### IP ADDRESS FINDER

### A PROJECT REPORT

*Submitted by*

**ABHAY SHAJI [Reg No: RA2112704010006]**

**SUJAL SINGH [Reg No: RA2112704010001]**

**ABHAY SHAJI [Reg No: RA2112704010017]**

*Under the Guidance of*

### Dr. PAUL T SHEEBA

#### (Assistant Professor, Department of Data Science and Business Systems)

*In partial fulfillment of the Requirements f +or the Degree of*

## BACHELOR OF TECHNOLOGY COMPUTER SCIENCE AND BUSINESS SYSTEMS



**DEPARTMENT OF DATA SCIENCE AND BUSINESS SYSTEMS**

## FACULTY OF ENGINEERING AND TECHNOLOGY SRM INSTITUTE OF SCIENCE AND TECHNOLOGY NOVEMBER 2022

Department of Data Science and Business Systems

###### SRM Institute of Science & Technology Own Work\* Declaration Form

To be completed by the student for all assessments

###### Degree/ Course :

**Student Name :ABHAY SHAJI**

###### Registration Number :RA2112704010006

**Title of Work :IP ADDRESS FINDER**

We hereby certify that this assessment complies with the University’s Rules and Regulations relating to Academic misconduct and plagiarism\*\*, as listed in the University Website, Regulations, and the Education Committee guidelines.

We confirm that all the work contained in this assessment is our own except where indicated, and that we have met the following conditions:

* Clearly references / listed all sources as appropriate
* Referenced and put in inverted commas all quoted text (from books, web, etc)
* Given the sources of all pictures, data etc. that are not my own
* Not made any use of the report(s) or essay(s) of any other student(s) either past or present
* Acknowledged in appropriate places any help that I have received from others (e.g. fellow students, technicians, statisticians, external sources)
* Compiled with any other plagiarism criteria specified in the Course handbook / University website

We understand that any false claim for this work will be penalized in accordance with the University policies and regulations.

|  |
| --- |
| **DECLARATION:** |
| I am aware of and understand the University’s policy on Academic misconduct and plagiarism and I certify that this assessment is my / our own work, except where indicated by referring, and that I have followed the good academic practices noted above. |
| If you are working in a group, please write your registration numbers and sign with the date for every student in your group. |

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY KATTANKULATHUR-603203

BONAFIDE CERTIFICATE

Certified that this project report titled “**IP Address finder”** is the bonafide work of **“ABHAY SHAJI [Reg No:RA2112704010006],SUJAL SINGH[Reg No:RA2112704010001] AND VAMSHI GADDE [Reg No:RA2112704010006]**who carried out the project work under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form part of any other thesis or dissertation on the basis of which a degree or award was conferred on an earlier occasion for this or any other candidate.

Dr. Paul T Sheeba Dr. M. Lakshmi

##### GUIDE HEAD OF THE DEPARTMENT

Assistant Professor Dept. of DSBS Dept. of DSBS

Signature of Internal Examiner Signature of External Examiner

**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.

### ACKNOWLEDGEMENTS

We express our humble gratitude to **Dr C. Muthamizhchelvan**, Vice-Chancellor, SRM Institute of Science and Technology, for the facilities extended for the project work and his continued support. We extend our sincere thanks to Dean-CET, SRM Institute of Science and Technology, **Dr T.V.Gopal**, for his invaluable support.

We wish to thank **Dr Revathi Venkataraman**, Professor & Chairperson, School of Computing, SRM Institute of Science and Technology, for her support throughout the project work. We are incredibly grateful to our Head of the Department**, Dr M. Lakshmi** Professor, Department of Data Science and Business Systems, SRM Institute of Science and Technology, for her suggestions and encouragement at all the stages of the project work. We want to convey our thanks to our program coordinator **Dr.E.Sasikala,** Professor, Department of Data Science and Business Systems, SRM Institute of Science and Technology, for her input during the project reviews and support.

We register our immeasurable thanks to our Faculty Advisor**, Dr. R Rajkumar,** Assistant Professor, DSBS, SRM Institute of Science and Technology, for leading and helping us to complete our course.

