# CS305 2022 Spring PA 3

## Problem

In this assignment, you will implement a static routing algorithm using the Dijkstra's algorithm. You will be given the computer network topology information. You will need to create a routing table for this network and output the IP address of all devices, through which a packet from A traverses in order before it reaches the destination B.

## Implementation details

You should implement a routing algorithm by using Dijkstra with the input of a file.
You should use the following command to run your program.

```
python PA3.py input.txt
```

Please see the format of the input file and additional information below:

- All the IPs in input file are IPv4 addresses, and they are all in CIDR notation. A prefix is shown as a 4-octet quantity, just like a traditional IPv4 address or network number, followed by the "/" (slash) character and followed by a decimal value between 0 and 32 that describes the number of significant bits. The format is **addr1/network prefix**, for example: **67.100.3.8/24**

  67.100.3.8 is the IP address, and 24 is network prefix, they are separated by a slash "/".

- The first line is a list of IPv4 addresses, which are the addresses of the interfaces for hosts and routers in the network, separated by a space.
  **addr1/network prefix addr2/network prefix addr3/network prefix ……**

- The second line is a string illustrating the interfaces on each router. The interfaces are grouped by router, separated by a space. For example:

  **(interfaceAddress1/network prefix,interfaceAddress2/network prefix,interfaceAddress3/network prefix) (interfaceAddress4/network prefix,interfaceAddress5,interfaceAddress6/network prefix,interfaceAddress7/network prefix)** are 7 interfaces of two routers.

- The third line shows all the edges in this network. The format will be (node/network prefix,node/network prefix,cost). For example, **(node3/network prefix,node4/network prefix,cost1) (node1/network prefix, node2/network prefix,cost2) describe two edges in the network graph**.

- The forth line will contain a number **m** to indicate that the following **m** lines are testcases.

- There are two kinds of test cases:

  - One kind is the **PATH** testcase, in which you should give the path from a source to a destination using the Dijkstra's algorithm. The format is **PATH src dst**, separated by a space. The corresponding output should be **src addr1 addr2 addr3 …… dst**, **separated by a space**.

  - The other type is the **TABLE** testcase, in which you should output the routing table of a router. The input format is **TABLE (interfaceAddress1/network prefix,interfaceAddress2/network prefix,interfaceAddress3/network prefix)**. **(interfaceAddress1/network prefix,interfaceAddress2/network**

**prefix,interfaceAddress3/network prefix)** indicates a router and its interface, which is the same format as Line 2. The output format should be multiple lines and each line denotes an entry of the routing table of this router. Here is an example output format:

**subnet number of addr1/network prefix via InterfaceAddress3**
**subnet number of addr2/network prefix via InterfaceAddress1**
**subnet number of addr3/network prefix via InterfaceAddress3**
……

If the subnet is directly connected you should output:

**subnet number of addr4/network prefix is directly connected**

The output should be sorted by the lexicographical order of using the subnet number of target addresses

**addr1 addr2 addr3** in previous example.

**subnet number of addr1** is the masked result of addr1. Take 67.100.3.8/24 as an example, its network prefix length is 24 bits, the subnet mask is 255.255.255.0. After performing the bitwise AND operations of 67.100.3.8 and 255.255.255.0, we could get 67.100.3.0 as the subnet number.

- The cost to transmit a packet in the router is 0.
- Each host is directly and only connected to one router.
- The bit number of a subnet mask is determined by the number after slash in IP address. If two hosts have the same subnet number (after bitwise operations), they are in the same subnet; otherwise, they are in different subnets.
- You must follow the Dijkstra's algorithm.
- Reduce the redundant path in each router. For example, as shown in the figure below, if you start from 67.100.3.8 to 120.100.3.5 you may go across 67.100.3.3  120.100.3.3 directly  or 67.100.3.3  200.30.6.1 120.100.3.3. You should choose the previous one to avoid redundancy in this case.
- After the first round, you have already inserted some items in routing table. However, we may observe that some subnet numbers are similar to others. Take 67.100.3.0/24 and 67.100.2.0/24 for example, their first 23 bits are the same, so we may use the routing aggregation scheme to merge the two items into one item in the routing table.

