Graph Traversal Algorithms

Dr Ian Cornelius

Hello

Hello (1)

Learning Outcomes

1. Understand the concept of different algorithms for traversing a graph
2. Implement and use a graph traversing algorithm in their bodies of work

Graph Traversal

Graph Traversal (1)

Traversing a Graph

- Storing data within a graph is fairly easy
- However, they are not useful until you can search for data or find information about the interconnections
- Algorithms that interrogate the graph by following the edges are known as **traversal algorithms**

Graph Traversal (2)

Principles for Traversing a Graph

1. Start from a *root* node
    - this may be specific to the problem
        - i.e. the starting address in a route planner
    - or it could be the nth node in the graph
    - or it may be something intelligently selected based on the search parameters
        - i.e. looking for restaurants near-by in a graph consisting of points of interest, and there is a starting node that is specific to *restaurant* searches
2. There is a *goal* node
    - this may be the address you are trying to navigate to in the route planner
    - it may be parameters defining a goal node or a set of goal nodes
        - i.e. a list of universities at least 200 miles away from my parents' house
3. The graph is traversed from node to node, along the edges that connect them
    - this is done until the goal is reached
    - or a satisfactory node has met the parameters
    - or a large set of nodes which meet the parameters are found

Graph Traversal (3)

Approaches for Traversing a Graph

- There are various approaches to solving this problem of traversing a graph
  - however, there is no best method of doing this
- The nature of the data you are after matters when choosing a graph traversal algorithm
- The structure of the graph also matters
- The desired output *also* matters
  - this could be a single specific node
  - the first node that meets a set of parameters
  - several nodes that meet the set of parameters

Graph Traversal Algorithms

Graph Traversal Algorithms (1)

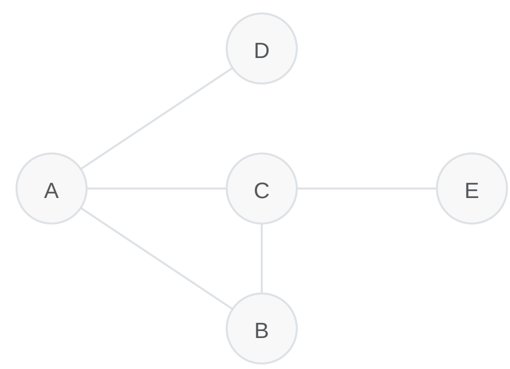
Depth-First Search (DFS)

- Standard DFS implementation will put each vertex of the graph into one of two categories:
  1. Visited
  2. Unvisited
- The algorithm will mark each vertex as visited whilst avoiding cycles
- The algorithm consists of the following steps:
  1. Add any of the graph vertices on top of a stack
  2. Take the top item of the stack and add it to the visited list
  3. Create a list of the adjacent vertices for that node
     - add any that are not in the visited list to the top of the stack
  4. Keep repeat steps 2 and 3 until the stack is empty

Graph Traversal Algorithms (2)

DFS on an Undirected Graph

- Click **Start** to proceed!



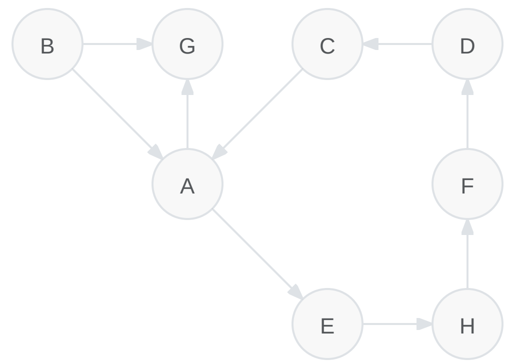
Start

Unvisited **Visited**

Graph Traversal Algorithms (3)

DFS on a Directed Graph

- Click **Start** to proceed!



