PROGRAMMING ASSIGNMENT 4
Mobile Ad-Hoc Network (MANET) Simulator

Subject : Recursions and collections
TAs: Cemil ZALLUHOGLU, Necva BOLUCU, Selim YILMAZ, Selma DILEK

Due Date: 13.12.2017 23:59:59
Click here to accept your 4th Assignment.

1 Introduction

Nowadays, almost all of us carry mobile devices that communicate to each other in a direct or indirect way. Mobile ad-hoc network (MANET) is a decentralized network which comprises of a collection of these devices/nodes that interact with each other, forming a temporary network without any aid of a centralized administration. Since the advent of it, MANETs are now integral components of many applications such as military applications, virtual conferences, and disaster recovery operations; they have become part of out daily lives.

The topology of a MANET is dynamic as the nodes belonging to MANET can enter or leave constantly. Devices, we call them as *nodes* henceforth, must route the packets to another in order to make the network fully connected. Two chosen nodes are said to be 'directly connected' to each other as long as they are located somewhere within the transmission ranges of each other so they can communicate directly; otherwise they would require other nodes to communicate. Hence, a route should be established first for the message delivery between two nodes.

Routing –a process of moving a packet of data from source to destination– in such dynamic, infrastructure-less mobile ad-hoc network has always been an attractive research area for scientists, it is known as **routing in MANET** in the literature. The quality of a route between sender and receiver nodes varies depending on the network or the problem type at hand.

In the following section, approaches for network and route construction as well as optimal route selection are described in detail.

2 AdHocSim Simulator

In this assignment, you are expected to implement a program that simulates MANET, we call simulator as **AdHocSim**. You first need to establish the network then find possible routes

between sender and receiver nodes finally choose optimal route among them. To accomplish this task you have to use **recursion** while constructing the possible routes.

In Section 2.1, how to construct nodes and determine their neighbors are explained; Section 2.2 describes the approach that you need to consider for finding possible routes. Finally, the selection criteria for choosing optimal route among possible routes are detailed in Section 2.3.

2.1 Setting-up the Nodes and the Neighbors

As mentioned, MANET comprises of nodes that communicate each other directly or indirectly by employing other nodes as routers in the case of they are not located within the communication range. In this assignment, a node carries three different information: i) *location*, ii) *transmission range*, and iii) *residual battery level*. For example, node A is located at `(120,45)` and has 80% battery level (see Fig. 1).

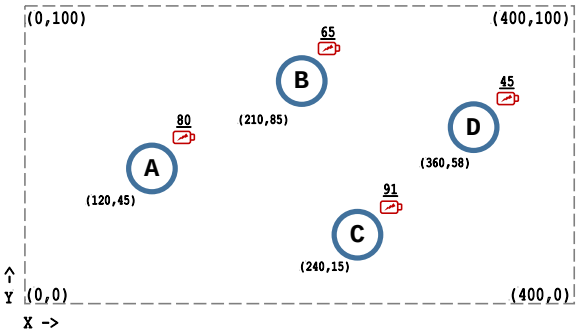


Figure 1: Nodes in MANET

In order to determine neighborhood of each node, you first need to examine the location and transmission range information. To say A is one of the neighbors of B, A should be within the transmission range of B. As mentioned, every node has location and transmission range information which is determined by east (x_1), west (x_2), north (y_1), and south (y_2) coordinate limits. For example `B` and `C` are the two neighbors of `A` (see Fig. 2).

Note: To represent neighborhood you are advised to construct a dictionary where key is node label and its corresponding value is its neighbors, as illustrated follows. There is an example of dictionary that stores relations:

```
'A': ['B', 'C']
'B': ['D', 'E']
'C': ['F', 'H']
'D': ['E', 'G']
'E': ['G']
'F': ['G', 'H']
'G': ['I']
'H': []
'I': []
```

2.2 Route Finding in MANET

You may think of a network as a directed graph where nodes represent vertices whereas communication links represent edges. A graph might be *cyclic* or *acyclic*. A graph is said

