# DSA LAB PROGRAMS-Trees-Graphs

By Sindhu Sankati AP19110010477 CSE-G

---

PROGRAM 30

## Title: Binary Tree Traversal

## Objective:

Create a binary tree and output the data with 3 tree traversals

## Explanation:

A  binary tree, is a tree in which no node can have more than two  children.
Pre-order: The travesal goes in format root, left,right.
In-order: The traversal goes in format left, root, right.
Post-order: The traversal goes in format left, right, root.

## Pseudo Code:

>PreOrder

```
1. PREORDER(root):
2.     IF root == NULL RETURN
3.     PRINT root->data
4.     PREORDER(root->left)
5.     PREORDER(root->right)
6. END
```

>InOrder

```
1. PREORDER(root):
2.     IF root == NULL RETURN
3.     PREORDER(root->left)
4.     PRINT root->data
5.     PREORDER(root->right)
6. END
```

>PostOrder

```
1. POSTORDER(root):
2.     IF root == NULL RETURN
3.      PREORDER(root->left)
4.     PREORDER(root->right)
5.     PRINT root->data
6.     END
```

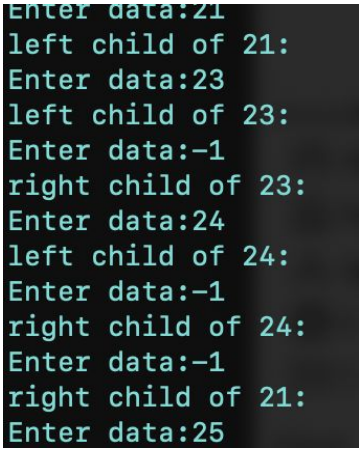
**Code:**

```c
#include <stdio.h>
#include <stdlib.h>
#define initmemory() (struct node*)malloc(sizeof(struct node))

struct node {

    int data;
    struct node *left;
    struct node *right;

};

struct node* insert(){
    struct node *newnode;
    int x;
    printf("Enter data:");
    scanf("%d",&x);

    if(x==-1)
        return NULL;

    newnode = initmemory();
    newnode->data = x;
    printf("left child of %d:\n",x);
    newnode->left = insert();
    printf("right child of %d:\n",x);
    newnode->right = insert();
    return newnode;

}

void postOrder(struct node *root) {
    if (root == NULL){
        return;
    }
    postOrder(root->left);
    postOrder(root->right);
    printf("%d ",root->data);
}

void inOrder(struct node *root) {
```

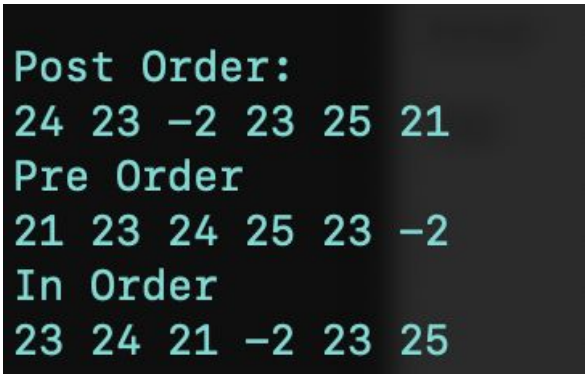
```c
    if(root == NULL) return;
    inOrder(root->left);
    printf("%d ",root->data);
    inOrder(root->right);
}

void preOrder( struct node *root) {
    if(root == NULL)return;
    printf("%d ",root->data);
    preOrder(root->left);
    preOrder(root->right);
}

int main() {

    struct node* root = insert();

    int num,i;
    int data;

    printf("\nPost Order:\n");
    postOrder(root);

    printf("\nPre Order\n");
    preOrder(root);

    printf("\nIn Order\n");
    inOrder(root);
    return 0;

}
```

## Outputs:

sample input




output




## Conclusion:

All the travesrals are are O(n) is worst, best, average cases.

PROGRAM 31

## Title: Binary Search Tree

## Objective:

Create a Binary Search Tree(BST) and search for a given value in BST.

## Explanation:

Searching for a node is similar to inserting a node. We start from root, and then go left or right until we find (or not find the node).

