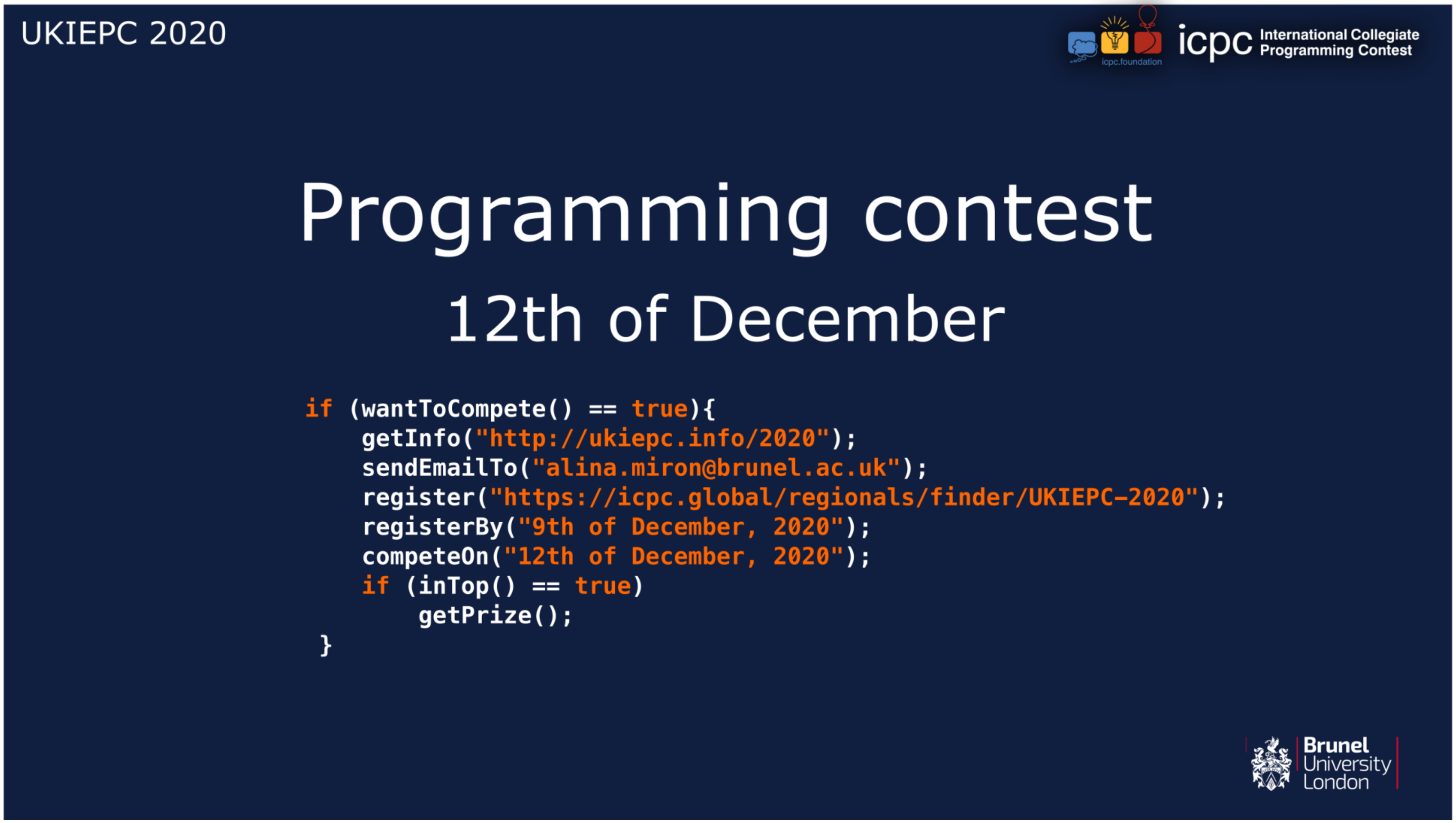
Algorithms and their Applications CS2004 (2020-2021)

