

CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

B-Tree

Giới thiệu

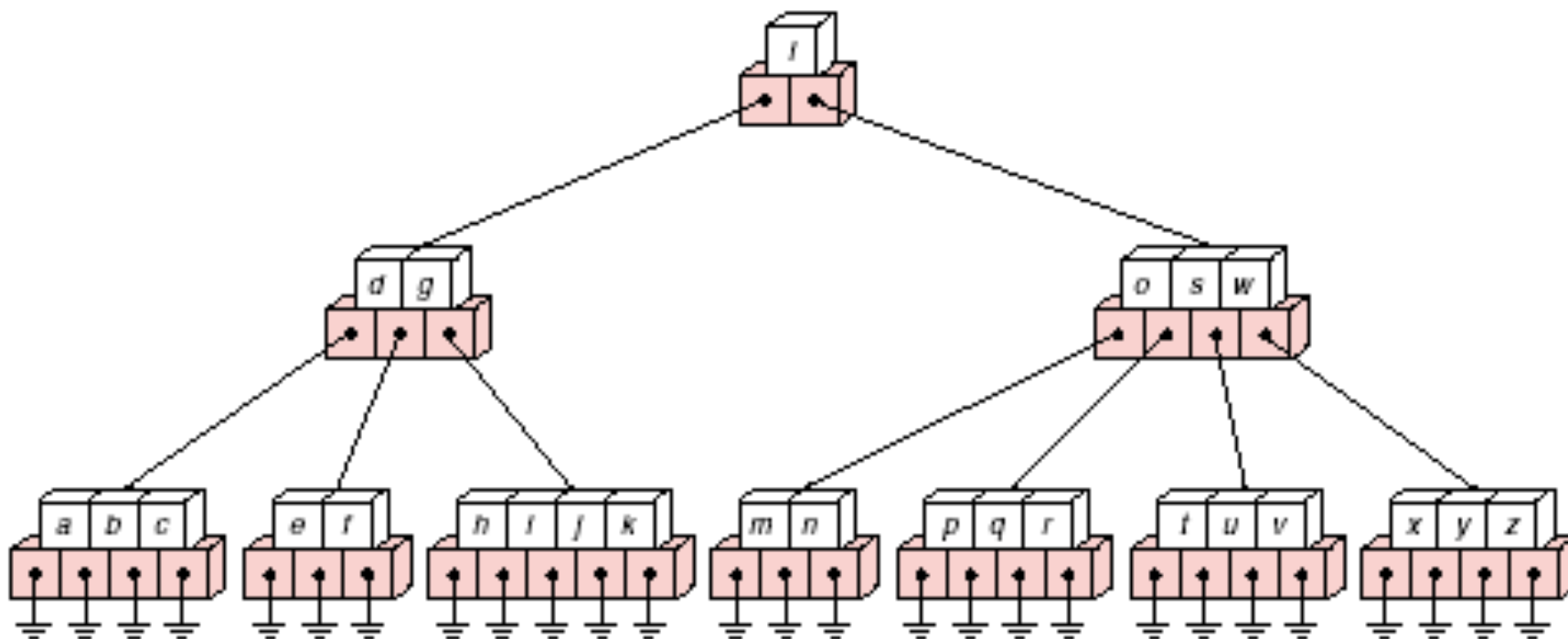
- ▶ Do R.Bayer và E.M.McCreight đưa ra năm 1972.
- ▶ B-tree là cấu trúc dữ liệu phù hợp cho việc lưu trữ và truy xuất dữ liệu trên bộ nhớ ngoài (đĩa cứng)
- ▶ Hạn chế số thao tác đọc mỗi lần tìm kiếm trên cây:
 - ▶ Mỗi lần truy xuất đọc toàn bộ dữ liệu trong 1 node (mỗi node chứa M phần tử (items))
 - ▶ Sử dụng thuật toán tìm nhị phân để tìm phần tử x (giá trị cần tìm).
- ▶ Chiều cao cây $\log_M N$, tăng M chiều cao cây giảm rất nhanh

Định nghĩa

Một B-tree bậc m là cây m nhánh tìm kiếm thỏa các điều kiện sau:

Cho số tự nhiên $k > 0$, B-Trees bậc m với $m = 2k+1$ là một cây thỏa mãn các tính chất:

- ▶ Tất cả node lá nằm trên cùng một mức.
- ▶ Tất cả các node, trừ node gốc và node lá, có ***tối thiểu* $k+1$ node con.**
- ▶ Tất cả các node có ***tối đa* m con.**
- ▶ Tất cả các node, trừ node gốc, có từ k cho đến $m - 1$ khóa (keys). Node gốc có từ 1 đến $m-1$ khóa.
- ▶ Một node không phải lá và có n khóa thì phải có $n+1$ node con.
- ▶ Tại mỗi khóa X bất kỳ, các khóa ở nhánh trái $< X <$ các khóa ở nhánh phải



B-Tree bậc 5 có 3 mức

Khai báo cấu trúc

```
typedef struct
```

```
{
```

```
    int count;                // số khoá của node hiện hành
```

```
    int Key[m-1];            // mảng lưu trữ các khoá của node
```

```
    int *Branch[m];          /* các con trỏ chỉ đến các cây con, m-Bậc của cây*/
```

```
} BNode;                    // Kiểu dữ liệu của node
```

```
typedef struct BNode *pBNode; // con trỏ node
```

```
pBNode *Root;               // con trỏ node gốc
```

Phép tìm kiếm

*Xét trong hình trên, khoá cần tìm là **X**. Giả sử nội dung của node nằm trong bộ nhớ. Với m đủ lớn ta sử dụng phương pháp tìm kiếm nhị phân, nếu m nhỏ ta sử dụng phương pháp tìm kiếm tuần tự. Nếu X không tìm thấy sẽ có 3 trường hợp sau xảy ra:*

- ▶ $K_i < X < K_{i+1}$ Tiếp tục tìm kiếm trên cây con C_i
- ▶ $K_m < X$. Tiếp tục tìm kiếm trên C_m
- ▶ $X < K_1$. tiếp tục tìm kiếm trên C_0

Quá trình này tiếp tục cho đến khi node được tìm thấy. Nếu đã đi đến node lá mà vẫn không tìm thấy khoá, việc tìm kiếm là thất bại.

Phép tìm kiếm

Cài đặt phép toán `nodesearch` :

Trả về vị trí nhỏ nhất của khóa trong nút `current` bắt đầu lớn hơn hay bằng `k`. Trường hợp `k` lớn hơn tất cả các khóa trong nút `current` thì trả về vị trí `current` -> `count`

```
int nodesearch (pBNode current, int k)
{
    int i;
    for(i=0;i<current->count &&
        current->key[i] < k; i++);
    return (i);
}
```

Phép tìm kiếm

Tìm khóa k trên B-Tree. Con trỏ current xuất phát từ gốc và đi xuống các nhánh cây con phù hợp để tìm khóa k có trong một nút current hay không.

Nếu có khóa k tại nút current trên cây:

- ▶ Biến **found** trả về giá trị TRUE
- ▶ Hàm **search()** trả về con trỏ chỉ nút current có chứa khóa k
- ▶ Biến **position** trả về vị trí của khóa k có trong nút current này

Tìm kiếm

Nếu không có khóa k trên cây:

- ▶ Lúc này **current=NULL** và q (nút cha của current) chỉ nút lá có thể thêm khóa k vào nút này được.
- ▶ Biến **found** trả về giá trị FALSE
- ▶ Hàm **search()** trả về con trỏ q là nút lá có thêm nút k vào
- ▶ Biến **position** trả về vị trí có thể chèn khóa k vào nút lá q này

Tìm kiếm

pBNode search(int k, int &position, int &found)

```
{
    int i;
    pBNode current, q;
    q = NULL;
    current = Root;
    while (current != NULL)
    {
        i = nodesearch (current, k);
        if(i < current->count && k == current->key[i]) //tim thay
        {
            found = TRUE;
            position = i;      // vi trí tìm thay khoa k
            return (current); // node co chua khoa k
        }
    }
}
```

Tìm kiếm

```
q = current;  
current = current ->Branch[i];  
}
```

*/*Khi thoát khỏi vòng lặp trên là không tìm thấy, lúc này current=NULL, q là node là có thể thêm khóa k vào node này, position là vị trí có thể chèn khóa k*/*

```
found = FALSE;  
position = i;  
return (q); //trả về node lá  
} //end search()
```

Duyệt cây

```
void traverse(pBNode proot)
```

```
{
```

```
    if (proot == NULL)    return;
```

```
    else { /* vòng lặp duyệt nhánh cây con Branch[i] và in khóa  
key[i] của node proot*/
```

```
        for(i = 0; i < proot -> count; i++)
```

```
        {
```

```
            traverse (proot ->Branch[i]);
```

```
            cout<< proot -> key[i];
```

```
        }
```

```
        traverse (proot -> Branch[proot -> count]); //duyet nhánh  
cây con cuối cùng của node proot
```

```
    }
```

```
}
```

Thêm

- ▶ Khóa mới sẽ được thêm vào node lá
 - ▶ Nếu chưa đầy -> thêm hoàn tất.
 - ▶ Nếu đầy -> tách node:
 - ▶ Khóa giữa node được lan truyền ngược lên node cha.
 - ▶ Trong trường hợp đặc biệt lan truyền đến tận gốc của B-Tree.
 - ▶ Nếu node gốc bị đầy, node gốc sẽ bị tách và dẫn đến việc tăng trưởng chiều cao của cây.
 - ▶ Phần còn lại chia thành 2 node cạnh nhau trong cùng 1 mức.

