CLC6

KariAnn Harjo

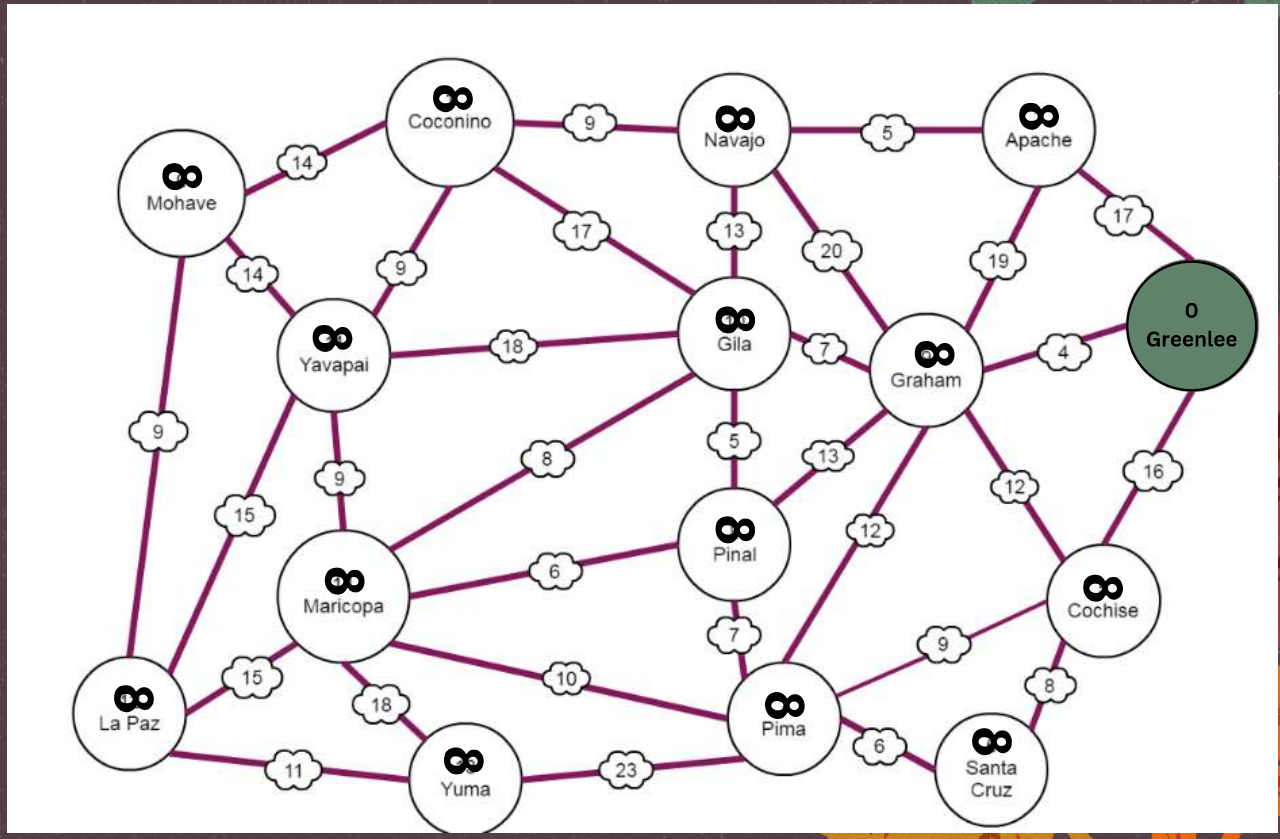
Dijkstra's Algorithm

Dijkstra's Algorithm is a graph search algorithm that finds the shortest path between a starting node and all other nodes in a weighted graph. It operates by systematically selecting the nearest vertex not yet processed, and performing a relaxation step to update neighboring vertices with the shortest path to them.

