FIGURA 6.23 Implementação do algoritmo de inserção. (*continuação*)

```
    BSTNode<T> *p = root, *prev = 0;
    while (p != 0) {  // encontre um no para inserir um novo no;
        prev = p;
        if (el < p->el)
            p = p->left;
        else p = p->right;
    }
    if (root == 0)     // a arvore esta vazia;
        root = new BSTNode<T>(el);
    else if (el < prev->el)
        prev->left  = new BSTNode<T>(el);
    else prev->right  = new BSTNode<T>(el);
}
```

FIGURA 6.24 Implementação do algoritmo para inserir nó em uma árvore alinhada.

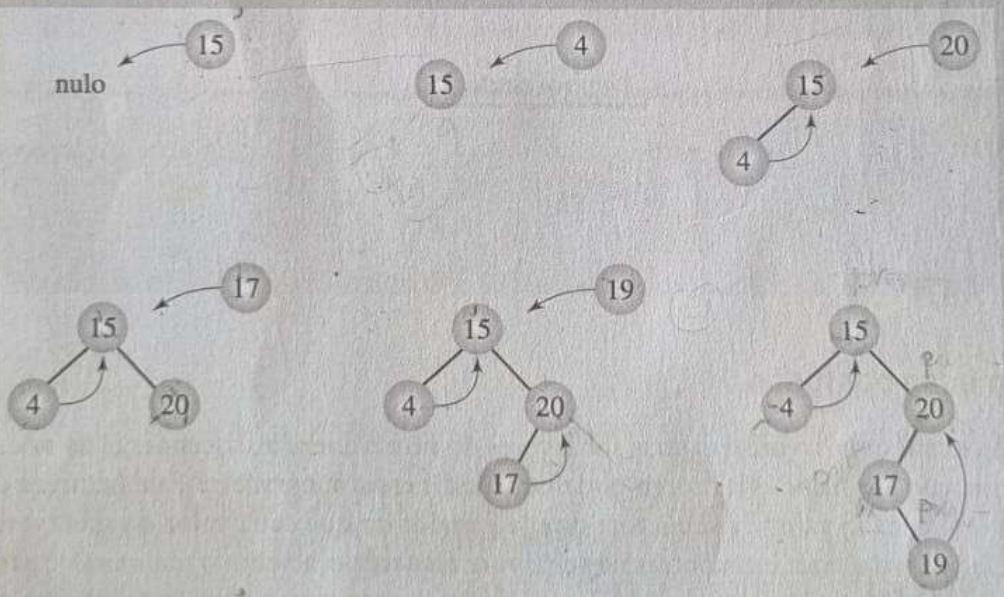
```
template<class T>
void ThreadedTree<T>::insert(const T& el) {
    ThreadedNode<T> *p, *prev = 0, *newNode;
    newNode = new ThreadedNode<T>(el);
    if (root == 0) {           // a árvore esta vazia;
        root = newNode;
        return;
    }
    p = root;                  // encontre um lugar para inserir newNode;
    while (p != 0) {
        prev = p;
        if (p->el > el)
            p = p->left;
        else if (p->successor == 0) // va para o no a direita somente se ele
            p = p->right;    // for um descendente, nao um sucessor;
        else break;          // nao siga o vinculo do sucessor;
    }
    if (prev-> el > el) {            // se newNode e o filho a esquerda de
        prev->left   = newNode;      // seu ascendente, o ascendente
        newNode->successor = 1;      // tambem se torna seu sucessor;
        newNode->right = prev;
    }
    else if (prev->successor == 1)   {// se o ascendente de newNode
        newNode->successor = 1;      // nao for o no mais a direita,
        prev->successor = 0;         // faca o sucessor do ascendente
        newNode->right = prev->right; // sucessor de newNode,
        prev->right = newNode;
    }
    else prev->right = newNode;      // caso contrario, ele nao tera sucessor;
}
```

Ao analisarmos o problema de percorrer as árvores binárias, três abordagens foram apresentadas: cruzamento com o auxílio de uma pilha, com o auxílio de linhas e através da transformação da árvore. A primeira abordagem não modifica a árvore durante o processo. A terceira a modifica, mas a restaura para a mesma condição de quando a iniciou. Somente a segunda abordagem necessita de alguma operação preparatória na árvore para se tornar praticável: exige linhas. Essas linhas podem ser criadas cada vez antes de o percurso iniciar sua tarefa, e removidas cada vez que é terminado. Se o percurso não é realizado com frequência, isto se torna uma opção viável. Outra abordagem é manter as linhas em todas as operações na árvore ao inserir um novo elemento na árvore binária de busca.

A função para inserir um nó em uma árvore alinhada é uma simples extensão de `insert()` para as árvores binárias de busca regulares, usada para ajustar as linhas sempre que possível. Esta função é para o percurso de árvore em in-ordem, e cuida somente dos sucessores, não dos predecessores.

Um nó com um filho à direita tem um sucessor em algum lugar na sua subárvore à direita. Em consequência, não necessita de uma linha de sucessor. Tais linhas são necessárias para permitir escalar a árvore, não para descê-la. Um nó sem filho à direita tem seu sucessor em algum lugar acima dele. Exceto para um nó, todos os nós sem filhos à direita terão linhas para os seus sucessores. Se um nó se torna o filho à direita de outro, ele herda o sucessor de seu novo ascendente. Se um nó se torna um filho à esquerda de outro, este ascendente se torna seu sucessor. A Figura 6.24 mostra a implementação deste algoritmo. As primeiras poucas inserções estão na Figura 6.25.

FIGURA 6.25 Inserindo nós em uma árvore alinhada.



6.6 Remoção

Remoção de um nó é outra operação necessária para manter uma árvore binária de busca. O nível de complexidade em realizar a operação depende da posição do nó a ser removido da árvore. É muito mais difícil remover um nó que tem duas subárvores do que uma folha; a complexidade do algoritmo de remoção é proporcional ao número de filhos que o nó tem. Existem três casos de remoção de um nó da árvore binária de busca:

1. O nó é uma folha e não tem filhos. Este é o caso mais fácil de se tratar. O ponteiro apropriado de seu ascendente é ajustado para nulo e o nó removido por `delete`, como na Figura 6.26.