Algo for ICPC

**Declare Two Decleration**

int[][] myArray = new int[3][5] ;

int[][] myArray = { {0,0,0,0,0}, {0,0,0,0,0}, {0,0,0,0,0} };

**Array Filling At Once**

int [] myarray = new int[num\_elts];

Arrays.fill(myarray, 42);

**Summing two array**

import java.util.stream.\*;

int[] a = {10,20,30,40,50};

int sum = IntStream.of(a).sum();

System.out.println("The sum is " + sum);

**Convert One Base to Another**

System.out.println(Integer.toString(Integer.parseInt("A", 16), 2));

**Convert DecimalFormat**

System.out.println((new java.text.DecimalFormat(“#.00”)).format(3.112121212));

**String Builder Usage**

(new StringBuilder(String.valueOf(palindrome)).reverse().toString()).equals(String.valueOf(palindrome));

**Convert character to ASCII**

int ascii = (int) character;

**Convert ASCII to character**

char character = (char)ascii;

**Template**

-BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

int input = Integer.parseInt(br.readLine());

while (input != -1)

{

input = Integer.parseInt(br.readLine());

}

}

**Template 2**

-BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

int testcase = Integer.parseInt(br.readLine());

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

String line = br.readLine();

}

**Prime Generator**

-public class PrimeGenerator {

public static void main (String[] args)

{

int i =0;

int num =0;

//Empty String

String primeNumbers = "";

int c=0;

for (i = 1; i <= 1000; i+=2)

{

int counter=0;

for(num =i; num>=1; num-=2)

{

if(i%num==0)

{

counter = counter + 1;

}

}

if (counter ==2)

{

//Appended the Prime number to the String

primeNumbers = primeNumbers + i + " ";

c++;

}

}

System.out.println("Prime numbers from 1 to 100 are :");

System.out.println(primeNumbers);

System.out.println(c);

}

}

**Polygon**

var xPts = [4, 4, 8, 8, -4,-4];

var yPts = [6, -4, -4, -8, -8, 6];

var a = polygonArea(xPts, yPts, 6); // 6 -> A NAR

alert("Area = " + a);

function polygonArea(X, Y, numPoints)

{

area = 0; // Accumulates area in the loop

j = numPoints-1; // The last vertex is the 'previous' one to the first

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

{ area = area + (X[j]+X[i]) \* (Y[j]-Y[i]);

j = i; //j is previous vertex to i

}

return area/2;

}

**ArrayList**

import java.util.\*;

public class ArrayListDemo {

public static void main(String args[]) {

// create an array list

ArrayList al = new ArrayList();

System.out.println("Initial size of al: " + al.size());

// add elements to the array list

al.add("C");

al.add("A");

al.add("E");

al.add("B");

al.add("D");

al.add("F");

al.add(1, "A2");

System.out.println("Size of al after additions: " + al.size());

// display the array list

System.out.println("Contents of al: " + al);

// Remove elements from the array list

al.remove("F");

al.remove(2);

System.out.println("Size of al after deletions: " + al.size());

System.out.println("Contents of al: " + al);

}

}

Can use

* boolean contains(Object o)

**Hash**

import java.util.\*;

public class HashMapDemo {

public static void main(String args[]) {

// Create a hash map

HashMap hm = new HashMap();

// Put elements to the map

hm.put("Zara", new Double(3434.34));

hm.put("Mahnaz", new Double(123.22));

hm.put("Ayan", new Double(1378.00));

hm.put("Daisy", new Double(99.22));

hm.put("Qadir", new Double(-19.08));

// Get a set of the entries

Set set = hm.entrySet();

// Get an iterator

Iterator i = set.iterator();

// Display elements

while(i.hasNext()) {

Map.Entry me = (Map.Entry)i.next();

System.out.print(me.getKey() + ": ");

System.out.println(me.getValue());

}

System.out.println();

// Deposit 1000 into Zara's account

double balance = ((Double)hm.get("Zara")).doubleValue();

hm.put("Zara", new Double(balance + 1000));

System.out.println("Zara's new balance: " + hm.get("Zara"));

