RA1911026010029
Thejaswin S

EXP-4 Implementation and Analysis of DFS and BFS for an application

AIM

To implement and Analyse DFS(Depth for Search) and BFS(Breadth for Search) for an application.

BFS

BFS stands for Breadth-First Search is a vertex-based technique for finding the shortest path in the graph. It uses a Queue data structure that follows first in first out. In BFS, one vertex is selected at a time when it is visited and marked then its adjacent are visited and stored in the queue. It is slower than DFS.

DFS

DFS stands for Depth First Search is an edge-based technique. It uses the Stack data structure, performs two stages, first visited vertices are pushed into the stack, and second if there is no vertices then visited vertices are popped.

Algorithm:

So, basically you have to use Queue as a data structure

Follow algorithm for BFS

1. Start with one node of graph. It may be travers from left to right.
2. So there are two factrors which are important for traversing .. first that the node is visited or not and second one is there any adjecent nodes for current node
3. If we are visiting current node then add node into queue
4. If the current node having adjecent nodes then add those nodes into queue
5. If there are no more adjecent are found then print the current node and place the pointer to first node of queue
6. Repeat these until all nodes get traversed.

For implementation of DFS you must know Stack data structure

Follow these steps

1. In this traversal you must know the operation of stack
2. Start from one of the graph node it may be start from left to right
3. Visit the current node and push it to stack
4. Check the top of the stack having adjecent nodes
5. If yes then push the adjecent nodes into stack

6. If no then print the top of the stack and pop it from stack
7. Repeat these steps until you will visit all node

Program:

```
graph = {
  '0' : [ '1','3'],
  '1' : ['3','2','5','6'],
  '2' : ['1','3','4','5'],
  '3' : ['0','1','2','4'],
  '4' : ['3','2','6'],
  '5' : ['1','2'],
  '6' : ['1','4']
}

visited_bfs = []
queue = []

def bfs(visited_bfs, graph, node):
  visited_bfs.append(node)
  queue.append(node)

  while queue:
    s = queue.pop(0)
    print (s, end = " ")
    for neighbour in graph[s]:
      if neighbour not in visited_bfs:
        visited_bfs.append(neighbour)
        queue.append(neighbour)

visited = set()

def dfs(visited, graph, node):
    if node not in visited:
        print (node, end=" ")
        visited.add(node)
        for neighbour in graph[node]:
            dfs(visited, graph, neighbour)

print("BFS:" , end =" ")
bfs(visited_bfs, graph, '0')
print('\n')
print("DFS:" , end =" ")
dfs(visited, graph, '0')
print('\n')
```

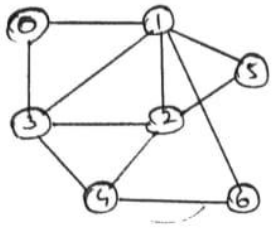
exp4.py × bash - "ip-172-31-2-88" × exp

```python
graph = {
  '0' : [ '1','3'],
  '1' : ['3','2','5','6'],
  '2' : ['1','3','4','5'],
  '3' : ['0','1','2','4'],
  '4' : ['3','2','6'],
  '5' : ['1','2'],
  '6' : ['1','4']
}

visited_bfs = []
queue = []

def bfs(visited_bfs, graph, node):
  visited_bfs.append(node)
  queue.append(node)

  while queue:
    s = queue.pop(0)
    print (s, end = " ")

    for neighbour in graph[s]:
      if neighbour not in visited_bfs:
        visited_bfs.append(neighbour)
        queue.append(neighbour)

visited = set()

def dfs(visited, graph, node):
    if node not in visited:
        print (node, end=" ")
        visited.add(node)
        for neighbour in graph[node]:
            dfs(visited, graph, neighbour)

print("BFS:" , end =" ")
bfs(visited_bfs, graph, '0')
print('\n')
print("DFS:" , end =" ")
```

Output:

```
RA1911026010022:~/environment/RA1911026010029/exp4 $ python3 exp4.py
BFS: 0 1 3 2 5 6 4

DFS: 0 1 3 2 4 6 5
```

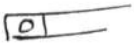
RA1911026010029
Thejaswin S



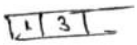
BFS

If 0 is considered as root node.

0 is inserted into queue

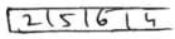


0 is popped and neighbours of 0 is added



1 is popped & neighbours of 1 is added

3 is popped and neighbours of 3 is added, Existing neighbours ignored.



after that all elements in queue is ~~pop~~ popped out.

0 1 3 2 5 6 4 – Order

DFS

If 0 is considered as root node.

push 0 into stack 

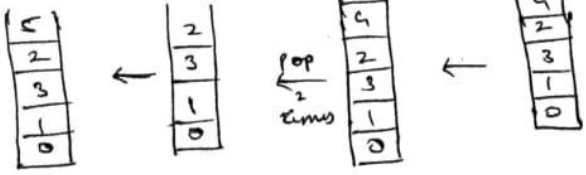
Neighbours of 0 are 1 and 3. consider 1 and traverse further

neighbours of 1 – 3, 2, 5, 6

push 3 into stack 

neighbours of 3 unvisited – 2, 4, . push 2 into stack

neighbours of 2 unvisited – 4, 6, 5 . push 4 into stack



Now backtrack, check for unvisited nodes. Since all nodes are visited, we can pop all out.