Our inexpressible respect and thanks to my guide, **Dr. Paul T Sheeba,** Assistant Professor, DSBS, SRM Institute of Science and Technology, for providing me with an opportunity to pursue my project under his mentorship. He provided me with the freedom and support to explore the research topics of my interest. His passion for solving problems and making a difference in the world has always been inspiring. We sincerely thank the Data Science and Business Systems staff and students, SRM Institute of Science and Technology, for their help during our project. Finally, we would like to thank parents, family members, and friends for their unconditional love, constant support, and encouragement.

ABHAY SAHJI

SUJAL SINGH

VAMSHI GADDE

## TABLE OF CONTENTS

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| TABLE OF CONTENTS | | | | |
| CHAPTER NO. | | | TITLE | PAGE NO. |
|  | ABSTRACT | |  | iii |
|  | LIST OF TABLES | |  | xvii |
|  | LIST OF FIGURES | |  | xviii |
|  | LIST OF ABBREVIATIONS | |  | ix |
| 1. | INTRODUCTION | |  |  |
|  | 1.1 | SPLAY-TREE |  | 1 |
| 2 | LITERATURE SURVEY | |  |  |
| 3 | SYSTEM DESIGN  3.1 BLOCK DIAGRAM | |  |  |
|  | 3.2 | ARCHITECTURE |  |  |

|  |  |  |
| --- | --- | --- |
| 4 | OUTPUT | 47 |
| 5 | CONCLUSION 48 | |
| 6 | FUTURE ENHANCEMENTS 49 | |
| 7 | REFERENCES 50 | |
|  | APPENDIX 51 | |
|  |  |  |

|  |  |
| --- | --- |
|  | **LIST OF FIGURES** |
| 4.1.1 | LAYERED ARCHITECTURE OF SKILL VERIFY PLATFORM… 11 |
| 4.3.1 | THE SYSTEM ARCHITECTURE OF SKILLVERIFY 16 |
| 4.3.2 | OVERVIEW OF THE SMART CONTRACTS OF SKILLCHECK 17 |
|  |  |

## ABBREVIATIONS

|  |  |
| --- | --- |
| **MOOC** | Massive Open Online Courses |
| **Dapps** | Decentralized Apps |
| **JS** | JavaScript |
| **IPC** | Inter-process communication |
| **SC** | Smart Contract |
| **MVC** | Model View Controller |
| **DMA** | Trusted third parties |
| **IETF** | The Internet Engineering Task Force |

--

**CHAPTER 1**

**INTRODUCTION**

**1.1 Splay Tree:**

A **splay tree** is a binary search tree with the additional property that recently accessed elements are quick to access again. Like self-balancing binary search trees, a splay tree performs basic operations such as insertion, look-up and removal in (log *n*) amortized time. For random access patterns drawn from a non-uniform random distribution, their amortized time can be faster than logarithmic, proportional to the entropy of the access pattern. For many patterns of non-random operations, also, splay trees can take better than logarithmic time, without requiring advance knowledge of the pattern. According to the unproven dynamic optimality conjecture, their performance on all access patterns is within a constant factor of the best possible performance that could be achieved by any other self-adjusting binary search tree, even one selected to fit that pattern. The splay tree was invented by Daniel Sleator and Robert Tarjan in 1985.

All normal operations on a binary search tree are combined with one basic operation, called *splaying*. Splaying the tree for a certain element rearranges the tree so that the element is placed at the root of the tree. One way to do this with the basic search operation is to first perform a standard binary tree search for the element in question, and then use tree rotations in a specific fashion to bring the element to the top. Alternatively, a top-down algorithm can combine the search and the tree reorganization into a single phase.

### General Objective

To use splay trees (fastest data structure) to achieve an algorithm which can search through data as fast as possible . it uses the principle that the most occuring Ip address stays at the top and so the time complexity of searching decreases eventually. to keep the most recurring element on the root node the data structure uses splaying operation.

## CHAPTER 2

## LITERATURE STUDY

# 2.1 Splay Tree

Like the AVL tree, the splay tree is not actually a distinct data structure, but rather reimplements the BST insert, delete, and search methods to improve the performance of a BST. The goal of these revised methods is to provide guarantees on the time required by a series of operations, thereby avoiding the worst-case linear time behavior of standard BST operations. No single operation in the splay tree is guaranteed to be efficient. Instead, the splay tree access rules guarantee that a series of mm operations will take O(mlogn)O(mlogn) time for a tree of nn nodes whenever m≥nm≥n. Thus, a single insert or search operation could take O(n)O(n) time. However, mm such operations are guaranteed to require a total of O(mlogn)O(mlog⁡n) time, for an average cost of O(logn)O(log⁡n) per access operation. This is a desirable performance guarantee for any search-tree structure.

