Introduktion til diskret matematik og algoritmer: Problem Set 4

Due: Wednesday April 3 at 17:15 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. Never, ever just state the answer, but always 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!

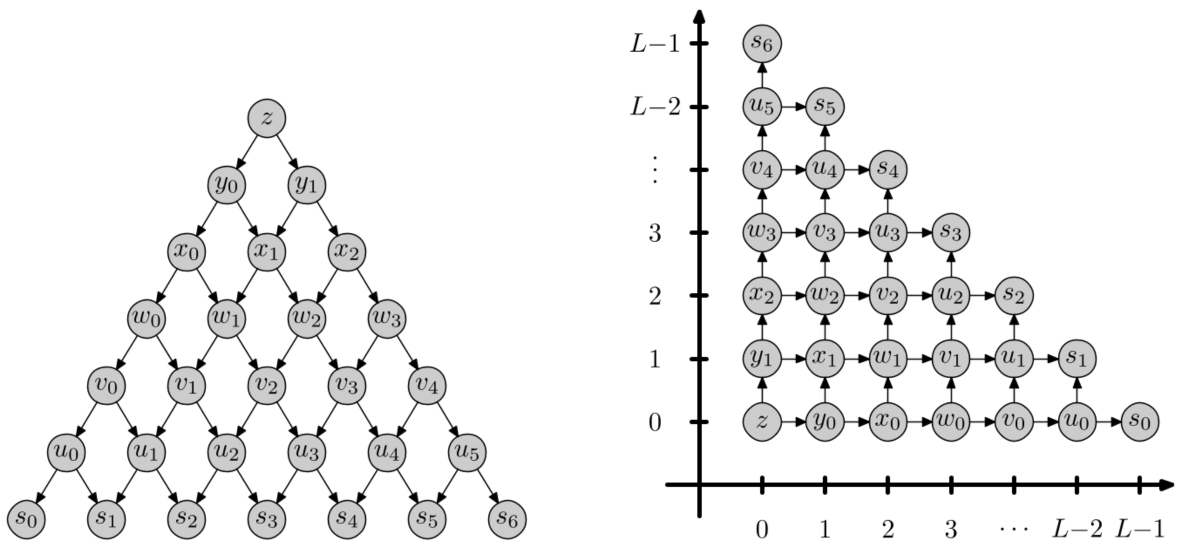
1 (100 p) Consider a directed graph that consists of L layers numbered from 0 up to $L-1$, with $i+1$ vertices in every layer i numbered from 0 to i, and with outgoing edges from vertex j in layer i to vertices j and $j+1$ in layer $i+1$. See Figure 1a for an illustration of this graph with 7 layers. We can agree to call this graph a *lattice graph* (because it can be obtained from a fragment of the integer lattice in 2 dimensions as shown in Figure 1b, but this is actually completely irrelevant to this problem).

Suppose we start in the unique source vertex on level 0 and walk along edges in the graph, flipping a fair coin at every vertex to decide whether to go left or right, until we reach one of the sinks in the last layer. For instance, in Figure 1a the walk "left–left–right–left–right–right" would visit vertices z, y_0, x_0, w_1, v_1, u_2, and end in s_3.

1a For every vertex s_i in the lattice graph in Figure 1a, calculate the probability that such a walk ends in vertex s_i. Which vertex is the walk most likely to end up in?

1b For a lattice graph with L layers, where $L \geq 2$ is a positive integer, calculate the probability that a walk as described above ends in vertex j in layer $L-1$ for $j = 0, 1, \ldots, L-1$.

Hint: Use the fact that all walks are equally likely to turn this into a counting problem.



(a) Lattice graph with 7 layers. (b) Graph as connected points in 2D integer lattice.

Figure 1: Example graph for Problem 1.

2 (80 p) In this problem we wish to understand algorithms for minimum spanning trees.

2a Consider the graph G in Figure 2, which is given to us in adjacency list representation, with the neighbour lists sorted so that vertices are encountered in lexicographic order. (For instance, the neighbour list of c is (a, b, d, e) sorted in that order.)

Run Prim's algorithm on the graph G, starting with the vertex a. Show the minimum spanning 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.

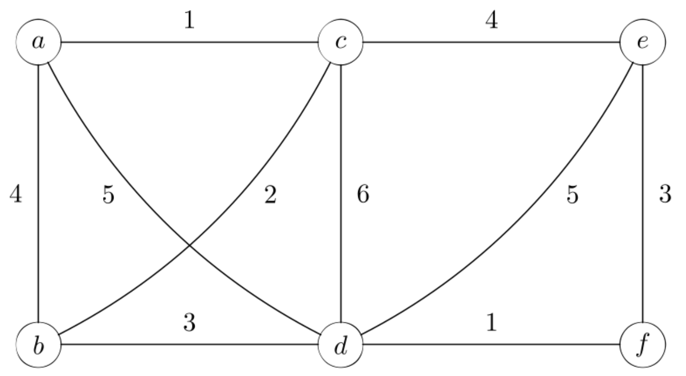


Figure 2: Graph G on which to run Prim's algorithm in Problem 2a.

2b Jakob has designed a fairly nifty MST algorithm that works as follows for $G = (V, E)$:

1. Initialize $T = \emptyset$ and $S = \{v\}$, where v is a randomly chosen vertex of the graph.
2. Iterate $|V| - 1$ times:
   (a) Let w be the vertex most recently added to S.
   (b) Among all edges from w to vertices x not yet added to S, pick the edge (w, x) with lowest cost and add to T, and add x to S with predecessor w.
   (c) If w does not have any neighbours not already in S, go back to the predecessor of w, and then to the predecessor of the predecessor, et cetera, until you find a vertex z that has at least one neighbor not already in S. Use that vertex z instead of w in this iteration.

Jakob claims that T as computed by this algorithm is a minimum spanning tree, and that the algorithm is also faster than Prim's algorithm since it runs in time $O(|V| + |E|)$. Determine whether Jakob's algorithm is valid by giving a proof of correctness or a counter-example. Regardless of whether the algorithm is correct or not, did Jakob get the time complexity analysis right? Motivate your answer clearly.

3 (80 p) Consider the graph in Figure 3. 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.

3a 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.

3b 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.

3c Suppose that the graph in Figure 3 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?

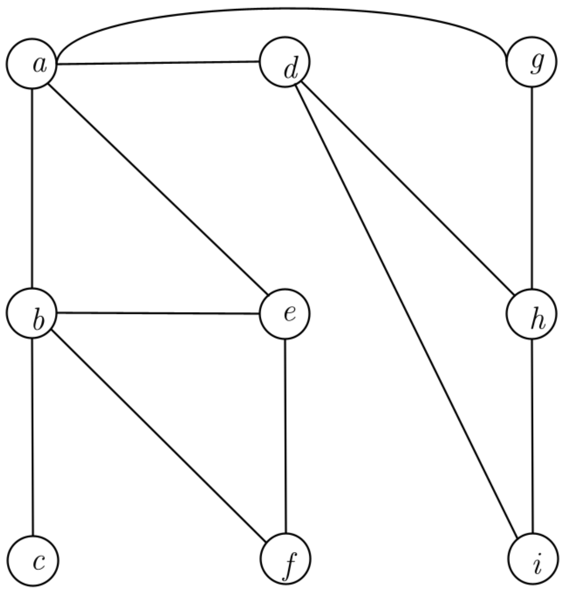


Figure 3: Undirected unweighted graph for graph traversals in Problem 3.