

**Department of Computer Science**

**2019-20**

**M.Sc-IT(Part-I)**

Course : Data Structure and Algorithms

Course Code: MIT 11

Name: Vedank Naik

RollNo:SP192221

|  |  |  |  |
| --- | --- | --- | --- |
| Sr. No | Date | Title/Topic | Page  No. |
| 1.1 | 4-7-19 | Demonetization problem | 3 |
| 1.2 | 4-7-19 | Subsequence of number | 5 |
| 1.3 | 9-7-19 | Occurrence of string | 7 |
| 2.1 | 11-7-19 | Instagram problem | 10 |
| 2.2 | 11-7-19 | Pattern sum | 12 |
| 2.3 | 11-7-19 | Dhananjay’s Magical Word | 14 |
| 3.1 | 17-7-19 | Tower of Hanoi | 17 |
| 3.2 | 17-7-19 | Staircase problem | 19 |
| 4.1 | 19-7-19 | Stacks | 21 |
| 4.2 | 19-7-19 | Bracket validation | 23 |
| 5.1 | 24-7-19 | Loop in list | 25 |
| 5.2 | 24-7-19 | Rotate list | 28 |
| 6 | 30-7-19 | Binary Search Tree | 31 |
| 7 | 8-8-19 | AVL Tree | 34 |
| 8 | 20-8-19 | Huffman coding | 41 |
| 9.1 | 19-9-19 | Open addressing | 44 |
| 9.2 | 24-9-19 | Flight problem | 47 |

**Sr.No: 1-1**

**Date: 4-7-19**

**Problem Statement:**

Due to the demonetization move, there is a long queue of people in front of ATMs. Due to withdrawal limit per person per day, people come in groups to withdraw money. Groups come one by one and line up behind the already present queue. The groups have a strange way of arranging themselves. In a particular group, the group members arrange themselves in increasing order of their height (not necessarily strictly increasing).

Swapy observes a long queue standing in front of curious kid, he wants to count the total number of groups present in the queue waiting to withdraw money. Since groups are standing behind each other, one cannot differentiate between different groups and the exact count cannot be given. Can you tell him the minimum number of groups that can be observed in the queue?

**Source Code:**

package groups;

public class Groups {

public static void main(String[] args) {

// TODO code application logic here

int i,count,group=1;

Scanner h=new Scanner(System.in);

System.out.println("Number of people:");

count=h.nextInt();

int height[] = new int[count];

System.out.println("Enter height");

for(i=0;i<count;i++)

{

height[i] = h.nextInt();

// System.out.println("height"+height[i]);

}

int j;

for(i=0;i<count;i++)

{

j=i+1;

if(j==count)

break;

if(height[i]>height[j])

{

group=group+1;

}

System.out.println("Groups:"+group);

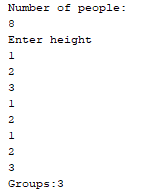
}

// System.out.println("Groups:"+group);

}

}

**Screenshot of Output:**

****

**Sr.No: 1-2**

**Date: 4-7-19**

**Problem Statement:**

You are given an array A of length N. You need to find maximum length of subsequence of A

which contains anyone digit common in all the elements of that subsequence. You are not

allowed to count leading zeroes.

**Source Code:**

/\*

\* To change this license header, choose License Headers in Project Properties.

\* To change this template file, choose Tools | Templates

\* and open the template in the editor.

\*/

package subseq;

import java.util.\*;

/\*\*

\*

\* @author Student

\*/

public class Subseq {

/\*\*

\* @param args the command line arguments

\*/

public static void main(String[] args) {

// TODO code application logic here

int i,n,temp,temp2;

Scanner h=new Scanner(System.in);

System.out.println("Number of elements:");

n=h.nextInt();

int num[] = new int[n];

Integer num2[] = new Integer[100];

System.out.println("Enter elements");

for(i=0;i<n;i++)

{

num[i] = h.nextInt();

}

int j=0;

for(i=0;i<n;i++)

{

temp=num[i];

while(temp>0)

{

temp2=temp%10;

num2[j]=temp2;

j++;

temp=temp/10;

}

}

int freq1 = Collections.frequency(Arrays.asList(num2),1);

int freq2 = Collections.frequency(Arrays.asList(num2),2);

int freq3 = Collections.frequency(Arrays.asList(num2),3);

int freq4 = Collections.frequency(Arrays.asList(num2),4);

int freq5 = Collections.frequency(Arrays.asList(num2),5);

int freq6 = Collections.frequency(Arrays.asList(num2),6);

int freq7 = Collections.frequency(Arrays.asList(num2),7);

int freq8 = Collections.frequency(Arrays.asList(num2),8);

int freq9 = Collections.frequency(Arrays.asList(num2),9);

int test[]={freq1,freq2,freq3,freq4,freq5,freq6,freq7,freq8,freq9};

int max=0;

for(i = 0; i < 9; i++)

{

if(max < test[i])

{

max = test[i];

}

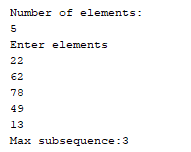
}

System.out.println("Max subsequence:"+max);

}

}

**Screenshot of Output:**

****

**Sr.No:1-3**

**Date: 9-7-19**

**Problem Statement:**

You are given an n\*m grid which contains lower case English letters. How many times does

the phrase saba appear horizontally, vertically, and diagonally in the grid?

**Source Code:**

/\*

\* To change this license header, choose License Headers in Project Properties.

\* To change this template file, choose Tools | Templates

\* and open the template in the editor.

