UTCS UIL Contest 2014 - Possible Solutions

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

public class Collatz {

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

        Scanner s = new Scanner(new File("collatz.dat"));
        while(s.hasNextInt()) {
            long org = s.nextInt();
            long n = org;
            int steps = 0;
            while(n != 1) {
                if(n <= 0)
                    throw new RuntimeException("n <= 0 for starting value:
" + org);
                assert n >= 0 : "UH OVERFLOWED: " + org;
                steps++;
                if(n % 2 == 0)
                    n /= 2;
                else {
                    n = n * 3 + 1;
                }
            }
            System.out.println(steps);
        }
        s.close();
    }
}

import java.io.*;
import java.util.*;

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

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

        for(int i = 1; i <= numSets; i++){
            System.out.print("DATA SET " + i + " ");
            int numCrops = s.nextInt();
            s.nextLine();
            double max = -1;
            String best = "";
            for(int j = 0; j < numCrops; j++) {
                String crop = s.nextLine().trim();
```

```
                int hours = s.nextInt();
                int minutes = s.nextInt();
                int priceFor10 = s.nextInt();
                s.nextLine();
                minutes += hours * 60;
                double goldPerMinutePerPlot = 1.0 * priceFor10 / minutes /
5; // 10 units requires 5 plots
                goldPerMinutePerPlot = ((int) (goldPerMinutePerPlot *
100)) / 100.0;
                if(goldPerMinutePerPlot > max) {
                    max = goldPerMinutePerPlot;
                    best = crop;
                }
            }
            System.out.print(best + " ");
            System.out.printf("%4.2f", max);
            System.out.println();
        }
        s.close();
    }


