DS Assignment

Released: 7 October
Assignment Deadline: 6 November, 4pm
Feedback return: 27 November

This assignment is worth 25% of the final mark. This assignment has two parts: part one - a programming exercise and part two - a set of theoretical exercises. The assignment is marked out of 100. The programming exercise is worth 70% and the theoretical part 30% of the assignment.

1. Programming part (70 marks)

In this assignment, you have to create a simulation of a wireless network, compute a minimum spanning tree in that simulation, and broadcast messages in the network. For example, each node in the network may be a sensor unit, with the capability to sense the ambient temperature, pressure etc and store, receive and send data over wireless links. Each sensor is powered by a battery.

You will be given the precise location and the remaining battery life of each sensor node. Each sensor node can communicate with any other sensor nodes present in a radius **R = 10** meters around its location.

The programming assignment has two parts: First is computing the minimum spanning tree in the network where weights of edges are their lengths. Second is to carry out a given sequence of broadcasts in the network. Nodes may run out of energy in this second phase.

Overall, your tasks are:

1. Read the node locations and description from an input file
2. Have a protocol to “create” the network, so that every node knows its neighbors. (you will need suitable features in your simulation to ensure proper communication.)
3. Compute a minimum spanning tree of the network.
4. The input file will also have a sequence of broadcast messages to be sent to all nodes in the network. You have to carry out these broadcasts using as little energy as possible.

In the simulation environment, the nodes can send and receive the following types of messages:

- send “discover” messages in the wireless environment, to find out what other nodes are around them in the R radius. A “discover” message goes to all nodes in distance R
- respond to “discover” messages with their exact position
- send “neighbor” messages, to specific neighbor nodes within distance R
- receive “neighbor” messages (we assume acknowledgements are not necessary and

any message sent is always delivered)

The “neighbor” messages can carry multiple payloads (data): broadcasts, sensor data load, messages for computing MST etc. You can use freely these for MST construction.

In addition, there is a cellular *base station* that sends broadcast beacon messages to be used in specific circumstances in MST construction. Note that the base station’s communications are not restricted to UDG, they can directly transmit to everyone.

1.1. Distributed Minimum Spanning Tree (50 marks)

In the MST construction, your simulator will **not** consider the energy costs. The messages in constructing MST are small and we are assuming that these cost negligible energy. You should simulate these as consuming zero energy. The task is to implement the ***SynchGHS*** algorithm for computing the Minimum Spanning Tree (MST) on the wireless sensors network. This task will be performed in a distributed manner, with all nodes involved in the process by exchanging messages with their neighboring nodes.

Each node starts by knowing only its location. They can send and receive discovery and neighbor messages with the other nodes in their proximity as described above.

A quick overview of the steps required:

1. The base-station alerts all the nodes to start constructing the MST
2. Each node discovers the other nodes in its proximity
3. The leader of a connected component broadcasts a message inside the component for each node to identify a new edge to add to the MST
4. Each node chooses the link with the lowest weight to add, and sends it to the leader of its component in a convergecast
5. The leader of a component chooses which link to add and broadcasts (floods in the tree) its decision.
6. Once all the leaders have made their decisions, the beacon broadcasts a “merge” message. All the leaders now flood (in the tree) their id, and in each connected component, the leader with the highest id becomes the leader of the entire component for the next level.
7. The base-station waits for all the components to finish before moving to the next level and broadcasts a beacon to start next level
8. The next level starts again from step 3.
9. The algorithm finishes when there is no mode link added to the MST in a level.

In step 4, the nodes use convergecast in GHS, however, if you want to simplify this step, you are allowed to use a flood (inside the tree.) In steps 6, we suggest a broadcast based method for electing leader of a new component, because this is simpler than the original GHS algorithm.

You are free to use either.

Note that the base-station is assumed to automatically know when certain stages of computation has finished. This is merely to simplify your implementation. In a real system, we will need specific protocols to detect these. (We will cover termination detection later in class.)

For the entire process, you can assume a synchronous communication model. Meaning that, there is a global clock that every node knows (eg. broadcast by the base station). Nodes send messages in rounds. That is, all nodes send their messages simultaneously in one round. Then they perform computations on messages received in the current round. Then they proceed with transmission of next round of messages etc.

Note that the version of GHS we had covered in class was also in synchronous model. The description we have given here is further simplified for easier programming. You can see the appendix for a description of the algorithm.

Your simulator should produce a log file "**log.txt**" with the following information describing the progress of the algorithm:

- when the base-station alerts the leaders to proceed to the next level:
  bs {NodeID1}, {NodeID2}, ….
  {NodeID} being the id of each node the base-station is communicating to.
- when a new leader is elected:
  elected {NodeID}
  {NodeID} is the id of the newly elected leader.
- when a new edge is added to a connected component
  added {NodeID1} - {NodeID2}
  {NodeID} is the id of each the nodes at the end of the edge.

Your log file should follow the exact format given here, including keywords (highlighted in **bold** above). These will be evaluated by an automated checker script which will not be able to read the logs if they do not conform to the format.

