DSA PROJECT Report

Title: IP Address Finder

Sourav Payla (E21CSEU0846)

Batch:Eb29

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

Objective: To use splay trees (fastest data structure) to achieve the aim.

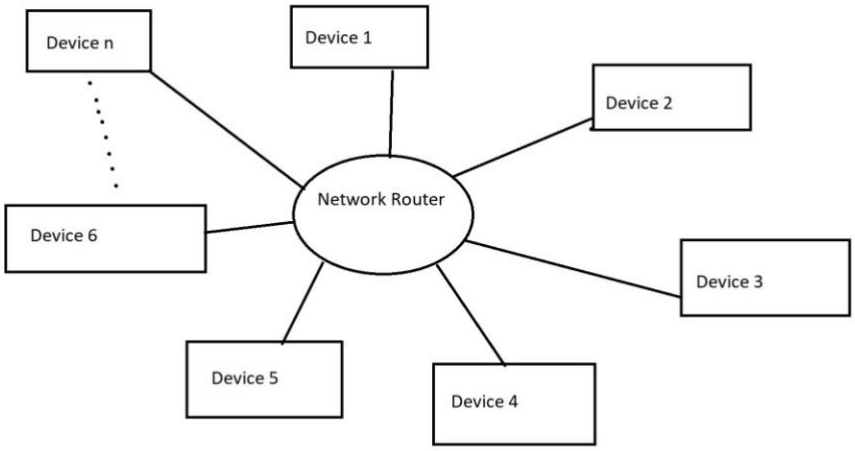
Abstract: With the help of splay tree data structure, we would create a tree whose nodes are embedded with the Ip address of the device that are connect to a specific network router. In our code we have taken 11 devices connected to one network router and so there would be some common part in the Ip 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.

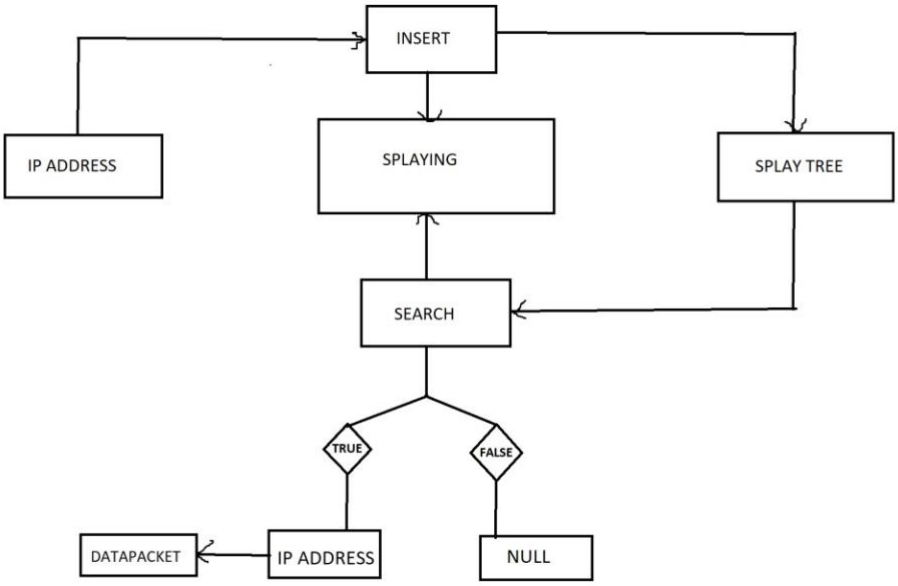
Basic principle: it uses the principle that the most occuring Ip address stays at the top and so the time complexity of searching decreases eventually.

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

Architecture:



Block Diagram:



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.

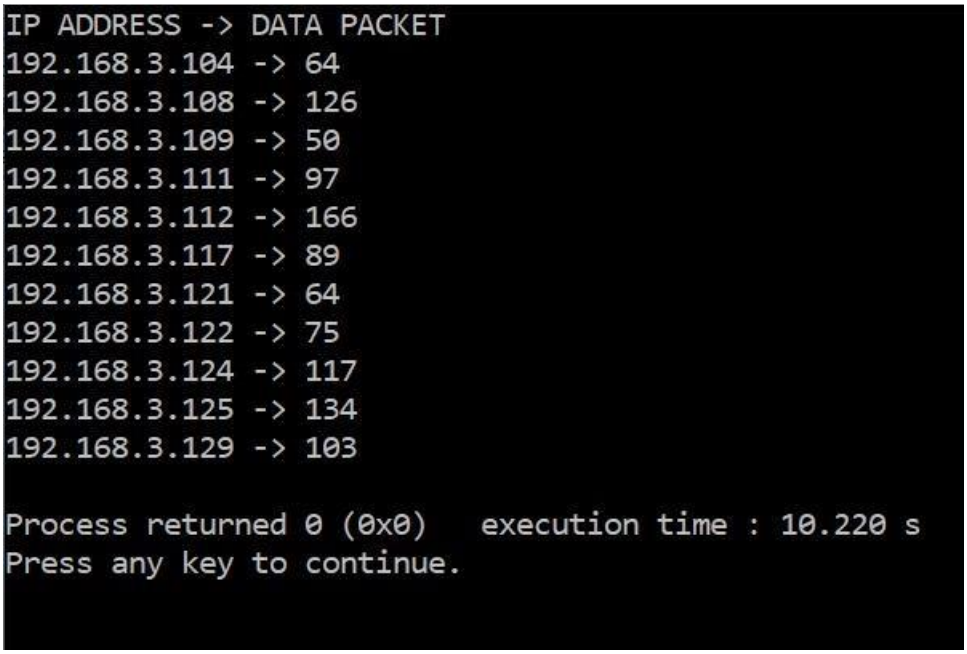
Implementation Module:

```
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	

Conclusion: 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:

www.codesdope.com

Jenny's lecture – Youtube.