00ACE928 -> node: 25, 30 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
--------------------------------------------------------------------------------
```

```
Inserting 35 into the tree...

Tree currently:
00ACEB58 -> node: 20, -1 | ptr1 = 00ACEA68, ptr2 = 00ACECE8, ptr3 = 00000000
00ACEA68 -> node: 15, -1 | ptr1 = 00ACEAB8, ptr2 = 00ACE608, ptr3 = 00000000
00ACEAB8 -> node: 5, 10 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00ACE608 -> node: 15, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00ACECE8 -> node: 25, 30 | ptr1 = 00ACEC98, ptr2 = 00ACE928, ptr3 = 00ACE978
00ACEC98 -> node: 20, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00ACE928 -> node: 25, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00ACE978 -> node: 30, 35 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
--------------------------------------------------------------------------------
Inserting 1 into the tree...

Tree currently:
00ACEB58 -> node: 20, -1 | ptr1 = 00ACEA68, ptr2 = 00ACECE8, ptr3 = 00000000
00ACEA68 -> node: 5, 15 | ptr1 = 00ACEAB8, ptr2 = 00ACED38, ptr3 = 00ACE608
00ACEAB8 -> node: 1, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00ACED38 -> node: 5, 10 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00ACE608 -> node: 15, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00ACECE8 -> node: 25, 30 | ptr1 = 00ACEC98, ptr2 = 00ACE928, ptr3 = 00ACE978
00ACEC98 -> node: 20, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00ACE928 -> node: 25, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00ACE978 -> node: 30, 35 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
--------------------------------------------------------------------------------
Inserting 40 into the tree...

Tree currently:
00ACEB58 -> node: 20, 30 | ptr1 = 00ACEA68, ptr2 = 00ACECE8, ptr3 = 00ACE658
00ACEA68 -> node: 5, 15 | ptr1 = 00ACEAB8, ptr2 = 00ACED38, ptr3 = 00ACE608
00ACEAB8 -> node: 1, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00ACED38 -> node: 5, 10 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00ACE608 -> node: 15, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00ACECE8 -> node: 25, -1 | ptr1 = 00ACEC98, ptr2 = 00ACE928, ptr3 = 00000000
00ACEC98 -> node: 20, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00ACE928 -> node: 25, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00ACE658 -> node: 35, -1 | ptr1 = 00ACE978, ptr2 = 00ACE9C8, ptr3 = 00000000
00ACE978 -> node: 30, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00ACE9C8 -> node: 35, 40 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
--------------------------------------------------------------------------------
```

```
--------------------------------------------------------------------------------
Inserting 50 into the tree...

Tree currently:
00ACEB58 -> node: 20, 30 | ptr1 = 00ACEA68, ptr2 = 00ACECE8, ptr3 = 00ACE658
00ACEA68 -> node: 5, 15 | ptr1 = 00ACEAB8, ptr2 = 00ACED38, ptr3 = 00ACE608
00ACEAB8 -> node: 1, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00ACED38 -> node: 5, 10 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00ACE608 -> node: 15, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00ACECE8 -> node: 25, -1 | ptr1 = 00ACEC98, ptr2 = 00ACE928, ptr3 = 00000000
00ACEC98 -> node: 20, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00ACE928 -> node: 25, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00ACE658 -> node: 35, 40 | ptr1 = 00ACE978, ptr2 = 00ACE9C8, ptr3 = 00ACE6F8
00ACE978 -> node: 30, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00ACE9C8 -> node: 35, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00ACE6F8 -> node: 40, 50 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
--------------------------------------------------------------------------------
Inserting 60 into the tree...

Tree currently:
00ACEA18 -> node: 30, -1 | ptr1 = 00ACEB58, ptr2 = 00ACEB08, ptr3 = 00000000
00ACEB58 -> node: 20, -1 | ptr1 = 00ACEA68, ptr2 = 00ACECE8, ptr3 = 00000000
00ACEA68 -> node: 5, 15 | ptr1 = 00ACEAB8, ptr2 = 00ACED38, ptr3 = 00ACE608
00ACEAB8 -> node: 1, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00ACED38 -> node: 5, 10 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00ACE608 -> node: 15, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00ACECE8 -> node: 25, -1 | ptr1 = 00ACEC98, ptr2 = 00ACE928, ptr3 = 00000000
00ACEC98 -> node: 20, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00ACE928 -> node: 25, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00ACEB08 -> node: 40, -1 | ptr1 = 00ACE658, ptr2 = 00ACE798, ptr3 = 00000000
00ACE658 -> node: 35, -1 | ptr1 = 00ACE978, ptr2 = 00ACE9C8, ptr3 = 00000000
00ACE978 -> node: 30, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00ACE9C8 -> node: 35, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00ACE798 -> node: 50, -1 | ptr1 = 00ACE6F8, ptr2 = 00ACE748, ptr3 = 00000000
00ACE6F8 -> node: 40, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00ACE748 -> node: 50, 60 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
--------------------------------------------------------------------------------
```

```
--------------------------------------------------------------------------------
Inserting 55 into the tree...