\*/

package saba;

/\*\*

\*

\*

\*/

import java.util.\*;

public class Saba {

/\*\*

\* @param args the command line arguments

\*/

public static void main(String[] args) {

// TODO code application logic here

String search="saba",value;

int counth=0,countv=0,countd=0,r,c,i,j;

System.out.println("Enter number of rows and columns:");

Scanner s=new Scanner(System.in);

r=s.nextInt();

c=s.nextInt();

String[][] mat = new String[r][c];

String test;

System.out.println("Enter elements:");

for(i=0; i<r; i++)

{

for(j=0; j<c; j++)

{

test=s.nextLine();

mat[i][j] = s.next();

}

}

for(i=0; i<r; i++)

{

for(j=0; j<c; j++)

{

System.out.print(mat[i][j]+ " ");

}

System.out.println();

}

for(i=0;i<r;i++)

for(j=0;j<c;j++)

{

if(j+3<c)

{

value=mat[i][j]+mat[i][j+1]+mat[i][j+2]+mat[i][j+3];

if(value.equals(search))

counth++;

}

if(i+3<r)

{

value=mat[i][j]+mat[i+1][j]+mat[i+2][j]+mat[i+3][j];

if(value.equals(search))

countv++;

}

if(i+3<r&&j+3<c)

{

value=mat[i][j]+mat[i+1][j+1]+mat[i+2][j+2]+mat[i+3][j+3];

if(value.equals(search))

countd++;

}

if(j-3>=0)

{

value=mat[i][j]+mat[i][j-1]+mat[i][j-2]+mat[i][j-3];

if(value.equals(search))

counth++;

}

if(i-3>=0)

{

value=mat[i][j]+mat[i-1][j]+mat[i-2][j]+mat[i-3][j];

if(value.equals(search))

countv++;

}

if(i-3>=0&&j-3>=0)

{

value=mat[i][j]+mat[i-1][j-1]+mat[i-2][j-2]+mat[i-3][j-3];

if(value.equals(search))

countd++;

}

}

System.out.println("Horizontal:"+counth);

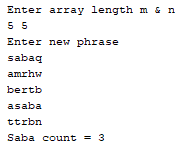
System.out.println("Vertical:"+countv);

System.out.println("Diagonal:"+countd);

}

}

**Screenshot of Output:**

****

**Sr.No: 2-1**

**Date:11-07-19**

**Problem Statement:**

Mr.X wants to change his profile picture on Instagram. Now Instagram has some restriction

over the dimension of picture that we can upload.

Minimum dimension of the picture can be L x L, where L is the length of the side of square.

Now Roy has N photos of various dimensions.

Dimension of a photo is denoted as W x H

where W - width of the photo and H - Height of the photo

When any photo is uploaded following events may occur:

[1] If any of the width or height is less than L, user is prompted to upload another one. Print

&quot;UPLOAD ANOTHER&quot; in this case.

[2] If width and height, both are large enough and

(a) if the photo is already square then it is accepted. Print &quot;ACCEPTED&quot; in this case.

(b) else user is prompted to crop it. Print &quot;CROP IT&quot; in this case.

Given L, N, W and H as input, print appropriate text as output.

**Source Code:**

/\*

\* To change this license header, choose License Headers in Project Properties.

\* To change this template file, choose Tools | Templates

\* and open the template in the editor.

\*/

package insta;

import java.util.Scanner;

/\*\*

\*

\* @author Student

\*/

public class Insta {

/\*\*

\* @param args the command line arguments

\*/

public static void main(String[] args) {

// TODO code application logic here

int l,n,i=0,j=0;

Scanner s=new Scanner(System.in);

System.out.println("Enter min L:");

l=s.nextInt();

System.out.println("Enter number of photos:");

n=s.nextInt();

int[][] a=new int[n][2];

System.out.println("Enter dimensions of photos:");

for(i=0;i<n;i++)

for(j=0;j<2;j++)

{

a[i][j]=s.nextInt();

}

for(i=0;i<n;i++)

for(j=0;j<2;j++)

{

if(a[i][j]<l || a[i][1]<l)

{

System.out.println("Upload another");

break;

}

if(a[i][j]>l || a[i][j]>l)

{

System.out.println("Crop it");

break;

}

if(a[i][j]>=l && a[i][j]==a[i][1])

{

System.out.println("Accepted");

break;

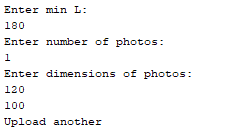
}

}

}

}

**Screenshot of Output:**

****

**Sr.No: 2-2**

**Date:11-07-19**

**Problem Statement:**

Problem Statement

Given a 6\*6 2D Array

1 1 1 0 00

0 1 0 0 00

1 1 1 0 00

0 0 0 0 00

0 0 0 0 00

0 0 0 0 00

We define a pattern in 2D Array to be a subset of values with indices falling in this pattern in 's graphical representation:

a b c

  d

e f g

There are 16 patterns in 2D Array . Calculate the sum for every pattern in the 2D Array , then print the maximum patterns sum.

**Source Code:**

/\*

\* To change this license header, choose License Headers in Project Properties.

\* To change this template file, choose Tools | Templates

\* and open the template in the editor.

