Task 1 Part A

The map function contains three inputs are key, value, and context:

```
public void map(Object key, Text value, Context context) throws IOException, InterruptedException {
    StringTokenizer itr = new StringTokenizer(value.toString(), "\t");
    itr.nextToken();
    word.set(itr.nextToken());
    String weight = itr.nextToken();
    context.write(word, new IntWritable(Integer.parseInt(weight)));
}
```

The mapper implementation processes each line at a time, where it obtains three tokens from the value input. The first one represents the Source and is skipped because it is unnecessary for this prompt. It uses the next token to set the Target value. Lastly, the Weight is set using the third token. Then the context input is used to map each Target with its Weight, such that the mapping is <Target, Weight>.

The reduce function contains the three inputs as well: key, values, context:

```
public void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException, InterruptedException {
    int max = 0;
    for (IntWritable val : values) {
        int tmp = val.get();
        if (tmp > max) {
            max = tmp;
        }
    }
    result.set(max);
    context.write(key, result);
}
```

The reduce implementation receives an iterable list of the weights for each target, and maps the largest weight to the target. For example, for the key 51 it will receive <1, 1, 3>, since 3 is the max, it will use context to write 51 and 3 together: <51, 3>.

Task 1 Part B:

```
function map (source, target) {
//for a given node, find all of the nodes whose source matches this node's target value
        for node in nodes
                if (node.target == source) {
                        directNeighbors += node;
        }
        context.write(node, directNeighbors);
}

function reduce (key, directNeighborsList) {
        for (neighbor in directNeighborsList) {
                context.write(key.target, neighbor.source);
        }
}
```

The map function evaluates each node and creates a list of nodes whose source value matches the node's target value. This key-value pair is then passed into the reduce function, where the reduce function will map the 2-hop neighbors by matching the key's target with the neighbor's source. The neighbor's target and the key's source should be the same to satisfy the 2-hop condition.