Unlike the AVL tree, the splay tree is not guaranteed to be height balanced. What is guaranteed is that the total cost of the entire series of accesses will be cheap. Ultimately, it is the cost of the series of operations that matters, not whether the tree is balanced. Maintaining balance is really done only for the sake of reaching this time efficiency goal.

The splay tree access functions operate in a manner reminiscent of the [move-to-front](https://opendsa-server.cs.vt.edu/ODSA/Books/Everything/html/Glossary.html#term-move-to-front) rule for [self-organizing lists](https://opendsa-server.cs.vt.edu/ODSA/Books/Everything/html/SelfOrg.html#selforg), and of the path compression technique for managing a series of [Union/Find](https://opendsa-server.cs.vt.edu/ODSA/Books/Everything/html/UnionFind.html#unionfind) operations. These access functions tend to make the tree more balanced, but an individual access will not necessarily result in a more balanced tree.

Whenever a node SS is accessed (e.g., when SS is inserted, deleted, or is the goal of a search), the splay tree performs a process called [splaying](https://opendsa-server.cs.vt.edu/ODSA/Books/Everything/html/Glossary.html#term-splaying). Splaying moves SS to the root of the BST. When SS is being deleted, splaying moves the parent of SS to the root. As in the AVL tree, a splay of node SS consists of a series of [rotations](https://opendsa-server.cs.vt.edu/ODSA/Books/Everything/html/Glossary.html#term-rotation). A rotation moves SS higher in the tree by adjusting its position with respect to its parent and grandparent. A side effect of the rotations is a tendency to balance the tree. There are three types of rotation.

A [single rotation](https://opendsa-server.cs.vt.edu/ODSA/Books/Everything/html/Glossary.html#term-single-rotation) is performed only if SS is a child of the root node. The single rotation is illustrated by Figure [26.3.1](https://opendsa-server.cs.vt.edu/ODSA/Books/Everything/html/Splay.html#singprom). It basically switches SS with its parent in a way that retains the BST property. While Figure [26.3.1](https://opendsa-server.cs.vt.edu/ODSA/Books/Everything/html/Splay.html#singprom)

 is slightly different from Figure [26.2.2](https://opendsa-server.cs.vt.edu/ODSA/Books/Everything/html/AVL.html#avlsingle), in fact the splay tree single rotation is identical to the AVL tree single rotation.

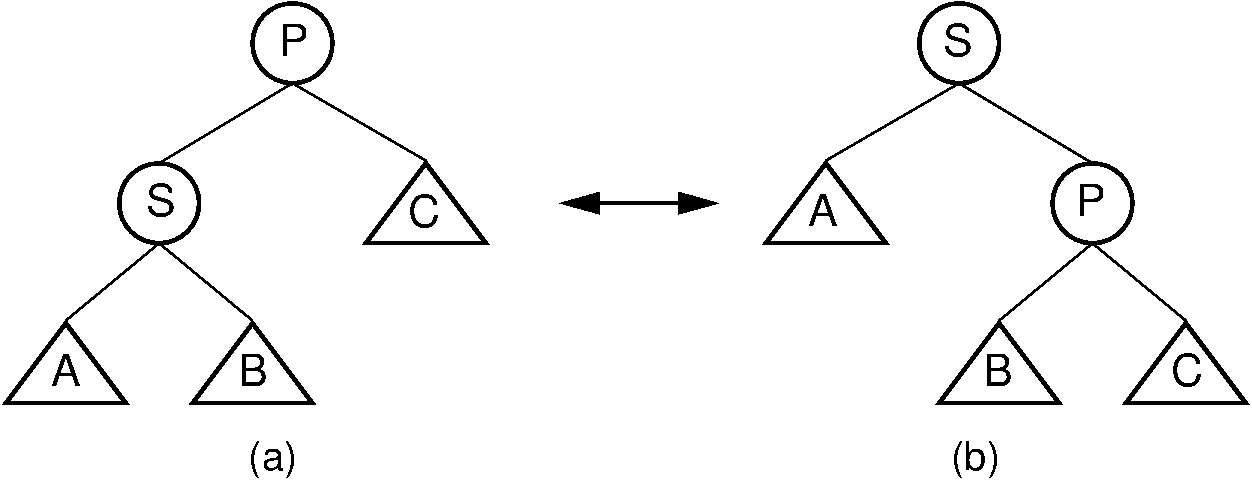
**[](https://opendsa-server.cs.vt.edu/ODSA/Books/Everything/html/_images/SingRot.png)**

