Diskret Matematik og Formelle Sprog: Exam April 9, 2022

Due: Saturday April 9 at 13:00 CET.
Submission: Please write your solutions on paper. Leave ample margins on all sides, and make sure your handwriting is legible. *Start your solution of every new problem on a new sheet of paper. Please make sure to mark every sheet with your name, KU ID, exam number or something else that uniquely identifies your exam, so that it is easy to see for every sheet which exam submission it is part of.* Please try to be precise and to the point in your solutions and refrain from vague statements. Write your solutions in such a way that a fellow student of yours could read, understand, and verify your solutions. In addition to what is stated below, the general rules in the official course information always apply.
Collaboration: All problems should be solved individually. No communication or collaboration is allowed, and solutions will be checked for plagiarism.
Reference material: All course material is allowed, including textbooks, lecture notes, exercise sheets, and individual notes. You are *not* supposed, and will not need, to copy substantial parts of text verbatim, and if your solutions make heavy use of reference material, then make sure to provide clear references to what you are using. Please note that you deviate from definitions and algorithms as described in the course material at your own risk!
Grading: A score of 220 points is guaranteed to be enough to pass the exam.
About the problems: Note that the problem are of quite different types, and are *not sorted in increasing order of difficulty. Please read through the whole exam first,* before you start working on any single problem, so that you can plan which order of working on the problems makes most sense to you. Partial answers to problems can also give points, and you can get a top grade even without solving all problems. Please do not hesitate to alert the exam administrators if any problem statement is unclear. Good luck!

1 (70 p) In the following snippet of code `A` is an array indexed from 1 to `A.size` that contains elements that can be compared, and `B` is intended to be an auxiliary array of the same type and size. The functions `floor` and `ceiling` round down and up, respectively, to the nearest integer.

```
m := A.size
for i := 1 upto m
    B[i] := A[i]
while (m > 1)
    for i := 1 upto floor(m / 2)
        if (B[2 * i - 1] >= B[2 * i])
            B[i] := B[2 * i - 1]
        else
            B[i] := B[2 * i]
    if (m mod 2 == 1)
        B[ceiling(m / 2)] := B[m]
    m := ceiling(m / 2)
return B[1]
```

1a Explain in plain language what the algorithm above does (i.e., do not just rewrite the pseudocode word by word, so that your text is just a more verbose version of the pseudocode, but interpret what the code is doing and why). In particular, how can you describe the element `B[1]` that is returned at the end of the algorithm?

1b Provide an asymptotic analysis of the running time as a function of the array size m.

1c Can you improve the code to run asymptotically faster while retaining the same functionality? How much faster can you get the algorithm to run? Analyse the time complexity of your new algorithm. Can you prove that it is asymptotically optimal?

2 (100 p) After the less than stellar performance by Jakob at the DIKU 50th anniversary outreach event, only 12 brave children registered for a follow-up event that was arranged earlier this semester with the purpose of conveying the excitement of computer science to the younger generation. In view of this, the head of the Algorithms & Complexity Section Mikkel Thorup decided to take charge of the organization. In this problem we will study how Mikkel's follow-up event was arranged.

2a Since Mikkel is an avid mushroom picker, he took the 12 children to the forest to collect mushrooms. Being a good host, he of course made sure that every child found at least one mushroom. When everybody returned to DIKU, it turned out that the children had collected exactly 77 mushrooms together. Mikkel explained to the children that this meant there had to be at least two kids who had collected the same number of mushrooms. Can you describe in detail how he could have proven such a claim?

2b When all mushrooms had been cleaned, Mikkel taught the children what it means for two positive integers to be relatively prime. He then wrote the numbers $1, 2, 3, \ldots, 22$ on 22 sheets of paper and performed the following experiment:

- First, all the 22 sheets of papers were randomly shuffled.
- Then each of the 12 children randomly picked one sheet of paper.
- Finally, the children tried to identify a pair amongst themselves who held sheets of paper with relatively prime numbers.

