Map & Tree

Môn học: Advanced Programming with JavaScript

Mục tiêu

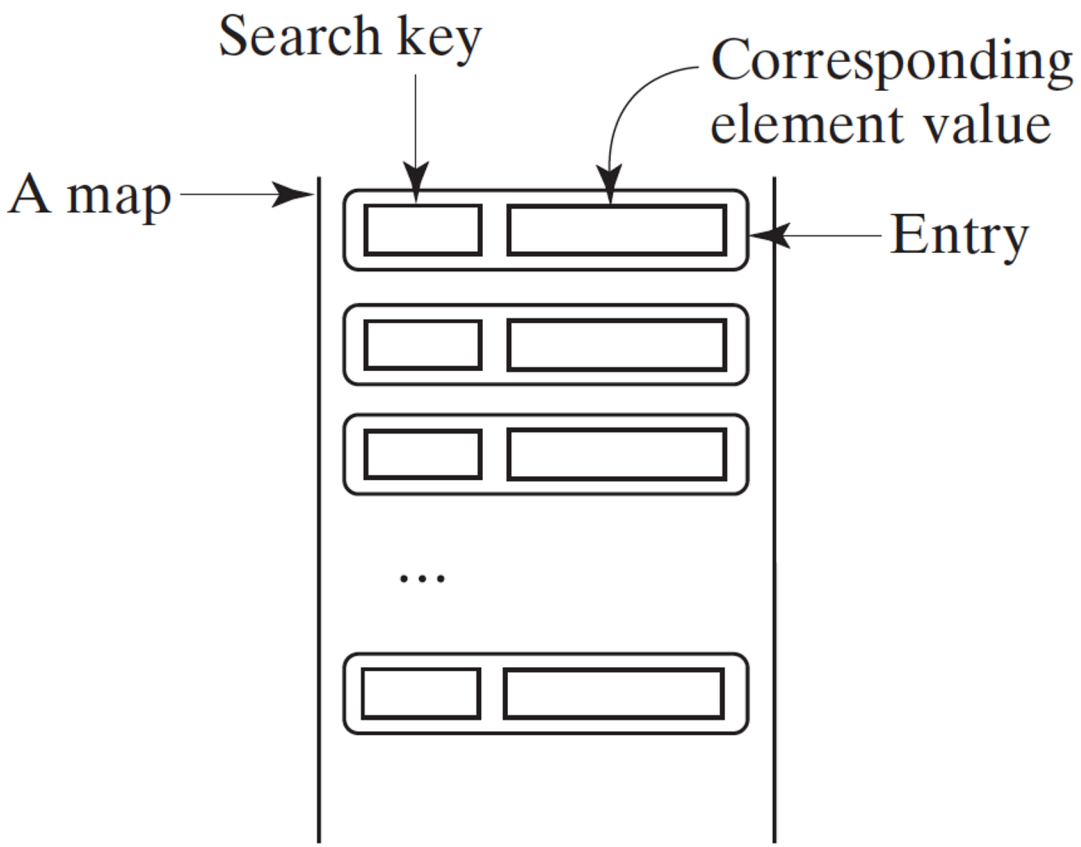
- Trình bày được cấu trúc dữ liệu Map
- Sử dụng được Map trong JavaScript
- Trình bày được cấu trúc dữ liệu Tree
- Triển khai được cấu trúc dữ liệu Tree

