Example solutions for exam problems

Foundations of computing, Algorithms and Data Structures

Rasmus Pagh

December 21, 2010

Abstract

Below are example solutions to some exam problems that would have been given (close to) maximum points. In each answer are sections typeset in italics. These are the most essential parts of each answer, and roughly what would be required for a passing grade on the particular question.

Slash Maze, June 2009 exam

We can consider the slash maze as a graph, where each grey square is a vertex, and adjacent squares have an edge between them unless there is a slash or backslash on the common edge. The vertices may be identified by their coordinates. (Concretely, if we suppose that the height and width of a square is 1, there are squares with center point $(x - 0.5, y)$ for $x = 1, \dots, w$, $y = 0, \dots, h$, and there are squares with center point $(x, y - 0.5)$ for $x = 0, \dots, w$, $y = 1, \dots, h$. Each (back)slash blocks two edges, whose end vertices can be computed from the coordinates of the slash. If we create a hash table containing these edges, the edges incident to a vertex can be computed in $O(1)$ time. We can also use a hash table to map coordinates to vertex identifiers in a range of size $O(hw)$.

To find the cycles, we perform a DFS starting from a vertex in each connected component. The DFS will either find that the component is acyclic, or that it contains a cycle. (In fact, the component will *be* a cycle in this case.) In the latter case, the length of the cycle can be found as in Sedgewick Algorithm X, or simply as the height of the recursion stack. While doing this, it is trivial to count the cycles, and find the longest cycle.

To explore each connected component exactly once we keep the markings between BFS runs, and only initiate BFS on unmarked vertices. This means that *the running time becomes linear in the size of the graph, which is $O(hw)$.* This is also the (expected) time for reading the input and constructing the graph. *The space usage is $O(hw)$*, since we need to store two blocked edges for each input character.

Analysis and big-O 1, January 2010 exam

Information is lacking about the constant factor in the big-O expression. We could compute or measure the space usage for a small value of n, e.g., $n = 1000$. Then input size $n = 1000k$ would require roughly k^2 times more space. Of course, this assumes that n^2 is the dominating term in the big-O expression, also for small values of n.

Analysis and big-O 5, January 2010 exam

The running time is $O(n^2)$ since there are two nested loops, that each runs for at most n iterations. This is also a precise estimate: for $i < n/4$ the inner loop makes at least $n/2$ iterations, so the conditional statement is rune at least $n^2/8 = \Omega(n^2)$ times.

Analysis and big-O 8, January 2010 exam

We assume that the sorting is done by an $O(n \log n)$ time algorithm, such as mergesort. *The running time $T(n)$ of the algorithm on an input of size n satisfies the following recurrence relation:*

$$T(n) \leq O(n \log n) + 2T(n/2)$$

Observe that the $O(n)$ time to combine results is dominated by the time for the sorting algorithm. The recurrence is similar to that of mergesort, with an additional $O(\log n)$ factor multiplied. Therefore, the running time is $O(n \log^2 n)$. By considering the recursion tree, we see that the running time is in fact $\Omega(n \log n)$ for the first $\frac{1}{2} \log n$ levels of the recursion, so this analysis is tight.

Correctness 3, January 2010 exam

The invariant is that $(i-1)^2 \leq n$. This is true initially, since $i = 1$, and after each iteration because of the conditional statement that returns when $i^2 > n$.

Toolbox 1, January 2010 exam

A hash table can be used to count the number of occurrences of each item, in $O(1)$ time per occurrence. This time usage is in expectation. Then a selection algorithm can be used on the counts, to find the kth highest count, after which it is trivial to find the corresponding elements in $O(n)$ time. If we exploit that all numbers are less than n^2, we can instead count by radix sorting the numbers in $O(n)$ time (worst case).