KØBENHAVNS UNIVERSITET

Diskret Matematik og Formelle Sprog: Problem Set 4

Due: Monday March 21 at 12:59 CET.
Submission: Please submit your solutions via *Absalon* as PDF file. State your name and e-mail address close to the top of the first page. Solutions should be written in LaTeX or some other math-aware typesetting system with reasonable margins on all sides (at least 2.5 cm). Please try to be precise and to the point in your solutions and refrain from vague statements. Make sure to explain your reasoning. *Write so that a fellow student of yours can read, understand, and verify your solutions.* In addition to what is stated below, the general rules for problem sets stated on *Absalon* always apply.
Collaboration: Discussions of ideas in groups of two to three people are allowed—and indeed, encouraged—but you should always write up your solutions completely on your own, from start to finish, and you should understand all aspects of them fully. It is not allowed to compose draft solutions together and then continue editing individually, or to share any text, formulas, or pseudocode. Also, no such material may be downloaded from the internet and/or used verbatim. Submitted solutions will be checked for plagiarism.
Grading: A score of 120 points is guaranteed to be enough to pass this problem set.
Questions: Please do not hesitate to ask the instructor or TAs if any problem statement is unclear, but please make sure to send private messages — sometimes specific enough questions could give away the solution to your fellow students, and we want all of you to benefit from working on, and learning from, the problems. Good luck!

1 (60 p) The purpose of this problem is to gain an understanding of strongly connected components in directed graphs.

1a Compute the strongly connected components of the graph in Figure 1 by making a dry-run of the algorithm in CLRS (which is also discussed in the part of the lecture notes that we did not have time to discuss in class). Make sure to explain carefully the different steps in the algorithm execution, including in which order vertices are dealt with during graph traversal and why, and also what the final output is.

We assume that 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) sorted in this order.)

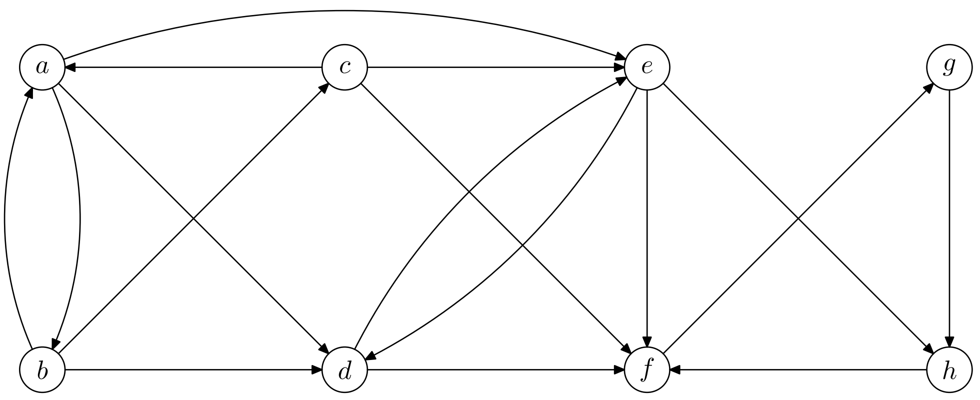

Figure 1: Directed graph for which to compute strongly connected components in Problem 1a.

1b In order to get a deeper understanding of operations on Boolean matrices, Jakob has performed some fairly extensive experiments on adjacency matrices A_G for small directed graphs G. He now claims to have made the discovery that if he computes

$$\bigvee_{i=1}^{\infty} (A_G)^i_{\odot} = A_G \vee (A_G \odot A_G) \vee (A_G \odot A_G \odot A_G) \vee (A_G \odot A_G \odot A_G \odot A_G) \vee \ldots \ ,$$

then it holds (possibly after reordering the vertices in G, corresponding to swapping rows and columns in A_G) that this matrix can be written on the form

$$\bigvee_{i=1}^{\infty} (A_G)^i_{\odot} = \begin{pmatrix} M_{1,1} & M_{1,2} & \cdots & M_{1,s} \\ M_{2,1} & M_{2,2} & \cdots & M_{2,s} \\ \vdots & \vdots & \ddots & \vdots \\ M_{s,1} & M_{s,2} & \cdots & M_{s,s} \end{pmatrix}$$

where the submatrices $M_{i,j}$ provide information about the strongly connected components of G in the following sense. If G has s strongly connnected components C_1, C_2, $\ldots$, C_s of sizes n_1, n_2, $\ldots$, n_s, respectively, then:

- Each matrix $M_{i,j}$ has dimensions $n_i \times n_j$.
- If there is a path from some $u \in C_i$ to some $v \in C_j$ in G, then $M_{i,j}$ contains 1s everywhere. (In particular, all matrices $M_{i,i}$ on the diagonal contain only 1s.)
- If there is no path from any $u \in C_i$ to any $v \in C_j$ in G, then $M_{i,j}$ contains 0s everywhere.

Sadly, Jakob is completely unable to explain this amazing fact, and he also cannot determine any upper bound on the time complexity of computing $\bigvee_{i=1}^{\infty} (A_G)^i_{\odot}$.

