Part a)

First, the map splits each line by tabs. It then looks in the second column for the target node and the third column for the associated weight, which it interprets numerically. Essentially, it's just stripping out the source node, since that doesn't matter.

The reduce function then iterates through all the values from the map that have the same target name and finds the maximum weight. It's much the same as the word count, except with a max instead of sum.

Part b)

Mapper:

```
For each line{
if "Student" in index 0
     keyindex=2
     tag = 2
     data = index[1] //name
else
     keyindex=1
     tag = 1
     data = index[2] //dept name
}
```

The core strategy for the map is to clearly mark the dept line with a tag and make sure the key is correctly set for each line. This is done easily by the fact that all the student share a value in index 0.

Reducer:

```
For each unique key{

     Dept_line = find line with tag == 1
     for all other lines:

          Create list of [dept_num, name]
          append Dept_line[2] //dept name
```

```
        convert to string
        write [key, string] to context
```

The core strategy in the reduce function is to identify the line that is belonging to the dept information and append it to the student lines. The tutorial did this with a special comparator, but I would feel more comfortable doing it within the reduce function itself by searching through all the records with the same key as a first step. (I would still want to implement the key as a TaggedKey object so that I could store a tag in the key.) Once the dept is identified, it's a simple matter to append that line's data to the data from the student lines.