**M. Ali. Arslan**

**19F-0348**

**Lab-10**

**Task # 01**

#include <iostream>

using namespace std;

struct Node

{

int val;

Node\* next;

};

struct Edge {

int src, dest;

};

class Graph

{

Node\* getNode(int dest, Node\* head)

{

Node\* newNode = new Node;

newNode->val = dest;

newNode->next = head;

return newNode;

}

int N;

public:

Node\*\* head;

Graph(Edge edges[], int n, int N)

{

head = new Node \* [N]();

this->N = N;

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

head[i] = nullptr;

}

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

{

int src = edges[i].src;

int dest = edges[i].dest;

Node\* newNode = getNode(dest, head[src]);

head[src] = newNode;

}

}

~Graph() {

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

delete[] head[i];

}

delete[] head;

}

};

void printList(Node\* ptr)

{

while (ptr != nullptr)

{

cout << ptr->val << " -> ";

ptr = ptr->next;

}

cout << endl;

}

int main()

{

Edge edges[] =

{

{ 0, 1 }, { 1, 2 }, { 2, 0 }, { 2, 1 },{ 3, 2 }, { 4, 5 }, { 5, 4 }

};

int N = 6;

int n = sizeof(edges) / sizeof(edges[0]);

Graph graph(edges, n, N);

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

{

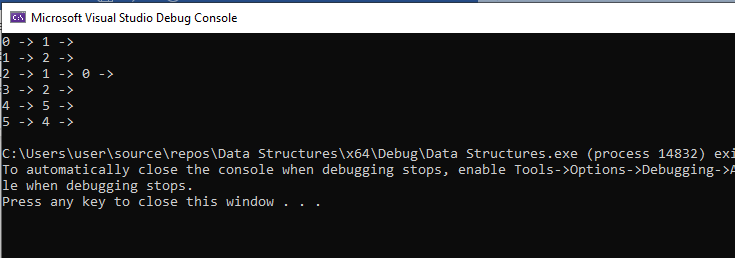
cout << i << " -> ";

printList(graph.head[i]);

}

return 0;

}



**Task # 02**

#include <iostream>

using namespace std;

class Node

{

public:

int data;

Node\* link;

};

Node\* front = NULL;

Node\* rear = NULL;

class Queue

{

private:

int count;

public:

Queue()

: count(0)

{}

bool isEmpty() {

if (front == NULL && rear == NULL)

{

return true;

}

else

{

return false;

}

};

bool IsFull() const

{

Node\* temp;

try {

temp = new Node;

delete temp;

return false;

}

catch (bad\_alloc exception)

{

return true;

}

}

void enqueue(int d)

{

Node\* ptr = new Node;

if (isEmpty())

{

ptr->data = d;

front = rear = ptr;

}

else

{

ptr->data = d;

rear->link = ptr;

rear = ptr;

}

};

int dequeue()

{

int data;

if (isEmpty())

{

//cout << "Queue is empty, cannot dequeue!" << endl;

return -1;

}

else if (front == rear)

{

data = front->data;

//cout << "Dequeued: " << data << endl;

front = rear = NULL;

return data;

}

else

{

Node\* ptr = front;

data = front->data;

front = front->link;

ptr = NULL;

//cout << "Dequeued: " << data << endl;

return data;

}

};

void display()

{

if (isEmpty())

{

cout << "Queue is empty!" << endl;

}

else

{

cout << front->data << endl;

}

};

};

void BFS(int G[][7], int start, int n)

{

char a = NULL;

Queue q1;

int i = start, j;

int visited[7] = { 0 };

a = 65 + i;

cout << a << " ";

visited[i] = 1;

q1.enqueue(i);

while (!q1.isEmpty())

{

i = q1.dequeue();

for (j = 1; j < n; j++)

{

if (G[i][j] == 1 && visited[j] == 0)

{

a = 65 + j;

cout << a << " ";

visited[j] = 1;

q1.enqueue(j);

}

}

}

};

void BFS(int G[][6], int start, int n)

{

char a = NULL;

Queue q1;

int i = start, j;

int visited[6] = { 0 };

a = 65 + i;

cout << a << " ";

visited[i] = 1;

q1.enqueue(i);

while (!q1.isEmpty())

{

i = q1.dequeue();

for (j = 1; j < n; j++)

{

if (G[i][j] == 1 && visited[j] == 0)

{

a = 65 + j;

cout << a << " ";

visited[j] = 1;

q1.enqueue(j);

}

}

}

};

void DFS(int G[][7], int start, int n)

{

char a = NULL;

static int visited[7] = { 0 };

int j;

if (visited[start] == 0)

{

a = 65 + start;

cout << a << " ";

visited[start] = 1;

for (j = 1; j < n; j++)