}

}

can use

* boolean containsKey(Object key)
* boolean containsValue(Object value)
* Object put(Object key, Object value)
* boolean isEmpty()

- int size()

**Stack**

public class StackTest {

static void showpush(Stack st, int a) {

st.push(new Integer(a));

System.out.println("push(" + a + ")");

System.out.println("stack: " + st);

}

static void showpop(Stack st) {

System.out.print("pop -> ");

Integer a = (Integer) st.pop();

System.out.println(a);

System.out.println("stack: " + st);

}

public static void main(String args[]) {

Stack st = new Stack();

System.out.println("stack: " + st);

showpush(st, 42);

showpush(st, 66);

showpush(st, 99);

showpop(st);

showpop(st);

showpop(st);

try {

showpop(st);

}catch (EmptyStackException e) {

System.out.println("empty stack");

}

}

}

**Sorting**

**Bubble Sort**

// Bubble Sort Algorithm in Ascending Order

public static int[] CrunchifyBubbleSortAsceMethod(int[] crunchifyArr) {

int temp;

for (int i = 0; i < crunchifyArr.length - 1; i++) {

for (int j = 1; j < crunchifyArr.length - i; j++) {

if (crunchifyArr[j - 1] > crunchifyArr[j]) {

temp = crunchifyArr[j - 1];

crunchifyArr[j - 1] = crunchifyArr[j];

crunchifyArr[j] = temp;

}

}

log("Iteration " + (i + 1) + ": " + Arrays.toString(crunchifyArr));

}

return crunchifyArr;

}

**QuickSort**

public class QuickSort {

    private int array[];

    private int length;

    public void sort(int[] inputArr) {

        if (inputArr == null || inputArr.length == 0) {

            return;

        }

        this.array = inputArr;

        length = inputArr.length;

        quickSort(0, length - 1);

    }

    private void quickSort(int lowerIndex, int higherIndex) {

        int i = lowerIndex;0

        int j = higherIndex;

        // calculate pivot number, I am taking pivot as middle index number

        int pivot = array[lowerIndex+(higherIndex-lowerIndex)/2];

        // Divide into two arrays

        while (i <= j) {

            /\*\*

             \* In each iteration, we will identify a number from left side which

             \* is greater then the pivot value, and also we will identify a number

             \* from right side which is less then the pivot value. Once the search

             \* is done, then we exchange both numbers.

             \*/

            while (array[i] < pivot) {

                i++;

            }

            while (array[j] > pivot) {

                j--;

            }

            if (i <= j) {

                exchangeNumbers(i, j);

                //move index to next position on both sides

                i++;

                j--;

            }

        }

        // call quickSort() method recursively

        if (lowerIndex < j)

            quickSort(lowerIndex, j);

        if (i < higherIndex)

            quickSort(i, higherIndex);

    }

    private void exchangeNumbers(int i, int j) {

        int temp = array[i];

        array[i] = array[j];

        array[j] = temp;

    }

    public static void main(String a[]){

        MyQuickSort sorter = new MyQuickSort();

        int[] input = {24,2,45,20,56,75,2,56,99,53,12};

        sorter.sort(input);

        for(int i:input){

            System.out.print(i);

            System.out.print(" ");

        }

    }

}

Vector

import java.util.\*;

public class VectorDemo {

public static void main(String args[]) {

// initial size is 3, increment is 2

Vector v = new Vector(3, 2);

System.out.println("Initial size: " + v.size());

System.out.println("Initial capacity: " + v.capacity());

v.addElement(new Integer(1));

v.addElement(new Integer(2));

v.addElement(new Integer(3));

v.addElement(new Integer(4));

System.out.println("Capacity after four additions: " + v.capacity());

v.addElement(new Double(5.45));

System.out.println("Current capacity: " + v.capacity());

v.addElement(new Double(6.08));

v.addElement(new Integer(7));

System.out.println("Current capacity: " + v.capacity());

v.addElement(new Float(9.4));

v.addElement(new Integer(10));

System.out.println("Current capacity: " + v.capacity());

v.addElement(new Integer(11));

v.addElement(new Integer(12));

System.out.println("First element: " + (Integer)v.firstElement());

System.out.println("Last element: " + (Integer)v.lastElement());

if(v.contains(new Integer(3)))

System.out.println("Vector contains 3.");

// enumerate the elements in the vector.

