Vikram Thirumaran

Prof. Thanh H. Nguyen

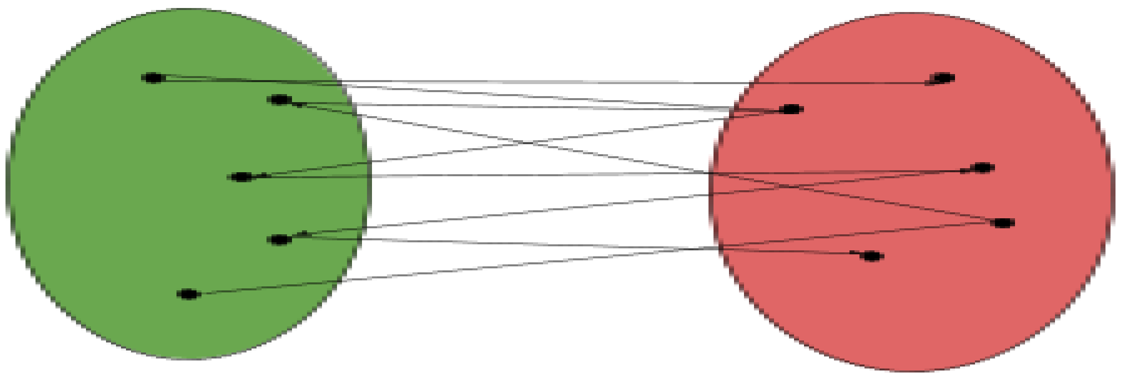
CS 315

January 19th, 2025

HW1

1.1: Your job is to give an O(n + s) time algorithm to decide whether these determinations are consistent with the gastropods belonging to just two species. Provide an explanation on how your algorithm works, its pseudocode, and an analysis on the runtime performance of the algorithm.

We can model our problem as a graph, with the slugs as vertices and the species determinations as edges. (With g_i & g_j different species, connect them with an edge). Doing this creates a bipartite graph, a concept we learned in Discrete Math. Bipartite graphs are graphs with 2 color groups of vertices, with no vertices that share an edge(adjacent) touching each other. Our colors will be one of the two species that the slugs can be. Using a searching algorithm like Breadth First Search or Depth First Search we can parse through our graph and check our structure. If it follows our set parameters of a bipartite graph, it works. My google drawing ex:



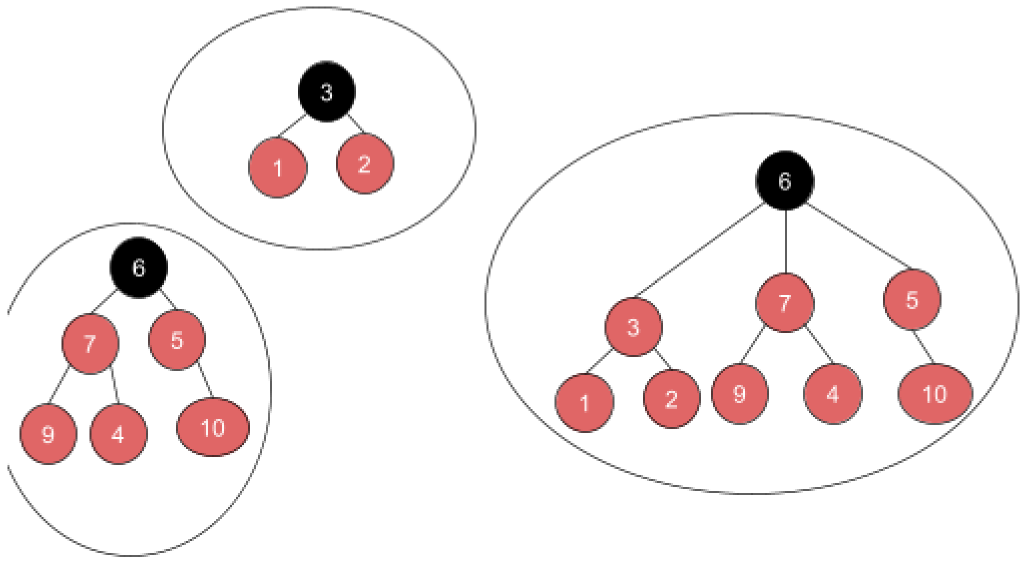
Algorithm Steps(Pseudocode)

```
1. Initialize:
    a. Create an empty queue
    b. Create an array showing all nodes uncolored
    c. From a starting node assign it a color, mark it visited and then enqueue it.
2. Traverse:
    a. While the queue is not empty:
        i. Dequeue a node
        ii. For each neighbor
        iii. if its neighbor is uncolored:
            1. Assign the opposite color to a neighbor
            2. Enqueue the neighbor.
        iv. If the neighbor is already colored and has the same color as the
            current_node:
            1. The graph is not bipartite, so we return false
3. Repeat:
    a. Continue for all connected components.
    b. If no conflicts are found(queue empty), the graph is bipartite.
```

