***Due date: 11:00pm Friday November 16, 2007***

## Introduction

Freddy Frog is sitting on a stone in the middle of a lake. Suddenly he notices Fiona Frog who is sitting on another stone. He wants to visit her, but since the water is dirty and full of tourists' sunscreen, he decides to avoid swimming and instead reach her by jumping.

Unfortunately Fiona's stone is out of his jump range. Therefore Freddy wants to use other stones as intermediate stops and reach her by a sequence of several small jumps.

To execute a given sequence of jumps, a frog's jump range obviously must be at least as long as the longest jump occurring in the sequence.

You are given Freddy's maximum jump range, the coordinates of Freddy's stone, Fiona's stone and all other stones in the lake. Your job is to find a sequence of jumps that Freddy can use to travel to Fiona's stone.

If there are multiple sequences that Freddy can use, he will choose the one that uses the fewest jumps.

## Program Input

The input will contain one or more test cases. The first line of each test case will contain the number of stones *n, where* $0 < n < 1000$. The next *n* lines each contain two integers $x_i, y_i$ representing the coordinates of the $i^{th}$ stones. Stone #1 is Freddy's stone, stone #2 is Fiona's stone, the other *n-2* stones are unoccupied. The last line of each case will contain a single integer specifying Freddy's maximum jump distance.

Test cases are separated by a single line with nothing on it (empty line).

Note: the sample input below is pretty minimal. you need to make sure your program works with larger and more varied test sets.

## Program Output

For each test case display a line indicating the order of stones that Freddy should use to travel to Fiona.

Obviously the sequence will always start at Freddy's stone (1) and end at Fiona's stone (2).

If Freddy is unable to reach Fiona, print a line that says "Freddy cannot reach Fiona".

## Sample Input

```
2
0 0
3 4
6

3
17 4
19 5
18 5
2

3
2 4
10 5
6 4
4
```

*Note:* *Freddy is always on the first stone listed, and Fiona is always on the second one.*

*For example, in the second scenario Freddy is on 17 4 and Fiona is on 19 5.*

*In the third scenario Freddy is on 2 4 and Fiona is on 10 5.*

## Sample Output

```
Case 1: 1 2
Case 2: 1 3 2
Case 3: Freddy cannot reach Fiona
```

## Other Requirements and Guidelines

1. You may work in pairs, or by yourself. If you work with someone else, each person is expected to contribute equally to the assignment. Reports of partner-exploitation should be directed to the course instructor. The names and ID's of both partners must be included as per the submission guidelines below.
2. For this assignment you must use Java, and your program must compile using the Java 6 release.
3. Your program must read the input from a text file. The name of the text file will be passed as the only command line argument (as per the following submission requirements).
4. You are free to use any of the data structures or algorithms in the Java Collections Framework, or you can develop your own structures and algorithms if you prefer.
5. You are free to use the graph class implementation given to you in class, or you can write your own.
6. You are not required to include an efficiency analysis with your submission ( this time I just need the program).
7. Your program will be evaluated against the instructors test files, which will not be provided in advance. It is up to you to understand the requirements and make sure your solution meets them.
8. Programming style, design, technique, and inline documentation are up to you. This is not a course in software engineering or software design, however, you are expected to write code that can be easily understood by the instructor. Also, since this is an algorithm course, we will be looking at the overall efficiency and sophistication of the algorithms and structures that you use.
9. Each source code file in your submission must include a header block listing the author(s), date, and description of the file contents.
10. These requirements are subject to change. Any clarifications or changes will be announced in WebCT, and documented in a new version of this file. It is your responsibility to monitor WebCT so that you are aware of any assignment updates or changes.

## Submission Details

To receive full marks, your assignment must be submitted to WebCT before ***11:00pm Friday November 16, 2007.*** WebCT will be configured such that assignments submitted after this time will be marked as ***late***. Late assignments will be graded as follows:

- for each day (24hrs or portion thereof) the assignment is late, 15 marks will be deducted, up to a maximum of 75. Programs that are more than 5 days late will not be graded.

There will be no exceptions to this policy, regardless of your reason. It is up to you to manage your time and get the assignment submitted ***before*** the deadline.

Your submitted assignment will consist of a single zip file. When your submitted zip file is unzipped, it should create a directory with your real name(s) (initial & last name). For example, if I was working with Albert on the assignment, unzipping will create a directory called: rneilson_awei (or something similar).

Within this directory I expect to find a directory containing all the Java source files required to build your program. Your submission must not include java SDK files, and must meet the following criteria:

a. Your "main" function must be in a class called Main, in a package called "frog", and in a subdirectory called "frog" under the directory created when your submitted file is unzipped. In the above example, this means main should be in the file rneilson_awei\frog\Main.java

b. Note that packages correspond to subdirectories. This means you need to put the line: package frog; in Main.java. You will probably need to do the same with the other source files.

*How do I test your assignment?*

1. I don't use NetBeans. I just compile your program using javac. Your program must compile fine when I use the following commands to compile it: javac frog\Main.java (assuming I am in directory rneilson_awei).
2. Next I will execute your program, passing the name of one of my input files, and redirecting the output to a file. For example: java frog\Main a2data.txt > rneilson_awei_a1data_out.txt

## Grading and Evaluation

Your program will be graded according to the marks distribution shown on the next page. The instructor's test data is designed to make sure you understood and followed the requirements. Don't assume anything!

Within each category, marks will be assigned for choice of data structures and algorithms, as well as correctness and efficiency of the solution(s).

Programs that do not compile will receive a grade of zero.

Programs that are submitted late will receive a grade of zero.

Programs that do not read the input file (ie: crash on input) will receive a grade of zero.

Programs that do not accept the file name as a command line argument will receive a maximum grade of 13/100.

### Mark Distribution

Ability to follow instructions 10%

- Did you follow instructions for naming of directories, building, etc?

Program Design 20%

The program design portion of your grade will be based on (but not limited to) things like

- Choice of algorithms and data structures.
- Modularity, readability, and overall organization of the program.

Program Efficiency 10%

The program efficiency portion of your grade will be based on (but not limited to) things like

- Overall efficiency of Solution (more efficient solutions receive higher grades).

Correctness of Solution 60%

The correctness portion of your grade will be based on (***but not limited to***) things like

- Program runs without error / exception.
- An appropriate graph is created.
- Algorithms to find the path work correctly.
- Formatting of output.

## Summary of Changes to this File

1. changed sample input for the second scenario. It was incorrect.
2. changed sample input for third scenario. It was also incorrect!