**V Venkateswarlu Gupta**

**1602-19-733-119**

**CSE-B**

**Data Structures Lab**

Week – 9

***Prelab Questions***

1. **Define a graph?**

A graph is a data structure that consists of the following two components:  
1.A finite set of vertices also called as nodes.  
2.A finite set of ordered pair of the form (u, v) called as edge. The pair is ordered because (u, v) is not the same as (v, u) in case of a directed graph(di-graph). The pair of the form (u, v) indicates that there is an edge from vertex u to vertex v. The edges may contain weight/value/cost.

1. **What are different ways of representing a graph.**
2. Adjacency matrix
3. Adjacency list
4. **Define indegree and outdegree.**

### Indegree of a Graph

Indegree of vertex V is the number of edges which are coming into the vertex V

**Notation** − deg−(V).

### Outdegree of a Graph

Outdegree of vertex V is the number of edges which are going out from the vertex V.

**Notation** − deg+(V).

1. **What is a complete graph.**

A **complete graph** is a [simple](https://en.wikipedia.org/wiki/Simple_graph) [undirected graph](https://en.wikipedia.org/wiki/Undirected_graph) in which every pair of distinct [vertices](https://en.wikipedia.org/wiki/Vertex_(graph_theory)) is connected by a unique [edge](https://en.wikipedia.org/wiki/Edge_(graph_theory)). A **complete digraph** is a [directed graph](https://en.wikipedia.org/wiki/Directed_graph) in which every pair of distinct vertices is connected by a pair of unique edges (one in each direction).

1. **Differentiate between BFS and DFS.**

Breadth First Search (BFS) algorithm traverses a graph in a breadthward motion and uses a queue to remember to get the next vertex to start a search when a dead end occurs in any iteration.

Depth First Search (DFS) algorithm traverses a graph in a depthward motion and uses a stack to remember to get the next vertex to start a search when a dead end occurs in any iteration

