Red Scare! Report

by Group N: Joshua James Medilo Calba, Mohtashim Haider, Jobayer Hossain, Miina Johanna Mäkinen, Samikshya Rana, and Courage Räsänen

Results

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

Table 1: Results for all graphs with $n \geq 500$

instance	n	A	F	M	N	S
bht.txt	5757	false	0	NP-HARD	6	true
common-1-1000.txt	1000	false	-1	NP-HARD	-1	false
common-1-1500.txt	1500	false	-1	NP-HARD	-1	false
common-1-2000.txt	2000	false	-1	NP-HARD	-1	false
common-1-2500.txt	2500	false	1	NP-HARD	6	true
common-1-3000.txt	3000	false	1	NP-HARD	6	true
common-1-3500.txt	3500	false	1	NP-HARD	6	true
common-1-4000.txt	4000	false	1	NP-HARD	6	true
common-1-4500.txt	4500	true	1	NP-HARD	6	true
common-1-500.txt	500	false	-1	NP-HARD	-1	false
common-1-5000.txt	5000	true	1	NP-HARD	6	true
common-1-5757.txt	5757	true	1	NP-HARD	6	true
common-2-1000.txt	1000	true	1	NP-HARD	4	true
common-2-1500.txt	1500	true	1	NP-HARD	4	true
common-2-2000.txt	2000	true	1	NP-HARD	4	true
common-2-2500.txt	2500	true	1	NP-HARD	4	true
common-2-3000.txt	3000	true	1	NP-HARD	4	true
common-2-3500.txt	3500	true	1	NP-HARD	4	true
common-2-4000.txt	4000	true	1	NP-HARD	4	true
common-2-4500.txt	4500	true	1	NP-HARD	4	true
common-2-500.txt	500	true	1	NP-HARD	4	true
common-2-5000.txt	5000	true	1	NP-HARD	4	true
common-2-5757.txt	5757	true	1	NP-HARD	4	true
gnm-1000-1500-0.txt	1000	false	1	NP-HARD	-1	true
gnm-1000-1500-1.txt	1000	false	2	NP-HARD	-1	true
gnm-1000-2000-0.txt	1000	false	0	NP-HARD	7	true
gnm-1000-2000-1.txt	1000	false	2	NP-HARD	-1	true
gnm-2000-3000-0.txt	2000	false	0	NP-HARD	8	true
gnm-2000-3000-1.txt	2000	true	2	NP-HARD	-1	true
gnm-2000-4000-0.txt	2000	false	0	NP-HARD	6	true