Start

Unvisited

Visited

Graph Traversal Algorithms (4)

Time Complexity of DFS

- The time complexity of DFS is represented in the form $O(V + E)$, where
  - V is the number of nodes
  - E is the number of edges
- The space complexity of DFS is $O(V)$, where V is the number of nodes
- DFS is useful in the following applications:
  - detecting a cycle in a graph
  - topological sorting
  - finding *strongly connected* components of a graph
  - path finding

Graph Traversal Algorithms (5)

Breadth-First Search (BFS)

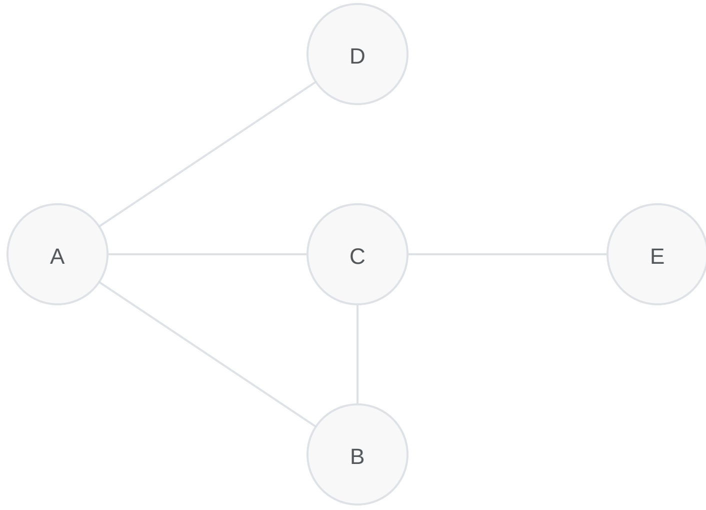
- Standard BFS implementation will put each vertex of the graph into one of two categories:
  1. Visited
  2. Not Visited (Queue)
- The algorithm will mark each vertex as visited whilst avoiding cycles
- The algorithm consists of the following steps:
  1. Start by putting any one of the graph’s vertices to the back of a queue
  2. Take the front item of the queue and add it to the visited list
  3. Create a list of those vertices adjacent nodes and add any which are not in the visited list to the back of the queue
  4. Keep repeating steps 2 and 3 until the queue is empty

Graph Traversal Algorithms (6)

BFS on an Undirected Graph

- Click **Start** to proceed!

Unvisited **Visited**



Start

Graph Traversal Algorithms (7)

Time Complexity of BFS

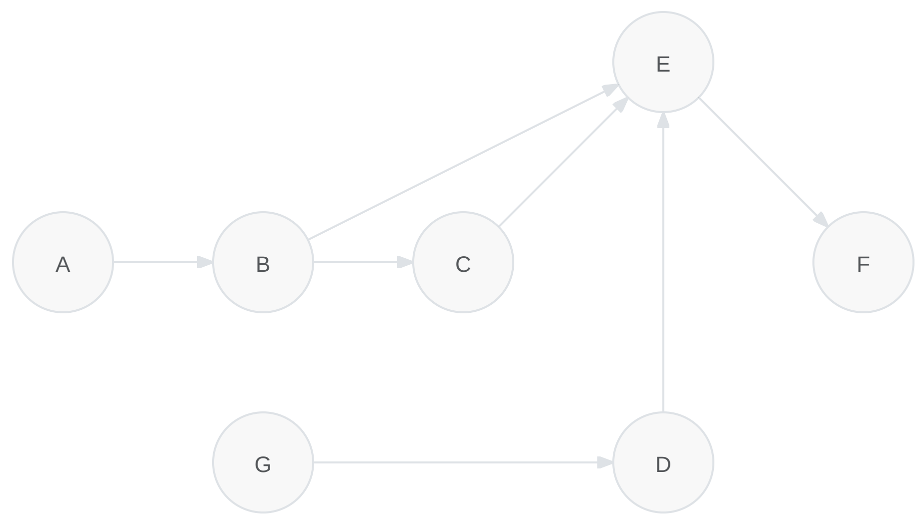
- The time complexity of BFS is represented in the form $O(V+E)$, where
  - V is the number of nodes
  - E is the number of edges
- The space complexity of BFS is $O(V)$, where V is the number of nodes
- BFS is useful in the following applications:
  - shortest path in a graph (the least number of edges)
  - peer to peer (P2P) networks
  - search engine crawlers

