- **Additional data structure**
  - **Queue and Stack**
  - **Linked list**
- **Graph traversal**
  - **Depth first traversal**
  - **Breadth first traversal**

Queue and stack

- In computer science, a priority queue is an abstract data type.
- Each element has a priority associated with it.

  e.g. in software systems, e.g. requests at a web server.
- Important to efficient algorithm design, especially for solving complex problems over graph.
- Since data structure is needed to organise data efficiently.
- Queue and stack are used for implementing priority queue.
- The queue in data structure is similar to queues which happen a lot in real life, e.g. checkout and banks.

Queue

- A queue is a linear data structure that stores items in First In First Out (FIFO) manner.
- Add and remove operations
  - Add is called enqueue, at the tail.
  - Remove is called dequeue, at the head
- Applications
  - serving incoming requests in stores
  - reservation services
  - operating systems for scheduling processes and disks
  - search algorithm

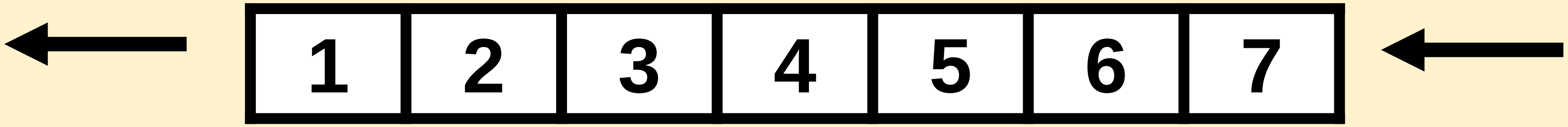
dequeue()

enqueue()

Queue head

Queue tail

Example (queue)

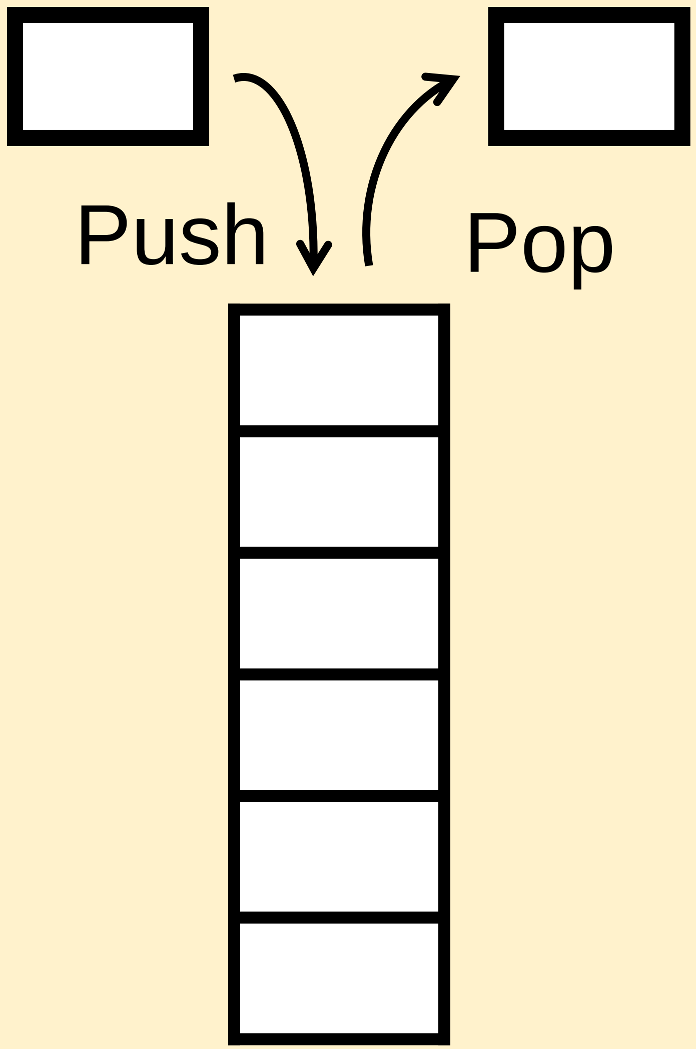


1= dequeue()

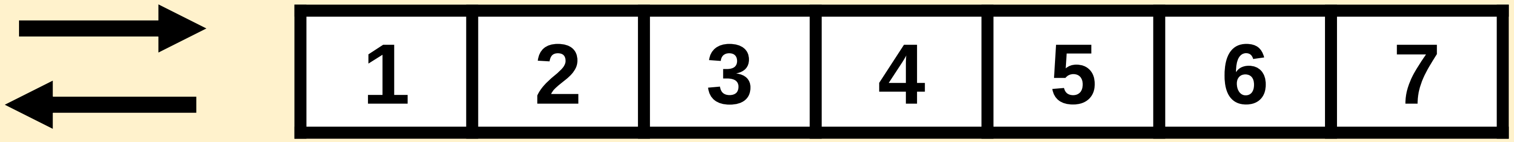
enqueue(8)

Stack

- A stack is a linear data structure that stores items in Last In First Out (LIFO) manner.
- Add and remove operations
  - Add is called push, at the top.
  - Remove is called pop, at the the top
- Has applications
  - Evaluation of expression
  - Backtracking
  - Memory management



Example (stack)

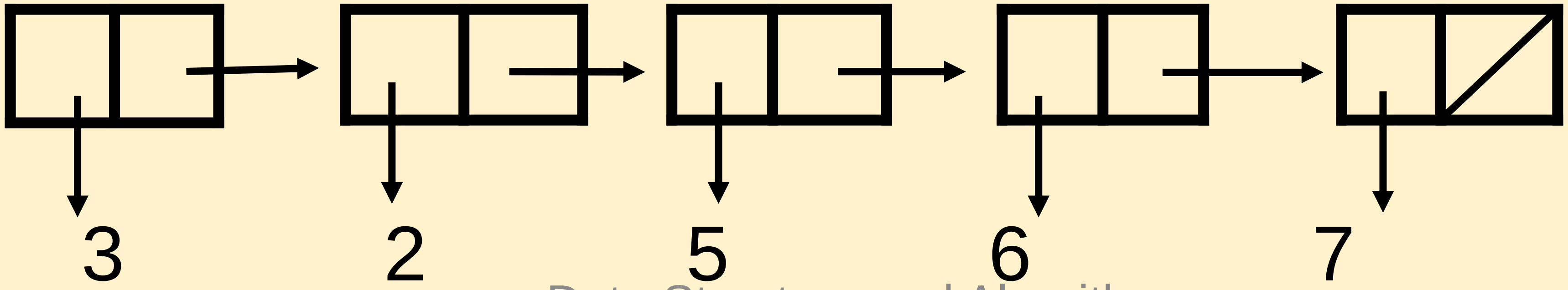


1= pop()

push(8)

Linked list

- Linked list are used in many ways, e.g. in dynamic memory.
- Used for implementing queues and stacks.
- Examples of a list could be
  [3,2,5 ,6,7] or [apple, carrots, milk, cheese]
- Non empty linked lists are represented by two cells.
- The first cell contains a pointer to a list element and the second cell contains a pointer to either the empty list or another two-cell.



Traversal Techniques for Tree and Graphs

- Trees and Graphs are models on which algorithmic solutions for many problems are constructed.
- Traversal: All nodes of a tree/graph are examined/evaluated
- e.g Locate all the neighbours of a vertex *V* in a graph
- Binary Tree traversals
  - Inorder,
  - Postorder,
  - Preorder
- Trees are a special case of a graph
- General Tree has many children at each node

- Graph traversal is a generalization of tree traversal except we have to keep track of the visited vertices.
- Different from search: only a subset of vertices (nodes) are examined.
- Graph traversal techniques
  - Depth–first search (DFS)
  - Breadth–first search (BFS)
- Applications: Strongly connected components, topological sorting, critical path analysis
- They prove the basis of most simple, efficient graph algorithms.