}
```

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

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

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

        for(int i = 1; i <= numSets; i++){
            int stations = s.nextInt();
            double gallons = s.nextDouble();
            double mpg = s.nextDouble();
            double wearPerMile = s.nextDouble();

            double minCost = Double.MAX_VALUE;
            int index = -1;

            for(int j = 1; j <= stations; j++) {

                double distance = s.nextDouble() * 2;
                double price = s.nextDouble();
                double cost = price * gallons + distance / mpg * price;
                cost += distance * wearPerMile;
                if(cost < minCost) {
                    minCost = cost;
                    index = j;
                }
                // System.out.println(j + " " + cost);
            }
            // System.out.println(minCost);
            System.out.print(i + ": " + index + " ");
            System.out.printf("%.2f", minCost);
            System.out.println();
        }
        s.close();
    }


}
```

```
import java.io.*;
import java.util.*;

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

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

        for(int i = 1; i <= numSets; i++){
            String[] clubsAsStrings = s.nextLine().trim().split("\\s+");
            int[] clubs = new int[clubsAsStrings.length];
            for(int j = 0; j < clubs.length; j++)
                clubs[j] = Integer.parseInt(clubsAsStrings[j]);
            int distance = s.nextInt();
            s.nextLine();
            int result = minHits(distance, clubs);
            if(result == -1)
                System.out.println("NOT POSSIBLE");
            else
                System.out.println(result);
        }
        s.close();
    }

    public static int minHits(int distance, int[] clubs) {
        // base case, 0 hits required to hit ball 0 distance
        if(distance == 0)
            return 0;
        // if distance is negative can't do it
        else if(distance < 0)
            return -1;
        else{
            int best = distance + 1; // best has to be better than this if
possible
            // try all the clubs
            for(int i = 0; i < clubs.length; i++) {
                int current = 1 + minHits(distance - clubs[i], clubs);
                if(current != 0 && current < best)
                    best = current;

            }
            if(best == distance + 1)
                // never found an answer!
                best = -1;
            return best;
        }
    }



}
```

```
public class Levels {

    static final int LINE_LENGTH = 64;

    public static void main(String[] args){
        String[] levels =
        {       "OVERRIDING.HASHCODE().AND.NOT.OVERRIDING.EQUALS()",
                "NO.TRUE.REFERENCE.PARAMETERS",
                "ArrayIndexOutOfBoundsException",
                "THE.GREAT.PRIMITIVE-OBJECT.SCHISM",
                "OVERLOADING.INSTEAD.OF.OVERRIDING.EQUALS()",
                "LENGTH()..LENGTH.OR..SIZE().WOE.IS.US",
                "==.AND..EQUALS()-TEARS.UNNUMBERED",
                "NullPointerException.OF.DOOM",
                "ClassCastException"
        };
        for(int j = 0; j < levels.length; j++)
            drawLevel(j, levels[j]);
    }

    private static void drawLevel(int level, String string) {
        int periods = level * 2;
        int stars = LINE_LENGTH - periods * 2;
        printTopAndBotton(periods, stars);
        printMiddle(periods, stars, string);
        printTopAndBotton(periods, stars);

    }

    private static void printMiddle(int periods, int stars, String string)
{
        printString(".", periods);
        printString("*", 1);
        int innerPeriods = stars - 2 - string.length();
        int startPeriods = innerPeriods / 2;
        int endPeriods = innerPeriods / 2 + innerPeriods % 2;

        printString(".", startPeriods);
        printString(string, 1);
        printString(".", endPeriods);
        printString("*", 1);
        printString(".", periods);

        System.out.println();
    }

    private static void printTopAndBotton(int periods, int stars) {
        printString(".", periods);
        printString("*", stars);
        printString(".", periods);
        System.out.println();
    }

    private static void printString(String s, int reps) {
```

```
        for(int i = 0; i < reps; i++)
            System.out.print(s);
    }


}
```

```
import java.io.*;
import java.util.*;

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

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

        for(int i = 1; i <= numSets; i++){
            double[][] distances = new double[s.nextInt()][s.nextInt()];
            s.nextLine();
            String[] temp = s.nextLine().trim().split("\\s+");
            assert temp.length % 2 == 0 : "NOT EVEN NUMBER OF
COORDINATES";
            int[] coordinates = new int[temp.length];
            for(int j = 0; j < temp.length; j++)
                coordinates[j] = Integer.parseInt(temp[j]);

            calcDistances(distances, coordinates);
            // showDistances(distances);
            showMin(distances);

        }
        s.close();
    }

    private static void showDistances(double[][] distances) {
        for(int r = 0; r < distances.length; r++) {
            for(int c = 0; c < distances[0].length; c++) {
                System.out.print(distances[r][c] + " ");
            }
            System.out.println();
        }
        System.out.println();
    }

    private static void showMin(double[][] distances) {
        double minD = Double.MAX_VALUE;
        int rr = -1;
        int cr = -1;
        double diff = 0;
        for(int r = 0; r < distances.length; r++)
            for(int c = 0; c < distances[0].length; c++)
                if(distances[r][c] < minD) {
                    diff = distances[r][c] - minD;
                    minD = distances[r][c];
                    rr = r;
                    cr = c;
                }
        // System.out.println(diff);
        System.out.println(rr + " " + cr);
    }
```

```
    private static void calcDistances(double[][] distances, int[]
coordinates) {
        for(int r = 0; r < distances.length; r++)
            for(int c = 0; c < distances[0].length; c++)
                for(int i = 0; i < coordinates.length; i += 2) {

                    int rs = coordinates[i];

                    assert 0 <= rs && rs < distances.length : "ROW OUT OF
BOUNDS. rows = " + distances.length + " row data: " + rs;
                    int cs = coordinates[i + 1];
                    assert 0 <= cs && cs < distances[0].length : "COL OUT
OF BOUNDS. cols = " + distances[0].length + " col data: " + cs;
                    // System.out.println(distances[rs][cs]);
                    double tempD = Math.pow(r - rs, 2);
                    tempD += Math.pow(c - cs, 2);
                    distances[r][c] += Math.sqrt(tempD);
                }
    }


}
```

```
import java.io.*;
import java.util.*;

public class Points {

    private static final String ALLOWED = "WDL";

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

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

        for(int i = 1; i <= numSets; i++){
            int winPoints = s.nextInt();
            int drawPoints = s.nextInt();
            int lossPoints = s.nextInt();
            s.nextLine();
            String results = s.nextLine().trim();
            int total = 0;
            for(int j = 0; j < results.length(); j++) {
                char ch = results.charAt(j);
                assert ALLOWED.indexOf(ch) != -1 : "BAD CHAR IN RESULTS: "
+ i + " " + ch;
                if(ch == 'W')
                    total += winPoints;
                else if(ch == 'D')
                    total += drawPoints;
                else
                    total += lossPoints;
            }
            System.out.println(total);
        }
        s.close();
    }


}
```