Enumeration vEnum = v.elements();

System.out.println("\nElements in vector:");

while(vEnum.hasMoreElements())

System.out.print(vEnum.nextElement() + " ");

System.out.println();

}

}

Set

-

**Queue**

-public static void main(String[] args) {

Queue queueA = new LinkedList();

queueA.add("element 0");

queueA.add("element 1");

queueA.add("element 2");

queueA.remove();

queueA.add("element 0");

//access via Iterator

Iterator iterator = queueA.iterator(); // Data Output

while(iterator.hasNext()){

String element = (String) iterator.next();

}

//access via new for-loop

for(Object object : queueA) {

String element = (String) object;

System.out.println(element);

}

}

**Shortest Path**

import java.util.StringTokenizer;

import java.util.Vector;

public class ICPCTest {

public static void main(String[] args) {

int a[][] = {

{10000, 8, 10000, 10, 1},

{8, 10000, 6, 1, 4},

{10000, 6, 10000, 1, 15},

{10, 1, 1, 10000, 8},

{1, 4, 15, 8, 10000}};

String b[][] = new String[a.length][a.length];

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

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

if (a[i][j] < 10000) {

b[i][j] = (i + 1) + "," + (j + 1);

}

}

}

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

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

if (i != j) {

for (int k = 0; k < a.length; k++) {

if ( j != k && i != k) {

if (a[i][j] + a[j][k] <= a[i][k]) {

a[i][k] = a[i][j] + a[j][k];

a[k][i] = a[i][j] + a[j][k];

b[i][k] = (i + 1) + "," + (j + 1) + "," + (k + 1);

b[k][i] = (k + 1) + "," + (j + 1) + "," + (i + 1);

}

}

System.out.println("");

}

}

}

}

System.out.println("");

System.out.println("");

//Shortest weight output

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

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

System.out.print(a[i][j] + "\t");

}

System.out.println("");

}

System.out.println("");

System.out.println("");

//Shortest path output

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

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

System.out.print(b[i][j] + "\t");

}

System.out.println("");

}

print1(b);

}

static void print1(String a[][]) {

int s = 1 - 1, d = 3 - 1;

Vector v = new Vector();

StringTokenizer st = new StringTokenizer(a[s][d], ",");

while (st.hasMoreTokens()) {

v.add(st.nextToken().toString());

}

while (v.size() > 1) {

int a1 = Integer.parseInt(v.get(0).toString());

int a2 = Integer.parseInt(v.get(1).toString());

if ((a1 + "," + a2).equals(a[a1 - 1][a2 - 1])) {

System.out.print(a1 + "=>");

v.removeElementAt(0);

} else {

v.removeElementAt(0);

v.removeElementAt(0);

st = new StringTokenizer(a[a1-1][a2-1], ",");

Vector vvv=new Vector();

while (st.hasMoreTokens()) {

vvv.add(st.nextToken().toString());

}

for(int i=vvv.size()-1;i>=0;i--)v.insertElementAt(vvv.get(i), 0);

}

}

System.out.println(v.get(0));

}

}

**Minimun spanning tree**

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

public class min\_cost\_spanning\_tree {

static void begin() throws IOException{

BufferedReader br=new BufferedReader(new InputStreamReader(System.in));

int testcases=Integer.parseInt(br.readLine());

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

String node\_edge[]=br.readLine().split(" ");

int st[]=new int[Integer.parseInt(node\_edge[1])];

int end[]=new int[st.length];

int cost[]=new int[st.length];

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

String data[]=br.readLine().split(" ");

st[j]=Integer.parseInt(data[0]);

end[j]=Integer.parseInt(data[1]);

cost[j]=Integer.parseInt(data[2]);

}

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

for(int k=0;k<st.length-j-1;k++){

if(cost[k]>cost[k+1]){

int t=cost[k];

cost[k]=cost[k+1];

cost[k+1]=t;

t=st[k];

st[k]=st[k+1];

st[k+1]=t;

t=end[k];

end[k]=end[k+1];

end[k+1]=t;

