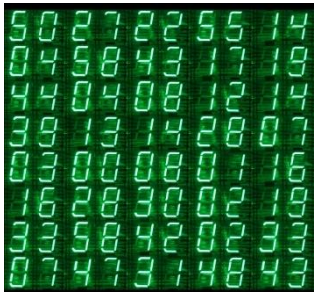
國立清華大學
NATIONAL TSING HUA UNIVERSITY

Matrix to Tree Converter

https://acm.cs.nthu.edu.tw/problem/11901/

Data Structures Assignment

NTHU EE and CS

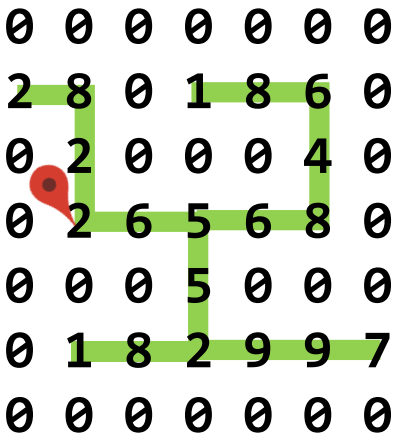


Overview

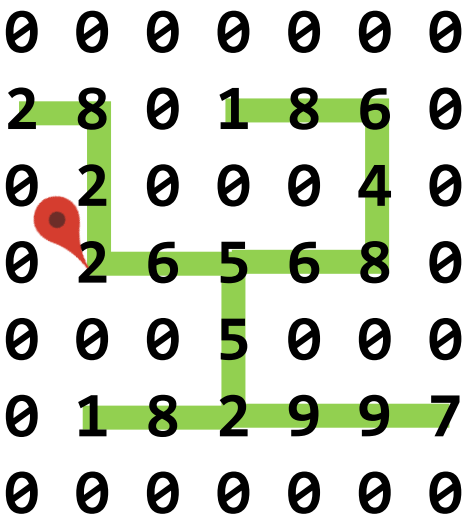
- Given
  - A matrix of digits
  - A starting position
  - Traversal method
- Task
  - Convert the nonzero digits of the matrix into a tree
    - The input matrix guarantees no cycle
  - Print out the digits based on the specified tree traversal methods

Matrix and Tree Specification

- Each matrix cell contains a digit value ranged from 0 to 9
- The starting position of the matrix represents the root of the tree
  - The starting position cannot be 0
- Each tree node can have up to four children
  - Left, Down, Right, Up
- Take the right figure as an example
  - The root is 2, and it has two children
    - Up for 2 and right for 6