## Pseudo Code:

>Sum N natural numbers

```
1. Search(N, T) :
2.  false   if T is empty
3.   true  if T = N
4.  search(N, T.left) if N < T
5.  search(N, T.right) if N > T
6. END
```

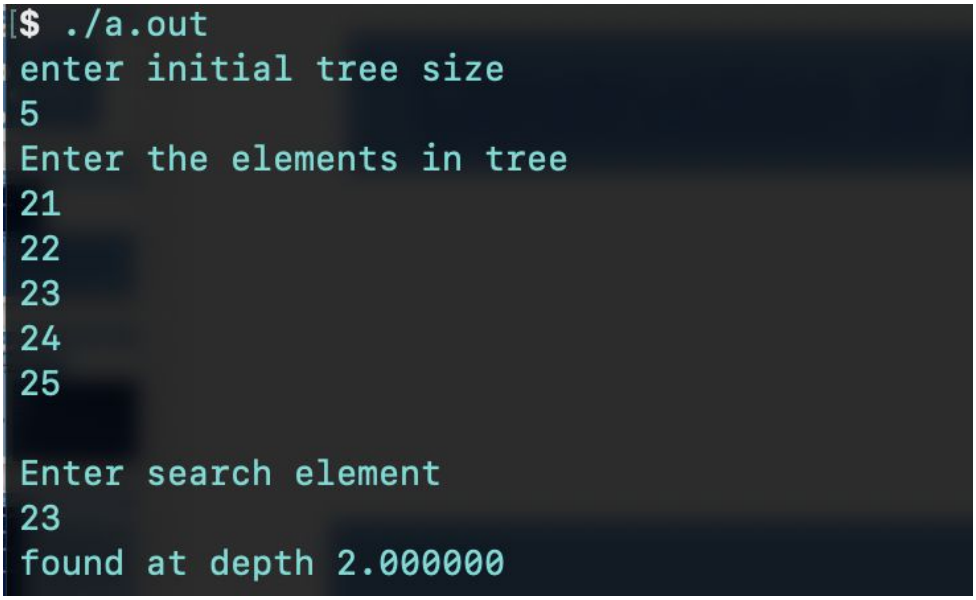
## Code:

```c
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#define initmemory() (struct node*)malloc(sizeof(struct node))

typedef struct node {

    int data;
    struct node *left;
    struct node *right;

}node;

node* insert(node* root, int data) {

    if(root == NULL) {

        node* node = initmemory();

        node->data = data;
```

```
            node->left = NULL;
            node->right = NULL;
            return node;

        } else {

            if(data <= root->data) {
                root->left = insert(root->left, data);
            }

            else {
                root->right = insert(root->right, data);;
            }
            return root;
        }
    }
    int bstSearch(node* root, int search)
    {
        if (root == NULL)
        return 0;
        if(root->data == search)
        return 1;
        if (root->data < search)
        return 2*(bstSearch(root->right, search));
        return 2*(bstSearch(root->left, search));
    }
    int main(int argc, char const *argv[])
    {
        node* root = NULL;

        int num,i,search,data,pos;

        printf("enter initial tree size\n");
        scanf("%d", &num);

        printf("Enter the elements in tree\n");
        for(i=0;i<num;i++){
            scanf("%d", &data);
```

```
            root = insert(root, data);
        }

        printf("\nEnter search element\n");
        scanf("%d",&search);
        pos = bstSearch(root,search);
        printf("found at depth %f\n",log2(pos));


        return 0;
    }
```

**Outputs:**




**Conclusion:**

Time complexity for serching is O(log n). This search algorthm wont be efficent to find the exat location in the tree.

PROGRAM 33

## Title: Shortest paths in graphs

## Objective:

Write a program to find All-to-all Shortest paths in a Graph.

## Explanation:

BFS is a traversing algorithm where you should start traversing from a selected node (source or starting node) and traverse the graph layerwise thus exploring the neighbour nodes (nodes which are directly connected to source node). You must then move towards the next-level neighbour nodes.

## Pseudo Code:

>BFS

```
1. BFS (G, s)                  //Where G is the graph and s is the source
   node
2.       let Q be queue.
3.       Q.enqueue( s ) //Inserting s in queue until all its neighbour
   vertices are marked.
4.
5.       mark s as visited.
6.       while ( Q is not empty)
7.            //Removing that vertex from queue,whose neighbour will be
   visited now
8.            v  =  Q.dequeue( )
9.
10.            //processing all the neighbours of v
11.            for all neighbours w of v in Graph G
12.                 if w is not visited
13.                           Q.enqueue( w )             //Stores w in Q to
   further visit its neighbour
14.                           mark w as visited.
15.  D
```

## Code:

```c
#include <stdio.h>
#include <stdlib.h>
#define SIZE 40

struct queue {
  int items[SIZE];
  int front;
```

