COMP 26120

Two hours

UNIVERSITY OF MANCHESTER
SCHOOL OF COMPUTER SCIENCE

Algorithms and Imperative Programming

Date: Thursday 1st June 2017

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

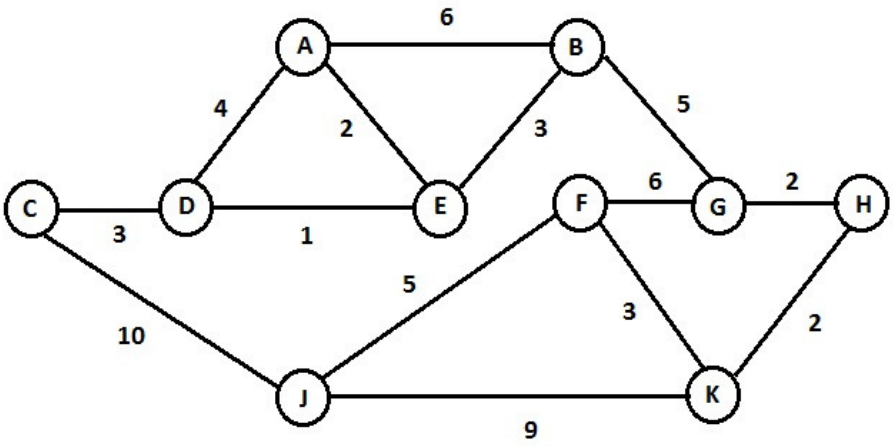
The use of electronic calculators is permitted provided they are not programmable and do not store text

[PTO]

1. Path-finding in graphs

   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. Assume in your answer that the priority queue that is used in Dijkstra's algorithm, and the functions that operate on it, are given. (4 marks)

   b) A road network with distances between towns is represented by the following undirected weighted graph:



   The driver wants to go from the town *A* to the town *J* and the navigation system calculates the shortest route to be *A–E–D–C–J* of length 16. En route, exactly half way between the towns *D* and *C*, the navigation system receives a message that the road *C–J* is closed because of an accident, and recalculates the new route. Give the progression of Dijkstra's algorithm, step by step, to determine the new shortest path. In your answer, assume that the driver needs first to go to the town *C* to turn around. The priority queue does not need to be shown explicitly in your answer, but the intermediate weights for all nodes should be presented.

   (10 marks)

   c) Give the time complexity of Dijkstra's algorithm when applied to the problem of finding the shortest path between a given node and all other nodes in an undirected, connected, weighted graph with *N* nodes and *E* edges. Contrast two different implementations of the priority queue: as a list and as a heap. Define sparse and dense graphs by expressing the asymptotic relationship between *N* and *E* in such graphs. Based on this, compare the asymptotic complexity of:

   - Dijkstra's algorithm for finding the shortest path between *all pairs* of nodes (assuming that the priority queue is implemented as a heap), and
   - the Floyd-Walshall algorithm (also known as Floyd's algorithm).

   (6 marks)

2. Applying Knapsack techniques

The UK government has recently set up a *Commission on Academic Quality* (CAQ) to rank academics using an official database of scientific publications. Each publication in the database has: (i) a list of one or more authors, (ii) a number of pages (a positive integer) and (iii) a number of times this publication has been cited elsewhere (a non-negative integer - the *citation number*).

The original plans provided for each UK academic to submit to the CAQ a list of publications of which they are an author, up to a limit of ℓ pages in total. An academic's CAQ score is then just the sum of the citation numbers of the submitted publications. (It does not matter if those publications have other authors.) The strange decision was made to allow submission of just *parts* of publications (specified by a whole number of pages), with citations being counted proportionately. Thus, Dr. Smith might (for some reason) decide to submit 31 pages' worth of an 80 page publication with 103 citations, which will then score $31 \cdot 103/80 = 39.9125$ points. Dr. Smith can submit any number of part- (or whole-) publications, up to the total page limit of ℓ.

a) Using pseudocode, give an algorithm for computing, for any academic with n publications, a list of (whole- and part-) publications which will maximize their CAQ score. State precisely the complexity of your algorithm, as a function of ℓ and n. You may assume that all the data for the academic's publications has already been extracted from the database as a list of records (but this may be in any order).

(5 marks)

b) In a revised version of the CAQ rules, it was decided that only *whole* publications could be submitted (no part-publications), but still with a strict limit of ℓ pages in total. Explain clearly how you would compute, in time polynomially bounded by n and ℓ, a list of publications an academic should submit under this new scheme to maximise their CAQ score. A full program, or pseudocode, is not required.

(5 marks)

c) Following user-feedback, a further change was made to the CAQ rules. It was decided that the limit should not be ℓ *pages*, but $L = 2000\ell$ *characters*. Indeed, the official database had an additional, hidden field listing, for each publication, the number of characters it contains, and this field was duly made visible.

