**Generalized BFS**

GBFS playground states the following

* For any graph that belongs to the set of graphs where d(s,t) > N/C

Where d(s,t) is the shortest path between s and t, s being source and t being destination

* N=> # nodes
* N is a multiple of C, both N and C are integers
* Claim is that there exists a layer of node/s which if removed will disconnect s and t
* And that this layer consists of at most C-1 nodes
* Claim: there exists a set of C-1 nodes which, if removed, will disconnect s and t. Reason: we don’t want to give away the solution.

**Purpose**

* The purpose of the playground is to explore undirected graphs and the intuition that nodes that are far apart have a more tenuous connection than nodes that are close together. It is about the susceptibility of paths to the deletion of nodes.
* The purpose of this playground is to test the solve method and the provide method in the avatars.
* How quickly can your avatar solve a graph, can it solve it within the round time limit?
* How big a graph can your avatar generate?
* Does it pass the belongsTo and valid methods?
* Can you generate a graph that passes the belongsTo method these 2 methods and still doesn’t have a valid solution?
* If so…kudos to you..and you deserve to win

**Playground attribute/element definitions**

* **Instance** consists of a graph, a source node “s” and a destination node “t”
* **Graph** is a list of adjacencies
* **Adjacency** is a parent node and its neighbors
* **Node** is a string. an identifier which ultimately is a string
* **Solution** is a list of nodes which if removed disconnect s and t
* **InstanceSet** is the setof graphs that … consists of C which is an integer value

**InstanceSet:**

belongsTo => This method checks if the #nodes in the graph is within the range specified in the config file

 It also checks that the #nodes in the graph is a multiple of C

 And finally it checks if the distance(s,t) is greater than N/C, N => #nodes in the graph

Note that this method has been changed, in that it now takes in the config along with instance

valid => This method checks if the “C” value is within range specified in the config file

**Instance:**

quality => This checks the quality of the solution

 First off it checks if the solution layer length is less than or equal to C-1

 If so then does removing the layer disconnect s and t, if it does we return a quality 1 else 0

valid => This just checks if removing the layer (as provided in the solution) disconnects s and t, if so it’s valid else invalid

**Solution:**

Solution is just a list of nodes that when removed from the graph disconnects s and t. Note that the solution layer may **not contain s or t.**

**Baby Avatar:**

creategraph => This is a helper method which creates a static 10 node graph that satisfies all the above mentioned properties and will pass the valid?? and belongsTo methods.

solve => DO NOT TALK ABOUT LAYER This method as the name suggests solves a given graph, as in it finds the layer of nodes which when removed disconnect s and t and it also checks the solution layer length is less than or equal to C-1.

provide => provides a graph with N nodes where N is a multiple of C, C being part of the instance set and can be derived from the claim. It can use the creategraph method to do so

**Configuration parameters:**

The claims must be new is set to false since we currently can generate graphs of up to 1k nodes which limits the range of C, and hence we can’t have it generate a new C each time.

scg\_config[

domain:gbfs.GBFSDomain

protocols: scg.protocol.ForAllExistsEqual

tournamentStyle: full round-robin

turnDuration: 60 //seconds

maxNumAvatars: 20

minStrengthening: 0.01

initialReputation: 100.0

maxReputation: 1000.0

reputationFactor: 0.4

minProposals: 1

maxProposals: 5

numRounds: 9

proposedClaimMustBeNew: false

minConfidence: 0.5

]

gbfs.GBFSConfig {{ gbfs\_config[

minN: 4

maxN: 1000

minC: 4

maxC: 20

]}}