A
PROJECT
REPORT
ON

IP ADDRESS FINDER

Submitted in partial fulfillment of the requirement for the IV semester

of

BACHELOR OF

TECHNOLOGY

IN

DESIGN AND ANALYSIS OF ALGORITHM

Submitted By:

AFRAZ TANVIR (RA2011003010499)

ANKUSH SAHOO (RA2011003010531)

SHASHANK SINGH (RA2011003010541)

Under the supervision of

Dr. B. Pandeeswari

(Asst. Professor)

DEPARTMENT OF COMPUTING AND TECHNOLOGY, SRM

INSTITUTE OF SCIENCE AND TECHNOLOGY SESSION – 2022

School of Computing

SRM IST, Kattankulathur – 603 203

Course Code: 18CSC204J

Course Name: Design and Analysis of Algorithm

Title of Project	**IP ADRESS FINDER**
Team Members	AFRAZ TANVIR, ANKUR SAHOO, SHASHANK SINGH
Register Number	RA2011003010499, RA2011003010531, RA2011003010541
Date of Experiment	25/05/2022

Staff Signature with date

Aim: To make an algorithm which can search through a data as fast as possible

Team Members:

S No	Register No	Name	Role
1	RA2011003010499	AFRAZ TANVIR	Rep
2	RA2011003010531	ANKUSH SAHOO	Member
3	RA2011003010541	SHASHANK SINGH	Member

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY
S.R.M NAGAR, KATTANKULATHUR – 602 203

BONAFIDE CERTIFICATE

Certified to be bonafide record of the work done by ***Afraz Tanvir [RA2011003010499], Ankush Sahoo [RA2011003010531], Shashank Singh [RA2011003010541]*** *of* ***School of Computing,*** *B.tech degree course in the practical 18CSC204J-Design and Analysis of Algorithms in SRM Institute of Science and Technology, Kattankulathur during the academic year 2021-22.*

Date:25/06/2022 **Lab Incharge:**

Submitted for university examination held in _____SRM Institute of Science and Technology, Kattankulathur.

ABSTRACT

With the help of splay tree data structure, we would create a tree whose nodes are embedded with the Ip address of the device that are connect to a specific network router. In our code we have taken 11 devices connected to one network router and so there would be some common part in the Ip address of each of the devices. Now, router gets some specific data packets from the net which is supposed to be given to a specified device and so it uses searching operation to find the correct Ip address. To increase the speed of this process we use splay tress for searching and inserting the Ip addresses. It is the fastest data structure for searching operation. Therefore, the router sends the data packet to the specified Ip address when multiple devices are connected. Here we have used the random function to input the data packets so that there is no input function required and the processes is completely automatic as it takes place in network router.

TABLE OF CONTENTS

CONTRIBUTION TABLE

AFRAZ TANVIR RA2011003010499	SHASHANK SINGH RA2011003010541	ANKUSH SAHOO RA2011003010531
• Identifying the problem Statement • Devices the algorithm to work. • Analyse time complexity. • Give final touch and complies the report.	• Identifying the suitable data structure for our problem statement. • Optimize the code from Afraz Tanvir. • Preparing the report.	• Identifying algorithm strategy towards our problem. • Run Test cases on code and verification. • Preparing the ppts i.e review 1 and review 2

PROJECT DEFINITION

To make an algorithm which can search through a data as fast as possible. To use splay tree (fastest data structure) to achieve the aim.

BASIC PRINCIPLE

It uses the principle that the most occurring Ip address stays at the top and so the time complexity of searching decreases eventually.

