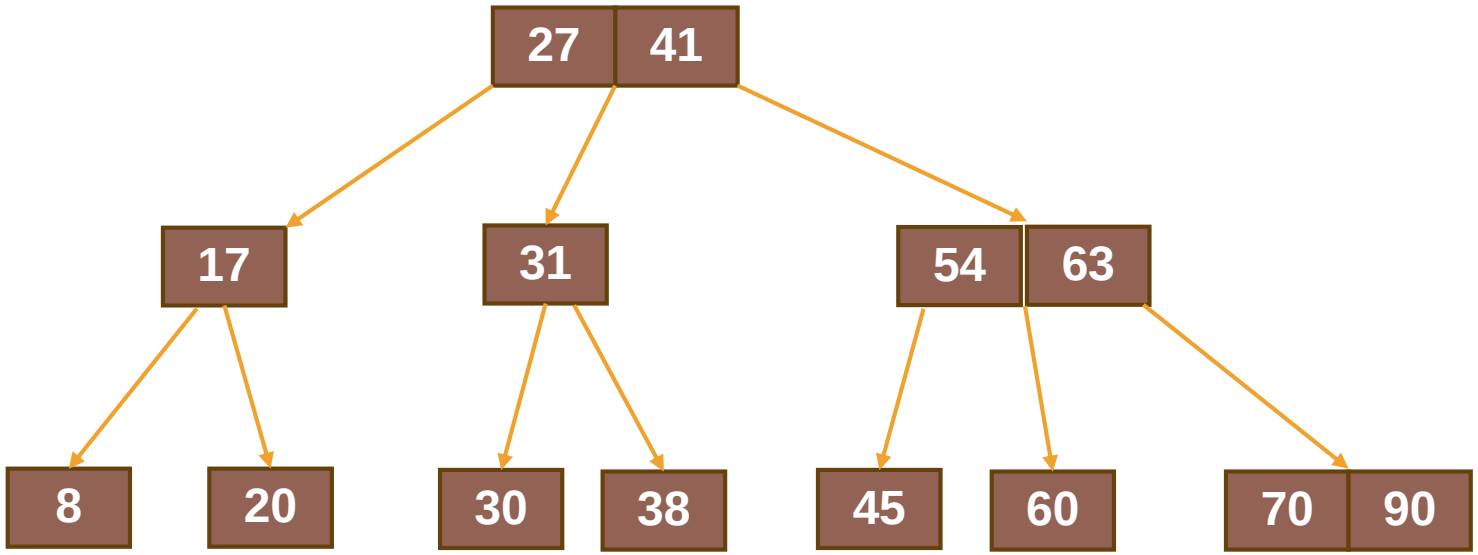
5 B - TREE

B-tree là một cây tìm kiếm cân bằng tổng quát, mỗi nút có thể có nhiều khóa và các khóa được sắp xếp tăng dần.

Đặc điểm nổi bật của B-tree là mỗi nút có thể chứa nhiều khóa và nhiều con. Điều này giúp giảm chiều cao của cây một cách đáng kể, từ đó giảm số lần truy cập đĩa khi tìm kiếm.



5 B - TREE

Các đặc trưng chính của B-tree

Một B-tree có các đặc trưng sau đây, được xác định bởi một tham số gọi là **bậc** của cây (t hoặc m).

Số khóa của nút: Tối đa m−1 khóa, tối thiểu [m/2]−1 khóa (trừ nút gốc).

Số con của nút: Tối đa m con, tối thiểu [m/2] con (trừ nút gốc).

Tính cân bằng: Tất cả các nút lá nằm trên cùng một mức.

Nút gốc: Có ít nhất 1 khóa và 2 con (trừ khi cây rỗng).

5 B - TREE

Các đặc trưng chính của B-tree

1. Nút (Node)

Mỗi nút trong B-tree có thể chứa một số lượng khóa (key) và con trỏ tới các nút con.

- Mỗi nút có tối đa m−1 khóa và m con.
- Các khóa trong mỗi nút được sắp xếp theo thứ tự tăng dần.

2. Cấu trúc Nút

- Một nút không phải là lá (non-leaf node) với k khóa sẽ có k+1 con.
- Mỗi khóa K_i trong một nút sẽ phân chia các khóa của cây con: tất cả các khóa trong cây con thứ i (con trỏ C_i) nhỏ hơn K_i, và tất cả các khóa trong cây con thứ i+1 (con trỏ C_{i+1}) lớn hơn K_i.

5 B - TREE

Các đặc trưng chính của B-tree

3. Bậc của cây (Order of the Tree)

Bậc của cây, thường ký hiệu là **m** (hoặc đôi khi 2t), là số con tối đa mà một nút có thể có.

- Số lượng khóa tối đa trong một nút là m−1.
- Số lượng khóa tối thiểu trong một nút là ⌈m/2⌉−1 (với nút gốc, số khóa tối thiểu là 1).

4. Tính cân bằng (Balanced Structure)

Tất cả các nút lá đều nằm trên cùng một mức (cùng một độ sâu). Điều này đảm bảo rằng đường đi từ gốc đến bất kỳ nút lá nào đều có cùng chiều dài, giúp cho thời gian tìm kiếm luôn ổn định và hiệu quả.

5 B - TREE

Các đặc trưng chính của B-tree

5. Điều kiện của nút gốc (Root Node)

Nút gốc phải có ít nhất 1 khóa (tức là 2 con) trừ khi nó là nút duy nhất của cây (trường hợp cây rỗng hoặc chỉ có 1 nút).

6. Điều kiện của các nút khác (Non-root Nodes)

Mỗi nút không phải là gốc và không phải là lá phải có ít nhất $\lceil m/2 \rceil$ con. Điều này tương đương với việc mỗi nút có ít nhất $\lceil m/2 \rceil - 1$ khóa.

5 B - TREE

Cài đặt B-tree bằng DSLK

```
struct NODE
{
        int *keys;          // Mảng chứa khóa
        int m;              // Bậc của cây
        NODE **children;    // Con trỏ tới các con
        int n;              // Số lượng khóa hiện có
        bool leaf;          // true nếu là nút lá
};
// Tạo nút mới
NODE* create_node(int m, bool leaf)
{
        NODE* P = new NODE;
        P->m = m;
        P->leaf = leaf;
        P->keys = new int[m - 1];// tối đa m-1 khóa
        P->children = new NODE*[m];// tối đa m con
        P->n = 0;
        return P;
}
```

5 B - TREE

Thuật toán tìm kiếm trên B-tree

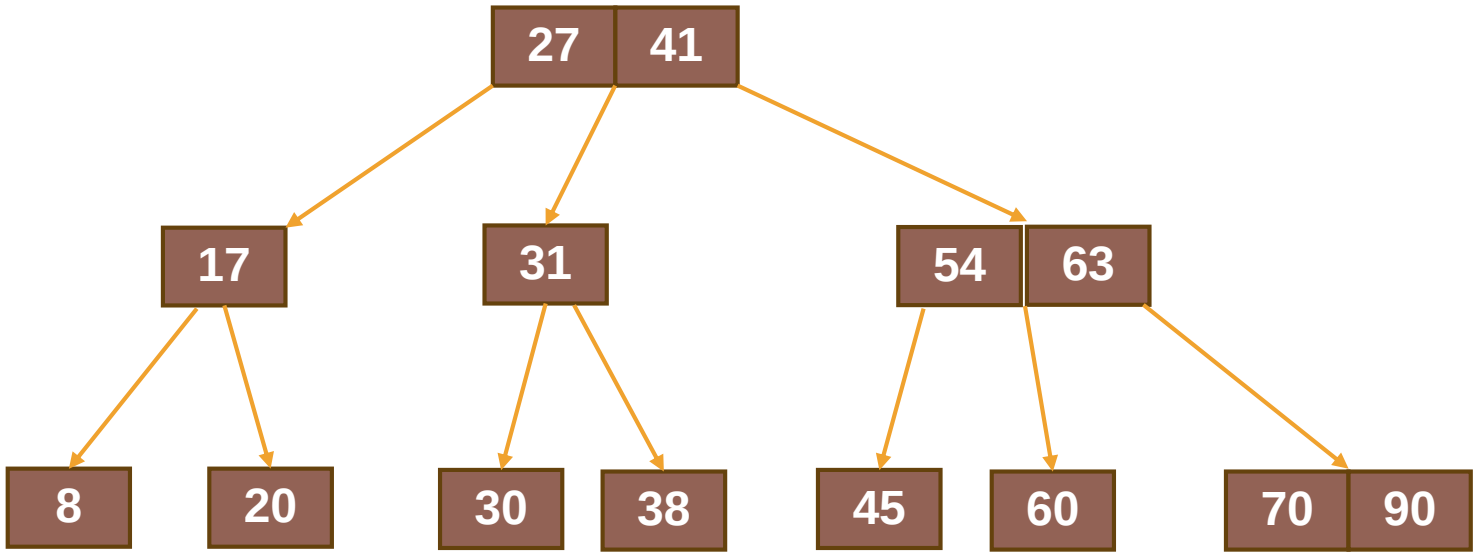
1. Bắt đầu từ nút gốc (root).
2. Trong mỗi nút, duyệt qua các khóa.
   - Nếu tìm thấy khóa cần tìm, thao tác kết thúc.
   - Nếu khóa cần tìm nhỏ hơn khóa đầu tiên của nút, di chuyển đến nút con đầu tiên.
   - Nếu khóa cần tìm nằm giữa hai khóa K_i và K_{i+1}, di chuyển đến nút con nằm giữa hai khóa đó.
   - Nếu khóa cần tìm lớn hơn tất cả các khóa trong nút, di chuyển đến nút con cuối cùng.
3. Lặp lại bước 2 cho đến khi tìm thấy khóa hoặc đến một nút lá mà không tìm thấy. Nếu không tìm thấy ở nút lá, khóa đó không tồn tại trong cây.

5 B - TREE

Thuật toán tìm kiếm trên B-tree

Ví dụ: Tìm khóa 45 trong B-tree sau:

45

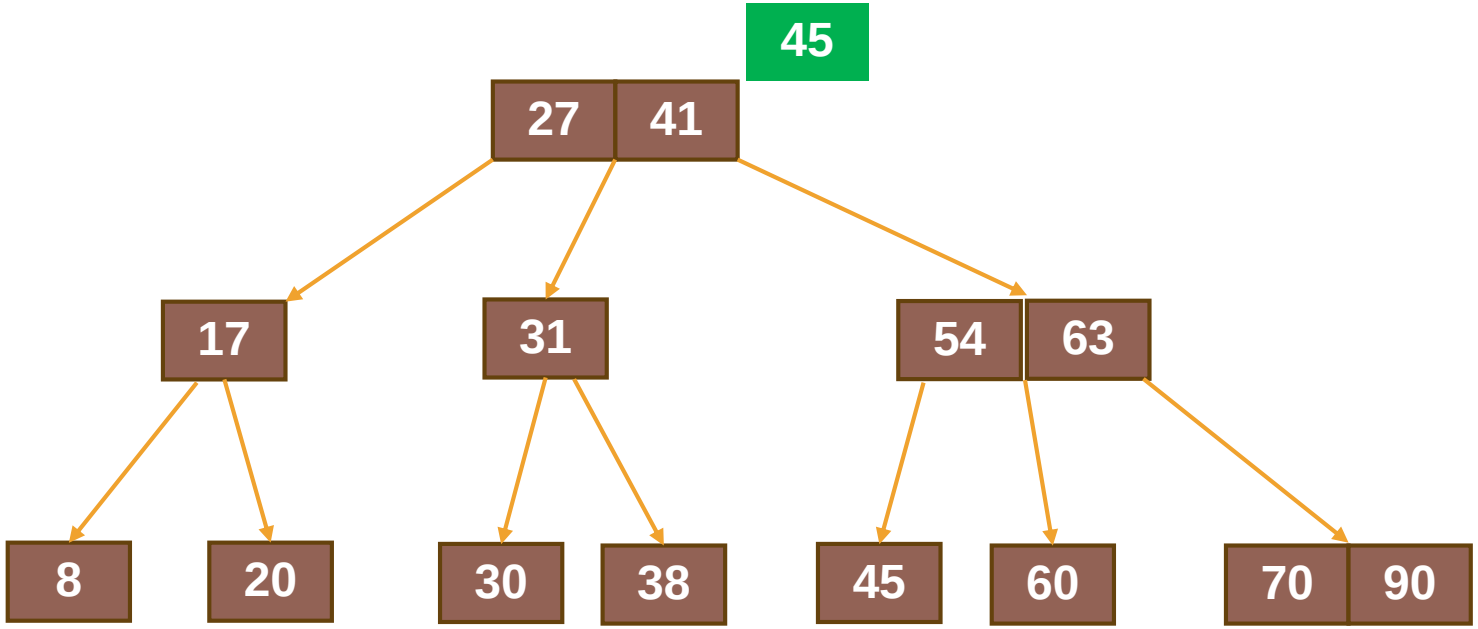


5 B - TREE

Thuật toán tìm kiếm trên B-tree

Ví dụ: Tìm khóa 45 trong B-tree sau:

