School of Computing and Information Systems
COMP90038 Algorithms and Complexity Tutorial Week 5

20–24 August 2018

Plan

Questions 23–26 are related to brute-force problem solving. There may not be time to cover more than two. The rest of the questions mostly relate to graphs and search.

The exercises

23. Consider the *subset-sum problem*: Given a set S of positive integers, and a positive integer t, find a subset $S' \subseteq S$ such that $\sum S' = t$, or determine that there is no such subset. Design an exhaustive-search algorithm to solve this problem. Assuming that addition is a constant-time operation, what is the complexity of your algorithm?

24. Consider the *partition problem*: Given n positive integers, partition them into two disjoint subsets so that the sum of one subset is the same as the sum of the other, or determine that no such partition exists. Designing an exhaustive search algorithm to solve this problem seems somewhat harder than doing the same for the subset-sum problem. Show, however, that there is a simple w s, try to *reduce* th

25. Consider the *cli* k, determine whether the graph contains a *clique* of size k, that is, G has a c k nodes. Design an exhaustive-search algorithm to solve this problem.

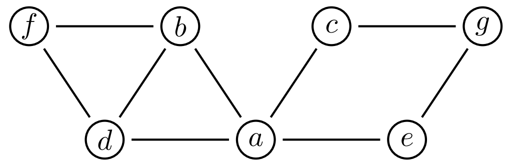
26. Consider the special clique problem of finding triangles in a gra a clique of size 3). Show that this problem can be solved in time $O(|V|^3|E|)$.

27. Draw the undirected graph whose adjacency matrix is

	a	b	c	d	e
a	0	1	1	0	0
b	1	0	1	0	0
c	1	1	0	1	1
d	0	0	1	0	1
e	0	0	1	1	0

Starting at node a, traverse the graph by depth-first search, resolving ties by taking nodes in alphabetical order.

28. Consider this graph:



(a) Write down the adjacency matrix representation for this graph, as well as the adjacency list representation (assume nodes are kept in alphabetical order in the lists).

(b) Starting at node a, traverse the graph by depth-first search, resolving ties by taking nodes in alphabetical order. Along the way, construct the depth-first search tree. Give the order in which nodes are pushed onto to traversal stack, and the order in which they are popped off.

(c) Traverse the graph by breadth-first search instead. Along the way, construct the depth-first search tree.

29. Explain how one can use depth-first search to identify the connected components of an undirected graph. Hint: Number the components from 1 and mark each node with its component number.

30. The function CYCLIC is intended to check whether a given undirected graph is cyclic.

function CYCLIC($\langle V, E \rangle$)
  mark each n [illegible]
  $count \leftarrow 0$
  for each v [illegible]
    if v is m[illegible]
      $cyclic \leftarrow$ HASCYCLES(v)
      if $cyclic$ **then**
        return *True*
  return *False*

function HASCYCLES(v)
  $count \leftarrow count + 1$
  mark v with $count$
  for each edge (v, w) **do** ▷ w is v's neighbour
    if w's mark is greater than 0 **then**
      return *True* ▷ w has been visited before
    if HASCYCLES(w) **then**
      return *True* ▷ a cycle can be reached from w
  return *False*

Show, through a worked example, that the algorithm is incorrect. A later exercise will ask you to develop a correct algorithm for this problem.

31. (Optional—state space search.) Given an 8-pint jug full of water, and two empty jugs of 5- and 3-pint capacity, get exactly 4 pints of water in one of the jugs by completely filling up and/or emptying jugs into others. Solve this problem using breadth-first search.

32. (Optional—use of induction.) If we have a finite collection of (infinite) straight lines in the plane, those lines will split the plane into a number of (finite and/or infinite) regions. Two regions are *neighbours* iff they have an edge in common. Show that, for any number of lines, and no matter how they are placed, it is possible to colour all regions, using only two colours, so that no neighbours have the same colour.