CS310 Final Project

Graph App: *Graph Analysis and Visualization Suite*

Overview

This document describes the “Graph Analyzer and Visualizer” application, an AWS-based project that enables users to generate, examine, and depict graphs. The application allows users to upload files describing graphs, run graph analysis algorithms on uploaded graphs, and download image files visualizing existing graphs.

This project pertains to graphs as an abstract data type. That is, the application works with collections of vertices and edges that compose specific graph objects. The scope of this project includes weighted, undirected (WU) graphs.

The application supports three non-trivial operations intended to ease the process of working with graph objects:

1. **Generation**: Graphs can be randomly generated by the application using parameters specified by the user. In particular, the user can specify the type of graph they wish to produce (e.g. “bipartite”) and optionally the number of vertices and edges to include.
2. **Analysis**: User-uploaded graphs can be analyzed by the application using common graph algorithms such as Dijkstra’s algorithm and Prim’s algorithm. Users can specify the type of analysis to perform (e.g. cycle detection) and will receive the results of the examination on the specified graph.
3. **Visualization**: Graphs are best understood when depicted visually, but generating tidy visualizations is tedious for humans. This application allows users generate visualizations of uploaded graphs by calling a single API endpoint.

Components

User interaction with the application can be performed directly through the REST API or by using the provided Python3 client. In either case, the application and the user will interact with three types of files: **graph data (JSON) files**, **graph visualization (PNG) files**, and **graph analysis (JSON)** files.

When users upload or download graphs, they provide or receive JSON files that unambiguously describe the shape of the graph. Graph data files must adhere to the format provided below.

Note:

- All vertex identifiers are integer values
- All edges have positive weights and are formatted as arrays: [start, end, weight]
- At most one edge (A, B) or (B, A) for any two vertices A and B may be provided

```json
{
    “vertices”: [
        0, 4, 2, 10, -1            // example vertices
    ],
    “edges”: [
        [0, 4, 10.4], [10, -1, 3]  // example edges
```

```
      ]
}
```

Additionally, when retrieving the results of a graph analysis, users will receive JSON files that contain the analysis results. Analysis results files will always adhere to the format provided below, although the exact format of the value under the “data” key depends on the analysis type. See Appendix B for a full description of analysis results file formats.

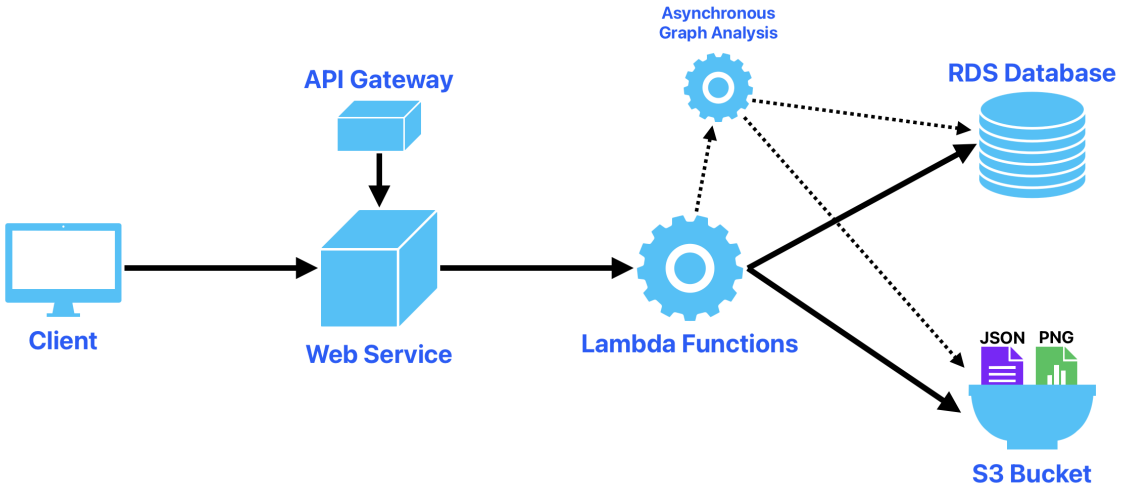
```json
{
      “type”: “mst”,
      “data”: [
            [0, 4], [4, -1]            // example edges composing MST
      ]
}
```

The application is split into four tiers: client, server, compute, and data. On the client tier, users will upload and download graph data files as well as download graph visualization files. The server tier is powered by API Gateway and provides a REST API through which clients can interact with the application.

