Algorithms Exam TIN093/DIT600

Course: Algorithms

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

Date, time: 22nd October 2016, 14:00–18:00

Building: M

Responsible teacher: Peter Damaschke, Tel. 5405

Examiner: Peter Damaschke

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

Time for questions: around 15:00 and around 16:30.

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:

- 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.
- Facts that are known from the course material can be used.
  You don't have to repeat their proofs.

Remark: Points are not always proportional to the length or difficulty of a solution, but they also reflect the importance of a topic or skill.

Good luck!

Problem 1 (10 points)

Let $e(A, B)$ denote the edit distance of the words A and B, that is, the number of insertions, deletions and mismatches in an optimal alignment of A and B. Let AC denote the concatenation of the words A and C, that is, C is appended to A without a gap. Example: If $A = soft$ and $C = link$ then $AC = softlink$.

1.1. Prove that $e(AC, BC) = e(A, B)$ holds for any three words A, B, C. Hint: An equation is equivalent to two inequalities. (4 points)

1.2. Nowadays algorithms are much discussed in the media, in connection with big data analysis, and it (really!) happens sometimes that *algorithms* are confused with *logarithms*. Let's see how similar the words are:

What is the edit distance, and an optimal alignment, between the words *algorithm* and *logarithm*? But do not only give the result. Most importantly, explain how you obtained it, and what makes you sure that you gave the optimal solution. (6 points)

Problem 2 (10 points)

We define the following problem called Redundant Storage. Three disks, each with capacity U, are available, and n files of sizes $w_1, \ldots, w_n$ have to be stored there. (All mentioned numbers are integers.) It is required to store two copies of every file, on two different disks. The reason is obvious: In the event that one disk crashes, still at least one copy of every file remains.

Give a dynamic programming algorithm for Redundant Storage. That is, the algorithm shall assign two copies of every file to two different disks, such that the capacities are not exceeded, or report that the problem instance has no solution.

We propose already a function with Boolean values; feel free to use it: Let $R(k, x, y, z) = 1$ if copies of the first k files can be assigned such that exactly x, y, z units of space are used on the three disks. Let $R(k, x, y, z) = 0$ if this is not possible.

2.1. Give a "recursive" formula that actually computes this function R. Make sure that you also explain your formula. (6 points)

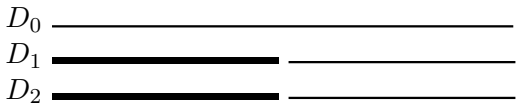
2.2. Derive a time bound for the resulting dynamic programming algorithm. It should be polynomial in n and U. (4 points)

Problem 3 (8 points)

We continue with Redundant Storage, as defined in Problem 2.

Let D_0, D_1, D_2 denote the three disks. Consider instances of Redundant Storage with the following specific properties: U is an even number, one big file with size $U/2$ exists, and the files have total size $3U/2$. (Hence, if the instance has a solution, then all disks will be full.)

After assigning the big file to two disks, say D_1 and D_2, the remaining capacity on D_0 is still U, whereas both D_1 and D_2 have the remaining capacity $U/2$ (see picture below). For each of the other files, the two copies must be put either on D_0 and D_1, or on D_0 and D_2. (We cannot put them on D_1 and D_2, since this would force us to assign both copies of some other file to D_0.) You can use this fact without proof.



Next, we define the problem Half-Half Subset Sum: Given n positive numbers $w_1, \ldots, w_n$, does there exist a subset with sum $W = \sum_{i=1}^{n} w_i/2$?
Half-Half Subset Sum is NP-complete; you can use this fact without proof.

3.1. Explain why Redundant Storage belongs to the complexity class $\mathcal{NP}$. (2 points)

3.2. Give a polynomial-time reduction from Half-Half Subset Sum to Redundant Storage. Hint: Use the special instances discussed above. (4 points)

3.3. The result of 3.2 implies that Redundant Storage is NP-complete. But, on the other hand, in 2. we have solved the problem in polynomial time in n and U. Doesn't this imply $\mathcal{P} = \mathcal{NP}$? Explain. (2 points)

Problem 4 (8 points)

Let S be a sorted set of n elements. That is, we have $S = \{s_1, \ldots, s_n\}$, where $s_1 < s_2 < \ldots < s_n$ is promised, In order to find a desired element x in S, one could do *ternary search* instead of binary search. This algorithm is described as follows. Compare x with $s_{n/3}$ and with $s_{2n/3}$. Depending on the results of these two comparisons, search for x recursively, either in the left third, or in the central third, or in the right third of S.

4.1. Write down the recurrence equation for the number $T(n)$ of comparisons, and explain all terms in this equation. That is: Where do the terms come from? (5 points)

4.2. Give the solution $T(n)$ in O-notation and state clearly how you obtained this solution. (3 points)

Problem 5 (8 points)

Lisa Pathfinder and her friends, in total k persons, live at different places in town. One day they want to meet at one place. Their sense of fairness is so overwhelming that they want to agree on a meeting point that has exactly the same distance to each of their homes. We model the street net as a graph where all edges have unit lengths. Thus our friends face the following graph problem.

Given a graph $G = (V, E)$ with n nodes and m edges, and a subset $T \subset V$ of k "terminal" nodes. All edges have length 1. Find a node $v \in V$ that has the same distance to all terminal nodes: for the chosen node v, all distences $d(t, v)$, $t \in T$, shall be equal. Moreover, $d(t, v)$ shall be minimized if a solution v exists, and otherwise the algorithm must recognize that no solution v exists. (In the latter case the friends have to be less strict and relax their equality condition. But you need not further consider that case.)

It is obvious that one can compute all pairwise distances by n times breadth-first-search (BFS), and then search for a node v that satisfies the conditions, altogether in $O(nm)$ time. However, one can easily do faster:

Propose an algorithm that solves the problem already in $O(km)$ time. Of course, it has to be correct and must not miss an optimal solution. Also give enough details such that you can establish the time bound. (8 points)

Problem 6 (16 points)

Algorithms for the various versions of the Shortest Path problem need to consider all edges (in order not to miss some shortest path) and therefore have the edge number m in their time bounds. But for certain graphs we may be able to diminish the edge number by some preprocessing, thus making the application of the "main" algorithm faster. Here is one such idea:

Let $G = (V, E)$ be an undirected graph with unit edge lengths. (All edges have length 1.) Assume that we know already some clique $C \subset V$, with k nodes. Now we change the graph as follows. We add an extra node c, delete all edges inside C, and join all nodes of C by new edges with the new node c. All these new edges get the length 1/2. Let G' denote this modified graph.

5.1. Prove that the distance between any two nodes in G equals their distance in G'. (That is, our manipulation has not changed the distances, hence it is correct to compute shortest paths in the smaller graph G' rather than in G.) (5 points)

The following questions can be answered even if you did not manage the proof in 5.1.

5.2. How large is the difference of the edge numbers in G and G'? How large must k be, in order to make the edge number in G' smaller than in G?
(3 points)

5.3. So far we have assumed that some clique C is already known. Now consider a scenario where this is not the case, and only the graph G is given. Would it be a good idea to first compute a largest clique C in G, then apply the preprocessing suggested above, and finally compute shortest paths? Give some brief and motivated statement, in favour of or against this idea. (4 points)

5.4. Consider the following problem called k-Clique: Given a graph, find a clique with k nodes. (Here k is constant, and only the graph is the instance. Note that, in the "usual" Clique problem, k is not constant but is part of the instance.)

Is the k-Clique problem with constant k still NP-complete? If you think "yes", give a polynomial-time reduction from the usual Clique problem. If you think "no", give a polynomial-time algorithm. Of course, only one of these alternatives can be true. (4 points)

Solutions (attached after the exam)

1.1. To any optimal alignment of A and B we can append the string C without adding new mismatches, therefore $e(AC, BC) \leq e(A, B)$. The opposite inequality $e(AC, BC) \geq e(A, B)$ holds because A and B is a substring of AC and BC, respectively. (4 points)

1.2. Due to 1.1 it suffices to determine the edit distance of *algo* and *loga*. Since we know already that the dynamic programming algorithm for string comparison is correct, it simply remains to apply it to these short strings. (The calculations themselves are omitted here, and they may also be omitted in the written solution.) The edit distance is 3. The optimal alignment is not unique: (*algo*, *loga*) and (*al-go*, *-loga*) are some. (6 points)

