# Cấu trúc dữ liệu và giải thuật

## B-CÂY

Giảng viên:
Văn Chí Nam – Nguyễn Thị Hồng Nhung – Đặng Nguyễn Đức Tiến

# Nội dung trình bày



Cây tìm kiếm m-nhánh

B-Cây

Các thao tác trên B-cây

# Cây tìm kiếm m-nhánh

m-way search tree

m-way tree

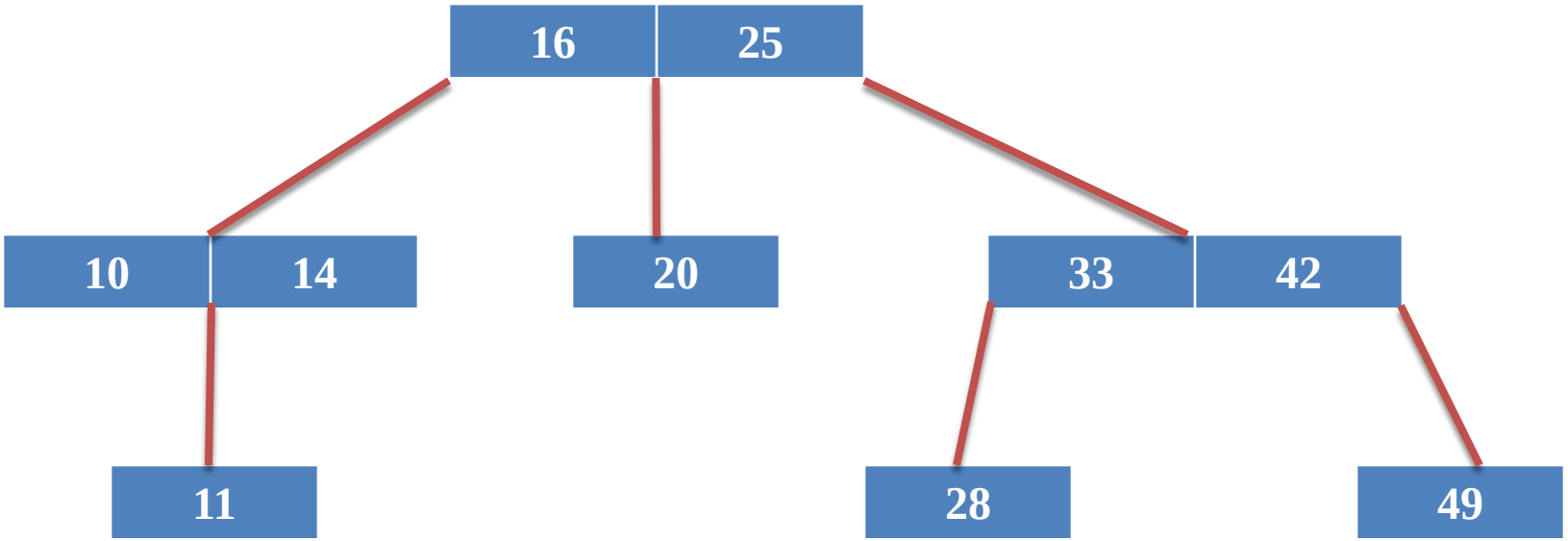
# Định nghĩa



- Cây tìm kiếm m-nhánh là cây có tính chất:
  - Có tối đa *m-1* khóa trong mỗi node ($v_1$, $v_2$,.., $v_k$) ($k \leq m-1$).
  - Các giá trị khóa trong node được tổ chức có thứ tự ($v_1 < v_2 < ... < v_k$).
  - Một node có *k* khóa thì sẽ có *k + 1* cây con (các cây con có thể rỗng).
    - Các cây con đặt giữa hai giá trị khóa.
    - Hai cây con nằm ở hai đầu của dãy khóa
    - Mỗi khóa sẽ có cây con trái và cây con phải.
      - Các giá trị của cây con trái sẽ nhỏ hơn giá trị của khóa.
      - Các giá trị của cây con phải sẽ lớn hơn giá trị của khóa.

# Ví dụ






**Cây tìm kiếm 3-nhánh**

# Thao tác trên cây



- Tìm kiếm
- Thêm phần tử
- Xóa phần tử

# Tìm kiếm



- Tổng quát hóa từ trường hợp cây nhị phân tìm kiếm
  - X là giá trị cần tìm
  - Nếu $X < v_1$ thì tìm X bên nhánh trái của $v_1$.
  - Ngược lại, nếu $X > v_k$ thì tìm X bên nhánh phải của $v_k$.
  - Nếu $X = v_i$ thì thông báo tìm thấy.
  - Nếu $v_i < X < v_{i+1}$ thì tìm X tại cây con nằm giữa $v_i$ và $v_{i+1}$.

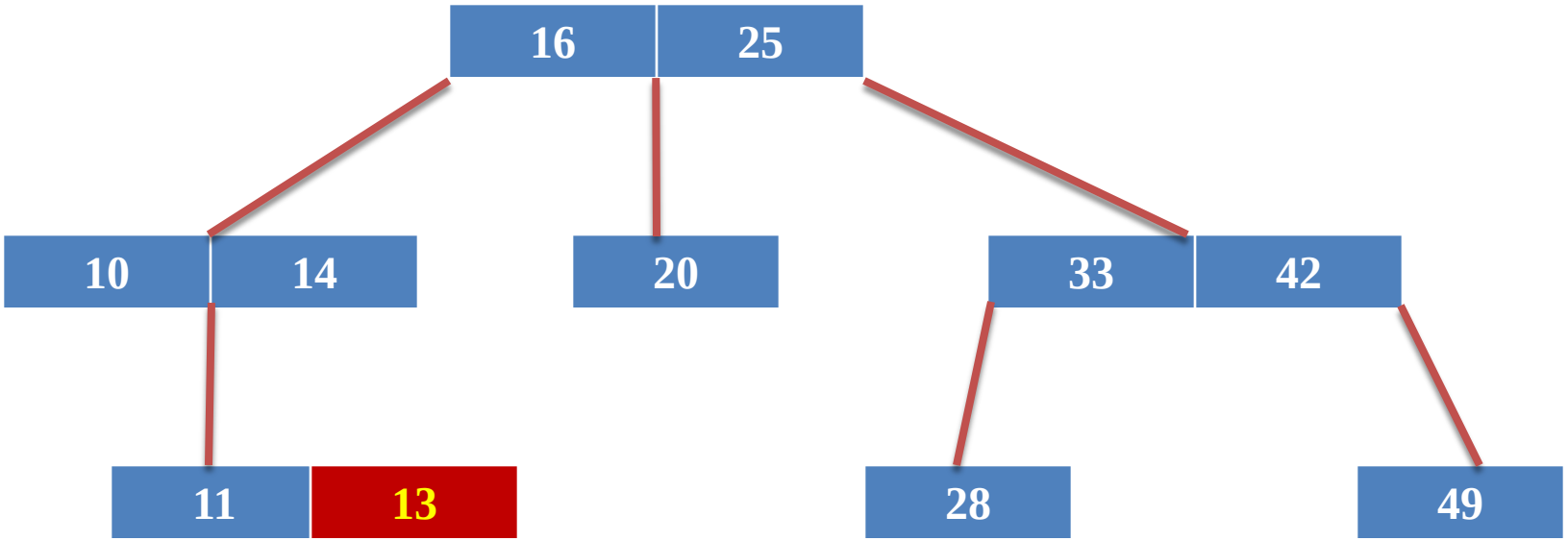
# Thêm phần tử



- Tổng quát hóa từ trường hợp cây nhị phân tìm kiếm
  - X là giá trị cần thêm vào cây.
  - Duyệt cây tìm X trên cây.
    - Nếu X đã tồn tại trên cây thì không thêm.
    - Nếu X chưa tồn tại (tìm thấy node rỗng) thì
      - Nếu node cha (của node rỗng tìm thấy) còn có thể thêm X vào thì thêm X vào node cha.
      - Ngược lại, tạo node mới và thêm X vào node đó.

# Thêm phần tử

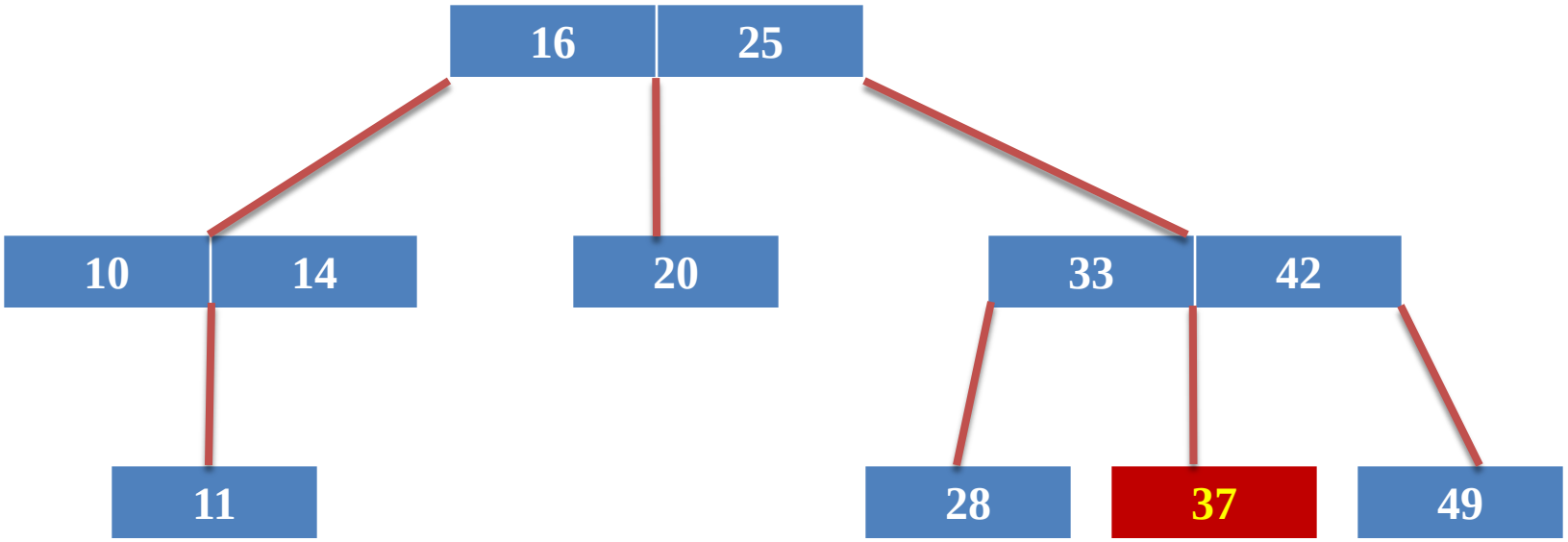





**Thêm vào giá trị 13**

# Thêm phần tử






**Thêm vào giá trị 37**

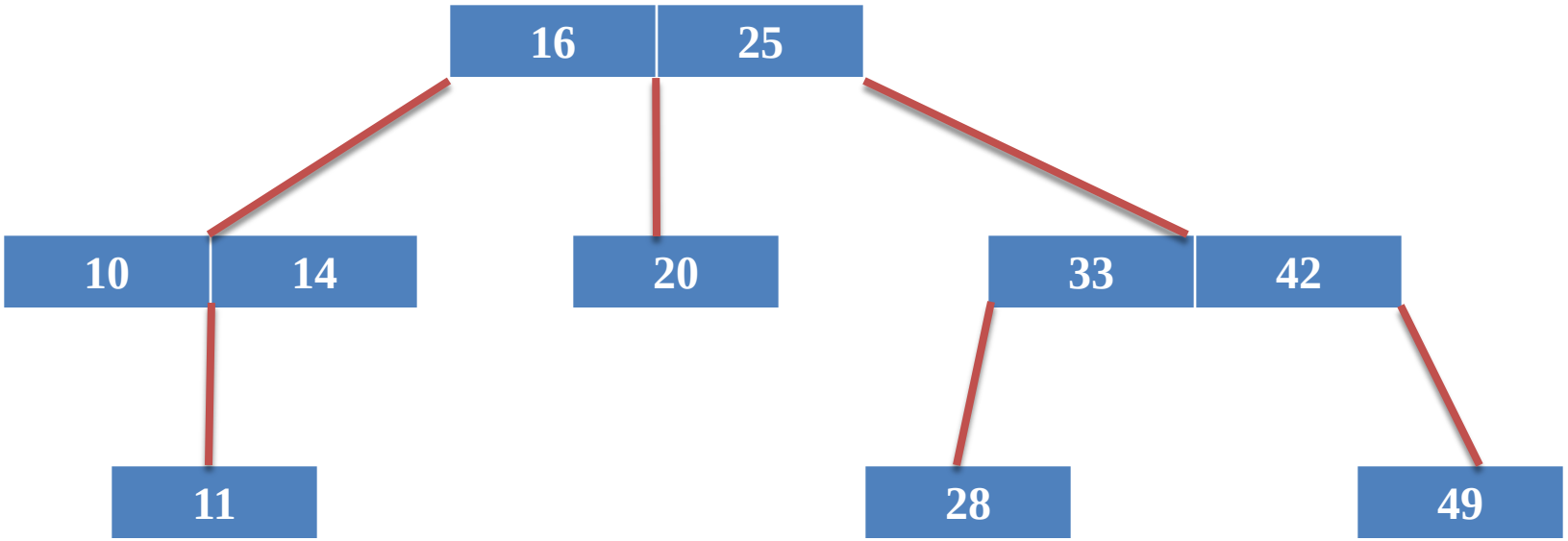
# Xóa phần tử



- Tương tự cây nhị phân tìm kiếm
  - Tìm vị trí của phần tử X cần xóa.
  - Nếu X nằm giữa hai cây con rỗng thì xóa X.
  - Nếu X có cây con, thay thế X bằng:
    - Phần tử lớn nhất bên cây con trái của X hoặc
    - Phần tử nhỏ nhất bên cây con phải của X

