Algorithms Exam TIN093/DIT602

2019-03-20

Course: Algorithms

Course code: TIN093 (CTH), DIT602 (GU)

Date, time: 2019-03-20, 8.30 am

Building: SB Multi Hall in "Samhällsbyggnadshuset", Sven Hultins gata 8

Responsible teacher: Birgit Grohe, Tel. 2037

Examiner: Birgit Grohe

Exam aids: one A4 paper (both sides) with own handwritten notes, dictionary, printouts of the Lecture Notes, printout of lecture slides. Printout of Assignments (possibly with own annotations).

Time for questions: around 9.30 and around 11.30.

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): Date 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 a complex algorithm.
  Instead *explain* how the algorithm works and/or write pseudo-code.
- 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: The number of points is not always proportional to the length or difficulty of a solution, but it may also be influenced by the importance of the topics and skills.

Good luck!

1 True or false? [10 points]

Assume that all graphs $G = (V, E)$ in this task are connected and undirected with $|V| = n$ and $|E| = m$ and edge costs c_{ij} for edges $(ij) \in E$.

1.1 Given a graph G with costs $c_{ij} = k$, $k \geq 0$. Then a minimum spanning tree (MST) can be found in $O(m + n)$.
If you think the statement is true, argue why (i.e. give an idea for an algorithm including an short argument why the running time is correct). If you think the statement is false, give an argument why it cannot be true. [3 points]

1.2 Given a graph G. Given the edge e^* with the highest cost, that is $c_{e^*} > c_e$ for all $e \neq e^*$. Then the edge e^* can never be part of any MST of G.
If you think the statement is true, give an argument or short proof. If you think the statement is false, give a counterexample. [3 points]

1.3 Given a graph G where the costs may not all be different. If the costs are not unique, there can in general be many different MST's. Suppose you are given a spanning tree $T \subseteq G$ with the guarantee that each edge $e \in T$ belongs to *some* MST in G.

1.3.1 Draw an example of a graph G that has several different MST's (also, draw at least two different MST's). [1 point]

1.3.2 Can we conclude that T itself must be an MST in G? Give a proof or a counterexample. [3 points]

2 Dynamic farmer [10 points]

Suppose that a farmer can decide from year to year which types of crops (plants) shall be grown. With different crops, the farmer can earn different amounts of money. Assume for simplicity that these prices are known in advance. However, the farmer cannot simply grow the same, most profitable crop every year. This is because every crop changes the condition of the soil, for instance, the amounts of nutrients. The sequence of crops must be planned with wisdom, such that the soil remains in good shape. Abstraction leads to the following problem:

We are given a directed graph with k nodes, and with positive edge weights $c(u, v)$, which is specified for every directed edge (u, v). A token is placed on some node s. In a *move* we can bring the token from its current place at node u to any node v, provided that (u, v) is a directed edge, and earn $c(u, v)$ units of money. Furthermore, a number n is given. The problem is to find a sequence of n moves, starting in s, that maximises the amount of money we can earn.

Give a dynamic programming algorithm for this problem, and analyse its time. It is enough to define and compute a suitable function (usually called OPT), and to motivate the time bound. You are not expected to describe the entire algorithm including backtracing. But do not forget that computing OPT requires correct initial values and an "inductive" step.

Hint: Probably your function will need two arguments: the number of moves and the current (last) node where the token is located.

3 Divide and conquer [11 points]

Your friends Anna Clever and Rolf Genius have discovered a new version of Merge sort that they have concluded must be more efficient than the textbook version. The running time of the original version of Merge sort is $O(n \log n)$ for n items, where the second factor comes from the number of recursive calls. Their idea is: if they split the items to be sorted into 3 parts instead of 2, then the algorithm will hit the base case of the recursion faster. Maybe 4 parts is even better, then maybe there will be only half of the recursive calls? Anna and Rolf are not good at recurrences and they ask you to help them find out if their idea works.

Before they manage to book a time with you, they develop their idea even further: If it is a good idea to divide into more than 2 parts, why not divide into n parts? Then you do not have to do any recursive calls at all! Did they find an $O(n)$ algorithm for sorting?

3.1 Write down pseudo-code for Merge sort with the idea to divide into 3 instead of into 2 parts. [4 points]

3.2 Write down the recurrence equation $T()$ for the algorithm in 3.1. [1 point]

3.3 Solve the recurrence stated in 3.2 and write down the running time in terms of $O()$. [2 points]

3.4 Write down the recurrence $T(n)$ for the idea to divide into 4 parts and solve the recurrence. Solve the recurrence and give the running time in $O()$ [2 point]

3.5 What about the idea to divide into n parts? Can you write down an algorithm that runs in $O(n)$ according to the idea described above?
Either write an algorithm in pseudo-code including running time analysis or give an argument why it is not possible. [2 points]

4 Tour scheduling [10 points]

A band is trying to schedule a tour. For each day of the tour, a venue has proposed a date and set a price p_i for a show. The different venues do not all pay the same amount. The band is trying to find a schedule for their tour: for every day of the tour, they have to decide if they accept the show offered that day or not. However, because of the time that it takes to travel between cities, the band cannot play two days in a row.

4.1 If the tour lasts n days, how many ways to schedule it are there (including schedules that are obviously sub-optimal, such as not playing any show)? It is enough to give the answer as a recursive function of n. [3 points]

4.2 The band manager proposes the following greedy strategy: "among the first two days, pick the venue that offers the best remuneration. Then, take the remaining possible offers for the rest of the tour (excluding the first 2 or 3 days, depending on which show you just picked) and repeat that procedure until all the offers have been considered." Does this algorithm always give the most profitable schedule? Give a proof or a counter example. [2 points]

4.3 Give a dynamic programming algorithm to compute the most profitable schedule. [4 points]

4.4 What is the running time of your algorithm in terms of $O()$? [1 point]

5 A round tour in a graph [10 points]

The Travelling salesperson problem has been studied extensively within the field of algorithms and optimisation. It is considered difficult to solve. It is usually formulated as an optimisation problem, but below we give a decision version:

Travelling salesperson problem (TSP)
Given n cities (nodes in a graph $G = (V, E)$) and distances between each pair of cities (integer weights $c_{ij} \geq 0$ on the edges). Given a positive integer B. Is there a round tour in this graph that starts at an arbitrary start city, visits all other cities exactly once and returns to the start city and where the length of this tour is at most B?

Prove that TSP is NP-complete. You may choose any NP-complete problem for the reduction, but for your convenience we include a suggestion below.

Hamilton Cycle (HC)
Given an undirected graph $G = (V, E)$ does there exist a cycle that visits every node exactly once?

5.1 Argue that TSP is in NP. [2 points]

5.2 Describe a reduction from a know NP-complete problem. Make sure that the reduction needs only polynomial time. [4 points]

5.3 Prove that the reduction is correct, that is: Show that any given instance of the known NP-complete problem chosen in 5.2 is equivalent to the instance of TSP. [4 points]

6 Breaking objects - revisited [9 points]

Remember the breaking-objects-problem from the last exercise session? Here a short reminder:

A certain industrial product will be broken if it is exposed to a mechanical shock, with a force larger than some unknown value f. If the force is at most f, then the object remains intact. We want to figure out the exact value of f by testing a number of objects. For each test, the outcome can either be *object intact* or *object broke*. Supposed that the value of f can be any integer in the interval $[1 \dots n]$. In the exercise session we constructed an algorithm with running time $O(\sqrt{n})$ that allowed us to find the value for f by breaking only two objects. The idea was to divide the interval $[1 \dots n]$ into $\sqrt{n}$ intervals of size $\sqrt{n}$ and perform two linear searches, one to find the right interval and then to find the right value for f within that interval. The entire algorithm to find f had running time $O(\sqrt{n})$.

6.1 Assume a slightly different goal: you are asked to minimise the number of tests, but are allowed to break more than two objects. What is the minimum number of tests that you have to perform? Describe your algorithm in words or in pseudo-code and give a running time analysis. [3 points]

6.2 Let B be the number of tests needed in your answer in 6.1. Assume now that you are allowed to break b objects $2 < b < B$. Give an algorithm that minimises the number of tests to find f? Write down your idea in words *and* in pseudo-code. Give the running time in $O()$ including explanation. [6 points]

Short solutions

1.1 True. If all costs are the same, then all trees have the same cost, i.e. km, and thus we can pick any tree as MST using e.g. BFS or DFS.

1.2 Not true. Counter example: If G itself is a tree, then all edges must be part of the MST, thus also e^*.

1.3.1 A graph with 3 nodes and 3 edges and equal costs.

1.3.2 Not true. Counter example: A graph with four nodes a, b, c, d and edge costs $C_{ab} = c_{cd} = 1$ and $c_{bc} = c_{bd} = 2$.

