COMP20007 Design of Algorithms

Week 5 Workshop

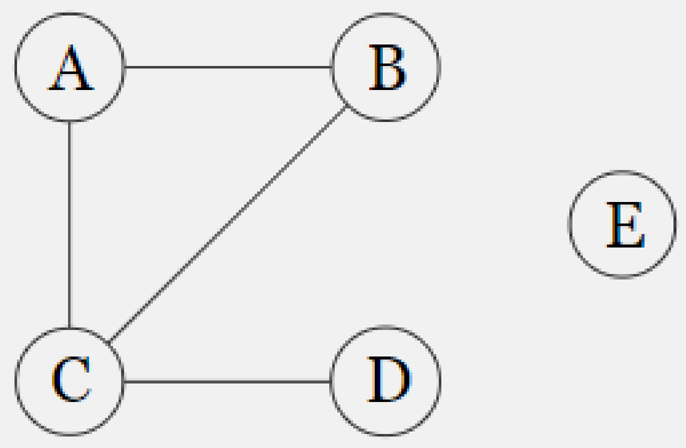
Tutorial

We'll begin the tutorial by going over any graph representation questions from last week which we didn't get to.

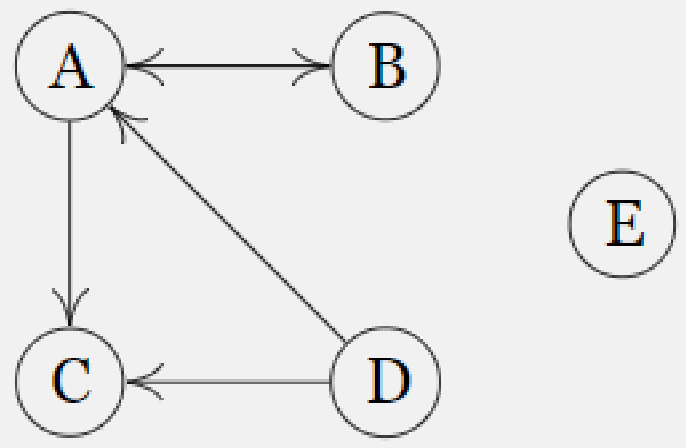
1. Graph representations

Consider the following graphs.

a. An undirected graph:



b. A directed graph:



Give their representations as:

i. adjacency lists

ii. adjacency matrices

iii. sets of vertices and edges

Which node from (a) has the highest degree? Which node from (b) has the highest in-degree?

2. Graph representations continued

Different graph representations are favourable for different applications. What is the time complexity of the following operations if we use (i) adjacency lists (ii) adjacency matrices or (iii) sets of vertices and edges?

a. determining whether the graph is *complete*

b. determining whether the graph has an *isolated node*

Assume that the graphs are undirected. A *complete* graph is one in which there is an edge between every pair of nodes. An *isolated node* is a node which is not adjacent to any other node.

Assume that the graph has n vertices and m edges.

3. Sparse and dense graphs (Homework)

We consider a graph to be *sparse* if the number of edges, m, is order $\Theta(n)$. On the other hand we say a graph is *dense* if $m \in \Theta(n^2)$.

Give examples of types of graphs which are sparse and types which are dense.

What is the space complexity (in terms of n) of a sparse graph if we store it using:

i. adjacency lists

ii. adjacency matrix

iii. sets of vertices and edges

4. Depth First Search and Breadth First Search

List the order of the nodes visited in the following graph when the following search algorithms are run:

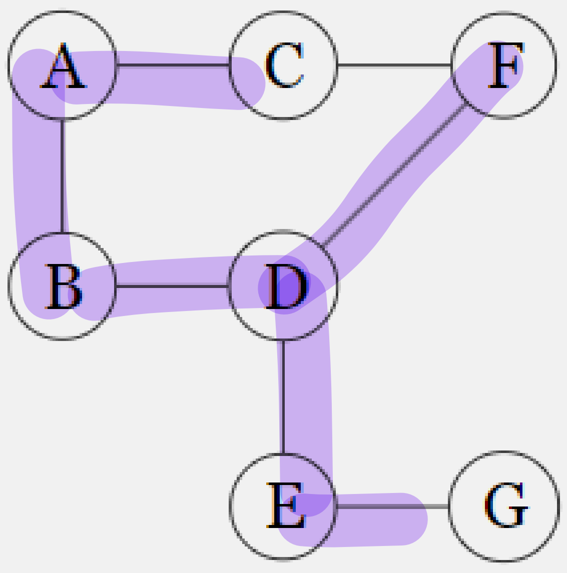
i. depth first search

ii. breadth first search

Start at *A* and break ties by choosing nodes in alphabetic order.

