DATA STRUCTURES AND ALGORITHMS

- LECTURE 10 -

(2nd year of study)

Prof. Igor Lazov, PhD

Skopje, 2023/24

Contents

10.1. Breath First Search (BFS)

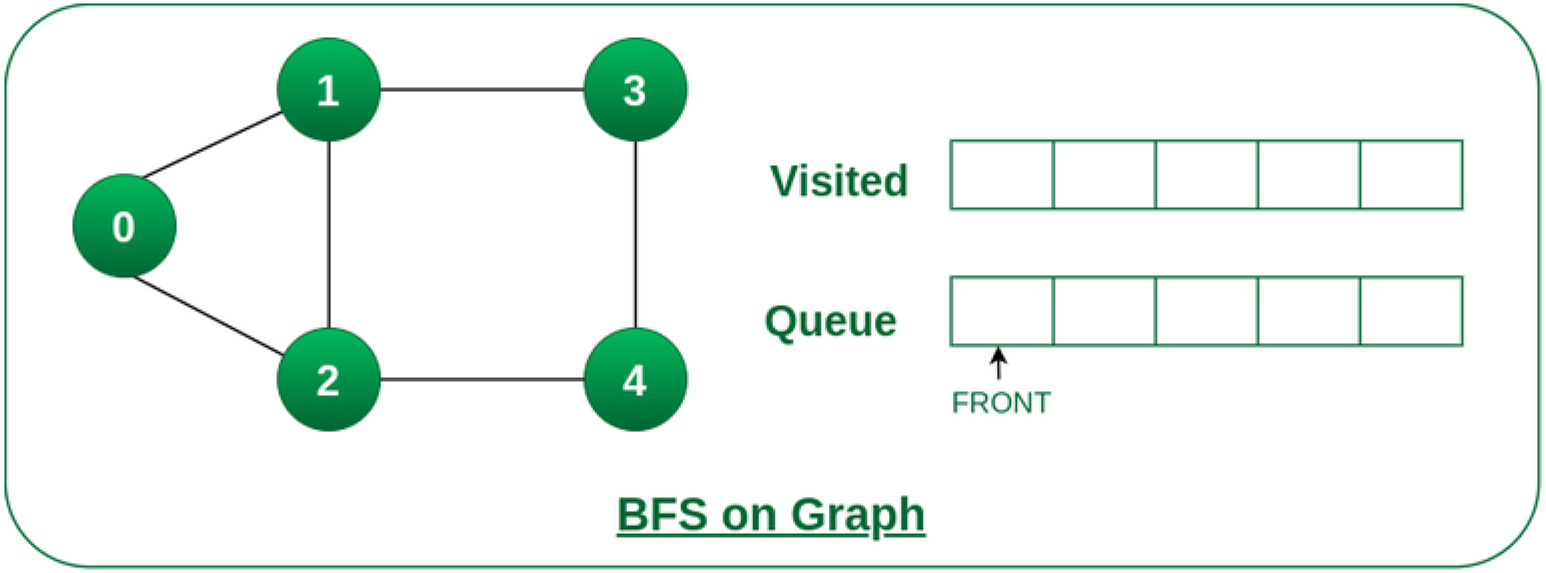
- The **Breadth First Search (BFS)** algorithm is used to search a graph data structure for a node that meets a set of criteria. It starts at the root of the graph and visits all nodes at the current depth level before moving on to the nodes at the next depth level.
  - Breadth-First Traversal (or Search) for a graph is similar to the Breadth-First Traversal of a tree.
  - The only catch here is, that, unlike trees, graphs may contain cycles, so we may come to the same node again. To avoid processing a node more than once, we divide the vertices into two categories: *Visited* and *Not visited*.

10.1. Breath First Search (BFS)

- A boolean visited array is used to mark the visited vertices. For simplicity, it is assumed that all vertices are reachable from the starting vertex. BFS uses a **queue data structure** for traversal.
- Starting from the root, all the nodes at a *particular level* are visited *first*, and then the nodes of the *next level* are traversed till all the nodes are visited.
- To do this a queue is used. All the adjacent unvisited nodes of the current level are pushed into the queue, and the nodes of the current level are marked visited and popped from the queue.

