# 20MCA 135 DATASTRUCTURES LAB

# EXAMINATION

SUBMITTED BY :

SNEHA ROY

TKM20MCA-2039

MCA137

Git link - https://github.com/26sneharoy/data-structures/upload/main

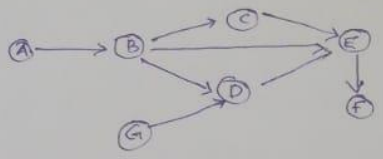
Git name- 26sneharoy

Q1)consider a directed acyclic graph as given. Develop a program to implement the topological sorting

# 20CMCA 135 DATA STRUCTURES LAB

SNEHA ROY.
TKM20MCA-2039.
MCA 1st.

Q1) Consider a directed acyclic graph. G1 given in the following figure.




Develop a program to implement topological sorting.

Algorithm –

step-1 . enter the number of vertices.

step-2 find the adjacency matrix of the given graph.

| | A | B | C | D | E | F | G |
|---|---|---|---|---|---|---|---|
| A | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| B | 0 | 0 | 1 | 1 | 1 | 0 | 0 |
| C | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| D | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| E | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| F | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| G | 0 | 0 | 0 | 1 | 0 | 0 | 0 |

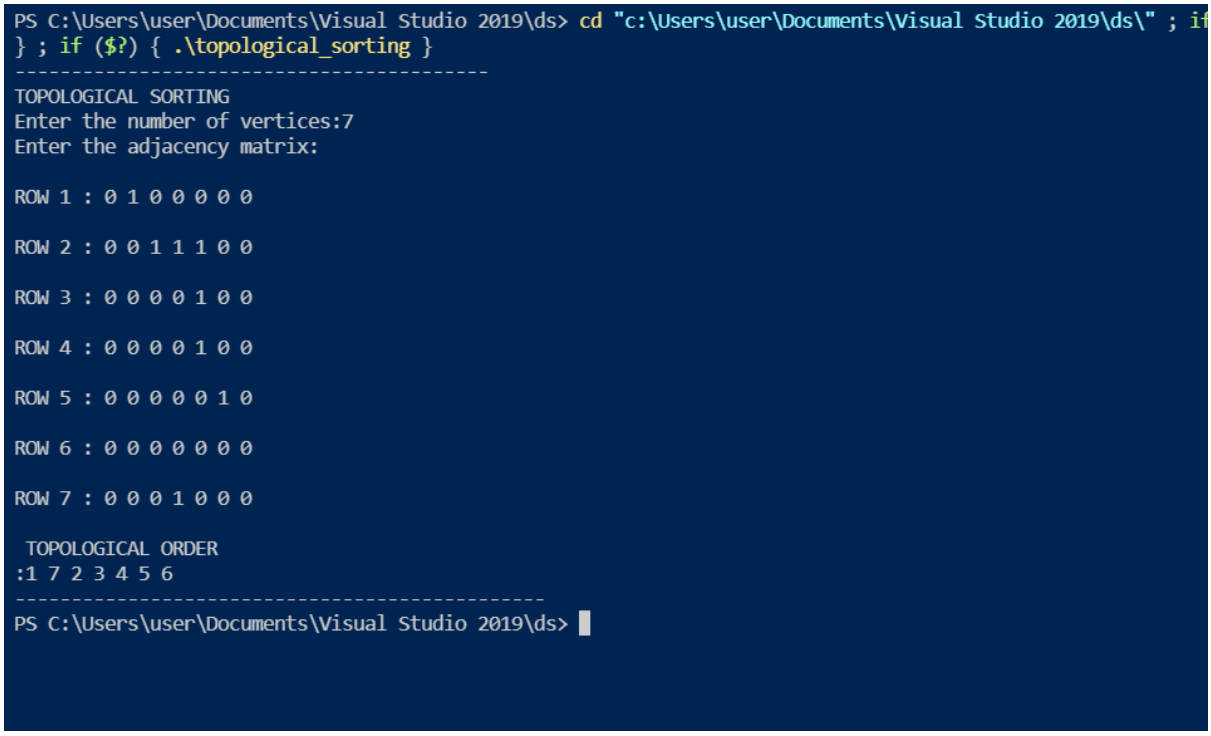
Step-3 . find the indegree of the vertices and select the vertex with degree 0 . & decrease the gree count of adjacent vertex . add the vertex to the solution . Repeat the process . for the last vertex .