2.1. We can compute $R(k, x, y, z)$ as follows. There are three possibilities to assign the two copies of the kth file to two of the disks. Prior to that step, the space used up on the two selected disks was smaller by exactly w_k units. Therefore $R(k, x, y, z)$ equals the clause
$R(k-1, x-w_k, y-w_k, z) \vee R(k-1, x-w_k, y, z-w_k) \vee R(k-1, x, y-w_k, z-w_k)$.
(Only the recursion was requested. Initializations may be omitted, as well as the remark that the instance has a solution if and only if $R(k, x, y, z) = 1$ for some x, y, z.) (6 points)

2.2. The number of values of R to be computed is $O(nU^3)$ since $k \leq n$ and $x, y, z \leq U$. For each value we obviously need $O(1)$ arithmetic and Boolean operations. Hence the total time is $O(nU^3)$, if these operations are considered elementary. (4 points)

Special remark: One can improve the algorithm to run in $O(nU^2)$ time, using that z is uniquely determined by x, y and k. It is not expected to recognize this possibility.

3.1. An assignment of file copies to the disks can be verified in polynomial time: We check that every file is on two disks, and for every disk we add the sizes and compare the sum to U. (2 points)

3.2. Consider any instance of Half-Half Subset Sum, that is, n numbers w_i, $(i = 1, \ldots, n)$. Let $W = \sum_{i=1}^{n} w_i/2$. For the reduction we take $U := 2W$, we let the w_i be the sizes of our n files, and let W be the size of yet another file. This defines an instance of Redundant Storage of the proposed special form. Trivially, the instance is constructed in polynomial time. Since, for each file, exactly one copy must be placed on either D_1 or D_2, and the remaining space W must be filled exactly, we conclude that the instance of Redundant

Storage permits a solution if and only if a subset of the w_i has exactly the sum W. (4 points)

3.3. No, because the time bound $O(nU^3)$ is not polynomial in the input size. The number U may be exponential in n. (2 points)

4.1. $T(n) = T(n/3) + 2$, or just $T(n) = T(n/3) + O(1)$. Only one of the three possible recursive calls to an instance of size $n/3$ is done, hence the factor of $T(n/3)$ is 1. The additive term accounts for the 2 comparisons done prior to the recursive call. (5 points)

4.2. We need not do calculations from scratch. The master theorem yields $T(n) = O(n^0 \log n) = O(\log n)$, since $1 = 3^0$. (3 points)

5. We do k times BFS, only with the nodes in T as roots. This costs $O(km)$ time and yields all distances $d(t, v)$ for all $t \in T$ and $v \in V$. We store these distances in a $k \times n$ array. Then we check for every v whether the distances to all t are equal. Among all successful nodes we pick the minimum. If no such v is found, we know that no solution exists. This final phase costs only $O(kn)$ time, which is contained in $O(km)$. (8 points)

6.1.[1] Consider any shortest path P between two nodes x and y in G. Since P is a shortest path, it uses at most one edge uv with $u, v \in C$. In G' we can replace this edge with the path uc, cv of length $1/2 + 1/2 = 1$. Hence $d(x, y)$ in G' is no larger than in G. Similarly, consider any shortest path P' between two nodes x and y in G'. Since P' is a shortest path, it uses the node c at most once. Let $u, v \in C$ be the two neighbors of c on P'. In G we can replace the path uc, cv with the edge uv. Hence $d(x, y)$ in G is no larger than in G'. (5 points)

6.2. A clique with k nodes has $k(k-1)/2$ edges, and we replaced them with k new edges, hence the difference is $k(k-3)/2$. It follows that we save edges already if $k \geq 4$. (3 points)

6.3. Unfortunately, the Clique problem is NP-hard. Thus, in this way we cannot save time for shortest-path calculations in the worst case. However, we may find some cliques by fast heuristic methods. According to 6.2, replacing rather small cliques will already save some edges. (4 points)

6.4. It can be solved in $O(n^k)$ time, which is polynomial because k is fixed. The algorithm is trivial: Test all $O(n^k)$ subsets with k nodes for being a clique. (4 points)

[1] There was an obvious numbering mistake in the exam paper. Here the intended numbers 6.x are used.