Name – Shrinivas Hatyalikar

Div - CS-B

Roll no.- 24

PRN- 12110883

WAP to Perform BFS and DFS traversals on Graph using Adjacency Matrix and Adjacency Lists.

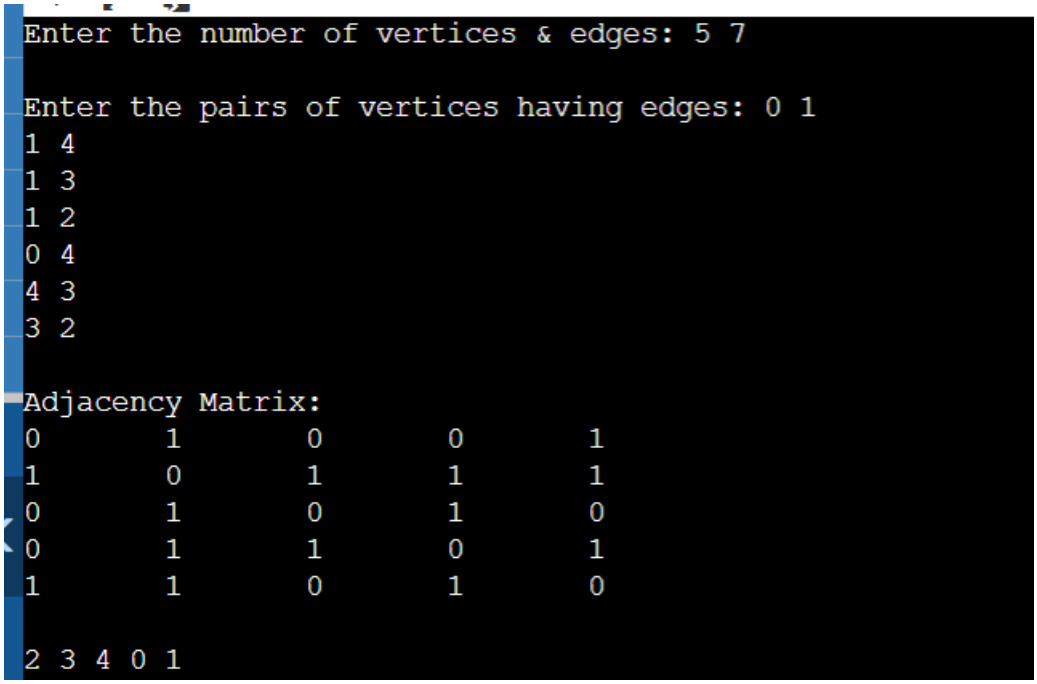
DFS using Adjacency Matrix:

```
#include <stdio.h>
#include <stdlib.h>

struct Node{
   int data;
   struct Node* next;
};
struct Node* top = NULL;

void push(int x){
   struct Node* temp = (struct Node*)malloc(sizeof(struct Node));
   temp->data=x;
   temp->next=top;
   top=temp;
}

void pop(){
   struct Node* temp;
   if(top==NULL){
      printf("\nUnderflow\n");
   }
   else{
      temp=top;
      top=top->next;
```

```
        free(temp);
    }
}

int visited[5] = {0};

void dfs(int i, int n, int adjMat[][n]) {
    push(i);
    visited[i]=1;
    while(top!=NULL){
        int curr_node=top->data;
        pop();
        printf("%d ",curr_node);
        for(int j=0;j<n;j++){
            if(adjMat[curr_node][j]==1 && visited[j]==0){ // fixed comparison
operators
                push(j);
                visited[j]=1;
            }
        }
    }
}

int main() {
    int n, m;
    printf("Enter the number of vertices & edges: ");
    scanf("%d %d", &n, &m);
    int adjMat[n][n];
    printf("\nEnter the pairs of vertices having edges: ");
    for (int i = 0; i < m; i++) {
        int u, v;
        scanf("%d %d", &u, &v);
        adjMat[u][v] = 1;
        adjMat[v][u] = 1;
    }
    printf("\nAdjacency Matrix:\n");
    for (int i = 0; i < n; i++) {
```

```
        for (int j = 0; j < n; j++) {
            printf("%d\t", adjMat[i][j]);
        }
        printf("\n");
    }
    printf("\n");
    dfs(2, n, adjMat);
    return 0;
}
```