Đầu tiên tìm ở nút gốc: khóa cần tìm lớn hơn tất cả các khóa nên ta sẽ tìm ở nút con cuối cùng

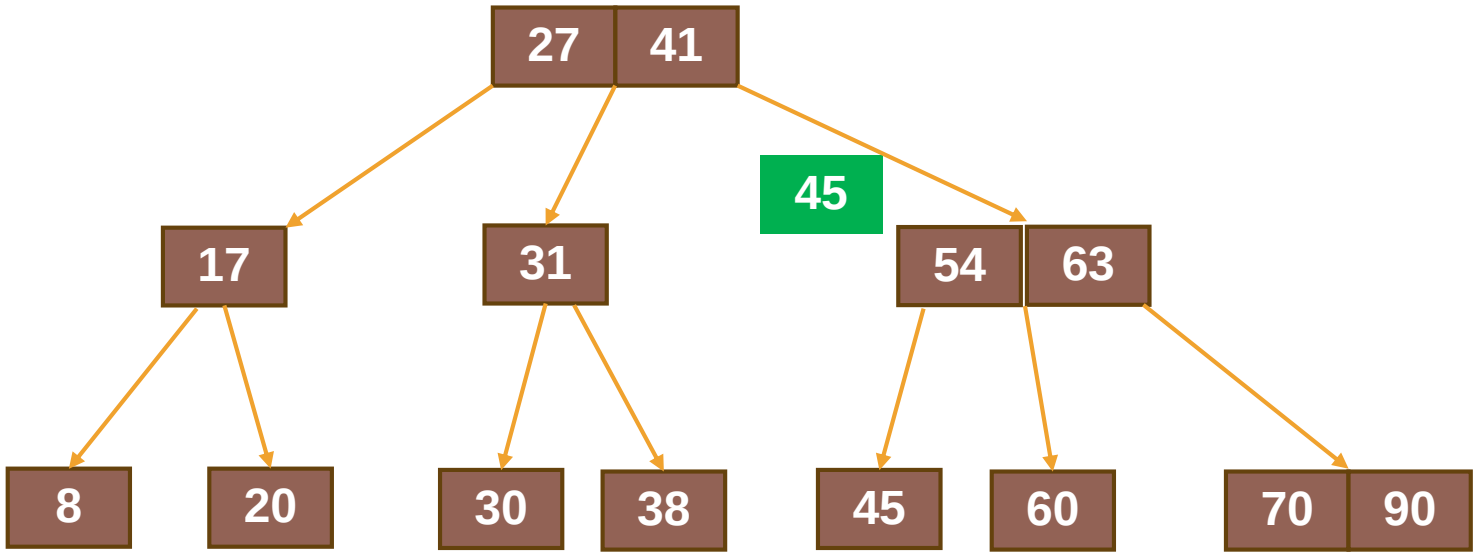


5 B - TREE

Thuật toán tìm kiếm trên B-tree

Ví dụ: Tìm khóa 45 trong B-tree sau:

Khóa cần tìm cũng nhỏ hơn tất cả các khóa trên nút đang tìm (các khóa 54, 63) nên ta sẽ tìm ở nút con đầu tiên của nút này.

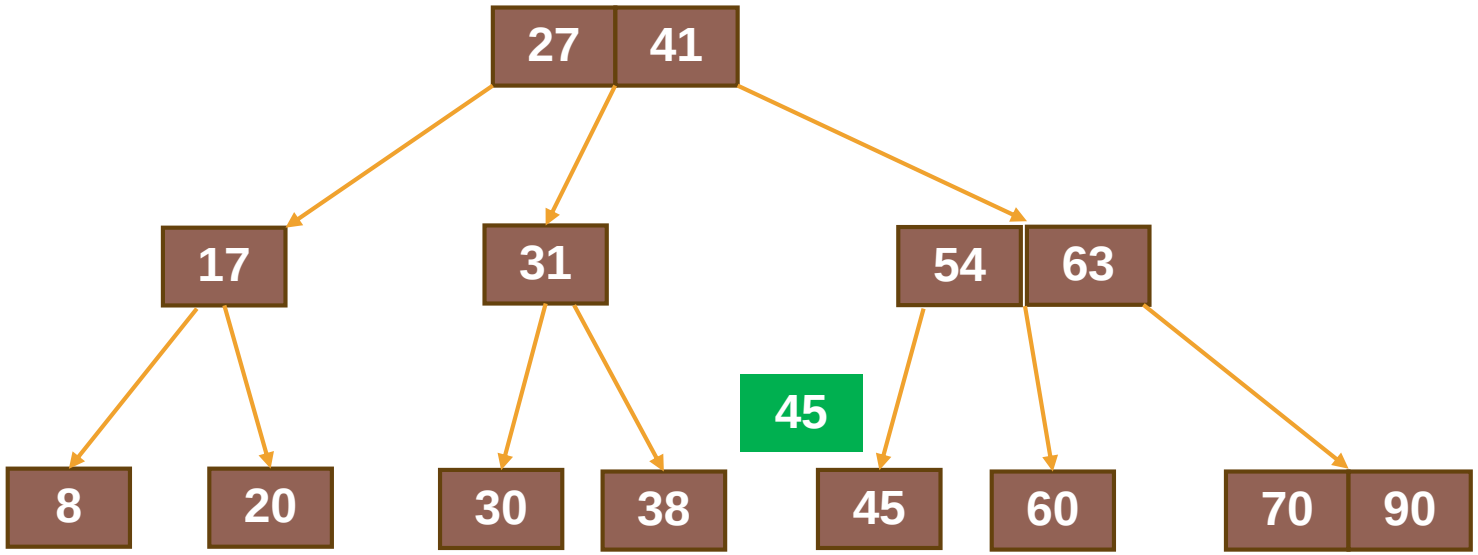


5 B - TREE

Thuật toán tìm kiếm trên B-tree

Ví dụ: Tìm khóa 45 trong B-tree sau:

Nút tìm kiếm có duy nhất khóa 45 và bằng giá trị ta cần tìm



5 B - TREE

```
// Tìm kiếm khóa
NODE* search(NODE* root, int k)
{
        if (root == NULL)
                return NULL;
        int i = 0;
        while (i < root->n && k > root->keys[i])
                i++;
        if (i < root->n && root->keys[i] == k)
                return root;
        if (root->leaf)
                return NULL;
        return search(root->children[i], k);
}
```

5 B - TREE

Thuật toán chèn khóa vào B-tree

Nguyên tắc:

- Không vi phạm số khóa tối đa.

- Luôn đảm bảo tính cân bằng.

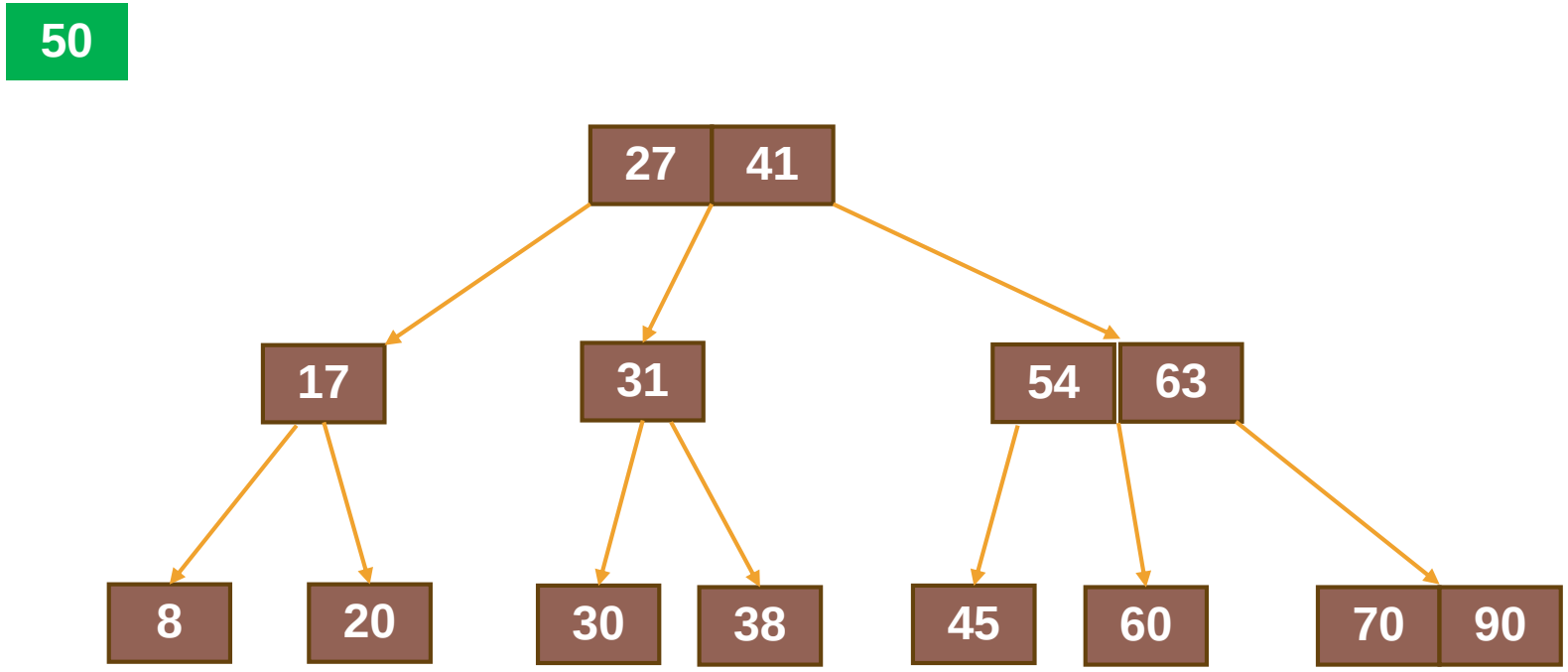
Việc chèn luôn được thực hiện tại một nút lá. Thao tác này có thể phức tạp nếu nút lá đã đầy**.** Các bước thực hiện như sau:

1. Tìm vị trí nút lá thích hợp để chèn khóa mới.
2. Nếu nút lá đó **chưa đầy** (số khóa < m−1), chỉ cần chèn khóa vào và sắp xếp lại các khóa trong nút.
3. Nếu nút lá đã **đầy** (số khóa = m−1), ta phải thực hiện **tách nút (split)**:
   - Chia nút đầy thành hai nút mới.
   - Đẩy khóa ở giữa lên nút cha.
   - Nếu nút cha cũng đầy, quá trình tách này sẽ tiếp tục "lan truyền" lên trên (splitting propagates upward) cho đến khi gặp một nút chưa đầy, hoặc tạo thành một nút gốc mới.

