Graphs

Introduction to Graphs

- Suppose you want to go to Bombay from Jaipur by bus with minimum number of transfers.
- You can do that by:
- Modelling the problem as a graph
- Solve the problem using BFS

Some real life arrangements

- A LAN system
- Internet
- Railway network
- Telephone network
- Road connections
- Bus service
- Air network
- Electric network

Graph: a data structure to solve many real-life problems??

- A graph consists of a set V of vertices (nodes) and a set E of edges (arcs)
- It is denoted as G = (V,E)
- Directed graph: A graph is called directed if the pair of nodes that make up the arcs are ordered pairs <A,B>,
- The head of each arrow represents the second node in the ordered pair and tail represents the first node in the pair.
- Directed graph = {<A,B>, <A,C>, <B,D>}
- Unordered pairs or undirected graph is represented as: {(A,B),(A,C),(B,D))
- You would have already studied graphs in mathematics, graph is basically a relation

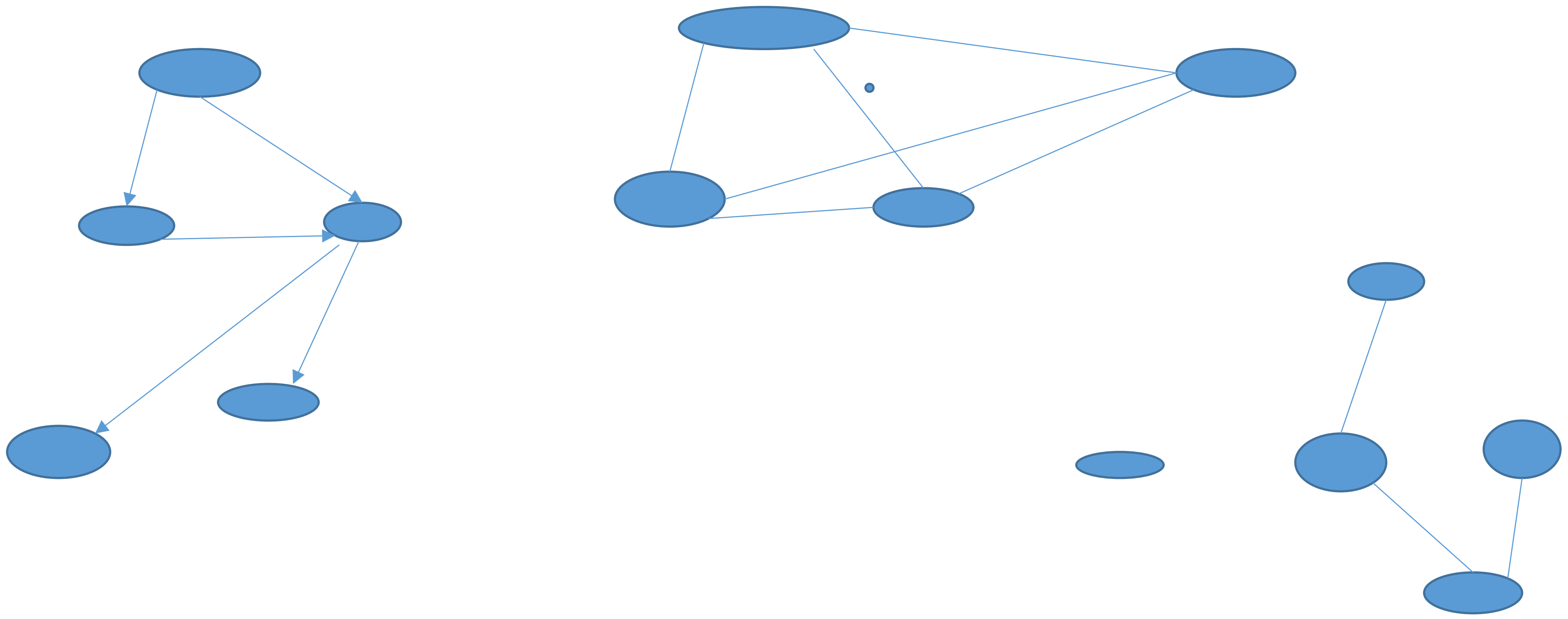
Cont..

- Graph represents a relation among different objects of similar types
- For Example: We have a set of cities
- Define a relation “connected”: a city x is “connected” to city y if there is a road between them
- Define a relation “connected”: a country x is “connected” to country y if there is a flight communication between them
- A student is related to another student if they belong to same batch or same section or same branch

Terminology related to graph

- Incident: A node n is incident to an arc if n is one of the two nodes in the arc
- Degree: Degree of a node is no. of arcs incident to it
- Indegree: Indegree of a node n is no. of arcs that have n as a head
- Outdegree: Outdegree of a node n is no. of arcs that have n as a tail
- Adjacent: A node n is adjacent to node m if there is an edge from m to n, n is called successor of m and m is called predecessor of n
- In some situations, there may be a number associated with the arc which is called weight of the arc.
- Path: A path of length k from node a to node b is defined as sequence of k+1 nodes such that n^1 = a and n^{K+1} = b and adjacent(n^i,n^{i+1}) is true for all i between 1 and k.