Graph Theory Continued

Graph Theory Continued (1)

Recap: Directed Acyclic Graphs (DAG)

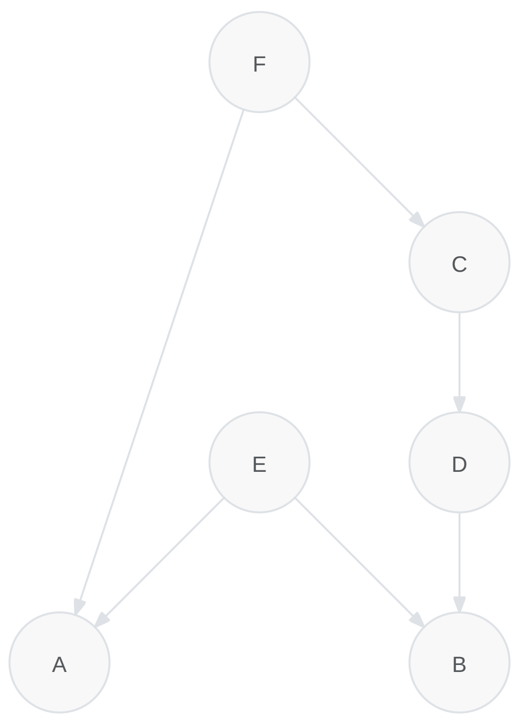
- DAGs are *directed* graphs with no cycles
  - hence the term **acyclic**
- Testing for no cycles can be achieved by:
  - a Depth-First Search (DFS)
  - if a directed graph has a cycle, then a back arc will always be encountered in any depth-first search of the graph



Graph Theory Continued (2)

Topological Sorting

- A process of assigning a linear order to the vertices of a DAG
- Time complexity is $O(M+N)$, where
  - M is the number of edges
  - N is the number of nodes
- Follows a set order of principles:
  1. Identify a node with no incoming connections
  2. Add that node to the topological sort list
  3. Remove the node from the graph
  4. Repeat

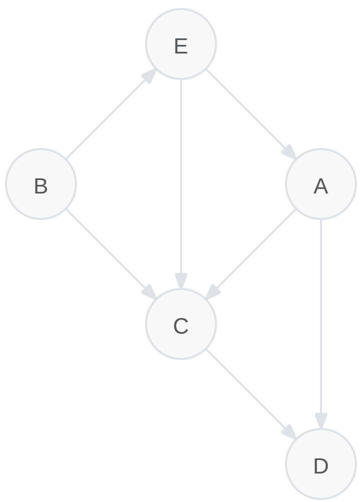


- Topological Sort:
  - `['F', 'C', 'D', 'B', 'A', 'E']`
  - `['F', 'C', 'D', 'B', 'E', 'A']`

Graph Theory Continued (3)

Walkthrough: Topological Sorting

- Click **Start** to proceed!

