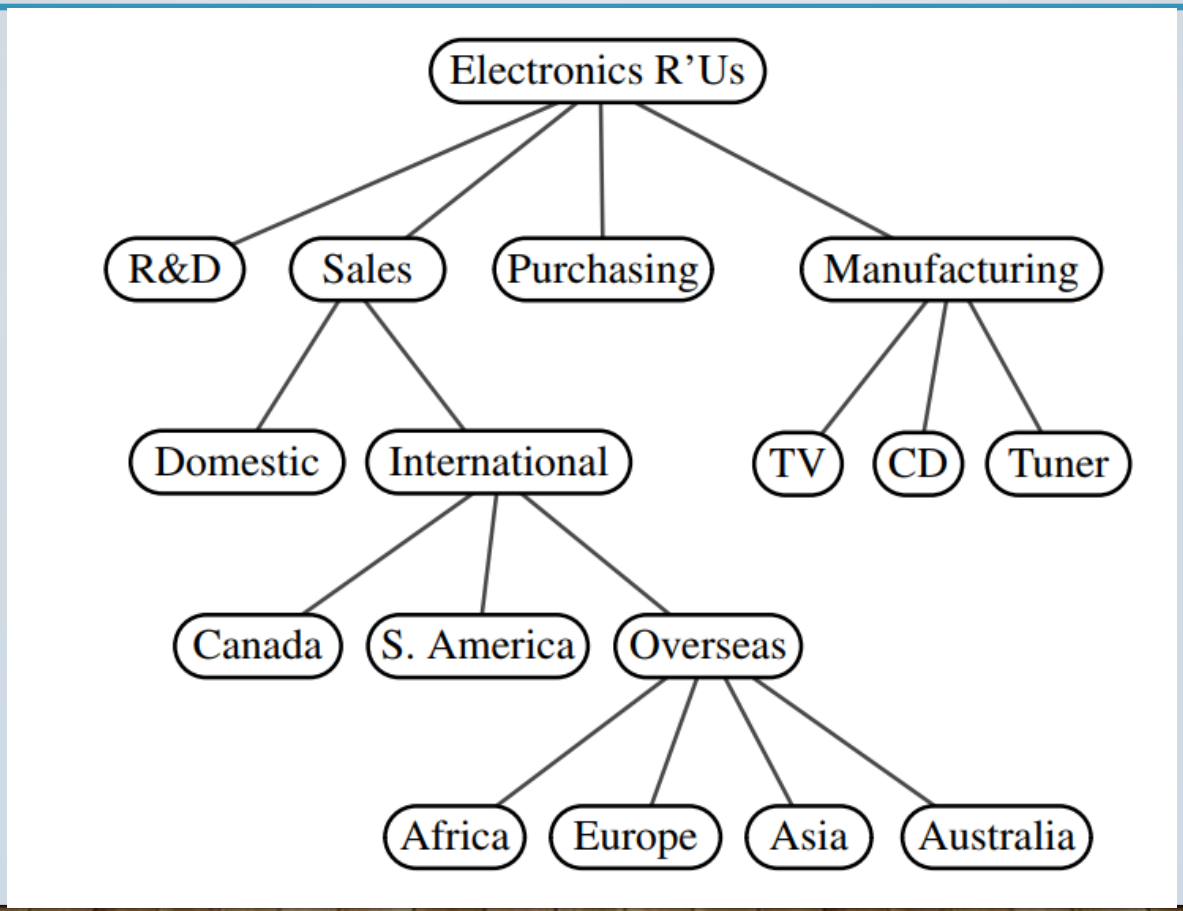
# Tree

TUAN NGUYEN

# Outline

- Linked List
- Double Linked List

# Tree


Electronics R'Us
R&D
Sales
Purchasing
Manufacturing
Domestic
International
TV
CD
Tuner
Canada
S. America
Overseas
Africa
Europe
Asia
Australia

