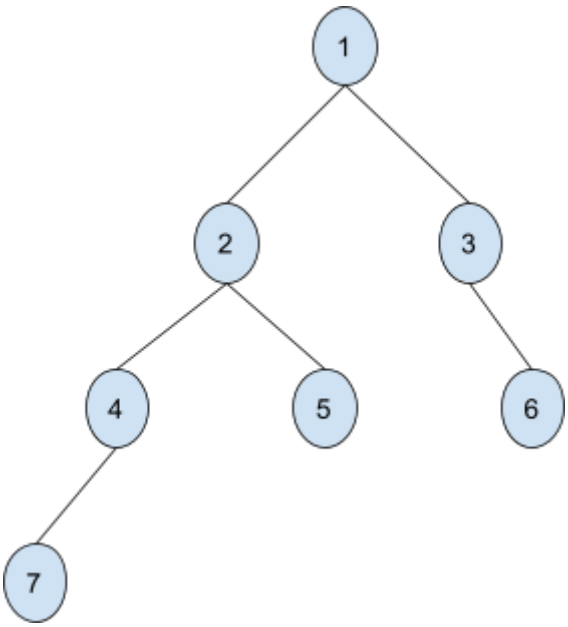
1 - Path Search

You are given a tree where each node is identified by a unique integer value. You must write a function that returns all of the unique pathways from the tree's root to each of the nodes specified by the values in a given array.



For example, in the picture presented above, when given the values [5, 4, 3] your function should return the following array:
[‘1->2->5’, ‘1->2->4’, ‘1->3’]

If a path is contained within another then it should be excluded from the returned list. For example when given the values [4, 7] the aforementioned function should return:
[‘1->2->4->7’]
and **not**
[‘1->2->4’, ‘1->2->4->7’]
since ‘1->2->4’ exists within the ‘1->2->4->7’ path.

You should write your code in the designated area inside *test_1/path_search.js*.

Your solution should run in O(n) and it is advised to use recursion.

You should **not** define trees with duplicate Node ids.

2 - Path Search Server

Use ExpressJS in order to develop an HTTP server which listens to incoming requests on Port 7070. You should implement the API: **GET** */api/tree/path-search*.
The aforementioned API call expects two parameters:

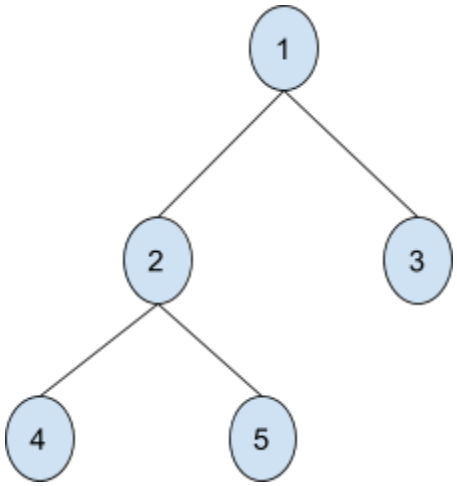
1. **tree:** A stringified JSON object representing a tree structure
2. **pathsToFind:** An array with nodes to search for in the tree

The server uses the function developed in the previous exercise in order to calculate and return the paths in the tree based on the **pathsToFind** variable.

The **tree** object is defined as follows:

- The tree node ids are depicted as the keys of the object
- If a tree node has kids then its value should be an object
- The keys of the objects under a tree node are the ids of its children tree nodes
- If a tree node does not have any kids then its value should be null

For example the object `{ 1: { 2: { 4: null, 5: null }, 3: null }}`
represents the following tree:



The folder test_2 contains some basic configuration for your Server. You may use the function provided in the src/utils.js file (`jsonTreeToTree`) in order to transform the **tree** parameter into a Tree Node structure similar to the one of the previous exercise.

When the server is complete you may test it by building a docker image out of it and then running it in the background:

1. docker build . -t path_search_server
2. docker run -p 7070:7070 -d path_search_server

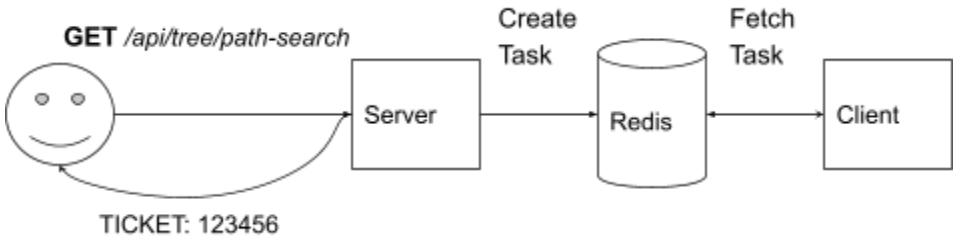
You can use Postman in order to send API requests.

