DELFT UNIVERSITY OF TECHNOLOGY
Faculty of Electrical Engineering, Mathematics and Computer Science

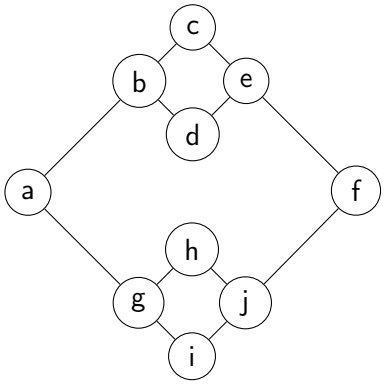
Exam IN4301 Advanced Algorithms Part II

20 November 2020 | 09:00–11:00

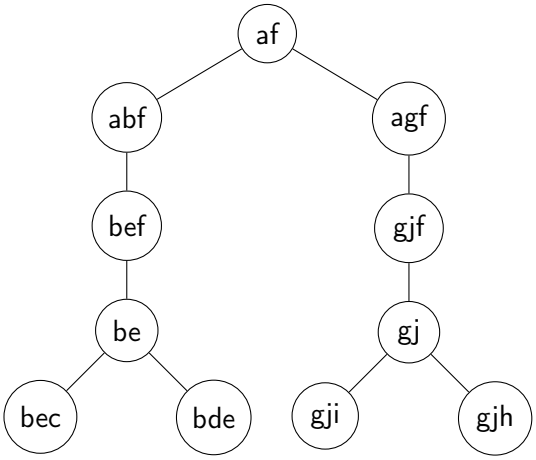
- Start each question on a new page.
- This is an open-book individual examination with 4 questions worth 24 points in total.
- If your score is n points, then your grade for this exam will be $1 + \frac{9}{24}n$.
- Use of course books, readers, notes, and slides is permitted.
- Use of computing devices, including calculators, mobile devices and laptops, is permitted.
- Use of websites such as but not limited to Google, Stack Exchange and Wikipedia is not permitted.
- By submitting your answers this exam, you agree to the following Code of Honour pledge: "I promise that I have not used unauthorized help from people or other sources for completing my exam. I created the submitted answers all by myself during the time slot that was allocated."

  Name: _____ Signature: _____
- Oral fraud checks will occur immediately after the exam. **If you have not signed up for an oral interview, do so now, because if you forget, your exam is *invalid*.**
- Write clearly, use correct English, and avoid verbose explanations. Giving irrelevant information may lead to a reduction in your score. *Almost all question parts can be answered in a few lines!*
- This exam covers Chapters 10 of Kleinberg, J. and Tardos, E. (2005), *Algorithm Design*, all information on the slides of the Part II of the course, everything discussed in lectures of Part II, and the set of papers as described in the study guide for Part II.
- This exam assumes a familiarity with the stated background of the course.
- The total number of pages of this exam is 4 (excluding this front page).
- Exam prepared by M. de Weerdt and N. Yorke-Smith. ©2020 TU Delft.

1. (4 points) Give a tree decomposition of the following graph that has the lowest width you can find, explain why this is a correct tree decomposition (hint: you don't need to give the definition itself, but you may use it for your explanation).



Solution: The answer maybe given by a tree like displayed below, or given in the set notation where both the tree structure and the bags need to be defined: $T = (V, E)$ with $V = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11\}$,
$E = \{(1, 2), (2, 3), (3, 4), (4, 5), (4, 6), (1, 7), (7, 8), (8, 9), (9, 10), (9, 11)\}$ and the bags $\{af\}, \{abf\}, \{bef\}, \{be\}, \{bec\}, \{bde\}, \{agf\}, \{gjf\}, \{gj\}, \{gji\}, \{gjh\}$.



Explanation: this tree meets the three other properties of a tree decomposition:

- every vertex is represented in one of the bags of the tree nodes
- for every edge, there is a tree node with both end points of that edge
- for every vertex, the bags of tree nodes containing this vertex is a connected subtree: vertices c,d,h, and i occur only in the leaves; vertices b, e, g, j occur only in the complete subtree in one of the main branches, vertices a and f occur only in the upper part of the tree
- Assign $5 - w$ points for a correct tree decomposition with width w, and 1 point for a correct explanation (using the definition).

©2020 TU Delft

- Assign no points if only vertex and edge coverage are correct, but not coherence and no tree structure is given.
- Assign 1 point if a tree structure with bags is given, but the proposed decomposition is not coherent.
- Assign up to 2 points if a set of bags is given and it meets all conditions including coherence, but the tree structure is not explicit.

2. (6 points) We consider the traveling salesperson problem (TSP) with dependencies. Given are a set S of n cities and the distances $d_{ij} \geq 0$ between each pair of cities $i, j \in S$. In this variant of the problem, additionally we are given a set of dependencies: pairs of cities (i, j) for which city i needs to be visited before city j (it is given that city 1 has no dependencies). The problem is to find a sequence of all the cities (starting and ending in city 1) which minimizes the total distance while meeting the dependencies.

   Describe how you would approach this problem using dynamic programming (i.e., provide a recursive function defining the value of the optimal solution and explain how to use it).

Solution: This is in fact a combination of TSP and scheduling with precedences.

A backward definition: Call the following with $\min_{i \in S \setminus \{1\}} OPT(S, i) + d_{i1}$:
OPT$(\{i\}, i) = d_{1i}$ and otherwise OPT$(S, i) = \min_{j \in LAST(S \setminus \{i\})} \{d_{ji} + \text{OPT}(S \setminus \{i\}, j)\}$. Here LAST$(S)$ denotes the cities in S on which no other cities depend.