① DFS:

A B D E G F C

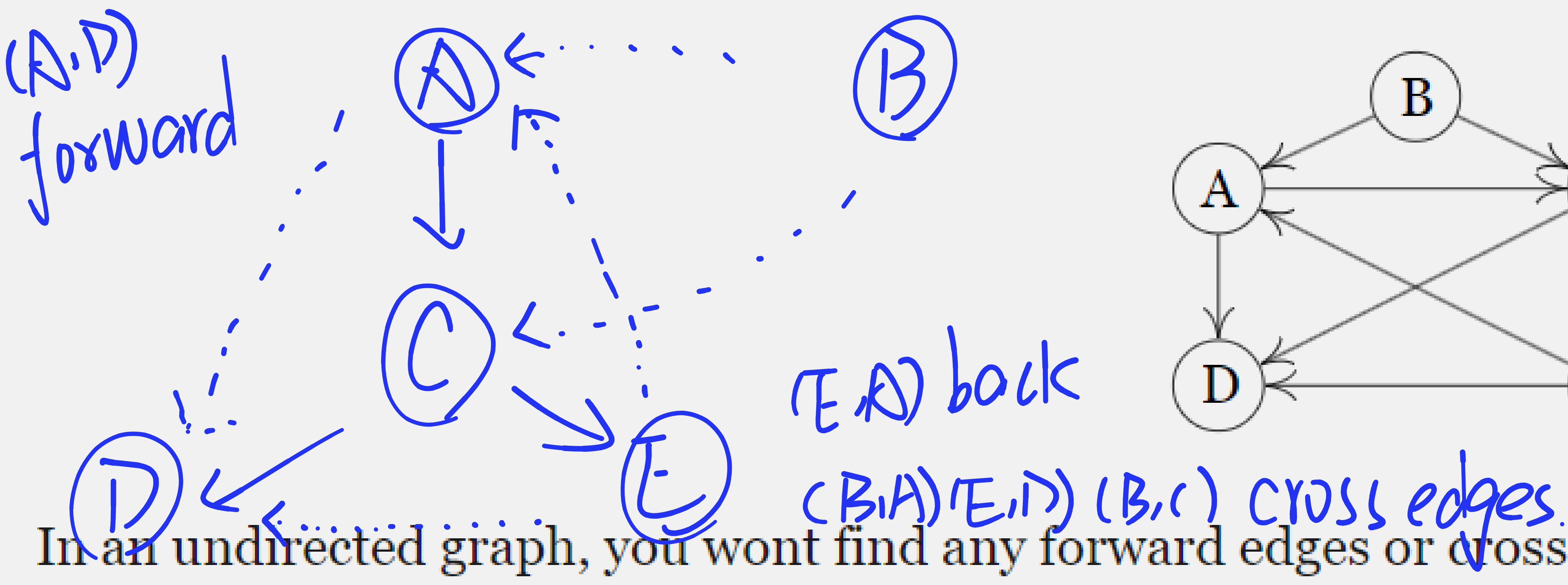


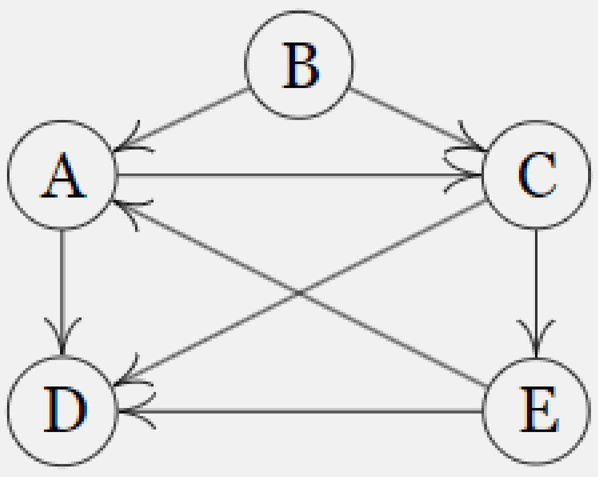
② BFS:

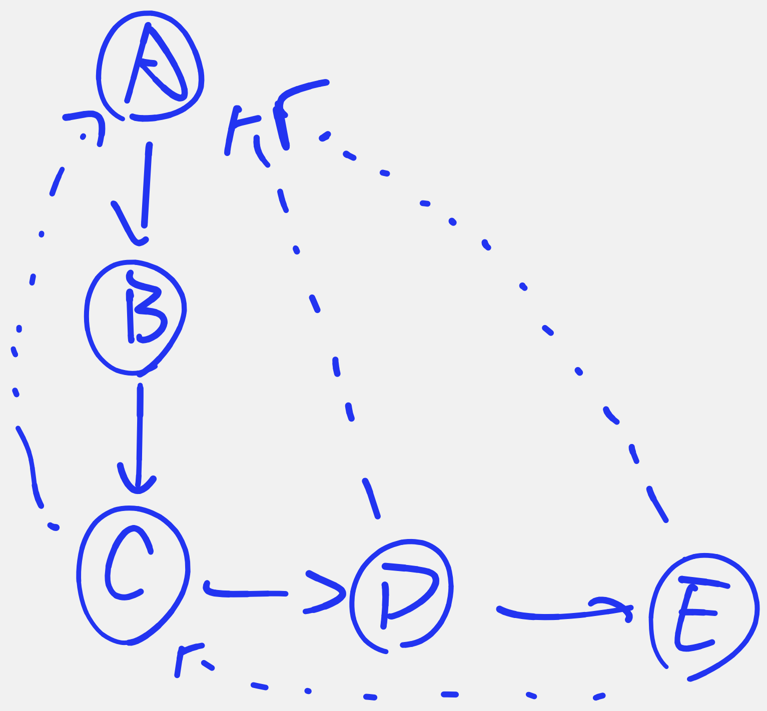
A B C D F E G

5. Tree, Back, Forward and Cross Edges

A depth-first search of a directed graph can be represented as a tree (or a collection of trees, *i.e.*, a forest). Each edge of the graph can then be classified as a *tree edge*, a *back edge*, a *forward edge*, or a *cross edge*. A tree edge is an edge to a previously un-visited node, a back edge is an edge from a node to an ancestor, a forward edge is an edge to a non-child descendent and a cross edge is an edge to a node in a different sub-tree (*i.e.*, neither a descendent nor an ancestor). Draw a depth-first search tree based on the following graph, and classify its edges into these categories. Begin the depth-first search at A.







In an undirected graph, you wont find any forward edges or cross edges. Why is this true? You might like to consider the graph above, with each of its edges replaced by undirected edges.

6. Finding Cycles

Explain how one can also use breadth-first search to see whether an undirected graph is cyclic. Which of the two traversals, depth-first and breadth-first, will be able to find cycles faster? (If there is no clear winner, give an example where one is better (DFS) and another example where the other is better.) (BFS)

find the back edge → find one cycle. ← find a cross edges

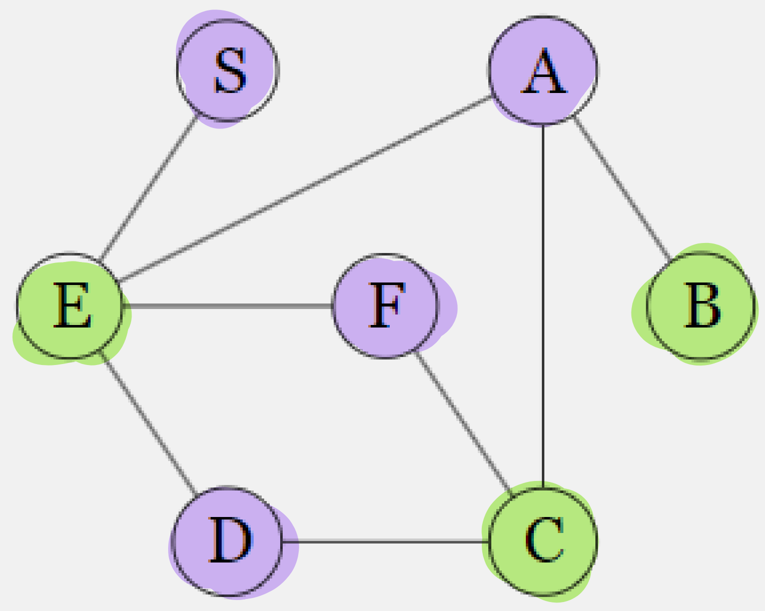
7. 2-Colourability

Design an algorithm to check whether an undirected graph is 2-colourable, that is, whether its nodes can be coloured with just 2 colours in such a way that no edge connects two nodes of the same colour.

To get a feel for the problem, try to 2-colour the following graph.

DFS / BFS

f(x) and x have opposite colors.



