```
package entry;

import java.io.File;
import java.io.FileNotFoundException;
import java.io.PrintWriter;
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

public class Main {

	private static String fileName = "ProbA";

	private static double[] weight;
	private static double[][] position;
	private static int[] solution;

	private static List<Double> bestMeasures;
	private static List<Double> tempMeasures;

	public static void main(String args[]) {
		weightReader("src/data/" + fileName + ".txt");
		positionReader("src/data/Positions.txt");
		bestMeasures = new ArrayList<Double>();
		tempMeasures = new ArrayList<Double>();

		nextDescentSearch(solution);
		nextDescentSearch(solution);
		write();

		nextDescentSearch200("ProbA");
		nextDescentSearch200("ProbB");
		nextDescentSearch200("ProbC");

		tabuSearch();
	}

	public static void weightReader(String path) {
		File file = new File(path);
		try {
			Scanner scanner = new Scanner(file);
			int lineNumber = Integer.parseInt(scanner.nextLine());
			weight = new double[lineNumber + 1];
			solution = new int[120];
```

```
            weight[0] = 0;
            for (int i = 1; i < lineNumber + 1; i++) {
                weight[i] = Double.parseDouble(scanner.nextLine().trim());
                solution[i - 1] = i;
            }
            scanner.close();
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        }
    }

    public static void positionReader(String path) {
        File file = new File(path);
        try {
            Scanner scanner = new Scanner(file);
            int lineNumber = Integer.parseInt(scanner.nextLine().trim());
            position = new double[lineNumber][2];
            for (int i = 0; i < lineNumber; i++) {
                String[] temp = scanner.nextLine().split("¥¥s+");
                position[i][0] = Double.parseDouble(temp[1]);
                position[i][1] = Double.parseDouble(temp[2]);
            }
            scanner.close();
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        }
    }

    public static void tabuSearch() {
        weightReader("src/data/ProbA.txt");
        bestMeasures = new ArrayList<Double>();
        tempMeasures = new ArrayList<Double>();
        boolean[][] flag = new boolean[120][120];
        int banListSize = Math.min(20, weight.length / 3);
        int[][] banList = new int[banListSize][2];
        int count = 0;
        int worsen = 50000;

        double currentX = 0;
        double currentY = 0;
        double sumWeight = 0;
        for (int i = 0; i < 120; i++) {
            currentX += weight[solution[i]] * position[i][0];
            currentY += weight[solution[i]] * position[i][1];
```

```
                sumWeight += weight[solution[i]];
            }
            int[] current = solution.clone();
            while (count < 100000 && count < worsen * 2) {
                int[] currentPosition = new int[2];
                double iterationBestX = Double.POSITIVE_INFINITY;
                double iterationBestY = Double.POSITIVE_INFINITY;
                double tempMeasure = 0;
                // Find the best solution in each iteration with the tabu list
                for (int a = 0; a < 120; a++) {
                    for (int b = a + 1; b < 120; b++) {
                        if (flag[a][b] || (weight[current[a]] == 0 && weight[current[b]] == 0) || a
== b - 60) {
                            continue;
                        } else {
                            double tempX = currentX + weight[current[a]] * position[b][0]
                                    + weight[current[b]] * position[a][0] - weight[current[a]]
* position[a][0]
                                    - weight[current[b]] * position[b][0];
                            double tempY = currentY + weight[current[a]] * position[b][1]
                                    + weight[current[b]] * position[a][1] - weight[current[a]]
* position[a][1]
                                    - weight[current[b]] * position[b][1];
                            tempMeasure = Math.abs(tempX) + 5 * Math.abs(tempY);
                            double iterationBestMeasure = Math.abs(iterationBestX) + 5 *
Math.abs(iterationBestY);
                            if (tempMeasure < iterationBestMeasure) {
                                iterationBestX = tempX;
                                iterationBestY = tempY;
                                currentPosition[0] = a;
                                currentPosition[1] = b;
                            }
                        }
                    }
                }
                // check whether the solution become worse and change the stop condition
                double currentMeasure = Math.abs(currentX) + 5 * Math.abs(currentY);
                double iterationBestMeasure = Math.abs(iterationBestX) + 5 *
Math.abs(iterationBestY);
                tempMeasures.add(Math.log10(currentMeasure/sumWeight));
                bestMeasures.add(Math.log10(tempMeasure/sumWeight));
                if (currentMeasure < iterationBestMeasure && worsen == 50000) {
                    worsen = count;
                }
```

```
                // change the current solution to the best solution in this iteration
                int temp = current[currentPosition[0]];
                current[currentPosition[0]] = current[currentPosition[1]];
                current[currentPosition[1]] = temp;
                currentX = iterationBestX;
                currentY = iterationBestY;
                // Add the swap to the ban list and using the ban list to update the matrix
                flag[banList[count % banListSize][0]][banList[count % banListSize][1]] = false;
                flag[currentPosition[0]][currentPosition[1]] = true;
                banList[count % banListSize][0] = currentPosition[0];
                banList[count % banListSize][1] = currentPosition[1];
                // Increase the iteration count
                count++;
            }

            try {
                PrintWriter pw = new PrintWriter("src/output/TabuSearch.txt");
                for (int i = 0; i < tempMeasures.size(); i++) {
                        pw.write(bestMeasures.get(i) + "    ");
                        pw.write(tempMeasures.get(i) + "¥n");
                }
                pw.flush();
                pw.close();
            } catch (FileNotFoundException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }

    public static void nextDescentSearch(int[] s) {
        shuffle(s);
        double bestX = 0;
        double bestY = 0;
        double sumWeight = 0;
        for (int i = 0; i < 120; i++) {
            bestX += weight[s[i]] * position[i][0];
            bestY += weight[s[i]] * position[i][1];