# Tree Definition

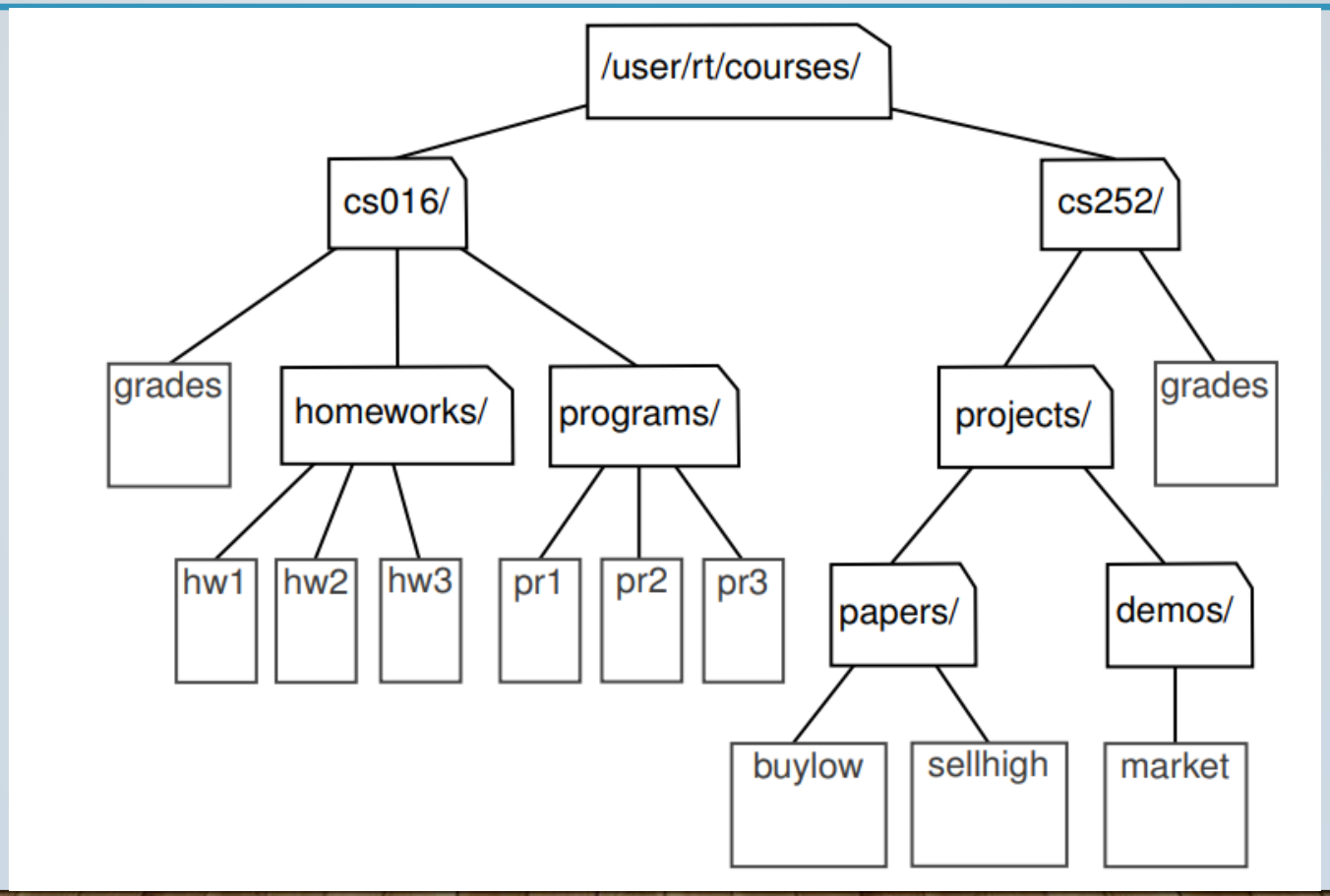
We define a tree T as a set of nodes storing elements such that the nodes have a parent-child relationship that satisfies the following properties:

- If T is nonempty, it has a special node, called the root of T , that has no parent.
- Each node v of T different from the root has a unique parent node w; every node with parent w is a child of w.