1.2. Energy budget restricted communication (20 marks)

In this part, your task is to carry out broadcast transmissions given in the input file. Each transmission simply gives the node that initiates the broadcast. Starting from that node, the broadcast must reach all nodes in the network. How you do this is up to you. Your goal is to minimize energy usage and keep nodes alive as long as possible. A node *dies* when its energy reserve drops below minimum energy budget.

The sensor data transferred between nodes is in a large volume, unlike control messages such as in the MST construction. If you use other "control" messages, those costs can be ignored in

this case too, however, the broadcast message consumes energy in going from one node to another. Let us take the energy cost of a transmission between neighbors to be:

Energy_cost (distance) = distance * 1.2 (2)

The assumption is that only the sender pays for this energy cost from its energy budget, with no implications on the energy budget of the receiver. So, if the link between Node1 and Node2 has an energy cost of 12 energy units and Node1 has a budget of 210 energy units left and Node2 has 355 energy units, after Node1 sends to Node2 then budget for Node1 becomes 198 while the budget for Node2 remains 355.

Nodes can not function with less energy than a minimal budget (MB), which is defined at the start of the input file. Once a node goes below this value (< MB) then it has energy for just to inform the nodes in its proximity that it's going down. From that point this node should not be used for any other forms of communication by the network. Your network should cope with the lose of nodes. Also, the simulator will print in the log file this information:

node down {NodeID}

where {NodeID} is the id of the node going down.

Additionally, the simulator prints in the log file information about the transfers of sensor data:

data from {NodeID1} **to** {NodeID2}, energy: {EnergyBudget1}

where {NodeID} indicate the id of the sender (first) and the receiver (second), {EnergyBudget} indicates the remaining energy on the sender node; for each of the sensor data transfers during a broadcast.

Instructions from the input file should be executed in order. We assume that the second broadcast starts only after all activities of the first broadcast has ended. Here again, you can assume that the bases station coordinates this. (Also, all communications are synchronous, as before.)

Example:

Node 1 is connected to node 2 and node 2 is connected to node 3. Each node has 30, 56 and 119 energy respectively. The link cost of 1-2 is 7 energy units and the cost of 2-3 is 3 energy units. MB for this simulation is 25.

When sending the sensor data from 1 to 2 the energy budget of node 1 becomes 23. Because the energy budget of node 1 is less than MB this will be going down. When sensing from 2 to 3, the energy budget of node 2 becomes 53.

The simulator will produce the following output:

data from 1 to 2, energy: 23
node down 1

data from 2 to 3, energy 53.

Implementation details
Input file:

Mimimum Budget
NodeID, position_x, position_y, energy
NodeID, position_x, position_y, energy
NodeID, position_x, position_y, energy
…
bcst from NodeID
bcst from NodeID
…

where, on the first line is the minimum budget of a node before it goes down. The next lines will contain the configuration of a sensor node representing the unique id of the node (an integer value), the X and Y coordinates (real numbers) and the energy budget left (real value). After the node configuration lines, until the end of the file there will be communication instructions for sensor data broadcasts (marked by "bcst") starting from a node indicated by the NodeID. An examples of input file:

7
node 1, 5.43, 6.23, 20.3
node 2, 12.2, 6.2, 190.2
node 5, 14.3, 10.4, 19.4
bcst from 1
bcst from 2
bcst from 1
bcst from 5

Correct implementation of these tasks can be performed with multi-threaded or single-threaded implementations. In a multi-threaded simulation, each sensor node is simulated by a thread, whereas the simulator itself is another thread that simulates message transmission etc. In a single threaded implementation, the simulator also simulates the actions of each sensor node.

You are free to make your own assumptions and modeling of the communication protocol as long as this is in the line with the general description of the assignment. Please include a short description of your assumptions and your choices in a ReadMe.txt file. Also, commenting your code will help us understand it much faster. Points will be given for code aesthetics and good explanation.

Annexes

1. **The GHS Algorithm** is a distributed algorithm for the computation of the MST on a given connected undirected graph. The algorithm assumes that the weights of the edges of the graph are distinct. This assumption can be implemented by taking weight=(distance, id) format.

Each node of the graph knows its adjacent neighbors and their distances (therefore weights). The algorithm works in levels. At level k the various connected components constitute a spanning forest, meaning that each component is a tree (that is a subgraph of the MST). Each tree has a leader and the id of the leader is the id of the whole tree.

At level 0 each node is a component (i.e., tree). To get level k from level k-1, every leader (i.e., every component) of level k-1 performs a search in its spanning tree to find its minimum-weight outgoing edge. When this node is found among the outgoing edges of the component, the node that has this outgoing edge must be aware (through a broadcast). This node then informs the node on the other side of the edge (which belongs to another component and must inform its own leader).

When all the components of level k-1 have found their minimum-weight outgoing edges, the components are combined to build the level k spanning forest. A new leader must then be chosen for each component of level k. It can be shown that during the merging process of components, two components will select each other (i.e., they will have the same edge as minimum weight outgoing edge). The new leader is the node of this edge that has the highest Id. Once the new leader is selected, its id is broadcasted to the component.