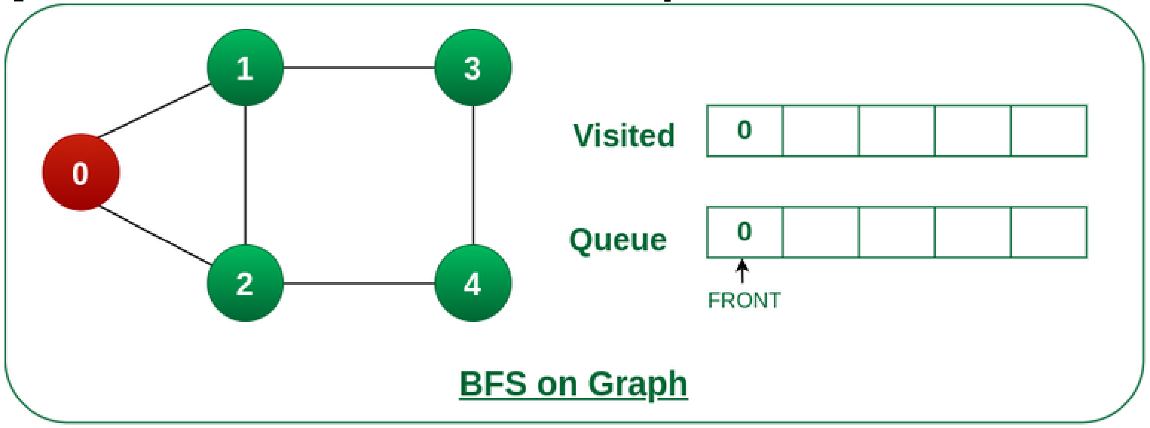
10.1. Breath First Search (BFS)

- Let us understand the working of the algorithm with the help of the following example.
  - **Step1:** Initially *queue* and visited arrays are empty.

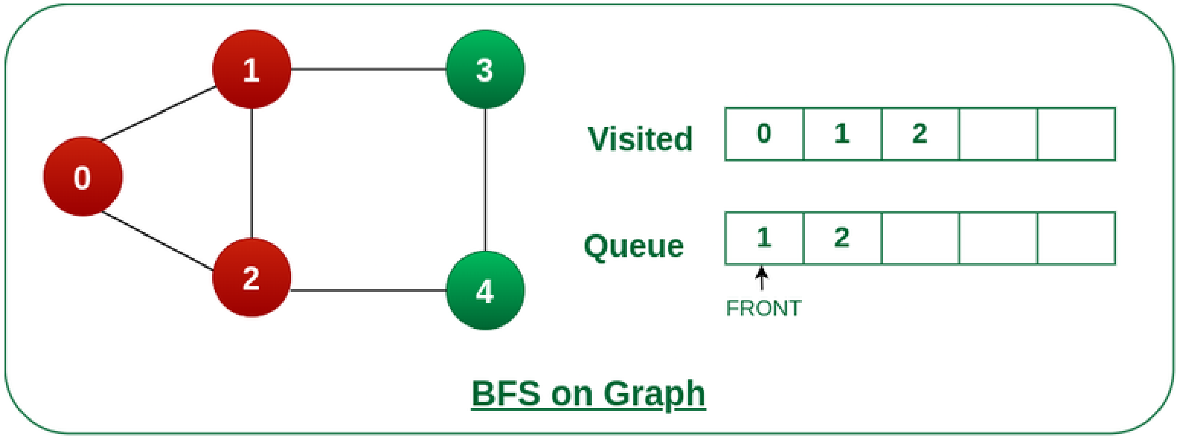


10.1. Breath First Search (BFS)

- **Step2:** Push node 0 into queue and mark it visited.

