Q1.a

The *map* function takes a key, which is the position in the file, a *Text* value, which is the line of text (for example, “51<tab>117<tab>1” in the toy example (see figure), 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. In the toy example below, (51,1), (51,1), (51,3), (151, 51), (151,79), and (130,10) are written as (key, value)-pairs. After this, a combine step occurs where the values are aggregated by key.

The *reduce* function takes a key, which corresponds to a distinct source node, and an iterable collection of IntWritable values, which correspond to edge weights, as well as a Context object. In the example, (51, [1,1,3]) is one such set of inputs. 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. In the toy example below, (51,3), (151,79), and (130,10) are written as results.

src	*tgt*	*weight*
51	117	1
51	194	1
51	299	3
151	230	51
151	194	79
130	51	10

map() →

key	*value*
51	1
51	1
51	3
151	51
151	79
130	10

combine() →

key	*values*
51	[1,1,3]
151	[51,79]
130	[10]

reduce() ↓

key	*values*
51	3
151	79
130	10

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 a line in the file (for example, “4<tab>3”)), and Context object. 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. For example, (4,”3:1”), (3,”4:0”) would be written in the below example. In the combine phase (see below), values are grouped by key.

The *reduce* function then takes a key, an iterable collection of values, and a 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. For example, (1, [“2:1”,”2:0”]) in the example. 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 an (inNode, outNode) pair for every outbound node that does not equal itself. In the example, this produces (4,1), (1,3), (4,2), (3,1), and (4,3) as resultant (key, value)-pairs.