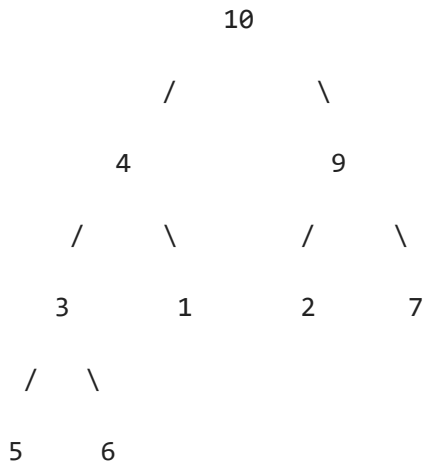
Homework 3

20173821 Ka Yoon Kim



In [3]:
```
graph = {10:[4,9],
         4:[3,1],
         9:[2,7],
         3:[5,6],
         1:None,
         2:None,
         5:None,
         6:None,
         7:None}
```

In [4]:
```
root = 10
```

1. Construct your tree using your own programming language and perform BFS from root

In [5]:
```
from collections import deque

def BFS(graph,root):
    visited = []
    queue = deque([root])

    while queue:
        n = queue.popleft()
        visited.append(n)
        if graph[n] == None: continue
        else:
            for i in range(len(graph[n])):
                queue.append(graph[n][i])
    return visited
```

image

In [6]:
```
print(BFS(graph,root))
```

```
[10, 4, 9, 3, 1, 2, 7, 5, 6]
```

2. Construct your tree using your own programming language and perform DFS from root

In [7]:
```
def DFS(graph, root):
    visited = []
    stack = [root]
    while stack:
        n=stack.pop()
        visited.append(n)
        if graph[n] == None: continue
        else:
            for i in range(len(graph[n])):
                stack.append(graph[n][i])
    return visited
```

If the order starts in right side, the result could be as follows.

In [8]:
```
DFS(graph,root)
```

Out[8]:
```
[10, 9, 7, 2, 4, 1, 3, 6, 5]
```

3. Explain how you would do BFS and DFS on general graph

- explain how you would read the input and how you keep the data
- explain how you are going to implement BFS, DFS , and how it is different from (1, 2)

I would like to read the input graph and transform into adjacency list that is composed of list of vertices.

On the previous BFS and DFS that which input was the tree, I simply implemented the tree using dictionary, shape of {node(parent):[child]}. On the other hand, for the general graph as an input, I will use the dictionary to transform and keep the data as I did before, but in the shape of {node A :[a node which node A is heading(edge)]}. If I know the input graph, I will directly add the nodes and edges by myself. If I don't and input graphs are random, I would like to make a class named 'Graph' which is dictionary of list(using defaultdict(list) from collections library) and append the nodes and edges. With transformed input data, I will put these as a input on BFS and DFS.

For implementing BFS, I will use the data structure called queue because it is FIFO and fits on BFS. While queue is empty, every node in graph will be enqueued and marked as visited. While dequeue the node, enqueue the adjacent nodes and mark as visited. On the other hand, for implementing DFS, the stack is needed as a data structure because it is LIFO and fits on DFS. While stack is being empty, every node in graph will be pushed and marked as visited. While it pops, push the adjacent nodes and mark as visited. These implementation has a difference from (1,2) which is the time complexity. Previous tree BFS and DFS only needed O(Node) because they visit every node exactly once. However, BFS and DFS on general graph takes O(Vertices + Edges) of time complexity.