Part a

Map function:

Input: src tgt weight

Output: tgt as key and weight as value

Split each line in the graph file into tokens separated by a tab. Map function uses the second parameter as a key and third parameter as a value.

Example:

Input:

src	tgt	weight
100	10	3
110	10	3
200	10	1
150	130	30
110	130	67
10	101	15

Output:

<inboundEmail, weight>

<10, 3>

<10, 3>

<10, 1>

<130, 30>

<130, 67>

<101, 15>

Reduce Function:

Input: unique tgt as key and list of weight of this node's inbound edges

Output: tgt as key and min inbound edge weight as value

Input:

<10, <3, 3, 1>>

<130, <30,67>>

<101, <15>>

Output

<10, 1>

<130, 30>

<101, 15>

Part b

Mapper Function:

Use D_ID as the key. Combine "Student" or "Department" and student name or department name together as value.

Input of the algorithm:

Student, Alice, 1234

Student, Bob, 1234
Department, 1123, CSE
Department, 1234, CS
Student, Carol, 1123

Mapping function output:
<1234, S-Alice>
<1234, S-Bob>
<1123, D-CSE>
<1234, D-CS>
<1123, S-Carol>

Reducer Function:
Set keys to IntWritable and WritableComparator would normally handle the numerical ordering ascending order. For every key, first find the department name and then iterate over all student names to get the final <D_ID, Student_Name, D_Name> tuple.

Input of reducer:
<1234, <S-Alice, S-Bob, D-CS>>
<1123, <D-CSE, S-Carol>>

Output of reducer:
<1123, Carol, CSE>
<1234, Alice, CS>
<1234, Bob, CS>

Pseudo code:

```
Mapper (Object, Text, IntWritable, Text):
for line in file:
    if (line [0] == "Student")
        <key, value> ← <line[2], "S"+ "-" + line[1]>
    else
        <key, value> ← <line[1], "D" + "-" + line[2]>

Reducer (IntWritable, Text, IntWritable, Text):
for value in values:
    tokens [] = value.split("-")
    if (tokens [0] == 'D')    D_Name = tokens[1]
    if (tokens [0] == 'S')    context.write(key, tokens[1], D_Name)
```