Red Scare

In general

We used the following data structure for alle the problems:
Graph data structure: A modified version of algs4 directed graph, merged with its SymbolGraph. https://algs4.cs.princeton.edu/code/
We use a directed graph because it can handle both directed and undirected graphs. For undirected graphs, whenever vertices A and B are connected, we add an edge from A to B and an edge from B to A.
Most of the algorithms we run use the fact that the vertices are indexed from 0 through n-1, with n being number of vertices. But since the vertices in red scare can have strings as names, we use a modified symbol graph. In our graph there is a map from vertex names to indices and its inverse from indices to names.
To handle red vertices, we have an additional map that maps from a vertex to a boolean indicating whether the given vertex is red or not.
The graph keeps track of inputs, such as count of vertices and edges. It has an adjacency list for each vertex to keep track of edges.

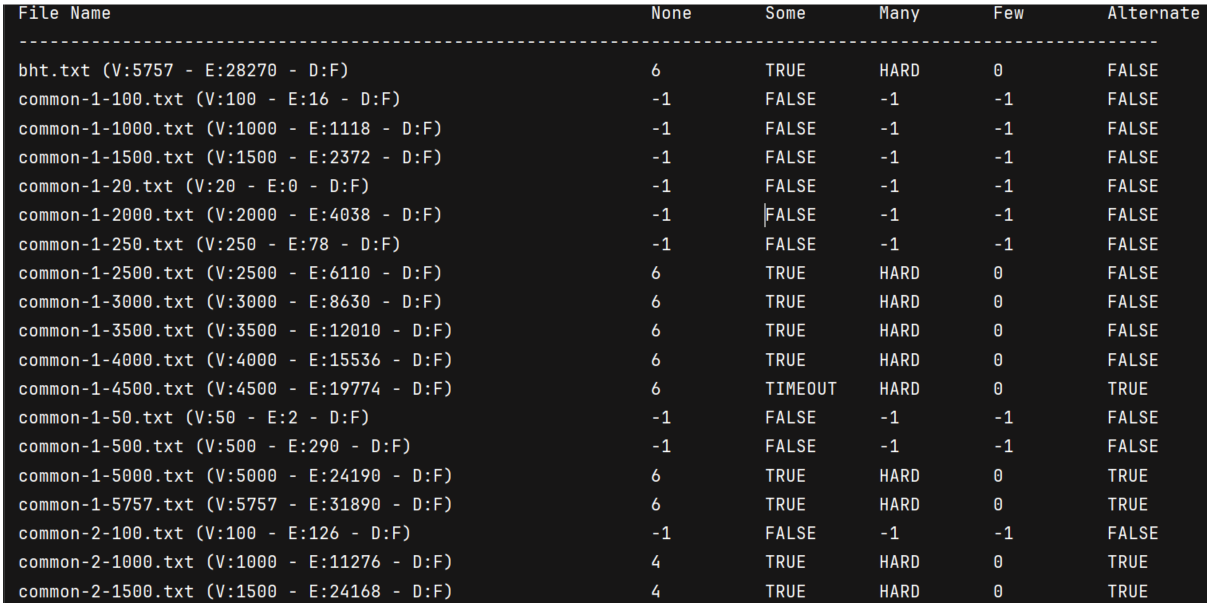
Total space: O(V + E), where V = vertices and E = edges.

The input is only parsed once per file, and each method then receives a copy of the graph, so different exercises won't affect the same graph.

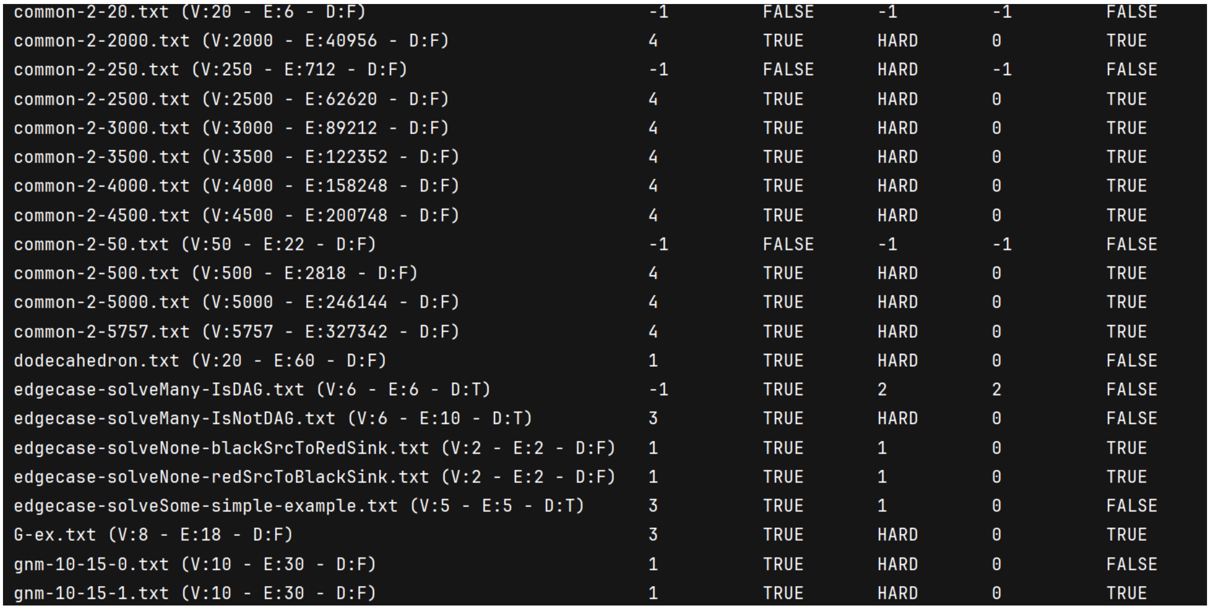
Each exercise also has its own test method, which is simply known answers manually run by the group on some of the sample cases, which helped us debugging through solving the exercises. Some of the exercises also contains cross-exercise checks, such as for the Many problem. We know that If there is a path from s to t (found in None), and also no such path with red vertices (found in Some), we know the longest path with red vertices is 0 (answer for Many)!