5 B - TREE

Thuật toán chèn khóa vào B-tree

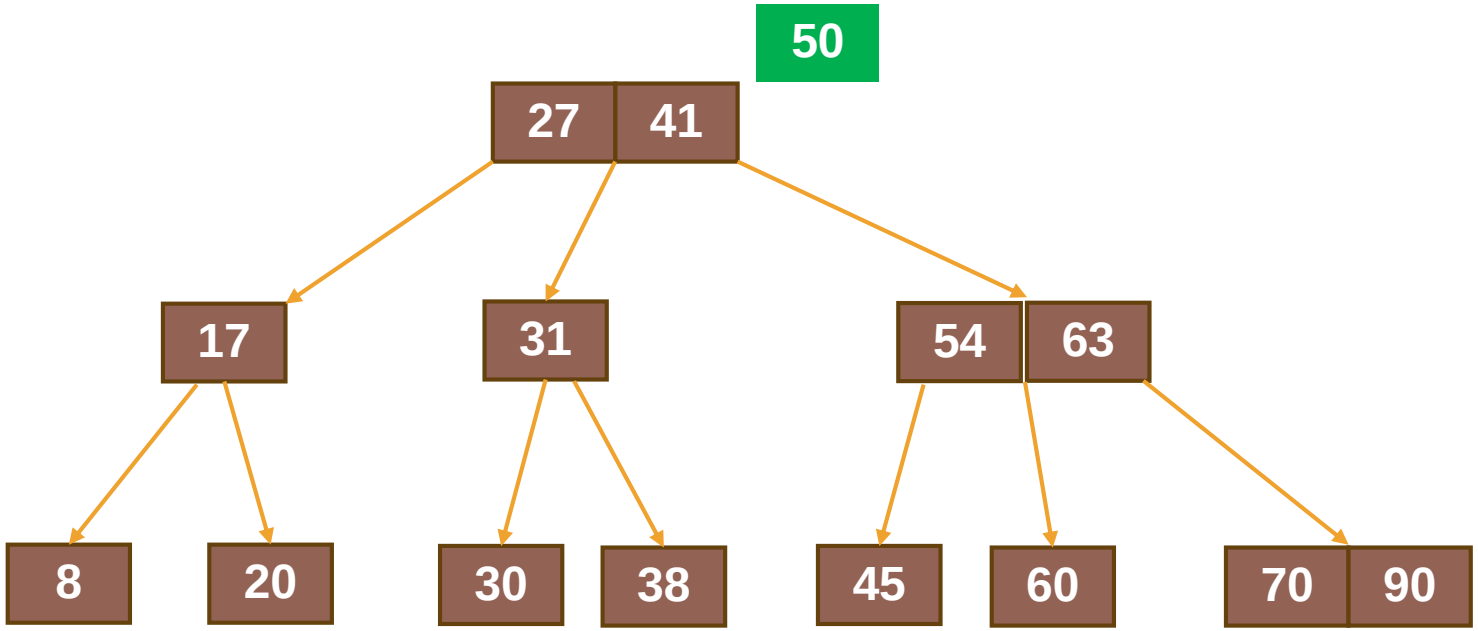
Ví dụ: Chèn nút có khóa 50 vào B-tree sau (bậc 3).



5 B - TREE

Thuật toán chèn khóa vào B-tree

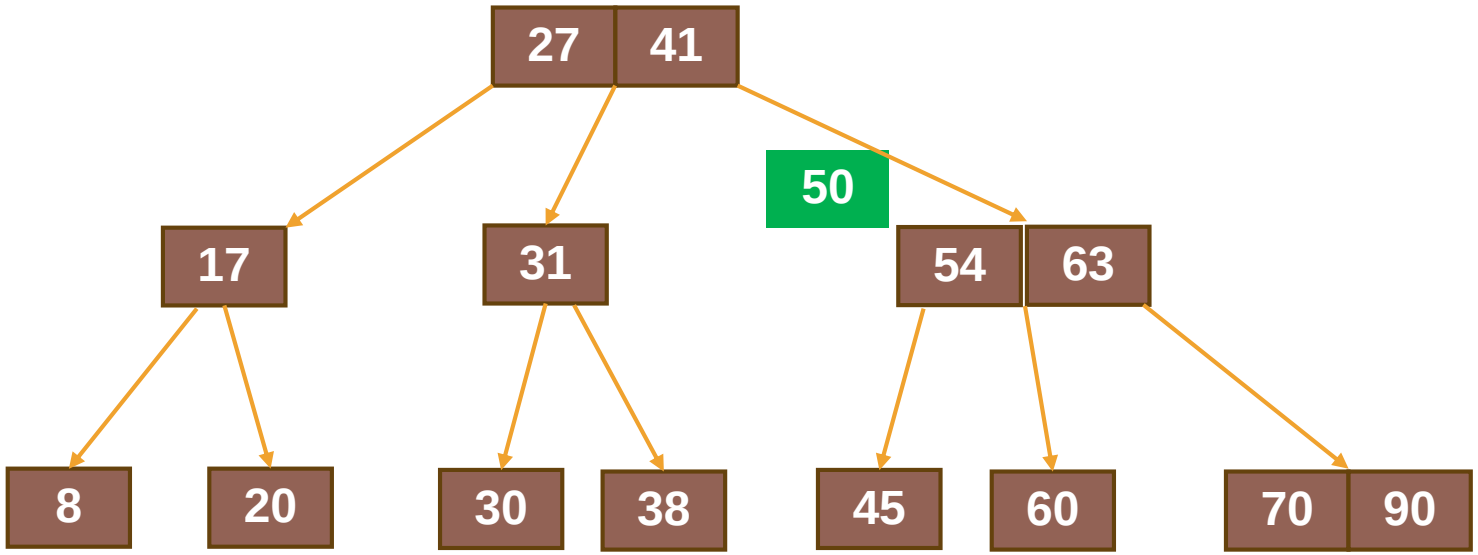
Ví dụ: Chèn nút có khóa 50 vào B-tree sau (bậc 3).



5 B - TREE

Thuật toán chèn khóa vào B-tree

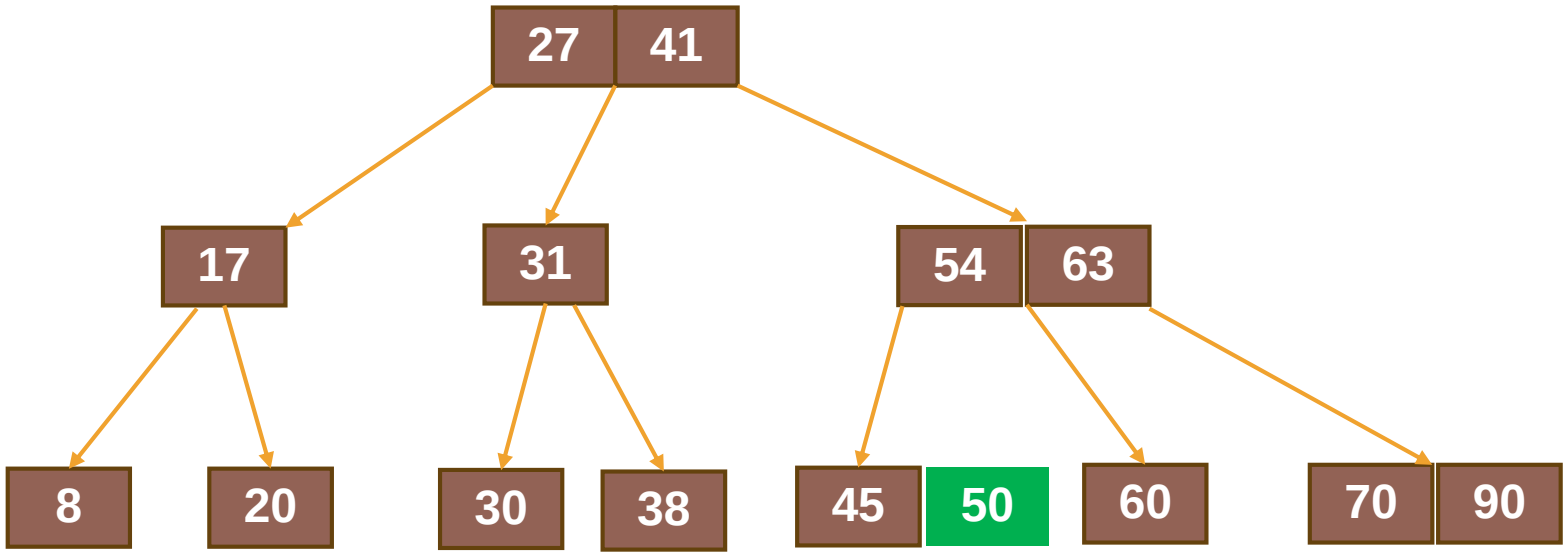
Ví dụ: Chèn nút có khóa 50 vào B-tree sau (bậc 3).



5 B - TREE

Thuật toán chèn khóa vào B-tree

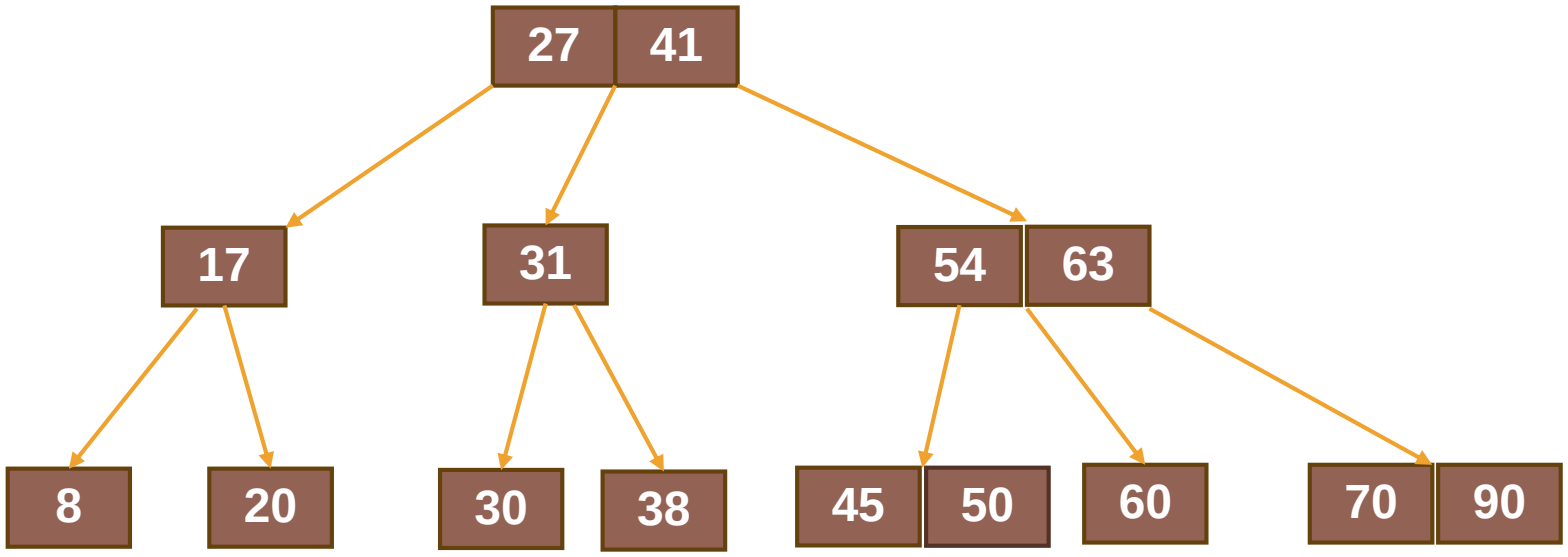
Ví dụ: Chèn nút có khóa 50 vào B-tree sau (bậc 3).