When client requests are received by the web service, Lambda functions composing the compute tier are activated. A separate Lambda function exists for (and is triggered by) each of the API endpoints.

Each of the Lambda functions must access the RDS database in the data tier to either write or read stored graph information. Many of the functions use the database information to identify graph data and visualization files in the S3 bucket, which is also on the data tier. While some Lambda functions primarily retrieve and deliver files, others (such as those for random graph generation and graph analysis) execute algorithms and other forms of computation.

The following diagram portrays the application’s four-tiered design. In general, graph files flow between the client and the S3 bucket, with Lambda functions identifying, placing, and retrieving files from the bucket as necessary. Movement of files in and out of the S3 bucket is usually guarded by calls to the RDS database (for example, Lambda functions might need to identify which file to retrieve or check if an analysis job is complete, both of which are determined through database queries).



Note that one of the endpoints, `/analysis/:graphid/:type`, is asynchronous. When called, this endpoint generates a graph analysis job of the specified type but does not wait for the job to finish before responding to the client. Graphs may be large, and graph analysis might take an arbitrarily-long time. To retrieve graph analysis results, clients must poll the `/results/:jobid` endpoint, which will return the current status of the job and the results if available.

Internally, graph analysis is performed by an additional lambda function that is triggered by the lambda function corresponding to the `/analysis/:graphid/:type` endpoint. Using a separate lambda function allows the original lambda function to exit before the analysis is complete (achieving asynchronous execution).

Database

The data tier of the application includes both an RDS database and an S3 bucket. The S3 bucket is used to store graph data (JSON) files and graph visualization (PNG) files, while the database is used to identify these files within the database and to track the status of graph analysis jobs. The RDS database "`graphapp`" contains two tables for storing graph information and graph analysis jobs.

The first table is named "`graphs`". Each row of this table represents one graph stored in the data tier of the application (note that the edges and vertices of the graph are not stored in the database, but in the S3 bucket). The table has three columns:

- `graphid`: the primary key of the graph
- `datafilekey`: the name of the graph data file in the S3 bucket
- `visualfilekey`: the name of the graph visualization file in the S3 bucket (if any)

A given graph stored in the application's data tier can only have one corresponding visualization. Initially, no visualization exists and the value for `visualfilekey` is NULL. If the user requests a visualization, the pre-existing visualization will be returned if one exists; otherwise, a new visualization will be produced and stored in the S3 bucket (and `visualfilekey` will no longer be NULL).

The following table lists the three columns of the "`graphs`" table and two example rows:

graphid *int, PK*	**datafilekey** *varchar(256)*	**visualfilekey** *varchar(256), nullable*
10001	e2969a6d-cfed-47e3-83c6-3d525652d83a.json	fef7a334-3eb4-4c2a-b4f1-00efef320bf7.png
10002	49495072-371f-4c01-899d-28168f657c8c.json	NULL

The second table is named "`jobs`". Each row of this table represents one graph analysis job stored in the data tier of the application. The table has four columns:

- `jobid`: the primary key of the analysis job
- `graphid`: the foreign key of the graph being analyzed
- `status`: the status of the job (either "processing", "completed", or "error")
- `resultsfilekey`: the name of the graph analysis file in the S3 bucket (if any)

A given job stored in the application's data tier can only have one corresponding analysis result file. Initially, no result file exists and the value for `resultsfilekey` is NULL. During the processing of the graph analysis job, the state of the job will eventually be updated to either "completed" or "error", after which the user can retrieve the results (a results JSON file containing the analysis data or error message, respectively).

The following table lists the four columns of the "`jobs`" table and two example rows:

jobid *int, PK*	**graphid** *int, FK*	**status** *varchar(256)*	**resultsfilekey** *varchar(256), nullable*
80001	10001	completed	fef7a334-3eb4-4c2a-b4f1-00efef320bf7.png
80002	10002	processing	NULL

Summarized REST API

The following REST API can be used to interact with the application. The provided Python3 client internally makes requests to these API endpoints.