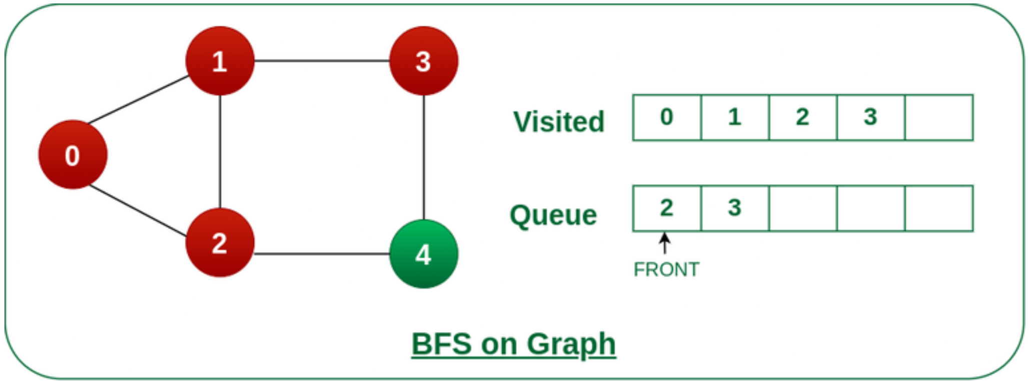


- **Step 3:** Remove node 0 from the front of queue and visit the unvisited neighbours and push them into queue.

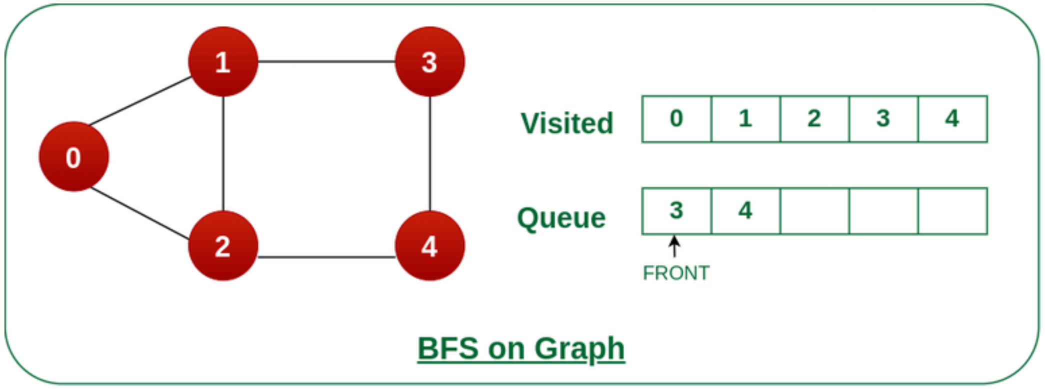


10.1. Breath First Search (BFS)

- **Step 4:** Remove node 1 from the front of queue and visit the unvisited neighbours and push them into queue.

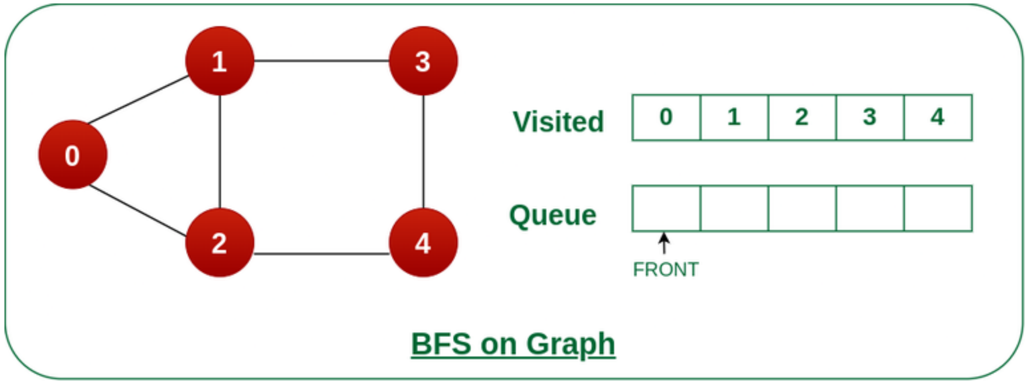


- **Step 5:** Remove node 2 from the front of queue and visit the unvisited neighbours and push them into queue.



