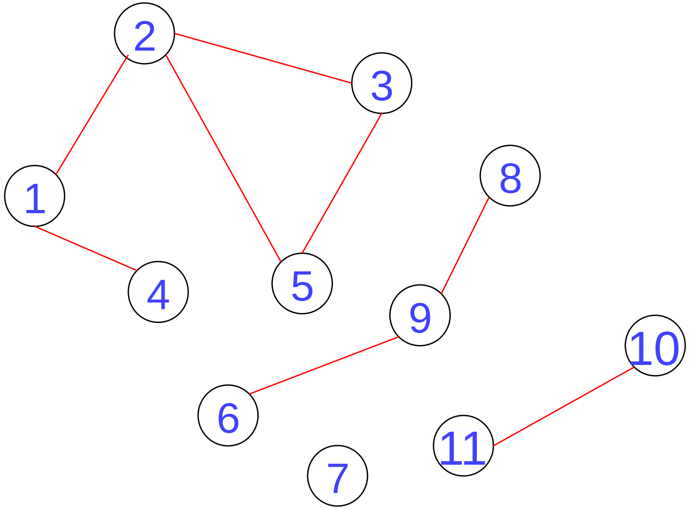
Graph Algorithms: Breadth-First Search (BFS)

Dr. Shrutilipi Bhattacharjee, Assistant Professor, NIT Karnataka, India

Graph Searching

- A vertex ***v*** is reachable from vertex ***u*** iff there is a path from ***u*** to ***v***
- A search method starts at a given vertex ***u*** and visits/labels/marks every vertex that is reachable from ***v***



<u>Graph Search Methods</u>

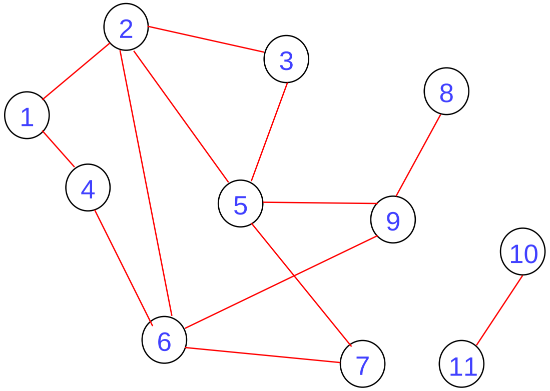
- Many graph problems solved using a search method
  - Path from one vertex to another
  - Is the graph connected?
  - Find a spanning tree
  - etc.
- Commonly used search methods:
  - Breadth-first search
  - Depth-first search

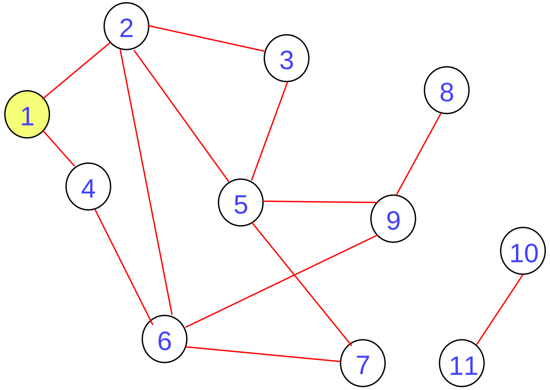
Dispensers

- Graph searches require a data structure that temporarily holds vertices that are neighbors of previously visited vertices, or edges that leave previously visited vertices
- The data structures that are used form a family of abstract data types called ***dispensers***
- A ***dispenser*** is an abstract data type that supports insert, delete, and retrieve operations
- The ***dispenser*** family is characterized by the fact that whenever it is not empty, only one of its data items is accessible for delete or retrieve operations
- Stacks, queues, and priority queues are the best-known abstract data types in the ***dispenser*** family
- They determine the accessible item in a different ways:
  - In a stack, it is the item that was inserted last
  - In a queue, it is the item that was inserted first
  - In a priority queue, it is the item that has the highest priority
    - the priority of an item in a priority queue may be specified as a parameter when the item is inserted, or it may be determined from the item's data contents