Dijkstra's Algorithm

01 Initialization

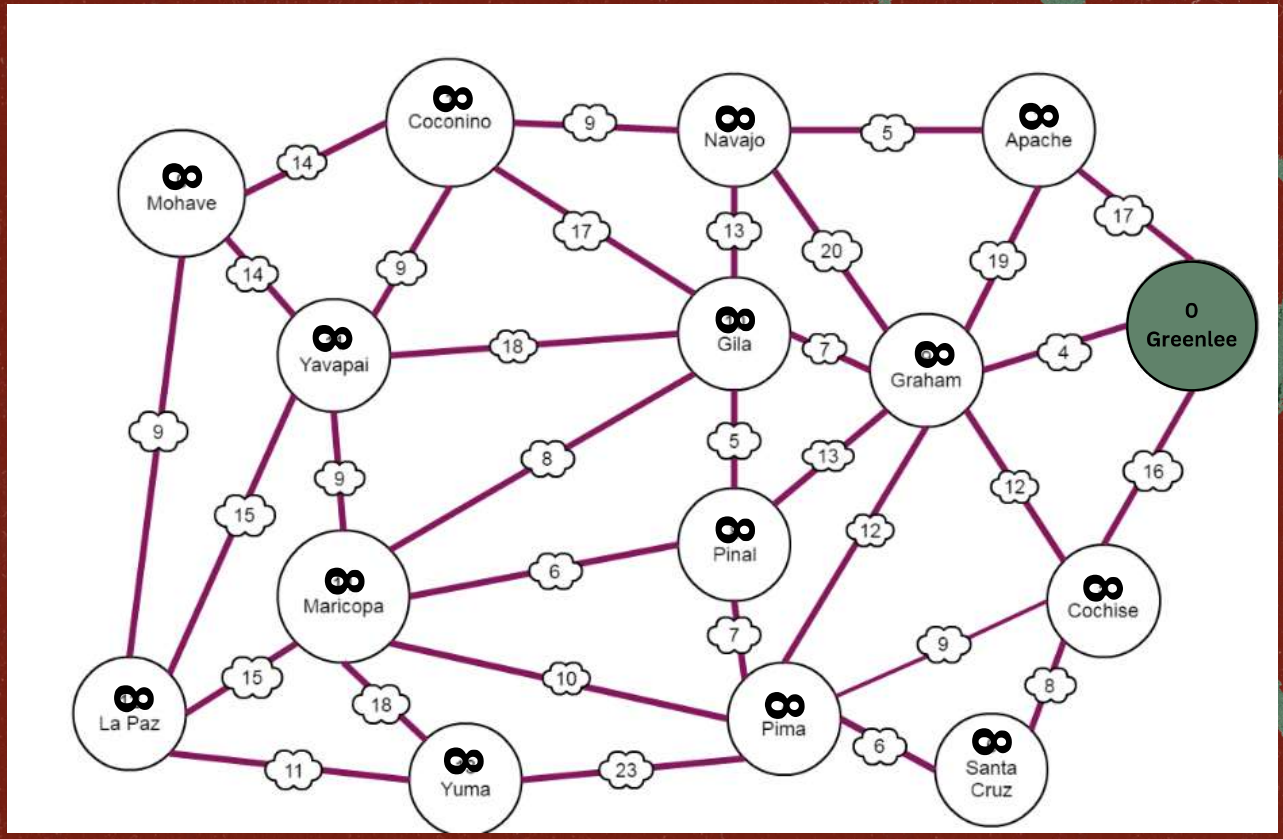
- Start with Greenlee as the "current" node, set its distance to 0, and the distances to all other nodes as infinity. Set the predecessor of each node to null.