10.1. Breath First Search (BFS)

- **Step 6:** Remove node 3 from the front of queue and visit the unvisited neighbours and push them into queue.
  - As we can see that every neighbours of node 3 is visited, so move to the next node that is in the front of the queue.
- **Steps 7:** Remove node 4 from the front of queue and visit the unvisited neighbours and push them into queue.
  - As we can see that every neighbours of node 4 are visited, so move to the next node that is in the front of the queue.



- Now, Queue is empty. So, terminate the iteration process.

10.1. Breath First Search (BFS)

- **Time Complexity:** O(V+E), where V is the number of *nodes*, and E is the number of *edges*.
- **Auxiliary Space:** O(V).
- **Advantages of BFS:**
  - BFS will never get trapped exploring the useful path forever.
  - If there is a solution, BFS will definitely find it.
  - If there is more than one solution, then BFS can find the minimal one that requires less number of steps.
  - Low storage requirement – linear with depth.
- **Disadvantages of BFS:**
  - The main drawback of BFS is its memory requirement.

10.1. Breath First Search (BFS)

- **Applications of Breadth First Search:**
  - **Shortest Path and Minimum Spanning Tree for unweighted graph:** In an unweighted graph, the shortest path is the path with the least number of edges. With Breadth First, we always reach a vertex from a given source using the minimum number of edges. Also, in the case of unweighted graphs, any spanning tree is Minimum Spanning Tree and we can use either Depth or Breadth first traversal for finding a spanning tree.
  - **Minimum Spanning Tree for weighted graphs:** We can also find Minimum Spanning Tree for weighted graphs using BFT, but the condition is that the weight should be non-negative and the same for each pair of vertices.

10.1. Breath First Search (BFS)

- **Peer-to-Peer Networks:** In Peer-to-Peer Networks like *BitTorrent*, BFS is used to find all neighbor nodes.
- **Crawlers in Search Engines:** Crawlers build an index using Breadth First. The idea is to start from the source page and follow all links from the source and keep doing the same. Depth First Traversal can also be used for crawlers, but the advantage of Breadth First Traversal is, the depth or levels of the built tree can be limited.
- **Social Networking Websites:** In social networks, to find people within a given distance 'k' from a person using BFS till 'k' levels.
- **GPS systems:** BFS is used to find all neighboring locations.
- **Broadcasting in Network:** In networks, a broadcasted packet follows BFS to reach all nodes.

10.2. Depth First Search (DFS)

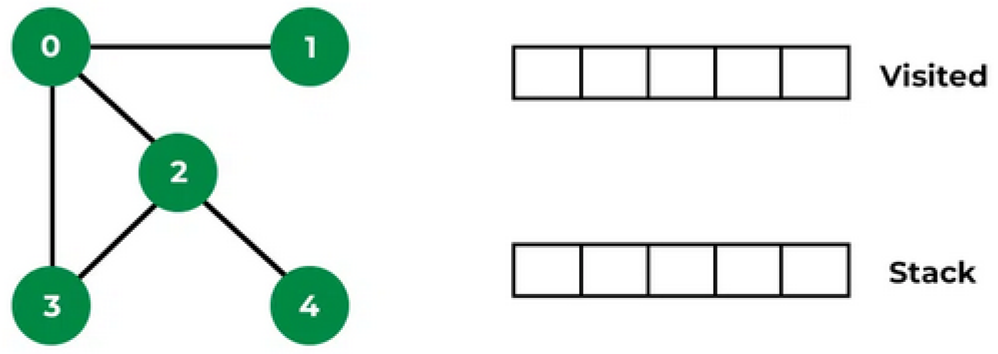
- **Depth First Search (or, DFS)** for a graph is similar to *Depth First Traversal of a tree*.
- The only catch here is, that, unlike trees, graphs may contain cycles (a node may be visited twice).
- To avoid processing a node more than once, use a boolean visited array.
- A graph can have more than one DFS traversal.