5 B - TREE

Thuật toán chèn khóa vào B-tree

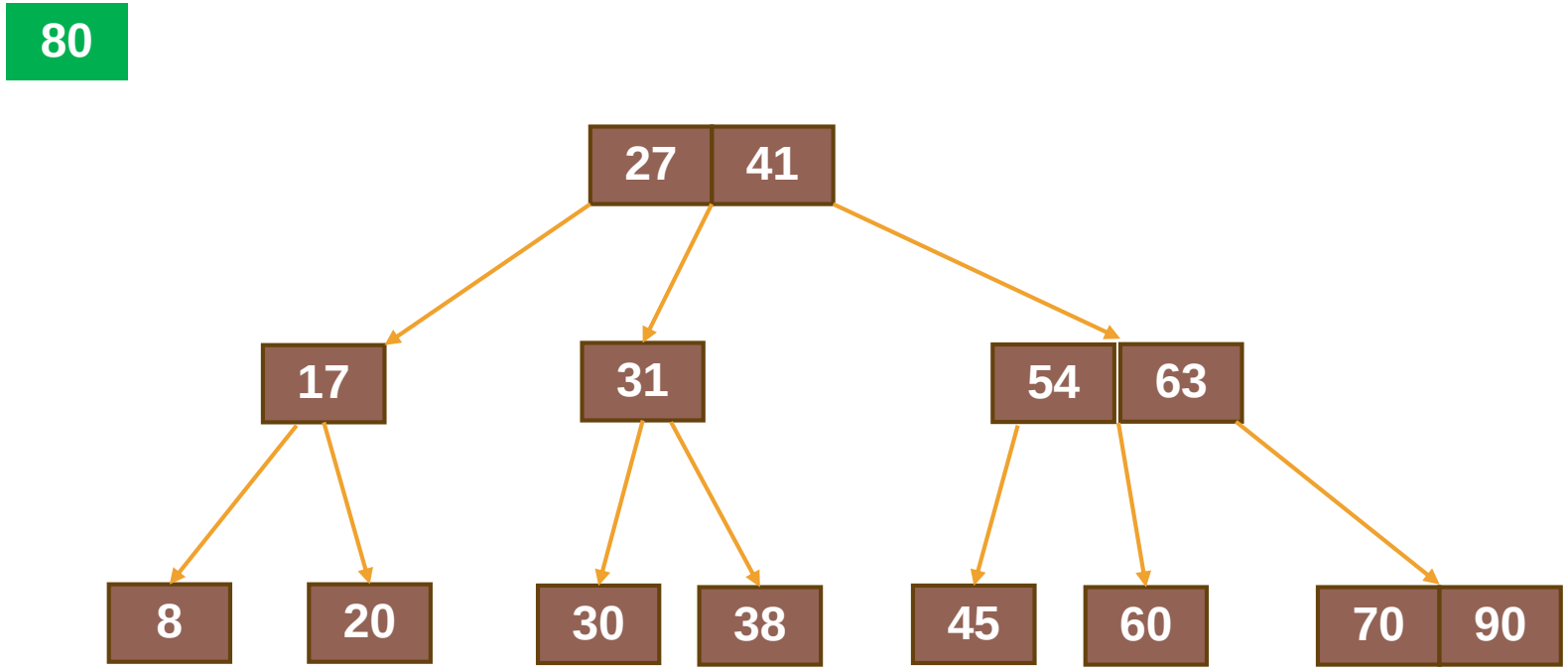
Ví dụ: Chèn nút có khóa 50 vào B-tree sau (bậc 3).



5 B - TREE

Thuật toán chèn khóa vào B-tree

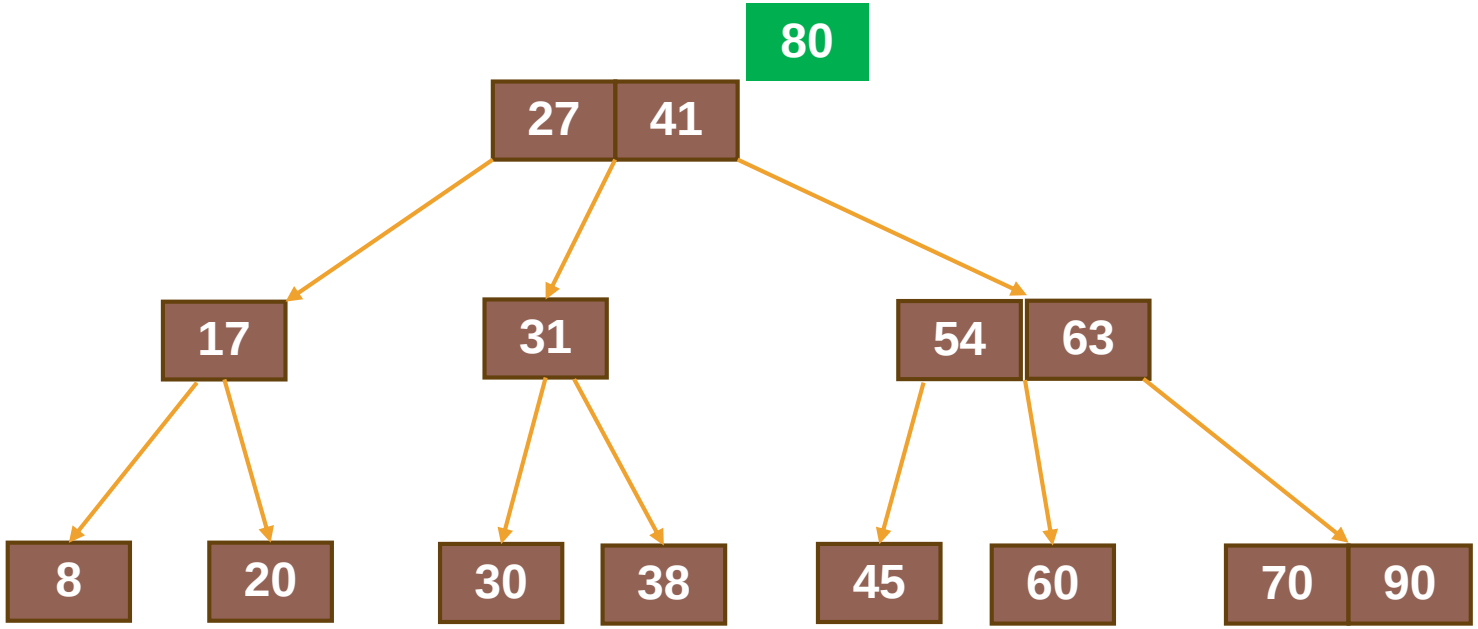
Ví dụ: Chèn nút có khóa 80 vào B-tree sau (bậc 3).



5 B - TREE

Thuật toán chèn khóa vào B-tree

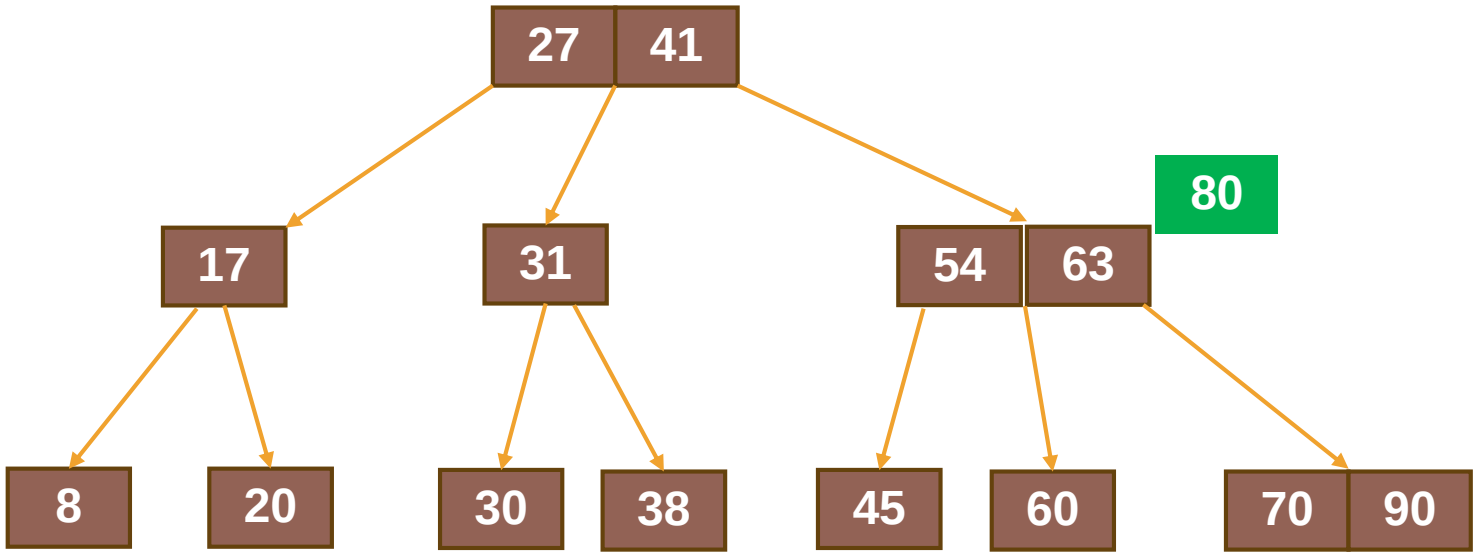
Ví dụ: Chèn nút có khóa 80 vào B-tree sau (bậc 3).