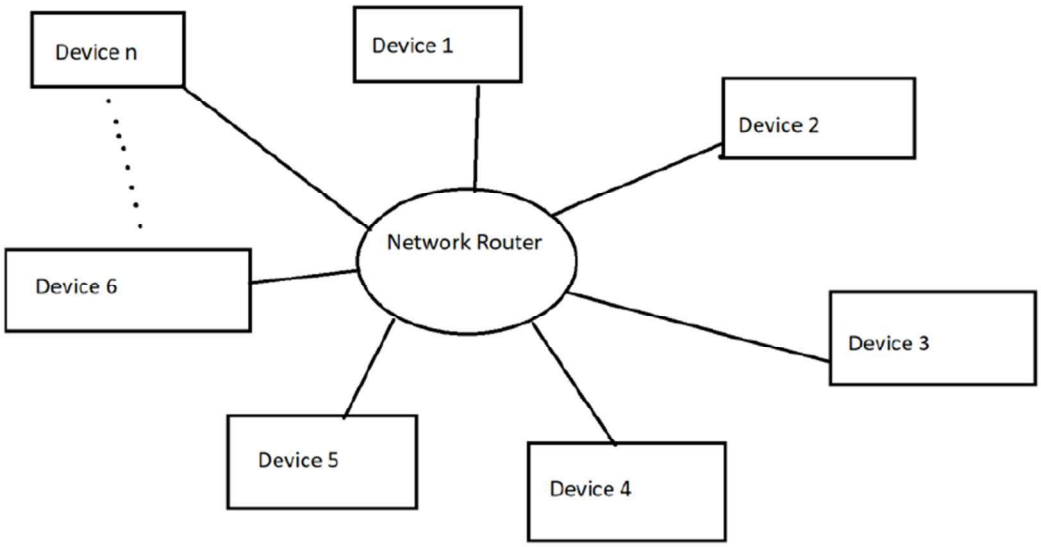
PROBLEM EXPLANATION

Basic principle: It uses the principle that the most occurring Ip address stays at the top and so the time complexity of searching decreases eventually.

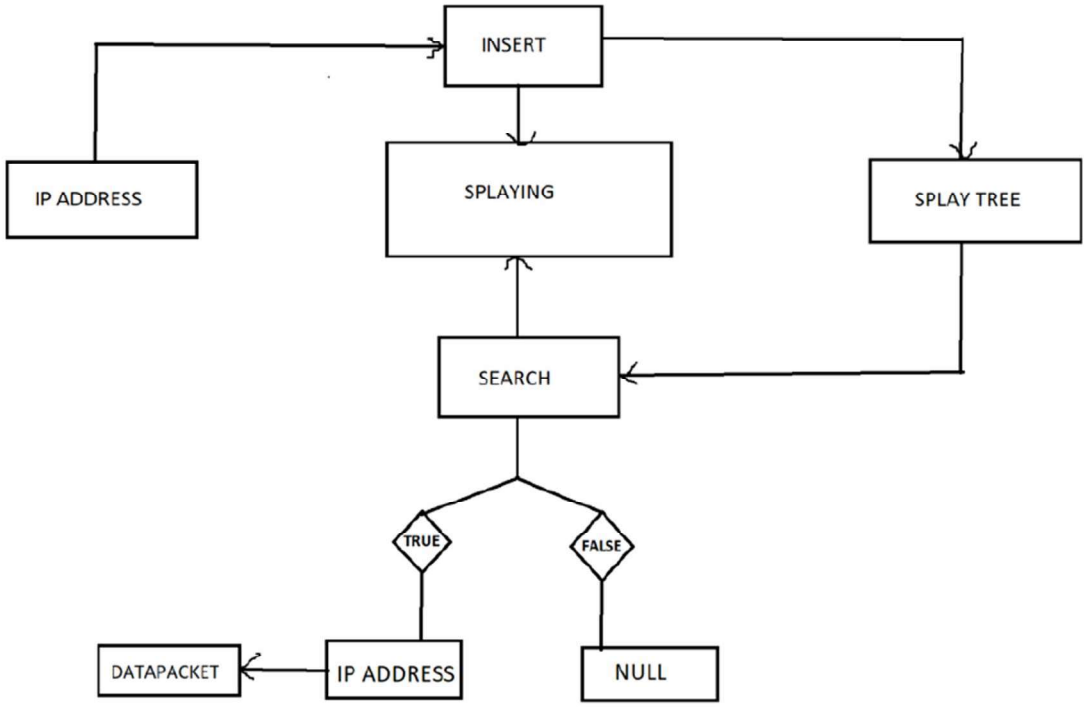
Methodology: To keep the most recurring element on the root node the data structure uses splaying operation.

For example :- If an IP address say 192.168.3.104 occurred most time it should be replaced with the root so that the next time while searching the same IP the searching time will be reduced.