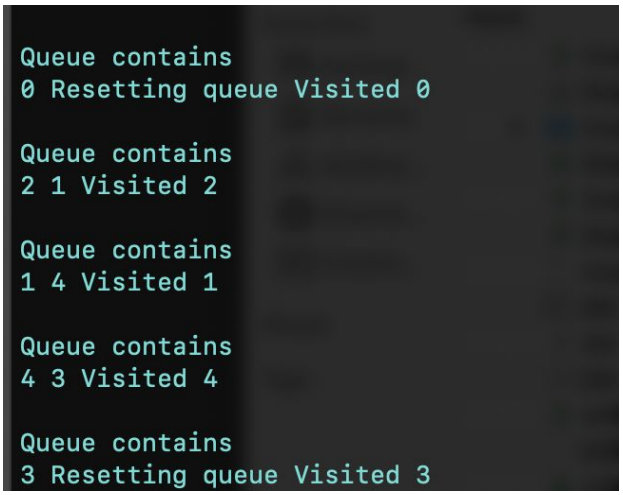
```
  int rear;
};

struct queue* createQueue();
void enqueue(struct queue* q, int);
int dequeue(struct queue* q);
void display(struct queue* q);
int isEmpty(struct queue* q);
void printQueue(struct queue* q);

struct node {
  int vertex;
  struct node* next;
};

struct node* createNode(int);

struct Graph {
  int numVertices;
  struct node** adjLists;
  int* visited;
};

void bfs(struct Graph* graph, int startVertex) {
  struct queue* q = createQueue();

  graph->visited[startVertex] = 1;
  enqueue(q, startVertex);

  while (!isEmpty(q)) {
    printQueue(q);
    int currentVertex = dequeue(q);
    printf("Visited %d\n", currentVertex);

    struct node* temp = graph->adjLists[currentVertex];

    while (temp) {
      int adjVertex = temp->vertex;

      if (graph->visited[adjVertex] == 0) {
        graph->visited[adjVertex] = 1;
        enqueue(q, adjVertex);
```

```
      }
      temp = temp->next;
    }
  }
}

// Creating a node
struct node* createNode(int v) {
  struct node* newNode = malloc(sizeof(struct node));
  newNode->vertex = v;
  newNode->next = NULL;
  return newNode;
}

// Creating a graph
struct Graph* createGraph(int vertices) {
  struct Graph* graph = malloc(sizeof(struct Graph));
  graph->numVertices = vertices;

  graph->adjLists = malloc(vertices * sizeof(struct node*));
  graph->visited = malloc(vertices * sizeof(int));

  int i;
  for (i = 0; i < vertices; i++) {
    graph->adjLists[i] = NULL;
    graph->visited[i] = 0;
  }

  return graph;
}

void addEdge(struct Graph* graph, int src, int dest) {
  struct node* newNode = createNode(dest);
  newNode->next = graph->adjLists[src];
  graph->adjLists[src] = newNode;

  newNode = createNode(src);
  newNode->next = graph->adjLists[dest];
  graph->adjLists[dest] = newNode;
}

struct queue* createQueue() {
```

```c
  struct queue* q = malloc(sizeof(struct queue));
  q->front = -1;
  q->rear = -1;
  return q;
}

int isEmpty(struct queue* q) {
  if (q->rear == -1)
    return 1;
  else
    return 0;
}

void enqueue(struct queue* q, int value) {
  if (q->rear == SIZE - 1)
    printf("\nQueue is Full!!");
  else {
    if (q->front == -1)
      q->front = 0;
    q->rear++;
    q->items[q->rear] = value;
  }
}

int dequeue(struct queue* q) {
  int item;
  if (isEmpty(q)) {
    printf("Queue is empty");
    item = -1;
  } else {
    item = q->items[q->front];
    q->front++;
    if (q->front > q->rear) {
      printf("Resetting queue ");
      q->front = q->rear = -1;
    }
  }
  return item;
}

void printQueue(struct queue* q) {
  int i = q->front;
```

```
    if (isEmpty(q)) {
      printf("Queue is empty");
    } else {
      printf("\nQueue contains \n");
      for (i = q->front; i < q->rear + 1; i++) {
        printf("%d ", q->items[i]);
      }
    }
  }

  int main() {
    struct Graph* graph = createGraph(6);
    addEdge(graph, 0, 1);
    addEdge(graph, 0, 2);
    addEdge(graph, 1, 2);
    addEdge(graph, 1, 4);
    addEdge(graph, 1, 3);
    addEdge(graph, 2, 4);
    addEdge(graph, 3, 4);

    bfs(graph, 0);

    return 0;
  }
```

**Outputs:**




**Conclusion:**

Time complexity for BFS is O(n+e)