Árvores Múltiplas

Árvores-B

Prof. Edson Alves - UnB/FGA

2019

Sumário

Árvores-B

Árvores múltiplas

- Segundo a definição formal de árvores, não há restrição quanto ao número de filhos que um nó pode ter
- Uma árvore múltipla de ordem m é um árvore cujos nós possuem, no máximo, m filhos
- As árvores binárias de busca que são árvores múltiplas de ordem 2 que impõem condições sobre as chaves dos nós com o intuito de agilizar o processo de busca.
- As árvores binárias de busca podem ser generalizadas como árvores de busca de ordem m

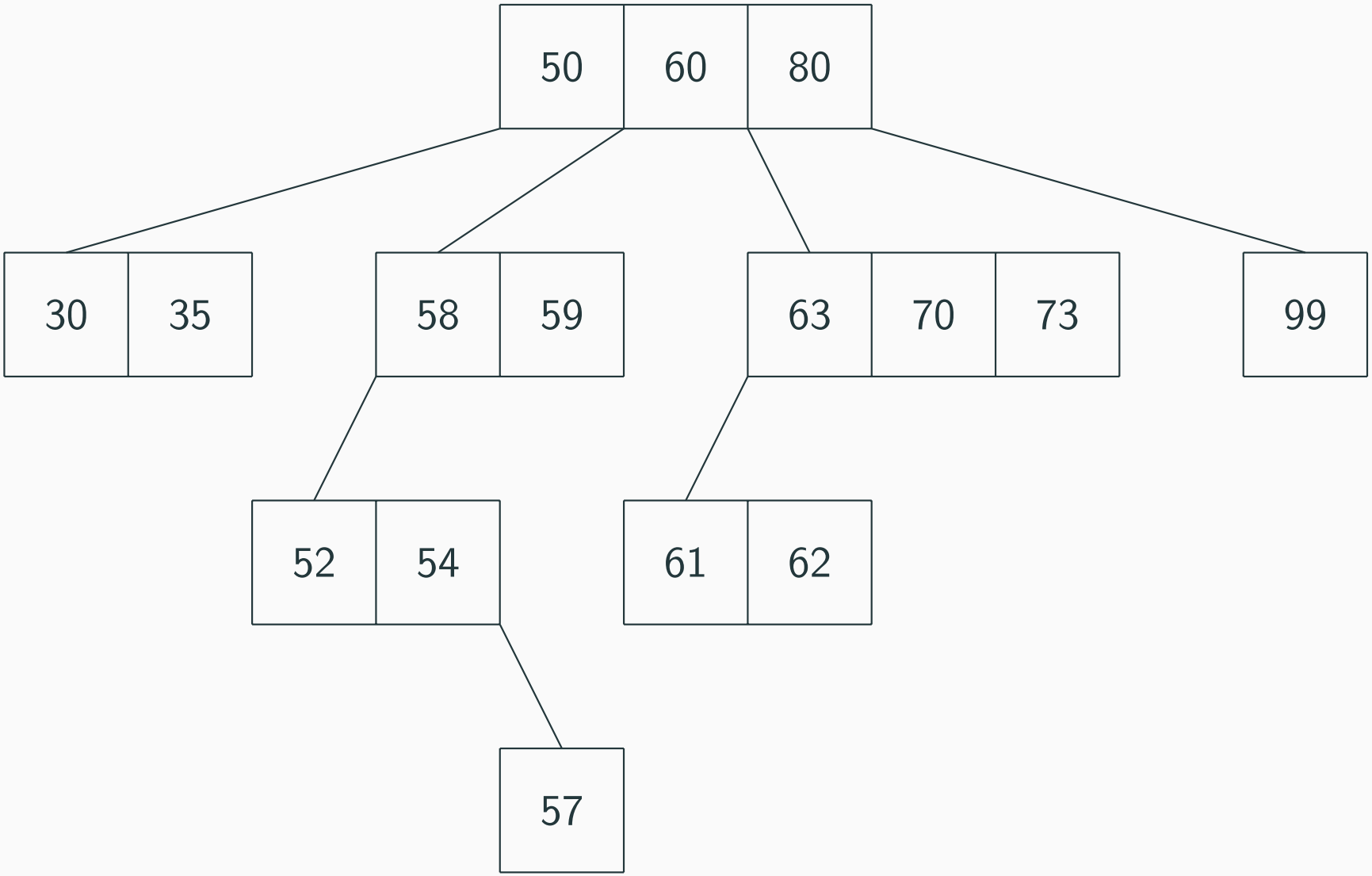
Árvores de busca de ordem m

Definição

Uma árvore de busca de ordem m é uma árvore que satisfaz as seguintes condições:

1. Cada nó tem, no máximo, m filhos e $m - 1$ chaves.
2. As chaves de cada nó são armazenadas em ordem crescente.
3. As chaves dos i primeiros filhos são menores do que a chave i.
4. As chaves dos $m - i$ últimos filhos são maiores do que a chave i.

Exemplo de árvore de busca de ordem 4



Notas sobre árvores de busca de ordem m

- As árvores de busca de ordem m tem o mesmo objetivo das árvores de busca binárias: aumentar a eficiência da rotina de busca
- Observe que, em cada nó, é preciso localizar, a partir da informação a ser encontrada e das chaves armazenadas, identificar o filho que dará sequência a busca
- A ordenação das chaves permite esta identificação em ordem $O(\log m)$, desde que o contêiner que armazena as chaves permita a busca binária
- Assim como as árvores binárias de busca, as árvores de busca de ordem m também podem ter problemas relativos ao balanceamento
- Para evitar tal problemas, existem especilizações destas árvores, como as árvores-B