ARCHITECTURE:



BLOCK DIAGRAM:



DESIGN TECHNIQUES AND ALGORITHM

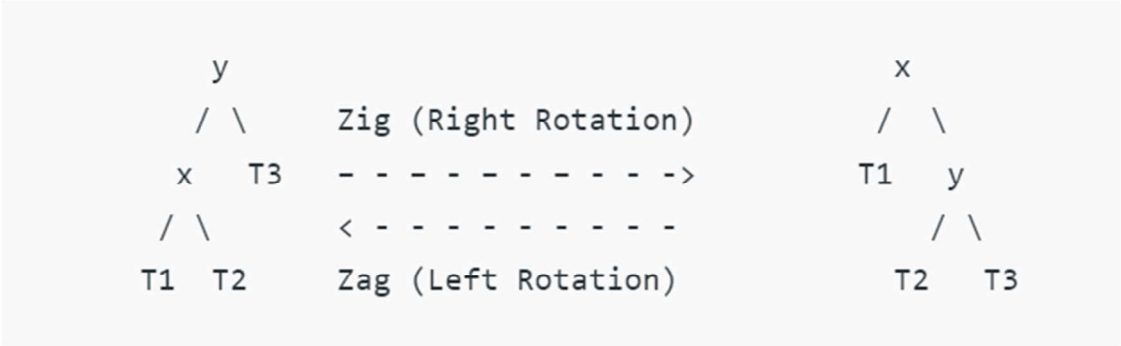
We will use greedy method to perform splaying.

The main idea of splay tree is to bring the recently accessed item to root of the tree, this makes the recently searched item to be accessible in O(1) time if accessed again. The idea is to use locality of reference (In a typical application, 80% of the access are to 20% of the items). Imagine a situation where we have millions or billions of keys and only few of them are accessed frequently, which is very likely in many practical applications.

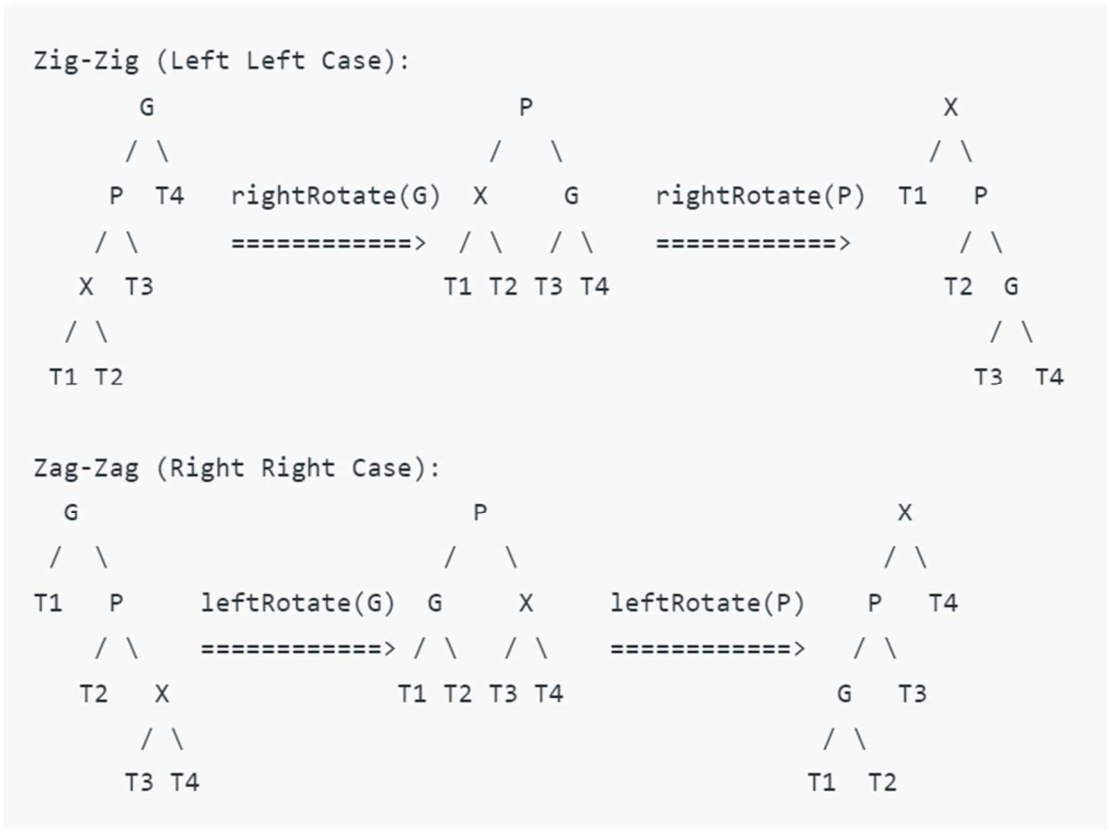
All splay tree operations run in O(log n) time on average, where n is the number of entries in the tree. Any single operation can take Theta(n) time in the worst case.