1. **Write the sequence in which vertices are visited in bfs and dfs.**

A

/ \

B C

/ / \

D E F

**BFS:-A,B,C,D,E,F**

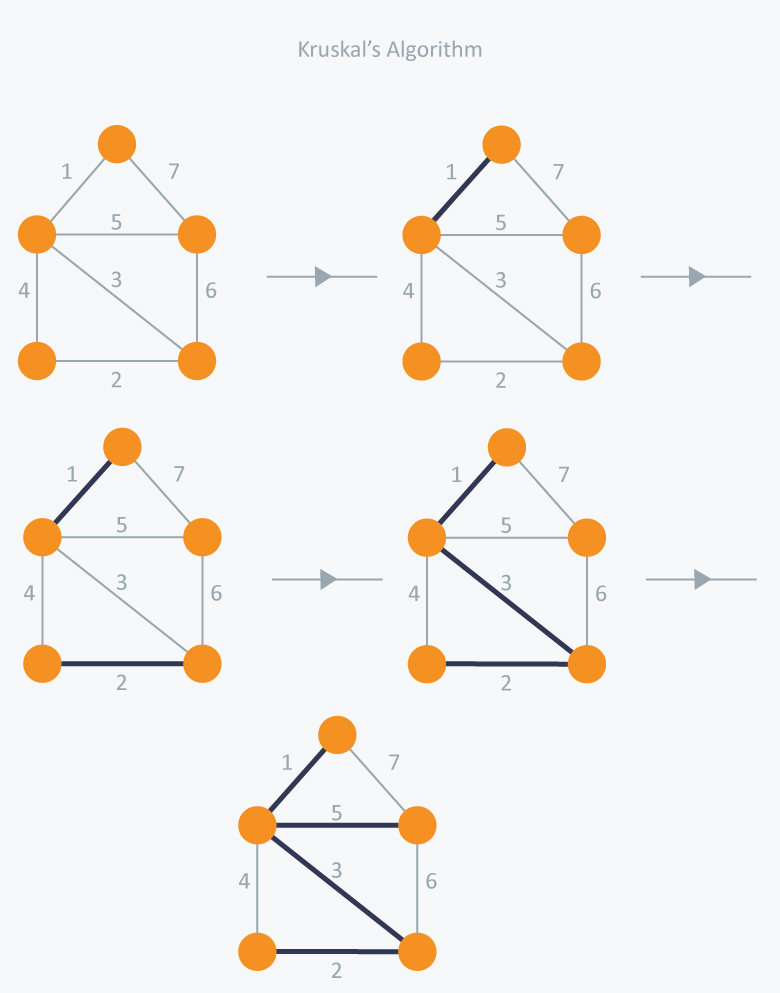
**DFS:-A,B,D,C,E,F**

1. **Applications of graph data structure.**

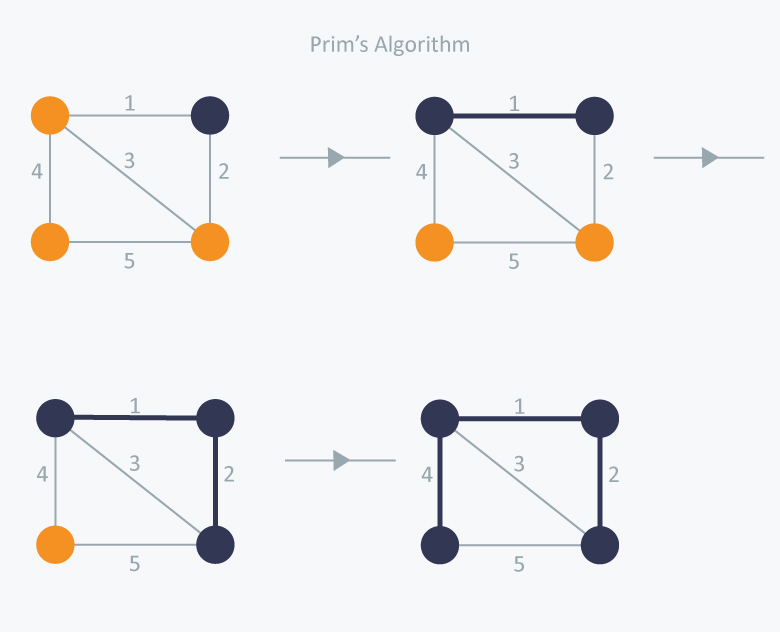
* In **Computer science** graphs are used to represent the flow of computation.
* **Google maps** uses graphs for building transportation systems, where intersection of two(or more) roads are considered to be a vertex and the road connecting two vertices is considered to be an edge, thus their navigation system is based on the algorithm to calculate the shortest path between two vertices.
* In **Facebook**, users are considered to be the vertices and if they are friends then there is an edge running between them. Facebook’s Friend suggestion algorithm uses graph theory. Facebook is an example of **undirected graph**.
* In **World Wide Web**, web pages are considered to be the vertices. There is an edge from a page u to other page v if there is a link of page v on page u. This is an example of **Directed graph**. It was the basic idea behind [Google](https://www.geeksforgeeks.org/page-rank-algorithm-implementation/)

1. **What are the datastructures used for implementing BFS and DFS?**
2. BFS-queues

DFS-Stacks

1. **Construct a minimum cost spanning tree using Prims algorithm and krukshals algorithm consider weighted graph.**
2. 

PRIMS algorithm based:-



Programs:

1.BFS

#include <stdio.h>

#include <stdlib.h>

*struct* node

{

*int* vertex;

*struct* node \*next;

};

*struct* graph

{

*int* noOfVertices;

*struct* node \*\*lists;

*int* \*visited;

};

*struct* node \*createNode(*int* *v*)

{

*struct* node \*newNode = (*struct* node \*)malloc(sizeof(*struct* node));

    newNode->vertex = *v*;

    newNode->next = NULL;

    return newNode;

}

*struct* graph \*createGraph(*int* *vertices*)

{

*struct* graph \*g = (*struct* graph \*)malloc(sizeof(*struct* graph));

    g->noOfVertices = *vertices*;

    g->lists = (*struct* node \*\*)malloc(*vertices* \* sizeof(*struct* node \*));

    g->visited = (*int* \*)malloc(*vertices* \* sizeof(*int*));