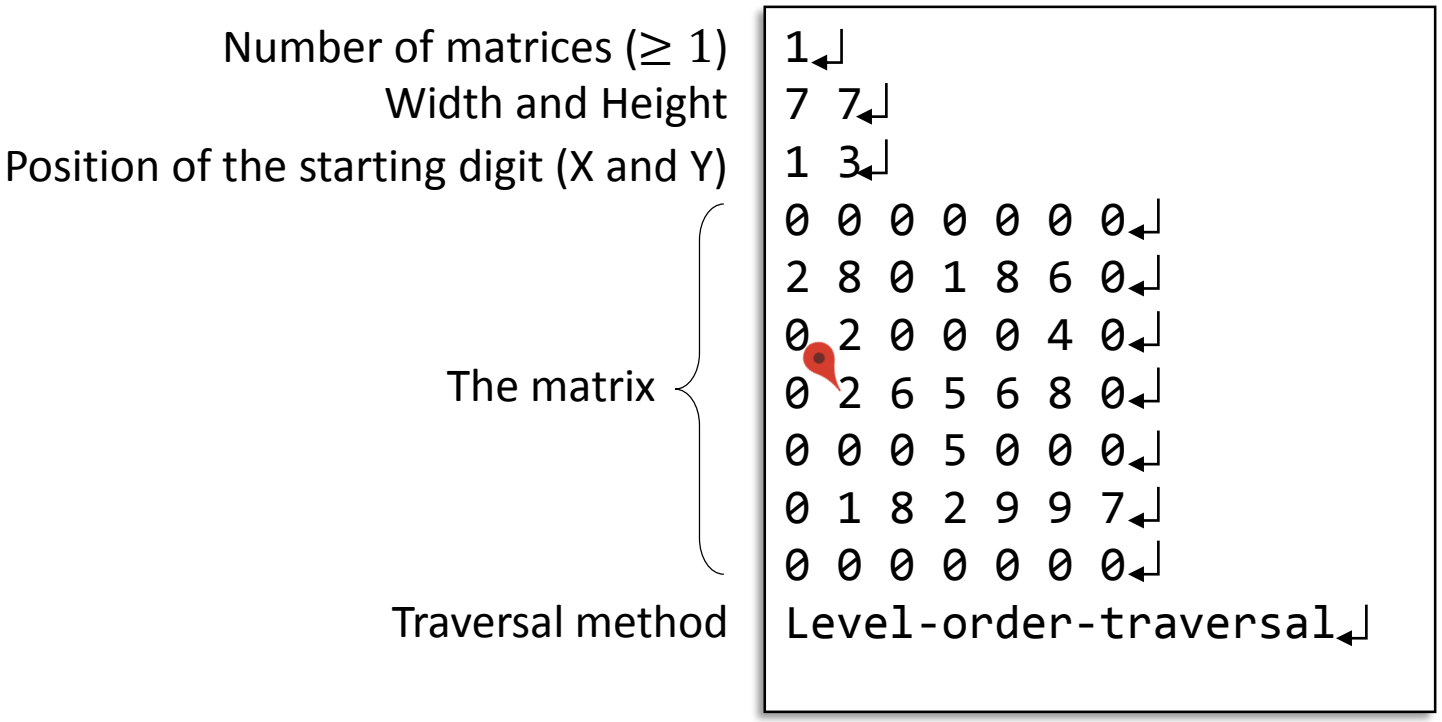
Example



Tree Node Format

Digit			
Left	**D**own	**R**ight	**U**p

Sample Input

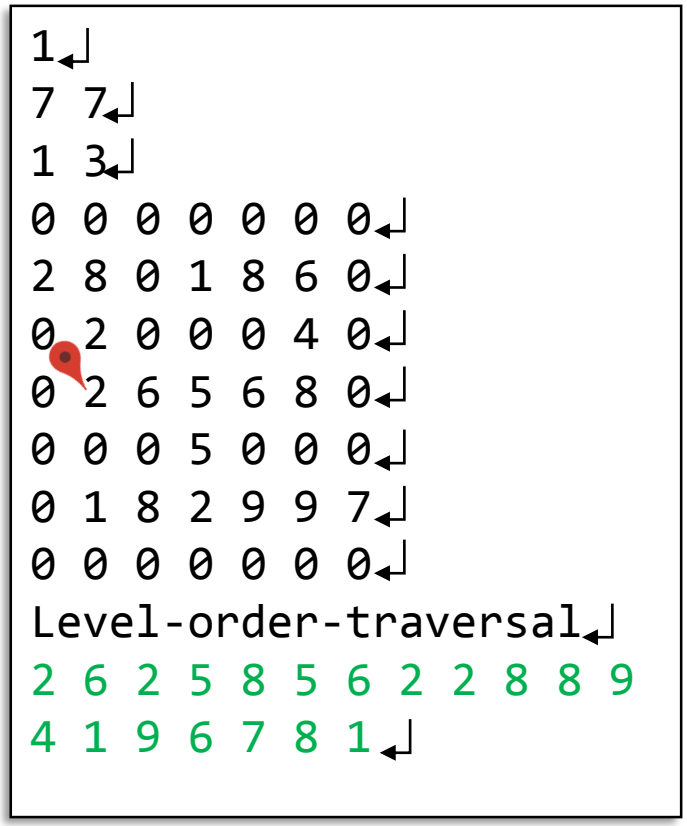


Traversal method can be one of the following:

- "Level-order-traversal"
- "Pre-order-traversal"
- "Post-order-traversal"

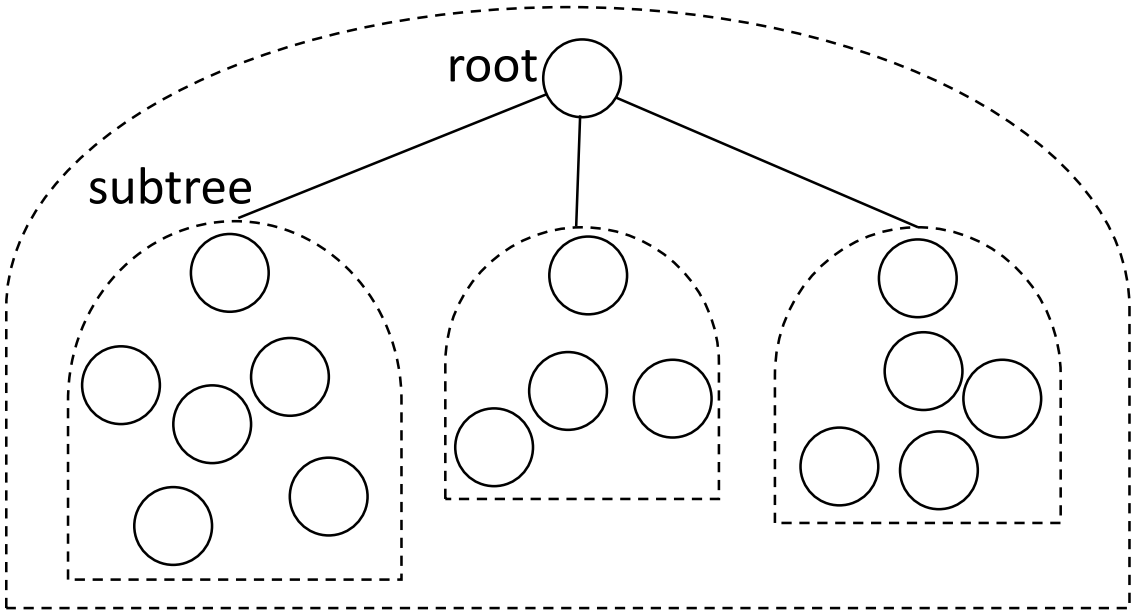
Sample Output

- Repeat the inputs and additionally print out the tree traversal



Tree Definition

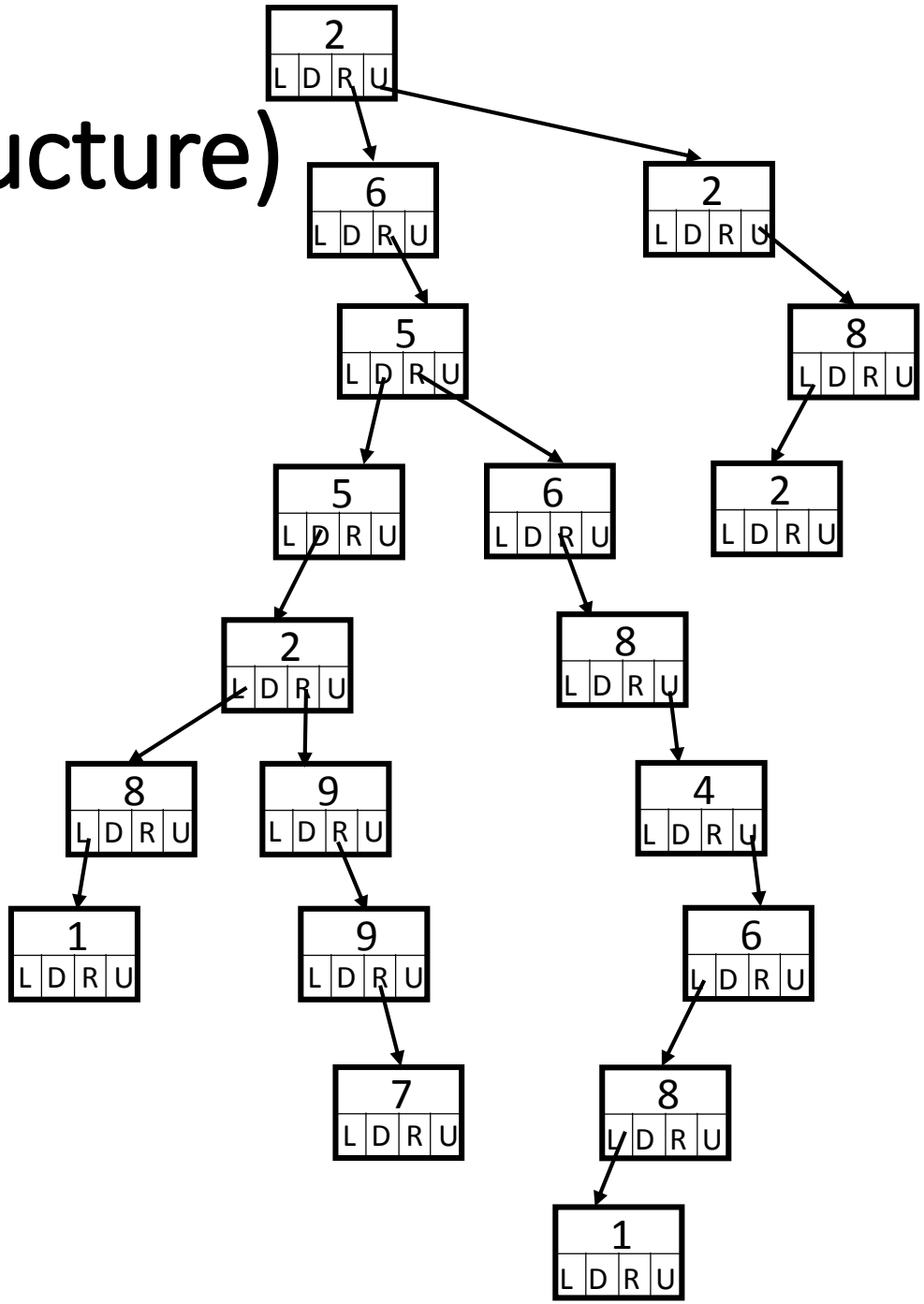
- A finite set of one or more nodes such that
  - There is a specially designated node called the root
  - The remaining nodes are partitioned into $n \geq 0$ disjoint sets, T_1, …, T_n, where each of these sets is a tree (i.e., subtree).