Search Operation

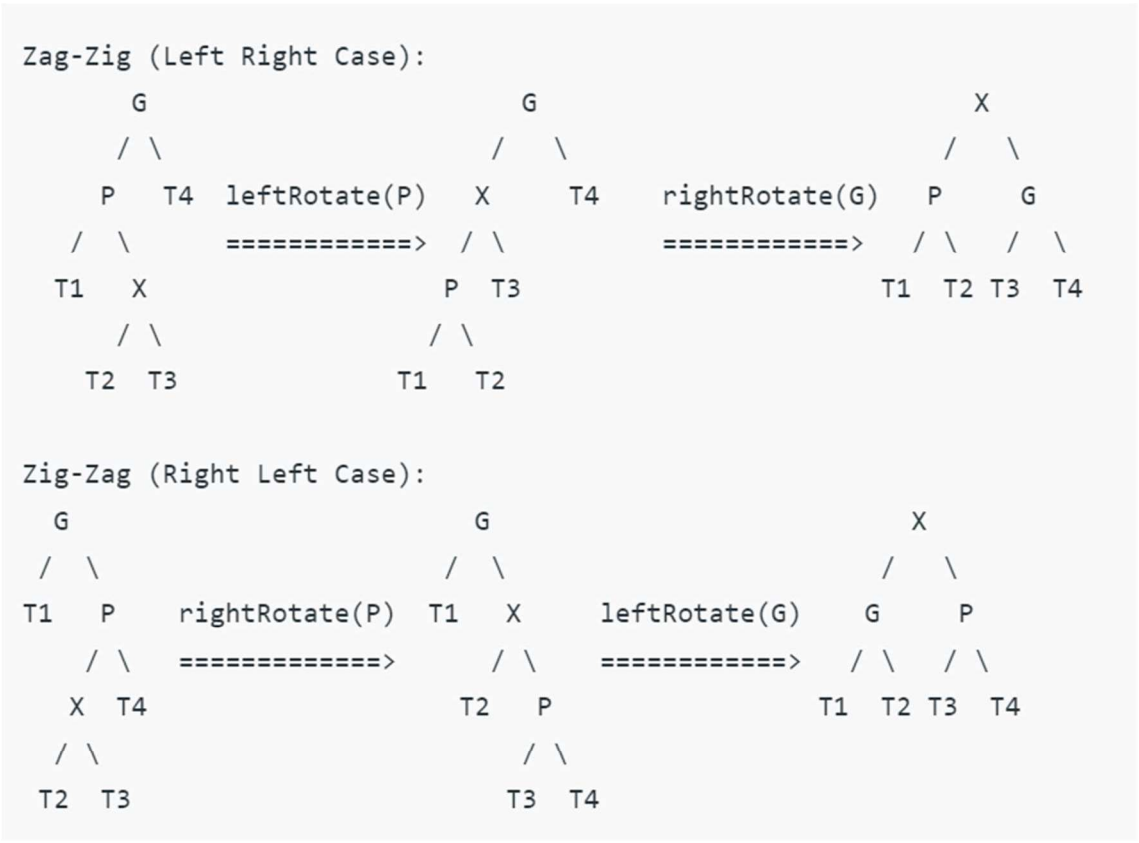
The search operation in Splay tree does the standard BST search, in addition to search, it also splays (move a node to the root). If the search is successful, then the node that is found is splayed and becomes the new root. Else the last node accessed prior to reaching the NULL is splayed and becomes the new root.
There are following cases for the node being accessed.
1) Node is root We simply return the root, don't do anything else as the accessed node is already root.
2) Zig: ***Node is child of root*** (the node has no grandparent). Node is either a left child of root (we do a right rotation) or node is a right child of its parent (we do a left rotation).
T1, T2 and T3 are subtrees of the tree rooted with y (on left side) or x (on right side)