# Xóa phần tử

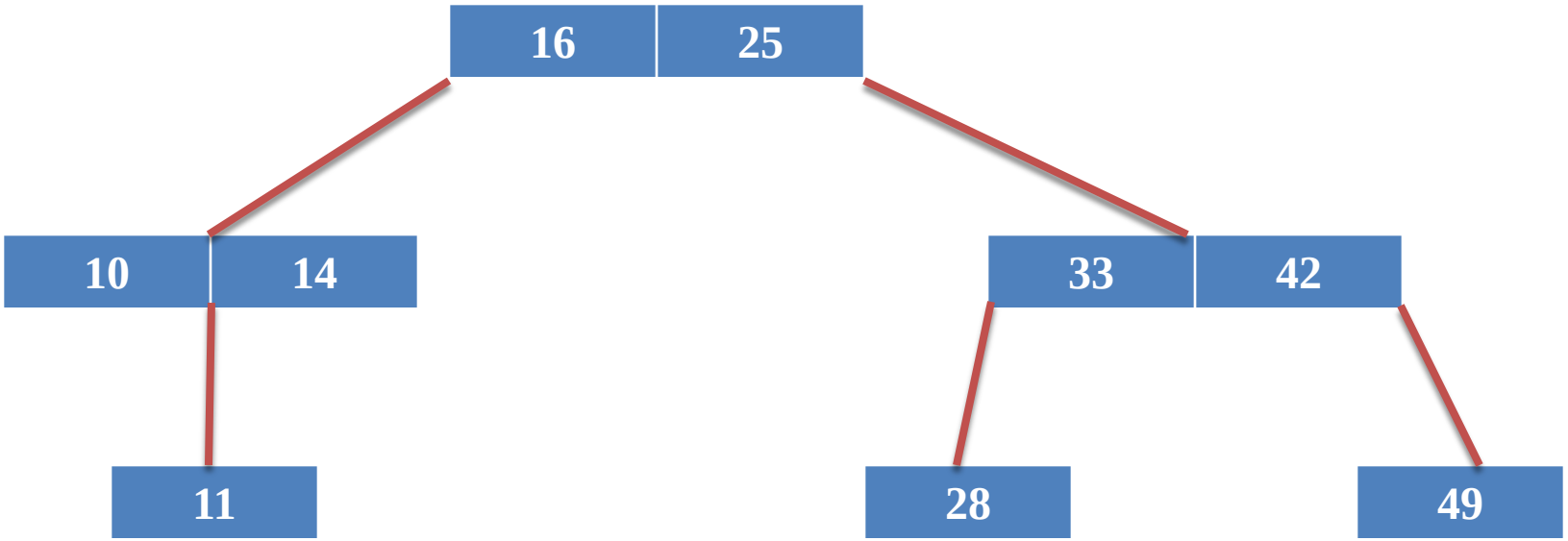





**Xóa giá trị 20**

# Xóa phần tử

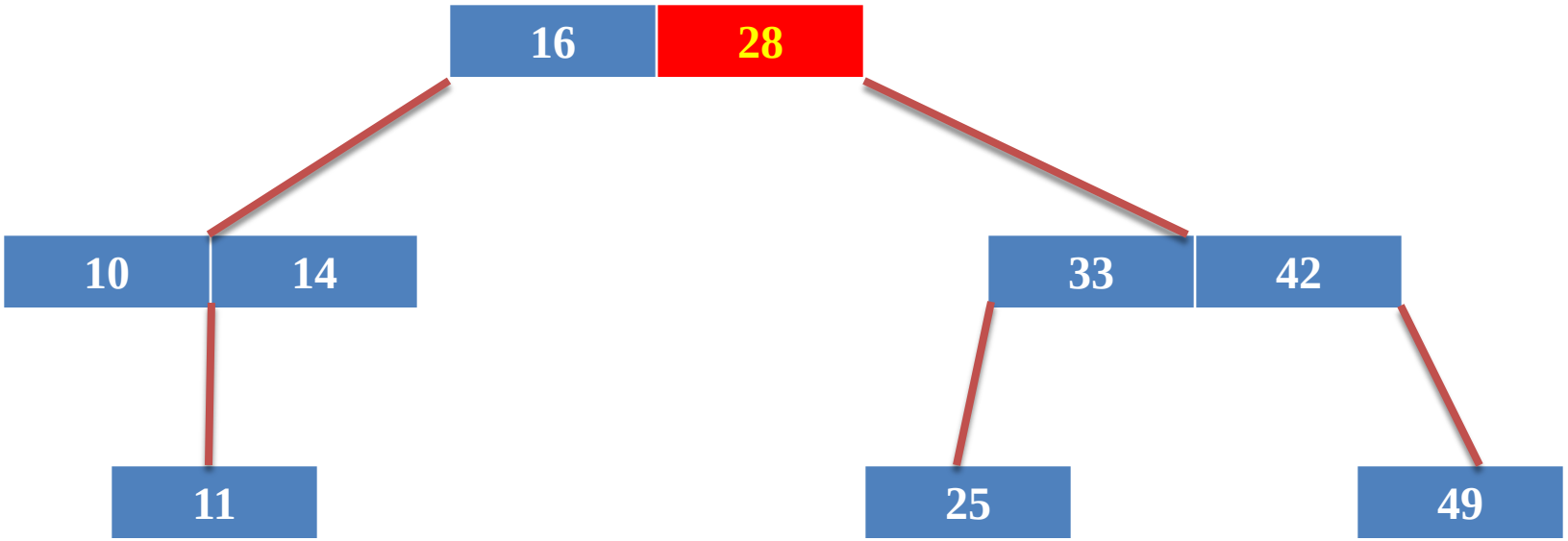





**Xóa giá trị 25**

# Xóa phần tử

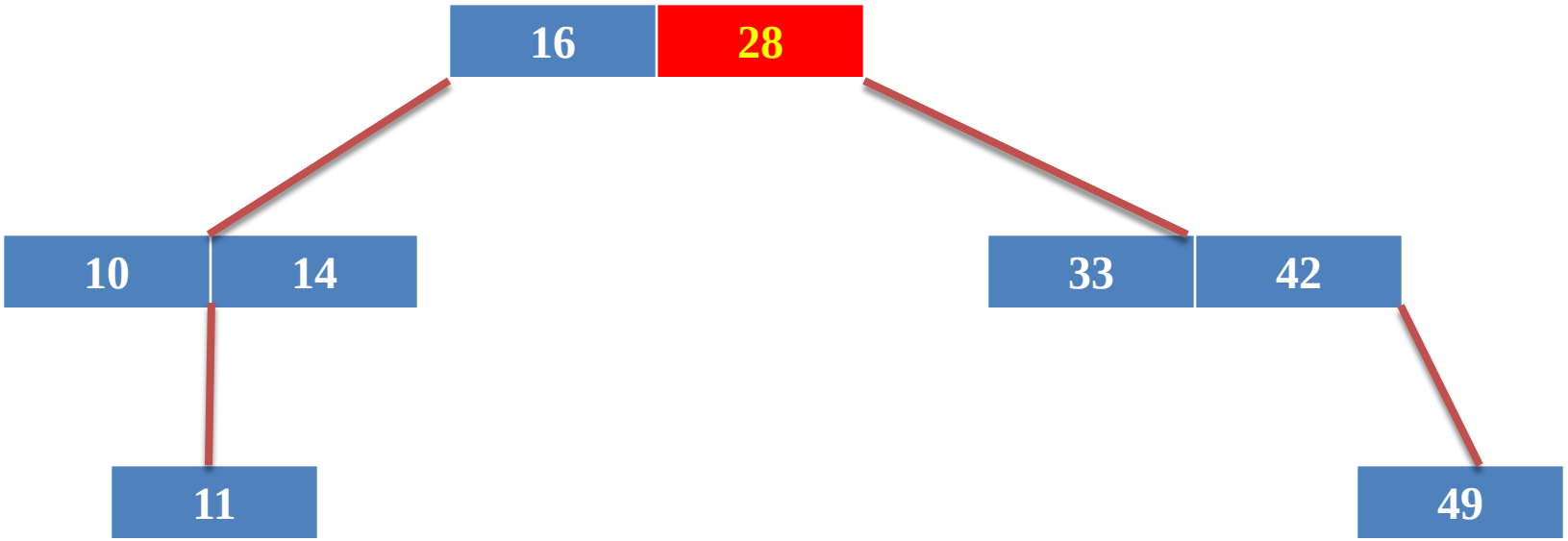





**Xóa giá trị 25**

# Xóa phần tử






**Xóa giá trị 25**

# B-Cây

B-tree

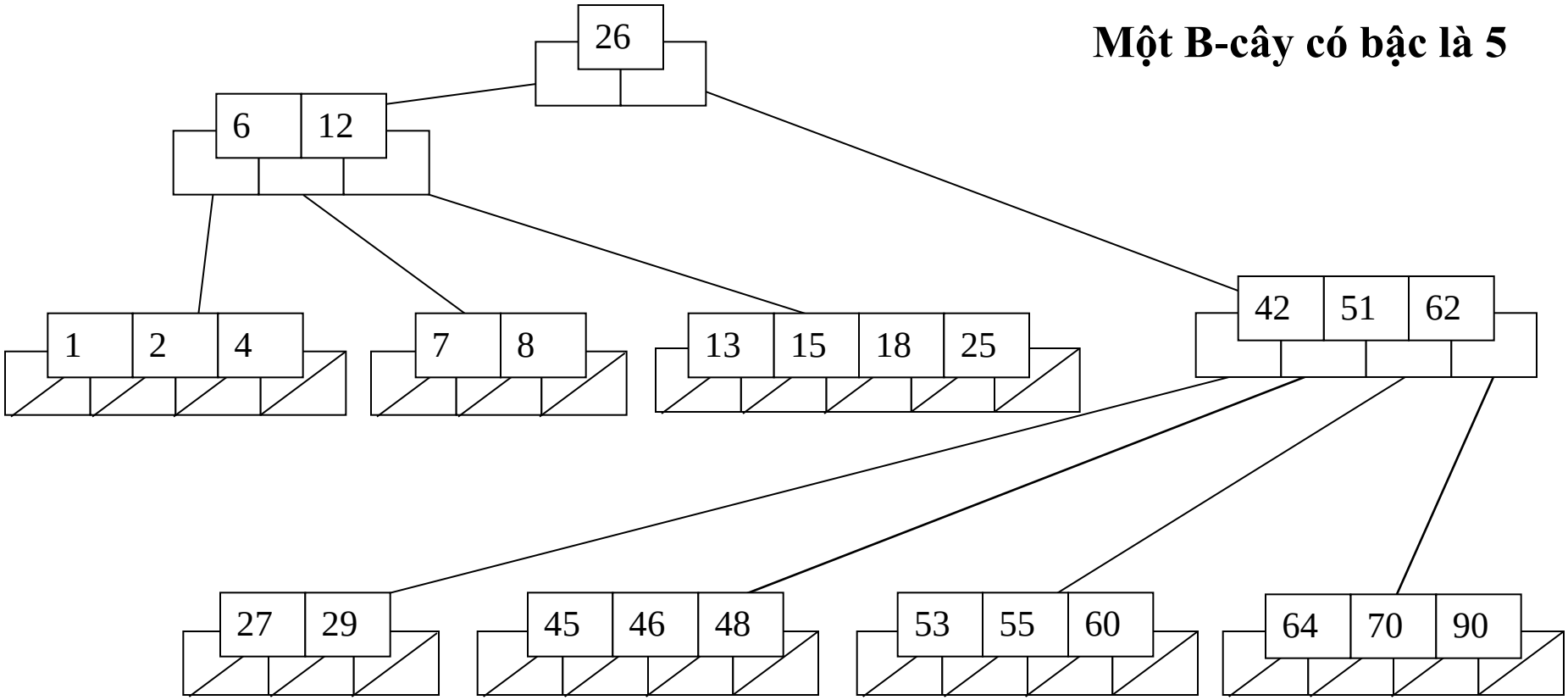
# Định nghĩa



- B-cây bậc m là 1 cây tìm kiếm m-nhánh (m>2) thỏa:
  - Nút gốc có ít nhất 1 khóa
  - Tất cả các cây con rỗng ở cùng một mức.
  - Tất cả các node (trừ node gốc) có ít nhất $\lceil \mathbf{m/2} \rceil$ cây con (có ít nhất $\lceil \mathbf{m/2} \rceil$ **-1** khóa).
- Giá trị *m* thường là lẻ.