Other node relationships

- Two nodes that are children of the same parent are siblings.
- A node v is external if v has no children.

# Tree definition (1)

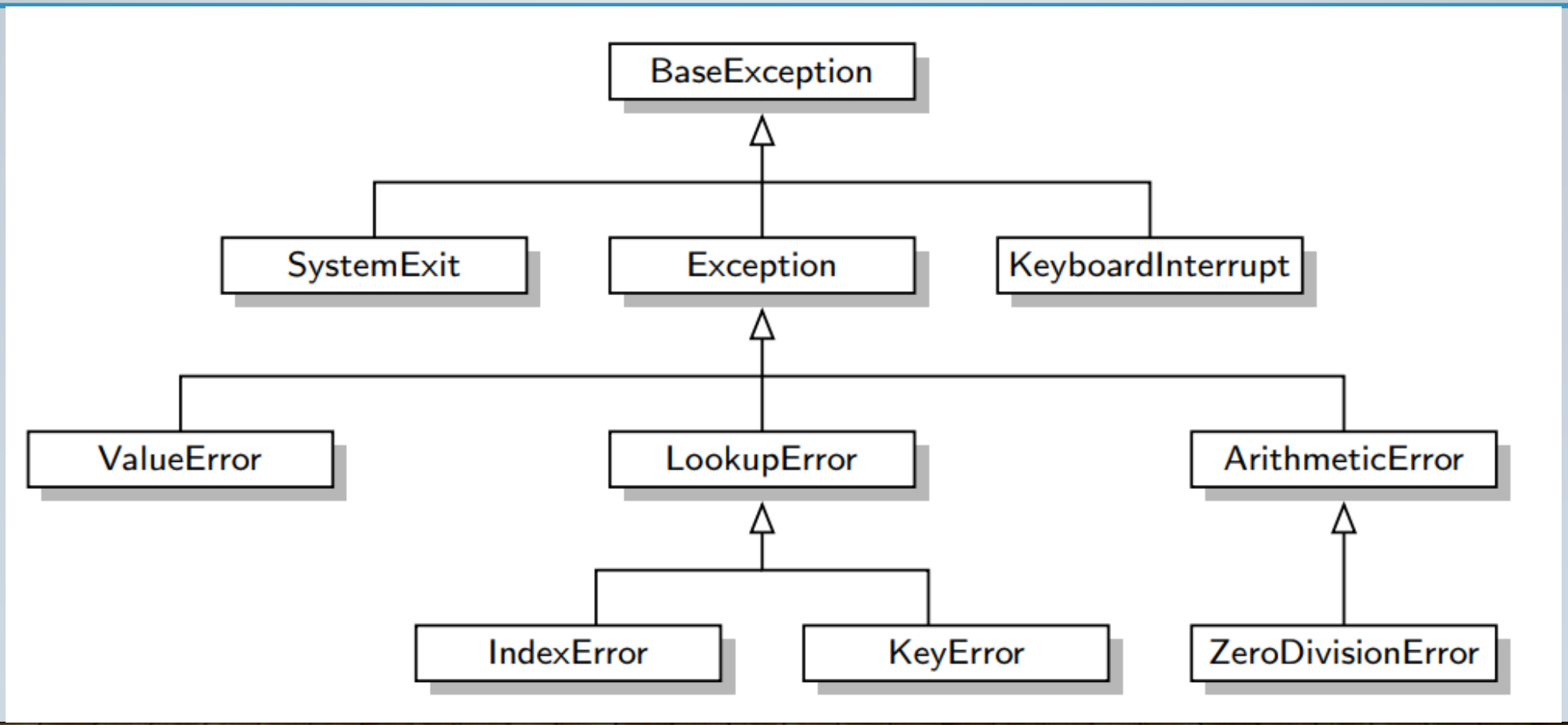


# Edges and Paths

- An edge of tree T is a pair of nodes (u,v) such that u is the parent of v, or vice versa.

