Example sheet 4

Graphs and path finding

Algorithms—DJW[*]—2021/2022

Questions labelled ∘ are warmup questions. Questions labelled ∗ are more challenging, and optional.

Question 1°. Read the definitions of *directed acyclic graph* and *tree* from lecture notes. Draw an example of each of the following, or explain why no example exists:
(i) A directed acyclic graph with 8 vertices and 10 edges
(ii) An undirected tree with 8 vertices and 10 edges
(iii) A graph without cycles that is *not* a tree

Question 2°. Give pseudocode for a function `dfs_recurse_path(`g, s, t`)` based on `dfs_recurse`, that returns a path from s to t.

Question 3. Modify your function from question 2 so that it does not visit any further vertices once it has reached the destination vertex t.

Question 4. Do `dfs` and `dfs_recurse` (as given in lecture notes) always visit vertices in the same order? Either prove they do, or give an example of a graph where they do not. You may assume that there is an ordering on vertices, and that v`.neighbours` returns a sorted list of v's neighbouring vertices.

If they do not, then modify `dfs` so they do. Give pseudocode.

Question 5. Consider a directed graph in which every vertex v is labelled with a real number x_v. For each vertex v, define m_v to be the minimum value of x_u among all vertices u such that either $u = v$ or u is reachable via some path from v. Give an $O(E + V \log V)$-time algorithm that computes m_v for all vertices v.

[Extra challenge, going beyond what's taught in this course: give an $O(V + E)$ algorithm.]

Question 6. Modify `bfs_path(`g, s, t`)` to find *all* shortest paths from s to t. *[This is an optional tick. See the Ticks tab on the course website, item `ex4-bfs`, for submission details.]*

Question 7*. Breadth-first search (as presented in lecture notes) is supposed to pop vertices in order of distance from the start vertex. Formally, if $v_{(i)}$ is the ith vertex it pops, we want $i \le j \Rightarrow d(v_{(i)}) \le d(v_{(j)})$, where $d(v)$ is the distance to v. Prove rigorously that this is the case. *[Like many proofs of correctness, this relies on induction. The trick is finding the right inductive hypothesis.]*

Question 8 (FS)°. In a directed graph with edge weights, give a formal proof of the triangle inequality

$$\text{distance}(u \text{ to } v) \le \text{distance}(u \text{ to } w) + \text{cost}(w \to v) \quad \text{for all vertices } u, v, w \text{ with } w \to v.$$

Make sure your proof covers the cases where no path exists.

Question 9. There is a missing step in the proof of Dijkstra's algorithm, as given in lecture notes. The proof looks at the state of program execution at the point in time at which some vertex v fails the assertion on line 9. Call this time t. It uses the equation

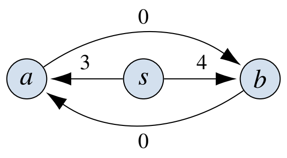
$$u_{i-1}\texttt{.distance} = \text{distance}(s \text{ to } u_{i-1}),$$

and justifies it by saying "the assertion on line 9 didn't fail when u_{i-1} was popped." Let t' be the time when u_{i-1} was popped, and explain carefully why u_{i-1}`.distance` cannot change between t' and t.

[*]Questions labelled FS are from Prof Stajano. Questions labelled CLRS are from *Introduction to Algorithms, 3rd ed.* by Cormen, Leiserson, Rivest and Stein.

Question 10. Suppose we have implemented Dijkstra's algorithm using a priority queue for which `push` and `decreaskey` have running time $\Theta(1)$, and `popmin` has running time $\Theta(\log n)$ where n is the number of items in the queue. Construct a sequence of graphs indexed by k, where the kth graph has $|V| = k$ and $|E| = \Theta(k^\alpha)$, such that Dijkstra's algorithm has running time $\Omega(k^\alpha + k\log k)$. Here α is a constant, $1 \le \alpha \le 2$.