0 1 3 2 4 6 5 – Order

RA1911026010029
Thejaswin S

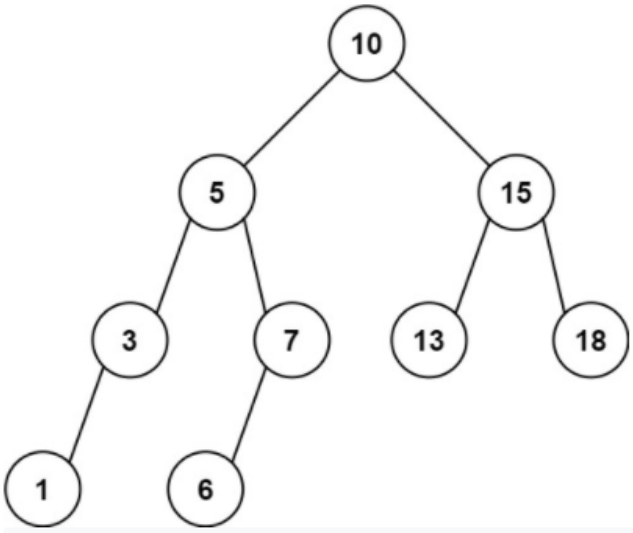
Application

RANGE SUM OF BINARY SEARCH TREE

Aim:

Given the root node of a binary search tree, return the sum of values of all nodes with a value in the range [low, high] using depth-first and then breadth-first search.
While:

- The number of nodes in the tree is in the range [1, 2 * 10^4].
- 1 <= Node.Val <= 10^5
- 1 <= low <= high <= 10^5
- All Node.value are unique.




```
Input: root = [10,5,15,3,7,13,18,1,null,6], low = 6, high = 10
```

Algorithm :

1. We traverse the tree using a depth first search.
2. If node.value falls outside the range [L, R], (for example node.val < L), then we know that only the right branch could have nodes with value inside [L, R].
3. We showcase two implementations - one using a recursive algorithm, and one using an
iterative one.
4. Time Complexity: O(N)O(N), where NN is the number of nodes in the tree.
5. Space Complexity: O(N)O(N)
6. For the recursive implementation, the recursion will consume additional space in the function call stack. In the worst case, the tree is of chain shape, and we will reach all the way down to the leaf node.

7. For the iterative implementation, essentially we are doing a BFS (Breadth-First Search)
traversal, where the stack will contain no more than two levels of the nodes. The maximal
number of nodes in a binary tree is N/2.
8. Therefore, the maximal space needed for the stack would be O(N)O(N).

DFS

```
class TreeNode:
    def __init__(self, val=0, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right

# RECURSIVE APPROACH

def rangeSumBST029(root, L, R):
    ans = 0
    stack = [root]
    while stack:
        node = stack.pop()
        if node:
            if L <= node.val <= R:
                ans += node.val
            if L < node.val:
                stack.append(node.left)
            if node.val < R:
                stack.append(node.right)
    return ans

bst = TreeNode(10)
bst.left = TreeNode(5)
bst.right = TreeNode(15)
bst.left.left = TreeNode(3)
bst.left.right = TreeNode(7)
bst.right.left = TreeNode(13)
bst.right.right = TreeNode(18)
bst.left.left.left = TreeNode(1)
bst.left.right.left = TreeNode(6)
mn = int(input("Enter the Lower value of the range : "))
mx = int(input("Enter the Higher value of the range : "))
```

```
dfs = rangeSumBST029(bst, mn, mx)
print(f"The sum of the nodes in the range {mn} and {mx} is {dfs}")
```

Output:

```
RA1911026010022:~/environment/RA1911026010029/exp4 $ python3 dfs.py
Enter the Lower value of the range : 6
Enter the Higher value of the range : 10
The sum of the nodes in the range 6 and 10 is 23
```

BFS

```
class TreeNode:
    def __init__(self, val=0, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right

class Solution029(object):
    def rangeSumBST(self, root, L, R):
        if root == None:
            return 0
        res = 0
        q = [root]
        while q:
            next = []
            for node in q:
                if L <= node.val <= R:
                    res += node.val
                if node.left:
                    next.append(node.left)
                if node.right:
                    next.append(node.right)
            q = next
        return res

bst = TreeNode(10)
bst.left = TreeNode(5)
bst.right = TreeNode(15)
bst.left.left = TreeNode(3)
bst.left.right = TreeNode(7)
bst.right.left = TreeNode(13)
bst.right.right = TreeNode(18)
bst.left.left.left = TreeNode(1)
```

```
bst.left.right.left = TreeNode(6)
mn = int(input("Enter the Lower value of the range : "))
mx = int(input("Enter the Higher value of the range : "))
bfs=Solution029().rangeSumBST(bst,mn,mx)
print(f"The sum of the nodes in the range {mn} and {mx} is {bfs}")
```

Output:

```
RA1911026010022:~/environment/RA1911026010029/exp4 $ python3 bfs.py
Enter the Lower value of the range : 6
Enter the Higher value of the range : 10
The sum of the nodes in the range 6 and 10 is 23
```

```
Input: root = [10,5,15,3,7,13,18,1,null,6], low = 6, high = 10
Output: 23
Explanation: Nodes 6, 7, and 10 are in the range [6, 10]. 6 + 7 + 10 = 23.
```

Observation:

Thus concept of Depth for search and breadth for search is studied and application is implemented and understood thought this experiment.

Result :

Program for DFS and BFS has been implemented and Successfully found the sum of nodes in a binary search tree between any given range (min, max) using both depth first search and breadth first search approach.