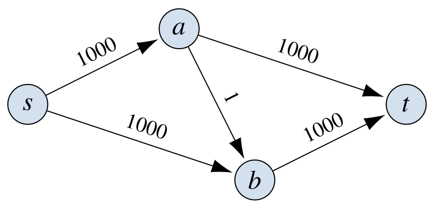
Example sheet 5

Flows and subgraphs

Algorithms—DJW*—2023/2024

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∘. We are given a directed graph, and a source vertex and a sink vertex. Each edge has a capacity $c_E(u \to v) \geq 0$, and each vertex (excluding the source and the sink) also has a capacity $c_V(v) \geq 0$. In addition to the usual flow constraints, we require that the total flow through a vertex be $\leq$ its capacity. We wish to find a maximum flow from source to sink.

Explain how to translate this problem into a max-flow problem of the sort we studied in section 6.2.

Question 5. 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 studied in section 6.2.

*Questions labelled FS are from Dr Stajano.

Question 6*. 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 such failures that will prevent travel between Kings Cross and Embankment.

(i) Explain how the tube map g can be translated into a suitable flow network g', with Kings Cross the source and Embankment the sink, such that a set of signal failures preventing travel in g is translated into a cut in g'. *[Hint. Remember that a cut is a partition of the vertices, not an arbitrary selection of edges.]*

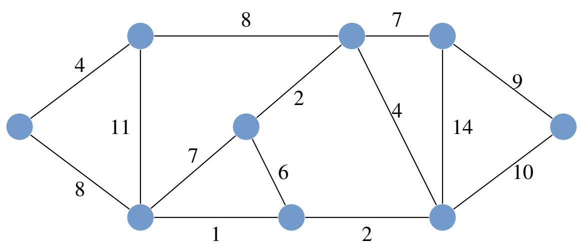
(ii) Explain how a cut in g' can be translated into a set of travel-preventing signal failures in g, such that the number of signal failures is $\leq$ the capacity of the cut.

(iii) Suppose we take the minimum cut in g' and translate it into a set of travel-preventing signal failures in g. Show that this is the minimum number of travel-preventing signal failures.

(iv) Find a maximum flow on your directed graph. Hence state the minimum number of signal failures that will prevent travel. *[Hint. You should run Ford–Fulkerson by hand, with sensibly-chosen augmenting paths.]*

Question 7*. In the context of Question 5, 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 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?

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

Question 10. Write out a formal proof of correctness of Prim's algorithm. You may use without proof the theorem stated in lecture notes: "Suppose we have a forest F and a cut C such that (i) C contains no edges of F, and (ii) there exists a MST containing F. If we add to F a min-weight edge among those that cross C, then the result is still part of a MST."

[Note: the proof of this theorem is not examinable, but the application to Prim's algorithm is examinable. Also note that the term cut *has two slightly different definitions, one for flow networks, one for spanning trees.]*

Question 11°. If we take a DAG and reverse all the edges, do we get a DAG? Justify your answer.

Question 12°. 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`, 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. insert `totalorder.append(v)` immediately after the line that sets `v.visited=True`.

Question 13. 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.