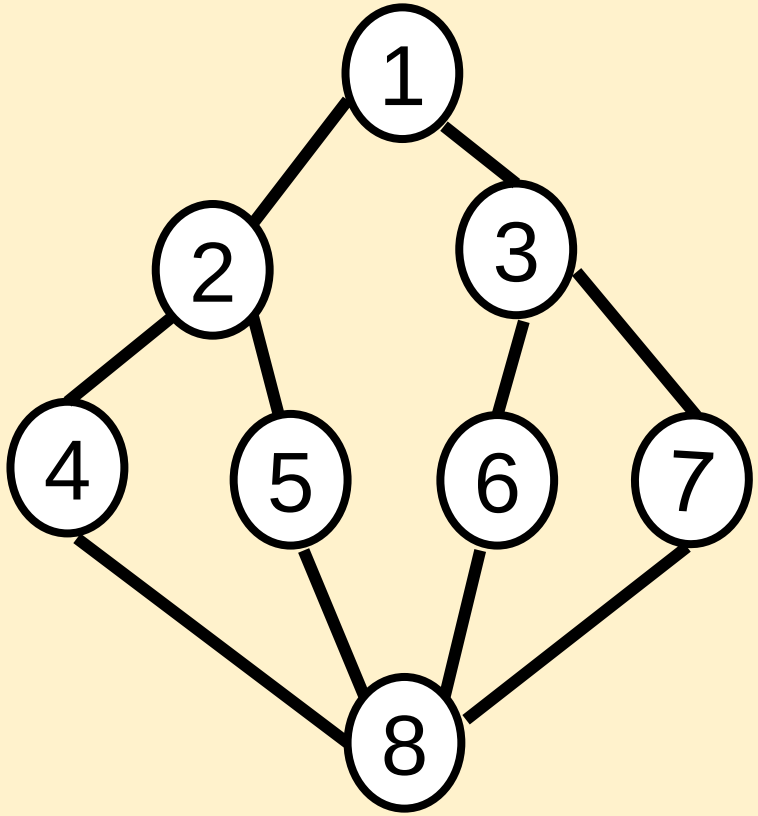
Depth first search (DFS)

- Depth-first search (DFS) is an algorithm for traversing or searching tree or graph data structures.
- starts at the root node
- Can select some arbitrary node as the root node in the case of a graph
- explores as far as possible along each branch before backtracking, after the branch is fully explored.
- This is expressed as a recursion.
- Can be implemented using a stack.

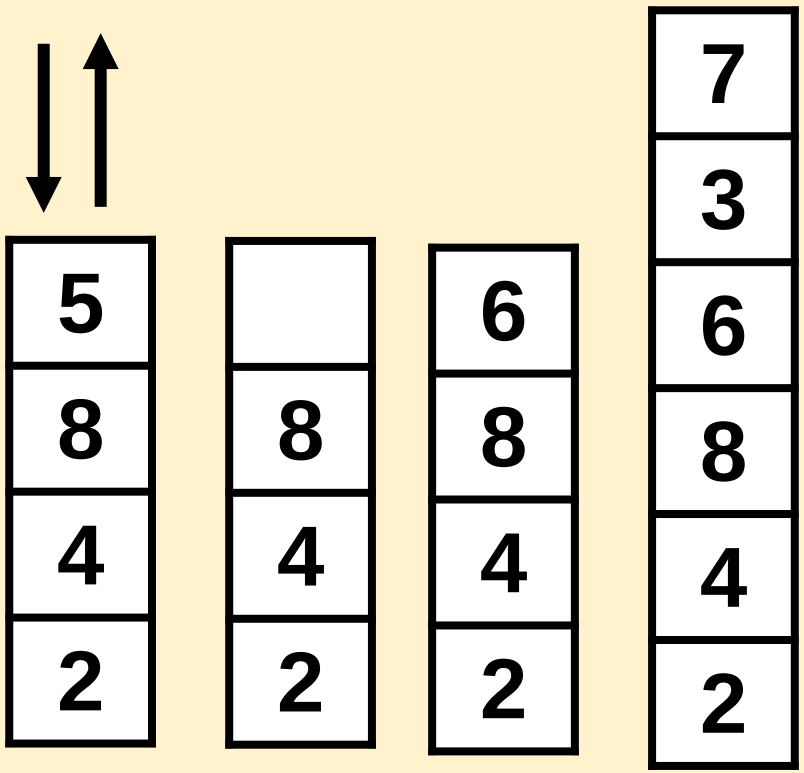
```
Algorithm DFS(v: vertex; G:Graph)
   Visited[v] = 1;
   for all vertices w adjacent to v {
       if(Visited[w] = 0)
      DFS(w; G);
}
```

- *The algorithm conduct a depth first search of G;*
- *starting with vertex v*
- *a vertex i if visited is marked by setting visited[i]=1;*
- *initially all visited[i] =0;*

