Algorithms Exam TIN093/DIT602

Course: Algorithms

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

Date, time: 18 March 2020, morning 8:30-12:30

Building: Samhällsbyggnad, SB Multisal

Responsible teacher: Birgit Grohe, Tel. 2037

Examiner: Birgit Grohe

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

Time for questions: Around one hour after start and one hour before end.

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.
- 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.
- 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 Given a connected, undirected graph $G = (V, E)$. Is it possible to find some node $v \in V$ such that removing node v and all its incident edges will not result in an disconnected graph (i.e. the resulting graph will still be connected)? If you think this is true give an algorithm (and possibly an example graph to show how the algorithm works). If you think it is not true, give a short proof/argument. [3 points]

1.2 Given an connected, undirected graph $G = (V, E)$ with edge costs $c_{ij}, (ij) \in E$ where the edge costs are allowed to be both positive and negative. Do any of the miniumum spanning tree (MST) algorithms presented in class give the correct MST? If you think it is true, present a short proof or argument why the algorithm(s) work. If you think it is not true, present a counter example. [3 points]

1.3 Given an connected, undirected graph $G = (V, E)$ with non-negative edge costs $c_{ij}, (ij) \in E$. If the edge costs are non-unique, then there can be different MSTs with the same value. If the edge costs are unique, then the MST is unique, that is only one MST with the minimum value exists (these two facts are true, you don't have to prove them). Here is the True-or-False question: If the edge costs are non-unique, then there are always at least two MSTs with the same value. If you think it is true, give a short proof/argument. If you think it is not true, give a counter example [3 points]

1.4 Given two functions $f(n) = n^k$ and $g(n) = n^{k+1}$, $k \geq 0$ integer. Then $f(n) \in O(g(n))$ and $g(n) \in O(f(n))$. [3 points]

1.5 It is possible to sort an array that only consists of 0's and 1's in linear time. If you think it is true, give an algorithm and a short explanation why it is linear. If you think it is not true, give a proof/argument why it cannot be true. [3 points]

2 Longest Common Sub-sequence [10 points]

Consider a sequence of letters or numbers. A *sub-sequence* of a given sequence is just the given sequence with 0 or more elements left out. Formally, given two sequences $X = \langle x_1 \dots x_m \rangle$ and $Z = \langle z_1 \dots {}_k \rangle$ (assume $m \geq k$), then Z is a sub-sequence of X if there exists a strictly increasing sequence $\langle i_1, \dots, i_k \rangle$ of indices of X such that for all $j = 1 \dots k$ we have $x_j = z_j$.

For example, consider sequence $X = \langle A, B, C, B, D, A, B \rangle$. Then $Z = \langle B, C, D, B \rangle$ is a sub-sequence of X with indices $\langle 2, 3, 5, 7 \rangle$.

Given two sequences X and Y, we say Z is a *common sub-sequence* if Z is a sub-sequence of both X and Y.

Now consider that we want to find a *longest common sub-sequence (LCS)* of two sequences X and Y, i.e., the common sub-sequence with the maximum length.

2.1 A brute force solution would be to enumerate all sub-sequences of X and compare them to all sub-sequences of Y. How many sub-sequences would one have to enumerate and how many comparisons would that result in, in terms of $O()$? [2 points]

2.2 Give a dynamic programming algorithm to solve the LCS problem and analyse its running time in terms of $O()$. You do not need to give a full pseudo-code, a suitable function $OPT()$ and an explain in words is enough. Do not forget the initial condition(s). [8 points]

3 Recurrence Relations [10 points]

Suppose that to solve a given problem, we are given three different divide and conquer algorithms. Let the size of the input be n.

- Algorithm A finds a solution by dividing it into five sub-problems, each of half the size of its input problem, recursively solves each of these five sub-problems, and then combines their solution to form the solution of the input problem in linear time.
- Algorithm B finds a solution by recursively solving two sub-problems of size $n-1$ each, and then combines their solutions to form the solution of the input problem in a constant time.
- Algorithm C finds a solution by recursively solving 9 sub-problems of size $n/3$, and then combines their solutions to form the solution of the input problem in quadratic time.

3.1 For each of the algorithms A, B and C, write down the recurrence relation. [3 points]

3.2 Solve the recurrence relations and give the running time in terms of $O()$ for each of the algorithms (you may use the master theorem if applicable). [6 points]

3.3 Which of the algorithms would you choose if asked to implement an as efficient algorithm as possible? Give a short explanation of your choice. [1 point]

4 Maximum Bottleneck Paths [8 points]

Assume you work for a company that operates vehicles which carry wide load. Your task is to find paths from a starting location to a target location, but instead of finding the shortest or cheapest path you are only interested in paths that contain wide streets.