            sumWeight += weight[s[i]];
        }
        int[] bestSolution = s.clone();
        double bestMeasure = Math.abs(bestX) + 5 * Math.abs(bestY);
        boolean flag = true;
        while (flag) {
            flag = false;
```

```
                for (int a = 0; a < 120; a++) {
                    for (int b = a + 1; b < 120; b++) {
                        if (a == b - 60) {
                            continue;
                        } else {
                            double tempX = bestX + weight[bestSolution[a]] * position[b][0]
                                    + weight[bestSolution[b]] * position[a][0] -
weight[bestSolution[a]] * position[a][0]
                                    - weight[bestSolution[b]] * position[b][0];
                            double tempY = bestY + weight[bestSolution[a]] * position[b][1]
                                    + weight[bestSolution[b]] * position[a][1] -
weight[bestSolution[a]] * position[a][1]
                                    - weight[bestSolution[b]] * position[b][1];
                            double tempMeasure = Math.abs(tempX) + 5 * Math.abs(tempY);
                            if (tempMeasure < bestMeasure) {
                                int temp = bestSolution[a];
                                bestSolution[a] = bestSolution[b];
                                bestSolution[b] = temp;
                                bestX = tempX;
                                bestY = tempY;
                                bestMeasure = tempMeasure;
                                flag = true;
                            }
                            bestMeasures.add(Math.log10(bestMeasure/sumWeight));
                            tempMeasures.add(Math.log10(tempMeasure/sumWeight));
                        }
                    }
                }
            }
        }

        public static void nextDescentSearch200(String name) {
            weightReader("src/data/" + name + ".txt");
            double bestX = 0;
            double bestY = 0;
            double sumWeight = 0;
            for (int i = 0; i < 120; i++) {
                bestX += weight[solution[i]] * position[i][0];
                bestY += weight[solution[i]] * position[i][1];
                sumWeight += weight[solution[i]];
            }
            int[] bestSolution = solution.clone();
            double bestMeasure = Math.abs(bestX) + 5 * Math.abs(bestY);
            for (int i = 0; i < 200; i++) {
```

```
                shuffle(solution);
                double iterationBestX = 0;
                double iterationBestY = 0;
                for (int j = 0; j < 120; j++) {
                        iterationBestX += weight[solution[j]] * position[j][0];
                        iterationBestY += weight[solution[j]] * position[j][1];
                }
                int[] iterationBestSolution = solution.clone();
                double iterationBestMeasure = Math.abs(iterationBestX) + 5 *
Math.abs(iterationBestY);
                boolean flag = true;
                while (flag) {
                        flag = false;
                        for (int a = 0; a < 120; a++) {
                                for (int b = a + 1; b < 120; b++) {
                                        if (a == b - 60) {
                                                continue;
                                        } else {
                                                double tempX = iterationBestX +
weight[iterationBestSolution[a]] * position[b][0]
                                                        + weight[iterationBestSolution[b]] * position[a][0]
                                                        - weight[iterationBestSolution[a]] * position[a][0]
                                                        - weight[iterationBestSolution[b]] * position[b][0];
                                                double tempY = iterationBestY +
weight[iterationBestSolution[a]] * position[b][1]
                                                        + weight[iterationBestSolution[b]] * position[a][1]
                                                        - weight[iterationBestSolution[a]] * position[a][1]
                                                        - weight[iterationBestSolution[b]] * position[b][1];
                                                double tempMeasure = Math.abs(tempX) + 5 *
Math.abs(tempY);
                                                if (tempMeasure < iterationBestMeasure) {
                                                        int temp = iterationBestSolution[a];
                                                        iterationBestSolution[a] = iterationBestSolution[b];
                                                        iterationBestSolution[b] = temp;
                                                        iterationBestX = tempX;
                                                        iterationBestY = tempY;
                                                        iterationBestMeasure = tempMeasure;
                                                        flag = true;
                                                }
                                        }
                                }
                        }
                }
                if (iterationBestMeasure < bestMeasure) {
```

```
                    bestSolution = iterationBestSolution;
                    bestMeasure = iterationBestMeasure;
                }
            }
            try {
                PrintWriter pw = new PrintWriter("src/output/" + name + "output.txt");
                pw.write(bestMeasure/sumWeight + "¥n");
                for (int i = 0; i < bestSolution.length; i++) {
                    pw.write(bestSolution[i] + "¥n");
                }
                pw.flush();
                pw.close();
            } catch (FileNotFoundException e) {
                e.printStackTrace();
            }
        }

        public static void write() {
            try {
                PrintWriter pw = new PrintWriter("src/output/output.txt");
                for (int i = 0; i < bestMeasures.size(); i++) {
                    pw.write(bestMeasures.get(i) + "   ");
                    pw.write(tempMeasures.get(i) + "¥n");
                }
                pw.flush();
                pw.close();
            } catch (FileNotFoundException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }

        public static void shuffle(int[] s) {
            for (int i = 0; i < 119; i++) {
                int j = i + 1 + (int) (Math.random() * (118 - i));
                int temp = s[i];
                s[i] = s[j];
                s[j] = temp;
            }
        }
    }
```