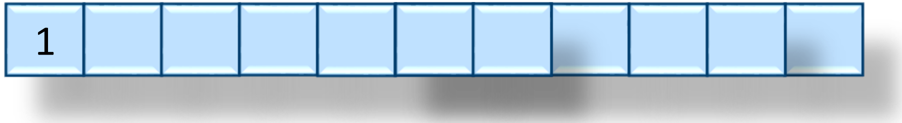
Breadth-first Search (BFS)

- Visit start vertex and put into a ***FIFO queue***
- Repeatedly remove a vertex from the queue, visit its unvisited adjacent vertices, put newly visited vertices into the queue
- Start search at vertex ***1***





Visit/mark/label start vertex and put in a FIFO queue ***Q***



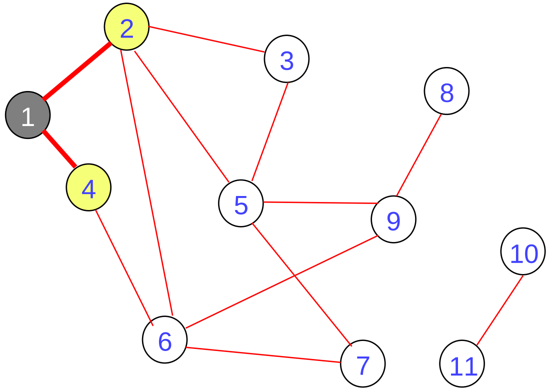
BFS(**G**, **root**)

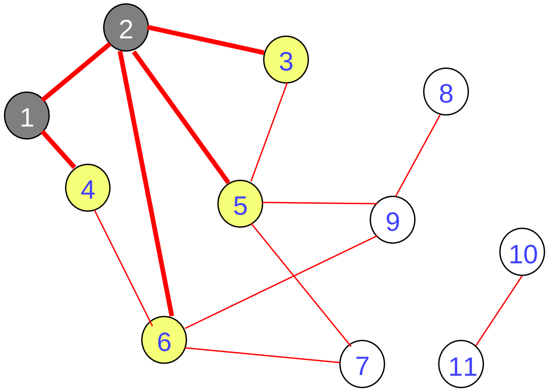
let **Q** be a queue
label root as discovered
Q.enqueue(***root***)
while *Q* is not empty **do**
    v := ***Q***.dequeue()
    if *v* is the goal **then**
        return *v*
    for all edges from ***v*** to ***w*** in G.adjacentEdges(***v***) **do**
        if *w* is not labeled as discovered **then**
            label ***w*** as discovered
            Q.enqueue(***w***)

Output: Goal state. The parent links trace the shortest path back to ***root***

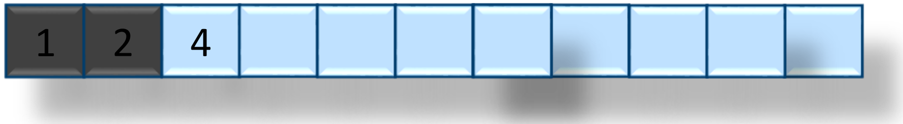
Breadth-first Search (BFS)

- Remove ***1*** from ***Q***, visit adjacent unvisited vertices, put in ***Q***
- Remove ***2*** from ***Q***, visit adjacent unvisited vertices, put in ***Q***



