ShanghaiTech University

CS101 Algorithms and Data Structures
Fall 2024

Homework 4

Due date: October 30, 2024, at 23:59

1. Please write your solutions in English.
2. Submit your solutions to Gradescope.
3. Set your FULL name to your Chinese name and your STUDENT ID correctly in Gradescope account settings.
4. If you want to submit a handwritten version, scan it clearly. `CamScanner` is recommended.
5. We recommend you to write in LaTeX.
6. When submitting, match your solutions to the problems correctly.
7. No late submission will be accepted.
8. Violations to any of the above may result in zero points.

1. (12 points) Multiple Choices

Each question has **one or more** correct answer(s). Select all the correct answer(s). For each question, you will get 0 points if you select one or more wrong answers, but you will get 1 point if you select a non-empty subset of the correct answers.

Write your answers in the following table.

(a)	(b)	(c)	(d)

(a) (3') Which of the following statements about **trees** is(are) true?

A. The degree of a node is equal to the number of its descendants.

B. The depth of a node is always positive.

C. Siblings always have the same depth.

D. None of the above.

(b) (3') Which of the following statements about **binary trees** is(are) true?

A. A perfect binary tree with n nodes has height $O(n)$.

B. Given a binary tree with height h. Let n be the number of nodes in this tree, then: $h + 1 \leq n \leq 2^h + 1$.

C. In a binary tree, the maximum number of nodes with depth k is $2k$.

D. None of the above.

(c) (3') Which of the following statements is(are) true?

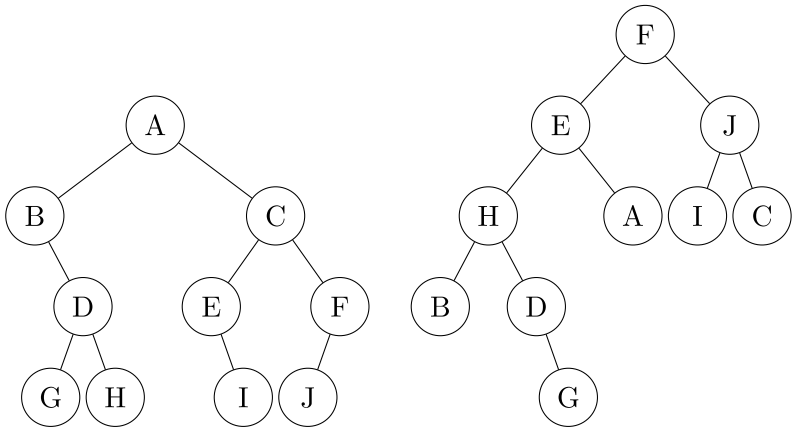
A. The ancestors of a node can never include a leaf node.

B. For any two nodes in a tree, there exists exactly one path between them.

C. A binary tree is a full binary tree if and only if every node has an odd number of descendants.

D. None of the above.

(d) (3') Which traversals of the left tree and right tree, will produce the same sequence node name?



A. left: Post-order, right: Pre-order

B. left: In-order, right: Pre-order

C. left: Post-order, right: In-order

D. left: In-order, right: Post-order

.

2. (8 points) Making binary trees grow

(a) (3') Given the in-order and pre-order traversal of a binary tree T are **AECBFDGH** and **ABCEDFGH** respectively.
Draw the tree T.

Solution:

(b) (3') Given the in-order and post-order traversal of a binary tree T are **XAKQHDPGTBIN** and **AXKHPDQTNIBG** respectively.
Draw the tree T.

Solution:

(c) (2') Given the pre-order and post-order traversal of a binary tree T, can you decide the tree T? If yes, please describe an algorithm to construct T; if no, please provide a counterexample.

Solution:

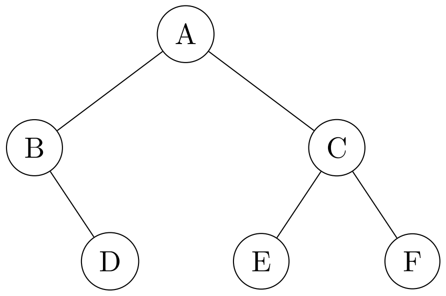
3. (10 points) Run DFS and BFS

Answer the following questions for the tree shown below **according to the definition specified in the lecture slides**.

