Laboratory 8

Graphs

I. OBJECTIVES:

In this laboratory, you will

- understand and apply the concepts of graphs
- implement the different operations of graphs such as adding and removing of vertices
- creating an edge between vertices
- implement Warshall's Algorithm in determining the possible paths between vertices
- implement Dijkstra algorithm to derive the shortest path between vertices
- apply the graph ADT in different applications

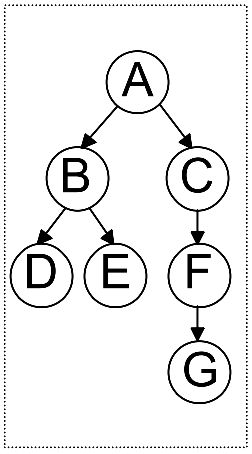
II. DISCUSSION:

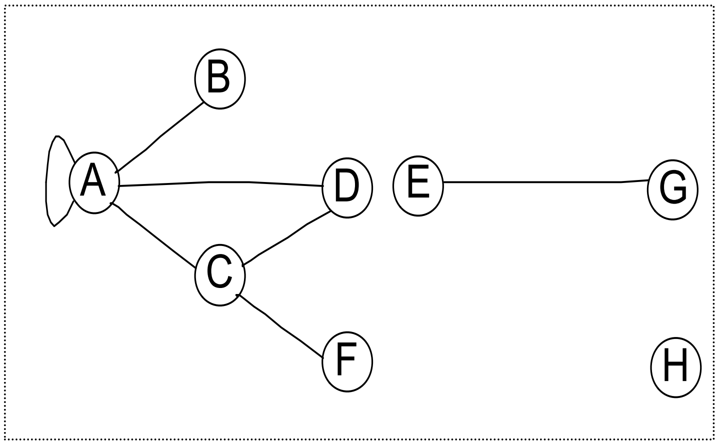
Graph

It is a collection of vertices (or nodes) and the connections between them.

- It represents one limitation of trees. Trees can only represent relations of hierarchical type, such as relations between parent and child.
- A graph need not be a tree but tree must be a graph.
- A node need not have any arcs associated with it.

Examples:



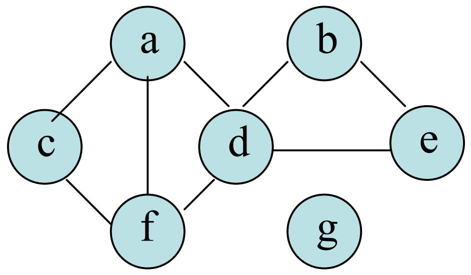


Directed Graph

Undirected Graph

Graph Representation

From the given graph, you can represent it using adjacency matrix and adjacency list tables.



Adjacency Matrix – it represents every possible ordered pair of nodes.

Example:

	a	b	c	d	e	f	g
a	0	0	1	1	0	1	0
b	0	0	0	1	1	0	0
c	1	0	0	0	0	1	0
d	1	1	0	0	1	1	0
e	0	1	0	1	0	0	0
f	1	0	1	1	0	0	0
g	0	0	0	0	0	0	0

Adjacency List – it specifies all nodes adjacent to each node of the graph.

Example:

a	c	d	f	
b	d	e		
c	a	f		
d	a	b	e	f
e	b	d		
f	a	c	d	
g				

Graph Implementation

Implementation of graph requires assumption that the number of nodes in the graph is constant so that arcs may be added or deleted but nodes may not.

DECLARATION:

```
#define MAXNODES 50
struct node{
  /* information associated with each node*/
};
struct arc{
  int adj;
  /* information associated with each arc*/
};
struct graph{
  struct node nodes[MAXNODES];
  struct arc arcs[MAXNODES][MAXNODES];
};
struct graph g;
```

FUNCTIONS:

```
void join(int adj[][MAXNODES],int node1,int node2) {
  /* add an arc from node1 to node2*/
 adj[node1][node2]=TRUE;
};//end join
void remv(int adj[][MAXNODES],int node1,int node2) {
  /* delete arc from node1 to node2 if one exists*/
 adj[node1][node2]=FALSE;
};//end remv
int  adjacent(int adj[][MAXNODES],int node1,int node2) {
  return((adj[node1][node2]==TRUE)? TRUE:FALSE);
};//end adjacent
```

Warshall's Algorithm

The path *k* is define such that path *k[i][j]* is true if and only if there is a path from node *i* to *j* that does not pass through any nodes numbered higher than *k*.