Tree currently:
00ACEA18 -> node: 30, -1 | ptr1 = 00ACEB58, ptr2 = 00ACEB08, ptr3 = 00000000
00ACEB58 -> node: 20, -1 | ptr1 = 00ACEA68, ptr2 = 00ACECE8, ptr3 = 00000000
00ACEA68 -> node: 5, 15 | ptr1 = 00ACEAB8, ptr2 = 00ACED38, ptr3 = 00ACE608
00ACEAB8 -> node: 1, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00ACED38 -> node: 5, 10 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00ACE608 -> node: 15, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00ACECE8 -> node: 25, -1 | ptr1 = 00ACEC98, ptr2 = 00ACE928, ptr3 = 00000000
00ACEC98 -> node: 20, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00ACE928 -> node: 25, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00ACEB08 -> node: 40, -1 | ptr1 = 00ACE658, ptr2 = 00ACE798, ptr3 = 00000000
00ACE658 -> node: 35, -1 | ptr1 = 00ACE978, ptr2 = 00ACE9C8, ptr3 = 00000000
00ACE978 -> node: 30, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00ACE9C8 -> node: 35, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00ACE798 -> node: 50, 55 | ptr1 = 00ACE6F8, ptr2 = 00ACE748, ptr3 = 00ACEBA8
00ACE6F8 -> node: 40, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00ACE748 -> node: 50, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00ACEBA8 -> node: 55, 60 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
--------------------------------------------------------------------------------

Final Tree after insertions:
00ACEA18 -> node: 30, -1 | ptr1 = 00ACEB58, ptr2 = 00ACEB08, ptr3 = 00000000
00ACEB58 -> node: 20, -1 | ptr1 = 00ACEA68, ptr2 = 00ACECE8, ptr3 = 00000000
00ACEA68 -> node: 5, 15 | ptr1 = 00ACEAB8, ptr2 = 00ACED38, ptr3 = 00ACE608
00ACEAB8 -> node: 1, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00ACED38 -> node: 5, 10 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00ACE608 -> node: 15, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00ACECE8 -> node: 25, -1 | ptr1 = 00ACEC98, ptr2 = 00ACE928, ptr3 = 00000000
00ACEC98 -> node: 20, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00ACE928 -> node: 25, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00ACEB08 -> node: 40, -1 | ptr1 = 00ACE658, ptr2 = 00ACE798, ptr3 = 00000000
00ACE658 -> node: 35, -1 | ptr1 = 00ACE978, ptr2 = 00ACE9C8, ptr3 = 00000000
00ACE978 -> node: 30, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00ACE9C8 -> node: 35, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00ACE798 -> node: 50, 55 | ptr1 = 00ACE6F8, ptr2 = 00ACE748, ptr3 = 00ACEBA8
00ACE6F8 -> node: 40, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00ACE748 -> node: 50, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00ACEBA8 -> node: 55, 60 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
```

- **Cenário 6:** Inserção, busca e remoção de nós

Entrada: Inserção: {15, 20, 5, 30, 25, 10, 35, 1, 40, 50, 60, 55}
Busca: Chaves para busca: {25, 100}
Remoções: Chaves para remoção: {20, 5, 40, 100, -5}

Esperado:
- A árvore deve ser inserida com as chaves e balanceada.
- Busca por valores existentes (25) deve retornar o nó com a chave.
- Busca por valores inexistentes (100) deve retornar NULL.
- Após as remoções de valores existentes, a árvore deve ser reestruturada corretamente.

- Remoções de valores inexistentes (100 e -5) devem resultar em uma mensagem indicando que as chaves não foram encontradas, sem alterar a árvore.

Saída:

```
Case 6:
Inserting 15 into the tree...

Tree currently:
00C81230 -> node: 15, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
--------------------------------------------------------------------------------
Inserting 20 into the tree...

Tree currently:
00C81230 -> node: 15, 20 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
--------------------------------------------------------------------------------
Inserting 5 into the tree...