Example



Example

- **Cyclic path**: A path from a node to itself is called cycle
- **If a graph contains a cycle, it is called cyclic otherwise it is called acyclic**
- **Connected graph**: A graph is called connected if there exists a path from any vertex to any vertex.

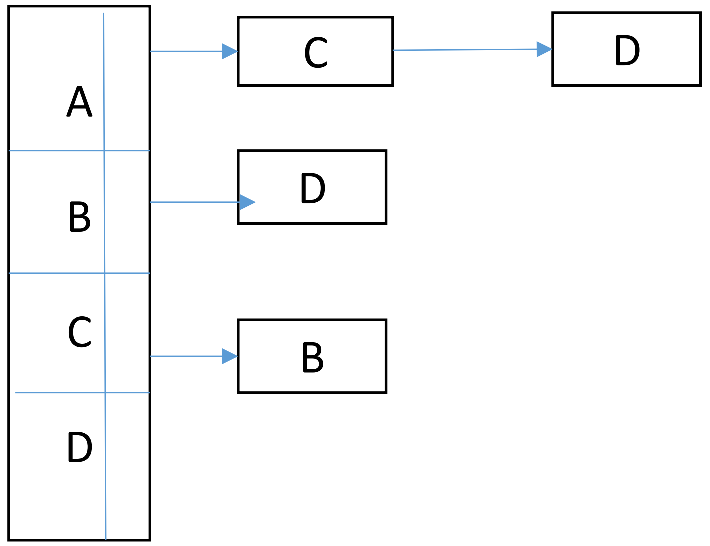
Graph representation(Using table)

- **Adjacency Matrix**: The adjacency matrix A for a graph G = (V, E) with n vertices is an nxn matrix of bits such that:
- Aij = 1 if there is an edge from vi to vj
- Aij = 0 if there is not an edge from vi to vj

- Note: obviously we will have the properties of node declared
- int s_graph[50][50];
- struct student s[50];
- struct student{ int rollno; char name[20], int age;};

Adjacency list representation

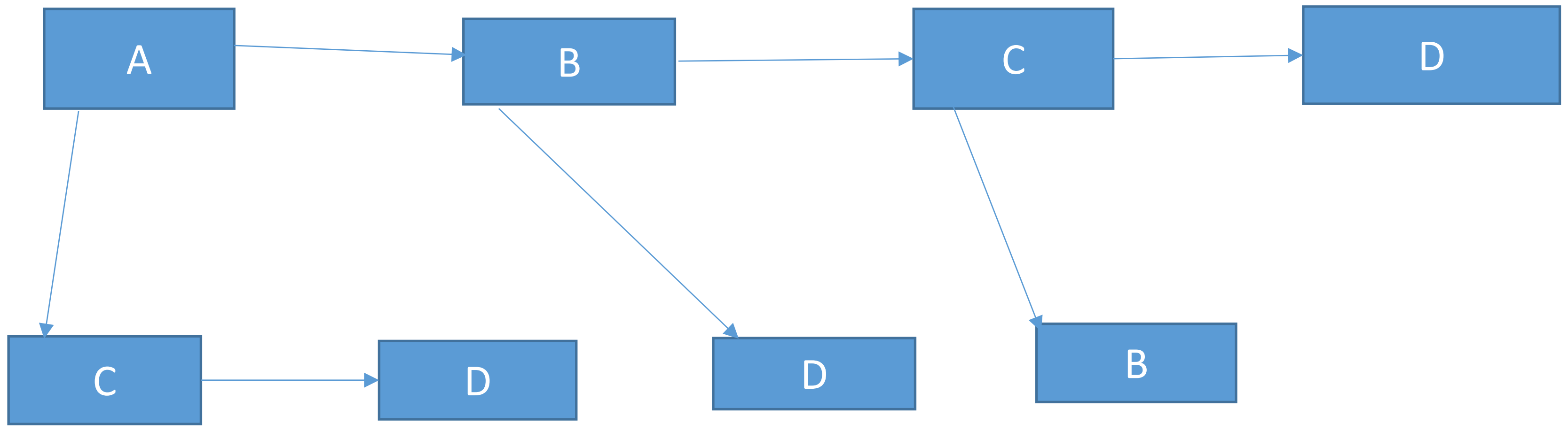
- In this case, graph is stored as a linked structure.
- We store all the vertices in an array and then for each vertex, we form a linked list of its adjacent vertices