Output:



DFS Using Adjacency List:

```
#include <stdio.h>
#include <stdlib.h>

struct nod
{
    int data;
    struct nod *next;
```

```
} *top = NULL;

void push(int x)
{
   struct nod *p;
   p = (struct nod *)malloc(sizeof(struct nod));
   p->data = x;
   if (top == NULL)
   {
      p->next = NULL;
   }
   else
   {
      p->next = top;
   }
   top = p;
}

int pop()
{
   if (top == NULL)
   {
      printf("No element in stack");
   }
   else
   {
      struct nod *p = top;
      int data = top->data;
      top = top->next;
      free(p);
      return data;
   }
}

void display()
{
   if (top == NULL)
```

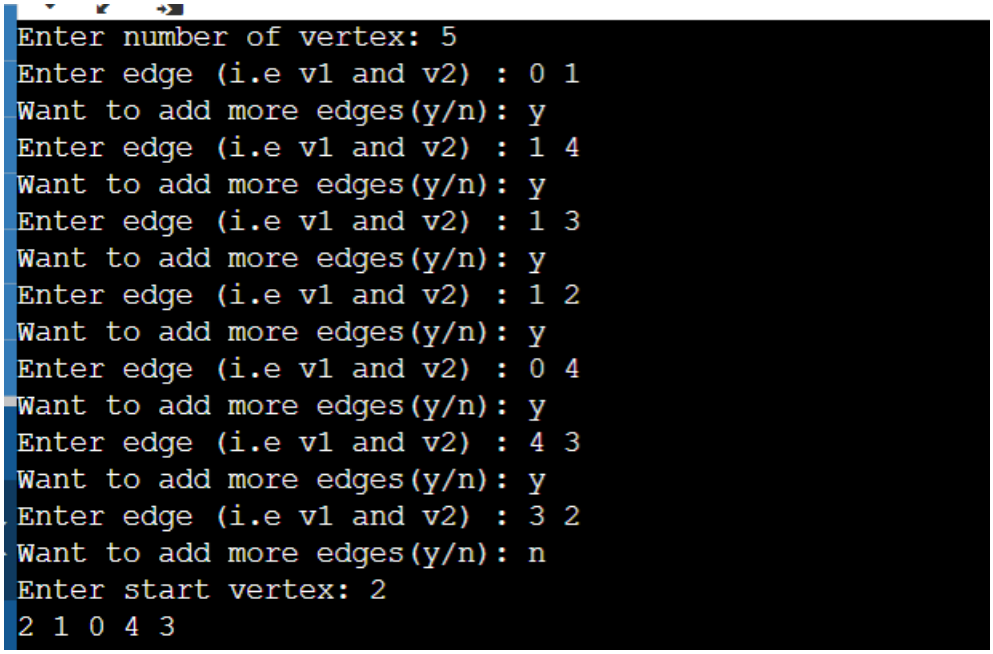
```
    {
        printf("No element in stack");
    }
    else
    {
        struct nod *p = top;
        while (p->next != NULL)
        {
            printf("%d\n", p->data);
            p = p->next;
        }
        printf("%d", p->data);
    }
}

struct node
{
    int vertex;
    struct node *next;
} *A[10];

void init(struct node *A[], int n)
{
    int i;
    for (int i = 0; i < n; i++)
    {
        A[i] = NULL;
    }
}

void create(struct node *A[])
{
    struct node *new, *p;
    char ch;
    do
    {
        int v1, v2;
```

```
    printf("Enter edge (i.e v1 and v2) : ");
    scanf("%d%d", &v1, &v2);
    new = (struct node *)malloc(sizeof(struct node));
    new->vertex = v2;
    new->next = NULL;
    p = A[v1];
    if (p == NULL)
    {
       A[v1] = new;
    }
    else
    {
       while (p->next != NULL)
       {
          p = p->next;
       }
       p->next = new;
    }
    new = (struct node *)malloc(sizeof(struct node));
    new->vertex = v1;
    new->next = NULL;
    p = A[v2];
    if (p == NULL)
    {
       A[v2] = new;
    }
    else
    {
       while (p->next != NULL)
       {
          p = p->next;
       }
       p->next = new;
    }
    printf("Want to add more edges(y/n): ");
    scanf(" %c", &ch);
 } while (ch == 'y' || ch == 'Y');
```

