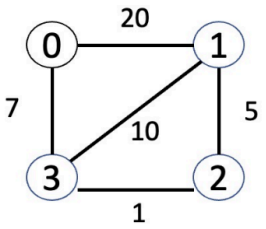
Take-home:

In this lab, you are going to develop and use the Distance Vector (DV) routing algorithm. You will work with a network topology with multiple nodes (routers) as the input and then implement the DV routing at every node (router) in a distributed manner.

To test your code, you are provided a sample network topology, which can be found in `topo.txt`, and the expected output can be found in the `sample_output.txt`

Inputs

$$\begin{matrix} 0 & 20 & -1 & 7 \\ 20 & 0 & 5 & 10 \\ -1 & 5 & 0 & 1 \\ 7 & 10 & 1 & 0 \end{matrix}$$



Note that the N nodes are marked as 0, 1, 2, · · · , N −1. So the link cost of node i to all the nodes 0, 1, 2, · · · , N −1 in order is given at the i + 1th row of this matrix. For example, the number “20” in the first row and the second column is the link cost between node 0 and node 1. The cost from any node to itself is 0, as shown in the matrix. A value of -1 indicates no link exists between the two nodes. Each node only has a local view at the start of the run.

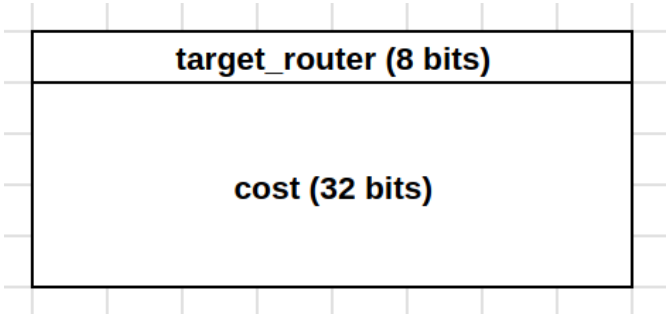
Implement the following functions in `impl/impl.c`.

1. Packet processing (1 mark)
    a. `unsigned char* serialize(struct packet* pkt)`: Function for serializing the packet using the header structure mentioned as follows. As you can see, this is a much more simplified version of the packet structure used in the RaSh protocol. (0.5 marks)

Header

source_router (8 bits)	dest_router (8 bits)
num_entries (8 bits)	distance_vector (variable)

DV_Entry (**Each** entry in the payload <distance_vector>)



The struct for the packet has been provided in `shared/types.h`

i. **source_router (8 bits)**: Represents the source router (0,1,2..)
ii. **dest_router (8 bits)**: Represents the destination router receiving the DV
iii. **num_entries (8 bits)**: Represents the total number of entries in DV where a valid entry exists (integer)
iv. **distance_vector**: Represents the payload for the packet
    1. **target_router (8 bits)**
    2. **cost (32 bits)**: The cost advertised by the source_router to target_router

Example:

```
Packet: {
        source_router: 0,
        dest_router: 1,
        num_entries: 2,
        distance_vector: [
        {
                target_router: 3,
                cost: 128 (0x80)
        },
        {
                target_router: 2,
                cost: 12345 (0x3039)
        }
        ]
}

Buffer: 0x00
        0x01
```

0x02
0x03
0x00
0x00
0x00
0x80
0x02
0x00
0x00
0x30
0x39

b. `struct packet* deserialize(unsigned char* buffer)`: Function for deserializing the packet (0.5 marks)

2. Router initialization (2 marks)
   `void router_init(struct distance_table *my_dt, int my_node, int *my_link_costs, int num_nodes)`:
   Function to initialize the distance table for a node, which is called at the beginning for each router in the topology. Once initialized, we call `send2neighbor` **for each** neighboring link to broadcast the node's distance vector.

You are provided with a helper function: `void send2neighbor(unsigned char* buffer)`, that must be called to send a packet to the neighboring routers.

Inputs

a. A pointer to the local distance_table: `my_dt`
   i. The distance table is a struct holding a 2D array of costs (`num_nodes` x `num_nodes`), which has already been initialized to a matrix of -1s with a diagonal of 0s (as each router can reach itself).
b. The index of the router in the topology: `my_node`
c. A pointer to their neighboring links: `my_link_costs`
   i. It is an array of size `num_nodes` that represents the costs to nodes reachable **DIRECTLY** by the router. If an entry is -1, it implies that there is no direct link to that router. It is (0,20,-1,7) for Node 0 in the input example.
d. The total number of nodes: `num_nodes`

Example Node 0 should firstly initialize its distance table by setting node 0's distance vector in the distance table (First row) to (0,20,-1,7), representing the minimum distance from node 0 to all nodes in the topology. It should leave all the remaining rows as it is, as we still do not have any data from the neighboring nodes. Lastly, node 0 needs to send its distance vector in a routing packet to its **connected neighbors** using `send2neighbor`.

