Unit 9: Graph

a. Introduction
b. Representation of Graph
   - Array
   - Linked List
c. Traversal
   - Depth First Search
   - Breadth First Search
d. Minimum spanning tree
   - Kruskal's Algorithm

Introduction

Definition of Graph

- A graph ***G*** consists of a set ***V*** of vertices (nodes) and a set ***E*** of edges (or arcs) which is a pair of vertices.
- We write ***G=(V, E)***, ***V*** is a finite and non-empty set of vertices, and ***E*** is a set of pairs of vertices called edges.
- Thus, ***V(G)***, read as ***V*** of ***G***, is set of vertices and ***E(G)***, read as ***E*** of ***G***, is set of edges.
- An edge ***e1=(v1, v2)***, is a pair of vertices.

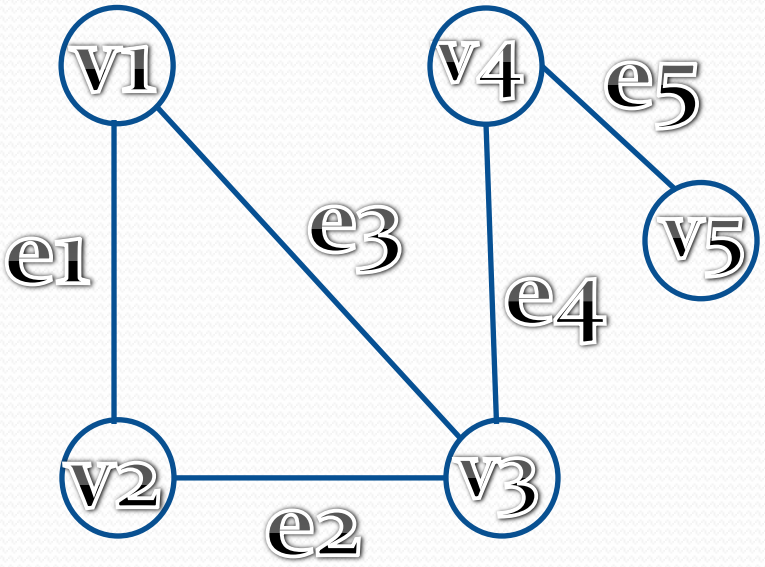

Fig: Graph

V(G)={v1,v2,v3,v4,v5}

E(G)={e1,e2,e3,e4,e5}

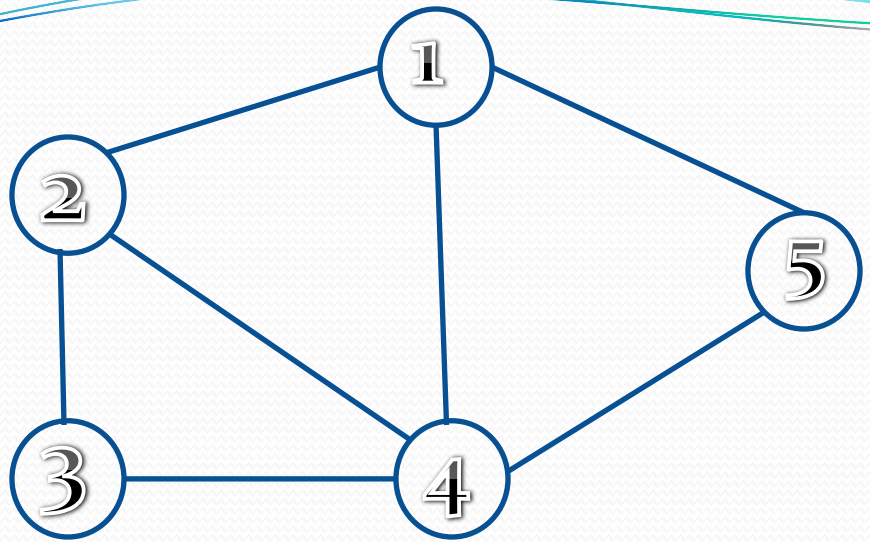

Fig: Graph

Here,

V(G)={1, 2, 3, 4, 5}

E(G)={(1,2),(1,4),(1,5),(2,3),(2,4),(3,4),(4,5)}

Here, the edge incident with node 1 and node 2 is written as (1,2); we could also have written (2,1) instead of (1,2). This is same for all the edges. In this case, the ordering of the vertices is not significant. This is the case of a *directed graph*.

