Graph Searching

CSE 373

Data Structures

Lecture 20

Readings

- Reading
  - Sections 9.5 and 9.6

Graph Searching

- Find Properties of Graphs
  - Spanning trees
  - Connected components
  - Bipartite structure
  - Biconnected components
- Applications
  - Finding the web graph – used by Google and others
  - Garbage collection – used in Java run time system
  - Alternating paths for matching

Graph Searching Methodology Breadth-First Search (BFS)

- Breadth-First Search (BFS)
  - › Use a queue to explore neighbors of source vertex, then neighbors of neighbors etc.
  - › All nodes at a given distance (in number of edges) are explored before we go further

Graph Searching Methodology Depth-First Search (DFS)

- Depth-First Search (DFS)
  - Searches down one path as deep as possible
  - When no nodes available, it backtracks
  - When backtracking, it explores side-paths that were not taken
  - Uses a stack (instead of a queue in BFS)
  - Allows an easy recursive implementation

Depth First Search Algorithm

- Recursive marking algorithm
- Initially every vertex is unmarked

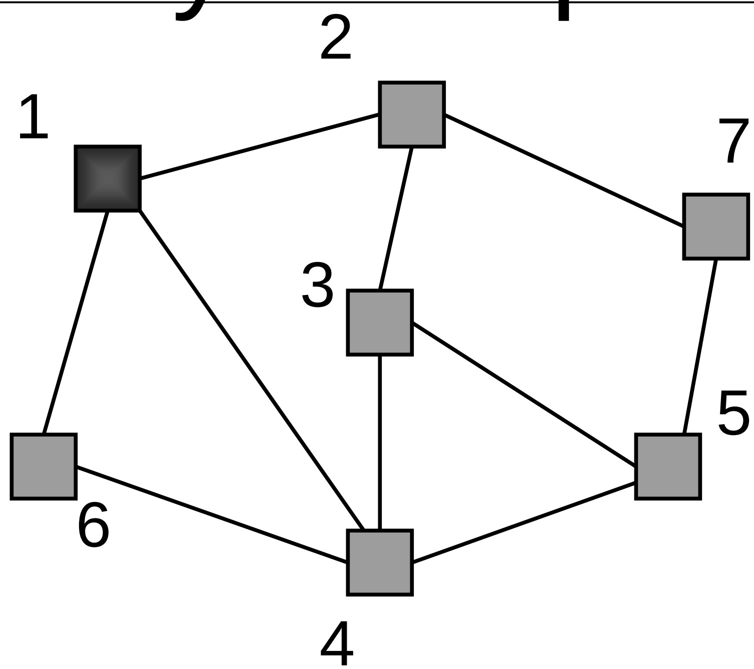
```
DFS(i: vertex)
   mark i;
   for each j adjacent to i do
      if j is unmarked then DFS(j)
end{DFS}
```

Marks all vertices reachable from i

DFS Application: Spanning Tree

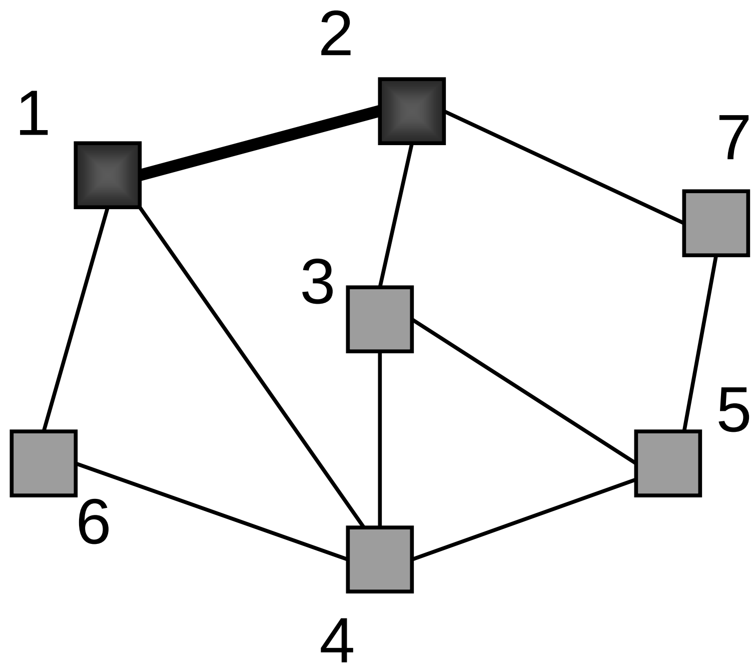
- Given a (undirected) graph G(V,E) a spanning tree of G is a graph G'(V',E')
  - V' = V, the tree touches all vertices (spans) the graph
  - E' is a subset of E such G' is connected and there is no cycle in G'
  - A graph is connected if given any two vertices u and v, there is a path from u to v

