Name: Shehab Hasanul Haque
ID: u2180613
Algorithms and Data Structure

Practical 11:

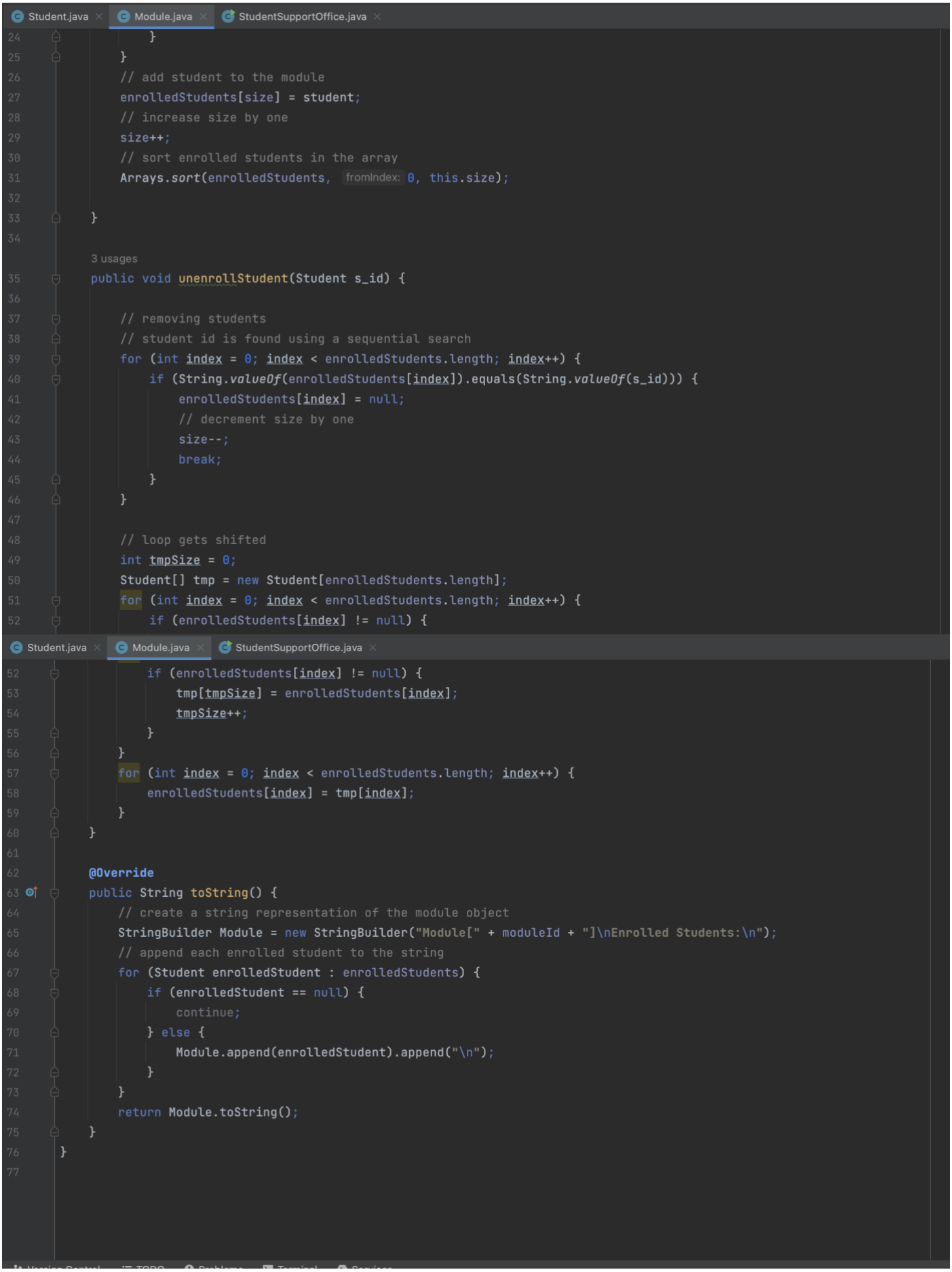
This is a student support office, which stores the Module ID and Student ID. Students will be enrolled on modules and then according to the assignment, some students will be disenrolled to a particular module and again enrolled. This project contains 3 classes:

1. Module
2. Student
3. StudentSupportOffice

Module:

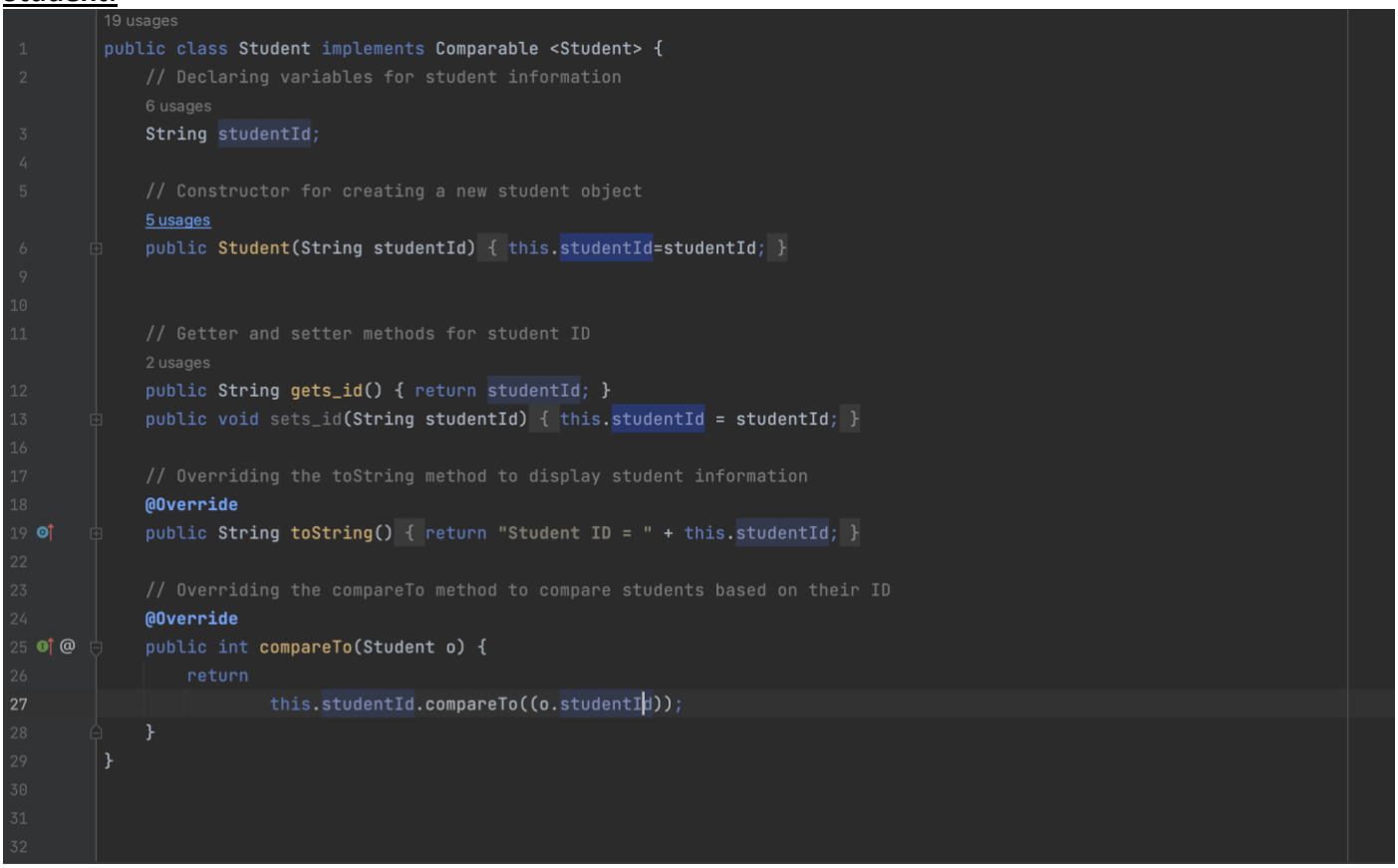
Student.java | Module.java | StudentSupportOffice.java

```
1   import java.util.ArrayList;
2   import java.util.Arrays;
3
    6 usages
4   public class Module {
5       // module identifier
        2 usages
6       String moduleId;
7       // array of students enrolled in the module
        14 usages
8       Student[] enrolledStudents;
9       // number of students currently enrolled in the module
        6 usages
10      int size;
11
        3 usages
12      public Module(String moduleId) {
13          this.moduleId = moduleId;
14          this.enrolledStudents = new Student[5];
15          this.size = 0;
16      }
17
        14 usages
18      public void enrolledStudents(Student student) {
19          //checking if student is already enrolled in the module
20          for (int index = 0; index < size; index++) {
21              if (enrolledStudents[index].gets_id().equals(student.gets_id())) {
22                  System.out.println("\nError during enrollment\nStudent enrolled already\n\n");
23                  return;
24              }
```



This class is called Module which illustrates a module in a system for enrolling students. The module identification, the array of students enrolled in the module, and the total number of students enrolled in the module are all fields in the class. It provides methods for enrolling and dropping students, as well as one that returns a string that represents the module and the students who are enrolled in it.

Student:



```
19 usages
public class Student implements Comparable <Student> {
    // Declaring variables for student information
    6 usages
    String studentId;

    // Constructor for creating a new student object
    5 usages
    public Student(String studentId) { this.studentId=studentId; }


    // Getter and setter methods for student ID
    2 usages
    public String gets_id() { return studentId; }
    public void sets_id(String studentId) { this.studentId = studentId; }

    // Overriding the toString method to display student information
    @Override
    public String toString() { return "Student ID = " + this.studentId; }

    // Overriding the compareTo method to compare students based on their ID
    @Override
    public int compareTo(Student o) {
        return
                this.studentId.compareTo((o.studentId));
    }
}
```

This class 'Student' represents the students in a student enrolment system. This class provides methods to acquire and set the student's ID as well as a field for the student's ID. Additionally, the Student class provides two methods: compareTo and function toString() that compares two students based on their IDs. The function toString() method returns a string representation of the student. The compareTo method should be implemented since the student class implements the comparable interface. This makes it possible to sort the students according to their ID.