- A path of T is a sequence of nodes such that any two consecutive nodes in the sequence form an edge.

# Example


BaseException
SystemExit
Exception
KeyboardInterrupt
ValueError
LookupError
ArithmeticError
IndexError
KeyError
ZeroDivisionError

# Tree Abstract Data Type

| | |
|---|---|
| **T.root( ):** | Return the position of the root of tree T, or None if T is empty. |
| **T.is_root(p):** | Return True if position p is the root of Tree T. |
| **T.parent(p):** | Return the position of the parent of position p, or None if p is the root of T. |
| **T.num_children(p):** | Return the number of children of position p. |
| **T.children(p):** | Generate an iteration of the children of position p. |
| **T.is_leaf(p):** | Return True if position p does not have any children. |
| **len(T):** | Return the number of positions (and hence elements) that are contained in tree T. |
| **T.is_empty( ):** | Return True if tree T does not contain any positions. |
| **T.positions( ):** | Generate an iteration of all *positions* of tree T. |
| **iter(T):** | Generate an iteration of all *elements* stored within tree T. |

# Depth

The depth of p is the number of ancestors of p, excluding p itself.

- If p is the root, then the depth of p is 0.
- Otherwise, the depth of p is one plus the depth of the parent of p.

```
def depth(self, p):
  """Return the number of levels separating Position p from the root."""
  if self.is_root(p):
    return 0
  else:
    return 1 + self.depth(self.parent(p))
```

# Height

The height of a position p in a tree T is also defined recursively:

- If p is a leaf, then the height of p is 0.
- Otherwise, the height of p is one more than the maximum of the heights of p's children.

The height of a nonempty tree T is equal to the maximum of the depths of its leaf positions.

# Height (1)

```
def _height1(self):                    # works, but O(n^2) worst-case time
  """Return the height of the tree."""
  return max(self.depth(p) for p in self.positions( ) if self.is_leaf(p))
```

```
def _height2(self, p):                 # time is linear in size of subtree
  """Return the height of the subtree rooted at Position p."""
  if self.is_leaf(p):
    return 0
  else:
    return 1 + max(self._height2(c) for c in self.children(p))
```

