The University of New South Wales - COMP9312 - 24T2 - **Data Analytics for Graphs**

[Project Homepage](#) / **Q1**

Q1 (10 marks)

Design a solution to answer the shortest distance query in large graphs.

Problem Statement

In this question, you need to design a solution to compute the shortest distance between two query vertices in a **large undirected unweighted graph**.

Background & Marking Criteria

Given an input graph, you are expected to preprocess the graph and build an index structure to speed up shortest distance queries for some arbitrary pairs of vertices. You can do anything you like to preprocess the graph data.

The marking tutors will evaluate the goodness of your solution. A good solution should consider query processing time, index space, and index construction time. For example, you may implement an online algorithm based on BFS. There is no index space and indexing time cost, but the query efficiency is low. In another extreme example, you may just precompute the shortest distance for all possible pairs of query vertices. The query time is optimal, but the index space is unacceptable since we are dealing with big graphs. By big graphs, we mean graphs with at least millions of vertices. You can think about how much memory (RAM) we need to store the shortest distance for all pairs of vertices if there are around 10 million vertices. Therefore, the goal is to **improve the query efficiency** while **the additional space usage is acceptable** for million-scale graphs in a commercial machine. The index construction efficiency depends on the structure of your index. Try to design an efficient algorithm to compute your index.

Online VS Offline

We focus on the static graphs without any updates. Normally, the offline (index-based) method is preferable in this setting. Even though preprocessing the graph takes much more time compared with a single shortest distance query processing, there may exist a considerable amount of queries, which can benefit from the precomputed index. If you have no idea about how to design an index-based solution, you should at least try to implement a good online (do not preprocess the graph) algorithm.

In addition to writing out the code, it is also important to understand what you have achieved. You need to provide a **document** to describe your solution. The document helps us understand your code and lets us know you have already got an idea to solve the problem at least, which may provide you with some marks even if your code cannot run correctly. The document can be in any format. The content should include but not be limited to your **design ideas, theoretical analysis for index space complexity, query time complexity, and indexing time complexity**. Example figures (e.g., index structure) will be helpful if you design some complex solutions. You are also welcome to discuss multiple potential solutions/baselines and demonstrate why your solution is the best.

Doing this Project

Open the code template file Q1.ipynb and make a copy of the file in your own Google Drive. You need to implement a class named `ShortestDistanceQuery` with at least two functions, `preprocess()` and `query()`. Below is the code template for `ShortestDistanceQuery` shown in Q1.ipynb. You can also find the input graph data structure defined in the `UndirectedUnweightedGraph` class and an example of how we test your code. The file is running on the Google Colab platform, which has already integrated the Python environment. As shown in our tutorials, you can directly write and run your code without any configuration. You can directly add any description for your solution and theoretical analysis in your Q1.ipynb instead of creating a separate PDF report. Ask your tutor if you have any question to run the code.

```
################################################################
# You can import any Python Standard Library modules~
from collections import deque
################################################################
```

```
class ShortestDistanceQuery(object):
    def __init__(self, G):
        self.G = G
        ############################################################
        # TODO: You may add some index data structures here~
        # analyze the space usage/complexity of the index (all
        ############################################################
        self.preprocess(G)

    def preprocess(self, G):
        ############################################################
        # TODO: Your code here~
        # precompute some index data structures for G and use t
        # analyze the time complexity of preprocess()
        ############################################################
        return

    def query(self, vertex_u, vertex_v):
        shortest_distance = 0
        ############################################################
        # TODO: Your code here~
        # Input: vertex_u, vertex_v
        # Output: the shortest distance between vertex_u and ve
        # analyze the time complexity of query()
        # Initialize a queue for BFS
        ############################################################
        return shortest_distance

    ############################################################
    # You can define any auxiliary functions~
    ############################################################
```

Required Files

Compress all related files for Q1 (`Q1.ipynb` and an optional `Q1.pdf` if you want to create a document in a separate file) into `Q1.zip`, and submit it on Moodle.

Towards the Full-Mark Solution

Full marks will be given for an efficient implementation of **a minimal two-hop labeling index** with the correct theoretical analysis or **other methods which are proven not worse than the two-hop index**. The two-hop labeling index has been discussed for reachability queries in Topic 2.1. The challenge is to extend the idea for the shortest distance queries.

Other Marking Criteria

a. Only correctly implementing an online algorithm (without any valuable code to preprocess the graph) is worth at most 4 points.
b. Any reasonable algorithm is acceptable as long as you provide a detailed description and analysis in the report. The mark is provided on a case-by-case basis. Usually, a reasonable and correct index-based solution is worth at least 6 points.

Notes

a. A detailed report is required to describe your algorithm, which at least analyses the time complexity of query processing and index construction, and the space complexity of the index. You can directly write them in `Q1.ipynb` or in a separate `Q1.pdf`.
b. You can add additional variables and functions in the `ShortestDistanceQuery`. Do not change the input/output format in the code template.
c. We may use a different and much larger dataset to test your algorithm.

END OF QUESTION