**LAB – 5**

**NAME – TUSHAR MAROO**

**SECTION – A**

**REGISTRATION NUMBER – 190905238**

**ROLL NUMBER – 40**

**Q1.**

**i)**

**ALGO**

Get dfs for a particular node and print it in reverse order. Do this for all nodes. Same visited array should be used when applying dfs to all nodes.

**CODE**

#include <stdio.h>

#include <stdlib.h>

#define MAX 100

struct Stack

{

int top;

unsigned capacity;

int \*array;

};

struct Stack \*createStack(unsigned capacity)

{

struct Stack \*stack = (struct Stack \*)malloc(sizeof(struct Stack));

stack->capacity = capacity;

stack->top = -1;

stack->array = (int \*)malloc(stack->capacity \* sizeof(int));

return stack;

}

int isFull(struct Stack \*stack)

{

return stack->top == stack->capacity - 1;

}

int isEmpty(struct Stack \*stack)

{

return stack->top == -1;

}

void push(struct Stack \*stack, int item)

{

if (isFull(stack))

return;

stack->array[++stack->top] = item;

// printf("%d pushed to stack\n", item);

}

int pop(struct Stack \*stack)

{

if (isEmpty(stack))

return INT\_MIN;

return stack->array[stack->top--];

}

int peek(struct Stack \*stack)

{

if (isEmpty(stack))

return INT\_MIN;

return stack->array[stack->top];

}

typedef struct listNode ListNode;

struct listNode

{

ListNode \*next;

int data;

};

ListNode \*createNode(int x)

{

ListNode \*temp = (ListNode \*)malloc(sizeof(ListNode));

temp->data = x;

temp->next = NULL;

return temp;

}

ListNode \*insert(ListNode \*head, int x)

{

if (head == NULL)

{

head = createNode(x);

return head;

}

ListNode \*temp = head;

while (temp->next != NULL)

temp = temp->next;

temp->next = createNode(x);

return head;

}

void print(ListNode \*head)

{

ListNode \*temp = head;

while (temp != NULL)

{

printf("%d --> ", temp->data);

temp = temp->next;

}

printf("NULL\n");

}

void dfs(ListNode \*\*adjList, int visited[],struct Stack \*DFS, int v)

{

int idx = 0;

struct Stack \*stack = createStack(100);

push(stack, v);

while (!isEmpty(stack))

{

v = pop(stack);

if (visited[v])

{

continue;

}

visited[v] = 1;

push(DFS,v);

ListNode \*temp = adjList[v];

while (temp != NULL)

{

if (!visited[temp->data])

push(stack, temp->data);

temp = temp->next;

}

}

}

void TSusingDFS(ListNode \*\*adjList, int visited[], int v)

{

int i;

struct Stack \*DFS = createStack(v);

struct Stack \*TS = createStack(v);

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

{

dfs(adjList, visited, DFS, i);

printf("\nDFS : ");

while(!isEmpty(DFS))

{

int v = pop(DFS);

printf("%d ",v);

push(TS,v);

}

}

printf("\nTopological sort using DFS : ");

while(!isEmpty(TS))

{

printf("%d ",pop(TS));

}

}

int main()

{

int v, e, i, j;

printf("Enter the number of vertices : ");

scanf("%d", &v);

ListNode \*\*adjList = malloc(v \* sizeof(ListNode \*));

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

adjList[i] = NULL;

printf("Enter the number of edges : ");

scanf("%d", &e);

int start, end, visited[v];

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

visited[i] = 0;

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

{

printf("Enter start and end vertex of edge %d : ", i + 1);

scanf("%d%d", &start, &end);

adjList[start] = insert(adjList[start], end);

}

printf("\nAdjacency List : \n");

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

{

printf("%d : ", i);

print(adjList[i]);

}

printf("\n");

TSusingDFS(adjList,visited,v);

}

**OUTPUT**

****

**ii)**

**ALGO**

Get indegree for all nodes. Print nodes with indegree 0 and also store these in a queue. Pop queue and do the same after removing the popped element from graph and calculating new indegree till this queue is empty.

**CODE**

#include <stdio.h>

#include <stdlib.h>

#define MAX 100

struct Queue

{

int front, rear, size;

unsigned capacity;

int \*array;

};

struct Queue \*createQueue(unsigned capacity)

{

struct Queue \*queue = (struct Queue \*)malloc(

sizeof(struct Queue));

queue->capacity = capacity;

queue->front = queue->size = 0;

queue->rear = capacity - 1;

queue->array = (int \*)malloc(

queue->capacity \* sizeof(int));

return queue;

}

int isFull(struct Queue \*queue)

{

return (queue->size == queue->capacity);

}

int isEmpty(struct Queue \*queue)

{

return (queue->size == 0);

}

void enqueue(struct Queue \*queue, int item)

{

if (isFull(queue))

return;

queue->rear = (queue->rear + 1) % queue->capacity;

queue->array[queue->rear] = item;

queue->size = queue->size + 1;

// printf("%d enqueued to queue\n", item);