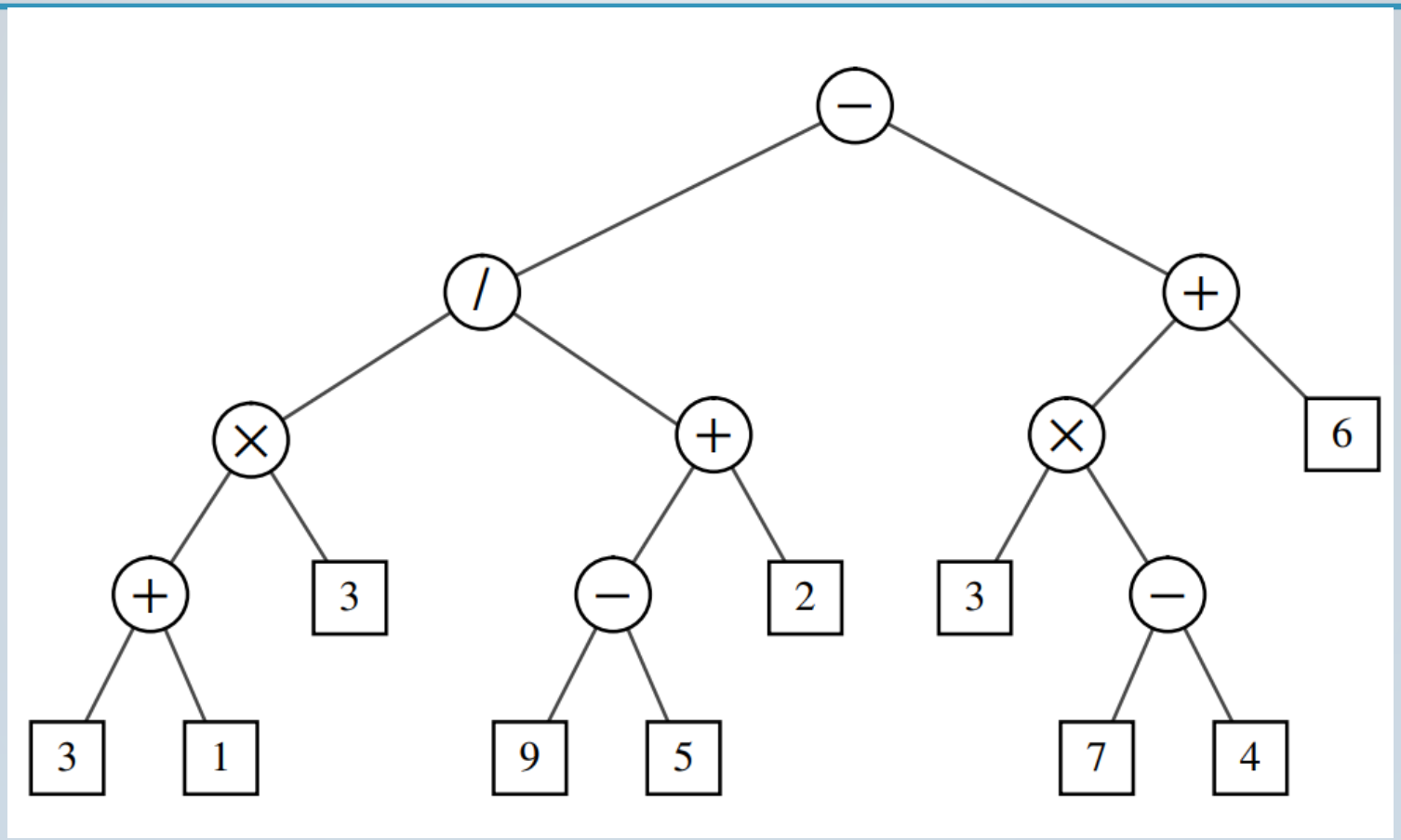
# Binary Tree

A binary tree is an ordered tree with the following properties:

- Every node has at most two children.
- Each child node is labeled as being either a left child or a right child.
- A left child precedes a right child in the order of children of a node.

The subtree rooted at a left or right child of an internal node v is called a left subtree or right subtree.