For a packet sent from say, node 0 to node 1, the buffer sent to `send2neighbor` should be constructed as follows:

A. Allocate memory for the packet
B. Fill in appropriate values for the *source_router=0* and *dest_router=1* in the header
C. For each **positive cost** in node 0's distance vector, add a *dv_entry* to the *distance_vector* array and increment *total_entries*. Here, *total_entries* will be 2, and the *distance_vector* array will look like [{target_router: 1, cost: 20}, {target_router: 3, cost: 7}]
D. Now use `serialize` to return a pointer to the packet_buffer, and call `send2neighbor`.

3. Distance Vector Update (3 marks)
   `void router_update(struct distance_table *my_dt, int my_node, unsigned char* packet_buffer, int *my_link_costs, int num_nodes)`: Function to update the distance vector after receiving routing packets. This method will be called for every received routing packet. It deserializes the input, which is a routing packet, to update the distance table based on the DV algorithm. If its own minimum distance to any of the other nodes is updated, the node informs its directly connected neighbors of this change in minimum cost by sending them a routing packet via `send2neighbor`. Therefore, in the given example, nodes 0 and 1 will communicate with each other, but nodes 0 and 2 will not.

Inputs

a. An incoming serialized packet: `packet_buffer`
b. The latest local distance_table: `my_dt`
c. All other inputs are the same as `router_init`

The DV algorithm written by you will be evaluated in multiple iterations. The first iteration (k=0) will be complete when `router_init` is called on all the nodes. Any `send2neighbor` call in this stage will be evaluated at k=1 via `router_update`, and calls at k=1 will be evaluated at k=2 and so on. Once all the processing for an iteration is complete, the program will **AUTOMATICALLY** print out the *distance vectors* of each node onto the terminal (The distance vector of a node i is the ith row in node i's distance vector table).

Program output (Converges at k=2)

```
k=0:
node-0.my_dt[0]: 0 20 -1 7
node-1.my_dt[1]: 20 0 5 10
node-2.my_dt[2]: -1 5 0 1
node-3.my_dt[3]: 7 10 1 0
k=1:
node-0.my_dt[0]: 0 17 8 7
node-1.my_dt[1]: 17 0 5 6
node-2.my_dt[2]: 8 5 0 1
node-3.my_dt[3]: 7 6 1 0
k=2:
node-0.my_dt[0]: 0 13 8 7
node-1.my_dt[1]: 13 0 5 6
node-2.my_dt[2]: 8 5 0 1
node-3.my_dt[3]: 7 6 1 0
k=3:
node-0.my_dt[0]: 0 13 8 7
node-1.my_dt[1]: 13 0 5 6
node-2.my_dt[2]: 8 5 0 1
node-3.my_dt[3]: 7 6 1 0
k=4:
node-0.my_dt[0]: 0 13 8 7
node-1.my_dt[1]: 13 0 5 6
node-2.my_dt[2]: 8 5 0 1
node-3.my_dt[3]: 7 6 1 0
k=5:
node-0.my_dt[0]: 0 13 8 7
node-1.my_dt[1]: 13 0 5 6
node-2.my_dt[2]: 8 5 0 1
node-3.my_dt[3]: 7 6 1 0
```

Testing your Code

Run:

1. “make” in the project root directory to compile the program
2. “make test” to test your code on the sample input `topo.txt`

Look out for WARNINGS in the output window which might arise due to incorrect packets being sent to `send2neighbor`, that can be used to debug your program.

For the curious:
The output executable “dv” takes in 2 arguments

1. The number of iterations to run
2. The input topo text file

Note: Your code will be tested against other network topologies with an arbitrary number of nodes.

Submission Guidelines:

Submit `impl.c` in a zip: BITSID_CNLab8(Ex: 2021A7PS1234G_CNLab8.zip). There should be no other files. We will be only evaluating the following functions:

1. `unsigned char* serialize(struct packet* pkt)`
2. `struct packet* deserialize(unsigned char* buffer)`
3. `void router_init(struct distance_table *my_dt, int my_node, int *my_link_costs, int num_nodes)`
4. `void router_update(struct distance_table *my_dt, int my_node, unsigned char* packet_buffer, int *my_link_costs, int num_nodes)`

Make sure not to define any more functions. You are only allowed to call the functions defined in `impl.h`