What will be the effect of using the same approach as the one you used in part (b), but with a character limit of L instead of a page-limit of ℓ? Give an alternative technique which will work well if, in practice, most academics hit the character limit with a small number of publications. (5 marks)

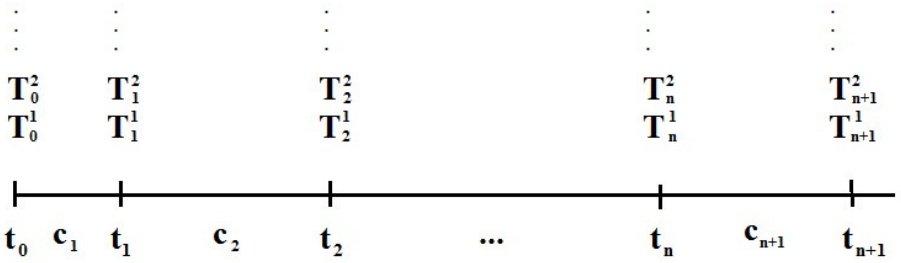
d) Some unscrupulous university departments found a ruse to maximize their scores. They decided that, on new publications, they would simply list the entire department as authors; that way, every academic in the department would get the score of the top-performing colleague. When government officials heard of this, they changed the rules so that no department could submit a publication twice. Thus, if Drs. Smith and Jones in the same department author a publication together, it could be submitted *either* by Dr. Smith or by Dr. Jones, but not both. However, the character limit was dropped and replaced with a limit on the number k of (whole) publications that any academic could submit.

Explain clearly how you would compute, for a given department, a list of publications every academic in that department should submit to the CAQ so as to maximize the department's total CAQ score. The submissions must conform to the new rules. Give the complexity of your algorithm as a function of: the number m of academics in the department, the total number N of publications submitted by academics in the department, and the maximum number k of publications submitted by each academic. What techniques can you use to avoid needless search?

(5 marks)

3. Trees and hashing

Consider the following simplified scenario for task scheduling in a computer system with a single processor (CPU). We assume that all the CPU time is assigned to the execution of users' tasks (i.e. the time taken to perform task scheduling is neglected). The CPU divides time into "slots" of varying duration $c_{i+1} = t_{i+1} - t_i$, $i = 0, 1, 2, \ldots$ (see the picture below). At the start time t_i of each slot c_{i+1}, a number of tasks T_i^j, $j = 0, 1, \ldots$ with execution times τ_i^j arrive ready for execution. In addition, there may be tasks waiting from previous times but which have not yet been executed. During a single CPU slot *only one task may be executed* and its execution time must not exceed the time allocated for the current slot. The execution times of some tasks may be longer than individual CPU slots, and such tasks cannot be executed across multiple CPU slots. Tasks which are not executed at a particular time wait for succeeding time slots. *Task scheduling* aims to provide the best utilisation of the CPU.



a) Explain how priority queues can be used to implement task scheduling. In your answer, give a suitable choice of a key (that represents the priority of a task in the scheduling) associated with each task in the queue, which will maximise the use of each CPU slot. In addition, explain the operations required to update the priority queue at the start of each slot. Hint: the old and the newly arrived tasks at t_i need to be combined together in a new priority queue and the new keys associated with all the tasks need to be created; these keys should guarantee that the task sent to the execution represents the best fit among all tasks based on the current slot size c_{i+1} and the chosen task's execution time. Your answer can be either a clear explanation or a pseudocode program. (10 marks)

b) Consider the following set of tasks and set of CPU slots:

Process (T_i^j)	T_0^1	T_0^2	T_0^3	T_3^1	T_3^2	T_5^1	T_5^2
Arrival time	0	0	0	16	16	33	33
Execution time (τ_i^j)	4	10	2	3	7	12	5

Slot start time (t_i)	0	6	10	16	20	33	46	52
Slot length (c_{i+1})	6	4	6	4	13	13	6	10

Apply the scheduling algorithm from part (a) to perform scheduling of these tasks on the given CPU time resources. In your answer, consider that a priority queue is realised as a heap. For each discrete time t_i, $i = 0, \ldots, 7$, give the content of the old priority queue, and the values of the new keys for each old and (possibly) newly arrived tasks. Based on the newly computed keys, draw the final priority queue (the intermediate steps taken during the creation of the heap are not required), and indicate which task (if any) is sent to the execution. Calculate the CPU usage, as a percentage, achieved by scheduling these tasks. (10 marks)

4. Graphs and graph algorithms

a) Explain the following traversal techniques for *finite directed graphs*. Your traversal should visit all the nodes of the graph, restarting the traversal where necessary. You need not give programs, but you should explain clearly the rules which determine the order that nodes are visited.

i. Depth-first Search (DFS),
ii. Breadth-first Search (BFS), and
iii. Priority Search, for graphs where each node has a non-negative integer priority assigned to it.

(6 marks)

b) Give an explicit algorithm for performing DFS of finite directed graphs which visits all the nodes and numbers them in the order that they are first encountered. You may either present a program or express the algorithm in pseudocode.
(5 marks)

c) Using Big-Oh notation, express the worst-case time complexity of DFS in terms of the number of nodes N, and the number of edges E, of a graph, using the following two representations of graphs:

i. Adjacency lists, and
ii. Adjacency matrices.

In each case, explain why your answer is correct. (3 marks)

d) A *topological sorting* of the nodes of a finite directed graph is a list of the nodes, each occurring exactly once in the list, such that, if there is an edge from node s to node t, then s is before t in the list.

Explain clearly why a directed graph with a topological sorting must be acyclic.
(2 marks)

Given an acyclic finite directed graph, explain how, using a DFS or otherwise, a topological sorting may be constructed. (4 marks)

END OF EXAMINATION