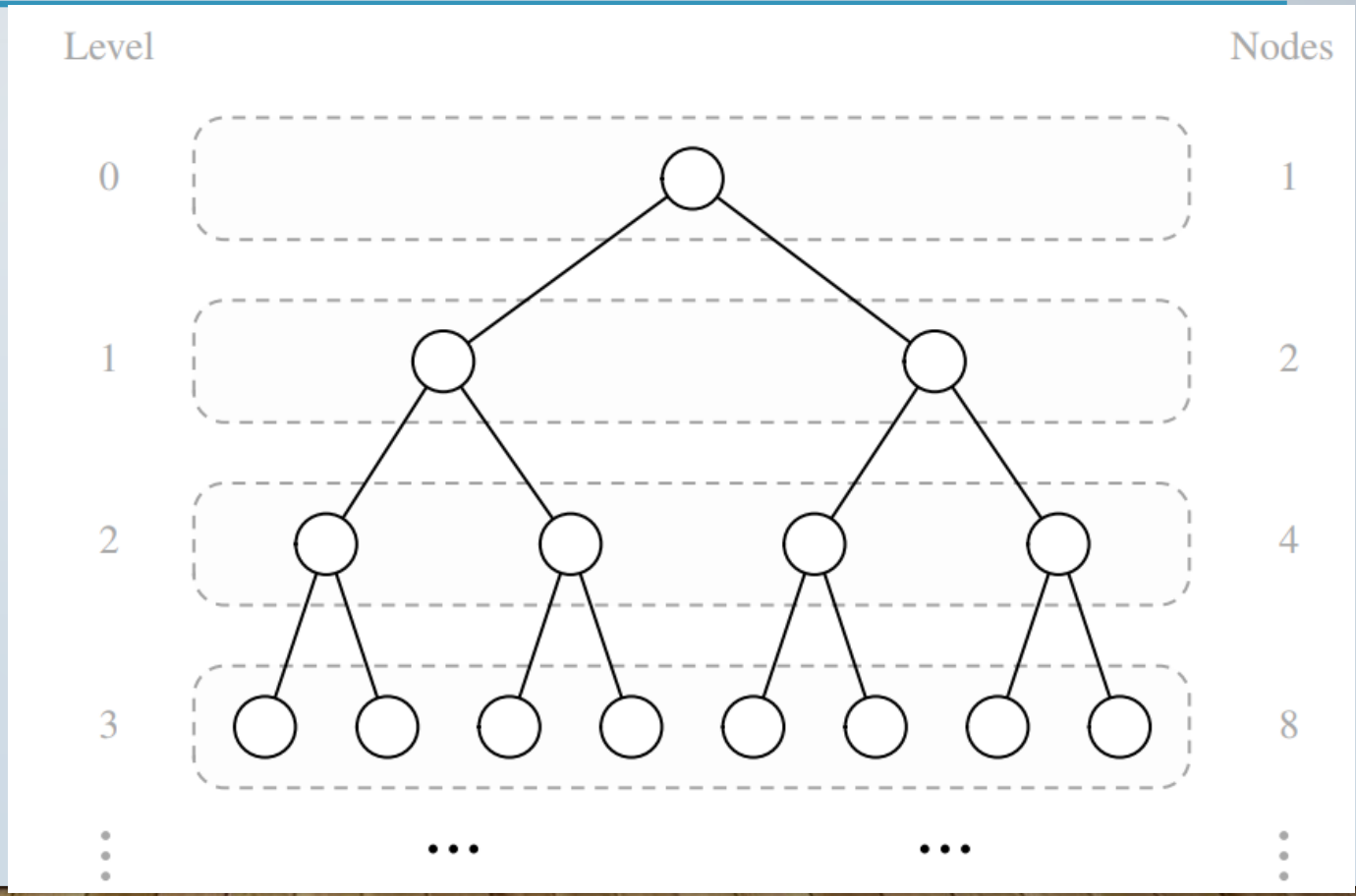
# Binary Tree (1)