Results

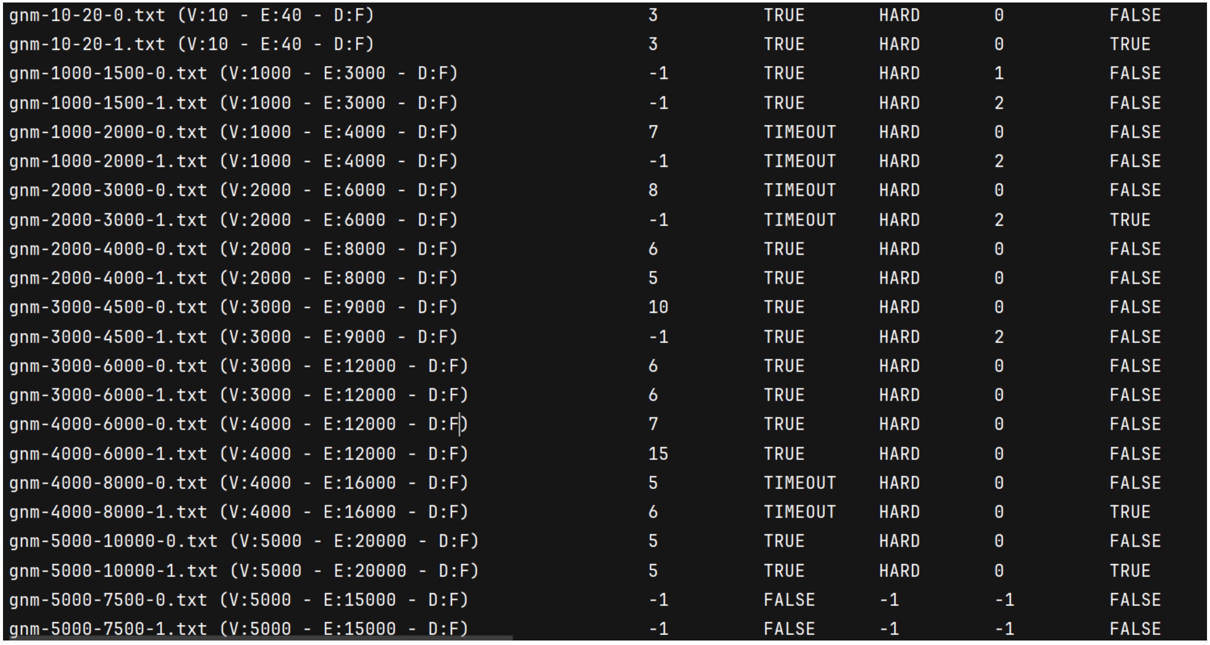
Format:
1st column: Filename + (count of vertices, count of edges, isDirectedOrUndirected)
2nd column: Answer for None. Positive integer if reachable, and -1 if a simple path from S to T doesn't exist.
3rd column: Answer for Some. Either True, False or Timeout, whereas timeout denotes that we stopped checking for an answer after 2 seconds of computation time on that specific file.
4th column: Answer for Many. -1 denotes no path from S to T. Any other integer denotes the number of red vertices from S to T. "HARD" denotes, the graph was not acyclic, and thus couldn't be solved in polynomial time.
5th column: Answer for Few. An integer denoting fewest number of red vertices from S to T. -1 means no simple path from S to T exists.
6th column: Answer for Alternate. Either true or false, if an color-alternating simple path from S to T exists.