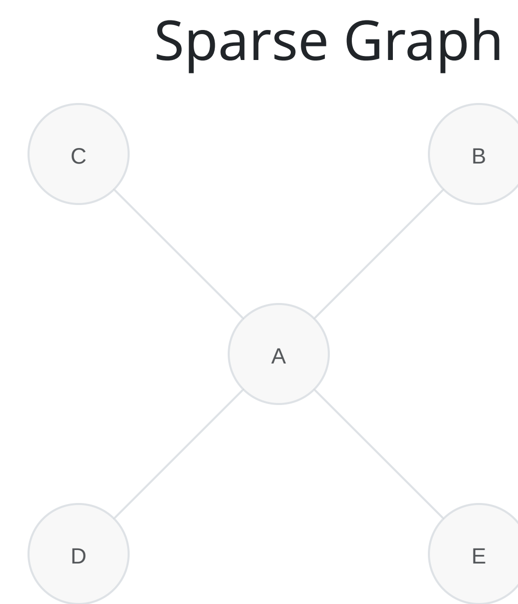


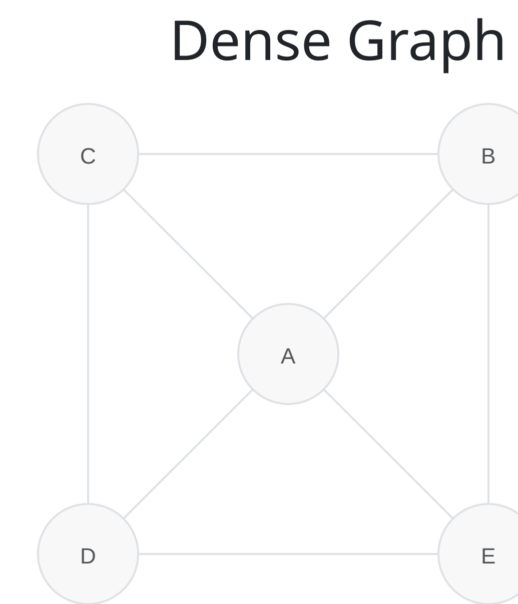
Start

Graph Theory Continued (4)

Density of Graphs

- Graph density tells us how *full* the graph is
  - i.e. how many connections exist in relation to the number of nodes
- To formulate how dense a graph is, we need to know the **size** and **order** of a graph
  - the size is the number of edges, $|E|$
  - the order is the number of vertices, $|V|$





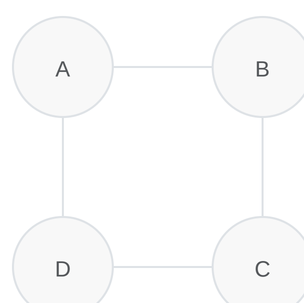
Graph Theory Continued (5)

Spanning Trees

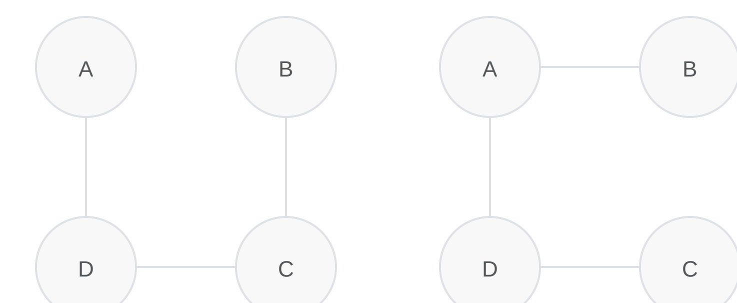
- A spanning tree is a subgraph of an undirected, connected graph
  - it will include all vertices of the graph with a minimum possible number of edges
  - edges may contain weights or not
- The total number of spanning trees is equal to $n^{(n-2)}$
  - where n, is the number of vertices

Example of a Spanning Tree

- Normal Graph



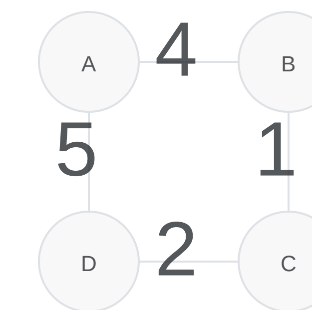
- Sub-Graphs



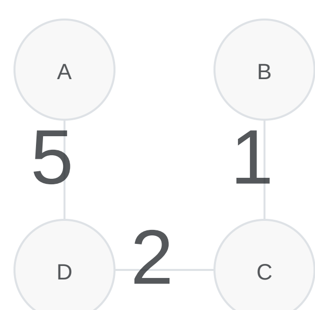
Graph Theory Continued (6)

Minimum Spanning Tree (MST)