Dr Mahir Arzoky

7.1 Classic Algorithms - Graph Traversal

Notices



UKIEPC 2020

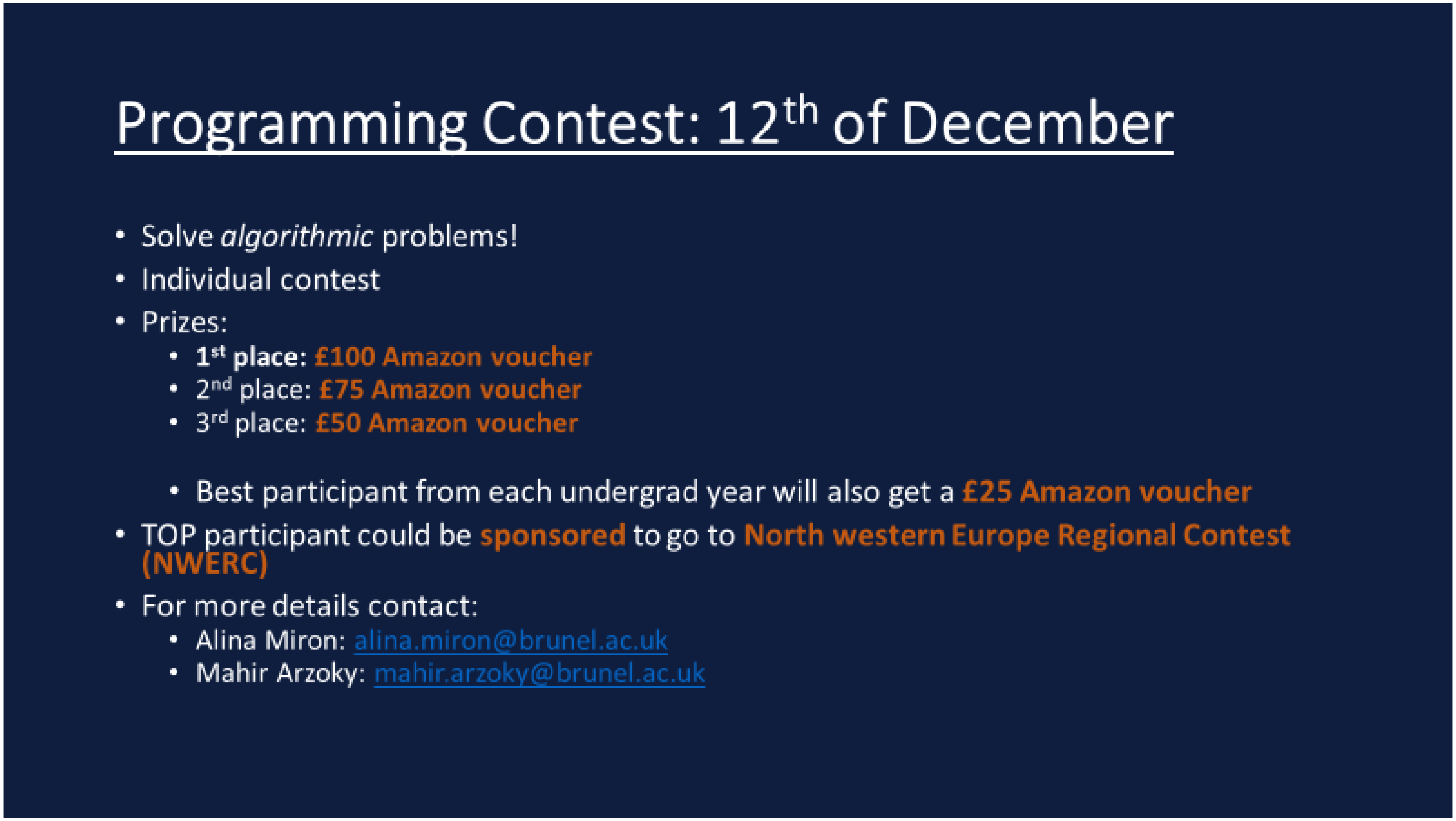
icpc.foundation icpc International Collegiate Programming Contest

Programming contest

12th of December

```
if (wantToCompete() == true){
    getInfo("http://ukiepc.info/2020");
    sendEmailTo("alina.miron@brunel.ac.uk");
    register("https://icpc.global/regionals/finder/UKIEPC-2020");
    registerBy("9th of December, 2020");
    competeOn("12th of December, 2020");
    if (inTop() == true)
        getPrize();
 }
```

Brunel University London



Programming Contest: 12th of December

- Solve *algorithmic* problems!
- Individual contest
- Prizes:
    - **1st place: £100 Amazon voucher**
    - 2nd place: **£75 Amazon voucher**
    - 3rd place: **£50 Amazon voucher**

    - Best participant from each undergrad year will also get a **£25 Amazon voucher**
- TOP participant could be **sponsored** to go to **North western Europe Regional Contest (NWERC)**
- For more details contact:
    - Alina Miron: alina.miron@brunel.ac.uk
    - Mahir Arzoky: mahir.arzoky@brunel.ac.uk

Class Test CR II
will be released today!

Previously On CS2004…

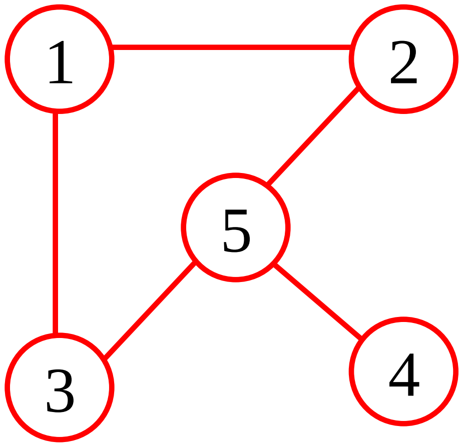
- So far we have looked at:
  - Concepts of Computation and Algorithms
  - Comparing algorithms
  - Some mathematical foundation
  - The Big-Oh notation
  - Computational Complexity
  - Data structures
  - Last lecture we also looked at Sorting Algorithms

Graph Traversal

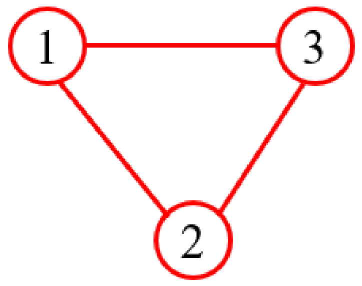
- Within this lecture we are going to look at a number of classic graph traversal algorithms:
  - Depth-First
  - Exhaustive Search
  - Breadth-First
  - A* (A-Star)
  - Minimum Spanning Trees

Recap – What is a Graph – Part 1

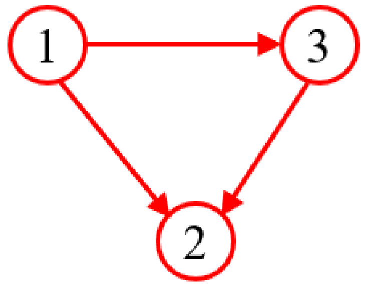
- A graph $G = (V,E)$ is composed of:
  - V: a set of vertices or nodes
  - E: a set of edges or lines connecting the vertices in V
  - An edge $e=(u,v)$ is a connection between the vertices u and v
- In the example:
  - $V=\{1,2,3,4,5\}$
  - $E=\{(1,2),(1,3),(3,5),(2,5),(5,4)\}$