}

}

}

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

//for(int j=0;j<st.length;j++){

// System.out.println(st[j]+" "+end[j]+" "+cost[j]);

//}

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

int co=2;

int test[]=new int[Integer.parseInt(node\_edge[0])];

int ans=cost[0];

test[st[0]-1]=test[end[0]-1]=1;

System.out.println(st[0]+" "+end[0]);

for(int j=1;j<st.length;j++){

if(test[st[j]-1]==0 || test[end[j]-1]==0){

int copy;

if(test[st[j]-1]==0)test[st[j]-1]=co;

else{

copy=test[st[j]-1];

for(int k=0;k<test.length;k++){

if(test[k]==copy)test[k]=co;

}

}

if(test[end[j]-1]==0)test[end[j]-1]=co;

else{

copy=test[end[j]-1];

for(int k=0;k<test.length;k++){

if(test[k]==copy)test[k]=co;

}

}

System.out.println(st[j]+" "+end[j]);

co++;

ans+=cost[j];

}

else if(test[st[j]-1]!=test[end[j]-1]){

int start=test[st[j]-1];

int en=test[end[j]-1];

for(int k=0;k<test.length;k++){

if(test[k]==start)test[k]=co;

if(test[k]==en)test[k]=co;

}

System.out.println(st[j]+" "+end[j]);

co++;

ans+=cost[j];

}

//for(int k=0;k<test.length;k++){

// System.out.print(test[k]+" ");

//}

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

//if(co==test.length)break;

}

//System.out.println(ans);

}

}

public static void main(String[] args) throws IOException {

min\_cost\_spanning\_tree a=new min\_cost\_spanning\_tree();

a.begin();

}

}

**PreOrder / InOrder / PostOrder**

import java.util.Vector;

public class TreeOrder {

static int left[] = new int[] {

2,4,6,8,0,0,10,0,0,0

};

static int right[] = new int[] {

3,5,7,0,0,9,0,0,0,0

};

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

static int count = 1;

static void inOrder(int vertex) {

//Left Order

if(left[vertex-1] != 0)

{

inOrder(left[vertex-1]);

}

//Root Order

number[vertex-1] = count;

count++;

//Right Order

if(right[vertex-1] != 0) {

inOrder(right[vertex-1]);

}

}

static void preOrder(int vertex) {

//Root Order

number[vertex-1] = count;

count++;

//Left Order

if(left[vertex-1] != 0)

{

inOrder(left[vertex-1]);

}

//Right Order

if(right[vertex-1] != 0) {

inOrder(right[vertex-1]);

}

}

static void postOrder(int vertex) {

//Left Order

if(left[vertex-1] != 0)

{

inOrder(left[vertex-1]);

}

//Right Order

if(right[vertex-1] != 0) {

inOrder(right[vertex-1]);

}

//Root Order

number[vertex-1] = count;

count++;

}

public static void main(String[] args) {

TreeOrder a = new TreeOrder();

a.inOrder(1);

Vector v = new Vector();

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

v.add(number[i]);

}

for (int i = 1; i <= 10; i++) {

System.out.print((v.indexOf(i)+1)+" ");

}

a.preOrder(1);

v = new Vector();

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

v.add(number[i]);

}

for (int i = 1; i <= 10; i++) {

System.out.print((v.indexOf(i)+1)+" ");

}

a.postOrder(1);

v = new Vector();

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

v.add(number[i]);

}

for (int i = 1; i <= 10; i++) {

System.out.print((v.indexOf(i)+1)+" ");

}

// TreeOrder TreeOrder.inOrder(count);

}

}

**Parallelogram**

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

18public class Parallelogram {

public static void main(String[] args) throws IOException {

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

int testcase = Integer.parseInt(br.readLine());

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

int coordinate = Integer.parseInt(br.readLine());

int x[] = new int[coordinate];

int y[] = new int[coordinate];

for (int j = 0; j < coordinate; j++) {

String data[] = br.readLine().split(" ");

x[j] = Integer.parseInt(data[0]);

y[j] = Integer.parseInt(data[1]);