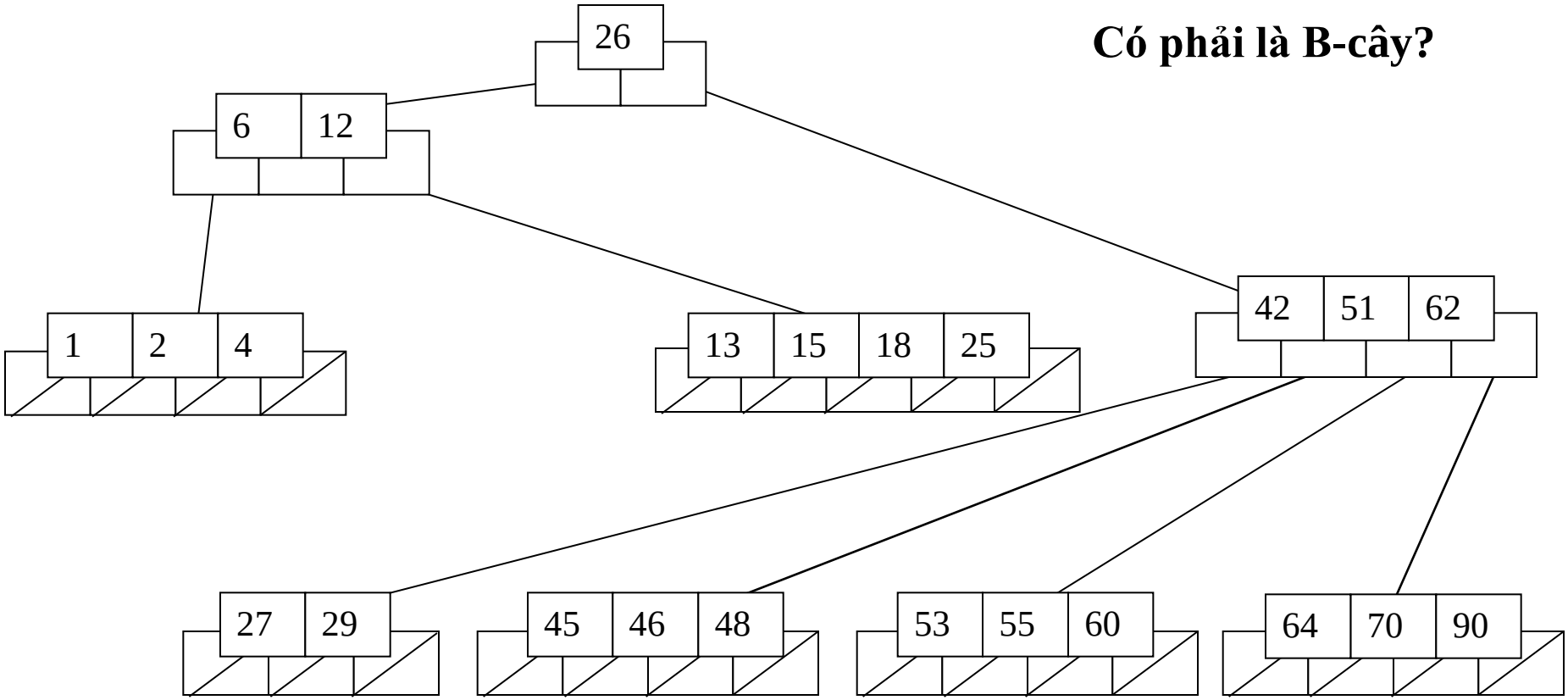
# Ví dụ




Một B-cây có bậc là 5
26
6
12
1
2
4
7
8
13
15
18
25
42
51
62
27
29
45
46
48
53
55
60
64
70
90

# Ví dụ




Có phải là B-cây?
26
6
12
1
2
4
13
15
18
25
42
51
62
27
29
45
46
48
53
55
60
64
70
90

# Thao tác trên cây



- Tìm kiếm
  - Thực hiện tương tự trên cây tìm kiếm m-nhánh.
- Thêm phần tử
- Xóa phần tử

# Thêm phần tử



- Thêm phần tử vào node lá.
- Nếu node lá bị tràn thì
  - Tách thành 2 node mới.
  - Khóa chính giữa được đưa lên node cha.
- Thực hiện tương tự nếu node cha bị tràn.
- Nếu node gốc bị tràn thì tạo một node gốc mới (có 1 khóa duy nhất là khóa chính giữa của node cũ)

# Thêm phần tử - Ví dụ

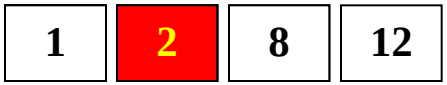


- Tạo B-cây bậc 5 gồm các phần tử theo thứ tự sau:
  - 1, 12, 8, 2, 25, 5, 14, 28, 17

**1 12 8 2** 25 5 14
28 17

# Thêm phần tử - Ví dụ



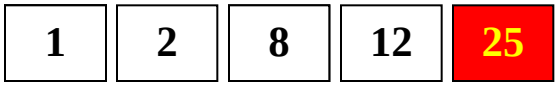


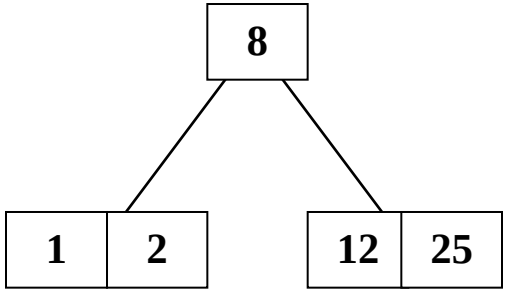

Thêm **1** Thêm **12** Thêm **8** Thêm **2**

1 12 8 2 **25** 5 14
28 17

# Thêm phần tử - Ví dụ





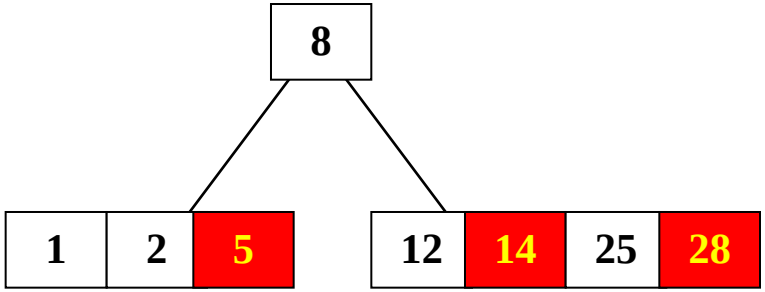




Thêm **25**

1 12 8 2 25 **5** **14**
**28** 17

# Thêm phần tử - Ví dụ



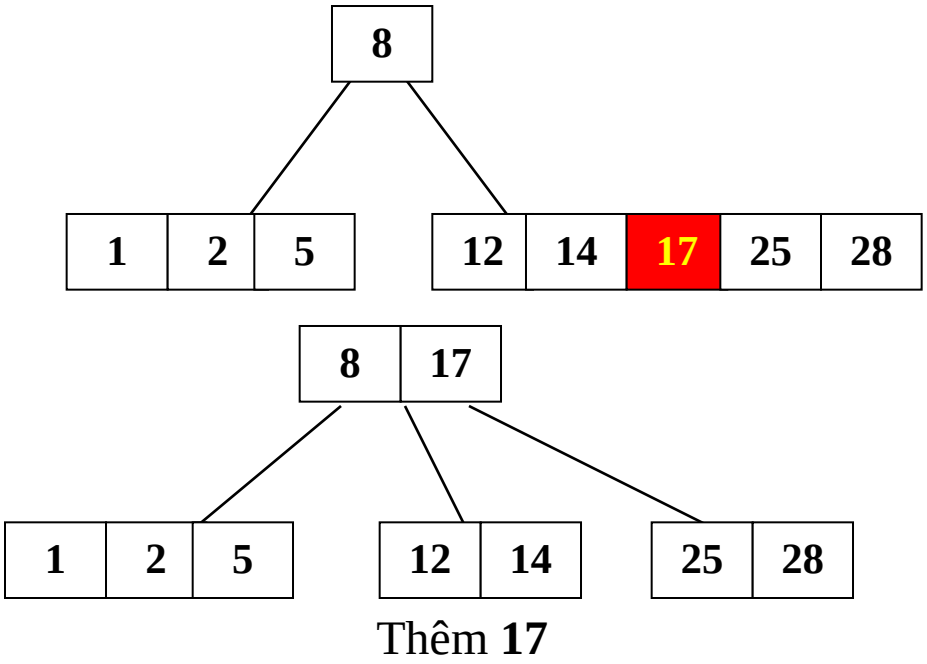



Thêm **5, 14, 28**

1 12 8 2 25 5 14
28 **17**

# Thêm phần tử - Ví dụ






Thêm **17**

# Xóa phần tử



- Thực hiện tương tự cây tìm kiếm m-nhánh.

- Xét hai trường hợp:
  - Khóa thuộc node lá
  - Khóa thuộc node trong

# Xóa phần tử



- Khóa thuộc node lá:
  - Xóa khóa khỏi node chứa khóa.
  - Sau khi xóa, nếu node chứa khóa mới xóa có số khóa không đủ (ít hơn $\lceil m/2 \rceil - 1$ khóa) thì:
    - Mượn khóa từ node bên cạnh (Node bên cạnh dư khóa).
    - Nhập khóa với node bên cạnh cùng với khóa cha (Node bên cạnh KHÔNG dư khóa).

# Xóa phần tử



- Sau khi xóa khóa dẫn đến việc các node trong cây bị thiếu khóa: cân bằng lại cây từ dưới lên.
  - TH1: Nếu node kề phải dư khóa
    - Thêm khóa cha của 2 node vào node bị thiếu.
    - Lấy khóa đầu của node kề phải lên thay cho khóa cha ở node cha.
  - TH2: Nếu node kề trái dư khóa: làm tương tự trường hợp trên
  - TH3: Nếu cả node kề trái và phải đều không dư khóa:
    - Tạo node mới chứa khóa của node bị thiếu, tất cả khóa của 1 node kề nó và khóa cha của 2 node này.
    - Xóa khóa cha của 2 node ở node cha, và thay 2 node con vừa bị nhập bằng node mới tạo.