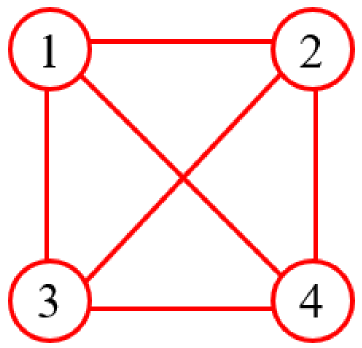
Recap – What is a Graph – Part 2



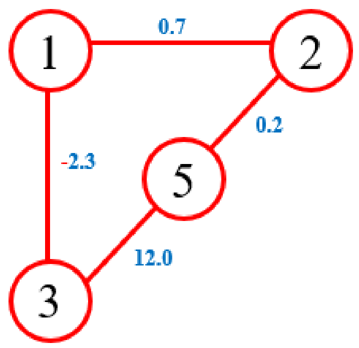
Undirected
(Edges go both ways)



Directed
(Edges have arrows)



Complete
(There is an edge between all nodes)



Weighted
(Each edge has a value associated to it)

What is “Search” in Graphs?

- Start at the source node and search until we find the target node
- When we work with graphs, we often wish to do something to each node in the graph exactly once
  - Visit each node **once and only once**
- Examples:
  - A piece of information that needs to be distributed to all the computers on a network
  - Look for information among computer in a network
- We will cover the concept of a “general” search in more detail in future lectures

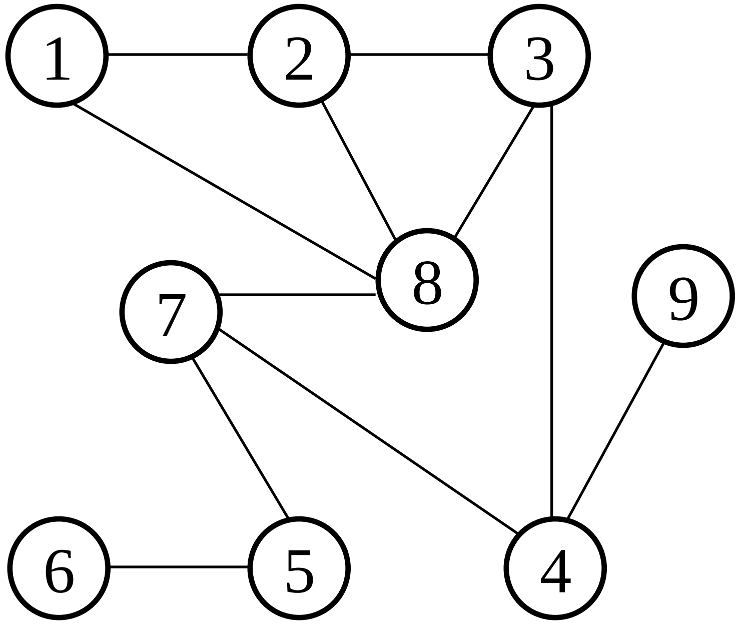
What is “Depth-First”?

- ❑ Depth-First: the traversal will go as far as possible down a path until a “**Dead End**” is reached
- ❑ In an undirected graph:
  - ❑ A node is a dead end if all of the nodes adjacent to it have already been visited
- ❑ In a directed graph:
  - ❑ If a node has no outgoing edges, it is called a dead end

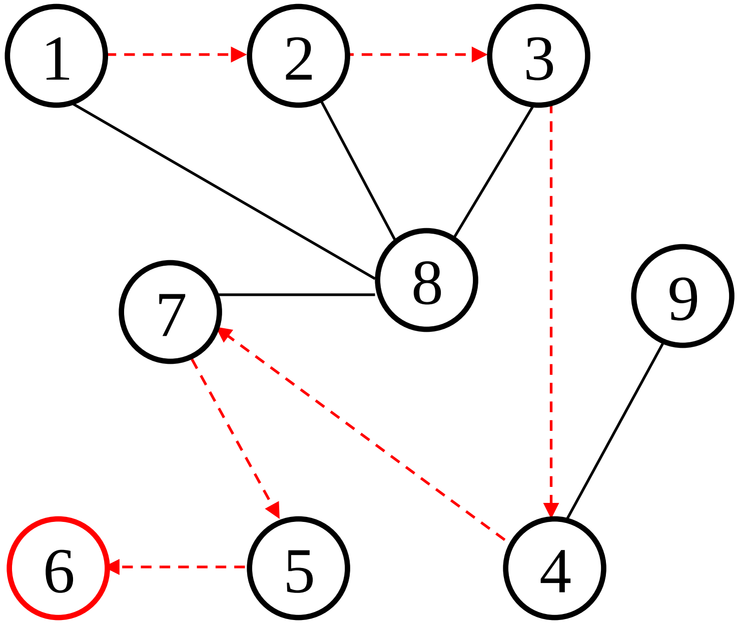
What is “Depth-First Search” (DFS)?