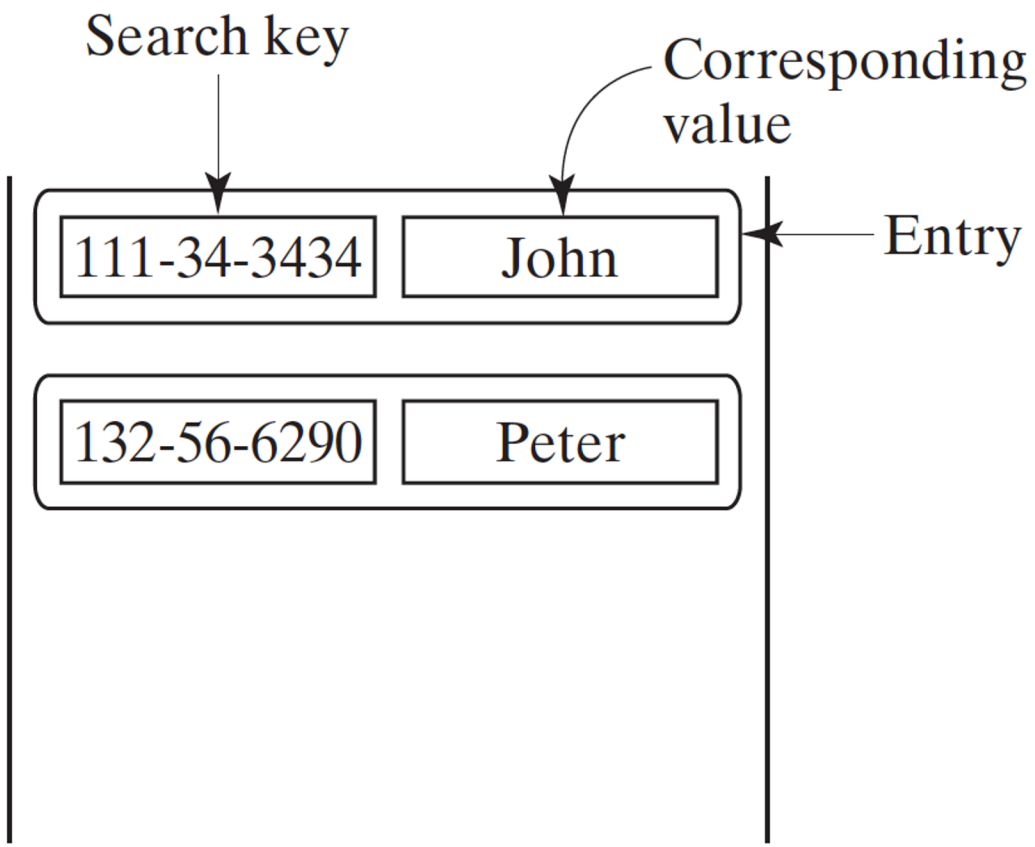
Map

Map

- Map là cấu trúc dữ liệu lưu trữ danh sách các cặp key/value
- Map cho phép thực hiện các hành động truy xuất, xóa và cập nhật các cặp key/value rất hiệu quả thông qua việc sử dụng key
- Map không cho phép 2 key trùng lặp
- Mỗi key tương ứng với một value
- Một cặp key-value được gọi là một Entry

Map: Ví dụ


Search key
Corresponding element value
A map
Entry
...


Search key
Corresponding value
Entry
111-34-3434
John
132-56-6290
Peter

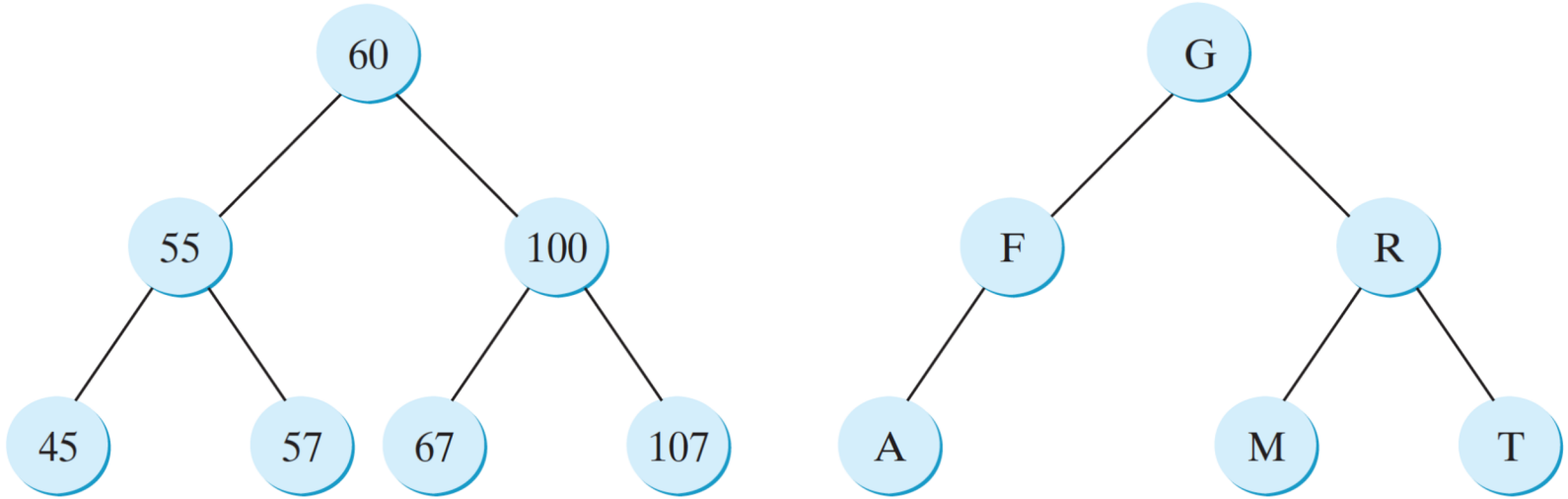
Ví dụ sử dụng Map

```
let nameAgeMapping = new Map<string, number>();

//1. Add entries
nameAgeMapping.set("Lokesh", 37);
nameAgeMapping.set("Raj", 35);
nameAgeMapping.set("John", 40);

//2. Get entries
let age = nameAgeMapping.get("John");      // age = 40

//3. Check entry by Key
nameAgeMapping.has("Lokesh");              // true
nameAgeMapping.has("Brian");               // false

//4. Size of the Map
let count = nameAgeMapping.size;           // count = 3
```