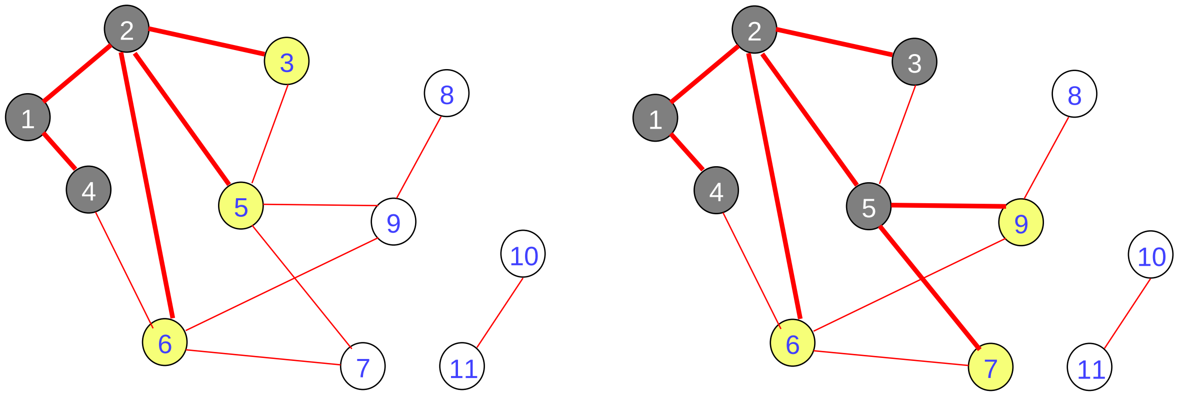


Visit/mark/label start vertex and put in a FIFO queue

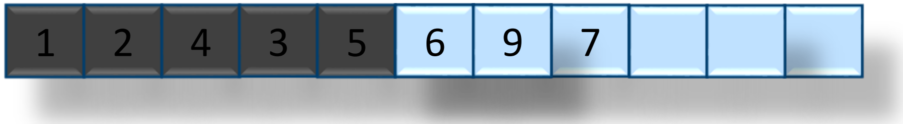


Breadth-first Search (BFS)

- Remove ***4*** from ***Q***, visit adjacent unvisited vertices, put in ***Q***
- Remove ***3*** from ***Q***, visit adjacent unvisited vertices, put in ***Q***
- Remove ***5*** from ***Q***, visit adjacent unvisited vertices, put in ***Q***

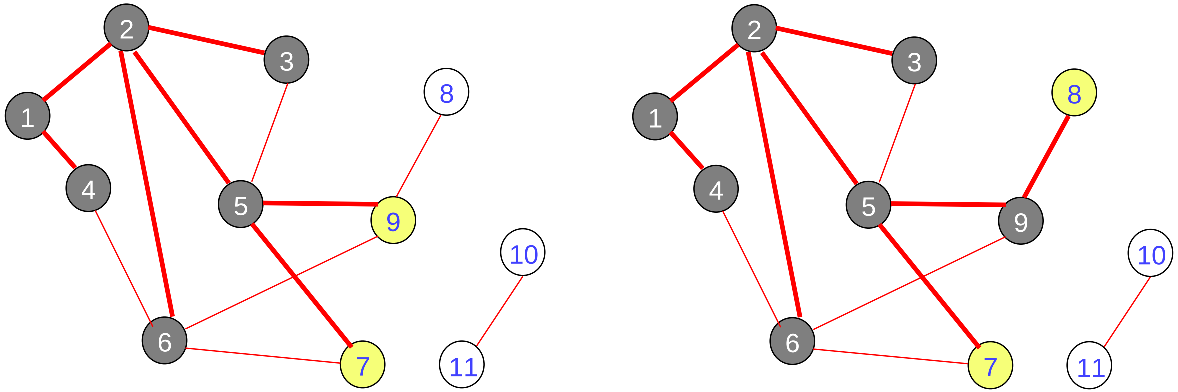


Visit/mark/label start vertex and put in a FIFO queue



Breadth-first Search (BFS)

- Remove ***6*** from ***Q***, visit adjacent unvisited vertices, put in ***Q***
- Remove ***9*** from ***Q***, visit adjacent unvisited vertices, put in ***Q***
- Remove ***7*** from ***Q***, visit adjacent unvisited vertices, put in ***Q***

