CHUCK

```
import java.io.File;
import java.io.IOException;
import java.util.Scanner;

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

        Scanner s = new Scanner(new File("chuck.dat"));
        int numSets = s.nextInt();
        s.nextLine();

        for(int i = 1; i <= numSets; i++){
            int num = s.nextInt();
            assert num >= 0 && num <= 2000000000 : "num may not be negative";
            s.nextLine();
            int count = 0;
            for(int base = 2; base <= 16; base++) {
                String newNum = Integer.toString(num, base);
                // System.out.println(newNum);
                String revNewNum = new StringBuffer(newNum).reverse().toString();
                if(newNum.equals(revNewNum))
                    count++;
            }
            System.out.println(count);
        }
        s.close();
    }
}
```

COUNT

```
import java.io.*;
import java.util.*;
/**
* author: Moises Holguin
* date: Monday, November 26, 2012
* UTCS Problem Counts Them Up
*/
public class Count {
    public static void main(String[] args) throws IOException {
        Scanner s = new Scanner(new File("count.dat"));

        int numSets = Integer.parseInt(s.nextLine());

        for(int i = 0; i < numSets; i++) {
            String line = s.nextLine();
            int num1 = 0;
            int num2 = 0;

            //Handles the permutations
            if (line.contains("P")) {
                num1 = Integer.parseInt(line.substring(0, line.indexOf("P")));
                num2 = Integer.parseInt(line.substring(line.indexOf("P") + 1));

                assert 1 <= num1 && num1 <= 60 && 0 <= num2 && num2 <= 60;

                long answer = factorial(num1, num1 - num2);

                //This is how a person would normally tackle the problem
                //long answer = factorial(num1)/factorial(num1-num2);

                // BUT, that leads to overflow in many cases

                System.out.println(answer);
            }
            //Handles the combinations
            else {
                num1 = Integer.parseInt(line.substring(0, line.indexOf("C")));
                num2 = Integer.parseInt(line.substring(line.indexOf("C") + 1));
                long answer = factorial(num1, Math.max(num1 - num2,
num2))/factorial(Math.min(num1 - num2, num2));

                //This is how a person would normally tackle the problem
                //long answer = factorial(num1)/(factorial(num1-
num2)*factorial(num2));

                // BUT, that leads to overflow in many cases

                System.out.println(answer);
            }
        }
    }

    public static long factorial(int x) {
        if(x==0)
```

```
            return 1;
        else
            return x * factorial(x-1);
    }

    // avoid overflow
    public static long factorial(int x, int y) {
        if(x==0 || x==y)
            return 1;
        else
            return x * factorial(x-1, y);
    }
}
```

CUBES

```
import java.io.File;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.Scanner;

public class Cubes {

    public static Scanner key = new Scanner(System.in);

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

        Scanner s = new Scanner(new File("cubes.dat"));
        int numSets = s.nextInt();
        s.nextLine();

        // Stopwatch st = new Stopwatch();

        for(int i = 1; i <= numSets; i++){
            String[] numsAsStrings = s.nextLine().trim().split("\\s+");

            int[] energies = new int[numsAsStrings.length];
            for(int j = 0; j < numsAsStrings.length; j++)
                energies[j] = Integer.parseInt(numsAsStrings[j]);
            Arrays.sort(energies);
            // st.start();
            System.out.println(minBoxes(energies));
            // st.stop();
            // System.out.println(st);
        }
        s.close();
    }

    private static int minBoxes(int[] energies) {
        int boxes = 1;
        boolean solved = false;
        while(!solved) {
            solved = canWork(energies, boxes);
            // System.out.println(boxes);
            if(!solved) {
                boxes++;
                // System.out.println(boxes);
            }
        }
        return boxes;
    }

    private static boolean canWork(int[] energies, int numBoxes) {
        List<Integer>[] boxes = (List<Integer>[]) (new List[numBoxes]);
        for(int i = 0; i < boxes.length; i++)
            boxes[i] = new ArrayList<Integer>();

        return canWork(0, energies, boxes);
```

```
    }

    private static boolean canWork(int index, int[] energies,
                                                List<Integer>[] boxes) {
        if(index == energies.length) {
            // System.out.println(Arrays.toString(boxes));
            return true;
        }


        for(int b = 0; b < boxes.length; b++) {

            boxes[b].add(energies[index]);

            // System.out.println(Arrays.toString(boxes));
            // key.nextLine();

            if(noBadTriples(boxes) && canWork(index + 1, energies, boxes)) {
                return true;
            }
            boxes[b].remove(boxes[b].size() - 1);
        }
        return false;
    }

    private static boolean noBadTriples(List<Integer>[] boxes) {
        for(List<Integer> box : boxes) {
            if(badBox(box))
                return false;
        }
        return true;
    }

    private static boolean badBox(List<Integer> box) {
        for(int i = 0; i < box.size(); i++)
            for(int j = i + 1; j < box.size(); j++)
                for(int k = j + 1; k < box.size(); k++)
                    if(badTriple(box.get(i), box.get(j), box.get(k)))
                        return true;
        return false;
    }

    private static boolean badTriple(int x, int y,int z) {
        return x + y == z || x + z == y || y + z == x;
    }


}
```