\*/

package patternsum;

/\*\*

\*

\* @author

\*/

import java.util.\*;

public class Patternsum {

/\*\*

\* @param args the command line arguments

\*/

public static void main(String[] args) {

// TODO code application logic here

int[][] a=new int[6][6];

int i,j,temp=0,max=0;

Scanner s=new Scanner(System.in);

System.out.println("Enter elements:");

for(i=0;i<6;i++)

for(j=0;j<6;j++)

{

a[i][j]=s.nextInt();

}

for(i=0;i<6;i++)

for(j=0;j<6;j++)

{

if(i+2<6&&j+2<6)

{

temp=a[i][j]+a[i][j+1]+a[i][j+2]+a[i+1][j+1]+a[i+2][j]+a[i+2][j+1]+a[i+2][j+2];

if(temp>=max)

{

max=temp;

}

}

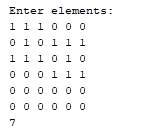
}

System.out.println(max);

}

}

**Screenshot of Output:**

****

**Sr.No:2-3**

**Date:11-07-19**

**Problem Statement:**

Dhananjay has recently learned about ASCII values.He is very fond of experimenting. With

his knowledge of ASCII values and character he has developed a special word and named it

Dhananjays Magical word.

A word which consist of alphabets whose ASCII values is a prime number is an Dhananjays

Magical word. An alphabet is Dhananjays Magical alphabet if its ASCII value is prime.

Dhananjay;s nature is to boast about the things he know or have learnt about. So just to

defame his friends he gives few string to his friends and ask them to convert it to Dhananjay;s

Magical word. None of his friends would like to get insulted. Help them to convert the given

strings to Dhananjays Magical Word.

Rules for converting:

1.Each character should be replaced by the nearest Dhananjay;s Magical alphabet.

2.If the character is equidistant with 2 Magical alphabets. The one with lower ASCII value will be considered as its replacement.

**Source Code:**

/\*

\* To change this license header, choose License Headers in Project Properties.

\* To change this template file, choose Tools | Templates

\* and open the template in the editor.

\*/

package dhanmag;

import java.util.Arrays;

import java.util.Scanner;

import java.math.\*;

/\*\*

\*

\* @author Student

\*/

public class Dhanmag {

/\*\*

\* @param args the command line arguments

\*/

static long nextprime(long n)

{

BigInteger b=new BigInteger(String.valueOf(n));

return Long.parseLong(b.nextProbablePrime().toString());

}

public static void main(String[] args) {

// TODO code application logic here

int n,c,i,j,m,flag=0;

Scanner s=new Scanner(System.in);

System.out.println("Enter number of names:");

n=s.nextInt();

System.out.println("Enter number of char:");

c=s.nextInt();

String[] a=new String[n];

String b;

char[][] x=new char[n][c];

int[][] q=new int[n][c];

BigInteger[][] q2=new BigInteger[n][c];

BigInteger qq;

System.out.println("Enter names:");

for(i=0;i<n;i++)

{

b=s.next();

a[i]=b;

}

for(i=0;i<n;i++)

for(j=0;j<c;j++)

{

x[i][j]=a[i].charAt(j);

}

// for(i=0;i<n;i++)

// for(j=0;j<c;j++)

// {

// System.out.println(x[i][j]);

// }

for(i=0;i<n;i++)

for(j=0;j<c;j++)

{

q[i][j]=(int)x[i][j];

}

long[] p=new long[100];

long[] ascii=new long[100];

char[] dhan=new char[100];

int k=0;

for(i=0;i<n;i++)

for(j=0;j<c;j++)

{

p[k]=q[i][j];

ascii[k]=nextprime(p[k]);

k++;

}

/// for(i=0;i<k;i++)

// System.out.println("a"+ascii[i]);

for(i=0;i<k;i++)

dhan[i]=(char)ascii[i];

for(i=0;i<k;i++)

{

if(i%c==0)

System.out.println();

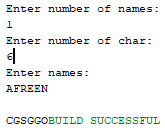
System.out.print(dhan[i]);

}

}

}

**Screenshot of Output:**

****

**Sr.No: 3-1**

**Date: 17-07-19**

**Problem Statement:**

Tower of Hanoi consists of three pegs or towers with n disks placed one over the other.

The objective of the puzzle is to move the stack to another peg following these simple rules.

1: Only one disk can be moved at a time.

2: No disk can be placed on top of the smaller disk.

**Source Code:**

/\*

\* To change this license header, choose License Headers in Project Properties.

\* To change this template file, choose Tools | Templates

\* and open the template in the editor.

\*/

package hanoi;

import java.util.Scanner;

/\*\*

\*

\* @author Student

\*/

public class Hanoi {

/\*\*

\* @param args the command line arguments

\*/

public void tow(int n,String begin,String aux,String end) {

if (n==1)

{

System.out.println(begin+ "->" +end);

}

else

{

tow(n-1,begin,end,aux);

System.out.println(begin+ "->" +end);

tow(n-1,aux,begin,end);

}

}

public static void main(String[] args) {

// TODO code application logic here

int n,steps=0;

Hanoi h=new Hanoi();

System.out.println("Enter number of rings:");

Scanner s=new Scanner(System.in);

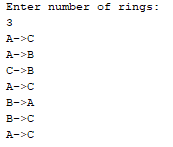
n=s.nextInt();

h.tow(n,"A","B","C");

}

}

**Screenshot of Output:.**

****

**Sr.No: 3-2**

**Date:17-07-19**

**Problem Statement:**

Simran is running up a staircase with N steps, and can hop(jump) either 1 step, 2 steps or 3

steps at a time.You have to count, how many possible ways Simran can run up to the stairs.

**Source Code:**

/\*

\* To change this license header, choose License Headers in Project Properties.

\* To change this template file, choose Tools | Templates

\* and open the template in the editor.