Example of DFS: Graph connectivity and spanning tree



DFS(1)

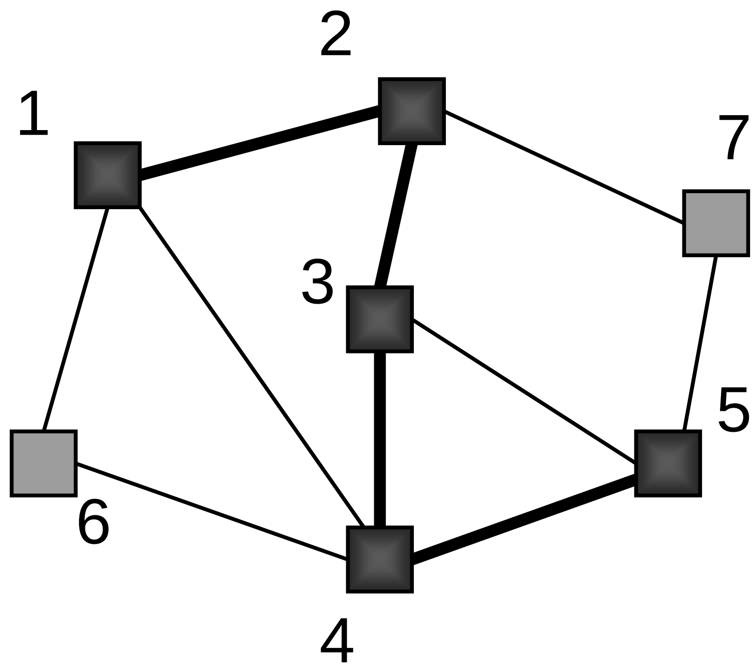
Example Step 2



DFS(1)
DFS(2)

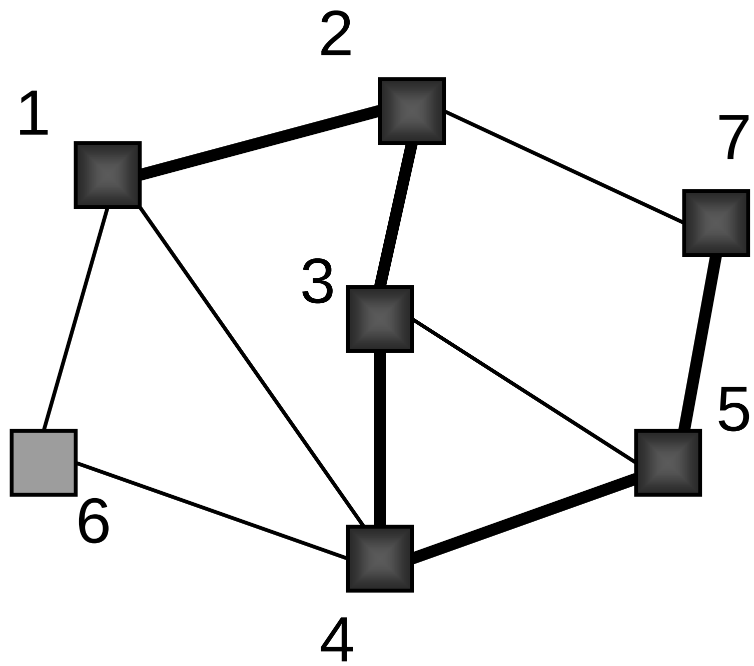
Red links will define the spanning tree if the graph is connected