This experiment was repeated several times, and every single time some pair of children found that they had drawn relatively prime numbers. Together with the children, Mikkel discussed whether this was just a weird coincidence or whether this always has to happen. What was the conclusion of this discussion? Please make sure to provide formal proofs backing up any claims you make.

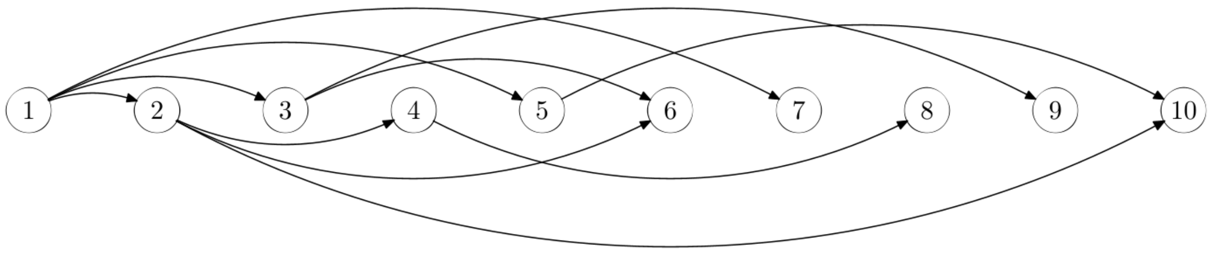

Figure 1: Directed graph D_S representing relation S in Problem 4.

3 (50 p) In this problem we consider formulas in propositional logic. Decide for each of the formulas below whether it is tautological or not and then do the following:

- If the formula is a tautology, prove this by either (i) presenting a full truth table for all subformulas analogously to how we did it in class, or (ii) providing an explanation based on the rules and equivalences we have learned. You only need to to one of (i) or (ii), but you are free to do both if you like, and crisp and clear explanations can compensate for minor slips in the truth table.

- If the formula is *not* a tautology, present a falsifying assignment. Also, explain how you can change a single connective in the formula to turn it into a tautology, and try to provide a natural language description of what the tautology you obtain in this way encodes.

3a $(p \to (q \land r)) \to ((q \lor \neg p) \land (r \lor \neg p))$

3b $((p \land q) \to r) \to ((r \lor \neg p) \land (r \lor \neg q))$

(Note that $\to$ denotes logical implication. and $\neg$ denotes logical negation. Negation is assumed to bind harder than the binary connectives, but other than that all formulas are fully parenthesized for clarity.)

4 (50 p) Consider the relation S described by the directed graph D_S in Figure 1.

4a Write down the matrix representation M_S of the relation S and describe briefly but clearly how you construct this matrix.

4b Let us write T to denote the transitive closure of the relation S. What is the matrix representation of T?

4c Now let R be the reflexive closure of the relation T. What is the matrix representation of R?

4d Can you explain in words what the relation R is by describing how it can be interpreted? (In particular, is it similar to anything we have discussed during the course?)

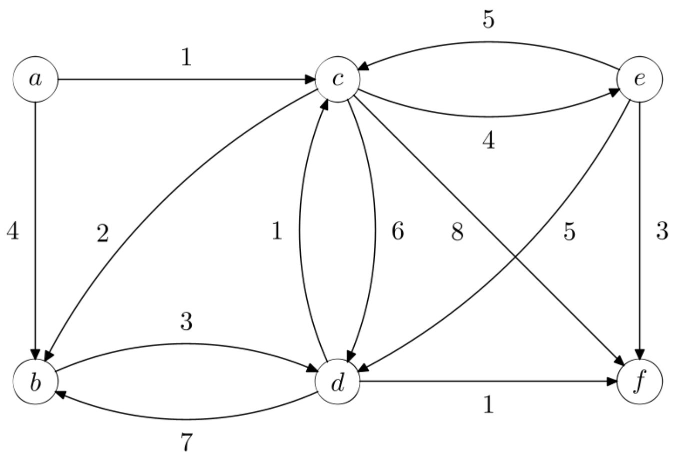

Figure 2: Directed graph D on which to run Dijkstra's algorithm in Problem 5a.

