Two hours

UNIVERSITY OF MANCHESTER
SCHOOL OF COMPUTER SCIENCE

Algorithms and Imperative Programming

Date:	Friday 3rd June 2016
Time:	14:00 - 16:00

Please answer THREE Questions from the FOUR Questions provided

Use a SEPARATE answerbook for EACH Question

This is a CLOSED book examination

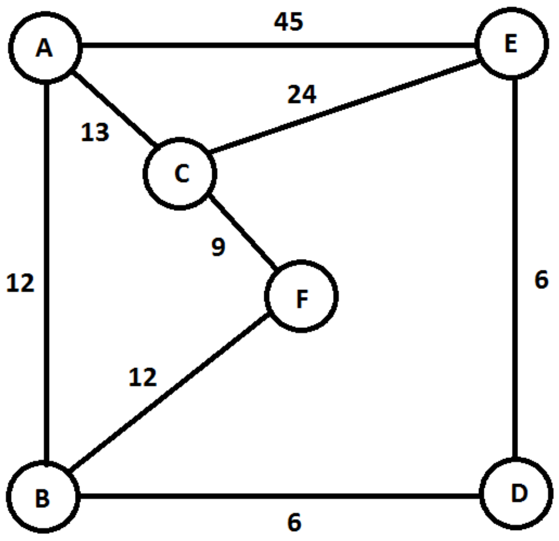
The use of electronic calculators is permitted provided they are not programmable and do not store text

[PTO]

1. Graph algorithms.

   a) Write a pseudocode description of Dijkstra's algorithm for finding the shortest path between a given node and all other nodes in a connected, undirected, weighted graph G. Assume in your answer that you are given the functions `remove_smallest()` and `insert(int k)` for handling the priority queue in Dijkstra's algorithm.
   (5 marks)

   b) Consider the following weighted undirected graph:



   A traveller needs to go from town A to town E, but also needs to visit town D (before, or after visiting E). Explain how this problem can be solved by a single application of Dijkstra's algorithm and a simple heuristic argument. (Hint: Show that, due to the structure of the graph G, the shortest path between towns D and E is a direct path of the length 6.) Show the progression of Dijkstra's algorithm applied to solve this problem, step by step, and draw the contents of the priority queue (implemented as a heap) at each step. (10 marks)

c) Suppose that a graph represents a street network in a small town, with the nodes being where streets join and the weights being street lengths. A tourist wants to undertake a sightseeing tour of the town *by passing along all of the streets*.

How can the tourist ensure that the tour has minimum overall distance? Describe how Dijkstra's algorithm can be used in this context. Hint: Firstly, establish a connection between the number of edges incident at nodes and the requirement to walk along certain streets more than once, i.e. what is the condition on the number of streets at a junction that allows us to pass along all streets at the junction without revisiting streets. Then group the nodes adjacent to the edges that need to be traversed more than once into pairs and deduce how to minimise the path distances between such pairs. (5 marks)

2. Knapsack problems.

   a) Let C be a positive integer. What is the size of (i.e. number of bits in) the standard binary representation of C? (2 marks)

   b) Let $a_1, \dots, a_n$ be a list of items with (positive integral) 'weights' $w(a_1), \dots, w(a_n)$ and (positive integral) 'values' $v(a_1), \dots, v(a_n)$. For all i ($1 \leq i \leq n$) and all non-negative integers C, denote by $V(i,C)$ the maximum value of

   $$\sum\{v(a_j) \mid j \in J\} \qquad (1)$$

   where J ranges over all subsets of $\{1, \dots, i\}$ such that $\sum\{w(a_j) \mid j \in J\} \leq C$. Write an expression for $V(1,C)$ in terms of the number C, the 'weight' $w(a_1)$ and 'value' $v(a_1)$. (2 marks)

   c) Write a recursive expression for $V(i+1,C)$ in terms of $V(i,D)$ for suitable values of $D \leq C$, together with the 'weight' $w(a_{i+1})$ and 'value' $v(a_{i+1})$. (4 marks)

   d) From your answers to Questions 2b and 2c, write a *recursive* algorithm (in pseudocode) that, for a given collection of 'weights' $w(a_1), \dots, w(a_n)$ and 'values' $v(a_1), \dots, v(a_n)$, and a given non-negative integer C, computes $V(n,C)$. Your algorithm must use only a polynomial amount of memory (as a function of the sizes of the inputs). You need not prove that your algorithm uses only a polynomial amount of memory. (4 marks)

   e) Write an *iterative* algorithm (in pseudocode) that, for a given collection of 'weights' $w(a_1), \dots, w(a_n)$ and 'values' $v(a_1), \dots, v(a_n)$, and a given non-negative integer C, computes $V(n,C)$. Your algorithm must run in time bounded by a polynomial function of the quantity $C + n + \sum_{i=1}^{n}(\log w(a_i) + \log v(a_i))$. Briefly justify the claim that your algorithm has the prescribed running time. (6 marks)

   f) Why is it not correct to say that algorithm given in Question 2e runs in polynomial time? (2 marks)

3. Balanced trees.

   Consider the following sequence of keys:

   `25,38,11,26,31,46,57`

   We wish to insert these keys in the given order (from left to right) into an AVL tree.

   a) Show the process of insertion, step by step, and the trees created at each step. (10 marks)

   b) Write an algorithm (in pseudocode) that implements the following method for an AVL tree: `findAllInRange(T,k1,k2)`. The method should return an iterator of size s pointing to the elements of a tree `T` with the keys k that satisfy $\texttt{k1} \leq k \leq \texttt{k2}$. The elements in the iterator should have their keys sorted in ascending order. In designing your algorithm, you may wish to preprocess the tree before searching. Your algorithm should have asymptotic complexity $O(\log(n)+s)$, where n is the number of nodes in the AVL tree. (10 marks)

4. Tree searching.

   a) Explain clearly what is meant by the following search strategies for *rooted finite trees*. You need not give code for the search strategies, but should explain how the order of visiting nodes is determined.

      i. Depth-first search (DFS), and

      ii. Breadth-first search (BFS).

   (4 marks)

   b) Now consider trees whose nodes are assigned numerical priorities. Explain clearly what is meant by a *priority search* over such a tree. Illustrate your answer with an example tree and a traversal by a priority search. (3 marks)

   c) A *priority queue* is a data structure storing items, each item having a numerical priority. It has the same operations as a queue (*push*, *pop*, *top*, *empty*), except that the *top* operation returns the item with highest priority, and the *pop* operation removes this item.

   Give a pseudocode description of an algorithm for the priority search of rooted finite trees with nodes labelled with priorities, using the operations of a priority queue.

   (5 marks)

   d) Explain clearly what is meant by *heuristic search* in algorithm design.

   (3 marks)

   e) Consider the following problem. We are given a positive integer *N* and a set of positive integers *S*. The task is to select a subset of numbers from *S* whose sum is exactly *N*. If no such selection can be made, then we return with failure.

   Explain how selecting numbers from *S* one-by-one to try to satisfy the required property may be formulated as a *tree searching* problem, and how a *heuristic search* may be based on the principle of choosing the next largest appropriate number available. In particular, describe how to construct the search tree and which of the search strategies (DFS, BFS or priority search) your heuristic search employs.

   (5 marks)