  **Before routing aggregation**, we have two items in the routing table:

```
67.100.3.0/24 via 120.100.3.3
67.100.2.0/24 via 120.100.3.3
```

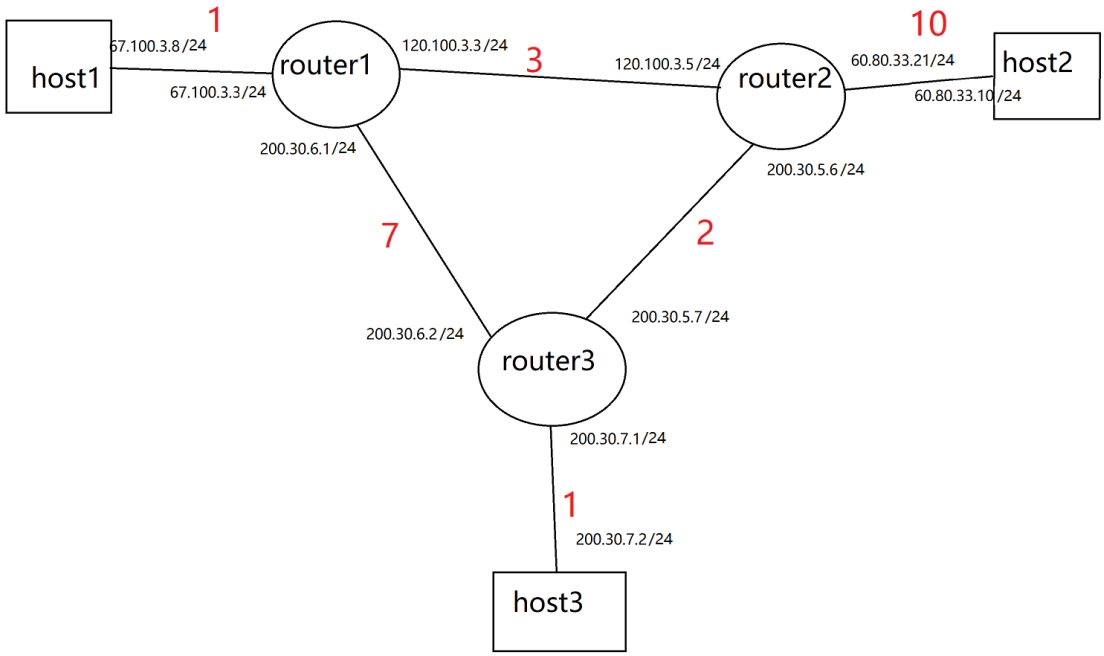
  **After routing aggregation**, we only have one item in the routing table:

```
67.100.2.0/23 via 120.100.3.3
```

- We should follow **FOUR rules** to make routing aggregation:
  - **Rule 1**. The router exit must be the same.
  - **Rule 2.** The common prefix length of the two (or more) subnets is at least 16 bits.
  - **Rule 3.** You should find the longest prefix length of those subnets.
  - **Rule 4.** You should not change the route path of the same pair of src and dst compared with before routing aggregation.
- Each subnet is made by either two interfaces of a router or one interface of a router and one host.

- For query of Path, it is guaranteed that the input of src and dst will only be two interfaces of hosts.
- For query of TABLE, we guarantee that the shortest path to a subnet is unique.

**Example**




For this example, the input is in a file named input.txt:

```
1: 67.100.3.8/24 67.100.3.3/24 120.100.3.3/24 120.100.3.5/24 60.80.33.21/24
200.30.5.6/24 200.30.5.7/24 200.30.6.1/24 200.30.6.2/24 200.30.7.1/24
200.30.7.2/24 60.80.33.10/24

2: ('67.100.3.3/24','120.100.3.3/24','200.30.6.1/24')
('120.100.3.5/24','60.80.33.21/24','200.30.5.6/24')
('200.30.5.7/24','200.30.6.2/24','200.30.7.1/24')

3: ('67.100.3.8/24','67.100.3.3/24',1) ('120.100.3.3/24','120.100.3.5/24',3)
('200.30.6.1/24','200.30.6.2/24',7) ('200.30.5.6/24','200.30.5.7/24',2)
('60.80.33.21/24','60.80.33.10/24',10) ('200.30.7.1/24','200.30.7.2/24',1)

4: 2

5: PATH 67.100.3.8/24 200.30.7.2/24

6: TABLE ('67.100.3.3/24','120.100.3.3/24','200.30.6.1/24')
```

**Note that the line numbers 1-6 are just for demonstration purpose. In the real tests, there will not be line numbers.**

The first line includes all the interfaces in the picture and the second line includes all the interfaces of every router.