5 (70 p) In this problem we wish to understand Dijkstra's algorithm.

5a Assume that we are given the directed graph D in Figure 2. The graph is given to us in adjacency list representation, with the out-neighbours in each adjacency list sorted in lexicographic order, so that vertices are encountered in this order when going through neighbour lists. (For instance, the out-neighbour list of d is (b, c, f) sorted in that order.)

Run Dijkstra's algorithm by hand on the graph D as we have learned in class, starting in the vertex a. Show the directed tree T produced at the end of the algorithm. Use a heap for the priority queue implementation. Assume that in the array representing the heap, the vertices are initially listed in lexicographic order. During the execution of the algorithm, describe for every vertex which neighbours are being considered and how they are dealt with. Show for the first two dequeued vertices how the heap changes after the dequeueing operations and all ensuing key value updates in the priority queue. For the rest of the vertices, you should describe how the algorithm considers all neighbours of the dequeued vertices and how key values are updated, but you do not need to draw any heaps.

5b Suppose that we have another type of weighted directed graph $G = (V, E, w)$ where the weight function $w : V \to \mathbb{R}_{\geq 0}$ is defined not on the edges but on the vertices. The cost of a path $P = v_0 \to v_1 \to \ldots \to v_{n-1} \to v_n$ is $\sum_{i=0}^{n-1} w(v_i)$. As before, we are interested in finding the cheapest paths from some start vertex s to all other vertices in the graph.

Design an algorithm that solves this problem as efficiently as possible (for any directed graph G with non-negative vertex weights). Prove that your algorithm is correct, and analyze its time complexity.

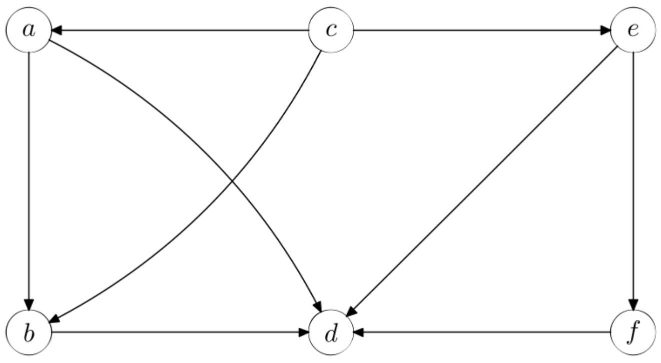

Figure 3: Directed graph H for which to compute a topological ordering in Problem 6a.

6 (90 p) In this problem we wish to understand how to compute topological orderings and/or strongly connected components of directed graphs.

6a Consider the graph H in Figure 3. The graph is again given to us with out-neighbour adjacency lists sorted in lexicographic order, so that this is the order in which vertices are encountered when going through neighbour lists. Run the topological sort algorithm on H and explain the details of the execution analogously to how we did this in class (including which recursive calls are made). Report what the resulting topological ordering is.

6b In this subproblem, we want develop our ability to parse new definitions of operations on graphs and apply these operations.

Suppose that $G = (V, E)$ is any directed graph and let $U \subseteq V$ be a subset of the vertices. Let the *contraction of G on U* be the graph $\mathsf{contract}(G, U) = (V', E')$ defined on the vertex set

$$V' = (V \setminus U) \cup \{v_U\} ,$$

where v_U is a new vertex, and with edges

$$\begin{aligned} E' = \quad & \big(E \cap ((V \setminus U) \times (V \setminus U))\big) \\ \cup\ & \big\{(v_U, w) \,\big|\, w \in V \setminus U \text{ and } \exists u \in U \text{ such that } (u, w) \in E\big\} \\ \cup\ & \big\{(w, v_U) \,\big|\, w \in V \setminus U \text{ and } \exists u \in U \text{ such that } (w, u) \in E\big\} . \end{aligned}$$

Demonstrate that you understand this definition by drawing $\mathsf{contract}(G, U_1)$ for the graph G in Figure 4 on page 6 and for $U_1 = \{a, b, d\}$. Also, draw $\mathsf{contract}(G, U_2)$ for the same graph G and $U_2 = \{e, f\}$.