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

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

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

        for(int i = 1; i <= numSets; i++){
            System.out.print(i + " ");
            char[][] maze = getMaze(s);
//            for(char[] row : maze)
//                System.out.println(Arrays.toString(row));
            SPoint start = findStart(maze);
            boolean[] escape = {false, false};
            solveMaze(start.r, start.c, maze, escape);
            String result = "";
            result += escape[0] ? "YES " : "NO ";
            result += escape[1] ? "YES" : "NO";
            System.out.println(result);
        }
        s.close();
    }

    private static final int[][] directions = {{-1, 0, 1, 0},
                                               {0, 1, 0, -1}};


    private static boolean solveMaze(int r, int c, char[][] maze,
boolean[] escape) {
        if(!inbounds(maze, r, c))
            return false;
        // impassable??
        if(maze[r][c] == '*')
            return false;

        // found exit??
        if(maze[r][c] == 'E') {
            escape[0] = true;
            if(!redPresent(maze))
                escape[1] = true;
            return true;
        }
        // change current cell because we are going to try and leave
        char old = maze[r][c];
        if(old == 'R')
            maze[r][c] = 'G';
        else if(old == 'G')
            maze[r][c] = '*';
        else
```

```
            throw new RuntimeException("ENTERED BAD CELL: r: " + r + ", c:
" + c);

        // not on exit, try four directions
        for(int i =0; i < directions[0].length; i++) {
            boolean foundExit = solveMaze(r + directions[0][i],
                    c + directions[1][i], maze, escape);
            if(foundExit && escape[1]) {
                // found way out and got all clues, all done
                return true;
            }
        }
        // never found way out, so reset cell
        maze[r][c] = old;
        return false;
    }

    private static boolean redPresent(char[][] maze) {
        for(char[] row : maze)
            for(char ch : row)
                if(ch == 'R')
                    return true;
        return false;
    }

    private static boolean inbounds(char[][] maze, int r, int c) {
        return 0 <= r && r < maze.length && 0 <= c && c < maze[r].length;
    }

    private static SPoint findStart(char[][] maze) {
        for(int r = 0; r < maze.length; r++)
            for(int c = 0; c < maze[r].length; c++)
                if(maze[r][c] == 'S') {
                    maze[r][c] = 'R';
                    return new SPoint(r, c);
                }
        return null;
    }

    private static char[][] getMaze(Scanner s) {
        int rows = s.nextInt();
        s.nextLine();
        char[][] result = new char[rows][];
        result[0] = s.nextLine().toCharArray();
        for(int r = 1; r < rows; r++) {
            result[r] = s.nextLine().toCharArray();
            assert result[r].length == result[0].length : "BAD LENGTH ON
ROW IN PRISON: "
                    + r + " " + Arrays.toString(result[r]);
        }
        return result;
    }

    private static class SPoint {
```

```
        private int r, c;

        private SPoint(int ri, int ci) {
            r = ri;
            c = ci;
        }
    }

