Ollscoil na hÉireann
The National University of Ireland

Coláiste na hOllscoile, Corcaigh
University College, Cork

End-of-Term Examination
Period 1, 2010

CS4407 Analysis of Algorithms

Prof. G. Provan

Attempt all questions

Total marks: 100

60 minutes

Please answer all questions
Points for each question are indicated by [xx]

1. [**15**] Let $f(n)$ and $g(n)$ be asymptotically positive functions. Prove or disprove:
   $f(n)+ g(n) = \Theta(min\ (f(n), g(n))$.

   For the upper bound, we have
   $$f(n)+ g(n) \le max\ \{f(n), g(n)\} + max\ \{f(n), g(n)\}$$
   $$\le 2\ max\ \{f(n), g(n)\}\ \forall n \ge 0 \qquad (1)$$
   For the lower bound, since f and g are positive functions, we have
   $$f(n)+ g(n) \ge f(n) \quad and \quad f(n)+ g(n) \ge g(n)\ \ \forall n \ge 0 \qquad (2)$$
   Combining bounds (1) and (2) we obtain
   $$0 \le max\ \{f(n), g(n)\} < f(n) + g(n) \le 2\ max\ \{f(n), g(n)\}\ \forall n \ge 0$$
   By definition, we have
   $$f(n)+ g(n) = \Theta(max\ (f(n), g(n))$$

   Hence the claim is false.

2. [**30**] A directed graph $G(V,E)$ is *singly-connected* if there is at most one simple path between any two vertices, where a simple path from u to v is a sequence of edges starting at u and ending at v with no loops.
   a. [**15**] Give an algorithm for deciding whether a graph G is *singly-connected.*
   b. [**15**] Prove the complexity of your algorithm.

   SOLUTION:
   (a) Since the graph is directed, we must test each vertex to check if it has a simple path to the other vertices in the graph. To test the presence of simple paths, we can run Depth-first search (DFS) or Breadth-First Search (BFS) to compute the paths. In DFS, a forward edge or a cross edge means the graph is not ***singly-connected***.

   (b) ***Complexity:*** Running Depth-first search (DFS) from a vertex has complexity of $O(|\boldsymbol{(V+E)}|)$. Since we run DFS from each vertex, the total complexity is $O(|\boldsymbol{V\cdot(V+E)}|)$

3. [**30**] Consider the Change Problem in the land of Elbonia. The input to this problem is an integer L. The output should be the minimum-cardinality collection of coins required to make L shillings of change (that is, you want to use as few coins as possible). In Elbonia the coins are worth 1, 5, 10, 20, 25, 50 Shillings. Assume that you have an unlimited number of coins of each type. So for example, to make change for 234 Shillings the greedy algorithms would take four 50 shilling coins, one 25 shilling coin, one 5 shilling coin, and four 1 shilling coins.

   a. [**10**] For a greedy algorithm, what property P must be satisfied to ensure a globally optimal solution?
   b. [**10**] Define property P for this problem.
   c. [**10**] Formally prove or disprove that a greedy algorithm (that takes as many coins as possible from the highest denominations) correctly solves the Change Problem.

   SOLUTION:

a. The greedy algorithm must satisfy the optimal substructure property *P*, which means that any sub-solution of a complete solution must be optimal.
b. For this problem, if the collection C of coins, if we have $c=max\{C\} \leq L-\Sigma jd$, where we have already chosen *j* coins of denomination *d*, then *c* is in a solution.
c.

Given a set of integer coin values $\{w_1, w_2, ..., w_n\}$ where $w_1 = 1$ and $w_j < w_{j+1}$ for $1 \leq j \leq n - 1$, and a positive integer *W*, find a set of non-negative integers $\{x_1, x_2, ..., x_n\}$ which minimize

$$\sum_{j=1}^{n} x_j \quad \text{subject to} \quad \sum_{j=1}^{n} w_j x_j = W.$$

Pseudocode: The greedy strategy consists of always taking the largest denomination that we can at the time. We repeat this until we have the correct amount of change. We assume that we have k types of coin.

```
MAKE-CHANGE(M)
    1 for i = k downto 1
    2   count[i]  ← M/di
    3   M   ← M − count[i] · di
```

Since for each denomination we calculate the number of coins we can take until we would make more than the amount of change asked for, this operation takes time O(*k*).

We disprove the claim by providing a counter-example. The greedy algorithm is not optimal for the problem of making change with the minimum number of coins when the denominations are 1*;* 5*;* 10*;* 20*;* 25*;* and 50. In order to make 40 Shillings, the greedy algorithm would use three coins of 25, 10, and 5 shillings. The optimal solution is to use two 20- shilling coins.

4. **[25]** Given a weighted graph *G=(V,E)* and a minimum spanning tree *T* for *G*, suppose that we add a new vertex *v* and connect it to *k*<|*V*| of the vertices in *G*, to form a new graph *G'*.
   a. [**10**] Define an algorithm to efficiently find the minimum spanning tree of the modified graph *G'*.
   b. [**5**] What is the complexity of this algorithm?
   c. [**10**] Show that your algorithm is correct.

(a) Call the new portion of graph added (v, E*). Call the set of vertices in G that E* is incident on V*. Let the minimum-cost edge of E* have cost μ. Remove all edges in the minimum spanning tree (MST) T with cost > μ, to create a new MST, called T', with edges E'. Create a sorted list of available edges A, in terms of ascending cost, where A contains the edges not in the MST together with E*, i.e., A = (E-E') ∪ E*. We assume that A has |A| elements.

Since T' must be part of a minimum spanning tree (MST) for G, all we need to do is add the least-cost edges in A to create a new MST for G'. We add the edges in A by cost, and return the least-cost edges e' using Prim's algorithm.

(b) The complexity of this algorithm is $O(|A|^2)$ with a simple adjacency matrix data structure, or $\Theta(|A| \log|V|)$ with a binary heap.
(c) This algorithm is correct since

- we know that T' must be included in the new MST by the property of MSTs;
- adding the least cost edges in A results in T' using Prim's algorithm being an MST. There will be no loops induced by the properties of the algorithm.