Is Jakob right about his claim? If so, present a concise and clear explanation to help Jakob see why this is so. If he is wrong, give a simple, concrete counter-example. Also, regardless of whether Jakob is correct or not, can you provide an efficient algorithm for computing $\bigvee_{i=1}^{\infty} (A_G)^i_{\odot}$ (for the plain adjacency matrix A_G without any row or column swaps) together with as tight an upper bound as possible for the time complexity?

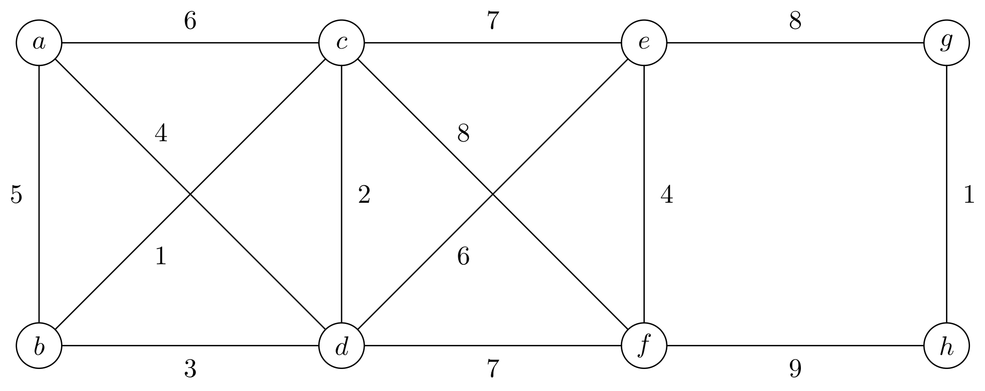

Figure 2: Undirected graph for which to compute minimum spanning tree in Problem 2a.

2 (80 p) The purpose of this problem is to deepen our understanding of minimum spanning trees in undirected graphs.

2a Generate a minimum spanning tree by running Kruskal's algorithm by hand on the graph in Figure 2. Assume that edges of the same weight are sorted in lexicographic order (so that for three hypothetical edges (u, v), (u, w), and (v, w) of the same weight, we would have (u, v) coming before (u, w), which would in turn come before (v, w)).

Describe how the forest changes at each step (but you do not need to describe in detail how the set operations are implemented). Also show the final tree produced by the algorithm.

2b Suppose that some vertex v with several neighbours in a graph G has a unique neighbour u such that the edge (u, v) has strictly smaller weight than any other edge incident to v. Is it true that the edge (u, v) must be included in any minimum spanning tree? Prove this or give a simple counter-example.

2c Suppose that some vertex v with several neighbours in a graph G has a unique neighbour u such that the edge (u, v) has strictly larger weight than any other edge incident to v. Is it true that the edge (u, v) can never be included in any minimum spanning tree? Prove this or give a simple counter-example.

2d Suppose that T is a minimum spanning tree for a weighted, undirected graph G. Modify G by adding some constant $c \in \mathbb{R}^+$ to all edge weights. Is T still a minimum spanning tree? Prove this or give a simple counter-example.

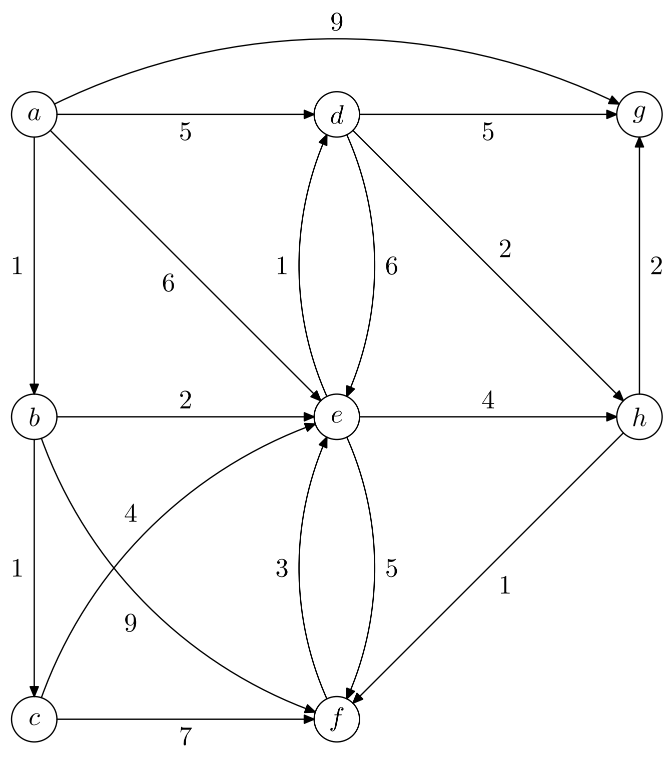

Figure 3: Directed graph for Dijkstra's algorithm in Problem 3a.

3 (100 p) Assume that we are given the directed graph in Figure 3. 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 this order.)

3a 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.

3b Consider the directed tree of shortest paths T produced in Problem 3a. 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.

3c Consider the directed tree of shortest paths T produced in Problem 3a. 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.

3d 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.