gnm-2000-4000-1.txt	2000	false	0	NP-HARD	5	true
gnm-3000-4500-0.txt	3000	false	0	NP-HARD	10	true
gnm-3000-4500-1.txt	3000	false	2	NP-HARD	-1	true
gnm-3000-6000-0.txt	3000	false	0	NP-HARD	6	true
gnm-3000-6000-1.txt	3000	false	2	NP-HARD	6	true
gnm-4000-6000-0.txt	4000	false	0	NP-HARD	7	true
gnm-4000-6000-1.txt	4000	false	1	NP-HARD	15	true
gnm-4000-8000-0.txt	4000	false	0	NP-HARD	5	true
gnm-4000-8000-1.txt	4000	true	2	NP-HARD	6	true
gnm-5000-10000-0.txt	5000	false	2	NP-HARD	5	true
gnm-5000-10000-1.txt	5000	true	1	NP-HARD	5	true
gnm-5000-7500-0.txt	5000	false	-1	NP-HARD	-1	false
gnm-5000-7500-1.txt	5000	false	-1	NP-HARD	-1	false
grid-25-0.txt	625	true	0	NP-HARD	324	true
grid-25-1.txt	625	true	0	NP-HARD	123	true
grid-25-2.txt	625	true	5	NP-HARD	-1	true
grid-50-0.txt	2500	false	0	NP-HARD	1249	true
grid-50-1.txt	2500	false	0	NP-HARD	521	true
grid-50-2.txt	2500	false	11	NP-HARD	-1	true
increase-n500-1.txt	500	true	2	16	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	NP-HARD	-1	false
rusty-1-2500.txt	2500	false	-1	NP-HARD	-1	false
rusty-1-3000.txt	3000	false	0	NP-HARD	14	true
rusty-1-3500.txt	3500	false	0	NP-HARD	14	true
rusty-1-4000.txt	4000	false	0	NP-HARD	13	true
rusty-1-4500.txt	4500	false	0	NP-HARD	7	true
rusty-1-5000.txt	5000	false	0	NP-HARD	7	true
rusty-1-5757.txt	5757	false	0	NP-HARD	7	true
rusty-2-2000.txt	2000	false	0	NP-HARD	5	true
rusty-2-2500.txt	2500	false	0	NP-HARD	4	true
rusty-2-3000.txt	3000	false	0	NP-HARD	4	true
rusty-2-3500.txt	3500	false	0	NP-HARD	4	true
rusty-2-4000.txt	4000	false	0	NP-HARD	4	true
rusty-2-4500.txt	4500	false	0	NP-HARD	4	true
rusty-2-5000.txt	5000	false	0	NP-HARD	4	true
rusty-2-5757.txt	5757	false	0	NP-HARD	4	true
smallworld-30-0.txt	900	false	0	NP-HARD	9	true
smallworld-30-1.txt	900	true	1	NP-HARD	11	true
smallworld-40-0.txt	1600	false	0	NP-HARD	8	true
smallworld-40-1.txt	1600	true	1	NP-HARD	13	true
smallworld-50-0.txt	2500	false	0	NP-HARD	3	true
smallworld-50-1.txt	2500	true	2	NP-HARD	-1	true

wall-n-100.txt	800	false	0	NP-HARD	1	true
wall-n-1000.txt	8000	false	0	NP-HARD	1	true
wall-n-10000.txt	80000	false	0	NP-HARD	1	true
wall-p-100.txt	602	false	0	NP-HARD	1	true
wall-p-1000.txt	6002	false	0	NP-HARD	1	true
wall-p-10000.txt	60002	false	0	NP-HARD	1	true
wall-z-100.txt	701	false	0	NP-HARD	1	true
wall-z-1000.txt	7001	false	0	NP-HARD	1	true
wall-z-10000.txt	70001	false	0	NP-HARD	1	true

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 we was unable to find a solution within reasonable time.

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

Methods

Problem: None

For the **None** problem, we solved each instance G by performing a Breadth-First Search (BFS) starting from vertex s. During the exploration of neighbors for a current vertex u (where u has not been visited), a neighbor v was added to the queue only if $v \notin R$ or if $v = t$. This condition ensures that while the start vertex s and the target vertex t are permitted to be red, no intermediate vertex on the path belongs to R. If the queue became empty without reaching t, the algorithm returned -1.

The running time of this algorithm is $O(n + m)$, as it visits each vertex and edge at most once.

Problem: Some

For the **Some** problem, we solved each instance G by performing up to two standard Breadth-First Search (BFS) traversals on the graph for each red vertex r: first to determine if there is a path from s to r, and second to determine if there is a path from r to t. If a red vertex r was found where both reachability checks returned true, the algorithm immediately returned "true".

The running time of this algorithm is $O(|R| \cdot (n + m))$, because in the worst-case scenario, two BFS traversals are performed for every red vertex in the graph.

Problem: Few

For the **Few** problem, we solved each instance G using Dijkstra's Algorithm. We assigned a traversal "cost" based on vertex color:

vertices in R (red) had a weight of 1, while vertices not in R had a weight of 0. The goal was to find a path that minimized the total vertex weight.

To implement this, we used a binary heap (min-priority queue) to efficiently explore the graph. The priority queue stored tuples of `(current_red_count, vertex)`, ensuring that paths with fewer red vertices were prioritized. The running time of this algorithm is $O(m \log n)$ due to the priority queue operations.

Problem: Alternate

For the **Alternate** problem, we use BFS starting from s over the graph's adjacency list. Each queue entry stores the current vertex u and it's color (red or non-red). For every neighbor v of u we enqueue v only if its color is not equal to the color of u. This ensures that consequtive vertices along the explored path alternate in color. If we reach t during BFS, we return true; if the queue empties without reaching t, we return false. The time complexity of this algorithm is $O(n + m)$, since we process each vertex and edge a constant number of times.