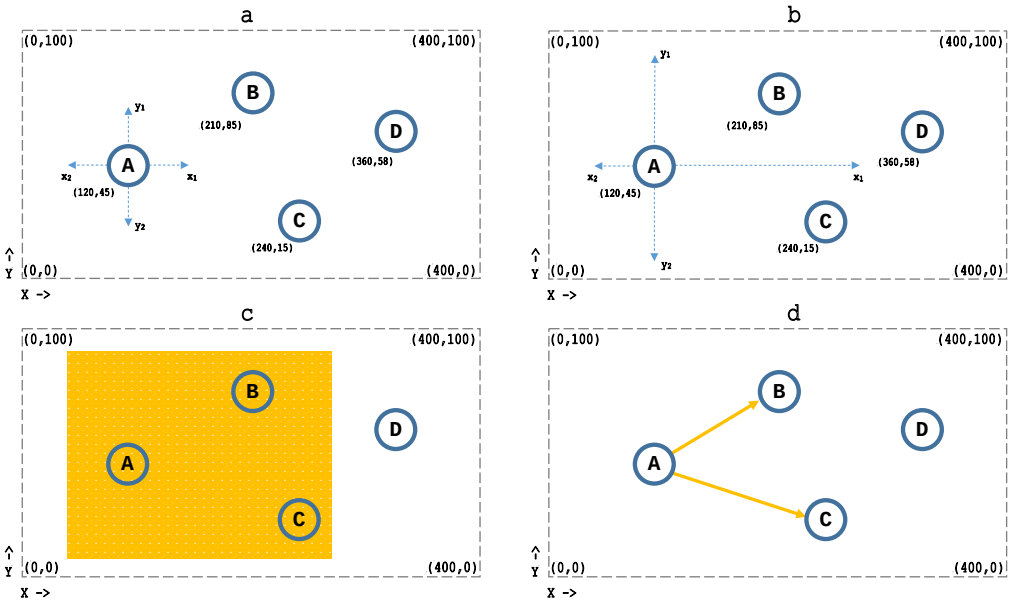


Figure 2: Determining relations in a MANET

to be cyclic if there exists some vertices that are connected in a closed chain (see Fig. 2for an example of acyclic graph). In this assignment, you will obtain cyclic or acyclic network once you set-up nodes and their neighbors successfully. Concretely speaking, there might be a case where the next node is a node that has already been visited before. In such case, you should design your program such that it refuses to visit such nodes in order to avoid infinite loop problem.

After setting up the nodes and determining of neighbors of every node, we obtain a network ready-to-use. Before transmission of first packet begins, it is needed to find a path through which packets are transferred. This path is called as *route* that starts and ends with a sender and destination nodes, respectively. Remember that route consists of only sender and destination nodes if they are within each other's transmission range. But, on the other, they employ other nodes as intermediate nodes by treating them routers for transmission of a collection of packets.

To clarify the routing process in MANET, we make use of Fig. 3. From the figure, sender node `A` is ready to initiate a data transfer to destination node `I` and there exist four routes between `A` and `I`. (Note: There might be cases where there is no route between any given two nodes.) A step by step route finding process is demonstrated with figures at the end of this document. With a similar way, you are expected to find every possible routes, using **recursion**, with its cost whose calculation is explained in the following section.

2.3 Selecting an Optimal Route

This process is similar to what actually navigation application does. Suppose Alice wants to visit Bob and uses his application to take a route to Bob's house. Application returns Alice's request with a route within a series of possible routes, which is often shortest path