1. We visit the starting node and then proceed to follow links through the graph until we reach a dead end
2. We then back up along our path until we find an unvisited adjacent node, and continue in that new direction
3. The process completes when we back up to the starting node and all the nodes adjacent to it have been visited
4. If presented with two choices, we choose the node with numerically and/or alphabetically smaller label (just for convenience)

DFS Undirected Graphs – Part 1

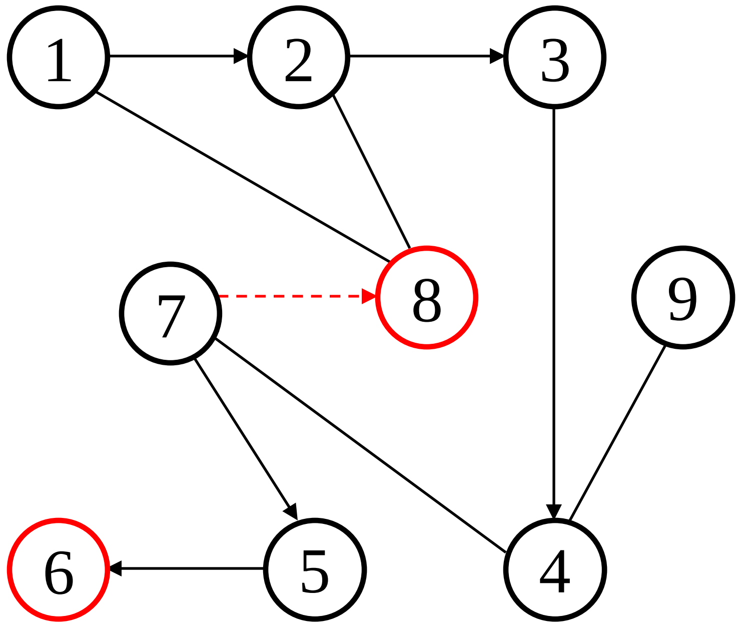


Depth-first search

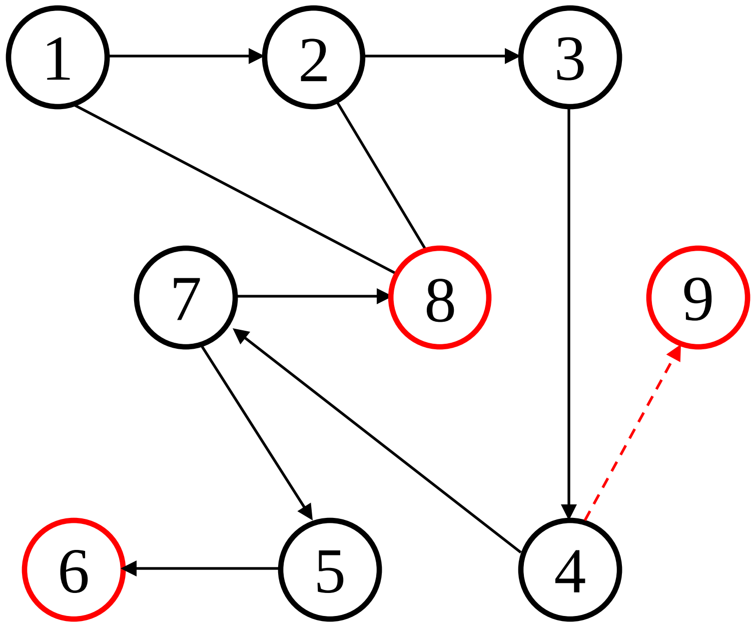


1,2,3,4,7,5,6, then backtrack

DFS Undirected Graphs – Part 2

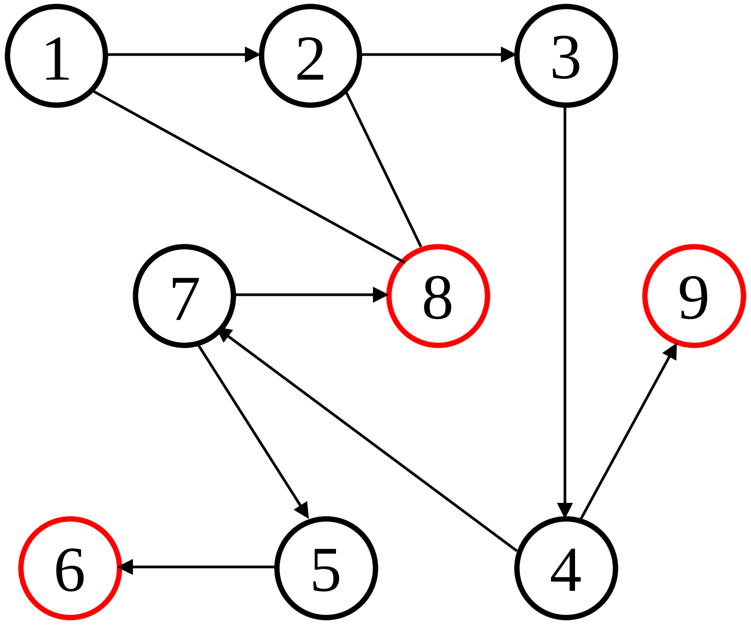


Backtrack but don’t visit nodes 5,7, and find node 8 hasn’t been visited



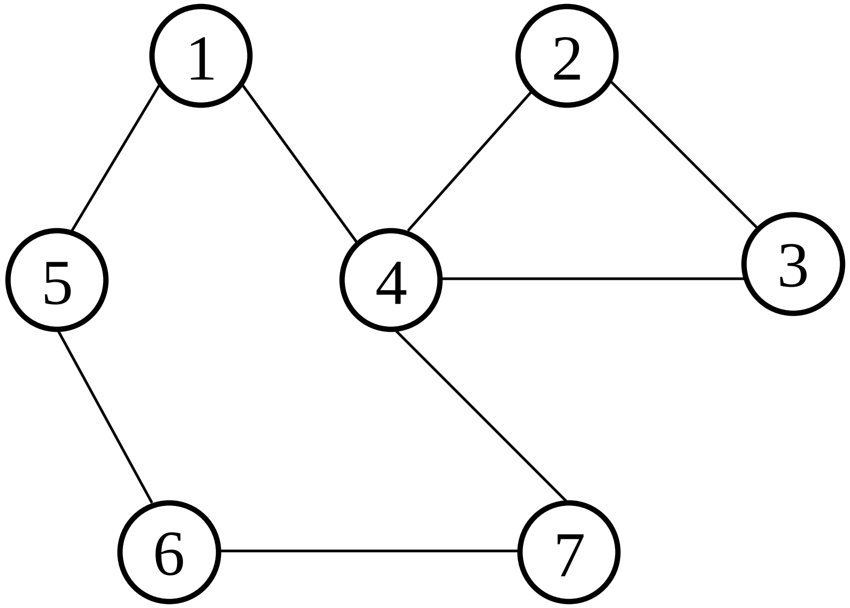
Backtrack, don’t visit nodes 7, 4, until node 9 that hasn’t been visited

DFS Undirected Graphs – Part 3



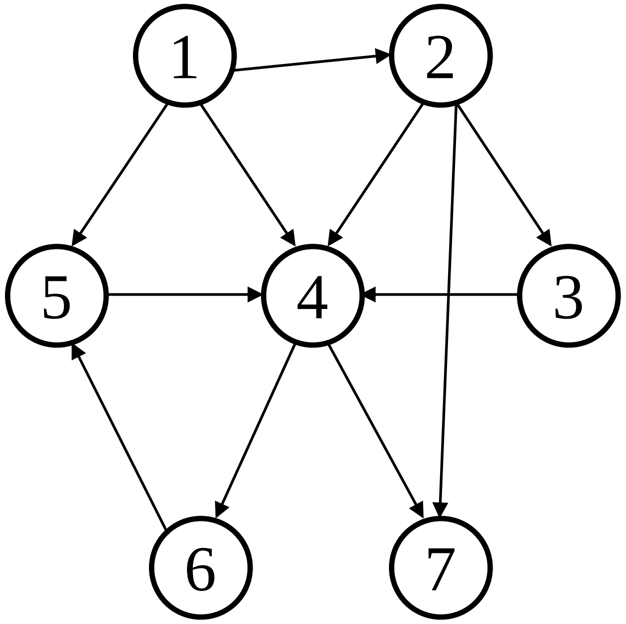
- We continue to back up, without visiting nodes 4,3,2, until we reach the starting node
- Since all nodes adjacent to node 1 have been visited, we are done
- The order that the nodes are visited: 1,2,3,4,7,5,6,8,9

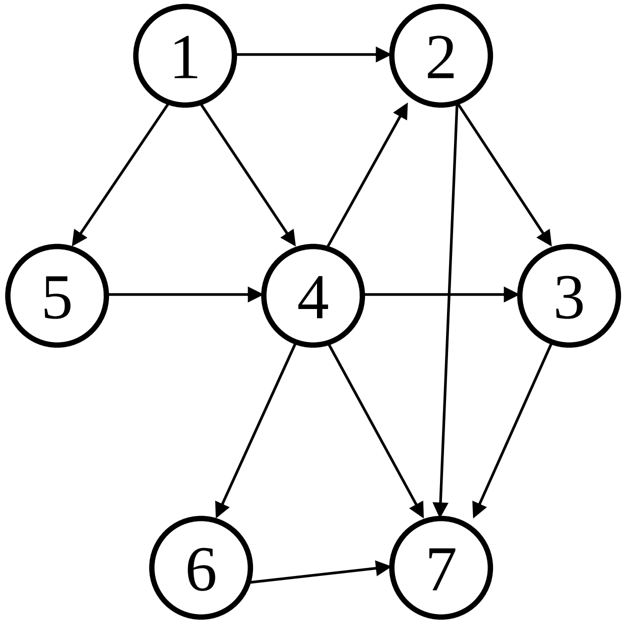
DFS Undirected Graphs – Part 4



- ❑ Starting from the node 1, what is the order that the nodes will be first visited?
- ❑ 1,4,2,3,7,6,5

DFS Directed Graphs