# Xóa phần tử

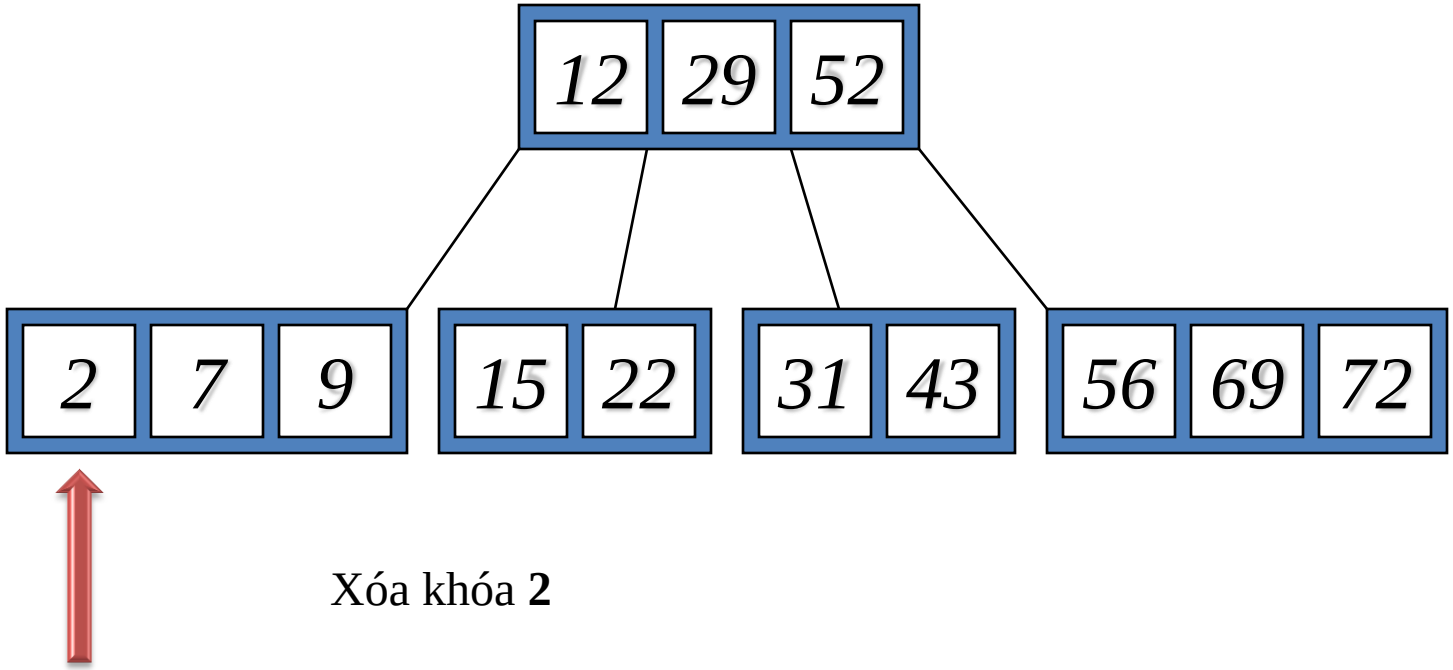


- Khóa thuộc node trong:
  - Khóa bị xóa có các node bên nhánh con trái và nhánh con phải có số khóa tối thiểu (⌈**m/2**⌉**-1** khóa): nhập khóa của 2 node con.
  - Ngược lại: tìm phần tử thế mạng và thực hiện **cân bằng lại cây** như trường hợp xóa khóa thuộc node lá.

# Xóa phần tử - Ví dụ



- Cho B-cây bậc 5:

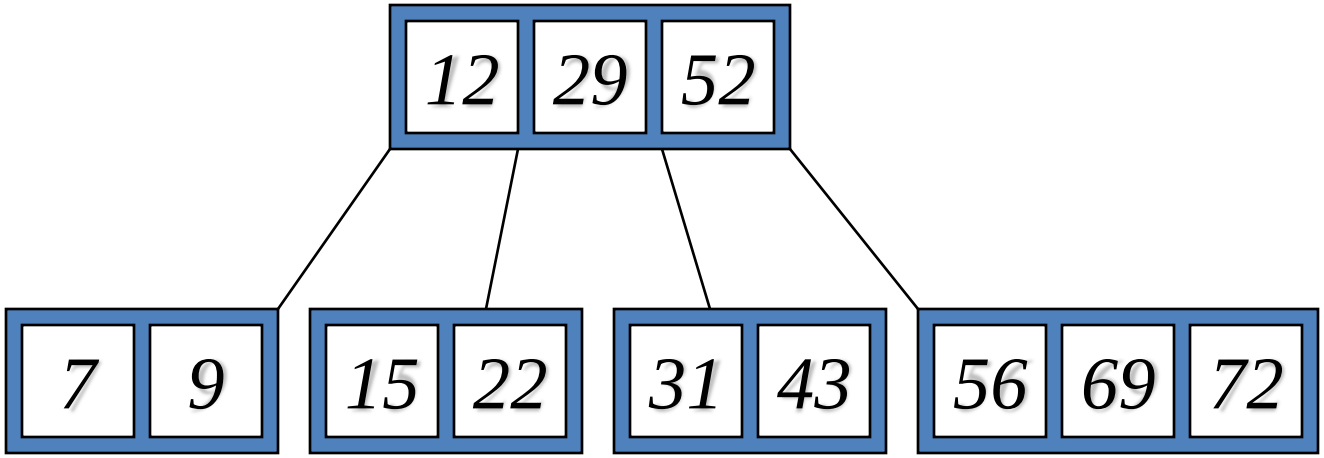



Xóa khóa **2**

# Xóa phần tử - Ví dụ



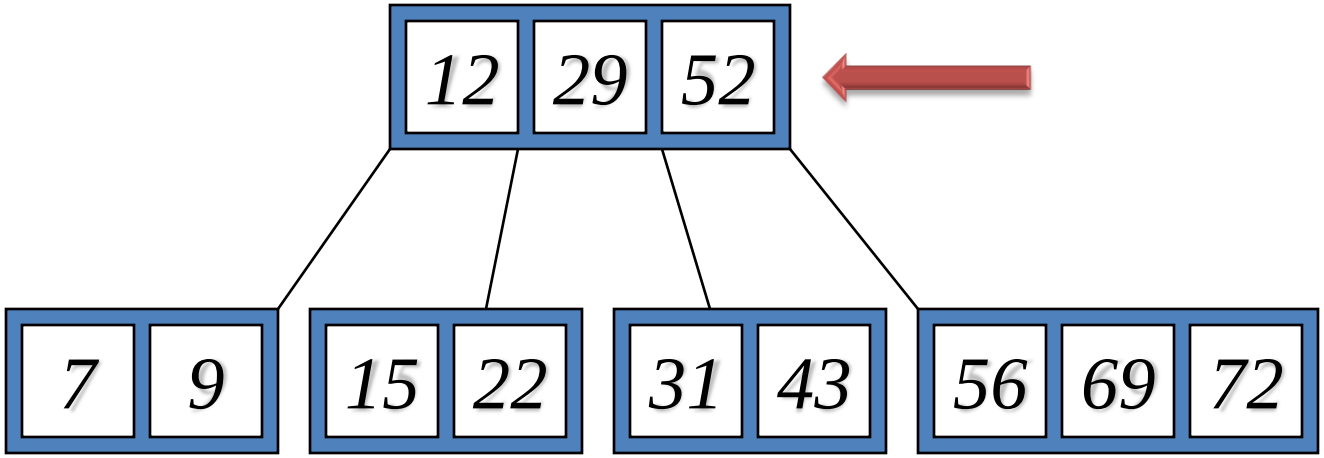
- Cho B-cây bậc 5:



# Xóa phần tử - Ví dụ



- Cho B-cây bậc 5:

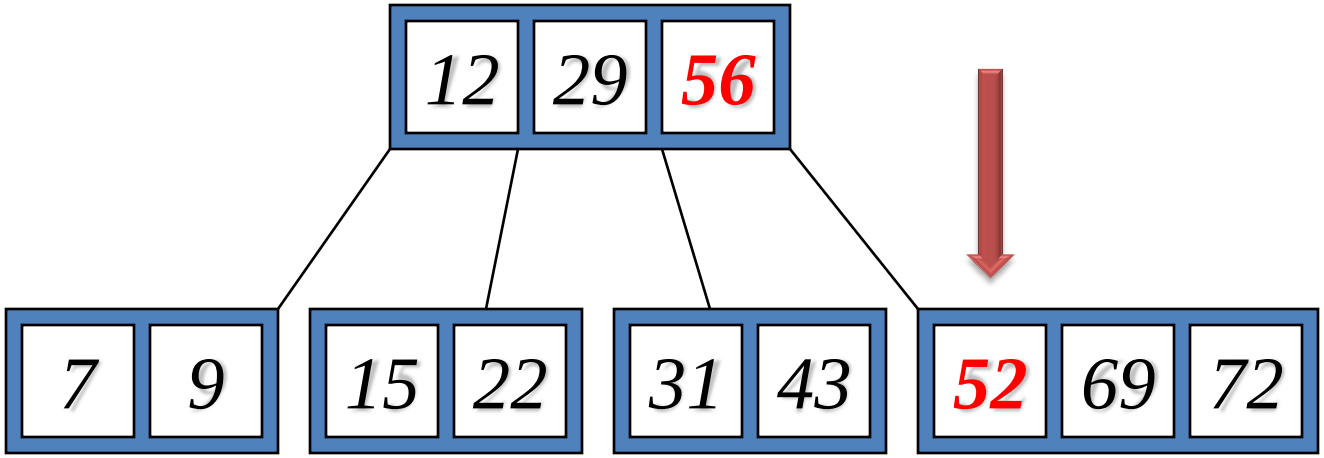



Xóa khóa **52**

# Xóa phần tử - Ví dụ



- Cho B-cây bậc 5:

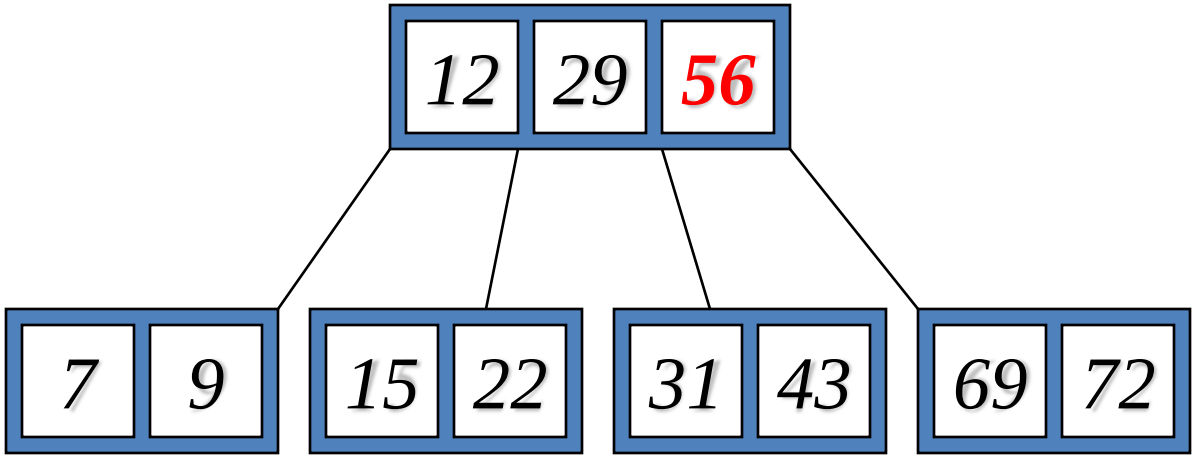



Xóa khóa **52**. Thay thế bằng **56**

# Xóa phần tử - Ví dụ



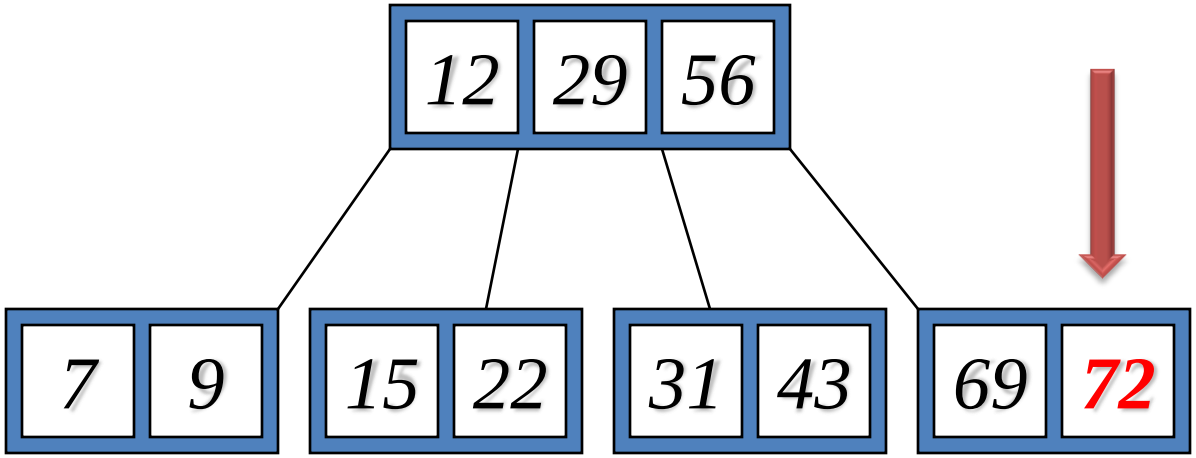
- Cho B-cây bậc 5:



# Xóa phần tử - Ví dụ



- Cho B-cây bậc 5:

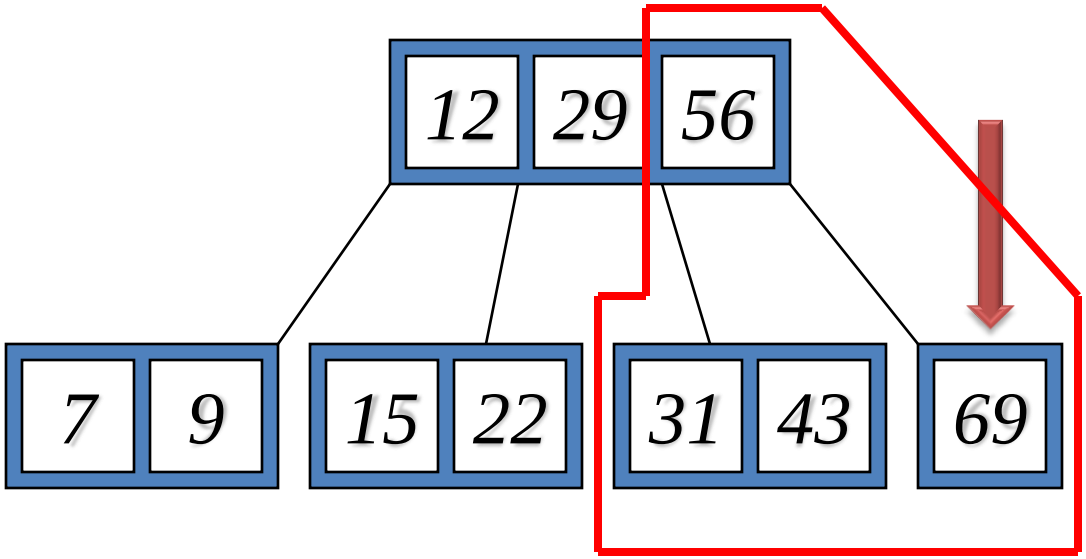



Xóa khóa **72**

# Xóa phần tử - Ví dụ



- Cho B-cây bậc 5:

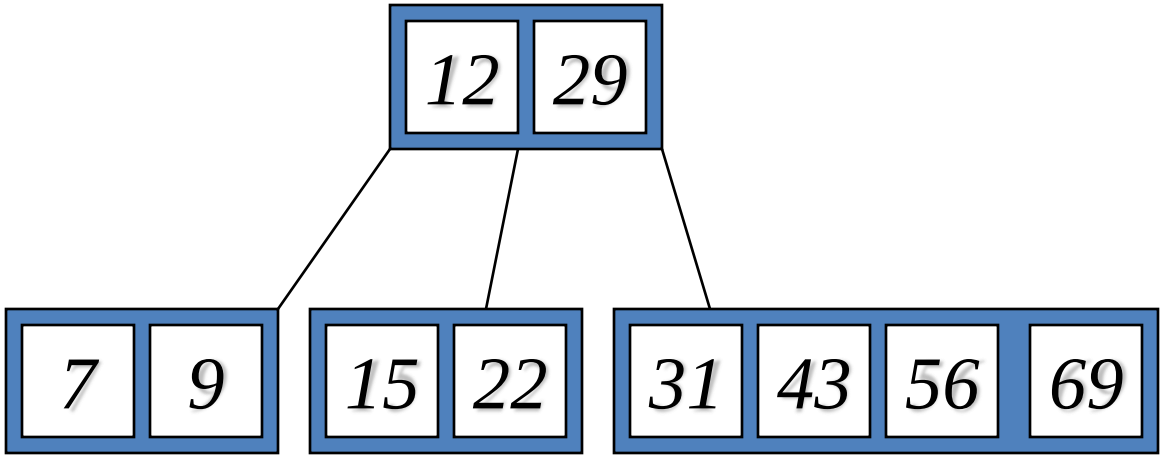



**Ít khóa -> Nhập node**

# Xóa phần tử - Ví dụ



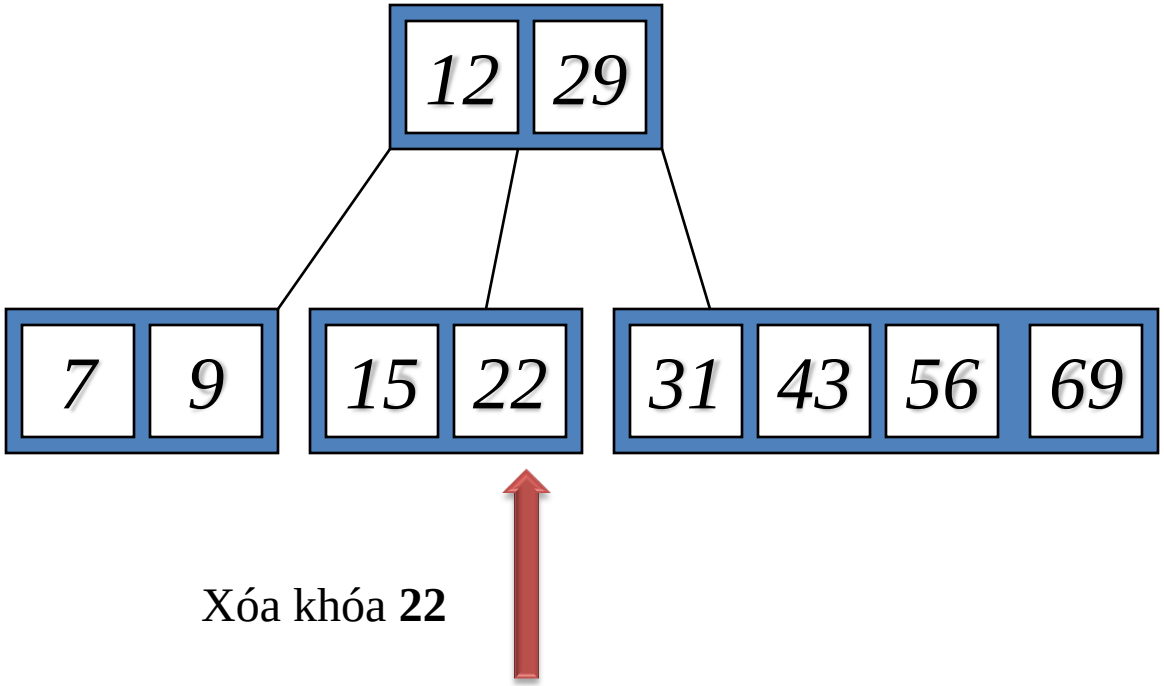
- Cho B-cây bậc 5:



# Xóa phần tử - Ví dụ



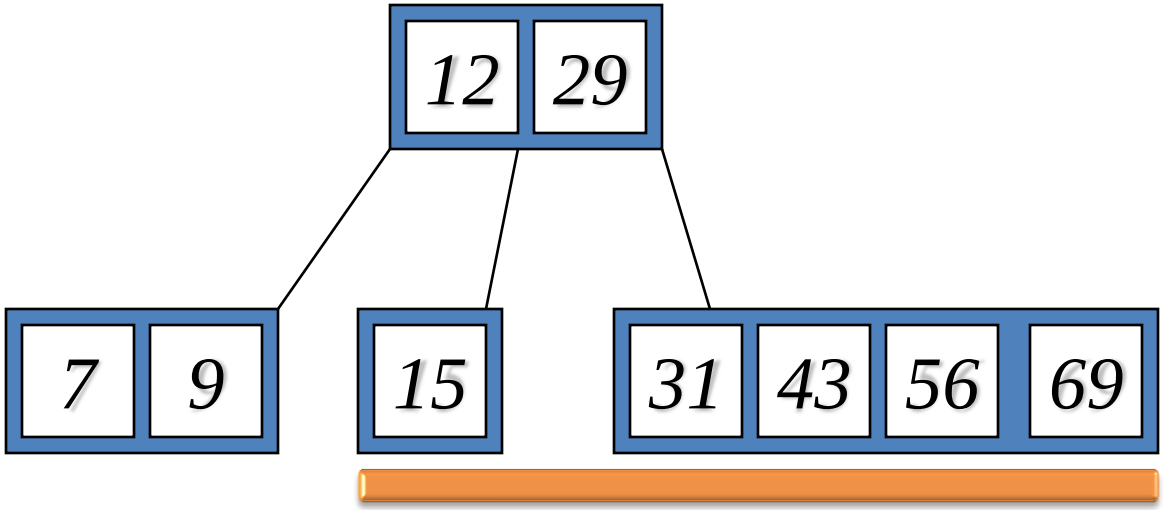
- Cho B-cây bậc 5:



# Xóa phần tử - Ví dụ



- Cho B-cây bậc 5:

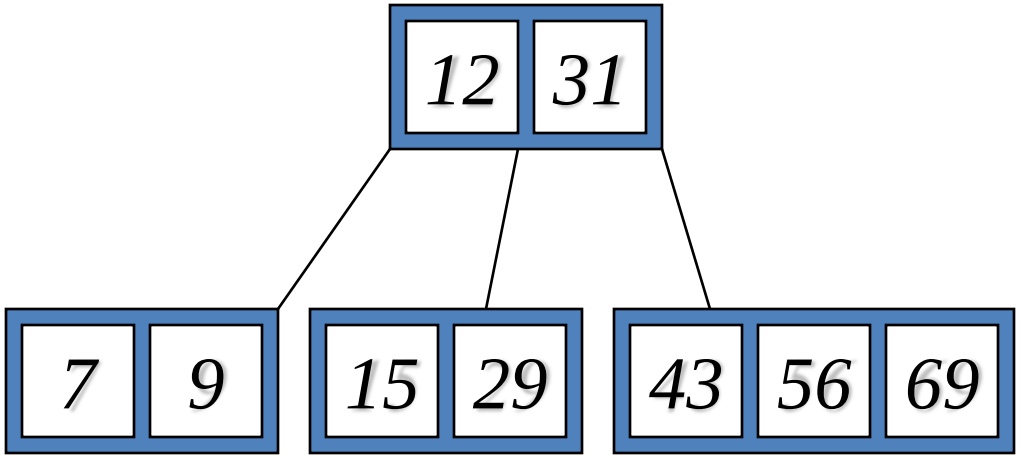



Mượn khóa

# Xóa phần tử - Ví dụ



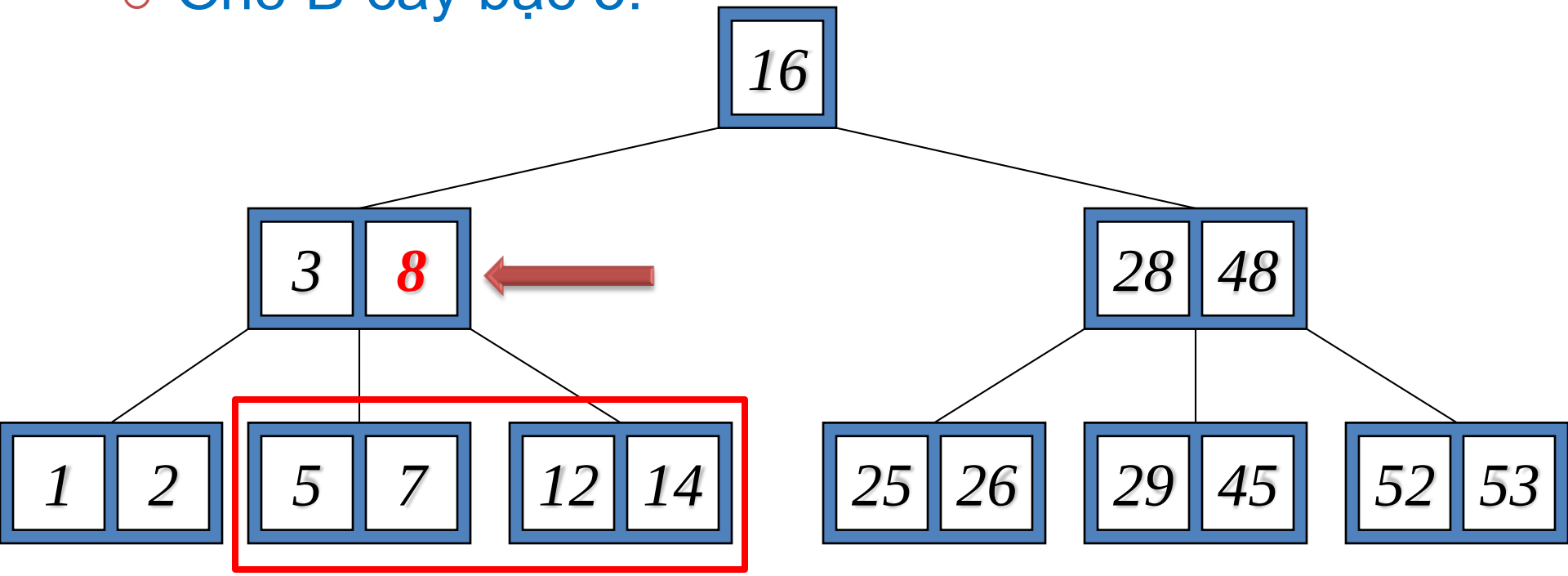
- Cho B-cây bậc 5:



# Xóa phần tử - Ví dụ



- Cho B-cây bậc 5:

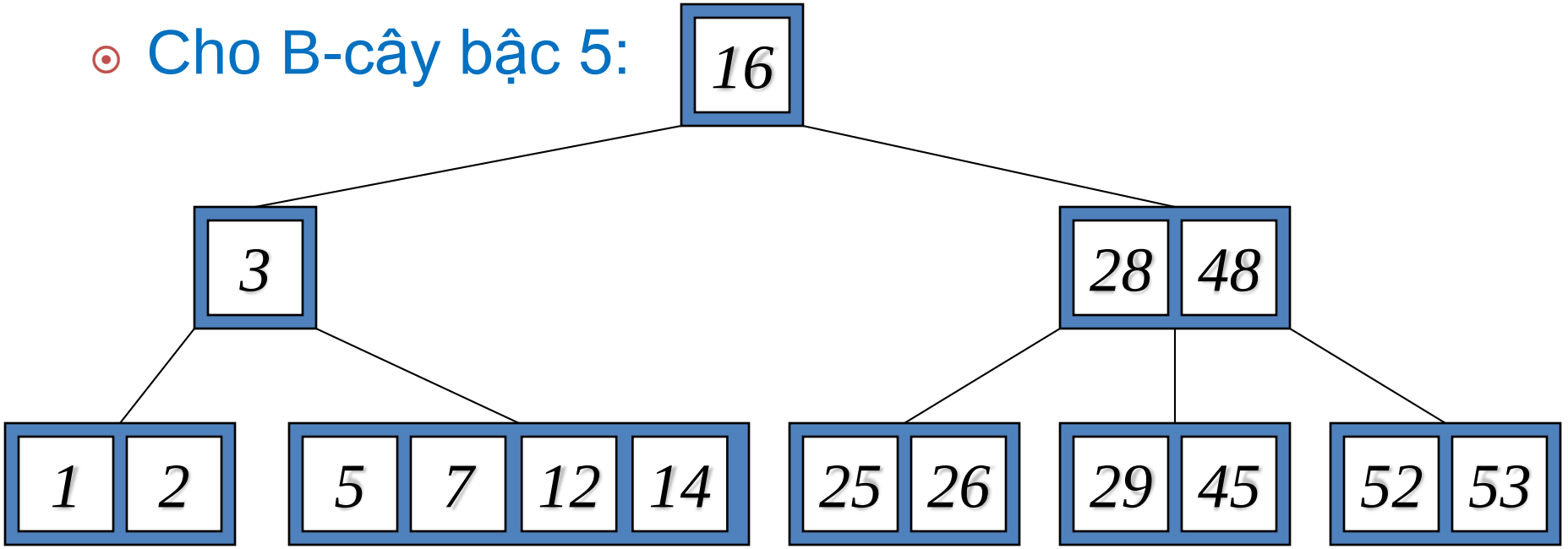



Xóa khóa **8** ⇒ Nhập 2 node con của 8 lại

# Xóa phần tử - Ví dụ



- Cho B-cây bậc 5:



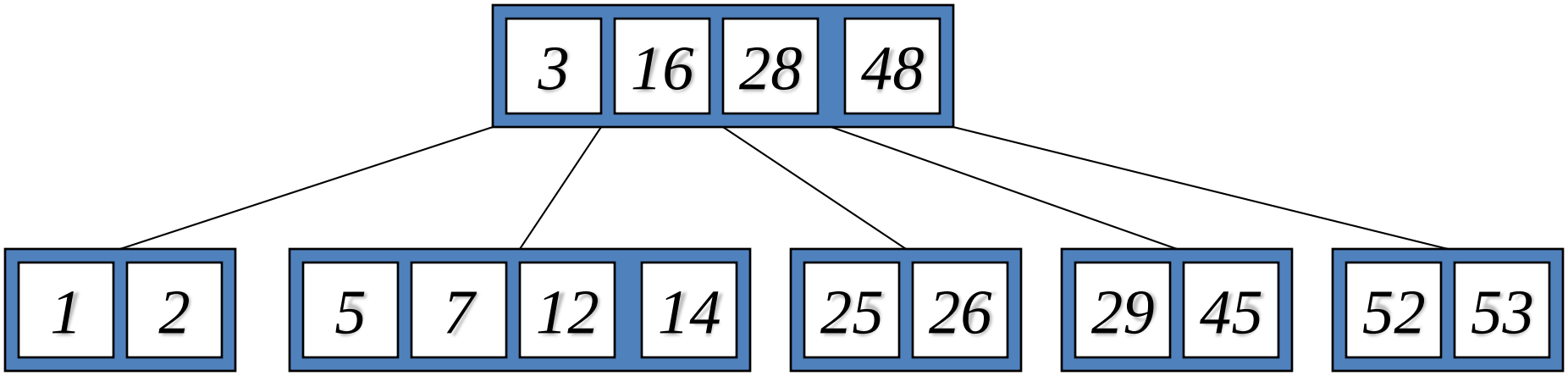

Nhận xét?

Node 3 bị thiếu khóa → Cân bằng lại cây theo TH3

# Xóa phần tử - Ví dụ



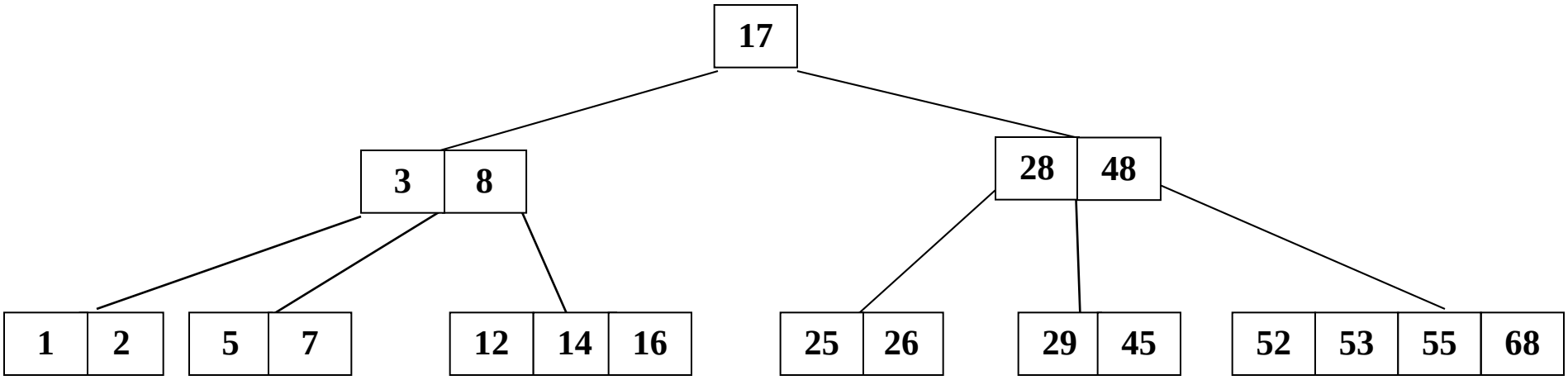
- Cho B-cây bậc 5: tạo node mới → hạ độ cao cây



# Xóa phần tử - Ví dụ



- Cho B-cây bậc 5 như dưới đây:
  - Xóa 28 rồi xóa 48



# Ý nghĩa



- B-cây là dạng cây cân bằng, phù hợp với việc lưu trữ trên đĩa.
- B-cây tiêu tốn số phép truy xuất đĩa tối thiểu cho các thao tác.
- Có thể quản lý số phần tử rất lớn.

# Ứng dụng



- Xây dựng cấu trúc chỉ mục trong các hệ quản trị cơ sở dữ liệu

# Hỏi – Đáp

# Cây B+

B+ Tree

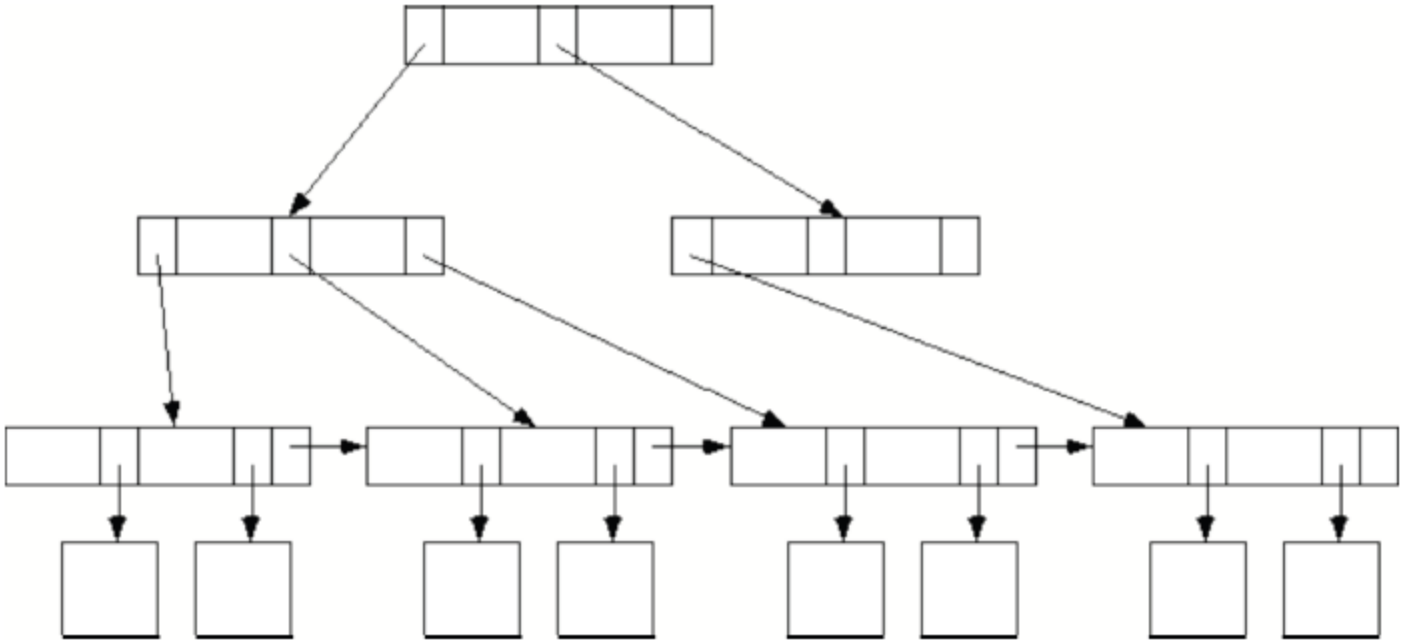
# Đặc điểm



- Thỏa tính chất của B-Cây.
- Cấu trúc của node lá và node trong có điểm khác nhau:
  - Thông tin chỉ mục chứa ở các node trong (không phải là node lá).
  - Con trỏ đến vùng dữ liệu của các khóa CHỈ được chứa ở node lá.
- Giữa các node lá thường có các liên kết qua lại (giống danh sách liên kết)

