public class **Permutation**

{

public static void main(String[] args)

{

String str = "ABC";

int n = str.length();

Permutation permutation = new Permutation();

permutation.permute(str, 0, n-1);

}

private void permute(String str, int l, int r)

{

if (l == r)

System.out.println(str);

else

{

for (int i = l; i <= r; i++)

{

str = swap(str,l,i);

permute(str, l+1, r);

str = swap(str,l,i);

}

}

}

public String swap(String a, int i, int j)

{

char temp;

char[] charArray = a.toCharArray();

temp = charArray[i] ;

charArray[i] = charArray[j];

charArray[j] = temp;

return String.valueOf(charArray);

}

}

----------------------------------------------------------------------------

class **Combination** {

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

int end, int index, int r)

{

if (index == r)

{

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

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

System.out.println("");

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);

}

/\*Driver function to check for above function\*/

public static void main (String[] args) {

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

int r = 3;

int n = arr.length;

printCombination(arr, n, r);

}

}

-----------------------------------------------------------------------------

public class **infixToPost** {

static int Prec(char ch) {

switch (ch) {

case '+':

case '-':

return 1;

case '\*':

case '/':

return 2;

case '^':

return 3;

}

return -1;

}

static String inToPost(String exp) {

String result = new String("");

Stack<Character> stack = new Stack<>();

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

char c = exp.charAt(i);

if (Character.isLetterOrDigit(c))

result += c;

else if (c == '(')

stack.push(c);

else if (c == ')') {

while (!stack.isEmpty() && stack.peek() != '(')

result += stack.pop();

if (!stack.isEmpty() && stack.peek() != '(')

return "Invalid Expression"; // invalid expression

else

stack.pop();

}

else {

while (!stack.isEmpty() && Prec(c) <= Prec(stack.peek()))

result += stack.pop();

stack.push(c);

}

}

while (!stack.isEmpty())

result += stack.pop();

return result;

}

public static void main(String[] args) {

String exp = "1+2\*3/4";

String exp1="(84-2)/54+72-9\*100";

System.out.println(inToPost(exp));

System.out.println(inToPost(exp1));

}

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Infix to Pre**

StringBuilder exp1 = new StringBuilder("((90-1)\*11)/3+4\*2");

String exp = exp1.reverse().toString();

char []expArray = exp.toCharArray();

System.out.println(exp);

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

if(expArray[i]=='('){

expArray[i]=')';

}else if(expArray[i]==')'){

expArray[i]='(';

}

}

exp=String.valueOf(expArray);

System.out.println(exp);

exp=inToPost(exp);

System.out.println(exp);

exp1= new StringBuilder(exp);

exp = exp1.reverse().toString();

System.out.println(exp);

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

public class **PostCal** {

static String pC(String exp) {

Stack<String> stack = new Stack<>();

String[] useExp = new String[exp.length()];

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

useExp[i] = Character.toString(exp.charAt(i));

}

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

if (Character.isLetterOrDigit(exp.charAt(i))) {

stack.push(useExp[i]);

}

else {

String s1 = stack.pop();

String s2 = stack.pop();

Double d1 = Double.parseDouble(s1);

Double d2 = Double.parseDouble(s2);

Double d=0.0;

switch (useExp[i]) {

case "+":d = d2 + d1;break;

case "-":d = d2 - d1;break;

case "\*":d = d2 \* d1;break;

case "/":d = d2 / d1;break;

}

stack.push(d+"");

}

}

return stack.pop();

}

public static void main(String[] args) {

String exp = "123\*4/+";

System.out.println(pC(exp));

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

public class **prefixCal** {

static boolean isOperator(String x) {

switch (x) {

case "+":case "-":case "\*":case "/":case "^":

return true;

}

return false;

}

static String prefixToInfix(String exp){

String arr[]= new String[199];

StringTokenizer st=new StringTokenizer(exp,"\_");

int number =0;

while(st.hasMoreTokens()){

arr[number++]=st.nextToken().toString();