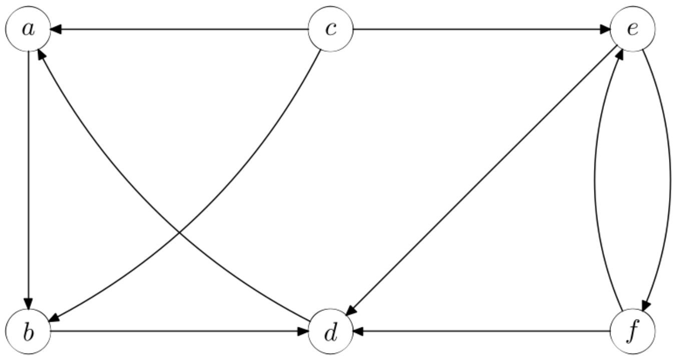

Figure 4: Directed graph G for which to compute contractions in Problem 6b.

6c Consider the following idea for an algorithm for computing the strongly connected components of a directed graph $G = (V, E)$:

1. Set $G' = G$ and label every vertex $v \in V$ by the singleton vertex set $\mathsf{vlabels}(v) = \{v\}$. Set s to be the first vertex of G (sorted in alphabetical order, say).
2. Run topological sort on G' starting with the vertex s. If this call succeeds, output $\{\mathsf{vlabels}(v) \mid v \in V(G')\}$ as the strongly connected components of G and terminate.
3. Otherwise, fix the back edge that made the topological sort fail and use this edge to find a cycle C in G'.
4. Update G' to $G' = \mathsf{contract}(G', C)$ and label the new vertex v_C by $\mathsf{vlabels}(v_C) = \bigcup_{u \in C} \mathsf{vlabels}(u)$.
5. Set s to be the new vertex v_C and go to 2.

Does this algorithm compute strongly connected components correctly? Argue why it is correct or provide a simple counter-example. (For partial credit, you can instead run the algorithm on the graph in Figure 4 and report what it does for that particular graph.)

Regardless of what the algorithm does, can you analyse the time complexity? Is the algorithm even guaranteed to terminate? If you find that the algorithm will always terminate, determine whether it will run faster or slower than the algorithm for strongly connected components that is covered in CLRS and the lecture notes.

Regardless of what the algorithm does, is it guaranteed to terminate, and if so what is the time complexity? Will the algorithm run faster or slower than the algorithm for strongly connected components that is covered in CLRS and the lecture notes.

7 (60 p) Consider the regular expression $(a(bb|c^*)d^*)^*$ and determine which of the words below belong to the language generated by this regular expression. Motivate your answers briefly but clearly by explaining for each string how it can be generated or arguing why it is impossible.

1. $bbacccd$
2. $abbdaccc$
3. $aaaccc$
4. $abbcccd$
5. $abbddcccddd$
6. $abbabba$

8 (90 p) Consider the following context-free grammars, where a, b, c are terminals, S, T, U are non-terminals, and S is the starting symbol.

Grammar 1:

$$S \to TaT \tag{1a}$$
$$T \to bT \tag{1b}$$
$$T \to ccT \tag{1c}$$
$$T \to \tag{1d}$$

Grammar 2:

$$S \to TaT \tag{2a}$$
$$T \to bTb \tag{2b}$$
$$T \to U \tag{2c}$$
$$U \to cU \tag{2d}$$
$$U \to \tag{2e}$$

Grammar 3:

$$S \to aTa \tag{3a}$$
$$T \to TbT \tag{3b}$$
$$T \to U \tag{3c}$$
$$U \to cUc \tag{3d}$$
$$U \to \tag{3e}$$

Which of these grammars generate regular languages? For each grammar, write a regular expression that generates the same language (and explain why), or argue why the language generated by the grammar is not regular. In your regular expressions, please use *only* the concatenation, alternative (|) and star (*) operators, and not the syntactic sugar extra operators that we just mentioned in class but never utilized.