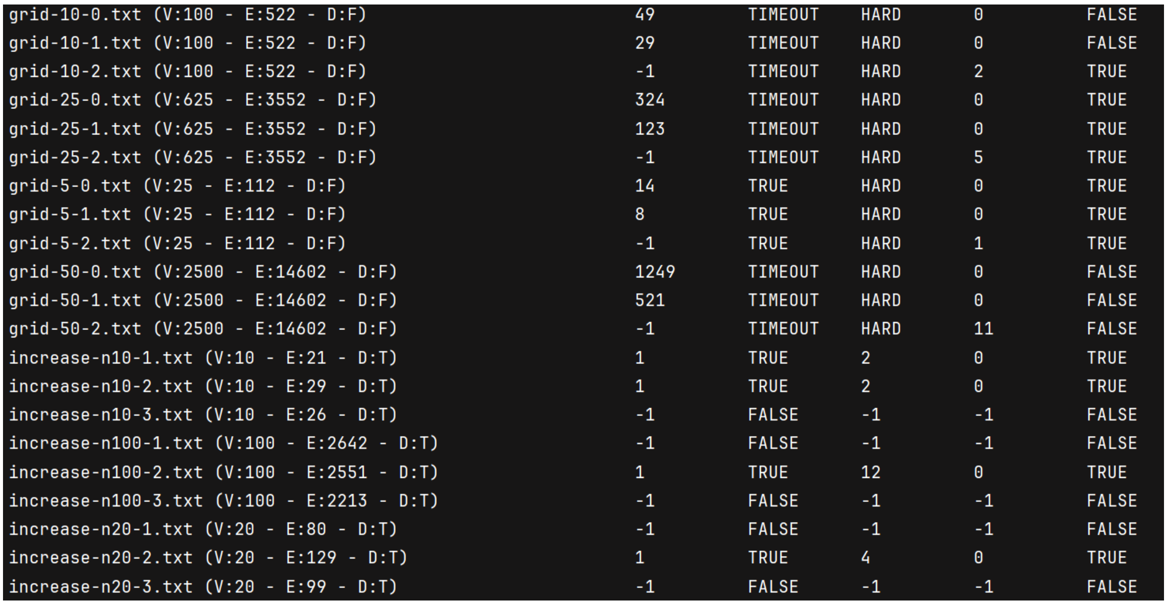
```
File Name                                                    None       Some       Many       Few        Alternate
--------------------------------------------------------------------------------------------------------------
bht.txt (V:5757 - E:28270 - D:F)                             6          TRUE       HARD       0          FALSE
common-1-100.txt (V:100 - E:16 - D:F)                        -1         FALSE      -1         -1         FALSE
common-1-1000.txt (V:1000 - E:1118 - D:F)                    -1         FALSE      -1         -1         FALSE
common-1-1500.txt (V:1500 - E:2372 - D:F)                    -1         FALSE      -1         -1         FALSE
common-1-20.txt (V:20 - E:0 - D:F)                           -1         FALSE      -1         -1         FALSE
common-1-2000.txt (V:2000 - E:4038 - D:F)                    -1         FALSE      -1         -1         FALSE
common-1-250.txt (V:250 - E:78 - D:F)                        -1         FALSE      -1         -1         FALSE
common-1-2500.txt (V:2500 - E:6110 - D:F)                    6          TRUE       HARD       0          FALSE
common-1-3000.txt (V:3000 - E:8630 - D:F)                    6          TRUE       HARD       0          FALSE
common-1-3500.txt (V:3500 - E:12010 - D:F)                   6          TRUE       HARD       0          FALSE
common-1-4000.txt (V:4000 - E:15536 - D:F)                   6          TRUE       HARD       0          FALSE
common-1-4500.txt (V:4500 - E:19774 - D:F)                   6          TIMEOUT    HARD       0          TRUE
common-1-50.txt (V:50 - E:2 - D:F)                           -1         FALSE      -1         -1         FALSE
common-1-500.txt (V:500 - E:290 - D:F)                       -1         FALSE      -1         -1         FALSE
common-1-5000.txt (V:5000 - E:24190 - D:F)                   6          TRUE       HARD       0          TRUE
common-1-5757.txt (V:5757 - E:31890 - D:F)                   6          TRUE       HARD       0          TRUE
common-2-100.txt (V:100 - E:126 - D:F)                       -1         FALSE      -1         -1         FALSE
common-2-1000.txt (V:1000 - E:11276 - D:F)                   4          TRUE       HARD       0          TRUE
common-2-1500.txt (V:1500 - E:24168 - D:F)                   4          TRUE       HARD       0          TRUE
```



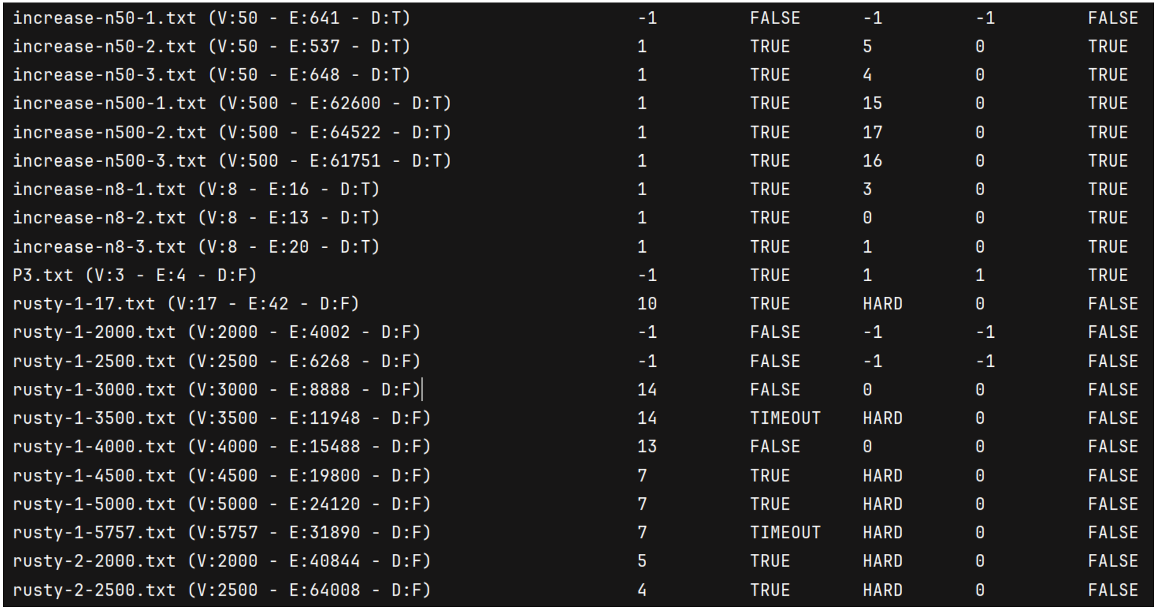
```
common-2-20.txt (V:20 - E:6 - D:F)                           -1         FALSE      -1         -1         FALSE
common-2-2000.txt (V:2000 - E:40956 - D:F)                   4          TRUE       HARD       0          TRUE
common-2-250.txt (V:250 - E:712 - D:F)                       -1         FALSE      HARD       -1         FALSE
common-2-2500.txt (V:2500 - E:62620 - D:F)                   4          TRUE       HARD       0          TRUE
common-2-3000.txt (V:3000 - E:89212 - D:F)                   4          TRUE       HARD       0          TRUE
common-2-3500.txt (V:3500 - E:122352 - D:F)                  4          TRUE       HARD       0          TRUE
common-2-4000.txt (V:4000 - E:158248 - D:F)                  4          TRUE       HARD       0          TRUE
common-2-4500.txt (V:4500 - E:200748 - D:F)                  4          TRUE       HARD       0          TRUE
common-2-50.txt (V:50 - E:22 - D:F)                          -1         FALSE      -1         -1         FALSE
common-2-500.txt (V:500 - E:2818 - D:F)                      4          TRUE       HARD       0          TRUE
common-2-5000.txt (V:5000 - E:246144 - D:F)                  4          TRUE       HARD       0          TRUE
common-2-5757.txt (V:5757 - E:327342 - D:F)                  4          TRUE       HARD       0          TRUE
dodecahedron.txt (V:20 - E:60 - D:F)                         1          TRUE       HARD       0          FALSE
edgecase-solveMany-IsDAG.txt (V:6 - E:6 - D:T)               -1         TRUE       2          2          FALSE
edgecase-solveMany-IsNotDAG.txt (V:6 - E:10 - D:T)           3          TRUE       HARD       0          FALSE
edgecase-solveNone-blackSrcToRedSink.txt (V:2 - E:2 - D:F)   1          TRUE       1          0          TRUE
edgecase-solveNone-redSrcToBlackSink.txt (V:2 - E:2 - D:F)   1          TRUE       1          0          TRUE
edgecase-solveSome-simple-example.txt (V:5 - E:5 - D:T)      3          TRUE       1          0          FALSE
G-ex.txt (V:8 - E:18 - D:F)                                  3          TRUE       HARD       0          TRUE
gnm-10-15-0.txt (V:10 - E:30 - D:F)                          1          TRUE       HARD       0          FALSE
gnm-10-15-1.txt (V:10 - E:30 - D:F)                          1          TRUE       HARD       0          TRUE
```



