University of Freiburg
Dept. of Computer Science
Prof. Dr. F. Kuhn
S. Faour, M. Fuchs, A. Malyusz

Algorithm Theory
Exercise Sheet 6

Due: Monday, 29th of November 2024, 10:00 am

Assumption: You may assume that calculations with real numbers can be performed with arbitrary precision in constant time.

Exercise 1: Efficient Sorting in a Connected System *(8 Points)*

We are given a system of N elements, each uniquely labeled 1 to N. Initially, the elements are distributed across N locations, also labeled 1 to N. Let $p = (p_1, p_2, \ldots, p_N)$ represent a permutation of $(1, 2, \ldots, N)$, where p_i denotes the current location of element i. The objective is to sort the elements such that element i is at location i for all $i \in \{1, 2, \ldots, N\}$.
The system is also equipped with a set of bidirectional connections, called *tunnels*, which can be used to swap elements between locations. Each tunnel connects two distinct locations (a_i, b_i) and has an associated *capacity* w_i. The capacity of an edge has nothing to do with the numbers on the vertices it is just a number. The sorting process involves using these tunnels to perform swaps.
The goal is to determine the maximum minimal tunnel capacity that allows the elements to sort themselves into their designated locations. In other words, given a solution take the smallest capacity edge used in the swaps. You wish to maximize this number.

(a) Prove that if for every element i, location p_i is already in the same connected component as location i we can sort. *(2 Points)*

(b) Solve the problem if you can only use tunnels with capacity $\geq w$, determine whether the system can be sorted using only tunnels with capacity at least w. This should be accomplished in $O(N + M \cdot \alpha(N))$ amortized time, where $\alpha(N)$ is the inverse Ackermann function. *(2 Points)*

(c) Solve the full problem, determine the maximum minimal tunnel capacity required to sort the system. This should be accomplished in $O(\log M \cdot (N + M \cdot \alpha(N)))$ amortized time. *(3 Points)*

,where M is the number of tunnels.

Example

If you are given the permutation $[2, 3, 1]$ and tunnels $(1, 2)$ of weight 1, $(2, 3)$ of weight 2 and $(1, 3)$ of weight 5. If we choose the tunnels $(1, 2)$ and $(2, 3)$ with weights 1 and 2 respectively then the minimum is 1. If you choose $(1, 3)$ with weight 5 and $(2, 3)$ with weight 2 then the minimum is 2. You want to maximize this minimum.

Exercise 2: Dynamic Friendship and Enmity Maintenance *(12 Points)*

You are tasked with maintaining dynamic relationships between individuals during a peace talk. The relationships are governed by rules defining friendships and enmities. The operations available must preserve these rules and provide answers efficiently.
The problem involves n individuals, with operations performed to establish or query their relationships. Implement the following operations, ensuring all constraints are respected:

- setFriends(x, y): Declares x and y as friends.
- setEnemies(x, y): Declares x and y as enemies.
- areFriends(x, y): Queries whether x and y are friends.
- areEnemies(x, y): Queries whether x and y are enemies.

The relationships between individuals are subject to the following rules:

- Friendship ($\sim$) is an equivalence relation:
  - **Transitivity:** The friends of my friends are my friends as well.
    $$x \sim y \land y \sim z \implies x \sim z$$
  - **Symmetry:** Friendship is mutual.
    $$x \sim y \implies y \sim x$$
  - **Reflexivity:** Everyone is a friend of himself.
    $$x \sim x$$
- Enmity ($*$) is symmetric and irreflexive:
  - **Symmetry:** Hatred is mutual.
    $$x * y \implies y * x$$
  - **Irreflexivity:** Nobody is an enemy of himself.
    $$\neg(x * x)$$
- Interaction rules:
  - **Common Enemy Rule:** A common enemy makes two people friends.
    $$x * y \land y * z \implies x \sim z$$
  - **Enemy of a Friend Rule:** An enemy of a friend is an enemy.
    $$x \sim y \land y * z \implies x * z$$

If setFriends(x, y): Declares x and y as friends, setEnemies(x, y): Declares x and y as enemies, would try to set enemies to friends or in reverse then the algorithm should recognize this. Simply output that this is wrong according to the current knowledge. Create an algorithm that implements these operations given N individuals and Q queries (operation calling) in $O((Q+N)\alpha(N))$ amortized time.

Hint: Use the Disjoint Set Union (DSU) data structure to keep track of friends and enemies and use dummy variables to sign who is the enemies of a person.

Exercise 3: Problem for the exercise session *(0 Points)*

A country has N cities and M power plants, which we collectively call *places*. The places are numbered $1, 2, \ldots, N + M$, among which Places $1, 2, \ldots, N$ are the cities and Places $N + 1, N + 2, \ldots, N + M$ are the power plants.
This country has E power lines. Power Line i ($1 \leq i \leq E$) connects Place U_i and Place V_i bidirectionally. A city is said to be *electrified* if one can reach at least one of the power plants from the city using some power lines.
Now, Q events will happen. In the i-th ($1 \leq i \leq Q$) event, Power Line X_i breaks, making it unusable. Once a power line breaks, it remains broken in the succeeding events.
Find the number of electrified cities immediately after each event. The running time should be $O(Q\alpha(N + M))$.