{

if (G[start][j] == 1 && visited[j] == 0)

DFS(G, j, n);

}

}

}

void DFS(int G[][6], int start, int n)

{

char a = NULL;

static int visited[6] = { 0 };

int j;

if (visited[start] == 0)

{

a = 65 + start;

cout << a << " ";

visited[start] = 1;

for (j = 1; j < n; j++)

{

if (G[start][j] == 1 && visited[j] == 0)

DFS(G, j, n);

}

}

}

int main()

{

cout << "\t\t\t\t\tGRAPH TRAVERSALS" << endl << endl;

int G1[7][7] = {

{0,1,0,0,1,1,0},

{1,0,1,0,0,0,0},

{0,1,0,1,0,0,0},

{0,0,1,0,0,0,0},

{1,0,0,0,0,0,0},

{1,0,0,0,0,0,1},

{0,0,0,0,0,1,0} };

int G2[6][6] =

{

{0,1,1,1,0,0},

{1,0,1,0,1,0},

{1,1,0,0,0,0},

{1,0,0,0,1,1},

{0,1,0,1,0,0},

{0,0,0,1,0,0}

};

cout << "Graph 1 BFS : ";

BFS(G1, 0, 7);

cout << endl << endl;

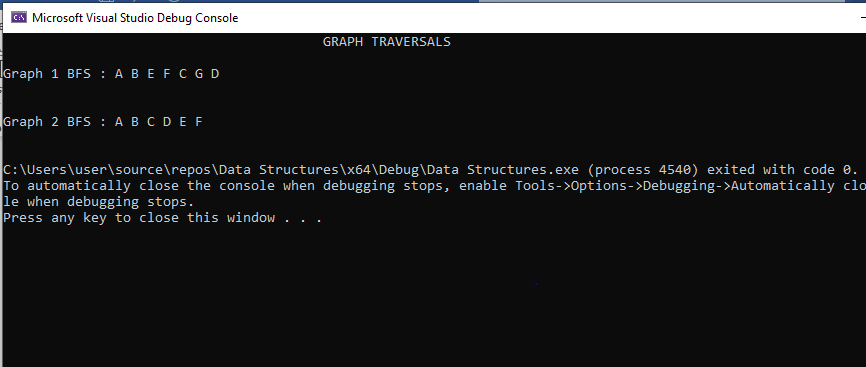
cout << "\nGraph 2 BFS : ";

BFS(G2, 0, 6);

cout << endl << endl;

return 0;

}



**Task # 03**

#include <iostream>

using namespace std;

class Node

{

public:

int data;

Node\* link;

};

Node\* front = NULL;

Node\* rear = NULL;

class Queue

{

private:

int count;

public:

Queue()

: count(0)

{}

bool isEmpty() {

if (front == NULL && rear == NULL)

{

return true;

}

else

{

return false;

}

};

bool IsFull() const

{

Node\* temp;

try {

temp = new Node;

delete temp;

return false;

}

catch (bad\_alloc exception)

{

return true;

}

}

void enqueue(int d)

{

Node\* ptr = new Node;

if (isEmpty())

{

ptr->data = d;

front = rear = ptr;

}

else

{

ptr->data = d;

rear->link = ptr;

rear = ptr;

}

};

int dequeue()

{

int data;

if (isEmpty())

{

//cout << "Queue is empty, cannot dequeue!" << endl;

return -1;

}

else if (front == rear)

{

data = front->data;

//cout << "Dequeued: " << data << endl;

front = rear = NULL;

return data;

}

else

{

Node\* ptr = front;

data = front->data;

front = front->link;

ptr = NULL;

//cout << "Dequeued: " << data << endl;

return data;

}

};

void display()

{

if (isEmpty())

{

cout << "Queue is empty!" << endl;

}

else

{

cout << front->data << endl;

}

};

};

void BFS(int G[][7], int start, int n)

{

char a = NULL;

Queue q1;

int i = start, j;

int visited[7] = { 0 };

a = 65 + i;

cout << a << " ";

visited[i] = 1;

q1.enqueue(i);

while (!q1.isEmpty())

{

i = q1.dequeue();

for (j = 1; j < n; j++)

{

if (G[i][j] == 1 && visited[j] == 0)

{

a = 65 + j;

cout << a << " ";

visited[j] = 1;

q1.enqueue(j);

}

}

}

};

void BFS(int G[][6], int start, int n)

{

char a = NULL;

Queue q1;

int i = start, j;

int visited[6] = { 0 };

a = 65 + i;

cout << a << " ";

visited[i] = 1;

q1.enqueue(i);

while (!q1.isEmpty())

{

i = q1.dequeue();

for (j = 1; j < n; j++)

{

if (G[i][j] == 1 && visited[j] == 0)