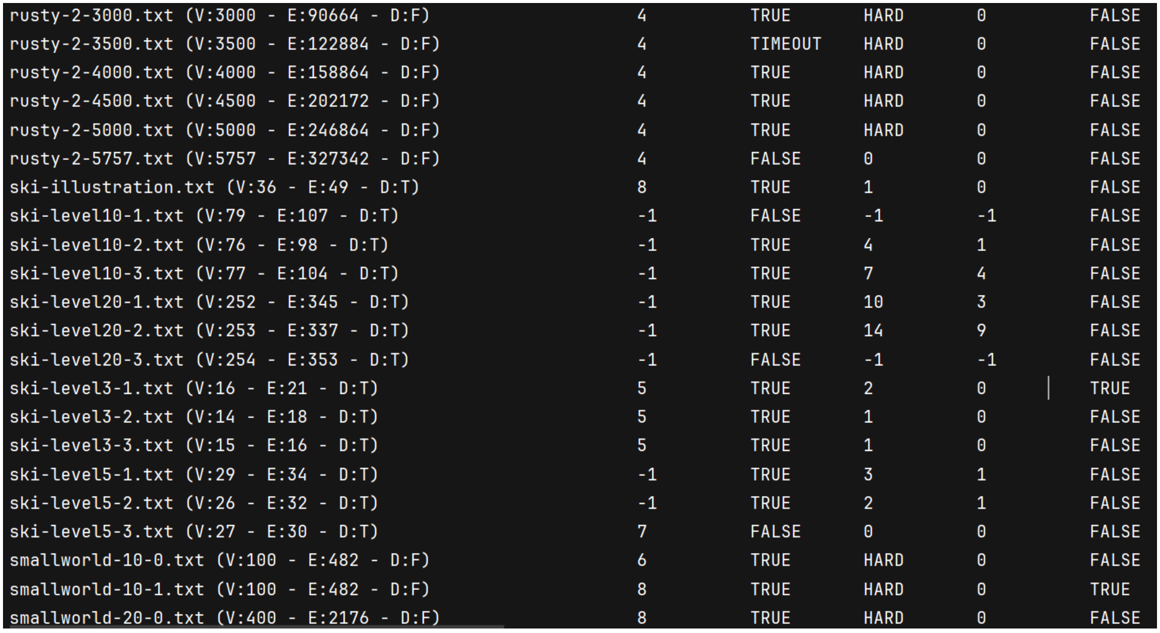
```
gnm-10-20-0.txt (V:10 - E:40 - D:F)                          3          TRUE       HARD       0          FALSE
gnm-10-20-1.txt (V:10 - E:40 - D:F)                          3          TRUE       HARD       0          TRUE
gnm-1000-1500-0.txt (V:1000 - E:3000 - D:F)                  -1         TRUE       HARD       1          FALSE
gnm-1000-1500-1.txt (V:1000 - E:3000 - D:F)                  -1         TRUE       HARD       2          FALSE
gnm-1000-2000-0.txt (V:1000 - E:4000 - D:F)                  7          TIMEOUT    HARD       0          FALSE
gnm-1000-2000-1.txt (V:1000 - E:4000 - D:F)                  -1         TIMEOUT    HARD       2          FALSE
gnm-2000-3000-0.txt (V:2000 - E:6000 - D:F)                  8          TIMEOUT    HARD       0          FALSE
gnm-2000-3000-1.txt (V:2000 - E:6000 - D:F)                  -1         TIMEOUT    HARD       2          TRUE
gnm-2000-4000-0.txt (V:2000 - E:8000 - D:F)                  6          TRUE       HARD       0          FALSE
gnm-2000-4000-1.txt (V:2000 - E:8000 - D:F)                  5          TRUE       HARD       0          FALSE
gnm-3000-4500-0.txt (V:3000 - E:9000 - D:F)                  10         TRUE       HARD       0          FALSE
gnm-3000-4500-1.txt (V:3000 - E:9000 - D:F)                  -1         TRUE       HARD       2          FALSE
gnm-3000-6000-0.txt (V:3000 - E:12000 - D:F)                 6          TRUE       HARD       0          FALSE
gnm-3000-6000-1.txt (V:3000 - E:12000 - D:F)                 6          TRUE       HARD       0          FALSE
gnm-4000-6000-0.txt (V:4000 - E:12000 - D:F)                 7          TRUE       HARD       0          FALSE
gnm-4000-6000-1.txt (V:4000 - E:12000 - D:F)                 15         TRUE       HARD       0          FALSE
gnm-4000-8000-0.txt (V:4000 - E:16000 - D:F)                 5          TIMEOUT    HARD       0          FALSE
gnm-4000-8000-1.txt (V:4000 - E:16000 - D:F)                 6          TIMEOUT    HARD       0          TRUE
gnm-5000-10000-0.txt (V:5000 - E:20000 - D:F)                5          TRUE       HARD       0          FALSE
gnm-5000-10000-1.txt (V:5000 - E:20000 - D:F)                5          TRUE       HARD       0          TRUE
gnm-5000-7500-0.txt (V:5000 - E:15000 - D:F)                 -1         FALSE      -1         -1         FALSE
gnm-5000-7500-1.txt (V:5000 - E:15000 - D:F)                 -1         FALSE      -1         -1         FALSE
```