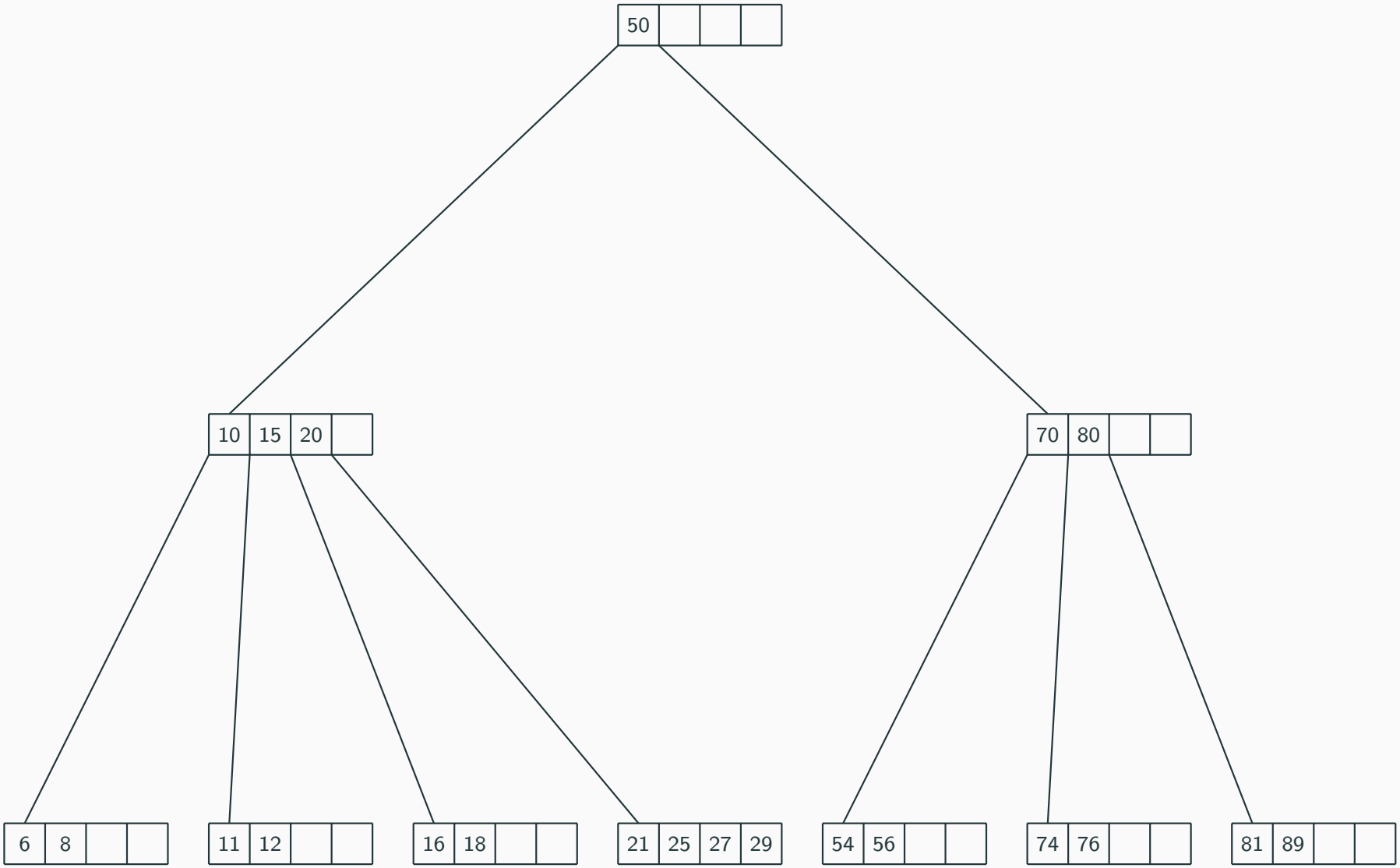
Definição de Árvores-B

Definição

Uma árvore-B de ordem m é uma árvore de busca de ordem m com as seguintes propriedades:

1. A raiz tem, no mínimo, dois filhos, caso não seja uma folha.
2. Cada nó que não é nem folha nem raiz tem $k-1$ chaves e k ponteiros para subárvores, onde $\lceil m/2 \rceil \leq k \leq m$.
3. Cada folha tem $k-1$ chaves, onde $\lceil m/2 \rceil \leq k \leq m$.
4. Todas as folhas estão no mesmo nível.

Exemplo de árvore-B de ordem 5



Árvores-B

- As árvores-B foram propostas por Bayer e McCreigh em 1972.
- O número de chaves armazenadas em uma árvore-B é proporcional a metade de sua capacidade máxima
- Devido às suas propriedades, uma árvore-B tem poucos níveis
- Uma árvore-B está sempre perfeitamente balanceada
- Um nó de uma árvore-B possui dois contêiners: um para armzenar as $m - 1$ chaves e outro para os m ponteiros para os filhos
- Na implementação dos nós de árvores-B costuma-se adicionar informações extras que facilitem a manutenção da árvore, como o número de chaves do nó e uma indicação se o nó é folha ou não

Exemplo de implementação de uma árvore-B

```cpp
#include <bits/stdc++.h>

using namespace std;

template<typename T, size_t M>
class BTree {
private:
    struct Node {
        bool leaf;
        Node *parent;
        vector<T> keys;
        vector<Node *> children;

        Node(bool is_leaf = true) : leaf(is_leaf), parent(nullptr) {}
    };

    Node *root;

public:
    // A árvore é inicializada com um nó sem nenhuma chave armazenada
    BTree() : root(new Node()) {}
```

Exemplo de implementação de uma árvore-B

```
23      // Complexidade O(log N log M)
24      bool search(const T& info) const
25      {
26          auto node = search(root, info);
27
28          return binary_search(node->keys.begin(), node->keys.end(), info);
29      }
30
31  private:
32      // Procura pelo nó onde a informação deveria estar
33      Node * search(Node *node, const T& info) const
34      {
35          if (node->leaf)
36              return node;
37
38          auto i = lower_bound(node->keys.begin(), node->keys.end(), info)
39              - node->keys.begin();
40
41          return search(node->children[i], info);
42      }
43
```

