Data Structures and Algorithms

ต้นไม้

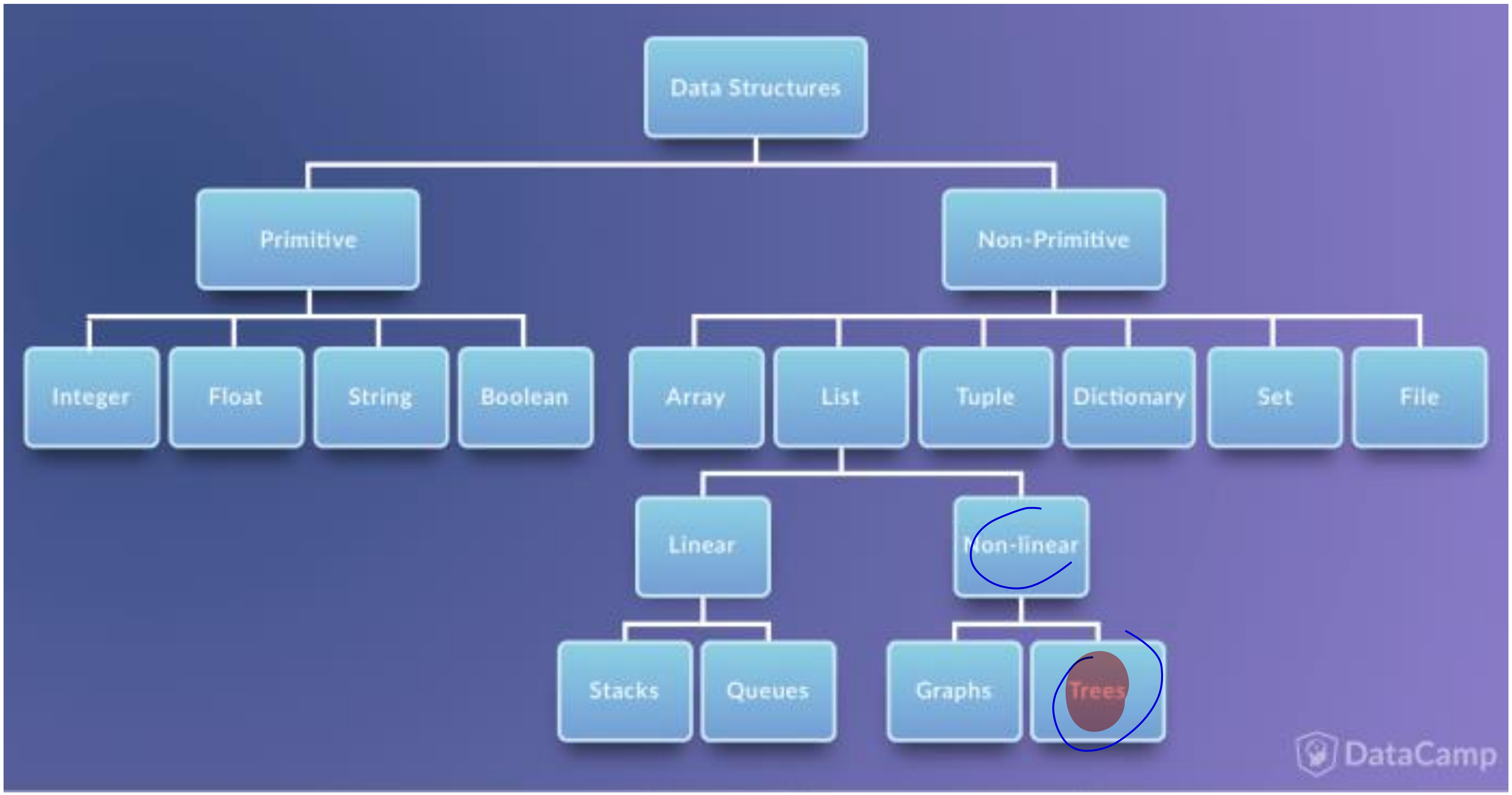
Lecture 16: Trees

Nopadon Juneam
Department of Computer Science
Kasetsart university

Outlines

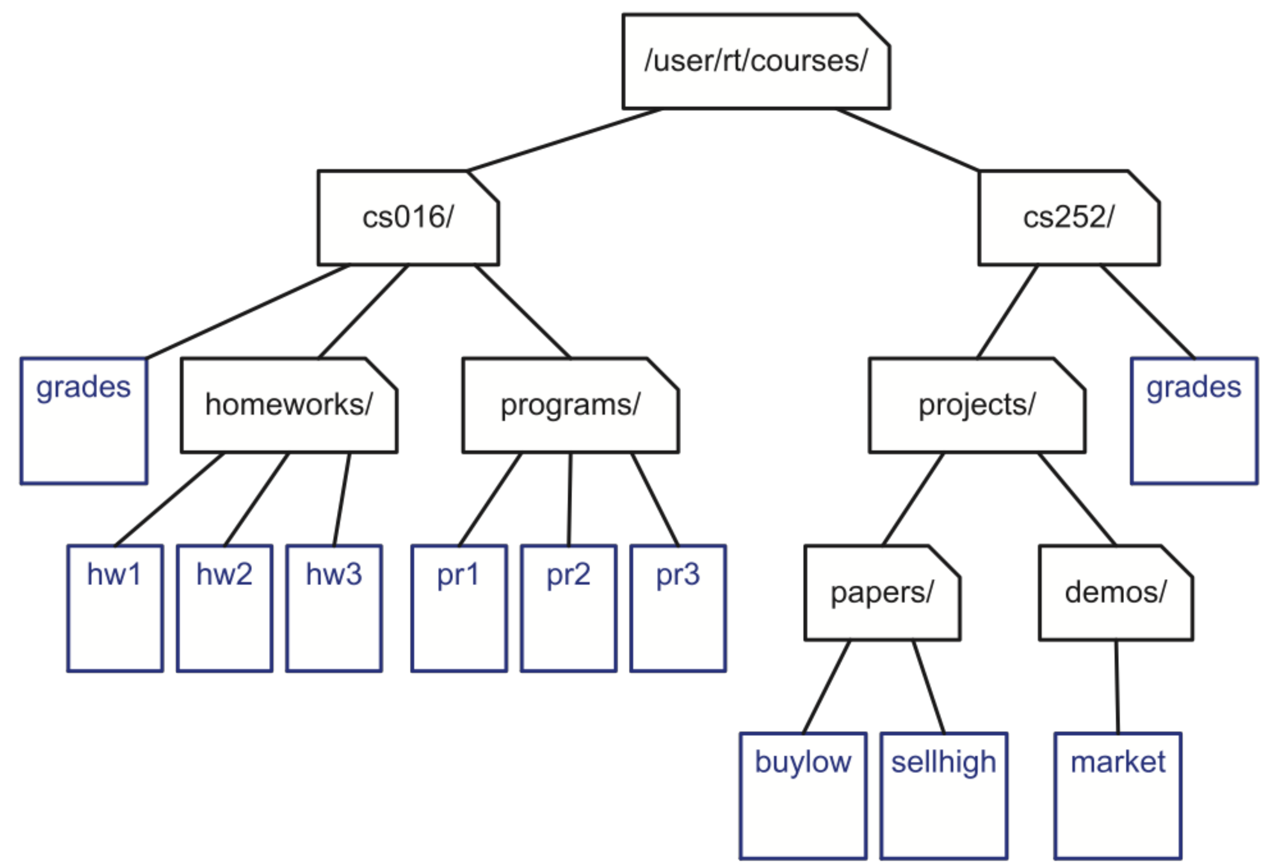
- Trees: basic terminology and notations
  - Free trees อิสระ
  - Rooted trees มีราก
  - Ordered trees มีลำดับ
- Data structures for representing trees
  - Linked structures
- Basic operation on rooted trees
  - Create a rooted tree