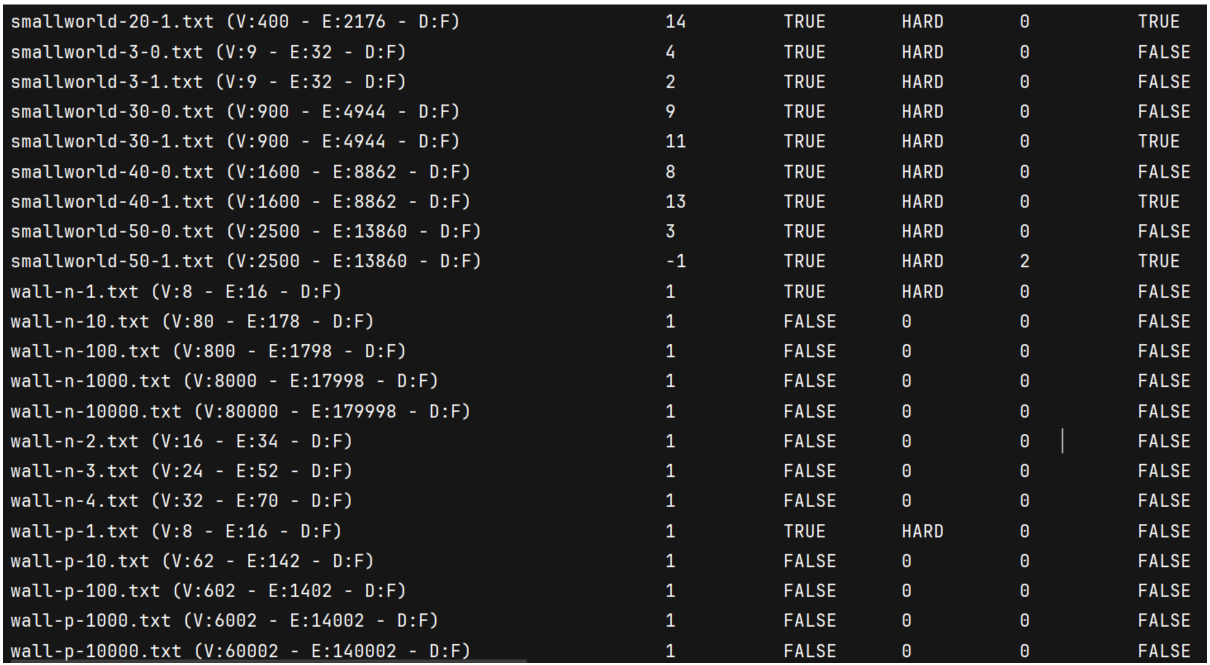
```
grid-10-0.txt (V:100 - E:522 - D:F)                     49      TIMEOUT   HARD      0         FALSE
grid-10-1.txt (V:100 - E:522 - D:F)                     29      TIMEOUT   HARD      0         FALSE
grid-10-2.txt (V:100 - E:522 - D:F)                     -1      TIMEOUT   HARD      2         TRUE
grid-25-0.txt (V:625 - E:3552 - D:F)                    324     TIMEOUT   HARD      0         TRUE
grid-25-1.txt (V:625 - E:3552 - D:F)                    123     TIMEOUT   HARD      0         TRUE
grid-25-2.txt (V:625 - E:3552 - D:F)                    -1      TIMEOUT   HARD      5         TRUE
grid-5-0.txt (V:25 - E:112 - D:F)                       14      TRUE      HARD      0         TRUE
grid-5-1.txt (V:25 - E:112 - D:F)                       8       TRUE      HARD      0         TRUE
grid-5-2.txt (V:25 - E:112 - D:F)                       -1      TRUE      HARD      1         TRUE
grid-50-0.txt (V:2500 - E:14602 - D:F)                  1249    TIMEOUT   HARD      0         FALSE
grid-50-1.txt (V:2500 - E:14602 - D:F)                  521     TIMEOUT   HARD      0         FALSE
grid-50-2.txt (V:2500 - E:14602 - D:F)                  -1      TIMEOUT   HARD      11        FALSE
increase-n10-1.txt (V:10 - E:21 - D:T)                  1       TRUE      2         0         TRUE
increase-n10-2.txt (V:10 - E:29 - D:T)                  1       TRUE      2         0         TRUE
increase-n10-3.txt (V:10 - E:26 - D:T)                  -1      FALSE     -1        -1        FALSE
increase-n100-1.txt (V:100 - E:2642 - D:T)              -1      FALSE     -1        -1        FALSE
increase-n100-2.txt (V:100 - E:2551 - D:T)              1       TRUE      12        0         TRUE
increase-n100-3.txt (V:100 - E:2213 - D:T)              -1      FALSE     -1        -1        FALSE
increase-n20-1.txt (V:20 - E:80 - D:T)                  -1      FALSE     -1        -1        FALSE
increase-n20-2.txt (V:20 - E:129 - D:T)                 1       TRUE      4         0         TRUE
increase-n20-3.txt (V:20 - E:99 - D:T)                  -1      FALSE     -1        -1        FALSE
```