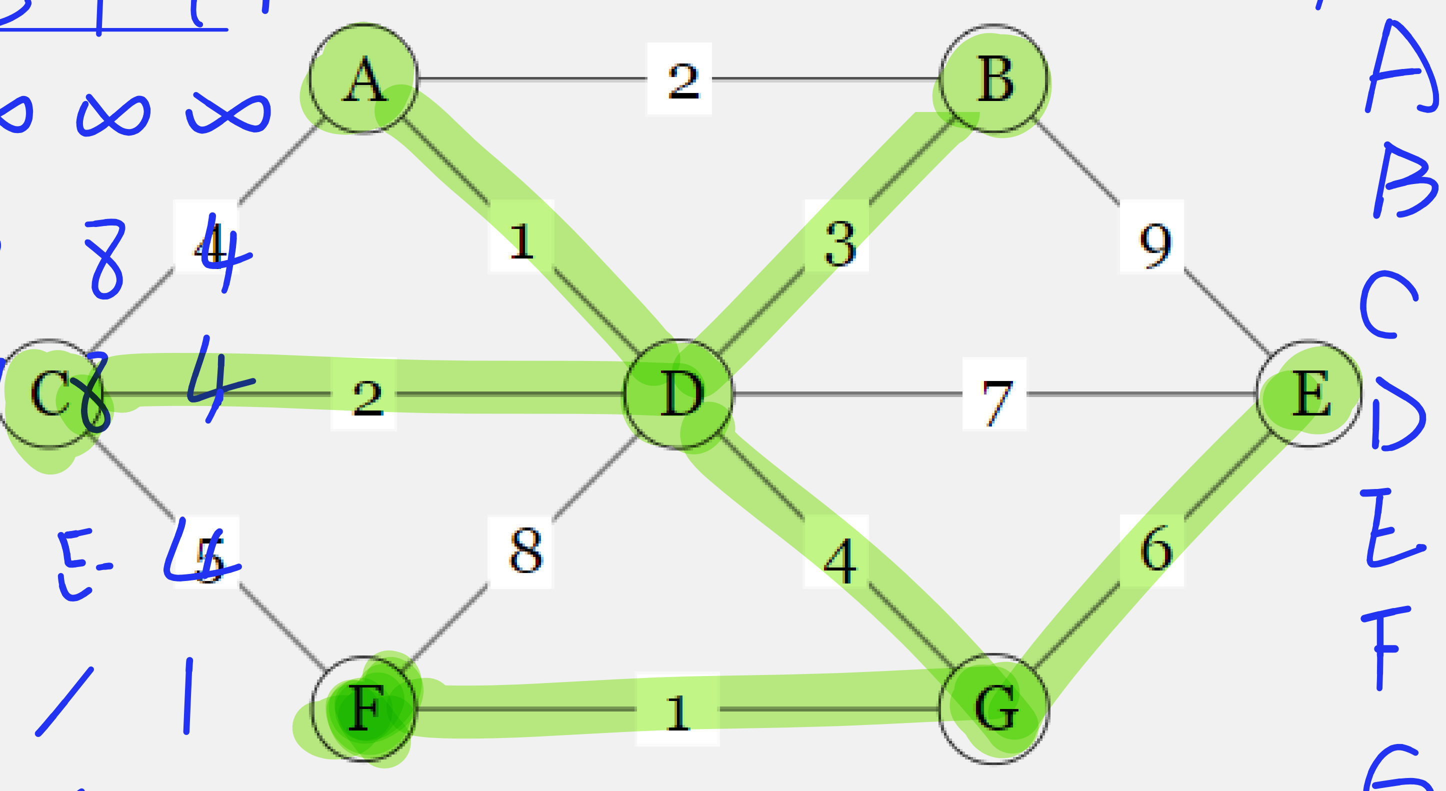
Do you expect we could extend such an algorithm to check if a graph is 3-Colourable, or in general: *k*-Colourable?

8. Minimum Spanning Tree with Prim's Algorithm

Prim's algorithm finds a minimum spanning tree for a weighted graph. Discuss what is meant by the terms 'tree', 'spanning tree', and 'minimum spanning tree'.

Run Prim's algorithm on the following graph, using A as the starting node. What is the resulting minimum spanning tree for this graph? What is the cost of this minimum spanning tree?

set	A	B	C	D	E	F	G
A	/	2	4	1	∞	∞	∞
D	/	2	2	/	7	8	4
B	/	/	2	/	7	8	4
C	/	/	/	/	7	5	4
F	/	/	/	/	7	/	1
G	/	/	/	/	6	/	/
E	/	/	/	/	/	/	/



Node				
A	∞	∞	∞	8
B	∞	9	9	9
C	∞	∞	∞	9
D	∞	7	7	
E	0			
F	∞	∞	7	7
G	∞	6		

9. Single Source Shortest Path with Dijkstra's Algorithm

Dijkstra's algorithm computes the shortest path to each node in a graph from a single starting node (the 'source'). Trace Dijkstra's algorithm on the graph from Question 8, with node E as the source. Repeat the algorithm with node A as the source. How long is the shortest path from E to A? How about A to F?

E to A: E→D→A $\boxed{8}$ A to F: A→D→G→F $\boxed{6}$