KØBENHAVNS UNIVERSITET

Diskret Matematik og Formelle Sprog: Problem Set 5

Due: Thursday March 30 at 23:59 CET .
Submission: Please submit your solutions via *Absalon* as a 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 or generated via the internet to be used in draft or final solutions. 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 (70 p) In this problem we wish to understand Dijkstra's algorithm.

1a (40 p) Assume that we are given the directed graph D in Figure 1. 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 done in Jakob's notes and video lectures, 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.

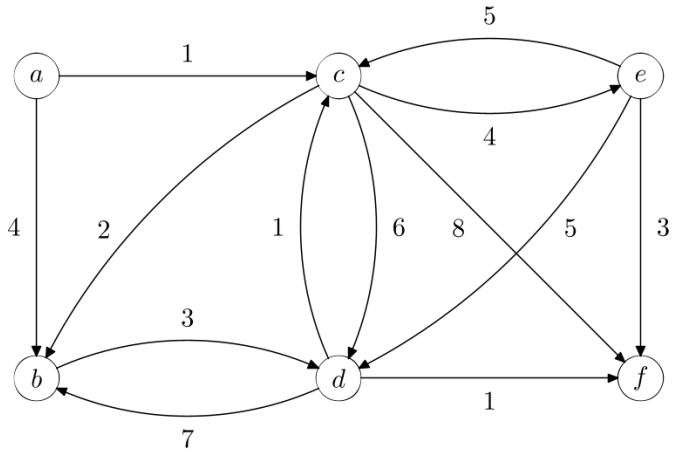

Figure 1: Directed graph D on which to run Dijkstra's algorithm in Problem 1a.

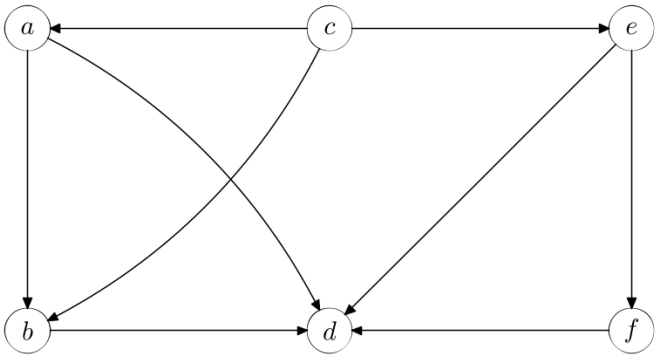

Figure 2: Directed graph H for which to compute a topological ordering in Problem 2a.

1b (30 p) 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.

2 (90 p) In this problem we wish to understand how to compute topological orderings and/or strongly connected components of directed graphs.

2a (30 p) Consider the graph H in Figure 2. 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 this is done in Jakob's lecture notes (including which recursive calls are made). Report what the resulting topological ordering is.

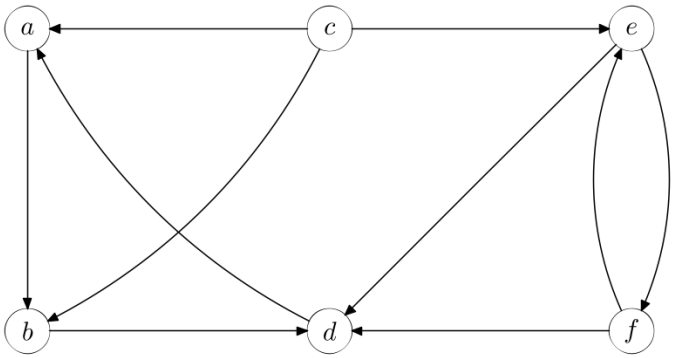

Figure 3: Directed graph G for which to compute contractions in Problem 2b.

2b (10 p) 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 3 on page 3 and for $U_1 = \{a, b, d\}$ and also explaining how you constructed the graph $\mathsf{contract}(G, U_1)$. Also, draw $\mathsf{contract}(G, U_2)$ for the same graph G in Figure 3 and $U_2 = \{e, f\}$ and explain how you constructed this graph.

2c (50 p) 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 3 and report what it does for that particular graph.)

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?

3 (60 p) On the final problem set, Jakob wanted to add a couple of examples of the power of inductive proofs. This did not go so well, however, since Jakob's proofs turned out to be a little bit too strong. Spceifically, Jakob was able to use mathematical induction to show

1. that all swans have the same colour (presumably white, so that there are no black swans after all), and
2. that all positive integers are in fact equal.

Both of these claims are fairly disturbing from a mathematical point of view. Please help Jakob by pointing out clearly what goes wrong in his induction proofs below.

3a **Theorem.** All swans have the same colour.

Proof. We prove by induction over n that any set of n swans have the same colour.

For the base case, if we have a set of $n = 1$ swan, then this swan vacuously has the same colour as itself.

For the induction step, assume as our induction hypothesis that all sets of n swans have the same colour, and consider a set of $n + 1$ swans. Fix some swan S_1 in this set. If we remove S_1, then we have n swans left, and by the induction hypothesis they all have the same colour. Let C be this colour.

Now consider another swan S_2 in the set. If we remove S_2 from our set of $n + 1$ swans instead of S_1, then we again have n swans left, and they all have the same colour by the induction hypothesis. Since S_1 is one of the swans in this set, it must have the same colour C as all the others. Hence, all $n + 1$ swans have the same colour.

It follows by the principle of mathematical induction that any set of n swans must always have the same colour.

3b **Theorem.** All positive integers are equal.

Proof. By induction over n.

For the base case, the positive integer 1 is equal to itself.

For the induction step, assume as our induction hypothesis that $n - 1 = n$. Adding 1 to both sides of this equality, we derive that $n = n + 1$.

It follows from this by the principle of mathematical induction that for all integers n the equality $n = n + 1$ holds. But then by transitivity we obtain that all integers are equal, and the claim in the theorem statement has thus been established.