For brevity, this section lists only a high-level description of each endpoint. **To view detailed information about the request and response formats of each endpoint, please refer to appendix A.**

POST `/graph`

Creates a new graph on the server-side that is instantiated using the graph data supplied in the request body.

GET `/graph/:graphid`

Returns the graph data file associated with "graphid" primary key provided in the request.

DELETE `/graph/:graphid`

Deletes all information associated with the graph corresponding to the "graphid" primary key provided in the request.

GET `/graphs`

Returns all graphs in the "graphs" table in the application's RDS database.

GET `/jobs`

Returns all jobs in the "jobs" table in the application's RDS database.

DELETE `/jobs`

Deletes all jobs stored in the “jobs” table in the application’s RDS database and all corresponding results files from the S3 bucket.

GET `/visual/:graphid`

Returns the graph visualization file associated with “graphid” primary key provided in the request.

GET `/random/:type`

Returns a graph data file containing vertices and edges that compose a randomly-generated graph of the specified type.

GET `/analysis/:graphid/:type`

Starts an asynchronous job that analyzes the given graph in the way specified by the request.

GET `/results/:jobid`

Retrieves the results of the specified graph analysis job, if available, and reports on the job’s status.

Appendix A: Full REST API

POST /graph

Description

- Creates a new graph on the server-side that is instantiated using the graph data supplied in the request body.
- When called, a new row representing the graph is created in the RDS database using the provided filename. The provided graph data file is stored under a new, unique name in the S3 bucket.

Request

- Method: **POST**
- Parameters: None
- Body: JSON-serialized data containing the bytes of the graph data file itself. For example, the following is a potential body:

```json
{
      "data": "KMTANCjExCjEyCjEzCjE0CjE1CjE2CjE"
}
```

Response

- Success
  - Returned when the request was appropriately formed and the server generated the associated database row and S3 bucket item successfully.
  - Status code: **200 OK**
  - Body: JSON-serialized data containing the message "success" and the primary key of the new graph row in the RDS database. This "graphid" primary key is used to refer to this graph in calls to other API endpoints. For example, the following is a potential body:

```json
{
   "message": "success",
   "graphid": 10001
}
```

- Client-side failure
  - Returned when the request was malformed, such as due to not having the appropriate items in the JSON-serialized body or the described graph being invalid (e.g. an edge references a non-existent vertex).
  - Status code: **400 Bad Request**
  - Body: JSON-serialized data containing a message explaining the first problem identified by the server when parsing the request and -1 for the graphid. For example, the following is a potential body:

```json
{
   "message": "no data key provided in the body",
   "graphid": -1
}
```

- Server-side failure
  - Returned when the request was appropriately formed but the server encountered an error when trying to create the associated database row and S3 bucket item.
  - Status code: **500 Internal Server Error**
  - Body: JSON-serialized data containing the error message raised on the server side and -1 for the graphid. For example, the following is a potential body:

```json
{
   “message”: “cannot use identifier before it is defined...”,
   “graphid”: -1
}
```

GET /graph/:graphid

Description

- Returns the graph data file associated with “graphid” primary key provided in the request.
- When called, the primary key is used to query the RDS database to identify the graph data file in the S3 bucket. The file data is then returned as JSON-serialized body data in the response.

Request

- Method: **GET**
- Parameters:
  - *graphid*, a required URL parameter specifying the graph to retrieve
- Body: None

Response

- Success
  - Returned when the request was appropriately formed, the graphid existed in the database, and the graph data file bytes could be retrieved.
  - Status code: **200 OK**
  - Body: JSON-serialized data containing the message “success” and the bytes of the graph data file. For example, the following is a potential body:

```json
{
   “message”: “success”,
   “data”: “KMTANCjExCjEyCjEzCjE0CjE1CjE2CjE”
}
```

- Invalid parameter
  - Returned when the request contained a value for the graphid parameter that did not correspond to any graph in the RDS database.
  - Status code: **404 Not Found**
  - Body: JSON-serialized data containing a message explaining that the graphid did not exist in the database and an empty string for the file data. For example, the following is a potential body:

```json
{
   “message”: “graphid does not exist in the database”,
   “data”: “”
}
```

