TSP

1. Input

`graph`: an `Object` of format `{vertexName: [[neighborName, neighborDistance], ...], ...}`, you may refer to `testcase.json` as an example. Note: the expected $|V| < 256, |E| < 1024$.
`selectedVertices`: an `Array` of unique vertices of format `[vertexName1, vertexName2, ...]`. Note: all vertices are in `graph`. Note: the expected $|selectedVertices| < \sqrt{|V|}$.

2. Output

`orderedSelectedVertices`: a `Array` of format `[vertexName1, vertexName2, ...]` sorted according to the approximated traveling salesman tour.

3. Libraries and functions you may use

- Priority Queue: [js-priority-queue]
- Dijkstra shortest path (as below)

```
// Dijkstra shortest path from src to tgt on triangulationGraph
function getTourPath(triangulationGraph, src, tgt) {
    // TODO: check it is a valid triangulation
    const prevDict = {};
    const queue = new PriorityQueue({ comparator: function(a, b) { return -
    // Init queue from src
    queue.queue([src, 0, null]);
    // Dijkstra update
    while(queue.length !== 0) {
        const [tempNode, tempDist, prevNode] = queue.dequeue();
        if (prevDict.hasOwnProperty(tempNode)) { // already visited.
            continue;
        }
        prevDict[tempNode] = prevNode;
        if (tempNode.normalize() === tgt.normalize()) { // get to tgt, done
            break;
        }

        for (const [nextNode, dist] of triangulationGraph[tempNode]) {
```

```
            if (prevDict.hasOwnProperty(nextNode)) { // already visited.
                continue;
            }
            queue.queue([nextNode, tempDist + dist, tempNode])
        };
    }
    // get path back from tgt
    let tempNode = tgt;
    const path = [];
    while (tempNode !== null) {
        path.unshift(tempNode);
        tempNode = prevDict[tempNode];
    };
    return path;
}
```

4. High-level Algorithm

1. Compute pair-wise shortest distance on `selectedVertices`. (If necessary, you may modify `getTourPath` to also return distances.) Then, you can form a complete graph `selectedGraph` on `selectedVertices`.
2. Compute minimum spanning tree `selected tree` on `selectedGraph`. Then you may arbitrarily pick a vertex as the `root`.
3. Return a pre-order walk on `selected tree`, and append `root` at the end.

5. Reference

https://www.geeksforgeeks.org/travelling-salesman-problem-set-2-approximate-using-mst/