Algorithms Re-Exam TIN093/DIT600

Course: Algorithms

Course code: TIN 093 (CTH), DIT 600 (GU)

Date, time: 7th January 2016, 8:30–12:30

Building: M

Responsible teacher: Peter Damaschke, Tel. 5405.

Examiner: Peter Damaschke.

Exam aids: a handwritten A4 paper (both sides), dictionary, the printed Lecture Notes; any edition of Kleinberg, Tardos: "Algorithm Design".

Time for questions: around 9:30 and around 11:00.

Solutions: will appear on the course homepage.

Results: will appear in ladok.

Point limits: CTH: 28 for 3, 38 for 4, 48 for 5; GU: 28 for G, 48 for VG; PhD students: 38. Maximum: 60.

Inspection of grading (exam review): times will be announced on the course homepage.

Instructions and Advice:

- Numbers in parentheses are the points.
- First read through all problems, such that you know what is unclear to you and what to ask the responsible teacher.
- Write solutions in English.
- Start every new problem on a new sheet of paper.
- Write your exam number on every sheet.
- Write legible. Unreadable solutions will not get points.
- Answer precisely and to the point, without digressions.
  Unnecessary additional writing may obscure the actual solutions.
- Motivate all claims and answers.
- Strictly avoid code for describing an algorithm.
  Instead *explain* how the algorithm works.
- If you cannot manage a problem completely, still provide your approach or partial solution to earn some points.

Good luck!

Problem 1: Binary Search Inside Arithmetic (12)

For simplicity assume in the following that a multiplication of two integers with d digits costs $O(d^2)$ time, albeit faster algorithms are known.

Let n be a given integer with k digits. We wish to compute $x = \lfloor \sqrt{n} \rfloor$, the integer part of the square root of n. One simple idea is to test for an integer x whether $x^2 < n$ or $x^2 > n$ and, depending on the result, to increase or decrease x. In this way we can find the correct x by binary search.

We will figure out a time bound for the algorithm sketched above. Note that the input length is the number k of digits, and digit operations are the elementary operations to be counted here. In detail:

1.1. How "long" are the tested numbers x? That is, specify the number of digits in O-notation. (2)

1.2. How many steps of binary search are needed? Express this again as a function of k, in O-notation. (2)

1.3. How many digit operations are needed in each step of binary search? Do not forget that comparisons of numbers cost time as well. (3)

1.4. Finally, put things together to obtain an overall time bound. It should be polynomially bounded in k. (2)

1.5. Discuss: What changes in the time bound if we wish to compute the cubic root $x = \lfloor \sqrt[3]{n} \rfloor$ instead? (3)

Problem 2: Reduction to Longest Paths (14)

In the Weighted Interval Scheduling problem we are given n intervals $[s_i, f_i]$, $i = 1, \ldots, n$, each with a positive weight v_i, and the goal is to select a subset of pairwise disjoint intervals with maximum total weight. We can assume that all $2n$ numbers s_i and f_i are distinct.

Instead of solving the problem by a special-purpose dynamic programming algorithm we may get the idea to reduce it to Longest Paths in DAGs as follows, and then solve the latter problem.

Reduction: Create a node for each s_i and f_i. For each i, create a directed edge of length v_i from s_i to f_i. For all pairs f_i, s_j with $f_i < s_j$, create a directed edge of length 0 from f_i to s_j. Also create two special nodes a and b, and directed edges of length 0 from a to every s_i, and from every f_i to b.

2.1. Show that this graph is, in fact, a DAG. (3)

2.2. Show equivalence of the two problems. Assertion: The Weighted Interval Scheduling instance has a solution with total weight at least L, if and only if the DAG has a directed path from a to b of length at least L. (7)

2.3. How much time (as a function of n) would it take to solve Weighted Interval Scheduling in this way? Note that both the reduction and the Longest Path algorithm need time. (4)

Problem 3: Obstacles in a Grid (12)

Imagine a usual (x, y)-coordinate system in the plane. In the following, a unit square means a square with corners (i, j), $(i+1, j)$, $(i, j+1)$, $(i+1, j+1)$, where i and j are integers. We only consider a finite part of the plane: the $n \times n$ grid with corners $(0, 0)$, $(n, 0)$, $(0, n)$, (n, n).

Suppose we want to walk from point $(0, 0)$ to point (n, n), but some unit squares are blocked by obstacles. We can always walk from (i, j) to $(i+1, j)$ or to $(i, j+1)$; clearly, the distance is 1. We can also walk diagonally from (i, j) to $(i+1, j+1)$, *but only if that square is not blocked.* In this case the distance is $\sqrt{2}$.

We define the function $OPT(i, j)$ as the length of a shortest path from $(0, 0)$ to (i, j). The information which of the n^2 unit squares are blocked is given.

3.1. Derive a recursive formula for $OPT(i, j)$ and explain its correctness. The correctness argument can be informal but should contain all necessary thoughts. (7)

3.2. What is the running time (as a function of n) of the resulting dynamic programming algorithm that computes a shortest path from $(0, 0)$ to (n, n)? Do not forget the backtracing phase. Also state clearly what you count as elementary operations. (5)

Remark: Similarly to Problem 2 we might reduce this one to *Shortest* Paths in graphs, but this would unnecessarily cost some extra time for constructing the graph. Dynamic programming is faster in practice.

Problem 4: Many Robots Moving on a Street (10)

Imagine that n robots are placed at points with coordinates $x_1, \ldots, x_n$ on the line. Our task is to move them to n given points $y_1, \ldots, y_n$ on the same line. The robots are identical. It is not specified which robot to be placed on which point, the only demand is to occupy all given points $y_1, \ldots, y_n$. However we want to minimize the sum of distances traveled by the n robots.

4.1. Suppose for a moment that the given coordinates are sorted, that is: $x_1 < \ldots < x_n$ and $y_1 < \ldots < y_n$. Prove that the optimal solution is to move the robot from x_i to y_i, for all $i = 1, \ldots, n$.
Hint: Assume that an optimal solution is different from the proposed one, and then provide an exchange argument. (5)

4.2. Now we do not suppose any more that the coordinates are sorted. How much time does it take to compute an optimal solution (including the sum of travel distances), and how do you proceed? – Use the result of 4.1. The time must be polynomially bounded in n. (3)

4.3. How much time would a trivial exhaustive search algorithm need, that simply tests all one-to-one assignments of x-values to y-values and finally takes the best? Is this time still polynomial in n? (2)

Problem 5: The Traveling Salesman – a Special Case (12)

Yet another type of problem related to shortest paths is the Traveling Salesman Problem (TSP). In a special case called *Graphic TSP*, only an undirected graph $G = (V, E)$ with n nodes is given, along with an integer t. All edges have unit length. The distance $d(u, v)$ between two nodes u and v is the length of a shortest path connecting u with v. A *tour* is any permutation $(v_1, v_2, \ldots, v_n)$ of V. The problem is whether G admits a tour (permutation of V) that starts and ends in the same node and has total travel distance at most t:

$$d(v_n, v_1) + \sum_{i=1}^{n-1} d(v_i, v_{i+1}) \leq t$$

(In the optimization version of Graphic TSP we would ask for a shortest tour in G, without a predefined threshold t.) Note that a tour visits the nodes $v_1, v_2, \ldots, v_n$ in this order, but may also visit other nodes repeatedly in order to get from each v_i to v_{i+1}. Equivalently we can say that a tour visits every node *at least* once. **See next page.**

5.1. Show that Graphic TSP belongs to the complexity class NP. (2)

5.2. Give a polynomial-time reduction from the Hamilitonian Cycle problem to Graphic TSP. (7)

5.3. What do 5.1 and 5.2 together imply for the computational complexity of Graphic TSP? (3)

Solutions (attached after the exam)

1.1. The numbers x still have $O(k)$ digits. It “does not help” that we have $x \approx \sqrt{n}$ most of the time, because $\sqrt{n}$ has about $k/2$ digits. (2)

1.2. Since $n = 10^k$, we have to do $O(\log n) = O(k)$ tests during binary search. (2)

1.3. Each step involves a multiplication (squaring x) in $O(k^2)$ time and a comparison of x^2 with n, in $O(k)$ time. These time bounds follow from the lengths in 1.1. Together this is $O(k^2 + k) = O(k^2)$. (3)