Since both DFS and BFS algorithms have a runtime of $O(n + s)$ and making the graph itself takes the same time, the total runtime stays at $O(n+s)$.

1.2: Here modify your solution to the previous problem to make the same test of whether the gastropods could belong to just two species, but where the types of determinations tell you Same or Diff. That is, you are given a list of s determinations of the form • (i, j, Diff) indicating that gi and gj belong to different species, or • (i, j, Same) indicating gi and gj belong to the same species

The algorithm extends the previous approach by incorporating determination types ("Same" or "Diff") into the graph representation. Each edge in the adjacency list is represented as a tuple (neighbor, determination), where "Same" indicates that two nodes must have the same color, and "Diff" indicates that they must have opposite colors. During a depth-first search (DFS), the algorithm propagates colors according to the determination type: a "Same" edge assigns the same color to the neighbor as the current node, while a "Diff" edge assigns the opposite color. If a conflict occurs during this process, the algorithm halts and returns False. If all nodes are successfully processed without conflicts, the graph is deemed consistent, and the algorithm returns True. Google Drawing:



Algorithm:

1. Initialize Union-Find
    a. Create a Union-Find instance for nnn elements.
    b. parent[x] starts as x, indicating each node is its own set initially.
    c. rank[x] is initialized to 0 to track tree depth.
2. Process "Same" Constraints
    a. Iterate through the constraints list.
    b. For every (u,v,t)where t = Same :
        i. Perform Union(u, v) to merge the sets containing uuu and vvv.
3. Process "Diff" Constraints
    a. Iterate through the constraints list again.
    b. For every (u,v,t)where t = Diff :
        i. Perform Find(u) and Find(v):
            1. If Find(u) = Find(v) the constraint is inconsistent. Return False
4. Return Result
    a. If all constraints are processed without inconsistencies, return True.

The runtime for union and find are O(s) (constant related to the size of theset, or determinations). A DFS/BFS algorithm that is used in find would return the avg runtime of O(n + s).

2. Provide an efficient algorithm that determines whether their records contain such a contradiction. Provide an explanation on how your algorithm works and an analysis on the runtime performance of the algorithm. For this question, a pseudocode is not required.

The problem of determining the consistency of mine operation records can be modeled as a graph problem where mines are represented as nodes and constraints as edges. Simultaneous operation constraints indicate that two mines belong to the same equivalence class, forming strongly connected components (SCCs) in an undirected graph. Temporal order constraints impose directed edges, indicating that one mine closed before another opened. By identifying SCCs, we treat all mines within the same component as a single "super-node" in a condensed directed graph. A valid record requires that this condensed graph be acyclic, as cycles would represent contradictory temporal constraints (e.g., a mine closing before it opens). Using efficient graph algorithms, we construct the graph, condense it into SCCs, and perform a cycle check using a DFS or BFS. If a cycle is detected, it implies a logical inconsistency in the constraints, confirming the records are contradictory. If no cycle exists, the constraints are consistent, allowing the historians to proceed with confidence.

The runtime analysis reveals that graph construction takes O(n+s)O(n + s)O(n+s), where nnn is the number of mines and sss is the number of constraints. Identifying strongly connected components (SCCs) and detecting cycles also run in O(n+s)O(n + s)O(n+s). Thus, the total runtime of the algorithm is O(n+s)O(n + s)O(n+s), making it efficient for large datasets.

4o