Thêm

Tạo B-Tree bậc 5 từ dãy các khóa sau: **20 40 10 30** 15 35 7 26 18
22 5 42 13 46 27 8 32 38 24 45 25

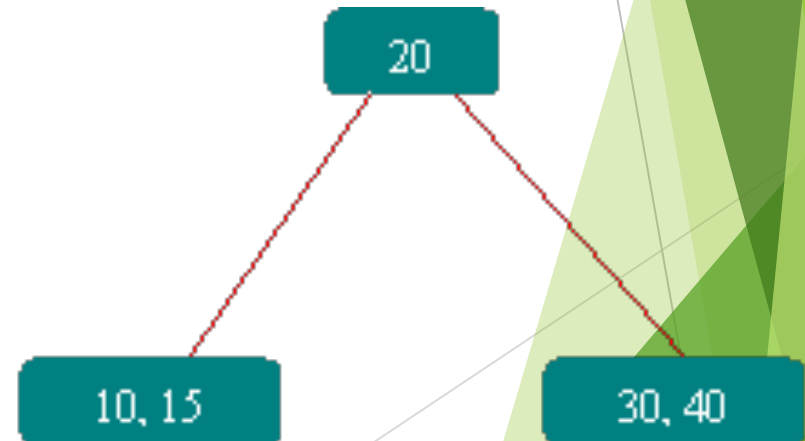
Thêm
20 40
10 30

10, 20, 30, 40



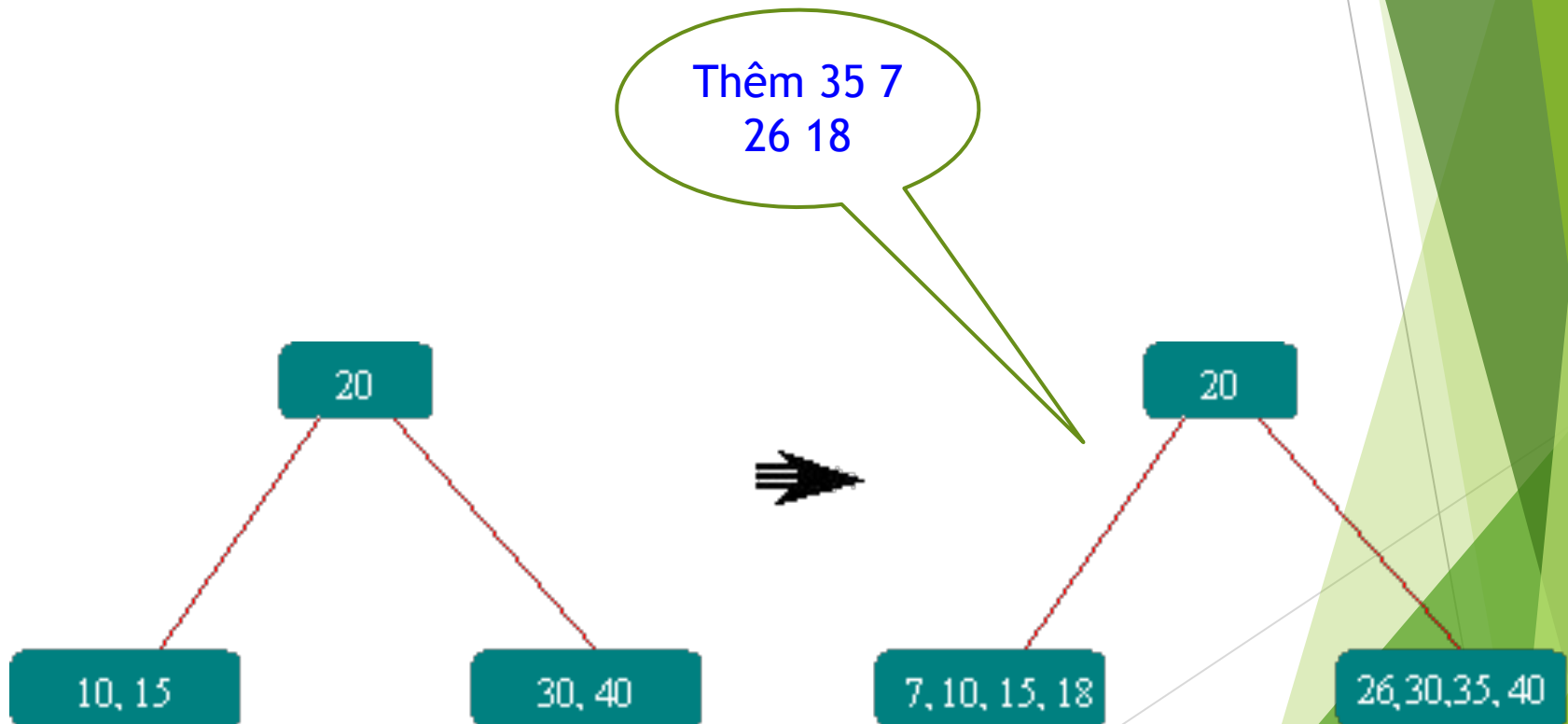
Thêm 15 -> đầy trang -> tách

- Khóa giữa 20 đưa lên node cha
- 10, 15 là node con trái của 20
- 30, 40 là node con phải của 20



Thêm

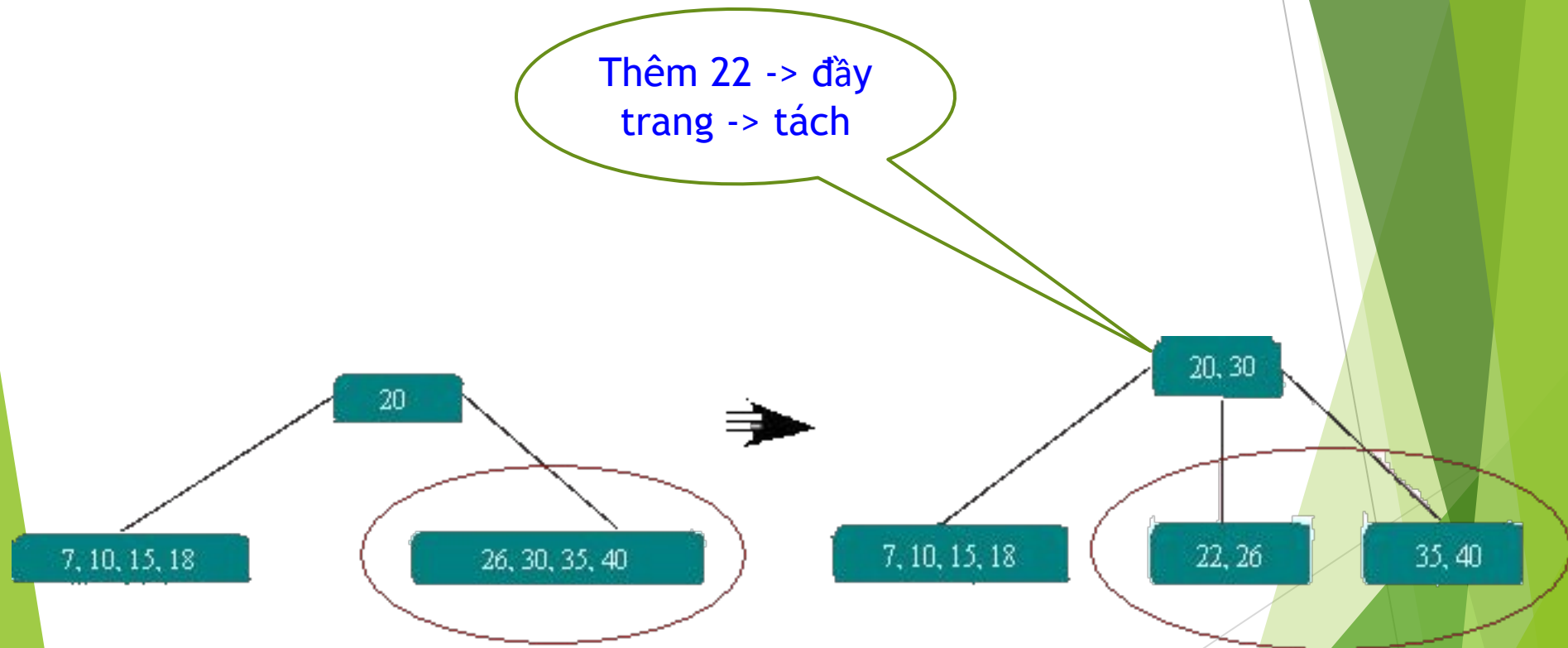
- Tạo B-Tree bậc 5 từ dãy các khóa sau : 20 40 10 30 15 **35 7 26**
18 22 5 42 13 46 27 8 32 38 24 45 25