Question 11 (CLRS-24.3-4). A contractor has written a program that she claims solves the shortest path problem, on directed graphs with edge weights ≥ 0. The program produces v.`distance` and v.`come_from` for every vertex v in a graph, reporting distances and paths from a given start vertex s. Give a $O(V + E)$-time algorithm to test whether or not the output of the contractor's program is correct.

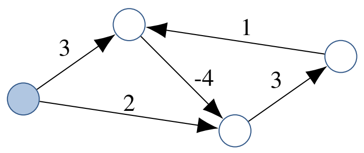


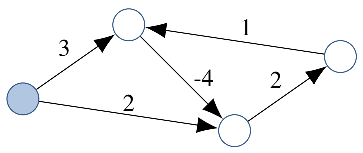
As a test of your algorithm, what will its output be for the graph above, when the contractor's program produces `s.come_from=None`, `a.come_from=b`, `b.come_from=a`, `s.distance=0`, `a.distance=1`, `b.distance=1`?

Question 12. We are given a directed graph where each edge is labelled with a weight, and where the vertices are numbered $1,\dots,n$. Assume it contains no negative weight cycles. Define $F_{ij}(k)$ to be the minimum weight path from i to j, such that every intermediate vertex is in the set $\{1,\dots,k\}$. Give a dynamic programming equation for $F_{ij}(k)$, and a suitable definition for $F_{ij}(0)$.

Question 13°. By hand, run both Dijkstra's algorithm and the Bellman-Ford algorithm on each of the graphs below, starting from the shaded vertex. The labels indicate edge costs, and one is negative. Does Dijkstra's algorithm correctly compute minimum weights?

[Run `dijkstra` *as described in lectures, which loops until* `toexplore` *is empty. Some textbooks use different versions of Dijkstra's algorithm, that terminate once a destination vertex has been popped, or that don't allow popped vertices to re-enter* `toexplore`*.]*





Question 14. In the course of running the Bellman-Ford algorithm, is the following assertion true? "Pick some vertex v, and consider the first time at which the algorithm reaches line 7 with v`.minweight` correct i.e. equal to the true minimum weight from the start vertex to v. After one subsequent pass of relaxing all the edges, u`.minweight` is correct for all $u \in \text{neighbours}(v)$."

If it is true, prove it. If not, provide a counterexample.

Question 15. Consider running Dijkstra's algorithm on a graph which may have negative edge weights, but which has no negative-weight cycle. Assuming it terminates, prove that the minweights it computes are correct. *[Hint. Use a similar argument to the Bellman-Ford proof.]*

Question 16 (CLRS-25.3-4)°. An engineer friend tells you there is a simpler way to reweight edges than the method used in Johnson's algorithm. Let w^* be the minimum weight of all edges in the graph, and just define $w'(u \to v) = w(u \to v) - w^*$ for all edges $u \to v$. What is wrong with your friend's idea?

Question 17*. Consider running Dijkstra's algorithm as in question 15. Show that its asymptotic worst-case running time is worse than that of Bellman-Ford. *[You should pick some $\alpha \in [1,2]$, and construct a sequence of graphs as in question 10, and show that the running time on your graphs is asymptotically larger than $VE = k^{1+\alpha}$. There is in fact a construction in which the running time grows exponentially in k.]*

Question 18*. Consider a graph without edge weights, and write $d(u, v)$ for the length of the shortest path from u to v. The *diameter* of the graph is defined to be $\max_{u,v\in V} d(u, v)$. Give an efficient algorithm to compute the diameter of an undirected tree, and analyse its running time. *[Hint. One way to solve this is by using breadth-first search, twice.]*

Question 19*. The Bellman-Ford code given in lecture notes will report "Negative cycle detected" if there is a negative-weight cycle reachable from the start vertex. Modify the code so that, in such cases, it returns a negative-weight cycle, rather than just reporting that one exists.