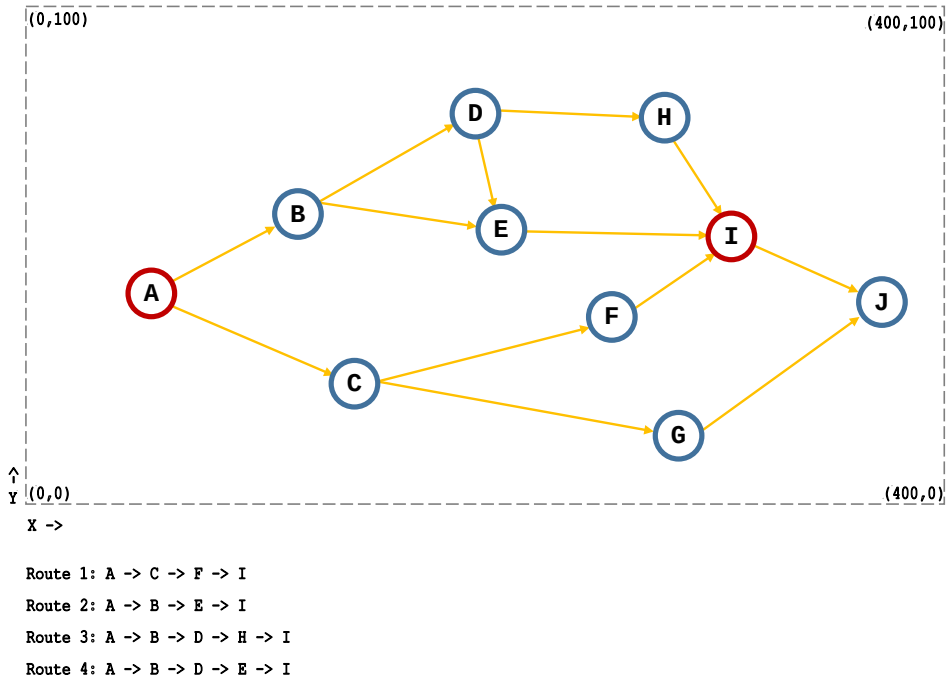


Figure 3: Steps of route finding process in a MANET

with respect to the *time* or *distance*. In this scenario time and distance are the two costs that determine the quality of the routes. Similar to this scenario, one needs to determine one of the routes to use for packet transmission depending on the their costs after finding all routes. The cost of a route is the sum of the cost of every communication links (L) between nodes (see Eq. 1 and Fig. 4).

$$Cost(Route_r) = \sum Cost(i,j) \quad | \quad i,j \in L \tag{1}$$

You will consider *distance* and *residual battery level* for the calculation of cost of a link between nodes i and j given by:.

$$Cost(i,j) = \frac{dist_{i,j}}{battery_j} \tag{2}$$

where $dist_{i,j}$ is a distance between the nodes i and j. As these nodes are actually represented by two points on the Cartesian plane (i_x and i_y are two points of node i whereas j_x and j_y are the two points of node j), you can get the distance between them using Euclidean distance:

$$dist_{i,j} = \sqrt{(i_x - j_x)^2 + (i_y - j_y)^2} \tag{3}$$

In MANETs, apart from initial data transfer, it is also necessary to find routes every time a link breakage occurs. Link breakage problem arises mostly when a node leaves the route through which data packets are transferred. A node becomes unreachable either if it runs