- Server-side failure
  - Returned when the request was appropriately formed but the server encountered an error when trying to query the RDS database and retrieve the graph data file from the S3 bucket.
  - Status code: **500 Internal Server Error**
  - Body: JSON-serialized data containing the error message raised on the server side and an empty string for the file data. For example, the following is a potential body:

```json
{
   “message”: “cannot use identifier before it is defined...”,
```

```
    "data": ""
}
```

DELETE /graph/:graphid

Description

- Deletes all information associated with the graph corresponding to the "graphid" primary key provided in the request.
- When called, the primary key is used to query the RDS database to identify the graph data file and visualization file in the S3 bucket. Both files, if they exist, are deleted from the S3 bucket, and the graph row in the RDS database is also deleted.

Request

- Method: **DELETE**
- Parameters:
  - *graphid*, a required URL parameter specifying the graph to delete
- Body: None

Response

- Success
  - Returned when the request was appropriately formed, the graphid existed in the database, and the graph data was successfully deleted.
  - Status code: **200 OK**
  - Body: JSON-serialized data containing the message "success". For example, the following is a potential body:

```json
{
    "message": "success"
}
```

- Invalid parameter
  - Returned when the request contained a value for the graphid parameter that did not correspond to any graph in the RDS database.
  - Status code: **404 Not Found**
  - Body: JSON-serialized data containing a message explaining that the graphid did not exist in the database. For example, the following is a potential body:

```json
{
    "message": "graphid does not exist in the database"
}
```

- Server-side failure
  - Returned when the request was appropriately formed but the server encountered an error when trying to query the RDS database and identify the graph files in the S3 bucket.
  - Status code: **500 Internal Server Error**
  - Body: JSON-serialized data containing the error message raised on the server side. For example, the following is a potential body:

```json
{
    "message": "cannot use identifier before it is defined..."
}
```

GET /graphs

Description

- Returns all graphs in the “graphs” table in the application’s RDS database.
- When called, the RDS database is queried to collect all rows of the “graphs” table. The rows are then returned as JSON-serialized body data in the response.

Request

- Method: GET
- Parameters: None
- Body: None

Response

- Success
  - Returned when the request was appropriately formed and the graph rows could be retrieved from the database.
  - Status code: 200 OK
  - Body: JSON-serialized data containing the message “success” and the contents of each row of the “graphs” table. For example, the following is a potential body:

```json
{
   “message”: “success”,
   “data”: [
        {
             “graphid”: 10001,
             “datafilekey”: “e2969a6d-cfed-47e.json”
             “visualfilekey”: “fef7a334-3eb4b4f1-00.png”
        }
   ]
}
```

- Server-side failure
  - Returned when the request was appropriately formed but the server encountered an error when trying to query the RDS database for the graph rows.
  - Status code: 500 Internal Server Error
  - Body: JSON-serialized data containing the error message raised on the server side and an empty string for the graph rows data. For example, the following is a potential body:

```json
{
   “message”: “cannot use identifier before it is defined...”,
   “data”: “”
}
```

GET /jobs

Description

- Returns all jobs in the “jobs” table in the application’s RDS database.
- When called, the RDS database is queried to collect all rows of the “jobs” table. The rows are then returned as JSON-serialized body data in the response.

Request

- Method: GET
- Parameters: None
- Body: None

Response

- Success
  - Returned when the request was appropriately formed and the jobs rows could be retrieved from the database.
  - Status code: **200 OK**
  - Body: JSON-serialized data containing the message “success” and the contents of each row of the “jobs” table. For example, the following is a potential body:

```json
{
   “message”: “success”,
   “data”: [
         {
               “jobid”: 80001,
               “graphid”: 10001,
               “status”: “error”
               “resultsfilekey”: “e2969a6d-cfed-47e.json”
         }
   ]
}
```

- Server-side failure
  - Returned when the request was appropriately formed but the server encountered an error when trying to query the RDS database for the job rows.
  - Status code: **500 Internal Server Error**
  - Body: JSON-serialized data containing the error message raised on the server side and an empty string for the job rows data. For example, the following is a potential body:

```json
{
   “message”: “cannot use identifier before it is defined...”,
   “data”: “”
}
```

DELETE /jobs

Description

- Deletes all jobs stored in the “jobs” table in the application’s RDS database and all corresponding results files from the S3 bucket.
- When called, the RDS database is queried to delete all rows of the “jobs” table. For each job row present in the database, the corresponding results file is deleted from the S3 bucket if it exists.