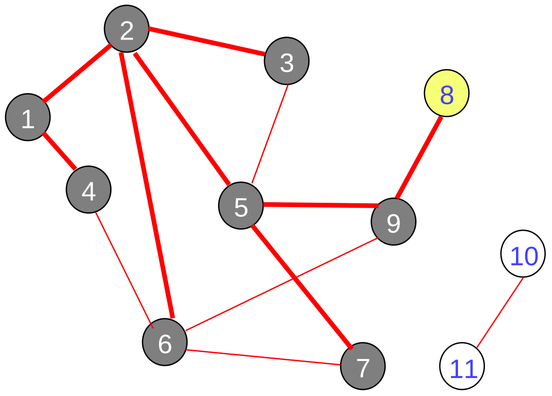


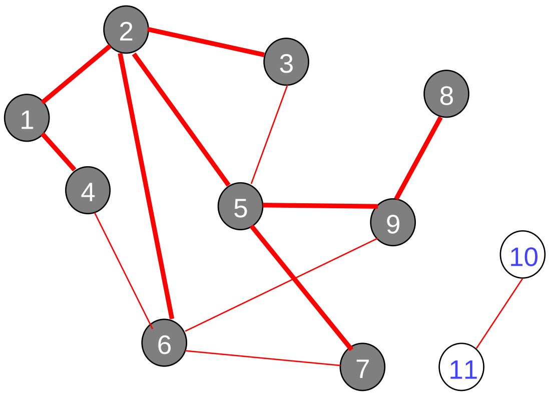
Visit/mark/label start vertex and put in a FIFO queue

1	2	4	3	5	6	9	7	8		

Breadth-first Search (BFS)

- Remove ***8*** from ***Q***, visit adjacent unvisited vertices, put in ***Q***
- Return to the invoking method

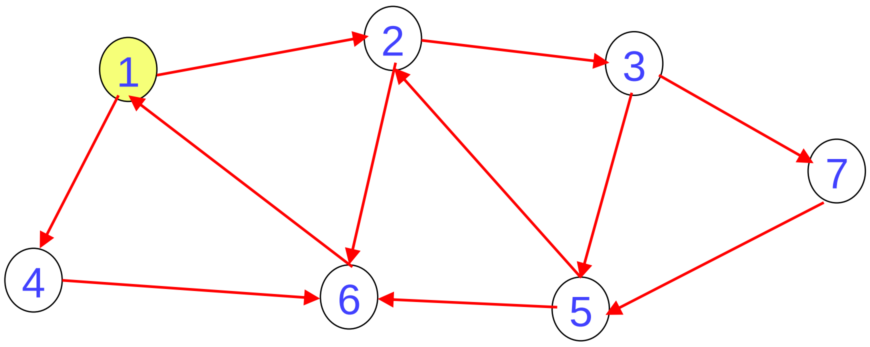


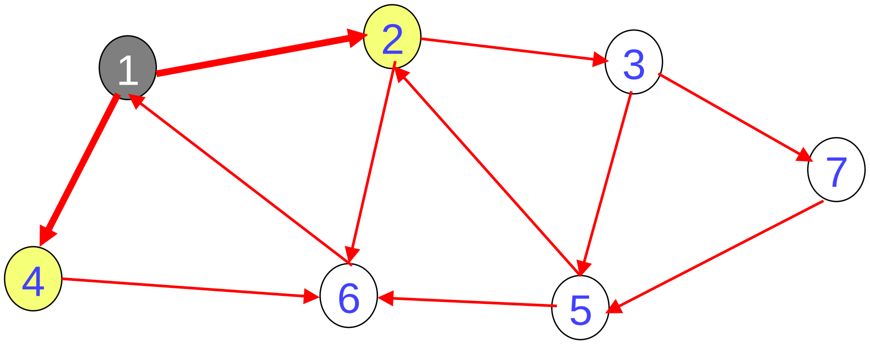


Visit/mark/label start vertex and put in a FIFO queue

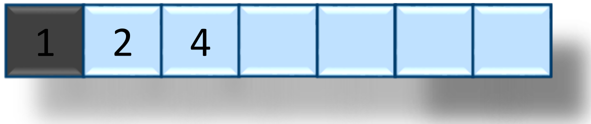
Breadth-first Search (BFS)