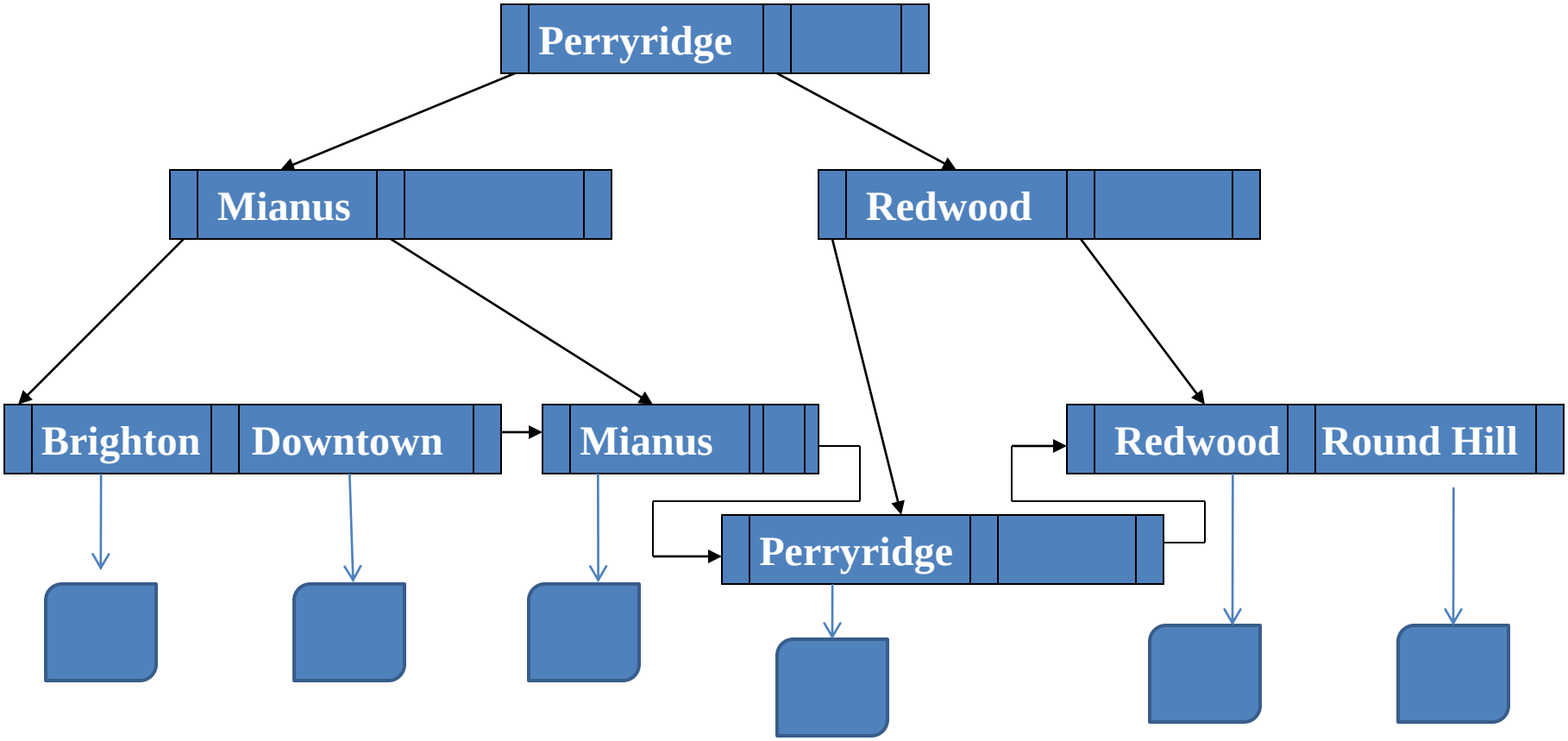
# Ví dụ





# Ví dụ

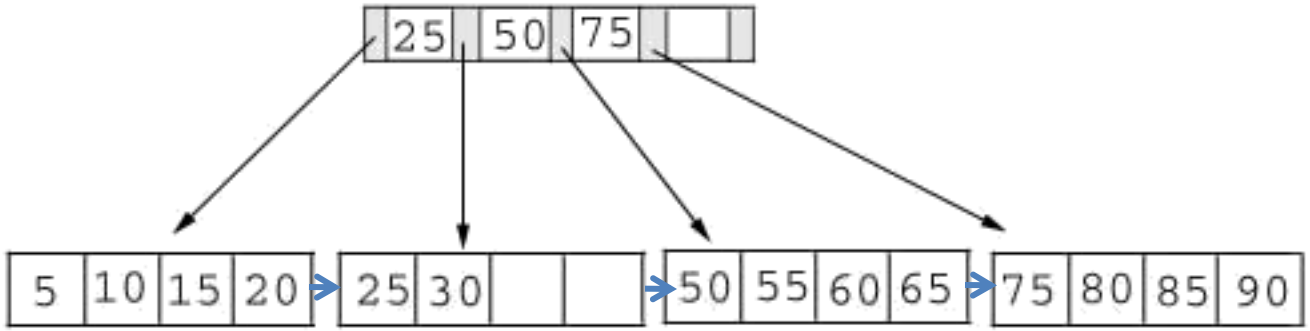



Perryridge
Mianus
Redwood
Brighton
Downtown
Mianus
Redwood
Round Hill
Perryridge

# Các thao tác trên cây B+



- Thao tác thêm, xóa phần tử:
  - Thực hiện gần giống B-cây.
  - Lưu ý: các khóa nằm ở cây con phải có thể có giá trị **lớn hơn hoặc bằng** khóa trên node cha tương ứng.



# Tập tin chỉ mục IDX

Tập tin chỉ mục của hệ quản trị cơ sở dữ liệu FoxPro

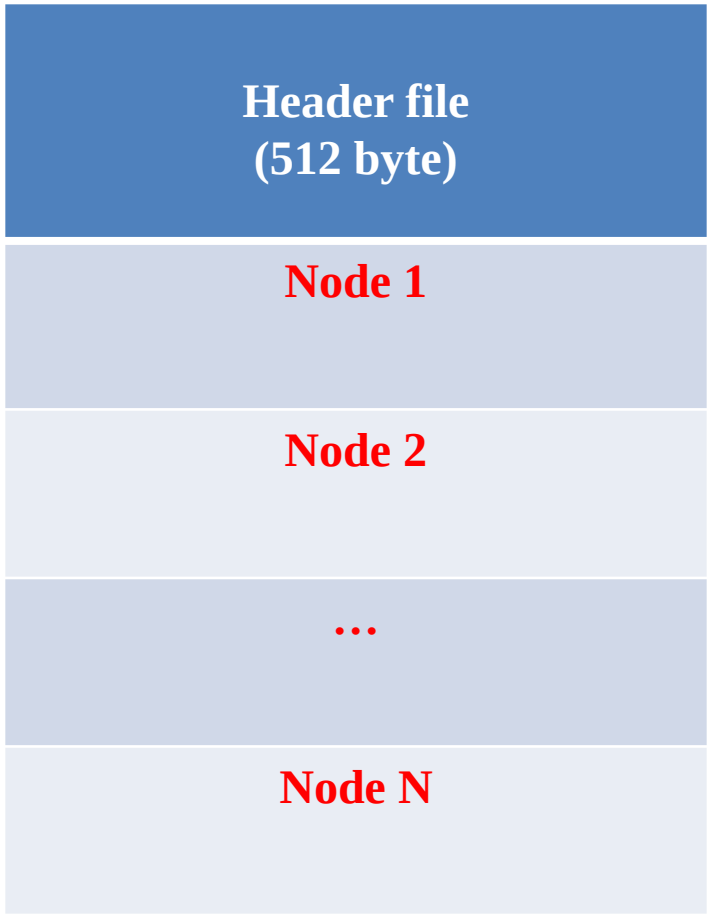
# Tập tin chỉ mục IDX



- Tập tin định dạng .IDX của hệ quản trị cơ sở dữ liệu Foxpro.
- Lưu trữ chỉ mục cho dữ liệu sử dụng cấu trúc cây B+.
- Tham khảo: **Index File Structrue (.idx)** trong MSDN

# Cấu trúc tập tin

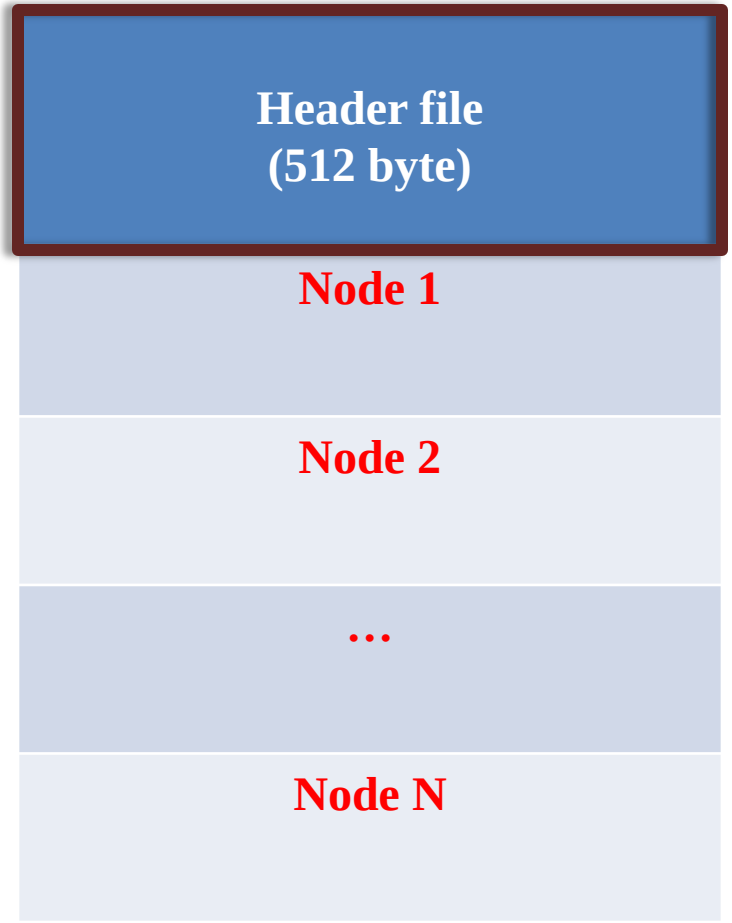





- Gồm 2 phần chính:
  - Header
  - Dữ liệu: tập hợp các node của cây B+
- Kích thước:
  - Header: 512 byte
  - Node: 512 byte

# Thông tin header

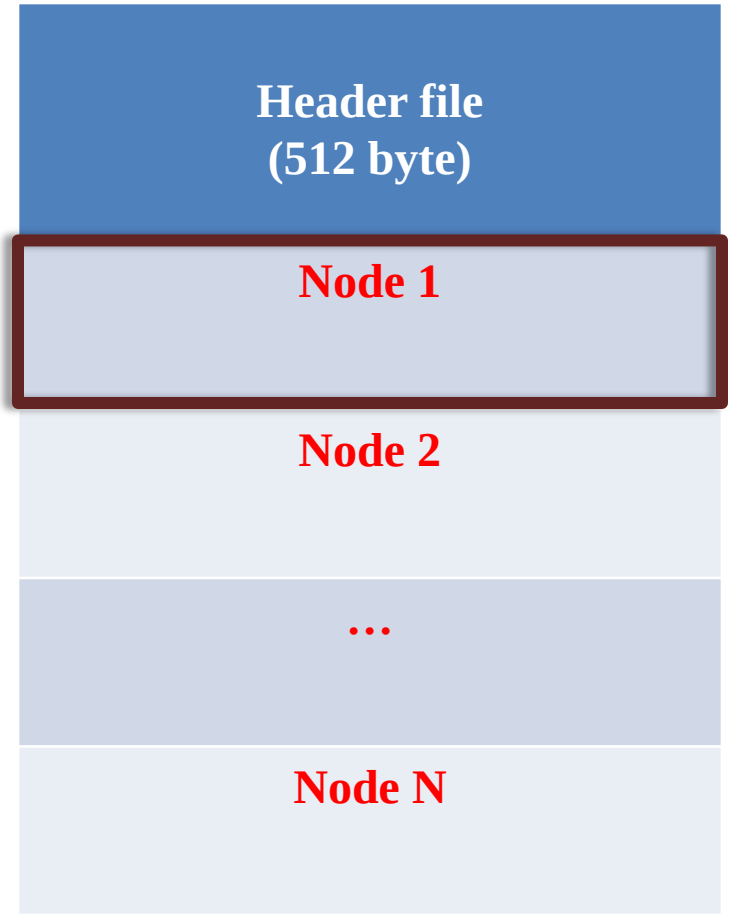





| | |
|---|---|
| **00 – 03** | **Pointer to the root node** |
| **04 – 07** | Pointer to the free node list ( -1 if not present) |
| **08 – 11** | Pointer to the end of file (file size) |
| **12 – 13** | **Length of key** |
| **14** | Index options (any of the following numeric values or their sums):<br>1 – a unique index<br>8 – index has FOR clause |
| **15** | Index signature (for future use) |
| **16 – 235** | Key expression (uncompiled; up to 220 characters) |
| **236 – 455** | FOR expression (uncompiled; up to 220 characters ending with a null value byte) |
| **456 – 511** | Unused |