Classification of Data Structures



Trees: Informal Introduction (1)

- Trees are *non-linear*, but a ***hierarchical*** data structure
  ลำดับชั้น
- Trees are a breakthrough in data organization. They allow implementing a host of algorithms which are much faster than when operating on linear data structures
- Trees provide a natural organization for file systems, GUI, databases, websites, etc.

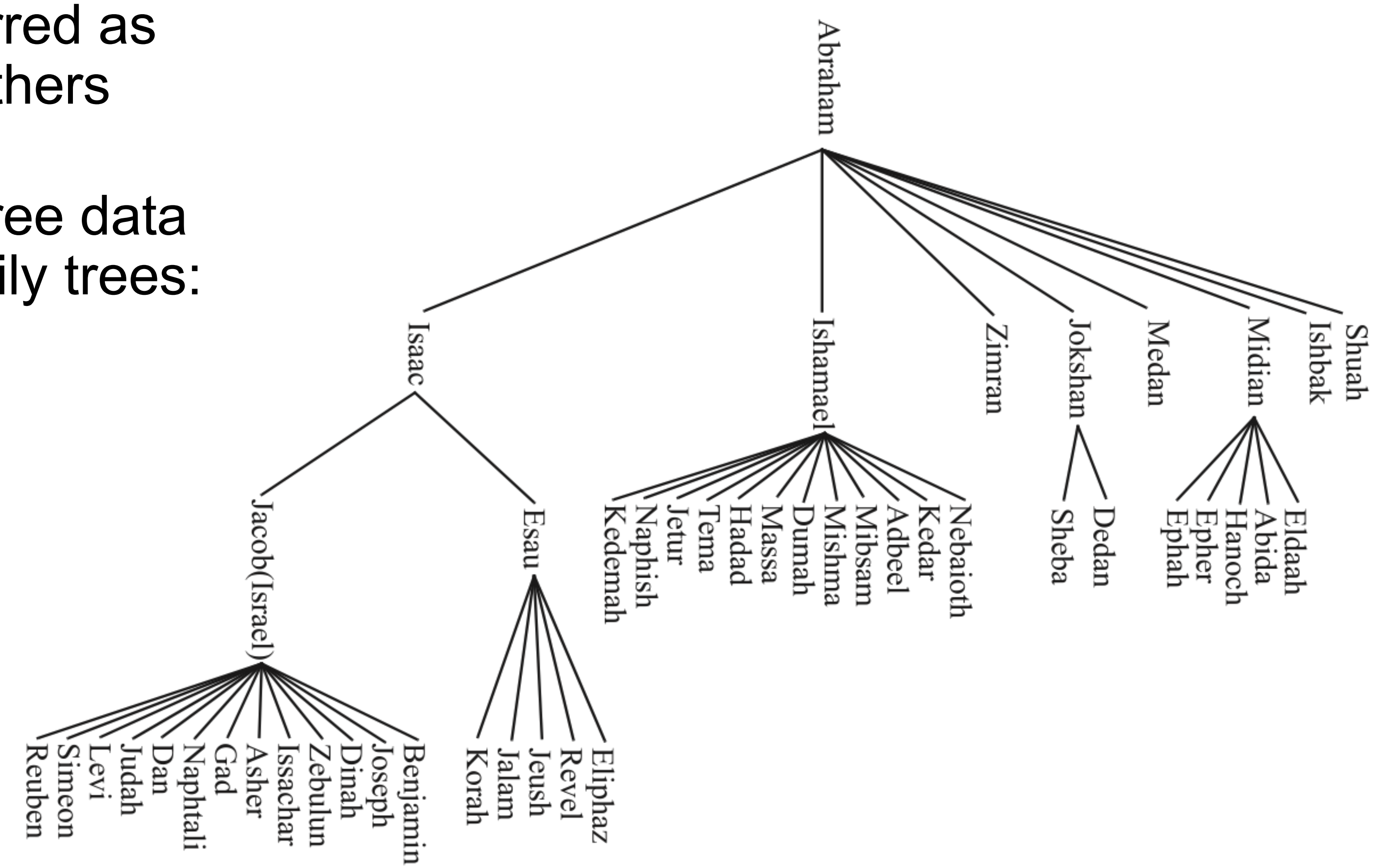


Trees: Informal Introduction (2)

- The relationships in a tree are hierarchical. This means some objects in the tree are referred as being “*above*” or “*below*” others

- The main terminology for tree data structures comes from family trees:

  - **Parent** พ่อแม่
  - **Child** ลูก
  - **Ancestor** บรรพบุรุษ
  - **Descendant** ลูกหลาน
  - **Siblings** พี่น้อง (มีพ่อเดียวกัน)