Figure 26.3.1: Splay tree single rotation.

This rotation takes place only when the node being splayed is a child of the root. Here, node SS is promoted to the root, rotating with node PP. Because the value of SS is less than the value of PP, PP must become SS ‘s right child. The positions of subtrees AA, BB, and ;math:C are altered as appropriate to maintain the BST property, but the contents of these subtrees remains unchanged. (a) The original tree with PP as the parent. (b) The tree after a rotation takes place. Performing a single rotation a second time will return the tree to its original shape. Equivalently, if (b) is the initial configuration of the tree (i.e., SS is at the root and PP is its right child), then (a) shows the result of a single rotation to splay PP to the root.

Unlike the AVL tree, the splay tree requires two types of double rotation. Double rotations involve SS, its parent (call it PP), and SS ‘s grandparent (call it GG). The effect of a double rotation is to move SS up two levels in the tree.

The first double rotation is called a zigzagrotationzigzagrotation. It takes place when either of the following two conditions are met:

1. SS is the left child of PP, and PP is the right child of GG.
2. SS is the right child of PP, and PP is the left child of GG.

In other words, a zigzag rotation is used when GG, PP, and SS form a zigzag. The zigzag rotation is illustrated by Figure .

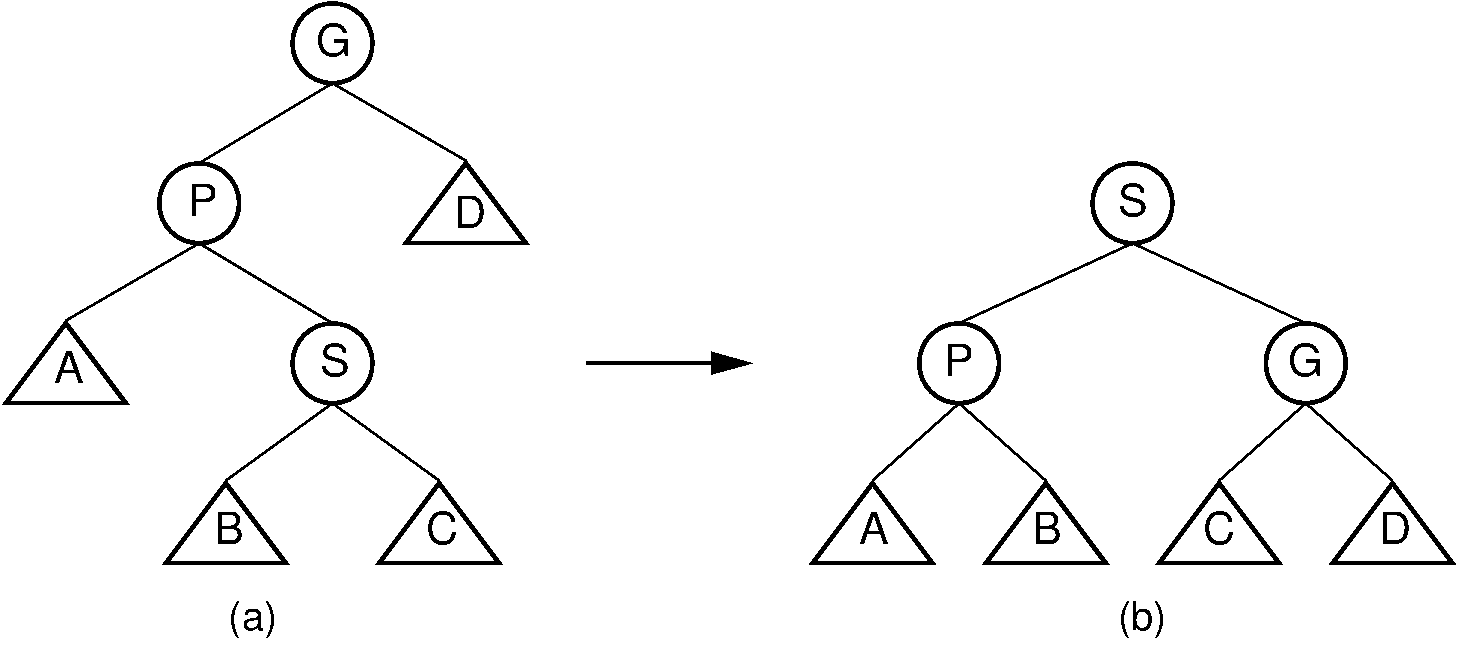
**[](https://opendsa-server.cs.vt.edu/ODSA/Books/Everything/html/_images/ZigZag.png)**