Thêm

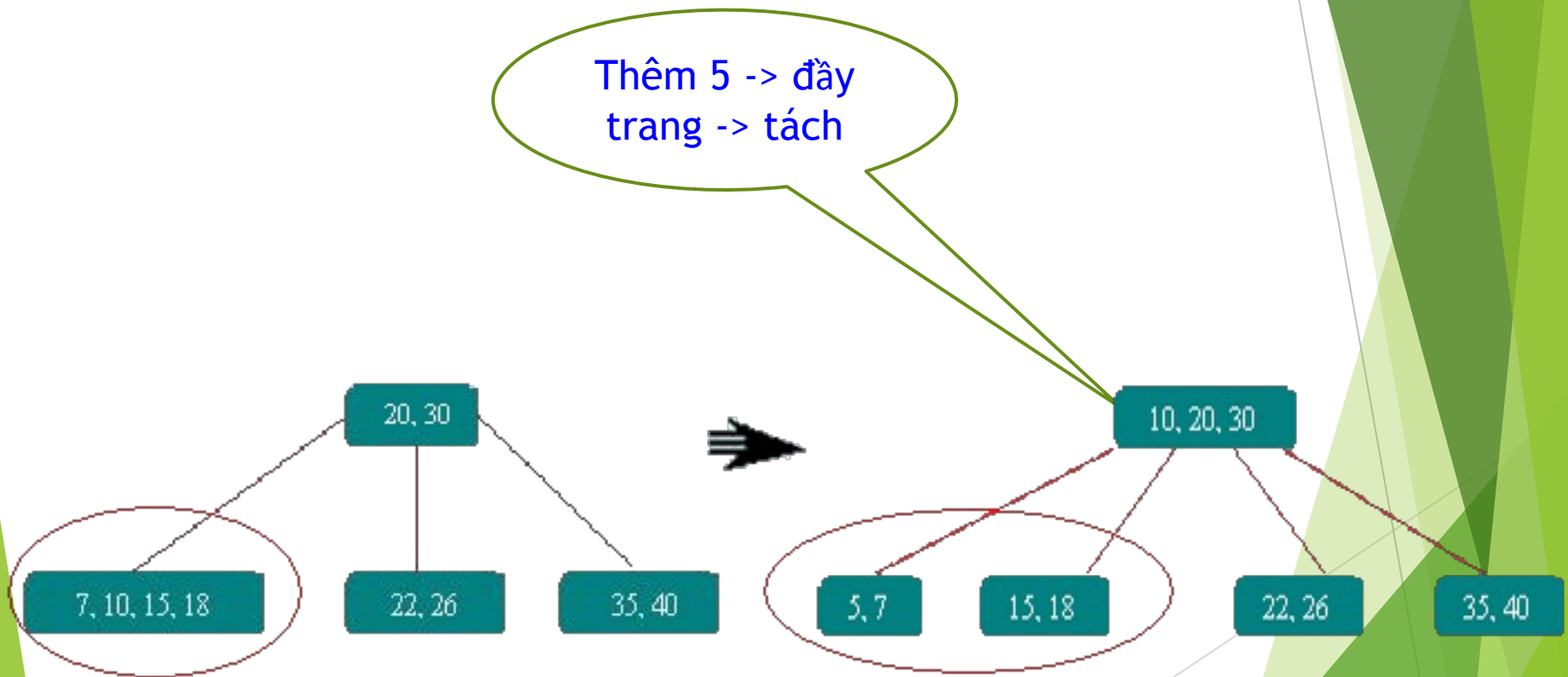
- Tạo B-Tree bậc 5 từ dãy các khóa sau : 20 40 10 30 15 35 7 26 18 **22** 5 42 13 46 27 8 32 38 24 45 25

Thêm 22 -> đầy
trang -> tách



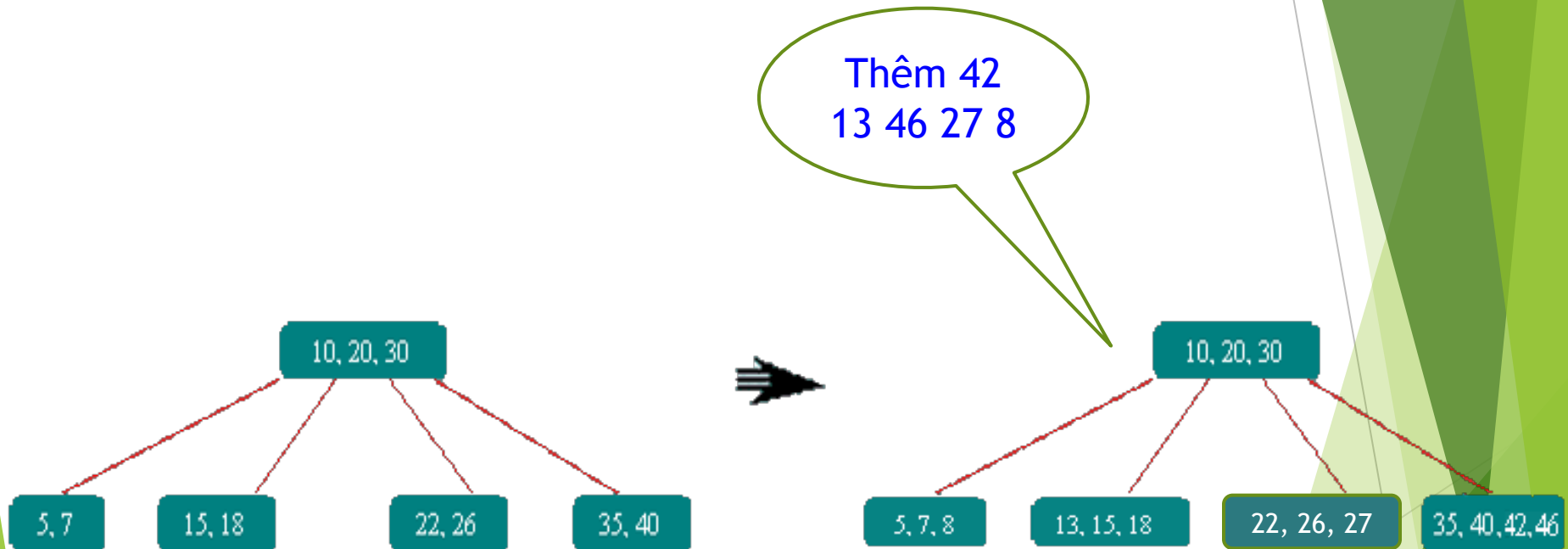
Thêm

- Tạo B-Tree bậc 5 từ dãy các khóa sau : 20 40 10 30 15 35 7 26 18 22 **5** 42 13 46 27 8 32 38 24 45 25



Thêm

- Tạo B-Tree bậc 5 từ dãy các khóa sau : 20 40 10 30 15 35 7 26 18 22 5 **42 13 46 27 8** 32 38 24 45 25



Thêm

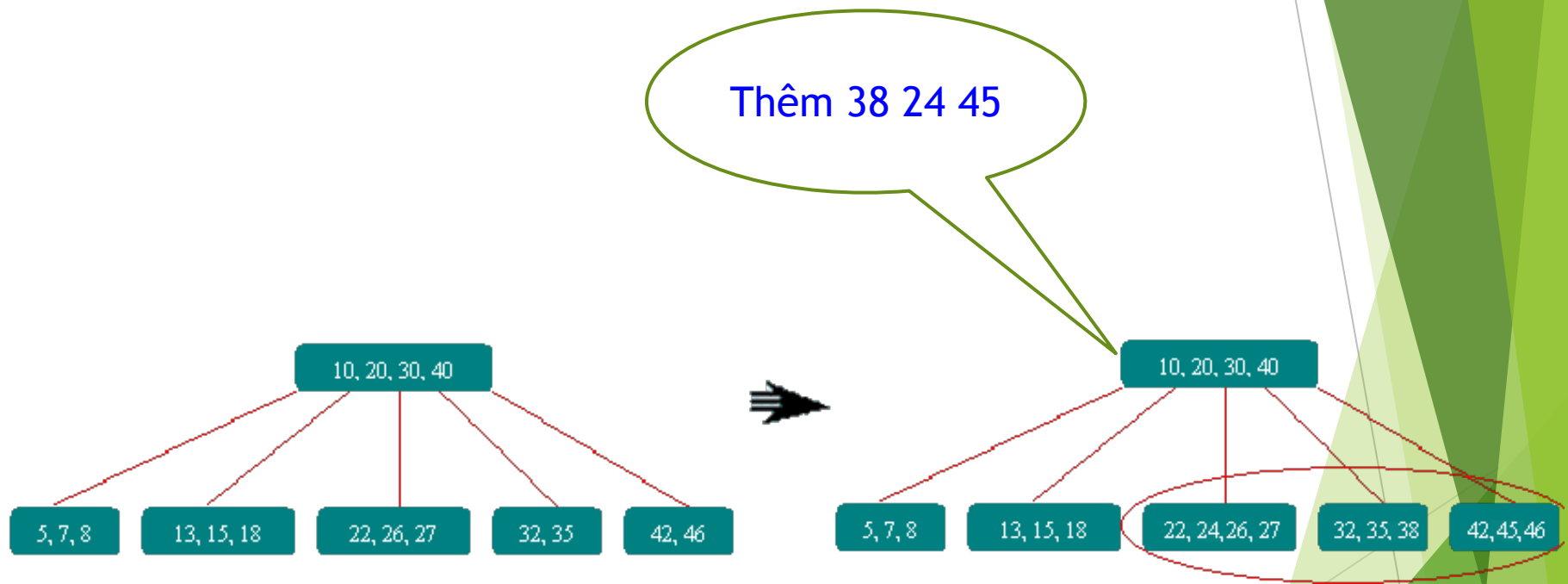
- Tạo B-Tree bậc 5 từ dãy các khóa sau : 20 40 10 30 15 35 7 26 18 22 5 42 13 46 27 8 **32** 38 24 45 25

Thêm 32 -> đầy
trang -> tách



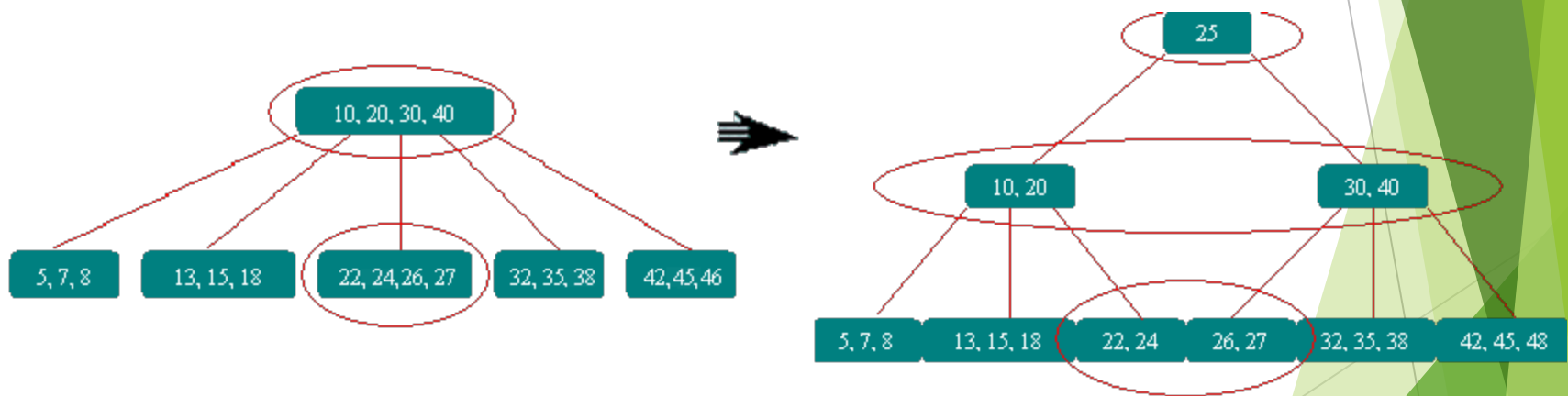
Thêm

- Tạo B-Tree bậc 5 từ dãy các khóa sau : 20 40 10 30 15 35 7 26 18 22 5 42 13 46 27 8 32 **38 24 45** 25



Thêm

- ▶ Tạo B-Tree bậc 5 từ dãy các khóa sau : 20 40 10 30 15 35 7 26 18 22 5 42 13 46 27 8 32 38 24 45 **25**
- ▶ Thêm 25 vào node (22, 24 26, 27) làm node này bị đầy -> tách và 25 được đưa lên node cha. Nút (10, 20, 30, 40) được tách thành 2 node.