Example Step 5



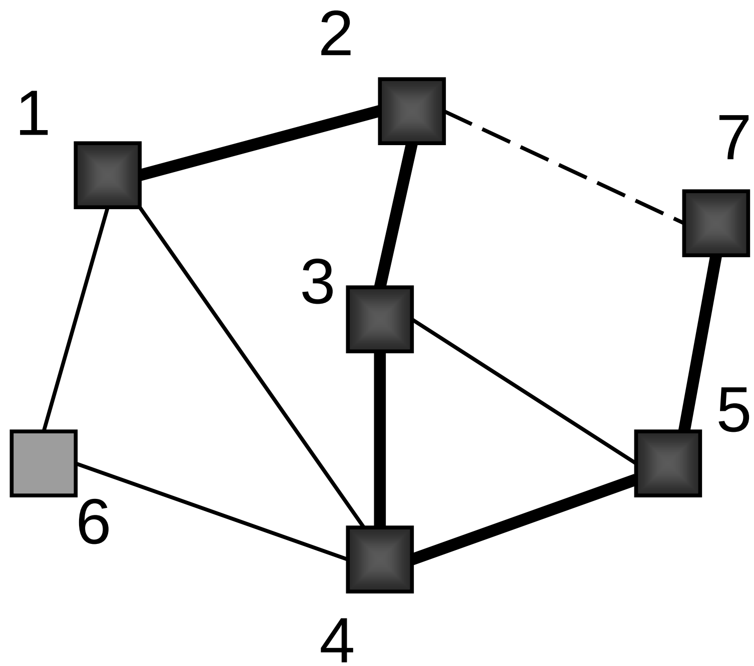
DFS(1)
DFS(2)
DFS(3)
DFS(4)
DFS(5)

Example Steps 6 and 7



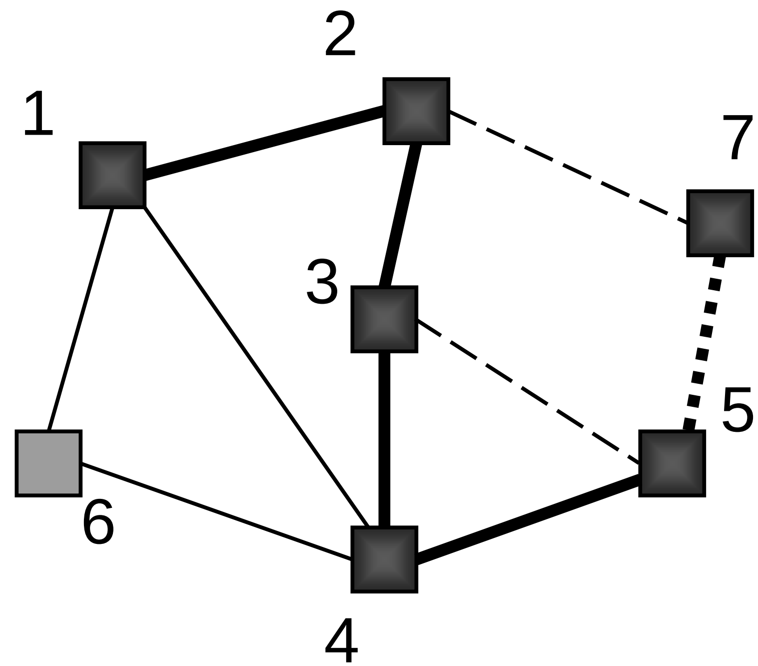
DFS(1)
DFS(2)
DFS(3)
DFS(4)
DFS(5)
~~DFS(3)~~
DFS(7)