Dijkstra's Algorithm

02 Exploration

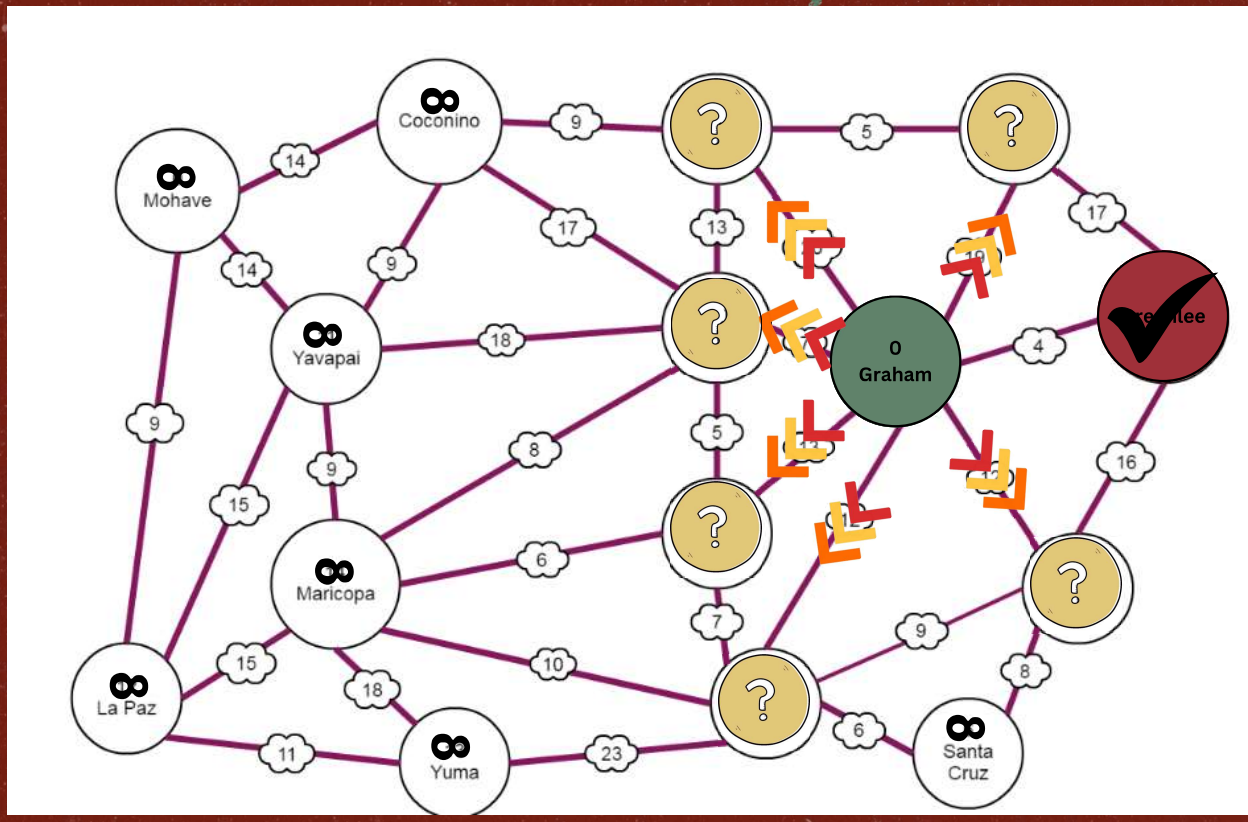
- From Greenlee, explore all the connected nodes (neighbors). Update their distances from Greenlee if it's shorter than what's currently recorded.



Dijkstra's Algorithm

02 Exploration

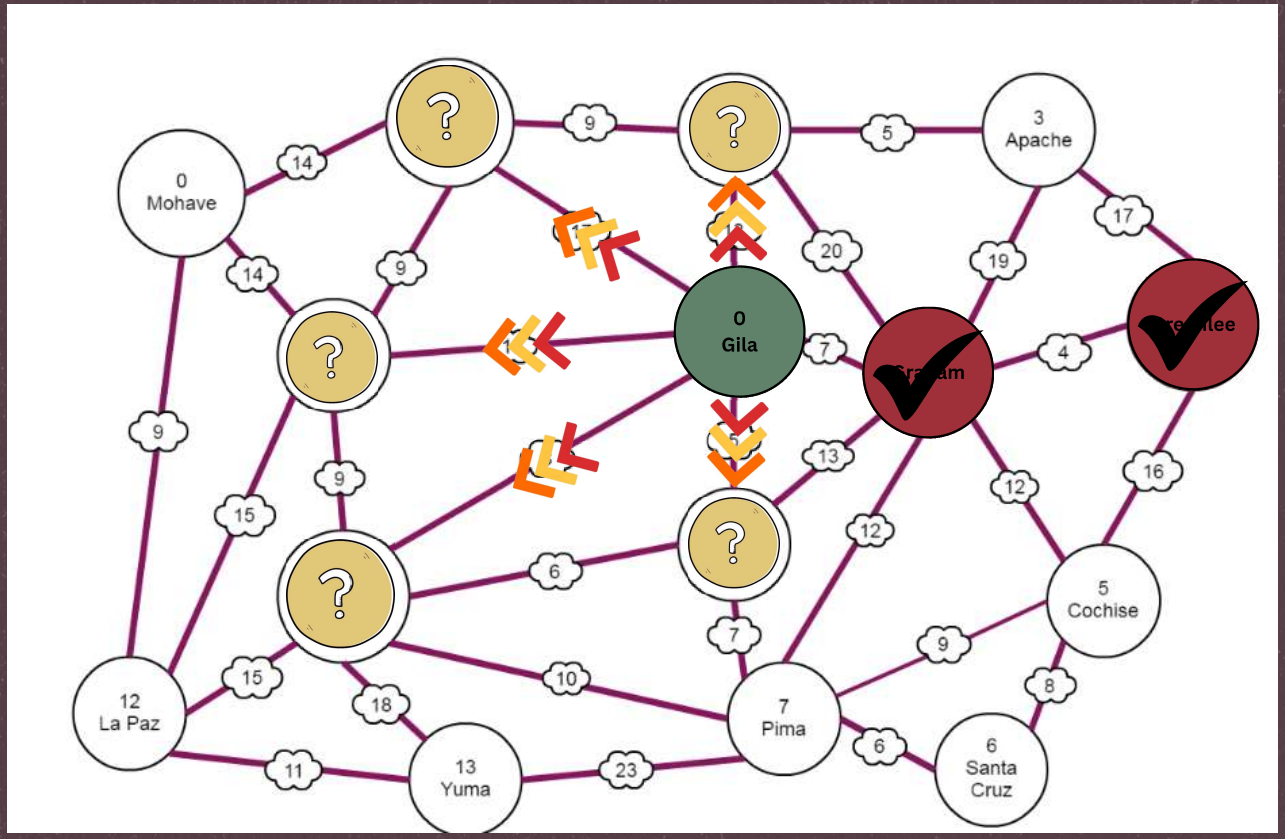
- Choose the node with the smallest tentative distance as the new "current" node, mark Greenlee as visited, and repeat the process of updating distances.