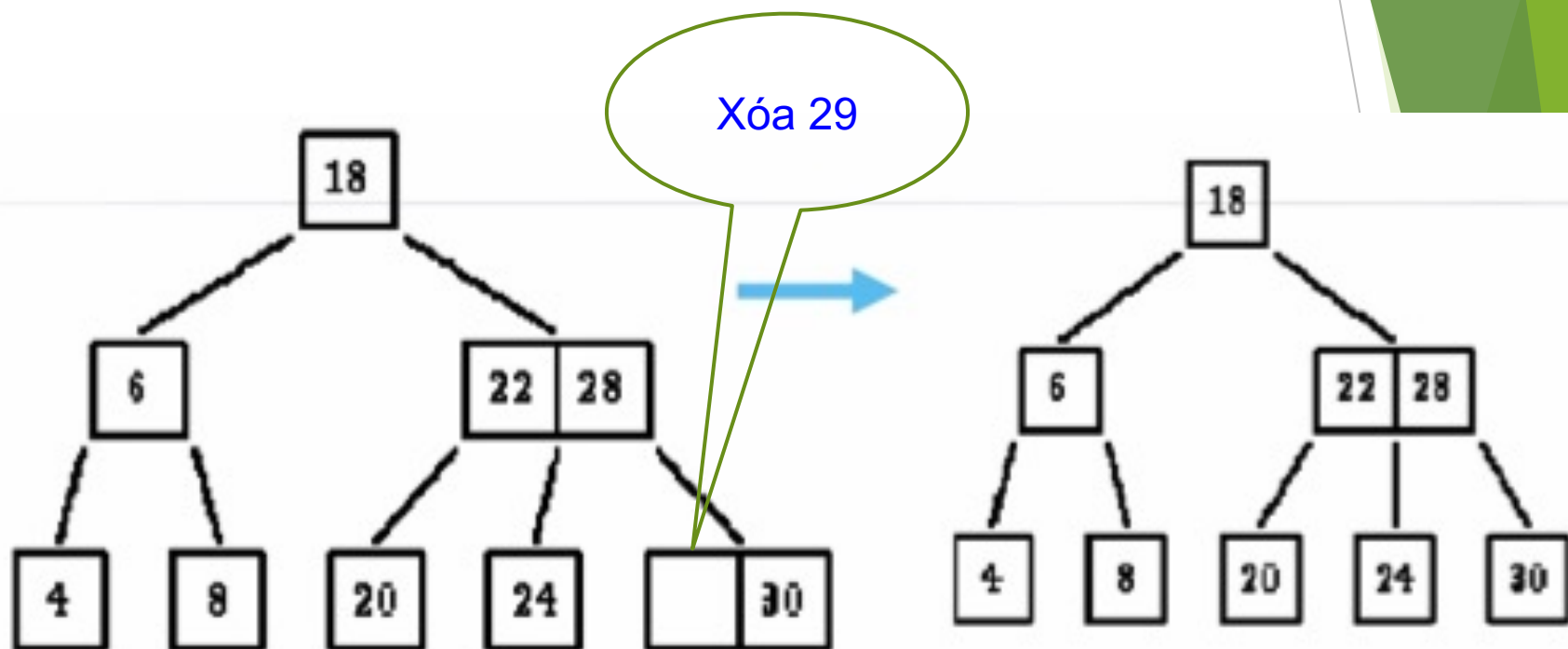


Xóa

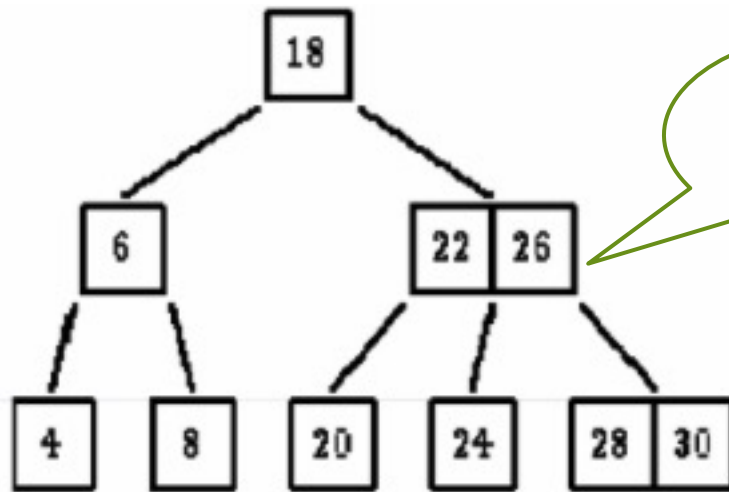
- ▶ Khóa cần xóa nằm trên node lá -> Xóa bình thường
- ▶ Khóa cần xóa không trên node lá:
 - ▶ Tìm phần tử thay thế: trái nhất của cây con bên phải hoặc phải nhất của cây con bên trái.

Xoá

- Xoá 1 khoá trên trang lá (B-Tree bậc 3)

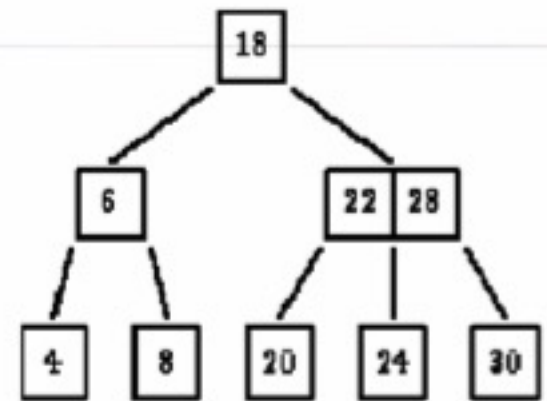
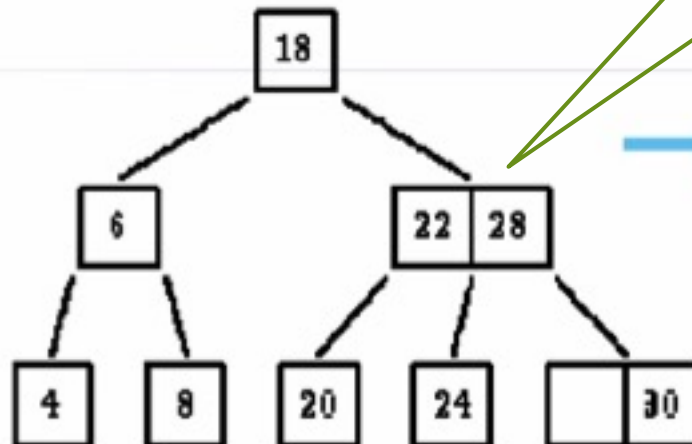


Xoá



Xoá 26,
không phải
trên trang
lá

Phần tử thay
thế 28: trái
nhất trên cây
con phải

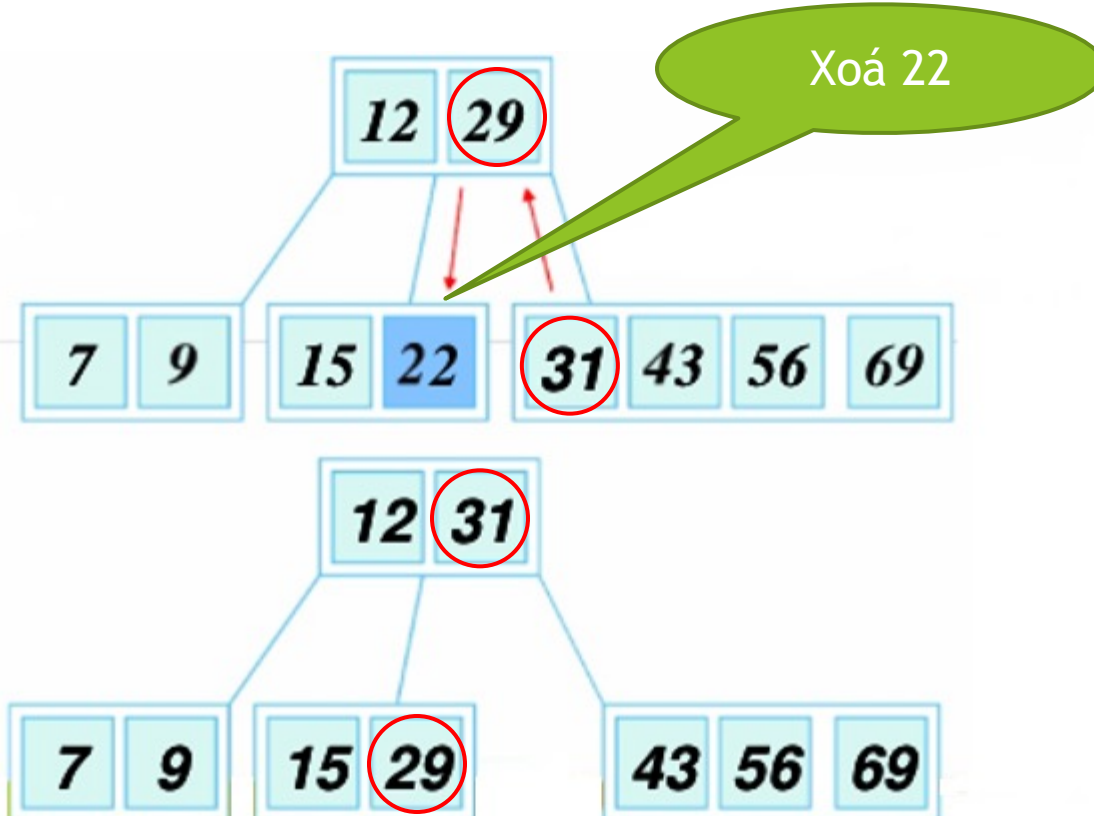


Cân bằng cây sau khi xóa

- ▶ Sau khi xóa, node bị thiếu (vi phạm điều kiện B-tree):
 - ▶ Hoặc chuyển dời phần tử từ node thừa
 - ▶ Hoặc ghép với node bên cạnh (trái/phải)

