1. Criação da Estrutura e Criação de Novo Nó

- Linha de destaque: `struct No* criarNo(int dado)`
- A função `criarNo` cria uma estrutura de nó e define os ponteiros de `esquerda` e `direita` como `NULL`.

Diagrama de sequência

```
sequenceDiagram
    main -> criarNo: chama criarNo(dado)
    criarNo -> malloc: aloca espaço para o nó
    criarNo -> criarNo: inicializa dados e ponteiros
    criarNo -> main: retorna o novo nó
```

Fluxograma de `criarNo`

```
graph TD
  Start["Início"]
  A["Aloca espaço na memória"]
  B["Inicializa dado e ponteiros"]
  C["Retorna o novo nó"]
  Start --> A --> B --> C
```

2. Inserir Nó

- Linha de destaque: `struct No* inserir(struct No* raiz, int dado)`
- A função `inserir` insere um novo valor na árvore, posicionando-o de acordo com seu valor.

Diagrama de sequência

```
sequenceDiagram
    main -> inserir: chama inserir(raiz, dado)
    inserir -> criarNo: chama criarNo caso raiz seja NULL
    inserir -> inserir: decide ir para esquerda ou direita
    inserir -> main: retorna a raiz atualizada
```

Fluxograma de `inserir`

```
graph TD
  A["Início"]
  B["Raiz é NULL?"]
  C["Cria novo nó"]
  D["Dado < raiz->dado"]
  E["Vai para a esquerda"]
  F["Vai para a direita"]
  G["Retorna a raiz atualizada"]

  A --> B
  B -- Sim --> C --> G
  B -- Não --> D
  D -- Sim --> E --> G
  D -- Não --> F --> G
```

3. Buscar Nó

- Linha de destaque: `struct No* buscar(struct No* raiz, int dado)`
- A função `buscar` localiza um nó com o valor especificado na árvore.

Diagrama de sequência

```
sequenceDiagram
    main -> buscar: chama buscar(raiz, dado)
    buscar -> buscar: verifica valor do dado
    buscar -> main: retorna o nó encontrado ou NULL
```

Fluxograma de `buscar`

```
graph TD
  A["Início"]
  B["Raiz é NULL ou raiz->dado == dado?"]
  C["Verifica lado esquerdo"]
  D["Verifica lado direito"]
  E["Retorna o nó"]

  A --> B
```

```
  B -- Sim --> E
  B -- Não --> C --> D --> E
```

4. Excluir Nó

- Linha de destaque: `struct No* removerNo(struct No* raiz, int dado)`
- A função `removerNo` remove um nó específico da árvore, ajustando a árvore conforme necessário.

Diagrama de sequência

```
sequenceDiagram
    main -> removerNo: chama removerNo(raiz, dado)
    removerNo -> removerNo: localiza o nó
    removerNo -> encontrarMinimo: encontra substituto
    removerNo -> free: libera o nó
    removerNo -> main: retorna raiz ajustada
```

Fluxograma de `removerNo`

```
graph TD
  A["Início"]
  B["Raiz é NULL?"]
  C["Dado < raiz->dado"]
  D["Remover do lado esquerdo"]
  E["Remover do lado direito"]
  F["Substitui pelo menor da direita"]
  G["Libera memória do nó"]
  H["Retorna raiz ajustada"]

  A --> B
  B -- Sim --> H
  B -- Não --> C
  C -- Sim --> D --> H
  C -- Não --> E --> F --> G --> H
```

5. Imprimir Árvore em Ordem

- Linha de destaque: `void emOrdem(struct No* raiz)`
- A função `emOrdem` percorre a árvore em ordem crescente.

Diagrama de sequência

```
sequenceDiagram
    main -> emOrdem: chama emOrdem(raiz)
    emOrdem -> emOrdem: recursão esquerda
    emOrdem -> printf: imprime dado
    emOrdem -> emOrdem: recursão direita
    emOrdem -> main: retorna
```

Fluxograma de `emOrdem`

```
graph TD
  A["Início"]
  B["Raiz é NULL?"]
  C["Percorre à esquerda"]
  D["Imprime dado"]
  E["Percorre à direita"]
  F["Retorna"]

  A --> B
  B -- Não --> C --> D --> E --> F
  B -- Sim --> F
```