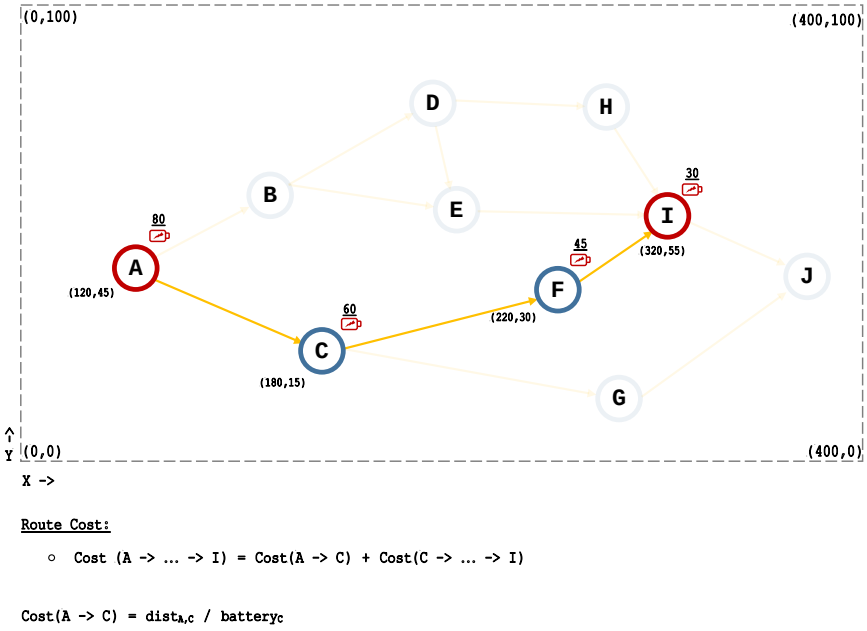


Figure 4: An example of a route cost

out of its battery or goes beyond the transmission range of a node from which it receives packets. That's why distance and residual battery level has been intentionally included in the calculation of every link cost in this assignment. As the probability of a node being exceeded transmission range gets lowered with a decrease in distance between two nodes, it is rational to select a link having a more closer one. Similarly, a link destined to a node having higher residual battery level should also be selected as it is lower probability of such node being left due to the drainage of battery. Therefore, the optimal route ($Route_{OPT}$) selection among possible routes (R) is done as follows:

$$Route_{OPT} = \arg\min_{r} \; Cost(Route_r) \mid r \in R \tag{4}$$

2.4 Simulator

The simulator you are asked to implement should start the simulation just after the network is established in line with the commands provided in a file named **commands** and should terminate itself once whole data transfer is completed. Here, you need to evaluate five types of commands:

- CRNODE (Create a new node): It generates a new node and requires four arguments to work properly. The first argument gives node a label (say i); second argument defines its locations as x and y coordinates (i.e., i_x and i_y); transmission range is determined in third argument in order of (x_1;x_2;y_1;y_2). Finally, residual battery level is set by forth argument.

  ```
  CRNODE y 120;40 100;0;60;40 65
  ```

Every time you evaluate CRNODE command, the following text should be displayed on the screen:

```
COMMAND *CRNODE*: New node y is created
```

- SEND (Send data): It initiates data transfer from a source node to a destination node. It takes three arguments; the first two defines source and destination node labels, respectively. The last argument is the amount of data (in Byte) to be transferred.

  ```
  SEND x y 1002
  ```

  Every time you evaluate SEND command the following text should be displayed on the screen:

  ```
  COMMAND *SEND*: Data is ready to send from x to y
  ```

- MOVE (Move node): It relocates a node to a pair of points that is defined as an argument provided with itself. There exist two arguments first of which is the node label whereas other defines its new location as x and y coordinates.

  ```
  MOVE x 165;70
  ```

  Every time you evaluate MOVE command the following text should be displayed on the screen:

  ```
  COMMAND *MOVE*: The location of node x is changed
  ```

- CHBTTRY (Change battery level): It simply charges or discharges the battery of a given node. It takes two arguments: i) node label and ii) new battery level.

  ```
  CHBTTRY x 90
  ```

  Every time you evaluate CHBTTRY command the following text should be displayed on the screen:

  ```
  COMMAND *CHBTTRY*: Battery level of node x is changed to 90
  ```

- RMNODE (Removes an existing node): It vanishes an existing node that is determined by an argument.

  ```
  RMNODE y
  ```

  Every time you evaluate RMNODE command the following text should be displayed on the screen:

  ```
  COMMAND *RMNODE*: Node y is removed
  ```

Note: Commands and their parameters are separated by <TAB> characters, each parameter is separated from each other by <TAB> characters as well.

Once you read **commands**, you will see one more argument at the beginning of every line. It represents the time (in Second) that points out when a command of interest should be applied. For example, `20 CHBTTRY x4 90` command line should be evaluated such that the residual battery level of node `x4` will be set to `90` when simulator reaches 20th second. The command file will always begin with SEND and CRNODE so that you set-up network (see Section 2.1) and begin the transmission of packets.