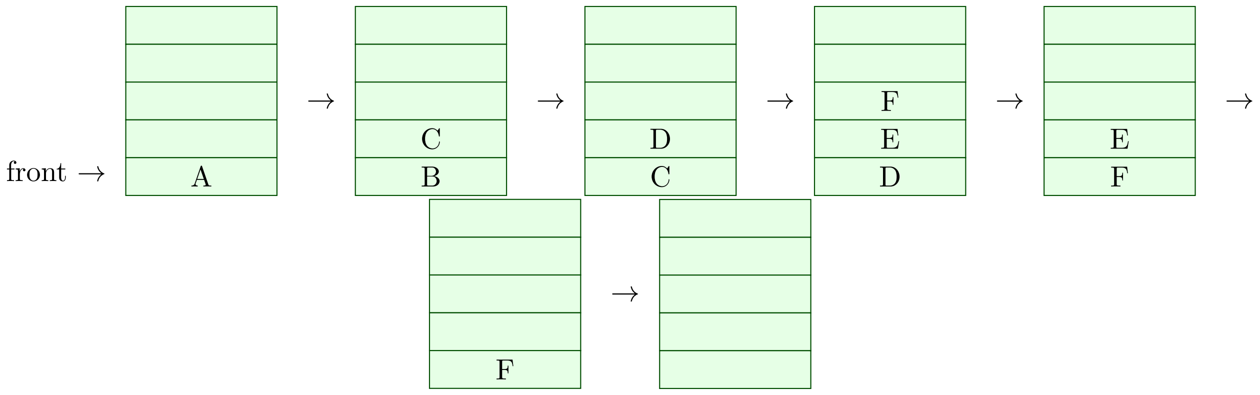
Note: Form your answer in the following steps.

1. Decide on an appropriate **data structure** to implement the traversal.
2. **Popping a node** and **pushing a sequence of children** can be considered as one single step.
3. When doing **Breadth First Traversal**, push children of a node into the data structure in **alphabetical order**; when doing **Depth First Traversal**, push children of a node into the data structure in **reverse alphabetical order**.

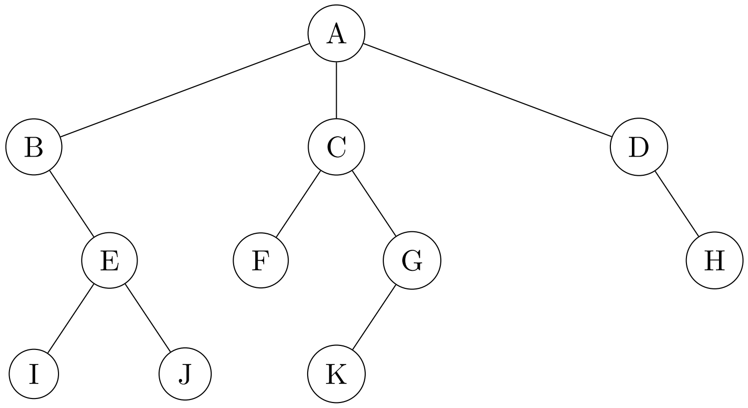
Example: Given a tree with root **A**:



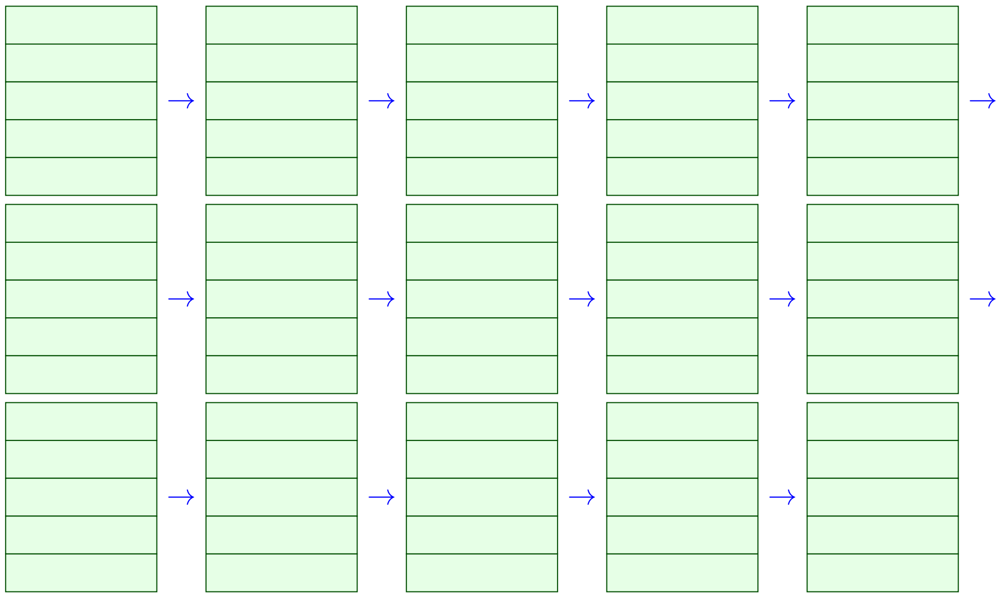
The process of doing **Breadth First Traversal** is:



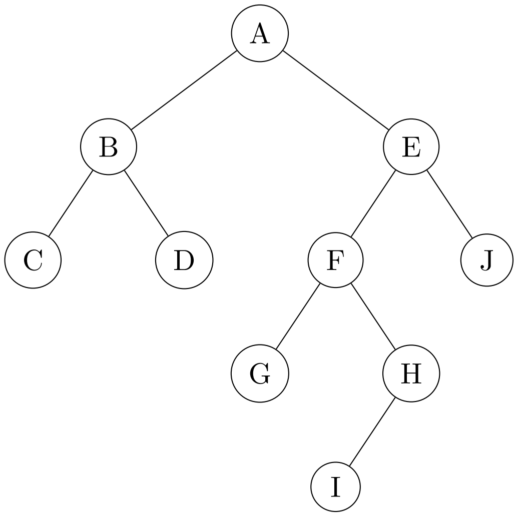
(a) (5') Run **Pre-order Depth First Traversal** on the tree with root **A** and draw the whole process in the space below. (Note: it's not required to use all the blank cells.)



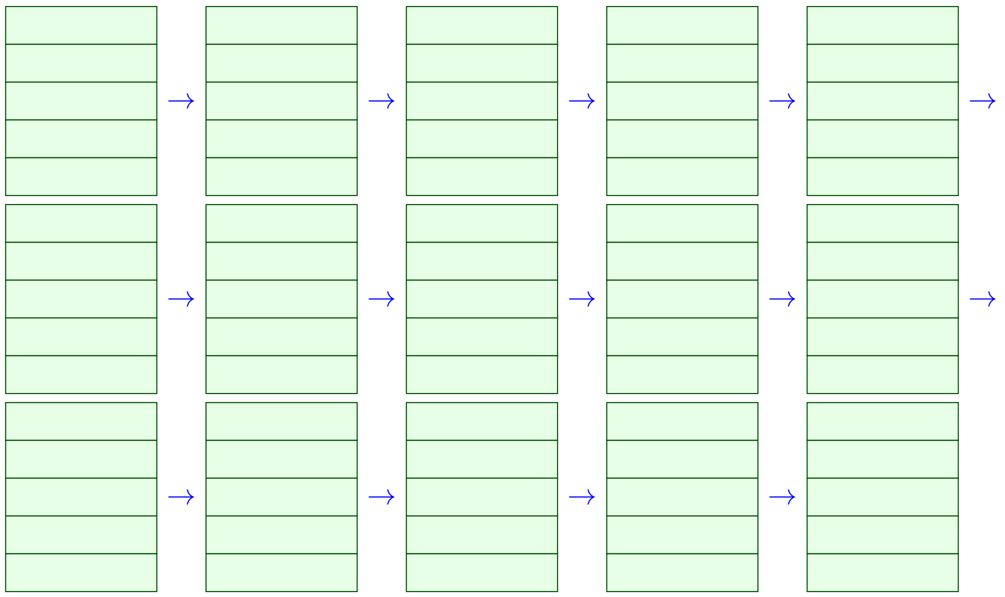
Solution:



(b) (5') Run **Breadth First Traversal** on the tree with root **A** and draw the whole process in the space below. (Note: it's not required to use all the blank cells.)



Solution:



4. (13 points) Array Storage

Unlike arbitrary n-ary trees, binary trees can be easily stored within an array.

(a) (6') Complete the code below:

```
struct BinaryTree {
    int data[SIZE]{};

    // Return the index of the root node
    size_t head() {
        return 1;
    }

    // Return the index of the left child
    size_t left_child_idx(size_t idx) {
        return ________________;
        // Fill in the formula for the left child index
    }

    // Return the index of the right child
    size_t right_child_idx(size_t idx) {
        return ________________;
        // Fill in the formula for the right child index
    }

    // Return the index of the parent node
    size_t parent_idx(size_t idx) {
        return ________________;
        // Fill in the formula for the parent index
    }
};
```

Solution:

```
size_t left_child_idx(size_t idx) {

    return ___________________________;

}
size_t right_child_idx(size_t idx) {

    return ___________________________;

}
size_t parent_idx(size_t idx) {

    return ___________________________;

}
```

(b) (3') To ensure the code functions correctly for all trees with n nodes, what should the minimum **SIZE** be? You should justify your answer correctly.

Solution:

(c) (4') Consider a complete binary tree, the maximum index in this array is 2025, what is the height and number of leaf nodes of this tree? You should justify your answer correctly.

Solution: