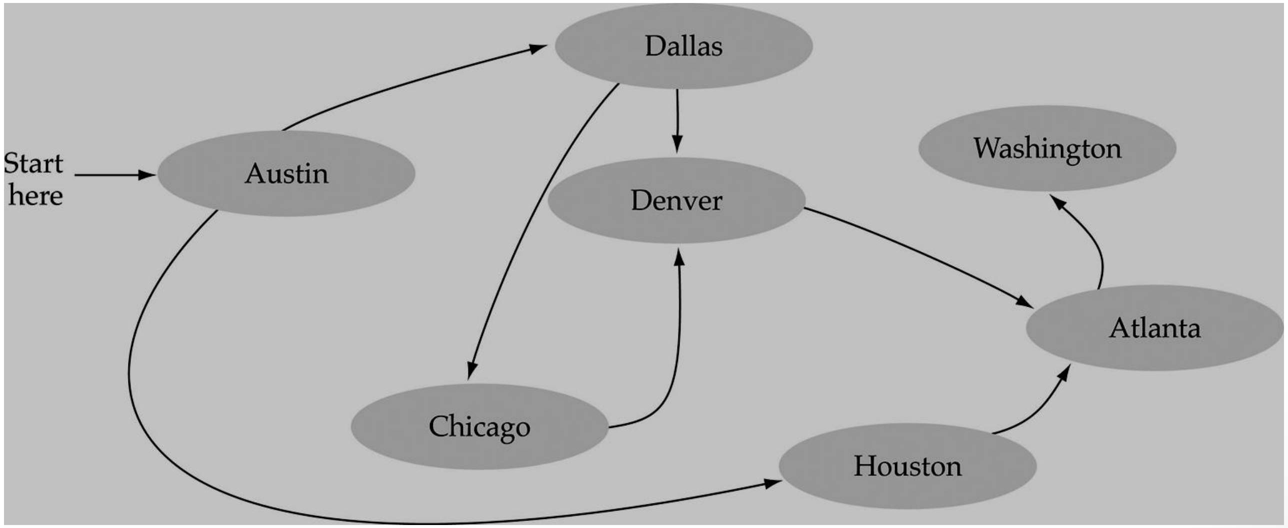
Graph

- Motivation and Terminology
- Representations
- Traversals
- Three Problems

What is a graph?

- A data structure that consists of a set of nodes (*vertices*) and a set of edges that relate the nodes to each other
- The set of edges describes relationships among the vertices



Formal definition of graphs

- A graph G is defined as follows:

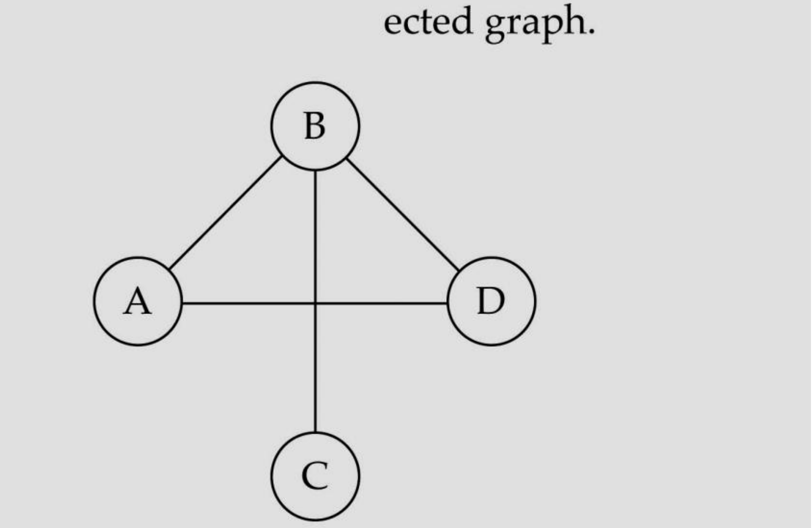
$$G=(V,E)$$

V(G): a finite, nonempty set of vertices

E(G): a set of edges (pairs of vertices)

Directed vs. undirected graphs

- When the edges in a graph have no direction, the graph is called *undirected*

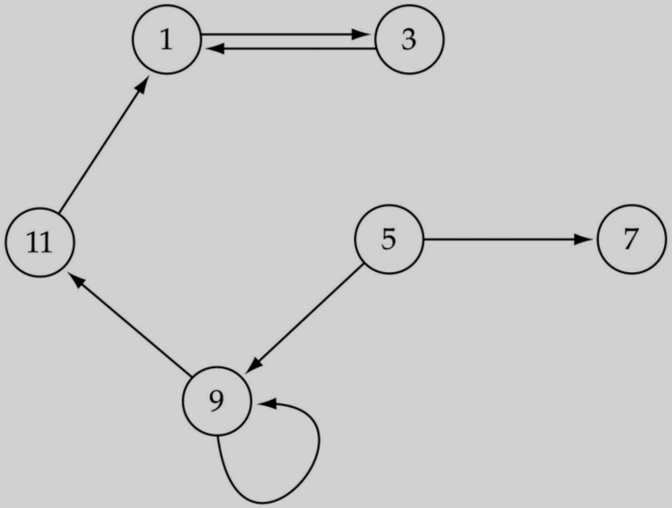


V(Graph1) = { A, B, C, D }
E(Graph1) = { (A, B), (A, D), (B, C), (B, D) }

Directed vs. undirected graphs (cont.)

- When the edges in a graph have a direction, the graph is called *directed* (or *digraph*)

(b) Graph2 is a directed graph.



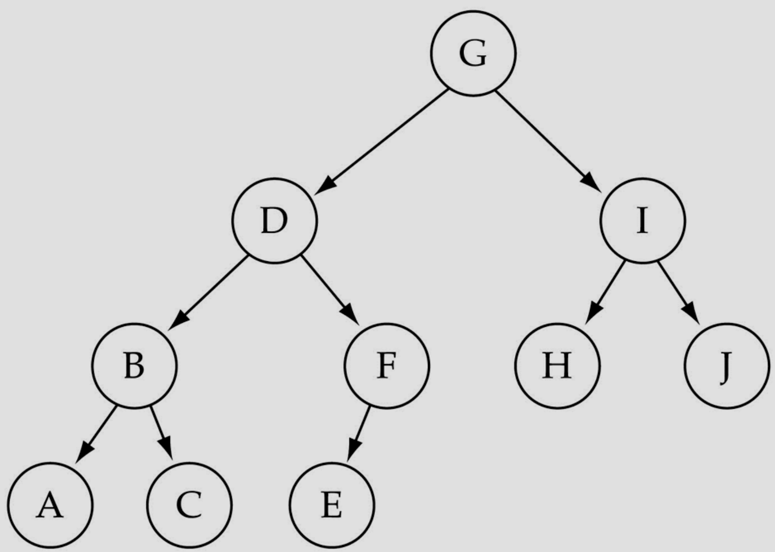
V(Graph2) = { 1, 3, 5, 7, 9, 11 }

E(Graph2) = {(1,3) (3,1) (5,9) (9,11) (5,7) 1), (9, 9), (11, 1) }

Trees vs graphs

- Trees are special cases of graphs!!

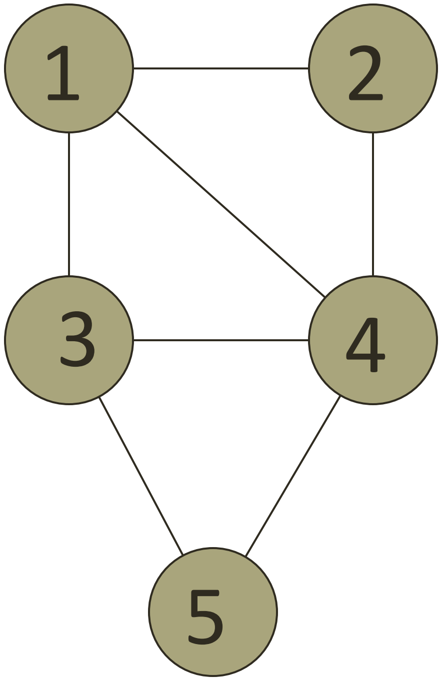
(c) Graph3 is a directed graph.



V(Graph3) = { A, B, C, D, E, F, G, H, I, J }
E(Graph3) = { (G, D), (G, J), (D, B), (D, F) (I, H), (I, J), (B, A), (B, C), (F, E) }

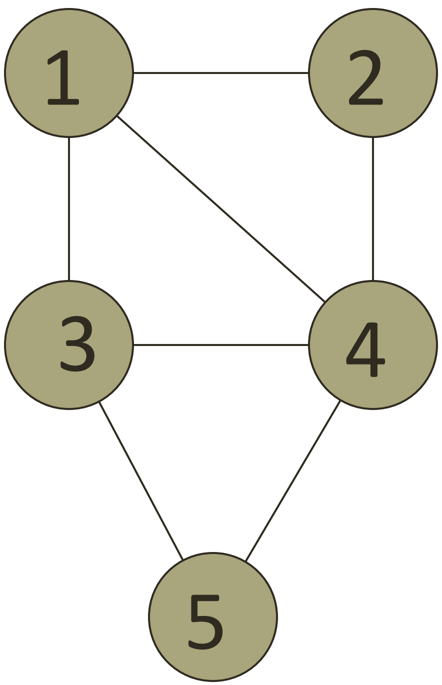
Representation

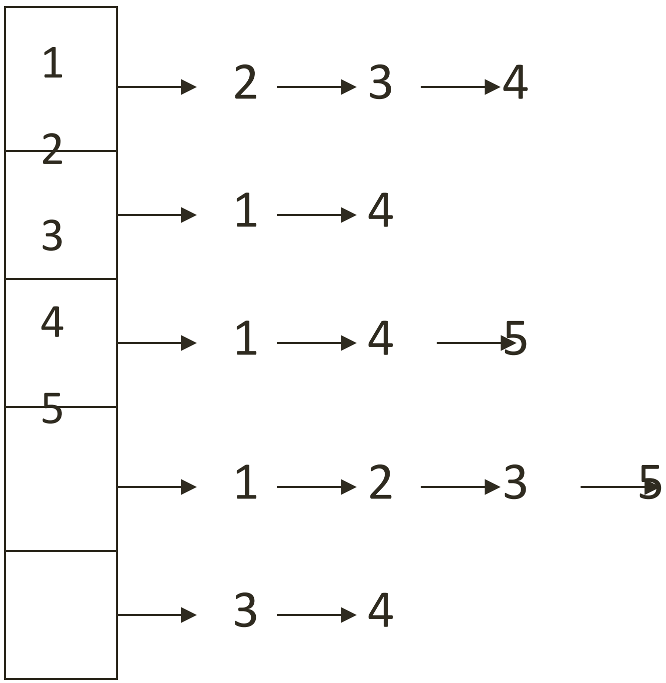
Adjacency Matrix



	1	2	3	4	5
1	0	1	1	1	0
2	1	0	0	1	0
3	1	0	0	1	1
4	1	1	1	0	1
5	0	0	1	1	0

Adjacency List





Adjacency matrix vs. adjacency list representation

- **Adjacency matrix**
  - Good for dense graphs --$|E|$~$O(|V|^2)$
  - Memory requirements: $O(|V| + |E|) = O(|V|^2)$
  - Connectivity between two vertices can be tested quickly
- **Adjacency list**
  - Good for sparse graphs -- $|E|$~$O(|V|)$
  - Memory requirements: $O(|V| + |E|)=O(|V|)$
  - Vertices adjacent to another vertex can be found quickly

Search

DEPTH-FIRST SEARCH

- **Depth-first search**, or **DFS**, is a way to traverse the graph.
- The principle of the algorithm is quite simple: to go forward (in depth) while there is such possibility, otherwise to backtrack.

DEPTH-FIRST SEARCH

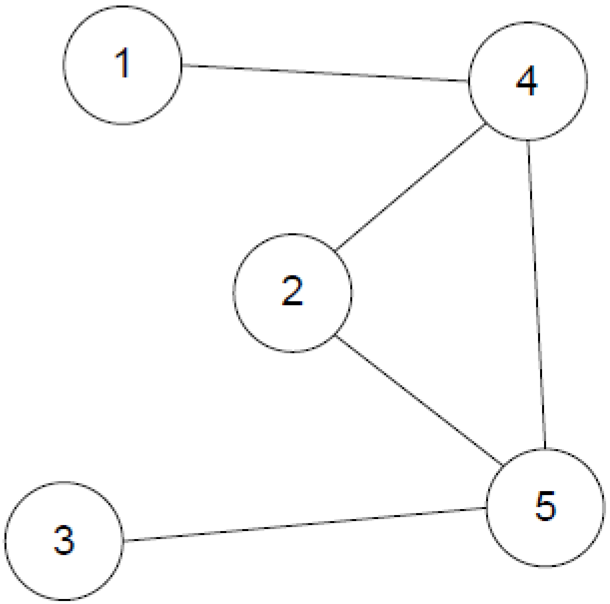
- **Algorithm**
  - In DFS, each vertex has three possible colors representing its state:
  - white: vertex is unvisited;
  - gray: vertex is in progress;
  - black: DFS has finished processing the vertex.

DEPTH-FIRST SEARCH

- Initially all vertices are white (unvisited). DFS starts in arbitrary vertex and runs as follows:

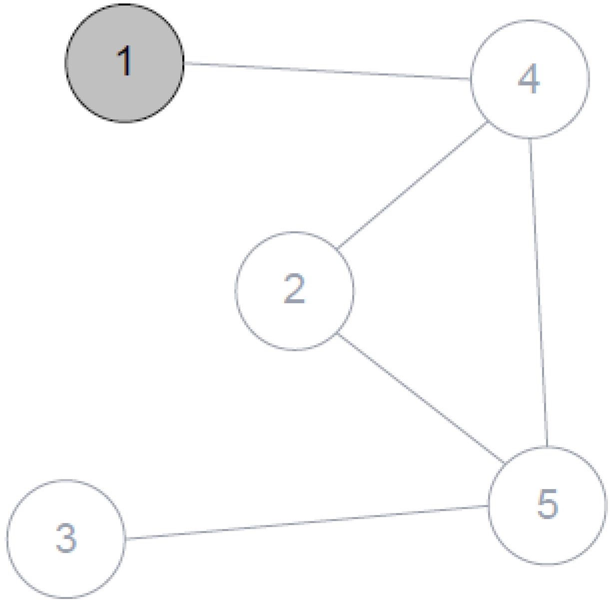
1. Mark vertex **u** as gray (visited).
2. For each edge **(u, v)**, where **v** is white, run depth-first search for **v** recursively.
3. Mark vertex **u** as black and backtrack to the parent.

DEPTH-FIRST SEARCH



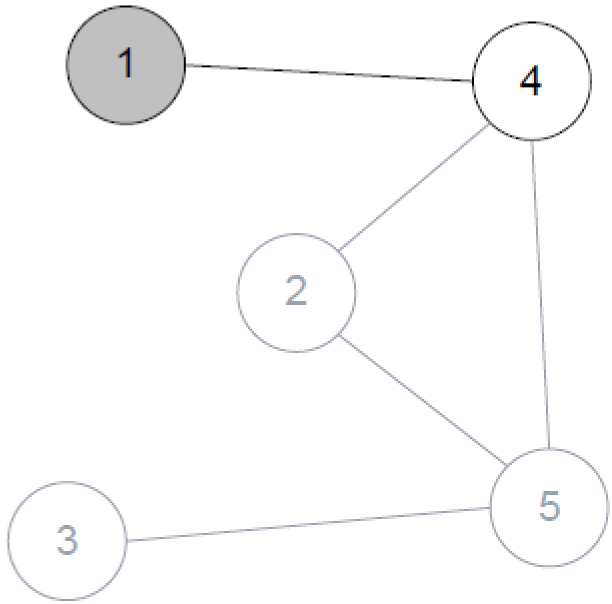
- Start from a vertex with number 1

DEPTH-FIRST SEARCH



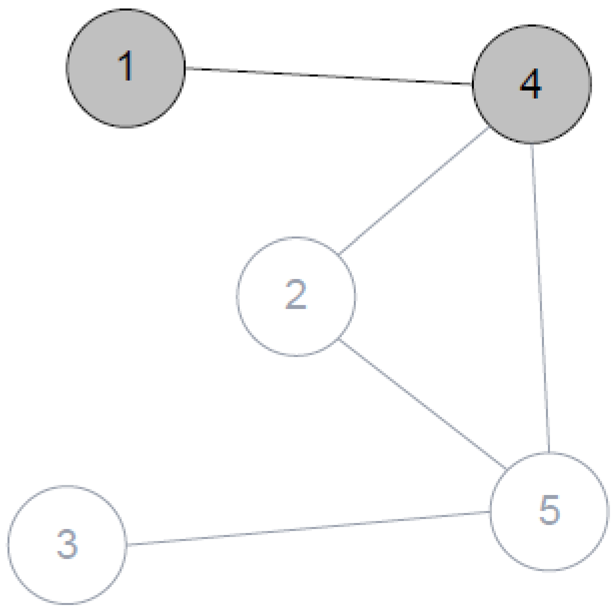
- Mark vertex 1 as gray.

DEPTH-FIRST SEARCH



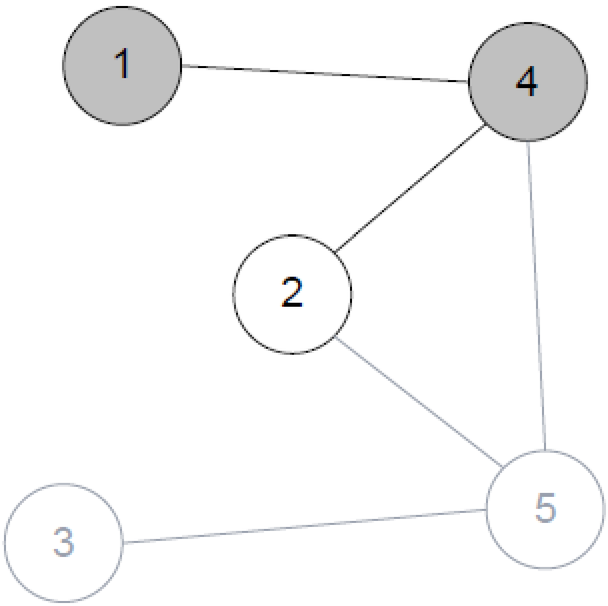
- There is an edge (1, 4) and a vertex 4 is unvisited.
- Go there.

DEPTH-FIRST SEARCH



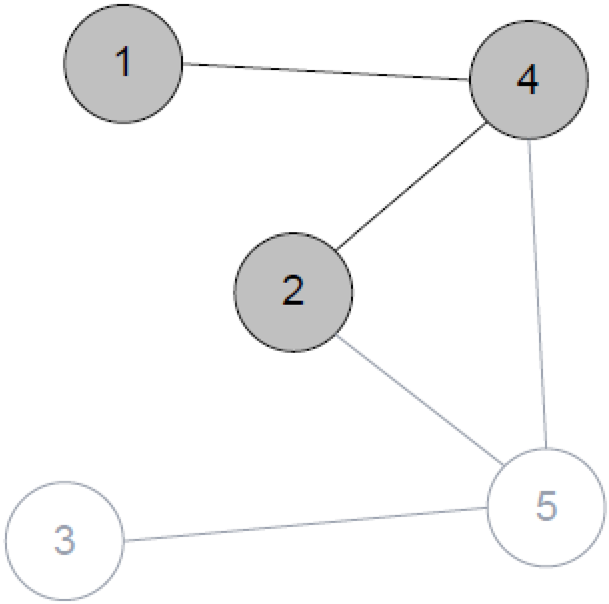
- Mark the vertex 4 as gray.

DEPTH-FIRST SEARCH



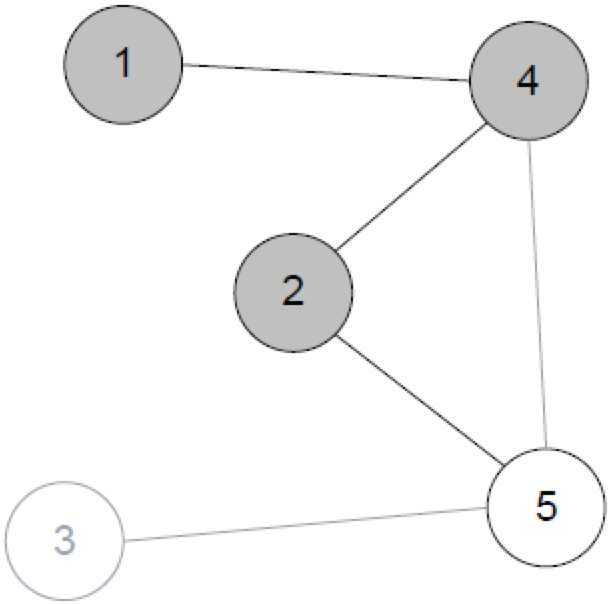
- There is an edge (4, 2) and vertex a 2 is unvisited.
- Go there.

DEPTH-FIRST SEARCH



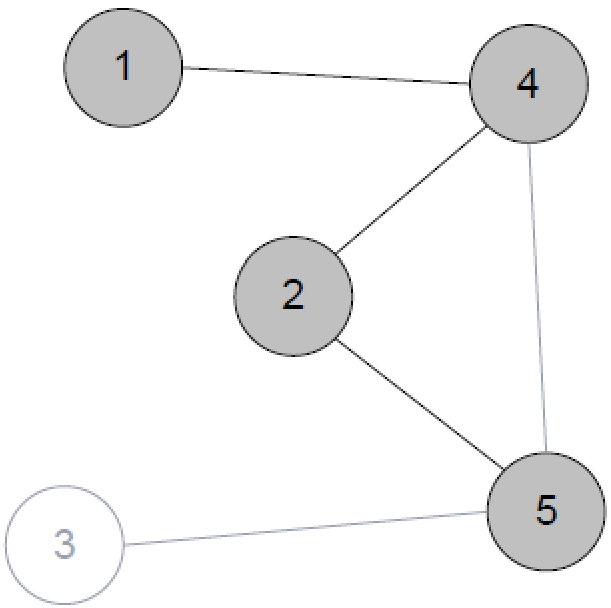
- Mark the vertex 2 as gray.

DEPTH-FIRST SEARCH



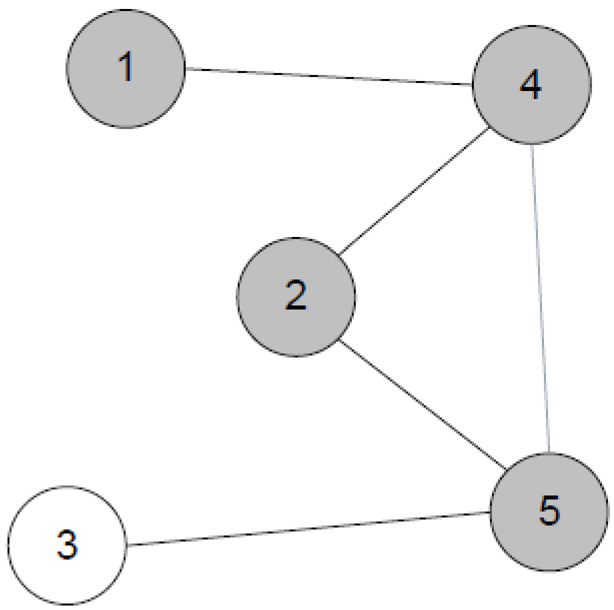
- There is an edge (2, 5) and a vertex 5 is unvisited.
- Go there.