The third line includes all the cost of edges in this graph topology.

The forth line denotes the number of testcases.

The fifth line includes a **PATH** testcase, including a start node and a end node.

The sixth line includes a **TABLE** testcase for routing table, which is a router and its interface.

For this input, the expected output before routing aggregation should be:

```
67.100.3.8 67.100.3.3 120.100.3.3 120.100.3.5 200.30.5.6 200.30.5.7 200.30.7.1
200.30.7.2
120.100.3.0/24 is directly connected
200.30.5.0/24 via 120.100.3.3
200.30.6.0/24 is directly connected
200.30.7.0/24 via 120.100.3.3
60.80.33.0/24 via 120.100.3.3
67.100.3.0/24 is directly connected
```

Since the items including 200.30.5.0/24 and 200.30.7.0/24 fit all the four rules of routing aggregation, we could aggregate them into one item as **200.30.4.0/22 via 120.100.3.3**.

So, the expected output after routing aggregation should be:

```
67.100.3.8 67.100.3.3 120.100.3.3 120.100.3.5 200.30.5.6 200.30.5.7 200.30.7.1
200.30.7.2
120.100.3.0/24 is directly connected
200.30.5.0/24 via 120.100.3.3
200.30.6.0/24 is directly connected
200.30.7.0/24 via 120.100.3.3
60.80.33.0/24 via 120.100.3.3
67.100.3.0/24 is directly connected
After
120.100.3.0/24 is directly connected
200.30.4.0/22 via 120.100.3.3
200.30.6.0/24 is directly connected
60.80.33.0/24 via 120.100.3.3
67.100.3.0/24 is directly connected
```

You should display the routing tables both before and after routing aggregation so that we could score separately, and use "After" to separate them.

**Note that all the IP addresses have a 24-bit subnet mask in this demo, but in the real tests, subnet mask bit number will vary.**

**Environments**

- Python 3.9.7

# Test details

## Input format

The inputs are all strings. Please read the input format carefully.

The first line is all the IP addresses, separated by a space.

The second line includes the IP address of all interfaces of the routers. The interfaces of each routers are separated by a space, and in each router, the interfaces are separated by a comma and surrounded with a bracket.

The third line includes all the edges. For each edge, the format is (src addr,dst addr,cost). Each edge is separated by a space.

The forth line is the number of testcases **m**.

The following **m** lines describe the testcases.

The format of querying PATH:  PATH src dst, separated by a space.

The format of querying TABLE:  TABLE (interface1,interface2,.....) .

**Tips: you can use eval() to convert a string after splitting it to tuple.**

For the example above, the real input is :

```
67.100.3.8/24 67.100.3.3/24 120.100.3.3/24 120.100.3.5/24 60.80.33.21/24
200.30.5.6/24 200.30.5.7/24 200.30.6.1/24 200.30.6.2/24 200.30.7.1/24
200.30.7.2/24 60.80.33.10/24
('67.100.3.3/24','120.100.3.3/24','200.30.6.1/24')
('120.100.3.5/24','60.80.33.21/24','200.30.5.6/24')
('200.30.5.7/24','200.30.6.2/24','200.30.7.1/24')
('67.100.3.8/24','67.100.3.3/24',1) ('120.100.3.3/24','120.100.3.5/24',3)
('200.30.6.1/24','200.30.6.2/24',7) ('200.30.5.6/24','200.30.5.7/24',2)
('60.80.33.21/24','60.80.33.10/24',10) ('200.30.7.1/24','200.30.7.2/24',1)
2
PATH 67.100.3.8/24 200.30.7.2/24
TABLE ('67.100.3.3/24','120.100.3.3/24','200.30.6.1/24')
```

All the strings and information is read from a file as args in command line. you may use:

```
with open(sys.argv[1], 'r') as f:
```

and read from the inputs.

## Output format

For the query of **PATH**, you should output a serial of IP addresses to indicate the path from the source node to the destination node, separated by a space.

**src addr1 addr2 addr3 ⋯ dst**

For the query of **TABLE** you should output multiple lines and each line includes an entry of the routing table:

**subnet number of addr/network prefix via interfaceAddress**, separated by a space

or

**subnet number of addr/network prefix is directly connected**

It is sorted in the lexicographical order of the **subnet number of addr**.

It should not include the interfaces addresses of the router itself as a target.

# What to submit

- An implementation file named PA3.py