}

int dequeue(struct Queue \*queue)

{

if (isEmpty(queue))

return INT\_MIN;

int item = queue->array[queue->front];

queue->front = (queue->front + 1) % queue->capacity;

queue->size = queue->size - 1;

return item;

}

int front(struct Queue \*queue)

{

if (isEmpty(queue))

return INT\_MIN;

return queue->array[queue->front];

}

int rear(struct Queue \*queue)

{

if (isEmpty(queue))

return INT\_MIN;

return queue->array[queue->rear];

}

typedef struct listNode ListNode;

struct listNode

{

ListNode \*next;

int data;

};

ListNode \*createNode(int x)

{

ListNode \*temp = (ListNode \*)malloc(sizeof(ListNode));

temp->data = x;

temp->next = NULL;

return temp;

}

ListNode \*insert(ListNode \*head, int x)

{

if (head == NULL)

{

head = createNode(x);

return head;

}

ListNode \*temp = head;

while (temp->next != NULL)

temp = temp->next;

temp->next = createNode(x);

return head;

}

void print(ListNode \*head)

{

ListNode \*temp = head;

while (temp != NULL)

{

printf("%d --> ", temp->data);

temp = temp->next;

}

printf("NULL\n");

}

void TSUsingSourceRemoval(ListNode \*\*adjList, int v)

{

int indegree[v], i;

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

indegree[i] = 0;

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

{

ListNode \*temp = adjList[i];

while (temp != NULL)

{

indegree[temp->data]++;

temp = temp->next;

}

free(temp);

}

struct Queue \*queue = createQueue(1000);

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

if (indegree[i] == 0)

enqueue(queue, i);

int TS[v], idx = 0, cnt = 0;

while (!isEmpty(queue))

{

int u = dequeue(queue);

TS[idx++] = u;

ListNode \*temp = adjList[u];

while (temp != NULL)

{

if (--indegree[temp->data] == 0)

enqueue(queue, temp->data);

temp = temp->next;

}

cnt++;

}

if (cnt != v)

printf("There is a cycle in the graph.");

else

{

printf("Topological sort using source removal technique : ");

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

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

}

}

int main()

{

int v, e, i, j;

printf("Enter the number of vertices : ");

scanf("%d", &v);

ListNode \*\*adjList = malloc(v \* sizeof(ListNode \*));

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

adjList[i] = NULL;

printf("Enter the number of edges : ");

scanf("%d", &e);

int start, end, visited[v];

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

visited[i] = 0;

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

{

printf("Enter start and end vertex of edge %d : ", i + 1);

scanf("%d%d", &start, &end);

adjList[start] = insert(adjList[start], end);

}

printf("\nAdjacency List : \n");

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

{

printf("%d : ", i);

print(adjList[i]);

}

printf("\n");

TSUsingSourceRemoval(adjList, v);

}

**OUTPUT**

****

**Q2.**

**ALGO**

The diameter of the tree would be the maximum of:

1. 1 + height of left subtree + height of right subtree
2. Diameter of right subtree
3. Diameter of left subtree

Call diameter function recursively to get the diameter of tree.

**CODE**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

struct Node

{

int val;

struct Node \*left, \*right;

};

struct Node \*newNode(int value)

{

struct Node \*node = (struct Node \*)malloc(sizeof(struct Node));

node->val = value;

node->left = NULL;

node->right = NULL;

return (node);

}

int max(int a, int b)

{

return (a > b) ? a : b;

}

int height(struct Node \*node)

{

if (node == NULL)

return 0;

return 1 + max(height(node->left), height(node->right));

}

int diameter(struct Node \*tree)

{

if (tree == NULL)

return 0;

int Lheight = height(tree->left);

int Rheight = height(tree->right);

int Ldiam = diameter(tree->left);

int Rdiam = diameter(tree->right);

return max(Lheight + Rheight + 1, max(Ldiam, Rdiam));

}

int main()

{

struct Node \*root = newNode(1);

int t, i, error = 0;

printf("Enter the the number of nodes except root node : ");

scanf("%d", &t);

while (t--)

{

struct Node \*temp = root;

char str[10];

printf("Enter the location of node : ");

scanf("%s", str);

for (i = 0; i < strlen(str) - 1; i++)

{

if (str[i] == 'L' || str[i] == 'l')

{

temp = temp->left;

}

else if (str[i] == 'R' || str[i] == 'r')

{

temp = temp->right;

}

else

{

printf("Invalid String\n");

break;

}

if (temp == NULL)

{

printf("Invalid String\n");

error = 1;

break;

}

}

if (error == 1)

{

error = 0;

continue;

}

if (str[i] == 'L' || str[i] == 'l')

{

temp->left = newNode(1);

}

else if (str[i] == 'R' || str[i] == 'r')

{

temp->right = newNode(1);

}

else

{

printf("Invalid String\n");

continue;

}

printf("Node Added\n");

}

printf("Diameter of the given binary tree is %d\n", diameter(root));

return 0;

}

**OUTPUT**