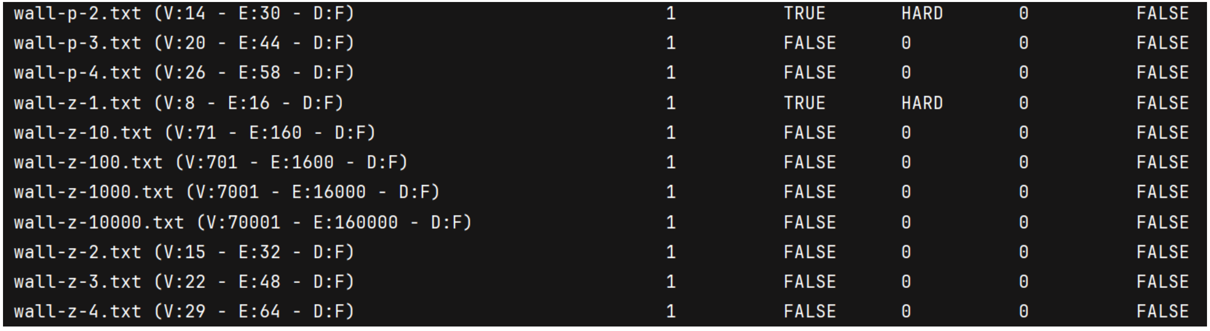
```
increase-n50-1.txt (V:50 - E:641 - D:T)                 -1      FALSE     -1        -1        FALSE
increase-n50-2.txt (V:50 - E:537 - D:T)                 1       TRUE      5         0         TRUE
increase-n50-3.txt (V:50 - E:648 - D:T)                 1       TRUE      4         0         TRUE
increase-n500-1.txt (V:500 - E:62600 - D:T)             1       TRUE      15        0         TRUE
increase-n500-2.txt (V:500 - E:64522 - D:T)             1       TRUE      17        0         TRUE
increase-n500-3.txt (V:500 - E:61751 - D:T)             1       TRUE      16        0         TRUE
increase-n8-1.txt (V:8 - E:16 - D:T)                    1       TRUE      3         0         TRUE
increase-n8-2.txt (V:8 - E:13 - D:T)                    1       TRUE      0         0         TRUE
increase-n8-3.txt (V:8 - E:20 - D:T)                    1       TRUE      1         0         TRUE
P3.txt (V:3 - E:4 - D:F)                                -1      TRUE      1         1         TRUE
rusty-1-17.txt (V:17 - E:42 - D:F)                      10      TRUE      HARD      0         FALSE
rusty-1-2000.txt (V:2000 - E:4002 - D:F)                -1      FALSE     -1        -1        FALSE
rusty-1-2500.txt (V:2500 - E:6268 - D:F)                -1      FALSE     -1        -1        FALSE
rusty-1-3000.txt (V:3000 - E:8888 - D:F)                14      FALSE     0         0         FALSE
rusty-1-3500.txt (V:3500 - E:11948 - D:F)               14      TIMEOUT   HARD      0         FALSE
rusty-1-4000.txt (V:4000 - E:15488 - D:F)               13      FALSE     0         0         FALSE
rusty-1-4500.txt (V:4500 - E:19800 - D:F)               7       TRUE      HARD      0         FALSE
rusty-1-5000.txt (V:5000 - E:24120 - D:F)               7       TRUE      HARD      0         FALSE
rusty-1-5757.txt (V:5757 - E:31890 - D:F)               7       TIMEOUT   HARD      0         FALSE
rusty-2-2000.txt (V:2000 - E:40844 - D:F)               5       TRUE      HARD      0         FALSE
rusty-2-2500.txt (V:2500 - E:64008 - D:F)               4       TRUE      HARD      0         FALSE
```



```
rusty-2-3000.txt (V:3000 - E:90664 - D:F)               4       TRUE      HARD      0         FALSE
rusty-2-3500.txt (V:3500 - E:122884 - D:F)              4       TIMEOUT   HARD      0         FALSE
rusty-2-4000.txt (V:4000 - E:158864 - D:F)              4       TRUE      HARD      0         FALSE
rusty-2-4500.txt (V:4500 - E:202172 - D:F)              4       TRUE      HARD      0         FALSE
rusty-2-5000.txt (V:5000 - E:246864 - D:F)              4       TRUE      HARD      0         FALSE
rusty-2-5757.txt (V:5757 - E:327342 - D:F)              4       FALSE     0         0         FALSE
ski-illustration.txt (V:36 - E:49 - D:T)                8       TRUE      1         0         FALSE
ski-level10-1.txt (V:79 - E:107 - D:T)                  -1      FALSE     -1        -1        FALSE
ski-level10-2.txt (V:76 - E:98 - D:T)                   -1      TRUE      4         1         FALSE
ski-level10-3.txt (V:77 - E:104 - D:T)                  -1      TRUE      7         4         FALSE
ski-level20-1.txt (V:252 - E:345 - D:T)                 -1      TRUE      10        3         FALSE
ski-level20-2.txt (V:253 - E:337 - D:T)                 -1      TRUE      14        9         FALSE
ski-level20-3.txt (V:254 - E:353 - D:T)                 -1      FALSE     -1        -1        FALSE
ski-level3-1.txt (V:16 - E:21 - D:T)                    5       TRUE      2         0         TRUE
ski-level3-2.txt (V:14 - E:18 - D:T)                    5       TRUE      1         0         FALSE
ski-level3-3.txt (V:15 - E:16 - D:T)                    5       TRUE      1         0         FALSE
ski-level5-1.txt (V:29 - E:34 - D:T)                    -1      TRUE      3         1         FALSE
ski-level5-2.txt (V:26 - E:32 - D:T)                    -1      TRUE      2         1         FALSE
ski-level5-3.txt (V:27 - E:30 - D:T)                    7       FALSE     0         0         FALSE
smallworld-10-0.txt (V:100 - E:482 - D:F)               6       TRUE      HARD      0         FALSE
smallworld-10-1.txt (V:100 - E:482 - D:F)               8       TRUE      HARD      0         TRUE
smallworld-20-0.txt (V:400 - E:2176 - D:F)              8       TRUE      HARD      0         FALSE
```



```
smallworld-20-1.txt (V:400 - E:2176 - D:F)          14      TRUE    HARD    0       TRUE
smallworld-3-0.txt (V:9 - E:32 - D:F)               4       TRUE    HARD    0       FALSE
smallworld-3-1.txt (V:9 - E:32 - D:F)               2       TRUE    HARD    0       FALSE
smallworld-30-0.txt (V:900 - E:4944 - D:F)          9       TRUE    HARD    0       FALSE
smallworld-30-1.txt (V:900 - E:4944 - D:F)          11      TRUE    HARD    0       TRUE
smallworld-40-0.txt (V:1600 - E:8862 - D:F)         8       TRUE    HARD    0       FALSE
smallworld-40-1.txt (V:1600 - E:8862 - D:F)         13      TRUE    HARD    0       TRUE
smallworld-50-0.txt (V:2500 - E:13860 - D:F)        3       TRUE    HARD    0       FALSE
smallworld-50-1.txt (V:2500 - E:13860 - D:F)        -1      TRUE    HARD    2       TRUE
wall-n-1.txt (V:8 - E:16 - D:F)                     1       TRUE    HARD    0       FALSE
wall-n-10.txt (V:80 - E:178 - D:F)                  1       FALSE   0       0       FALSE
wall-n-100.txt (V:800 - E:1798 - D:F)               1       FALSE   0       0       FALSE
wall-n-1000.txt (V:8000 - E:17998 - D:F)            1       FALSE   0       0       FALSE
wall-n-10000.txt (V:80000 - E:179998 - D:F)         1       FALSE   0       0       FALSE
wall-n-2.txt (V:16 - E:34 - D:F)                    1       FALSE   0       0       FALSE
wall-n-3.txt (V:24 - E:52 - D:F)                    1       FALSE   0       0       FALSE
wall-n-4.txt (V:32 - E:70 - D:F)                    1       FALSE   0       0       FALSE
wall-p-1.txt (V:8 - E:16 - D:F)                     1       TRUE    HARD    0       FALSE
wall-p-10.txt (V:62 - E:142 - D:F)                  1       FALSE   0       0       FALSE
wall-p-100.txt (V:602 - E:1402 - D:F)               1       FALSE   0       0       FALSE
wall-p-1000.txt (V:6002 - E:14002 - D:F)            1       FALSE   0       0       FALSE
wall-p-10000.txt (V:60002 - E:140002 - D:F)         1       FALSE   0       0       FALSE
```



```
wall-p-2.txt (V:14 - E:30 - D:F)                    1       TRUE    HARD    0       FALSE
wall-p-3.txt (V:20 - E:44 - D:F)                    1       FALSE   0       0       FALSE
wall-p-4.txt (V:26 - E:58 - D:F)                    1       FALSE   0       0       FALSE
wall-z-1.txt (V:8 - E:16 - D:F)                     1       TRUE    HARD    0       FALSE
wall-z-10.txt (V:71 - E:160 - D:F)                  1       FALSE   0       0       FALSE
wall-z-100.txt (V:701 - E:1600 - D:F)               1       FALSE   0       0       FALSE
wall-z-1000.txt (V:7001 - E:16000 - D:F)            1       FALSE   0       0       FALSE
wall-z-10000.txt (V:70001 - E:160000 - D:F)         1       FALSE   0       0       FALSE
wall-z-2.txt (V:15 - E:32 - D:F)                    1       FALSE   0       0       FALSE
wall-z-3.txt (V:22 - E:48 - D:F)                    1       FALSE   0       0       FALSE
wall-z-4.txt (V:29 - E:64 - D:F)                    1       FALSE   0       0       FALSE
```