}

int count = 0;

int xp1 = 0;

int xp2 = 0;

int loop=0;

for (int j = 0; j < coordinate; j++) {

for (int k = j + 1; k < coordinate; k++) {

xp1 = x[j] - x[k];

for (int l = 0; l < coordinate; l++) {

if (l != j && l != k) {

for (int m = 0; m < coordinate; m++) {

loop++;

if (m != j && m != k && m != l) {

if(!((x[j]==x[k]&& x[l]==x[m] && x[j]==x[l]) || (y[j]==y[k]&& y[l]==y[m] && y[j]==y[l]))){

xp2 = x[l] - x[m];

if (xp1 == xp2) {

int yp1 = y[j] - y[k];

int yp2 = y[l] - y[m];

if (yp1 == yp2) {

count++;

//System.out.println(j+" "+k+" "+l+" "+m);

}

}

}

}

}

}

}

}

}

System.out.println(count/4+" "+loop);

}

}

}

**ASCII Table**

**Grid**

package Revision;

public class JavaApplication1 {

static int count=1;

static int a[][]=new int[][]{{1,0,0,0,0},{0,1,1,0,0},{0,0,1,0,1},{1,0,0,0,1},{0,1,0,1,0}};

static int b[][]=new int[a.length][a.length];

static void r(int i,int j){

b[i][j]=count;

if(j>0 && b[i][j-1]<=0 && a[i][j-1]==1) r(i,j-1); //left

if(j<a.length-1 && b[i][j+1]<=0 && a[i][j+1]==1) r(i,j+1); //right

if(i>0 && b[i-1][j]<=0 && a[i-1][j]==1) r(i-1,j);//above

if(i<a.length-1 && b[i+1][j]<=0 && a[i+1][j]==1) r(i+1,j);//bottom

//diagonally

if(i>0 && j>0 && b[i-1][j-1]<=0 && a[i-1][j-1]==1) r(i-1,j-1); //left top

if(i<a.length-1 && j>0 && b[i+1][j-1]<=0 && a[i+1][j-1]==1) r(i+1,j-1); //right top

if(i>0 && j<a.length-1 && b[i-1][j+1]<=0 && a[i-1][j+1]==1) r(i-1,j+1);//left buttom

if(i<a.length-1 && j<a.length-1 && b[i+1][j+1]<=0 && a[i+1][j+1]==1) r(i+1,j+1);//right bottom

}

public static void main(String[] args) {

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

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

if(a[j][i]==1 && b[j][i]<=0) {

r(j,i); count++;

}

}

}

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

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

System.out.print(b[i][j]+"\t");

}

System.out.println("");

} // Output

}

}

**Java Regular Expression**

import java.util.regex.Matcher;

import java.util.regex.Pattern;

public class RegexMatches {

public static void main( String args[] ) {

// String to be scanned to find the pattern.

String line = "This order was placed for QT3000! OK?";

String pattern = "(.\*)(\\d+)(.\*)";

// Create a Pattern object

Pattern r = Pattern.compile(pattern);

// Now create matcher object.

Matcher m = r.matcher(line);

if (m.find( )) {

System.out.println("Found value: " + m.group(0) );

System.out.println("Found value: " + m.group(1) );

System.out.println("Found value: " + m.group(2) );

}else {

System.out.println("NO MATCH");