2 We define $OPT(i, v)$ to be the maximum amount that can be earned by a sequence of i moves ending in node v. We have $OPT(0, s) = 0$, and we set $OPT(0, v) := -\infty$ for every $v \neq s$ to indicate that it is impossible to start at any other node. (It would be a mistake to set $OPT(0, v) := 0$ "because zero is nothing" – then we have no information about the start node.) The function is then computed by $OPT(i, v) = \max(OPT(i-1, u) + c(u, v))$ where the maximum is taken over all nodes u such that (u, v) is a directed edge. We have to compute $O(kn)$ values, and every computation may take $O(k)$ time, hence the overall time is $O(k^2 n)$.

3.1 Assume wlog that n is a power of 3. Merge() is the merge function of the textbook version of Mergesort.
Merge3Sort(List l){
If length of l is 1, then return l (base case)
for $i = 1, 2, 3$ do l_i = Merge3Sort(i^{th} third of l)
l_{tmp} = Merge(l_1, l_2)
return Merge(l_{tmp}, l_2) }
It is also possible to do a direct merge of all three lists simultaneously.

3.2 $T(n) = 3T(n/3) + cn$, we divide the problem in three parts of equal size. The calls to Merge(), or the direct merge of three lists, take $O(n)$ together.

3.3 Using the Master theorem yields $O(n \log n)$.

3.3 $T(n) = 4T(n/4) + cn$, also $O(n \log n)$ (Master theorem)

3.5 In the lecture, we have seen a lower bound for comparison based sorting: $O(n \log n)$, which could be argument enough why their idea will not work. Even if we would find an algorithm based on the idea above, the conquer-step would not be possible to do in linear time since the number of sub-problems is not constant any more. The conquer step requires essentially pairwise comparison of all elements, thus $O(n^2)$ in total. (Thus, it would have been better to choose textbook Mergesort.)

4.1 Let $f(n)$ denote the number of possible schedules over n days. The nth day can have a show or not. In the first case, the previous day cannot

have a show, so there are $f(n-2)$ such schedules. In the second case, there is no constraint, so there are $f(n-1)$ such schedules. Therefore we have: $f(n) = f(n-1) + f(n-2)$.
To define f we also need to give the two base cases, $f(0) = 1$ and $f(1) = 2$.

4.2 Consider a three day tour with $p_1 = 1$, $p_2 = 2$ and $p_3 = 2$. The greedy will pick the solution where only the second show is played, for a profit of 2. The optimal solution is to play on day 1 and 3 for a profit of 3.

4.3 Let (n) denote the value of the most profitable schedule over the first n days. For the base cases, we have $\mathrm{OPT}(0) = 0$ and $\mathrm{OPT}(1) = p_1$. On the nth day, we can choose to play the show or not. In the first case, the previous day cannot have a show, so we get $p_n + \mathrm{OPT}(n-2)$. In the second case, the value is $\mathrm{OPT}(n-1)$.

Thus $OPT(n) = \max\{\mathrm{OPT}(n-2) + p_n, \mathrm{OPT}(n-1)\}$.

The OPT-values can be computed efficiently by an algorithm that iterates from 2 to n. The previously computed values can be stored in an array of size n, but since only the last two values are needed, it is also possible to use only two variables for storage.

4.4 $O(n)$: Compute n values, each takes a constant number of operations.

5.1 Given a tour, we can in polynomial time verify if it is a proper round tour (visits all nodes exactly once and returns to the start node) and if the length is at most B, by traversing all nodes in the graph, and edges in the tour.

5.2 Reduction by using HC: For each node in the graph of HC, create a node in the graph of TSP. For each edge in the graph of HC, create an edge with cost 1 in the TSP graph. For each pair of nodes in the HC graph that do not have an edge between them, create an edge with cost 2 (or infinity) in the TSP graph. Let $B = n$. This construction can be cone in polynomial time.

5.3 There is a solution for TSP with tour length at most $B = n$ if and only if there is a Hamilton cycle in the HC graph. If you are interested to see a very detailed solution, see Garey and Johnson: Computers and Intractability: A Guide to the Theory of NP-completeness.

6.1 Use binary search, this will break $B = \log n$ objects.

6.2 Use binary search until only 2 objects are left. Then switch to the strategy given in the problem description. This will break $(b-2)+2\sqrt{\frac{n}{2^{b-2}}}$ objects which results in running time $O(b + \sqrt{\frac{n}{2^b}})$. (If b is close to 2, this will be $O(\sqrt{n})$ and if it is close to B, then it will be similar to 6.1.)