Note that in our description, we simplified this last part. So now, instead of this protocol, you can simply have all the leaders of level k-1 flood the newly created connected component. They then select the leader with largest id. (but you are also free to use the real GHT protocol)

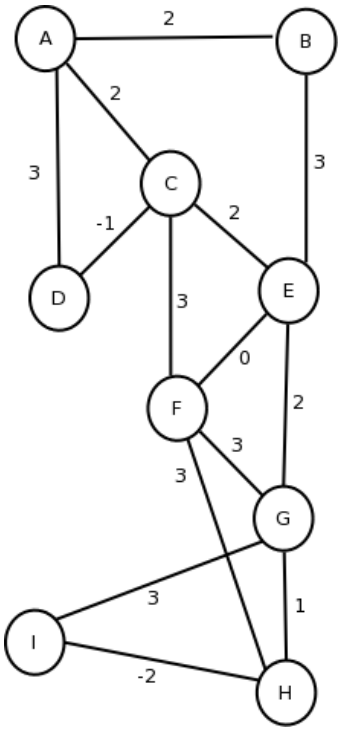
The algorithm terminates when there is a single component containing all the nodes in the network and searching for a minimum weight outgoing edge fails.

2. The Euclidean distance between the sensor node S1 with coordinates (x1, y1) and sensor S2 with (x2, y2) is: $\sqrt{(x1 - x2)^2 + (y1 - y2)^2}$. You have to use this for MST construction.

2. Theoretical part (30 marks)

2.1 If V is a vector clock, prove that a $\rightarrow$b if and only if V(a) ≤ V(b) [8 marks]

2.2 Show that Lamport's mutual exclusion algorithm satisfies the Liveness property. Assuming that all channels are FIFO and there are no failures in channels or processes. [8 marks]



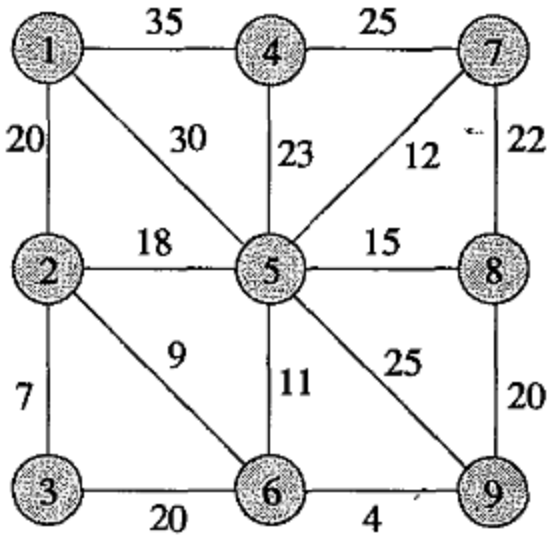
2.3

The figure above shows a weighted network graph. What is the weighted diameter of this network? (that is, diameter of the network, where the weight of an edge represents its length.) Which is the path that realizes the diameter? What will be the diameter if the graph was unweighted, and what will be the corresponding path? [4 marks]

2.4 Answer any one: [10 marks]

(a) Show that in a Unit Disk Graph, the minimum spanning tree (where the weights are lengths of the edges) cannot have any vertex with more than 6 edges.

(b) Using Prim's algorithm, find the minimum spanning tree of the following network:



3. Submission:

You are allowed to use C, C++, java or Python. If you have a strong reason to use a different language, discuss with us. However, we strongly prefer that you use one of the above. The code will be tested on DICE. If it does not run directly on DICE, we cannot evaluate it.

The submission should contain the following in a single folder:

1. Your *source code*. NOT the executable. The source code must be well commented.
2. A shell script called run.sh. This must compile and run your program when executed, and should take the input file as a parameter e.g.

   ./run.sh input.txt

3. A readme file (max 1 page in 12 pt font, 1 inch margin)
   - 3.1. Explaining the design of your simulator, how your code is structured etc.
   - 3.2. Also explain your algorithm and any specific choices you made in computing MST.
   - 3.3. Explain your strategy in broadcast and adaptation to node failures in part 1.2: Energy budget restricted broadcasts
4. The theory part as a single pdf file: *ds_theory_<your uun>.pdf*

To submit this, put everything in a single zipped folder and use from DICE:

submit ds 1 *ds_assignment_<your uun>.zip*

4. University regulations:

On good Scholarly Practice. Please remember the University requirement as regards all assessed work. Details about this can be found at:

http://www.ed.ac.uk/schools-departments/academic-services/students/undergraduate/discipline/academic-misconduct

and at:

http://www.inf.ed.ac.uk/admin/ITO/DivisionalGuidelinesPlagiarism.html.

Remember, if you use ideas from elsewhere (including other students), cite them. And try not use too much of these. The regulation says you can pick up “general ideas” but not “pivotal ideas”. But “general” and “pivotal” are very subjective and depends very much on the person making the judgement. Play safe and avoid getting into trouble.