Tree (Linked Structure)

```
class TreeNode {
friend class Tree;
private:
    int data;
    TreeNode * leftChild;
    TreeNode * rightChild;
    TreeNode * …;
};

class Tree{
public:
    // tree operations
private:
    TreeNode * root;
};
```



Tree Node Format

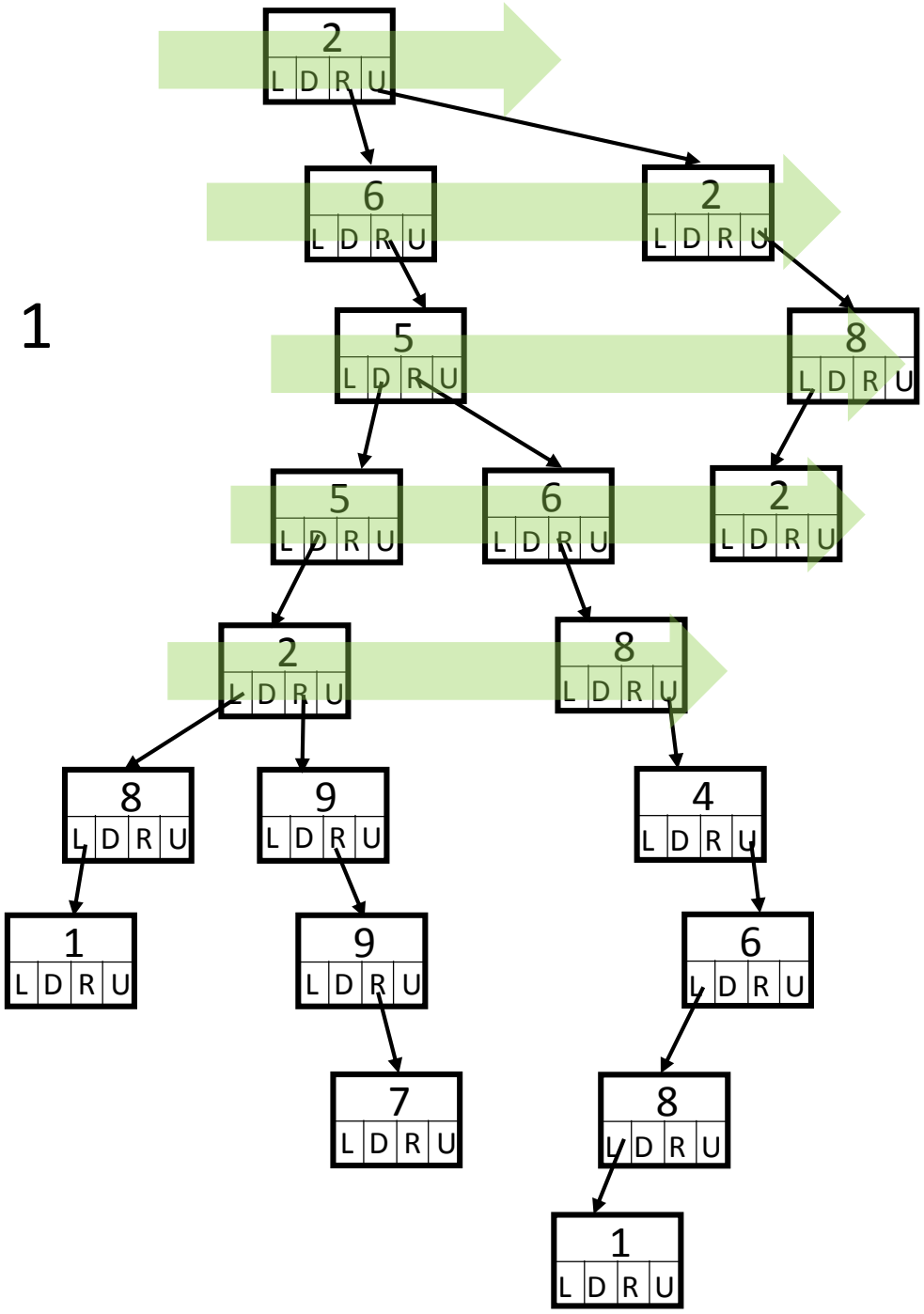
Digit			
Left	**D**own	**R**ight	**U**p

Tree Traversal

- Objective
  - Convert a tree to a sequence based on a predefined rule
- Common method
  - Level order
  - Pre order
  - Post order
  - (In order)