Inserção em árvores-B

Há 3 casos a serem tratados na inserção de um elemento em uma árvore-B, uma vez localizado o nó onde deve ocorrer a inserção:

1. O nó é uma folha com espaço livre: A estrutura da árvore não é alterada. Pode ser necessário transpor algumas chaves para que se mantenha a ordem crescente das mesmas.
2. O nó é uma folha sem espaço livre: O nó deve ser dividido em dois nós. O novo nó deve receber a metade superior do antigo nó (já contabilizado o novo elemento), enquanto que a maior chave restante no antigo nó é migrada para o nó pai. Também deve-se adicionar uma referência ao novo nó no pai.
3. O nó é a raiz e ela está sem espaço livre: Deve-se proceder como no caso de uma folha cheia, dividindo o nó em dois. Deve-se criar um novo nó para ser a nova raiz, e este nó fará o papel do pai no caso anterior. Esta é a única inserção que pode alterar a altura da árvore.

Exemplo de inserção no primeiro caso, $m = 4$

Elemento a ser inserido: 50

Exemplo de inserção no primeiro caso, $m = 4$

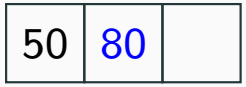
Elemento a ser inserido: 50

Exemplo de inserção no primeiro caso, $m = 4$

Elemento a ser inserido: 80

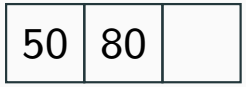
Exemplo de inserção no primeiro caso, $m = 4$

Elemento a ser inserido: 80



Exemplo de inserção no primeiro caso, $m = 4$

Elemento a ser inserido: 30



Exemplo de inserção no primeiro caso, $m = 4$

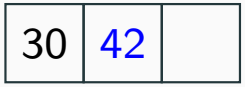
Elemento a ser inserido: 30

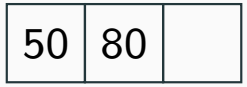
Exemplo de inserção no terceiro caso, $m = 4$

Elemento a ser inserido: 42

Exemplo de inserção no terceiro caso, $m = 4$

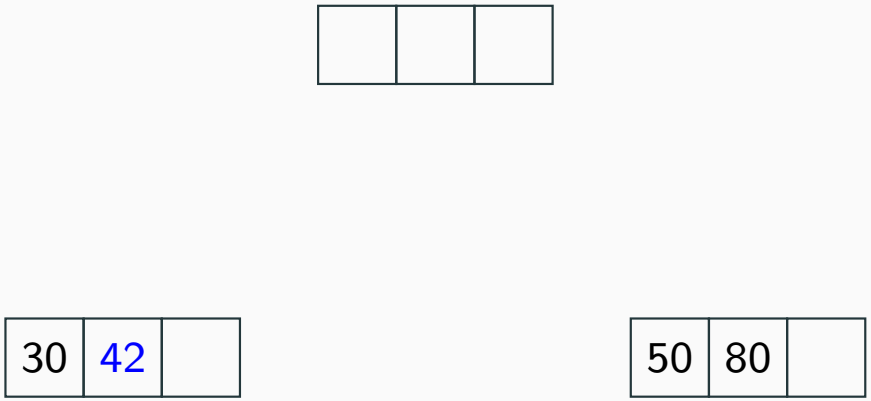
Elemento a ser inserido: 42





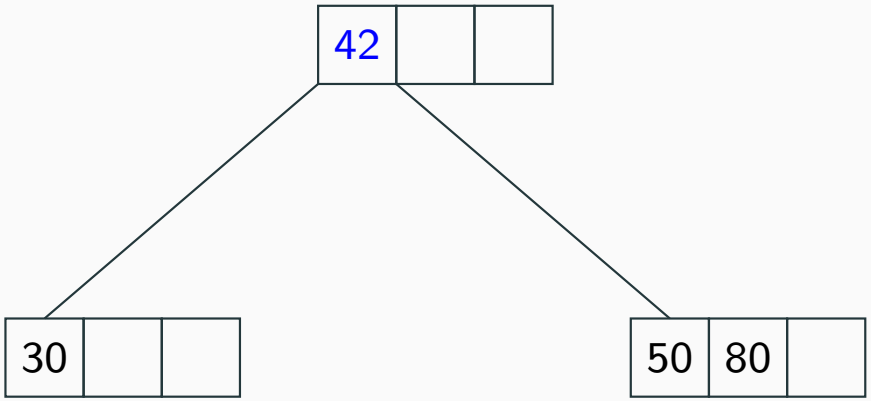
Exemplo de inserção no terceiro caso, $m = 4$

