4.3a Perform depth-first, pre-order depth-first and post-order depth-first traversals on the tree shown in Figure 1.

```
Figure 1. A general tree.
```

4.3b What is the maximum size of the queue if a queue is used for performing a breadth-first traversal on the tree in Figure 1?

4.3c You are given that a tree has pre- and post-order depth first traversals of

- A B D E G C F
- D G E B F C A

respectively. Can you determine the original tree from this information?

Hint: if x is a descendant of y, then where will y sit relative to x in both of these orders?

4.3c You are given that a tree has pre-order depth-first and breadth-first traversals of

- A B C D E G F
- A B C D E F G

respectively. Can you determine the original tree from this information?

4.3e Write a traversal that prints out the leaf nodes in the order in which they appear in an ordered general tree from left-to-right.

```cpp
template <typename Type>
void Simple_tree<Type>::print_leaves() const {
```
4.3f Right a depth-first traversal that:

1. Prints out the elements stored in the nodes at depth $n$ where $n$ is a parameter passed by the user, and
2. Does not visit any nodes beyond depth $n$.

Hint: What information do you have to pass to the children?

```cpp
template <typename Type>
void Simple_tree<Type>::print_at_depth( int n ) const {

4.3g For each of the following, indicate whether the function prints the nodes in a pre-order depth-first traversal order, post-order depth-first traversal order, or a breadth-first traversal order. The stacks and queues work as expected from class. Assume that the Simple_tree data structure uses a doubly linked list where each node has both next and previous pointers.

```cpp
template<typename Type>
void Simple_tree<Type>::first_traversal() {
    Single_list< Simple_tree * > list;
    list.push_front( this );

    while ( !list.empty() ) {
        Simple_tree *ptr = list.pop_front();
        std::cout << ptr->retrieve() << std::endl;

        for ( Double_node< Simple_tree * > *node = ptr->children.tail();
            node != nullptr;
            node = node->previous() ) {
            list.push_front( node->retrieve() );
        }
    }
}

template<typename Type>
void Simple_tree<Type>::second_traversal() {
    std::cout << ptr->retrieve() << std::endl;

    for ( Double_node< Simple_tree * > *node = children.head();
        node != nullptr;
        node = node->next() ) {
        node->retrieve()->second_traversal();
    }
}
template<typename Type>
void Simple_tree<Type>::third_traversal() {
    Single_list< Simple_tree * > list;
    list.push_back( this );

    while ( !list.empty() ) {
        Simple_tree *ptr = list.pop_front();
        std::cout << ptr->retrieve() << list::endl;

        for ( Double_node< Simple_tree * > *node = ptr->children.head();
            node != nullptr;
            node = node->next() ) {
            list.push_back( node->retrieve() );
        }
    }
}

template<typename Type>
void Simple_tree<Type>::fourth_traversal() {
    Single_list< Simple_tree * > list;
    list.push_front( this );

    while ( !list.empty() ) {
        Simple_tree *ptr = list.pop_front();

        for ( Double_node< Simple_tree * > *node = ptr->children.tail();
            node != nullptr;
            node = node->previous() ) {
            list.push_front( node->retrieve() );
        }
        std::cout << ptr->retrieve() << std::endl;
    }
}

template<typename Type>
void Simple_tree<Type>::fifth_traversal() {
    for ( Double_node< Simple_tree * > *node = children.head();
        node != nullptr;
        node = node->next() ) {
        node->retrieve()->fifth_traversal();
    }
    std::cout << ptr->retrieve() << std::endl;
}

4.3h Under what conditions would a pre-order and a breadth-first traversal be the same?
4.3 Suppose a directory structure has \( N \) files stored in \( n \) directories. Answer the following questions:

a. What is the run time of a traversal that prints out the directory names?
b. What is the run time of a traversal that prints out the file names?
c. How does your answer to Part b of this question change if you know that \( N = \Omega(n) \)?