# ComS 3110: Homework 4
# Due: November 19
# Points: 80

## I) Introduction

In this assignment, you will be writing a Java program to find shortest paths in weighted directed graphs with non-negative edge weights. Your task is to implement Dijkstra's algorithm. You will also need to implement a priority queue using a binary heap that supports an asymptotically efficient decrease key operation.

## II) Details

In the provided zip file, you will find a `Graph` interface and a `PurePriorityQueue` interface provided for you. There is also a `Tuple` class that you are free to use, but should not be modified. You should not change these interfaces.

You will also find two classes implementing these interfaces, `DirectedGraph` and `BinaryMinHeap` respectively. You should not change the signatures of any given methods.

### II. a) DirectedGraph

The `DirectedGraph` class implements `Graph` as an adjacency list. Many of the basic methods are implemented for you. You are required to implement the `dijkstras` method yourself.

The signature of the method used for finding single-source shortest paths is `public Tuple<Map<V, Double>, Map<V, V>> dijkstras(V source, Map<Tuple<V, V>, Double> weights)`.

- The first argument, `source`, is the starting vertex. This method should compute the shortest path from `source` to every vertex in the graph that can be reached from it.
- The second argument, `weights`, represents a mapping from edges to their weights. You may assume that the weights are non-negative, and you do not need to validate the input. As you will notice, a `DirectedGraph` itself does not hold edge weights, hence this argument. This allows for easily evaluating the same underlying graph using different edge weightings.
- The return type is a `Tuple`.
  - The first item in the tuple is a mapping from vertices to their distance from `source`.
  - The second item in the tuple is a mapping from vertices to their predecessors as determined by Dijkstra's algorithm. This can be used to reconstruct the shortest `source` to any reachable vertex.

Your implementation of Dijkstra's algorithm should run in $O(E \log V)$ time, assuming that the core `HashMap` and `HashSet` operations are $O(1)$. Your implementation should not add vertices to your queue multiple times, hence the `keyDecreased` method present in our priority queue interface.

Example usage:

```
// Initialize the graph.
Graph<String> g = new DirectedGraph<String>();
String v1 = "a";
```

```
String v2 = "b";
g.addVertex(v1);
g.addVertex(v2);
g.addEdge(v1, v2);

Map<Tuple<String,String>, Double> weights = new HashMap<Tuple<String,String>, Double>();
weights.add(Tuple.create(v1, v2), 2);

/*
   The first item in the tuple is a mapping:
   a -> 0, b -> 2
   The second item in the tuple is a mapping:
   b -> a
*/
var distsAndPreds = g.dijkstras(v1, weights);

// Add another edge.
g.addEdge(v2, v1);
// Add a weight for it.
weights.put(Tuple.create(v2,v1), 3);
/*
   The first item in the tuple is a mapping:
   a -> 3, b -> 0
   The second item in the tuple is a mapping:
   a -> b
*/
distsAndPreds = g.dijkstras(v2, weights);
```

## II. b) BinaryMinHeap

The `BinaryMinHeap` class implements `PurePriorityQueue` stored in an array. Notably, it includes a `keyDecreased` method suitable for usage in `Graph.dijkstras()`. The `add()`, `extractMin()`, and `keyDecreased()` methods should run in $O(\log n)$ time.

Because there is no constraint on the type of Object it may hold, the constructor takes a `Comparator` object to be used for comparing its contents.

Example usage:

```
HashMap<String, Integer> m = new HashMap<String, Integer>();

String x = "obj1";
int j = 11;
m.put(x, j);

String y = "obj2";
int k = 12;
m.put(y, k);

var q = new BinaryMinHeap<String>((l, r) -> Integer.compare(m.get(l), m.get(r)));
q.add(x);
q.add(y);
m.put(y, k-2);
q.keyDecreased(y);

System.out.println(q.extractMin()); // Prints "obj2".
```

## III) Submission Format

Your source code *must* belong to the `edu.iastate.coms3110.hw4` package. You should *not* alter or remove the public api of any file you've been given. Your submission *must* be a `zip` file containing your source (ending in `.java`) files and the folder structure corresponding to the package name. Do **not** submit your compiled (usually ending in `.class`) files.

## IV) Late Policy

The late policy for this homework is the same as for written assignments.

An assignment that is submitted one day late will get a penalty of 10%. An assignment that is submitted two days late will get a penalty of 20%. Assignments submitted after two days will not be graded and will get no points. For example, if an assignment is due on Friday by midnight, a submission on Saturday will be penalized by 10%, and a submission on Sunday will be penalized by 20%. A submission on Monday will get no points.