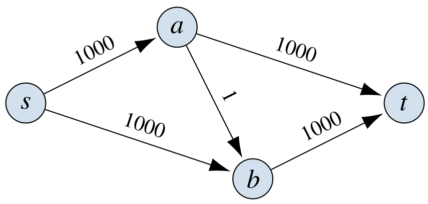
Example sheet 6

Flows. Subgraphs.

Algorithms—DJW*—2018/2019

Questions labelled ∘ are warmup questions. Questions labelled ∗ involve more thinking and you are not expected to tackle them all.

Question 1°. Use the Ford-Fulkerson algorithm, by hand, to find the maximum flow from s to t in the following graph. How many iterations did you take? What is the largest number of iterations it might take, with unfortunate choice of augmenting path?



Question 2°. Consider a flow f on a directed graph with source vertex s and sink vertex t. Let $f(u \to v)$ be the flow on edge $u \to v$, and set $f(u \to v) = 0$ if there is no such edge.

(i) < Show that

$$\sum_{v \neq s,t} \left(\sum_w f(v \to w) - \sum_u f(u \to v) \right) = 0.$$

(ii) The value of the flow is defined to be the net flow out of s,

$$\text{value}(f) = \sum_w f(s \to w) - \sum_u f(u \to s).$$

Prove that this is equal to the net flow into t. *[Hint. Add the left hand side of the equation from part (i).]*

Question 3. The code for `ford_fulkerson` as given in the handout has a bug: lines 27–39, which augment the flow, rely on an unstated assumption about the augmenting path. Give an example which makes the code fail. State the required assumption, and prove that the assertion on line 39 is correct, i.e. that after augmenting we still have a valid flow.

Question 4. The Russian mathematician A.N. Tolstoĭ introduced the following problem in 1930. Consider a directed graph with edge capacities, representing the rail network. There are three types of vertex: supplies, demands, and ordinary interconnection points. There is a single type of cargo we wish to carry. Each demand vertex v has a requirement $d_v > 0$. Each supply vertex v has a maximum amount it can produce $s_v > 0$. Tolstoĭ asked: can the demands be met, given the supplies and graph and capacities, and if so then what flow will achieve this?

Explain how to translate Tolstoĭ's problem into a max-flow problem of the sort we have studied.

Question 5. In the London tube system (including DLR and Overground), there are occasional signal failures that prevent travel in either direction between a pair of adjacent stations. We would like to know the minimum number of disruptions that will prevent travel between Kings Cross and Embankment.

*Questions labelled FS are from Dr Stajano.

(i) Translate the tube map into a suitable directed graph with capacities, such that a travel-disruption set of n disruptions is translated into a cut of capacity n. *[Hint. Remember that a cut is a partition of the vertices, not an arbitrary selection of edges.]*

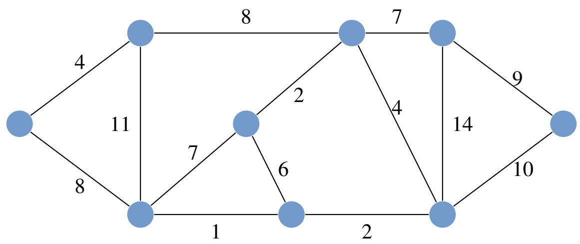
(ii) Show that a minimum cut corresponds to a minimal set of travel-preventing disruptions.

(iii) Find a maximum flow on your directed graph. Hence state the minimum number of disruptions that will prevent travel.

Question 6*. In the context of Question 4, a dispute has arisen in the central planning committee. Comrade A who oversees the factories insists that each demand vertex must receive precisely d_v, no more and no less. Comrade B who oversees the trains insists that each demand vertex v must be prepared to receive a surplus flow, more than d_v, so as not to constrain the flows on the train system any more than necessary. Does your solution satisfy Comrade A or Comrade B? How would you satisfy the other?

Question 7*. Devise an algorithm that takes as input a flow f on a network, and produces as output a decomposition $[(\lambda_1, p_1), \ldots, (\lambda_n, p_n)]$ where each p_i is a path from the source to the sink, and each λ_i is a positive number. The decomposition must satisfy $f = \sum_i \lambda_i p_i$, by which we mean "put flow λ_i along path p_i, and add together all these flows-along-paths, and the answer must be equal to f". Explain why your algorithm works.

Question 8 (FS55, FS56)°. Try to find, by hand, a minimum spanning tree for this graph. Now run, by hand, the Kruskal and Prim algorithms.



Question 9. An engineer friend tells you "Prim's algorithm is based on Dijkstra's algorithm, which requires edge weights to be ≥ 0. If some edge weights are < 0, we should first add some constant weight c to each edge so that all weights are ≥ 0, then run Prim's algorithm."

(i) Your friend's algorithm will produce a MST for the modified graph. Is this an MST for the original graph? *[Hint. Read step 3 in the analysis of Johnson's algorithm.]*

(ii) What would happen if you run Prim's algorithm on a graph where some weights are negative? Justify your answer.

Question 10. In a connected undirected graph with edge weights ≥ 0, let $u \leftrightarrow v$ be a minimum-weight edge. Show that $u \leftrightarrow v$ belongs to a minimum spanning tree.

Question 11°. Here are two buggy ways to code topological sort. For each, give an example to show why it's buggy.

(i) Pick some vertex s with no incoming edges. Simply run `dfs_recurse` from Section 5.2, starting at this node, and add an extra line `totalorder.prepend(v)` as we did in `toposort`.

(ii) Run `dfs_recurse_all`, but order nodes in order of when they are visited, i.e. remove `totalorder.prepend(v)` on line 17, and insert `totalorder.append(v)` immediately after the line that sets `v.visited=True`.

Question 12. Give pseudocode for an algorithm that takes as input an arbitrary directed graph g, and returns a boolean indicating whether or not g is a DAG.

Question 13*. The code for `toposort` is based on `dfs_recurse`. If we base it instead on the stack-based `dfs` from Section 5.2, and insert the line `totalorder.prepend(v)` on line 13 (after the iteration over `v`'s neighbours), would we obtain a total order? If so, justify your answer. If not, give a counterexample, and pseudocode for a proper stack-based `toposort`.