DEPTH-FIRST SEARCH



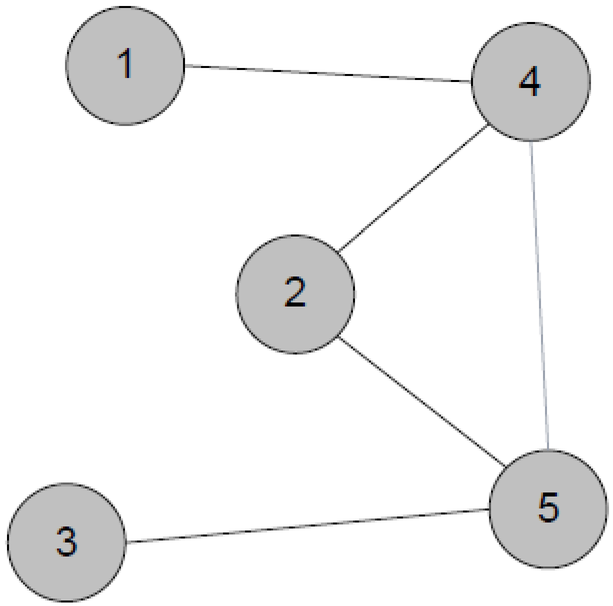
- Mark the vertex 5 as gray.

DEPTH-FIRST SEARCH



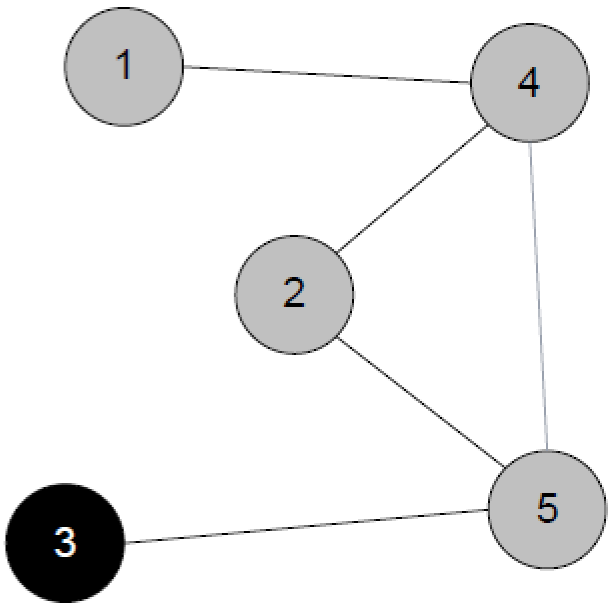
- There is an edge (5, 3) and a vertex 3 is unvisited.
- Go there.

DEPTH-FIRST SEARCH



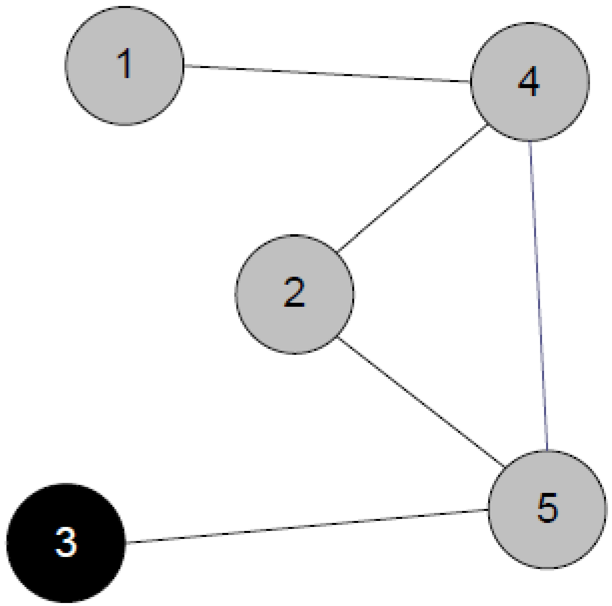
- Mark the vertex 3 as gray.