- Struct graphnode{ char node_data; struct graphnode *next;}

Struct graphnode G[10]

Linked representation of graph



Graph Traversal

- Traversing means visiting each node in some systematic way
- The elements of the graph to be visited are generally the nodes of graph.
- It is always possible to traverse the graph nodes efficiently in an implementation dependent manner (Adjacency matrix, Adjacency list or linked rep)
- But, we need a traversal that corresponds to the graph structure.
- But defining a traversal that relates to the structure of graph is more complex than tree or any other data structure due to three reasons:

Problems with traversal

- There is no ‘root node’ or first node of graph
- After deciding the first node, only those nodes which are reachable from the starting node can be traversed
- There may remain other nodes in the graph that have not been visited because those may not be connected
- This means we may have to select another starting node
- There is no natural order among successors of a particular node.
- Thus there is no apriori order in which the successors of a particular node should be visited.

Cont..

- A node of a graph may have more than one predecessor and therefore, it is possible for a node to be visited before one of its predecessor
- In a tree, we never encounter a node more than once but while traversing a graph, there may be a possibility that we reach a node more than once. So, we need to keep status of each node whether visited or not
- Unlike tree, in graph, for the same traversal technique, we may have more than one traversal

Breadth First traversal or search

- BFT or BFS is a technique that begins at the root (user defined start node) and explores all the neighboring or adjacent nodes or successor nodes. Then for each of those nodes, the algorithm explores all the unexplored nodes of these nodes until all the node are visited or goal node is found.
- This method visits all successors of a visited node before visiting any successors of any of those successors.

BFS/BFT algorithm

A Boolean array is maintained for visited nodes

1. Choose the starting node
2. Visit/take all the nodes adjacent to that starting node
3. Continue the same, for all other adjacent nodes which are adjacent to starting node
4. Maintain the status of vising nodes
5. Suppose V0 is the starting node and V11, V12, V13 are the adjacent nodes of V0….

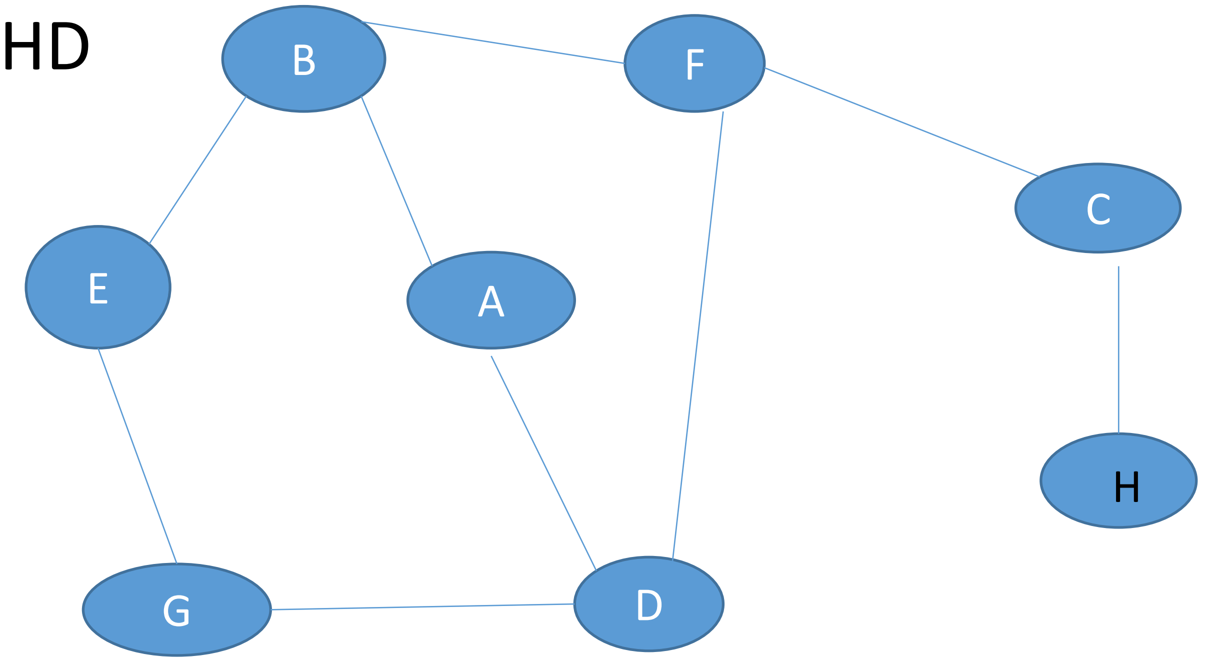
BFS through Queue

1. Insert starting node in the queue
2. Delete an element from queue and insert all the successors of starting node in the queue

