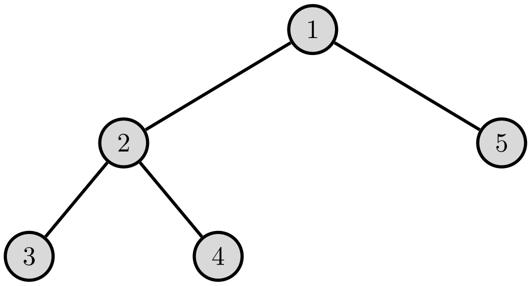
1 Morris Inorder Tree Traversal

1.1 source code

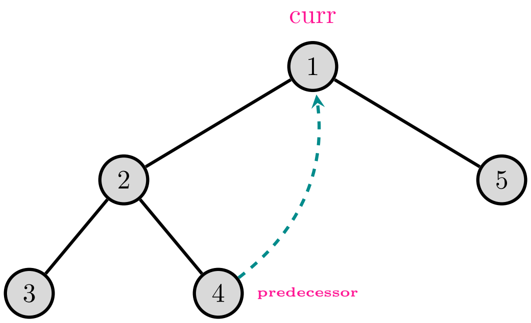
```
/* Find predecessor */
Node* find_inorder_predecessor(Node* curr)
{
  Node* predecessor=curr->left;
  while (predecessor->right != NULL && predecessor->right != curr)
    predecessor=predecessor->right;

    return predecessor;
}

/* Morris Inorder traversal */
void morris_inorder_traversal(Node* root)
{
  Node *curr;
  curr = root;

  while (curr!=NULL)
  {
    /* if curr doesn't have left child */
    if (curr->left==NULL)
    {
      cout<<curr->data<<"␣";   iteration 3
      curr=curr->right;   curr moves from iteration 3 to iteration 4
    }
    else
    {
        /* finding inorder predecessor */
    Node* predecessor=find_inorder_predecessor(curr);

      /* Make curr as the right child of its inorder predecessor */
      if(predecessor->right==NULL)
      {
        predecessor->right=curr;
        curr=curr->left;
      }
      else
      {
        predecessor->right=NULL;   remove iteration 4 link
        cout<<curr->data<<"␣";
        curr=curr->right;   move curr to iteration 5
      }
    }
  }
}

int main()
{
  Node* root = new_node(1);
  root->left = new_node(2);
  root->right = new_node(5);
  root->left->left = new_node(3);
  root->left->right = new_node(4);
 /* this input tree is shown in above figure */
  cout<<"\nMorris␣Inorder␣Traversal␣of␣the␣graph:␣";
  morris_inorder_traversal(root);

  return 0;
}
```

1.2 Input Tree



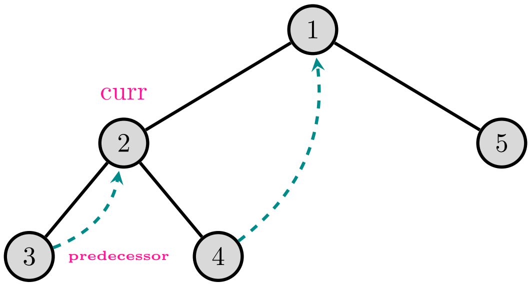
1.3 Iterations

1.3.1 Iteration 1



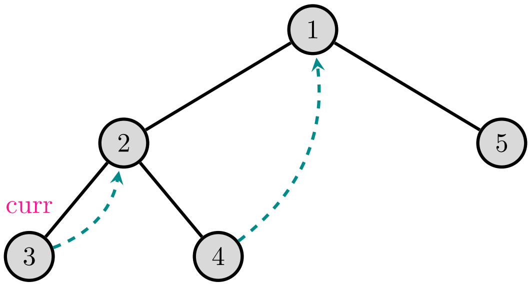
`Output: -`

1.3.2 iteration 2



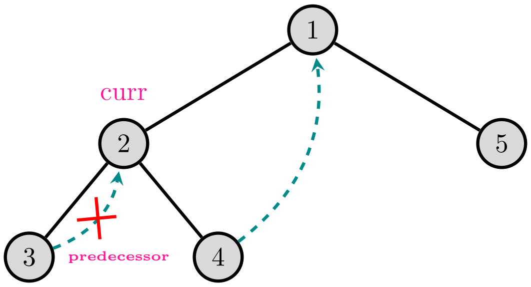
`Output: -`

1.3.3 iteration 3



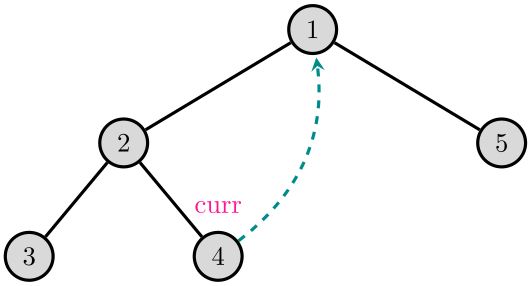
```
Output: 3
```

1.3.4 iteration 4



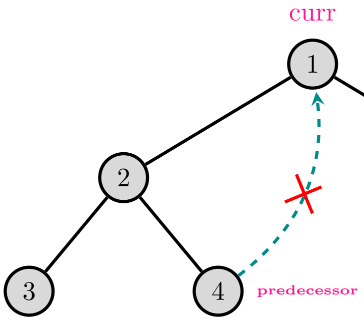
```
Output: 3 2
```

1.3.5 iteration 5



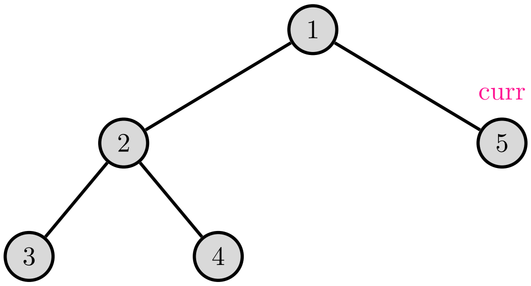
```
Output: 3 2 4
```

1.3.6 iteration 6



```
Output:  3 2 4 1
```

1.3.7 iteration 7



```
Output:  3 2 4 1 5
```