KØBENHAVNS UNIVERSITET

Diskret Matematik og Formelle Sprog: Exam April 14, 2021

Due: Wednesday April 14 at 7 pm CET.
Submission: Please submit your solutions via *Digital eksamen*. It will be helpful if you state clearly your full name and your KU ID at the top of the first page. From the point of view of grading, we are of course grateful if you typeset your solutions in LaTeX or some other math-aware typesetting system. This is *not* a requirement, however, and how you write your solutions will not affect the grading as long as you make sure that everything that you write is clearly legible (and leave reasonable margins on all sides). 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 exam aids and support material are allowed, including what you can find on the internet (except that, as stated above, no communication is allowed). You are *not* supposed to copy material verbatim from the internet, however, and you should provide clear references for any material used. Please note that you deviate from definitions and algorithms as described in the course material at your own risk!
Grading: A score of 360 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 do not hesitate to send a private message on *Absalon* if any problem statement is unclear. Good luck!

1 (80 p) The DMFS main instructor is so excited by the simplicity of insertion sort and the efficiency of merge sort that he wants to get the best of both worlds. To this end, he has designed a sorting algorithm `MERGE-INSERTION-SORT` for arrays `A` that can be described in pseudocode as follows (where we assume that arrays are indexed from 1 to n and contain elements that can be compared using the operator “>”):

```
MERGE-INSERTION-SORT (A)
    if (size (A) > 1)
        mid := floor ((1 + size (A)) / 2)
        L   := new array of size mid
        R   := new array of size size (A) - mid
        for (i := 1 upto mid)
            L[i] := A[i]
        for (i := mid+1 upto size (A))
            R[i-mid] := A[i]
        MERGE-INSERTION-SORT (L)
        MERGE-INSERTION-SORT (R)
        A := MERGE-INSERT (L, R)
```

The subroutine `MERGE-INSERT` used above is as follows:

```
MERGE-INSERT (L, R)
    s := size (L)
    t := size (R)
    M := new array of size s + t
    for (i := 1 upto s)
        M[i] := L[i]
    for (i := s+1 upto s+t)
        j    := i
        M[j] := R[j-s]
        while (j > 1 and M[j-1] > M[j])
            tmp    := M[j-1]
            M[j-1] := M[j]
            M[j]   := tmp
            j      := j-1
    return M
```

1a Argue that `MERGE-INSERTION-SORT` is in fact a valid algorithm in that it will terminate on any input array `A` and return some result. Then explain what `MERGE-INSERTION-SORT` does to `A`. In particular, does it actually sort the array correctly or not? Argue why the algorithm is correct sorting algorithm, or give a counter-example showing that it is not.

1b Analyze the time complexity of `MERGE-INSERTION-SORT`. Give the best asymptotic upper bound that you can find. Provide a lower bound by constructing a family of instances that will force the algorithm to run as asymptotically slowly as possible. Can you get your upper and lower bounds to match asymptotically? (Please make sure to include all necessary details in your analysis, but note that partial answers will also give credits.)

2 (40 p) Consider the sequence $(a_n)_{n \in \mathbb{N}^+}$ defined by

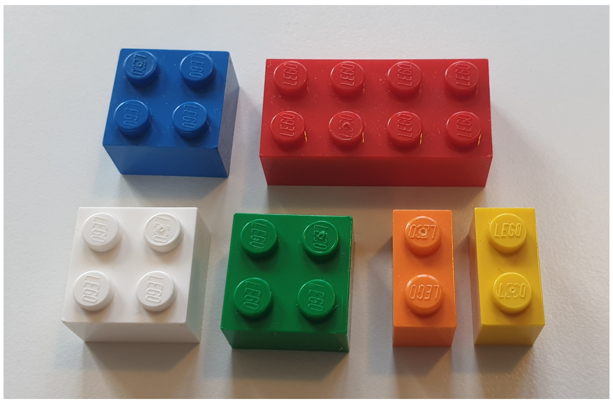
$$a_i = \begin{cases} 1 & \text{if } i = 1; \\ 3 & \text{if } i = 2; \\ 2a_{i-1} - a_{i-2} & \text{if } i > 2. \end{cases}$$

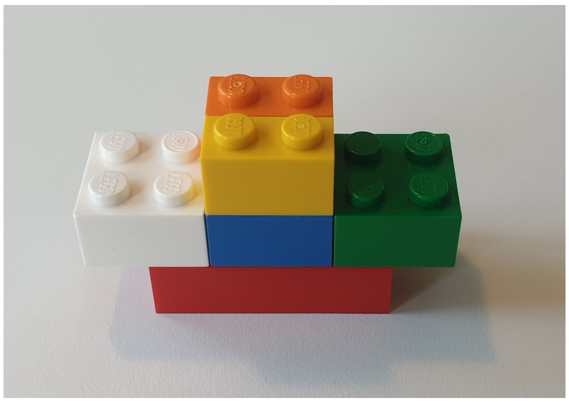
Find a closed expression (i.e., a formula) for a_n and prove that your formula is correct.