Any time the node is created, removed, re-localized, or battery level of it is changed; you need to initiate route finding process, which is naturally expected. Once at least one of these cases occurs, you must print out i) relations in the network, ii) possible routes found, and iii) optimal route chosen in the following format:

```
NODES & THEIR NEIGHBORS: x1 -> x2, x3 | x2 -> x4, x5 | x3 -> x4 | x4 -> x5, x7 | x5 -> x7 | x6
-> x7, x8 | x7 -> x9 | x8 ->  | x9 ->  |
5 ROUTE(S) FOUND:
ROUTE 1: x1 -> x2 -> x4 -> x5 -> x7 -> x9  COST: 4.9219
ROUTE 2: x1 -> x2 -> x4 -> x7 -> x9  COST: 5.0150
ROUTE 3: x1 -> x2 -> x5 -> x7 -> x9  COST: 4.1031
ROUTE 4: x1 -> x3 -> x4 -> x5 -> x7 -> x9  COST: 4.5047
ROUTE 5: x1 -> x3 -> x4 -> x7 -> x9  COST: 4.5978
SELECTED ROUTE (ROUTE 3): x1 -> x2 -> x5 -> x7 -> x9
```

Note: A path might not be found between end-points at every time. In such case simulator should terminate itself just after the message below is displayed on the screen.

```
NO ROUTE FROM x TO y FOUND.
```

Before simulator starts you need to calculate total number of packets (NP) to transfer whole data considering the equation below:

$$NP = \left\lceil \frac{DS}{PS} \right\rceil \tag{5}$$

