1. *(from DPV)* Here’s a problem that occurs in automatic program analysis. For a set of variables $x_1, x_2, \ldots, x_n$ you are given some equality constraints of the form “$x_i = x_j$” and some disequality constraints of the form “$x_i \neq x_j$”. Is it possible to satisfy all of them?

For example, the constraints

$$x_1 = x_2, x_2 = x_3, x_3 = x_4, x_1 \neq x_4$$

cannot be satisfied. Give an algorithm that takes as input $m$ constraints over $n$ variables and decides whether the constraints can be satisfied.

2. *(from Er)* Suppose we want to maintain an array $X[1 \ldots n]$ of bits, which are all initially zero, subject to the following operations.

- **Lookup**($i$): Given an index $i$, return $X[i]$.
- **Blacken**($i$): Given an index $i < n$, set $X[i] \leftarrow 1$.
- **NextWhite**($i$): Given an index $i$, return the smallest index $j$ such that $X[j] = 0$.

(Because we never change $X[n]$, such an index always exists.)

If we use the array $X[1 \ldots n]$ itself as the only data structure, it is trivial to implement **Lookup** and **Blacken** in $O(1)$ time and **NextWhite** in $O(n)$ time. But you can do better! Describe data structures that support **Lookup** in $O(1)$ worst-case time and the other two operations in the following time bounds. (We want a different data structure for each set of time bounds, not one data structure that satisfies all bounds simultaneously!)

(a) The worst-case time for both **Blacken** and **NextWhite** is $O(\log n)$.

(d) The worst-case time for **Blacken** is $O(1)$, and the amortized time for **NextWhite** is $O(\alpha(n))$.

(Hints)

- (a) think of a self-balancing search tree
- (a) you may need the **Successor** function
- (d) $\alpha(n)$ can be replaced by $\log^* n$
- (d) the amortized bound did not depend on the **Union** function being done by-rank
- (d) there is no **Whiten**.

3. *(from Er)* Consider the following simpler alternative to splaying:

**MoveToRoot**($v$):

```plaintext```
while parent(v) != null
    single rotate at v
```plaintext

```
Prove that the amortized cost of MOVE TO ROOT in an \( n \)-node binary tree can be \( \Omega(n) \). That is, prove that for any integer \( k \), there is a sequence of \( k \) MOVE TO ROOT operations that require \( \Omega(kn) \) time to execute.