Problem: Many For problem *Many*, we solve each instance G by first classifying the graph. If the graph is a DAG, we compute the maximum number of red vertices along an s, t-path in polynomial time. If the graph is undirected and acyclic (a tree), we also compute the solution efficiently using a modified BFS. However, if the graph contains any kind of cycle (directed or undirected), or if edges are mixed ($\rightarrow$ and $--$), then the problem becomes intractable, and our program returns **"NP-HARD"**, indicating that we cannot solve it in polynomial time.

In the DAG case, we perform a Dynamic Programming traversal over a topological ordering. For every vertex, we store the maximum number of red vertices collected so far on a path from s. When processing neighbours, the red count is updated and propagated only if it improves the best value stored for that node. This ensures efficiency by avoiding unnecessary revisits.

In the undirected acyclic case (tree), we run a Breadth-First Search (BFS) starting from s, carrying forward the cumulative number of red nodes. Since trees have a unique simple path between any two vertices, the first successful reach of t gives the optimal red count.

Building the graph and detecting DAG/tree structure both require $O(V + E)$ time. The DP traversal on DAGs and BFS on trees also run in $O(V + E)$, as each edge and vertex is processed at most once.

NP-hardness reduction

Claim. *Many* is NP-hard.

Proof. We reduce from *Hamiltonian Path*, which is known to be NP-hard.

Let $G = (V, E)$ be an instance of Hamiltonian Path, where $V = \{v_1, \ldots, v_n\}$, $E = \{e_1 \ldots e_m\}$, $n = |V|$ and $m = |E|$. We construct an instance $G' = (V', E')$ of *Many* by letting $G' = G$, that is, $V' = V$ and $E' = E$. We define the red set as $R = V'$, i.e. all vertices of G' are marked red.

Since a Hamiltonian path is a simple path that visits all vertices in V exactly once, we want the *Many* instance to return the value n if and only if such a path exists.

We choose start and end vertices $s, t \in V'$ such that any Hamiltonian path in G becomes an $s \to t$ simple path in G'.

If G contains a Hamiltonian path p, then p visits every vertex of V, and since every vertex is red in the *Many* instance, the path visits all n red vertices. Therefore, *Many* returns the value n.

Conversely, if *Many* returns n, then there exists an $s \to t$ simple path p' in G' that visits all n red vertices. Because the vertices are distinct on a simple path, p' visits all vertices of V' exactly once. Thus p' is a Hamiltonian path in G'.

The construction clearly runs in polynomial time, since we only copy the vertex and edge sets and mark each vertex red. This is $O(n + m)$ time.

So we can conclude that *Hamiltonian Path* $\leq_p$ *Many* based on our reduction.

References

1. GeeksforGeeks, *Binary Heap – Data Structure and Algorithms Tutorials*, 2024, GeeksforGeeks. Available at: `https://www.geeksforgeeks.org/dsa/binary-heap/`.

2. GeeksforGeeks, *Dijkstra's Shortest Path Algorithm*, 2024, GeeksforGeeks. Available at: `https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/`.

3. Stack Overflow, *Read text file and parse in Python*, 2018, Stack Overflow. Available at: `https://stackoverflow.com/questions/51345024/read-text-file-and-parse-in-python`.

4. WilliamFiset, *Breadth First Search (BFS): Graph Theory*, 2018, YouTube. Available at: `https://www.youtube.com/watch?v=Uh2ebFW8OYM`.

5. GeeksforGeeks, *Breadth First Search or BFS for a Graph*, 2024, GeeksforGeeks. Available at: `https://www.geeksforgeeks.org/breadth-first-search-or-bfs-for-a-graph/`.

6. GeeksforGeeks, *Proof that Hamiltonian Path is NP-Complete* 2025, GeeksforGeeks. Available at: `https://www.geeksforgeeks.org/`

theory-of-computation/proof-hamiltonian-path-np-complete/