The order that the nodes will be first visited: 1,2,3,4,6,5,7

If undirected: 1,2,3,4,5,6,7

The order that the nodes will be first visited: 1,2,3,7,4,6,5

If undirected: 1,2,3,4,5,6,7

Algorithm DFS(G, n)

```
Algorithm 1. DFS(G,n)
Input: G- the graph
       n- the current node
1) Visit(n) [Do something to/at node n...]
2) Mark(n)
3) For every edge nm (from n to m) in G do
4)    If m is not marked then
5)       DFS(G,m)
6)    End If
7) End For
Output: Depends on the purpose of the
        search...
```

DFS Running Time Analysis

- Remember:
  - DFS is called on each node exactly once
  - n nodes : $O(n)$
  - Every edge is examined exactly twice, once from each of its vertices/nodes
  - m edges: $O(2m)$
- $O(n) + O(2m) \equiv O(n+m)$
- If we assume that the work done as we visit each node is the most complex part of the process, the traversal is of order $O(n)$

Other Types of Graph “Search”

- ❑ There are other ways to search a graph other than visiting all nodes
- ❑ Finding a path between two nodes
- ❑ Finding the shortest path between two nodes (number of edges)
- ❑ Finding the cheapest path between two nodes (a function of edge weight)

The Exhaustive Search

- ❑ The exhaustive search systematically evaluates every possible path in a graph
- ❑ This methods is guaranteed to find what we are looking for
- ❑ However this method is unsuitable for most real world problems