5 B - TREE

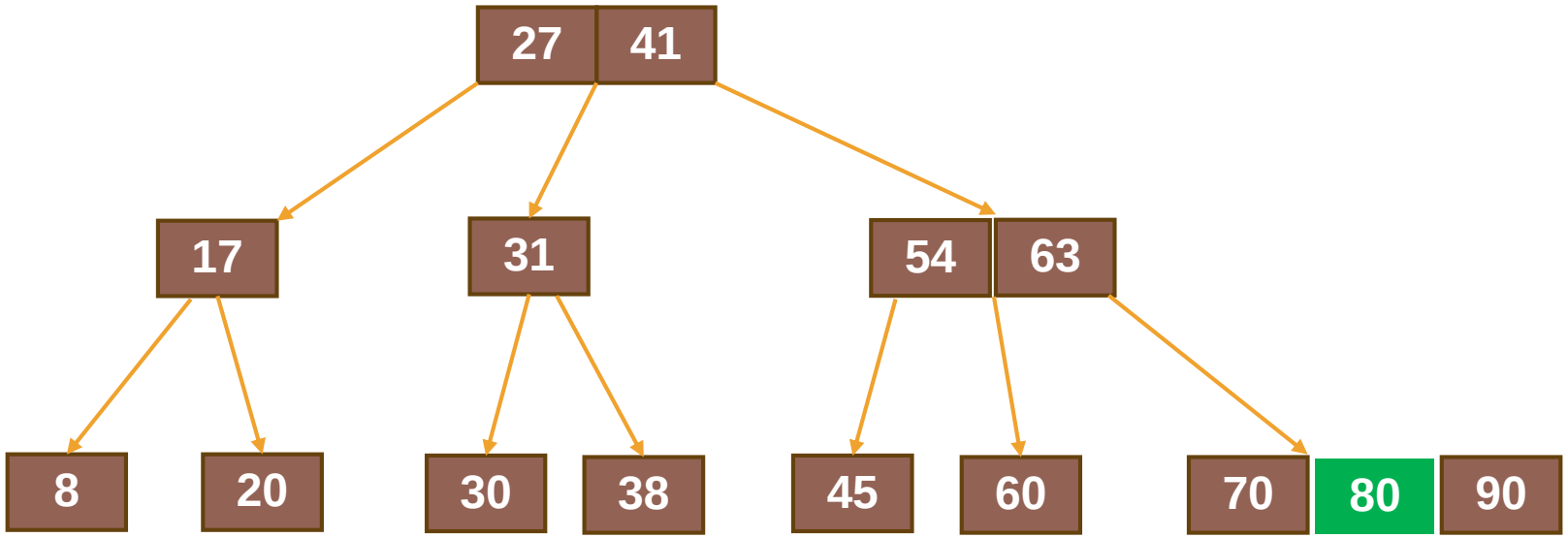
Thuật toán chèn khóa vào B-tree

Ví dụ: Chèn nút có khóa 80 vào B-tree sau (bậc 3).



5 B - TREE

Thuật toán chèn khóa vào B-tree

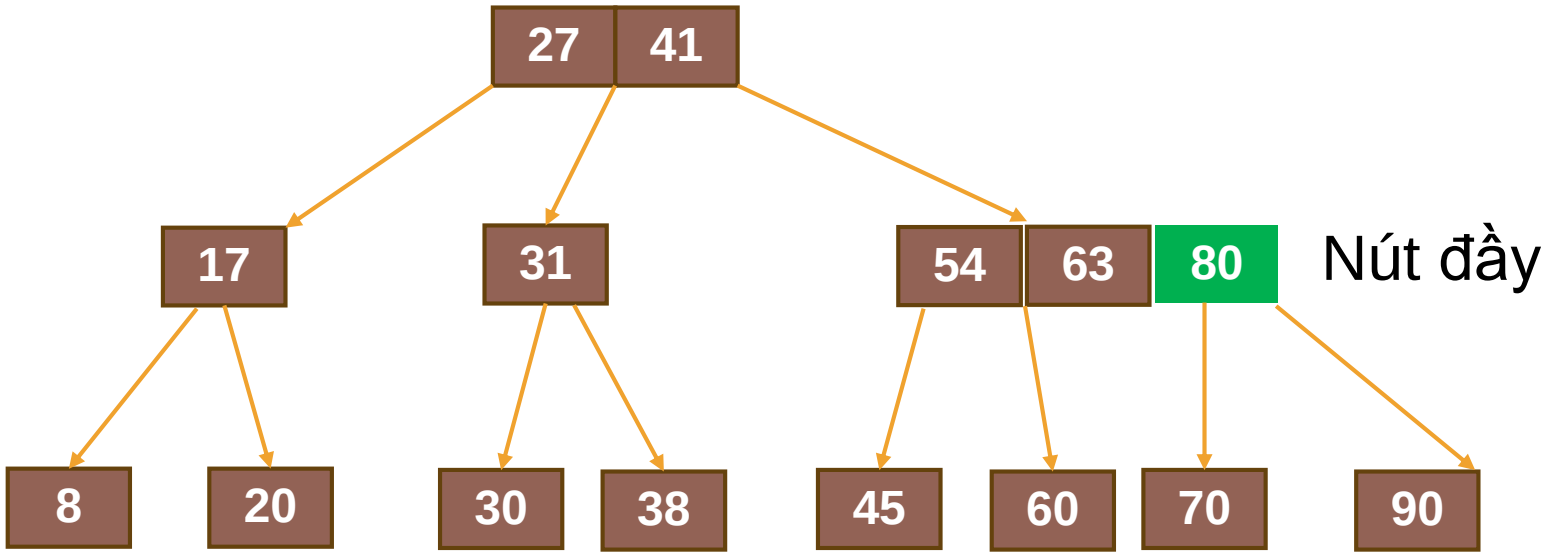


Nút đầy

5 B - TREE

Thuật toán chèn khóa vào B-tree

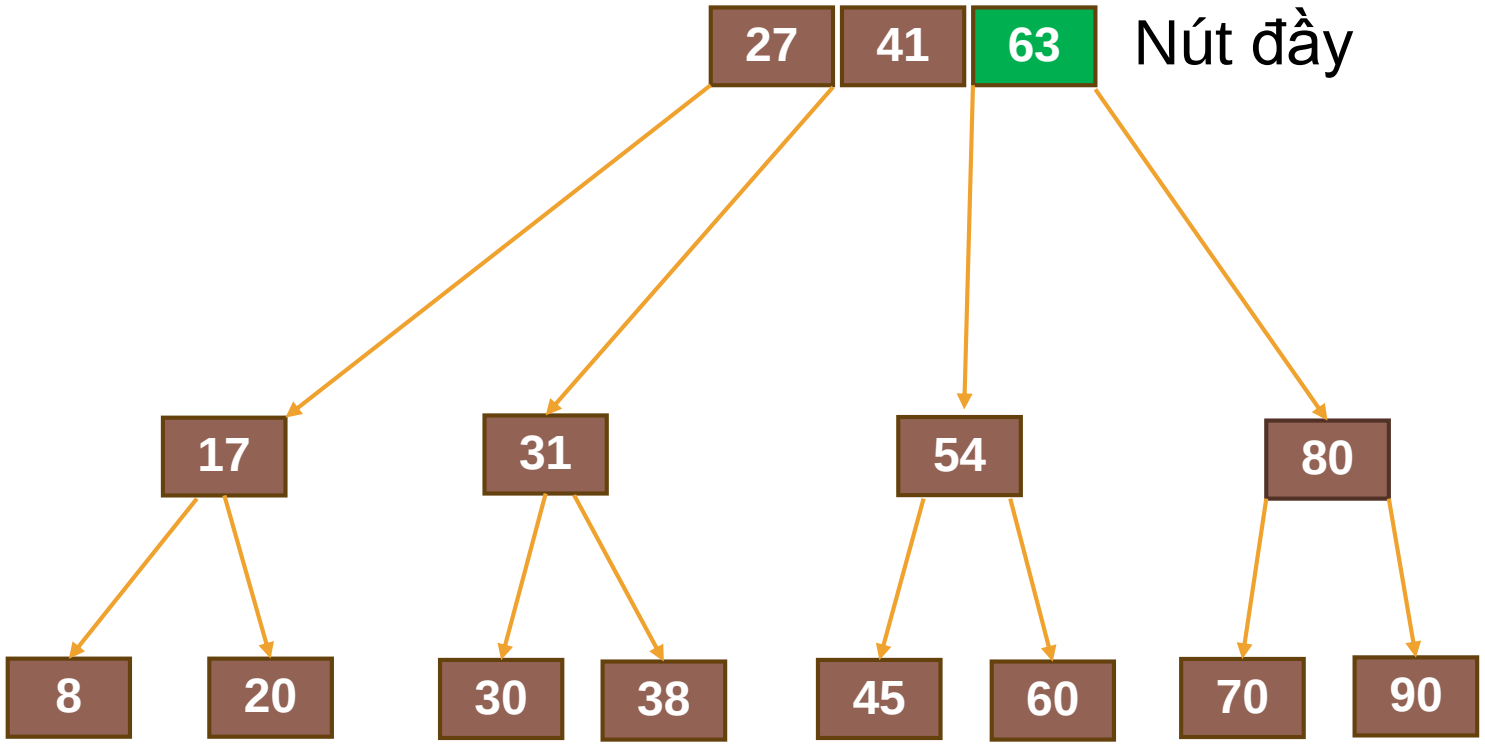
Ví dụ: Chèn nút có khóa 80 vào B-tree sau (bậc 3).



5 B - TREE

Thuật toán chèn khóa vào B-tree

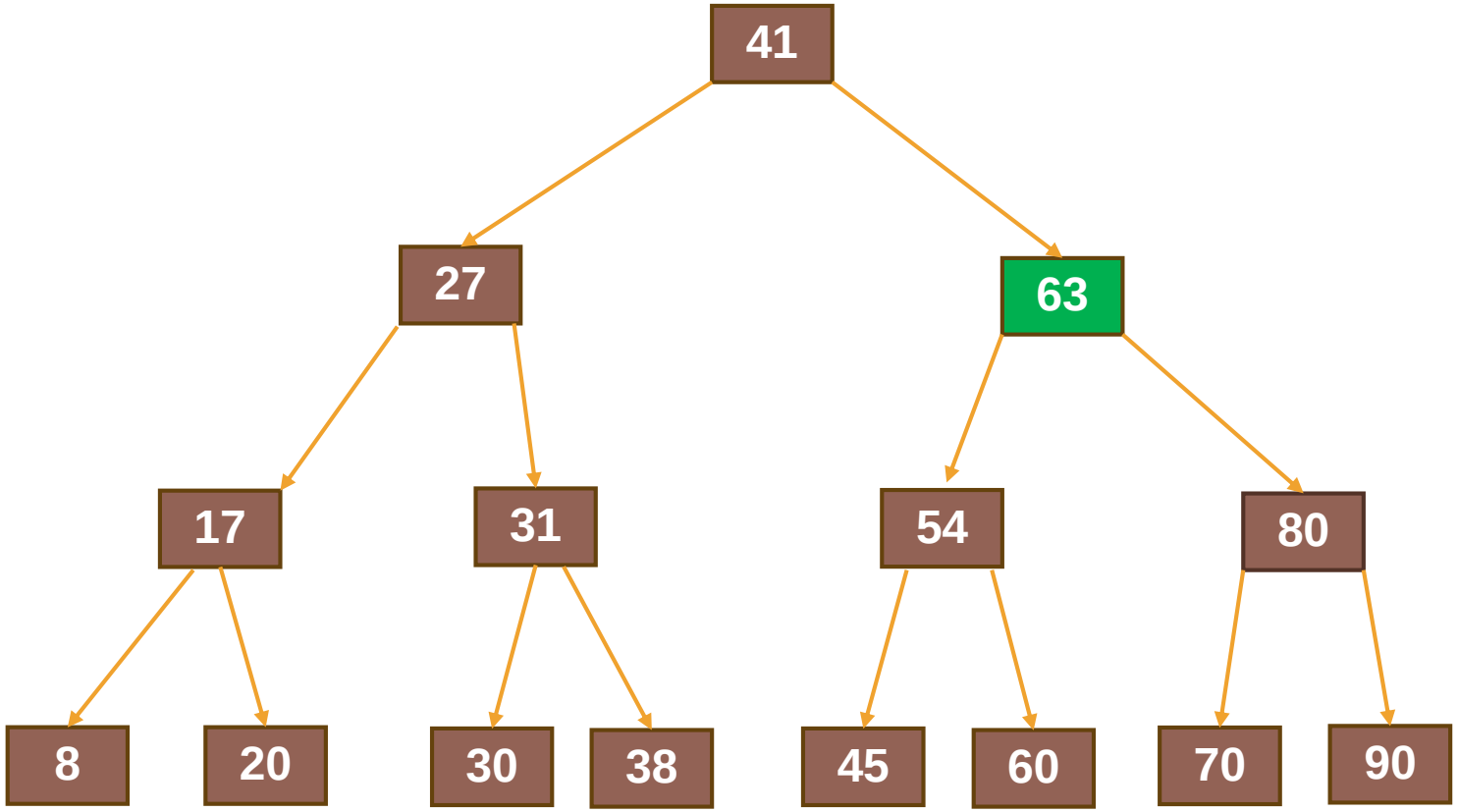
Ví dụ: Chèn nút có khóa 80 vào B-tree sau (bậc 3).