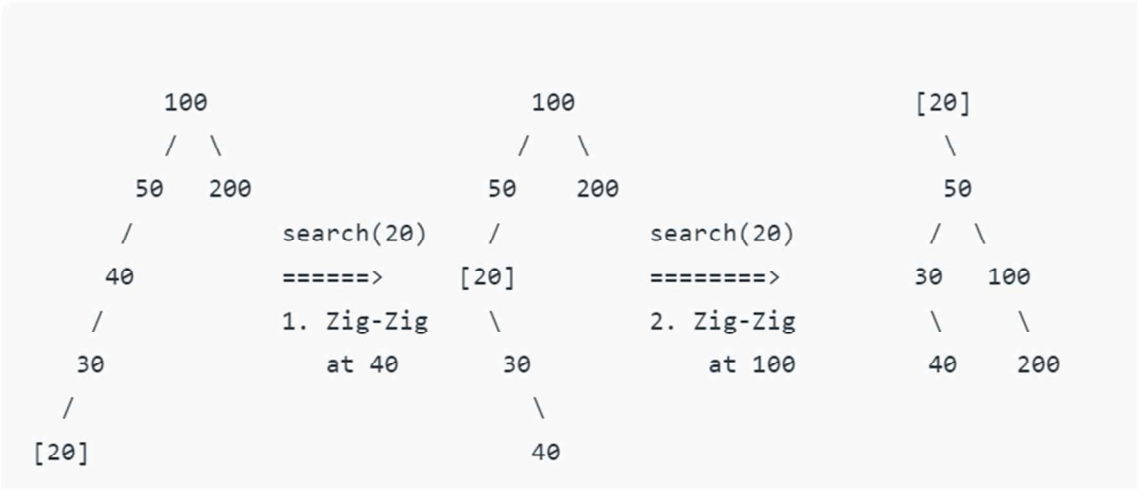
3) ***Node has both parent and grandparent***. There can be following subcases.
……..**3.a) Zig-Zig and Zag-Zag** Node is left child of parent and parent is also left child of grand parent (Two right rotations) OR node is right child of its parent and parent is also right child of grand parent (Two Left Rotations).



.**3.b) Zig-Zag and Zag-Zig** Node is right child of parent and parent is left child of grand parent (Left Rotation followed by right rotation) OR node is left child of its parent and parent is right child of grand parent (Right Rotation followed by left rotation).


Zag-Zig (Left Right Case):
G
P T4
T1 X
T2 T3
leftRotate(P)
G
X T4
P T3
T1 T2
rightRotate(G)
X
P G
T1 T2 T3 T4
Zig-Zag (Right Left Case):
G
T1 P
X T4
T2 T3
rightRotate(P)
G
T1 X
T2 P
T3 T4
leftRotate(G)
X
G P
T1 T2 T3 T4

EXAMPLE:


100
50 200
40
30
[20]
search(20)
======>
1. Zig-Zig
at 40
100
50 200
[20]
30
40
search(20)
========>
2. Zig-Zig
at 100
[20]
50
30 100
40 200