Example:
GET *<host>/api/tree/path-search?tree="{1:{2:{4:null,5:null},3:null}}"?pathsToFind=4&pathsToFind=5*
>>>> [“1->2->4”, “1->2->5”]

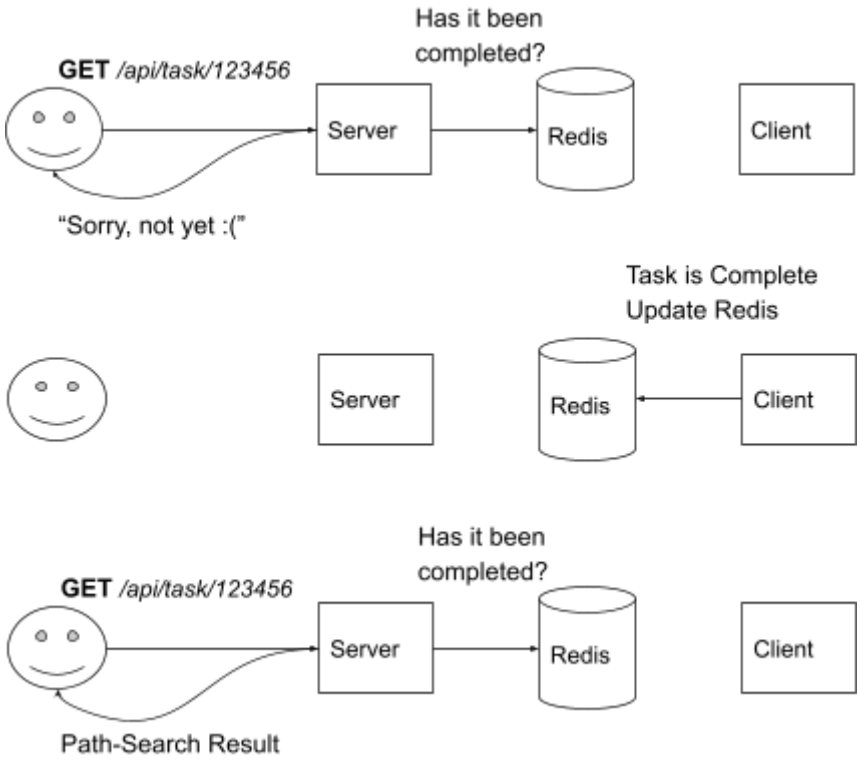
3 - Path Search Task Based Approach

Similar to Task 2 but the computation of the path should not happen on the server itself but on a separate worker. You should use Bull, a Task Manager based on Redis. The workflow is described below:

1. Client makes a GET */api/tree/path-search* request
2. Server receives the request and creates a task for it. Gives a ticket (a random integer) to the Client.



3. The worker fetches the task and calculates the answer. When the answer is created the client writes it in Redis (use ioredis as a redis client).
4. The client sends a message to another API: GET /api/task/<ticket> to check if the job has been completed and to see its result. The server should check whether the worker has finished the job and inserted its result into redis. On success it should return the result back to the client. If the worker has not finished the task execution yet it should return a message to the client.



In this exercise you have to do the following:

1. Research Bull and its usage
2. Update the *test_3/server/index.js* file and *insert two new APIs:*
    a. GET */api/tree/path-search*: Similar to test 2, however it now creates a Bull task and returns an integer (ticket) to the client)
    b. GET */api/task/<ticket>*: Where *ticket* is an integer. This api call checks the Redis Storage to see if a task is complete (you can use the ticket id to uniquely identi the Redis entry)
3. Update the *test_3/worker/index.js* file. You should create a task handler which performs a computation identical to this of the second exercise. After the calculation of the result it should store the data in Redis.

You can use the existing docker-compose file for the development (simply execute the command: `docker-compose up`

4 - Bonus

Make the necessary changes to the previous architecture so that the server responds to GET */api/tree/path-search* requests directly with the results calculated by the worker. The API request should stay open until the worker asynchronously calculates and stores the answer in redis.