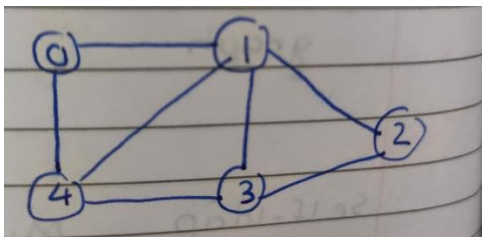
```
}

void dfs(struct node *A[], int n)
{
   struct node *p;
   int visited[10], i;
   for (i = 0; i < n; i++)
   {
      visited[i] = 0;
   }
   printf("Enter start vertex: ");
   scanf("%d", &i);
   push(i);
   visited[i] = 1;
   printf("%d ", i);
   do
   {
      p = A[i];
      while (p)
      {
         if (visited[p->vertex] == 0)
         {
            push(p->vertex);
            printf("%d ", p->vertex);
            visited[p->vertex] = 1;
            i = p->vertex;
            break;
         }
         else
         {
            p = p->next;
         }
         if (p == NULL)
         {
            pop();
            if (top != NULL)
            {
```

```
            i = top->data;
          }
        }
      }

   } while (top != NULL);
}

int main(int argc, char const *argv[])
{
   int n;
   printf("Enter number of vertex: ");
   scanf("%d", &n);
   init(A, n);
   create(A);
   dfs(A, n);
   return 0;
}
```

Output:



Graph:



BFS Using Adjacency Matrix:

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#define max 9999
struct node
{
   int data;
   struct node *next;
};
struct node *front = NULL, *rear = NULL;

void enqueue(int x)
{
   struct node *t;
   t = (struct node *)malloc(sizeof(struct node));
   if (t == NULL)
   {
      printf("Queue is Full\n");
   }
   else
   {
      t->data = x;
      t->next = NULL;
      if (front == NULL)
      {
         front = rear = t;
      }
      else
```

```
        {
            rear->next = t;
            rear = t;
        }
    }
}

int dequeue()
{
    int x = -1;
    struct node *t;

    if (front == NULL)
        printf("Queue is Empty\n");
    else
    {
        x = front->data;
        t = front;
        front = front->next;
        free(t);
    }
    return x;
}
int isEmpty()
{
    return front == NULL;
}

void BFS(int G[7][7], int v, int n)
{
    int visited[n];
    for (int i = 0; i < n; i++)
    {
        visited[i] = 0;
    }
    printf("%d ", v);
    enqueue(v);
```

```
    while (!isEmpty())
    {
        v = dequeue();
        for (int i = 0; i < n; i++)
        {
            if (G[v][i] == 1 && visited[i] == 0)
            {
                printf("%d ", i);
                visited[i] = 1;
                enqueue(i);
            }
        }
    }
    printf("\n");
}

void DFS(int G[7][7], int v, int n)
{
    static int visited[10] = {0};
    if (visited[v] == 0)
    {
        printf("%d ", v);
        visited[v] = 1;
        for (int i = 1; i < n; i++)
        {
            if (G[v][i] == 1 && visited[i] == 0)
            {
                DFS(G, i, n);
            }
        }
    }
}

int main()
{
    // int v;
    // printf("Enter number of vertices: ");
```

```
    // scanf("%d", &v);
    // int G[v + 1][v + 1];
    // printf("Enter graph data in matrix form:    \n");
    // for (int i = 0; i <= v; i++)
    // {
    //     for (int j = 0; j <= v; j++)
    //     {
    //         scanf("%d", &G[i][j]);
    //     }
    // }
    int G[7][7]={{0,0,0,0,0,0,0},
            {0,0,1,1,0,0,0},
            {0,1,0,0,1,0,0},
            {0,1,0,0,1,0,0},
            {0,0,1,1,0,1,1},
            {0,0,0,0,1,0,0},
            {0,0,0,0,1,0,0}};
    printf("BFS\n");
    BFS(G, 4, 7);
    // printf("DFS\n");
    // DFS(G,k,v);

    return 0;
}
```

Output:

```
BFS
4 2 3 5 6 1 4
```

BFS Using Adjacency List:

```
#include <stdio.h>
#include <stdlib.h>
```

```
struct nod
{
   int data;
   struct nod *next;
} *front = NULL, *rear = NULL;

void enqueue(int x)
{
   if (rear == NULL)
   {
      rear = (struct nod *)malloc(sizeof(struct nod));
      rear->data = x;
      rear->next = NULL;
      front = rear;
   }
   else
   {
      struct nod *temp = (struct nod *)malloc(sizeof(struct nod));
      temp->next = NULL;
      temp->data = x;
      rear->next = temp;
      rear = temp;
   }
}

int dequeue()
{
   int x;
   if (front == NULL)
   {
      printf("Queue is empty");
   }
   else if (front->next != NULL)
   {
      struct nod *temp = front;
      x = temp->data;
```

```
        front = front->next;
        free(temp);
    }
    else
    {
        x = front->data;
        free(front);
        front = NULL;
        rear = NULL;
    }
    return x;
}


struct node
{
    int vertex;
    struct node *next;
} *A[10];

void init(struct node *A[], int n)
{
    int i;
    for (int i = 0; i < n; i++)
    {
        A[i] = NULL;
    }
}

void create(struct node *A[])
{
    struct node *new, *p;
    char ch;
    do
    {
        int v1, v2;
        printf("Enter edge (i.e v1 and v2) : ");
```

```
        scanf("%d%d", &v1, &v2);
        new = (struct node *)malloc(sizeof(struct node));
        new->vertex = v2;
        new->next=NULL;
        p = A[v1];
        if (p == NULL)
        {
           A[v1] = new;
        }
        else
        {
           while (p->next != NULL)
           {
              p = p->next;
           }
           p->next = new;
        }
        new = (struct node *)malloc(sizeof(struct node));
        new->vertex = v1;
        new->next=NULL;
        p = A[v2];
        if (p == NULL)
        {
           A[v2] = new;
        }
        else
        {
           while (p->next != NULL)
           {
              p = p->next;
           }
           p->next = new;
        }
        printf("Want to add more edges(y/n): ");
        scanf(" %c", &ch);
    } while (ch == 'y' || ch == 'y');
}
```

```
void bfs(struct node *A[],int n){
   int visited[10],v;
   struct node *p;
   for(v=0;v<n;v++){
      visited[v]=0;
   }
   printf("Enter start vertex: ");
   scanf("%d",&v);
   enqueue(v);
   visited[v]=1;
   while(front!=NULL){
      v=dequeue();
      printf("%d ",v);
      p=A[v];
      while(p){
         if(visited[p->vertex]==0){
            enqueue(p->vertex);
            visited[p->vertex]=1;
         }
         p=p->next;
      }
   }
}

int main(int argc, char const *argv[])
{
   int n;
   printf("Enter number of vertex: ");
   scanf("%d", &n);
   init(A, n);
   create(A);
   bfs(A, n);
   return 0;
}
```

Output:

```
Enter number of vertex: 5
Enter edge (i.e v1 and v2) : 0 1
Want to add more edges(y/n): y
Enter edge (i.e v1 and v2) : 1 2
Want to add more edges(y/n): y
Enter edge (i.e v1 and v2) : 1 3
Want to add more edges(y/n): y
Enter edge (i.e v1 and v2) : 1 4
Want to add more edges(y/n): y
Enter edge (i.e v1 and v2) : 2 3
Want to add more edges(y/n): y
Enter edge (i.e v1 and v2) : 3 4
Want to add more edges(y/n): y
Enter edge (i.e v1 and v2) : 4 0
Want to add more edges(y/n): n
Enter start vertex: 0
0 1 4 2 3
```

Graph: