Algorithms Exam TIN093/DIT602

Course: Algorithms

Course code: TIN 093 (CTH), DIT 602 (GU)

Date, time: 19 March 2021, morning

Location: online

Responsible teacher/examiner: Birgit Grohe

Exam aids: This is an open book exam, all aids allowed except communication with other humans. **VERY IMPORTANT: You must provide references for everything that you use from a source other than your own thoughts!** You may of course use facts from the course book or other course material, in those cases it is ok to give a very short reference e.g. "course material", "course book", or similar.

Time and location for questions: Between 9:00-12:00 and between 13:00 and 13:30 (for those with prolonged exam time). Questions in the exam-zoom room, please.

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): 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.
- Write solutions to each problem in a separate document. There will be separate uploads in Canvas for each problem (all uploads must be in pdf format).
- If possible, type your solutions on a computer. Otherwise: write legibly, preferably use a scanner app such as CamScanner, Genius Scan or similar. In any case make sure if you scan/photograph that the result is not too dark and is fully readable. Unreadable solutions will not get points.
- Write your name on each document.
- 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 use pseudo-code if asked to.
- 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? [15 points]

1.1 Recall the Minimum spanning tree problem in an undirected connected graph. True or false? Motivate briefly by giving an argument or a counter-example!

1.1.1 If some of the edge weights in the graph are negative, then Prim´s algorithm will not always return a minimum spanning tree. [2 points]

1.1.2 If the edge weights in a graph are not all distinct, then there must be at least two different minimum spanning trees. [2 points]

1.1.3 If edge (i, j) is a minimum weight edge, then this edge must be among the edges of every minimum spanning tree. [2 points]

1.2 Recall the Knapsack problem: Given n items where each item has a weight w_i and a value v_i. Given further a number W. The goal is to select a subset S of items such that $\sum_{i \in S} w_i \leq W$ and the total value $\sum_{i \in S} v_i$ is maximised. For this general form, at present, there is no known polynomial algorithm.

Consider now a variant of the problem where $w_i = c$, i.e., the weights of all items are the same. Is it possible to find a polynomial time algorithm for this variant? If you think the answer is yes, give a description in words or pseudo-code and the running time $O()$. If you think the answer is no, explain why you think it is not possible. [3 points]

1.3 You discuss the Shortest path problem with your classmates. Assume the usual notation and $c_{ij} \geq 0$. Classmate Greedy Gustav comes up with a new idea for an algorithm for finding the shortest path from node s to some node t: Start in s and go to node p closest to s (meaning $c_{sp} \leq c_{sj} \forall j$). Then, repeat this idea, now starting from p. Repeat until you hit t, then stop.

1.3.1 Write pseudo-code for Greedy Gustav´s idea. [1 point]

1.3.2 What is the running time of this algorithm? Give $O()$ and explain. [2 points]

1.3.3 Does this algorithm solve the Shortest path problem? If you think it does, give a correctness proof (if you find it difficult to give a formal proof, try to argue instead). Otherwise give a counter-example. [3 points]

2 Finding an element in an array [8 points]

Given a sorted array A of size n with distinct integers, some of which possibly negative. We are looking for an algorithm that can find an index i, $1 \leq i \leq n$ with $A[i] = i$ if such an index exists. There is a trivial algorithm that runs in $O(n)$ time, but we are looking for something more efficient.

2.1 Describe your algorithm briefly in words and then in pseudo-code. [4 points]

2.2 What is the running time of your algorithm in terms of $O()$? Explain briefly. [2 points]

2.3 Argue why your algorithm solves the problem. No formal proof needed, just write in a way in which you would explain for a classmate. [2 points]

3 Babysitting [12 points]

Assume you want to attend a 1 day (online?) event and need someone to watch your kids. You call your friends and ask if they can help you out. None of your friends can make it the entire day, but several of them can help you during parts of the day. They give you their "from" and "to" availability and your goal will now be to cover as much as possible of the day by picking a subset of your friends. Unfortunately, your friends don´t like each other, so you can´t have any overlap of their babysitting shifts. But one friend helping you until, say 1 PM, then the next taking over at 1 PM, is ok.

More precisely: Given the event time interval $[0, K]$ and your n friend´s availability time slots $[a_i, b_i]$ (real numbers where $a_i \leq b_i$). Find a selection of the time slots that covers as much of the event time as possible but does not include any overlapping time slots. (Assume that the the following holds for the smallest a_s and for the largest b_l, respectively: $a_s \geq 0$ and $b_l \leq K$.)