StudentSupportOffice:

```java
import java.util.Collections;

public class StudentSupportOffice {
    public static void main(String[] args) {
        // Create student objects
        Student student1 = new Student( studentId: "U0000001");
        Student student2 = new Student( studentId: "U0000002");
        Student student3 = new Student( studentId: "U0000003");
        Student student4 = new Student( studentId: "U0000004");
        Student student5 = new Student( studentId: "U0000005");

        // Creating module objects
        Module module1 = new Module( moduleId: "CIS2206");
        Module module2 = new Module( moduleId: "CIS2360");
        Module module3 = new Module( moduleId: "CIS2205");

        // Enrolling students in modules
        module1.enrolledStudents(student1);
        module1.enrolledStudents(student5);

        module2.enrolledStudents(student1);
        module2.enrolledStudents(student3);
        module2.enrolledStudents(student4);

        module3.enrolledStudents(student2);
        module3.enrolledStudents(student4);
        module3.enrolledStudents(student5);

        System.out.println("Students before modification:" + "\n" + module3 + "\n" + module2 + "\n" + module1);

        module3.enrolledStudents(student5);

        System.out.println("Students before modification:" + "\n" + module3 + "\n" + module2 + "\n" + module1);

        // Disenrolling students from modules
        module1.unenrollStudent(student5);

        module1.enrolledStudents(student3);
        module1.enrolledStudents(student4);

        // Enrolling and disenrolling students in modules
        module2.enrolledStudents(student2);
        module2.enrolledStudents(student5);

        module3.unenrollStudent(student2);
        module3.unenrollStudent(student5);
        module3.enrolledStudents(student1);
        module3.enrolledStudents(student3);

        System.out.println("Students after modification:" + "\n" + module3 + "\n" + module2 + "\n" + module1);


    }
}
```

This is a class called StudentSupportOffice whose primary method creates several Student and Module objects and enrols or drops students into modules. The list of students in each module, both before and after the enrolment/disenrollment actions. Finally, the output is printed.

Practical 12:

This project contains 3 classes:

1. Dashboard
2. FootballLeague
3. Team

Dashboard:

```java
public class Dashboard {
    // this is the main method to run the program
    public static void main(String[] args) {
        // creating teams
        Team team1 = new Team( team_Name: "FC Barcelona");
        Team team2 = new Team( team_Name: "FC Bayern Munich");
        Team team3 = new Team( team_Name: "Chelsea FC");

        Team team4 = new Team( team_Name: "Paris Saint-Germain FC");
        Team team5 = new Team( team_Name: "Real Madrid CF");
        Team team6 = new Team( team_Name: "Man CIty FC");

        // creating the football league
        FootballLeague footballLeague = new FootballLeague("UEFA Champions League");

        // teams are being added to the league
        footballLeague.addFootballTeam(team1);
        footballLeague.addFootballTeam(team2);
        footballLeague.addFootballTeam(team3);
        footballLeague.addFootballTeam(team4);
        footballLeague.addFootballTeam(team5);
        footballLeague.addFootballTeam(team6);

        // matches and results
        footballLeague.win_Match(team2,team1);
        footballLeague.win_Match(team2,team4);
        footballLeague.win_Match(team4,team5);
        footballLeague.draw_Match(team1,team5);

        // sorting the teams in descending orders
```

```java
        // sorting the teams in descending orders
        footballLeague.descendingOrderSorting();

        // printing out the output of the league
        System.out.println(footballLeague);
    }
}
```

This class Dashboard with a main function adds a FootballLeague object to several Team instances that are created. The main focus is to record the results of some games played between league clubs. The program then prints out the league after sorting the teams in the league in decreasing order.

FootballLeague:

```java
import java.util.Collections;
import java.util.Comparator;
import java.util.LinkedList;
import java.util.List;


// creating class for football league
2 usages
public class FootballLeague {

    // declaring variables
    2 usages
    private String footballLeague;
    // creating a list of type Team to store a list of teams
    10 usages
    List<Team> footballListing = new LinkedList();

    // constructor to initialize a football league
    1 usage
    public FootballLeague(String footballLeague) { this.footballLeague = footballLeague; }
    // method to sort the list of teams in descending order based on points
    1 usage
    public void descendingOrderSorting(){
        // using sort method from collection class to sort the list of teams
        Collections.sort(footballListing, new Comparator<Team>() {
            @Override
            //Comparing the points of the two teams and returning a negative value
            public int compare(Team team1, Team team2) { return - (team1.getPoints()-team2.getPoints()); }
        });
    }
```

Dashboard.java | FootballLeague.java | Team.java

```java
    }

    // displaying the ranking of the team in the league
    public String toString() {
        String output = "";
        output += "FootballLeague: " + this.footballLeague + "\n";
        Integer teamPosition = 1;
        for (Team team : this.footballListing) {
            output += "\nRanking " + teamPosition.toString() + "\n";
            output += team;
            teamPosition += 1;
        }
        return output;
    }

    // method to add a team to the league
    6 usages
    public void addFootballTeam(Team teamToAdd) { this.footballListing.add(teamToAdd); }

    // method to remove a team from the league
    public void removeFootballTeam(Team teamToRemove) { this.footballListing.remove(teamToRemove); }

    // method to update the points of the winning and losing team
    3 usages
    public void win_Match(Team winningTeam, Team losingTeam) {
        for (int i = 0; i < this.footballListing.size(); i++) {
            if (footballListing.get(i).getTeamName().compareTo(winningTeam.getTeamName()) == 0) {
                winningTeam.incrementteamWin();
            }
            if (this.footballListing.get(i).getTeamName().compareTo(losingTeam.getTeamName()) == 0) {
```

```
            if (this.footballListing.get(i).getTeamName().compareTo(losingTeam.getTeamName()) == 0) {
                losingTeam.incrementteamLoss();
            }
        }
    }
    // method to update the points of the teams based on a draw result
    1 usage
    public void draw_Match(Team one, Team two) {
        for (int i = 0; i < this.footballListing.size(); i++) {
            if (footballListing.get(i).getTeamName().compareTo(one.getTeamName()) == 0) {
                one.incrementteamDraws();;
            }
            if (this.footballListing.get(i).getTeamName().compareTo(two.getTeamName()) == 0) {
                two.incrementteamDraws();
            }
        }
    }
}
```

The FootballLeague class, which symbolises a football league, has a list of Team objects that stand in for the league's participating teams. The FootballLeague class provides methods for adding and removing clubs from the league, recording the results of team-to-team matches, and ranking the teams in the league according to their point totals. The League's Team class, which represents a team, contains methods for updating the team's points based on the results of games.

Team:

Dashboard.java | FootballLeague.java | Team.java

```
25 usages
public class Team implements Comparable<Team>{

    3 usages
    private String team_Name;
    4 usages
    private int team_Wins;
    4 usages
    private int team_Draws;
    4 usages
    private int team_Losses;
    9 usages
    private Integer team_Points;

    // constructor is being set to initialize name, win, draw, loss and points
    6 usages
    public Team(String team_Name) {
        this.team_Name = team_Name;
        this.team_Wins = 0;
        this.team_Draws = 0;
        this.team_Losses = 0;
        this.team_Points = 0;
    }

    // getter for team name
    8 usages
    public String getTeamName() {
        return team_Name;
    }

```

The team class represents the team of the football club. The name of the team, the total number of wins, draws, losses, and points are all instance variables. It contains methods to compare the points of two teams as well as methods to update a team's points based on the result of a game (win, draw, or loss). The team object can also be returned as a string using the function toString() method.

Practical 13:

Calculator:

This programme uses a stack to construct a simple calculator. On a list of numbers, the calculator can add, subtract, multiply, and divide them. The calculator receives its input in the form of a string array, where each element is either a number or an operator. The numbers and operators are processed by the code, which stores them on a stack. Numbers are removed from the stack, the necessary operation is carried out, and the outcome is then pushed back onto the stack. When the string "end" is encountered at the top of the stack, the loop is broken. The code then displays the calculation's ultimate outcome.

rifat.java | Calculator.java

```java
import java.util.Stack;

public class Calculator {
    public static void main(String[] args) {

        String[] insert = {"4", "5", "+", "1", "2","*","-","5","/","end"};
        float totalValue = 0;

        // Initializing empty stack
        Stack<String> stack = new Stack();
        int i = 0;
        boolean b = stack.empty();
        while (true) {
            stack.push(insert[i]);
            System.out.println("\nTop:");
            System.out.println(insert[i]);
            i++;

            // perform addition if the stack is '+'
            if (stack.peek() == "+") {
                stack.pop();
                int number1 = Integer.parseInt((String) stack.pop());
                int number2 = Integer.parseInt((String) stack.pop());
                int value = number1 + number2;
                System.out.println("\nTop:");
                System.out.println(value);
                stack.push( item: "" + value);
            }

            // perform multiplication if the stack is '-'
```

rifat.java | Calculator.java

```
            // perform multiplication if the stack is '-'
            if (stack.peek() == "*") {
                stack.pop();
                int num1 = Integer.parseInt((String) stack.pop());
                int num2 = Integer.parseInt((String) stack.pop());
                int val = num1 * num2;
                System.out.println("\nTop:");
                System.out.println(val);
                stack.push( item: "" + val);
            }

            // perform subtraction if the stack is '-'
            if (stack.peek() == "-"){
                stack.pop();
                int numb1 = Integer.parseInt((String) stack.pop());
                int numb2 = Integer.parseInt((String) stack.pop());
                int v = numb2 - numb1;
                System.out.println("\nTop:");
                System.out.println(v);
                stack.push( item: "" + v);
            }

            // perform division if the stack is '/'
            if (stack.peek() == "/"){
                stack.pop();
                int numb1 = Integer.parseInt((String) stack.pop());
                int numb2 = Integer.parseInt((String) stack.pop());

                totalValue = (float)numb2/numb1;
                System.out.println("\nTop:");
                System.out.println("\nTop:");
                System.out.println(totalValue);
                stack.push( item: "" + totalValue);
            }

            // loop will break if the top of the stack is 'end'
            if(stack.peek() == "end"){
                break;
            }

        }
        System.out.println("\nFinal answer after calculating stacks: \n{(4+5)-(1*2)/5} is "+totalValue);
    }
}
```