\*/

package stairs2;

import java.util.Scanner;

/\*\*

\*

\* @author Student

\*/

public class Stairs2 {

static int count;

public static int count(int n)

{

if(n<=1)

{

return 1;

}

else

{

if(n==2)

{

return count(n-1)+count(n-2);

}

else

{

return count(n-1)+count(n-2)+count(n-3);

}

}

}

public static void main(String[] args) {

// TODO code application logic here

int n,steps=0;

Stairs2 st=new Stairs2();

System.out.println("Enter number of stairs:");

Scanner s=new Scanner(System.in);

n=s.nextInt();

System.out.println(count(n));

}

}

**Screenshot of Output:**

****

**Sr.No:4-1**

**Date:19-7-19**

**Problem Statement:**

You have an empty sequence, and you will be given N queries. Each query is one of these

three types:

1 x -Push the element x into the stack.

2 -Delete the element present at the top of the stack.

3 -Print the maximum element in the stack.

**Source Code:**

package stacks;

import java.util.Scanner;

import java.util.Stack;

public class Stacks {

public static void main(String[] args) {

// TODO code application logic here

int n,op,i,e,temp;

Stack<Integer> st = new Stack<Integer>();

Stack<Integer> st2 = new Stack<Integer>();

Scanner s=new Scanner(System.in);

System.out.println("Enter number of queries:");

n=s.nextInt();

for(i=0;i<n;i++)

{

System.out.println("Enter operation:\n1.Push 2.Pop 3.Max");

op=s.nextInt();

if(op==1)

{

System.out.println("Enter element:");

e=s.nextInt();

st.push(e);

if(st.size()==1)

{

st2.push(e);

}

if(e>st2.peek())

{

st2.push(e);

}

}

else if(op==2)

{

if(st.isEmpty())

{

System.out.println("Stack underflow");

System.exit(0);

}

else

{

temp=st.pop();

if(st2.peek()==temp)

{

st2.pop();

}

}

}

else if(op==3)

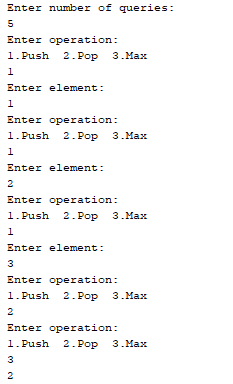
System.out.println(st2.peek());

}

}

}

**Screenshot of Output:**

****

**Sr.No:4-2**

**Date:19-7-19**

**Problem Statement:**

A bracket is considered to be any one of the following characters: (, ), {, }, [, or ].

Two brackets are considered to be a matched pair if the an opening bracket (i.e., (, [, or {) occurs to the left of a closing bracket (i.e., ), ], or }) of the exact same type. There are three types of matched pairs of brackets: [], {},and ().

A matching pair of brackets is not balanced if the set of brackets it encloses are not matched. For example, {[(])} is not balanced because the contents in between { and } are not balanced. The pair of square brackets encloses a single, unbalanced opening bracket, (, and the pair of parentheses encloses a single, unbalanced closing square bracket, ].

By this logic, we say a sequence of brackets is balanced if the following conditions are met:

It contains no unmatched brackets.

The subset of brackets enclosed within the confines of a matched pair of brackets is also a matched pair of brackets.

Given n strings of brackets, determine whether each sequence of

brackets is balanced. If a string is balanced, return YES. Otherwise, return NO.

**Source Code:**

import java.util.Scanner;

import java.util.Stack;

public class Stack3 {

public static void main(String[] args) {

int i;

Stack<Character> stk = new Stack();

Scanner input = new Scanner(System.in);

System.out.println("Enter n number of values");

int number = input.nextInt();

int num=number+1;

String a[]=new String[num];

for(i=0; i<num;i++)

{

a[i] = input.nextLine();

}

char ch;

for(i=0; i<num;i++)

{

String g=a[i];

OUTER\_LOOP:

for(int j=0;j<g.length();j++)

{

ch=g.charAt(j);

System.out.println(ch);

if(ch=='{' || ch=='(' || ch=='[')

{

stk.push(ch);

}

else

{

if(ch==']' && stk.peek()=='[' || ch=='}' && stk.peek()=='{' || ch==')' && stk.peek()=='(')

{

stk.pop();

if(stk.isEmpty()) {

System.out.println("balanced");

}

}

else

{

stk.clear();

System.out.println("unbalanced");

break OUTER\_LOOP;

}

}

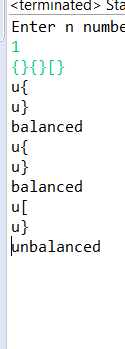
}

}

}

}

**Screenshot of Output:**



**Sr.No: 5-1**

**Date:24-7-19**

**Problem Statement:**

A linked list is said to contain a cycle if any node is visited more than once while traversing

the list.

Your function must return a boolean denoting whether or not there is a cycle in the list. If there

is a cycle, return true; otherwise, return false.

**Source Code:**

import java.util.LinkedList;

import java.util.Scanner;

public class linkedlist {

Node head;// head of list

Node link;

/\* Node Class \*/

class Node

{

int data;

Node next;

// Constructor to create a new node

Node(int d) {data = d; next = null; }

}

public void insert (int a) {

Node ent = new Node(a);

ent.next = null;

if (head == null)

{

head = ent;

link = ent;

}

else {

link.next=ent;

link=ent;

}

}

public void loop() {

Node last=head;

while(last.next != null) {

last=last.next;

}

last.next=head.next;

}

public void loopdetect() {

Node temp1 = head;

Node temp2 = head;

while(temp2.next.next != null) {

temp1=temp1.next;

temp2=temp2.next.next;

if(temp1==temp2){

System.*out*.println("loop detected");

break;

}

}

}

public void removeloop() {

Node temp1 = head;