Example: Depth First Search from vertex 1

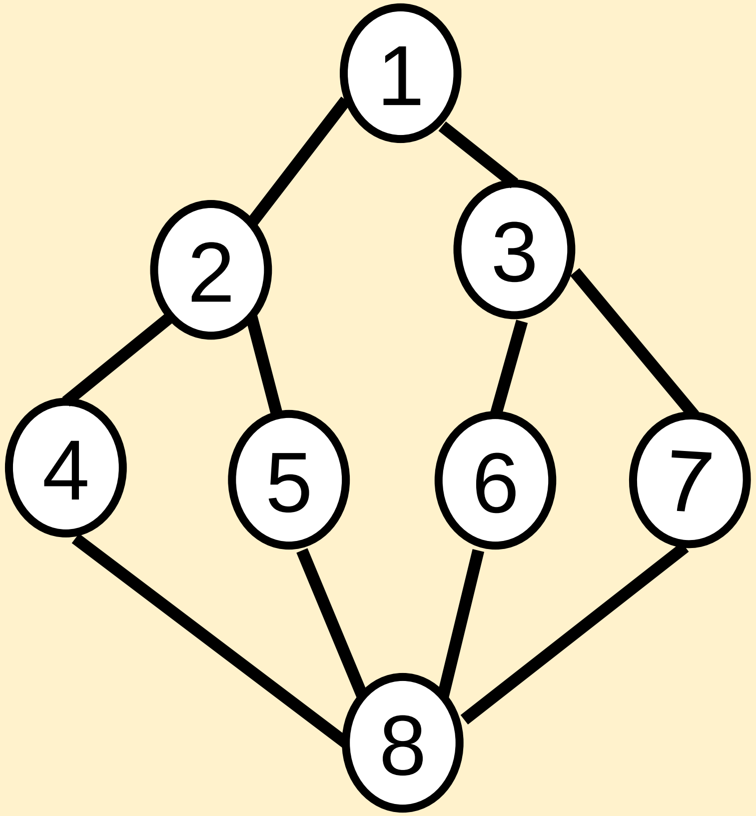


Stack of visited vertices from 1

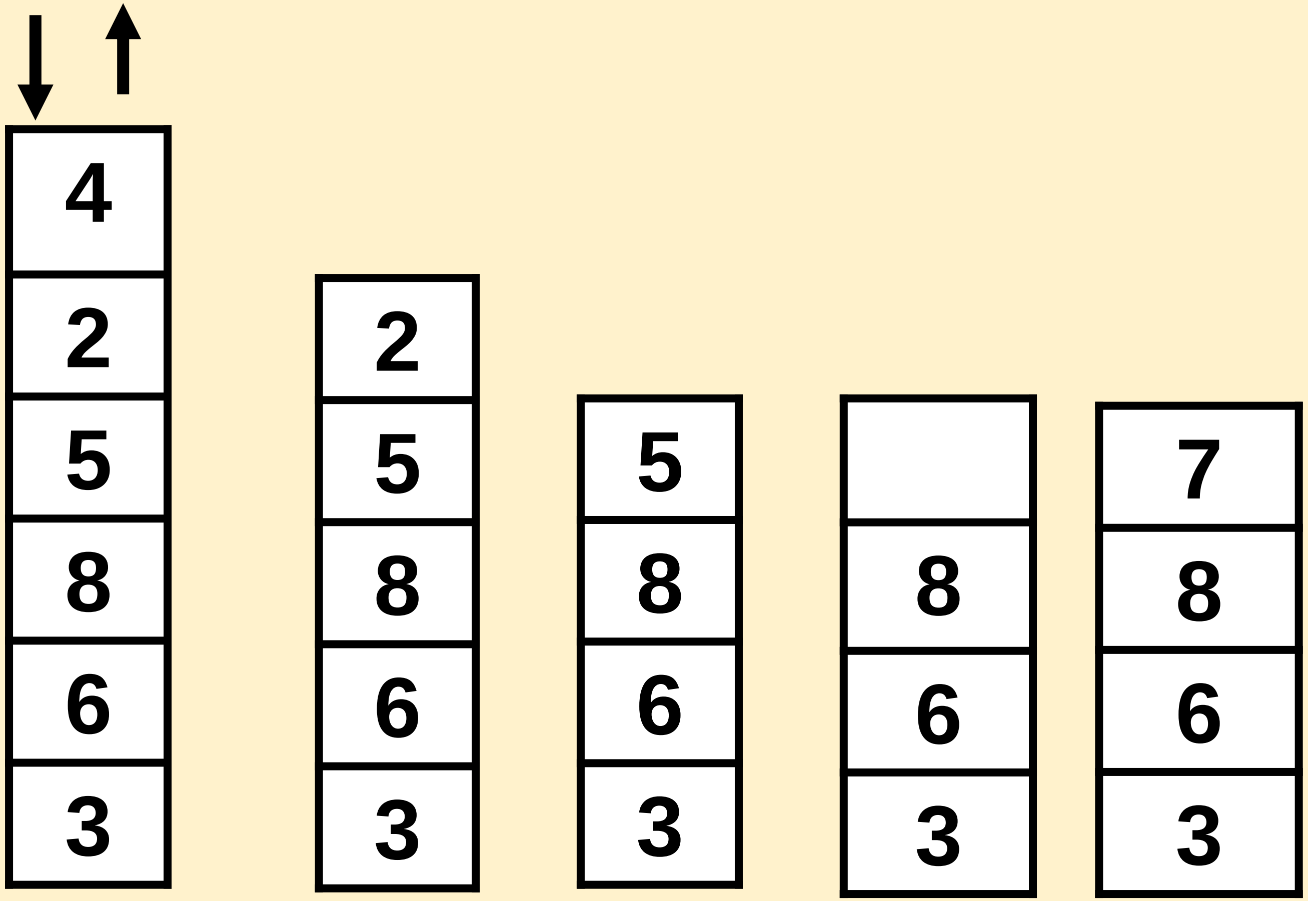


Vertices are visited vertices order 1,2,4,8,5,6,3, 7

Another possibility



Stack of visited vertices from 1



Vertices are visited vertices order:
1,3,6,8,5,2,4,7

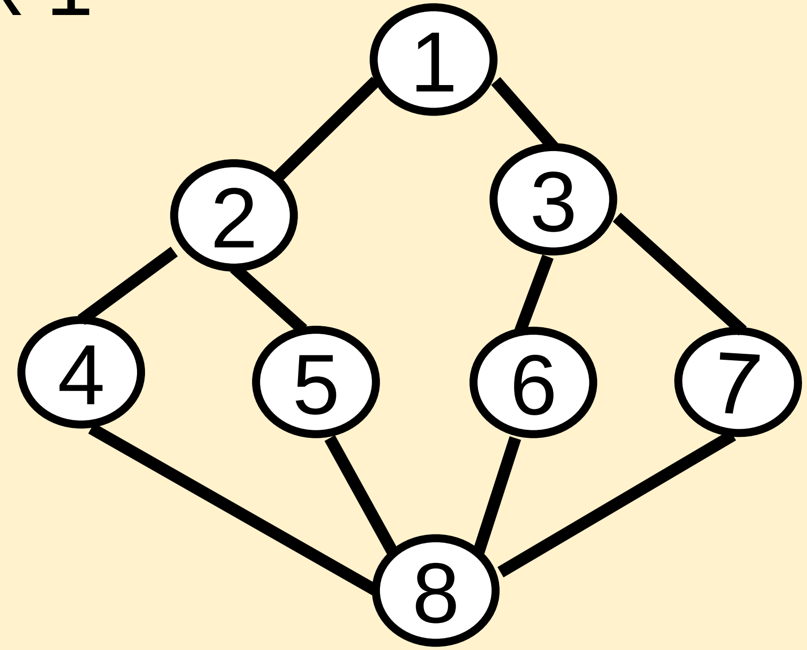
Breadth first search (BFS)

- Start at a vertex v– mark it as reached
- The vertex v is, as yet, unexplored
- When all the vertices adjacent to it (connected by an edge) have been visited, v has been explored(reached).
- Collect all the unvisited vertices adjacent to v and add them to a list.
- Take a vertex from the list and repeat the process
- When there are no vertices left in the list they have all been explored (reached).
- This yields the set of vertices that are “reach-able” from the start vertex v.

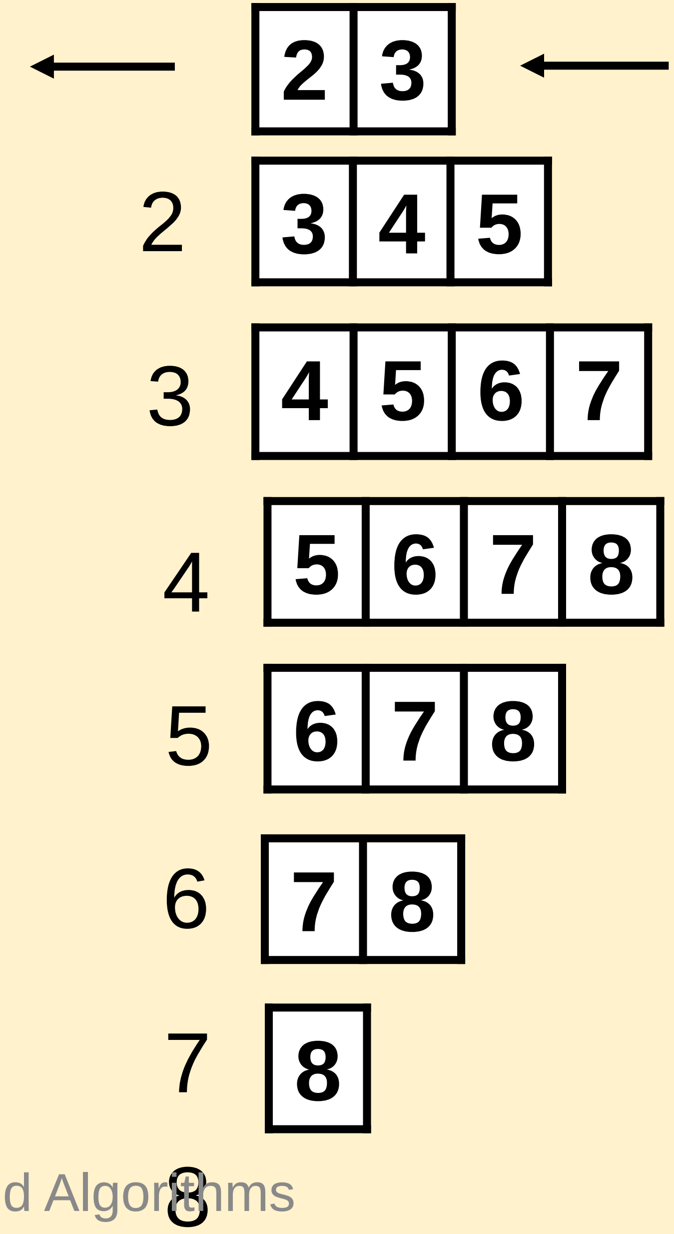
```
Algorithm BFS(v: vertex; G:Graph) {
    visited[v] = 1;
    u=v;
    MakeEmpty(Q);
    Insert(Q; v);
    while(Not IsEmpty(Q)) {
        for all vertices w adjacent to u{
        if(Visited[w] == 0){
         Insert(Q; w);visited[w] = 1;}
         }
   if(IsEmpty(Q)) u=Delete(Q);
   }
}
```

- The algorithm conduct a breadth first search of G;
- starting with vertex v
- a vertex i if visited is marked by setting visited[i]=1;
- initially all visited[i] =0;
- Putting all adjacent nodes into a queue.