- Start search at vertex ***1***
- Remove ***1*** from ***Q***, visit adjacent unvisited vertices, put in ***Q***



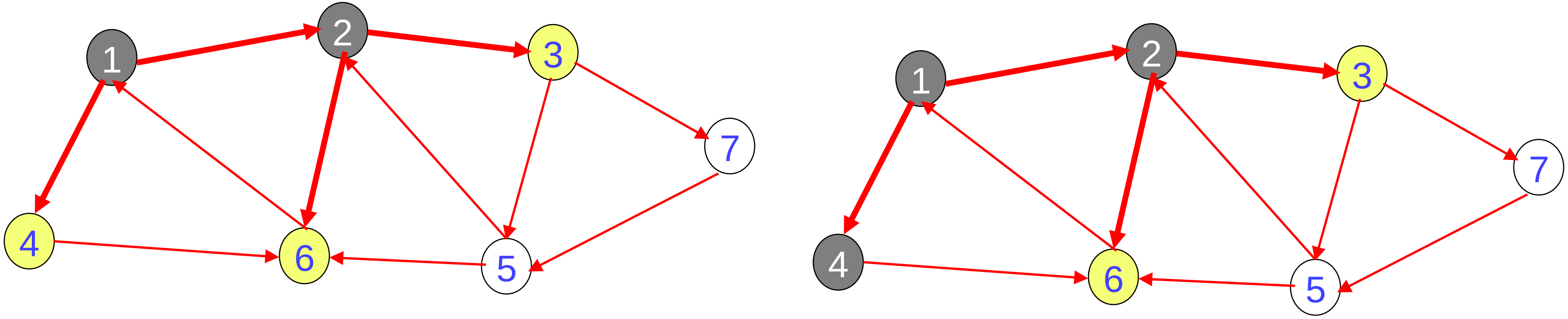


Visit/mark/label start vertex and put in a FIFO queue



Breadth-first Search (BFS)

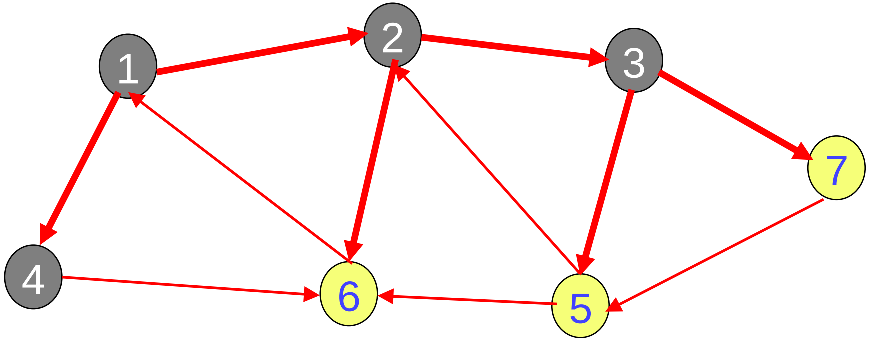
- Remove ***2*** from ***Q***, visit adjacent unvisited vertices, put in ***Q***
- Remove ***4*** from ***Q***, visit adjacent unvisited vertices, put in ***Q***

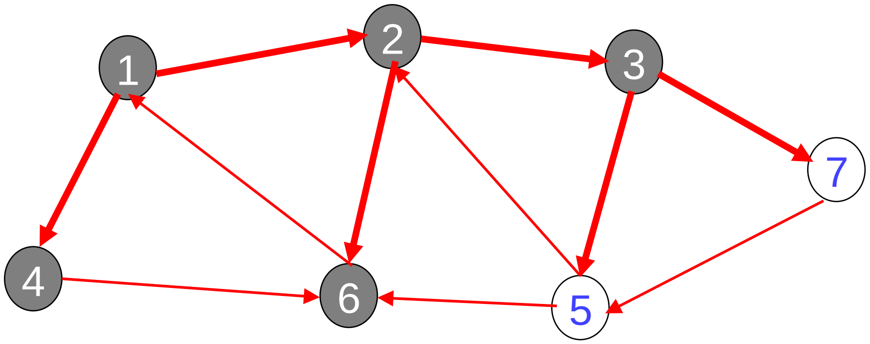


Visit/mark/label start vertex and put in a FIFO queue

Breadth-first Search (BFS)