*int* i;

    for (i = 0; i < *vertices*; i++)

    {

        g->lists[i] = NULL;

        g->visited[i] = 0;

    }

    return g;

}

*void* addEdge(*struct* graph \**g*, *int* *src*, *int* *dest*)

{

*struct* node \*newNode = createNode(*dest*);

    newNode->next = *g*->lists[*src*];

*g*->lists[*src*] = newNode;

    newNode = createNode(*src*);

    newNode->next = *g*->lists[*dest*];

*g*->lists[*dest*] = newNode;

}

*struct* Q

{

*int* elements[1000];

*int* front;

*int* rear;

};

*struct* Q \*createQ()

{

*struct* Q \*q = (*struct* Q \*)malloc(sizeof(*struct* Q));

    q->front = -1;

    q->rear = -1;

    return q;

}

*int* isEmpty(*struct* Q \**q*)

{

    return *q*->rear == -1 ? 1 : 0;

}

*void* enQ(*struct* Q \**q*, *int* *value*)

{

    if (*q*->rear == 1000 - 1)

    {

        printf("\nQ full\n");

    }

    else

    {

        if (*q*->front == -1)

            ++*q*->front;

*q*->elements[++*q*->rear] = *value*;

    }

}

*int* deQ(*struct* Q \**q*)

{

*int* removed;

    if (isEmpty(*q*))

    {

        printf("Q is empty\n");

        removed = -1;

    }

    else

    {

        removed = *q*->elements[*q*->front++];

        if (*q*->front > *q*->rear)

        {

*q*->front = *q*->rear = -1;

        }

    }

    return removed;

}

*void* printQ(*struct* Q \**q*)

{

    if (isEmpty(*q*))

    {

        printf("Queue is empty");

    }

    else

    {

        printf("\nQueue contains \n");

        for (*int* i = *q*->front; i < *q*->rear + 1; i++)

        {

            printf("%d ", *q*->elements[i]);

        }

    }

    printf("\n");

}

*void* BFS(*struct* graph \**g*, *int* *startVertex*)

{

*struct* Q \*q = createQ();

*int* visitingOrder[1000], i = 0;

*g*->visited[*startVertex*] = 1;

    enQ(q, *startVertex*);

    while (!isEmpty(q))

    {

        printQ(q);

*int* cur = deQ(q);

        visitingOrder[i++] = cur;

        visitingOrder[i] = -1;

        printf("Visited %d \n", cur);

*struct* node \*t = *g*->lists[cur];

        while (t)

        {

*int* adjvertex = t->vertex;

            if (*g*->visited[adjvertex] == 0)

            {

*g*->visited[adjvertex] = 1;

                enQ(q, adjvertex);

            }

            t = t->next;

        }

    }

}

*void* main()

{

*struct* graph \*graph = createGraph(10);

    addEdge(graph, 0, 1);

    addEdge(graph, 0, 2);

    addEdge(graph, 1, 2);

    addEdge(graph, 1, 4);

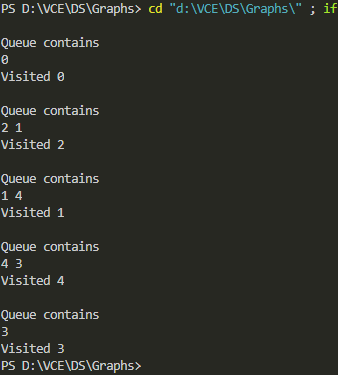
    addEdge(graph, 1, 3);

    addEdge(graph, 2, 4);

    addEdge(graph, 3, 4);

    BFS(graph, 0);

}



2.DFS

#include <stdio.h>

#include <stdlib.h>

*struct* node {

*int* vertex;

*struct* node\* next;

};

*struct* node\* createNode(*int* *v*) {

*struct* node\* newNode = malloc(sizeof(*struct* node));

  newNode->vertex = *v*;

  newNode->next = NULL;

  return newNode;

}

*struct* Graph {

*int* numVertices;

*int*\* visited;

