Candidate number: ______________

Final exam in TDT4120 Algorithms og data structures

Exam date	13 August 2012
Exam time	0900–1300
Grading date	3 Januar
Language	English
Concact during the exam	Magnus Lie Hetland (ph. 91851949)
Aids	No printed/handwritten; specific, simple calculator

! **Please read the entire exam before you start, plan your time, and prepare any questions for when the teacher comes to the exam room.** Make assumptions where necessary. Keep your answers short and concise. Long explanations that do not directly answer the questions are given little or no weight.

You may describe your algorithms in text, pseudocode or program code, as long as it is clear how the algorithm works. Short, abstract explanations can be just as good as extensive pseudocode, as long as they are precise enough. Running times are to be given in asymptotic notation, as precisely as possible.

a) Assume that you have a DAG G with n nodes and m edges. What is the running time for finding a topological sorting of G?

Answer (6%):

Assume that the undirected graph $G = (V, E)$ is connected. Assume that each node v in V is assigned one of its neighboring edges $e(v)$ in E, and that all these are distinct ($e(u) \neq e(v)$ if $u \neq v$).

b) Briefly explain why the nodes in V, together with their assigned edges, are not guaranteed to form a spanning tree for G.

Answer (6%):

The function F is defined by $F(n) = (F(n{-}1) + F(n))/2 + 1$, where $F(0) = 0$.

c) What is $F(42)$?

Answer (6%):

d) What is the running time of HEAPSORT in the worst case?

Answer (6%):

e) What kind of queue is normally used in Dijkstra's algorithm?

Answer (6%):

f) What is the worst-case running time (as a function of n) of inserting n elements into a hash table that already contains $n^{0.5}$ elements?

Answer (6%):

g) How will you find the number of possible paths from one node to another, if the pahts cannot share edges?

Answer (6%):

h) You can reduce from A to B in polynomial time. Then answers to A and B can be verified in polynomial time. Complete the following scenarios (under the assumption P ≠ NP).

Answer (6%): Check the box (□) by the most specific class that applies.

A is in …	**B is in …**
P	□ P / □ NPC / □ NP
NPC	□ P / □ NPC / □ NP
□ P / □ NPC / □ NP	**P**
□ P / □ NPC / □ NP	**NPC**

You are to compute distances between cities in road networks. Each road network is represented as an undirected graph $G = (V, E)$, where each edge $e = \{u, v\}$ in E has a weight $w(e)$ that corresponds to the distance (along the road) between u and v. You know that the number of roads connected to each city is limited (i.e., $O(1)$).

i) You are to solve the all-pairs shortest path problem for such graphs. Which algorithm would you use?

Answer (6%):

j) What is the running time of Prim's algorithm on a connected (undirected) graph with n nodes and m edges if you use an unsorted array as your priority queue, but still use adjacency lists? (You would need to scan the entire queue both to find the next node *and* to update the priority of a node.)

Answer (6%):

k) You have constructed a Huffman-tree over *n* symbols. If you for each node can choose freely which child edge is 0 and which is 1, how many different codes (that is, trees) can you construct?

Answer (7%):

Systems such as GNU Automake are used to pack and compile program code. In Automake you can have *conditional* subdirectories – directories with code that isn't necessarily compiled or used, depending on some (Boolean) condition. When you use Automake for building an archive (a tar- or zip-file) for distribution, all of these directories are included, because you can't know, *a priori*, whether they will be compiled or not, as this depends on conditions on the user's end. You have a colleague who still wants to try to optimize this process, by not packing subdirectories whose condition *can never be true*.

l) What will be the most important challenge in constructing such an algorithm?

Answer (7%):

You're writing a program for designing stairs. The number of steps (n), and their widths and depths, are given. The stairs must fit the environment in various ways, and there is therefore a c ost function $c_i(\cdot)$ attached to each step i, which yields the cost $c_i(h)$ for assigning the height h to the step i (measured from the ground). At the same time, there is a cost function $r(\cdot)$ for the difference in height between step k–1 and step k. If the difference is x then the cost is $r(x)$.

You are to fill in a height table $h[0\ldots n+1]$, where $h[0]$ and $h[n+1]$ are given, and where the total cost is $c_0(h[0]) + r(h[1]–h[0]) + c_1(h[1]) + \ldots + r(h[n+1]–h[n]) + c_{n+1}(h[n+1])$. All heights are measured in full cm. You can assume that $h[0] < h[1] < \ldots < h[n] < h[n+1]$ and that $h[n+1]$ is $O(1)$.

m) Sketch out an algorithm that is as efficient as possible, that constructs stairs with minimal total cost.

Answer (8%):

At the Department of Space Ships and Dinosaurs there are *n* students and *n* projects, and these are to be matched. The policy is to distribute the projects by drawing lots, and then let the students trade. To help the students trade, two lists are published, one of who is assigned to which project, and one with who applied for which project. (Only one student is assigned to each project, and vice versa, but more can have applied for the same.)

You want to help the students, and wish to construct an algorithm that solves the following problem: Given both the assignments and the list of applications, determine whether a student *S* can trade so that he gets the project he applied for. The trade may involve multiple students, but you can only assume that they will trade if they get what they applied for.

n) Describe an algorithm that solves the problem. What is the running time?

Answer (8%):

Let $G = (V, E)$ be a weighted, directed graph where all cycles have a positive weight sum. We say that two paths from i to j are distinct if the sequence of nodes you visit when following the paths are not exactly the same. (Thus, the paths must have some differing edges.) Describe an algorithm that, for each node pair i, j, finds the number of distinct shortest pahts from i to j in G. The algorithm is to compute n^2 answers, one for each of the n^2 node pairs. (The number of pahts from a node i to the same node can be either 0 or 1; that is up to you.)

o) Describe an efficient algorithm that solves the problem.

Answer (10%):