Fusão do nó dividido, nova raiz



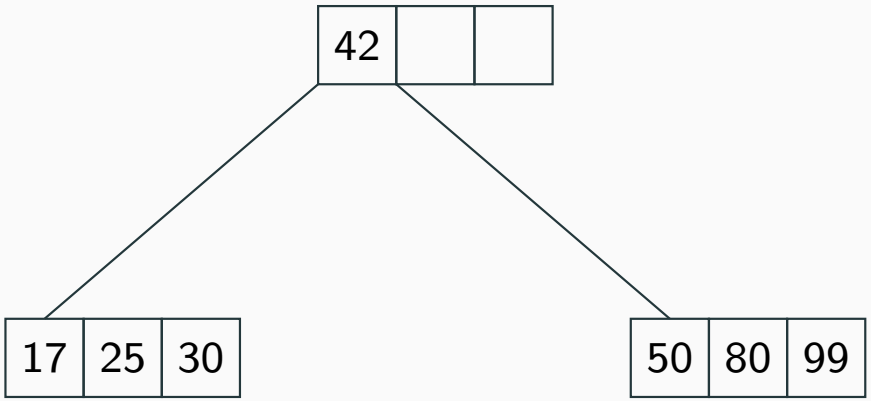
Exemplo de inserção no terceiro caso, $m = 4$

Fusão do nó dividido, nova raiz



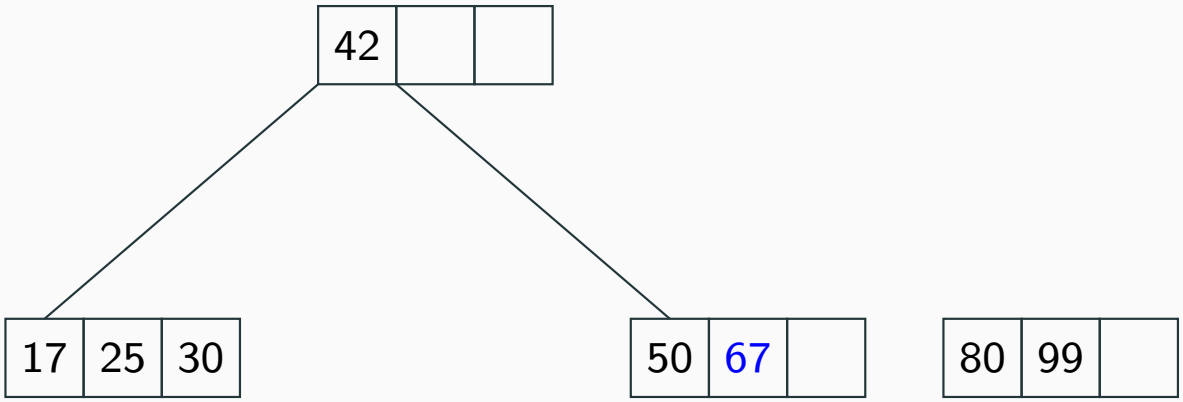
Exemplo de inserção no segundo caso, $m = 4$