10.2. Depth First Search (DFS)

- Depth-first search is an algorithm for traversing or searching tree or graph data structures.
- The algorithm starts at the root node (selecting some arbitrary node as the root node in the case of a graph) and explores as far as possible along each branch before backtracking.
- DFS uses a **stack data structure** for traversal.
- Let us understand the working of ***Depth First Search*** with the help of the following illustration:

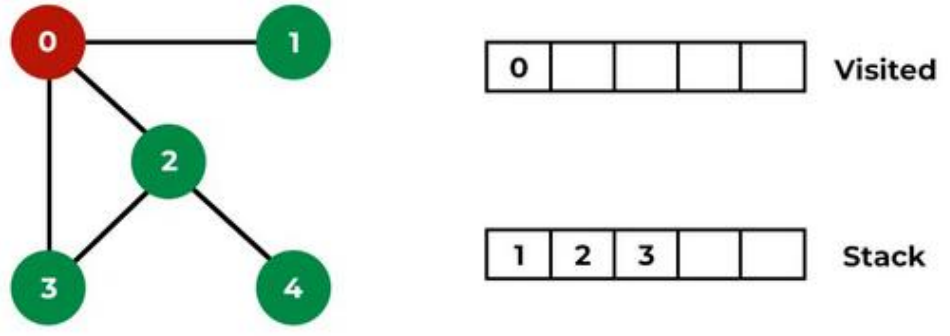
10.2. Depth First Search (DFS)

- **Step1:** Initially stack and visited arrays are empty.



DFS on Graph

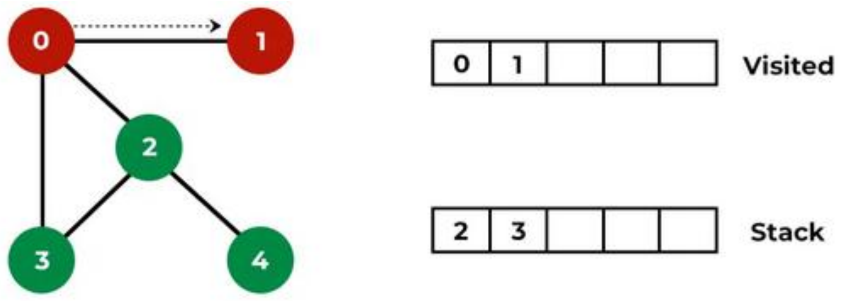
- **Step 2:** Visit 0 and put its adjacent nodes which are not visited yet into the stack.



DFS on Graph

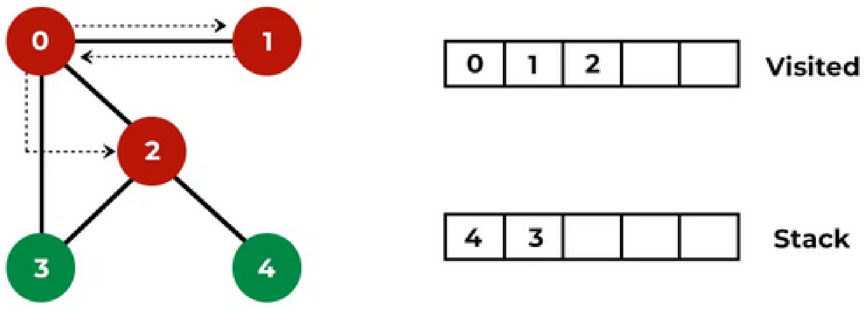
10.2. Depth First Search (DFS)

- **Step 3:** Now, Node 1 at the top of the stack, so visit node 1 and pop it from the stack and put all of its adjacent nodes which are not visited in the stack.