DIMENSIONS

```
import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;
import java.util.regex.Matcher;
import java.util.regex.Pattern;


public class Dimensions {

    private static String INT_OR_DOUBLE
                      = "[+]?[0-9]*\\.?[0-9]+([eE][+]?[0-9]+)?";

    public static void main(String args[]) throws FileNotFoundException {
        Scanner scan = new Scanner(new File("dimensions.dat"));
        int numTests = scan.nextInt();

        scan.nextLine();
        for (int test = 0; test < numTests; test++) {
            String dimensions = getDimensions(scan.nextLine());
            if (dimensions == null)
                System.out.println("NO DIMENSION FOUND");
            else System.out.println(dimensions);
        }
    }

    public static String getDimensions(String text) {
        Pattern p = Pattern.compile(INT_OR_DOUBLE + "x"
                            + INT_OR_DOUBLE + "x" + INT_OR_DOUBLE);
        Matcher m
      = p.matcher(text.replaceAll(" ", "").replaceAll("\t", "").toLowerCase());

        if (m.find())
            return m.group();
        return null;
    }
}
```

GOAL

```
import java.io.File;
import java.io.IOException;
import java.util.Scanner;

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

        Scanner s = new Scanner(new File("goals.dat"));
        int numSets = s.nextInt();

        for(int i = 1; i <= numSets; i++) {
            int numRides = s.nextInt();
            s.nextLine();
            // System.out.println(numRides);
            String[] rides = {};
            if(numRides > 0)
                rides = s.nextLine().split("\\s+");
            assert rides.length == numRides;
            int goal1 = s.nextInt();
            int goal2 = s.nextInt();
            s.nextLine();
            // System.out.println(goal1 + " " + goal2);
            int total = 0;
            boolean goal1Met = false;
            boolean goal2Met = false;
            for(int j = 0; j < rides.length; j++) {
                total += Integer.parseInt(rides[j]);
                if(!goal1Met && total >= goal1) {
                    goal1Met = true;
                    System.out.print((j + 1) + " ");
                }
                if(!goal2Met && total >= goal2) {
                    goal2Met = true;
                    System.out.print((j + 1) + " ");
                }
            }
            if(!goal1Met)
                System.out.print("FAIL ");
            if(!goal2Met)
                System.out.print("FAIL ");
            if(numRides != 0)
                System.out.println(total / numRides);
            else
                System.out.println(0);
        }

        s.close();
    }

}
```