5 B - TREE

Thuật toán chèn khóa vào B-tree

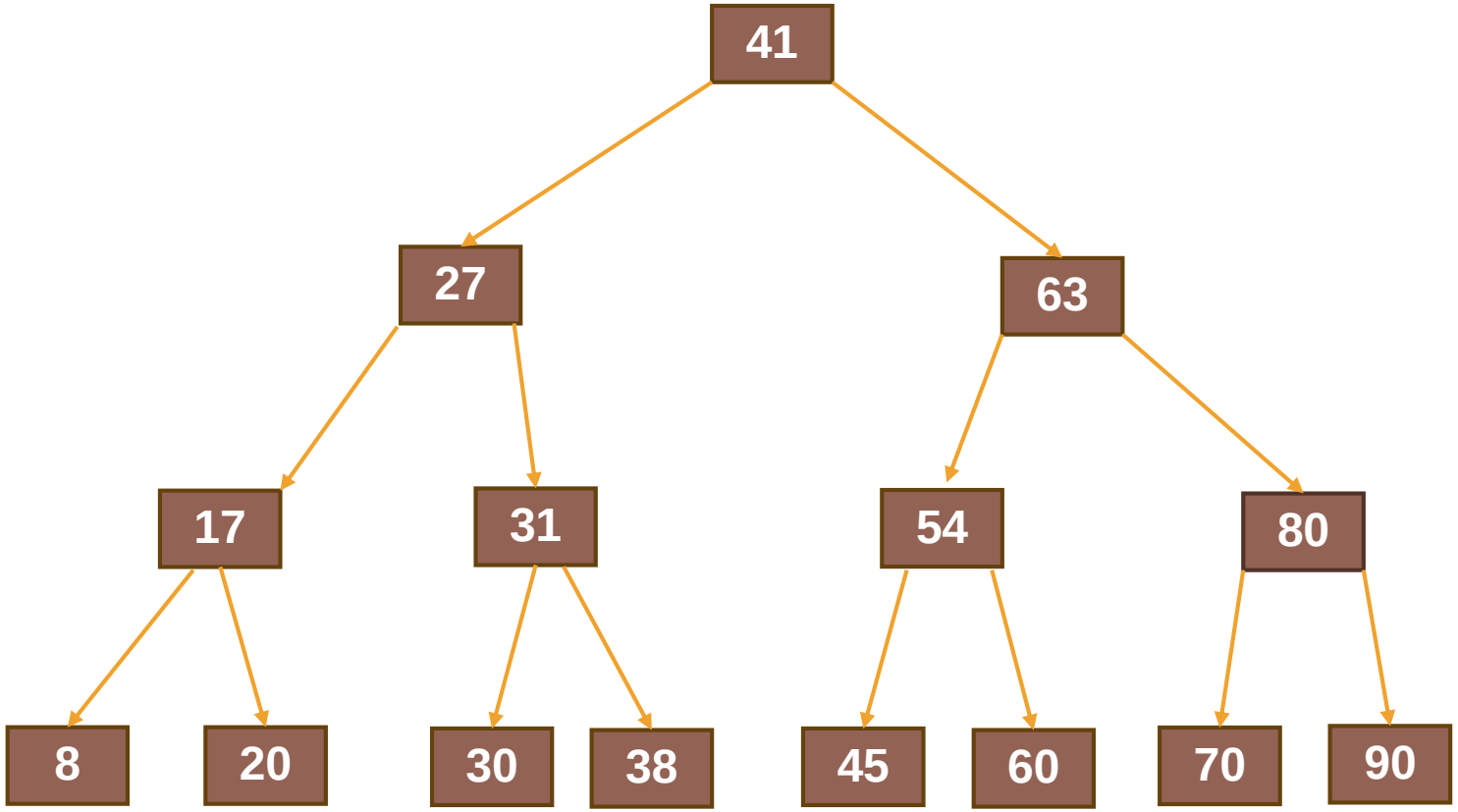
Ví dụ: Chèn nút có khóa 80 vào B-tree sau (bậc 3).



5 B - TREE

Thuật toán chèn khóa vào B-tree

Ví dụ: Chèn nút có khóa 80 vào B-tree sau (bậc 3).



5 B - TREE

Thuật toán xóa khóa khỏi B-tree

Nguyên tắc:

- Không vi phạm số khóa tối thiểu.

- Luôn đảm bảo tính cân bằng.

Xóa khóa trong B-tree là thao tác phức tạp nhất, vì nó có thể làm mất cân bằng cây. Có hai trường hợp chính:

- Trường hợp 1: Xóa khóa tại nút lá
- Trường hợp 2: Xóa khóa tại nút không phải lá

5 B - TREE

Thuật toán xóa khóa khỏi B-tree

Trường hợp 1: Xóa khóa tại nút lá:

Bước 1: Tìm và xóa khóa.

Bước 2: Kiểm tra số khóa còn lại trong nút.

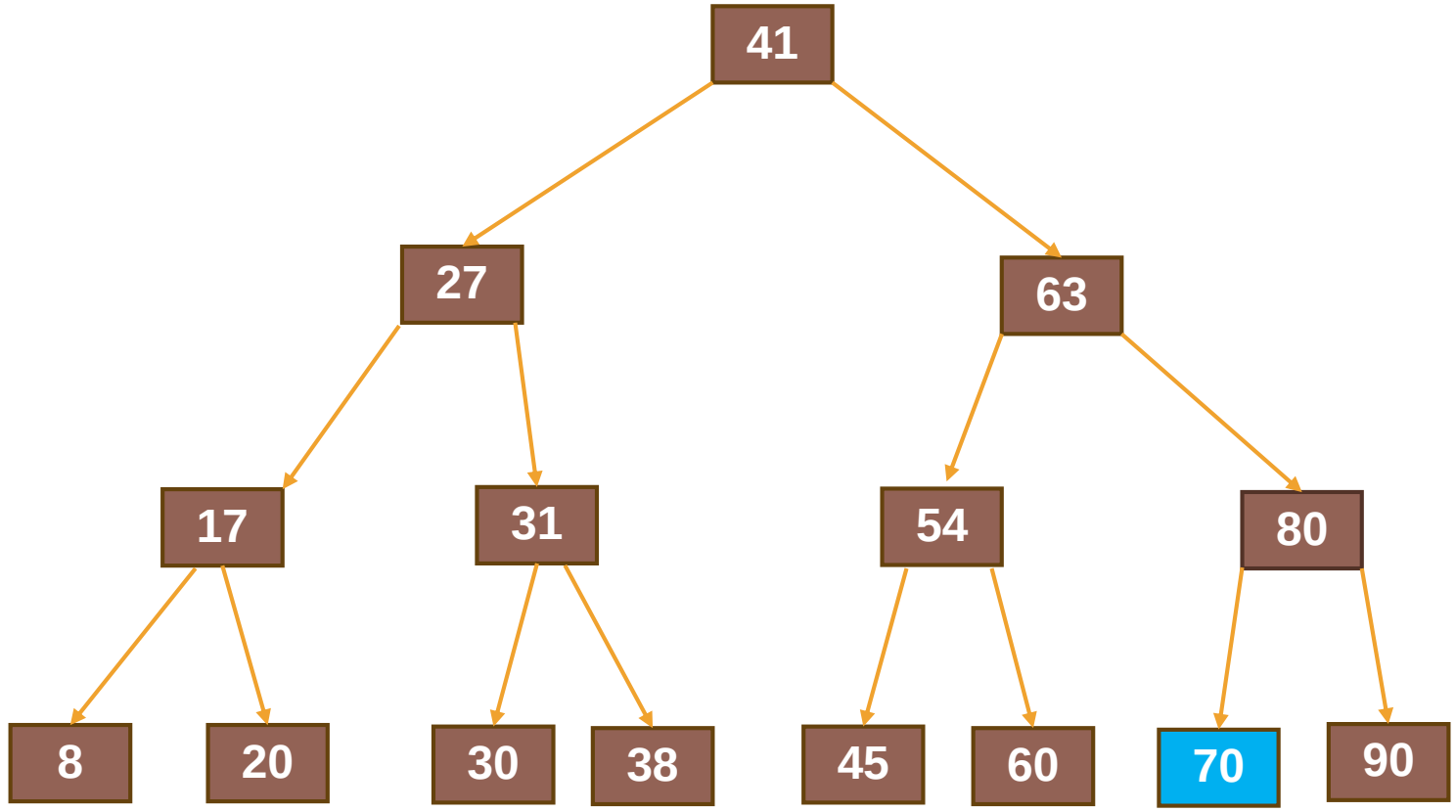
- Nếu nút vẫn có đủ số khóa tối thiểu ($\geq(m/2)-1$), xong.
- Nếu nút có ít khóa hơn mức tối thiểu, ta phải thực hiện **tái cân bằng**:
  - **Mượn (borrow)**: Nếu một nút anh em gần kề có thừa khóa, ta mượn một khóa từ nó và di chuyển một khóa từ nút cha xuống để cân bằng.
  - **Hợp nhất (merge)**: Nếu các nút anh em cũng chỉ có số khóa tối thiểu, ta hợp nhất nút hiện tại, nút anh em và một khóa từ nút cha thành một nút duy nhất. Thao tác này có thể tiếp tục "lan truyền" lên trên nếu nút cha cũng bị thiếu khóa.

5 B - TREE

Thuật toán xóa khóa khỏi B-tree

Trường hợp 1: Xóa khóa tại nút lá:

Ví dụ: Xóa nút có khóa 70 khỏi B-tree sau (bậc 3).

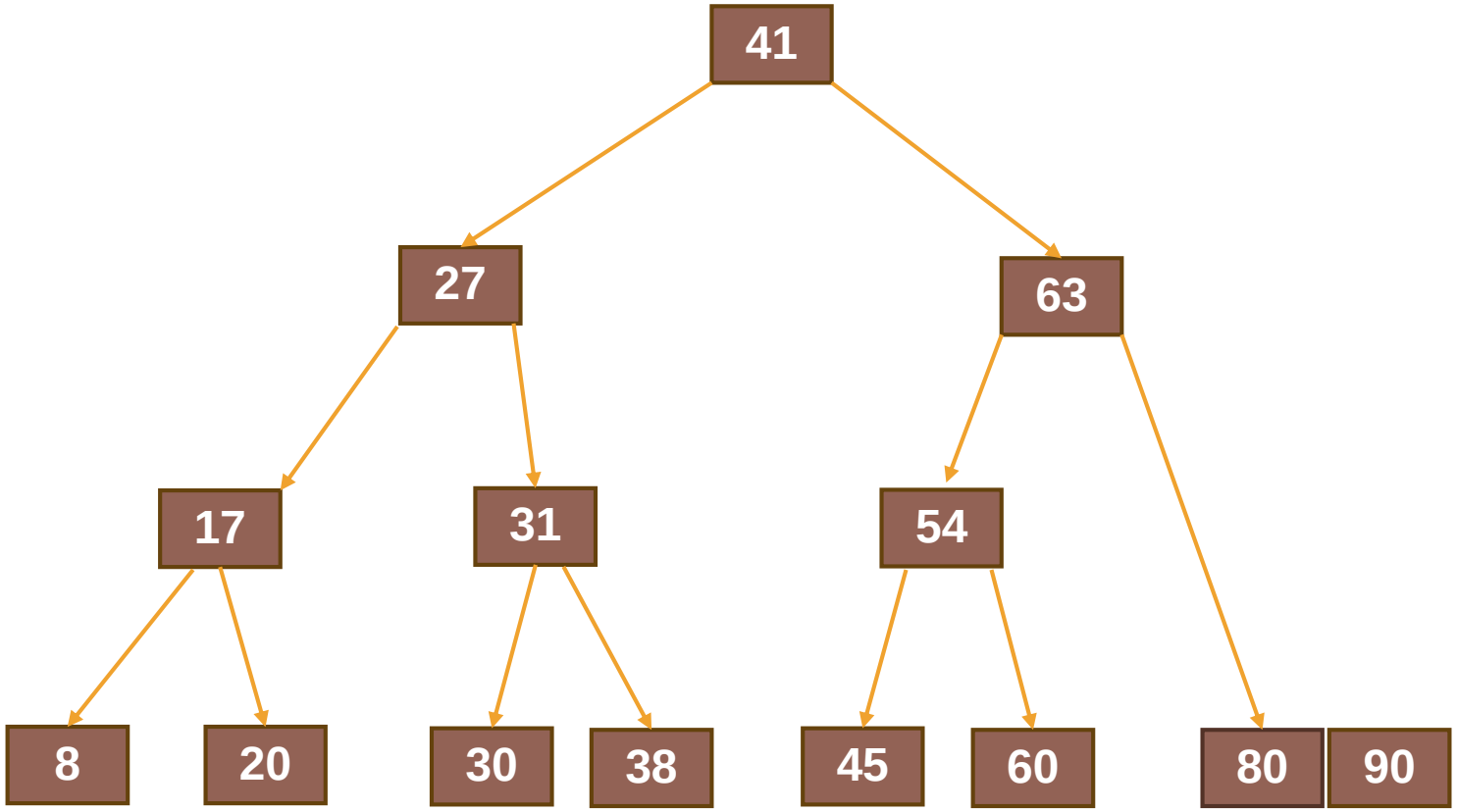


5 B - TREE

Thuật toán xóa khóa khỏi B-tree

Trường hợp 1: Xóa khóa tại nút lá:

Ví dụ: Xóa nút có khóa 70 khỏi B-tree sau (bậc 3).

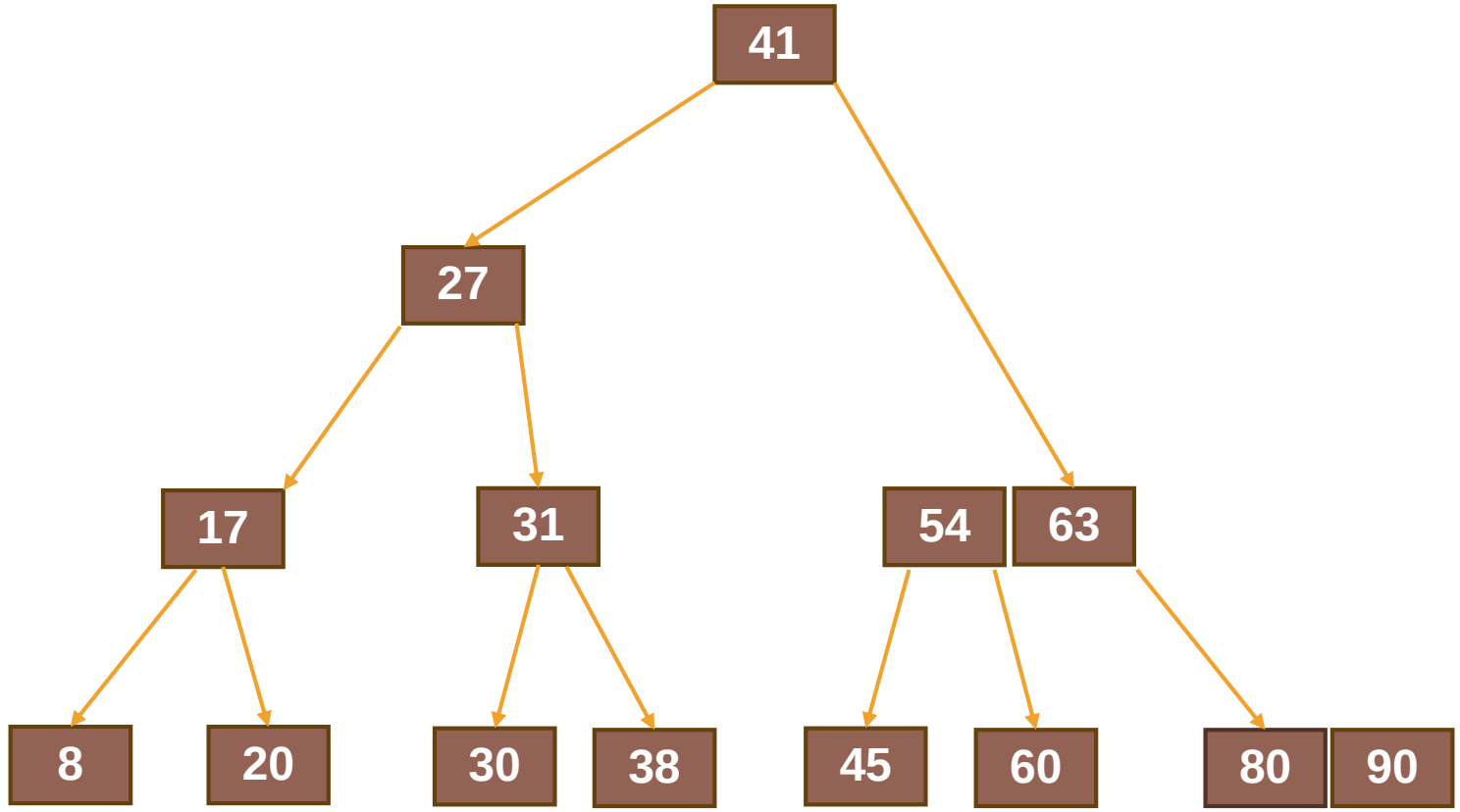


5 B - TREE

Thuật toán xóa khóa khỏi B-tree

Trường hợp 1: Xóa khóa tại nút lá:

Ví dụ: Xóa nút có khóa 70 khỏi B-tree sau (bậc 3).

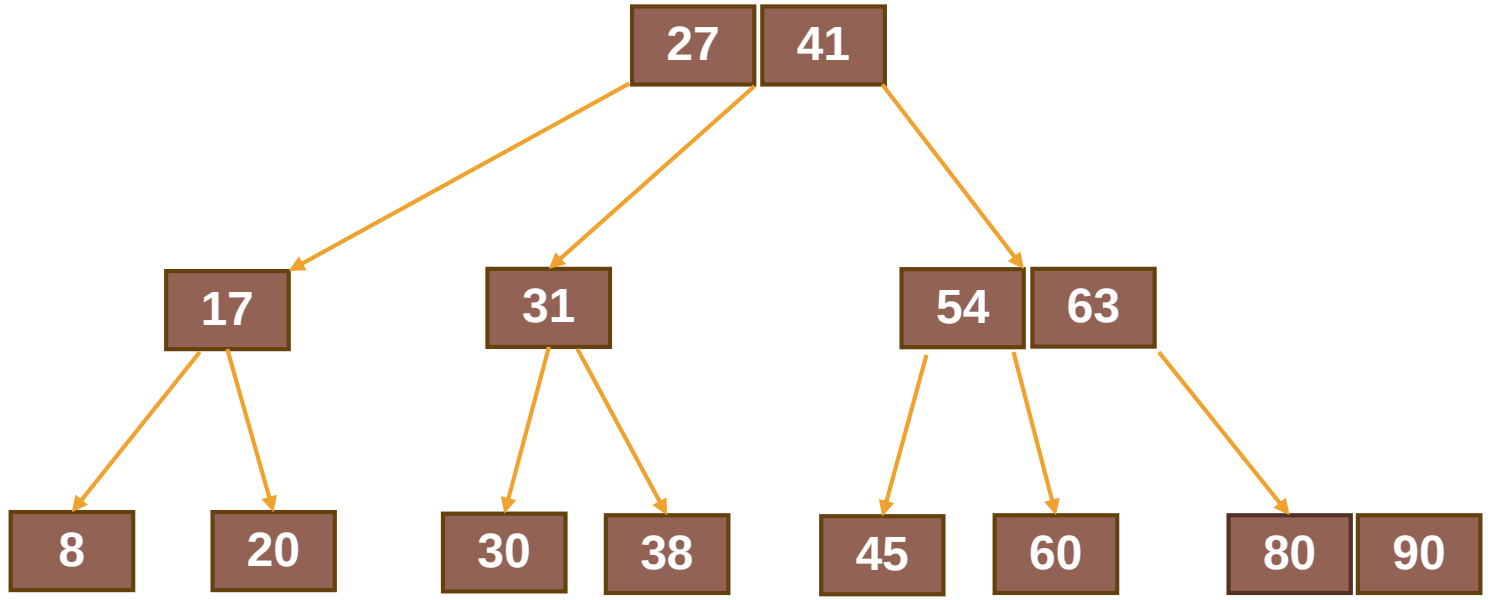


5 B - TREE

Thuật toán xóa khóa khỏi B-tree

Trường hợp 1: Xóa khóa tại nút lá:

Ví dụ: Xóa nút có khóa 70 khỏi B-tree sau (bậc 3).



5 B - TREE

Thuật toán xóa khóa khỏi B-tree

Trường hợp 2: Xóa khóa tại nút không phải lá

Bước 1: Tìm và xóa khóa.

Bước 2: Thay thế khóa đã xóa bằng **khóa lớn nhất** của cây con bên trái hoặc **khóa nhỏ nhất** của cây con bên phải.

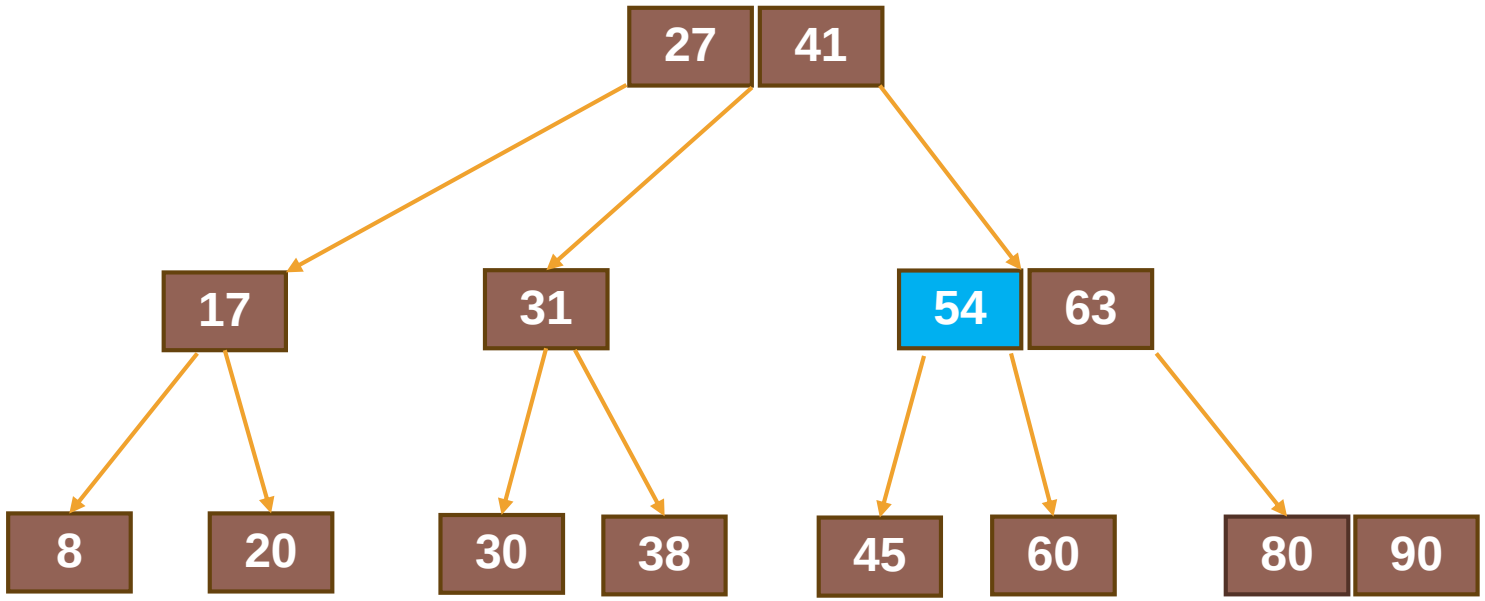
Bước 3: Xóa khóa đã thay thế đó ở nút lá (vị trí ban đầu của nó). Thao tác này sẽ trở về trường hợp 1.