C CODE:

```
//Starting Module:
#include <stdio.h>
#include <stdlib.h>
typedef struct node
{
   int ipAdd;
   int dataPacket;
   struct node *left;
   struct node *right;
   struct node *parent;
} node;


typedef struct splay_tree
{
   struct node *root;
} splay_tree;
// This is the starting module which includes the library files needed and the defining of the structures.
//Process module
node *new_node(int ipAdd)
{
   node *n = malloc(sizeof(node));
   n->ipAdd = ipAdd;
   n->parent = NULL;
   n->right = NULL;
   n->left = NULL;
   return n;
}


splay_tree *new_splay_tree()
{
```

```
    splay_tree *t = malloc(sizeof(splay_tree));
    t->root = NULL;
    return t;
}

node *maximum(splay_tree *t, node *x)
{
    while (x->right != NULL)
        x = x->right;
    return x;
}

void left_rotate(splay_tree *t, node *x)
{
    node *y = x->right;
    x->right = y->left;
    if (y->left != NULL)
    {
        y->left->parent = x;
    }
    y->parent = x->parent;
    if (x->parent == NULL)
    {
        t->root = y;
    }
    else if (x == x->parent->left)
    {
        x->parent->left = y;
    }
    else
    {
```

```
        x->parent->right = y;
    }
    y->left = x;
    x->parent = y;
}

void right_rotate(splay_tree *t, node *x)
{
    node *y = x->left;
    x->left = y->right;
    if (y->right != NULL)
    {
        y->right->parent = x;
    }
    y->parent = x->parent;
    if (x->parent == NULL)
    {
        t->root = y;
    }
    else if (x == x->parent->right)
    {
        x->parent->right = y;
    }
    else
    {
        x->parent->left = y;
    }
    y->right = x;
    x->parent = y;
}
```

```
void splay(splay_tree *t, node *n)
{
   while (n->parent != NULL)
   {
      if (n->parent == t->root)
      {
         if (n == n->parent->left)
         {
            right_rotate(t, n->parent);
         }
         else
         {
            left_rotate(t, n->parent);
         }
      }
      else
      {
         node *p = n->parent;
         node *g = p->parent;
         if (n->parent->left == n && p->parent->left == p)
         {
            right_rotate(t, g);
            right_rotate(t, p);
         }
         else if (n->parent->right == n && p->parent->right == p)
         {
            left_rotate(t, g);
            left_rotate(t, p);
         }
         else if (n->parent->right == n && p->parent->left == p)
         {
```

```
            left_rotate(t, p);
            right_rotate(t, g);
        }
        else if (n->parent->left == n && p->parent->right == p)
        {
            right_rotate(t, p);
            left_rotate(t, g);
        }
    }
  }
}

void insert(splay_tree *t, node *n)
{
  node *y = NULL;
  node *temp = t->root;
  while (temp != NULL)
  {
    y = temp;
    if (n->ipAdd < temp->ipAdd)
      temp = temp->left;
    else
      temp = temp->right;
  }
  n->parent = y;
  if (y == NULL)
    t->root = n;
  else if (n->ipAdd < y->ipAdd)
    y->left = n;
  else
    y->right = n;
```

```
    splay(t, n);
}

node *search(splay_tree *t, node *n, int x)
{
    if (x == n->ipAdd)
    {
        splay(t, n);
        return n;
    }
    else if (x < n->ipAdd)
        return search(t, n->left, x);
    else if (x > n->ipAdd)
        return search(t, n->right, x);
    else
        return NULL;
}

void inorder(splay_tree *t, node *n, char *cmn)
{
    if (n != NULL)
    {
        inorder(t, n->left, cmn);
        printf("%s%d -> %d\n", cmn, n->ipAdd,
                n->dataPacket);
        inorder(t, n->right, cmn);
    }
}
```