}

exp="";

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

exp+=arr[i];

}

System.out.println(exp);

Stack <String> stack = new Stack<>();

for(int i=number-1;i>=0;i--){

if(isOperator(arr[i])){

String s1=stack.pop();

String s2 = stack.pop();

Double n1=Double.parseDouble(s1);

Double n2=Double.parseDouble(s2);

Double ans=0.0;

if(arr[i].equals("+")){

ans=n1+n2;

}else if(arr[i].equals("-")){

ans=n1-n2;

}else if(arr[i].equals("\*")){

ans=n1\*n2;

}else if(arr[i].equals("/")){

ans = n1/n2;

}

stack.push(ans+"");

}else{

stack.push(arr[i]+"");

}

}

return stack.pop();

}

public static void main(String[] args) {

String exp1="+\_/\_\*\_-\_90\_1\_11\_3\_\*\_4\_2";

System.out.println(prefixToInfix(exp1));

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

public class **preToIn** {

static boolean isOperator(String x) {

switch (x) {

case "+":case "-":case "\*":case "/":case "^":

return true;

}

return false;

}

static String prefixToInfix(String exp){

String arr[]= new String[199];

StringTokenizer st=new StringTokenizer(exp,"\_");

int number =0;

while(st.hasMoreTokens()){

arr[number++]=st.nextToken().toString();

}

exp="";

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

exp+=arr[i];

}

System.out.println(exp);

Stack <String> stack = new Stack<>();

for(int i=number-1;i>=0;i--){

if(isOperator(arr[i])){

String s1=stack.pop();

String s2 = stack.pop();

stack.push("("+s1+arr[i]+s2+")");

}else{

stack.push(arr[i]+"");

}

}

return stack.pop();

}

public static void main(String[] args) {

String exp="+\_/\_-\_84\_2\_54\_-\_72\_\*\_9\_100t";

System.out.println(prefixToInfix(exp));

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Marix Multiply

**for** (**int** i = 0; i < aRows; i++) { // aRow

**for** (**int** j = 0; j < bColumns; j++) { // bColumn

**for** (**int** k = 0; k < aColumns; k++) { // aColumn

C[i][j] += A[i][k] \* B[k][j];}}}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Expression evaluation and format

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

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

ScriptEngineManager factory = new ScriptEngineManager();

// create a JavaScript engine

ScriptEngine engine = factory.getEngineByName("JavaScript");

System.out.println(engine.eval(br.readLine()) + "");

NumberFormat formatter = new DecimalFormat("#0.0000");

System.out.println(formatter.format(4));

}

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

//BIOnediacl

public class BIOmedical {

public static void main(String[] args){

int k=1;

for(int kk=0;kk<k;kk++){

int m = 4;

String s = "acgaagaacga";

String[] ss = {"acg","gaa","ac","cga"};

int[] a = new int[1001];

a[0]=1;

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

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

if (s.startsWith(ss[j],i)) {

if (a[i]!=0) {

a[i+ss[j].length()]=a[i+ss[j].length()]==0?

(a[i]+1):(Math.min(a[i+ss[j].length()], a[i+1]));

}

}

}

System.out.println("Data Set:"+(kk+1)+":");

int ans;

if ((ans=a[s.length()]-1)>0) {

System.out.println(ans);

}else{

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

}

}

}

}

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

//OrderTest

public class OrderTest {

static int I[] = new int[]{2,4,6,8,0,0,10,0,0,0};

static int r[] = new int[]{3,5,7,0,0,9,0,0,0,0};

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

static int count = 1;

static void in(int v){

if(I[v-1]!=0)

in(I[v-1]);

number[v-1]= count;

count++;

if(r[v-1]!=0)

in(r[v-1]);

}

static void pre(int v){

number[v-1]=count;

count++;

if(I[v-1]!=0)

pre(I[v-1]);

if(r[v-1]!=0)

pre(r[v-1]);