More formally: Given a directed graph $G = (V, E)$ with edge weights $w_{ij} \geq 0$, $(i, j) \in E$ (where w_{ij} denotes the width of corresponding street in the problem described above). Given further a start node s and a finish node t. The the *Maximum bottleneck problem* is about find a path from s to t where the minimum w_{ij}, called the *bottleneck*, is as large as possible.

Example: Given two $s - t$ paths P_1 and P_2, and let the edge weights of P_1 be $(5, 7, 3, 9, 11)$ and those of P_2 be $(6, 4, 4, 8)$. Then the bottleneck of P_1 is 3 and the bottleneck of P_2 is 4. The path with the largest bottleneck is thus P_2.

Design an algorithm that solves this problem and analyse its running time in terms of $O()$. Describe your algorithm both in words and in pseudo-code. (Hint: maybe this problem reminds you of another prominent problem within the field of algorithms.)

5 The Hitting Set Problem [10 points]

Given a set $A = \{a_1, ..., a_n\}$ and an accompanying collection of subsets of A, $\mathbf{B} = \{B_1, ..., B_m\}$ (meaning that $B_i \subseteq A$ for each $i = 1, ..., m$).
We define a **hitting set** H for the collection $\mathbf{B}$ to be a subset of A (so $H \subseteq A$) such that $H \cap B_i \neq \emptyset$ for each $i = 1, ..., m$. What this entails is that for each $i = 1, ..., m$ in the collection $\mathbf{B}$, H must contain at least one element from B_i, and it is in this sense that H is said to *hit* all the sets in the collection.
The Hitting Set problem (HS) asks the following question: given a set A and a collection of subsets $\mathbf{B}$, and given a number k, is there a hitting set $H \subseteq A$ for $\mathbf{B}$ such that $|H| \leq k$?

5.1 Show that HS is in NP. [2 points]

5.2 For some NP-complete problem Y, show that $Y \leq_p HS$, i.e.: give a polynomial-time reduction from the problem Y of your choosing to HC. Do so given the following steps:

5.2.1 Provide a translation from Y to HS: create an instance of HS given the input to Y. Show that the translation can be done in polynomial time.[4 points]

5.2.2 Show that if the translated instance of HS has a solution (a valid hitting set), then Y has a solution. That is to say:
*"If HS gives an answer **yes** for the translated instance, then Y also gives an answer **yes**."* [2 points]

5.2.3 Show that if there is a solution for Y, then the translated instance of HS has a solution (a valid hitting set). That is to say:
*"If Y gives an answer **yes**, then HS also gives an answer **yes** for the translated instance."* [2 points]

As suggestions, you could take one of the following NP-complete problems:

3-SAT Given a set of (disjunctive) clauses $C_1, ..., C_k$ over a set of Boolean variables $X = \{x_1, ..., x_n\}$, being each clause C_i of length 3. Is there a satisfying truth assignment for X satisfying all clauses $C_1, ..., C_k$?

Vertex Cover Given a graph $G = (V, E)$, a set $S \subseteq V$ is called a vertex cover if every edge $e \in E$ has at least one end in S (i.e.: if $e = (u, v)$ then at least one of u, v is in S). Given a number k, does G contain a vertex cover of size at most k?

Set Cover Given a set $U = \{u_1, ..., u_n\}$ of n elements, a collection $S_1, ..., S_m$ of subsets of U ($S_j \subseteq U$ for all $j = 1, ..., m$), and a number k. Is there a collection of at most k of these sets whose union is equal to all of U?

6 Running time analysis [7 points]

Let p be a positive integer. The following procedure computes 2^p:

```
power($p$) {
    $pow$ := 1;
    for $i$ := 1 to $p$ do
        $pow$ := pow * 2
    return $pow$
}
```

4.1 Assume that multiplication can be done in constant time. What is the running time of this algorithm in terms of the size of the input? (That is, first think of what is the size of the input?) Explain. [2 points].

4.2 Assume that multiplication cannot be done in constant time, but "costs" depending on the size of the numbers to be multiplied. Assume "$x * y$" costs $O(\text{size}(x) * \text{size}(y))$. What is the running time of this algorithm? Explain. [3 points]

4.3 Is this a polynomial algorithm or not? Why? [2 points]