Lab 5 : Decrease and Conquer : Dipesh Singh 190905520

Question 1 : Topological Sorting using DFS

```

#include <stdio.h>

#include <stdlib.h>

int isEmpty(int top)

{

    if (top == -1)

    {

        return 1;

    }

    return 0;

}

int isFull(int top, int capacity)

{

    if (top == capacity - 1)

    {

        return 1;

    }

    return 0;

}

void push(int \*\*Stack, int \*top, int \*capacity, int key)

{

    if (isFull(\*top, \*capacity))

    {

        \*Stack = (int \*)realloc(\*Stack, sizeof(int) \* (\*capacity) \* 2);

        \*capacity \*= 2;

    }

    (\*top)++;

    (\*Stack)[\*top] = key;

}

int pop(int \*\*Stack, int \*top)

{

    int temp = (\*Stack)[\*top];

    (\*top)--;

    return temp;

}

void display(int \*Stack, int top)

{

    if (isEmpty(top))

    {

    }

    else

    {

        printf("stack : ");

        int i;

        for (i = 0; i <= top; i++)

        {

            printf("%d ", \*(Stack + i));

        }

        printf("\n");

    }

}

void insertEdgeM(int \*\*matrix, int first, int second)

{

    matrix[first][second] = 1;

}

void displayMatrix(int \*\*matrix, int n)

{

    for (int i = -1; i < n; i++)

    {

        if (i != -1)

            printf("%d -> ", i);

        for (int j = 0; j < n; j++)

        {

            if (i == -1)

            {

                if (j == 0)

                {

                    printf("\t");

                }

                printf("%d\t", j);

                continue;

            }

            if (j == 0)

            {

                printf("\t");

            }

            printf("%d\t", matrix[i][j]);

        }

        printf("\n");

    }

}

void dfs(int \*\*matrix, int num, int \*\*Stack, int \*top, int \*capacity)

{

    int \*visited = (int \*)calloc(num, sizeof(int));

    for (int i = 0; i < num; i++)

    {

        visited[i] = 0;

    }

    char result[100];

    int resultIndex = 0;

    char popped[100];

    int poppedIndex = 0;

    push(Stack, top, capacity, 0);

    printf("pushed : %d\n", 0);

    char p = (char)('0' + 0);

    result[resultIndex++] = p;

    display(\*Stack, \*top);

    visited[0] = 1;

    int cur = \*Stack[\*top];

    int flag, ele;

    while (1)

    {

        if (!(isEmpty(\*top)))

        {

            flag = 0;

            for (int i = 0; i < num; i++)

            {

                if (visited[i] == 0 && matrix[cur][i] == 1)

                {

                    visited[i] = 1;

                    printf("pushed : %d\n", i);

                    p = (char)('0' + i);

                    result[resultIndex++] = p;

                    push(Stack, top, capacity, i);

                    display(\*Stack, \*top);

                    flag = 1;

                    break;

                }

            }

            if (flag == 0)

            {

                ele = pop(Stack, top);

                p = (char)('0' + ele);

                popped[poppedIndex++] = p;

                printf("popped : %d\n", ele);

                display(\*Stack, \*top);

            }

            cur = (\*Stack)[\*top];

        }

        else

        {

            flag = 1;

            for (int i = 0; i < num && flag; i++)

            {

                if (visited[i] == 0)

                {

                    visited[i] = 1;

                    printf("pushed : %d\n", i);

                    p = (char)('0' + i);

                    result[resultIndex++] = p;

                    push(Stack, top, capacity, i);

                    display(\*Stack, \*top);

                    flag = 0;

                    cur = (\*Stack)[\*top];

                    break;

                }

            }

            if (flag == 1)

            {

                break;

            }

        }

    }

    while (!(isEmpty(\*top)))

    {

        int rem = pop(Stack, top);

        p = (char)('0' + rem);

        popped[poppedIndex++] = p;

        printf("popped : %d\n", rem);

        display(\*Stack, \*top);

    }

    printf("The dfs is : ");

    for (int i = 0; i < resultIndex; i++)

    {

        printf("%c ", result[i]);

    }

    printf("\nThe traversal stack is : ");

    for (int i = 0; i < poppedIndex; i++)

    {

        printf("%c ", popped[i]);

    }

    printf("\n");

    for (int i = 0; i < poppedIndex - 1; i++)

    {

        for (int j = i + 1; j < poppedIndex; j++)

        {

            if (matrix[((int)(popped[i] - '0'))][((int)(popped[j] - '0'))] == 1)

            {

                printf("The graph is not a Directed Acyclic Graph, topological sort not possible.\n");

                return;

            }

        }

    }

    printf("The graph is a Directed Acyclic Graph, topoplogical sort is possible : ");

    for (int i = poppedIndex - 1; i >= 0; i--)

    {

        printf("%c ", popped[i]);

    }

    printf("\n");

}

int main()

{

    int num = 5;

    int \*\*matrix = (int \*\*)calloc(num, sizeof(int \*));

    for (int i = 0; i < num; i++)

    {

        matrix[i] = (int \*)calloc(num, sizeof(int));

        for (int j = 0; j < num; j++)

        {

            matrix[i][j] = 0;

        }

    }

    insertEdgeM(matrix, 0, 2);

    insertEdgeM(matrix, 1, 2);

    insertEdgeM(matrix, 2, 3);

    insertEdgeM(matrix, 2, 4);

    insertEdgeM(matrix, 3, 4);

    displayMatrix(matrix, num);

    int top = -1, capacity = 2;

    int \*Stack = (int \*)calloc(capacity, sizeof(int));

    dfs(matrix, num, &Stack, &top, &capacity);

    return 0;

}

```

Diagram

Description automatically generated

Text

Description automatically generated

A picture containing diagram

Description automatically generated

Text

Description automatically generated

The time complexity of DFS using adjacency matrix is O(V2) hence the time complexity of Topological sorting using DFS is O(V2).