Level Order

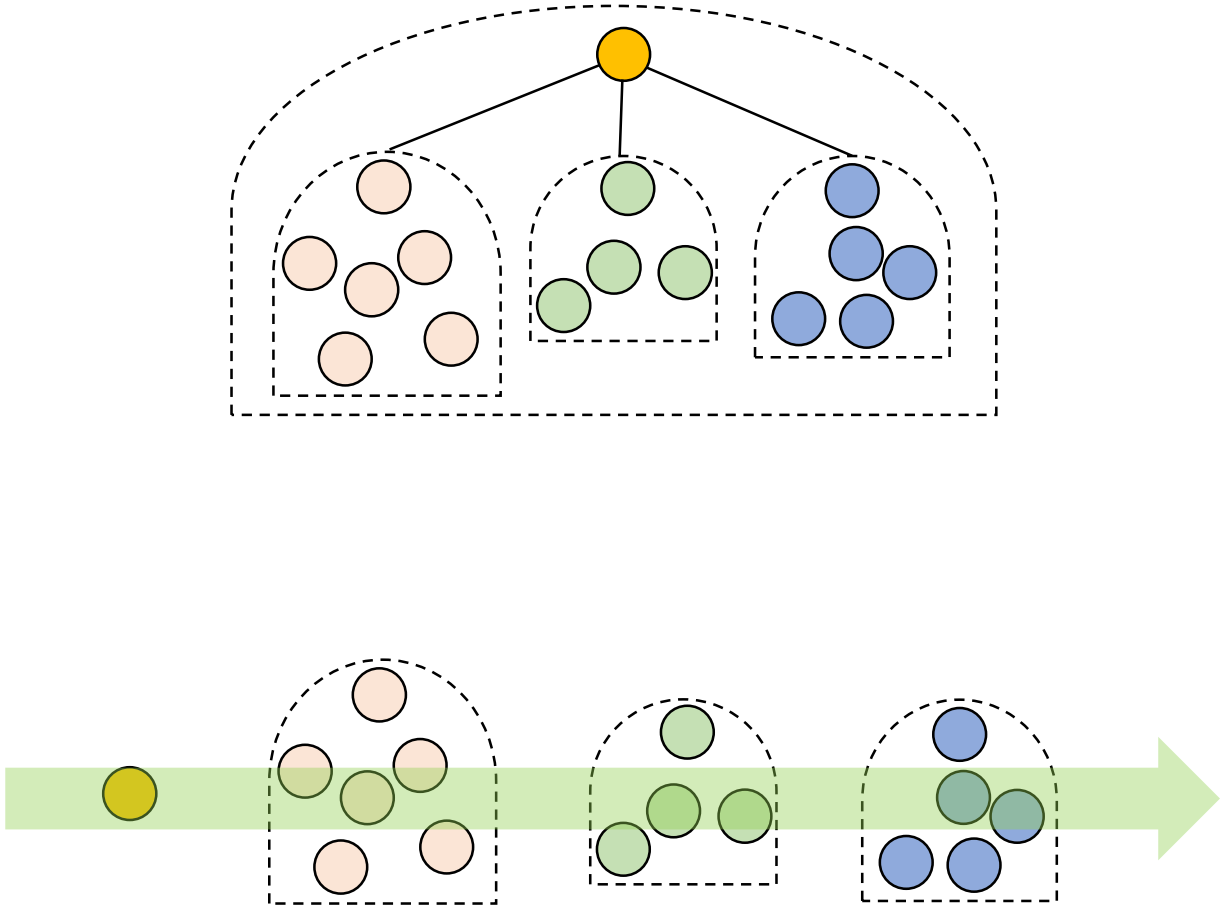
- 2 6 2 5 8 5 6 2 2 8 …. 7 8 1



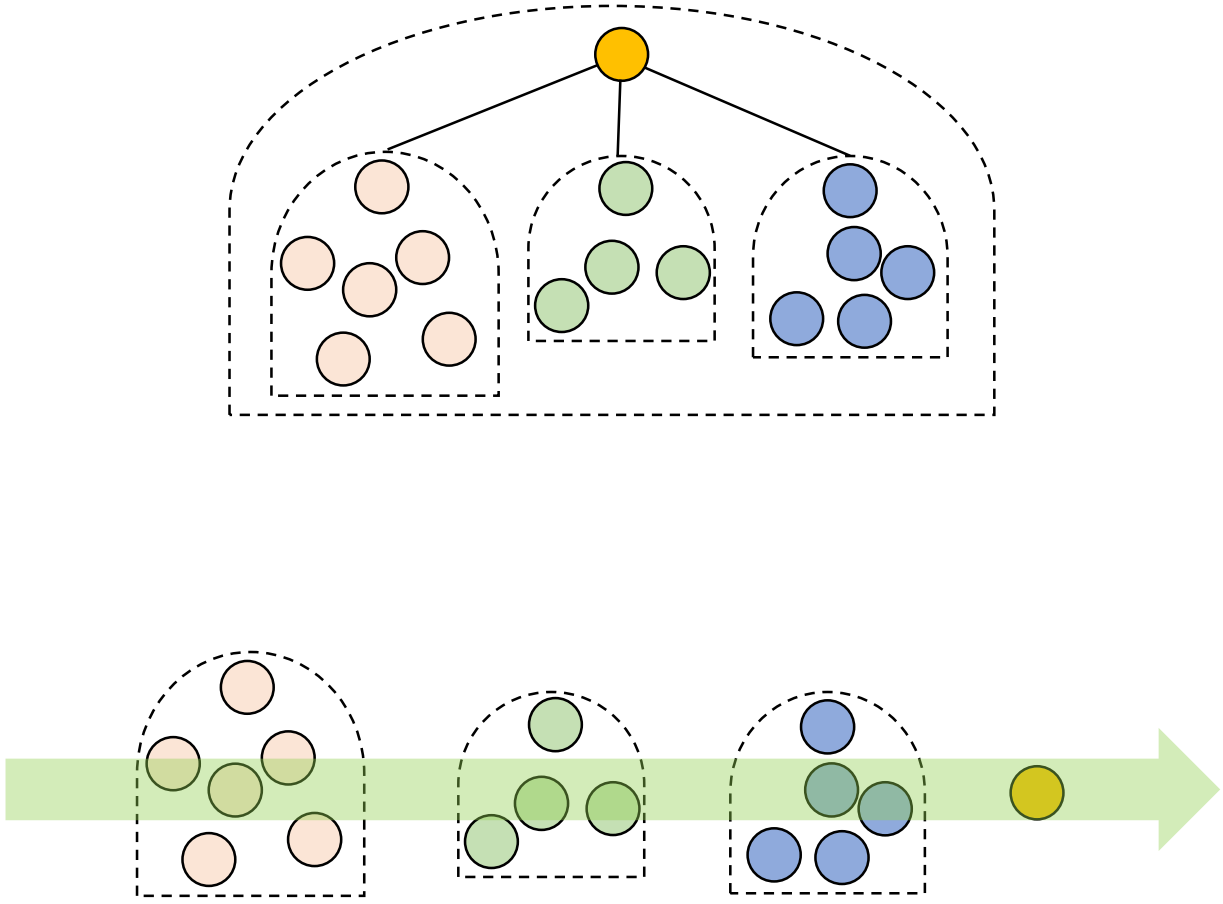
Pre-order and Post-order

- Pre-order:
  - Root goes first
  - Subtrees follows
- Post-order:
  - Subtrees go first
  - Root goes the last