คือ Graph จริงไม่ใช่ Trees จริงๆ

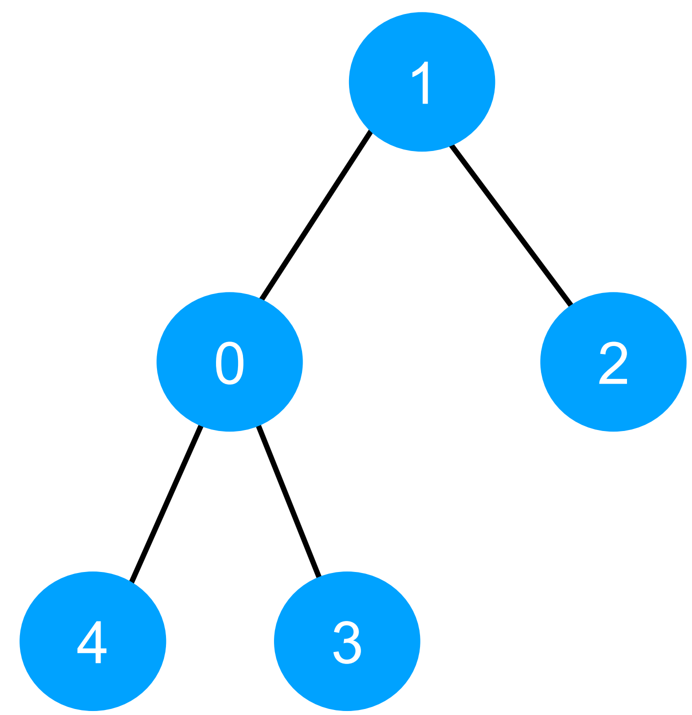
Trees (Free Trees)

cycle ต้องวิ่งเกิน u,v ได้มากกว่า 1 เส้นทาง

- A ***free tree*** (or simply just ***tree***) is a *connected undirected graph that has no cycle*

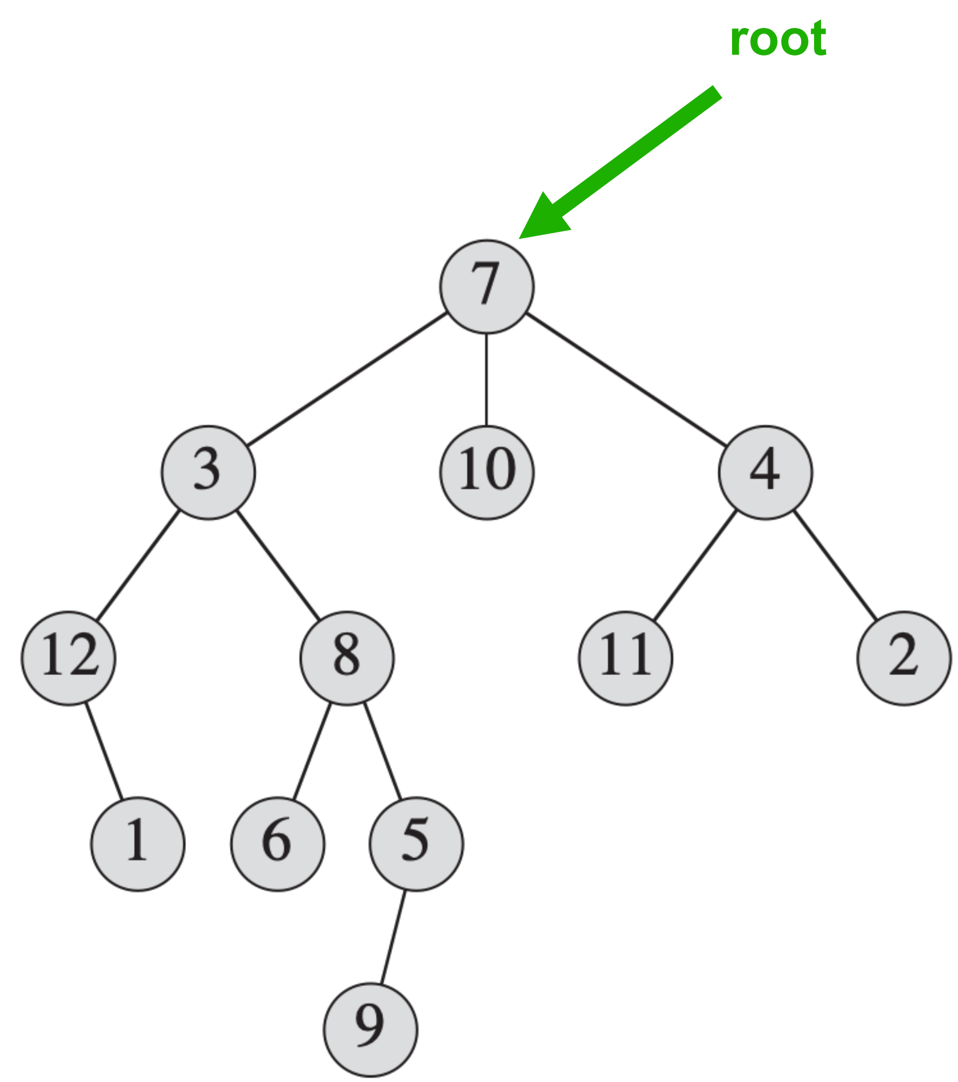
  จับประกันโครงสร้างตามด้านบน

- In the example, a free tree is given by (V, E), where
  - The set of vertices
    $V = \{0, 1, 2, 3, 4\}$
  - The set of edges
    $E = \{\{1,0\}, \{1,2\}, \{0,3\}, \{0,4\}\}$
- ***Remark*: Because a free tree is basically a graph, we can use data structures for graph to represent it so



Rooted Trees

- A ***rooted tree*** is a free tree in which one of the vertices is distinguished from the others
  - We call the distinguished
    ทุกอย่างเริ่มจาก Root
    vertex, the ***root*** of the tree (*the top element of the tree*)
  - We often refer to a vertex of a rooted tree as a ***node*** of the tree