Algorithm Exhaustive(G, n, p)

```
Algorithm 2. Exhaustive(G, n, p)
Input: G- the graph
       n- the current node
       p- the path so far
1) For every edge nm (from n to m) in G do
2)     If $m \notin p$ then
3)        $p = p \cup \{m\}$
4)        Exhaustive(G, m, p)
5)     End If
6) End For
Output: Depends on the purpose of the
        search...
```

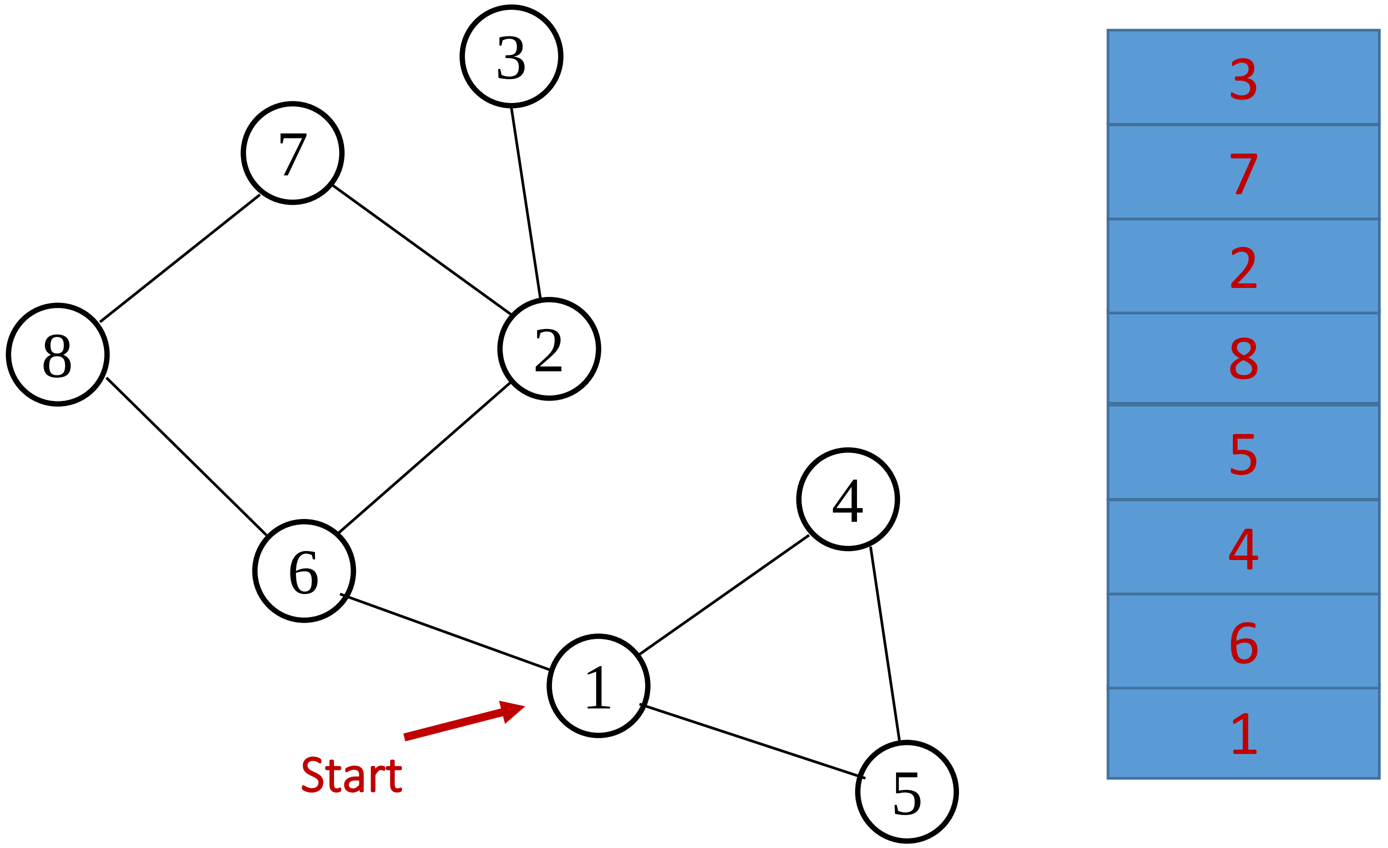
Exhaustive Running Time

- This is very complicated to compute
- It depends on how the nodes and edges are organised
- If we look at the worse case, that of a complete graph:
  - Here all the nodes are connected to each other
  - For the first node in the path there are n-1 edges to follow
  - For the second, n-2
  - For the third, n-3, ...
  - Hence we get $(n-1)(n-2)(n-3)\ldots1 = O((n-1)!)$

Breadth-First Search

- This search method considers neighbouring nodes first
    - All the neighbours of the start node are expanded first
    - Then the neighbours of the neighbours
    - Until the goal state is found
- This search is very expensive since all the partial paths being considered must be stored
- Similar to depth-first search, breadth-first search will eventually find a path to the goal, but it may not be the best path
- The algorithm is similar to the exhaustive search algorithm, but it stops when the goal node is reached
    - Exhaustive generates all paths

Breadth-First Search



Best-First Search

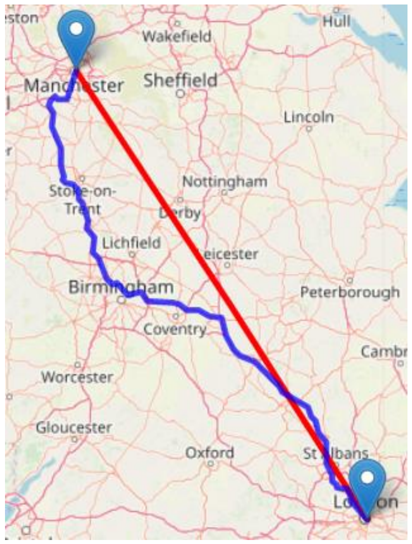
- Best-first search is an improvement upon depth-first search
- A heuristic is used to decide which node is explored next
    - A **heuristic** is a rule of thumb or best practice
    - We will look at heuristics in much more detail later
- For example nodes are expanded in order of lowest cost

