Assignment 3: Longest IP Prefix Matching

Introduction

Networking is a cornerstone of modern computing, with IP routing being a critical component. When internet users send requests, these are sent in the form of so-called *internet packets*, from one IP address to another. These internet packets contain the information that was sent in the request (for example, the text of a WhatsApp message that you want to send to your friend), and also some additional information (like the aforementioned source and destination IP addresses).

A key concept in networking is the concept of *routing*. Since the Internet (note the capital letter) consists of many different smaller networks, it is essential to be able to communicate between these smaller networks. Since there are many of these networks, however, we need to keep track of where to send the messages in order to reach the other networks. These redirection rules are stored in *routing tables*.

These routing tables typically contain several things, but for simplicity's sake, we will assume it only consists of two values per row: the *destination IP address with the subnet mask* and the *interface* through which the packet must travel in order to reach the final destination. Before you continue, make sure you understand what subnet masks are. Subnet masks help in determining the network segment of an IP address, essential for routing packets. You can find more information in this video and article.

When a new packet comes in, we look in the routing table and search for the longest (consecutive) match with the provided destination IP address, as this is the subnet that the IP address belongs to. Once the candidate has been found, we look at the second value in the entry in the routing table to get the interface we must travel to, and continue our search there.

An example of a (simplified) routing table can be found in Figure 1.

Subnet	**Interface**
`192.168.1.0/24`	`eth0`
`10.0.0.0/8`	`eth1`
`172.16.0.0/16`	`eth2`
`192.168.0.0/16`	`eth3`

Table 1: Routing table of the router shown in Figure 1.

This process is just one simple example of using a routing table to find the next (intermediate) destination. Given that billions (perhaps even trillions) of these internet packets are routed every second, an efficient system for processing these packets is crucial. Efficient routing requires sophisticated algorithms to quickly determine the best path for data packets.

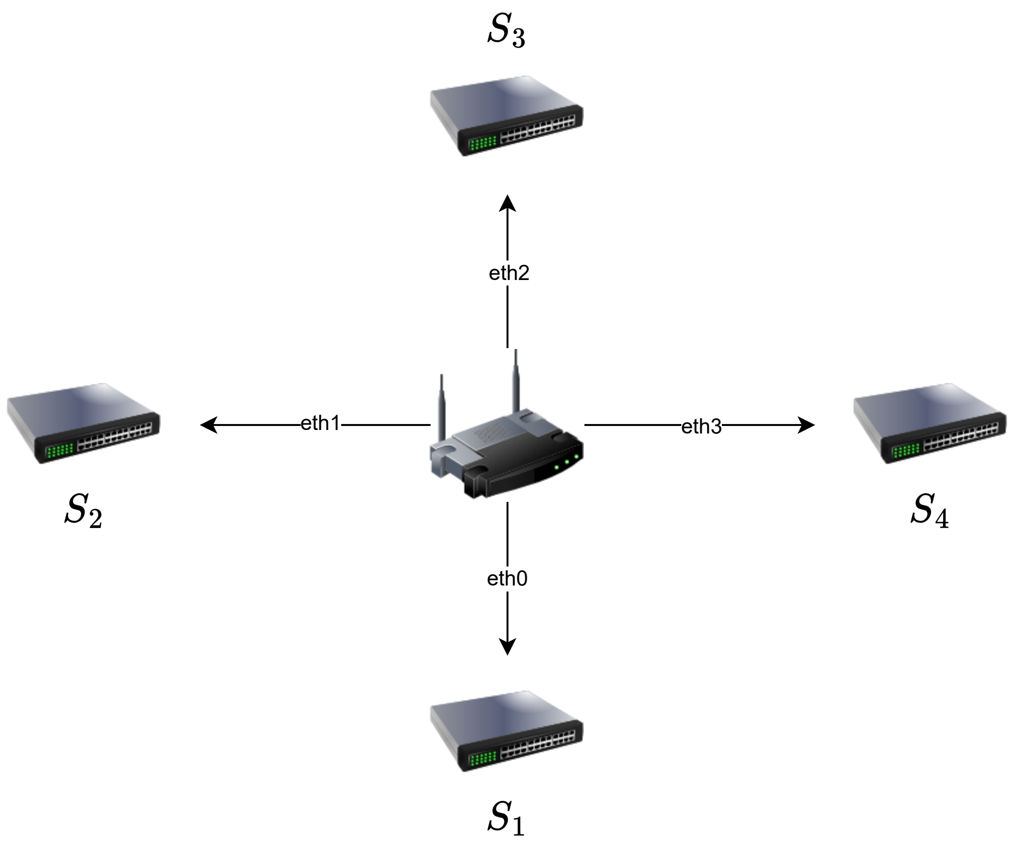

Figure 1: An example network architecture.

Objective

This assignment focuses on Longest Prefix Matching (LPM), a fundamental technique in routing that enables routers to select the most specific route for a packet. You will explore two approaches to implement LPM, highlighting the importance of data structures in networking. This assignment is split in two parts; the first part focuses on implementing a naive implementation, and the second one focuses on a more efficient approach.

Submission details

You can submit your program on Themis, as you are used to by now. As always, we will run Valgrind on each test case, so make sure that you handle memory management properly!

Part I

Naive implementation (2 points)

We will start by first make a naive implementation of LPM in networking. For the first part, you must do this by only using arrays, and no other data structures. There is no sample code, so you will have to start from scratch.

Input and Output Format

Input: the input starts off with a positive integer n, after which we have n lines of IP addresses with subnet masks and the corresponding interface number. We then read another another positive integer m, followed by m IP addresses.

Hint: an IP address with a subnet always follows the format `oct.oct.oct.oct/mask`, where `oct` and `masks` are some positive integers such that $0 \leq \texttt{oct} \leq 255$, and $0 \leq \texttt{mask} \leq 32$

Output: m routing numbers that correspond to the m requested IP addresses.

Example

Input:

```
4
192.168.1.0/24 0
10.0.0.0/8 1
172.16.0.0/16 2
192.168.0.0/16 3
3
192.168.1.128
192.168.0.255
10.255.255.254
```

Output:

```
0
3
1
```

Walk-through

- For `192.168.1.128`, it matches two subnets, namely `192.168.1.0/24` and `192.168.0.0/16`. However, the `/24` subnet is a longer (more specific) match, so it is routed to 0.
- `192.168.0.255` also appears to match both, but in this case, the more specific match is with the `/16` subnet, routing it to 3.
- `10.255.255.254` matches only the `10.0.0.0/8` subnet, directing it to 1.

Part II

Improved version (4 points)

As you may imagine, the previous approach works fine for smaller inputs, but when the number of IPs requested grows, the inefficiency of this approach becomes more apparent as well. Therefore, you will now improve this program by changing the approach and using search tries. Note that the input/output format is exactly the same for this part; there will be new test cases that cannot be passed with the naive implementation, however.

Hint: think about how you want to store the subnets in the search trie.

Part III

Report (4 points)

Additionally, you will write a report on this assignment as well. You can find the details about what these reports should look like on BrightSpace. Note that writing a report for this assignment is **mandatory**! Not submitting a report will result in your code submissions not being considered.