Tree

Binary Tree

- Tree lưu trữ dữ liệu trên các node
- Các node có mối quan hệ cha-con, node trên cùng được gọi là node gốc (root node)
- Binary Tree (Cây nhị phân) là cây mà mỗi node có 0, 1 hoặc 2 cây con (subtree)
- 2 cây con được gọi lần lượt là left-subtree (cây con trái) và right-subtree (cây con phải)

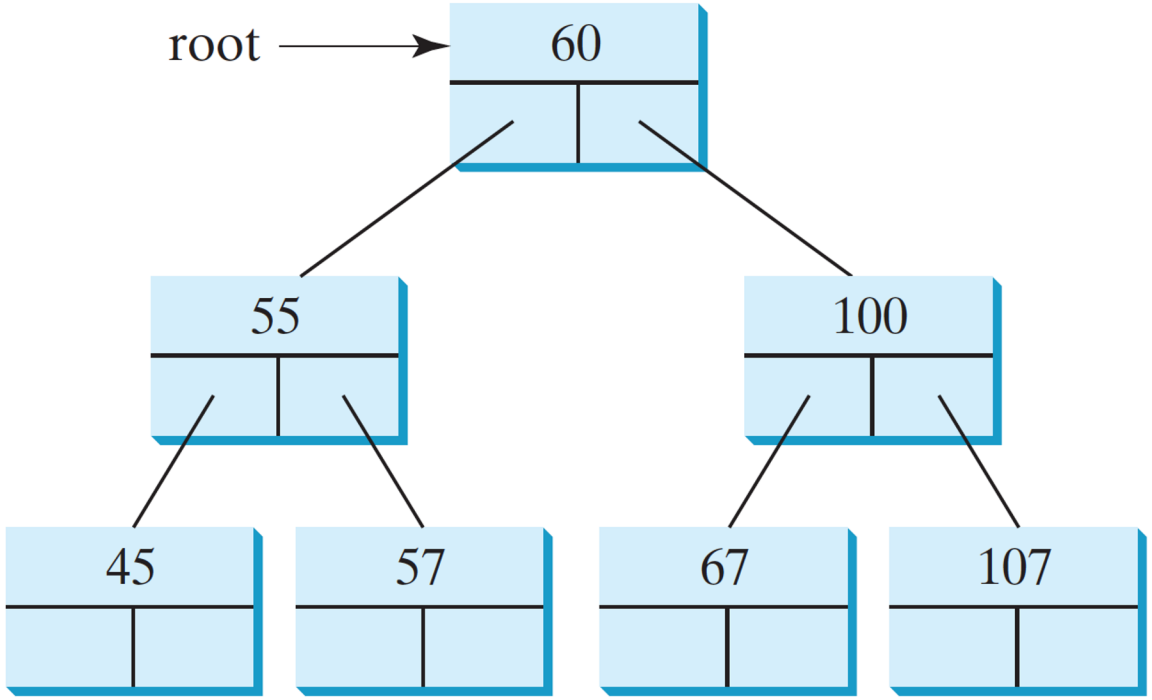


Các khái niệm

- Độ dài của đường đi (length of the path) là số lượng các cạnh
- Chiều sâu của node là độ dài của đường đi tính từ root node đến node đó
- Node anh em (sibling) là các node có cùng node cha
- Node không có node con thì gọi là node lá (leaf node)
- Chiều cao của cây là độ dài của đường đi từ node gốc đến node lá