# Cấu trúc node

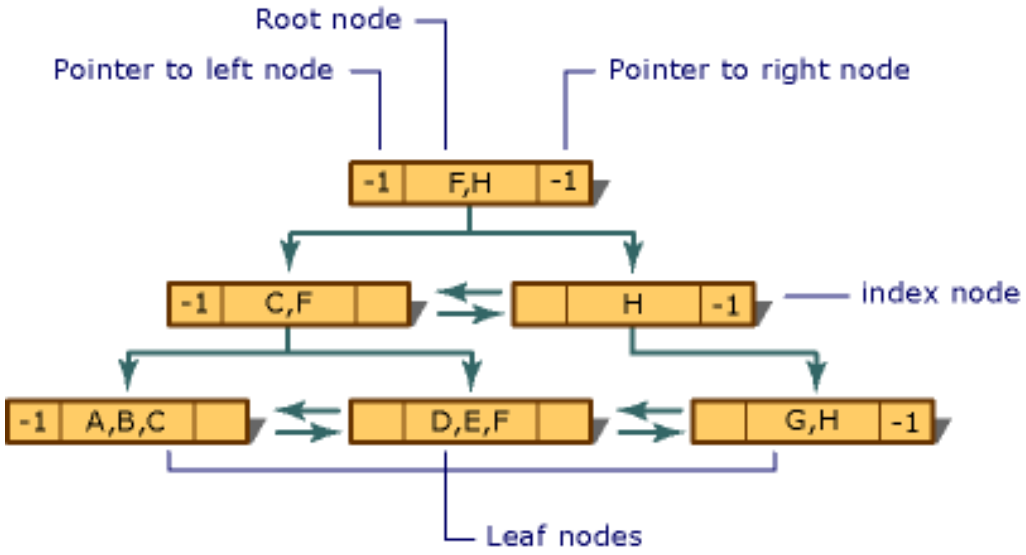





| | |
|---|---|
| **00 – 01** | Node attributes (any of the following numeric values or their sums):<br>0 – index node; 1 – root node; 2 – leaf node |
| **02 – 03** | Number of keys present (0, 1 or many) |
| **04 – 07** | Pointer to the node directly to left of the current node (on same level; -1 if not) |
| **08 – 11** | Pointer to the node directly to right of the current node (on same level; -1 if not) |
| **12 – 511** | Up to 500 characters containing the key value for the length of the key with a four-byte hexadecimal number stored in normal left-to-right format:<br>If the node is a leaf (attribute = 02 or 03) then the four bytes contain an actual table number in hexadecimal format; otherwise, the 4 bytes contain an intra-index pointer. |

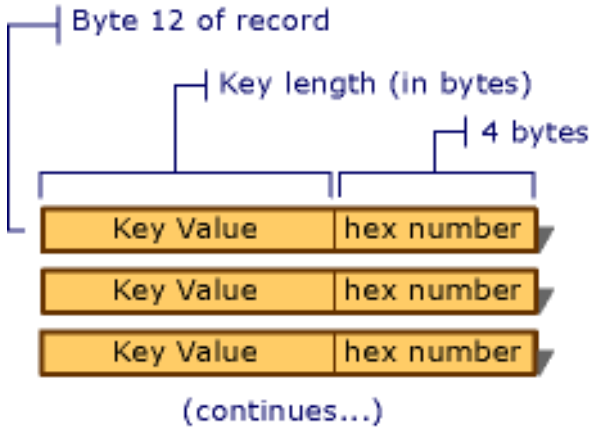
# Ví dụ






**Cấu trúc cây B+ của tập tin IDX**

**Phần tử chỉ mục của node**



# Hỏi và Đáp

# Chỉ mục

# Giới thiệu



**Hãy nêu những ví dụ thực tế đã biết về chỉ mục?**

# Giới thiệu



## Lợi ích của chỉ mục?

# Giới thiệu



- Các khuyết điểm lớn với cách lưu trữ trên tập tin tuần tự:
  - Tốc độ tìm kiếm chậm
  - Không an toàn
  - Không có nhiều tiêu chí tìm kiếm khác nhau

# Định nghĩa



- Chỉ mục:
  - Là một tập hợp các phần tử chỉ mục (Index entry)
  - Được tổ chức theo một qui tắc xác định nhằm tăng tốc độ tìm kiếm trên bảng dữ liệu

- Phần tử chỉ mục: là cấu trúc gồm tối thiểu 2 thuộc tính:
  - Khóa tìm kiếm: một (hay nhiều) thuộc tính, được dùng để tìm kiếm các mẫu tin trong bảng dữ liệu
  - Tham chiếu: con trỏ tham chiếu đến vị trí mẫu tin tương ứng với Khóa tìm kiếm

# Tính chất của chỉ mục



- Kích thước nhỏ hơn nhiều so với bảng dữ liệu
- Thực hiện tìm kiếm nhanh
- Cho phép ‘nhìn’ dữ liệu ở nhiều góc độ khác nhau. (Tìm kiếm trên nhiều khóa tìm kiếm khác nhau)

# Các loại chỉ mục

# Các loại chỉ mục

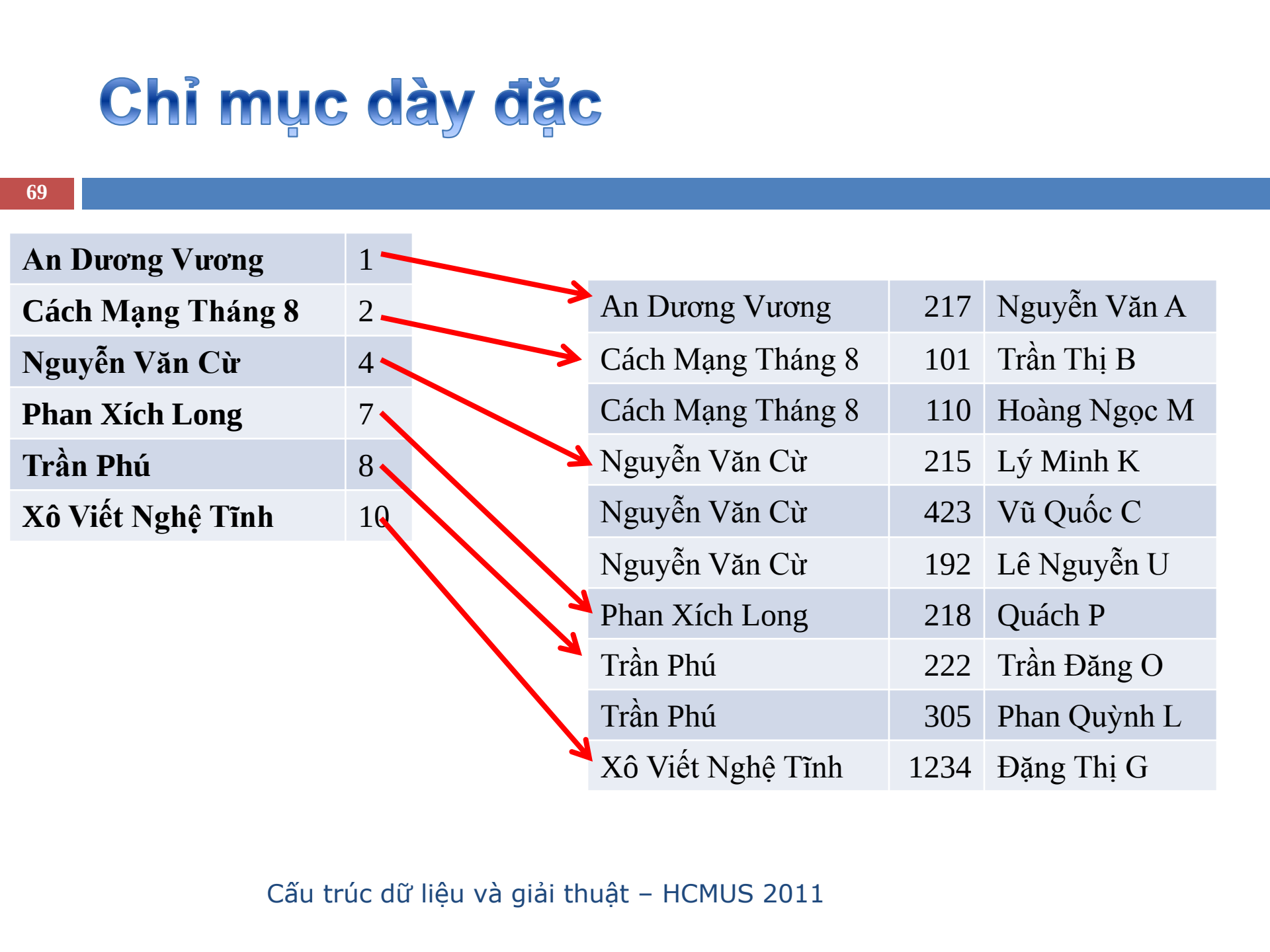


- Chỉ mục dày đặc (Dense index):
  - Các phần tử chỉ mục được tạo riêng cho mỗi khóa tìm kiếm
  - Trong trường hợp trùng khóa, tham chiếu sẽ trỏ đến bản ghi đầu tiên
- Chỉ mục thưa (Sparse index)
  - Tạo phần tử chỉ mục cho một nhóm các khóa tìm kiếm.
  - Sử dụng trong trường hợp khóa tìm kiếm đã được sắp xếp

# Chỉ mục dày đặc



| | |
|---|---|
| **An Dương Vương** | 1 |
| **Cách Mạng Tháng 8** | 2 |
| **Nguyễn Văn Cừ** | 4 |
| **Phan Xích Long** | 7 |
| **Trần Phú** | 8 |
| **Xô Viết Nghệ Tĩnh** | 10 |



| | | |
|---|---|---|
| An Dương Vương | 217 | Nguyễn Văn A |
| Cách Mạng Tháng 8 | 101 | Trần Thị B |
| Cách Mạng Tháng 8 | 110 | Hoàng Ngọc M |
| Nguyễn Văn Cừ | 215 | Lý Minh K |
| Nguyễn Văn Cừ | 423 | Vũ Quốc C |
| Nguyễn Văn Cừ | 192 | Lê Nguyễn U |
| Phan Xích Long | 218 | Quách P |
| Trần Phú | 222 | Trần Đăng O |
| Trần Phú | 305 | Phan Quỳnh L |
| Xô Viết Nghệ Tĩnh | 1234 | Đặng Thị G |

# Chỉ mục thưa



| | |
|---|---|
| **An Dương Vương** | 1 |
| **Nguyễn Văn Cừ** | 4 |
| **Phan Xích Long** | 7 |

| | | |
|---|---|---|
| An Dương Vương | 217 | Nguyễn Văn A |
| Cách Mạng Tháng 8 | 101 | Trần Thị B |
| Cách Mạng Tháng 8 | 110 | Hoàng Ngọc M |
| Nguyễn Văn Cừ | 215 | Lý Minh K |
| Nguyễn Văn Cừ | 423 | Vũ Quốc C |
| Nguyễn Văn Cừ | 192 | Lê Nguyễn U |
| Phan Xích Long | 218 | Quách P |
| Trần Phú | 222 | Trần Đăng O |
| Trần Phú | 305 | Phan Quỳnh L |
| Xô Viết Nghệ Tĩnh | 1234 | Đặng Thị G |

# Nhận xét



- Về cơ bản, chỉ mục dày đặc cho kết quả tìm kiếm nhanh hơn chỉ mục thưa.
- Chỉ mục thưa sử dụng ít không gian lưu trữ hơn.

->Kết hợp giữa chỉ mục dày và chỉ mục thưa: chỉ mục nhiều tầng.

# Chỉ mục nhiều tầng



- Vấn đề:
  - Ngay cả khi sử dụng chỉ mục thưa, tập tin chỉ mục cũng có thể có kích thước lớn.
    - Giả sử: dữ liệu có 100.000 bản ghi.
    - Đánh chỉ mục thưa cho từng khối 10 bản ghi

    ⇒Tổng cộng có 10.000 phần tử chỉ mục

  - Nếu kích thước chỉ mục không nằm thể nằm trọn trong bộ nhớ trong => tốn chi phí cho việc đọc tập tin.

    (Khoảng 14 lần cho dữ liệu 10.000 phần tử chỉ mục)

# Chỉ mục nhiều tầng



- Mục tiêu:
  - Giảm thiểu số lần truy cập bộ nhớ phụ
- Giải pháp:
  - Tạo **chỉ mục chính** cho dữ liệu (lưu trên bộ nhớ phụ)
  - Tạo **chỉ mục thưa** trên **chỉ mục chính** vừa tạo.
  - Chỉ mục thưa sẽ lưu trữ trên bộ nhớ chính.
  - Nếu kích thước của chỉ mục thưa lớn thì có thể tạo thêm chỉ mục thưa trên đấy.

# Chỉ mục nhiều tầng