Request

- Method: **DELETE**
- Parameters: None
- Body: None

Response

- Success
  - Returned when the request was appropriately formed and all jobs were successfully deleted from the database.
  - Status code: **200 OK**
  - Body: JSON-serialized data containing the message “success”. For example, the following is a potential body:

```json
{
   “message”: “success”
}
```

- Server-side failure
  - Returned when the request was appropriately formed but the server encountered an error when trying to delete all jobs from the database.
  - Status code: **500 Internal Server Error**
  - Body: JSON-serialized data containing the error message raised on the server side. For example, the following is a potential body:

```json
{
   “message”: “cannot use identifier before it is defined...”
}
```

GET /visual/:graphid

Description

- Returns the graph visualization file associated with “graphid” primary key provided in the request.
- When called, the primary key is used to query the RDS database to identify the graph visualization file in the S3 bucket, if any. If the visualization file does not already exist, then a PNG file containing a depiction of the graph is generated. The visualization file data is then returned as JSON-serialized body data in the response.

Request

- Method: **GET**
- Parameters:
  - *graphid*, a required URL parameter specifying the graph to visualize
- Body: None

Response

- Success
  - Returned when the request was appropriately formed, the graphid existed in the database, and the graph visualization file bytes could be retrieved.
  - Status code: **200 OK**
  - Body: JSON-serialized data containing the message “success” and the bytes of the graph visualization file. For example, the following is a potential body:

```json
{
   “message”: “success”,
   “data”: “KMTANCjExCjEyCjEzCjE0CjE1CjE2CjE”,
}
```

- Invalid parameter
  - Returned when the request contained a value for the graphid parameter that did not correspond to any graph in the RDS database.
  - Status code: **404 Not Found**
  - Body: JSON-serialized data containing a message explaining that the graphid did not exist in the database and an empty string for the file data. For example, the following is a potential body:

```json
{
   “message”: “graphid does not exist in the database”,
   “data”: “”,
}
```

- Server-side failure
  - Returned when the request was appropriately formed but the server encountered an error when trying to query the RDS database, retrieve the graph visualization file from the S3 bucket, and/or generate the visualization.
  - Status code: **500 Internal Server Error**
  - Body: JSON-serialized data containing the error message raised on the server side and an empty string for the file data. For example, the following is a potential body:

```json
{
   "message": "cannot use identifier before it is defined...",
   "data": "",
}
```

GET /random/:type

Description

- Returns a graph data file containing vertices and edges that compose a randomly-generated graph of the specified type.
- When called, a random graph is generated based on the type parameter (and optional vertex and edge counts) provided in the request. The generated graph is then returned in the format of a JSON-serialized graph data file. Additionally, the graph is saved in the database and S3 bucket.

Request

- Method: **GET**
- Parameters:
  - *type*, a required URL parameter specifying the type of graph to generated
    - Valid types include "any", "connected", "complete", "acyclic", "tree", and "bipartite"
  - *vertices*, an optional query parameter specifying number of vertices to generate
  - *edges*, an optional query parameter specifying number of edges to generate
    - Note: if "edges" is provided in the query string, then "vertices" must also be provided; however, "vertices" can be provided without specifying "edges"
- Body: None

Response

- Success
  - Returned when the request was appropriately formed and the random graph was generated and saved successfully.
  - Status code: **200 OK**
  - Body: JSON-serialized data containing the message "success", the bytes of the randomly-generated graph data file, and the graphid of the new graph row. For example, the following is a potential body:

```json
{
   "message": "success",
   "graphid": 10001,
   "data": "KMTANCjExCjEyCjEzCjE0CjE1CjE2CjE"
}
```

- Client-side failure
  - Returned when the request specified a type parameter that was not one of the supported types or when the requested number of vertices and edges were such that no valid graphs of the given type existed.
  - Status code: **400 Bad Request**

- Body: JSON-serialized data containing a message explaining the first problem identified by the server when parsing the request, an empty string for the graph data, and -1 for the graphid. For example, the following is a potential body:

```json
{
   “message”: “graph type is invalid”,
   “graphid”: -1,
   “data”: “”
}
```