Binary Search Tree (BST)

- Binary Search Tree (Cây tìm kiếm nhị phân) được biểu diễn bằng một tập các node liên kết với nhau
- Mỗi node chứa dữ liệu và 2 liên kết: 1 liên kết sang node con bên trái và. 1 liên kết sang node con bên phải



Triển khai BST

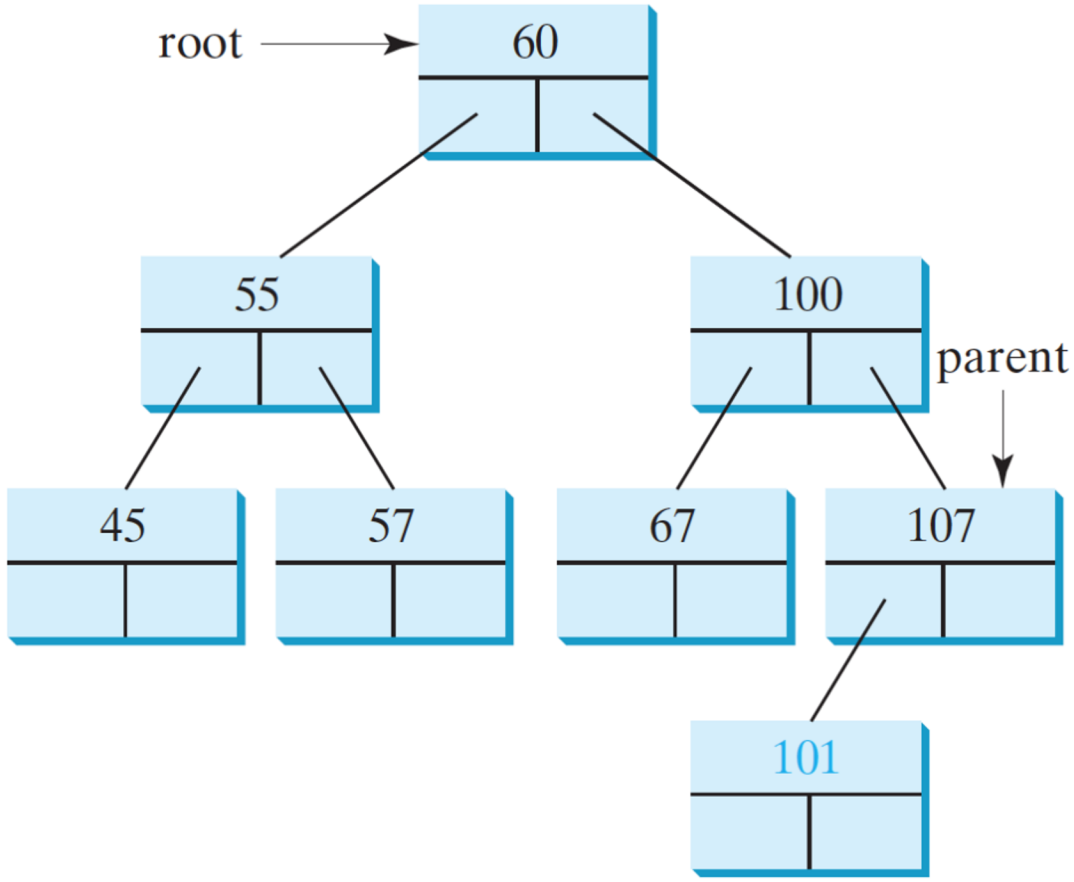
```
class TreeNode<T> {
    data: T;
    leftNode?: TreeNode<T>;
    rightNode?: TreeNode<T>;

    constructor(data: T) {
        this.data = data;
    }
}
```

```
const root = new TreeNode<number>(60);
root.leftNode = new TreeNode<number>(55);
root.rightNode = new TreeNode<number>(100);
```

