Algorithms TIN093/DIT093

Course: Algorithms

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

Date, time: 16 March 2022, 8.30

Location: on campus (HB4)

Responsible teacher/examiner: Birgit Grohe 031 772 2037

Exam aids: Dictionary, printouts of the Lecture Notes (possibly with own annotations), printouts of the Assignments (possibly with own annotations of the solutions of the assignments). And an additional A4 paper with own notes (both sides, handwritten or typed).

Time and location for questions: 9.30-10.30 and 11.30-12.30.

Solutions: will appear on the Canvas 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 Canvas page.

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 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 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? [12 points]

1.1 Given an undirected graph $G = (V, E)$ with edge weights $w_{ij} \geq 0$. Assume all edge weights are distinct. Let e^{max} be the edge with the largest edge cost. Let $G' = G(V, E')$ where $E' = E \backslash \{e^{max}\}$, i.e. E' is the original edge set without the most expensive edge.

Claim: The MST´s of G and G' are the same. If you think the claim is true, give an argument why. Otherwise give a counter example. [3 points]

1.2 Given a directed graph $G = (V, E)$ with edge weights $w_{ij} \geq 0$. Assume w_{ij} integer and all distinct. Given two nodes s and t, with $s \neq t$.

Claim: Then the shortest $s - t$-path is always unique. If you think the claim is true, given an argument why. Otherwise give a counter example. [3 points]

1.3 Given a graph $G = (V, E)$ where $|V| = n$ and $|E| = m$. Running times of simpler algorithms on graphs, such as e.g. Prim´s algorithm (see lecture notes 11), run in $O(m \log m)$, but are most commonly referred to as running in $O(m \log n)$.

1.3.1 The above statement indicates that $\log m \in O(\log n)$. Why does this make sense in the context of e.g. Prim´s algorithm? [3 points]

1.3.2 Is it generally true for any graph $G = (V, E)$, that $\log m \in O(\log n)$? Do not just answer yes or no, but also give an explanation. [3 points]

2 Minimum Spanning Tree Revisited [10 points]

Let the algorithm below be called Reverse-Delete-MST; it finds a minimum spanning tree in a undirected weighted graph $G = (V, E)$, $|V| = n$ and $|E| = m$. Assume G is connected and further assume that the edge weights w_{ij} are non-negative and distinct.

$T :=$ set of all edges in decreasing order
For each edge in T
    check if deleting the edge would disconnect the current graph $G = (V, T)$
    if no, delete the edge from T
return T

2.1 Prove that this algorithm returns a set of edges that form a minimum spanning tree. [5 points]

2.2 Suppose you are provided with the list of edges already sorted in decreasing order. Claim: For sparse graphs ($m \in O(n)$) it is possible to find a MST with the Reverse-Delete-MST algorithm in linear time. If you think it is true, describe your idea in words or pseudo-code, including arguments why the running time $O(n)$ holds. If you think it is not true, ague why not. [5 points]

3 Find the Special Entry [9 points]

Given a sorted array with n numbers where every number, except one, appears exactly twice and the remaining number appears only once. The goal is to find the special number, i.e. the entry that appears only once.

Examples: [4 4 5 6 6 9 9 10 10] or [1 1 3 3 4 4 6 6 9 9 12], the special entries are 5 and 12, respectively.

3.1 Design an algorithm that runs in sub-linear time. First describe your algorithm shortly in words, then give pseudo-code. Hint: divide and conquer may help. [4 points]

3.2 Explain why your algorithm works. No formal proof required, but argue why your algorithm always finds the number that only appears once. [3 points]

3.3 What is the running time of your algorithm in 3.1 in terms of $O()$? Explain shortly. [2 points]

4 Efficient Allocation of a Simulation Task [11 points]

Assume you are working as a freelance data scientist and want to run a large simulation. The simulation can be broken down into smaller simulation steps and you want to run as many of those steps as possible within a given time limit of n hours. You can rent time (in one-hour blocks) on two machines called X and Y. In hour i the access of machine X is limited by x_i simulation steps, and the same holds for Y (y_i). Furthermore, you can only process your simulation on one of the machines at a time, not on both in parallel. What makes things even more complicated: if you process the simulation on one machine and you want to switch to the other machine, the moving will cost you an entire one-hour block where the simulation can´t run on neither X or Y (i.e. switching machines means you loose one hour of computation time).
Terminology: Given n hours, let a *plan* be defined by a sequence of choices of either X, Y or "move", where the choices X and Y cannot occur in consecutive hours. A plan may start on either X or Y in the first hour.