Types of Graphs

1. ***Undirected Graph***: In an undirected graph, the pair of vertices representing any edge is unordered. Thus ***(v, w)*** and ***(w, v)*** represent the same edge.
2. ***Directed Graph***: In a directed graph, each edge is an ordered pair of vertices i.e., each edge is represented by a directed pair. If ***e=(v, w)***, then ***v*** is initial vertex and ***w*** is the final vertex. Thus ***(v, w)*** and ***(w, v)*** represent two different edges.

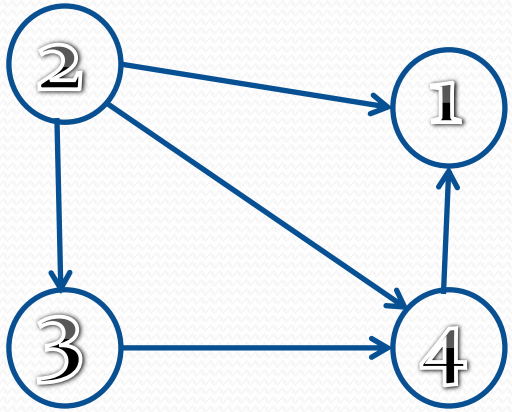

Fig: Directed Graph

TU Exam Question (2066)

- Differentiate between tree and graph. What are spanning forest and spanning tree. Explain MST (Minimum Cost Spanning Tree) problem.

Model Question (2008)

- Write a non-recursive depth-first traversal algorithm.

TU Exam Question (2065)

- Explain the Kruskal's algorithm.

TU Exam Question (2066)

- "To write an efficient program, we should know about data structures".

  Explain the above statement.

- **Ans:** If we are using an algorithm manually on some amount of data, we intuitively try to organize the data in a way that minimizes the number of steps that we need to take to search for the data.
- E.g.: Publishers offer dictionaries with the words listed in alphabetical order to minimize the length of time it takes us to look up a word.

- We can do the same thing for algorithms in our computer programs.
- Example: Finding a numeric value in a list.
  - If we assume that the list is unordered, we must search from the beginning to the end.
  - In worst case, we have to search the entire list
  - Algorithm is O(n), where n is size of array.
  - For example: Find a match with *value* in an unordered list

```
int list[6] = {7, 2, 9, 5, 6, 4};
for (int i=0; i<6, i++)
  if (value == list[i])
      printf("value found");
   else
      printf("value not found");
```

- If we assume that the list is ordered, we can still search the entire list from the beginning to the end to determine if we have a match.
- But, *we do not need to search that way*.
- Because the values are in numerical order, we can use a *binary search algorithm*.
- Algorithm is $O(\log_2 n)$, where n is size of array.

- Find a match with `value` in an ordered list

```
int list[6] = {2, 4, 5, 6, 7, 9};
int min = 0, max = list.length-1;
while (min <= max)
{
    if (value == list[(min+max)/2])
        statement;  // found it
    else
        if (value < list[(min+max)/2])
            max = (min+max)/2 - 1;
        else
            min = (min+max)/2 + 1;
}
statement;          // didn't find it
```

- The difference in the structure of the data between an unordered list and an ordered list can be used to reduce the algorithm complexity.
- This is the role of data structures and why we study them.
- We need to be as clever in organizing our data efficiently as we are in figuring out an algorithm for processing it efficiently
- It may take more time to complete one iteration of the second loop (due to more code in the body of the loop), but the growth rate of time taken with increasing size of the array is less.
- As *n* gets large, the growth rate eventually dominates the resulting time taken.
- That is why we ignore multiplication by or addition of constants in Big-O notation.