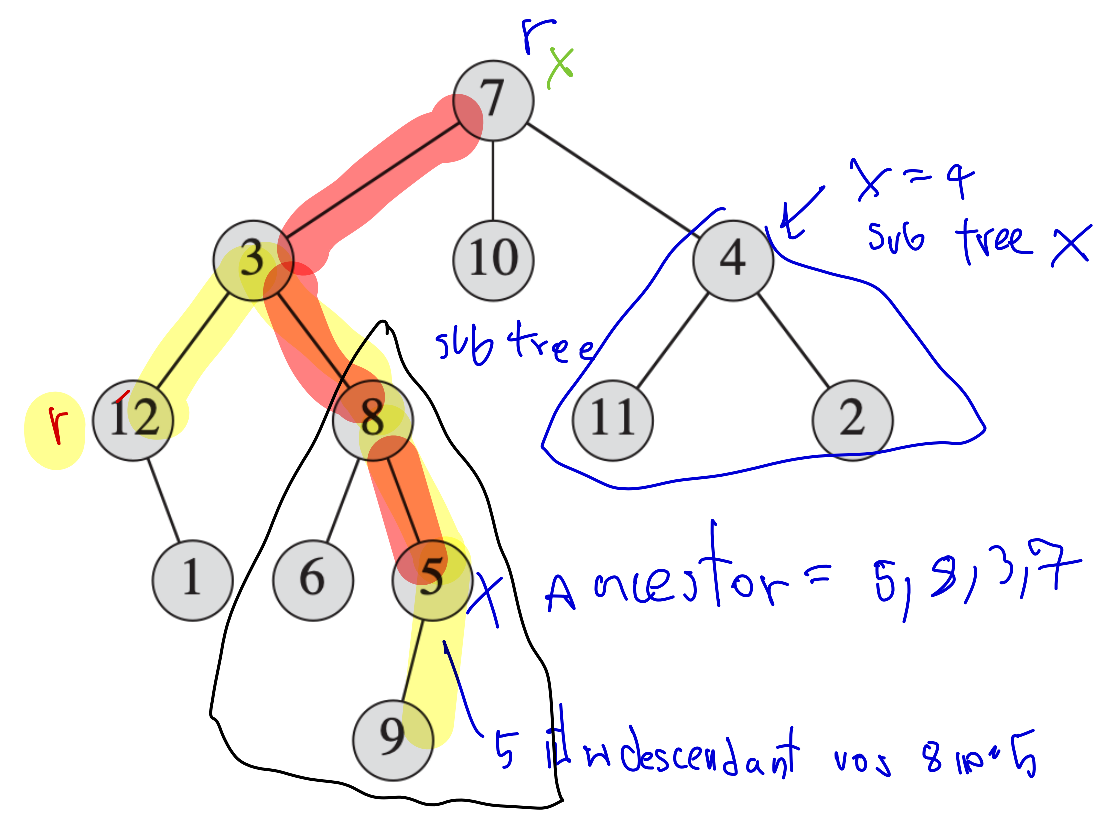


Rooted Tree Terminology (1)

พิจารณา

- Consider a node *x* in a rooted tree *T* with root *r* :
  - We call any node *y* on the unique simple path from *r* to *x* an ***ancestor*** of *x*

    บรรพบุรุษ
  - If *y* is an ancestor of *x*, then *x* is a ***descendant*** of *y* (every node is both an ancestor and a descendant of itself)
  - The ***subtree rooted at x*** is the tree induced by descendants of *x*, rooted at *x*
    - For example, the subtree rooted at node 8 in the figure contains nodes 8, 6, 5, and 9



Rooted Tree Terminology (2)

- If the last edge on the simple path from the root *r* of a tree *T* to a node *x* is (*y*, *x*), then *y* is the ***parent*** of *x*, and x is a ***child*** of *y*

  root Node พิเศษไม่มี Parent

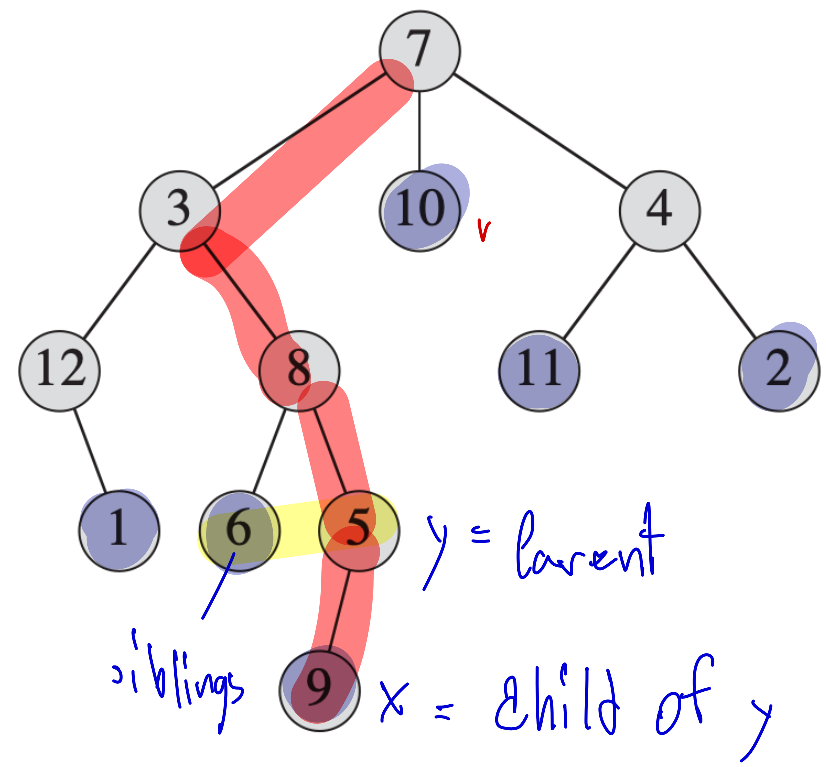
  - The root is the only node in *T* with no parent

- If two nodes have the same parent, they are ***siblings***

- A node with no children is a ***leaf*** or ***external node*** Node ที่ไม่มี children

  Node ที่ยังมีลูก

- A non-leaf node is an ***internal node***



Rooted Tree Terminology (3)