DEPTH-FIRST SEARCH



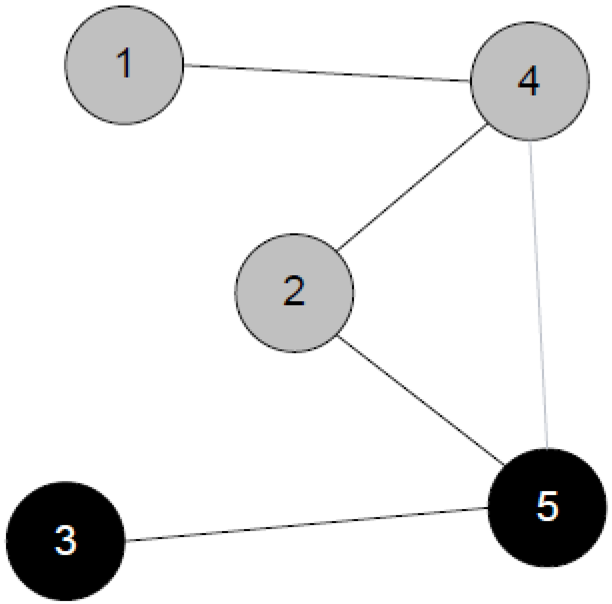
- There are no ways to go from the vertex **3**.
- Mark it as black and backtrack to the vertex **5**.

DEPTH-FIRST SEARCH



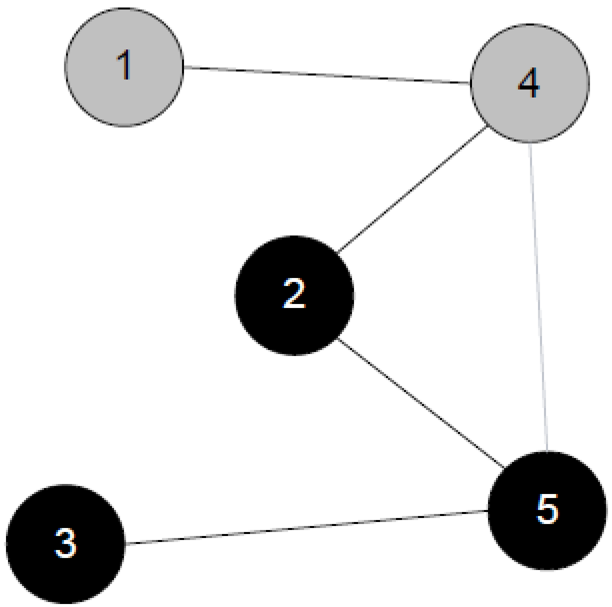
- There is an edge **(5, 4)**, but the vertex 4 is gray.

DEPTH-FIRST SEARCH



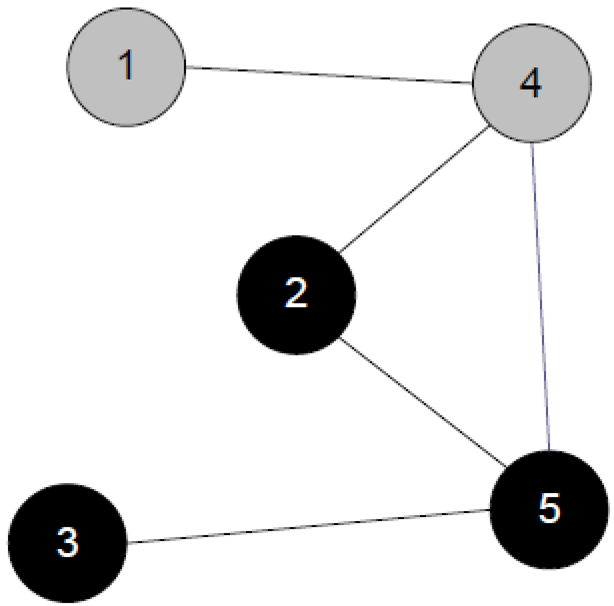
- There are no ways to go from the vertex **5.**
- Mark it as black and backtrack to the vertex **2**.

DEPTH-FIRST SEARCH



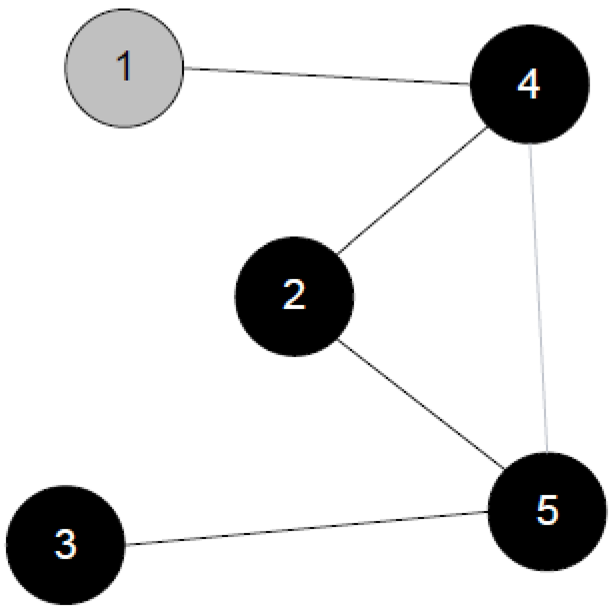
- There are no more edges, adjacent to vertex **2.**
- Mark it as black and backtrack to the vertex **4**.