Pre-Order

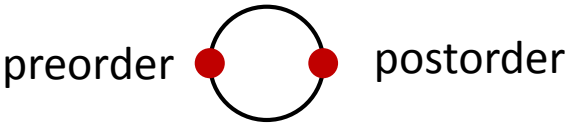


Post-Order

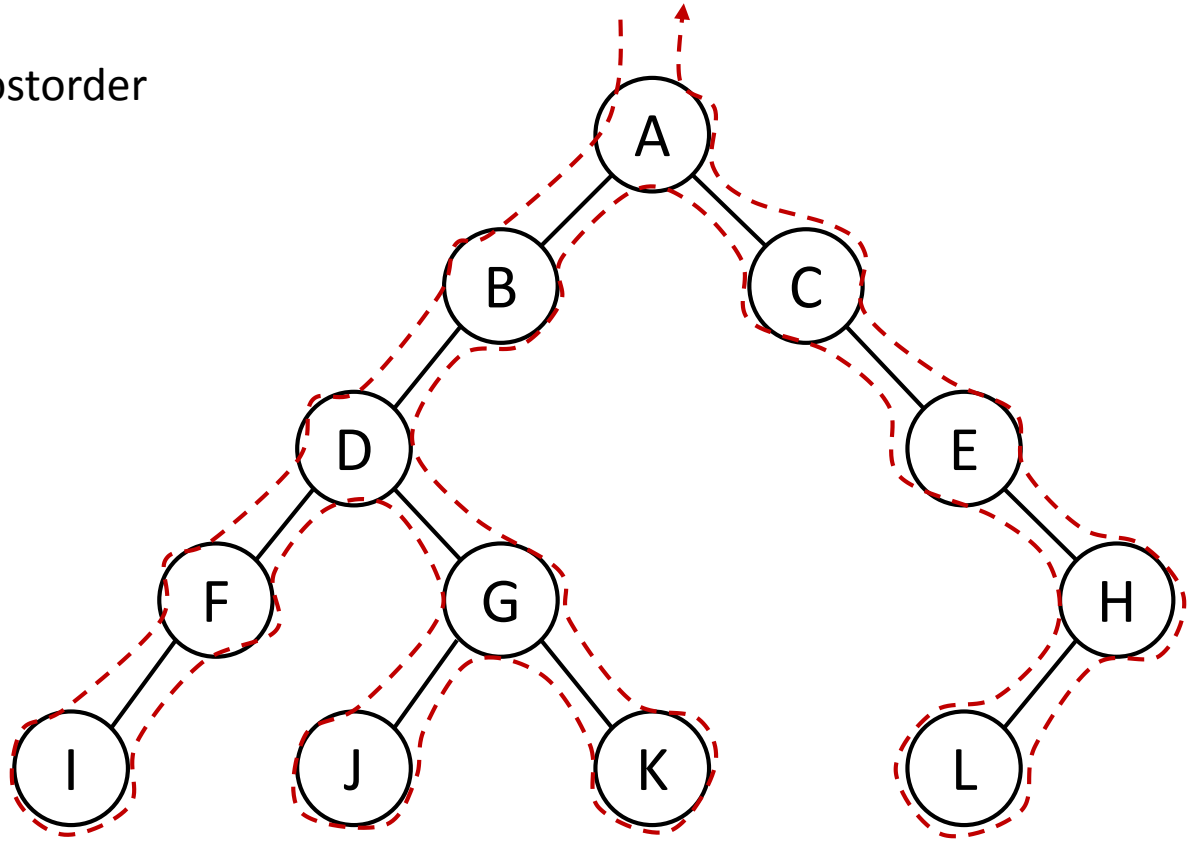


Tips for Preorder, & Postorder

- Attach a point to each node

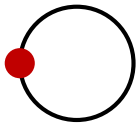


- Draw the contour of the tree