Example:

.	$hour1$	$hour2$	$hour3$	$hour4$
X	5	1	1	5
Y	2	1	8	10

The best plan would be to choose X for the first hour, then move to Y and stay there: $\{X, move, Y, Y\}$. Total no of simulation steps: $5 + 0 + 8 + 10 = 23$.

More formally: Given $x_1 \dots x_n$ and $y_1 \dots y_n$ (the number of simulation steps available at time i on X and Y, respectively), find an *optimal plan*, i.e. a plan that maximises the total number of simulation steps.

Design an efficient algorithm that takes values $x_1 \dots x_n$ and $y_1 \dots y_n$, and that returns the value of an optimal plan.

4.1 Define the $OPT()$ function(s), i.e. describe in words what $OPT()$ denotes in terms of the index/indices you choose. [2 points]

4.2 Write down a DP recurrence for your $OPT()$ function and the initial condition(s). [4 points]

4.3 Describe the entire algorithm in words or pseudo-code. If you choose to describe in words, give sufficient details so that we can understand your answer in 4.4. [3 points]

4.4 What is the running time of your algorithm in terms of $O()$? [2 points]

5 A Difficult Path [12 points]

Let the *Hamiltonian Circuit (HC)* problem be defined as follows: given a connected, undirected, unweighted graph $G = (V, E)$, does the graph contain a round tour that starts in some node, visits all other nodes exactly once and then returns to the start node?

Further, let the *Hamiltonian Path (HP)* problem be the same as HC with the only difference that we don´t require the last node to be connected to the start node, i.e. we are only looking after a path that goes through all nodes (exactly once!), not a round tour.

5.1 Make up two small examples with 5-10 nodes each: one where there exists a HP (mark the path in your example graph), and one where there doesn´t exist a HP. [2 point]

5.2 Show that HP is in NP. [2 points]

5.3 Pick a known NP-complete problem X and suggest a polynomial time reduction. You can choose any NP-complete problem, but if you want a hint you may try the Hamiltonian circuit, see the definition above (in that case you may consider adding some nodes and edges in the reduction). [4 points]

5.4 To complete the NP-completeness proof, show that X has a Yes-answer if and only if HP has a Yes-answer. [4 points]

6 Changing Graphs [6 points]

Given an undirected, connected graph $G = (V, E)$. Assume someone has already performed a Minimum spanning tree (MST) algorithm on G and tells you which edges are in the MST.

Now a new edge e_{vw} is added to the graph with $v, w \in V$. Let us call the resulting graph G'.

Give an efficient algorithm to answer the question if the MST's in G and G' are the same or not. Describe your algorithms in words, then write pseudo-code. What is the running time of your algorithm in terms of $O()$?

For the construction of your algorithm, you may assume that you get access to the details of the graph (and the MST) stored in convenient data structures, such as e.g. array or list of the edges in increasing order. Whatever you assume, mention it explicitly if you want to use it when arguing about the running time of your algorithm.

Short solutions (attached after the exam)

1.1 Counter example: e.g. if G is a tree, or if e^{max} is a so-called *bottleneck edge*, then removing e^{max} will disconnect the graph.

1.2 Counter example with nodes s, t, v: $c_{s,t} = 3, c_{sv} = 1, c_{vt} = 2$.

1.3.1 For a connected graph $|V| = n$, $|E) = m$ we have $n - 1 \leq m \leq n^2$ and $\log m \leq 2 \log n$.

1.3.2 Even if the graph is highly disconnected, the answer from 1.3.1 holds. (If we would allow the graph to have multiple edges between each pair of nodes, then the statement is not true any longer).

2.1 Consider edge e_{vw} removed by the RD algorithm. Then there must be a path from v to w in the remaining graph. The path, together with e_{vw} formed a cycle. RD removes only edges that break cycles and preserve connectivity. After considering all edges, the remaining edges form a tree that is a spanning tree (ST). The tree is also a *minimum* ST: For each removed edge that broke a cycle, the other edges on the cycle had smaller weight due to the order in which RD removes edges. The most expensive edge on a cycle cannot be in the MST (given all edge weights are different), see KT Lemma 4.20. Or use the lemma from lecture notes 11.