- 2 points for correct *main* (i.e., $\min_{i \in S \setminus \{1\}} OPT(S, i) + d_{i1}$)
- 2 points for a reasonable recursive formula and 2 points for correct use of LAST or FIRST
- If an answer is not concise (e.g. no pseudocode or mathematical expression) subtract one point.

3. (6 points) Consider the following problem. Given a bipartite graph $G = (V_S \cup V_T, E)$, find a minimum size set ('cover') $V' \subseteq V_S$ such that for each vertex $t \in V_T$ there is a neighbor $s \in V'$ (so with $(s, t) \in E$ and we may say that s covers in this case its neighbors in V_T).[1]

   Give two rules to reduce an instance of this problem *that are as general as you can think of* (without loss of optimality), and for each of these explain briefly why it is correct.

Solution:

1. Let $N(t)$ denote the neighbors of t in V_S. If $N(t) \subseteq N(t')$ then remove t' and all adjacent edges of t' from G. This rule is correct, because any solution needs a neighbor of t to be in the cover, and this will also cover t'.

[1] Note that this exact problem variant is not included in the course material.

©2020 TU Delft

2. Let $N(s)$ denote the neighbors of s in V_T. If $N(s) \subseteq N(s')$ then remove s and all adjacent edges of s from G. This is correct, because if s would be part of the cover, then it is never worse to have s' in the cover, because s' covers all neighbors of s.

Additionally, naive rules are possible such as for vertices with degree 0 or 1.

- Assign (up to) 2 points for each correct naive rule including explanation.
- Assign (up to) 4 points for one of the general rules including explanation.

4. If large, complex systems for which we have good models do not function properly, possibly broken components can be identified (partly) automatically.[2] Let us describe a system by a set of n components A. If such a system is not working correctly, a set of tests can be done. Each test i involves a subset of components $B_i \subseteq A$. Suppose that we are given m *failed* tests involving subsets $B_1, \ldots, B_m$, and that furthermore the maximum size of each subset is less than or equal to $c \leq n$, i.e., for all i, $1 \leq i \leq m$ it holds that $|B_i| \leq c$.

   We are considering the problem of deciding whether there is a subset of components H of size at most k that could explain all the failed tests, i.e., such that for all i, H has at least one component in common with B_i.

   Below we ask you to give a bounded search tree algorithm for this problem with a runtime of the form $O\left(f(c,k) \cdot p(n,m)\right)$, where $p\left(\cdot\right)$ is a polynomial function, and $f\left(\cdot\right)$ is an arbitrary function.

   (a) (6 points) Express the solution to a problem of size k and sets B recursively as a combination of solutions to one or more reduced problems of size $k-1$ (and a smaller set B') and explain why your answer is correct.

Solution: (4 points for correct solution, 2 points for a relevant motivation)
Deciding whether a set H of at most size k exists explaining a set $B = \{B_1, \ldots, B_m\}$ of faulty tests:

1. If $k \geq 0$ and $B = \emptyset$ return "yes".
2. If $k = 0$ and $B \neq \emptyset$ return "no".
3. Otherwise, we will branch (into at most c children) by picking any test from B, and then selecting a component $x \in B$ for each branch. For each x, we assume it to be in H, and thus remove all tests B_j that contain this component from the subproblem, arriving at the subproblem *reduced by* x of finding a set of size at most $k-1$ for the remaining tests B'. Return "yes" if and only if the answer to one of these recursive calls is "yes".

We only expect an informal motivation, but this can be formally proven to be correct by using induction over the following hypothesis: H is a k-component set with non-empty intersection with each of $B = \{B_1, B_2, \ldots, B_m\}$ if and only if for some $x \in H$, $H - \{x\}$

[2] You may think for example of debugging large software infrastructures.

©2020 TU Delft

is a $(k-1)$-component set with non-empty intersection of B reduced by x, i.e., B'. This bi-implication can be proven as follows:

- First, suppose H is a k-component set with non-empty intersection with each of $B_1, B_2, \ldots, B_m$. *(To prove the bi-implication in the forward direction.)* Let $x \in H$ be given. Define $H' = H \setminus \{x\}$ and let $B' = \{B_j \mid j = 1 \leq j \leq m,\ x_i \notin B_j\}$. For each $B_j \in B'$ we know that $B_j \cap H \neq \emptyset$ and $x \notin B_j$, so also $B_j \cap H' \neq \emptyset$. Also H' is of size $k-1$.
- For the other direction, assume that for some $x \in H$, $H - \{x\}$ is a $(k-1)$-component set with non-empty intersection of B reduced by x. The intersection of H with all sets in $B \setminus B'$ contains x and thus is non-empty. Therefore H has a non-empty intersection with all components in B.

More informally, the branching is complete, because we know at least one of the components x of the selected test needs to be in H, and the subproblem is reduced accordingly.

- Assign (up to) 2 points for correct base cases (no extra points for an explanation of these).

(b) (2 points) Give the runtime of this algorithm using $O(\cdot)$ and argue why it meets the desired form given above.

Solution: In each child, the number of tests m reduces by at least one, and there are at most c branches. The runtime of this algorithm satisfies $T(m,k) \leq cT(m-1,k-1) + O(cm)$, so it is $O\left(c^k \cdot kcm\right)$ (search tree of height k and branching factor c).
If the components are represented by a hashset, we can drop the factor m.

©2020 TU Delft End of exam