where DS and PS are the data size and packet size, respectively. As mentioned DS is provided with SEND command while PS is provided as a command-line argument.

The last thing worth to mention is that, in AdHocSim simulator, every simulation time corresponds to one second, and only one packet is allowed to send from source to destination in every second.

Note: You can design your implementation such that every second corresponds to one loop step. You need to print out time stamp at every simulation time in the following format:

```
SIMULATION TIME: <HH>:<MM>:<SS>
```

The command-line argument that runs your implementation should be as follows:

```
python AdHocSim.py PS
```

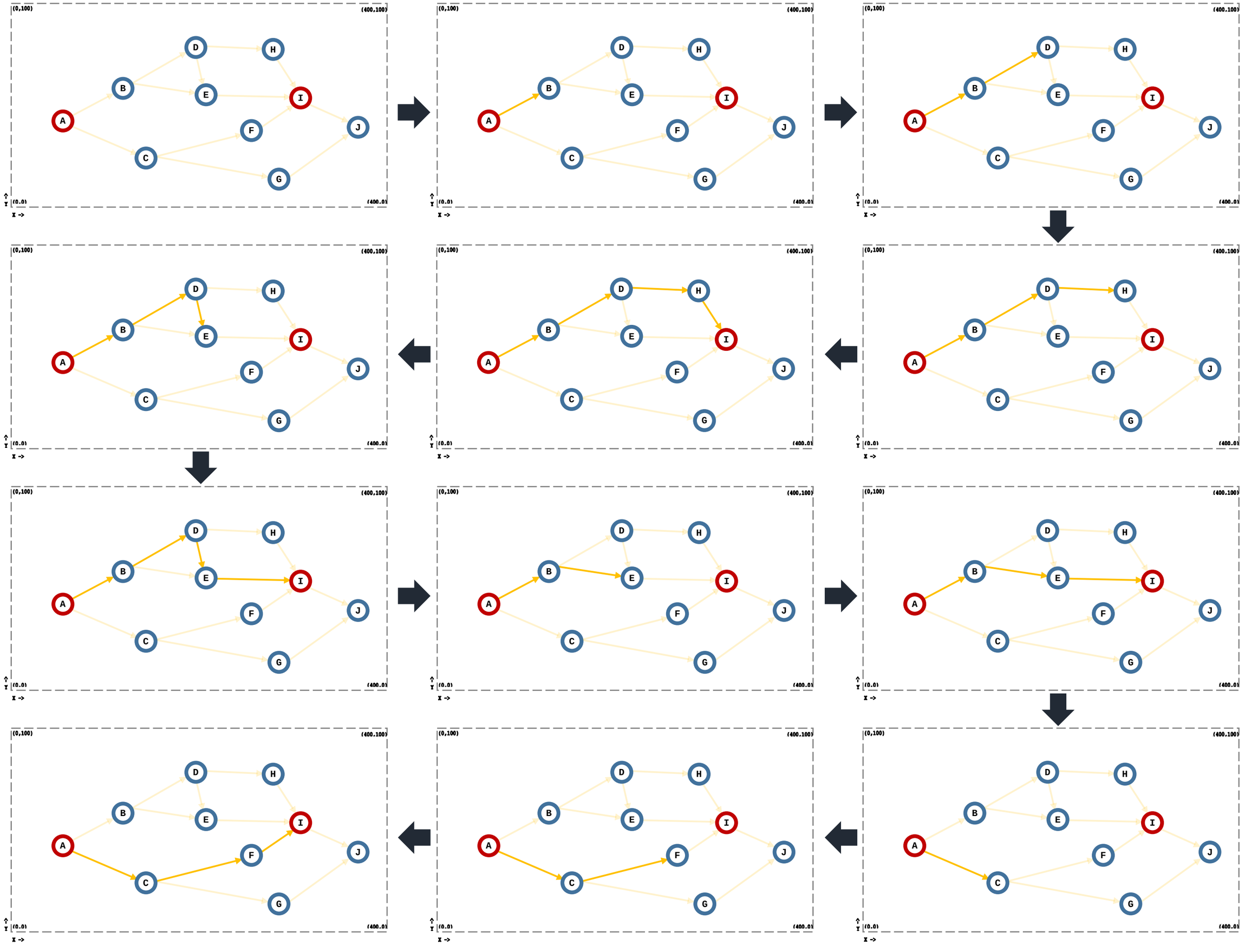
In the following page, an output sample that is created from simulator when these commands are evaluated:

```
0 CRNODE x1 120;40 100;0;60;40 65
0 CRNODE x2 200;50 80;10;30;5 40
0 CRNODE x3 150;15 120;0;25;10 88
0 CRNODE x4 240;60 100;10;0;40 35
0 CRNODE x5 260;45 70;0;10;25 98
0 CRNODE x6 225;15 75;10;5;10 70
0 CRNODE x7 273;20 75;0;0;0 68
0 CRNODE x8 221;5 0;0;0;0 68
0 CRNODE x9 345;20 0;0;0;0 68
0 SEND x1 x9 547
10 MOVE x3 165;70
20 CHBTTRY x4 90
32 RMNODE x5
```