- Remove ***3*** from ***Q***, visit adjacent unvisited vertices, put in ***Q***
- Remove ***6*** from ***Q***, visit adjacent unvisited vertices, put in ***Q***



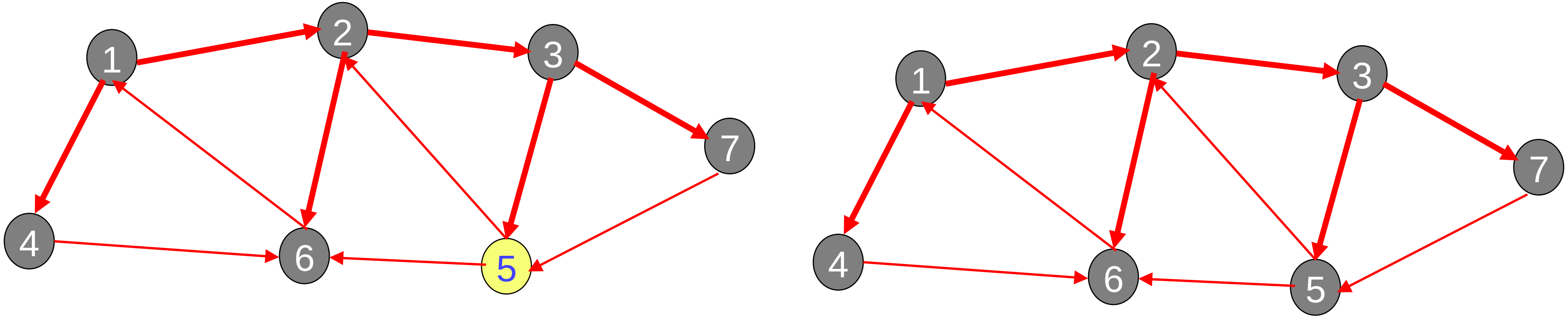


Visit/mark/label start vertex and put in a FIFO queue

1	2	4	3	6	7	5

Breadth-first Search (BFS)

- Remove ***7*** from ***Q***, visit adjacent unvisited vertices, put in ***Q***
- Remove ***5*** from ***Q***, visit adjacent unvisited vertices, put in ***Q***



Visit/mark/label start vertex and put in a FIFO queue

1	2	4	3	6	7	5

BFS Properties and Complexity

- All vertices reachable from the start vertex (including the start vertex) are visited

Time Complexity

- Each visited vertex is put on (and so removed from) the queue exactly once
- When a vertex is removed from the queue, we examine its adjacent vertices
    - ***O(N)*** if adjacency matrix used
    - ***O(vertex degree)*** if adjacency lists used
- Total time
    - ***O(MN)***, where ***M*** is number of vertices in the component that is searched (adjacency matrix)
    - ***O(N + sum of component vertex degrees)*** (adj. lists) = ***O(N + number of edges in component)***

Applications of BFS

- Shortest Path and Minimum Spanning Tree for unweighted graph
- Peer to Peer Networks
- Crawlers in Search Engines
- Social Networking Websites
- GPS Navigation systems
- Broadcasting in Network
- In Garbage Collection
- Cycle detection in undirected graph
- Ford–Fulkerson algorithm
- To test if a graph is Bipartite
- Path Finding
- Finding all nodes within one connected component

Application: Path from Vertex ***u*** to Vertex ***v***

- Start a breadth-first search at vertex ***u***
- Terminate when vertex ***v*** is visited or when ***Q*** becomes empty (whichever occurs first)
- Time
  - ***O(N²)*** when adjacency matrix used
  - ***O(N + E)*** when adjacency lists used (***E*** is number of edges)