Practical 14: Queuing

Queue:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
A	A																												
B	B	B																											
		C	C	C																									
			D	D	D	D	D	D																					
			E	E	E	E	E	E	E																					
			F	F	F	F	F	F	F	F	F	F																	
							G	G	G	G	G	G	G																	
											H	H	H	H	H														
												I	I	I	I	I													
															J	J	J	J	J										

Time unit	running	method	Return value	First <- Q <- last
0	A	Enqueue (A) Enqueue (B) Dequeue ()	-	(A)
1	A		-	(B)
2	B	Enqueue (C)	B	(C)
3	C	Enqueue (D) Enqueue (E) Enqueue (F)	C	(D) (E) (F)
4	C		-	
5	D	Dequeue ()	D	(E, F)
6	D		-	
7	D	Enqueue (G)		(E) (F) (G)
8	D		-	
9	E	Dequeue ()	E	(F) (G)
10	F	Dequeue ()	F	(G)
11	F	Enqueue (H)	-	(G) (H)
12	F	Enqueue (I) Enqueue (J)	-	(G) (H) (I) (J)
13	G	Dequeue ()	G	(H) (I) (J)
14	H	Dequeue ()	H	(I) (J)
15	H		-	
16	I	Dequeue (I)	I	(J)
17	J	Dequeue ()	J	-
18	J			

Priority queue:

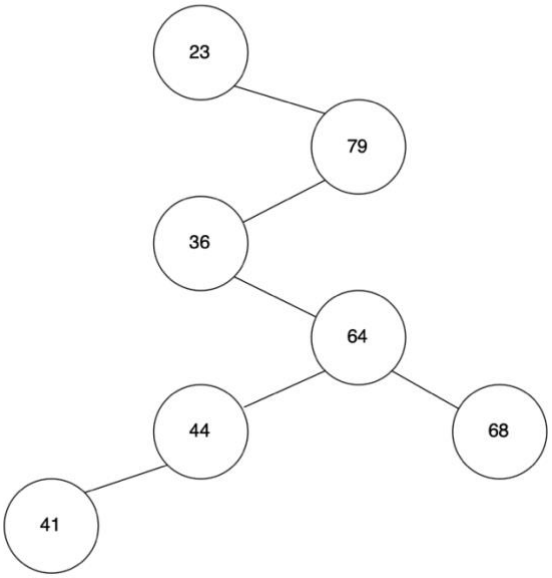
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
A	A	A																											
B																													
		C	C	C																									
			D	D	D	D	D	D	D	D	D	D	D																
			E	E	E																								
			F	F	F	F	F	F																					
							G	G	G																				
										H	H	H	H	H	H	H													
											I	I	I																
														J	J	J	J	J											

Waiting Queue:
A = 1-time units
B = 0-time units
C = 1-time units
D = 7-time units
E = 2-time units
F = 3-time units
G = 2-time units
H = 4-time units
I = 2-time units
J = 2-time units

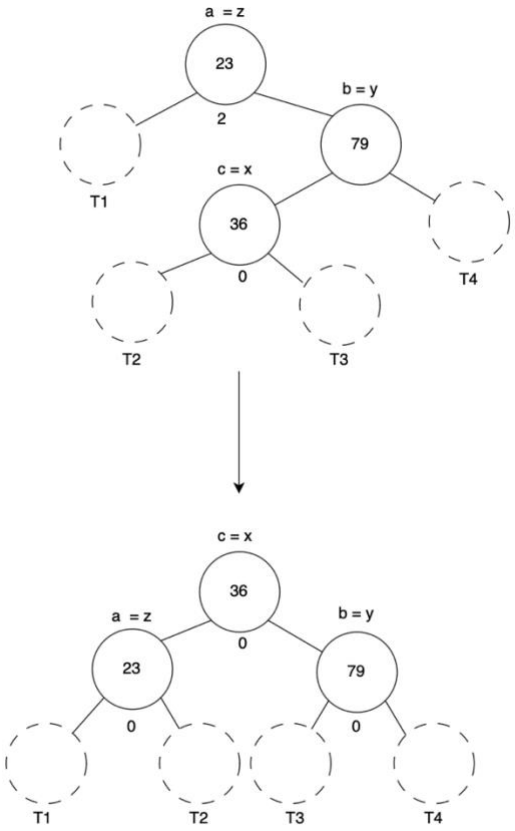
Total waiting time = 0 + 2 + 1 + 2 + 6 + 7 + 6 + 3 + 4 + 2 = 24-time units
Mean = 2.4-time units

Time unit	Running	Method	Return Value	First <- Q <- Last
0	B	Insert (2, A) Insert (1, B) Remove Min ()	- (1, B)	{(2, A)} {(1, B)}
1	A	Remove Min ()	{(2, A)}	-
2	A	Insert (2, C)		(2, C)
3	E	Insert (4, D) Insert (1, F) Insert (3, F) Remove Min ()	- - - {(1, E)}	{(2, C), (4, D), (1, E), (3, F)} - {(2, C), (4, D), (3, F)} -
4	C	Remove Min ()	{(2, C)}	{(4, D), (3, F)}
5	C	-	-	{(4, D), (3, F)}
6	F	Remove Min ()	{(3, F)}	{(4, D)}
7	F	Insert (G, 1)	-	{(4, D), (1, G)}
8	F	-	-	{(4, D), (1, G)}
9	G	Remove Min ()	{(1, G)}	{(4, D)}
10	D	Remove Min ()	{(4, D)}	-
11	D	Insert (2, H)	-	{(2, H)}
12	D	Insert (1, I)	-	{(2, H), (1, I)}
13	D	-	-	{(2, H), (1, I)}
14	I	Remove Min ()	{(1. I)}	{(2, H)}
15	H	Insert (3, J) Remove Min ()	- {(2, H)}	{(2, H), (3, J)} {(3, J)}
16	H	-	-	{(3, J)}
17	J	Remove Min ()	{(3, J)}	-
18	J	-	-	-
19	J	-	-	-
20				
21				
22				
23				
24				
25				
26				
27				
28				

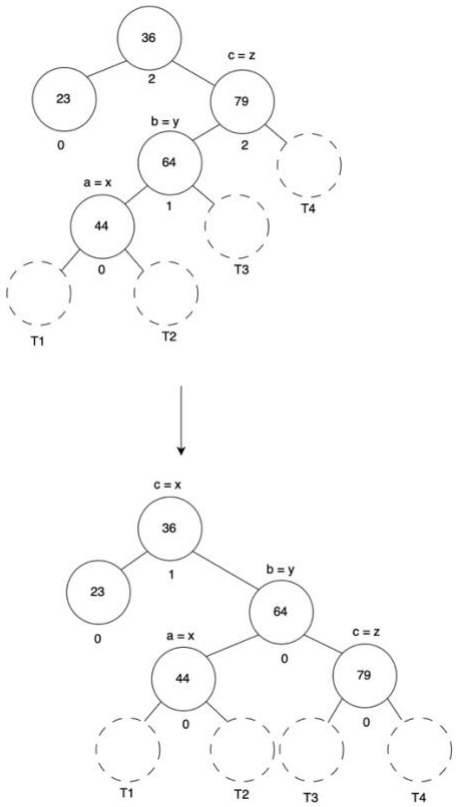
Practical 15: AVL Trees



Step 1:

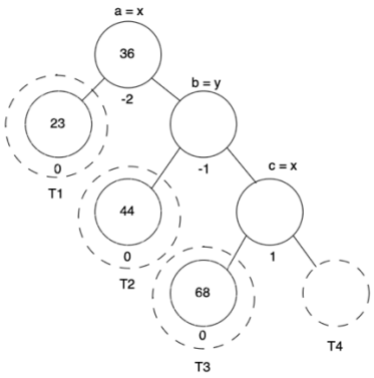


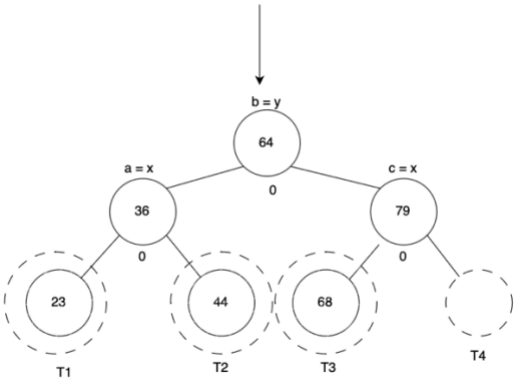
Step 2:


36
2
c = z
23
0
79
2
b = y
64
1
T4
a = x
44
0
T3
T1
T2

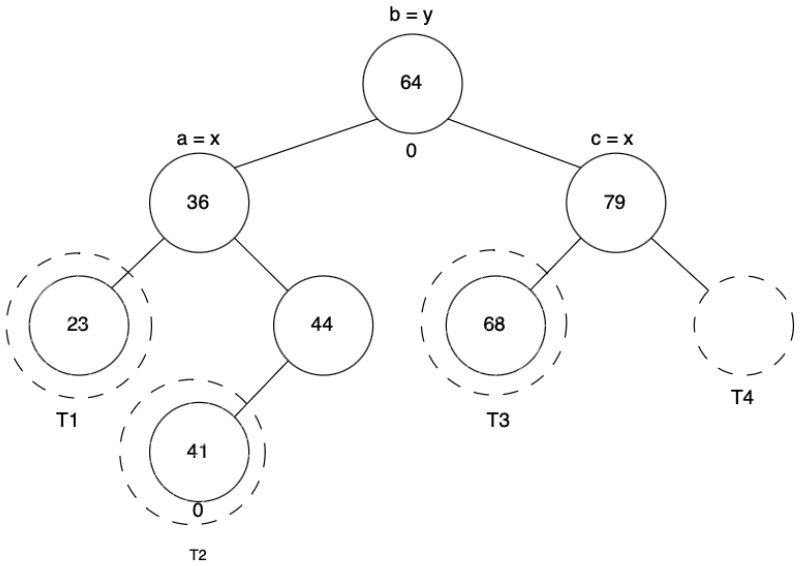
c = x
36
1
23
0
b = y
64
0
a = x
44
0
c = z
79
0
T1
T2
T3
T4

Step 3:


a = x
36
-2
b = y
23
0
T1
-1
c = x
44
0
T2
1
68
0
T3
T4


b = y
64
0
a = x
36
0
c = x
79
0
23
T1
44
T2
68
T3
T4

Step 4:



Practical 16:

Min Heap:

Array 35 inserted:

35 (0)	(1)	(2)	(3)	(4)	(5)	(6)

Array 12 inserted:

12 (0)	35 (1)	(2)	(3)	(4)	(5)	(6)

Array 73 inserted:

12 (0)	35 (1)	73 (2)				

Array 61 inserted:

12 (0)	35 (1)	73 (2)	61 (3)			

Array 23 inserted:

12 (0)	23 (1)	73 (2)	61 (3)	35 (4)		

Array 57 inserted:

12 (0)	23 (1)	57 (2)	61 (3)	35 (4)	73 (5)	

Array 29 inserted:

12 (0)	23 (1)	29 (2)	61 (3)	35 (4)	73 (5)	57 (6)

Max Heap:

Array 35 inserted:

35 (0)						

Array 12 inserted:

35 (0)	12 (1)					

Array 73 inserted:

73 (0)	12 (1)	35 (2)				

Array 61 inserted:

73 (0)	61 (1)	35 (2)	12 (3)			

Array 23 inserted:

73 (0)	61 (1)	35 (2)	12 (3)	23 (4)		

Array 57 inserted:

73 (0)	61 (1)	57 (2)	12 (3)	23 (4)	35 (5)	

Array 29 inserted:

73 (0)	61 (1)	57 (2)	12 (3)	23 (4)	35 (5)	29 (6)

Initial array after 35, 12, 73, 61, 23, 57, and 29 inserted:

73 (0)	61 (1)	57 (2)	29 (3)	23 (4)	35 (5)	12 (6)

Array 73 sorted:

61 (0)	29 (1)	57 (2)	23 (3)	35 (4)	12 (5)	73 (6)

Array 61 sorted:

57 (0)	29 (1)	35 (2)	23 (3)	12 (4)	73 (5)	61 (6)

Array 57 sorted:

35 (0)	29 (1)	12 (2)	23 (3)	73 (4)	61 (5)	57 (6)

Array 35 sorted:

29 (0)	23 (1)	12 (2)	73 (3)	61 (4)	57 (5)	35 (6)

Array 29 sorted:

23 (0)	12 (1)	73 (2)	61 (3)	57 (4)	35 (5)	29 (6)

Array 23 sorted:

12 (0)	73 (1)	61 (2)	57 (3)	35 (4)	29 (5)	23 (6)

Array 12 sorted:

73 (0)	61 (1)	57 (2)	35 (3)	29 (4)	23 (5)	12 (6)

Practical 17:

This project contains 3 classes:

1. PhoneBook
2. PhoneBookEntry
3. User

PhoneBook:

PhoneBook.java | PhoneBookEntry.java | User.java

```java
import java.util.*;

2 usages
public class PhoneBook {

    11 usages
    Map < String, PhoneBookEntry > phoneBookMap;
    1 usage
    public PhoneBook() {
        this.phoneBookMap = new HashMap();
    }

    // input function to store information in phonebook
    3 usages
    public void inputPhoneBookEntry(String f_name, String l_name, String phone_no, String home_address, String email_address){
        PhoneBookEntry phone_book_entries = new PhoneBookEntry(f_name,l_name, phone_no, home_address, email_address);
        // checking if the user is alraedy in the phonebook
        if (!this.phoneBookMap.containsKey(f_name))
            this.phoneBookMap.put(f_name, phone_book_entries);
        else
        {
            System.out.println("Matching entry detected");
            System.out.println("Entry already in the database");
        }
    }

    //update of entry
    1 usage
    public void updatephonebookentry(String f_name,String l_name ,String phone_no, String home_address,String email_address){
        PhoneBookEntry phone_book_entries = new PhoneBookEntry(f_name,l_name, phone_no, home_address,email_address);
```

PhoneBook.java | PhoneBookEntry.java | User.java

```java
    public void updatephonebookentry(String f_name,String l_name ,String phone_no, String home_address,String email_address
        PhoneBookEntry phone_book_entries = new PhoneBookEntry(f_name,l_name, phone_no, home_address,email_address);

        //to check that the user is already in the phonebook or not
        if (this.phoneBookMap.containsKey(f_name)) {
            this.phoneBookMap.put(f_name, phone_book_entries);
        }
        else
        {
            System.out.println("Unable to find the name ");
            System.out.println("This name is not in the  phonebook");
        }
    }

    //function for deleing entries
    1 usage
    public void deletephonebookentry(String name){

        if (this.phoneBookMap.containsKey(name)) {
            this.phoneBookMap.remove(name);
        }
        else
        {
            System.out.println("Unable to find the name ");
            System.out.println("This name is not in the  phonebook");
        }
    }

    //function for printing phonebook
    2 usages
```

PhoneBook.java | PhoneBookEntry.java | User.java

```
    //function for printing phonebook
    2 usages
    public void printphonebookentry(String name){

        //to check that the user is already in the phonebook or not
        if (this.phoneBookMap.containsKey(name)) {
            System.out.println(this.phoneBookMap.get(name));
        }
        else
        {
            System.out.println("Unable to find the name ");
            System.out.println("This name is not in the  phonebook");
        }
    }
    2 usages
    public void printAllPhoneBookEntries() {
        for (PhoneBookEntry entry : this.phoneBookMap.values()) {
            System.out.println(entry);
        }
    }

    @Override
    public String toString() {
        return "PhoneBook{" +
                "phoneBookMap=" + phoneBookMap +
                '}';
    }
}

```

A class called PhoneBook, that represents a phone book and contains a mapping of phone book entries, is created in this code. An instance of the PhoneBookEntry class, which stores details about a person including their first and last names, phone number, home address, and email address, is used to represent each phone book entry. The PhoneBook class includes methods for adding, editing, deleting, and printing phone book entries. It also includes a method for printing every phone book entry. The first names of the people are used as keys in a HashMap and the PhoneBookEntry objects are used as values in the PhoneBook class's storage of phone book entries.

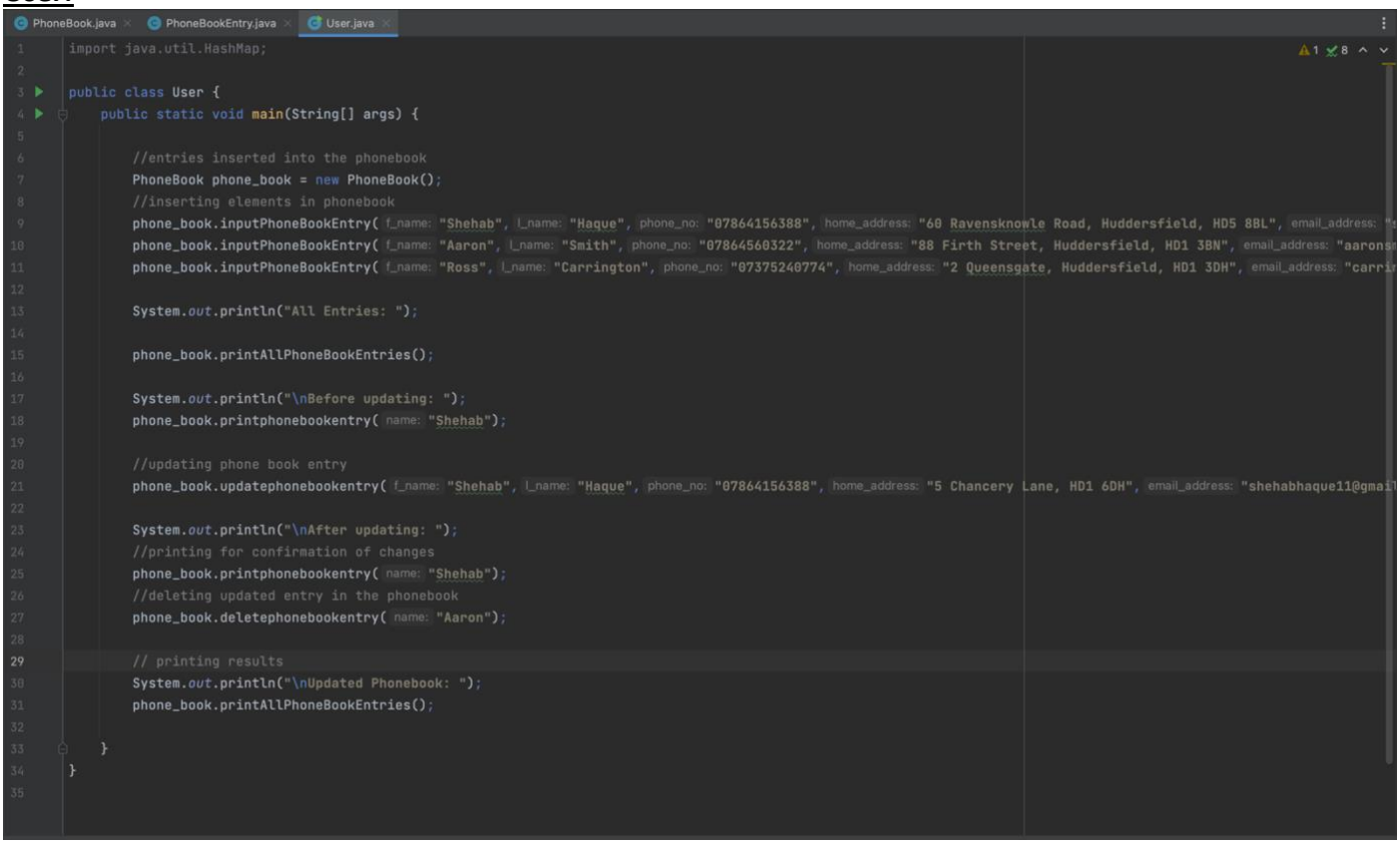
PhoneBookEntry:

PhoneBook.java | PhoneBookEntry.java | User.java

```java
import java.util.*;
6 usages
public class PhoneBookEntry {

    // declaring variales of phone book
    3 usages
    private String firstName;
    3 usages
    private String lastName;
    4 usages
    private String phoneNumber;
    3 usages
    private String homeAddress;
    5 usages
    private String emailAddress;

    // constructor to add information to the phonebook
    2 usages
    public PhoneBookEntry(String f_name,String l_name ,String phone_no, String home_address,String email_address) {

        this.firstName = f_name;
        this.lastName = l_name;
        this.phoneNumber = phone_no;
        this.homeAddress = home_address;
        this.emailAddress = email_address;

    }

    // method to get the information to the phonebook
    public String getName() { return firstName; }
    public String getName() { return firstName; }
    public String getEmail() { return emailAddress; }
    public String getAddress() {
        return homeAddress;
    }
    public void setPhone_number(String phone_number) { this.phoneNumber = phoneNumber; }
    public void setEmail(String email) { this.emailAddress = emailAddress; }
    public String getSurname() { return lastName; }

    // method to return a string representation of the phone book entry
    @Override
    public String toString() {
        return "\nPhoneBookEntry:" +
                "\nFirst Name: " + firstName  +
                "\nSurname: " + lastName +
                "\nPhone no: " + phoneNumber +
                "\nAddress: " + homeAddress +
                "\nEmail: " + emailAddress;
    }
}
```

This class named PhoneBookEntry that represents a phone book entry. There are various fields in the PhoneBookEntry class, including firstName, lastname, phoneNumber, home address, and emailAddress. A function Object() for the class lets users build a new PhoneBookEntry object and initialise its properties with supplied values. The class also includes a number of methods that let users create a string representation of the PhoneBookEntry object as well as obtain or alter the contents of its fields.

User:



```java
import java.util.HashMap;

public class User {
    public static void main(String[] args) {

        //entries inserted into the phonebook
        PhoneBook phone_book = new PhoneBook();
        //inserting elements in phonebook
        phone_book.inputPhoneBookEntry( f_name: "Shehab", l_name: "Haque", phone_no: "07864156388", home_address: "60 Ravensknowle Road, Huddersfield, HD5 8BL", email_address: "
        phone_book.inputPhoneBookEntry( f_name: "Aaron", l_name: "Smith", phone_no: "07864560322", home_address: "88 Firth Street, Huddersfield, HD1 3BN", email_address: "aaronsm
        phone_book.inputPhoneBookEntry( f_name: "Ross", l_name: "Carrington", phone_no: "07375240774", home_address: "2 Queensgate, Huddersfield, HD1 3DH", email_address: "carri

        System.out.println("All Entries: ");

        phone_book.printAllPhoneBookEntries();

        System.out.println("\nBefore updating: ");
        phone_book.printphonebookentry( name: "Shehab");

        //updating phone book entry
        phone_book.updatephonebookentry( f_name: "Shehab", l_name: "Haque", phone_no: "07864156388", home_address: "5 Chancery Lane, HD1 6DH", email_address: "shehabhaque11@gmail

        System.out.println("\nAfter updating: ");
        //printing for confirmation of changes
        phone_book.printphonebookentry( name: "Shehab");
        //deleting updated entry in the phonebook
        phone_book.deletephonebookentry( name: "Aaron");

        // printing results
        System.out.println("\nUpdated Phonebook: ");
        phone_book.printAllPhoneBookEntries();

    }
}
```

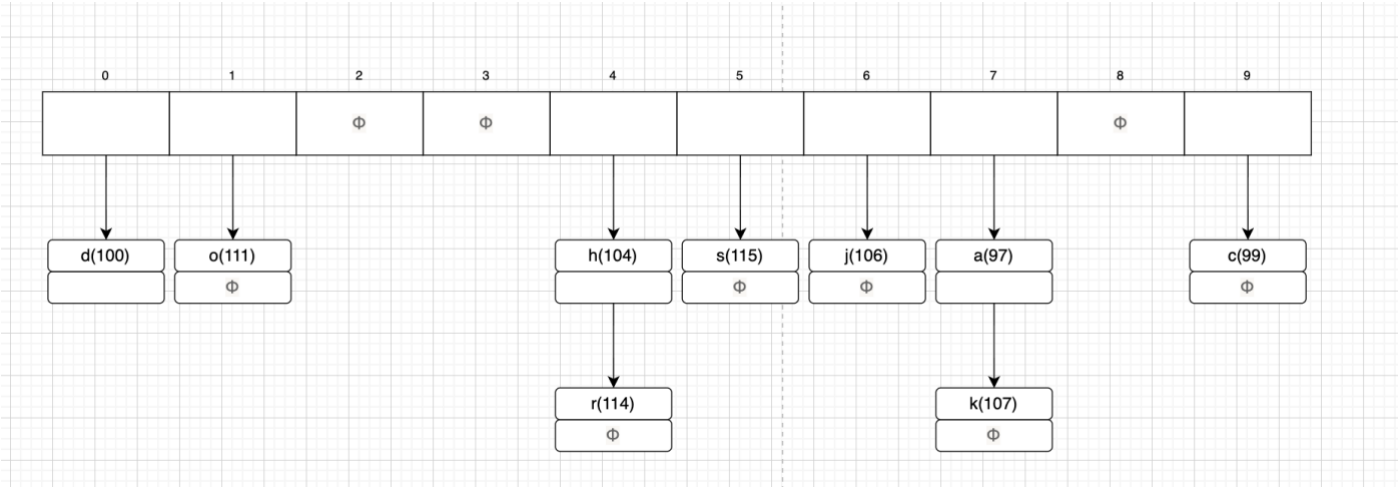
The main function of the User class creates a PhoneBook object, fill it with several PhoneBookEntry objects, update one entry, remove another item, and print the completed phone book.

Practical 18: hash tables

D (100)	X (120)	O (111)	C (99)	H (104)	R (114)	J (106)	A (97)	K(107)	S (115)
0	1	2	3	4	5	6	7	8	9

Linear probing calculations:

- The hash of the character ‘d’ is equal to its ASCII value of 100. When this value is trimmed to the last two digits (100 % 0), the result is 0. Therefore, the character ‘d’ is placed in the array at the index of 0.
- The hash of the character ‘h’ is equal to its ASCII value of 104. When this value is trimmed to the last two digits (104 % 0), the result is 4. Therefore, the character ‘h’ is placed in the array at the index of 4.
- The character ‘x’ hash is equal to its ASCII value of 120. This causes a collision with the previous character that was placed at index 0 in the array. To resolve this collision, a technique called linear probing is used, which involves searching for the next available index in the array. Therefore, the character ‘x’ is placed at index 1 in the array.
- The ASCII value of the hash character ‘r’ is 114. After trimming the value, it becomes 4 when it is modulo 10. As this results in a collision, linear probing finds the next available index in the array. Therefore, ‘r’ is placed in index 5 as a result.
- The hash of the character ‘a’ is equal to its ASCII value of 97. When this value is trimmed to the last two digits (97 % 10), the result is 7. Therefore, the character ‘h’ is placed in index 7 of the array.
- The character ‘j’ hash is equal to its ASCII value of 106. When this value is trimmed to the last two digits (106 % 10), the result is 6. Therefore, the character ‘h’ is placed in index 6 of the array.
- The ASCII value of the hash character ‘o’ is 111. After trimming the value, it becomes 1 when it is modulo 10. As this results in a collision, linear probing finds the next available index in the array. Therefore, ‘o’ is placed in index 2.
- The ASCII value of the hash character ‘o’ is 111. After trimming the value, it becomes 1 when it is modulo 10. As this results in a collision, linear probing finds the next available index in the array. Therefore, ‘o’ is placed in index 2.
- The ASCII value of the hash character ‘k’ is 107. After trimming the value, it becomes 7 when it is modulo 10. As this results in a collision, linear probing finds the next available index in the array. Therefore, ‘k’ is placed in index 8.
- The ASCII value of the hash character ‘s’ is 115. After trimming the value, it becomes 5 when it is modulo 10. As this results in a collision, linear probing finds the next available index in the array. Therefore, ‘s’ is placed in index 9 after searching through 5, 6, 7, and 8.
- The ASCII value of the hash character ‘c’ is 99. After trimming the value, it becomes 9 when it is modulo 10. As this results in a collision, linear probing finds the next available index in the array. Therefore, ‘c’ is placed in index 3 after searching through 9, 0, 1, and 3.

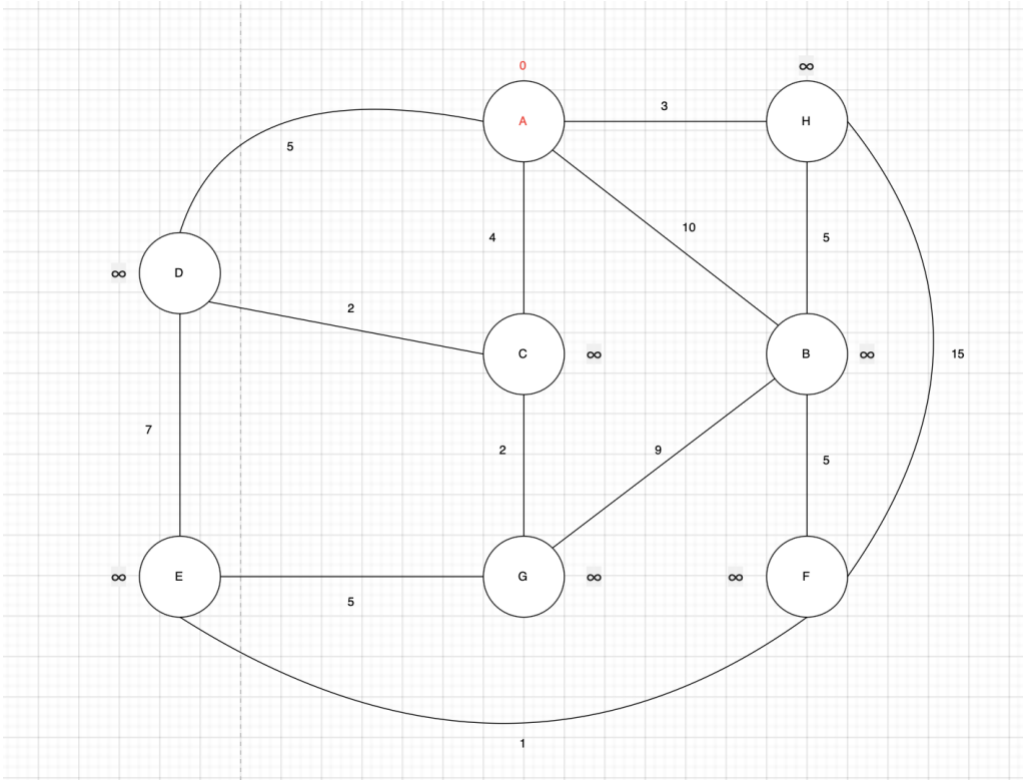


Chaining calculations are given as follows:

- ‘d’: hash (‘d’) = ASCII(‘d’) = 100, trimming gives 100 % 10 = 0, therefore, a new linked list is created at index 0.
- ‘h’: hash(‘h’) = ASCII(‘h’) = 104, trimming gives 104 % 10 = 4, therefore, a new linked list is created at index 4
- ‘x’: hash(‘x’) = ASCII(‘x’) = 120, trimming gives 120% 10 = 0 (collision), therefore, add ‘x’ to the linked list at index 0
- ‘r’: hash(‘r) = ASCII(‘r’) = 114, trimming gives 114 % 10 = 4, (collision), therefore, add ‘r’ to the linked list at index 4
- ‘a’: hash(‘a’) = ASCII(‘a’) = 97, trimming gives 97 % 10 = 7, therefore a new linked list is created at index 7
- ‘j’: hash(‘j’) = ASCII(‘j’) = 106, trimming gives 106 % 10 = 6, therefore, a new linked list is created at index 6
- o’: hash(‘o’) = ASCII(‘o’) = 111, trimming gives 111 % 10 = 1, therefore, a new linked list is created at index 1
- ‘k’: hash(‘k’) = ASCII(‘k’) = 107, trimming gives 107 % 10 = 7 (collision), therefore, add ‘k’ to the linked list at index 7
- ‘s’: hash(‘s’) = ASCII(‘s’) = 115, trimming gives 115 % 10 = 5, therefore, a new linked list is created at index 5
- ‘c’: hash(‘c’) = ASCII(‘c’) = 99, trimming gives 99% 10 = 9, therefore, a new linked list is created at index 9

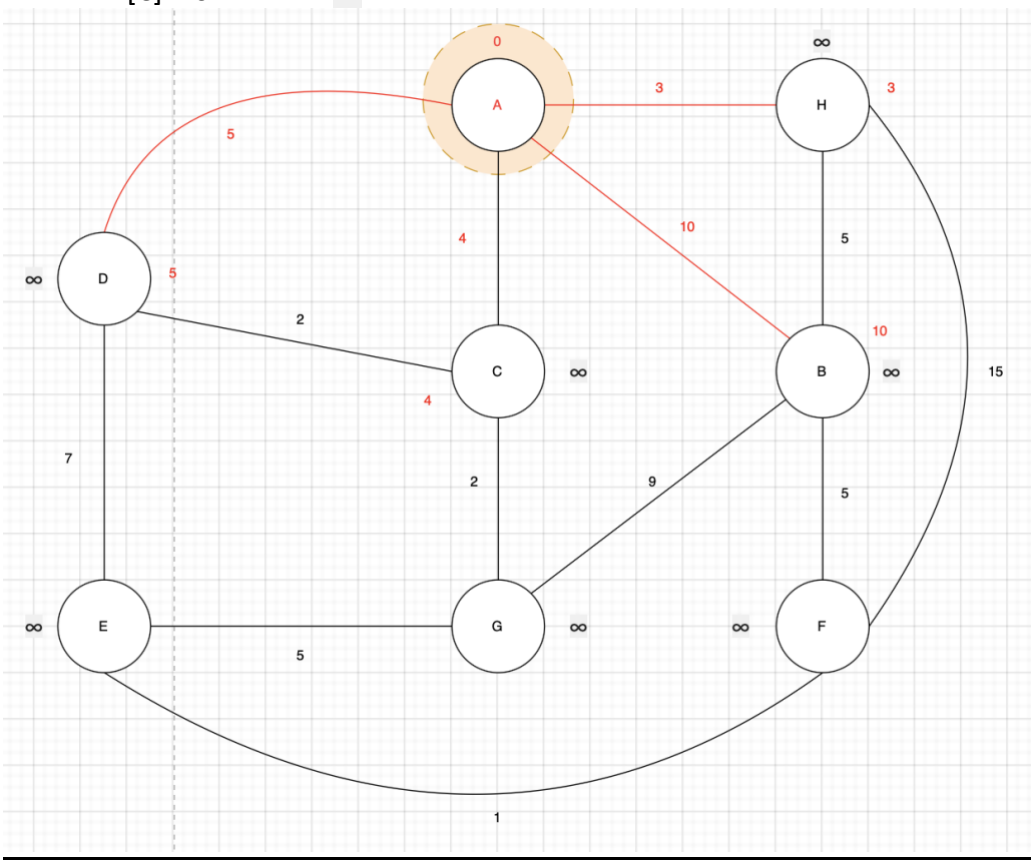
Practical 19:

Starting node: ‘A’



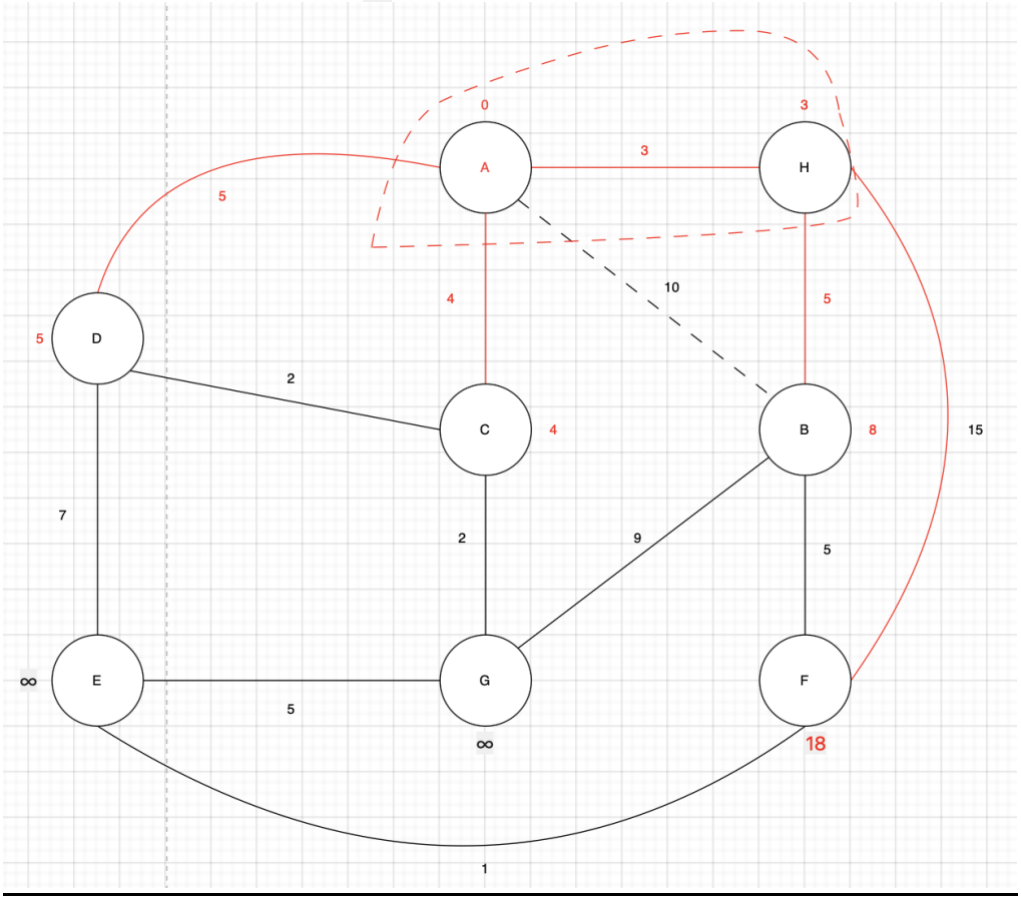
Cloud added to node ‘A’

- D [D] = 0 + 5 = 5 < ∞
- D [H} = 0 + 3 = 3 < ∞
- D [C] = 0 + 4 = 4 < ∞



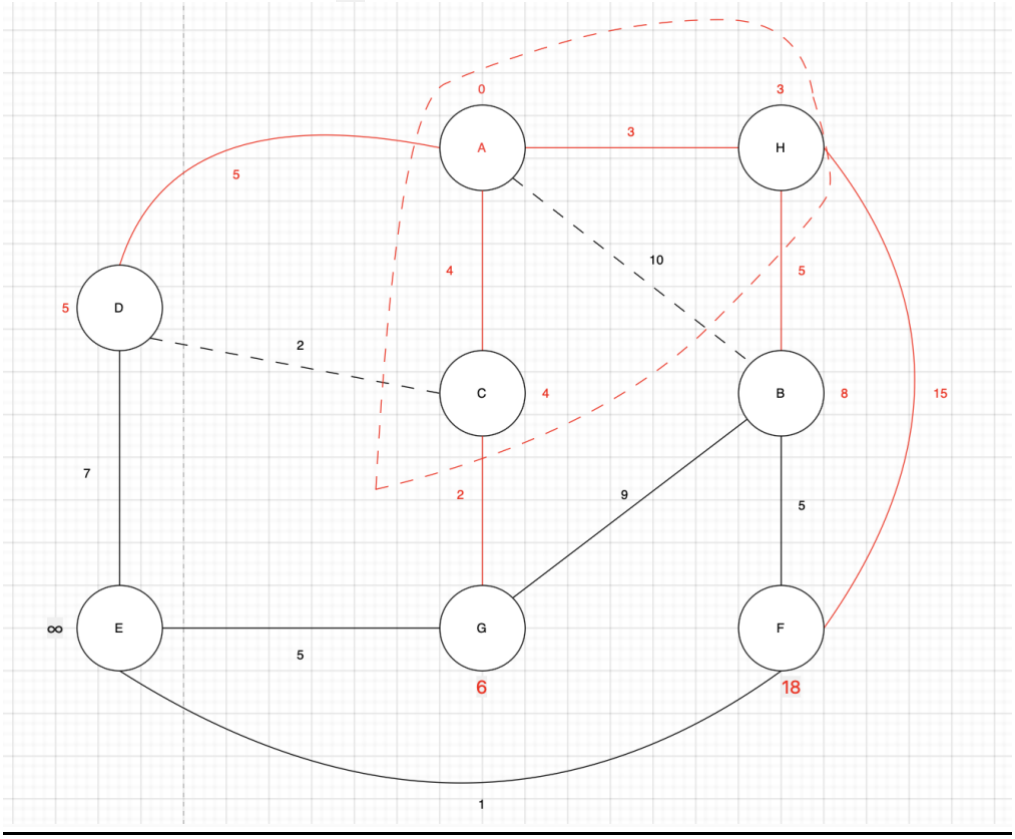
Cloud added to node ‘H’

- D [B] = 3 + 5 = 8 < 10
- D [F] = 3 + 15 = 18 < ∞



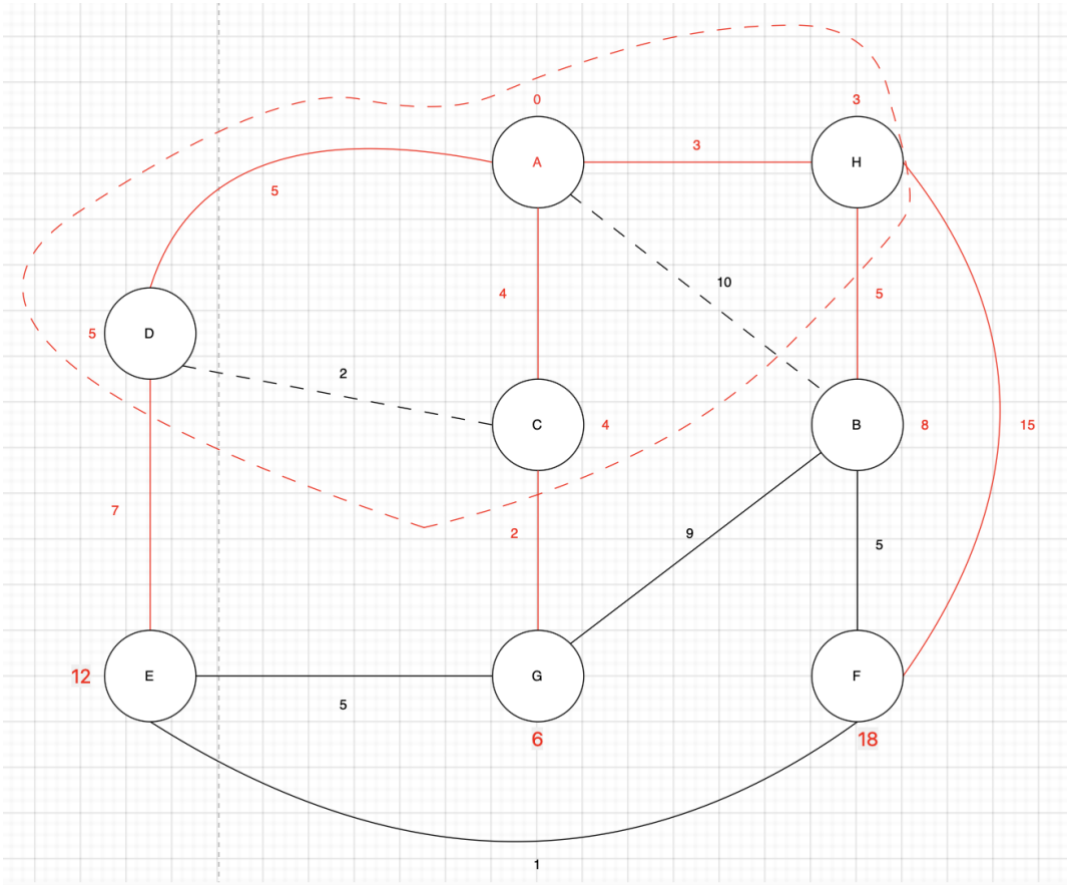
Cloud added to node ‘C’

- D [D] = 4 + 2 = 6 > 5
- D [G] = 4 + 2 = 6 < ∞



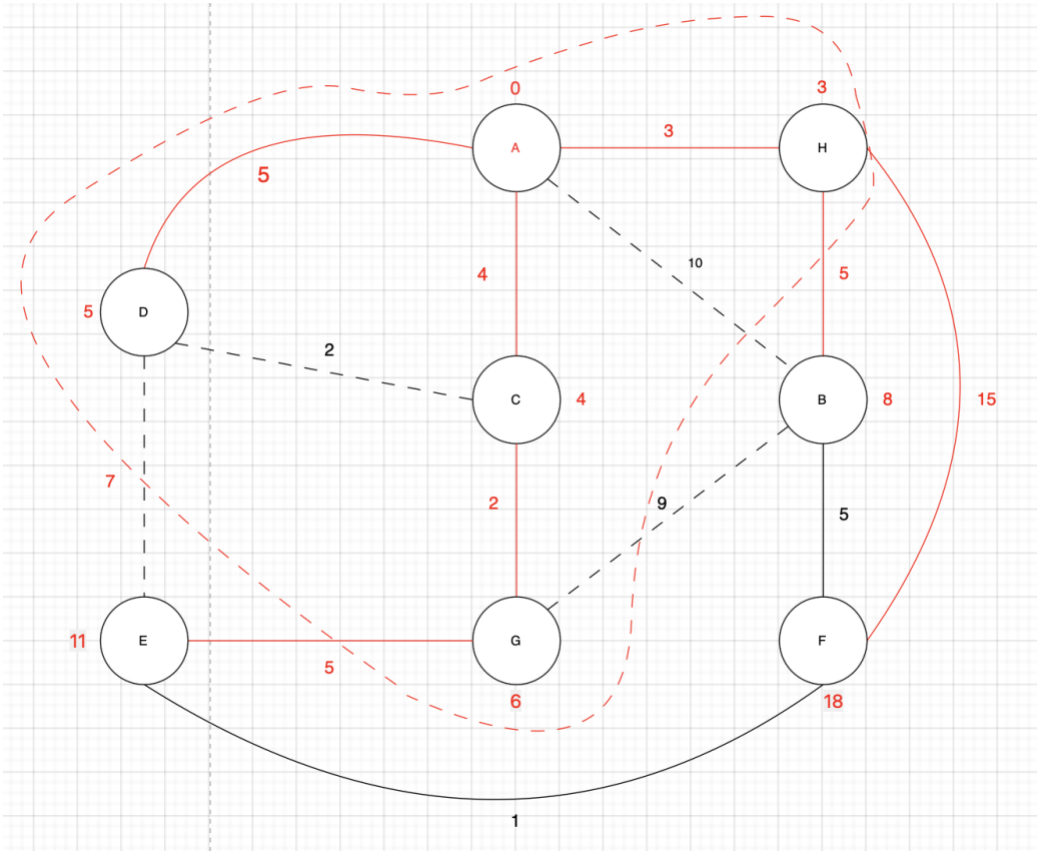
Cloud added to node ‘D’

- D [E] = 5 + 7 = 12



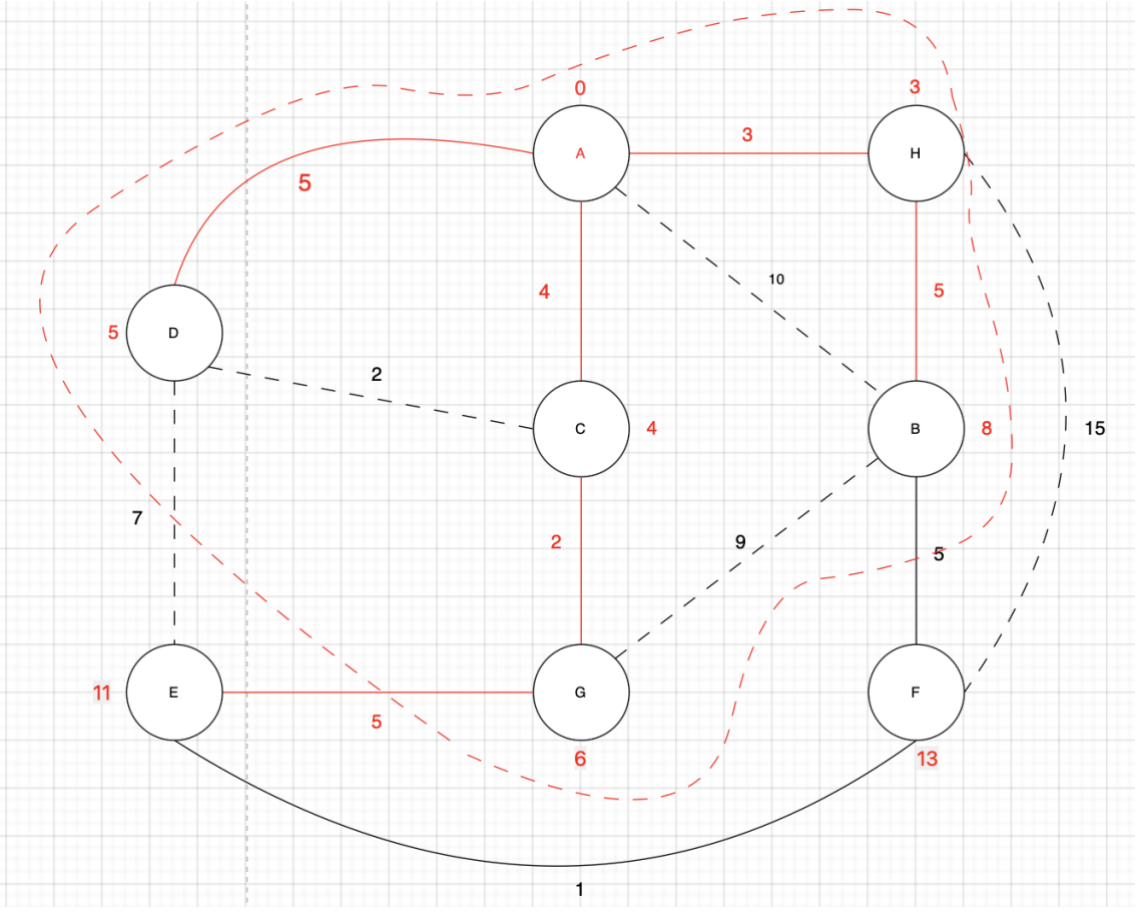
Cloud added to node ‘G’