}
```

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

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

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

        for(int i = 1; i <= numSets; i++){
            System.out.print(i + " ");
            String word = s.nextLine().trim();
            assert word.equals(word.toUpperCase()) : "BAD DATA";
            boolean a = a(word);
            boolean d = d(word);
            if(a)
                System.out.print("A");
            if(d)
                System.out.print("D");
            if(!a && !d)
                System.out.print("N");
            System.out.println();
        }

        s.close();
    }

    private static boolean d(String word) {
        for(int i = 1; i < word.length(); i++) {
            char first = word.charAt(i - 1);
            char second = word.charAt(i);
            if(second > first)
                return false;
        }
        return true;
    }

    private static boolean a(String word) {
        for(int i = 1; i < word.length(); i++) {
            char first = word.charAt(i - 1);
            char second = word.charAt(i);
            if(second < first)
                return false;
        }
        return true;
    }
}


public class Snap {
    public static void main(String[] args) {
        for(int i = 1; i <= 100; i++) {
```

```
            boolean d2 = i % 2 == 0;
            boolean d3 = i % 3 == 0;
            boolean d5 = i % 5 == 0;
            if(d2)
                System.out.print("SNAP");
            if(d3)
                System.out.print("CRACKLE");
            if(d5)
                System.out.print("POP");
            if(!d2 && !d3 && !d5)
                System.out.print(i);
            System.out.println();
        }
    }


}
```

```
import java.io.*;
import java.util.*;

public class Soccer {
    public static void main(String[] args) throws IOException{
        Scanner s = new Scanner(new File("soccer.dat"));
        int numSets = s.nextInt();
        s.nextLine();

        for(int i = 1; i <= numSets; i++){
            ArrayList<String> matches = new ArrayList<String>(6);
            for(int j = 0; j < 6; j++)
                matches.add(s.nextLine());
            assert matches.size() == 6;
            ArrayList<Results> teams = getResults(matches);
            assert teams.size() == 4 : teams.size() + "";
            Collections.sort(teams);
            // System.out.println(teams);
            String result = "";
            if(teams.get(0).compareTo(teams.get(1)) < 0 )
                // clear winner?
                result = teams.get(0).name;
            else if(teams.get(0).compareTo(teams.get(2)) == 0)
                // 3 way tie or more?
                result = "UNRESOLVED";
            else {
                // 2 way tie. Find match and see if winner
                String team1 = teams.get(0).name;
                String team2 = teams.get(1).name;
                String match = "";
                int j = 0;
                while(match.length() == 0 && j < teams.size()) {
                    String temp = matches.get(j);
                    if(temp.contains(team1) && temp.contains(team2))
                        match = temp;
                    j++;
                }
                result = breakHeadToHeadTie(match);
            }
            System.out.println("data set " + i + ":" + " " + result );
        }
        s.close();
    }


    private static String breakHeadToHeadTie(String match) {
        String result = "UNRESOLVED";
        String[] data = match.split("\\s+");
        String team1 = data[0];
        int goals1 = Integer.parseInt(data[1]);
        String team2 = data[2];
        int goals2 = Integer.parseInt(data[3]);
        if(goals1 > goals2)
            result = team1;
```

```
        else if(goals2 > goals1)
            result = team2;
        return result;
    }


    private static ArrayList<Results> getResults(ArrayList<String>
matches) {
        Map<String, Results> teams = new HashMap<String, Results>();
        for(int i = 0; i < matches.size(); i++) {
            String[] data = matches.get(i).split("\\s+");
            String team1 = data[0];
            int goals1 = Integer.parseInt(data[1]);
            String team2 = data[2];
            int goals2 = Integer.parseInt(data[3]);
            // draw okay as flag becuase teams are to be upper case
            String winner = (goals1 > goals2) ? team1 : (goals2 > goals1)
? team2 : "draw";
            Results r1 = (teams.containsKey(team1)) ? teams.get(team1) :
new Results(team1);