Dijkstra's Algorithm

03 Process Repetition

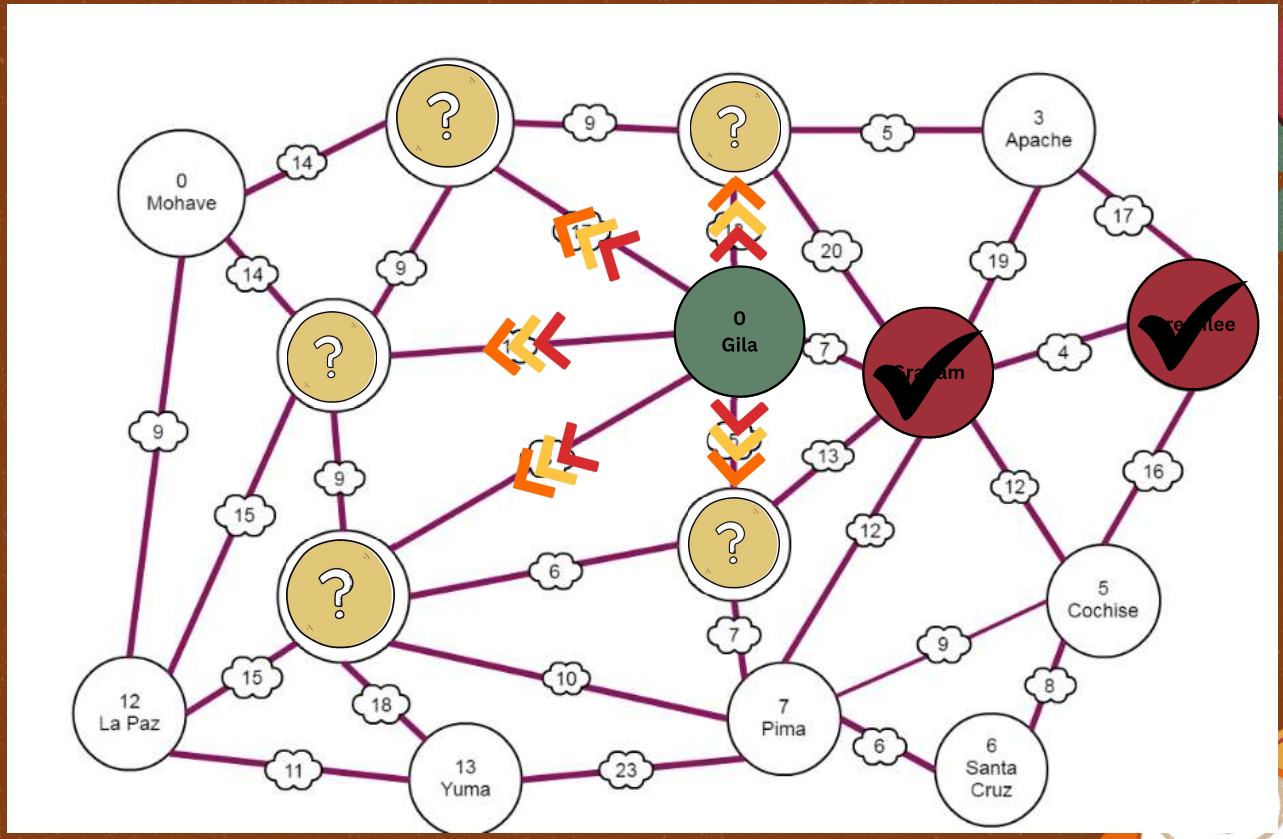
- Continue the process for each unvisited neighbor of the current node.
- After all neighbors are visited, move to the next unvisited node with the smallest tentative distance, marking the current node as visited.