GRADE

```
import java.io.File;
import java.io.IOException;
import java.util.Scanner;

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

        Scanner s = new Scanner(new File("grade.dat"));
        int numSets = s.nextInt();
        s.nextLine();

        for(int i = 1; i <= numSets; i++){
            System.out.println(s.nextLine());
            double hwp = s.nextInt() / 100.0;
            double qp = s.nextInt() / 100.0;
            double tp = s.nextInt() / 100.0;
            s.nextLine();
            int[] hw = stringsToInt(s);
            int[] qs = stringsToInt(s);
            assert hw != null & qs != null;
            int hwt = 0;
            for(int x : hw) {
                assert x >= 0;
                hwt += x;
            }
            int qt = 0;
            for(int x : qs) {
                assert x >= 0;
                qt += x;
            }
            double hwa = hwt * hwp / hw.length;
            double qa = qt * qp / qs.length;
            // System.out.println(hwa + " " + qa);
            int aNeed = (int) Math.ceil((90 - hwa - qa) / tp);
            int bNeed = (int) Math.ceil((80 - hwa - qa) / tp);
            int cNeed = (int) Math.ceil((70 - hwa - qa) / tp);
            System.out.println("A\t" + (aNeed > 100 ? "SORRY"
                                                   : aNeed < 0 ? 0: aNeed));
            System.out.println("B\t" + (bNeed > 100 ? "SORRY"
                                                   : bNeed < 0 ? 0: bNeed));
            System.out.println("C\t" + (cNeed > 100 ? "SORRY"
                                                   : cNeed < 0 ? 0: cNeed));
        }
        s.close();
    }

    public static int[] stringsToInt(Scanner s) {
        String[] strings = s.nextLine().trim().split("\\s+");
        int[] scores = new int[strings.length];
        for(int k = 0; k < strings.length; k++)
            scores[k] = Integer.parseInt(strings[k]);
        return scores;
    }
}
```

MAJORITY

```
import java.io.File;
import java.io.FileNotFoundException;
import java.util.HashMap;
import java.util.Map;
import java.util.Scanner;


public class Majority {
      public static void main(String args[]) throws FileNotFoundException {
            Scanner scan = new Scanner(new File("majority.dat"));
            int runs = scan.nextInt();
            scan.nextLine();

            for (int i = 0; i < runs; i++) {
                  String votes = scan.nextLine();
                  // printMajority(votes);
                  char c = possibleMajority(votes);
                  System.out.println(verifyMajority(votes, c));
            }
      }

      // alternate solution using map
      public static void printMajority(String votes) {
          Map<Character, Integer> voteCount = new HashMap<Character, Integer>();
          for(int i = 0; i < votes.length(); i++) {
              char ch = votes.charAt(i);
              if(voteCount.containsKey(ch))
                  voteCount.put(ch, voteCount.get(ch) + 1);
              else
                  voteCount.put(ch, 1);
          }
          int max = -1;
          char maxVote = ' ';
          for(char candidate : voteCount.keySet())
              if(voteCount.get(candidate) > max) {
                  max = voteCount.get(candidate) ;
                  maxVote = candidate;
              }
          // System.out.println(voteCount);
          if(1.0 * max / votes.length() > 0.5)
              System.out.println(maxVote);
          else
              System.out.println("NO MAJORITY");
      }

      public static char possibleMajority(String votes) {
            char majority = votes.charAt(0);
            int k = 1;

            for (int i = 1; i < votes.length(); i++) {
                  if (k == 0)
                        majority = votes.charAt(i);
                  if (votes.charAt(i) == majority)
                        k++;
```

```
                else k--;
            }
            return majority;
        }

        public static String verifyMajority(String votes, char c) {
            int instances = 0;
            for (char b : votes.toCharArray())
                if (b == c)
                    instances++;
            double percentage = (instances * 1.0) / votes.length();
            if (percentage > .5)
                return "" + c;
            else return "NO MAJORITY";
        }
}
```

