KØBENHAVNS UNIVERSITET

Diskret Matematik og Formelle Sprog: Problem Set 4

Due: Sunday March 21 at 23:59 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. *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 in the course information 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 pseudo-code. 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 160 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 on the problems. Good luck!

1 (80 p) Consider the graph in Figure 1. We assume that this graph is represented in adjacency list format, with the neighbours in each adjacency list sorted in lexicographic order (i.e., in the order $a, b, c, d, \ldots$), so this is the order in which vertices are encountered when going through neighbour lists.

1a (30 p) Run the code for depth-first search by hand on this graph, starting in the vertex a. Describe in every recursive call which neighbours are being considered and how they are dealt with. Show the spanning tree produced by the search.

1b (30 p) Run the code for breadth-first search by hand on this graph, also starting in the vertex a. Describe for every vertex which neighbours are being considered and how they are dealt with, and how the queue changes. Show the spanning tree produced by the search.

1c (20 p) Suppose that the graph in Figure 1 is modified to give every edge weight 1, and that we run Dijkstra's algorithm starting in vertex a. How does the tree computed by Dijkstra's algorithm compare to that produced by DFS? What about BFS?

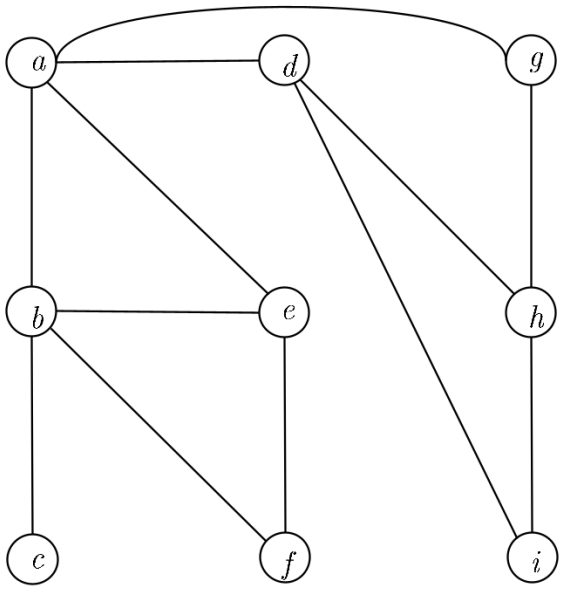

Figure 1: Undirected unweighted graph for graph traversals in Problem 1.

2 (110 p) Consider the graph in Figure 2. We assume that this graph is represented in adjacency list format, with the 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).

2a (30 p) Generate a minimum spanning tree by running Prim's algorithm by hand on this graph, starting in the vertex a. Use a heap for the priority queue implementation. Describe for every dequeueing operation which neighbours are being considered and how they are dealt with, and show how the heap changes. Also show the spanning tree produced by the algorithm.

2b (30 p) Generate a minimum spanning tree by running Kruskal's algorithm by hand on this graph. Assume that edges of the same weight are sorted in lexicographic order (so that we would have (a, b) coming before (a, d), which would in turn come before (b, a) for hypothetical edges of equal weight). 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 spanning tree produced by the algorithm.

2c (10 p) 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.

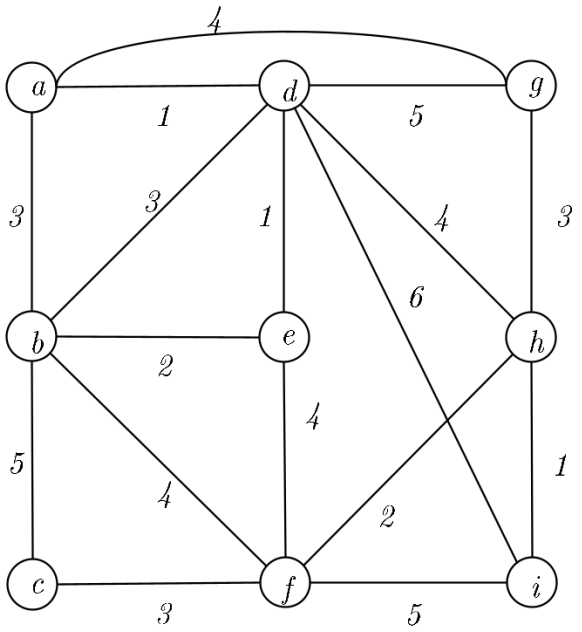

Figure 2: Undirected weighted graph for minimum spanning trees in Problem 2 and shortest paths in Problem 3.

2d (20 p) 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.

2e (20 p) 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.

3 (60 p) Consider again the graph in Figure 2 with the same assumptions.

3a (40 p) Run Dijkstra's algorithm by hand on this graph, starting in the vertex a. Use a heap for the priority queue implementation. Describe for every vertex which neighbours are being considered and how they are dealt with, and show how the heap changes. Also show the tree produced by the algorithm.

3b (20 p) If Dijkstra's algorithm is run on an undirected graph, is it true that the spanning tree produced is a minimum spanning tree? Prove this or give a simple counter-example.

4 (50 p) Let us return to our array $A = [5, 6, 4, 7, 3, 8, 2, 9, 1, 10]$ from problem set 2, which we still want to sort in increasing order.

4a (40 p) Run heap sort by hand on this array, and show in detail in every step of the algorithm what calls to the heap are made and how the array and the heap change.

4b (10 p) Suppose that we build a max-heap from any given array A. Is it true that once the array A has been converted to a correct max-heap, then considering the elements in reverse order (i.e., $A[n], A[n-1], \ldots, A[2], A[1]$) yields a correct min-heap? Prove this or give a simple counter-example.