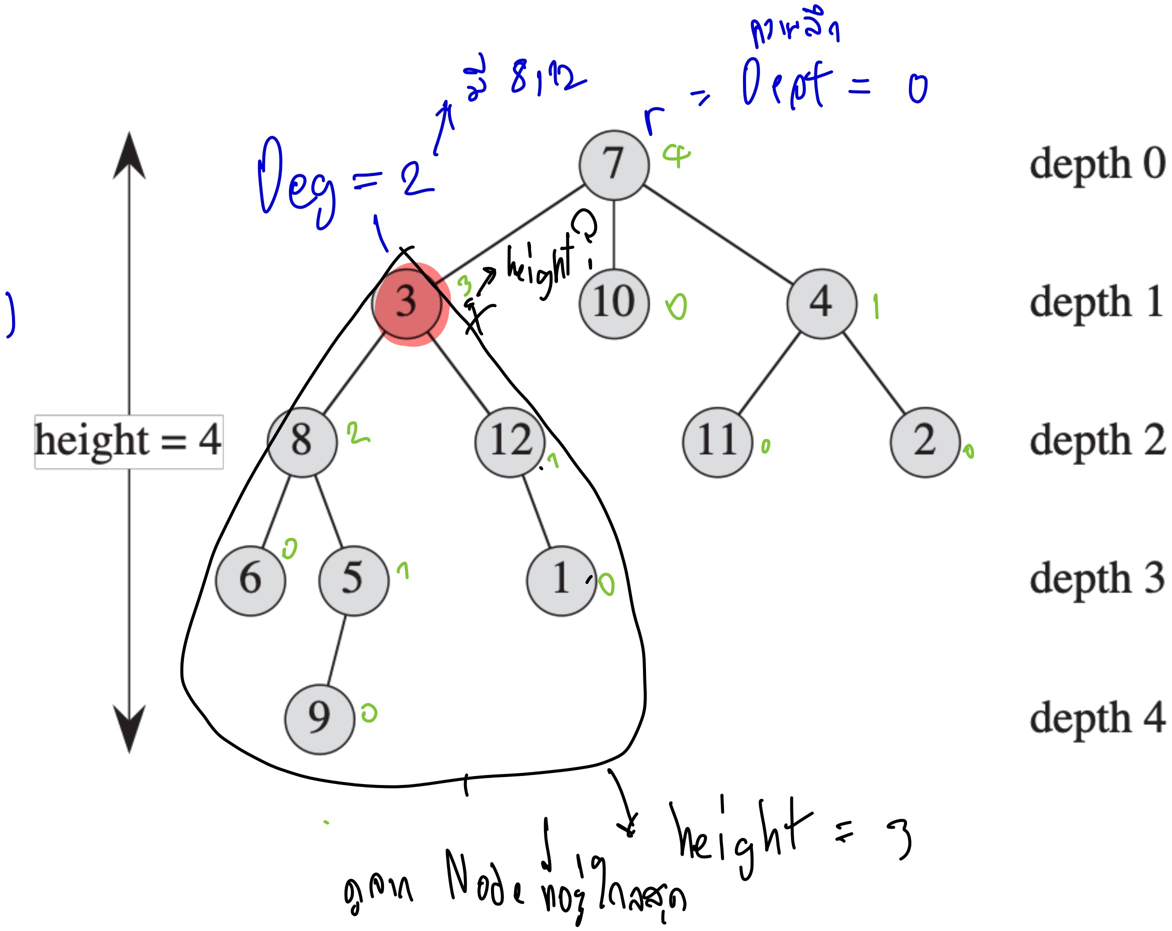
ลงไปข้างล่างมีลูกได้เท่าไหร่ = Degree

- The number of children of a node *x* in a rooted tree *T* equals the ***degree*** of *x*
- The length of the simple path from the root *r* to a node *x* is the ***depth*** of *x* in *T*

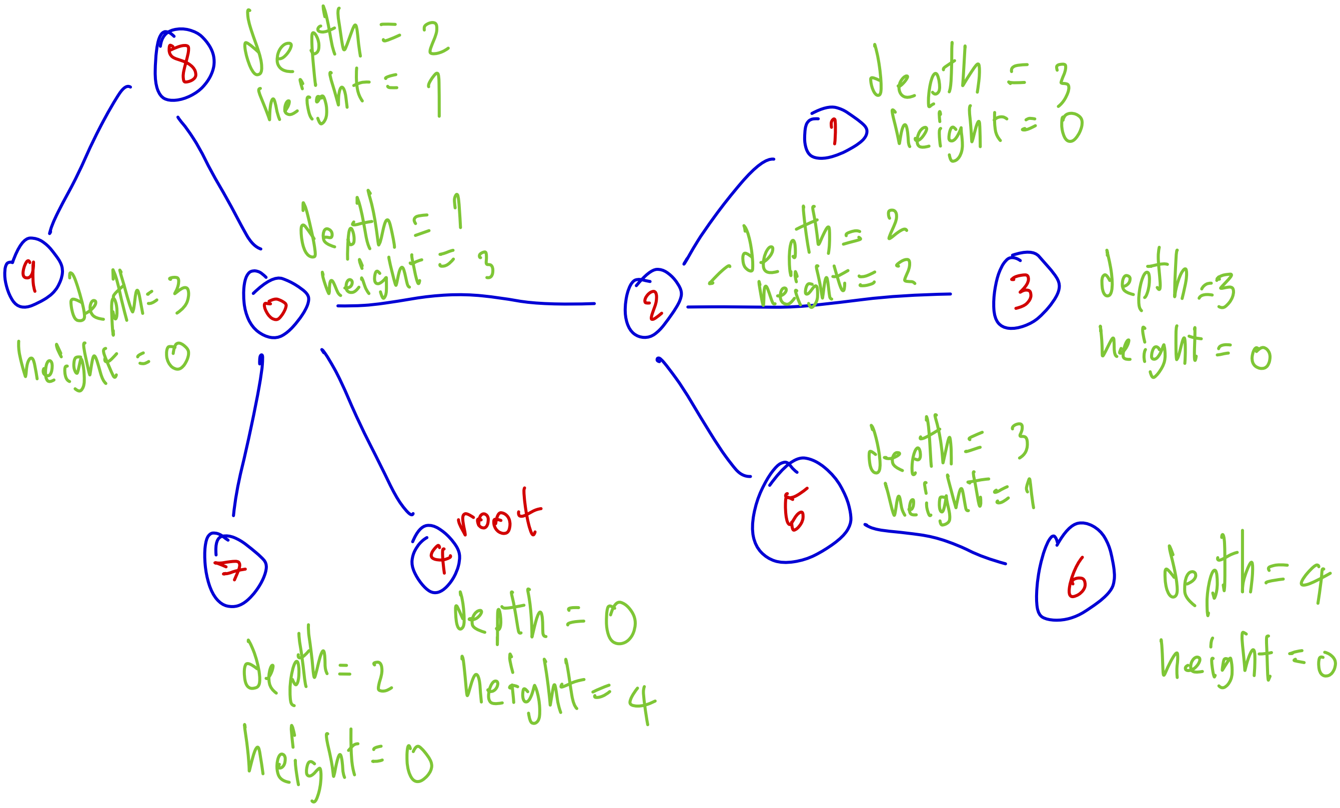
  ความลึก (นับจาก Root เท่าไหร่)

  กลุ่มของ Node ที่อยู่ในระดับความลึกเดียวกัน
- A ***level*** of a tree consists of all nodes at the same depth

