Q1.a -----

The map function takes a key, which is the position in the file, a *Text* value, which is the line of text, and a *Context* object, which is an abstraction that allows interacting with the Hadoop system. The function processes each line of the file line-by-line, splitting the line by the tab character, and saving the resultant values in an array. From this array, the source node and weight are extracted, and then written as a key-value pair of *IntWritables* by the Context object.

The reduce function takes a key, which corresponds to a distinct source node, and one or more iterable IntWritable values, which correspond to edge weights, as well as a Context object. First, the function retrieves an element from the values (weights), and initializes a **max** variable with a value of the element. Then, it iterates through the weights, replacing max with the current value if it is greater than max. Lastly, it writes the source node and max edge weight as a key-value pair of IntWritables.

Q1.b -----

```
def map(key, value, context):
  tkns = value.split("\t")

  src = tkns[0]
  tgt = tkns[1]

  context.write(src, tgt + ":1")
  context.write(tgt, src + ":0")

def reduce(key, values, context):
  inNodes = []
  outNodes = []

  for val in values:
    tkns = val.split(":")

    node = tkns[0]
    direc = tkns[1]

    if direc == 0: inNodes.append(node)
    else: outNodes.append(node)

  for inNode in inNodes:
    for outNode in outNodes:
      if inNode != outNode:
        context.write(inNode, outNode)
```

The map function takes a key (which is the position in the file), value (which is each line in the file), and Context object (which is used for writing to the Hadoop file system). First, we split the line and save the source and target nodes. Then, we write one (key, value) pair with the source as the key and the target concatenated with a ":1" as the value, identifying it as an outbound link. Next, we write a (key, value) pair with the target as the key and source concatenated with ":0" as the value, identifying it as an inbound link.

In the next phase of the computation (see Fig. 1), values are grouped by key.

The reduce function then takes a key, iterable collection of values, and Context object for writing to the filesystem. At this step, the key (which is a node) has a collection of <node:direction> pairs to work on. To determine the 2nd-degree out-neighbors, we first prepare arrays of inbound and outbound nodes from the values, by checking the direction value. When the direction value is a 0, it is an inbound link; otherwise, it is an outbound link. Next, for every inbound node, we write a (key, value) pair for every outbound node that does not equal itself.

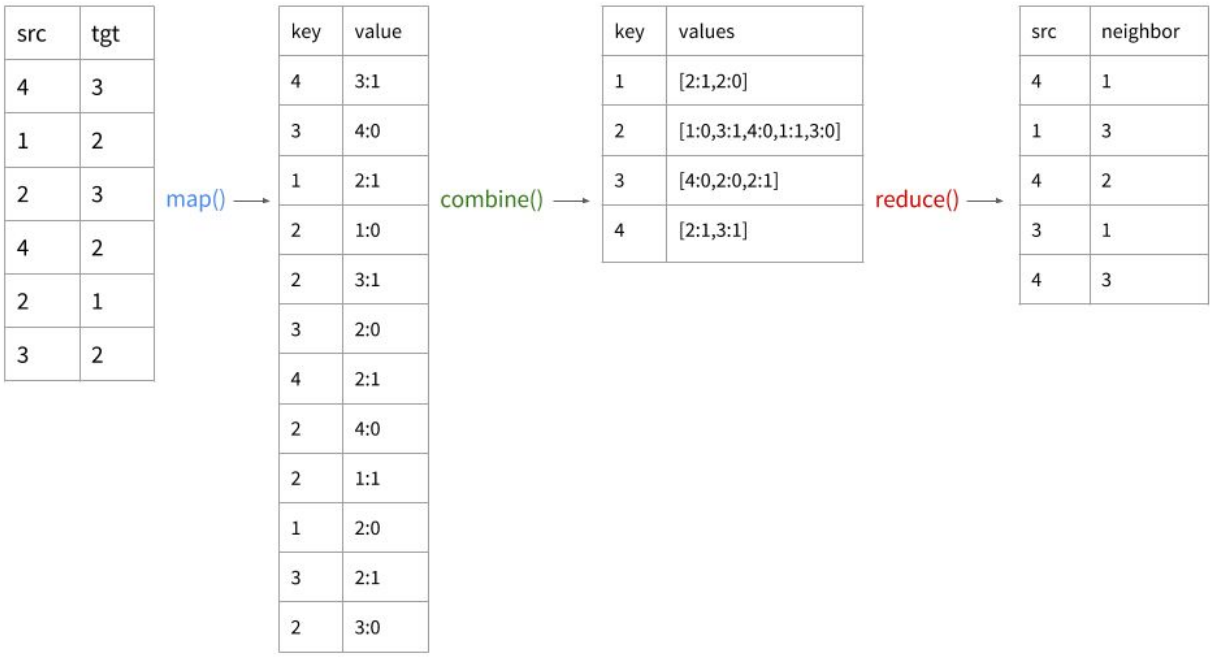

Fig. 1. Schematic of the input and output flows through the map, combine, and reduce phases of the computation.

For verification purposes, this solution was tested in Hadoop and performed as expected.