PROGRAM :

```
#include <stdio.h>

int main()
{
    int i, j, k, n;
    int adjacency[10][10], indegree[10], flag[10];
    int count = 0;
    printf("-----------------------------------------\n");
    printf("TOPOLOGICAL SORTING\n");

    printf("Enter the number of vertices:");
    scanf("%d", &n);

    printf("Enter the adjacency matrix:\n");
    for (i = 0; i < n; i++)
    {
        printf("\nROW %d : ", i + 1);
        for (j = 0; j < n; j++)
            scanf("%d", &adjacency[i][j]);
    }

    for (i = 0; i < n; i++)
    {
        indegree[i] = 0;
        flag[i] = 0;
    }

    for (i = 0; i < n; i++)
        for (j = 0; j < n; j++)
```

```
            indegree[i] = indegree[i] + adjacency[j][i];

    printf("\n TOPOLOGICAL ORDER \n:");

    while (count < n)
    {
        for (k = 0; k < n; k++)
        {
            if ((indegree[k] == 0) && (flag[k] == 0))
            {
                printf("%d ", (k + 1));
                flag[k] = 1;
            }

            for (i = 0; i < n; i++)
            {
                if (adjacency[i][k] == 1)
                    indegree[k]--;
            }
        }

        count++;
    }
    printf("\n----------------------------------------------");
    return 0;
}
```

OUTPUT


PS C:\Users\user\Documents\Visual Studio 2019\ds> cd "c:\Users\user\Documents\Visual Studio 2019\ds\" ; if
} ; if ($?) { .\topological_sorting }
TOPOLOGICAL SORTING
Enter the number of vertices:7
Enter the adjacency matrix:
ROW 1 : 0 1 0 0 0 0 0
ROW 2 : 0 0 1 1 1 0 0
ROW 3 : 0 0 0 0 1 0 0
ROW 4 : 0 0 0 0 1 0 0
ROW 5 : 0 0 0 0 0 1 0
ROW 6 : 0 0 0 0 0 0 0
ROW 7 : 0 0 0 1 0 0 0
TOPOLOGICAL ORDER
:1 7 2 3 4 5 6
PS C:\Users\user\Documents\Visual Studio 2019\ds>

Q2) write a program for doubly linked list and perform the following operations

a) Insert an element at a particular position
b) Search an element
c) Delete an element at the end of the list

2) Write a program for doubly linked list and perform the following operations.

(a) Insert an element at a particular position.

(b) Search an element.

(c) Delete ~~the~~ an element at the end of the list.

ALGORITHM :-

Step-1 - Create a function header to Insert an element in the begining :

Step-2 - Create a function to insert an element in a particular position.

Step-3 Create a function to search a particular element and to display its position.

Step-4. Create a function to delete an element from the end of the list.

Step 5 - Create another function to display all the elements in the list.

Step 6 - In the main function Give the provide a option to select the choice among these function.

Step 7 - Provide a exit choice to exit from the operation.