1.4. Due to 1.2 and 1.3, the total time is simply $O(k \cdot k^2) = O(k^3)$. (2)

1.5. The binary search procedure is the same, except that we use x^3 rather than x^2. Now a step involves two multiplications, but the time is still $O(k^2)$. Hence the overall time does not change, up to a constant factor. (3)

2.1. The s_i and f_i are already given as real numbers. We also represent node a and b as a number being smaller and larger, respecively, than all other numbers. Then, directed edges (x, y) exist only between numbers with $x < y$, hence the graph cannot have directed cycles. (3)

2.2. Consider a set X of disjoint intervals with total weight at least L. The path that visits a, all start and end points of the intervals in X in their natural order, and finally b, has length at least L, due to the definition of edge lengths. Conversely, consider a directed path P from a to b, of leangth at least L. Only edges that represent intervals have positive lengths, and none of these intervals can intersect, as the path goes strictly from left to right. Hence the “intervals on P” form a solution to Weighted Interval Scheduling with total weight at least L. (7)

2.3. Edges connect every f_i with every larger s_j, and there could be $m = O(n^2)$ such pairs of points. This is also the time bound of the reduction. A longest path in a DAG can be found in $O(m + n) = O(n^2)$ time. Thus the overall time would be $O(n^2)$. (4)

3.1. Let $b(i, j) := \infty$ if the square with upper right corner (i, j) is blocked, and $b(i, j) := 1$ otherwise. The $b(i, j)$ are given, and now we have:

$$OPT(i, j) = \min\{OPT(i-1, j)+1, OPT(i, j-1)+1, OPT(i-1, j-1)+b(i, j)\sqrt{2}\}.$$

These three options describe a horizontal, vertical, and diagonal step, where i and j can only increase. If these are the only options for the last move that reaches (i, j), the distance from $(0, 0)$ is the minimum of the three terms, by induction on i and j. We also need to argue that it is never beneficial to decrease i or j in a move: Any subpath that goes from i to a smaller $i' < i$ and later returns to i can be replaced by a subpath that is no longer and stays at i, and similarly for the j-coordinate. (7)

3.2. Counting table look-ups and arithmetic oeprations (+,min) with real numbers (truncating the irrational $\sqrt{2}$ after some decimals) as elementary operations, the computation of all $OPT(i, j)$ obviously costs $O(n^2)$ time. The backtracing phase traverses only one path in the table of $OPT(i, j)$ values, thus it costs $O(n)$ time. Together this is $O(n^2 + n) = O(n^2)$. (5)

4.1. In any solution deviating from the proposed one, there must exist two x-values $p < q$ and two y-values $r < s$, such that the robot from p moves to s, and the robot from q moves to r. Then one can check $|p - s| + |q - r| \geq |p - r| + |q - s|$. That means, it is never worse to move instead from p to r, and from q to s. (5)

4.2. Sort the x-values and y-values, respectively, in $O(n \log n)$ time, then match them as in 4.1, in $O(n)$ time, in total $O(n \log n)$. (3)

4.3. There are $n!$ such assignments. Computing and summing the distances needs $O(n)$ time for each, hence the total time is $O(n \cdot n!)$. This is not polynomial, as already $n!$ grows super-exponentially. (2)

5.1. Given a tour, we can verify in polynomial time that it includes all nodes of G. Since the shortest-path problem is solvable in polynomial time, we can also calculate the distances and add them in polynomial time. (2)

5.2. Let G, a graph with n nodes, be an instance of the Hamiltonian Cycle problem. We construct the instance (G, n) of Graphic TSP, that is, the graph is the same, and $t = n$. Trivially the reduction can be done in polynomial time, the only work is to count the nodes. G has a Hamiltonian path if and only if a tour of length at most n exists. (In summary, the same reduction as from Hamiltonian Cycle to general TSP stil works for Graphic TSP.) (7)

5.3. Since Graphic TSP is in NP and a polynomial-time reduction from an NP-complete problem (Hamiltionian Cycle) is established, Graphic TSP is NP-complete as well. (3)