Node temp2 = head;

while(temp2.next.next != null) {

temp1=temp1.next;

temp2=temp2.next.next;

if(temp1==temp2){

System.*out*.println("loop detected");

break;

}

}

temp2 = head.next;

int loopLength = 1;

while(temp2 != temp1) {

temp2=temp2.next;

loopLength++;

}

System.*out*.println("Loop length is " + loopLength);

temp2 = head.next;

while(temp2.next != temp1) {

temp2=temp2.next;

}

temp2.next=null;

System.*out*.println("Loop breaks");

}

public static void main(String[] args) {

linkedlist list = new linkedlist();

int n,i,m,c=0;

Scanner input = new Scanner(System.*in*);

System.*out*.println("how many numbers you want to insert in linked list");

n=input.nextInt();;

System.*out*.println("Enter numricals in linked list ");

for (i=0;i<n;i++)

{

m = input.nextInt();

list.insert (m);

}

list.loop();

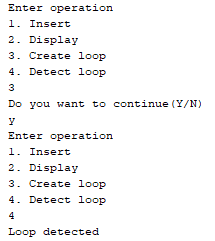
list.loopdetect();

list.removeloop();

}

}

**Screenshot of Output:**



**Sr.No:5-2**

**Date:24-7-19**

**Problem Statement:**

Given a linked list, rotate the list to the right by k places, where k is non-negative.

**Source Code:**

import java.util.InputMismatchException;

import java.util.Scanner;

public class Rotatelist {

Node head;// head of list

Node link;

/\* Node Class \*/

class Node {

int data;

Node next;

// Constructor to create a new node

Node(int d) {

data = d;

next = null;

}

}

public void insert(int a) {

Node ent = new Node(a);

ent.next = null;

if (head == null) {

head = ent;

link = ent;

} else {

link.next = ent;

link = ent;

}

}

public void rotate(int b) {

Node a = head;

Node x = null;

for (int i = 0; i < b; i++) {

while (a.next != null) {

x = a;

a = a.next;

}

x.next = null;

a.next = head;

head = a;

}

}

public void print() {

Node curr = head;

while (curr != null) {

System.out.println("node:" + curr.data);

curr = curr.next;

}

}

public static void main(String[] args) {

// TODO Auto-generated method stub

Rotatelist list = new Rotatelist();

int n, i, m, c = 0;

boolean st = false;

Scanner input = new Scanner(System.in);

while (!st) {

try {

System.out.println("Press : 1 :: To insert in list");

System.out.println("Press : 2 :: To rotate list");

System.out.println("Press : 3 :: To print list");

System.out.println("Press : 4 :: To stop");

n = input.nextInt();

switch (n) {

case 1:

System.out.println("Enter numricals in linked list ");

m = input.nextInt();

list.insert(m);

break;

case 2:

System.out.println("Enter number of rotations");

m = input.nextInt();

list.rotate(m);

break;

case 3:

System.out.println("Printing....");

list.print();

break;

case 4:

System.out.println("Stoping.......");

System.out.println("THANK YOU!!!!!");

st = true;

break;

default:

System.out.println("Choose correct number");

break;

}

} catch (InputMismatchException e) {

System.out.println("Enter numericals");

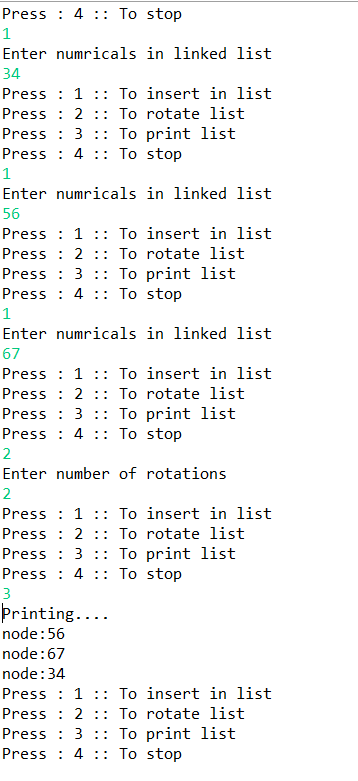
}

}

}

}

**Screenshot of Output:**



**Sr.No: 6**

**Date: 30-07-19**

**Problem Statement:** You are given a pointer to the root of a binary search tree and values to be inserted into the tree. Insert the values into their appropriate position in the binary search tree.

The program should be able to insert, delete nodes and display all the elements of the tree.

**Source Code:**

package bst;

import java.util.Scanner;

class node

{

int data;

node left,right;

node(int d)

{

data=d;

left=null;

right=null;

}

}

public class Bst {

/\*\*

\* @param args the command line arguments

\*/

node root=null;

node cp;

int data;

void insert(int key)

{

root=insertRec(root, key);

}

node insertRec(node root, int key)

{

if (root==null)

{

root=new node(key);

return root;

}

if(key<root.data)

root.left=insertRec(root.left, key);

else if(key>root.data)

root.right=insertRec(root.right, key);

return root;

}

void inorder()

{

inorderRec(root);

}

void inorderRec(node root)

{

if (root!=null)

{

inorderRec(root.left);

System.out.println(root.data);

inorderRec(root.right);

}

}

public static void main(String[] args) {

// TODO code application logic here

Bst a=new Bst();

Scanner s=new Scanner(System.in);

int choice,element;

char ch;

do

{

System.out.println("Enter operation");

System.out.println("1. Insert");

System.out.println("2. Display");

choice=s.nextInt();

switch(choice)

{

case 1:

System.out.println("Enter element:");

element=s.nextInt();

a.insert(element);

break;

case 2:

a.inorder();

break;

default :