- Server-side failure
  - Returned when the request was appropriately formed but the server encountered an error when trying to generate and save the graph.
  - Status code: **500 Internal Server Error**
  - Body: JSON-serialized data containing the error message raised on the server side, an empty strings for the graph data, and -1 for the graphid. For example, the following is a potential body:

```json
{
   “message”: “cannot use identifier before it is defined...”,
   “graphid”: -1,
   “data”: “”
}
```

GET `/analysis/:graphid/:type`

Description

- Starts an asynchronous job that analyzes the given graph in the way specified by the request.
- When called, the primary key is used to query the RDS database to identify the graph data file in the S3 bucket. Graph algorithms are then used to carry out the analysis requested by the client. When the analysis is completed, a results file is placed in the S3 bucket.
- Note: this endpoint is ***declarative***. Clients specify the type of analysis to perform but not the method of carrying out said analysis.
- Note: this endpoint is ***asynchronous***. To retrieve the results of an analysis job, the client must poll the `/results/:jobid` endpoint.

Request

- Method: **GET**
- Parameters:
  - *graphid*, a required URL parameter specifying the graph to analyze
  - *type*, a required URL parameter specifying the type of analysis to perform
    - Valid types include “is_connected”, “has_cycle”, “shortest_paths”, “reachable_nodes”, and “mst”
  - *root*, a sometimes-required query parameter identifying the root node for the analysis
    - Note: “root” is required when the analysis type is “shortest_paths” or “reachable_nodes” and ignored otherwise.
- Body: None

Response

- Success
  - Returned when the request was appropriately formed and the graphid existed in the database.
  - Status code: **200 OK**

- Body: JSON-serialized data containing the message “success” and the jobid of the created asynchronous analysis job. For example, the following is a potential body:

```json
{
   “message”: “success”,
   “jobid”: 80001
}
```

- Invalid parameter
  - Returned when the request contained a value for the graphid parameter that did not correspond to any graph in the RDS database.
  - Status code: **404 Not Found**
  - Body: JSON-serialized data containing a message explaining that the graphid did not exist in the database and -1 for the jobid. For example, the following is a potential body:

```json
{
   “message”: “graphid does not exist in the database”,
   “jobid”: -1
}
```

- Client-side failure
  - Returned when the request specified an invalid analysis type or did not supply a valid value for the optional “root” query parameter when the analysis type was “shortest_paths” or “reachable_nodes”.
  - Status code: **400 Bad Request**
  - Body: JSON-serialized data containing the first problem identified by the server when parsing the request and -1 for the jobid. For example, the following is a potential body:

```json
{
   “message”: “analysis type is invalid”,
   “jobid”: -1
}
```

- Server-side failure
  - Returned when the request was appropriately formed but the server encountered an error when trying to query the RDS database, retrieve the graph data file from the S3 bucket, and create the analysis job.
  - Status code: **500 Internal Server Error**
  - Body: JSON-serialized data containing the error message raised on the server side and -1 for the jobid. For example, the following is a potential body:

```json
{
   “message”: “cannot use identifier before it is defined...”,
   “jobid”: -1
}
```

GET `/results/:jobid`

Description

- Retrieves the results of the specified graph analysis job, if available, and reports on the job’s status.
- When called, the job primary key is used to query the RDS database for the job’s current status. If available, the results file containing the analysis of corresponding graph is retrieved from the S3 bucket and returned.

Request

- Method: **GET**
- Parameters:
  - *jobid*, a required URL parameter specifying the job from which to get the results
- Body: None

Response

- Success
  - Returned when the request was appropriately formed, the jobid existed in the database, and the results of the analysis was available.
  - Status code: **200 OK**
  - Body: JSON-serialized data containing the message “success” and the bytes of the analysis results file. For example, the following is a potential body:

```json
{
   “message”: “success”,
   “data”: “KMTANCjExCjEyCjEzCjE0CjE1CjE2CjE”
}
```

- Results not ready
  - Returned when the requested job does not yet have results to return to the client.
  - Status code: **481**
  - Body: JSON-serialized data containing a message explaining that the results for the jobid are not yet available and an empty string for the results file data. For example, the following is a potential body:

```json
{
   “message”: “results for jobid not yet available”,
   “data”: “”
}
```