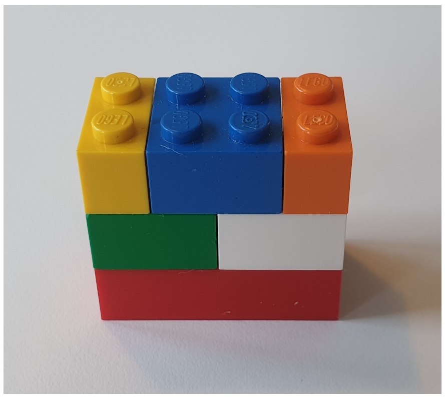
3 (50 p) Computer science exams can only be held virtually, but the Copenhagen Tivoli Gardens amusement park is apparently about to open for physical visits by the general public. This means that students frustrated by Zoom lectures and *Digital eksamen* exams will soon be able to head to the *bumper cars* (*radiobiler*) and let go of some steam by trying to bump into as many other cars (or as few cars) as possible.

Suppose that there are $n \geq 2$ bumper cars sharing an area of $m > n^2/2$ square meters.[1] An amazing mathematical fact is that after every bumper car round has finished, there are two different bumper cars that have bumped with exactly the same number of other (distinct) cars. Prove this!

[1]In reality, there are 18 bumper cars in the Copenhagen Tivoli Gardens sharing an area of 280 square meters, but this is not too relevant for this problem.


(a) A set of lego pieces.


(b) Invalid construction (not a cuboid).


(c) Valid construction of 3-layer cuboid.

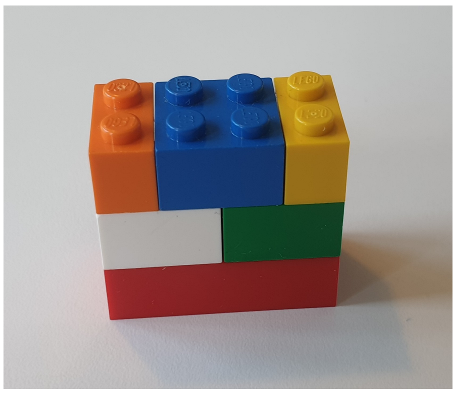

(d) Symmetric version of the same cuboid.

Figure 1: Lego pieces and constructions (and non-constructions) of cuboids for Problem 4.

4 (60 p) Another way in which Danish computer science students have been able to entertain themselves—already during lock-down—is to play with legos. In order to make this kind of activity more relevant from the point of view of discrete mathematics, rumour has it that the students often consider different sets of lego pieces (like the one in Figure 1a) and investigate in how many different ways cuboids (rectangular parallelepipeds) can be built from such pieces.

Inspired by this, let us consider the lego pieces in Figure 1a, which as we observe all have different colours. In this particular problem we want to build cuboids that are three layers high. The pieces should be put together in the standard lego way, with the lego studs pointing upwards and being fitted into the holes of any pieces above. Constructions like the one in Figure 1b are not valid—this particular construction does have three layers, and the studs point upwards and are fitted into the holes above when possible, but the resulting geometric shape is not a cuboid.

Two cuboids are considered to be the same if they can be rotated so that they become similar. Thus, the constructions in Figures 1c and 1d are both valid, but are considered to be the same cuboid, since rotating Figure 1c yields Figure 1d.

How many *different* cuboids can you build with the lego pieces in Figure 1a satisfying the restrictions described above? Please make sure to explain clearly how you reason to reach your answer.

5 (60 p) Decide for each of the propositional logic formulas below whether it is a tautology or a contradiction. If neither of these cases apply, then present one satisfying and one falsifying assignment for the formula. For a full score, please present truth tables for all the subformulas analogously to how we did it in class, and also try to *explain* why your answers are correct by interpreting what the formulas mean. Good explanations can compensate for minor slips in the truth tables.

(Note that logical not $\neg$ is assumed to bind harder than the binary connectives, but other than that all formulas are fully parenthesized for clarity. We write $\rightarrow$ for logical implication and $\leftrightarrow$ for equivalence.)

5a $((p \rightarrow q) \wedge (q \rightarrow r)) \leftrightarrow (p \rightarrow r)$

5b $(p \wedge (\neg q \vee r)) \rightarrow (q \rightarrow (p \wedge r))$

5c $((p \rightarrow q) \vee (r \rightarrow s)) \rightarrow ((p \vee r) \rightarrow (q \vee s))$

6 (70 p) In this problem we focus on relations. Suppose that $A = \{e_0, e_1, \ldots, e_5\}$ is a set of 6 elements and consider the relation R on A represented by the matrix

$$M_R = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

(where element e_i corresponds to row and column $i + 1$).

6a Let us write S to denote the symmetric closure of the relation R. What is the matrix representation of S? Can you explain in words what the relation S is by describing how it can be interpreted?

6b Now let T be the transitive closure of the relation S. What is the matrix representation of T? Can you explain in words what the relation T is by describing how it can be interpreted?

6c Suppose that we instead let T' be the transitive closure of the relation R, and then let S' be the symmetric closure of T'. Are S' and T the same relation? If they are not the same, show some way in which they differ. If they are the same, is it true that S' and T constructed in this way from some relation R on a set A will always be the same? Please make sure to motivate your answers clearly.