Figure 26.3.2: Splay tree zigzag rotation. (a) The original tree with SS, PP, and GG in zigzag formation. (b) The tree after the rotation takes place. The positions of subtrees AA, BB, CC, and DD are altered as appropriate to maintain the BST property.

The other double rotation is known as a [zigzig](https://opendsa-server.cs.vt.edu/ODSA/Books/Everything/html/Glossary.html" \l "term-zigzig" \t "_parent) rotation. A zigzig rotation takes place when either of the following two conditions are met:

1. SS is the left child of PP, which is in turn the left child of GG.
2. SS is the right child of PP, which is in turn the right child of GG.

Thus, a zigzig rotation takes place in those situations where a zigzag rotation is not appropriate. The zigzig rotation is illustrated by Figure [26.3.3](https://opendsa-server.cs.vt.edu/ODSA/Books/Everything/html/Splay.html#zigzig). While Figure [26.3.3](https://opendsa-server.cs.vt.edu/ODSA/Books/Everything/html/Splay.html#zigzig) appears somewhat different from Figure [26.2.3](https://opendsa-server.cs.vt.edu/ODSA/Books/Everything/html/AVL.html#avldouble), in fact the zigzig rotation is identical to the AVL tree double rotation.

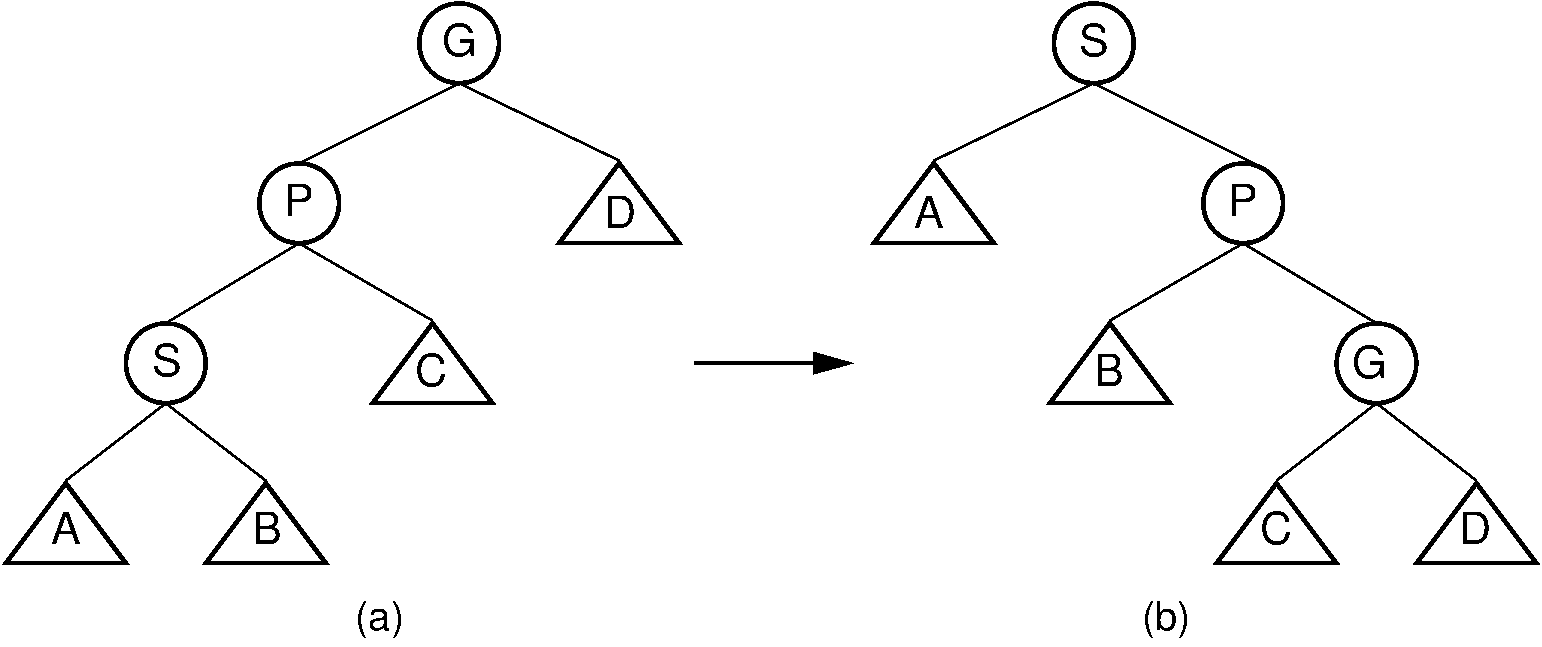
**[](https://opendsa-server.cs.vt.edu/ODSA/Books/Everything/html/_images/ZigZig.png)**

