Morris Traversal (Inorder)

🔍 Concept of Morris Traversal (Inorder):

Normally, inorder traversal (left → root → right) uses recursion or a stack. But Morris traversal uses **threading**: it temporarily modifies the tree to avoid stack/recursion and achieves **O(1) space complexity**.

🧠 Key idea:

- For each node, if `left` child exists:
  - Find its **inorder predecessor** (rightmost node in left subtree).
  - Make its `right` point to current node (create a thread).
  - Move current to left child.
- If `left` doesn't exist:
  - Visit the node, move to `right`.

When visiting the threaded node again:

- Remove the thread (restore original tree structure).
- Visit current node, go to right.

🌳 Example Tree:

Let's dry run on this tree:

```
    4
   / \
  2   5
 / \
1   3
```

Inorder Traversal should be: **1 2 3 4 5**

🧾 Code Explanation + Dry Run:

```
vector<int> inorderTraversal(TreeNode* root) {
    vector<int> ans;
    TreeNode* c = root;  // current node
    while(c != NULL){
```

Start at root (4):

🔁 First Iteration (c = 4):

- 4 has left child → go to left (2), find its **rightmost**:
  - 2 → right = 3 (3 has no right), so 3 is predecessor.
- `p->right` is NULL → make thread `3->right = 4`
- move `c = c->left` → `c = 2`

🔁 Second Iteration (c = 2):

- 2 has left → go to left (1), find rightmost (1 itself)
- `p->right` is NULL → make thread `1->right = 2`
- move `c = c->left` → `c = 1`

🔁 Third Iteration (c = 1):

- 1 has no left → `ans.push_back(1)` → `ans = [1]`
- move `c = c->right` → c = 2 (threaded)

🔁 Fourth Iteration (c = 2):

- 2 has left (1), go to rightmost of 1 → p = 1
- `p->right == c` → Thread exists → remove `1->right = NULL`
- `ans.push_back(2)` → `ans = [1, 2]`
- move `c = c->right` → `c = 3`

🔁 Fifth Iteration (c = 3):

- 3 has no left → `ans.push_back(3)` → `ans = [1, 2, 3]`
- move `c = c->right` → `c = 4` (threaded)

🔁 Sixth Iteration (c = 4):

- 4 has left (2), go to rightmost in left (3)
- `p->right == c` → Thread exists → remove it
- `ans.push_back(4)` → `ans = [1, 2, 3, 4]`
- move `c = c->right` → `c = 5`

🔁 Seventh Iteration (c = 5):

- 5 has no left → `ans.push_back(5)` → `ans = [1, 2, 3, 4, 5]`
- move `c = c->right` → `c = NULL`

✅ Final Output:

```
return ans; // [1, 2, 3, 4, 5]
```

🔁 Summary Flow:

```
1. Go left till null, while making threads
2. When you hit null or a threaded node:
   - Visit node
   - Restore right pointer if needed
   - Go right
```

✅ 🔁 Morris Inorder Traversal – Full Code with Dry Run

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode() : val(0), left(nullptr), right(nullptr) {}
 *     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
 *     TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left), right(ri
ght) {}
 * };
 */
class Solution {
public:
    vector<int> inorderTraversal(TreeNode* root) {
        vector<int> ans;
        TreeNode* c = root;  // current node
        while(c != NULL){
            if(c->left != NULL){
                TreeNode* p = c->left;
                while(p->right != NULL && p->right != c){
                    p = p->right;
                }
                if(p->right == NULL){
                    // Threading: link rightmost of left subtree to current
                    p->right = c;
                    c = c->left;
                }
                else{
                    // Thread already exists: remove and visit node
                    p->right = NULL;
                    ans.push_back(c->val);
```

```
                c = c->right;
            }
        }
        else{
            // No left child: just visit and go right
            ans.push_back(c->val);
            c = c->right;
        }
    }
    return ans;
  }
};
```

🧠 Morris Inorder Traversal Ka Basic Idea (Hinglish mein)

📌 Bina Recursion ya Stack ke Inorder Traversal kaise karein?

- Agar kisi node ka **left child hai**, to uska **inorder predecessor** (left subtree ka rightmost node) dhoondo.
- Us predecessor ke `right` pointer ko temporarily current node par point kara do (thread banao).
- Left subtree me jao.
- Jab wapas aate ho (via thread), to:
    - Thread tod do.
    - Node ko visit karo (value push_back).
    - Right subtree me jao.
- Agar left child nahi hai:
    - Visit karo aur right me jao.

⏱️ **Time Complexity:** O(n)

💾 **Space Complexity:** O(1) (kyunki koi stack nahi use kiya)

🌳 Example Tree:

```
    4
   / \
  2   5
 / \
1   3
```

Inorder (Left → Root → Right): `1 2 3 4 5`

🔁 Dry Run – Step by Step

▶️ First Iteration (c = 4)

- Left child hai → 2
- Rightmost in left = 3
- 3 ka right NULL hai → thread banao: `3->right = 4`
- Move to left → `c = 2`

▶️ Second Iteration (c = 2)

- Left child hai → 1
- Rightmost in left = 1
- 1 ka right NULL → `1->right = 2` (thread)
- Move to left → `c = 1`

▶️ Third Iteration (c = 1)

- No left → Visit `1` → `ans = [1]`
- Move to right → `c = 2` (threaded)

▶️ Fourth Iteration (c = 2)

- 1 ka thread exists → Remove it

- Visit `2` → `ans = [1, 2]`
- Move to right → `c = 3`

▶️ Fifth Iteration (c = 3)

- No left → Visit `3` → `ans = [1, 2, 3]`
- Move to right → `c = 4` (threaded)

▶️ Sixth Iteration (c = 4)

- 3 ka thread exists → Remove it
- Visit `4` → `ans = [1, 2, 3, 4]`
- Move to right → `c = 5`

▶️ Seventh Iteration (c = 5)

- No left → Visit `5` → `ans = [1, 2, 3, 4, 5]`
- Move to right → `c = NULL`

🧾 Final Output:

```
return ans;  // [1, 2, 3, 4, 5]
```

🔄 Overall Flow (Summary in Hinglish):

```
1️⃣ Left subtree me jaate jao, rightmost node dhoondo
2️⃣ Thread banao: rightmost node → current node
3️⃣ Jab thread dikh jaye, uska matlab left ho chuka hai traverse
4️⃣ Thread tod do, node visit karo
5️⃣ Right subtree me chale jao
```