Command-line argument:

```
python AdHocSim.py 15
```

```
******************************
AD-HOC NETWORK SIMULATOR - BEGIN
******************************
SIMULATION TIME: 00:00:00
        COMMAND *CRNODE*: New node x1 is created
        COMMAND *CRNODE*: New node x2 is created
        COMMAND *CRNODE*: New node x3 is created
        COMMAND *CRNODE*: New node x4 is created
        COMMAND *CRNODE*: New node x5 is created
        COMMAND *CRNODE*: New node x6 is created
        COMMAND *CRNODE*: New node x7 is created
        COMMAND *CRNODE*: New node x8 is created
        COMMAND *CRNODE*: New node x9 is created
        COMMAND *SEND*: Data is ready to send from x1 to x9
        NODES & THEIR NEIGHBORS: x1 -> x2, x3 | x2 -> x4, x5 | x3 -> x6, x8 |
        x4 -> x5, x7 | x5 -> x7 | x6 -> x7, x8 | x7 -> x9 | x8 ->  | x9 ->  |
        4 ROUTE(S) FOUND:
        ROUTE 1: x1 -> x2 -> x4 -> x5 -> x7 -> x9          COST: 4.9219
        ROUTE 2: x1 -> x2 -> x4 -> x7 -> x9   COST: 5.0150
        ROUTE 3: x1 -> x2 -> x5 -> x7 -> x9   COST: 4.1031
        ROUTE 4: x1 -> x3 -> x6 -> x7 -> x9   COST: 3.2837
        SELECTED ROUTE (ROUTE 4): x1 -> x3 -> x6 -> x7 -> x9
        PACKET 1 HAS BEEN SENT
        REMAINING DATA SIZE: 532.0 BYTE
SIMULATION TIME: 00:00:01
        PACKET 2 HAS BEEN SENT
        REMAINING DATA SIZE: 517.0 BYTE
SIMULATION TIME: 00:00:02
        PACKET 3 HAS BEEN SENT
        REMAINING DATA SIZE: 502.0 BYTE
SIMULATION TIME: 00:00:03
        PACKET 4 HAS BEEN SENT
        REMAINING DATA SIZE: 487.0 BYTE
SIMULATION TIME: 00:00:04
        PACKET 5 HAS BEEN SENT
        REMAINING DATA SIZE: 472.0 BYTE
SIMULATION TIME: 00:00:05
        PACKET 6 HAS BEEN SENT
        REMAINING DATA SIZE: 457.0 BYTE
SIMULATION TIME: 00:00:06
        PACKET 7 HAS BEEN SENT
        REMAINING DATA SIZE: 442.0 BYTE
SIMULATION TIME: 00:00:07
        PACKET 8 HAS BEEN SENT
        REMAINING DATA SIZE: 427.0 BYTE
SIMULATION TIME: 00:00:08
        PACKET 9 HAS BEEN SENT
        REMAINING DATA SIZE: 412.0 BYTE
SIMULATION TIME: 00:00:09
        PACKET 10 HAS BEEN SENT
        REMAINING DATA SIZE: 397.0 BYTE
SIMULATION TIME: 00:00:10
        COMMAND *MOVE*: The location of node x3 is changed
        NODES & THEIR NEIGHBORS: x1 -> x2, x3 | x2 -> x4, x5 | x3 -> x4 |
        x4 -> x5, x7| x5 -> x7 | x6 -> x7, x8 | x7 -> x9 | x8 ->  | x9 ->  |
        5 ROUTE(S) FOUND:
        ROUTE 1: x1 -> x2 -> x4 -> x5 -> x7 -> x9          COST: 4.9219
        ROUTE 2: x1 -> x2 -> x4 -> x7 -> x9   COST: 5.0150
        ROUTE 3: x1 -> x2 -> x5 -> x7 -> x9   COST: 4.1031
        ROUTE 4: x1 -> x3 -> x4 -> x5 -> x7 -> x9          COST: 4.5047
        ROUTE 5: x1 -> x3 -> x4 -> x7 -> x9   COST: 4.5978
        SELECTED ROUTE (ROUTE 3): x1 -> x2 -> x5 -> x7 -> x9
        PACKET 11 HAS BEEN SENT
        REMAINING DATA SIZE: 382.0 BYTE
SIMULATION TIME: 00:00:11
        PACKET 12 HAS BEEN SENT
        REMAINING DATA SIZE: 367.0 BYTE
SIMULATION TIME: 00:00:12
        PACKET 13 HAS BEEN SENT
        REMAINING DATA SIZE: 352.0 BYTE
SIMULATION TIME: 00:00:13
        PACKET 14 HAS BEEN SENT
        REMAINING DATA SIZE: 337.0 BYTE
SIMULATION TIME: 00:00:14
        PACKET 15 HAS BEEN SENT
        REMAINING DATA SIZE: 322.0 BYTE
SIMULATION TIME: 00:00:15
        PACKET 16 HAS BEEN SENT