Example: Breadth First Search from vertex 1



Queue for tracking visiting order



Vertices are visited vertices order: 1,2,3,4,5,6,7,8

- Graph traversal (continued)
  - Differences between DFS and BFS
  - Applications
    - ✓ Connected components
    - ✓ Spanning trees
- Analysis of BFS

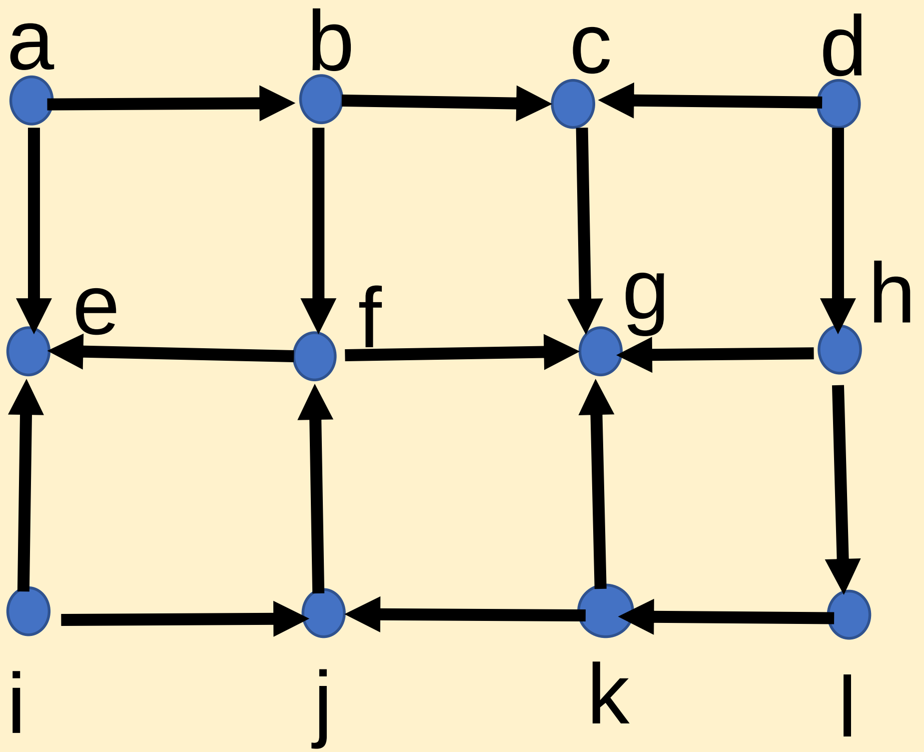
Difference between DFS and BFS

- Breadth first vs. Depth first
  - BFS : a node is fully explored before any other can begin.
  - DFS : exploration of a vertex v is suspended as soon as a new vertex u is reached.
    - ✓ exploration of the new vertex u begins.
    - ✓ exploration of v continues after u has been explored.
    - ✓ can be expressed as a recursive algorithm.
- Both are of $O(n)$ time complexity
- So which one to use and when ?
  - BFS is “better” at finding shortest path in a graph
  - DFS is “better” at answering connectivity queries (Determining if every pair of vertices can be connected by two disjoint paths)

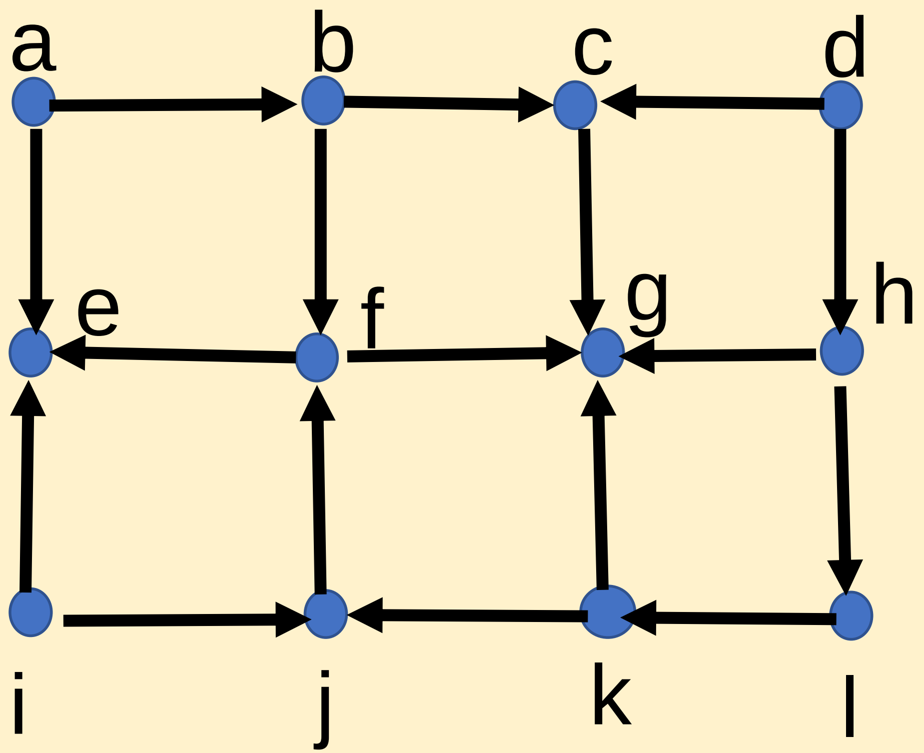
Applications

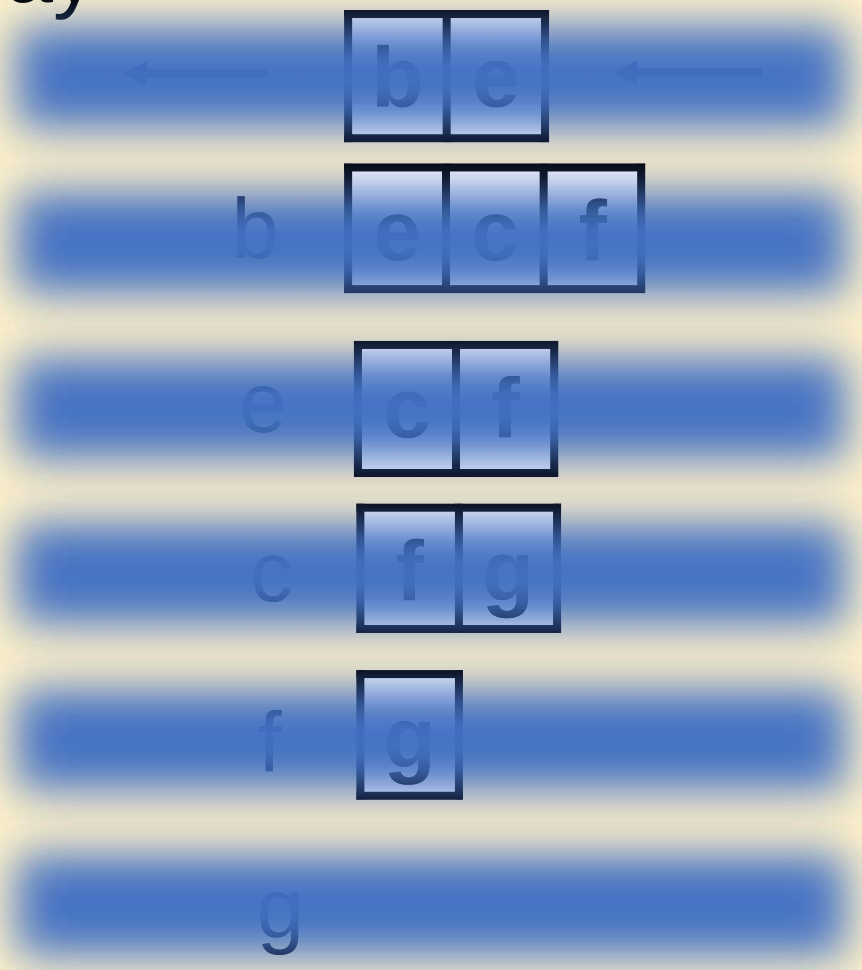
- IP Multicasting
  - Efficiently transmit (same) data to multiple computers.
  - requires that there are no loops in the network of computers and routers.
  - Construct a spanning tree with routers and computers as nodes and links between routers as edges.
- Web Spiders
  - Use a “web graph”
  - A web page is a node.
  - A (directed) edge between two nodes if there is a link in a web page pointing to another.
  - Either DFS or BFS can be used to “crawl the web” (Follow the links until no new links can be found.