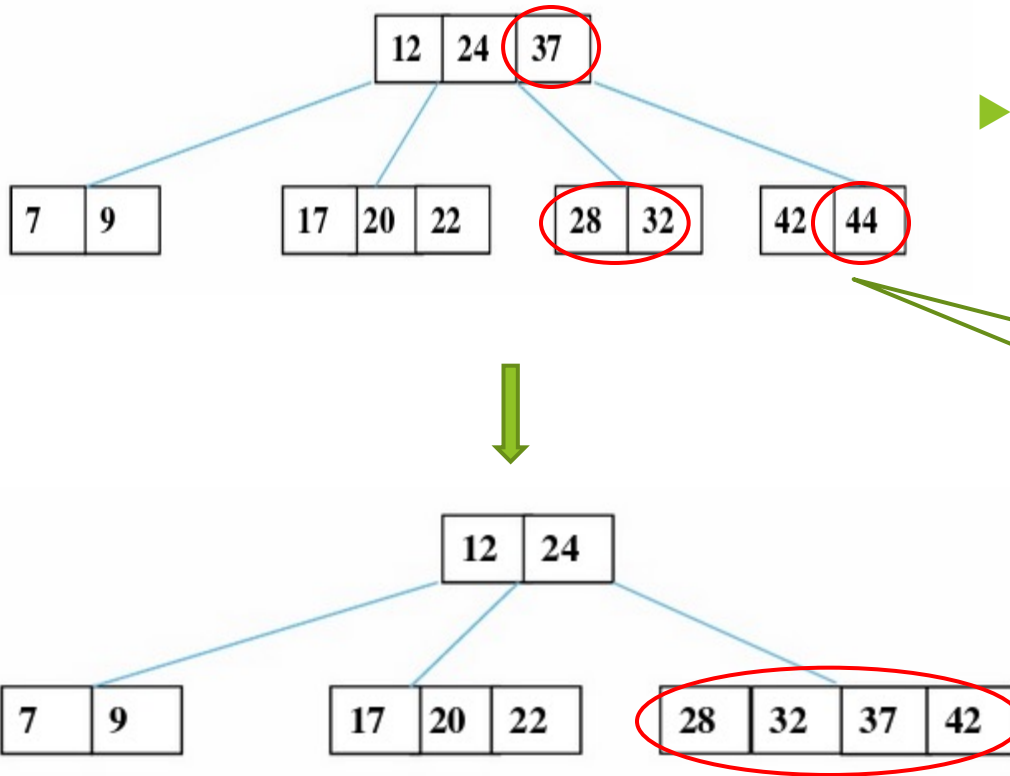
Cân bằng cây sau khi xoá

- ▶ Nếu một trong các **nút kế cận** nút đang xét (nút chứa khoá cần xoá) có số lượng khoá nhiều hơn số lượng tối thiểu:
 - ▶ Đưa 1 khoá của **nút kế cận** lên nút cha
 - ▶ Đưa 1 khoá ở nút cha xuống nút đang xét (thay thế khoá cần xoá)



Cân bằng cây sau khi xoá

- ▶ Tất cả nút kế cận nút đang xét có **số lượng khoá vừa đủ**:
 - ▶ Chọn 1 nút kế cận để **hợp nhất** với nút đang xét và khoá tương ứng ở nút cha.
 - ▶ Nếu nút cha thiếu khoá, lặp lại quá trình này.



Xoá 44: hợp
nhất 42, 37, 28
32

Cân bằng cây sau khi xóa

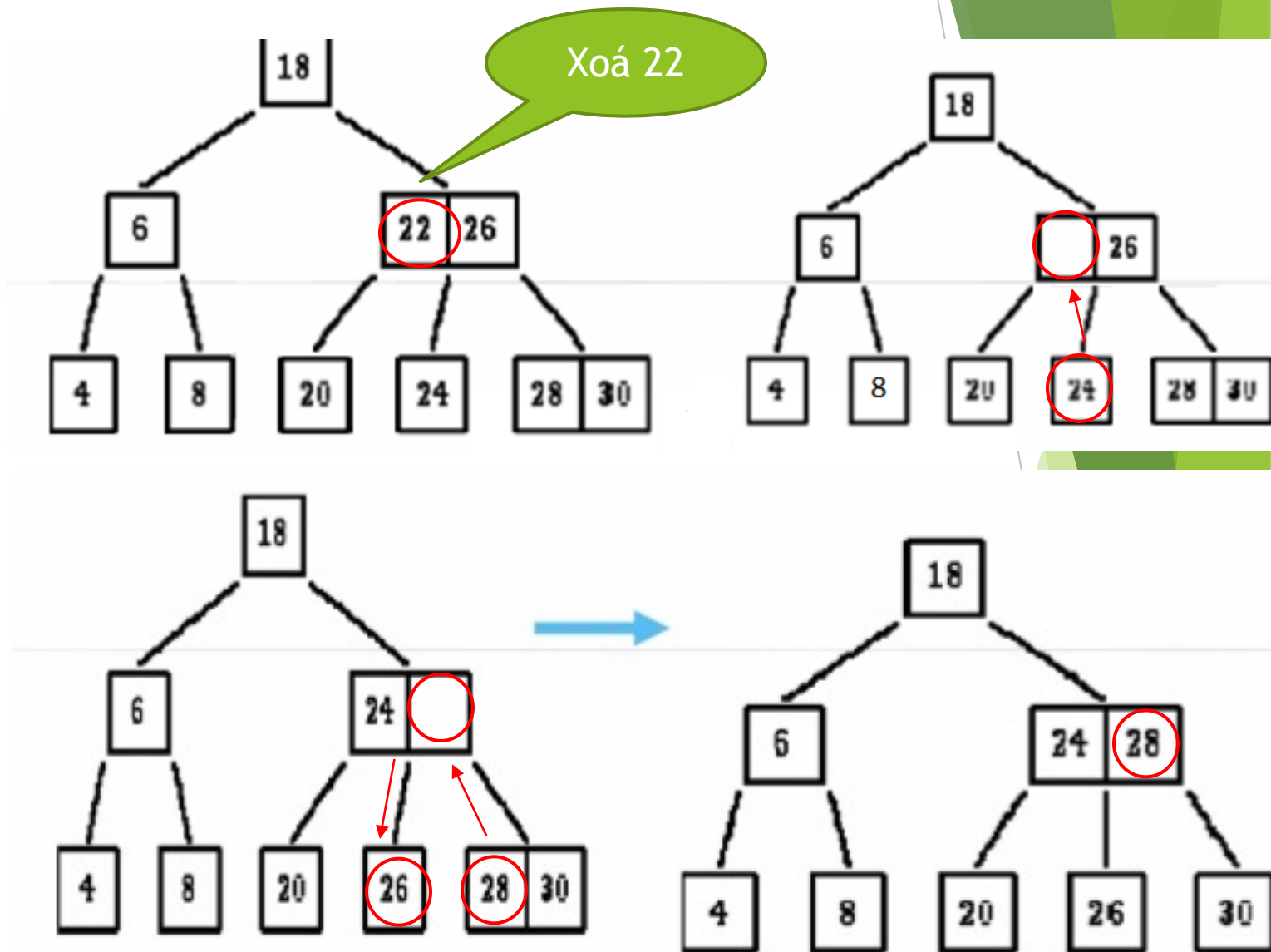
► Xóa 22 (b-tree bậc 3):

- Nút thay thế là 24: trái nhất của nhánh phải

→ thiếu lá.