OPTIMAL

```
import java.io.File;
import java.io.IOException;
import java.math.BigInteger;
import java.util.ArrayList;
import java.util.Scanner;
import java.util.TreeSet;

public class Optimal {
    public static void main(String args[]) throws IOException {
        Scanner scan = new Scanner(new File("optimal.dat"));

        int runs = scan.nextInt();

        // System.out.println(runs);
        for (int i = 0; i < runs; i++) {
            int numSequences = scan.nextInt();
            // System.out.println(numSequences);
            ArrayList<ArrayList<Integer>> sequences
                                = new ArrayList<ArrayList<Integer>>();


            // Builds sequences of primes
            for (int j = 0; j < numSequences; j++) {
                sequences.add(new ArrayList<Integer>());
                int lowerBound = scan.nextInt(), upperBound = scan.nextInt();
                // System.out.println(lowerBound + " " + upperBound);
                assert lowerBound >= 2;
                if (lowerBound <= 2)
                    sequences.get(j).add(2);
                if (lowerBound <= 2)
                    lowerBound = 3;
                if (lowerBound % 2 == 0)
                    lowerBound++;

                for (; lowerBound <= upperBound; lowerBound += 2)
                    if (isPrime(lowerBound))
                        sequences.get(j).add(lowerBound);
            }

            // Creates all permutations
            ArrayList<String> permutations = new ArrayList<String>();
            getPermutations(sequences, permutations, new String(), 0);

//            if(numSequences == 9) {
//                System.out.println(permutations);
//                System.out.println(permutations.size());
//            }
            // Stores and sorts all found primes
            TreeSet<Integer> primes = new TreeSet<Integer>();
            for (int j = permutations.size() - 1; j >= 0; j--) {
                int k = Integer.parseInt(permutations.get(j));
                if (isPrime(k))
                    primes.add(k);
            }
```

```
            // System.out.println(primes);
            if (primes.size() != 0)
                System.out.println(primes.size());
            else
                System.out.println("NO PRIMES");
        }
    }

    public static void getPermutations(ArrayList<ArrayList<Integer>> sequences,
            ArrayList<String> permutations, String s,
            int index) {
        // System.out.println(s);
        if (index >= sequences.size()) {
            if(s.length() > 0)
                permutations.add(s);
            return;
        }

        if (sequences.get(index).isEmpty())
            getPermutations(sequences, permutations, s, index + 1);
        else for (Integer i : sequences.get(index))
            getPermutations(sequences, permutations, s + i, index + 1);
    }

    // disregards all even numbers, and numbers less than 2
//    public static boolean isPrime(int n) {
//        if(n == 2)
//            return true;
//        else if(n % 2 == 0)
//            return false;
//        int limit = (int)(Math.sqrt(n) + 1);
//        for (int i = 3; i <= limit; i += 2)
//            if (n % i == 0)
//                return false;
//        return true;
//    }

    public static boolean isPrime(int n) {
        BigInteger num = new BigInteger(n + "");
        return num.isProbablePrime(1000);
    }
}
```

REBATE

```
import java.io.*;
import java.util.*;
import java.text.*;
/**
 * author: Moises Holguin
 * date: Monday, November 26, 2012
 * UTCS Problem
 *
 * Modified by Mike Scott
 */
public class Rebate {
    public static void main(String[] args) throws IOException {
        Scanner infile = new Scanner(new File("rebate.dat"));
        infile.nextLine();

        while(infile.hasNext()) {

            String name = infile.nextLine();
            int num = Integer.parseInt(infile.nextLine());
            double total = 0;

            while(num-->0)
                total+=Double.parseDouble(infile.nextLine());

            if(total < 0)
                System.out.println(name + " BAD DATA");
            else {
                int rebate = (int) (Math.ceil(total*.1));
                int fives = rebate / 5;
                if(rebate % 5 > 0)
                    fives++;
                rebate = fives * 5;
                // System.out.println(total);
                System.out.println(name + " " + rebate);
            }
        }
    }
}
```

ROYAL

```
public class Royal {
    public static void main(String[] args) {
        goalPost();
        System.out.println("**********");
        for(int i = 10; i <= 40; i += 10)
            drawLines(i + "");

        drawCenter();

        for(int i = 30; i >= 10; i -= 10)
            drawLines(i + "");

        drawLines("**");
        goalPost();

    }

    public static void drawCenter() {
        System.out.println("*.\\....\\.*");
        System.out.println("*.\\\"DKR\".*");
        System.out.println("****50****");
        System.out.println("*.\\\"DKR\".*");
        System.out.println("*.\\....\\.*");
        System.out.println("****40****");
    }

    public static void drawLines(String insert) {
        assert insert.length() == 2;
        System.out.println("*.\\....\\.*");
        System.out.println("*.\\....\\.*");
        System.out.println("****" + insert + "****");
    }

    public static void goalPost() {
        System.out.println("..#....#..");
        System.out.println("..#....#..");
        System.out.println("..######..");
        System.out.println("....#.....");
        System.out.println("....#.....");
    }
}
```

