Dept. of Computer Science and Engineering,

CS69011: Computing Lab 1

Assignment 1

Maximum Marks: 30

Consider an online meeting app or friendship recommender system which recommends meetings to people based on their geographical convenience, e.g. they walk / bike / travel on common routes within the city. They have a road network in the form of a graph $G(V, E)$, where $V = \{1, \dots, N\}$ is the set of vertices denoting endpoints of road segments, and $E \subseteq V \times V$ is the set edges denoting the road segments. The company also maintains a set of paths $L = \{p_1, \dots, p_M\}$, one corresponding to each of the M users. Each path is a sequence of adjacent vertices, and of arbitrary length. A path denotes the actual road segments which the user travels by: $p_i = (v_1, \dots, v_{n_i})$. Note that vertices can be repeated in the paths.

For recommendation, the company tries to find the **maximal common subpath**, between the paths p_i and p_j taken by two users i and j, and recommends users sharing a higher length of path to each other for meeting. Given a path $p = (v_1, \dots, v_n)$, a subpath is another path $q = (w_1, \dots, w_m)$, $m \leq n$, such that $v_i = w_1, \dots, v_{i+m-1} = w_m$ for some i. A maximal common subpath r between two paths p and q, satisfies:

- r is a subpath of both p and q.
- There is no other common subpath of p and q with length strictly larger than r.

Task 1:

The first task is to write a program which takes as input two such paths in the following format, read from an input file:

```
<User No. 1>: <vertex id 1>, <vertex id 2>, … , <vertex id m>
<User No. 2>: <vertex id 1>, <vertex id 2>, … , <vertex id n>
```

Where, m and n are the lengths of each path.
The program outputs a maximal common subpath.

Algorithm 1: **Marks: 10**

This problem is similar to longest common substring problem
(see: https://en.wikipedia.org/wiki/Longest_common_substring_problem)

One way solve it is using a dynamic programming based algorithm, which solves the problem by calculating the length of longest common subpath between prefixes of the two paths. Hence, $L(i, j)$ is the length of longest common subpath between the prefixes of paths $u_1, \dots, u_i$ and $v_1, \dots, v_j$.
One can see that if $u_i = v_j$, then $L(i, j) = L(i - 1, j - 1)$, else $L(i, j) = 0$. For more details, see section 15.4 in the book by Cormen et al.

Algorithm 2: **Marks: 10**

The second way of solving this problem is using a rolling string hashing function, similar to the one used in Rabin and Karp algorithm. The overall scheme is:

1. Pick a length l, and hash all l-length subpaths from both paths (u and v) into a hashtable and find the pairs of subpaths which collide. This can be done in $\Theta(m+n)$ time. Check that these subpaths are actually same and return a match.
2. Repeat the above step for all lengths l, and return a subpath of highest length.

Hashing algorithm: Rolling hash for a string $x_1, \ldots, x_n$, can be computed as: $\sum_{i=1}^{n} x_i d^i \ mod\ p$, where p is a prime and d is the alphabet size (in this case the total number of vertices).

Task 2: **Marks: 10**

In the second task, you are given a set of such paths. You have to find a maximal common subpath, which is common to all the input paths. The input format is:

```
<Number of Paths>
<User No. 1>: <vertex id 1>, <vertex id 2>, … , <vertex id n₁>
<User No. 2>: <vertex id 1>, <vertex id 2>, … , <vertex id n₂>
…
<User No. H>: <vertex id 1>, <vertex id 2>, … , <vertex id n_H>
```

Where, the number of paths is H, and each path is of length $n_1, n_2, \ldots, n_H$.

Algorithm:

Use the hashing-based approach described in Algorithm 2 of Task 1. However, in Step 1 return a match only if sub-paths from H paths have been matched in a hash bucket.

Submission:

Submit 3 c source files for: task 1 algorithm 1, task 1 algorithm 2, and task 2, respectively. Write your name and roll number in a comment at the top of the file along with compilation instructions. The program should read all input from a file and the name of the file should be first argument. You can assume that maximum number of vertices and maximum length of each path is 1000, and maximum number of users is 100.

Marks will be given for more understandable (logical order of functions, naming of variables, etc.), maintainable (proper comments) and concise (no code duplication) programs.