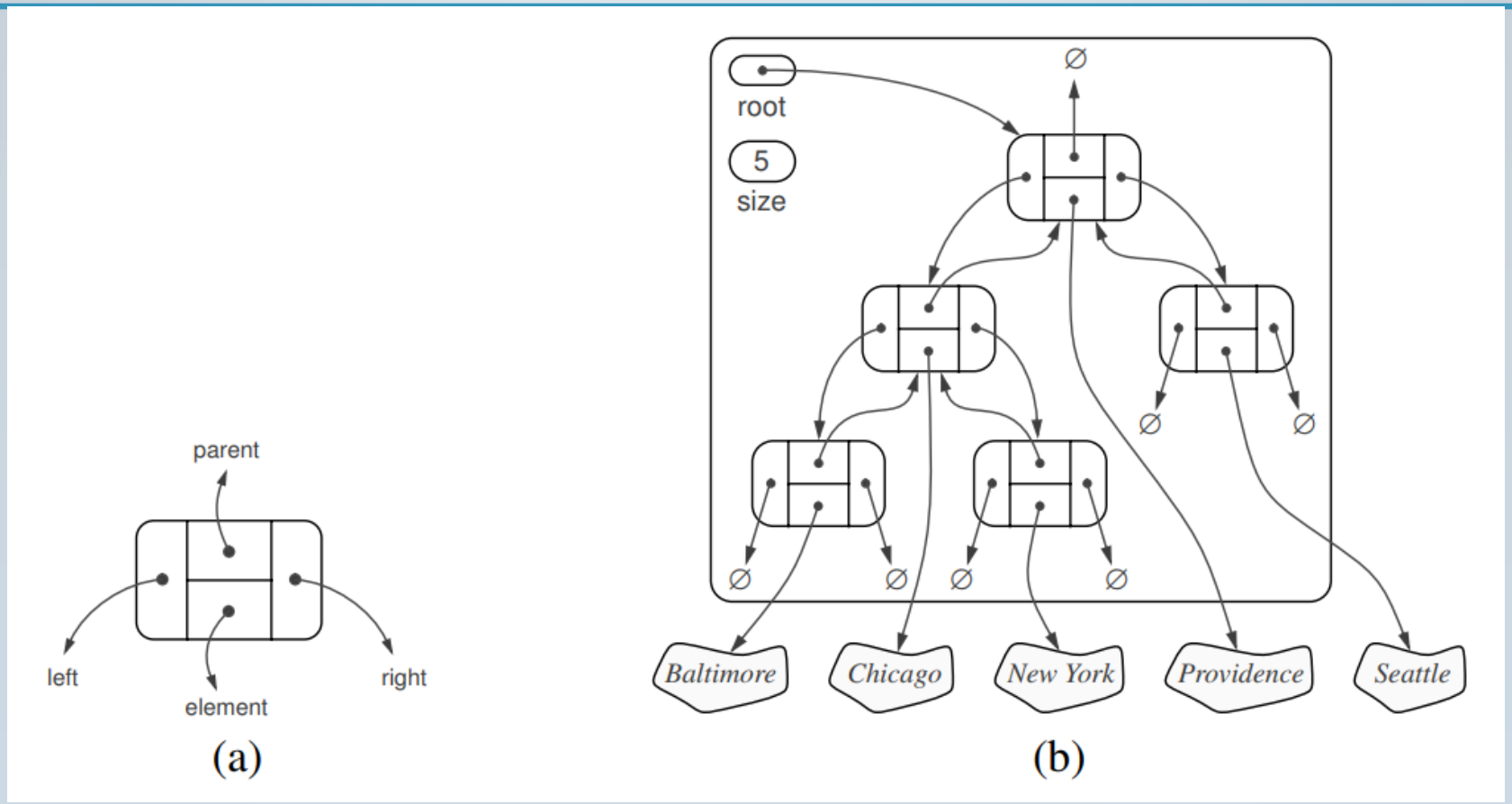
# Properties

In a binary tree, level 0 has at most one node (the root), level 1 has at most two nodes (the children of the root), level 2 has at most four nodes, and so on.

In general, level d has at most 2d nodes



# Binary Tree Implementation



# Binary Tree Implementation (1)

| | |
|---|---|
| **T.add_root(e):** | Create a root for an empty tree, storing e as the element, and return the position of that root; an error occurs if the tree is not empty. |
| **T.add_left(p, e):** | Create a new node storing element e, link the node as the left child of position p, and return the resulting position; an error occurs if p already has a left child. |
| **T.add_right(p, e):** | Create a new node storing element e, link the node as the right child of position p, and return the resulting position; an error occurs if p already has a right child. |
| **T.replace(p, e):** | Replace the element stored at position p with element e, and return the previously stored element. |
| **T.delete(p):** | Remove the node at position p, replacing it with its child, if any, and return the element that had been stored at p; an error occurs if p has two children. |
| **T.attach(p, T1, T2):** | Attach the internal structure of trees T1 and T2, respectively, as the left and right subtrees of leaf position p of T, and reset T1 and T2 to empty trees; an error condition occurs if p is not a leaf. |