Mandatory assignment 3

INF 2220, Fall 2011

Due: 15.11.2011

Published Date: 25.10.2011

Overview

In this assignment, you are going to simulate a network of machines which share data. Each machine on the network is identified by an integer. Each machine has a list of data elements it keeps, and is allocated with a capacity for storing data elements. The data elements are represented by integers as well. A machine can transmit data elements to another machine on the network if they are directly connected. Otherwise, there should exist a route through which they are connected. We assume that the connection between two machines is *bidirectional.* To transmit an element between two computers, it takes t milliseconds and c øre. Concurrent transmissions are allowed in the network. Each connection between two computers on the network is assumed to have *unlimited* bandwith, and therefore multiple elements can be transmitted at the same time.

Request and store data

Each machine on the network can *request* data as well as store data which is available for requests. Each machine is initialized with a set of data elements which are the original copies. A *successfully* served request means that for *each* of the requested data elements, there exists a route to reach it from the requesting machine. If a request is *successfully* served, the requesting machine will keep a copy of each of the requested elements locally, except for those are found locally in the machine. Each machine should know whether a data element is its own *original copy,* or a *local copy* of an element from another machine. Since each machine can only store *at most* `capacity`[1] number of elements in total, the element which is the *oldest local copy* has to be removed before any new local copy can be added if the capacity limit has been reached. NO *original* copy should be removed.

Requests

A *request* from a machine includes

- the identity of the machine from which the request originates,

[1] `capacity` is the maximum number of data elements each machine can store

- a search preference,
- data ownership preference, and
- a list of requested data elements.

With a search preference, one can choose to minimize the time (T) taken, or the cost (C) of a request, or both (B). The user can also specify the type of requested data elements he prefers: either an original copy (O) or any copy (A) that can be found.

A request example

An example of a request is "`1:C:O:57 15 296 83`" which means that `Machine 1` wants to use the lowest cost to get the original copy of data elements `57 15 296 83`. Your program will then print the result of the request. The result includes a list of machine identities for each requested data element. This list of identities describes the route from the machine in which the element is found to the requesting machine. The route should be optimal according to the search preference indicated by the user. The result also includes the time and the cost needed for each requested element, and the total time and cost taken for all the requested elements. The output of your program should be in this form

```
57: 5->2->1 (t=7, c=13)
15: 4->10->3->7->9->1 (t=11, c=8)
296: 1 (t=0, c=0)
83: 9->1 (t=2, cost=1)
Total time: 11, total cost = 22
```

The output shows that data element `57` is found in Machine `5` and is transmitted to the requesting machine (Machine `1` in this case) through Machine `2`. The time taken for a route is the sum of the time taken by all its segments. Similarly, the cost of a route is the sum of the cost of all its segments. Since the search preference in this example is to minimize the cost needed, the route `5->2->1` must be optimal in minimizing the cost for getting element `57`. Note that the data element `296` is found in the requesting machine, therefore, it does not take any time or cost to transmitt. Since concurrent transmissions are allowed, the total time taken for transmitting all the requested data is the *maximum* of all data elements' transmission time. In the example above, it takes the longest time to transmit data element `15`, and therefore the total time taken for the overall request is `11`. In addition, the total cost is the *sum* of the cost taken for all the elements, which is `22` in our example. Note also that although the routes for element `15` and element `83` overlap at the segment `9->1`, the cost of these two elements are considered separately. In case one or more requested elements are not found in any machine on the network, your program should show a list of unavailable elements to the user.

It works similarly if the search preference is to minimize the time. That means the route for getting each of the requested elements should be optimal in minimizing the `time` taken instead of the cost needed.