Tree currently:
00C81190 -> node: 15, -1 | ptr1 = 00C81230, ptr2 = 00C81C70, ptr3 = 00000000
00C81230 -> node: 5, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00C81C70 -> node: 15, 20 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
--------------------------------------------------------------------------------
Inserting 30 into the tree...

Tree currently:
00C81190 -> node: 15, 20 | ptr1 = 00C81230, ptr2 = 00C81C70, ptr3 = 00C81EF0
00C81230 -> node: 5, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00C81C70 -> node: 15, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00C81EF0 -> node: 20, 30 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
--------------------------------------------------------------------------------
Inserting 25 into the tree...

Tree currently:
00C81D60 -> node: 20, -1 | ptr1 = 00C81190, ptr2 = 00C81950, ptr3 = 00000000
00C81190 -> node: 15, -1 | ptr1 = 00C81230, ptr2 = 00C81C70, ptr3 = 00000000
00C81230 -> node: 5, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00C81C70 -> node: 15, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00C81950 -> node: 25, -1 | ptr1 = 00C81EF0, ptr2 = 00C81BD0, ptr3 = 00000000
00C81EF0 -> node: 20, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00C81BD0 -> node: 25, 30 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
--------------------------------------------------------------------------------
Inserting 10 into the tree...

Tree currently:
00C81D60 -> node: 20, -1 | ptr1 = 00C81190, ptr2 = 00C81950, ptr3 = 00000000
00C81190 -> node: 15, -1 | ptr1 = 00C81230, ptr2 = 00C81C70, ptr3 = 00000000
00C81230 -> node: 5, 10 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00C81C70 -> node: 15, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00C81950 -> node: 25, -1 | ptr1 = 00C81EF0, ptr2 = 00C81BD0, ptr3 = 00000000
00C81EF0 -> node: 20, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00C81BD0 -> node: 25, 30 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
--------------------------------------------------------------------------------
```

```
Inserting 35 into the tree...

Tree currently:
00C81D60 -> node: 20, -1 | ptr1 = 00C81190, ptr2 = 00C81950, ptr3 = 00000000
00C81190 -> node: 15, -1 | ptr1 = 00C81230, ptr2 = 00C81C70, ptr3 = 00000000
00C81230 -> node: 5, 10 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00C81C70 -> node: 15, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00C81950 -> node: 25, 30 | ptr1 = 00C81EF0, ptr2 = 00C81BD0, ptr3 = 00C818B0
00C81EF0 -> node: 20, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00C81BD0 -> node: 25, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00C818B0 -> node: 30, 35 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
--------------------------------------------------------------------------------
Inserting 1 into the tree...

Tree currently:
00C81D60 -> node: 20, -1 | ptr1 = 00C81190, ptr2 = 00C81950, ptr3 = 00000000
00C81190 -> node: 5, 15 | ptr1 = 00C81230, ptr2 = 00C81FE0, ptr3 = 00C81C70
00C81230 -> node: 1, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00C81FE0 -> node: 5, 10 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00C81C70 -> node: 15, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00C81950 -> node: 25, 30 | ptr1 = 00C81EF0, ptr2 = 00C81BD0, ptr3 = 00C818B0
00C81EF0 -> node: 20, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00C81BD0 -> node: 25, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00C818B0 -> node: 30, 35 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
--------------------------------------------------------------------------------
Inserting 40 into the tree...

Tree currently:
00C81D60 -> node: 20, 30 | ptr1 = 00C81190, ptr2 = 00C81950, ptr3 = 00C82170
00C81190 -> node: 5, 15 | ptr1 = 00C81230, ptr2 = 00C81FE0, ptr3 = 00C81C70
00C81230 -> node: 1, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00C81FE0 -> node: 5, 10 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00C81C70 -> node: 15, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00C81950 -> node: 25, -1 | ptr1 = 00C81EF0, ptr2 = 00C81BD0, ptr3 = 00000000
00C81EF0 -> node: 20, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00C81BD0 -> node: 25, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00C82170 -> node: 35, -1 | ptr1 = 00C818B0, ptr2 = 00C81900, ptr3 = 00000000
00C818B0 -> node: 30, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00C81900 -> node: 35, 40 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
--------------------------------------------------------------------------------
```

```
Inserting 50 into the tree...