2.2 Due to a typo in the question, this question has been taken out of the count for the grades, the grade boundaries have been adjusted accordingly.

3.1 The array has $n = 2k + 1$ entries. Look at the middle entry a_k and its neighbours. If $a_{k-1} < a_k < a_{k+1}$ then a_k is the special element. If $a_{k-1} = a_k$ then recursively continue searching in the left half $[a_1 \dots a_{k-2}]$ of the left half is of odd size. Otherwise right half. If $a_k = a_{k+1}$ accordingly.

Find_special($[a_1 \dots a_n]$)

If $n \leq 3$ Find special entry with adhoc method and Return it

Pick middle element a_k

If $a_{k-1} < a_k < a_{k+1}$ return a_k

Else if $a_{k-1} = a_k$

If $[a_1 \dots a_{k-2}]$ is of odd length then Find_special($[a_1 \dots a_{k-2}]$)

else Find_special ($[a_{k+1} \dots a_n]$)

Else // $a_k = a_{k+1}$

If $[a_{k+2} \dots a_n]$ is of odd length then Find_special($[a_{k+2} \dots a_n]$)

else Find_special($[a_1 \dots a_{k-1}]$)

3.2 Looking at the middle entry, we can find the special entry or remove a pair, thereby cutting the array in two halves, one of odd size and one of even size. The special entry must be in the in the odd-sized sub-array.

3.3 $O(logn)$ since array is cut into at least half in each iteration.

4.1 Define two functions $OPT(i, X)$ and $OPT(i, Y)$ to be the max value of the plan in hours $1 \dots i$ ending on machine X and Y, respectively (there exist other solutions).

4.2 $OPT(i, X) = x_i + \max\{OPT(i-1, X), OPT(i-2, Y)\}$ and $OPT(i, Y) = y_i + \max\{OPT(i-1, Y), OPT(i-2, X)\}$.

Initial conditions $OPT(1, X) = x_1$ and $OPT(1, Y) = y_1$.

4.3 Init $OPT(1, X) = x_1$ and $OPT(1, Y) = y_1$

for $i = 2 \dots n$

$OPT(i, X) = x_i + \max\{OPT(i-1, X), OPT(i-2, Y)\}$

$OPT(i, Y) = y_i + \max\{OPT(i-1, Y), OPT(i-2, X)\}$

Return $\max\{OPT(n, X), OPT(n, Y)\}$

4.4 The work inside the for-loop is constant, thus overall running time $O(n)$.

5.2 $|V| = n$. Follow the nodes on the path, mark each node visited and count the nodes. If the end of the path is reached and n nodes are visited without visiting a node multiple times, then the path is a HP. Walking the path and marking/checking/counting nodes is polynomial.

5.3 Add three new nodes: s, t and one more node: Pick an arbitrary node v and make a copy of it (v'). Connect v' to the same nodes as v. Then connect s and v. Connect t and v'. The addition of nodes and edges is polynomial.

5.4 Assume there is a HC in the HC graph, then there is a "path" from v to v that visits all nodes. Then there must be a path from v to v' in the HP graph. Since the HP graph contains the edges e_{sv} and $e_{tv'}$, there is a path from s to t.

Assume there is an path in the HP graph. Such a path must have its endpoints in s and t. The path will also contain all other nodes. Consequently, the second and second-last nodes on this path will be v and v'. Since v and v' where the same node in the HC graph, there must be a cycle in the HC graph.

6 The MST in G has n nodes and $n-1$ edges. Edge e_{vw} forms a cycle with the MST in G. Due to the cycle property, the most expensive edge must not be in the MST. Find the cycle by running BFS on the MST with root v, stop when w is reached. Find the most expensive e^{max} edge on this cycle (excluding e_{uv}). If $e^{max} = e_{uv}$ then the MST´s of G and G' are the same, otherwise not.

A tree structured graph has $O(n)$ edges, thus BFS runs in $O(n)$. Finding most expensive edge in the cycle is at most $O(n)$.