Example sheet 5

Graphs. Paths.
Algorithms—DJW*–2017/2018

Questions labelled $\circ$ are warmup questions. Questions labelled $*$ involve more thinking and you are not expected to tackle all of them. The questions are arranged in the order that the course is taught; consult your supervisor to find out which questions to answer for which supervision.

Question 1°. 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*. You've learned what $O(n)$ means when there's a single variable n that increases. How would you define $O(E+V\log V)$?

Question 3. Modify `bfs_path` to find *all* shortest paths from s to t.

Question 4*. 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. Use breadth-first search.]*

Question 5. Do the algorithms `dfs` and `dfs_recurse` (as given in the handout) always visit vertices in the same order? If not, modify `dfs` so that they do. Give pseudocode. **[Corrected on 2017-02-17. The original said 'modify `dfs_stack`'.]**

Question 6. Give pseudocode for a function `dfs_recurse_path(g,s,t)` based on `dfs_recurse`, that returns a shortest path from s to t. Modify your function so that it does not visit any further vertices once it has reached the destination vertex t.

Question 7. 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.

Question 8*. Give a $O(V+E)$ algorithm that solves Question 7. *[This goes beyond what is taught in lectures. Credit to Georgiev (matriculated 2016) for spotting this.]*

Question 9. There is a missing step in the proof of Dijkstra's algorithm, as given in the handout. 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 $u_{i-1}\texttt{.distance}$ cannot change between t' and t.

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

Question 10*. Consider a directed graph with edge weights that are ≥ 0, and suppose we want to find a shortest path from some start vertex s to a destination t. Let $h(v)$ be the distance from s to v for any vertex v. Write down a recursion from $h(v)$, and give pseudocode for a dynamic programming algorithm that uses recursion and memoization to compute $h(t)$. Comment on the relationship between this and Dijkstra's algorithm. **[Corrected on 2017-02-18. The original said 'uses recursion and memoization to compute $h(s)$'.]**

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. Give an $O(V+E)$-time algorithm to check the output of the contractor's program.

Question 12°. 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?

Question 13 (FS57)°. In a directed graph with edge weights, give a formal proof of the triangle inequality

$$d(u,v) \leq d(u,w) + c(w \to v) \quad \text{for all vertices } u, v, w \text{ with } w \to v$$

where $d(u,v)$ is the minimum weight of all paths from u to v (or ∞ if there are no such paths) and $c(w \to v)$ is the weight of edge $w \to v$. Make sure your proof covers the cases where no path exists.

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 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. Without looking at the handout, try to prove that the Bellman-Ford algorithm is correct. (The case without negative-weight cycles is easier.)

Question 16*. The Bellman-Ford code given in the handout 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.

Question 17 (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 18 (FS58, FS59). In Section 5.8, we defined $M_{ij}^{(n)}$ to be the minimum weight among all paths from i to j that have n or fewer edges, and we derived the recursion

$$M^{(1)} = W, \qquad M^{(n)} = M^{(n-1)} \otimes W$$

where $W_{ij} = 0$ if $i = j$, $W_{ij} = \text{weight}(i \to j)$ if there is an edge $i \to j$, and $w_{ij} = \infty$ otherwise. How can we define $M^{(0)}$ so that $M^{(1)} = M^{(0)} \otimes W$? What is the relationship between $M^{(0)}$ and the identity matrix I used in standard matrix multiplication?