Elemento a ser inserido: 67



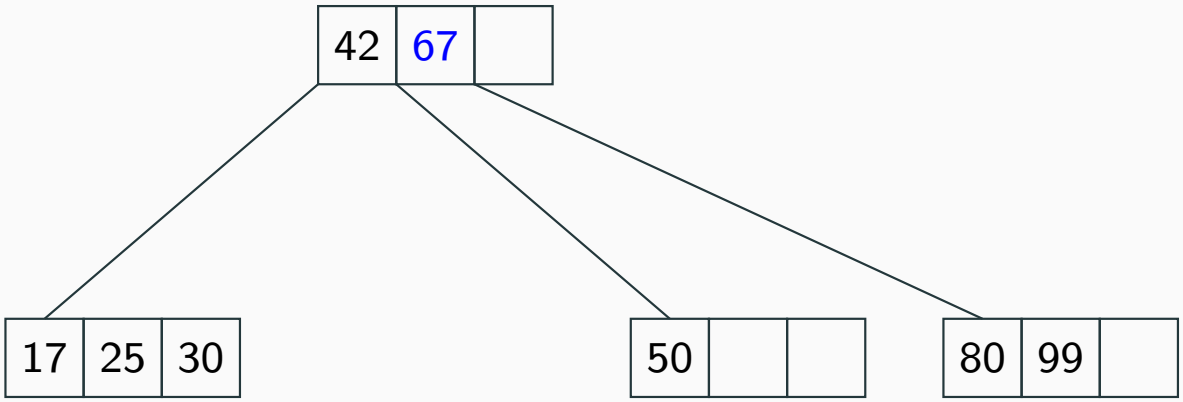
Exemplo de inserção no segundo caso, $m = 4$

Elemento a ser inserido: 67



Exemplo de inserção no segundo caso, $m = 4$

Elemento a ser inserido: 67



Implementação da inserção em uma árvore-B

```
44 public:
45     bool insert(const T& info)
46     {
47         // Não insere informações duplicadas
48         if (search(info))
49             return false;
50 
51         auto node = search(root, info);
52         return insert(info, node);
53     }
54 
55 private:
56     bool insert(const T& info, Node *node, Node* child = nullptr)
57     {
58         auto child_compare = [](const Node *a, const Node *b)
59             {
60                 if (a->keys.empty())
61                     return false;
62 
63                 if (b->keys.empty())
64                     return true;
```

Implementação da inserção em uma árvore-B

```
65
66                  return a->keys[0] < b->keys[0];
67              };
68
69          node->keys.push_back(info);
70          sort(node->keys.begin(), node->keys.end());
71
72          if (child)
73          {
74              node->children.push_back(child);
75              sort(node->children.begin(), node->children.end(),
76                  child_compare);
77          }
78
79          // Capacidade do nó superada: o nó deve ser dividido
80          if (node->keys.size() == M)
81          {
82              auto S = new Node(node->leaf);
83              auto half = M/2;
84
```

Implementação da inserção em uma árvore-B

```
85              // Divide as chaves
86              for (size_t i = half; i < M; ++i)
87              {
88                  S->keys.push_back(node->keys.back());
89                  node->keys.pop_back();
90              }
91
92              reverse(S->keys.begin(), S->keys.end());
93
94              // Determina o elemento do meio, que subirá para o pai
95              auto new_info = node->keys.back();
96              node->keys.pop_back();
97
98              // Divide os filhos, se necessário
99              if (node->leaf == false)
100             {
101                 for (size_t i = 0; i <= S->keys.size(); ++i)
102                 {
103                     S->children.push_back(node->children.back());
104                     node->children.pop_back();
105                 }
```

Implementação da inserção em uma árvore-B

```
107                 reverse(S->children.begin(), S->children.end());
108             }
109
110             if (node->parent)
111             {
112                 S->parent = node->parent;
113                 return insert(new_info, node->parent, S);
114             } else
115             {
116                 root = new Node(false);
117                 root->keys.push_back(new_info);
118                 root->children.push_back(node);
119                 root->children.push_back(S);
120
121                 node->parent = root;
122                 S->parent = root;
123             }
124         }
125
126         return true;
127     }
```

Referências

1. **DROZDEK**, Adam. *Algoritmos e Estruturas de Dados em C++*, 2002.
2. myUSF. Algorithm Visualization – B-Trees, acesso em 29/04/2019.
3. Wikipedia. B-tree, acesso em 29/04/2019.