Minimizing both cost and time

If the user would like to minimize both the time taken and the cost needed, we prioritize handling the request as cheap as possible regarding the cost, while also selecting the quickest routes without raising the cost. (Note: It is possible to handle the request the other way around, but we do not consider that here.) For our example, `22` is the minimum cost for requesting elements `57 15 296 83`. To shorten the total time taken for the request, we have to find a route for each requested element such that it is optimal in minimizing the time taken without raising the cost. In our example, we can see that the route for getting element `15` is critical because it takes the longest. As soon as we can find another route which takes shorter time than the current one without raising the cost (i.e., `8`), and it takes the least time with the given cost, then we have shortened the time taken for the request. For instance, if there exists two other alternatives for getting element `15` in the previous example:

```
15: 4->8->1 (t=8, c=8)
15: 6->1 (t=3, c=18)
```

We choose the first alternative instead of the second one even though the time taken is longer because the second alternative will raise the total cost of the request. If the time taken for the new route for getting element `15` is still higher than the time needed for other elements, then we have managed to handle the request as quickly as possible without paying higher cost. Otherwise, we have to repeat the same operation for the element which now takes the longest time.

Route update

Let's revisit the example above. Assume that the request `1:C:O:57 15 296 83` is successfully served. The user inputs another request `1:C:A:15`. Since a copy of data elements `57 15 296 83` is now stored locally in Machine `1`, and the user would like to have *any* copy (the search preference is `A`) of the data element `15`, the result of this request is

```
15: 1 (t=0, c=0)
Total time: 0, total cost = 0
```

However, if the request is `1:C:O:15` which indicates that the user wants to have the *original* copy of element `15`, the result of the request will be

```
15: 4->10->3->7->9->1 (t=11, c=8)
Total time: 11, total cost = 8
```

In addition, any other machine which has to go through Machine `1` for the *original copy* of data elements `57 15 296 83` can now get a *local copy* of these elements from Machine `1`.

Your Tasks

In this assignment, you have to implement your program in JAVA. You have to decide which data structures should be used in your implementation. Note that your choice of data structures *may* affect the efficiency of processing requests.

1. You are given two files: `config.txt` and `data.txt`, which you can find on the course website. File `config.txt` is the configuration of a network which is a sequence of numbers describing connections of each pair of machines on the network. Each connection entry in the configuration file `config.txt` is in the following form:

   ```
   machine_id machine_id time cost
   ```

   After your program has built the network, it should read the file `data.txt` which lists the data elements originally stored in each machine on the network. Each entry in the file `data.txt` looks like

   ```
   machine_id capacity list_of_data_elements
   ```

2. After configuring the network, your program should allow the user to enter a request in the form

   ```
   machine_id:search_preference:ownership_preference:  list_of_data_elements
   ```

   Remember that the three options for `search_preference` are `{ T, C, B }`, while the two options for `ownership_preference` are `{ O, A }`. Your program should respond to the user by printing on the screen the output we discussed earlier. Note that since the list of data elements stored in a machine may change according to different requests, the output of the same request may be different from time to time. In addition, your program should print an error message if the identity of the requesting machine does not exist in the network.

Deliver to

The assignment should be carried out individually and delivered to the teaching assistant responsible through `https://devilry.ifi.uio.no/`.

How to deliver

- You have to implement the assignment in JAVA.
- Your implementation should compile on the LINUX machines in the University, and run either with the command: `java Oblig3 config.txt data.txt` , or you have to specify in the `README` file exactly what commands should be used to compile and run the program.
- The delivery must only be in one file: either a `.tgz` or `.zip` archive (with one of those two file extensions). Follow the steps below to genenrate the archive:

1. `cp -R AssignmentFolder myusername`
2. `find myusername -name \*.class -delete`
3. `tar cvzf myusername.tgz myusername`

That is, first you copy the contents of `AssignmentFolder` to a folder with the same name as your username, then delete all compiled files from the `myusername` folder and pack the folder in a gzipped tar. For a `zip`-archive, replace the `tar`-command in step 3 with: `zip -r myusername.zip myusername`

- Your archive should contain
  - The source file(s) of your implementation.
  - A report, named either `report.txt` or `report.pdf` , in which you should argue your choice of the data structures you used in your implementation and the corresponding complexity. You should also discuss possible altenatives.
  - A file named `README` in which you should describe any peculiarities of your solution, for instance, things that may be missing, or assumptions made, or questions for the group teacher. If everything runs fine, state it in the `README` file.

Good luck!