            Results r2 = (teams.containsKey(team2)) ? teams.get(team2) :
new Results(team2);
            r1.goalsScored += goals1;
            r1.goalsAllowed += goals2;
            r2.goalsScored += goals2;
            r2.goalsAllowed += goals1;
            if(winner.equals(team1))
                r1.points += 3;
            else if(winner.equals(team2))
                r2.points += 3;
            else {
                r1.points++;
                r2.points++;
            }
            teams.put(team1, r1);
            teams.put(team2, r2);
        }
        ArrayList<Results> result = new ArrayList<Results>();
        for(String key : teams.keySet())
            result.add(teams.get(key));
        return result;
    }

    private static class Results implements Comparable<Results> {
        private String name;
        private int points;
        private int goalsScored;
        private int goalsAllowed;

        private Results(String n) { name = n; }

        public String scoreIndex () {
            return points + " " + goalDiff() + " " + goalsScored + " " +
goalsAllowed;
```

```
        }

        public int compareTo(Results other) {
            int result = other.points - points;
            if(result == 0)
                result = other.goalDiff() - goalDiff();
            if(result == 0)
                result = other.goalsScored - goalsScored;
            return result;
        }

        public int goalDiff() {
            return goalsScored - goalsAllowed;
        }

        public String toString() {
            return name + " " + scoreIndex();
        }
    }
}
```

```
import java.io.*;
import java.util.*;

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

        String alphabet = "abcdefghijklmnopqrstuvwxyz";

        Scanner s = new Scanner(new File("words.dat"));
        while(s.hasNextLine()) {
            String line = s.nextLine().trim();
            Scanner s2 = new Scanner(line);
            // System.out.println(line);
            int goodLetters = s2.nextInt();
            int badRunLength = s2.nextInt();
            int numDistinctLetters = s2.nextInt();
            String word = s2.next().trim();
            // System.out.println(word);
            if(goodLetters != alphabet.length() && containsBadChar(word,
goodLetters, alphabet)) {
                //System.out.println("BAD LETTERS");
                System.out.println("NOT GOOD");
            }
            else if(runLengthBad(word, goodLetters, alphabet,
badRunLength)) {
                //System.out.println("BAD RUN LENGTH");
                System.out.println("NOT GOOD");
            }
            else if(notEnoughDistinctChar(word, numDistinctLetters)) {
                //System.out.println("NOT ENOUGH DISTINCT LETTERS");
                System.out.println("NOT GOOD");
            }
            else
                System.out.println("GOOD");
            //System.out.println();
        }
        s.close();
    }


    private static boolean notEnoughDistinctChar(String word,
            int numDistinctLetters) {
        Set<Character> chars = new HashSet<Character>();
        for(int i = 0; i < word.length(); i++)
            chars.add(word.charAt(i));
        return chars.size() < numDistinctLetters;
    }

    private static boolean runLengthBad(String word, int n, String
alphabet,
            int badRunLength) {

        // If I were better at regular expressions I would use those
```

```
        // System.out.println(badRunLength);
        String goodLetters = alphabet.substring(0,n);
        for(int i = 0; i < goodLetters.length(); i++) {
            char currentLetter = goodLetters.charAt(i);
            String middle = "" + currentLetter;
            for(int j = 1; j < badRunLength; j++)
                middle += goodLetters.charAt(i);

            int currentPos = word.indexOf(middle);
            while(currentPos != -1) {
                int indexNextChar = currentPos + middle.length();
                int indexPrevChar = currentPos - 1;
                if( (indexPrevChar == -1 || word.charAt(indexPrevChar) !=
currentLetter)
                        && (indexNextChar == word.length() ||
word.charAt(indexNextChar) != currentLetter) )
                    return true;
                else
                    currentPos = word.indexOf(middle, currentPos + 1);
            }
        }
        return false;
    }

    public static boolean containsBadChar(String word, int n, String
alphabet) {
        String bad = ".*[" + alphabet.substring(n + 1) + "].*";
        // System.out.println(bad);
        return word.matches(bad);
    }
}
```