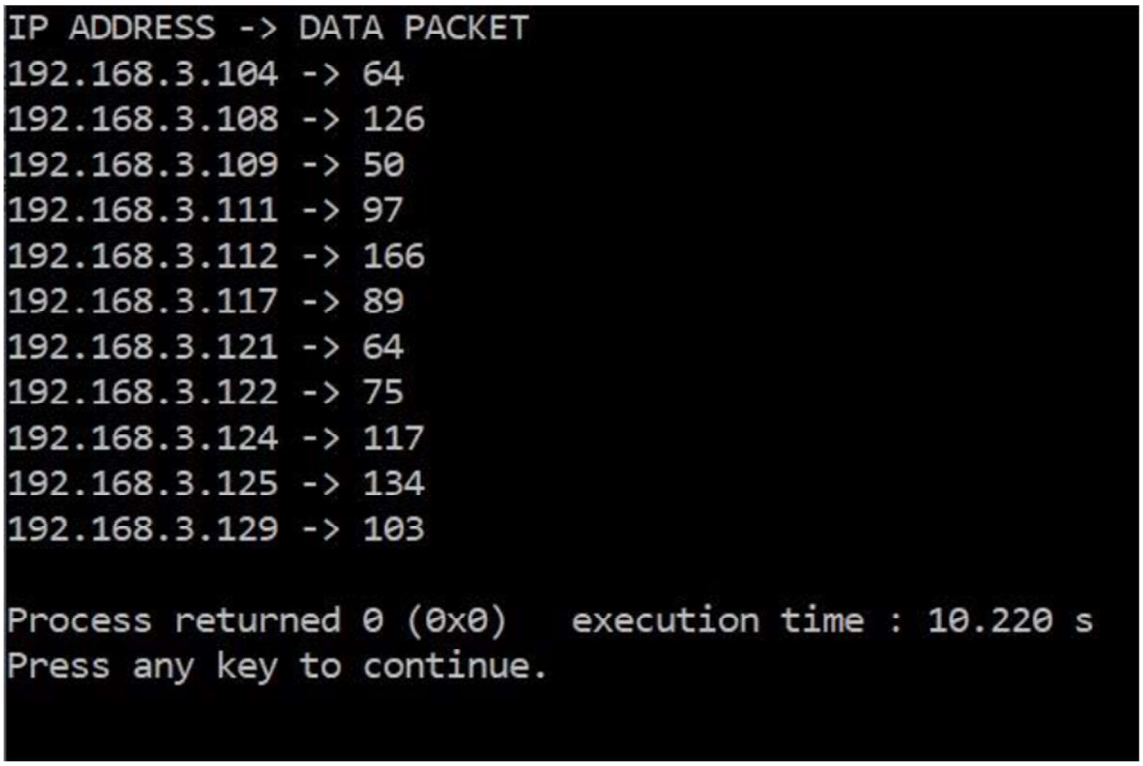
// This is the process module which includes all the functions (void, int, node* & splay_tree*) and processing takes place in these functions.

```
//Implementation Module
int main()
{
    char *cmn = "192.168.3.";
    splay_tree *t = new_splay_tree();
    node *a, *b, *c, *d, *e, *f, *g, *h, *i, *j, *k, *l, *m;
    a = new_node(104);
    b = new_node(112);
    c = new_node(117);
    d = new_node(124);
    e = new_node(121);
    f = new_node(108);
    g = new_node(109);
    h = new_node(111);
    i = new_node(122);
    j = new_node(125);
    k = new_node(129);
    insert(t, a);
    insert(t, b);
    insert(t, c);
    insert(t, d);
    insert(t, e);
    insert(t, f);
    insert(t, g);
    insert(t, h);
    insert(t, i);
    insert(t, j);
    insert(t, k);
    int x;
    int find[11] = {104, 112, 117, 124, 121, 108, 109, 111,
                    122, 125, 129};
```

```
    int add[11] = {a, b, c, d, e, f, g, h, i, j, k};
    srand(time(0));
    for (x = 0; x < 11; x++)
    {
        int data = rand() % 200;
        node *temp = search(t, add[x], find[x]);
        if (temp != NULL)
        {
            temp->dataPacket = data;
        }
    }
    printf("IP ADDRESS -> DATA PACKET\n");
    inorder(t, t->root, cmn);
    return 0;
}
```

// This is the implementation module of the int main(). This includes calling or implementing of the functions.

RESULT/OUTPUT:



This is the result obtained.

TIME COMPLEXITY ANALYSIS:

Function Name	NLOC	Complexity	Token #	Parameter #
new_node	8	1	46	
new_splay_tree	5	1	26	
maximum	5	2	29	
left_rotate	19	4	114	
right_rotate	19	4	114	
splay	32	12	225	
insert	19	5	110	
search	12	4	81	
inorder	7	2	61	
main	43	3	360	

All splay tree operations run in O(log n) time on average, where n is the number of entries in the tree. Any single operation can take Theta(n) time in the worst case.

CONCLUSION

The algorithm was successfully designed which makes searching fast was successfully designed. When multiple devices are connected in one router and millions of data packets are being sent in time of seconds, splay tree is the most convenient data structure to be used in such fields.

REFERENCES

- Geeks for Geeks
- Tutorial points
- Codeforces
- Thomas H cormen, Charles E Leiserson, Ronald L Revest, Clifford Stein, Introduction to Algorithms, 3rd., The MIT Press Cambridge, 2014