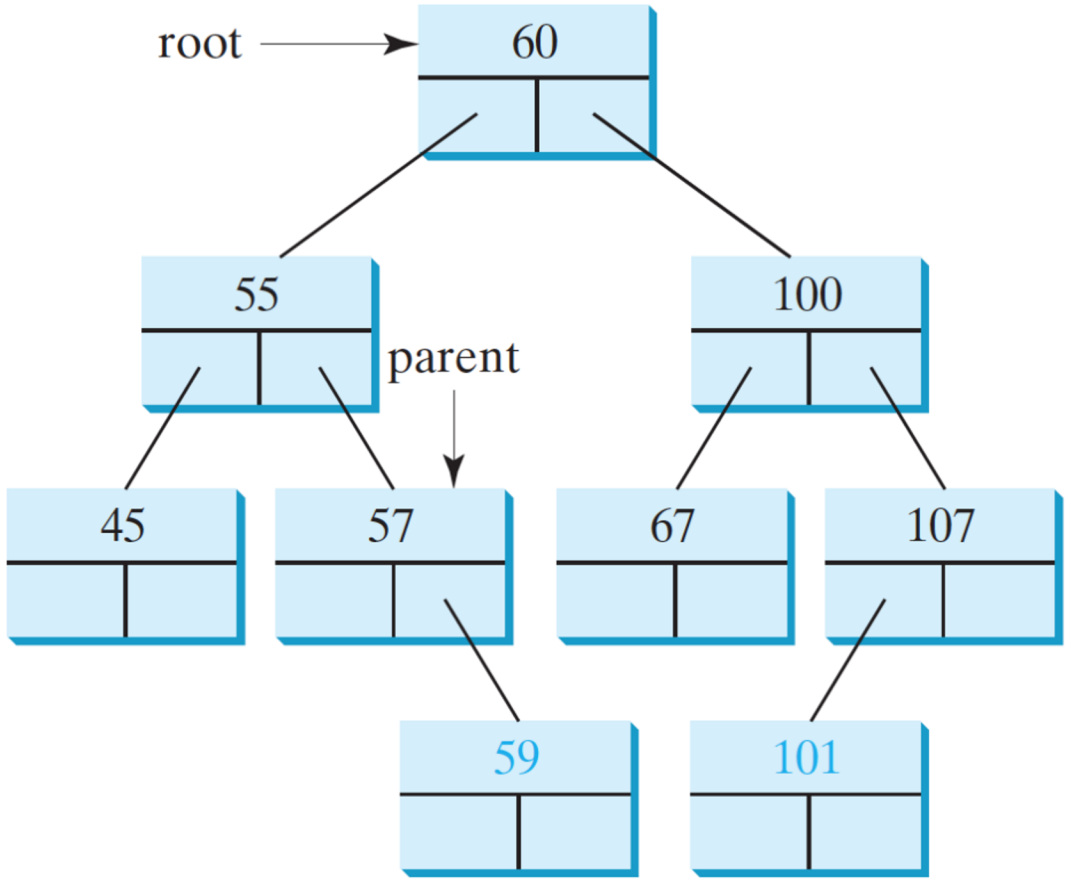
Tìm kiếm trên BST

```
search(data: T): TreeNode<T> | undefined {
    if (!this.root) return undefined;

    let current = this.root;

    while (this.comparator(data, current.data) !== 0) {
        if (this.comparator(data, current.data) === 1) {
            if (!current.rightNode) return;

            current = current.rightNode;
        } else {
            if (!current.leftNode) return;

            current = current.leftNode;
        }
    }

    return current;
}
```