3.1 Find a counter example for the Greedy strategy Smallest a_i first. [1 point]

3.2 Find a counter example for the Greedy strategy Smallest b_i first. [1 point]

3.3 Assume now that the time slots are sorted by increasing b_i. Devise a Dynamic programming algorithm to solve this problem. Explain in words what you want your $OPT()$ function to expresses in the context of this problem. [2 points]

3.4 Write down the initial condition(s). [2 points]

3.5 Write down the recurrence for $OPT()$. [2 points]

3.6 Write down pseudo-code for the entire algorithm. [2 points]

3.7 What is the running time of your algorithm in terms of $O()$? Do not forget to include the time for pre-processing (sorting or similar). [2 points]

4 Sequence Comparison [5 points]

Recall the sequence comparison problem and algorithm from the course material (lecture 4 slides or course book chapter 6.6 Sequence Alignment, Peter Damaschke´s lecture notes nr 5).

Apply the algorithm to the two strings "Lehrer" and "lärare" (which mean "teacher" in German and Swedish, respectively). Assume that the gap-penalty is 1, a mismatch of vowel-consonant or consonant-consonant costs 2, a mismatch of vowel-vowel costs 1 and a match costs 0. Further assume that you can ignore capital letters, that is, treat "L" and "l" as a match.

You can choose to either run the algorithm by hand (recommended and probably fastest given the short strings) or to write a short program or use a tool in case you find a free tool available on the internet. Reminder: if you use a tool or a program, you have to include a reference or your code.

Your answer should consist of the table showing all $OPT(i, j)$ values. It should contain the final alignment (the two strings written on top of each other showing in case there were gaps inserted in the strings). Your answer should also include the total cost for the solution and a statement if the solution is unique or not.

5 Cocktail Party [10 points]

Assume you want to win the Creative Cocktails award by finding a novel cocktail. Given n ingredients, the idea is to use as many as possible to create a new drink. But some of the ingredients don´t go well together. For each pair of ingredients, there is a number t_{ij}, $0 \leq t_{ij} \leq 1$ that indicates how well ingredients i and j go together (how tasty a mix of them would be). A 0 means that i and j fit very well in a drink whereas 1 indicates that the respective i and j would taste very bad when mixed together. A cocktail with m ingredients has a total "tasty-index" that is simply the sum of t_{ij} for all pairs of ingredients in the cocktail. The goal in the contest is to create a cocktail with many ingredients, but with low tasty-index.

Consider the decision version of the above problem and let us be more precise: Is there a cocktail with at least k ingredients that has a tasty-index of at most T?

5.1 Make up a small example with 4-5 ingredients including t_{ij} and write down the cocktail (combination of ingredients) with lowest T. [1 point]

5.2 Show that the decision version of the cocktail problem is in NP. [2 points]

5.3 Pick a known NP-complete X problem and describe a reduction to the cocktail problem. You can choose any NP-complete problem, but if you want a hint try Independent set (see course material or course book). [3 points]

5.4 Prove that X has a Yes-answer if and only if the cocktail problem has a Yes-answer. [4 points]

6 Changing Graphs [10 points]

Reality is dynamic and infrastructure changes frequently. An example would be a road network: new streets may be built, existing streets blocked due to e.g. construction work. Public transit may introduce new tram stops or close existing ones, a mobile operator may build new base stations etc..

6.1 Given an undirected connected weighted graph $G = (V, E)$ with edge weights $c_{ij} \geq 0$. Assume you are given a minimum spanning tree (MST) T. Assume further that one edge e is removed and let us call the resulting graph G'.

Give an efficient algorithm to compute the MST in G'. Describe your algorithm first in words, then write pseudo-code. What is the running time of your algorithm in terms of $O()$? (You may assume that you are provided with a list of the edges in G sorted in increasing order.) [5 points]

6.2 Given a directed connected weighted graph $G = (V, E)$ with edge weights $c_{ij} \geq 0$. Furthermore, there are two distinguished nodes s and t. Assume you are given information from a shortest path algorithm and you thus know the shortest path from s to t. Assume further a new node v is added to the graph along with an unknown number $r > 0$ of edges and non-negative edge weights with one endpoint in v. Call the resulting graph G'.

Give an efficient algorithm to answer the question if the shortest path from s to t in G will be different from the one in G'. Describe your algorithm first in words, then write pseudo-code. What is the running time of your algorithm in terms of $O()$? Argue why there is no better algorithm (in terms of $O()$) than the one you propose. [5 points]

Short solutions (attached after the exam)

1.1.1 False. Prim´s works on graphs with negative edge costs since the algorithm relies on differences between edge costs and in which order they are chosen, rather than the absolute value of the costs. One can add a large positive number M to all edge costs to make them positive: The number of edges is always $n-1$ in any MST and an offset of M on all edges changes only the cost of the MST, not the MST itself.