Tree currently:
00C81D60 -> node: 20, 30 | ptr1 = 00C81190, ptr2 = 00C81950, ptr3 = 00C82170
00C81190 -> node: 5, 15 | ptr1 = 00C81230, ptr2 = 00C81FE0, ptr3 = 00C81C70
00C81230 -> node: 1, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00C81FE0 -> node: 5, 10 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00C81C70 -> node: 15, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00C81950 -> node: 25, -1 | ptr1 = 00C81EF0, ptr2 = 00C81BD0, ptr3 = 00000000
00C81EF0 -> node: 20, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00C81BD0 -> node: 25, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00C82170 -> node: 35, 40 | ptr1 = 00C818B0, ptr2 = 00C81900, ptr3 = 00C81C20
00C818B0 -> node: 30, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00C81900 -> node: 35, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00C81C20 -> node: 40, 50 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
--------------------------------------------------------------------------------
Inserting 60 into the tree...

Tree currently:
00C81B80 -> node: 30, -1 | ptr1 = 00C81D60, ptr2 = 00C81A90, ptr3 = 00000000
00C81D60 -> node: 20, -1 | ptr1 = 00C81190, ptr2 = 00C81950, ptr3 = 00000000
00C81190 -> node: 5, 15 | ptr1 = 00C81230, ptr2 = 00C81FE0, ptr3 = 00C81C70
00C81230 -> node: 1, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00C81FE0 -> node: 5, 10 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00C81C70 -> node: 15, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00C81950 -> node: 25, -1 | ptr1 = 00C81EF0, ptr2 = 00C81BD0, ptr3 = 00000000
00C81EF0 -> node: 20, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00C81BD0 -> node: 25, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00C81A90 -> node: 40, -1 | ptr1 = 00C82170, ptr2 = 00C81A40, ptr3 = 00000000
00C82170 -> node: 35, -1 | ptr1 = 00C818B0, ptr2 = 00C81900, ptr3 = 00000000
00C818B0 -> node: 30, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00C81900 -> node: 35, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00C81A40 -> node: 50, -1 | ptr1 = 00C81C20, ptr2 = 00C82080, ptr3 = 00000000
00C81C20 -> node: 40, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00C82080 -> node: 50, 60 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
--------------------------------------------------------------------------------
```

```
Inserting 55 into the tree...

Tree currently:
00C81B80 -> node: 30, -1 | ptr1 = 00C81D60, ptr2 = 00C81A90, ptr3 = 00000000
00C81D60 -> node: 20, -1 | ptr1 = 00C81190, ptr2 = 00C81950, ptr3 = 00000000
00C81190 -> node: 5, 15 | ptr1 = 00C81230, ptr2 = 00C81FE0, ptr3 = 00C81C70
00C81230 -> node: 1, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00C81FE0 -> node: 5, 10 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00C81C70 -> node: 15, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00C81950 -> node: 25, -1 | ptr1 = 00C81EF0, ptr2 = 00C81BD0, ptr3 = 00000000
00C81EF0 -> node: 20, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00C81BD0 -> node: 25, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00C81A90 -> node: 40, -1 | ptr1 = 00C82170, ptr2 = 00C81A40, ptr3 = 00000000
00C82170 -> node: 35, -1 | ptr1 = 00C818B0, ptr2 = 00C81900, ptr3 = 00000000
00C818B0 -> node: 30, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00C81900 -> node: 35, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00C81A40 -> node: 50, 55 | ptr1 = 00C81C20, ptr2 = 00C82080, ptr3 = 00C820D0
00C81C20 -> node: 40, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00C82080 -> node: 50, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00C820D0 -> node: 55, 60 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
--------------------------------------------------------------------------------

Final Tree after insertions:
00C81B80 -> node: 30, -1 | ptr1 = 00C81D60, ptr2 = 00C81A90, ptr3 = 00000000
00C81D60 -> node: 20, -1 | ptr1 = 00C81190, ptr2 = 00C81950, ptr3 = 00000000
00C81190 -> node: 5, 15 | ptr1 = 00C81230, ptr2 = 00C81FE0, ptr3 = 00C81C70
00C81230 -> node: 1, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00C81FE0 -> node: 5, 10 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00C81C70 -> node: 15, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00C81950 -> node: 25, -1 | ptr1 = 00C81EF0, ptr2 = 00C81BD0, ptr3 = 00000000
00C81EF0 -> node: 20, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00C81BD0 -> node: 25, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00C81A90 -> node: 40, -1 | ptr1 = 00C82170, ptr2 = 00C81A40, ptr3 = 00000000
00C82170 -> node: 35, -1 | ptr1 = 00C818B0, ptr2 = 00C81900, ptr3 = 00000000
00C818B0 -> node: 30, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00C81900 -> node: 35, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00C81A40 -> node: 50, 55 | ptr1 = 00C81C20, ptr2 = 00C82080, ptr3 = 00C820D0
00C81C20 -> node: 40, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00C82080 -> node: 50, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00C820D0 -> node: 55, 60 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000


