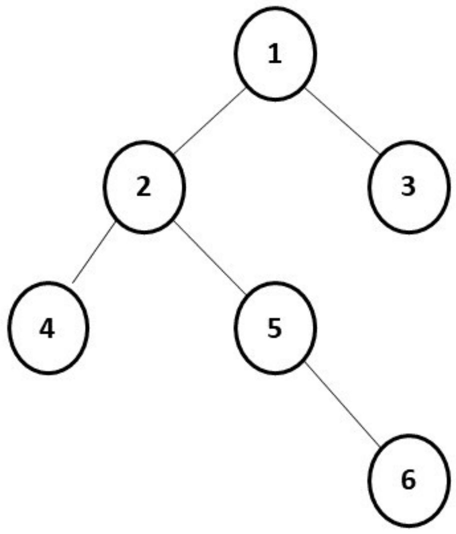
Morris Preorder Traversal of a Binary Tree

Problem Statement: Morris Preorder Traversal of a Binary tree. Given a Binary Tree, find the Morris preorder traversal of Binary Tree.

Example:



Output: Preorder Traversal of this binary tree will be:- 1,2,4,5,6,3

Pre-requisite: Morris Inorder Traversal

***Disclaimer**: Don't jump directly to the solution, try it out yourself first.*

Solution:

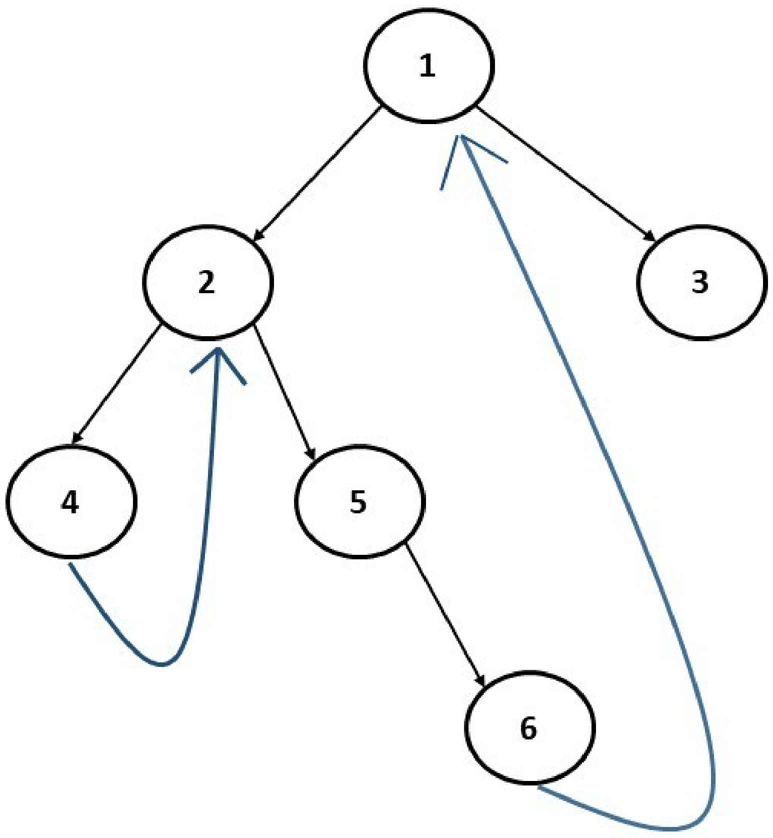
Intuition:

As we have learned **Morris's Inorder traversa**l in this tutorial, Morris's preorder traversal is quite similar. We need to understand the difference in the way we are traversing the tree.

In Morris Inorder Traversal, we are traversing the tree in the way:- Left, Root, Right. In Morris Preorder traversal we want to traverse the tree in the way:- Root, Left, Right. Therefore, the following code changes are required:

- When the current node has a left child: In Morris Inorder traversal we would simply make the new thread required and then move the current node to its left. When we return to the original node from the thread we print it. But in Morris preorder traversal, we want to print the root before visiting the left child, therefore after we set the thread, we first print the current node's value and then move it to its left.
- When the current node has no left child: This case remains exactly the same because if there is nothing to visit on the left, we will want to print the current node's value and move right in both traversals. Therefore, there will be no code modification.

The threaded binary tree will remain the same.



Approach:
The algorithm can be described as:
When we are currently at a node, the following cases can arise:

- **Case 1:** When the current node has no left subtree. In this scenario, there is nothing to be traversed on the left side, so we simply print the value of the current node and move to the right of the current node.
- **Case 2:** When there is a left subtree and the right-most child of this left subtree is pointing to null. In this case we need to set the right-most child to point to the current node, instead of NULL, print the current node value and move to the left of the current node.
- **Case 3:** When there is a left subtree and the right-most child of this left-subtree is already pointing to the current node. In this case we know that the left subtree is already visited so we need to reset the last node to NULL and move the current node to its right.

Note: Case 3 is very important as we need to remove the new links added to restore the original tree.
To summarize, at a node whether we have to move left or right is determined whether the node has a left subtree. If it doesn't we move to the right. If there is a left subtree then we see its rightmost child. If the rightmost child is pointing to NULL, we print the current node's value and move it to its left. If the rightmost child is already pointing towards the current node, we remove that link and move to the right of the current node. We will stop the execution when the current points to null and we have traversed the whole tree.
Code:

- C++ Code
- Java Code

```
#include <bits/stdc++.h>

using namespace std;

struct node {
   int data;
   struct node * left, * right;
};

vector < int > preorderTraversal(node * root) {
   vector < int > preorder;

   node * cur = root;
   while (cur != NULL) {
      if (cur -> left == NULL) {
         preorder.push_back(cur -> data);
         cur = cur -> right;
      } else {
         node * prev = cur -> left;
```

```
        while (prev -> right != NULL && prev -> right != cur) {
          prev = prev -> right;
        }

        if (prev -> right == NULL) {
          prev -> right = cur;
          preorder.push_back(cur -> data);
          cur = cur -> left;
        } else {
          prev -> right = NULL;
          cur = cur -> right;
        }
      }
    }
  return preorder;
}

struct node * newNode(int data) {
  struct node * node = (struct node * ) malloc(sizeof(struct node));
  node -> data = data;
  node -> left = NULL;
  node -> right = NULL;

  return (node);
}

int main() {

  struct node * root = newNode(1);
  root -> left = newNode(2);
  root -> right = newNode(3);
  root -> left -> left = newNode(4);
  root -> left -> right = newNode(5);
  root -> left -> right -> right = newNode(6);

  vector < int > preorder;
  preorder = preorderTraversal(root);

  cout << "The Preorder Traversal is: ";
  for (int i = 0; i < preorder.size(); i++) {
    cout << preorder[i] << " ";
  }

  return 0;
}
```

Output:

```
The Preorder Traversal is: 1 2 4 5 6 3
```

Time Complexity: O(N).

Space Complexity: O(1)