None

Problem in short: Shortest path in unweighted graph.
There are three obvious approaches, which are either:
1. When parsing the input, skip all red vertices
2. After parsing, remove all red vertices from the constructed graph
3. Modify the path finding algorithm to avoid red vertices

Implementation details:
We went with option 2.
We added the method 'removeAllRed' to our data structure, which simply removes all red vertices and their connected edges, except if it is the starting og ending vertex, and the dependent edges from/to the vertices, by going trough the graph in linear time.
After that, we simply used BFS (also from the algs4 library) from vertices s to t. Since the graph is unweighted, BFS returns the shortest path if there is one.

Works for: All provided graphs.

Running time: O(V+E)

Some

Problem in short: Path finding, with a condition. The simple path from S to T must contain at least 1 red vertex. This problem is NP-Hard.

Reduction: We don't honestly know. But our best guess is 3-SAT $<_p$ Some.
Our feeling is something like:
Given a 3-SAT instance we thought about how we could construct a graph that only passed through a red vertex if all Clauses were true. However we could not come up with anything that would work.

Implementation details:
Modified DFS from vertices s to t.
DFS is preferred to BFS, as it tries one path at a time. During traversal, we keep track of visited nodes, to prevent DFS from revisiting a vertex, which would break its simple path, as it requires no revisited vertexes. If it couldn't find a path from S to T with a red vertex, it will clear the visited tracker, making all vertexes available again, and will try a new path. As such, it might end up exploring all possible simple paths, checking if a red vertex is part of any of the paths. Normally DFS "marks" visited vertexes to avoid this. But this may exclude possible paths containing red vertices (see *Figure 1*), which is why we changed this part of the implementation.
Therefore this algorithm doesn't run in polynomial time, and as such a timeout has been implemented, which stops it from running after 2 seconds of attempting to find a valid simple path.

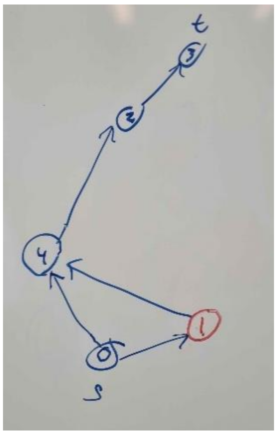


Figure 2: If DFS starts by finding the path 0,4,2,3 it will backtrack and mark all vertices in that path. Then it will continue to the red vertex 1, but since vertex 4 is marked it will terminate without finding a path containing the red vertex.