Example "Crawler": Trace the “crawl” for the following web graph based on DFS and BFS respectively (exam question type)

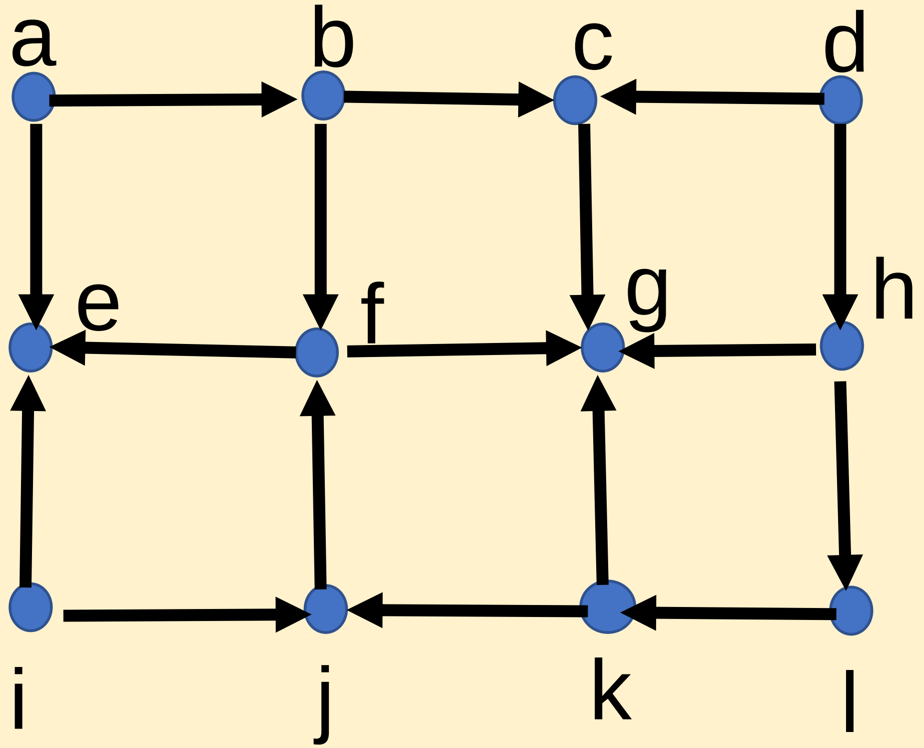


BFS: Pick a starting node, say it's 'a'

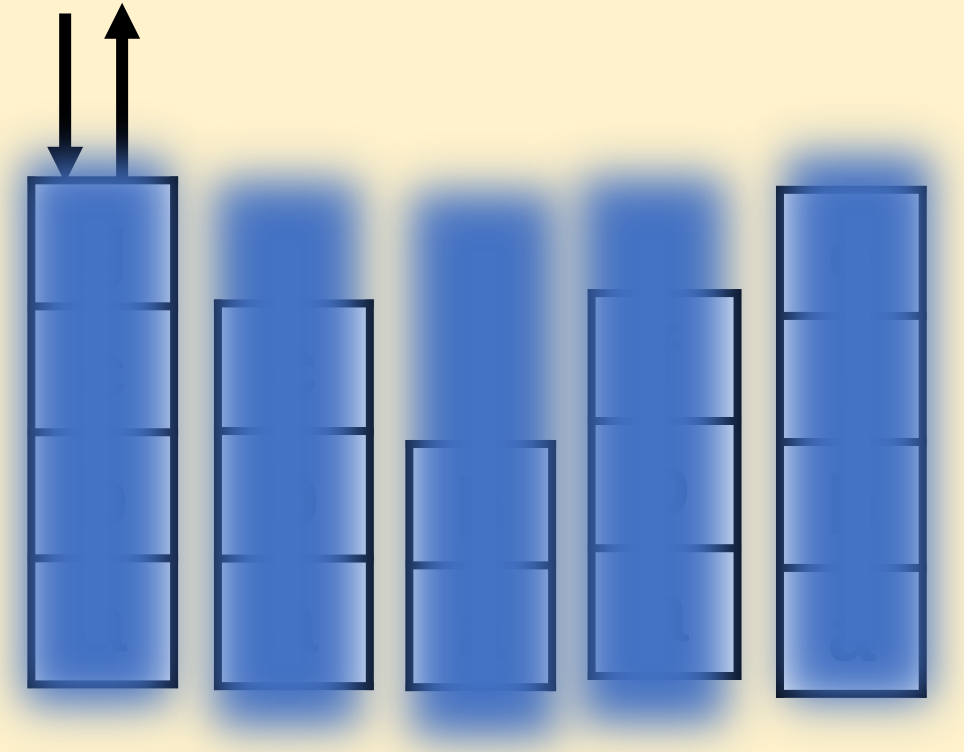




DFS: Pick a starting node, say it's 'a'



