Red Scare! Report

Bjarke Brodin (bjal)
Bjørnar Haugstad Jåtten (bjja)
Helle Friis (hefr)
and Simon Boye Jørgensen (sboj)

November 16, 2022

Results

The following table gives my results for all graphs of at least 500 vertices.

Instance name	n	A	F	M	N	S
bht.txt	5757	false	-1	?!	6	?!
common-1-1000.txt	1000	false	-1	-1	-1	false
common-1-1500.txt	1500	false	-1	-1	-1	false
common-1-2000.txt	2000	false	-1	-1	-1	false
common-1-2500.txt	2500	false	1	?!	6	?!
common-1-3000.txt	3000	false	1	?!	6	?!
common-1-3500.txt	3500	false	1	?!	6	?!
common-1-4000.txt	4000	false	1	?!	6	?!
common-1-4500.txt	4500	true	1	?!	6	?!
common-1-500.txt	500	false	-1	-1	-1	false
common-1-5000.txt	5000	true	1	?!	6	?!
common-1-5757.txt	5757	true	1	?!	6	?!
common-2-1000.txt	1000	true	1	?!	4	?!
common-2-1500.txt	1500	true	1	?!	4	?!
common-2-2000.txt	2000	true	1	?!	4	?!
common-2-2500.txt	2500	true	1	?!	4	?!
common-2-3000.txt	3000	true	1	?!	4	?!
common-2-3500.txt	3500	true	1	?!	4	?!
common-2-4000.txt	4000	true	1	?!	4	?!
common-2-4500.txt	4500	true	1	?!	4	?!
common-2-500.txt	500	true	1	?!	4	?!
common-2-5000.txt	5000	true	1	?!	4	?!
common-2-5757.txt	5757	true	1	?!	4	?!

⋮

Instance name	n	A	F	M	N	S
gnm-1000-1500-0.txt	1000	false	1	?!	-1	?!
gnm-1000-1500-1.txt	1000	false	2	?!	-1	?!
gnm-1000-2000-0.txt	1000	false	-1	?!	7	?!
gnm-1000-2000-1.txt	1000	false	1	?!	-1	?!
gnm-2000-3000-0.txt	2000	false	-1	?!	8	?!
gnm-2000-3000-1.txt	2000	true	2	?!	-1	?!
gnm-2000-4000-0.txt	2000	false	-1	?!	6	?!
gnm-2000-4000-1.txt	2000	false	-1	?!	5	?!
gnm-3000-4500-0.txt	3000	false	-1	?!	10	?!
gnm-3000-4500-1.txt	3000	false	2	?!	-1	?!
gnm-3000-6000-0.txt	3000	false	-1	?!	6	?!
gnm-3000-6000-1.txt	3000	false	1	?!	6	?!
gnm-4000-6000-0.txt	4000	false	-1	?!	7	?!
gnm-4000-6000-1.txt	4000	false	-1	?!	15	?!
gnm-4000-8000-0.txt	4000	false	-1	?!	5	?!
gnm-4000-8000-1.txt	4000	true	1	?!	6	?!
gnm-5000-10000-0.txt	5000	false	1	?!	5	?!
gnm-5000-10000-1.txt	5000	true	-1	?!	5	?!
gnm-5000-7500-0.txt	5000	false	-1	-1	-1	false
gnm-5000-7500-1.txt	5000	false	-1	-1	-1	false
grid-25-0.txt	625	true	-1	?!	324	?!
grid-25-1.txt	625	true	-1	?!	123	?!
grid-25-2.txt	625	true	5	?!	-1	?!
grid-50-0.txt	2500	false	-1	?!	1249	?!
grid-50-1.txt	2500	false	-1	?!	521	?!
grid-50-2.txt	2500	false	10	?!	-1	?!
increase-n500-1.txt	500	true	1	15	1	true
increase-n500-2.txt	500	true	1	17	1	true
increase-n500-3.txt	500	true	1	16	1	true
rusty-1-2000.txt	2000	false	-1	-1	-1	false
rusty-1-2500.txt	2500	false	-1	-1	-1	false
rusty-1-3000.txt	3000	false	-1	?!	14	?!
rusty-1-3500.txt	3500	false	-1	?!	14	?!
rusty-1-4000.txt	4000	false	-1	?!	13	?!
rusty-1-4500.txt	4500	false	-1	?!	7	?!
rusty-1-5000.txt	5000	false	-1	?!	7	?!
rusty-1-5757.txt	5757	false	-1	?!	7	?!