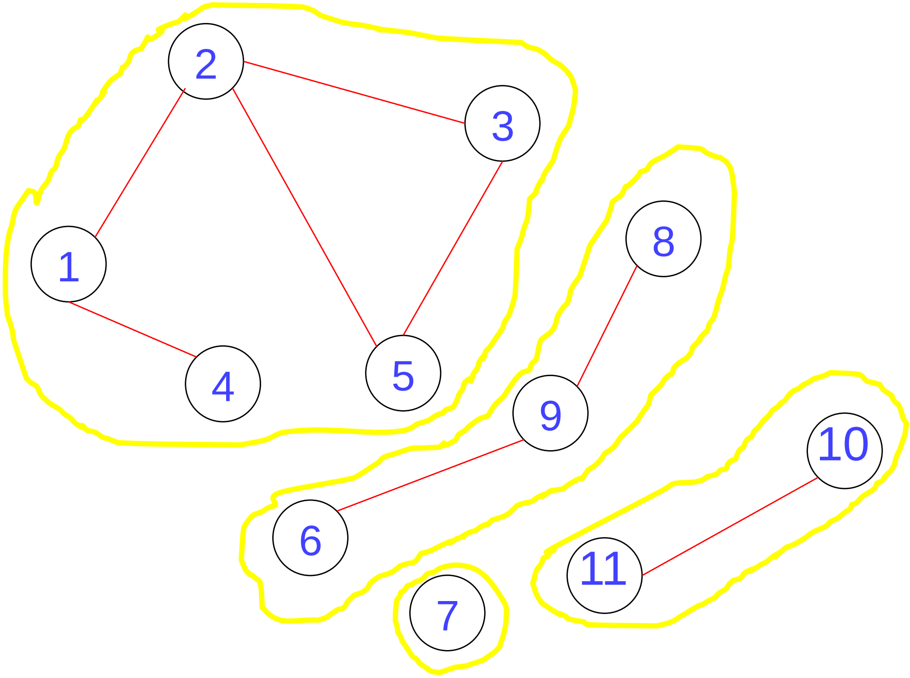
Application: Is The Graph Connected?

- Start a breadth-first search at any vertex of the graph
- Graph is connected iff all ***N*** vertices get visited
- Time
  - ***O(N²)*** when adjacency matrix used
  - ***O(N + E)*** when adjacency lists used (***E*** is number of edges)

Connected Components

- Start a breadth-first search at any as yet unvisited vertex of the graph
- Newly visited vertices (plus edges between them) define a component
- Repeat until all vertices are visited

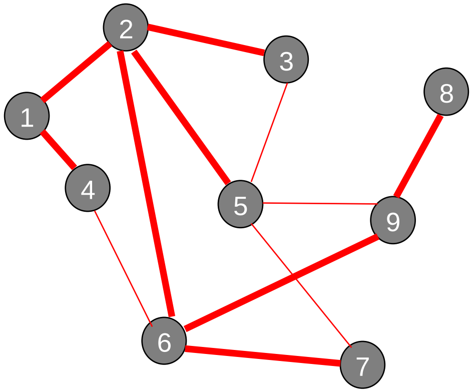
- Time complexity
  - ***O(N²)*** when adjacency matrix used
  - ***O(N + E)*** when adjacency lists used (***E*** is number of edges)



Spanning Tree

- Start a breadth-first search at any vertex of the graph
- If graph is connected, the ***N - 1*** edges used to get to unvisited vertices define a spanning tree (breadth-first spanning tree)

- Time complexity
  - ***O(N²)*** when adjacency matrix used
  - ***O(N + E)*** when adjacency lists used (***E*** is number of edges)



Breadth-first search from vertex ***1***

Breadth-first spanning tree

Next Lecture

Graph Algorithms: Depth-First Search (DFS)

Thank you for your attention...

Any question?

Contact:
Department of Information Technology, NITK Surathkal, India
6th Floor, Room: 13
Phone: +91-9477678768
E-mail: shrutilipi@nitk.edu.in, shrutilipi.bhattacharjee@tum.de