  ความสูง ≠ ความลึก (ไม่นับ Node)
- The ***height*** of a node in a tree is the number of edges on the longest simple downward path from the node to a leaf, and the height of a tree is the height of its root ทุก Node เป็น Root ของ subtree ได้
  - Hence, the height of a tree is also equal to the largest depth of any node in the tree

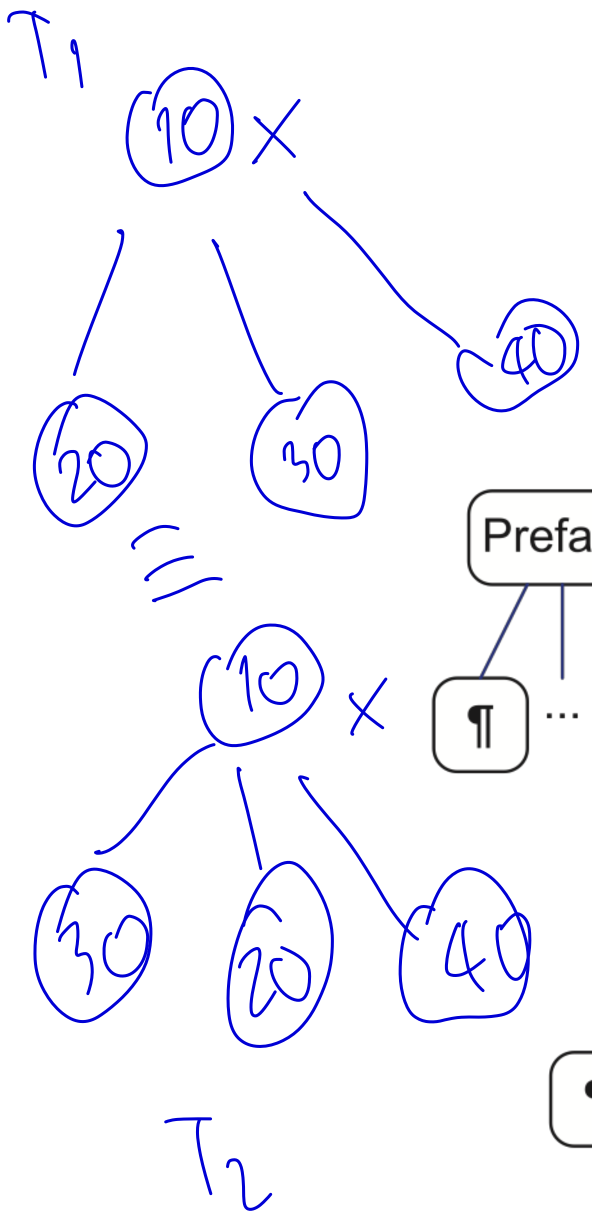


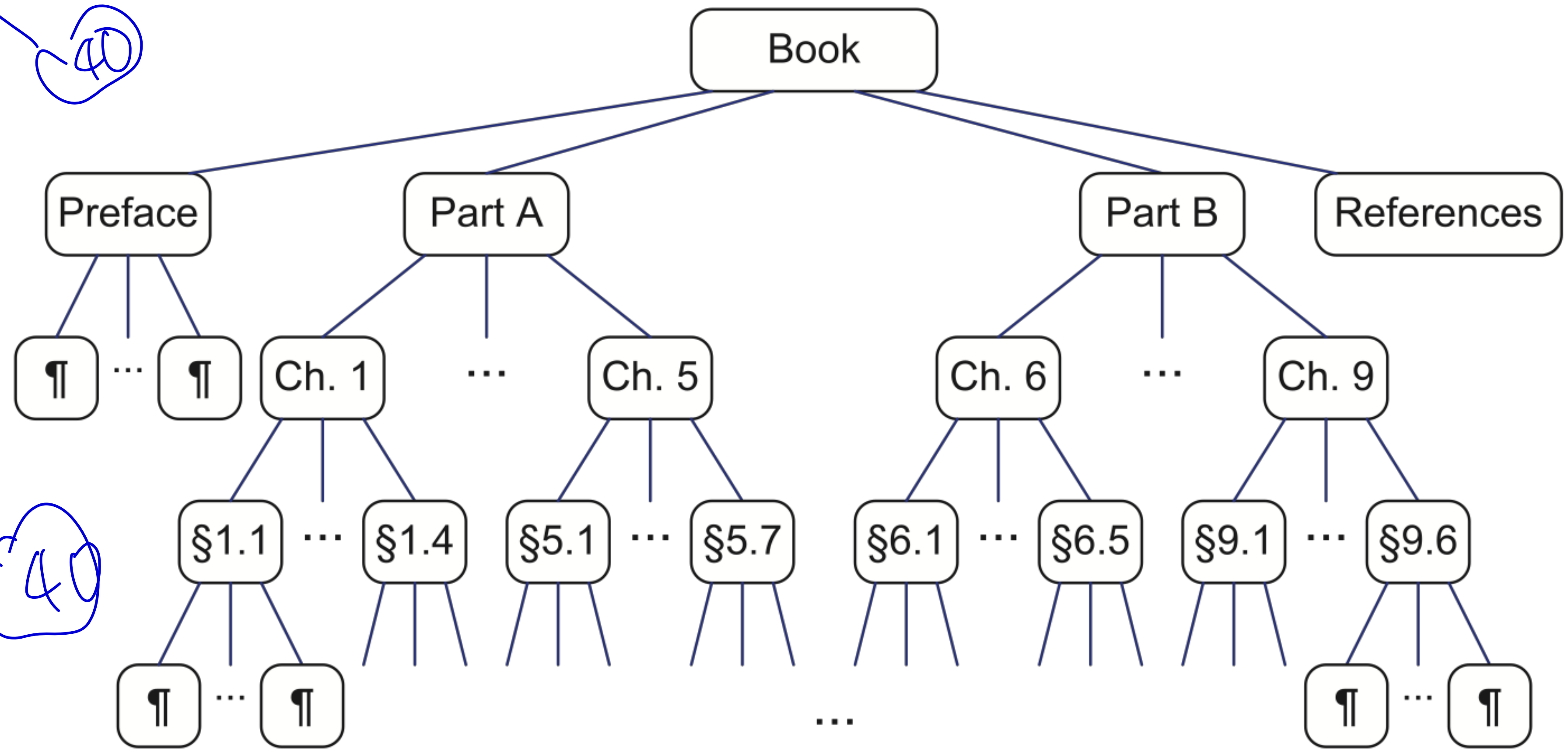
Quiz 3 : ให้เขียนค่า height และ depth ของแต่ละโหนด ใน tree ซึ่งมี Root เป็นโหนด