Example Steps 8 and 9



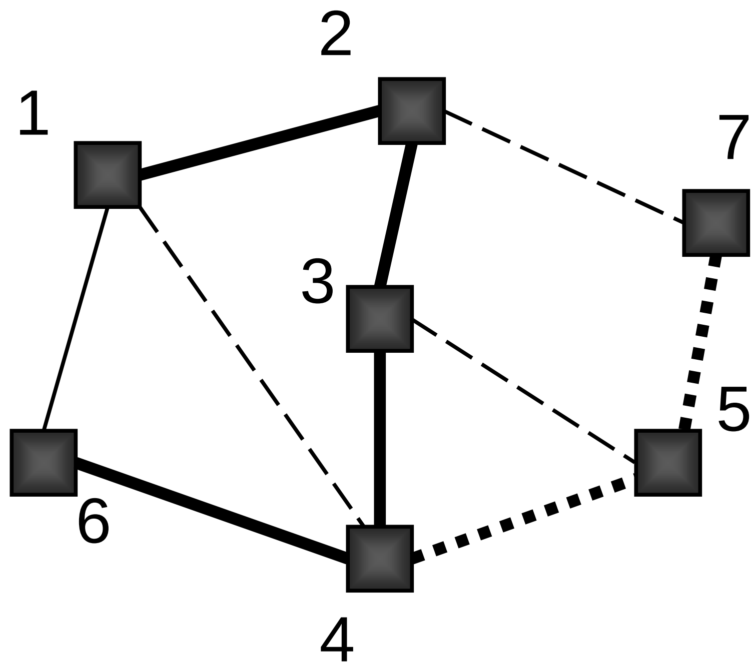
DFS(1)
DFS(2)
DFS(3)
DFS(4)
DFS(5)
DFS(7)

Example Step 10 (backtrack)



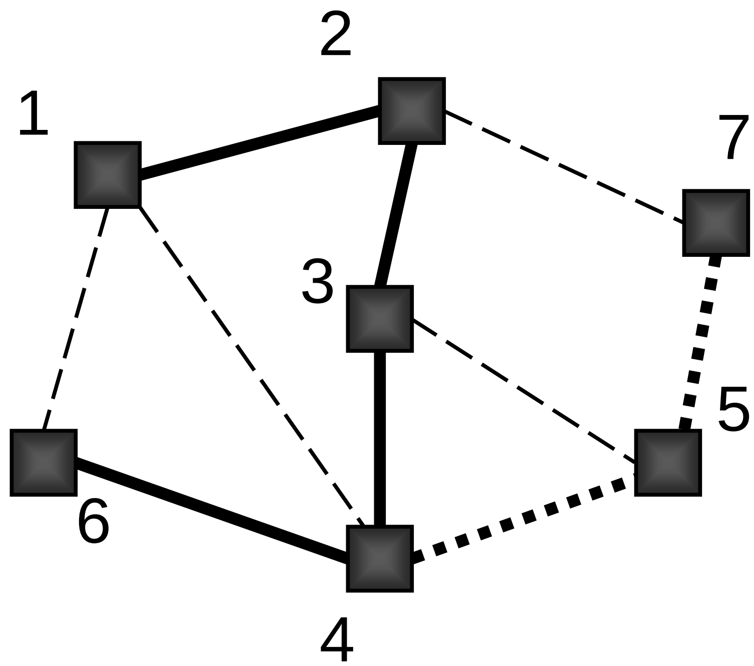
DFS(1)
DFS(2)
DFS(3)
DFS(4)
DFS(5)

Example Step 12



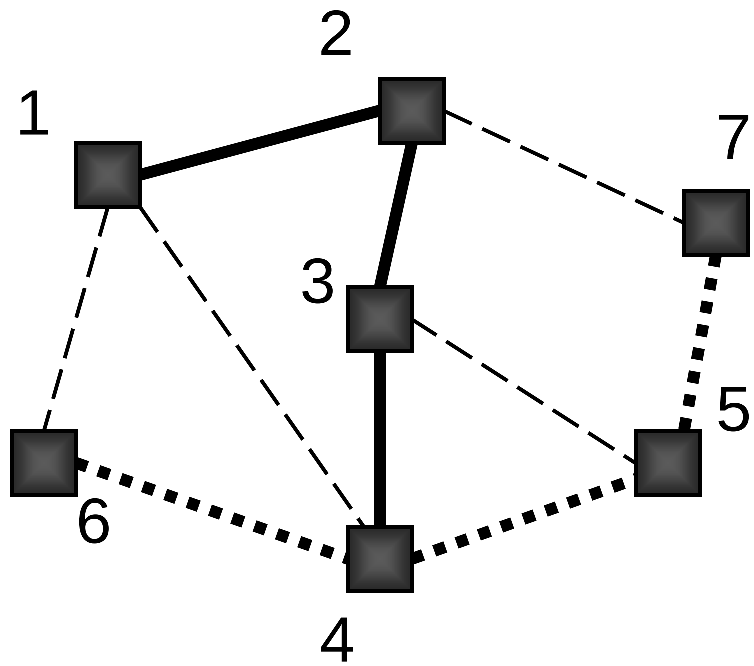
DFS(1)
DFS(2)
DFS(3)
DFS(4)
DFS(6)

Example Step 13



DFS(1)
DFS(2)
DFS(3)
DFS(4)
DFS(6)

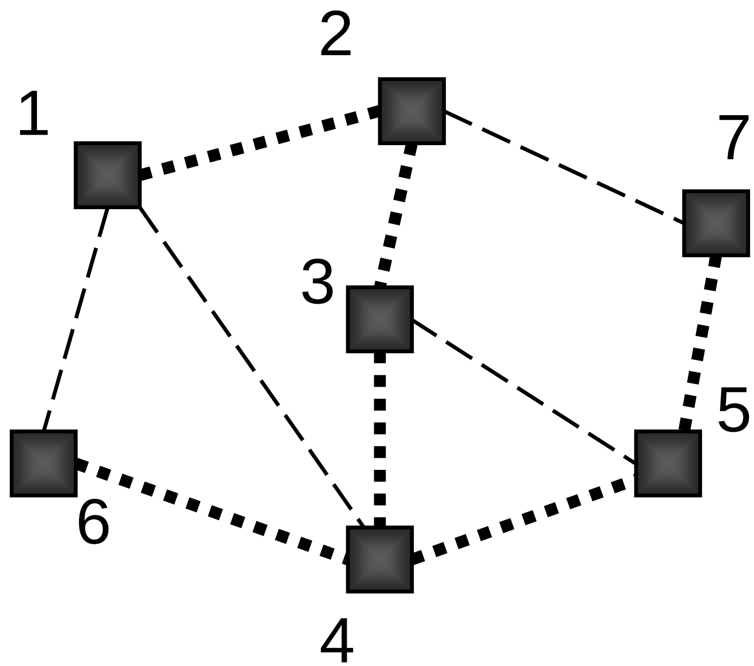
Example Step 14



DFS(1)
DFS(2)
DFS(3)
DFS(4)

Example Step 17

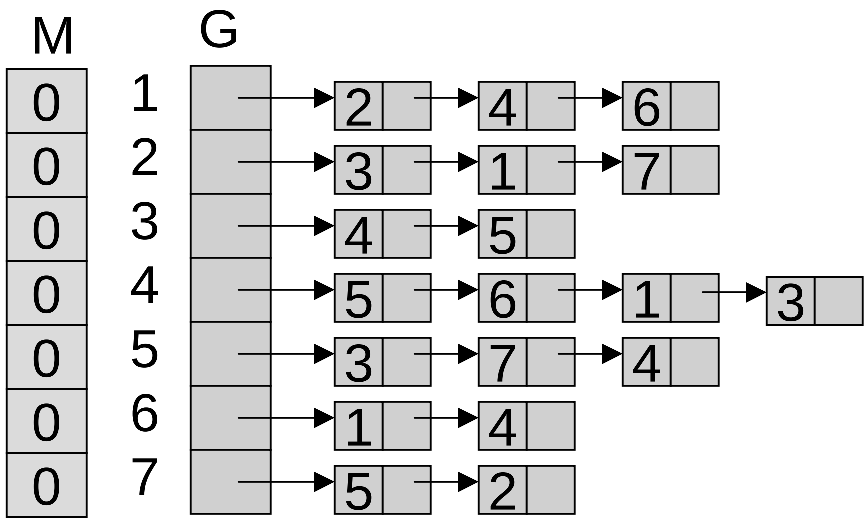
DFS(1)

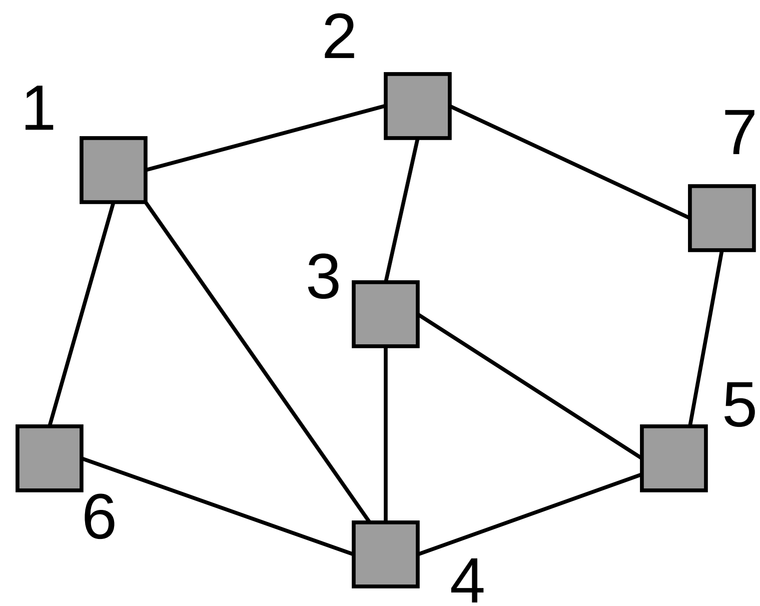


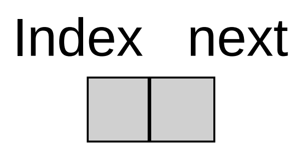
All nodes are marked so graph is connected;
red links define a spanning tree

Adjacency List Implementation

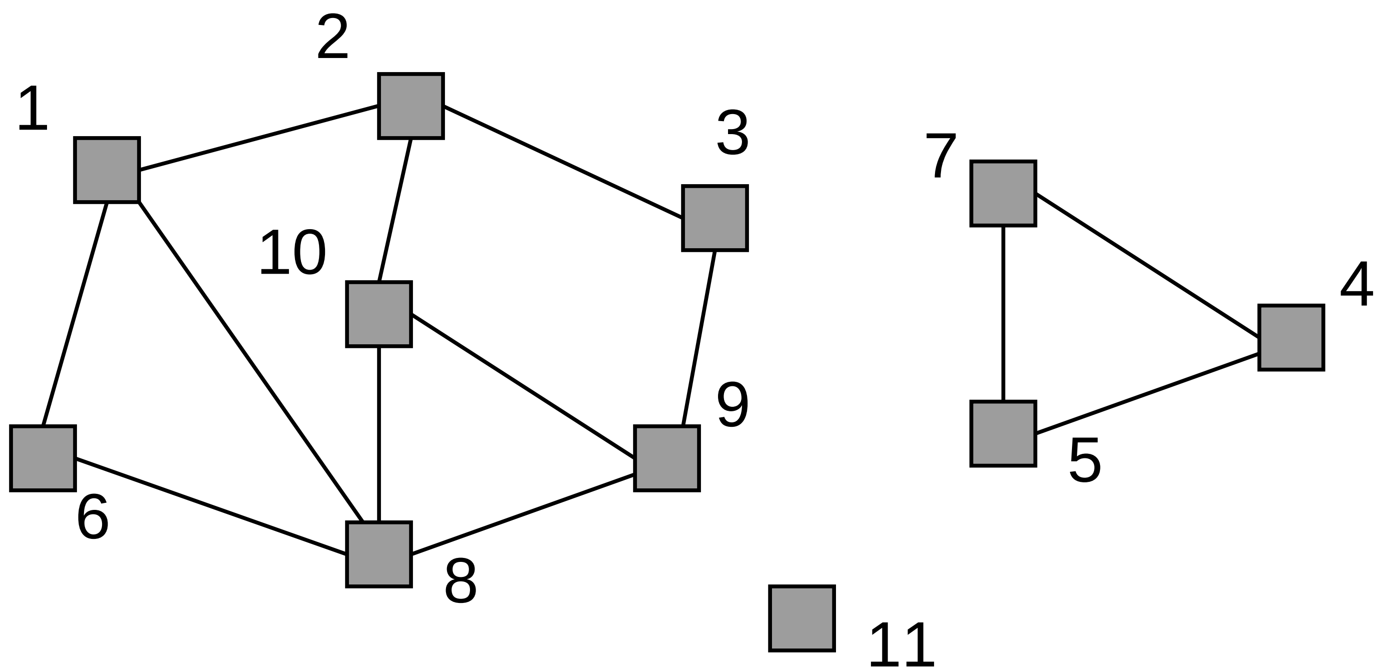
- Adjacency lists





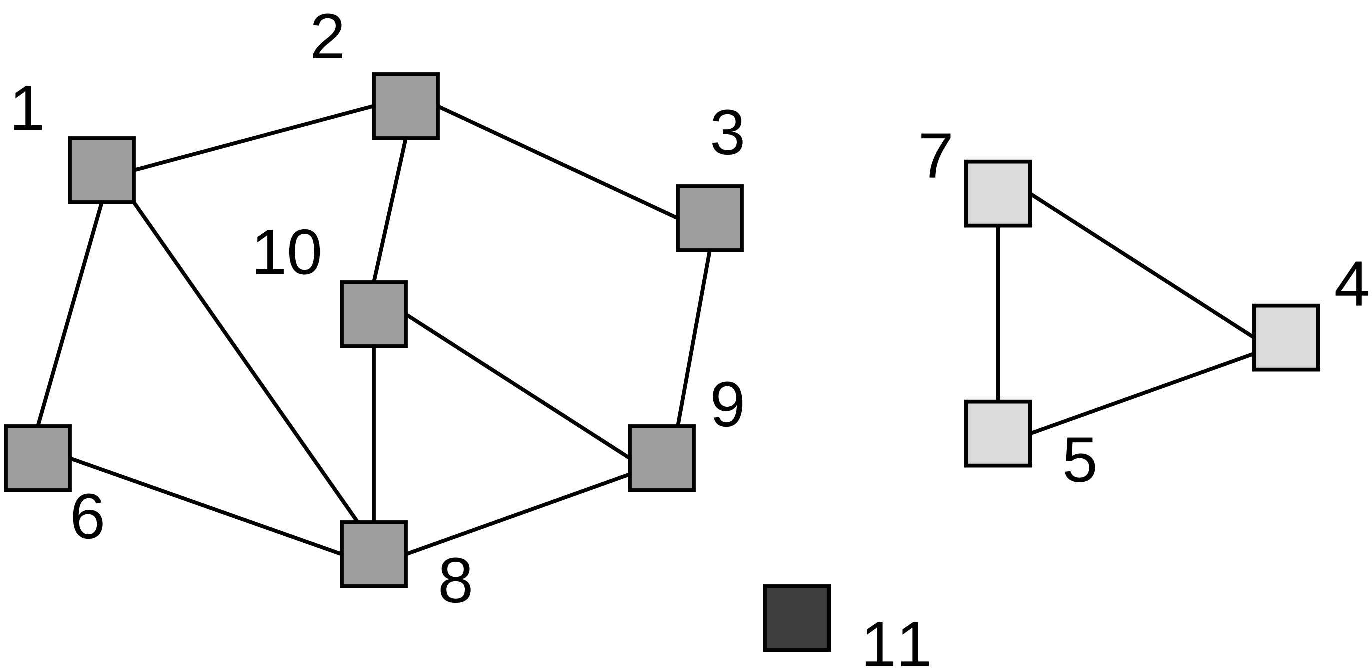


Connected Components



3 connected components

Connected Components



3 connected components are labeled

Depth-first Search for Labeling Connected components

```
Main {
i : integer
for i = 1 to n do M[i] := 0;
label := 1;
for i = 1 to n do
  if M[i] = 0 then DFS(G,M,i,label);
  label := label + 1;
}
DFS(G[]: node ptr array, M[]: int array, i,label: int) {
  v : node pointer;
  M[i] := label;
  v := G[i];
  while v ≠ null do
    if M[v.index] = 0 then DFS(G,M,v.index,label);
    v := v.next;
}
```

Performance DFS

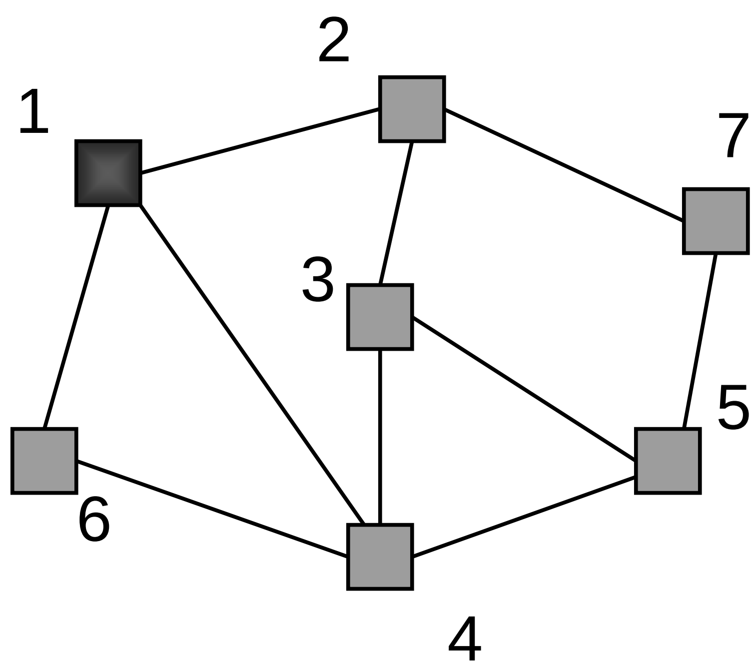
- n vertices and m edges
- Storage complexity $O(n + m)$
- Time complexity $O(n + m)$
- Linear Time!

Breadth-First Search

```
BFS
Initialize Q to be empty;
Enqueue(Q,1) and mark 1;
while Q is not empty do
    i := Dequeue(Q);
    for each j adjacent to i do
        if j is not marked then
            Enqueue(Q,j) and mark j;
end{BFS}
```

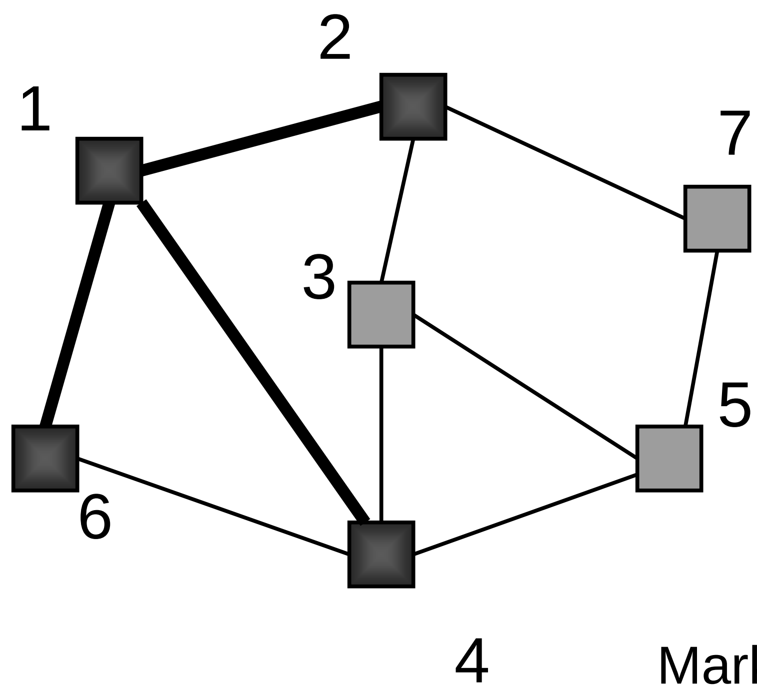
Can do Connectivity using BFS

- Uses a queue to order search



Queue = 1

Beginning of example



Queue = 2,4,6

Mark while on queue to avoid putting in queue more than once

Depth-First vs Breadth-First

- Depth-First
  - › Stack or recursion
  - › Many applications
- Breadth-First
  - › Queue (recursion no help)
  - › Can be used to find shortest paths from the start vertex
  - › Can be used to find short alternating paths for matching

Minimum Spanning Tree

- Edges are weighted: find minimum cost spanning tree
- Applications
  - › Find cheapest way to wire your house
  - › Find minimum cost to wire a message on the Internet