System.out.println("Invalid option");

break;

}

System.out.println("Do you want to continue(Y/N)");

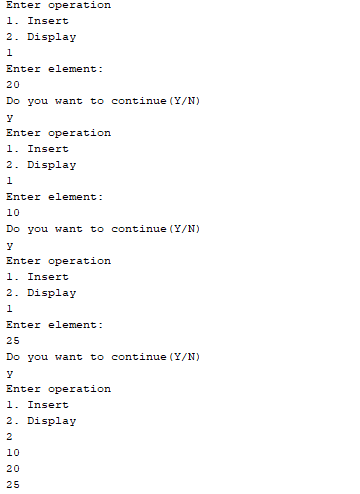
ch=s.next().charAt(0);

}while(ch=='Y'||ch=='y');

}

}

**Screenshot of Output:**

****

**References:**

<https://www.geeksforgeeks.org/binary-search-tree-set-1-search-and-insertion/>

**Sr.No:7**

**Date: 13-08-19**

**Problem Statement:** An AVL tree is a binary search tree in which the balance factor of every node, which is defined as the difference between the heights of the node’s left & right sub trees Construction of AVL tree is same as Binary search tree, but the difference is after inserting the new node into the tree or deleting a node from the tree, to check each node for the balance. If the tree is not balanced, we need to restructure or rebuild the tree for balancing using rotation.

**Source Code:**

package bstavl;

import static java.lang.Math.max;

import java.util.Scanner;

class node

{

int data,height;

node left,right;

node(int d)

{

data=d;

height=1;

left=null;

right=null;

}

}

public class Bstavl {

node root=null;

node cp;

int data;

int height (node N)

{

if (N == null)

return 0;

return N.height;

}

int getBalance(node N) {

if (N == null)

return 0;

return height(N.left) - height(N.right);

}

void insert(int key)

{

root=insertRec(root, key);

}

node insertRec(node root, int key)

{

if (root==null)

{

root=new node(key);

return root;

}

if(key<root.data)

root.left=insertRec(root.left, key);

else if(key>root.data)

root.right=insertRec(root.right, key);

root.height = Math.max(height(root.left), height(root.right)) + 1;

int balance = getBalance(root);

//Left left

if (balance > 1 && key < root.left.data)

return rightRotate(root);

//Right Right

if (balance < -1 && key > root.right.data)

return leftRotate(root);

//Left Right

if (balance > 1 && key > root.left.data) {

root.left = leftRotate(root.left);

return rightRotate(root);

}

//Right Left

if (balance < -1 && key < root.right.data) {

root.right = rightRotate(root.right);

return leftRotate(root);

}

return root;

}

node rightRotate(node a)

{

node b = a.left;

node c = b.right;

b.right = a;

a.left = c;

a.height = Math.max(height(a.left), height(a.right))+1;

b.height = Math.max(height(b.left), height(b.right))+1;

return b;

}

node leftRotate(node a)

{

node b = a.right;

node c = b.left;

b.left = a;

a.right = c;

a.height = Math.max(height(a.left), height(a.right))+1;

b.height = Math.max(height(b.left), height(b.right))+1;

return b;

}

void del(int key)

{

root=delRec(root, key);

}

node delRec(node root,int key)

{

if(key<root.data)

root.left=delRec(root.left,key);

else if(key>root.data)

root.right=delRec(root.right,key);

else

{

if(root.left==null)

root=root.right;

else if(root.right==null)

root=root.left;

else if(root.left!=null && root.right!=null)

{

root.data=maxval(root.left);

root.left=delRec(root.left,root.data);

}

}

root.height = Math.max(height(root.left), height(root.right)) + 1;

int balance = getBalance(root);

// Left Left Case

if (balance > 1 && getBalance(root.left) >= 0)

return rightRotate(root);

// Left Right Case

if (balance > 1 && getBalance(root.left) < 0)

{

root.left = leftRotate(root.left);

return rightRotate(root);

}

// Right Right Case

if (balance < -1 && getBalance(root.right) <= 0)

return leftRotate(root);

// Right Left Case

if (balance < -1 && getBalance(root.right) > 0)

{

root.right = rightRotate(root.right);

return leftRotate(root);

}

return root;

}

int maxval(node root)

{

int maxv=root.data;

while(root.right!=null)

root=root.right;

maxv=root.data;

return maxv;

}

void preorder()

{

preorderRec(root);

}

void preorderRec(node root)

{

if (root!=null)

{

System.out.println(root.data);

preorderRec(root.left);

preorderRec(root.right);

}

}

public static void main(String[] args) {

// TODO code application logic here

Bstavl a=new Bstavl();

Scanner s=new Scanner(System.in);

int choice,element,h;

char ch;

do

{

System.out.println("Enter operation");

System.out.println("1. Insert");

System.out.println("2. Preorder");

System.out.println("3. Delete");

System.out.println("4. Test insert");

// System.out.println("4. Height");

choice=s.nextInt();

switch(choice)

{

case 1:

System.out.println("Enter element:");

element=s.nextInt();

a.insert(element);

break;

case 2:

a.preorder();

break;

case 3:

System.out.println("Enter element:");

element=s.nextInt();

a.del(element);

break;

case 4:

a.insert(9);

a.insert(5);

a.insert(10);

a.insert(0);

a.insert(6);

a.insert(11);

a.insert(-1);

a.insert(1);

a.insert(2);

break;

default :

System.out.println("Invalid option");

break;

}

System.out.println("Do you want to continue(Y/N)");

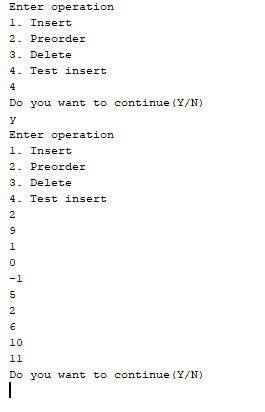
ch=s.next().charAt(0);

}while(ch=='Y'||ch=='y');

}

}

**Screenshot of Output:**

****

**References:**

<https://www.geeksforgeeks.org/avl-tree-set-1-insertion/>

**Sr.No: 8**

**Date: 20-08-19**

**Problem Statement**: Huffman encoding is a way to assign binary codes to symbols that reduces the overall number of bits used to encode a typical string of those symbols.