A* Search – Part 1

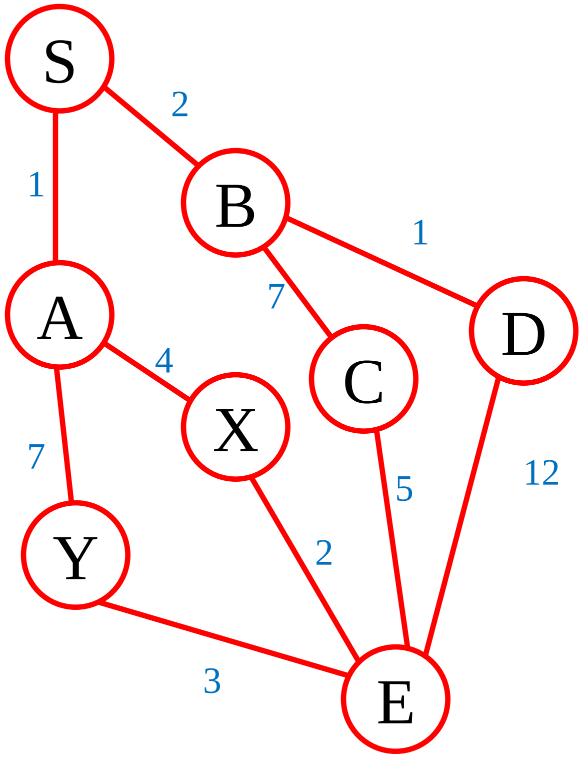
- The A* (A Star) search method is an example of a best-first search
- Very good for route finding applications
  - The AA website
  - The tube
- In order for the search to work, two estimates must be available
  - How much a partial path has cost
  - An estimate (**an under estimate**) of how far is left to the goal state
  - E.g. a lower bound on how far to travel
- The A* algorithm will find an optimal path without necessarily considering all the routes

A* Search – Part 2

- The algorithm scores each partial path with the following function: ***f* = *g* + *h***
- Where ***g*** is the cost so far and ***h*** is the estimate of the cost to the goal state. (E.g. as the crow-flies distance)
  - London to Manchester: Land transport 200.21 miles, crow-flies 163.05 miles
- If ***h*** is always zero (i.e. we do not have any information about how far away the goal is), then the algorithm becomes similar to depth-first search
- If ***g*** is always zero, then the algorithm is called a ***Greedy Search***



A* Search Example



- Values for *h* (distance to E):
  A:5, B:6, C:4, D:15, X:5, Y:8

Expand S – two choices

[S,A] *f*=1+5=6

[S,B] *f*=2+6=8

Expand [S,A] – two choices

[S,B] *f*=2+6=8

[S,A,X] *f*=(1+4)+5=10

[S,A,Y] *f*=(1+7)+8=16

Expand [S,B] – two choices

[S,A,X] *f*=(1+4)+5=10

[S,B,C] *f*= (2+7)+4=13

[S,A,Y] *f*=(1+7)+8=16

[S,B,D] *f*= (2+1)+15=18

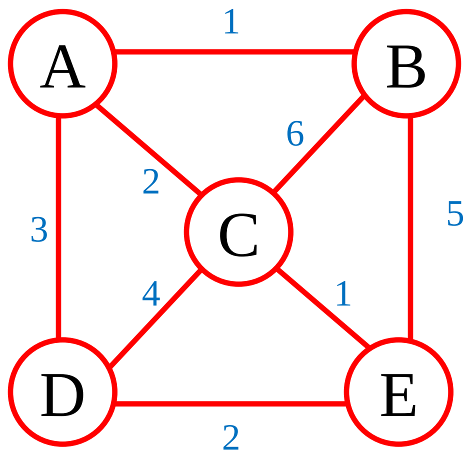
Expand [S,A,X] – one choice (E - GOAL!)

[S,A,X,E] is the best path... (costing 7)

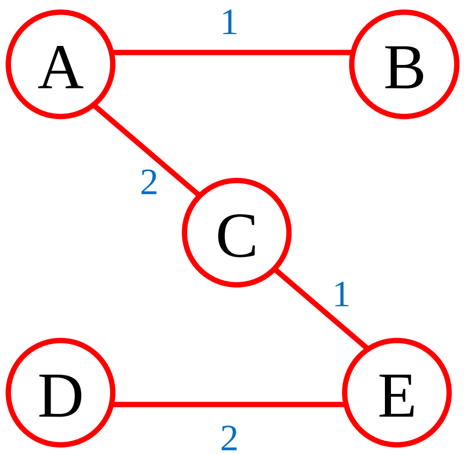
Minimum Spanning Trees

- ❑ A spanning tree is a sub-graph that is also a tree (no cycles) that contains all of the nodes of the super-graph
  - ❑ The super-graph is usually a **complete** graph
  - ❑ The edges can only come from the super-graph
- ❑ A graph may have many spanning trees
- ❑ The **cost** of a spanning tree is the sum of all the edge weights
- ❑ A minimum spanning tree (MST) is the spanning tree with the minimum cost

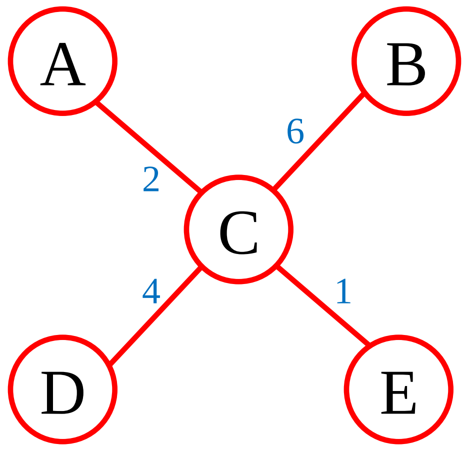
Minimum Spanning Tree Example



🡸Graph
(Should be complete…
…this isn't…!)