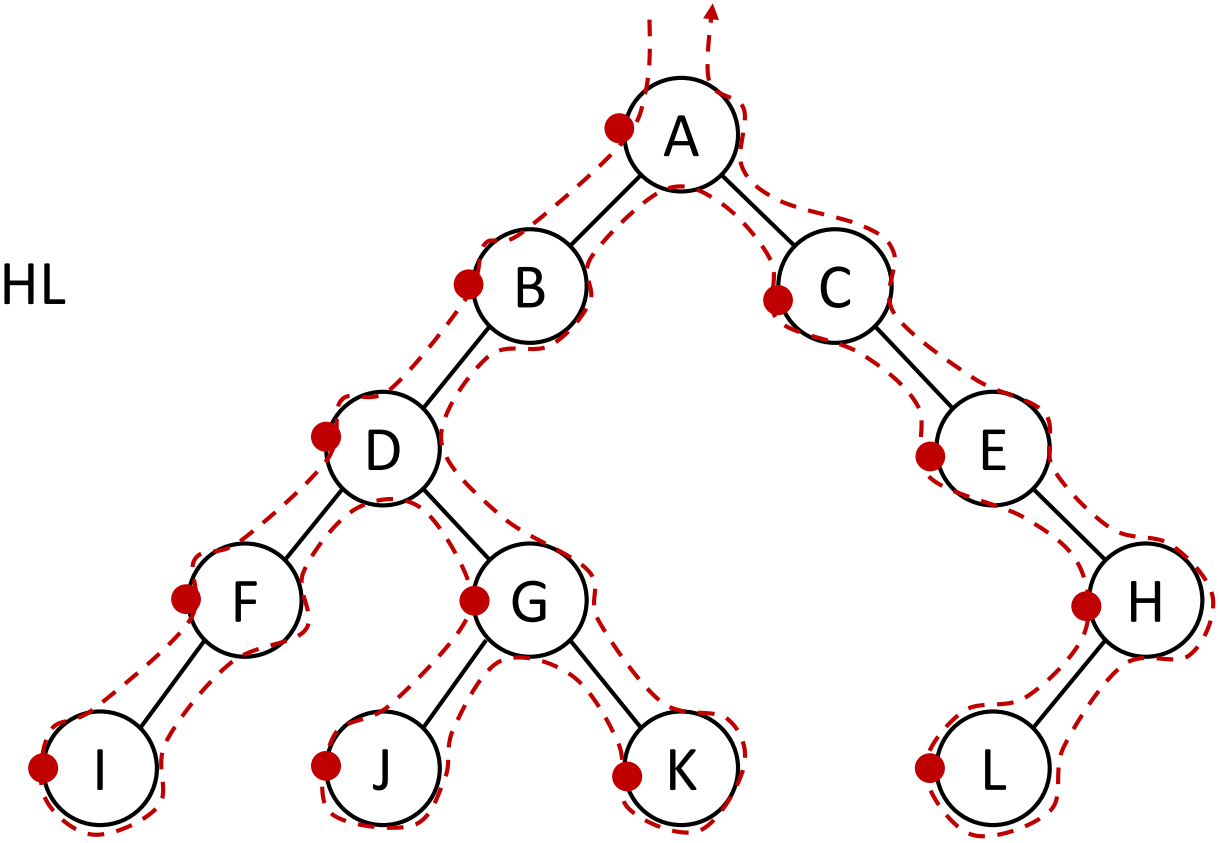
Tips for Preorder, & Postorder

- Attach a point to each node

preorder 

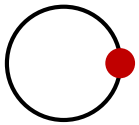
→ABDFIGJKCEHL

- Draw the contour of the tree



Tips for Preorder, & Postorder