5 B - TREE

Thuật toán xóa khóa khỏi B-tree

Trường hợp 2: Xóa khóa tại nút không phải lá

Ví dụ: Xóa khóa 54 khỏi cây B sau:

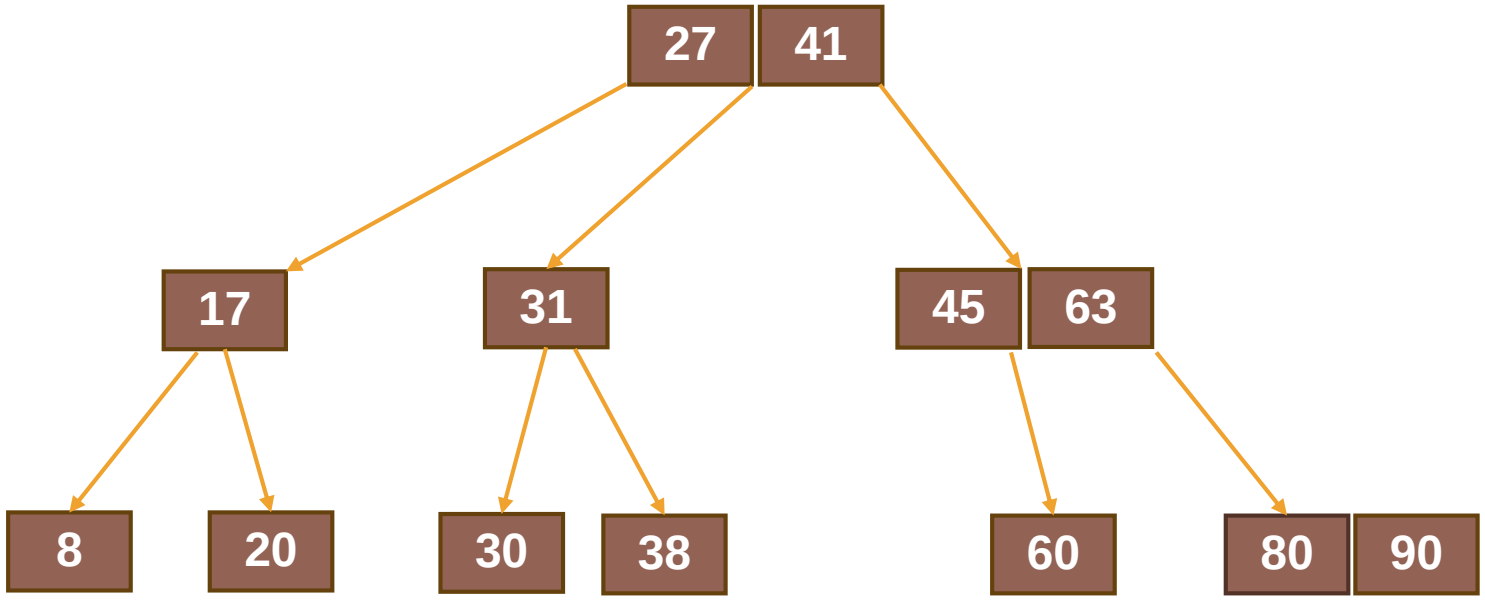


Thuật toán xóa khóa khỏi B-tree

Trường hợp 2: Xóa khóa tại nút không phải lá

Ví dụ: Xóa khóa 54 khỏi cây B sau:

Cách 1:

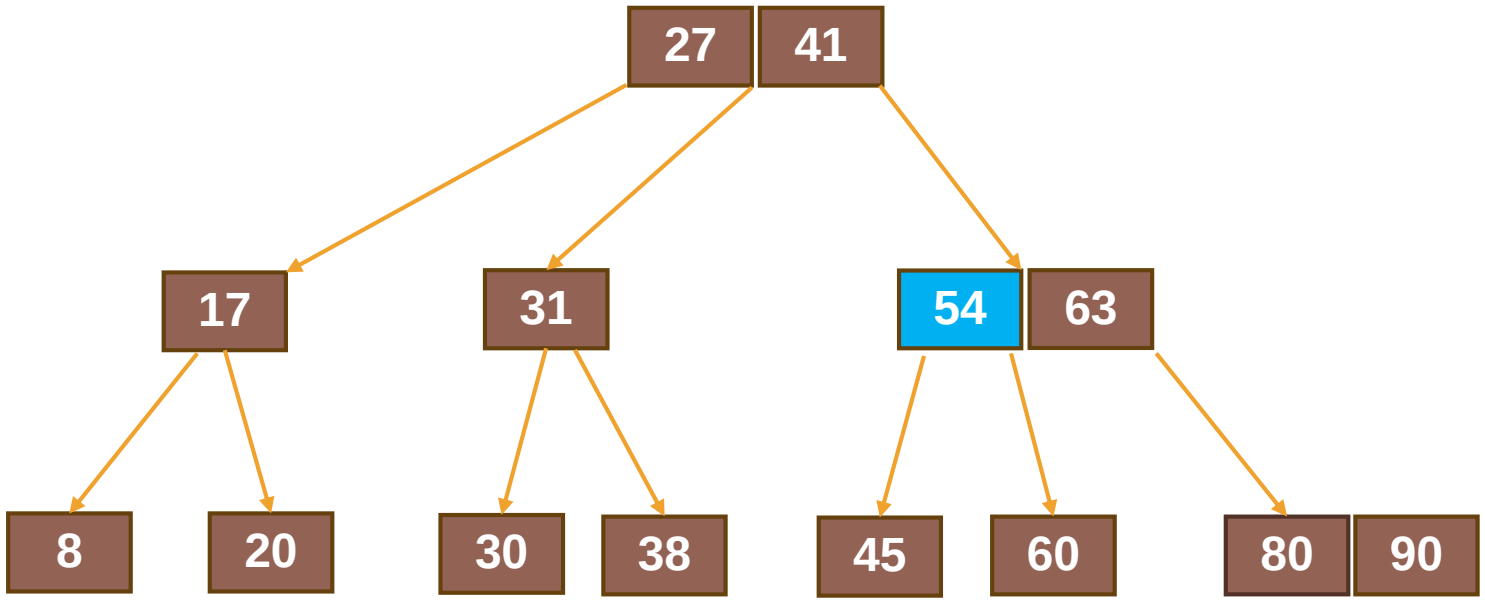


5 B - TREE

Thuật toán xóa khóa khỏi B-tree

Trường hợp 2: Xóa khóa tại nút không phải lá

Ví dụ: Xóa khóa 54 khỏi cây B sau:



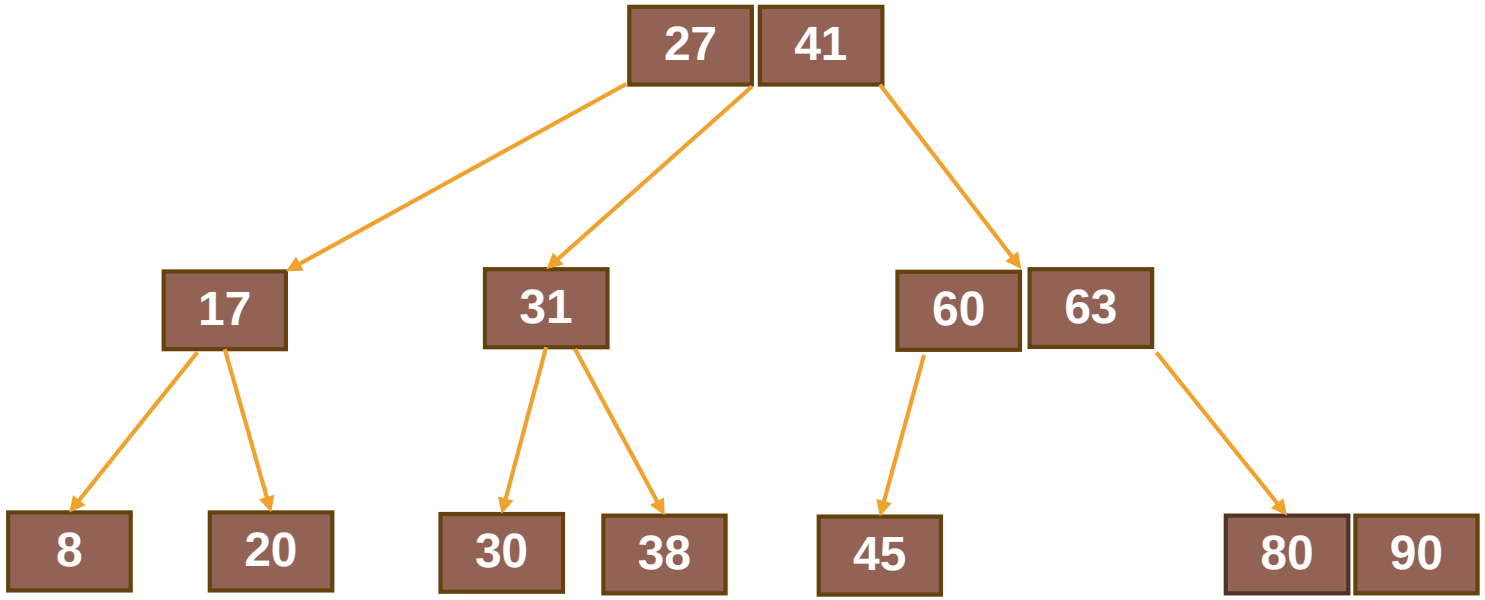
5 B - TREE

Thuật toán xóa khóa khỏi B-tree

Trường hợp 2: Xóa khóa tại nút không phải lá

Ví dụ: Xóa khóa 54 khỏi cây B sau:

Cách 2:



BÀI TẬP CHƯƠNG 5 - B TREE

Bài 1. Cho các số nguyên: 50, 25, 64, 31, 23, 29, 45, 27, 20, 26, 88, 56.

a. Vẽ cây B - Tree bậc 3 từ các khoá số nguyên trên.

b. Vẽ cây B - Tree khi chèn thêm khóa 60, 62.

c. Vẽ cây B - Tree khi xóa khóa 26, 50.

Bài 2. Cài đặt các thao tác: tìm kiếm, chèn, xóa một khóa số nguyên trên cây B- Tree bằng ngôn ngữ lập trình C++.