*struct* node\*\* adjLists;

};

*void* DFS(*struct* Graph\* *graph*, *int* *vertex*) {

*struct* node\* temp = *graph*->adjLists[*vertex*];

*graph*->visited[*vertex*] = 1;

  printf("Visited %d \n", *vertex*);

  while (temp != NULL) {

*int* connectedVertex = temp->vertex;

    if (*graph*->visited[connectedVertex] == 0) {

      DFS(*graph*, connectedVertex);

    }

    temp = temp->next;

  }

}

*struct* Graph\* createGraph(*int* *vertices*) {

*struct* Graph\* graph = malloc(sizeof(*struct* Graph));

  graph->numVertices = *vertices*;

  graph->adjLists = malloc(*vertices* \* sizeof(*struct* node\*));

  graph->visited = malloc(*vertices* \* sizeof(*int*));

*int* i;

  for (i = 0; i < *vertices*; i++) {

    graph->adjLists[i] = NULL;

    graph->visited[i] = 0;

  }

  return graph;

}

*void* addEdge(*struct* Graph\* *graph*, *int* *src*, *int* *dest*) {

*struct* node\* newNode = createNode(*dest*);

  newNode->next = *graph*->adjLists[*src*];

*graph*->adjLists[*src*] = newNode;

  newNode = createNode(*src*);

  newNode->next = *graph*->adjLists[*dest*];

*graph*->adjLists[*dest*] = newNode;

}

*void* printGraph(*struct* Graph\* *graph*) {

*int* v;

  for (v = 0; v < *graph*->numVertices; v++) {

*struct* node\* temp = *graph*->adjLists[v];

    printf("\n Adjacency list of vertex %d\n ", v);

    while (temp) {

      printf("%d -> ", temp->vertex);

      temp = temp->next;

    }

    printf("\n");

  }

}

*void* main() {

*struct* Graph\* graph = createGraph(4);

  addEdge(graph, 0, 1);

  addEdge(graph, 0, 2);

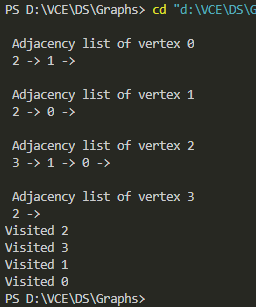
  addEdge(graph, 1, 2);

  addEdge(graph, 2, 3);

  printGraph(graph);

  DFS(graph, 2);

}



Lab program:

#include<stdio.h>

#include<stdlib.h>

#define INF 987654

*void* dijkstras(*int* \*\**graph*, *int* *n*, *int* *source*){

*int* i,j,max=0;

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

*int* \*distance=(*int* \*)calloc(*n*,sizeof(*int*));

    for(i=0;i<*n*;i++){

        distance[i]=INF;

    }

    distance[*source*]=0;

    for(i=0;i<*n*;i++){

*int* u=INF;

        for(j=0;j<*n*;j++){

            if(u>=distance[j]&&visited[j]==0){

                u=j;

                break;

            }

        }

        visited[u]=1;

        for(j=0;j<*n*;j++){

            max+=*graph*[i][j];

            if(visited[j]==0&&*graph*[u][j]!=0){

                if(distance[j]>distance[u]+*graph*[u][j]){

                    distance[j]=distance[u]+*graph*[u][j];

                }

            }

        }

    }

    for(i=0;i<*n*;i++){

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

    }

    printf("\n");

}

*void* main(){

*int* \*\*graph;

*int* i,j,n;

    FILE \*fp;

    fp=fopen("inputDijkstras1.txt","r");

    fscanf(fp,"%d",&n);

    graph=(*int* \*\*)malloc(n\*sizeof(*int* \*));

    for(i=0;i<n;i++){

        graph[i]=(*int* \*)malloc(sizeof(*int*)\*n);

        for(j=0;j<n;j++){

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

        }

    }

    printf("From A : ");

    dijkstras(graph, n, 0);

}

Input

5

0 3 0 0 0

0 0 2 4 0

0 0 0 0 5 `

0 0 5 0 0

0 0 0 2 0