PROGRAM-

```
#include <stdio.h>
#include <stdlib.h>
struct node
{
   struct node *prev;
   struct node *next;
   int data;
};
struct node *head;
void insertion_beginning();
void insertion_specified();
void deletion_last();
void search();
void display();
int main()
{
   printf("--------------------------------------\n");
   printf("DOUBLY LINKED LIST\n");
   int choice = 0;
   while (choice != 6)
   {
      printf("1.Insert in begining\n");
      printf("2.Insert at any particular position\n");
      printf("3.Deletion\n");
      printf("4.Search\n");
      printf("5.Show\n");
      printf("6.Exit\n");

      printf("\nEnter your choice?\n");
      scanf("\n%d", &choice);
      switch (choice)
      {
      case 1:
         printf("you have selected option 1 for the insertion at the begining \n ");
         insertion_beginning();

         break;
      case 2:
         printf("you have selected option 2 for the insertion in a particular position \n");
         insertion_specified();
         break;
      case 3:
         printf("you have selected option 3 for the deletion\n");
         deletion_last();
         break;
```

```
        case 4:
            printf("you have selected option 4 for searching \n");
            search();
            break;
        case 5:
            printf("you have selected option 5 for displaying \n");
            display();
            break;
        case 6:
            printf("----------------------------------------------");
            exit(0);
            break;
        default:
            printf("Please enter valid choice..");
        }
    }
}
void insertion_beginning()
{
    struct node *ptr;
    int item;
    ptr = (struct node *)malloc(sizeof(struct node));
    if (ptr == NULL)
    {
        printf("\nOVERFLOW");
    }
    else
    {
        printf("\nEnter Item value  : ");
        scanf("%d", &item);

        if (head == NULL)
        {
            ptr->next = NULL;
            ptr->prev = NULL;
            ptr->data = item;
            head = ptr;
        }
        else
        {
            ptr->data = item;
            ptr->prev = NULL;
            ptr->next = head;
            head->prev = ptr;
            head = ptr;
        }
```

```
        printf("\n----Node inserted-----\n");
    }
}

void insertion_specified()
{
    struct node *ptr, *temp;
    int item, loc, i;
    ptr = (struct node *)malloc(sizeof(struct node));
    if (ptr == NULL)
    {
        printf("\n OVERFLOW");
    }
    else
    {
        temp = head;
        printf("Enter the location : ");
        scanf("%d", &loc);
        for (i = 0; i < loc; i++)
        {
            temp = temp->next;
            if (temp == NULL)
            {
                printf("\n There are less than %d elements", loc);
                return;
            }
        }
        printf("Enter value : ");
        scanf("%d", &item);
        ptr->data = item;
        ptr->next = temp->next;
        ptr->prev = temp;
        temp->next = ptr;
        temp->next->prev = ptr;
        printf("\n-------node inserted---------\n");
    }
}
void deletion_last()
{
    struct node *ptr;
    if (head == NULL)
    {
        printf("\n UNDERFLOW");
    }
    else if (head->next == NULL)
    {
```

```
        head = NULL;
        free(head);
        printf("\n---node deleted-----\n");
    }
    else
    {
        ptr = head;
        head = head->next;
        head->prev = NULL;
        free(ptr);
        printf("\n-----node deleted------\n");
    }
}
void display()
{
    struct node *ptr;
    printf("\n --DISPLAYING --\n");
    ptr = head;
    while (ptr != NULL)
    {
        printf("%d\n", ptr->data);
        ptr = ptr->next;
    }
}
void search()
{
    struct node *ptr;
    int item, i = 0, flag;
    ptr = head;
    if (ptr == NULL)
    {
        printf("\n ---EMPTY LIST---\n");
    }
    else
    {
        printf("\n Item to be searched : \n");
        scanf("%d", &item);
        while (ptr != NULL)
        {
            if (ptr->data == item)
            {
                printf("\nitem found at location %d\n", i + 1);
                flag = 0;
                break;
            }
            else
```

```
            {
                flag = 1;
            }
            i++;
            ptr = ptr->next;
        }
        if (flag == 1)
        {
            printf("\n----Item not found---\n");
        }
    }
}
```

OUTPUT-

```
PS C:\Users\user\Documents\Visual Studio 2019\ds> cd "c:\Users\user\Documents\Vis
if ($?) { .\doubly_linkedlist }
-----------------------------------
DOUBLY LINKED LIST
1.Insert in begining
2.Insert at any particular position
3.Deletion
4.Search
5.Show
6.Exit

Enter your choice?
1
you have selected option 1 for the insertion at the begining

Enter Item value  : 5

----Node inserted-----
1.Insert in begining
2.Insert at any particular position
3.Deletion
4.Search
5.Show
6.Exit

Enter your choice?
1
you have selected option 1 for the insertion at the begining

Enter Item value  : 6

----Node inserted-----
1.Insert in begining
2.Insert at any particular position
3.Deletion
4.Search
5.Show
6.Exit
```

```
Enter your choice?
2
you have selected option 2 for the insertion in a particular position
Enter the location : 1
Enter value : 8

-------node inserted---------
1.Insert in begining
2.Insert at any particular position
3.Deletion
4.Search
5.Show
6.Exit

Enter your choice?
3
you have selected option 3 for the deletion

-----node deleted------
1.Insert in begining
2.Insert at any particular position
3.Deletion
4.Search
5.Show
6.Exit

Enter your choice?
5
you have selected option 5 for displaying

 --DISPLAYING --
5
8
1.Insert in begining
2.Insert at any particular position
3.Deletion
4.Search
5.Show
6.Exit

Enter your choice?
6
------------------------------------------------
PS C:\Users\user\Documents\Visual Studio 2019\ds>
```