Node of the desired key 25:
00C81BD0 -> node: 25, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000


Key 100 not found :(
```

```
Tree before deletion:
00C81B80 -> node: 30, -1 | ptr1 = 00C81D60, ptr2 = 00C81A90, ptr3 = 00000000
00C81D60 -> node: 20, -1 | ptr1 = 00C81190, ptr2 = 00C81950, ptr3 = 00000000
00C81190 -> node: 5, 15 | ptr1 = 00C81230, ptr2 = 00C81FE0, ptr3 = 00C81C70
00C81230 -> node: 1, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00C81FE0 -> node: 5, 10 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00C81C70 -> node: 15, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00C81950 -> node: 25, -1 | ptr1 = 00C81EF0, ptr2 = 00C81BD0, ptr3 = 00000000
00C81EF0 -> node: 20, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00C81BD0 -> node: 25, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00C81A90 -> node: 40, -1 | ptr1 = 00C82170, ptr2 = 00C81A40, ptr3 = 00000000
00C82170 -> node: 35, -1 | ptr1 = 00C818B0, ptr2 = 00C81900, ptr3 = 00000000
00C818B0 -> node: 30, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00C81900 -> node: 35, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00C81A40 -> node: 50, 55 | ptr1 = 00C81C20, ptr2 = 00C82080, ptr3 = 00C820D0
00C81C20 -> node: 40, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00C82080 -> node: 50, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00C820D0 -> node: 55, 60 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000

Tree after deletion of key 20:
00C81B80 -> node: 30, -1 | ptr1 = 00C81D60, ptr2 = 00C81A90, ptr3 = 00000000
00C81D60 -> node: 15, -1 | ptr1 = 00C81190, ptr2 = 00C81950, ptr3 = 00000000
00C81190 -> node: 5, -1 | ptr1 = 00C81230, ptr2 = 00C81FE0, ptr3 = 00000000
00C81230 -> node: 1, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00C81FE0 -> node: 5, 10 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00C81950 -> node: 25, -1 | ptr1 = 00C81C70, ptr2 = 00C81BD0, ptr3 = 00000000
00C81C70 -> node: 15, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00C81BD0 -> node: 25, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00C81A90 -> node: 40, -1 | ptr1 = 00C82170, ptr2 = 00C81A40, ptr3 = 00000000
00C82170 -> node: 35, -1 | ptr1 = 00C818B0, ptr2 = 00C81900, ptr3 = 00000000
00C818B0 -> node: 30, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00C81900 -> node: 35, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00C81A40 -> node: 50, 55 | ptr1 = 00C81C20, ptr2 = 00C82080, ptr3 = 00C820D0
00C81C20 -> node: 40, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00C82080 -> node: 50, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00C820D0 -> node: 55, 60 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
```

```
Tree after deletion of key 5:
00C81B80 -> node: 30, -1 | ptr1 = 00C81D60, ptr2 = 00C81A90, ptr3 = 00000000
00C81D60 -> node: 15, -1 | ptr1 = 00C81190, ptr2 = 00C81950, ptr3 = 00000000
00C81190 -> node: 10, -1 | ptr1 = 00C81230, ptr2 = 00C81FE0, ptr3 = 00000000
00C81230 -> node: 1, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00C81FE0 -> node: 10, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00C81950 -> node: 25, -1 | ptr1 = 00C81C70, ptr2 = 00C81BD0, ptr3 = 00000000
00C81C70 -> node: 15, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00C81BD0 -> node: 25, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00C81A90 -> node: 40, -1 | ptr1 = 00C82170, ptr2 = 00C81A40, ptr3 = 00000000
00C82170 -> node: 35, -1 | ptr1 = 00C818B0, ptr2 = 00C81900, ptr3 = 00000000
00C818B0 -> node: 30, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00C81900 -> node: 35, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00C81A40 -> node: 50, 55 | ptr1 = 00C81C20, ptr2 = 00C82080, ptr3 = 00C820D0
00C81C20 -> node: 40, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00C82080 -> node: 50, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00C820D0 -> node: 55, 60 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000

