Final exam in TDT4120 Algorithms og data structures

Exam date	18 August 2011
Exam time	0900–1300
Grading date	8 September
Language	English
Contact 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.

Problem 1

a) What is the running time of the MERGE procedure in mergesort? (Assume n elements.)

Answer (6%):

b) Give an exact solution to the recurrence $T(n) = 2T(n/2) + n$ (that is, without the use of asymptotic notation), where $n = 2^k$ for an integer $k > 1$, og $T(2) = 2$.

Answer (6%):

In the following sentences from the textbook, some words have been removed:

> «Even if the input is not drawn from a _____ distribution, _____ may still run in linear time. As long as the input has the property that the sum of the squares of the _____ sizes is linear in the total number of elements, equation ____ tells us that _____ will run in linear time.»

c) Which algorithm is being discussed?

Answer (7%):

d) What is the expected running time for a lookup in a hash table, if you assume that the number of slots in the table is at least proportional to the number of elements?

Answer (6%):

e) For dynamic programming to be correct, the problem must have so-called optimal substructure. For dynamic programming to be *useful* (that is, increase performance) another problem property is also required. Which?

Answer (7%):

f) Which one of the following expressions (1 through 3) is correct?

1. $T_p \leq T_1/P + T_\infty$
2. $T_p = T_1/P + T_\infty$
3. $T_p \geq T_1/P + T_\infty$

Answer (6%):

g) In the textbook, the shortest path problem is formulated as a linear program, based on a shortest-path algorithm from the curriculum—which one?

Answer (6%):

h) What is the sum $1 + 2 + \ldots + n$, expressed in asymptotic notation?

Answer (6%):

Problem 2

You are given two tables, A and P, with lengths of n and m, respectively. Each element $P[i]$ is an index, indicating an element in A (that element is $A[P[i]]$). You can assume that P is sorted. You are to modify A so it still contains the original elements, possibly in a different order (a permutation), so the elements indicated by P are in the first positions of A.

a) Describe an algorithm that solves the problem as efficiently as possible. What is the running time, as a function of m and n?

Answer (6%):

You are receiving a steady stream of integers over a network, and you wish, at each moment, to keep the m smallest values you have received so far. You wish to minimize the asymptotic running time (as a function of m) to process each element.

b) Describe an algorithm/data structure that solves this problem. Give the running time per element received, as a function of m, after at least m elements have been received.

Answer (6%):

Problem 3

Assume given a directed, acyclic graph (DAG) $G = (V, E)$, as well as nodes s and t in V.

a) How would you count the number of possible paths from s to t in G?

Answer (6%):

b) How would you count the (maximum) number of paths that can be followed *simultaneously* from s to t, if these paths cannot share edges?

Answer (6%):

Consider an undirected graph $G = (V, E \cup F)$, where $E \cap F = \varnothing$. (In other words, the edges of G may be partitioned into the non-overlapping sets E and F.)

The edges of E represent conflicts (E for enemies) and F represents friendships (F for friends). You wish to decide whether the nodes in V may be partitioned into two (non-overlapping) sets A and B, so that *none* of the edges in E connect two nodes in the same set, and so that *all* of the edges in F do. That is, for every edge $e = \{u, v\}$ in the graph G:

- If e is in E either u must be in A and v in B, or vice versa.
- If e is in F then u and v must be in the same set, either A or B.

c) Describe an algorithm (as efficient as possible) that either finds such a partition or that decides that this is impossible. Give the running time.

Answer (6%):

Problem 4

The following problem was given at last year's continuation exam:

> "You are inviting friends over for a party. You are considering a set of n candidates, but you know that each one of them will only have a nice time if he or she knows at least k others at the party. (You can assume that if A knows B then B automatically knows A.) Describe an algorithm that finds the greatest possible subset of the n friends, such that everyone in the subset knows at least k of the others, if such a subset exists. Briefly explain why the algorithm is correct and optimal."

You now wish to have a *small* get-together, and therefore wish to solve the same problem, except that you now wish to find a *smallest* possible subset where everyone knows at least k of the others.

a) Show (that is, explain briefly) that it is unrealistic to solve this new version of the problem.

Answer (6%):

The graph $T = (V, E)$ is a tree with root $r \in V$ and a weight function w over the nodes (that is, each node $v \in V$ has a weight $w(v)$). Note that this weight can be negative. Let $S = (U, F)$ be a subgraph of T. The weight sum of S is then the sum of $w(u)$ for all nodes u in U. You wish to find the weight sum of the heaviest connected subgraph of T that contains r.

b) Describe an algorithm that solves the problem and that is as efficient as possible. Give the running time.

Answer (7%):

Problem 5

In some types of biological sequences (like DNA) so-called *palindromic subsequences* are important. A palindrome is a sequence that is identical when it is reversed (for example, «`agnes i senga`») and a palindromic subsequence is a subsequence (of another sequence) that is a palindrome. The sequence «`CTATACGGTACGATA`» includes, for example, the palindromic subsequences «`TAT`» and «`AACGGCAA`», among others.

Note: a *subsequence* is not the same as a *substring*. If X is a subsequence of Y then all elements in X must be found in Y, in the same order, but not necessarily next to each other.

You now wish, given a sequence S with length n, to find the length of the *longest* palindromic subsequence P in S. (There may, of course, be several with the same length.)

a) Describe very briefly (feeling free to reference the curriculum) an algorithm that solves the problem as efficiently as possible. Argue very briefly that your solution is correct. Give the running time.

Answer (7%):