- Analysis error
  - Returned when the requested job terminated with a status of “error".
  - Status code: **482**
  - Body: JSON-serialized data containing a message explaining that the cause of the error is unknown and an empty string for the results file data. For example, the following is a potential body:

```json
{
   “message”: “jobid terminated due to an unknown error”,
   “data”: “”
}
```

- Invalid parameter
  - Returned when the request contained a value for the jobid parameter that did not correspond to any job in the RDS database.
  - Status code: **404 Not Found**
  - Body: JSON-serialized data containing a message explaining that the jobid did not exist in the database and an empty string for the results file data. For example, the following is a potential body:

```json
{
   “message”: “jobid does not exist in the database”,
   “data”: “”
}
```

- Server-side failure

- Returned when the request was appropriately formed but the server encountered an error when trying to query the RDS database or retrieve the results file from the S3 bucket.
- Status code: **500 Internal Server Error**
- Body: JSON-serialized data containing the error message raised on the server side and an empty string for the results file data. For example, the following is a potential body:

```json
{
   “message”: “cannot use identifier before it is defined...”,
   “data”: “”
}
```

Appendix B: Analysis Results Files

Graph analysis results files are returned by the `/results/:jobid` API endpoint when results become available. All types of graph analysis yield results files in the following JSON format, although the data corresponding to the "data" varies by type:

```json
{
      "type": "...",   // string containing analysis type
      "data": ...      // data containing analysis results
}
```

The remainder of this appendix lists in the format of the results returned by each type of graph analysis supported by the application.

CONNECTEDNESS

Description

- The "is_connected" analysis type returns a boolean indicating whether or not the graph is connected. That is, the boolean is true only when all nodes in the graph are reachable by all other nodes.

Results Format

- Results data is returned as a single boolean value. For example, the following is a possible results file:

```json
{
      "type": "is_connected",
      "data": false
}
```

CYCLE DETECTION

Description

- The "has_cycle" analysis type returns the first cycle found by the cycle detection algorithm or the boolean value "false" when no cycle exists.

Results Format

- Results data is returned as a single boolean value when the graph is acyclic or as a list of vertex identifiers composing the identified cycle. For example, the following is a possible results file:

```json
{
      "type": "has_cycle",
      "data": [5, -1, 3, -8, 5]
}
```

SHORTEST PATHS

Description

- The "shortest_paths" analysis type returns the root used when computing the shortest paths and the shortest distances and full shortest paths from the root to every vertex in the graph.

Results Format

- Results data is returned as a dictionary with key “root” to identify the root node and “paths” to identify the shortest paths.
- Within “paths”, each key is a vertex of the graph and the corresponding value is an array of [shortest distance, shortest path list]. When a node is not reachable, the shortest distance is -1 and the shortest path list is empty.
- For example, the following is a possible results file:

```json
{
      “type”: “shortest_paths”,
      “data”: {
            “root”: -1,
            “paths”: {
                  “5”: [100.1, [-1, 3, -8, 5]],
                  “2”: [-1, None],
                  “3”: [50, [-1, 3]],
                  “-8”: [87.3, [-1, 3, -8]],
                  “-1”: [0, [-1]],
            },
      }
}
```

REACHABILITY

Description

- The “reachable_nodes” analysis type returns the root used when detecting reachability and the list of vertices reachable from the root vertex.

Results Format

- Results data is returned as a dictionary with key “root” to identify the root node and “reachable” to identify the reachable vertices. For example, the following is a possible results file:

```json
{
      “type”: “reachable_nodes”,
      “data”: {
            “root”: 7,
            “reachable”: [-1, 3, -8, 5]
      }
}
```

MINIMUM SPANNING TREES

Description

- The “mst” analysis type returns the boolean “false” when the graph is not connected or the edges composing a minimum spanning tree of the graph when the graph is connected.

Results Format

- Results data is returned as a single boolean value when the graph is not connected or as a list of edges composing a minimum spanning tree. For example, the following is a possible results file:

```json
{
      “type”: “mst”,
      “data”: [[-1, 3, 5.4], [-1, 5, 10.33], [5, 4, 9]]
}
```