TUNNELS

```
import java.io.File;
import java.io.IOException;
import java.util.HashMap;
import java.util.LinkedList;
import java.util.Map;
import java.util.Queue;
import java.util.Scanner;

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

        Scanner s = new Scanner(new File("tunnels.dat"));
        int numSets = s.nextInt();
        s.nextLine();

        for(int i = 1; i <= numSets; i++){
            Graph g = new Graph(s.nextLine().trim());
            System.out.println(g.parts());
        }
        s.close();
    }

    private static final class Graph {
        boolean[][] adjMatrix;

        public Graph(String s) {
            Map<String, Integer> rows = new HashMap<String, Integer>();
            // find the end of the list
            int indexEnd = s.indexOf("},");
            String list = indexEnd >= 0 ? s.substring(1, indexEnd) : s;
            String pairs = indexEnd > 0 ? s.substring(indexEnd + 2) : "";

            String[] buildings = list.trim().split(",");
            int bID = 0;
            for(String b : buildings) {
                assert !rows.containsKey(b);
                rows.put(b, bID++);
            }

            adjMatrix = new boolean[rows.size()][rows.size()];

            pairs = pairs.replace("}", "").replace("{", "");
            String[] buildPairs = pairs.trim().split(",");
            // System.out.println(Arrays.toString(buildPairs));
            assert buildPairs.length % 2 == 0
                      || (buildPairs.length == 1
                      && buildPairs[0].equals(""))
                      : "number of buildings in pairs not even";

            if(buildPairs.length == 1)
                buildPairs = new String[0];

            for(int i = 0; i < buildPairs.length; i += 2) {
                String b1 = buildPairs[i];
```

```
            String b2 = buildPairs[i + 1];
            assert rows.containsKey(b1)
                  && rows.containsKey(b2)
                  : "building in pairs, not in list!!!";
            adjMatrix[rows.get(b1)][rows.get(b2)] = true;
            adjMatrix[rows.get(b2)][rows.get(b1)] = true;
        }
    }

    public int parts() {
        int count = 0;
        boolean[] used = new boolean[adjMatrix.length];
        for(int i = 0; i < used.length; i++) {
            if(!used[i]) {
                count++;
                used[i] = true;
                Queue<Integer> q = new LinkedList<Integer>();
                q.add(i);
                while(!q.isEmpty()) {
                    int row = q.remove();
                    for(int col = 0; col < adjMatrix[0].length; col++) {
                        if(adjMatrix[row][col] && !used[col]) {
                            q.add(col);
                            used[col] = true;
                        }
                    }
                }
            }
        }
        return count;
    }
  }
}
```

US

```
import java.io.File;
import java.io.IOException;
import java.util.Scanner;

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


        // int[] nums = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13,
                                                    14, 15, 16, 17, 18, 19, 20};
        int[] nums = {100, 200, 300, 400, 500, 1000, 2000, 10000, 20000};

        int count = 0;
        int num = 1;
        int index = 0;
        while(index < nums.length) {
            if(isSelfNumber(num)) {
                count++;
                if(count == nums[index]) {
                    System.out.println(nums[index] + " " + num);
                    index++;
                }
            }
            num++;
        }
    }

    private static boolean isSelfNumber(int limit) {
        for(int i = 1; i < limit; i++) {
            if(sums(limit, i))
                return true;
        }
        return false;
    }

    private static boolean sums(int limit, int i) {
        int digits = 0;
        int orgNum = i;
        while(i > 0) {
            digits += i % 10;
            i /= 10;
        }
        // System.out.println(digits + " " + orgNum + " " + limit);
        return digits + orgNum == limit;
    }


}
```