Homework Assignment #10: Programming Exercise #2

Due Date & Time

11:59PM Friday, January 4, 2019. Late submission will be penalized by 20% for each working day overdue.

Problem Description

Wen Chen is the captain of a rescue boat. One of his important tasks is to visit a group of islands once a day to check if everything is all right. Captain Wen starts from the west-most island, makes a pass to the east-most island visiting some of the islands, then makes a second pass from the east-most island back to the first one visiting the remaining islands. In each pass Captain Wen moves steadily east (in the first pass) or west (in the second pass), but moves as far north or south as he needs to reach the islands. The only complication is that there are two special islands where Wen gets fuel for his boat, so he must visit them in separate passes. Figure 1 shows the two special islands in pink (1 and 3) and one possible path Captain Wen could take. Calculate the length of the shortest path to visit all the islands in two passes when each island's location and the identification of the two special islands are given.

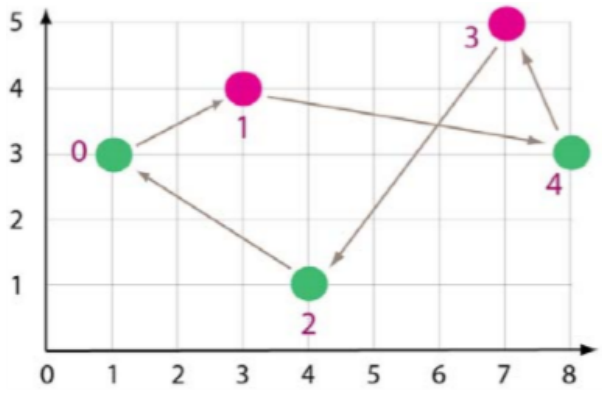

Figure 1: An example path

Input

An input has several lines. The first line has 3 integers n ($4 \leq n \leq 100$), b_1, and b_2 ($0 < b_1, b_2 < n - 1$ and $b_1 \neq b_2$), where n is the number of islands (numbered 0 to $n - 1$) and b_1 and b_2 are the two special islands. Following this, there are n lines containing the integer x- and y-coordinates of each island ($0 \leq x, y \leq 2000$), starting with island 0. No two islands have the same x-coordinate and they are in order from west-most to

east-most (that is, minimum x-coordinate to maximum x-coordinate).

Output

An output contains 1) the length of the shortest tour Captain Wen can take to visit all the islands, rounded and displayed to the nearest hundredth; 2) a space-separated list of the islands in the order that they should be visited, starting with island 0 and 1, and ending with island 0. The two parts are separated by a comma.

Sample input and output:

	case1	case2
input	5 1 3 1 3 3 4 4 1 7 5 8 3	5 3 2 0 10 3 14 4 7 7 10 8 12
output	18.18, 0 1 4 3 2 0	24.30, 0 1 3 4 2 0

Note that texts "case1", "case2", "input", and "output" are not included, only the numbers.

Notes

This assignment constitutes 4% of your grade. You may discuss the problem with others, but copying code is strictly forbidden.

Language Requirements

All the IM students are required to implement their algorithms in standard C/C++. For other students, implement your algorithms in one of the following languages.

- Standard C/C++
- Java 8
- Python 3

Do not use any library that is not included in the standard installation of compilers or interpreters. Never use a compiler-specific feature. Make sure that your code can be compiled by a standard compiler (or interpreted by a standard interpreter) without any specific argument, for example, "gcc b067050xx.c", "g++ b067050xx.cpp", "g++ b067050xx.cc", "javac b067050xx.java", or "python3 b067050xx.py".

Interface Requirements

Your application must accept a single argument, which is an input file. No other arguments will be provided. For example, "`./b067050xx case1`", "`java b067050xx case1`", or "`python3 b067050xx.py case1`". The application must output a single line as described above (that is, only numbers, spaces, and a comma). Do not output any other verbose messages.

File Requirements

You are required to provide the following three files:

- A single source file, named by your studnet ID (for example, `b067050xx.c` or `b067050xx.py`), containing the implementation.
- A README file describing how to compile your source file.
- A MS Word file or a PDF file describing algorithmic techniques applied in the implementation.

Submission Guidelines

- Pack the three required files, in a .zip file, named with the pattern "`b067050xx-hw10.zip`".
- Send the .zip file to [`r06725007@ntu.edu.tw`].

Grading

Your work will be graded according to its correctness, performance, and presentation. Before submission, you should have tested your program on several input cases. You should organize and document your program (preferably as comments in the source code) in such a way that other programmers, for example your classmates, can understand it. In the documentation, you may also want to explain how you have applied the algorithmic techniques, particularly design by induction and reduction, learned in class.

Below is a more specific grading policy:

Criteria	Score
incomplete, doesn't compile, or runtime errors	≤ 20
complete, compiles, but with incorrect results	≤ 60
correct but with an $> O(n^2)$-time algorithm	≤ 80
correct and with $O(n^2)$-time algorithm	≤ 100