DFS on Graph

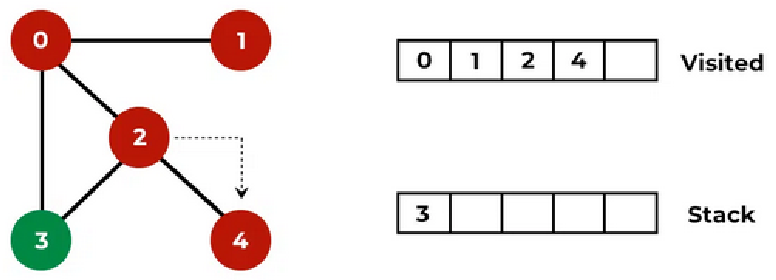
- **Step 4:** Now, Node 2 at the top of the stack, so visit node 2 and pop it from the stack and put all of its adjacent nodes which are not visited (i.e, 3, 4) in the stack.



DFS on Graph

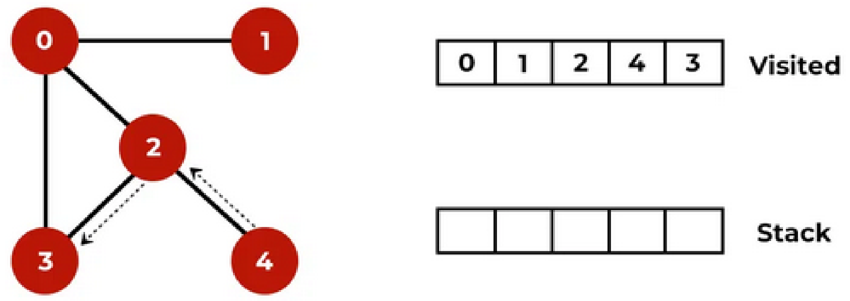
10.2. Depth First Search (DFS)

- **Step 5:** Now, Node 4 at the top of the stack, so visit node 4 and pop it from the stack and put all of its adjacent nodes which are not visited in the stack.



DFS on Graph

- **Step 6:** Now, Node 3 at the top of the stack, so visit node 3 and pop it from the stack and put all of its adjacent nodes which are not visited in the stack.



DFS on Graph

10.2. Depth First Search (DFS)

- Now, Stack becomes empty, which means we have visited all the nodes, and our DFS traversal ends.
- **Time complexity:** O(V+E), where V is the number of *vertices* and E is the number of *edges* in the graph.
- **Auxiliary Space:** O(V+E), since an extra visited array of size V is required, and stack size for iterative call to DFS function.

10.2. Depth First Search (DFS)

- **Advantages of DFS:**
  - Memory requirement is only linear with respect to the search graph. This is in contrast with BFS which requires more space. The reason is that the algorithm only needs to store a stack of nodes on the path from the root to the current node.
  - Practically DFS is time-limited, rather than space-limited.
  - If BFS finds solution without exploring much in a path, then the time and space it takes will be very less.
  - DFS requires less memory since only the nodes on the current path are stored. By chance, DFS may find a solution without examining much of the search space at all.

10.2. Depth First Search (DFS)

- **Disadvantages of DFS:**
  - The disadvantage of DFS is that there is a possibility that it may down the left-most path forever. Even a finite graph can generate an infinite tree. One solution to this problem is to impose a cutoff depth on the search. Although ideal cutoff is the solution depth *d* and this value is rarely known in advance of actually solving the problem. If the chosen cutoff depth is less than *d*, the algorithm will fail to find a solution, whereas if the cutoff depth is greater than *d*, a large price is paid in execution time, and the first solution found may not be an optimal one.
  - Depth-First Search is not guaranteed to find the solution.
  - There is no guarantee to find a minimal solution, if more than one solution.

10.2. Depth First Search (DFS)

- **Applications of Depth First Search:**
  - **Detecting cycle in a graph:** A graph has a cycle if and only if we see a back edge during DFS. So we can run DFS for the graph and check for back edges.
  - **Path Finding:** We can specialize the DFS algorithm to find a path between two given vertices *u* and *z*.
    - Call DFS(G, u) with *u* as the start vertex.
    - Use a stack S to keep track of the path between the start vertex and the current vertex.
    - As soon as destination vertex *z* is encountered, return the path as the contents of the stack.