- Đưa 1 khoá của nút kế cận lên nút cha

- Đưa 1 khoá ở nút cha xuống nút thiếu

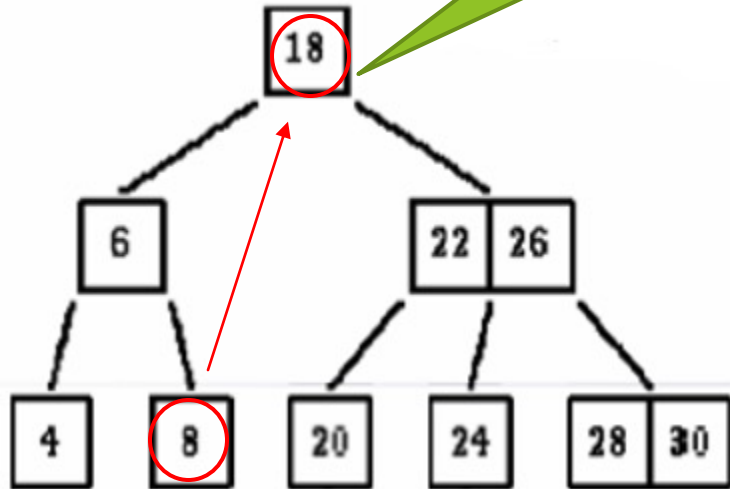


Cân bằng cây sau khi xoá

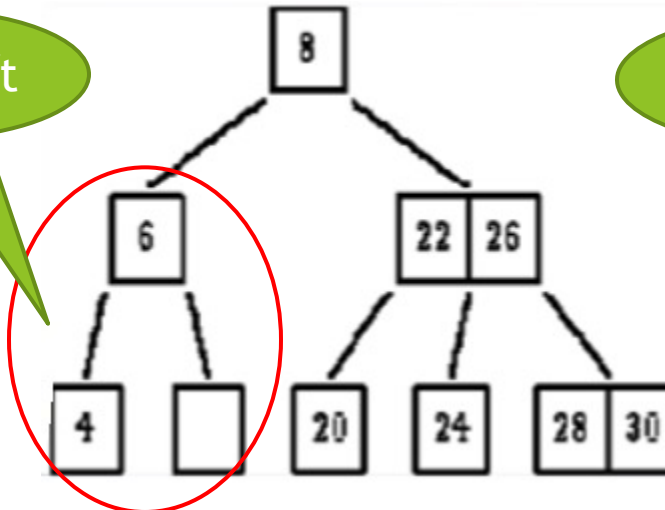
Xoá 18 đưa 8 lên thế

► Xoá 18:

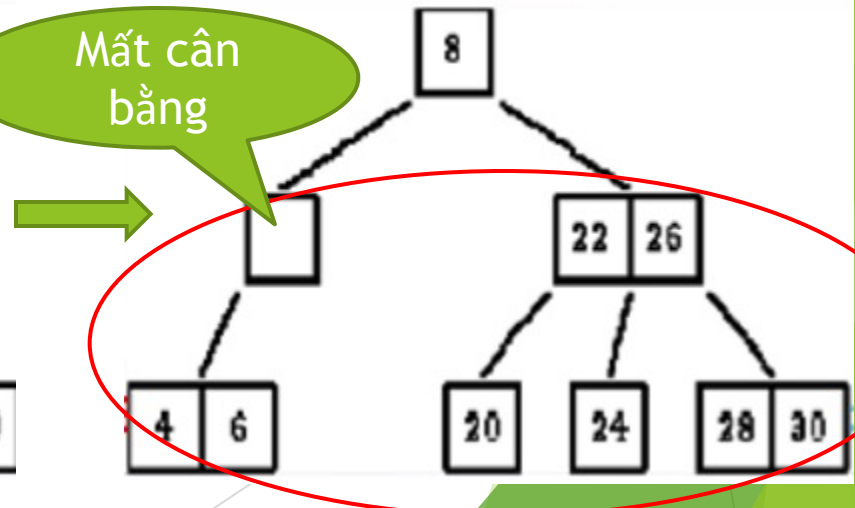
- **Đưa 8 lên thay** (phải nhất của cây con trái) -> thiếu lá
- Các nút có số lượng khóa vừa đủ: hợp nhất nút kế cận (4), nút đang xét (null) và khóa tương ứng ở nút cha (6) → cây mất cân bằng



Hợp nhất



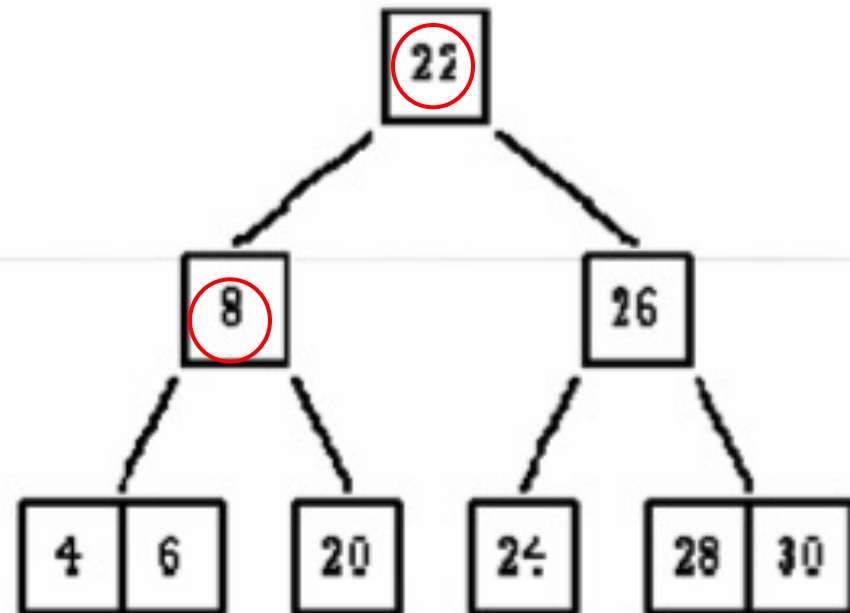
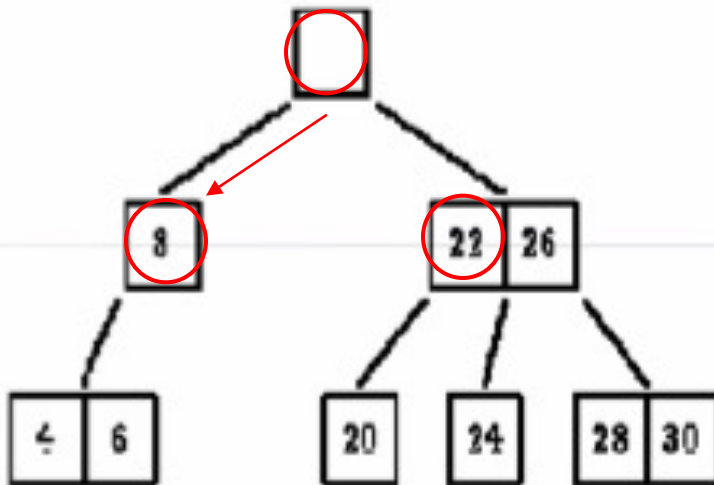
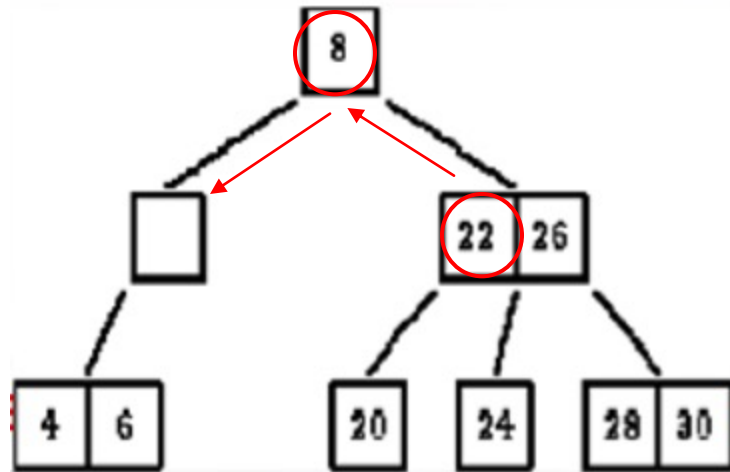
Mất cân bằng



Cân bằng cây sau khi xoá

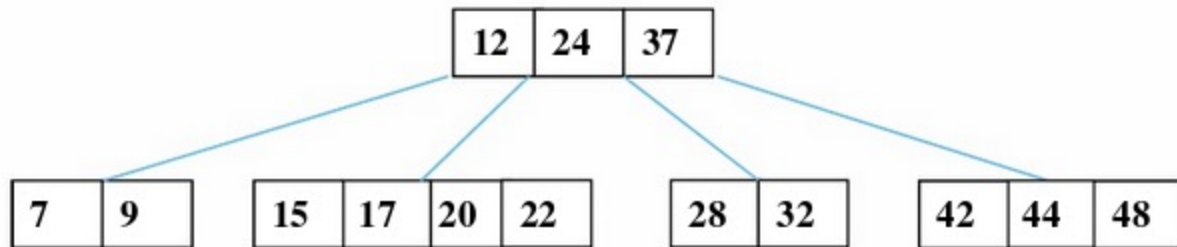
► Xoá 18 (tt)

- Đưa 1 khoá của nút kế cân (22 26) lên nút cha
- Đưa 1 khoá ở nút cha xuống nút thiếu

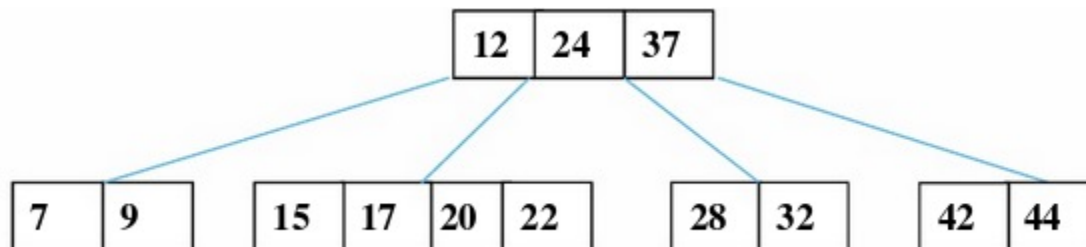


Xóa

► B-tree bậc 5:

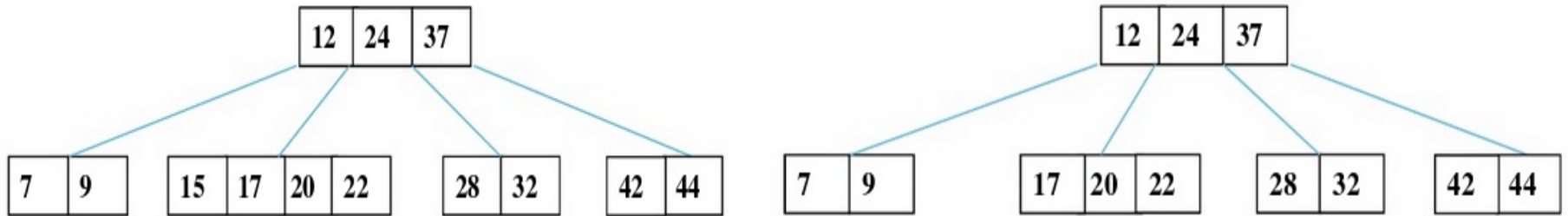


► Xóa 48: trên trang lá

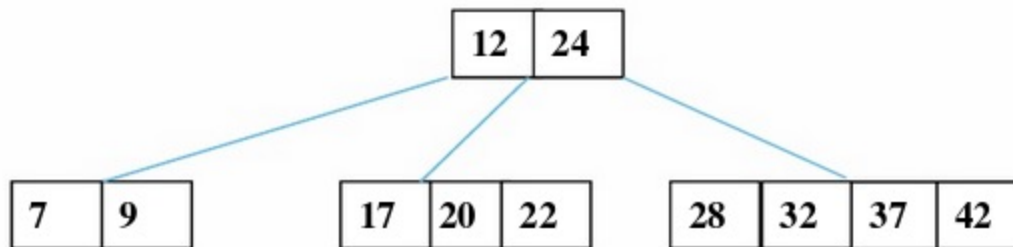


Xóa

► Xóa 15: trên trang lá

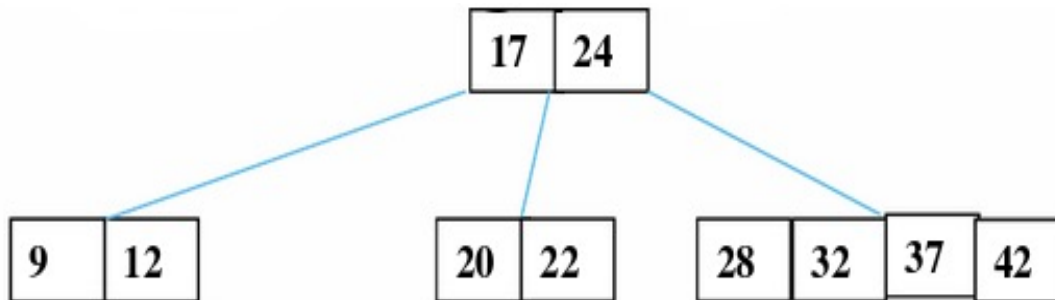
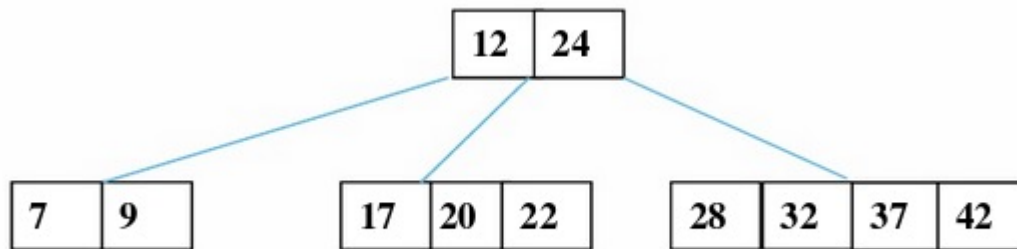


► Xóa 44: gộp trang



Xóa

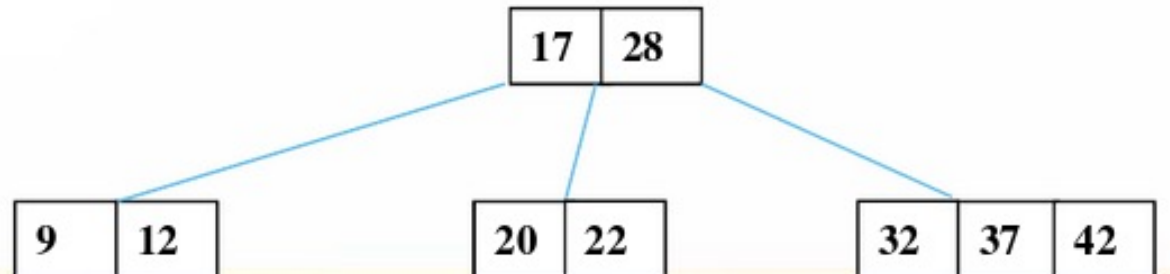
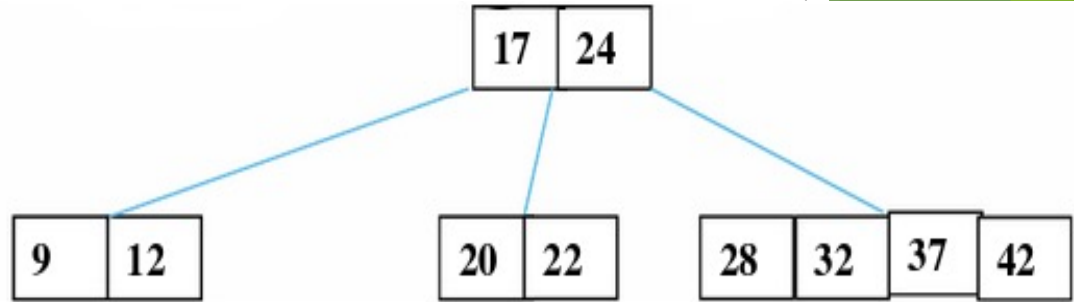
- Xóa 7: mượn trang phải



Xoá

► Xoá 24:

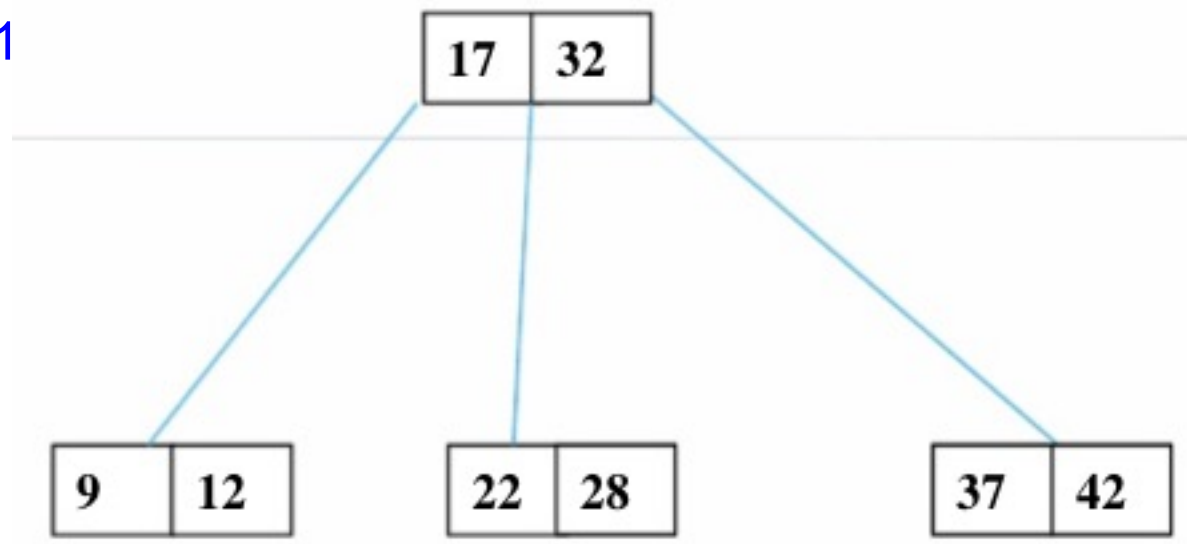
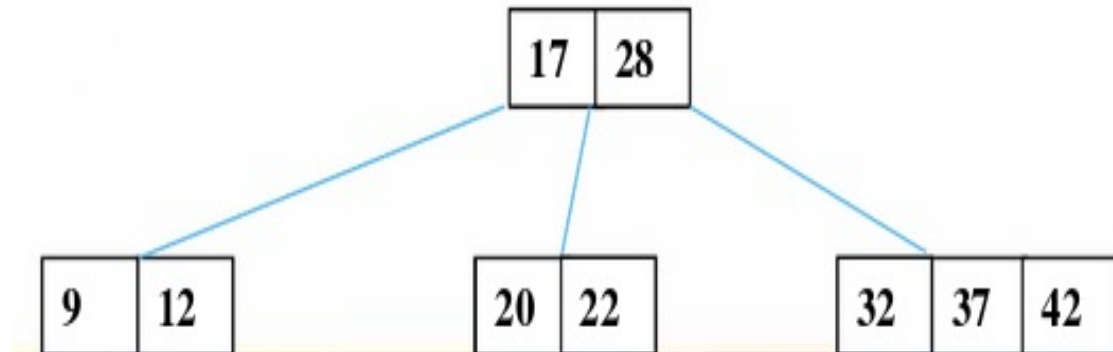
- Nếu đưa 22 lên thế, trang 22, 20 chỉ còn 1 phần tử (không hợp lệ)
- → Đưa 28 lên thế