Tree after deletion of key 40:
00C81B80 -> node: 30, -1 | ptr1 = 00C81D60, ptr2 = 00C81A90, ptr3 = 00000000
00C81D60 -> node: 15, -1 | ptr1 = 00C81190, ptr2 = 00C81950, ptr3 = 00000000
00C81190 -> node: 10, -1 | ptr1 = 00C81230, ptr2 = 00C81FE0, ptr3 = 00000000
00C81230 -> node: 1, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00C81FE0 -> node: 10, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00C81950 -> node: 25, -1 | ptr1 = 00C81C70, ptr2 = 00C81BD0, ptr3 = 00000000
00C81C70 -> node: 15, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00C81BD0 -> node: 25, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00C81A90 -> node: 50, -1 | ptr1 = 00C82170, ptr2 = 00C81A40, ptr3 = 00000000
00C82170 -> node: 35, -1 | ptr1 = 00C818B0, ptr2 = 00C81900, ptr3 = 00000000
00C818B0 -> node: 30, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00C81900 -> node: 35, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00C81A40 -> node: 55, -1 | ptr1 = 00C82080, ptr2 = 00C820D0, ptr3 = 00000000
00C82080 -> node: 50, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00C820D0 -> node: 55, 60 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000

Attempting to delete key 100 that is not in the tree:
Key not found :(

Attempting to delete key -5 that is not in the tree:
Key not found :(
```

```
Final Tree after deletions:
00C81B80 -> node: 30, -1 | ptr1 = 00C81D60, ptr2 = 00C81A90, ptr3 = 00000000
00C81D60 -> node: 15, -1 | ptr1 = 00C81190, ptr2 = 00C81950, ptr3 = 00000000
00C81190 -> node: 10, -1 | ptr1 = 00C81230, ptr2 = 00C81FE0, ptr3 = 00000000
00C81230 -> node: 1, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00C81FE0 -> node: 10, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00C81950 -> node: 25, -1 | ptr1 = 00C81C70, ptr2 = 00C81BD0, ptr3 = 00000000
00C81C70 -> node: 15, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00C81BD0 -> node: 25, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00C81A90 -> node: 50, -1 | ptr1 = 00C82170, ptr2 = 00C81A40, ptr3 = 00000000
00C82170 -> node: 35, -1 | ptr1 = 00C818B0, ptr2 = 00C81900, ptr3 = 00000000
00C818B0 -> node: 30, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00C81900 -> node: 35, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00C81A40 -> node: 55, -1 | ptr1 = 00C82080, ptr2 = 00C820D0, ptr3 = 00000000
00C82080 -> node: 50, -1 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
00C820D0 -> node: 55, 60 | ptr1 = 00000000, ptr2 = 00000000, ptr3 = 00000000
```

4. **Observações e conclusões**

Durante a implementação do código, alguns pontos importantes se destacaram:

- **Facilidades:** Utilizar o site de visualização de árvores B+ disponibilizado no EAD foi muito útil para checar se a árvore estava sendo montada corretamente. Ele permitiu validar visualmente a estrutura da árvore e detectar rapidamente quaisquer erros no processo de inserção ou remoção de nós.

- **Dificuldades:** Um dos maiores desafios foi gerenciar corretamente os ponteiros entre os nós, especialmente durante as operações de cisão e remoção. Cada nó possui múltiplos ponteiros (para filhos e para o pai), e a manipulação incorreta desses ponteiros pode levar a vazamentos de memória ou corrupções de dados. As operações de remoção e redistribuição de nós exigiram uma compreensão profunda das regras de balanceamento da árvore B+, o que demandou um esforço extra para garantir que a árvore fosse reestruturada corretamente após cada remoção.

- **Conclusão**:

A implementação da árvore B+ foi bem-sucedida, e todas as operações solicitadas (inserção, busca e remoção) foram corretamente implementadas. O processo de reestruturação da árvore, que inclui cisões e redistribuições, foi uma parte essencial para garantir que a árvore permanecesse balanceada, o que é um dos principais requisitos de uma árvore B+. A experiência também evidenciou a importância de uma boa gestão de ponteiros e da estrutura de dados subjacente. A utilização de testes foi crucial para validar o comportamento da árvore e corrigir possíveis erros, como o manejo inadequado dos ponteiros ou falhas na cisão de nós. Com a implementação bem-sucedida, podemos concluir que a árvore B+ é uma estrutura eficiente e adequada para gerenciar grandes volumes de dados de forma balanceada, garantindo boa performance nas operações de busca e inserção.