{

a = 65 + j;

cout << a << " ";

visited[j] = 1;

q1.enqueue(j);

}

}

}

};

void DFS(int G[][7], int start, int n)

{

char a = NULL;

static int visited[7] = { 0 };

int j;

if (visited[start] == 0)

{

a = 65 + start;

cout << a << " ";

visited[start] = 1;

for (j = 1; j < n; j++)

{

if (G[start][j] == 1 && visited[j] == 0)

DFS(G, j, n);

}

}

}

void DFS(int G[][6], int start, int n)

{

char a = NULL;

static int visited[6] = { 0 };

int j;

if (visited[start] == 0)

{

a = 65 + start;

cout << a << " ";

visited[start] = 1;

for (j = 1; j < n; j++)

{

if (G[start][j] == 1 && visited[j] == 0)

DFS(G, j, n);

}

}

}

int main()

{

cout << "\t\t\t\t\tGRAPH TRAVERSALS" << endl << endl;

int G1[7][7] = {

{0,1,0,0,1,1,0},

{1,0,1,0,0,0,0},

{0,1,0,1,0,0,0},

{0,0,1,0,0,0,0},

{1,0,0,0,0,0,0},

{1,0,0,0,0,0,1},

{0,0,0,0,0,1,0} };

int G2[6][6] =

{

{0,1,1,1,0,0},

{1,0,1,0,1,0},

{1,1,0,0,0,0},

{1,0,0,0,1,1},

{0,1,0,1,0,0},

{0,0,0,1,0,0}

};

cout << "Graph 1 DFS : ";

DFS(G1, 0, 7);

cout << endl << endl;

cout << "Graph 2 DFS : ";

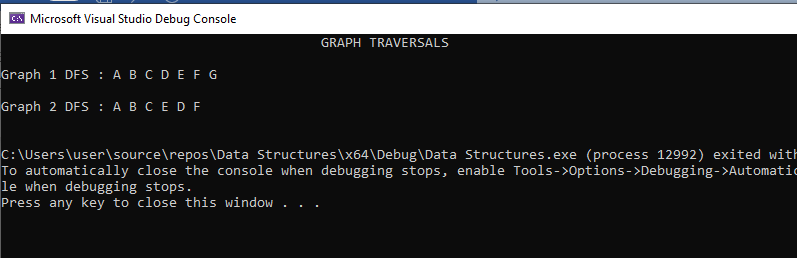
DFS(G2, 0, 6);

cout << endl << endl;

return 0;

return 0;

}



**Task # 04**

#include<iostream>

#include<list>

using namespace std;

int N;

class graph

{

int v;

list<int>\* adj;

public:

graph(int v)

{

this->v = v;

adj = new list<int>[v];

}

void add\_edge(int u, int v)

{

adj[u].push\_back(v);

adj[v].push\_back(u);

}

void print()

{

for (int v = 0; v < N; v++)

{

cout << v;

for (auto it = adj[v].begin(); it != adj[v].end(); it++)

{

cout << " -> " << (\*it) << " ";

}

cout << endl;

}

}

void DFS(int i, bool dfs[])

{

dfs[i] = true;

cout << i << " ";

list<int>::iterator it;

for (it = adj[i].begin(); it != adj[i].end(); it++)

{

if (dfs[\*it] == false)

{

DFS(\*it, dfs);

}

}

}

void BFS(int s)

{

int v = N;

bool\* visited = new bool[v];

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

visited[i] = false;

list<int> queue;

visited[s] = true;

queue.push\_back(s);

list<int>::iterator i;

while (!queue.empty())

{

s = queue.front();

cout << s << " ";

queue.pop\_front();

for (i = adj[s].begin(); i != adj[s].end(); ++i)

{

if (!visited[\*i])

{

visited[\*i] = true;

queue.push\_back(\*i);

}

}

}

}

};

int main()

{

N = 13;

graph g(N);

// in graph 2 i let ( A = 0 , B =1 , C =2 ,D=3 ,E=4 ,F=5,G =6,H=7,I=8,J=9,K=10,L=11,M=12 )

g.add\_edge(0, 1);

g.add\_edge(0, 2);

g.add\_edge(1, 3);

g.add\_edge(1, 4);

g.add\_edge(2, 5);

g.add\_edge(3, 6);

g.add\_edge(3, 7);

g.add\_edge(4, 8);

g.add\_edge(5, 9);

g.add\_edge(6, 10);

g.add\_edge(6, 11);

g.add\_edge(8, 12);

cout << "DFS : ";

bool dfs[13];

int i;

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

{

dfs[i] = false;

}

g.DFS(0, dfs);

cout << endl;

cout << "BFS : ";

g.BFS(0);

cout << endl;

system("pause");

return 0;

}

