Algorithm Design and Analysis

Homework #1

Due: 14:20, Thursday, October 4, 2012

Homework submission instructions:

- Submit your programming assignment (problem 1) to the Judgegirl System (http://katrina.csie.ntu.edu.tw/judgegirl/).
- Submit your answers of the writing problems and the programming report by one of the following ways:
  1. Submit an electronic copy to the CEIBA system before the deadline; or
  2. Submit a hard copy right before the class on the day of the deadline.

  If you submit by both ways, the TA will randomly select only one of them for grading.
- Please make sure that you have written down your name and school ID in the header of your documents. Otherwise, the TA will not grade them.
- If you choose to submit the answers of the writing problems via CEIBA, please combine the answers of all writing problems into only one file in the pdf format, with the file name in the format of "hw1 [ID].pdf" (e.g. "hw1 b99902010.pdf"). Otherwise, the TA will not grade them.
- What should be included in the report:
  1. Explain how your program works in detail.
  2. Derive the time complexity of your program and briefly explain why. No need for formal proof.
- If you are stuck with any problem, you are welcome to ask the TAs via e-mails or in person during the office hour. The TA may discuss with you and give you some useful hints. :D

Problem 1 - Manhaha Problem (programming assignment)

Description

In addition to the commonly used *Euclidean distance*, there is another definition to represent the distance between two points on a plane: *Manhattan distance* (曼哈頓距離). For any two point $p = (x_p, y_p)$ and $q = (x_q, y_q)$, Manhattan distance of p, q is defined as

$$d_M(p,q) = |x_p - x_q| + |y_p - y_q| .$$

It is more convenient to use Manhattan distance since computing the square roots is no longer necessary. However, the absolute value operation performed when calculating Manhattan distance can be troublesome as well. We sometimes need to consider different cases when calculating the distance. For example, when $x_p > x_q$ and $y_p > y_q$, we have $d_M(p,q) = x_p - x_q + y_p - y_q$. However, the same equation does not apply when $x_p > x_q$ and $y_p < y_q$, $x_p < x_q$ and $y_p > y_q$, or when $x_p < x_q$ and $y_p < y_q$.

In this problem, you are given *N* points on a 2D plane. Please write a program to find the nearest point (using the definition of Manhattan distance) for each given point and the distance between them.

Input

The first line contains an integer *N* ($2 \le N \le 1{,}000{,}000$).

The next *N* lines contain the x and y coordinates of these N points, $p_0, p_1, \dots, p_{N-1}$. Note that the indices of the points range from 0 to $N-1$, and the coordinates of all points are non-negative integers less than 1,000,000,000.

The format is as follows:

N

$x_0\ y_0$

$x_1\ y_1$

…

$x_{n-1}\ y_{n-1}$

Output

Please output *N* lines. Each line contains two numbers, separated by a space character. The first number in the $(i+1)$-th line is the index number of the nearest point to p_i. If there is more than one such point, you should output the one with the smallest index. The second number in the $(i+1)$-th line is the distance between p_i and the nearest point to it.

Sample Input

```
3
4 5
1 1
9 8
```

Sample Output

```
1 7
0 7
0 8
```

Grading

Your grade for this problem will be given based on the following two parts:
The judge result (10%)
The report & the code (10%)

Problem 2 - Super Safe

Peter has a secure safe called "Super Safe: Type-n". This type-n safe consists of n buttons, indexed from 1 to n. At the beginning, all buttons of this safe are pressed. Peter needs to release all the buttons to open the safe. However, this safe has some special mechanism so that one can only press or release the buttons (change the state of a button) under the following rules:

a) One can press/release button 1 at any moment.
b) One can press/release button 2 only when button 1 is pressed.
c) One can press/release button k ($3 \leq k \leq n$) only when button $1, 2 \ldots, k-2$ are all released, and button $k-1$ is pressed.

Press/release a button is counted as one step. Since Peter needs to perform the process every time he wants to open the safe, he wonders how many steps it would take to open the safe.