For example, if you use letters as symbols and have details of the frequency of occurrence of

those letters in typical strings, then you could just encode each letter with a fixed number of

bits, such as in ASCII codes. You can do better than this by encoding more frequently

occurring letters such as e and a, with smaller bit strings; and less frequently occurring letters

such as q and x with longer bit strings. Any string of letters will be encoded as a string of bits that are no-longer of the same length per letter. The Huffman coding scheme takes each symbol and its weight (or frequency of occurrence), and generates proper encodings for each symbol taking account of the weights of each symbol, so that higher weighted symbols have fewer bits in their encoding. A Huffman encoding can be computed by first creating a tree of nodes:

Huffman coding

Create a leaf node for each symbol and add it to the priority queue.

While there is more than one node in the queue:

Remove the node of highest priority (lowest probability) twice to get two nodes.

Create a new internal node with these two nodes as children and with probability equal to the

sum of the two nodes&#39; probabilities.

Add the new node to the queue.

The remaining node is the root node and the tree is complete.

**Source Code:**

package huffman;

import java.util.Comparator;

import java.util.HashMap;

import java.util.Iterator;

import java.util.Map;

import java.util.PriorityQueue;

import java.util.Scanner;

import java.util.Set;

class hnode

{

int freq;

char data;

hnode left,right;

hnode(char c,int d)

{

freq=d;

data=c;

left=null;

right=null;

}

}

class huffcomp implements Comparator<hnode>

{

@Override

public int compare(hnode n1,hnode n2)

{

if(n1.freq>n2.freq)

return 1;

else if(n1.freq<n2.freq)

return -1;

return 0;

}

}

public class Huffman {

public void printcode(hnode par,String s)

{

if(par.left==null&&par.right==null)

{

System.out.println(par.data+":"+s);

return;

}

printcode(par.left,s+"0");

printcode(par.right,s+"1");

}

public static void main(String[] args) {

HashMap<Character, Integer> map= new HashMap<Character, Integer>();

Huffman hu=new Huffman();

Scanner s=new Scanner(System.in);

int choice,element;

String st;

System.out.println("Enter string");

st=s.next();

PriorityQueue<hnode> pq = new PriorityQueue<hnode>(st.length(),new huffcomp());

for (int i = 0; i < st.length(); i++)

{

char ch = st.charAt(i);

if(map.containsKey(ch))

map.put(ch,map.get(ch)+1);

else

map.put(ch,1);

}

System.out.println(map);

Set<Character> keySet = map.keySet();

for (Map.Entry mapElement:map.entrySet())

{

hnode h=new hnode((char)mapElement.getKey(),(int)mapElement.getValue());

pq.add(h);

}

while(pq.size()>1)

{

hnode lc=pq.remove();

System.out.println(lc.data);

hnode rc=pq.remove();

System.out.println(rc.data);

hnode par=new hnode('-',lc.freq+rc.freq);

par.left=lc;

par.right=rc;

pq.add(par);

}

hnode test=pq.peek();

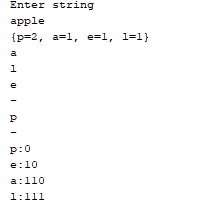
String str="";

hu.printcode(test,str);

}

}

**Screenshot of Output:**

****

**References:**

<https://www.geeksforgeeks.org/huffman-coding-greedy-algo-3/>

**Sr.No: 9-1**

**Date: 17-09-19**

**Problem Statement:**

In Open Addressing, all elements are stored in the hash table itself. So at any point, size of

the table must be greater than or equal to the total number of keys.

Insert(k): Keep probing until insert k.

Search(k): Keep probing until slot’s key doesn’t become equal to k or an empty slot is

reached.

Delete(k): Delete operation is interesting. If we simply delete a key, then search may fail.

So slots of deleted keys are marked specially as “deleted”.

Insert can insert an item in a deleted slot, but the search doesn’t stop at a deleted slot.

**Source Code:**

package openaddressing;

import java.util.Scanner;

public class OpenAddressing {

static int hashTable[] = new int[10];

public static void main(String[] args) {

// TODO code application logic here

OpenAddressing oa=new OpenAddressing();

for (int i=0;i<10;i++)

hashTable[i] = 0;

try {

Scanner input = new Scanner(System.in);

System.out.println("Enter number:");

for (int i=0;i<10;i++) {

int num=input.nextInt();

int index=num%10;

if(hashTable[index]==0){

hashTable[index] = num;

}

else {

while (hashTable[index] != 0) {

index++;

if (index > 9) {

index = 0;

}

}

hashTable[index] = num;

}

}

System.out.println("Enter the number for searching:");

int SearchNumber=input.nextInt();

oa.Search(SearchNumber, hashTable);

System.out.println("Enter the number for deletion:");

int DeleteNumber=input.nextInt();

oa.Delete(DeleteNumber, hashTable);

System.out.println("HashTable:");

for (int i=0;i<10;i++)

{

System.out.println(hashTable[i]);

}

} catch (NullPointerException e) {

System.out.println(e);

}

}

void Search(int SearchNumber,int hashTable[]) {

boolean Stop = false;

int KeyCode = SearchNumber % 10;

int NoNumber = KeyCode;

if (hashTable[KeyCode] == SearchNumber) {

System.out.println("The number is at " + KeyCode);

} else {

while (hashTable[KeyCode] != SearchNumber && !Stop) {

KeyCode++;

if (KeyCode > 9) {

KeyCode = 0;

} else if (NoNumber == KeyCode) {

Stop = true;

}

}

if (Stop == true) {

System.out.println("The number " + SearchNumber + " is not present");

} else {

System.out.println("The number is stored at " + KeyCode );

}

}

}

void Delete(int SearchNumber,int hashTable[]) {

boolean Stop = false;

int KeyCode = SearchNumber % 10;

int NoNumber = KeyCode;

if (hashTable[KeyCode] == SearchNumber) {

System.out.println("The number is at " + KeyCode);

} else {

while (hashTable[KeyCode] != SearchNumber && !Stop) {

KeyCode++;

if (KeyCode > 9) {

KeyCode = 0;

} else if (NoNumber == KeyCode) {

Stop = true;

}

}

if (Stop == true) {

System.out.println("The number " + SearchNumber + " is not present");

} else {

System.out.println("The number is stored at " + KeyCode );

hashTable[KeyCode]=0;

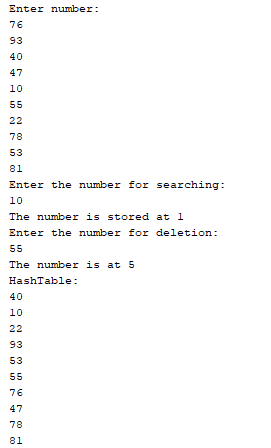
}

}

}

}

**Screenshot of Output:**

****

**References:**

<https://www.geeksforgeeks.org/hashing-set-3-open-addressing/>

**Sr.No: 9-2**

**Date: 24-09-19**

**Problem Statement:** There are several websites on the Internet that allow users to perform queries on flight databases to find flights between various cities, typically with the intent to buy a ticket. To make a query, a user specifies origin and destination cities, a departure date, and a departure time. To support such queries, we can model the flight database as a map,

where keys are Flight objects that contain fields corresponding to these four parameters.

That is, a key is a tuple k = (origin, destination, date, time).

Additional information about a flight, such as the flight number, the number of seats still

available in first (F) and coach (Y) class, the flight duration, and the fare, can be stored in

the value object.

Finding a requested flight is not simply a matter of finding an exact match for a requested

query. Although a user typically wants to exactly match the origin and destination cities, he

or she may have flexibility for the departure date, and certainly will have some flexibility for

the departure time on a specific day. We can handle such a query by ordering our keys

lexicographically. Then, an efficient implementation for a sorted map would be a good way

to satisfy users’ queries.

**Source Code:**

package flightgraph;

import java.util.ArrayList;

import java.util.HashMap;

import java.util.Iterator;

import java.util.LinkedList;

import java.util.List;

import java.util.Scanner;

class flight

{

String origin,dest;

int date,time;

flight(String ori,String desti,int da,int ti)

{

origin=ori;

dest=desti;

date=da;

time=ti;

}

}

public class FlightGraph {

static HashMap<Integer, List<flight>> map1 = new HashMap<Integer, List<flight>>();

public static void main(String[] args) {

// TODO code application logic here

int choice;

//flight object=null;

String origin="",destination="";

int date=0,time=0,flightNum=0;

Scanner input=new Scanner(System.in);

do{

System.out.println("Enter flight number:");

flightNum=input.nextInt();

System.out.println("Enter origin:");

origin=input.next();

System.out.println("Enter destination:");

destination=input.next();

System.out.println("Enter date:");

date=input.nextInt();

System.out.println("Enter time:");

time=input.nextInt();

System.out.println(flightNum);

System.out.println(origin);

System.out.println(destination);

System.out.println(date);

System.out.println(time);

flight object=new flight(origin,destination,date,time);

if (map1.containsKey(flightNum)) {

map1.get(flightNum).add(object);

} else {

LinkedList<flight> link=new LinkedList<flight>();

link.add(object);

map1.put(flightNum,link);

}

for (HashMap.Entry<Integer, List<flight>> entry : map1.entrySet())

{

int key=entry.getKey();

System.out.print(key + " : ");

Iterator it = entry.getValue().iterator();

while (it.hasNext()) {

flight pnode=(flight)it.next();

System.out.print(pnode.origin + " ");

System.out.print(pnode.dest + " ");

System.out.print(pnode.date + " ");

System.out.print(pnode.time + " ");

}

System.out.println();

}

System.out.println("Enter 1 to continue");

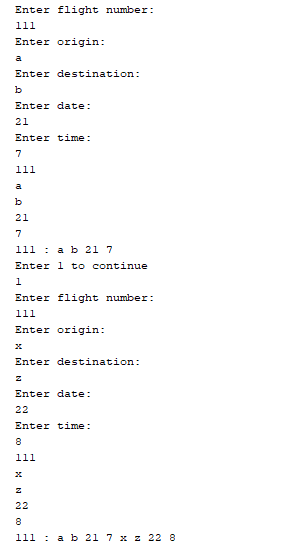
choice=input.nextInt();

}while(choice==1);

}

}

**Screenshot of Output:**

****

**References:**

<https://www.geeksforgeeks.org/hashing-set-3-open-addressing/>