Question 2 : Topological Sorting using Source Removal Technique

```

#include <stdio.h>

#include <stdlib.h>

void insertEdgeM(int \*\*matrix, int first, int second)

{

    matrix[first][second] = 1;

}

void displayMatrix(int \*\*matrix, int n)

{

    for (int i = -1; i < n; i++)

    {

        if (i != -1)

            printf("%d -> ", i);

        for (int j = 0; j < n; j++)

        {

            if (i == -1)

            {

                if (j == 0)

                {

                    printf("\t");

                }

                printf("%d\t", j);

                continue;

            }

            if (j == 0)

            {

                printf("\t");

            }

            printf("%d\t", matrix[i][j]);

        }

        printf("\n");

    }

}

int check(int \*\*matrix, int num, int node)

{

    int result = 1;

    for (int i = 0; i < num; i++)

    {

        if (matrix[i][node] == 1)

        {

            result = 0;

        }

    }

    return result;

}

void deleteEdges(int \*\*matrix, int num, int node)

{

    for (int i = 0; i < num; i++)

    {

        matrix[node][i] = 0;

    }

}

void topo(int \*\*matrix, int num)

{

    int \*removed = (int \*)calloc(num, sizeof(int));

    for (int i = 0; i < num; i++)

    {

        removed[i] = 0;

    }

    int popped[num];

    int poppedIndex = 0;

    for (int i = 0; i < num; i++)

    {

        if (removed[i] == 0 && check(matrix, num, i))

        {

            removed[i] = 1;

            popped[poppedIndex++] = i;

            deleteEdges(matrix, num, i);

            i = -1;

        }

    }

    for (int i = 0; i < num; i++)

    {

        if (removed[i] == 0)

        {

            printf("The graph is not a Directed Acyclic Graph, topological sort not possible.\n");

            return;

        }

    }

    printf("The graph is a Directed Acyclic Graph, topoplogical sort is possible : ");

    for (int i = 0; i < poppedIndex; i++)

    {

        printf("%d ", popped[i]);

    }

    printf("\n");

}

int main()

{

    int num = 5;

    int \*\*matrix = (int \*\*)calloc(num, sizeof(int \*));

    for (int i = 0; i < num; i++)

    {

        matrix[i] = (int \*)calloc(num, sizeof(int));

        for (int j = 0; j < num; j++)

        {

            matrix[i][j] = 0;

        }

    }

    insertEdgeM(matrix, 0, 2);

    insertEdgeM(matrix, 1, 2);

    insertEdgeM(matrix, 2, 3);

    insertEdgeM(matrix, 2, 4);

    insertEdgeM(matrix, 3, 4);

    displayMatrix(matrix, num);

    topo(matrix, num);

    return 0;

}

```

Diagram

Description automatically generated

Graphical user interface, text

Description automatically generated

A picture containing diagram

Description automatically generated

Text

Description automatically generated

Since for every vertex V, when deletion operation is underwent, we go through V vertices to remove the edge from the source to the destination, the time complexity of this algorithm is O(V2).

Question 3 : Find the diameter of a tree

```

#include <stdio.h>

#include <stdlib.h>

#include <limits.h>

typedef struct node

{

    int val;

    struct node \*left;

    struct node \*right;

} \* Node;

void inorder(Node n)

{

    if (n)

    {

        inorder(n->left);

        printf("%d ", n->val);

        inorder(n->right);

    }

}

Node insert()

{

    int val;

    int check;

    printf("Enter the element : ");

    scanf("%d", &val);

    Node n = (Node)malloc(sizeof(struct node));

    n->val = val;

    n->left = NULL;

    n->right = NULL;

    printf("Do you want to insert left child of %d? (1 for Yes, 0 for No) : ", val);

    scanf("%d", &check);

    if (check)

        n->left = insert();

    printf("Do you want to insert right child of %d? (1 for Yes, 0 for No) : ", val);

    scanf("%d", &check);

    if (check)

        n->right = insert();

    return n;

}

int max(int a, int b)

{

    return a > b ? a : b;

}

int height(Node head)

{

    if (head == NULL)

    {

        return 0;

    }

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

}

void diameter(Node cur, int \*max)

{

    if (cur)

    {

        int curDia = height(cur->left) + height(cur->right) + 1;

        if (curDia > \*max)

        {

            \*max = curDia;

        }

        diameter(cur->left, max);

        diameter(cur->right, max);

    }

}

int main()

{

    Node head = insert();

    printf("The inorder is : ");

    inorder(head);

    int diam = INT\_MIN;

    diameter(head, &diam);

    printf("\nThe diameter of the binary tree is : %d\n", diam);

    return 0;

}

```

Diagram

Description automatically generated

Text

Description automatically generated

Diagram

Description automatically generated

Text

Description automatically generatedA screenshot of a computer

Description automatically generated with medium confidence