Thêm phần tử vào BST



Chèn giá trị 101 vào BST

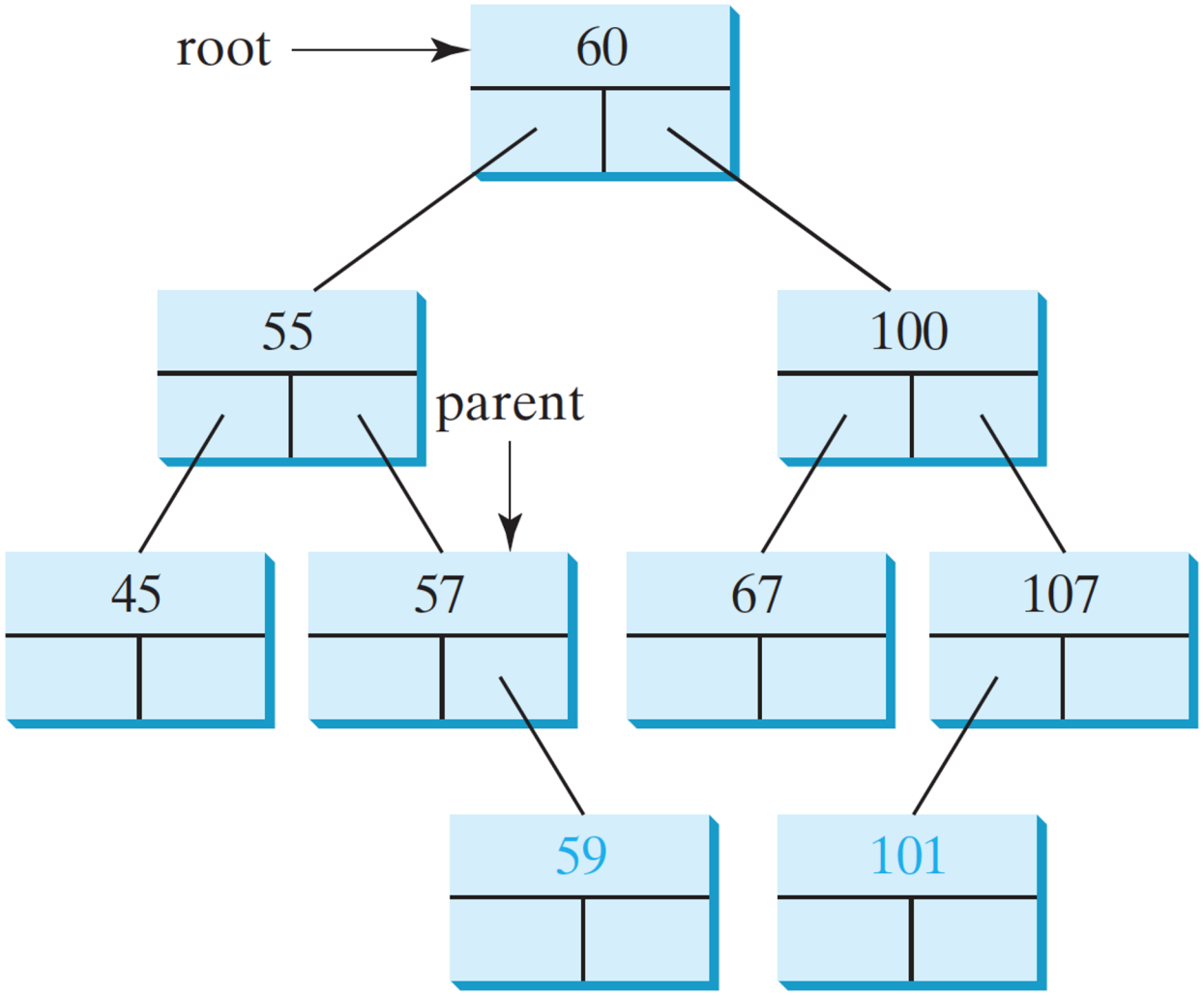


Chèn giá trị 59 vào BST

Duyệt cây

- Duyệt cây là thao tác đi qua từng node của cây, mỗi node được đi qua một lần duy nhất
- Có nhiều cách để duyệt cây, ví dụ:
  - Inorder: Duyệt theo thứ tự cây con trái->node hiện tại->cây con phải
  - Postorder: Duyệt theo thứ tự cây con trái->cây con phải->node hiện tại
  - Preorder: Duyệt theo thứ tự node hiện tại->cây con trái->cây con phải
  - Breath-first: Duyệt lần lượt theo từng level
  - ...

Duyệt cây: ví dụ



Inorder: 45 55 57 59 60 67 100 101 107

Postorder: 45. 59 57 55 67 101 107 100 60

Preorder: 60 55 45 57 59 100 67 107 101

Breath-first: 60 55 100 45 57 67 107 59 101

Tổng kết

- Map lưu trữ dữ liệu theo từng cặp key/value
- Mỗi cặp key/value được gọi là một Entry
- Thao tác truy xuất sử dụng key có hiệu suất cao
- Tree lưu trữ dữ liệu theo các node có liên kết cha-con với nhau
- Cây nhị phân là cây mà mỗi node có thể có 0, 1 hoặc 2 cây con