- D [B] = 6 + 9 = 15 > 8
- D [E] = 6 + 5 = 11 < 12



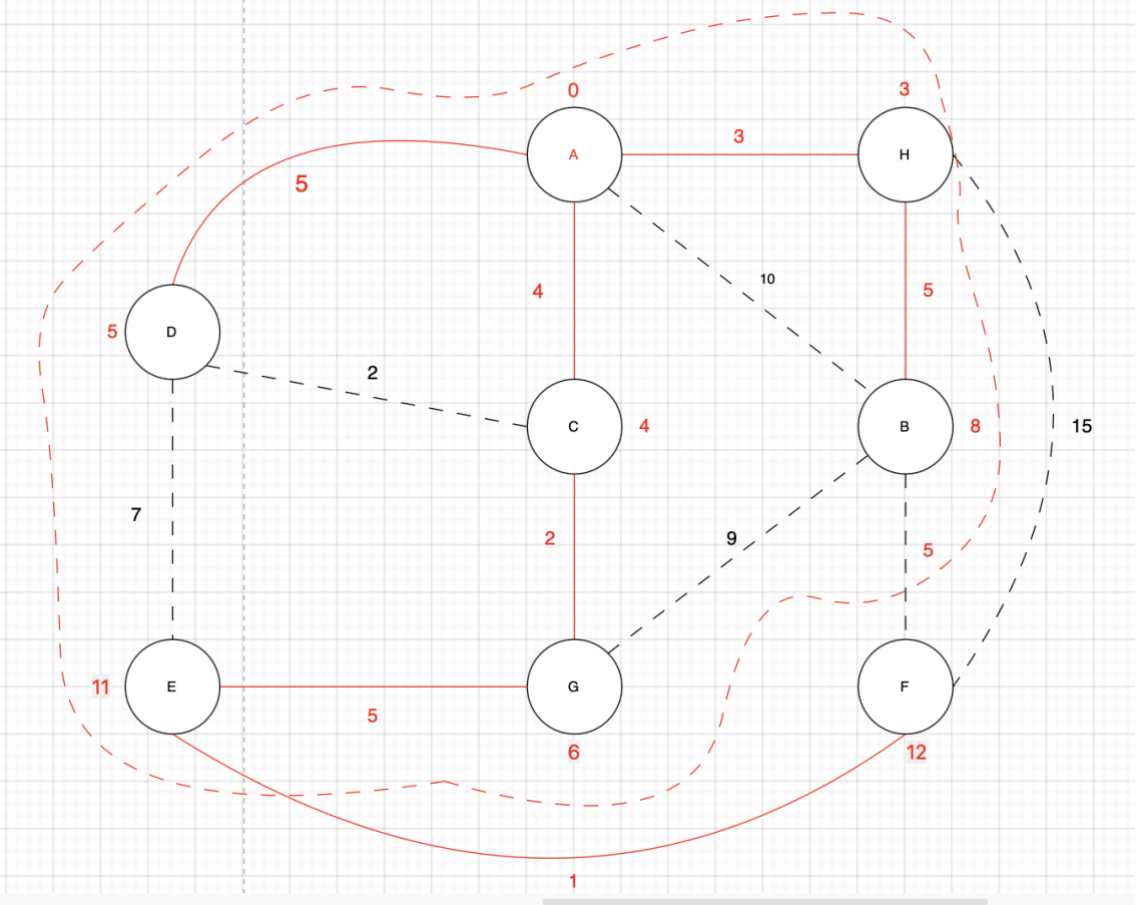
Cloud added to node ‘B’

- D [F] = 8 + 5 = 13 < 18

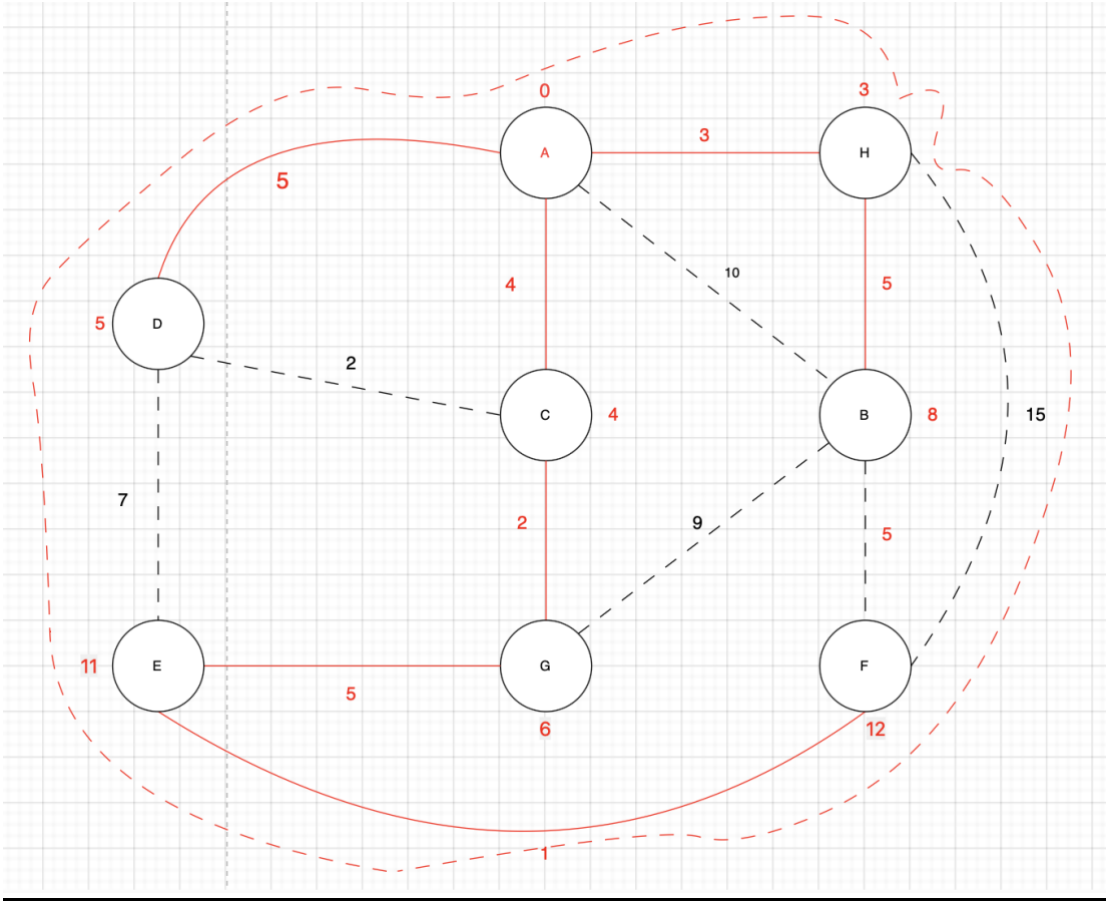


Cloud added to node ‘E’

- D [F] = 11 + 1 = 12 < 13



Cloud added to node ‘F’
No nodes left to be added to cloud



Practical 20:

The Sierpinski carpet pattern is be created and printed using the class SierpinskiCarpet defined in this code. Each square is recursively divided into nine smaller squares in the Sierpinski carpet pattern, and the middle square is then removed, leaving a cross-shaped space. The function Object() specifies the size of the carpet, then the createboard method splits the squares and removes the middle ones until each one is 1 size. This generated the pattern and is then printed to the terminal using the printboard function.

SierpinskiCarpet.java

```
2 usages
public class SierpinskiCarpet {
    5 usages
    static String[][] carpet;
    1 usage
    static int size;

    1 usage
    public SierpinskiCarpet(int size) {
        // creates 2D array
        this.size = size;
        carpet = new String[size][size];
        int row = 0;
        int col = 0;
        while (row < size) {
            col = 0;
            while (col < size) {
                this.carpet[row][col] = " * ";
                col++;
            }
            row++;
        }
    }

    1 usage
    public static void emptyblock(int currentsize, int a, int b) {
        // function to empty middle block
        int row = b + currentsize / 3;
        int col = 0;
        while (row <= b + 2 * (currentsize / 3) - 1) {
            col = a + currentsize / 3;
            while (col <= a + 2 * (currentsize / 3) - 1) {
                carpet[row][col] = "   ";
                col++;
            }
            row++;
```

SierpinskiCarpet.java

```
            row++;
        }
    }

    2 usages
    public static void createboard(String[][] carpet, int a, int b, int presentsize) {
        //empty center block
        emptyblock(presentsize, a, b);

        if (presentsize <= 1) {
            return;
        }

        int row = 0;
        int col = 0;
        // loops through all blcoks
        while (row < presentsize) {
            col = 0;
            while (col < presentsize) {
                // recursive function to move to the 3n-1x3n-1
                createboard(carpet,  a: row + b,  b: col + a,  presentsize: presentsize / 3);
                col += presentsize / 3;
            }
            row += presentsize / 3;
        }
    }

    1 usage
    public static void printboard(int size) {
        // loops through the arrays to prints
        int row = 0;
        int col = 0;
        String output = "";
        while (row < size) {
            col = 0;
            output = "";
            output = "";
            while (col < size) {
                output += carpet[row][col];
                col++;
            }
            System.out.println(output);
            row++;
        }
    }

    public static void main(String[] args) {
        System.out.println("Sierpinski Carpet: ");
        SierpinskiCarpet board = new SierpinskiCarpet( size: 27);
        createboard(carpet,  a: 0,  b: 0,  presentsize: 27);
        printboard( size: 27);
    }
}
```