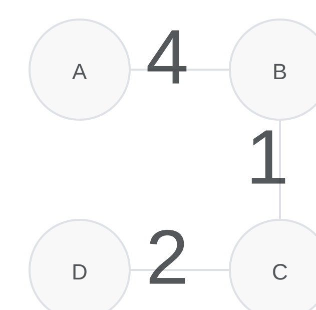
- A **minimum** spanning tree is a spanning tree whereby the minimum is the sum of weights that is the smallest



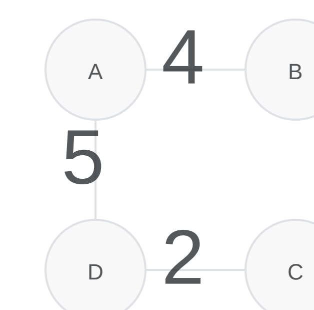
Sub-Graphs



- Sum of Weights: `8`



- Sum of Weights: `7`



- Sum of Weights: `11`

Dijkstra’s Algorithm

Dijkstra's Algorithm (1)

- An algorithm to find the shortest path between nodes in a graph
- Produces the shortest path tree
- Works on both **directed** and **non-directed** graphs
  - one condition: edges must have a non-negative weight
- Simply, it is used to find the **shortest** path with the **lowest** cost
- Dijkstra's Algorithm can be applied to the following:
  - Google Maps
  - IP Routing
  - Word Ladder Puzzles
  - Social Network Analysis

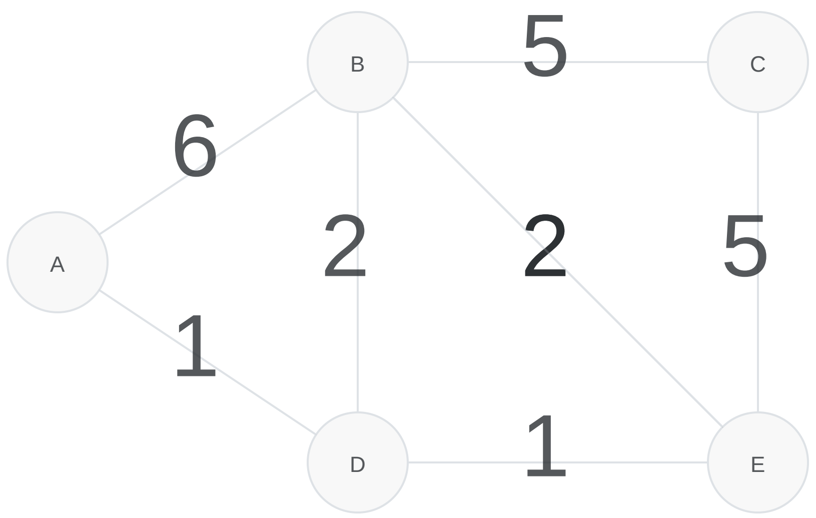
Dijkstra's Algorithm (2)

Algorithm Steps

1. Mark all nodes in the graph as unvisited
2. Pick a starting node, and set the current distance as 0 and all other nodes with infinity
3. Select the starting node and mark it as the current selected node
   - for this node, analyse all of its neighbours and measure their distances by summing the current distance
   - this is done with the current node with the weight of the edge to the neighbouring node
4. Compare the measured distance with the current distance assigned to the neighbouring node
   - mark this as the new current distance for the neighbouring node
5. Consider all the unvisited neighbouring nodes, and mark the current node as visited
   - **if** the destination node has been marked as visited, then stop
   - **else** choose an unvisited node marked with the least distance;
     - select it as the new current node and repeat the process from step 3

Dijkstra's Algorithm (3)

Walkthrough of Dijkstra



Unvisited **Visited**

Start

Goodbye

Goodbye (1)

Questions and Support

- Questions? Post them on the **Community Page** on Aula
- Additional Support? Visit the [Module Support Page]
- Contact Details:
  - Dr Ian Cornelius, ab6459@coventry.ac.uk