}

static void post(int v){

if(I[v-1]!=0)

post(I[v-1]);

if(r[v-1]!=0)

post(r[v-1]);

number[v-1]=count;

count++;

}

public static void main(String[] args){

OrderTest a = new OrderTest();

a.post(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)+" ");

}

}

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

//Shortpath

**public** **class** ShortestPath {

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

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

**int** testCase = Integer.*parseInt*(br.readLine());

StringTokenizer st;

**for** (**int** loop = 0; loop < testCase; loop++) {

st = **new** StringTokenizer(br.readLine(), " ");

**int** numberOfNode = Integer.*parseInt*(st.nextToken());

**int** numberOfEdge = Integer.*parseInt*(st.nextToken());

**int** numberOfQuestion = Integer.*parseInt*(st.nextToken());

**int**[][] array = **new** **int**[numberOfNode][numberOfNode];

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

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

array[i][j] = 500000;

}

}

String[][] bArray = **new** String[numberOfEdge][numberOfEdge];

**for** (**int** i = 0; i < numberOfEdge; i++) {

st = **new** StringTokenizer(br.readLine(), " ");

**int** s = Integer.*parseInt*(st.nextToken());

**int** e = Integer.*parseInt*(st.nextToken());

**int** data = Integer.*parseInt*(st.nextToken());

array[s - 1][e - 1] = data;

array[e - 1][s - 1] = data;

}

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

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

**if** (array[i][j] < 200000) {

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

}

}

}

System.***out***.println();

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

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

**if** (i != j) {

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

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

**if** (array[i][k] + array[k][j] < array[i][j]) {

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

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

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

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

}

}

}

}

}

}

**for** (**int** i = 0; i < numberOfQuestion; i++) {

st = **new** StringTokenizer(br.readLine(), " ");

**int** s = Integer.*parseInt*(st.nextToken());

**int** e = Integer.*parseInt*(st.nextToken());

System.***out***.println(array[s-1][e-1]);

*myFun*(bArray, s, e);

}

}

}

**public** **static** **void** myFun(String[][] data, **int** s1, **int** e1) {

**int** s = s1 - 1;

**int** e = e1 - 1;

**if** (s == e) {

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

} **else** {

Vector ans = **new** Vector<>();

Vector v = **new** Vector();

StringTokenizer st = **new** StringTokenizer(data[s][e], ",");

**while** (st.hasMoreTokens()) {

v.add(st.nextToken());

}

**while** (v.size() > 1) {

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

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

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

ans.add(v.get(0));

v.removeElementAt(0);

} **else** {

v.removeElementAt(0);

v.removeElementAt(0);

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

Vector vv = **new** Vector<>();

**while** (st.hasMoreTokens()) {

vv.add(st.nextToken());

}

**for** (**int** i = vv.size() - 1; i >= 0; i--) {

v.insertElementAt(vv.get(i), 0);

}

}

}

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

System.***out***.print(ans.get(i)+"=>");

}

System.***out***.print(e1);

}

}

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Shortest2

public static int[][] shortest(int g[][])

{

int [][]f=g;

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

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

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

{

if(f[i][j]<f[i][k]+f[k][j]){

g[i][j]=f[i][j];

g[j][i]=f[i][j];

}else{

g[i][j]=f[i][k]+f[k][j];

g[j][i]=f[i][k]+f[k][j];

}

}

}

f=g;

}

return g;

}

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

// TODO code application logic here

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

int n=Integer.parseInt(br.readLine().toString());

for(int root=0;root<n;root++){

StringTokenizer st=new StringTokenizer(br.readLine());

int v=Integer.parseInt(st.nextToken());

int e=Integer.parseInt(st.nextToken());

int c=Integer.parseInt(st.nextToken());

int g[][]=new int[v][v];

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

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

g[i][j]=10000;

}

}

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

st=new StringTokenizer(br.readLine());

int source=Integer.parseInt(st.nextToken());

int dest=Integer.parseInt(st.nextToken());

int value=Integer.parseInt(st.nextToken());

g[source-1][dest-1]=value;

g[dest-1][source-1]=value;