10.2. Depth First Search (DFS)

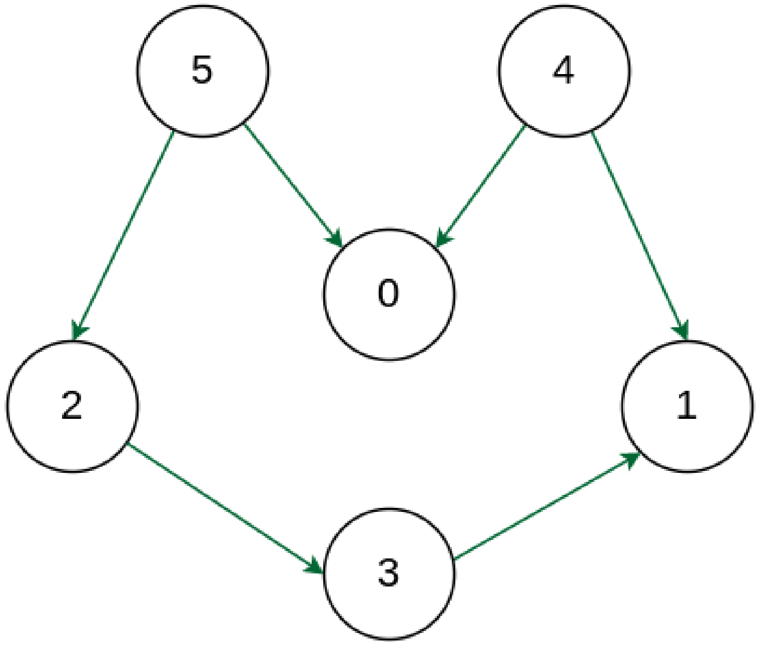
- **Finding Strongly Connected Components of a graph:** A directed graph is called strongly connected if there is a path from each vertex in the graph to every other vertex.
- **Solving puzzles with only one solution:** such as mazes (DFS can be adapted to find all solutions to a maze by only including nodes on the current path in the visited set).
- **Web crawlers:** DFS can be used in the implementation of web crawlers to explore the links on a website.
- **Maze generation:** DFS can be used to generate random mazes.
- **Model checking:** Depth-first search can be used in model checking, which is the process of checking that a model of a system meets a certain set of properties.
- **Backtracking:** DFS can be used in backtracking algorithms.

10.3. Topological Sorting

- **Topological Sorting:** Topological Sorting is mainly used for scheduling jobs from the given dependencies among jobs. In computer science, applications of this type arise, e.g., in instruction scheduling.
- The Topological sorting for **Directed Acyclic Graph (DAG)** is a linear ordering of vertices, such that for every directed edge *u-v*, vertex ***u*** comes before **v** in the ordering.
- **Note:** Topological Sorting for a graph is not possible if the graph is not a **DAG.**

10.3. Topological Sorting

- **Example**
  - **Input:** Graph
  - **Output:** 5 4 2 3 1 0
- *Explanation*: The first vertex in topological sorting is always a vertex with an in-degree of 0 (a vertex with no incoming edges).
- A topological sorting of the following graph is "5 4 2 3 1 0".
- There can be more than one topological sorting for a graph.
- Another sorting is "4 5 2 3 1 0".



10.3. Topological Sorting

- **Topological Sorting vs Depth First Traversal**
  - In ***DFS***, we print a *vertex* and then recursively call DFS for its adjacent vertices.
  - In ***topological sorting***, we need to print a *vertex* before its adjacent vertices.
  - *For example*, in the above given graph, the vertex '5' should be printed before vertex '0', but unlike DFS, the vertex '4' should also be printed before vertex '0'. So, topological sorting is different from DFS.
  - *For example*, a DFS of the shown graph is "5 2 3 1 0 4", but it is not a topological sorting.