1.1.2 False. Example: complete graph with three nodes, two of them cost=1 the third cost > 1.

1.1.3 False. Example: complete graph with three nodes and all costs=1.

1.2 Sort items according to decreasing value. Let $m = \lfloor W/c \rfloor$. Pick the first m items from the list. $O(n \log n)$. Correctness: Exchange argument: If we ignore an item among the first m items and instead pick an item later in the list, the total value will decrease.

1.3.1 Sketch of pseudo-code for Gustav´s idea (there are several other ways):

$i := s$; mark s visited; append s to H; $dist$ = inf;

while $i \neq t$ {

forall unvisited neighbours j of i do

if $dist > c_{ij}$ then { $dist := c_{ij}$; $\quad next\_node := j$; }

$i := next\_node$; mark i visited; append i to H;}

Return H.

1.3.2 The correct answer depends on the pseudo-code. For a general graph, the while and forall structure together look like $O(n^2)$: processing all nodes, and for each node process all its connected edges (at most $n-1$). However, given that nodes are marked visited, each edge is processed at most a constant number of times, thus in total $O(n+m)$ If the pseudo-code does not contain marking already visited nodes, and there are cycles, the procedure may run forever.

1.3.3 Counter-example.

2 Binary search based. While array not empty, probe the middle element. If $A[i] = i$ stop, if $A[i] < i$ ignore left half, otherwise ignore right half, repeat. $O(\log n)$ since we half the size of the array in each iteration. Why is works to ignore half of the elements in each iteration? Due to the array entries being all different together with the relation between index and size of entry (the array entry has to increase/decrease by 1 for each index).

3.1 and 3.2 Counter-examples.

3.3 Problem also known as scheduling by minimising idle-time. Let $OPT(i)$ denote the minimum idle-time using the first i intervals.

3.4 $OPT(0) = K$

3.5 Let $p(i)$ be the interval with largest index not overlapping with interval i (see Interval Scheduling from the lectures).

$OPT(i) = \min\{OPT(i-1), OPT(p(i)) - (b_i - a_i)\}$

3.6 pseudo-code (including sorting, calculation of $p(i)$ by forward scanning, $OPT(i)$ and backtracing to find the set of intervals).

3.7 Sorting $O(n \log n)$, calculation of $p(i)$ is $O(n)$, calculation of $OPT()$ array $O(n)$, backtracing (using suitable bookkeeping) $O(n)$. In total $O(n \log n)$.

One can also use the weighted Interval Scheduling problem: use the weights as the length of the intervals and re-use the algorithm from the course material with minor adjustments.

4 See separate file (image). Solution is unique since there is only one incoming arrow for each cell on the path to the last cell.

5.2 Given a solution, ie. a number of m ingredients and numbers k and T, one can easily check if $m \geq k$ (in $O(1)$) and if the tasty-index is $\leq T$ (at most $O(n^2)$).

5.3 From Independent Set. Let the nodes be the ingredients. Set $t_{ij} = 1$ if nodes i and j are connected with an edge and 0 otherwise. Set $T = 0$. The construction of the reduction is polynomial.

5.4 Given a solution to Cocktail with $T = 0$ and at least k ingredients means that all for all pairs (ij) of the ingredients in the cocktail we have $t_{ij} = 0$. Thus the corresponding nodes in the Ind Set graph form an ind set of size $\geq k$. Conversely, given an ind set of size k, then nodes in the ind set are not connected with an edge and thus their corresponding t_{ij} are $= 0$. Then T adds up to 0 as well. Thus we have a Yes answer to Cocktail if and only if there is a Yes answer to Ind set.

6.1 Check if the removed edge e was part of T ($O(n)$). If not, return T. Else: T decomposes into T_1 and T_2. Find the cheapest edge that connects the two sub-trees (if one of the T_i was a leaf node, then no MST can be found). The cost will be $\geq c_e$. Go through the sorted list of edges starting from edge f next in the list after e and pick the first edge that does not create a cycle in T_1 or T_2. Using union-find: at most $O(m)$ Find-operations in the worst case (w-c for a dense graph and if the removed edge produced equally large sub-trees).

6.2 Since we do not know neither r nor if the new edges are incoming or outgoing edges wrt v, nor do we know their cost, we cannot use the information from the SP for G. One can compute two SPs $s \to v$ and $v \to t$ by running e.g. Dijkstra´s twice, but it is more efficient to just run Dijkstra´s once on entire G'.