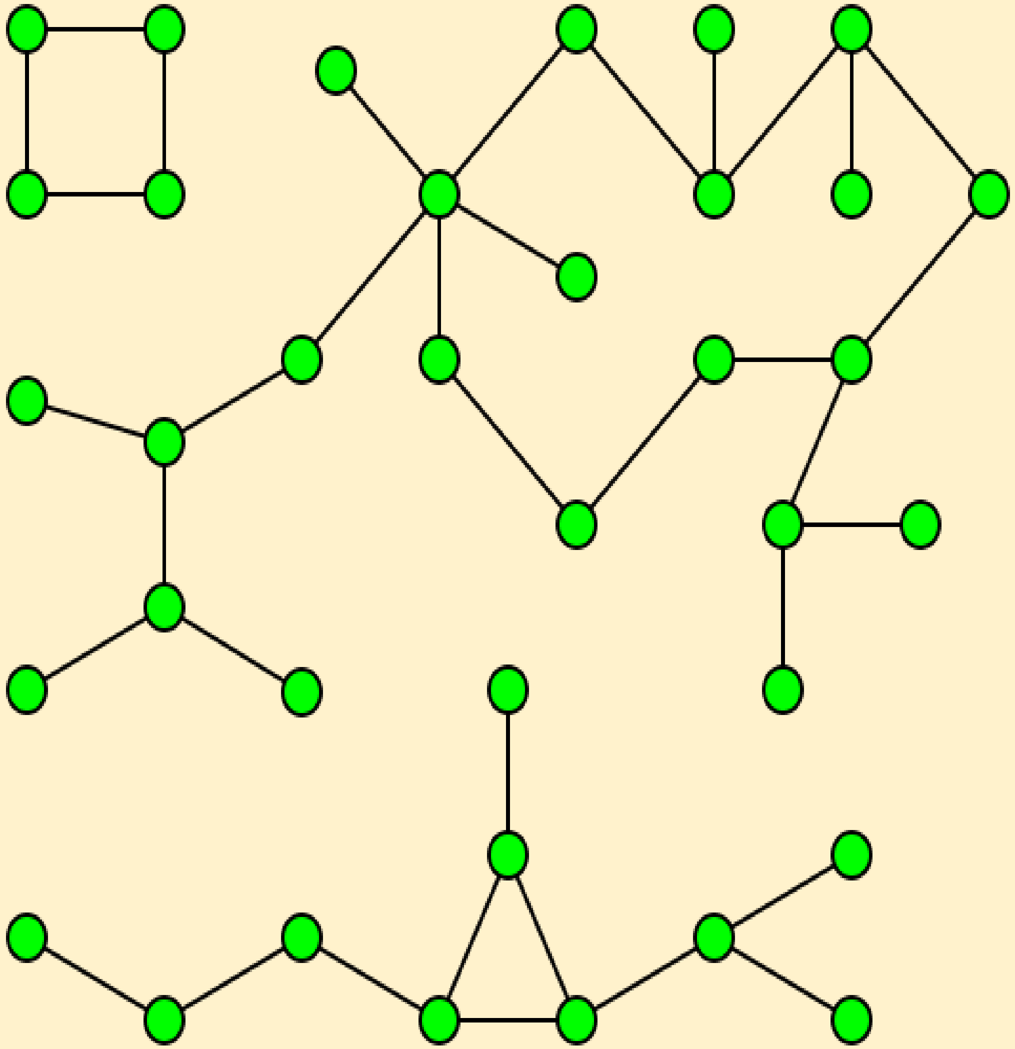
Stack of visited vertices from a



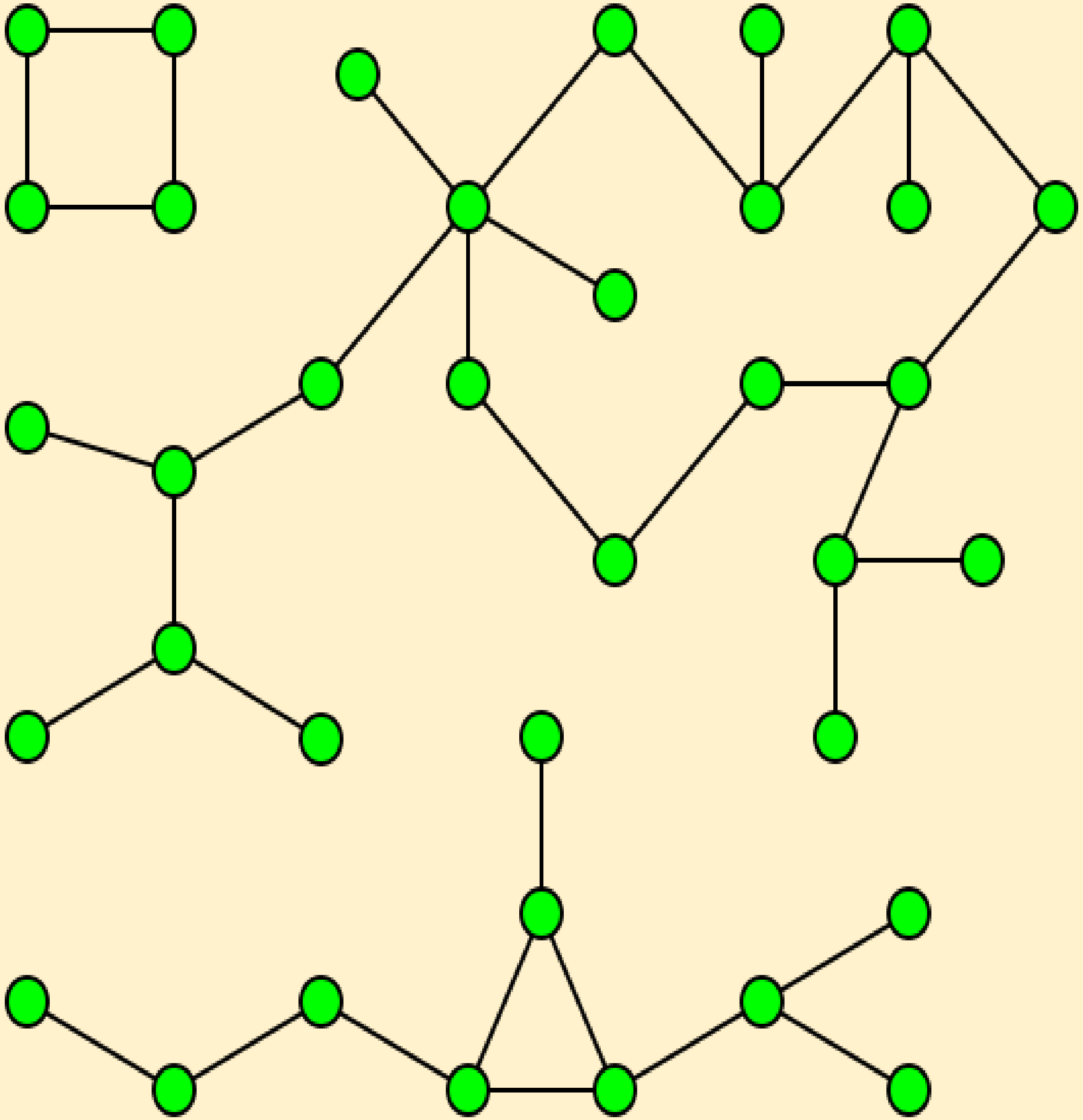
Applications

Connected components

- An undirected graph is connected if for every pair of vertices there exists a path between them.
- A connected component 'C' is a maximal connected subgraph.



- If G is connected undirected graph, then all vertices of G are visited on first call of BFS.
- If G is not connected, then we need at least two calls of BFS.
- An extension of BFS algorithm can be designed to find all the connected components



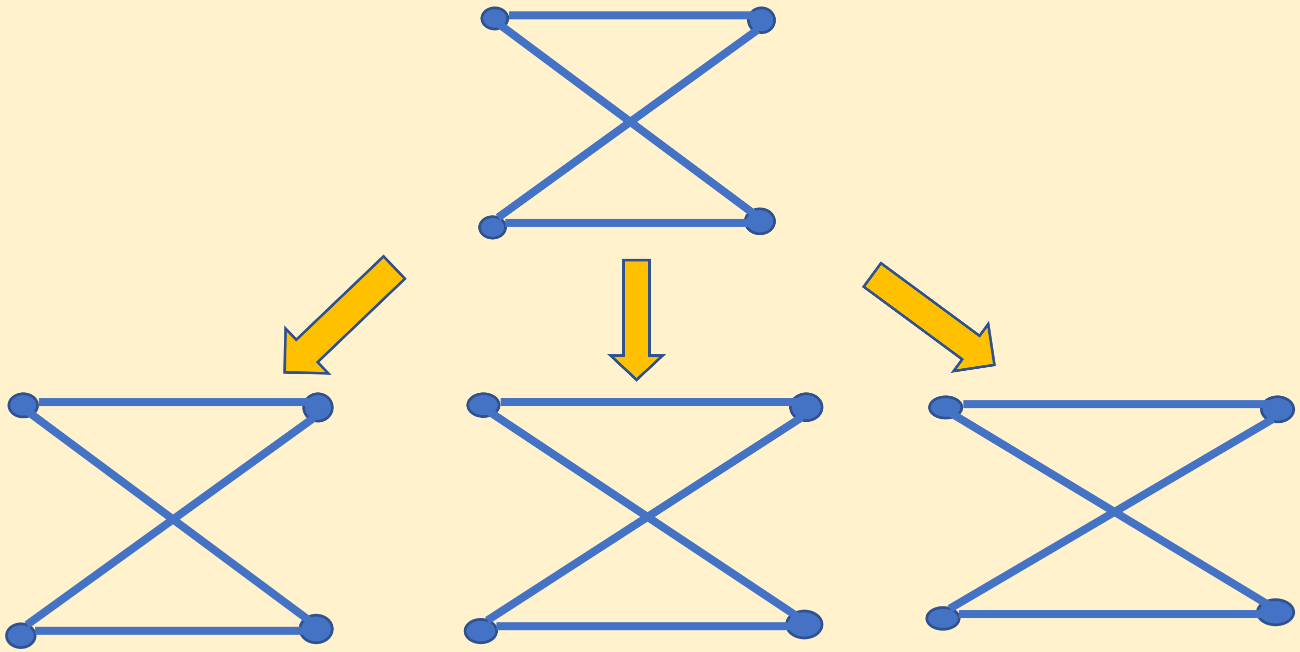
```
Algorithm BFT(G:Graph;n:integer){
    for(i= 1;i <=n; i+ +)
        Visited[i] = 1;
        for(i= 1;i <=n;i+ +)
         if(Visited[i] = 0) BFS(i, G);
}
```

- A complete traversal of an un-connected graph can be made by repeatedly calling BFS each time with an unvisited starting vertex.
- If *G* is connected then all vertices are visited in the first call of BFS.

```
Algorithm BFS(v: vertex; G:Graph) {
    visited[v] = 1;
    u=v;
    MakeEmpty(Q);
    Insert(Q; v);
    while(Not IsEmpty(Q)) {
        for all vertices w adjacent to u{
        if(Visited[w] == 0){
         Insert(Q; w);visited[w] = 1;}
          }
    if(Not IsEmpty(Q)) u=Delete(Q);
    }
}
```