Xoá

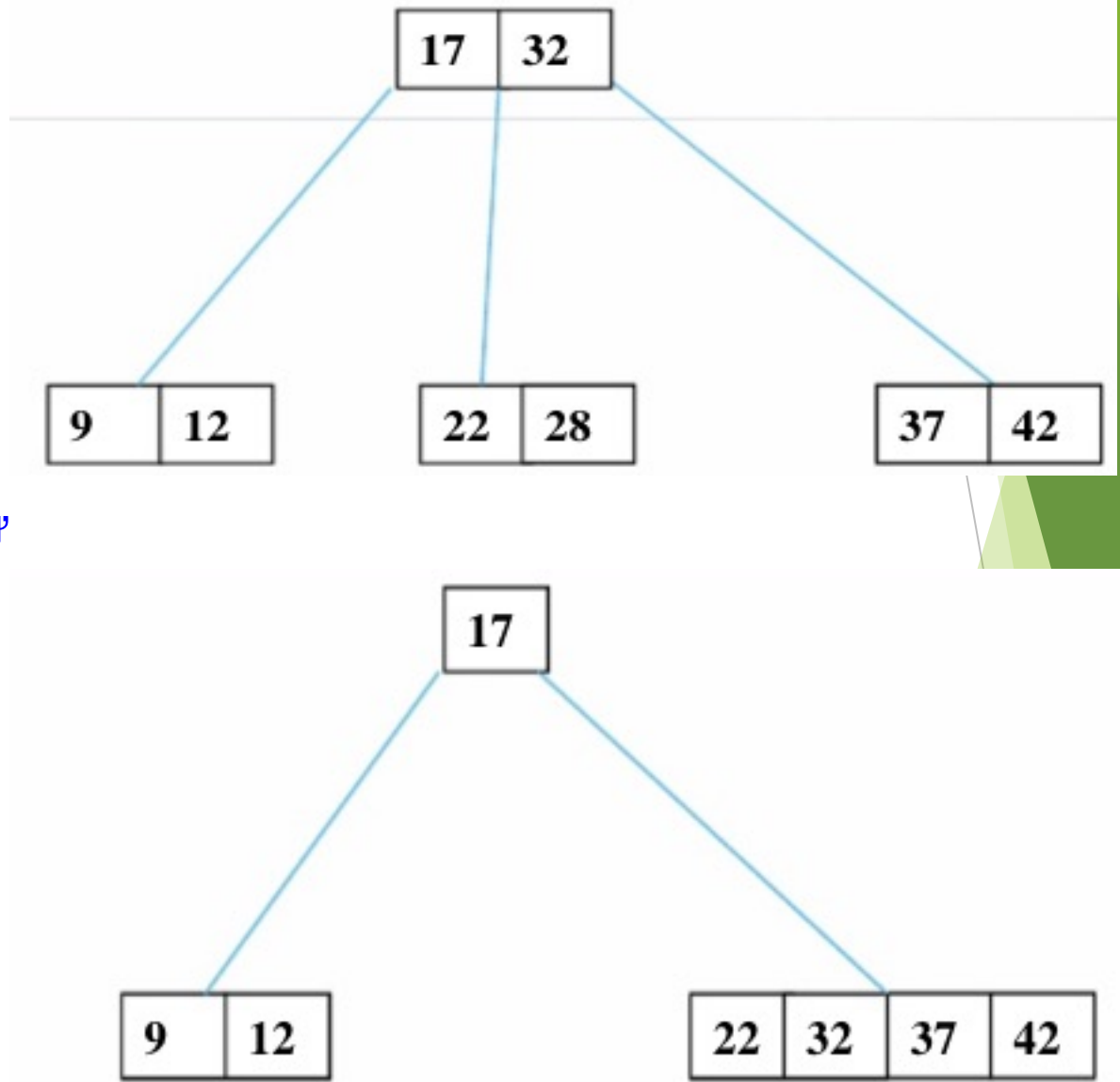
► Xoá 20:

- Trang 22 còn 1 phần tử là không hợp lệ
- Mượn trang phải 1 phần tử. Tức là mang 32 lên cha, đưa 28 xuống ghép với 22.



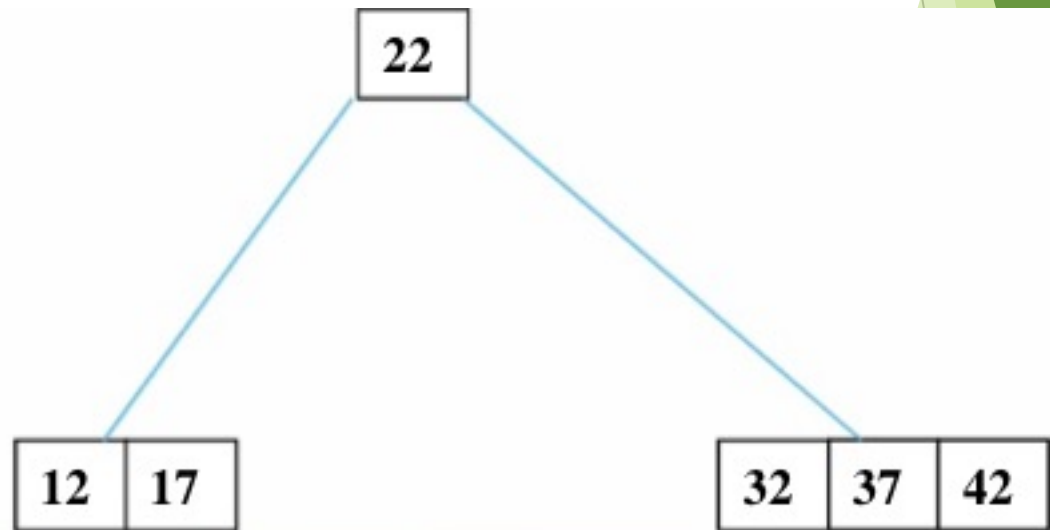
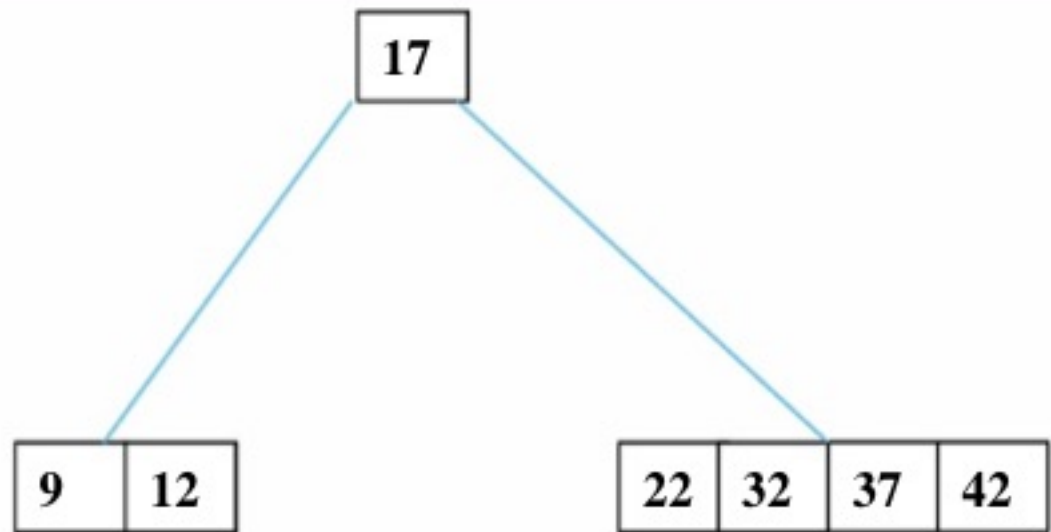
Xoá

- ▶ **Xoá 28:**
- ▶ Trang 22 còn 1 phần tử là không hợp lệ.
- ▶ Trang bên phải cũng có số phần tử vừa đủ → không thể mượn.
- ▶ Do đó, phải gộp trang.



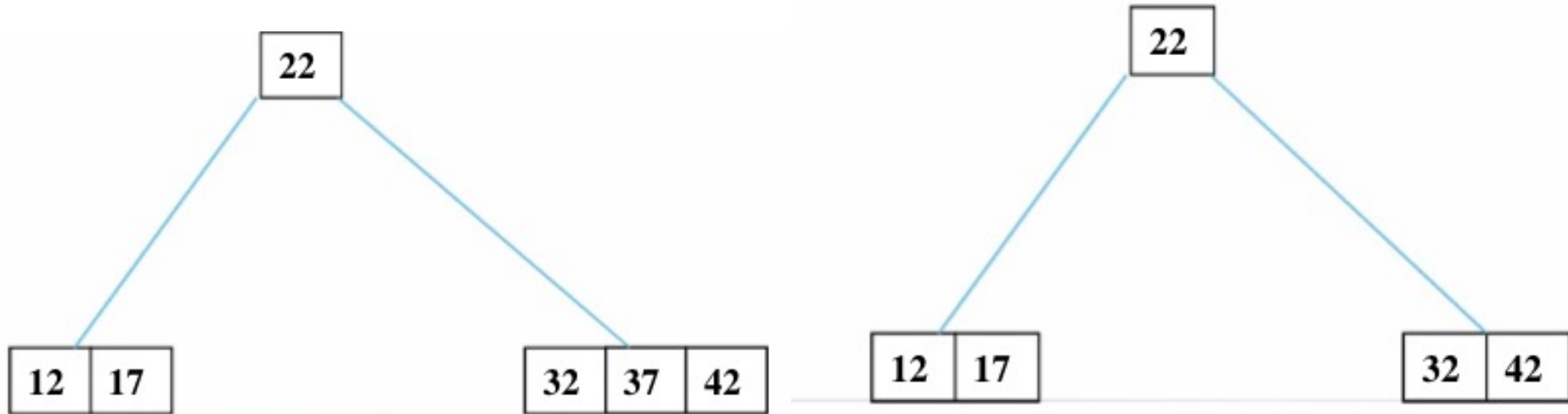
Xoá

- **Xoá 9:**
- Mượn phần tử của trang bên phải.

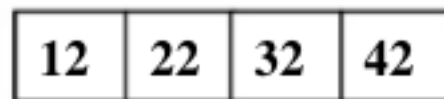


Xoá

► Xoá 37:



► Xoá 17: gộp trang.



Ứng dụng của B-Tree

- B-Tree có nhiều biến thể và cải tiến
 - B⁺-Tree
 - B^{*}-Tree
- Quản lý dữ liệu trên đĩa cứng, dữ liệu lớn

Ứng dụng của B-Tree

- File system - Hệ thống quản lý file trên đĩa cứng
 - Danh sách các block còn trống
 - File x đang nằm ở block số mấy?
- Windows:
 - NTFS, FAT32,...
- MacOS
 - HFS+
- Linux
 - Btrfs, Ext, xFS
- Other:
 - HFS, Reiser4, HAMMER, ...