1) (10%) If Peter knows how to open type-1, type-2, …, and type-$(n-1)$ safes, try to create a strategy to open type-n safe by using these methods. Let your strategy "seem" not to have any redundant steps, since later we will ask you to prove that your strategy is optimal.
2) (5%) Let a_i be the number of steps needed to open type-i safe with your strategy in 1). Please derive the recurrences of a_n; the terms in the recurrence can only include $a_i, 0 \leq i < n$.
3) (5%) Take a guess of the general form of a_n and use mathematical induction to prove it.
4) (Bonus 5%) Prove that your strategy from 1) is optimal. That is, your strategy takes a minimal number of steps.
5) (Bonus 5%) If at the beginning only button n is pressed and the other buttons are released, what is the minimal number of steps needed to open the type-n safe? Based on your answer, show that Peter always can open the safe successfully regardless of the states of the buttons at the beginning,

Problem 3 - Read Your Textbook

Give asymptotic upper and lower bounds for $T(n)$ in each of the following recurrences. Assume that $T(n)$ is constant for sufficiently small n . Make your bounds as tight as possible, and justify your answer. You can use any methods mentioned in the lecture.

1) (5%) $T(n) = 4T(n/3) + nlgn$

2) (5%) $T(n) = 3T(n/3) + n/lgn$
3) (5%) $T(n) = \ T(n/2) + T(n/4) + T(n/8) + n$
4) (5%) $T(n) = \sqrt{n}T(\sqrt{n}) + n$

Problem 4 - Dividing and Matching by Segments

On a plane, there are n amber (a kind of color) points, $a_1, a_2, \dots, a_n$, and n blue points, $b_1, b_2, \dots b_n$. The coordinates of these $2n$ points are given, and no three points taken from these points are collinear (任三點不共線). Your task is to connect each amber point to a blue point with a line segment, and at the end each blue point is connected to exactly one amber point. Since any intersection of line segments will lead to scary crosstalk, each line segment cannot intersect with any other line segment. In this problem, you are asked to design an algorithm to find a solution!

Assume that we can check if two line segments intersect in constant time. You can also assume that we can do some simple geometry computations in constant time, including computing the length of a segment, the angle between two segments, the dot product, and the cross product of two vectors.

1) (6%) Consider the following algorithm:

> ***ALGO_1***(n, $a[]$, $b[]$)
> ***for*** each $i \leftarrow 1$ to n
> ***do*** connect $a[i]$ to $b[i]$
> ***while*** there are two line segments – $a[i]$ connected to $b[j]$ and $a[x]$ connected to $b[y]$ – intersect
> ***do*** eliminate the original two connections and reconnect $a[i]$ to $b[y]$ and $a[x]$ to $b[j]$

Obviously this algorithm will give a correct solution … if it terminates! Please prove that this algorithm will terminate eventually, and show that for any given $2n$ points, a solution always exists.
(**Hint:** consider the total length of these n segments. When you reconnect a pair of intersected segments, how does the total length change?)

2) (14%) Design an algorithm that produces a solution. Analyze the complexity of your algorithm and show that your algorithm runs in $O(n^3)$ times.

Problem 5 – A Lovely Problem (Uh, probably not)

You are given 2 sets of **distinct** real numbers: $A = \{a_1, a_2, \ldots, a_{n+1}\}$ and $B = \{b_1, b_2, \ldots, b_n\}$. For any $a_k \in$ A, if $|\{a_i \in A \mid a_i < a_k\}| = |\{b_i \in B \mid b_i < a_k\}|$ (the number of elements which is less than a_k in A equals to the number of elements which is less than a_k in B), then we call a_k a *lovely number*. For example, assume that $A = \{3,2,7,4\}$ and $B = \{5,6,1\}$. Then 3 and 7 are both lovely numbers.

1) (5%) Prove that there exists at least one lovely number for any given A and B.
2) (5%) Assume that $\{a_1, a_2, \ldots, a_{n+1}\}$ are sorted and $\{b_1, b_2, \ldots, b_n\}$ are sorted (That is, $a_1 < a_2 < \ldots < a_{n+1}$ and $b_1 < b_2 < \cdots < b_n$). Design an algorithm that runs in $O(n)$ times to find a lovely number.
3) (5%) Assume that $\{a_1, a_2, \ldots, a_{n+1}\}$ are sorted but $\{b_1, b_2, \ldots, b_n\}$ are not. Design an algorithm that runs in $O(n)$ times to find a lovely number.
4) (5%) Consider the most general case: neither A nor B is sorted. Derive a lovely algorithm that runs $O(n)$ times to find a lovely number.