Red Scare! Report

by Alice Cooper.

Results

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

Instance name	n	A	F	M	N	S
rusty-5762	5,762	true	16	–	?	5
wall-p-10000	10,000					
⋮						

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

All algorithm and graph implementations are from the JGraphT library[1].

[1] https://jgrapht.org/

For problem A, I solved each instance G by removing all edges that go between vertices of same color, unless it is $s - -t$. This new graph is then run on a shortest-path algorithm. In this case Breadth-First Search (BFS). The running time of this algorithm is $O(V + E)$, and my implementation spends $\cdots$ seconds on the instance $\cdots$ with $n = \cdots$.

For problem F, I solved each instance G by adding edge weights. Weight of 1 for edges into red vertices, and weight of 0 for all other edges. Then we run Dijkstra to get the minimum path. The running time of this algorithm is $\cdots$, and my implementation spends $\cdots$ seconds on the instance $\cdots$ with $n = \cdots$.

For problem N, I solved each instance G by removing all red vertices from the graph, and then feeding it to a shortest-path algorithm. The running time of this algorithm is $O(V + E)$, and my implementation spends $\cdots$ seconds on the instance $\cdots$ with $n = \cdots$.

We solved problem S for all undirected graphs by creating a flow, and using Ford-Fulkerson. The flow graph is created by adding a new source vertex, and attaching it to both s and t, each with capacity 1. A new target is also created, and attached to a red vertex with capacity 2. This is then done for every red vertex until a flow of size 2 is found.

We were unable to solve problem M except for the acyclic instances. This is because, in generality, this problem is NP-hard. To see this, consider the following reduction from Hamiltonian paths to Many. Let g be a graph that we must find a Hamiltonian path for, with vertices V and edges E. To reduce it so a many problem, we set $R = V$. If the result of many problem is n, then the graph has a Hamiltonian path if $n = |V|$. For acyclic graphs, the problem can be solved by adding weights to the edges and running Bellman-Ford on it. Every edge going into red vertices gets a weight of -1. Every other edge gets a weight of 0. The algorithm finds the shortest path, which will include as many red vertices as possible. This is because paths with as many reds as possible, negative edges, will result in the lowest negative path.

References

1. *APLgraphlib—A library for Basic Graph Algorithms in APL*, version 2.11, 2016, Iverson Project, `github.com/iverson/APLgraphlib`.[2]

2. A. Lovelace, *Algorithms and Data Structures in Pascal*, Addison–Wesley 1881.

[2] If you use references to code, books, or papers, be professional about it. Use whatever style you want, but be consistent.