        REMAINING DATA SIZE: 307.0 BYTE
SIMULATION TIME: 00:00:16
        PACKET 17 HAS BEEN SENT
        REMAINING DATA SIZE: 292.0 BYTE
SIMULATION TIME: 00:00:17
        PACKET 18 HAS BEEN SENT
        REMAINING DATA SIZE: 277.0 BYTE
SIMULATION TIME: 00:00:18
        PACKET 19 HAS BEEN SENT
        REMAINING DATA SIZE: 262.0 BYTE
SIMULATION TIME: 00:00:19
        PACKET 20 HAS BEEN SENT
        REMAINING DATA SIZE: 247.0 BYTE
SIMULATION TIME: 00:00:20
        COMMAND *CHBTTRY*: Battery level of node x4 is changed to 90
        NODES & THEIR NEIGHBORS: x1 -> x2, x3 | x2 -> x4, x5 | x3 -> x4 |
        x4 -> x5, x7| x5 -> x7 | x6 -> x7, x8 | x7 -> x9 | x8 ->  | x9 ->  |
        5 ROUTE(S) FOUND:
        ROUTE 1: x1 -> x2 -> x4 -> x5 -> x7 -> x9          COST: 4.2020
        ROUTE 2: x1 -> x2 -> x4 -> x7 -> x9   COST: 4.2951
        ROUTE 3: x1 -> x2 -> x5 -> x7 -> x9   COST: 4.1031
        ROUTE 4: x1 -> x3 -> x4 -> x5 -> x7 -> x9          COST: 3.1836
        ROUTE 5: x1 -> x3 -> x4 -> x7 -> x9   COST: 3.2767
        SELECTED ROUTE (ROUTE 4): x1 -> x3 -> x4 -> x5 -> x7 -> x9
        PACKET 21 HAS BEEN SENT
        REMAINING DATA SIZE: 232.0 BYTE
SIMULATION TIME: 00:00:21
        PACKET 22 HAS BEEN SENT
        REMAINING DATA SIZE: 217.0 BYTE
SIMULATION TIME: 00:00:22
        PACKET 23 HAS BEEN SENT
        REMAINING DATA SIZE: 202.0 BYTE
SIMULATION TIME: 00:00:23
        PACKET 24 HAS BEEN SENT
        REMAINING DATA SIZE: 187.0 BYTE
SIMULATION TIME: 00:00:24
        PACKET 25 HAS BEEN SENT
        REMAINING DATA SIZE: 172.0 BYTE
SIMULATION TIME: 00:00:25
        PACKET 26 HAS BEEN SENT
        REMAINING DATA SIZE: 157.0 BYTE
SIMULATION TIME: 00:00:26
        PACKET 27 HAS BEEN SENT
        REMAINING DATA SIZE: 142.0 BYTE
SIMULATION TIME: 00:00:27
        PACKET 28 HAS BEEN SENT
        REMAINING DATA SIZE: 127.0 BYTE
SIMULATION TIME: 00:00:28
        PACKET 29 HAS BEEN SENT
        REMAINING DATA SIZE: 112.0 BYTE
SIMULATION TIME: 00:00:29
        PACKET 30 HAS BEEN SENT
        REMAINING DATA SIZE: 97.0 BYTE
SIMULATION TIME: 00:00:30
        PACKET 31 HAS BEEN SENT
        REMAINING DATA SIZE: 82.0 BYTE
SIMULATION TIME: 00:00:31
        PACKET 32 HAS BEEN SENT
        REMAINING DATA SIZE: 67.0 BYTE
SIMULATION TIME: 00:00:32
        COMMAND *RMNODE*: Node x5 is removed
        NODES & THEIR NEIGHBORS: x1 -> x2, x3 | x2 -> x4 | x3 -> x4 | x4 -> x7 |
        x6 -> x7, x8 | x7 -> x9 | x8 ->  | x9 ->  |
        2 ROUTE(S) FOUND:
        ROUTE 1: x1 -> x2 -> x4 -> x7 -> x9   COST: 4.2951
        ROUTE 2: x1 -> x3 -> x4 -> x7 -> x9   COST: 3.2767
        SELECTED ROUTE (ROUTE 2): x1 -> x3 -> x4 -> x7 -> x9
        PACKET 33 HAS BEEN SENT
        REMAINING DATA SIZE: 52.0 BYTE
SIMULATION TIME: 00:00:33
        PACKET 34 HAS BEEN SENT
        REMAINING DATA SIZE: 37.0 BYTE
SIMULATION TIME: 00:00:34
        PACKET 35 HAS BEEN SENT
        REMAINING DATA SIZE: 22.0 BYTE
SIMULATION TIME: 00:00:35
        PACKET 36 HAS BEEN SENT
        REMAINING DATA SIZE: 7.0 BYTE
SIMULATION TIME: 00:00:36
        PACKET 37 HAS BEEN SENT
        REMAINING DATA SIZE: 0.0 BYTE
******************************
AD-HOC NETWORK SIMULATOR - END
******************************
```


(0,100)
(400,100)
(0,0)
(400,0)
X ->
Y ->
A
B
C
D
E
F
G
H
I
J