Thus, path *k+1[i][j]* equals TRUE if and only if one of the following two conditions holds:

1. pathk[i][j] = TRUE
2. pathk[i][k+1]= TRUE and pathk[k+1][j]= TRUE

Dijkstra's Algorithm

A number of paths P1.. Pn from a vertex *v* are tried, and each time, the shortest path is chosen among them, which may mean that the same path Pi can be continued by adding one more edge to it. But if Pi turns out to be longer than any other path that can be tried, Pi is abandoned and this other path is tried by resuming from where it was left and by adding one more edge to it.

```
DijkstraAlgorithm (weighted simple Digraph, vertex first)
    for all vertices v
        currDist(v) = α;
        currDist(first)= 0;
        toBeChecked= all vertices;
    while toBeChecked is not empty
        v = a vertex in toBeChecked with minimal currDist(v);
    remove v from toBeChecked;
    for all vertices u adjacent to v and in toBeChecked
        if currDist(u) >currDist(v) + weight(edge(vu))
        currDist(u) = currDist(v) + weight (edge(vu));
  predecessor(u)=v;
```

III. Test and Debug

1. Create a test program that will implement the following *graphs* operations:

Command	Operations
`> +v`	`Insert vertex` *`v`*`.`
`> =v w wt`	`Insert an edge connecting vertices` *`v`* `and` *`w`*`.  The weight of this edge is` *`wt`*`.`
`> c`	`Clear the graph (empty graph)`
`> ? a`	`Display adjacency matrix`
`> ? v`	`Display adjacency list`
`> q`	`Quit the test program`

2. Do the following sample runs:

Commands	Results/Outputs
`> c`	Empty graph
`> +a`	a
`> +b`	b
`> =a b 2`	Edge = ab Wt=2
`> ? a`	Adjacency Matrix: ab a 1 b 0
`> ? v`	Adjacency List: a b b
`> q`	Exit/quit

IV. Supplementary Exercise

1. Implement the additional graph operations summarized below and add them in the program created in test and debug.

Command	Operations
`> - v`	`Remove vertex` *`v`*
`> ! r w`	`Remove the edge connecting vertices` *`v`* `and` *`w`*`.`

2. Prepare a test plan to verify the functions.

IV. Machine Problem

1. Assume one input containing four integers followed by any number of input lines with two integers each.

   First set of data:

   n = number of cities
   a,b = represent the two cities
   nr = number of roads (path)

   Second set of data:

   city1 = the first city
   city2 = the second city

It is desired to travel from the first city (a) to the second (b) using exactly *nr* roads.

Write a program that can determine whether there is a path of required length by which one can travel from the first of the given cities to the second using Warshall's Algorithm.

Graphs

DATA & RESULTS SHEET

(Tentative Laboratory Report)

Name: ____

Schedule:______________________________ Section:__________________

Date: _________________________________ Grade: __________________

Test & Debug:

Step No.	Answers/Results
1	Attach source codes
2	

Commands	Results/Outputs
`> c`	Empty graph
`> +a`	a
`> +b`	b
`> +c`	c
`> =a b 2`	Edge = ab Wt=2
`> ? a`	Adjacency Matrix: ab a 1 b 0
`> ? v`	Adjacency List: a b b
`> q`	Exit/quit

Supplementary Exercise:

Step No.	Answers/Results
1	Attach source codes
2	

Commands	Results/Outputs
`> c`	Empty graph
`> +a`	a
`> +b`	b
`> +c`	c
`> =a b 2`	Edge = ab Wt=2
`> -c`	vertex c deleted!
`> ? a`	Adjacency Matrix: ab a 1 b 0
`> ? v`	Adjacency List: a b b
`> ! a b`	Edge ab removed!
`> q`	Exit/quit

Machine Problem:

Topic	Completed?		Remarks
	YES	NO	

Note: Check the column YES if completed otherwise check the column NO.(for instructors use only)

INSTRUCTOR'S SIGNATURE: ____________________________