CS325: Analysis of Algorithms, Winter 2024

Group Assignment 2

Due: Thur, 2/8/24

Homework Policy:

1. Students should work on group assignments in groups of preferably three people. Each group submits to CANVAS a zip file that includes their source code and their *typeset* report. The report must include the name of all group members. Specifically, for this assignment your zipped folder should contain two files named `assignment2.pdf`, and `assignment2.py`. One submission from each group is sufficient.
2. The goal of the homework assignments is for you to learn solving algorithmic problems. So, I recommend spending sufficient time thinking about problems individually before discussing them with your friends.
3. You are allowed to discuss the problems with other groups, and you are allowed to use other resources, but you *must* cite them. Also, you must write everything in your own words, copying verbatim is plagiarism.
4. *I don't know policy:* you may write "I don't know" *and nothing else* to answer a question and receive 25 percent of the total points for that problem whereas a completely wrong answer will receive zero.
5. Algorithms should be explained in plain english. You can use pseudocodes if it helps your explanation, but the grader will not try to understand a complicated pseudocode.

Let $P[1 \dots m]$, and $Q[1 \dots n]$ be two sequences of points in the plane, where each $P[i]$ (or $Q[i]$) has an x and y coordinate $P[i].x$ and $P[i].y$ (or $Q[i].x$ and $Q[i].y$). These sequences specify the locations of two sequences of rocks across a river. Two frogs, Pfrog and Qfrog, would like to cross the river while staying connected to each other by holding onto a band. The frogs have specific rules for crossing the river: (1) Pfrog must jump on all the rocks in the sequence P in order, and (2) Qfrog to jump on all the rocks in the sequence Q in order. Specifically, Pfrog starts at $P[1]$ and Qfrog at $Q[1]$. At any step, if Pfrog is at $P[i]$ and Qfrog is at $Q[j]$ they can proceed in one of the following ways:

(A) Pfrog jumps forward to $P[i+1]$, and Qfrog stays at $Q[j]$,

(B) Qfrog jumps forward to $Q[j+1]$, and Pfrog stays at $P[i]$, or

(C) Pfrog and Qfrog jump forward together to $P[i+1]$ and $Q[j+1]$, respectively.

They continue this process until Pfrog is at $P[m]$ and Qfrog is at $Q[n]$. They are looking for a sufficiently long band that allows them to cross the river while staying connected with the band,

A band is called *useful* if Pfrog and Qfrog can hold onto it during the time that they traverse P and Q across the river, otherwise, it is deemed too short. The frogs what to purchase the shortest useful band. The input to your algorithm is P, Q, and a list available band lengths $L = \{\ell_1, \dots, \ell_t\}$. Your algorithm must find the shortest useful band in L.

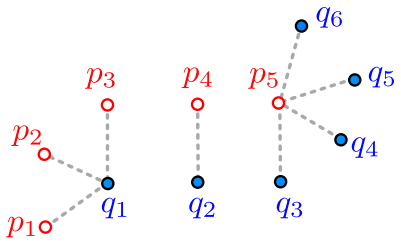

Figure 1: A valid sequence of moves for PFrog and Qfrog is $(P[1], Q[1]) \to (P[2], Q[1]) \to (P[3], Q[1]) \to (P[4], Q[2]) \to (P[5], Q[3]) \to (P[5], Q[4]) \to (P[5], Q[5]) \to (P[5], Q[6])$.

Report (60%). In your report, include the following items.

(1) Suppose P, Q and a real number ℓ are given. Let Cross$(P[1 \cdot\cdot m], Q[1 \cdot\cdot n], \ell)$ be *true* if and only if the frogs can traverse P and Q with a band of length ℓ. Come up with a recursive relation for Cross$(P[1 \cdot\cdot m], Q[1 \cdot\cdot n], \ell)$. Argue that your recursive relation is correct.

(2) Explain why the algorithm of part (1) is slow, and how you can speed it up (substantially) using Memoization.

(3) Change your memoized algorithm of part (2) into an iterative dynamic programming algorithm, and analyze it's running time. For full credit your running time should be $O(mn)$.

(4) Now, suppose that you are given a list of lengths $L = \{\ell_1, \ldots, \ell_t\}$, instead of just one number ℓ. Describe an efficient algorithm to pick the shortest useful band from L. For full credit your running time should be $O(t \log t + mn \log t)$.

Code (40%). Submit a python program for the problem outlined above. Your program will undergo testing against various test cases to assess both correctness and efficiency. For each test case, there's a time constraint—automatically stopping after 20 seconds if execution exceeds this limit. In such instances, the group will miss the points associated with that particular test case.

Note: it is important that your output is formatted as described below, since your codes will be tested automatically. You must implement the function "min_band_lenght" in the following code. The code you submit will be an implementation of this procedure in a file named "assignment2.py".

```
'''
    This file contains the template for Assignment2.  For testing it, I will place it
    in a different directory, call the function <min_band_length>, and check its output.
    So, you can add/remove  whatever you want to/from this file.  But, don't change the name
    of the file or the name/signature of the following function.

    Also, I will use <python3> to run this code.
'''

def min_band_length(input_file_path, output_file_path):
    '''
        This function will contain your code.  It wil read from the file <input_file_path>,
        and will write its output to the file <output_file_path>.
    '''
    pass

'''
    To test your function, you can uncomment the following command with the the input/output
    files paths that you want to read from/write to.
'''
# min_band_length('', '')
```

Input/Output The input file is composed of six lines in the following order.

- Line 1 of the input is a single integer $1 \leq m \leq 1000$.
- Line 2 specifies P. It is a list of m points (pair of integers) separated by commas.
- Line 3 of the input is a single integer $1 \leq n \leq 1000$.
- Line 4 specifies Q. It is a list of n points (pair of integers) separated by commas.
- Line 5 of the input is a single integer $1 \leq t \leq 1000$.
- Line 6 specifies L. It is a list of t integers separated by commas.

The output file must composed of one line that contains a single integer that is the length of the shortest useful band in L.

Sample Input (1):
2
(0,0),(2,0)
2
(0,1),(2,1)
6
5,0,1,3,12,5

Sample Output (1):
1

Sample Input (2):
2
(0,0),(2,0)
3
(0,1),(1,1),(2,1)
6
5,0,1,3,12,5

Sample Output (2):
3

Sample Input (3):
6
(0,0),(2,0),(3,0),(4,1),(3,2),(2,2)
5
(0,-1),(-1,0),(0,1),(2,1),(3,1)
8
5,0,1,3,2,5,12,18

Sample Output (3):
2