Dijkstra's Algorithm

04 Completion

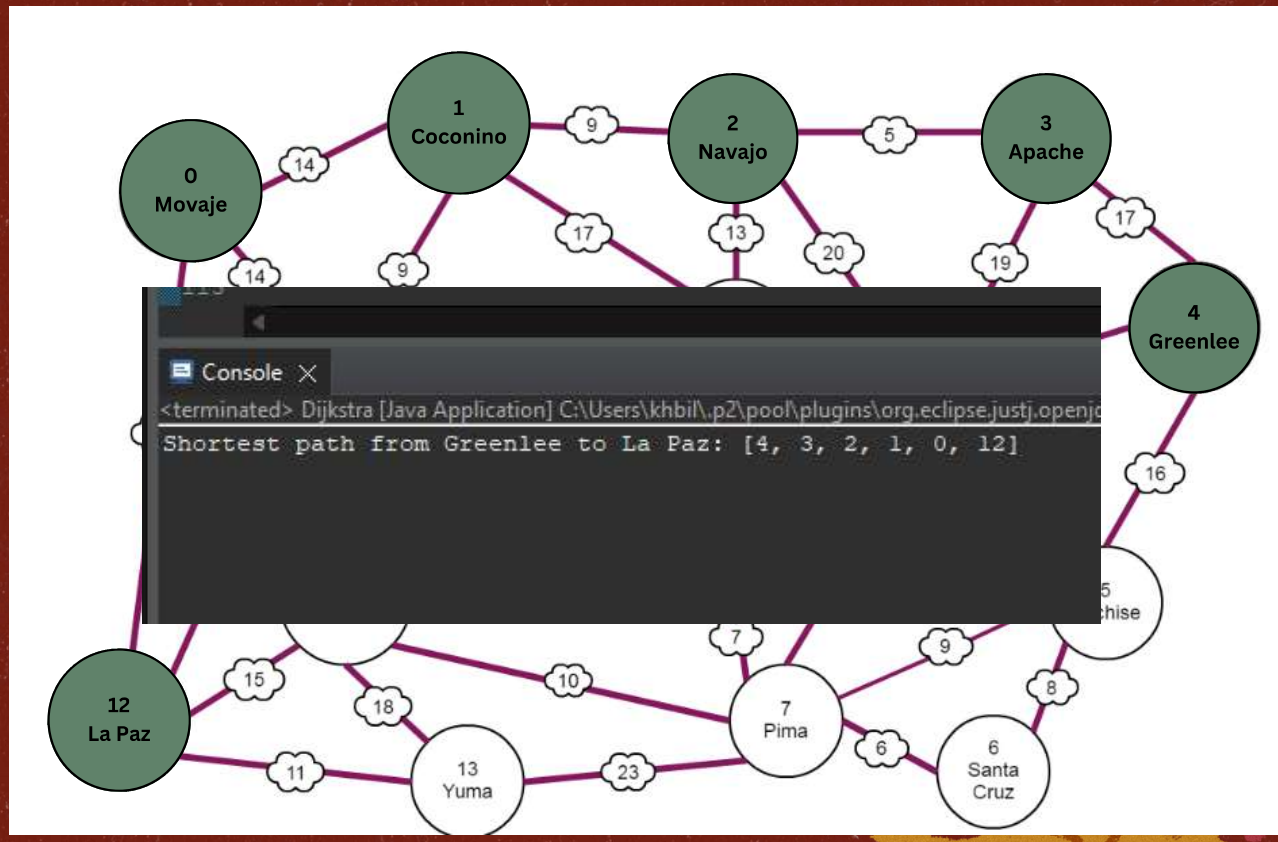
- The algorithm proceeds until it reaches La Paz, at which point the shortest path will have been determined.
- Backtrack from La Paz to Greenlee using the predecessor information to reconstruct the shortest path.



Dijkstra's Algorithm

05 Path Reconstruction

To reconstruct the path, start from La Paz and trace back using the predecessor nodes until you reach Greenlee.



Big-O Analysis

Time complexity that depends on the data structures used for the priority queue which stores the nodes.

- Using a Min-Heap Priority Queue (Binary Heap): The complexity is O((V + E) log V), where V is the number of vertices and E is the number of edges. This is because each insertion and decrease-key operation on the priority queue takes O(log V) time, and you need to perform these operations for every vertex and every edge.

- Using a Fibonacci Heap: The complexity improves to O(V log V + E), which is the best possible for this algorithm, because decrease-key operations take amortized constant time.


The end