}

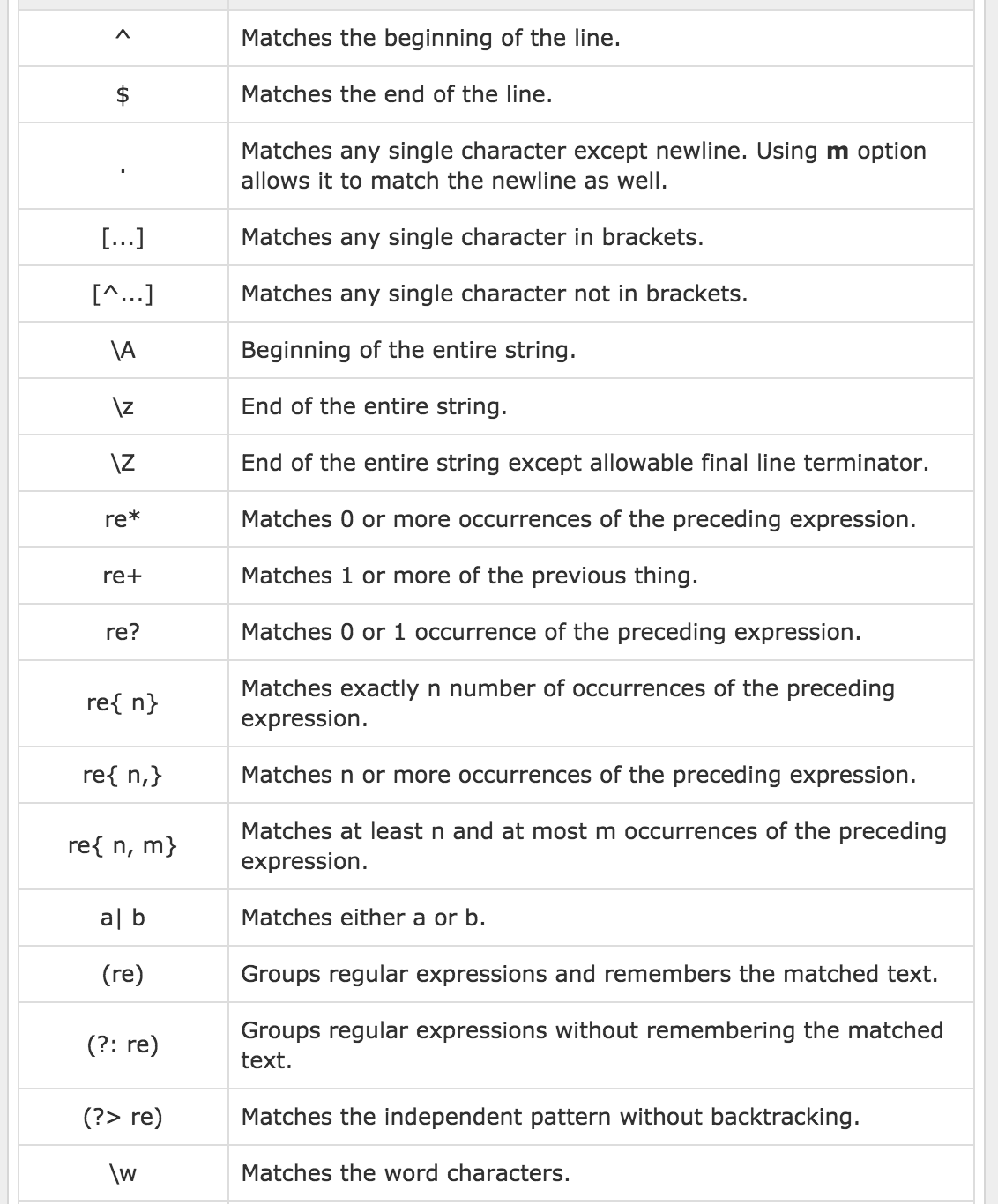
}

}

Found value: This order was placed for QT3000! OK?

Found value: This order was placed for QT300

Found value: 0



public class StringMatcher {

// returns true if the string matches exactly "true"

public boolean isTrue(String s){

return s.matches("true");

}

// returns true if the string matches exactly "true" or "True"

public boolean isTrueVersion2(String s){

return s.matches("[tT]rue");

}

// returns true if the string matches exactly "true" or "True"

// or "yes" or "Yes"

public boolean isTrueOrYes(String s){

return s.matches("[tT]rue|[yY]es");

}

// returns true if the string contains exactly "true"

public boolean containsTrue(String s){

return s.matches(".\*true.\*");

}

// returns true if the string contains of three letters

public boolean isThreeLetters(String s){

return s.matches("[a-zA-Z]{3}");

// simpler from for

// return s.matches("[a-Z][a-Z][a-Z]");

}

// returns true if the string does not have a number at the beginning

public boolean isNoNumberAtBeginning(String s){

return s.matches("^[^\\d].\*");

}

// returns true if the string contains a arbitrary number of characters except b

public boolean isIntersection(String s){

return s.matches("([\\w&&[^b]])\*");