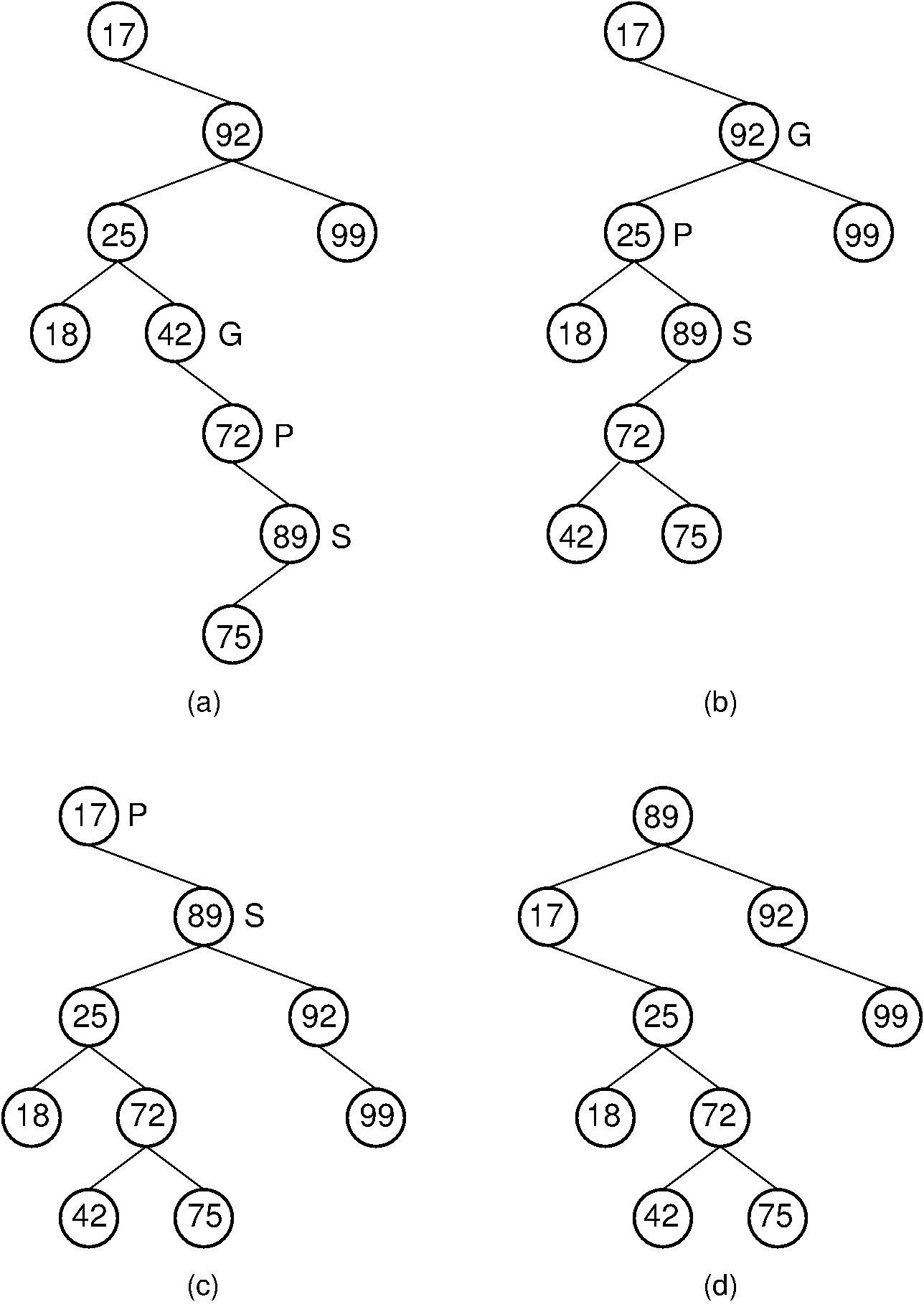
Figure 26.3.3: Splay tree zigzig rotation. (a) The original tree with SS, PP, and GG in zigzig formation. (b) The tree after the rotation takes place. The positions of subtrees AA, BB, CC, and DD are altered as appropriate to maintain the BST property.