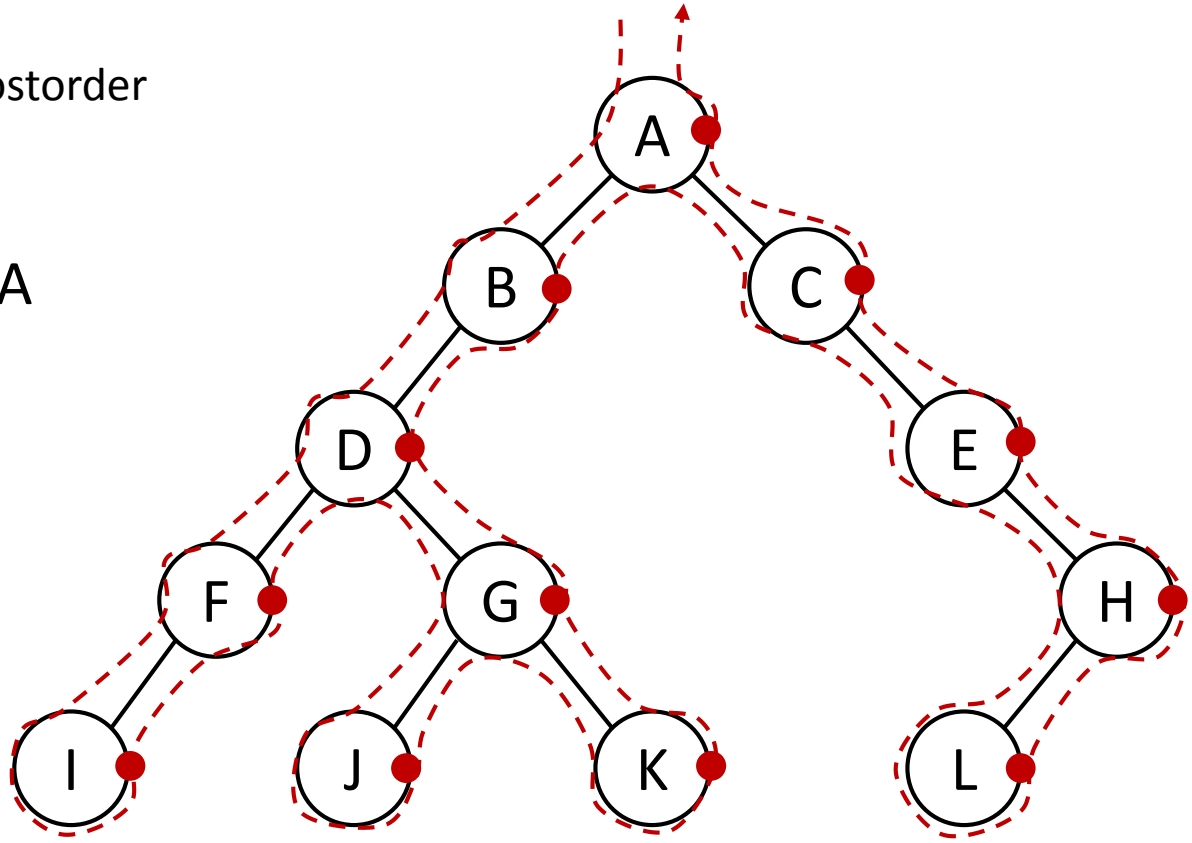
- Attach a point to each node



postorder

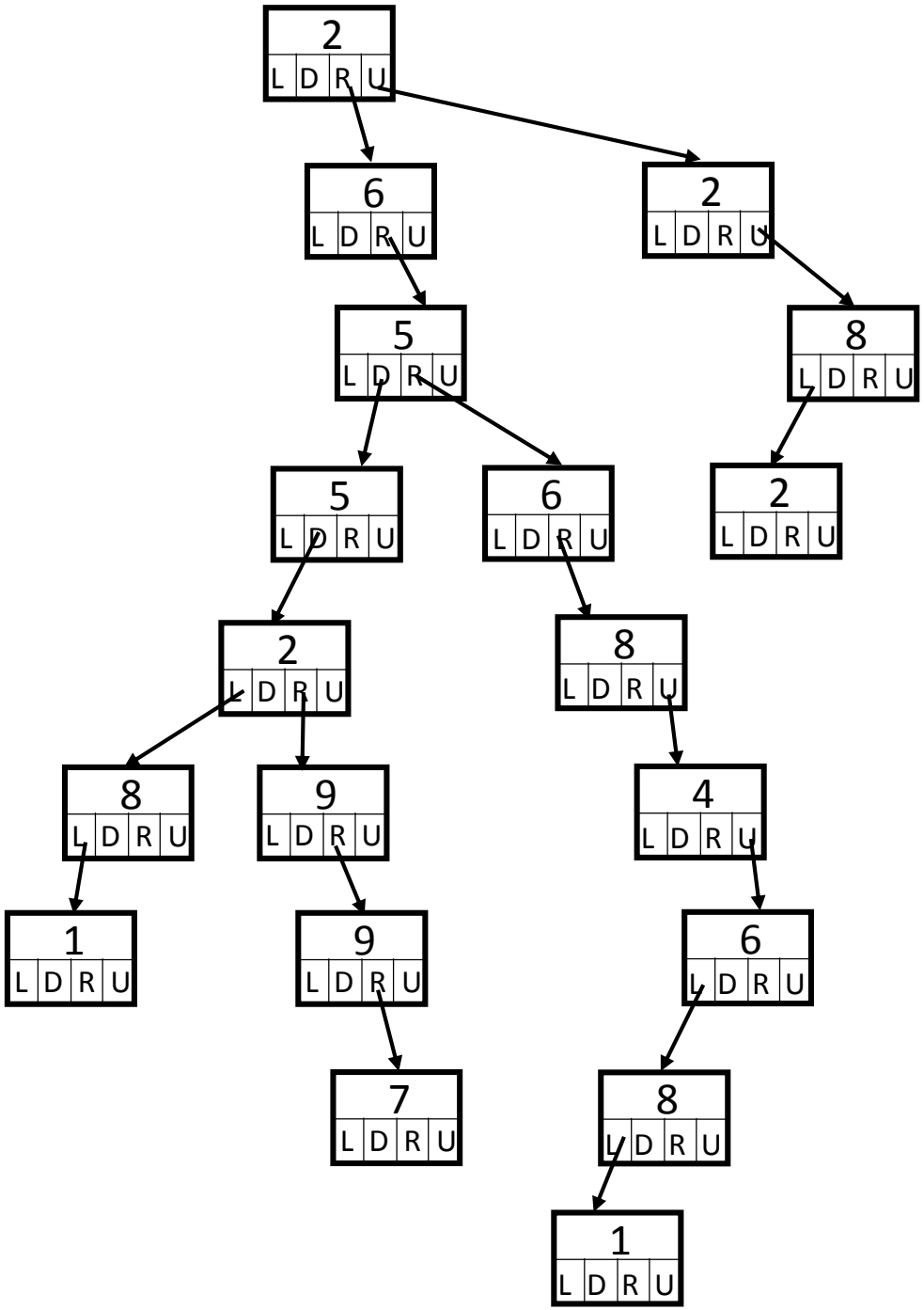
→IFJKGDBLHECA

- Draw the contour of the tree



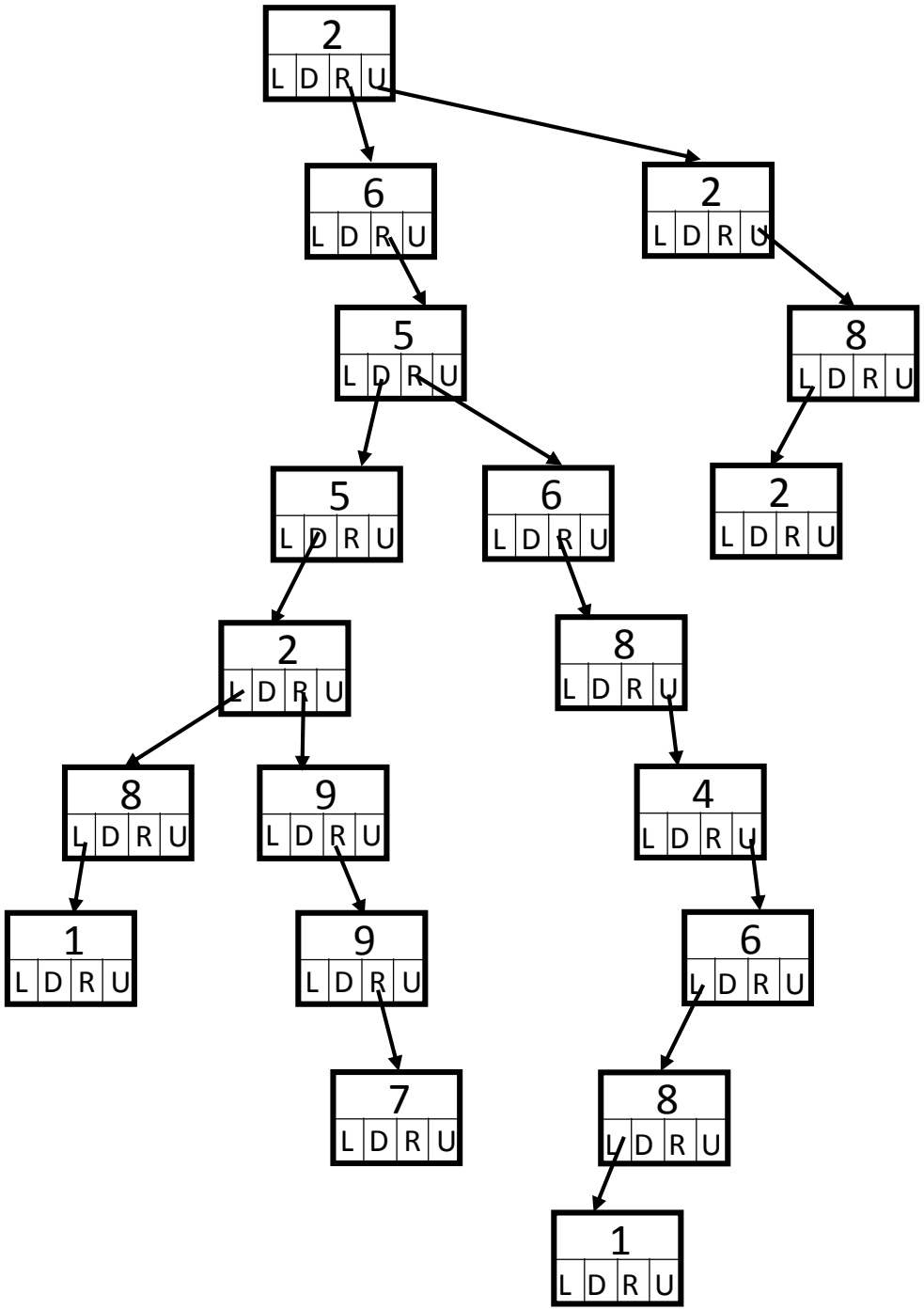
Level Order

- Pre-order?
- Post-order?