}

// returns true if the string contains a number less then 300

public boolean isLessThenThreeHundred(String s){

return s.matches("[^0-9]\*[12]?[0-9]{1,2}[^0-9]\*");

}

}

Biomedical Engieering

public class Icpc {

public static void main(String[] args) {

String test = "abc";

String[] test2 = {"a", "b", "c", "ab"};

String tmp1 = "";

int min = 0;

int count = 1;

int n = 0;

for (int i = 0; i < 4; i++) {

tmp1 = test2[i];

if (tmp1.equals(test.substring(0, tmp1.length()))) {

n++;

for (int j = 0; j < 3 && tmp1.length() < test.length(); j++) {

String tmp2 = tmp1 + test2[j];

if (tmp2.length() <= test.length()) {

if (tmp2.equals(test.substring(0, tmp2.length()))) {

j = 0;

n++;

tmp1 = tmp2;

} else {

tmp2 = tmp1;

}

}

}

if (tmp1.equals(test)) {

if (count == 1) {

min = n;

count++;

}

if (n > min) {

min = n;

}

} else {

n = 0;

}

}

n = 0;

}

System.out.println("Answer :" + min);

}

}

**UniquePermutation**

##############

import java.util.ArrayList;

import java.util.Arrays;

public class UniquePermutation {

static ArrayList<String> arlist=new ArrayList();

public static void main(String[] args) {

int[] input={1,1,2};

String resultSet="";

generatePermutation(resultSet,input,0);

for(int i=0;i<arlist.size();i++)

System.out.println(arlist.get(i));

System.out.println(arlist.size());

}

String[] store;

int storeCount=0;

private static void generatePermutation(String resultSet,int[] input,int start){

if(start==input.length){

// System.out.println(resultSet);

arlist.add(resultSet);

}

for(int i=start;i<input.length;i++){

if(i!=start&&input[i]==input[i-1])

continue;

swap(input,i,start);

generatePermutation(resultSet+String.valueOf(input[start])+",", input, start+1);

swap(input,i,start);

}

}

public static void swap(int[] arr,int i,int j){

int temp=arr[i];

arr[i]=arr[j];

arr[j]=temp;

}

}

**Non\_UniquePermutation**

##################

import java.util.ArrayList;

import java.util.Arrays;

import java.util.Scanner;

public class Non\_UniquePermutation {

static ArrayList<String> arlist=new ArrayList();

public static void Permute(int[] input, int startindex) {

int size = input.length;

String out="";

if (size == startindex + 1) {

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

// System.out.print(input[i]);

out+=input[i]+",";

}

// System.out.println();

arlist.add(out);

} else {

for (int i = startindex; i < size; i++) {

int temp = input[i];

input[i] = input[startindex];

input[startindex] = temp;

Permute(input, startindex + 1);

}

}

}

public static void main(String[] args) {

int[] input={1,2,2};

Permute(input, 0);

// System.out.println(counter + " number of permutations obtained");

System.out.println(arlist.size());

for(int i=0;i<arlist.size();i++)

System.out.println(arlist.get(i).toString());

}

}

**Combinations**

##########

import java.io.\*;

import java.util.ArrayList;

class Combinations {

static ArrayList<String> arlist=new ArrayList();

static void combinationUtil(int arr[], int data[], int start,

int end, int index, int r) {

String out="";

if (index == r) {

for (int j = 0; j < r; j++) {

//System.out.print(data[j] + ", ");

out+=data[j]+",";

}

arlist.add(out);

return;

}

for (int i = start; i <= end && end - i + 1 >= r - index; i++) {

data[index] = arr[i];

combinationUtil(arr, data, i + 1, end, index + 1, r);

}

}

static void printCombination(int arr[], int n, int r) {

int data[] = new int[r];

combinationUtil(arr, data, 0, n - 1, 0, r);

}

public static void main(String[] args) {

int arr[] = {1, 2, 3, 4, 5, 6};

int r = 4;

int n = arr.length;

printCombination(arr, n, r);

for(int i=0;i<arlist.size();i++)

System.out.println(arlist.get(i));

System.out.println("No of Comb:: " + arlist.size());

}

}