Spanning tree

- A tree is a connected undirected graph with no cycles
- A spanning tree T of an undirected graph G is a subgraph that is a tree which includes all vertices of G, with a minimum possible number of edges.
- e.g. three spanning trees of G



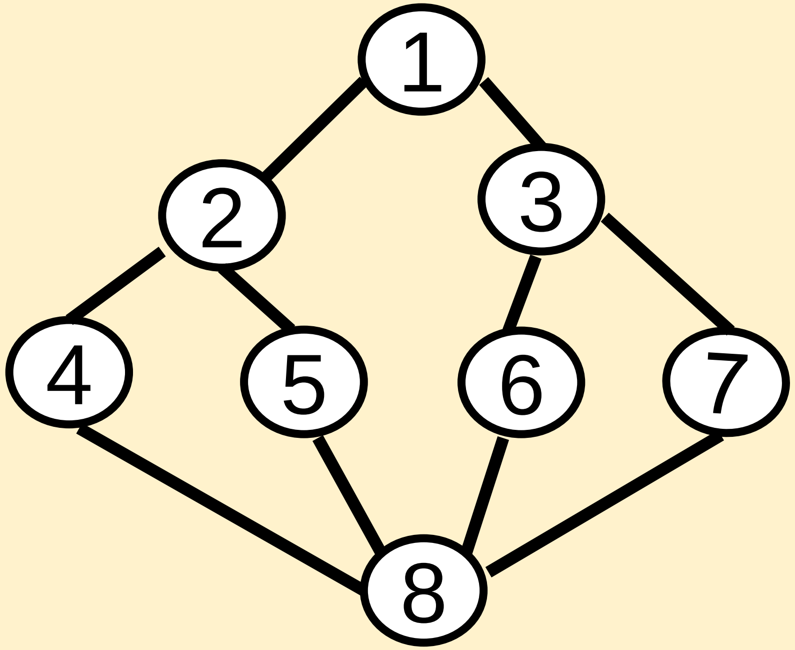
- A graph G has a spanning tree if and only if G is connected.

```
Algorithm BFS-Span(v: vertex; G:Graph)
{
    visited[v] = 1;
    u=v;
    MakeEmpty(Q);
    Insert(Q; v); t={};  //initially empty tree
    while(Not IsEmpty(Q)) {
        for all vertices w adjacent to u{
        if(Visited[w] == 0){
         Insert(Q; w);visited[w] = 1;
         t= t ∪ {(u,w)} ; //add forward edge
         }
   if(Not IsEmpty(Q)) u=Delete(Q);
   }
}
```

```
Algorithm BFS(v: vertex; G:Graph) {
    visited[v] = 1;
    u=v;
    MakeEmpty(Q);
    Insert(Q; v);
    while(Not IsEmpty(Q)) {
        for all vertices w adjacent to u{
        if(Visited[w] == 0){
         Insert(Q; w);visited[w] = 1;}
         }
   if(Not IsEmpty(Q)) u=Delete(Q);
   }
}
```

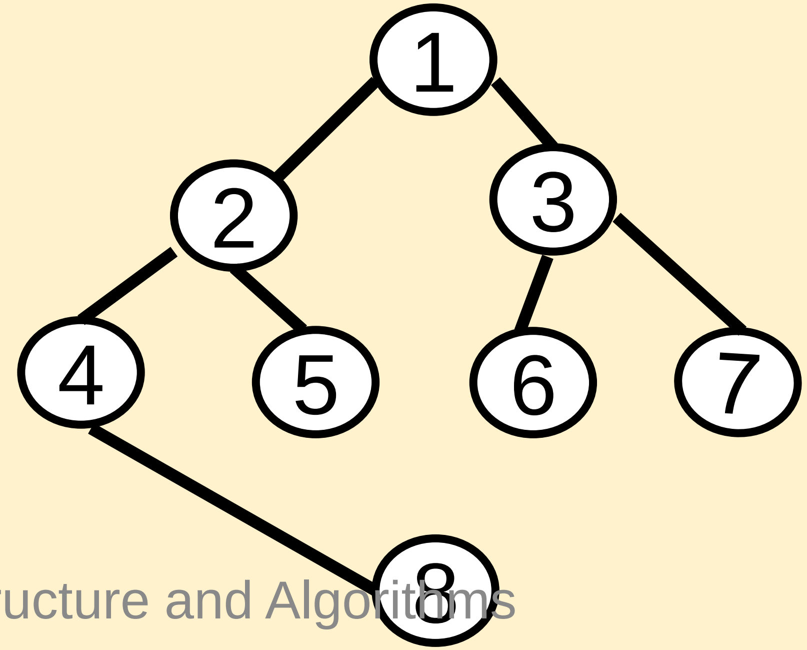
- A slight modification of BFS can be made to compute a spanning tree.

Example: Breadth First Spanning tree from vertex 1



Vertices are visited vertices order:
1,2,3,4,5,6,7,8

Added forward edges:
{1,2},{1,3},{2,4},{2,5},{3,6},{3,7},{4,8}

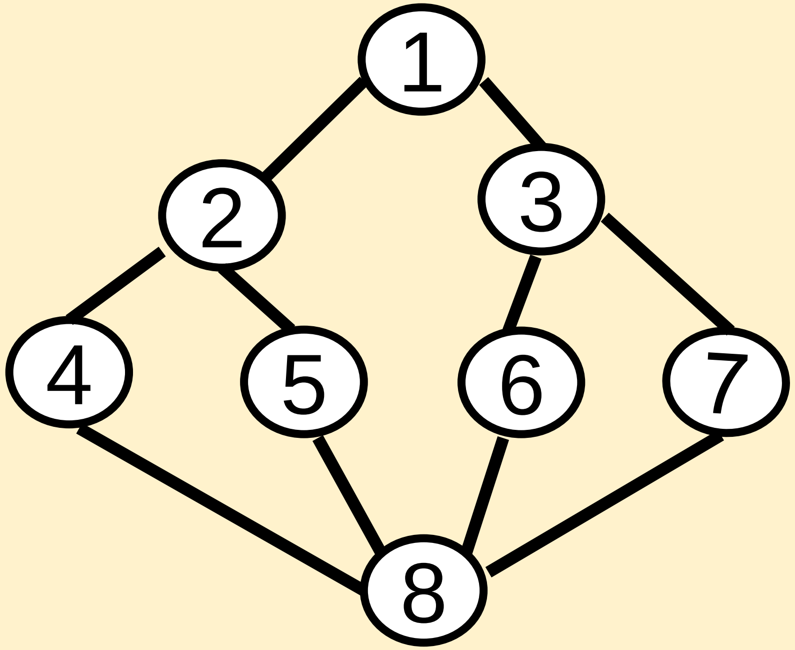


Extension: Modify DFS search to create a spanning tree algorithm

```
Algorithm DFS(v: vertex; G:Graph)
   Visited[v] = 1;
   for all vertices w adjacent to v {
          if(Visited[w] = 0)
         DFS(w; G);
}
```

- A slight modification of DFS can be made to compute a spanning tree.