⋮

Instance name	n	A	F	M	N	S
rusty-2-2000.txt	2000	false	-1	?!	5	?!
rusty-2-2500.txt	2500	false	-1	?!	4	?!
rusty-2-3000.txt	3000	false	-1	?!	4	?!
rusty-2-3500.txt	3500	false	-1	?!	4	?!
rusty-2-4000.txt	4000	false	-1	?!	4	?!
rusty-2-4500.txt	4500	false	-1	?!	4	?!
rusty-2-5000.txt	5000	false	-1	?!	4	?!
rusty-2-5757.txt	5757	false	-1	?!	4	?!
smallworld-30-0.txt	900	false	-1	?!	9	?!
smallworld-30-1.txt	900	true	-1	?!	11	?!
smallworld-40-0.txt	1600	false	-1	?!	8	?!
smallworld-40-1.txt	1600	true	-1	?!	13	?!
smallworld-50-0.txt	2500	false	-1	?!	3	?!
smallworld-50-1.txt	2500	true	1	?!	-1	?!
wall-n-100.txt	800	false	-1	?!	1	?!
wall-n-1000.txt	8000	false	-1	?!	1	?!
wall-p-100.txt	602	false	-1	?!	1	?!
wall-p-1000.txt	6002	false	-1	?!	1	?!
wall-p-10000.txt	60002	false	-1	?!	1	?!
wall-z-100.txt	701	false	-1	?!	1	?!
wall-z-1000.txt	7001	false	-1	?!	1	?!
wall-z-10000.txt	70001	false	-1	?!	1	?!

The columns are for the problems Alternate, Few, Many, None, and Some. The table entries either give the answer, or contain '?' for those cases where I was unable to find a solution within reasonable time. For those questions where there is a reason for my inability to find a good algorithm (because the problem is hard), I wrote '?!'.

For the complete table of all results, see the tab-separated text file `results.txt`.

Methods

For all solutions we process the input to produce a graph G, which we assume to be simple.

We preprocess each instance filtering out any irellevant components, and even return immediately if s and t are in different components. This is rather trivial to do in $O(|V| + |E| \log |E|)$ e.g. by using union-find.

None

We solve this problem efficiently for all graphs. Here we did not put any of the red vertex in the graph. Then when running the shortest path algorithm we were guaranteed to find the shortest path without any red vertexes in the path if there exists one.

We use BFS for the shortest path algorithm which has a running time of $O((|E| + |V|) \log |V|)$.

Alternate

To solve alternate, we first removed all reflective edges, meaning the edges that go from black to black or red to red. Afterwards, we applied a shortest path algorithm. If there exists a path between t and s, it must be alternating between black and red, as all the relfective edges are removed from the graph.

Running time is dominated by shortest path and is thus simply $O((|E| + |V|) \log |V|)$.

Few

To solve *few*, we realize that we can manipulate edge weights to allow a reduction to shortest path. We set weights to 1 for all arcs that are incident on a red vertex and all other weights to 0, then simply run an appropriate shortest path algorithm and return the distance. Running time is dominated by shortest path and is thus simply $O((|E| + |V|) \log |V|)$.

Many

We solve many for directed acyclic graphs, by using much the same approach as in few, only, instead of setting 1 weighted red arcs, we use -1 weighted red arcs. Running shortest path then yields the negation of our desired result. We take care to use a shortest path algorithm that does not stop early when reaching a valid path, but rather examines all options.

In the general case we cannot solve many, because the general case includes $R = G$. Solving many in this case implies solving the longest path problem exactly, and since this is known to be NP-hard we can safely give up.

Some

First we note that solving many implies solving some, since many $\geq$ 1 is equivalent to the result of some.

However we can utilize the maximum flow algorithm to solve some in polynomial time even when we may not be able to solve many. We construct a solution as follows:

- Ensure that all edges in the graph are bidirectional by filtering any that aren't, set all weights to 1.
- Ensure that each vertex v can only be used once, by splitting it into vertices v and $|V| + v$ such that all incoming edges go to v and all outgoing edges go from $|V| + v$. Then add a directional edge from v to $|V| + v$ with weight 1.
- Add a new source vertex s' and two new edges from s' to s and from s' to t both with weight 1.
- For each red vertex r compute maxflow(s', r) – if any flow is 2 then some must be 'true' (because every used original edge is bidirectional), if no such flow is found we can't however conclude that 'false' is the correct result, since we don't have sufficient information to make this claim.

In the general case the best solution we know of is enumerating all s-t paths, which is an exponential time algorithm, thus we give up in the general case.

References