Question 1:
c. after we have our prefix table (we can also use our binary tree from a. And b.)
we can build another trie such that every node consists from – 4 bits.
We will take our prefixes from the prefix table and divide them into multiple of 4 bits sections,
if we have a section with less than multiple of – 4 bits we will complete the section to multiple of –
4 by taking all possibilities, for example if we have the following prefixes:
0*A
011*B
101*C
010111*D
10*E
1111*F
so our nodes will be:
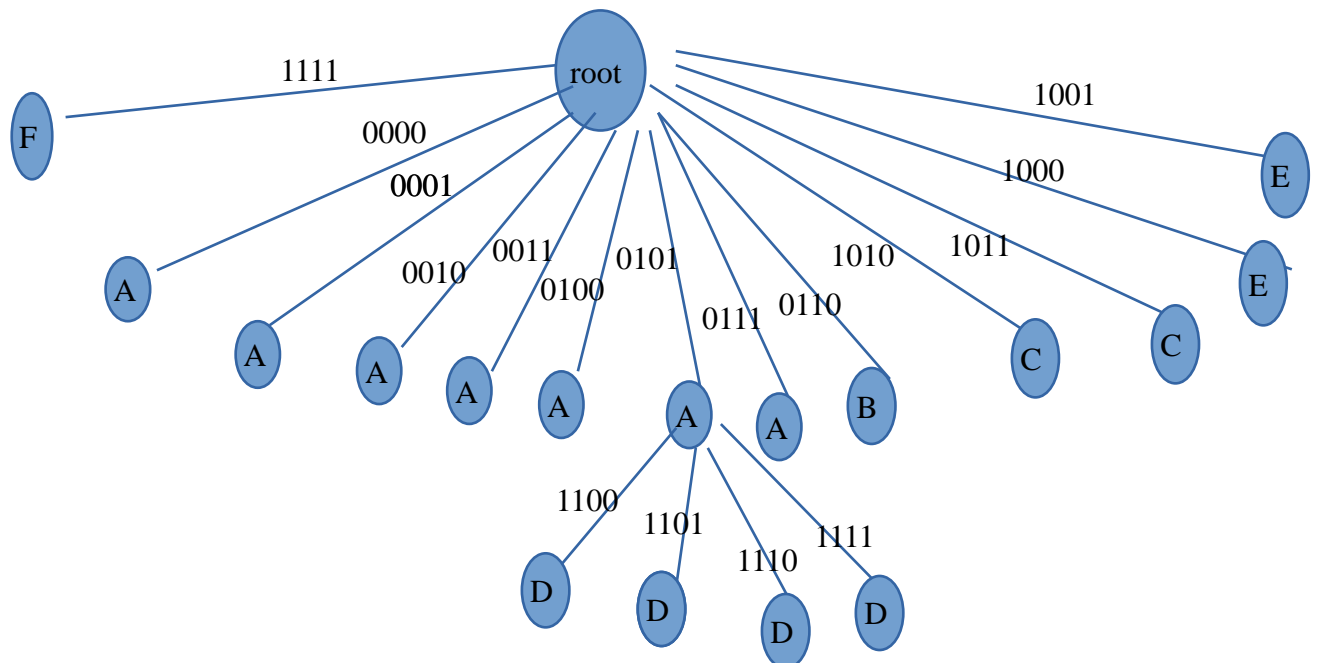0000 A,0001 A, 0010 A, 0011 A, 0100 A, 0101 A, 0111 A
0110 B, **0111 A**
1010 C, 1011 C
01011100 D,01011101 D, 01011110 D, 01011111 D
1000 E, 1001E, **1010 C ,1011 C**
1111 F

because we diveded every ip address into - 4 bits the maximum number of lookups is - 8 (32/4)
and the depth of the trie is also – 8.

but the required memory here is very big because for every non – 4 bit section we need to take all
possibilities to complete to – 4 bits and thats a lot of nodes.

d. another way to represent the prefix table is using hash table, and array with all the lengths we have in the prefix table. Say we have:

010100*

01100100*

01001110*

011*

so our array will contain 3 elements [8,6,3]

which are the lengths of our prefixes, and each prefix will enter to a hash table:

length	hash	hash tables
8	...	01100100 , 01001110
6	...	010100
3	...	011

then we will search from the biggest length (8 in our case) and if we found the right prefix in the hash table we finished if not we continue to the next biggest length.

This method is good when we have few prefixes in each length, but if we have a lot of prefixes in a certain length espcially in the bigger lengths, so the comparisson will take more time.

This metod can search in linear time and takes less memory than a trie.

We can improve this method to be even quicker using binary search on the array of lengths but the problem is how to use the binary search on the wanted length.

if we will use binary search we can serch in logarithmic time.