Note that zigzag rotations tend to make the tree more balanced, because they bring subtrees BB and CC up one level while moving subtree DD down one level. The result is often a reduction of the tree’s height by one. Zigzig promotions and single rotations do not typically reduce the height of the tree; they merely bring the newly accessed record toward the root.

Splaying node SS involves a series of double rotations until SS reaches either the root or the child of the root. Then, if necessary, a single rotation makes SS the root. This process tends to re-balance the tree. Regardless of balance, splaying will make frequently accessed nodes stay near the top of the tree, resulting in reduced access cost. Proof that the splay tree meets the guarantee of O(mlogn)O(mlog⁡n) is beyond the scope of our study.

**Example 26.3.1**

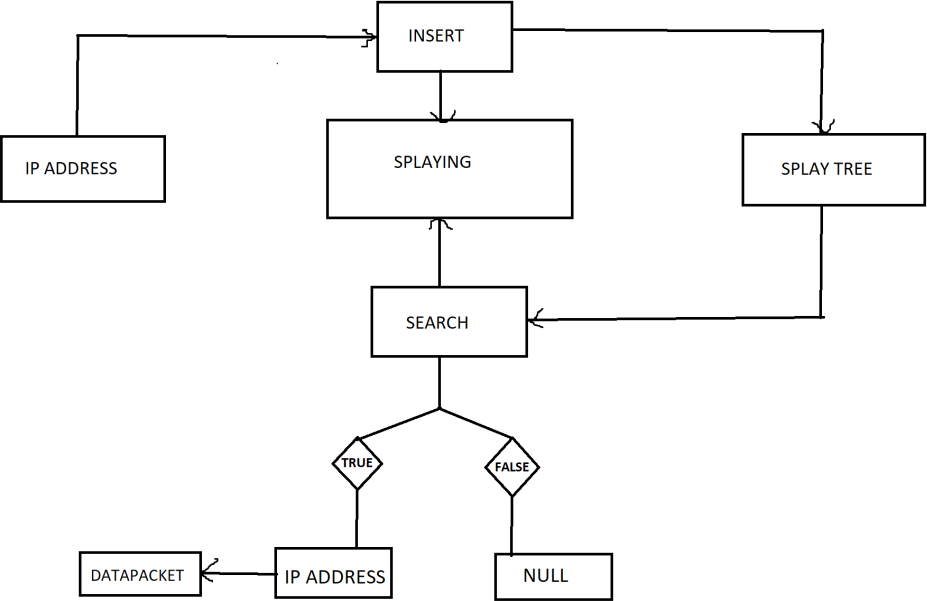
Consider a search for value 89 in the splay tree of Figure [26.3.4](https://opendsa-server.cs.vt.edu/ODSA/Books/Everything/html/Splay.html#splayex) (a). The splay tree’s search operation is identical to searching in a BST. However, once the value has been found, it is splayed to the root. Three rotations are required in this example. The first is a zigzig rotation, whose result is shown in Figure [26.3.4](https://opendsa-server.cs.vt.edu/ODSA/Books/Everything/html/Splay.html#splayex) (b). The second is a zigzag rotation, whose result is shown in Figure [26.3.4](https://opendsa-server.cs.vt.edu/ODSA/Books/Everything/html/Splay.html#splayex) (c). The final step is a single rotation resulting in the tree of Figure [26.3.4](https://opendsa-server.cs.vt.edu/ODSA/Books/Everything/html/Splay.html#splayex) (d). Notice that the splaying process has made the tree shallower.

