HW4 Q3 Wrote Muzi Zhao Read Jingbo Yang

Algorithm Explanation

We are going to use disjoint sets (Union-Find) implemented with a forest structure. Initially, we have n disjoint sets to represent the dinosaur bones. We will loop over the list L to find all pairs of the form $S(i, j)$ (same species) and union each pair. After all sets are properly unioned, we loop over L again to check the pairs of the form $D(k, l)$ (different species).

1. Firstly, for each pair $D(k, l)$, we check if bones k and l belong to the same set using the `Find` operation.
2. If they do belong to the same set, this is a contradiction, so we immediately return `Error Found`.
3. After processing all pairs and checking for possible conflicts, we count the number of disjoint sets by checking if each bone's representative is itself.

Pseudo-Code

```
def Find_Species(L):
    for each bone i from 1 to n:
        MakeSet(i)
    for each pair (S(i, j)) in L:
        Union(i, j)
    for each pair (D(k, l)) in L:
        if Find(k) == Find(l):
            return Error Found
    count = 0
    for each bone i from 1 to n:
        if Find(i) == i:
            count += 1
    return count
```

Listing 1: Pseudo-Code for Finding Species

Time Complexity Analysis

- Lines 2-3 will take $O(n)$ steps since `MakeSet(i)` takes $O(1)$ time per bone. It is used to create n disjoint sets, one for each bone.
- Lines 4-5 (for each $S(i,j)$) involve m pairs at most. According to the lecture slides, each `Union(i, j)` operation costs $O(\log^* n)$ due to path compression and weighted union, so the total cost of these steps is $O(m \log^* n)$.
- Lines 6-8 (for each $D(k,l)$) also involve m pairs at most. In the worst case, all bones are in the same species. Given that we perform $n-1$ union operations and $m \geq n$ find operations, according to the lecture slides, the total cost will be $O(m \log^* n)$. So the total steps for find(k) will not exceed $2O(m \log^* n)$
- Lines 9-13 (counting distinct sets) will take $O(m \log^* n)$, as discussed above, due to the use of `Find(i)` operations.

Therefore, the total time complexity of the algorithm is:

$$O(n) + 3O(m \log^* n) + O(m \log^* n) = O(m \log^* n)$$

Conclusion

Since $O(m \log^* n)$ grows much more slowly than $O(mn)$, especially for large inputs, we can say that this algorithm is asymptotically better than $O(mn)$.