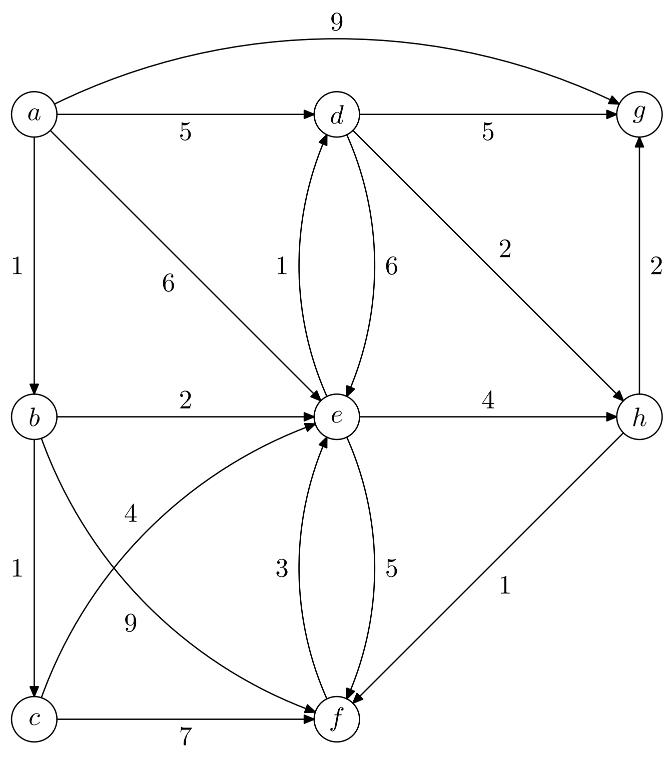


Figure 2: Directed graph for Problem 7a.

7 (100 p) Assume that we are given the directed graph 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 this is the order in which vertices are encountered when going through neighbour lists. (For instance, the out-neighbour list of a is $\{b, d, e, g\}$ sorted in that order.)

7a Run Dijkstra's algorithm by hand on this graph, starting in the vertex a. 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, it is sufficient to just describe how the key values are updated, without redrawing the heap after each operation, but you still have to describe how the algorithm considers all neighbours of the dequeued vertices. Finally, show the directed tree T produced at the end of the algorithm.

7b Consider the directed tree of shortest paths T produced in Problem 7a. Suppose that all edge weights in G are changed by some additive constant $c \in \mathbb{R}^+$. Is it true that T is still a directed tree of shortest paths for the modified graph? Please make sure to motivate your answer clearly.

7c Consider the directed tree of shortest paths T produced in Problem 7a. Suppose that all edge weights in G are changed by some multiplicative constant $c \in \mathbb{R}^+$. Is it true that T is still a directed tree of shortest paths for the modified graph? Please make sure to motivate your answer clearly.

7d Suppose that we want to give an extra bonus to paths with few hops, so that the length of a path is calculated as the sum of the weight of all the edges in the path *plus* the number of edges in the path. Describe an algorithm that can solve this problem (for any directed graph G with non-negative edge weights) and analyze its time complexity.

8 (60 p) Consider the regular expression $((ab)^*a)^* \mid (ab^*a^*b)$ 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. *abab*
2. *ababab*
3. *abbabbb*
4. *abbbbb*
5. *aaabaaba*
6. *abababa*

9 (120+ p) In this problem we want to construct a deterministic finite automaton (DFA) that recognizes the language generated by the regular expression $((ab^*)|(aac))^*$.

9a Translate this regular expression to a nondeterministic finite automaton (NFA) using the method from Mogensen's notes that we learned in class. Make sure to describe clearly which part of the regular expression corresponds to which part of the NFA, and which states are accepting. Please do not take any shortcuts without explaining what you are doing.

9b Translate the nondeterministic finite automaton to a deterministic finite automaton using the method from Mogensen's notes that we learned in class. Make sure to explain in detail how you perform the subset construction, so that it is possible to follow your line of reasoning. Present clearly the resulting DFA, with all states and transitions, and specify which states are accepting.

9c *Bonus problem (worth 40 p extra; might be hard):* How small a DFA can you produce that accepts precisely the language generated by the regular expression above? Can you prove that the size of your DFA is optimal? Note that that you can solve this problem even if you did not solve the other subproblems above.

10 (90 p) Consider the following context-free grammars, where a, b, c are terminals, B, S are non-terminals, and S is the starting symbol.

Grammar 1:

$$S \to aS \tag{1a}$$
$$S \to B \tag{1b}$$
$$B \to bcB \tag{1c}$$
$$B \to \tag{1d}$$

Grammar 2:

$$S \to aS \tag{2a}$$
$$S \to BS \tag{2b}$$
$$S \to B \tag{2c}$$
$$B \to bcB \tag{2d}$$
$$B \to \tag{2e}$$

Grammar 3:

$$S \to aS \tag{3a}$$
$$S \to BS \tag{3b}$$
$$S \to B \tag{3c}$$
$$B \to bBc \tag{3d}$$
$$B \to \tag{3e}$$

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.