INDEX

|  |  |  |
| --- | --- | --- |
| Sl.No | Experiment Title | Page No |
| 1.1 | Implementation of BFS in a digraph | 4-6 |
| 1.2 | Implementation of DFS in a digraph | 7-9 |
| 3 | Implementation of topological sort in a digraph | 10-12 |
| 4 | Implementation of Jhonson-Trotter algorithm | 13-15 |
| 5 |  | 16-20 |
| 6 | Delete operations in singly linked list | 21-26 |
| 7.1 | Sorting a singly linked list | 27-29 |
| 7.2 | Reversing a singly linked list | 30-32 |
| 7.3 | Concatenation of two singly linked lists | 32-35 |
| 8 | Implementation of stacks and queues using singly linked list | 36-41 |
| 9 | Implementation of doubly linked list | 42-47 |
| 10 | Implementation of binary search tree | 48-52 |

**PROGRAM 1**

Write program to do the following:

a. Print all the nodes reachable from a given starting node in a digraph using BFS method.

b. Print the DFS traversal of a given graph.

**1.1**

**(a) BFS Traversal**

**CODE**

#include<stdio.h>

#include<conio.h>

void insert\_rear(int q[],int \*r, int item, int size)

{

if(\*r==size)

printf("Queue overflow!\n");

else

{

\*r=\*r+1;

q[\*r]=item;

}

}

int delete\_front(int q[],int \*r, int \*f)

{

int del\_item=-1;

\*f=\*f+1;

del\_item=q[\*f];

return del\_item;

}

int isEmpty(int q[], int \*r, int \*f)

{

if(\*r==-1 || \*r==\*f)

return 1;

else

return 0;

}

void main()

{

int n,i,j,r=-1,f=-1;

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

scanf("%d",&n);

printf("Enter the adjacency matrix representing the graph:\n");

int graph[n][n];

int vis[n],q[n];

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

{

for(int j=0;j<n;j++)

{

scanf("%d",&graph[i][j]);

}

}

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

{

vis[i]=0;

}

printf(“The BFS travsersal is:\n”);

int k=0;

printf("%d ",k); // print the first node

vis[k]=1; //Make the first node visited

insert\_rear(q,&r,k,n); // Insert the node in the queue

while(isEmpty(q,&r,&f)==0) // if queue is not empty

{

int node=delete\_front(q,&r,&f); //remove node from queue

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

{

if(graph[node][j]==1 && vis[j]==0) /\*if the child of node removed exits and is not visited, make it visited.

1.print the child

2.make the node visited.

3.insert the child into the queue\*/

{

printf("%d ",j);

vis[j]=1;

insert\_rear(q,&r,j,n);

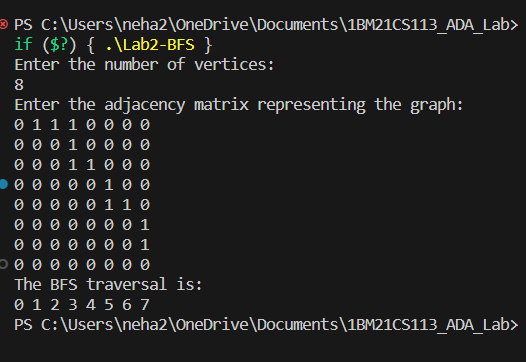
}

}

}

}

**OUTPUT**

****

**1.2**

**(b) DFS Traversal**

**CODE**

#include<stdio.h>

int graph[20][20];

void DFS(int i,int vis[],int n)

{

int j;

printf("%d ",i); // print the source node

vis[i]=1; // make the source node visited

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

{

if(graph[i][j]==1 && vis[j]==0) // for every adjacent vertex that is not visited

{

DFS(j,vis,n); // recursive call to DFS- because we need to print the nodes depth wise

}

}

}

void main()

{

int n,i,j,top=-1;

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

scanf("%d",&n);

printf("Enter the adjacency matrix representing the graph:\n");

int vis[n],st[n];

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

{

for(int j=0;j<n;j++)

{

scanf("%d",&graph[i][j]);

}

}

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

{

vis[i]=0;

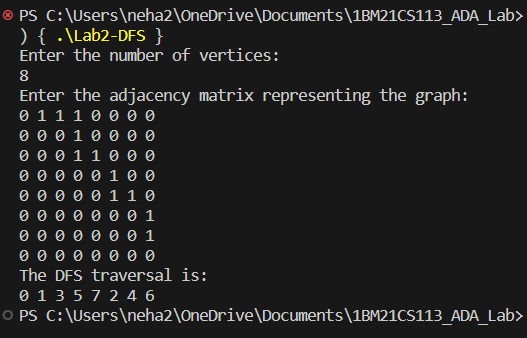
}

printf("The DFS traversal is:\n");

DFS(0,vis,n);

}

**OUTPUT**

****

**PROGRAM 2**

Write program to obtain the Topological ordering of vertices in a given digraph.