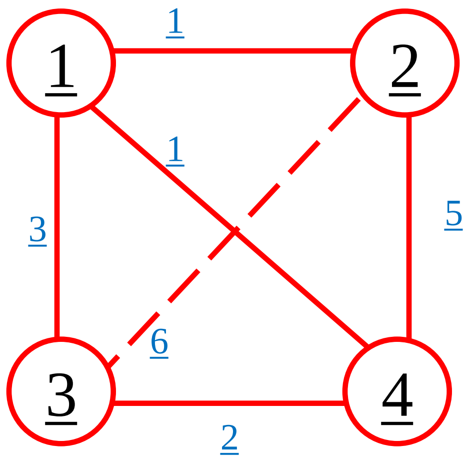


Minimum
Spanning Tree,
Cost = 6

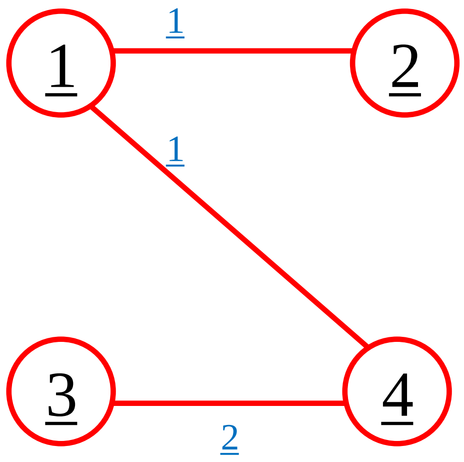


Spanning Tree, Cost = 13

Minimum Spanning Tree Example



Graph of cities

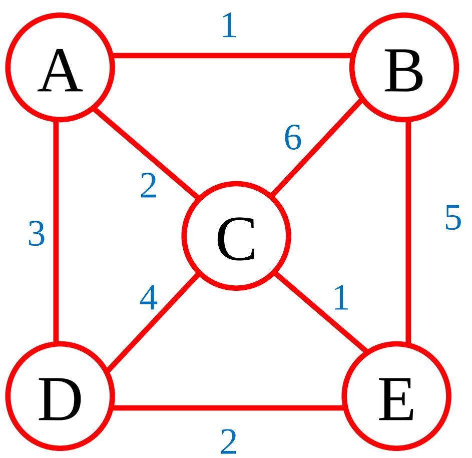


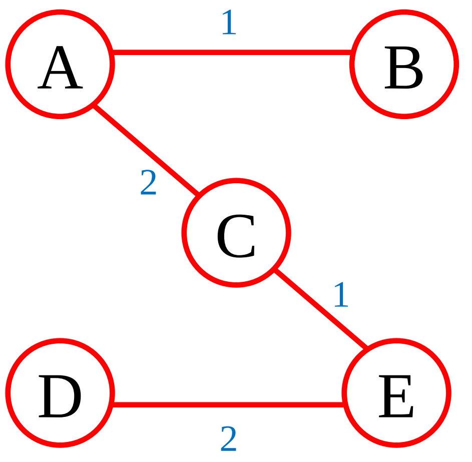
Minimum Spanning Tree, Cost = 4

Prim's Algorithm for MST

```
Algorithm 4. PrimsMST(G=(V,E))
Input: G- a weighted graph (V- vertices, E- Edges)
1) Let x be a random node from V
2) Let Vnew = {x}
3) Let Enew = {}
4) While Vnew ≠ V
5)     Choose an edge (u,v) with minimal weight,
       such that u ∈ Vnew and v ∉ Vnew
6)     Vnew = Vnew ∪ {v}
7)     Enew = Enew ∪ {(u,v)}
8) Wend
Output: Gnew = (Vnew,Enew)
```

Prim’s Algorithm Example





Randomly choose A
Vnew={A}
Enew={}

Select (A,B) (1)
Vnew={A,B}
Enew={(A,B)}

Select (A,C) (2)
Vnew={A,B,C}
Enew={{A,B),(A,C)}

Select (C,E) (1)
Vnew={A,B,C,E}
Enew={{A,B),(A,C),(C,E)}

Select (E,D) (2)
Vnew={A,B,C,E,D}
Enew={{A,B),(A,C),(C,E),(E,D)}

Applications of MST

- Network Design
  - E.g. computer networks, telecommunications networks, transportation networks, water supply networks, etc…
- CfA redshift survey
  - MSTs used for understanding the large-scale structure of the universe
- Approximation algorithms for NP-hard problems
  - E.g. Travelling salesperson problem
- Cancer imaging
  - The British Columbia Cancer Research centre uses them to describe the arrangements of nuclei in skin cells

This Weeks Laboratory

- ❑ This laboratory is also one of the worksheets you may be assessed on for **Task #1 and Task #2**
- ❑ You will be experimenting with Minimum Spanning Trees (MST)…
- ❑ **CodeRunner** class test CRII…

Next Lecture

- We will be looking at Floyd–Warshall and Dijkstra's algorithm