- Repeat step 2 until queue is empty

Example

- BFS: ABDGEFCH
- DFS: ABEGFCHD



BFS

- Start with A
- Mark it visited and put the node A in resultant string R (R: A)
- Put all the adjacent nodes of A into a queue
- Now delete a node from queue, mark it visited and add to R(say B)
- Put all the adjacent node of B into queue
- Continue with this process till all the nodes are visited or

Depth First traversal or search

- The depth first traversal or search progresses by expanding the starting node of G and then going deeper and deeper until goal node is found or until a node is encountered which has no successor
- DFS begins with a starting node A, then it examines nodes along a path P which begins at A. This means first neighbor of A is processed then a neighbor of neighbor of A is processed and so on.
- The algorithm proceeds like this until we reach dead-end.
- On reaching the dead end, we backtrack to find another path P’
- A stack is used to implement depth first traversal

DFS algorithm

A Boolean array is maintained for visited nodes

1. Choose the starting node
2. Visit one of the adjacent node to that starting node
3. Continue the same, for all other adjacent node which are adjacent to this node
4. Keep moving till a dead-end is reached
5. Suppose V0 is the starting node and V11, V12, V13 are the adjacent nodes of V0….

DFS

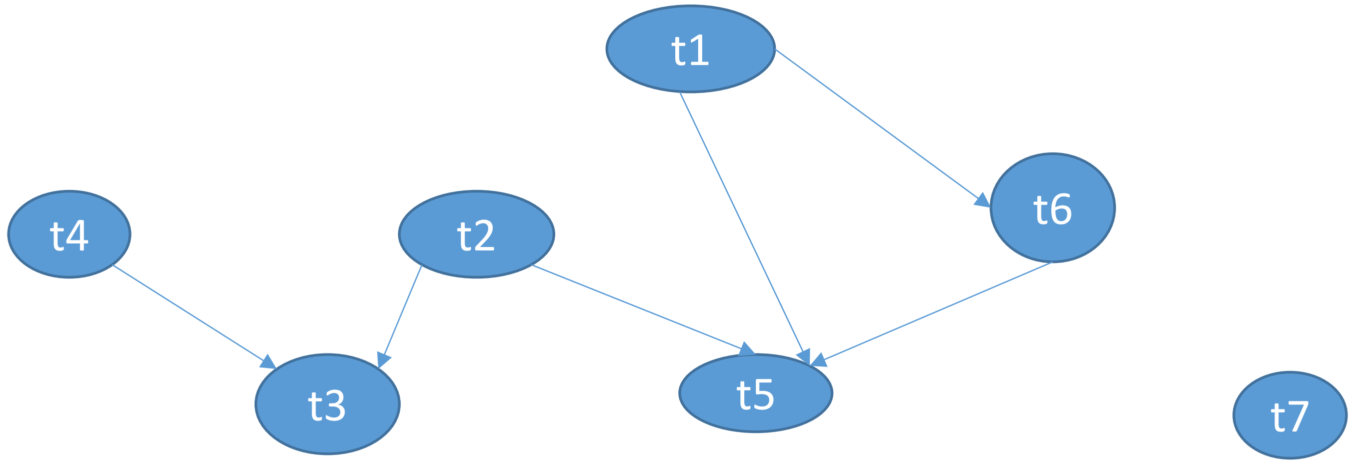
- Start with starting node A
- Put it in the stack, Mark it visited and put in the resultant string R
- Put a node adjacent to this node in stack
- Mark it visited and put in the resultant string R
- Then put adjacent node of this new node, if all the nodes already visited then pop it
- Continue with the same process till the stack is empty

Topological sorting

- How to solve the problem of course registration if pre-requisites of course have to be followed strictly
- how to schedule jobs if 7 jobs have to be done to complete a task and there is dependency among these jobs

Topological Sorting

- Let this graph shows the dependencies…



Topological Sorting

- If G is a directed graph with no directed cycles i.e. graph is directed and acyclic then topological ordering T of G is a sequential listing of all vertices such that for all u,v belonging to G, u precedes v in topological ordering.
- Thus topological ordering is a linear ordering of nodes of G such that if(u,v) is a edge then u always appears before v.

Algorithm to find Topological Sorting

- Find the in-degree of each vertex v in G
- Insert all vertices with zero in-degree in a queue
- Repeat 4 and 5 till queue is empty
- Remove front vertex from queue
- For each neighbor w of vertex v do:
- (a) decrease the in-degree of w by 1 and remove edge from v to w
- (b) If in-degree of w is zero, then add w to the rear of queue