DEPTH-FIRST SEARCH



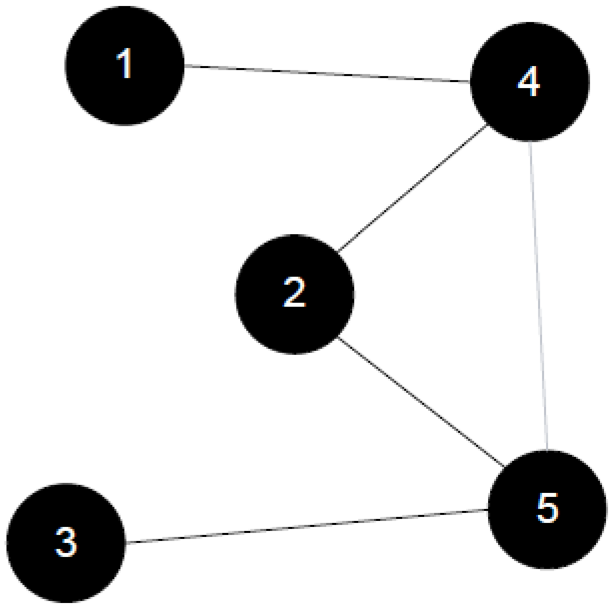
- There is an edge **(4, 5)**, but the vertex 5 is black.

DEPTH-FIRST SEARCH



- There are no more edges, adjacent to the vertex **4.**
- Mark it as black and backtrack to the vertex **1**.

DEPTH-FIRST SEARCH



- There are no more edges, adjacent to the vertex **1**.
- Mark it as black.
- DFS is over.

DEPTH-FIRST SEARCH

- As you can see from the example, DFS doesn't go through all edges.
- The vertices and edges, which depth-first search has visited is a **tree**.
  - This tree contains all vertices of the graph (if it is connected) and is called ***graph spanning tree***.

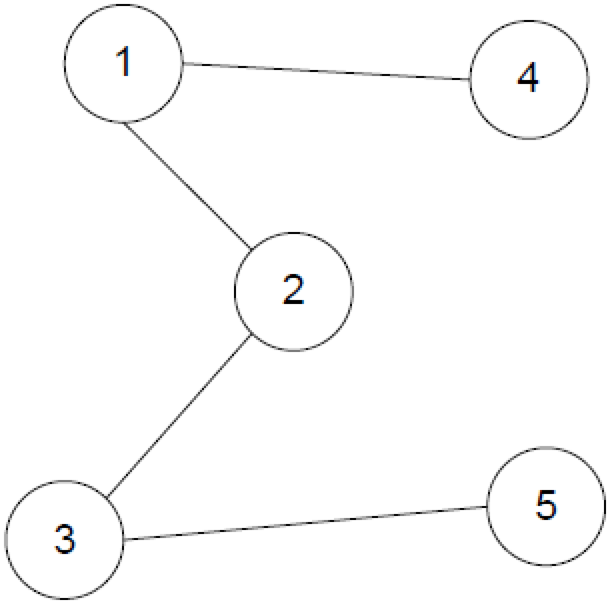
Breadth-First Search

BREADTH-FIRST SEARCH

- **Breadth-First Search** of a graph is similar to traversing a binary tree level-by-level.
    - All the nodes at any level, i, are visited before visiting the nodes at level i+1.

BREADTH-FIRST SEARCH

- The breadth-first ordering of the vertices of the following graph is as follows:
  - 1 2 4 3 5



BREADTH-FIRST SEARCH

- The breadth-first search traverses the graph from each vertex that is not visited.
  - Starting at the first vertex, the graph is traversed as much as possible
  - Then go to the next vertex that has not been visited.

BREADTH-FIRST SEARCH

- To implement the breadth-first search algorithm, we use a queue.

- The general algorithm is as follows:

1. for each vertex v in the graph
   if v is not visited
   add v to the queue

2. Mark v as visited

BREADTH-FIRST SEARCH

- The general algorithm is as follows (cont'):

    3. while the queue is not empty

        3. Remove vertex u from the queue

        4. Retrieve the vertices adjacent to u

        5. for each vertex w that is adjacent to u
           if w is not visited
              Add w to the queue
              Mark w as visited