COMP26120

Two hours

UNIVERSITY OF MANCHESTER
SCHOOL OF COMPUTER SCIENCE

Algorithms and Imperative Programming

Date: Friday 1st June 2012

Time: 14:00 - 16:00

Please answer THREE Questions from the FOUR questions provided
Use a SEPARATE answerbook for EACH Section

For full marks your answers should be concise as well as accurate.
Marks will be awarded for reasoning and method as well as being correct.

This is a CLOSED book examination

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

[PTO]

Section A

1. a) Give a definition of the 0/1 Knapsack Problem, describing the variables that specify an instance, the goal (or objective) and the constraints that must be observed.

(5 marks)

b) The Greedy algorithm for the 0/1 Knapsack problem is not guaranteed to find an optimal solution. Assuming that items are processed in descending order of value:weight ratio, (i) construct a 0/1 Knapsack problem instance where the Greedy algorithm *does* find an optimal solution, and (ii) construct an instance where it *does not* find an optimal solution. **In each case your instance must contain at least four items. You must give the items' values and weights, the Knapsack capacity, the optimal solution value, and the greedy solution value**.

(4 marks)

c) Explain the dynamic programming method for solving 0/1 Knapsack Problems. Your answer should include a table of values, clearly labelled, and should explain in specific terms how the values in the table are calculated, including the optimal solution's value. (You do not need to explain how to compute which items are packed in the optimal solution, only its value). (6 marks)

d) Complete the solution to the following 0/1 Knapsack problem using Branch and Bound. Use the fractional knapsack value to calculate upper bounds, and keep partial solutions in the priority queue in descending order of their upper bounds.

The items in descending order of value:weight ratio are

item	1	2	3	4	5
value	200	75	140	55	78
weight	100	50	100	40	60

and the capacity of the knapsack $C = 240$.

Below, the first three steps are done for you. Using the same format for your answer, show the remaining steps of the algorithm until termination. Also *indicate why it has terminated*. If any branch is infeasible, indicate this, and do not add the infeasible partial solution to the priority queue.

Step 1
Current best complete solution: 0
Priority Queue: empty
Upper bound left branch (1****): $200 + 75 + 0.9 \times 140 = 401$
Upper bound right branch (0****): $75 + 140 + 55 + 5/6 \times 78 = 335$.

Step 2
Current best complete solution: 0
Priority Queue: 1**** (401); 0**** (335)
Upper bound left branch (11***): $200 + 75 + 0.9 \times 140 = 401$
Upper bound right branch (10***): $200 + 140 + 55 = 395$

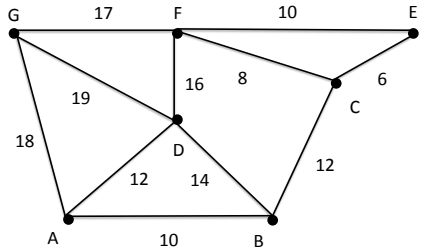
Step 3
Current best complete solution: 0
Priority Queue: 11*** (401); 10*** (395); 0**** (335)
Upper bound left branch (111**): $200 + 75 + 140 = 415$ INFEASIBLE
Upper bound right branch (110**): $200 + 75 + 55 + 50/60 \times 78 = 340$

Step 4...

(5 marks)

2. a) Give a pseudocode description of Dijkstra's algorithm for finding the shortest path between a given node and all other nodes in a graph G. Explain how Dijkstra's algorithm can be used to find the shortest path between all pairs of nodes in G. Assume in your answer that the functions `remove_smallest()`, `insert(int k)`, and `change(int k)` for handling the priority queue that is required by Dijkstra's algorithm are defined. (5 marks)

   b) Consider the following undirected weighted graph representing a road network with distances between the cities:



   A passenger needs to go from the town A to the town E, but also needs to visit the town F (before, or after visiting E). Explain how this problem can be solved by a single application of Dijkstra's algorithm and some simple heuristic argument. (Hint: Due to the structure of the given graph, the shortest path between F and E is a direct path of length 10). Show the progression of Dijkstra's algorithm applied to the solution of the above problem, step by step, and draw the content of the priority queue at each step. (7 marks)

   c) Give the algorithmic complexity of Dijkstra's algorithm when applied to the problem of finding the shortest path between *all* pairs of nodes in an undirected weighted graph with n nodes and e edges. (3 marks)

   d) Suggest a simple modification that would allow the application of Dijkstra's algorithm to the problem of finding the longest paths between each pair of nodes in an acyclic undirected graph. (Hint: If $a \leq b$, then $-a \geq -b$). Explain why this approach works only for acyclic graphs. (5 marks)

Section B

3. Consider the following set of words:

`dog,god,job,bee,fog,ivy,ace,cat,hat,egg`

We wish to insert these words in the given order (from left to right) into an ordered dictionary implemented as a binary search tree or an AVL tree.

a) Show the resulting tree from inserting the above words in the order shown into a binary search tree. (3 marks)

b) i) Show the process, step by step, of inserting the above words into an AVL tree. Use the first letter of a word as a key and assume the standard alphabetical order of letters. (5 marks)

ii) Now show the new AVL tree which results from the AVL tree you constructed in part (b)(i) when the word `fog` is deleted. (3 marks)

c) Give a pseudocode description of an algorithm `findElement(k,T)` for finding an element with the key k in the ordered dictionary T implemented either as a binary search tree or an AVL tree. (4 marks)

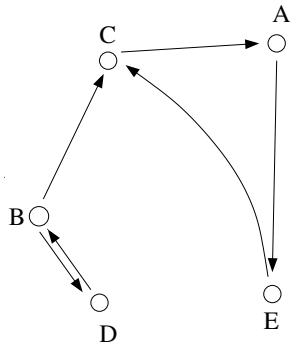
d) Give the worst case asymptotic complexity of the algorithm `findElement(k,T)` for T being both a binary search tree and an AVL tree. Give an example of input data that would lead to the worst case scenario for a binary search tree. (5 marks)

4. Graphs and graph algorithms.

a) Consider the following two representations of finite directed graphs

- Adjacency lists
- Adjacency matrices

i) Explain clearly what these representations are.
Show how the following example of a directed graph is presented using these two representations. (2 marks)



ii) For each of the following operations on finite directed graphs, explain clearly how it may be implemented using adjacency lists and using adjacency matrices. Give the worst-case time complexities of your algorithms, in terms of the number of nodes N and the number of edges E, for each representation and explain how you calculate this. (6 marks)

- Finding a node with maximum in-degree in the graph (the in-degree of a node n is the number of edges in the graph whose target is n).
- Finding whether the graph contains cycles of length 2.

b) Explain what is meant by a Depth-First Search (DFS) of a finite directed graph. (2 marks)

Give an explicit algorithm for performing DFS of finite directed graphs which numbers the nodes in the order that they are first encountered. You may either present a program or express the algorithm in pseudocode. (4 marks)

c) 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