EXPERIMENT NO:15

Date:

Aim: Program to find strongly connected components in a directed graph.

Program:

```
#include <stdio.h>

#include <string.h>

#include <stdbool.h>

#define ROW 5

#define COL 5

int i, j, k;

int isSafe(int M[][COL], int row, int col, bool visited[][COL])

{

return (row >= 0) && (row < ROW) &&

(col >= 0) && (col < COL) &&

(M[row][col] && !visited[row][col]);

}

void DFS(int M[][COL], int row, int col, bool visited[][COL])

{

static int rowNbr[] = { -1, -1, -1, 0, 0, 1, 1, 1};

static int colNbr[] = { -1, 0, 1, -1, 1, -1, 0, 1 };

visited[row][col] = true;

for (k = 0; k < 8; ++k)

if (isSafe(M, row + rowNbr[k], col + colNbr[k], visited))

DFS(M, row + rowNbr[k], col + colNbr[k], visited); }
```

```
int countIslands(int M[][COL])

{

bool visited[ROW][COL];

memset(visited, 0, sizeof(visited));

int count = 0;

for (i = 0; i < ROW; ++i) for (j = 0; j < COL; ++j)

if (M[i][j] && !visited[i][j])

{

DFS(M, i, j, visited);

++count;

}

return count;

}

int main()

{

int M[][COL] = { { 1, 1, 0, 0, 0 },

{ 0, 1, 0, 0, 1 },

{ 1, 0, 0, 1, 1 },

{ 0, 0, 0, 0, 0 },

{ 1, 0, 1, 0, 1 }

};

if(countIslands(M)>1)

{

printf("Graph is weakly connected.");
} else

{

printf("Graph is strongly connected.");

}

return 0;

}
```

Result:

The program is executed successfully and output is verified.

EXPERIMENT NO:16

Date:

Aim: Program to perform binary search tree operation.

Program:

```
#include <stdio.h>

#include <stdlib.h>

#include<malloc.h>

// structure of a node

struct node

{int data;

struct node *left;

struct node *right;

};

// globally initialized root pointer

struct node *root = NULL;

// function prototyping

struct node *create_node(int);

void insert(int);

struct node *delete (struct node *, int);

int search(int);

void inorder(struct node *);

void postorder();

void preorder();

struct node *smallest_node(struct node *);
struct node *largest_node(struct node *);
int get_data();
void main()
{int ch;
int data;
struct node* result = NULL;
   printf("\n\n------- Binary Search Tree ------\n");
   printf("\n1. Insert");
   printf("\n2. Delete");
   printf("\n3. Search");
   printf("\n\n-- Traversal --");
   printf("\n\n4. Inorder ");
   printf("\n\n5. Postorder ");
   printf("\n\n6. Preorder ");
   printf("\n7. Exit");
   do
   {printf("\n\nEnter Your Choice: ");
      scanf("%d", &ch);
      printf("\n");
      switch(ch)
      {
         case 1:
            data = get_data();
            insert(data);
```

```
    break;
case 2:
    data = get_data();
    root = delete(root, data);
    break;
 case 3:
    data = get_data();
    if (search(data) == 1)
    {
        printf("\nData was found!\n");
    }
    else
    {
        printf("\nData does not found!\n");
    }
    break;
case 8:
    result = largest_node(root);
    if (result != NULL)
    {
        printf("\nLargest Data: %d\n", result->data);
    }
    break;
```

```
          case 9:
              result = smallest_node(root);
              if (result != NULL)
              {
                  printf("\nSmallest Data: %d\n", result->data);
              }
              break;
           case 4:
              inorder(root);
              break;
           case 5:
              postorder(root);
              break;
          case 6:
              preorder(root);
              break;
           case 7:
              printf("\n\nProgram was terminated\n");
              break;
           default:
              printf("\n\tInvalid Choice\n");
              break;
      }
  }
```

```
    while(ch!=9);
}
// creates a new node
struct node *create_node(int data)
{
    struct node *new_node = (struct node *)malloc(sizeof(struct node));
     if (new_node == NULL)
    {
        printf("\nMemory for new node can't be allocated");
        return NULL;
    }
     new_node->data = data;
    new_node->left = NULL;
    new_node->right = NULL;
    return new_node;
}
// inserts the data in the BST
void insert(int data)
{
    struct node *new_node = create_node(data);
     if (new_node != NULL)
    {
        // if the root is empty then make a new node as the root node
        if (root == NULL)
```

```
{
    root = new_node;
    printf("\n* node having data %d was inserted\n", data);
    return;
}

struct node *temp = root;
struct node *prev = NULL;
// traverse through the BST to get the correct position for insertion
while (temp != NULL)
{
    prev = temp;
    if (data > temp->data)
    {
        temp = temp->right;
    }
    else
    {
        temp = temp->left;
    }
}
// found the last node where the new node should insert
if (data > prev->data)
{
```

```
            prev->right = new_node;
        }
        else
        {
            prev->left = new_node;
        }

        printf("\n* node having data %d was inserted\n", data);
    }
}
// deletes the given key node from the BST
struct node *delete (struct node *root, int key)
{
    if (root == NULL)
    {
        return root;
    }
    if (key < root->data)
    {
        root->left = delete (root->left, key);
    }
    else if (key > root->data)
    {
        root->right = delete (root->right, key);
    }
    else
    {
        if (root->left == NULL)
        {
            struct node *temp = root->right;
            free(root);
            return temp;
        }
        else if (root->right == NULL)
        {
            struct node *temp = root->left;
            free(root);
            return temp;
        }
        struct node *temp = smallest_node(root->right);
        root->data = temp->data;
        root->right = delete (root->right, temp->data);
    }
    return root;
 }
// search the given key node in BST
int search(int key)
{
    struct node *temp = root;
    while (temp != NULL)
    {
        if (key == temp->data)
        {
            return 1;
        }
        else if (key > temp->data)
        {
            temp = temp->right;
        }
        else
        {
            temp = temp->left;
        }
    }
    return 0;
}
// finds the node with the smallest value in BST
struct node *smallest_node(struct node *root)
{
    struct node *curr = root;
    while (curr != NULL && curr->left != NULL)
   {
        curr = curr->left;
    }
    return curr;
}
// finds the node with the largest value in BST
struct node *largest_node(struct node *root)
{
    struct node *curr = root;
    while (curr != NULL && curr->right != NULL)
    {
        curr = curr->right;
    }
    return curr;
}
// inorder traversal of the BST
void inorder(struct node *root)
{
    if (root == NULL)
    {
        return;
    }
    inorder(root->left);
    printf("%d ",  root->data);
    inorder(root->right);
```

```
}
// preorder traversal of the BST
void preorder(struct node *root)
{
    if (root == NULL)
    {
        return;
    }
    printf("%d ", root->data);
    preorder(root->left);
    preorder(root->right);
}
// postorder travsersal of the BST
void postorder(struct node *root)
{
    if (root == NULL)
    {
        return;
    }
    postorder(root->left);
    postorder(root->right);
    printf("%d ", root->data);
}
```

```
// getting data from the user
int get_data()
{
    int data;
    printf("\nEnter Data: ");
    scanf("%d", &data);
    return data;
}
```

Result:

The program is executed successfully and output is verified.

EXPERIMENT NO:17

Date:

Aim: Program to implement bit vector representation.

Program:

```
#include <stdio.h>

void main()

{

int U[5]={1,2,3,4,5},A[5]={1,0,0,1,1},B[5]={0,1,1,1,0},uni[5],ints[5],diffA[5],diffB[5],

i,compA[5],compB[5];

printf("The universal set=");

printf("{");

for(i=0;i<5;i++)

{

printf("%d ",U[i]);

}

printf("}");

printf("\nSet A=");

printf("{");

for(i=0;i<5;i++)

{

if(A[i]==1)

printf("%d ",U[i]);

}

printf("}");
printf("\nSet B");
printf("{");
for(i=0;i<5;i++)
{
if(B[i]==1)
printf("%d ",U[i]);
}
printf("}");
printf("\nUnion of A and B in Bit representation is: ");
for(i=0;i<5;i++)
{
uni[i]=A[i]|B[i];
printf("%d ",uni[i]);
}
printf("\nAUB={");
for(i=0;i<5;i++)
{
if(uni[i]==1)
printf("%d ",U[i]);
}
printf("}");
printf("\nIntersection of A and B in Bit representation is: ");
for(i=0;i<5;i++)
{
```

```
ints[i]=A[i]&B[i];

printf("%d ",ints[i]);

}

printf("\nAnB={");

for(i=0;i<5;i++)

{

if(ints[i]==1)

printf("%d ",U[i]);

}

printf("}");

printf("\nComplement of A in Bit representation is:");

for(i=0;i<5;i++)

{

compA[i]=1-A[i];

printf("%d ",compA[i]);

}

printf("\nA'={");

for(i=0;i<5;i++)

{

if(compA[i]==1)

printf("%d ",U[i]);

}

printf("}");

printf("\nComplement of B in Bit representation is: ");
```

```
for(i=0;i<5;i++)
{
compB[i]=1-B[i];
printf("%d ",compB[i]);
}
printf("\nB'={");
for(i=0;i<5;i++)
{
if(compB[i]==1)
printf("%d ",U[i]);
}
printf("}");
printf("\nDifference of A in Bit representation is:");
for(i=0;i<5;i++)
{
diffA[i]=A[i]&compB[i];
printf("%d ",diffA[i]);
}
printf("\nA-B={");
for(i=0;i<5;i++)
{
if(diffA[i]==1)
printf("%d ",U[i]);
}
printf("}");

printf("\nDifference of B in Bit representation is:");

for(i=0;i<5;i++)

{

diffB[i]=B[i]&compA[i];

printf("%d ",diffB[i]);

}

printf("\nB-A={");

for(i=0;i<5;i++)

{

if(diffB[i]==1)

printf("%d ",U[i]);

}

printf("}");

}
```

Result:

The program is executed successfully and output is verified.