กิตติคุณ บุญต่อญาติ 6510405334

Ordered Trees ลำดับของลูกมีผลต่อโครงสร้าง





- An ***ordered tree*** is a rooted tree in which the children of each node are ordered. That is, if a node has *k* children, then there is a first child, a second child, . . . , and a *k*-th child

5

10

10 เป็นลูกของ 5 ความคิด 0ไม่ ของ 5 ไว้

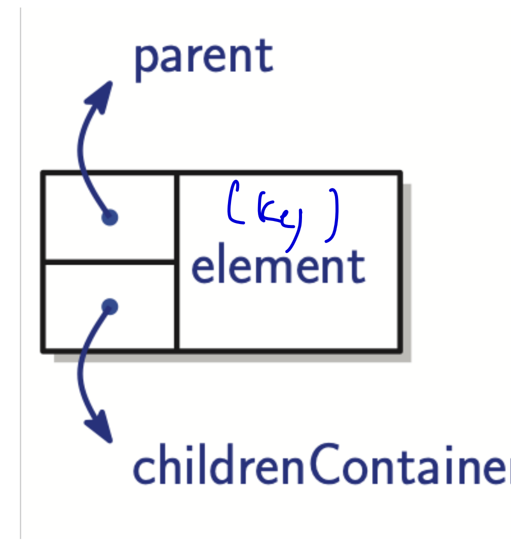
Linked Structure for General Trees

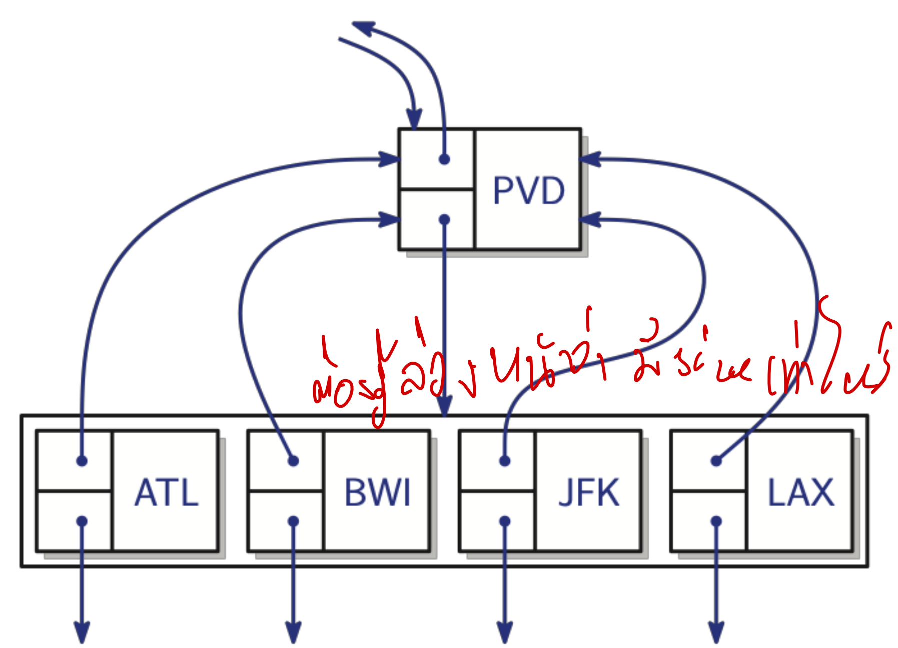
- A natural way to realize a tree *T* is to use a ***linked structure***, where we represent each node of *T* by an object *p* with the following fields:

  - A reference to the node's element (key)