**CODE**

#include<stdio.h>

#include<stdlib.h>

int s[100], j, res[100];

void AdjacencyMatrix(int a[][100], int n)

{

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

{

for(int j=0;j<n;j++)

{

scanf("%d",&a[i][j]);

}

}

return;

}

void dfs(int u, int n, int a[][100])

{

int v;

s[u] = 1;

for (v = 0; v < n ; v++) {

if (a[u][v] == 1 && s[v] == 0) {

dfs(v, n, a);

}

}

j += 1;

res[j] = u; // Store every dead node in the array

}

void topological\_order(int n, int a[][100])

{

int i, u;

for (i = 0; i < n; i++) {

s[i] = 0;

}

j = 0;

for (u = 0; u < n; u++) {

if (s[u] == 0) {

dfs(u, n, a);

}

}

}

void main() {

int a[100][100], n, i, j;

printf("Enter number of vertices:\n"); /\* READ NUMBER OF VERTICES \*/

scanf("%d", &n);

printf("Enter the adjacency matrix:\n");

AdjacencyMatrix(a, n);

topological\_order(n, a);

printf("The topological sort order is:\n");

for (i = n; i >=1; i--) /\*Inside the array 'res', we are adding the nodes that become dead from first to last.

But topological sort is the reverse order.So we are printing the array backwards.\*/

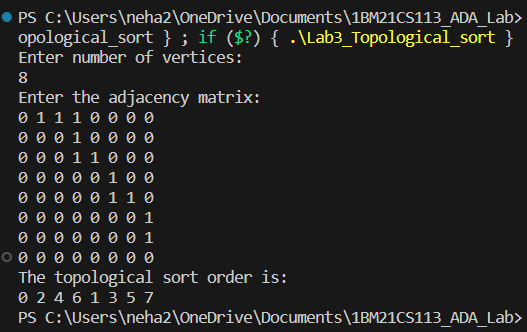
{

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

}

}

**OUTPUT**

****

**PROGRAM 3**

Implement Johnson Trotter algorithm to generate permutations.

**CODE**

#include <stdio.h>

#define RIGHT\_TO\_LEFT 0

#define LEFT\_TO\_RIGHT 1

void swap(int \*a, int \*b)

{

int temp = \*a;

\*a = \*b;

\*b = temp;

}

int searchArr(int a[], int n, int mobile)

{

for (int i = 0; i < n; i++) {

if (a[i] == mobile) {

return i + 1;

}

}

return -1; // Mobile not found

}

int getMobile(int a[], int dir[], int n)

{

int mobile\_prev = 0, mobile = 0;

for (int i = 0; i < n; i++) {

// Direction 0 represents RIGHT TO LEFT.

if (dir[a[i] - 1] == RIGHT\_TO\_LEFT && i != 0)

{

if (a[i] > a[i - 1] && a[i] > mobile\_prev)

{

mobile = a[i];

mobile\_prev = mobile;

}

}

if (dir[a[i] - 1] == LEFT\_TO\_RIGHT && i != n - 1)

{

if (a[i] > a[i + 1] && a[i] > mobile\_prev)

{

mobile = a[i];

mobile\_prev = mobile;

}

}

}

if (mobile == 0 && mobile\_prev == 0)

{

return 0; // No mobile element found

} else {

return mobile;

}

}

void printOnePerm(int a[], int dir[], int n)

{

int mobile = getMobile(a, dir, n);

int pos = searchArr(a, n, mobile);

if (dir[a[pos - 1] - 1] == RIGHT\_TO\_LEFT)

{

swap(&a[pos - 1], &a[pos - 2]);

} else if (dir[a[pos - 1] - 1] == LEFT\_TO\_RIGHT)

{

swap(&a[pos], &a[pos - 1]);

}

for (int i = 0; i < n; i++) {

if (a[i] > mobile) {

if (dir[a[i] - 1] == LEFT\_TO\_RIGHT)

{

dir[a[i] - 1] = RIGHT\_TO\_LEFT;

} else if (dir[a[i] - 1] == RIGHT\_TO\_LEFT)

{

dir[a[i] - 1] = LEFT\_TO\_RIGHT;

}

}

}

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

{

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

}

printf("\n");

}

int factorial(int n)

{

int res = 1;

for (int i = 1; i <= n; i++)

{

res = res \* i;

}

return res;

}

void printPermutation(int n)

{

int a[n];

int dir[n];

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

{

a[i] = i + 1;

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

}

printf("\n");

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

{

dir[i] = RIGHT\_TO\_LEFT;

}

for (int i = 1; i < factorial(n); i++)

{

printOnePerm(a, dir, n);

}

}

int main()

{

int n;

printf("Enter the value of n: ");

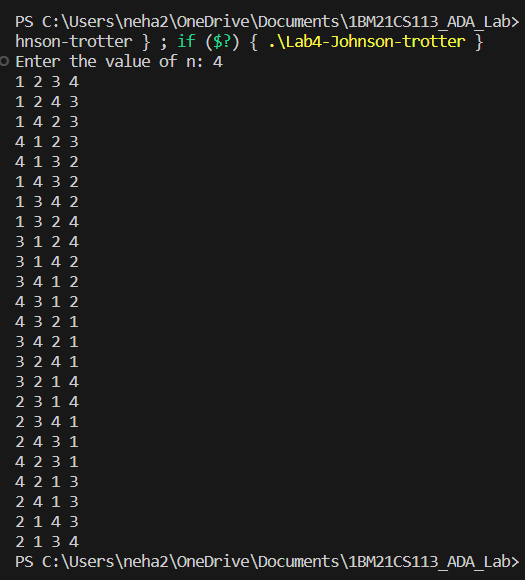
scanf("%d", &n);

printPermutation(n);

return 0;

}

**OUTPUT**

****

**PROGRAM 5**