**[](https://opendsa-server.cs.vt.edu/ODSA/Books/Everything/html/_images/SplayEx.png)**Figure 26.3.4: Example of splaying after performing a search in a splay tree. After finding the node with key value 89, that node is splayed to the root by performing three rotations. (a) The original splay tree. (b) The result of performing a zigzig rotation on the node with key value 89 in the tree of (a). (c) The result of performing a zigzag rotation on the node with key value 89 in the tree of (b). (d) The result of performing a single rotation on the node with key value 89 in the tree of (c). If the search had been for 91, the search would have been unsuccessful with the node storing key value 89 being that last one visited. In that case, the same splay operations would take place.

## CHAPTER 3

## SYSTEM DESIGN

**3.1 Block Diagram:**



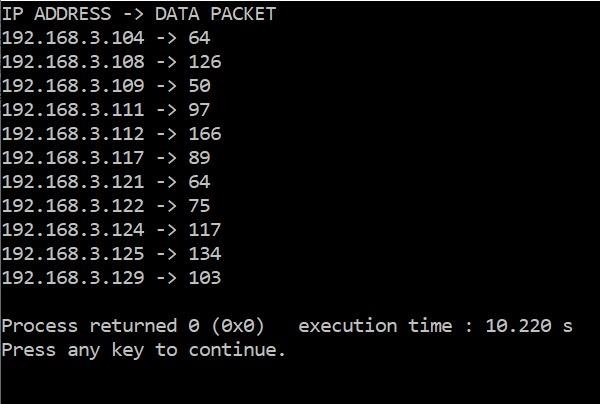
**3.2ARCHITECTURE**

### 

4.3.2 The System Architecture of SkillVerify

**CHAPTER 4**

**OUTPUT**



**FIG 4.1**

**CHAPTER 5**

**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.

## CHAPTER 6

## FUTURE ENHANCEMENT

The scale of the app market is gradually growing and has reached billions of dollars. In this study, we conducted a systematic descriptive analysis of 734 the apps over Ethereum. We find relationships among the app popularity, categories, open source and LoC, and explore reuse of smart contracts and source codes. Our findings provide valuable implications for different stakeholders in the decentralized application industry and in the research community of blockchain and decentralized applications. This paper mainly focuses on the descriptive analysis of the the apps. In future, we will focus questions about the app projects and source codes, such as how to build a the app project, how to keep synchronization on and off the chain, and how to improve throughput of the apps.

In general, source codes of dapps can be divided into two parts: smart contracts and the others. After development, smart contracts are compiled to bytecodes and then deployed to blockchain. Dapps provide end-users with services through UIs, do some operations and interact with blockchain through smart contracts. So open sources of dapps can be divided into two parts as well. Any smart contracts can be gotten from blockchain straightly, but they are only bytecodes. Developers can submit source codes to block explorers or open source code repositories to make smart contracts open-source.

## CHAPTER7

## REFERENCES

[2.1] Pilkington, Marc. "Blockchain technology: principles and applications." *Research handbook on digital transformations*. Edward Elgar Publishing, 2016.

[2.2] Yaga, Dylan, et al. "Blockchain technology overview." *arXiv preprint arXiv:1906.11078* (2019).

[2.3] Khandelwal, Harshita, et al. "Certificate verification system using blockchain." *Advances in Cybernetics, Cognition, and Machine Learning for Communication Technologies*. Springer, Singapore, 2020. 251-257.

[2.4] Adja, Yves Christian Elloh, et al. "A blockchain-based certificate revocation management and status verification system." *Computers & Security* 104 (2021): 102209.

|  |
| --- |
|  |
|  |
|  |  |

## APPENDIX

**SourceCode:**

## *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;*

## *}*