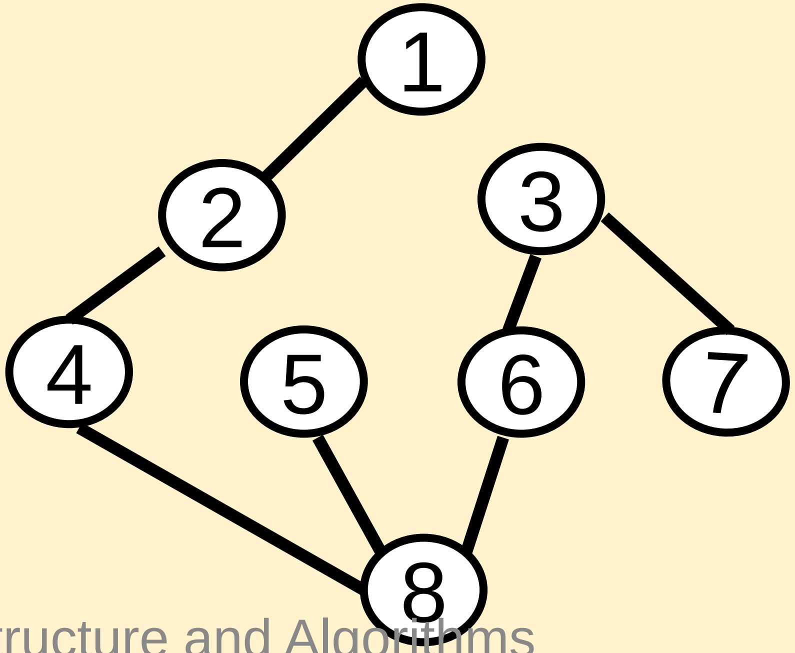
Example: Depth First Spanning tree from vertex 1



Vertices are visited vertices order 1,2,4,8,5,6,3, 7

Added forward edges:

{1,2},{2,4},{4,8},{8,5},{8,6},{6,3},{3,7}

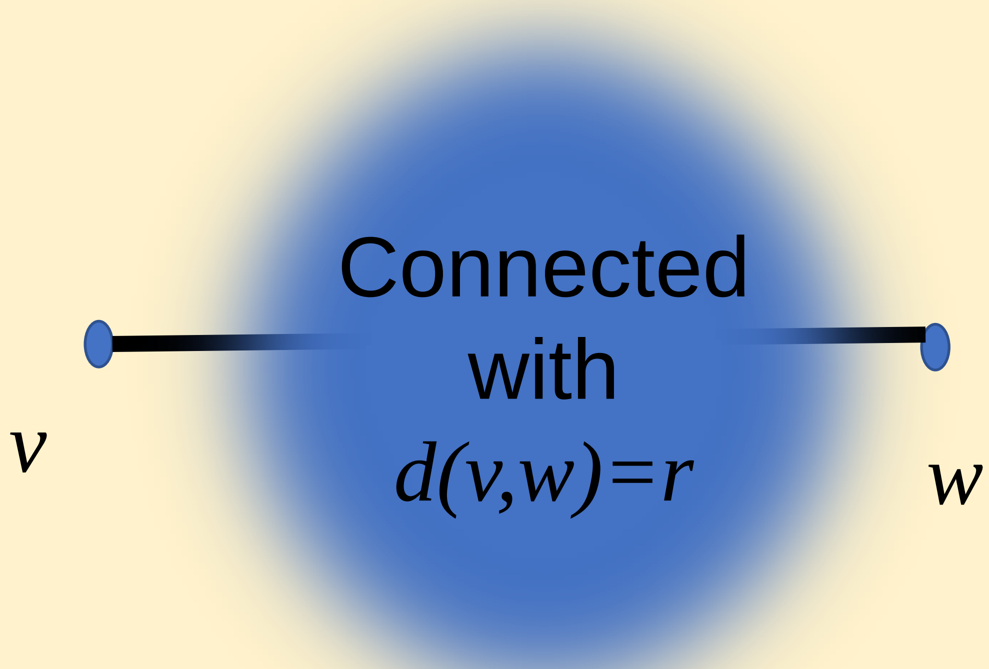


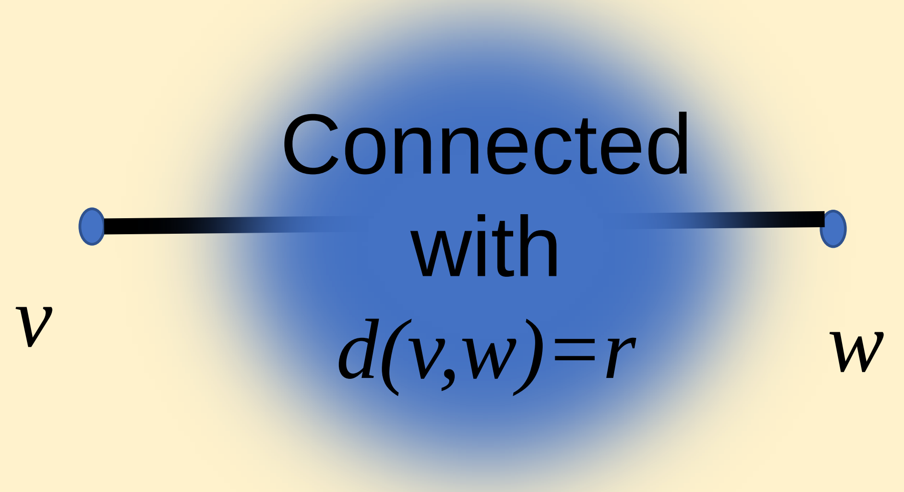
Analysis of BFS

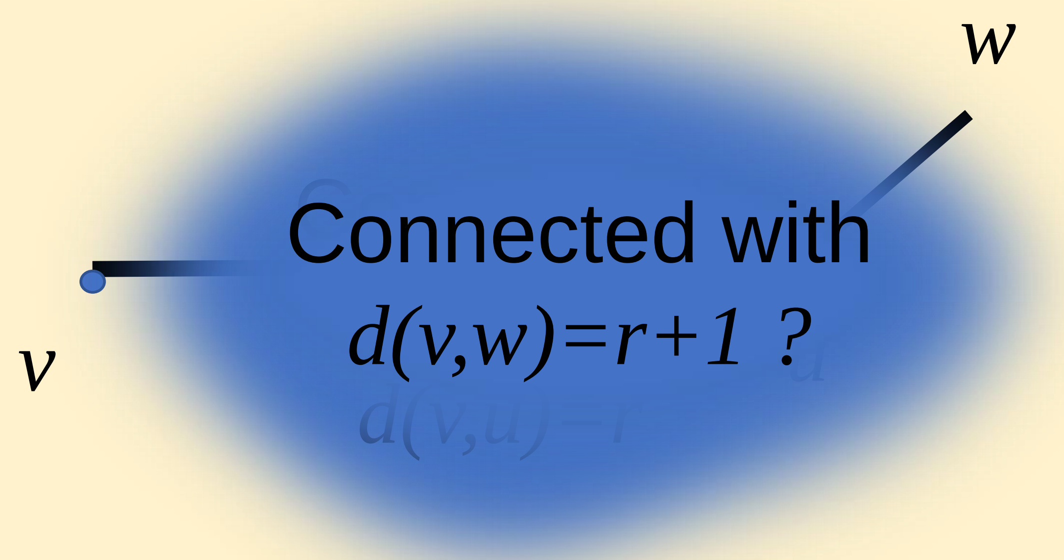
- The BFS algorithm:
  - Start at a vertex *v*– mark it as reached
  - The vertex *v* is, as yet, unexplored
  - When all the vertices adjacent to it (connected by an edge) have been visited, *v* has been explored(reached).
  - Collect all the unvisited vertices adjacent to *v* and add them to a list.
  - Take a vertex from the list and repeat the process
  - When there are no vertices left in the list they have all been explored (reached).
  - This yields the set of vertices that are “reach-able” from the start vertex *v*.
- Theorem: Algorithm BFS visits all reachable vertices.

- Mathematical Induction is a special way of proving things. It has only 2 steps:
  Step 1. Show it is true for first case, usually *n=1*
  Step 2. Show that if *n=k* is true , then *n=k+1* is also true
- Example:
  Step 1. The first domino falls
  Step 2. When any domino falls, the next domino falls
- So ... all dominos will fall!
- We will proof this theorem by method of induction.

- Proof is by induction based on the the distance of shortest path.
- Suppose $d(v, w)$ is the length (number of edges) of the shortest path from vertex v to a reachable vertex w.
- Basic step : Clearly, all w with $d(v; w) \leq 1$ are visited.
- Hypothesis : Assume all vertices w with $d(v; w) \leq r$ are visited.







- Inductive step : We now show that all w with $d(v; w) \leq r+1$ are also visited.
  - Suppose that $d(v, w) = r+1$ for some w, and let u be a vertex adjacent to w, $u \neq v$ and $r \geq 1$.
  - Then $d(v, u) = r$ (shortest path) and immediately prior to u being visited by BFS, u is put on the Queue.
  - Since the algorithm only stops when the Queue is empty, at some stage u is taken off the Queue and is explored and thus visits w.
  - The only way for w not to be visited is if it is not reachable.