Week 10 Tutorial Sheet

(To be completed during the Week 10 tutorial class)

Objectives: The tutorials, in general, give practice in problem solving, in analysis of algorithms and data structures, and in mathematics and logic useful in the above.

Instructions to the class: Aim to attempt these questions before the tutorial! It will probably not be possible to cover all questions unless the class has prepared them in advance. There are marks allocated towards active participation during the class. You **must** attempt the problems under **Assessed Preparation** section **before** your tutorial class and give your worked out solutions to your tutor at the start of the class – this is a hurdle and failing to attempt these problems before your tutorial will result in 0 mark for that class even if you actively participate in the class.

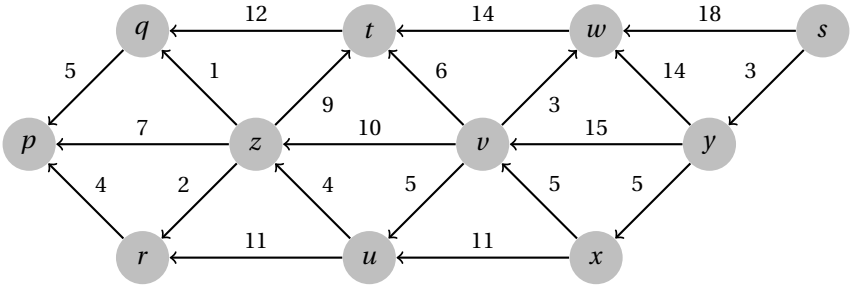
Instructions to Tutors:

1. The purpose of the tutorials is not to solve the practical exercises!
2. The purpose is to check answers, and to discuss particular sticking points, not to simply make answers available.

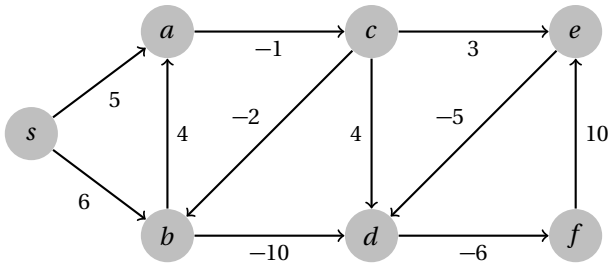
Supplementary problems: The supplementary problems provide additional practice for you to complete after your tutorial class, or as pre-exam revision. Problems that are marked as **(Advanced)** difficulty are beyond the difficulty that you would be expected to complete in the exam, but are nonetheless useful practice problems as they will teach you skills and concepts that you can apply to other problems.

Assessed Preparation

Problem 1. Use Dijkstra's algorithm to determine the shortest paths from vertex s to all other vertices in this graph. You should clearly indicate the order in which the vertices are visited by the algorithm, the resulting distances, and the shortest path tree produced.



Problem 2. Use Bellman-Ford to determine the shortest paths from vertex s to all other vertices in this graph. Afterwards, indicate to which vertices s has a well defined shortest path, and which do not by indicating the distance as $-\infty$. Draw the resulting shortest path tree containing the vertices with well defined shortest paths. For consistency, you should relax the edges in the following order: $s \to a$, $s \to b$, $a \to c$, $b \to a$, $b \to d$, $c \to b$, $c \to d$, $c \to e$, $d \to f$, $e \to d$ and $f \to e$.



Tutorial Problems

Problem 3. Give an example of a graph with negative weights on which Dijkstra's algorithm produces the wrong answer.

Problem 4. Consider the following algorithm for single-source shortest paths on a graph with negative weights:

- Find the minimum weight edge in the graph, say it has weight w
- Subtract w from the weight of every edge in the graph. The graph now has no negative weights
- Run Dijkstra's algorithm on the modified graph
- Add w back to the weight of the edges and compute the lengths of the resulting shortest paths

Prove by giving a counterexample that this algorithm is incorrect.

Problem 5. Suppose that we wish to solve the single-source shortest path problem for graphs with nonnegative bounded edge weights, i.e. we have $w(u,v) \leq c$ for some constant c. Explain how we can modify Dijkstra's algorithm to run in $O(V+E)$ time in this case. [Hint: Improve the priority queue]

Problem 6. Describe an algorithm that given a graph G determines whether or not G contains a negative cycle. Your algorithm should run in $O(VE)$ time.

Problem 7. Improve the Floyd-Warshall algorithm so that you can also reconstruct the shortest paths in addition to the distances. Your improvement should not worsen the time complexity of Floyd-Warshall, and you should be able to reconstruct a path of length k in $O(k)$ time.