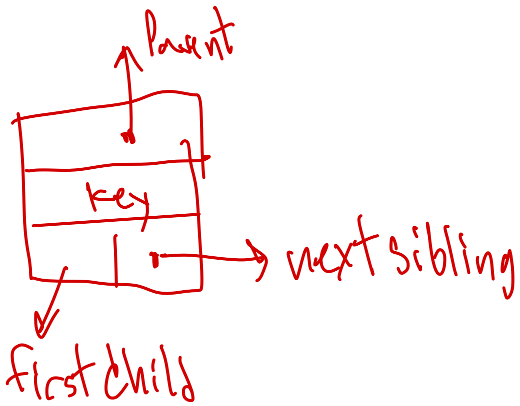
  - A link to the node's parent

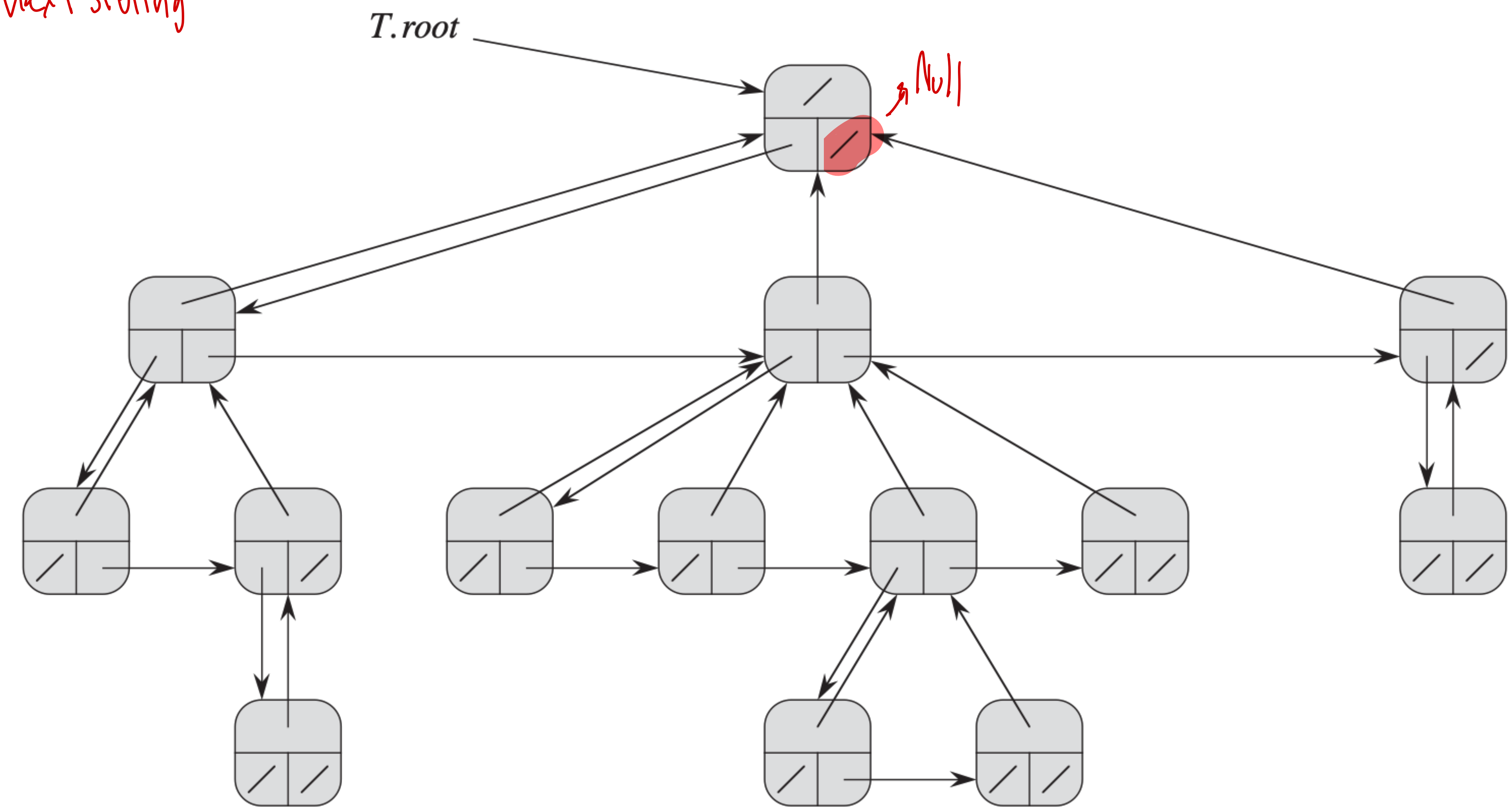
  - Some kind of collection (for example, a list or array) to store links to the node's children





Linked Structure for Rooted Trees



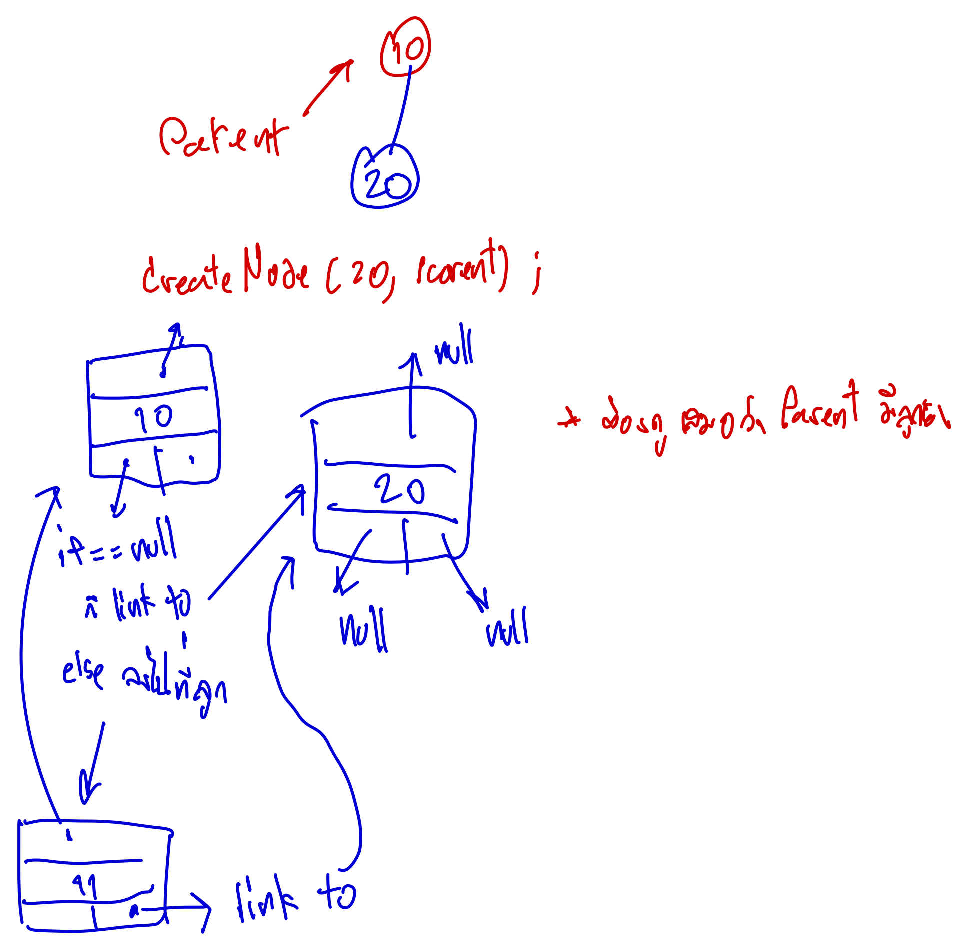


Basic Operation on Trees: Create a Rooted Tree (1)

```c
#include<stdlib.h>

struct node
{
   int key;
   struct node* parent;
   struct node* leftChild;
   struct node* rightSibling;
};

struct node* createNode(int key, struct node* parent) {
  // Allocate memory for new node
  struct node* node = (struct node*)malloc(sizeof(struct node));
  // Assign key to this node
  node->key = key;
  // Initialize parent
  node->parent = parent;
  // Initialize left child, and right sibling as NULL
  node->leftChild = NULL;
  node->rightSibling = NULL;
  // Set this node as a child to its parent
  if(node->parent != NULL) {
      if(node->parent->leftChild != NULL) {
         struct node* child = node->parent->leftChild;
         while(child->rightSibling != NULL) {
            child = child->rightSibling;
         }
         child->rightSibling = node;
      }
      else {
         node->parent->leftChild = node;
      }
   }
 return node;
}
```



Basic Operation on Trees: Create a Rooted Tree (2)

```
int main()
{
  /*create root*/
  struct node *root = createNode(1, NULL);
  /* following is the tree after the above statement

       1
  */

  createNode(2, root);
  createNode(3, root);
  /* 2 and 3 become children of 1
        1
       / \
      2   3
  */

  createNode(4, root->leftChild);
  /* 4 becomes left child of 2
        1
      /   \
     2     3
    /
   4
  */
  …
  return 0;
}
```