Main functions:

PrefixTree()

Receives a prefix table as an argument and returns the equivalent prefix tree.

Pseudo-code:

PrefixTree(prefix table)

Create prefix tree head node;

For(every table entry){

If(prefix is ‘e’)

Assign nextHop to tree head

Else{

Tree\_aux = tree\_root

For(each prefix[i]){

If(prefix[i] == 0)

Tree\_aux = tree\_aux->zero

If(prefix[i] == 1)

Tree\_aux = tree\_aux->one

} //end of for

} //end of else

Tree\_aux->nextHop = nextHop

} //end of for

Return tree\_root

PrintTable()

Receives a prefix tree and an empty address, prints the prefix tree in the terminal. This function is recursive in order to travel through all the prefix tree nodes.

PrintTable(prefix tree, empty address)

If(tree\_node->nextHop != -1) //-1 is equivalent to that node having no nextHop

Printf(prefix, nextHop)

If(tree\_node->zero != NULL)

PrintTable(tree\_node->zero, address + 0)

If(tree\_node->one != NULL)

PrintTable(tree\_node->one, address + 1)

LookUp()

Receives a prefix tree and an address, returns the corresponding address’ nextHop. An aux pointer travels to the corresponding prefix of the address, it then sees if there is a nextHop for that prefix or not printing the corresponding nextHop in case it exists.

LookUp(prefix tree, address)

For(each address[i])

If(address[i] == 0)

Tree\_aux = tree\_aux->zero

If(address[i] == 1)

Tree\_aux = tree\_aux->one

If(tree\_aux == NULL){

If(nextHop != -1) //-1 is equivalent to that node having no nextHop

Return nextHop

Else

Printf(“No nextHop found for this prefix”)

}

if(tree\_aux->nextHop != -1) //-1 is equivalent to that node having no nextHop

nextHop = tree\_aux->nextHop

}

InsertPrefix()

Receives a prefix tree, an address and a nextHop value, returns a prefix tree with the new prefix included. This function is similar to the previous one (LookUp()), but in the end, instead of returning the value of the nextHop, it changes the value of the current node’s nextHop to the desired one

DeletePrefix()

Receives a prefix tree and the address of the prefix to be deleted, returns a prefix tree without the prefix specified by the address.

DeletePrefix(prefix tree, address)

For(each address[i]){

If(prefix[i] == ‘0’)

Delete\_direction = LEFT

Tree\_aux == tree\_aux>zero

If(prefix[i] == ‘1’)

Delete\_direction = RIGHT

Tree\_aux == tree\_aux>one

}

If(node is not a leaf)

Tree\_aux->nextHop = NO\_HOP

Else

If(delete\_direction == LEFT)

freeNode(node->left)

if(delete\_direction == RIGHT)

freeNode(node->one)

Extra functions:

BinaryToTwoBit()

Receives a binary prefix tree and converts it to a two bit prefix tree which is returned.

PrintTableEven()