Level Order

- Pre-order
  - 2 6 5 5 2 8 1 9 9 7 6 8 4 6 8 1 2 8 2
- Post-order
  - 1 8 7 9 9 2 5 1 8 6 4 8 6 5 6 2 8 2 2



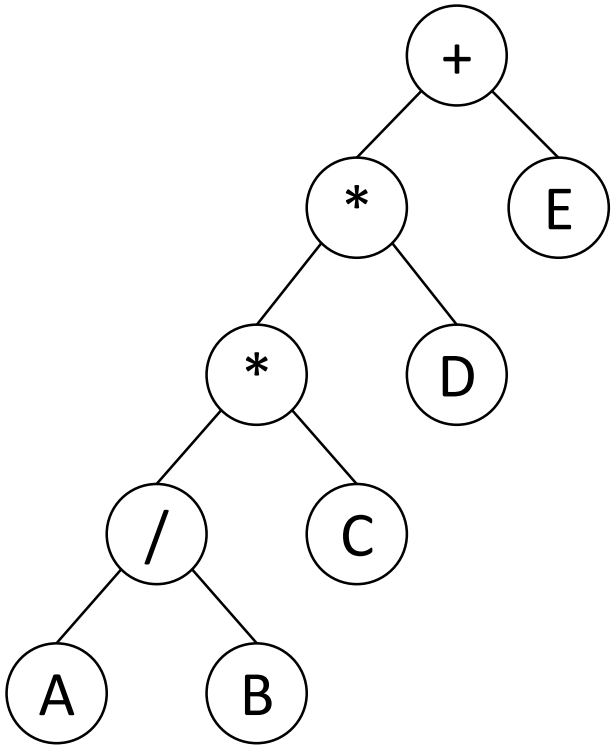
Level-Order Traversal Code

```
void Tree::LevelOrder()
{
  Queue<TreeNode*> q;

  TreeNode *currentNode = root;
  while (currentNode) {
    print(currentNode);

    for (each Child pointer)
      q.Push(the Child pointer);

    if (q.IsEmpty())
      return;
    currentNode = q.Front();
    q.Pop();
  }
}
```

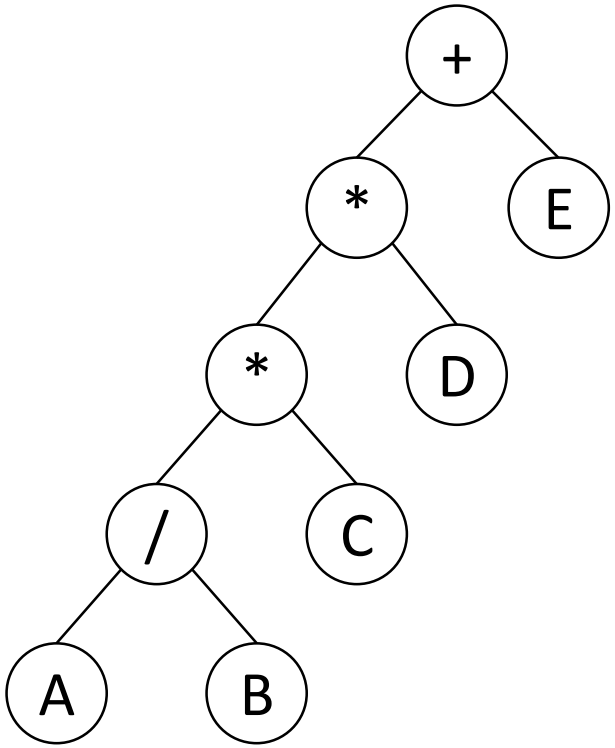


+ * E * D / C A B

Preorder

```
void Tree::Preorder()
{
   pre(root);
}
```

```
void Tree::pre(TreeNode * p)
{
  // this is a recursive function
  print(p);
  for (each Child pointer)
     pre(the Child pointer);
}
```

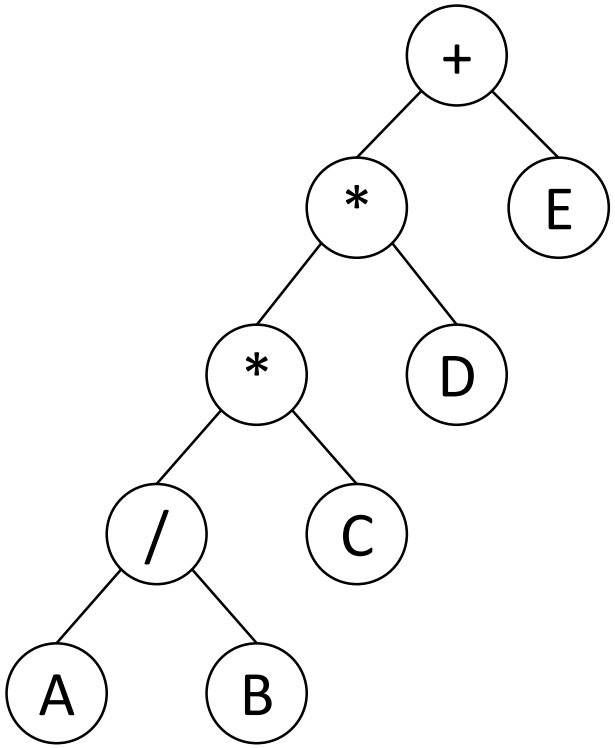


+ * * / A B C D E

Postorder

```
void Tree::Postorder()
{
   post(root);
}
```

```
void Tree::post(TreeNode * p)
{
  // this is a recursive function
  for (each Child pointer)
     post(the Child pointer);
  print(p);
}
```



A B / C * D * E +