Problem 8. Consider the problem of planning a cross-country road trip. You have a map of Australia consisting of the locations of towns, each of which has a petrol station, with the corresponding petrol prices (which may be different at each town). You are currently at a particular town s and would like to travel to town t. Your car has a fuel capacity of C litres, and for each road on the map, you know the amount of petrol it will take to travel along it. Your tank can only contain non-negative integer amounts of petrol, and all roads cost an integer amount of petrol to travel along. You can not travel along a road if you do not have enough petrol to make it all the way. You may refuel at any petrol station whether your tank is empty or not (but only to integer values), and you are not required to fill your tank. Assuming that your tank is initially empty, describe an algorithm for determining the cheapest way to travel to city t. [Hint: You should model the problem as a shortest path problem and use Dijkstra's algorithm.]

Supplementary Problems

Problem 9. Implement Dijkstra's algorithm and test your code's correctness by solving the following Codeforces problem: `http://codeforces.com/problemset/problem/20/C`.

Problem 10. Implement Bellman-Ford and test your code's correctness by solving the following problem on UVA

Online Judge: `https://uva.onlinejudge.org/index.php?option=onlinejudge&page=show_problem&problem=499`.

Problem 11. Consider a buggy implementation of Dijkstra's algorithm in which the distance estimate to a vertex is only updated the first time the vertex is discovered, and not in any subsequent relaxations. Give a graph on which this bug will cause the algorithm to not produce the correct answer.

Problem 12. Given a graph that contains a negative weight cycle, give an algorithm to determine the vertices of one such cycle. Your algorithm should run in $O(VE)$ time.

Problem 13. Add a post-processing step to Floyd-Warshall that sets $\text{dist}[u][v] = -\infty$ if there is an arbitrarily short path between vertex u and vertex v, i.e. if u can travel to v via a negative weight cycle. Your post processing should run in $O(V^3)$ time or better, i.e. it should not worsen the complexity of the overall algorithm.

Problem 14. Arbitrage is the process of exploiting conversion rates between commodities to make a profit. Arbitrage may occur over many steps. For example, one could purchase US dollars, convert it into Great British Pounds, and then back into Australian dollars. If the prices were right, this could result in a profit. Given a list of currencies and the best conversion rate between each pair, devise an algorithm that determines whether arbitrage is possible, i.e. whether or not you could make a profit. Your algorithm should run in $O(n^3)$ where n is the number of currencies.

Problem 15. In a directed graph with positive weights, a *vital arc* for a pair of vertices s and t is an edge whose removal from the graph would cause the shortest distance between s and t to increase. A *most vital arc* for the pair s, t is a vital arc whose removal causes the shortest distance between s and t to increase the most. For each of the following statements, decide whether they are true or false, and give a short proof possibly using an example graph.

(a) A vital arc must belong to a shortest path between s and t

(b) A vital arc must belong to all shortest paths between s and t

(c) A most vital arc (u, v) must be an edge with maximum weight $w(u, v)$ in the graph

(d) A most vital arc (u, v) must be an edge with maximum weight $w(u, v)$ on some shortest $s - t$ path

(e) **(Advanced)** Multiple most vital arcs may exist for the pair s and t

Now, devise an algorithm for finding one!

(f) **(Advanced)** Given a directed graph G, and a pair of vertices s and t, devise an algorithm that runs in $O((V + E)\log(E))$ time for finding a most vital arc if one exists.

Problem 16. (Advanced) An improvement to the Bellman-Ford algorithm is the so-called "Shortest Paths Faster Algorithm", or SPFA. SPFA works like Bellman-Ford, relaxing edges until no more improvements are made, but instead of relaxing every edge $|V| - 1$ times, we maintain a queue of vertices that have been relaxed. At each iteration, we take an item from the queue, relax each of its outgoing edges, and add any vertices whose distances changed to the queue if they are not in it already. The queue initially contains the source vertex s.

(a) Reason why SPFA is correct if the graph contains no negative-weight cycles.

(b) The algorithm described above will cycle forever in the presence of a negative-weight cycle. Add a simple condition to the algorithm to fix this and detect the presence of negative cycle if encountered.

(c) What is the worst-case time complexity of SPFA?

Problem 17. (Advanced) In this problem, we will derive a fast algorithm for the all-pairs shortest path problem on sparse graphs with negative weights[1]. We will assume for simplicity that the graph has no negative cycles, but

[1] This algorithm is known as Johnson's algorithm

the techniques here can easily be adapted to handle them. The key ingredient of the algorithm is based on the idea of *potentials*.

(a) Let $d[v] : v \in V$ be the distances to each vertex $v \in V$ from an arbitrary source vertex. Prove that for each edge $(u, v) \in E$

$$d[u] - d[v] + w(u, v) \geq 0.$$

We define the quantity $p[u] = d[u]$ as the *potential* of the vertex u.

(b) Give a simple algorithm to compute a set of valid potentials. Potentials should not be infinity for any vertex.

Suppose now that we modify the weight of every edge (u, v) in the graph to

$$w'(u, v) = p[u] - p[v] + w(u, v)$$

(c) Prove that a shortest path in the modified graph was also a shortest path in the original, unmodified graph.

(d) Deduce that we can solve the all-pairs shortest path problem on sparse graphs with negative weights in $O(V^2 \log(V))$ time.