Works for: All (though only terminates within 2 seconds when n, m are small)

Running time:
It basically tries to do all permutations of the vertices (except start and end are always the same). Such as if we have vertices 1,2,3,4,5 which are all connected, being the worst case.

Assuming s=1 and t=5 we try the following simple paths:
1,2,3,4,5
1,2,4,3,5
1,3,2,4,5
1,3,4,2,5
1,4,2,3,5
1,4,3,2,5

Therefore the running time is O((n-2)!)

Many

Problem in short/Reduction: Longest path problem
https://en.wikipedia.org/wiki/Longest_path_problem

Where longest here refers to number of red vertices, preferring as many as possible.

If the graph is acyclic:

Implementation details:

1. If directed acyclic graph:
Topological sort inspired by Algs4 library and by Kahns Algorithm implemented by geeksforgeeks. https://www.geeksforgeeks.org/topological-sorting-indegree-based-solution/
If there is a cycle in the graph the algorithm will detect this.

If there is a cycle the problem is NP hard and we return HARD. Otherwise we do the following:

Traverse the topological order of the vertices O(V), and foreach edge O(E), relaxes the edge, which is basically checking if a longer path has been found, by counting the weight of black edges as 0 and red edges as 1, and if so we update our maxRedVerticesToV [x] array. Black edges refer to edges going into black vertices and red edges are edges going into red vertices.

2. If undirected acyclic graph:

Run BFS. If at some point we reach a marked vertex it means there is a cycle and the problem is np hard and we return HARD. Otherwise there can only be at most one path from s to t. BFS will find the path if there is one. Then we count the red vertices on the path and return the count. Otherwise return -1.

Running time:
Topological order: O(V+E)
Traversal: O(V+E):
Therefore in total: O(V+E)

Works for: DAC's and undirected acyclic graphs

If graph is not acyclic:

We will show that Many is NP-complete by reducing st-longest path to Many. LP $\leq_P$ Many.

Reduction:

Given a weighted graph G. (For unweighted we can reduce to weighted by setting each edge to a weight of 1).

Create a new graph G' which is equal to G.

For each edge in G' with a weight of k, such that e connects vertex a and vertex b:

Remove e from G and add k red vertices r_1 through r_k. Now add the following edges: One that connects a to r_1. One that connects r_k to b. The remaining edges are added such that each vertex r_x where 0 < x < k is connected to r_x+1.

If the graph is directed and a has an outgoing edge to b, every added edge must also be directed such that a path from a to b is created. Otherwise, every added edge will be undirected.

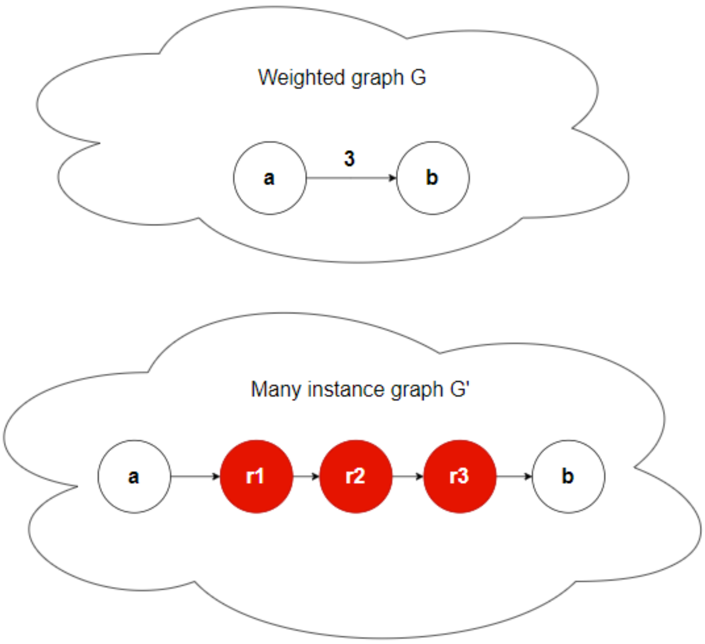


Figure 3 : Reduction from weighted graph to a graph for the Many problem.

The solution to the Many problem on G' will give us the answer to the st-longest path in G. The reduction itself has a running time of O(E*C) where C is the highest edge capacity, since we go through every edge and for each edge, we add at most C+1 many new vertices and edges.

Many can also be reduced to the longest path problem:

Given a Graph to the Many problem G.

Create a new graph G' which is equal to G but with the following change.

All edges going into a red vertex gets a weight of 1. All other edges get a weight of 0. If the s vertex is red, add a new vertex s' with an edge of weight 1 to s. The longest path from s' to t will be the one that goes through the most red vertices. Therefore, the length of the longest path on G' is also the solution to Many on G. This reduction has a running time of O(E) since we modify every edge and potentially add one extra vertex and edge to s in constant time.

Thus LP $\equiv_P$ Many

Few

Problem in short: Shortest Path in a Binary Weight Graph. Edges to black vertices have weight 0 and edges to red vertices have weight 1.

Implementation details:
We have a 0-1 BFS. It is BFS but instead of a normal queue it uses a simple priority queue that takes all 0's before 1's. This is implemented as a dequeue that adds 0's to the front and 1's to the back. This ensures that the algorithm prefers taking fewest amount of red vertices.

Works for: All graphs

Running time: O(V+E)

Alternate

Problem in short: Path finding with restriction.

Implementation details:
Modified BFS from algs4 library.
During traversal it checks color constraints on edges, by checking current vertex color of 'v', then color of neighbour 'w' (in constant time). Only if they alternate in color (color(v) != color(w)) do we proceed to handle the edge and enqueue the vertex w.

Works for: All graphs

Running time: O(V+E)