}

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

g[i][i]=0;

}

int a[][]=shortest(g);

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

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

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

}

System.out.println();

}

System.out.println("Case "+(root+1)+":");

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

st=new StringTokenizer(br.readLine());

int source=Integer.parseInt(st.nextToken());

int dest=Integer.parseInt(st.nextToken());

System.out.println(a[source-1][dest-1]);}}}}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

private static long gcd(long a, long b) {

while (b > 0) {

long temp = b;

b = a % b; // % is remainder

a = temp;

}

return a;

}

private static long lcm(long a, long b) {

return a \* (b / gcd(a, b));

}

private static void fibo(int data) {

int f = 1, s = 2, t = 0;

System.out.println(f + "\n" + s);

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

t = f + s;

System.out.println(t);

f = s;

s = t;

}

}

private static void primeFactor() {

int n = 6, d = 2;

while (n > 1)

if (n % d == 0) {

if (n == d) {

System.out.println(n);

}

else {

System.out.println(d + "\*");

}

n = n / d;

}

else

d++;

}

private static void Prime(int data) {

int d = 2;

while (data % d != 0) {

d++;

}

if (d == data)

System.out.println("Prime");

else

System.out.println("Not Prime");

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

public class **Castle**{

static int count = 1;

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

static int[][] b = new int[4][5];

static int row = 4, col = 5;

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

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

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

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

castle(i, j);

count++;

}

}

}

}

static void castle(int i, int j) {

b[i][j] = count;

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

castle(i, j - 1);// left

if (j < col - 1 && b[i][j + 1] <= 0 && a[i][j + 1] == 1)

castle(i, j + 1);// right

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

castle(i - 1, j);// above

if (i < row - 1 && b[i + 1][j] <= 0 && a[i + 1][j] == 1)

castle(i + 1, j);// bottom

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

castle(i - 1, j - 1);

if (i > 0 && j < col - 1 && b[i - 1][j + 1] <= 0 && a[i - 1][j + 1] == 1)

castle(i - 1, j + 1);

if (i < row - 1 && j > 0 && b[i + 1][j - 1] <= 0 && a[i + 1][j - 1] == 1)

castle(i + 1, j - 1);

if (i < row - 1 && j < col - 1 && b[i + 1][j + 1] <= 0 && a[i + 1][j + 1] == 1)

castle(i + 1, j + 1);

}

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Arrays.sort(a,**new** Comparator<**int**[]>(){

**public** **int** compare(**int** []x,**int** []y){

**if**(Integer.compare(x[1],y[1])){

**return** Integer.compare(x[2],y[2]);

}

**return** Integer.compare(x[1],y[1]);

}

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

private static void order(List<Person> persons) {

Collections.sort(persons, new Comparator() {

public int compare(Object o1, Object o2) {

String x1 = ((Person) o1).getName();

String x2 = ((Person) o2).getName();

int sComp = x1.compareTo(x2);

if (sComp != 0) {

return sComp;

}

Integer x1 = ((Person) o1).getAge();

Integer x2 = ((Person) o2).getAge();

return x1.compareTo(x2);

}

});

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**public** **class** Coin {

**static** **int** count( **int** S[], **int** m, **int** n )

{

// If n is 0 then there is 1 solution

// (do not include any coin)

**if** (n == 0)

**return** 1;

// If n is less than 0 then no

// solution exists

**if** (n < 0)

**return** 0;

// If there are no coins and n

// is greater than 0, then no

// solution exist

**if** (m <=0 && n >= 1)

**return** 0;

// count is sum of solutions (i)

// including S[m-1] (ii) excluding S[m-1]

**return** *count*( S, m - 1, n ) +

*count*( S, m, n-S[m-1] );

}

// Driver program to test above function

**public** **static** **void** main(String[] args)

{

**int** arr[] = {50, 100, 200};

**int** m = arr.length;

System.***out***.println( *count*(arr, m, 500));

}

}