PROJECT GOALS

Dataset: Last FM Asia Social Network
We have chosen a dataset from the Stanford Large Network Dataset Collection. Our dataset is from the popular UK music website: Last FM. To be more specific, our dataset contains information regarding the audience in Asian countries who use Last FM. Based on the user's taste in music, Last FM aims to create a profile for each of their users, keeping track of the music they like, radio stations they listen to, and etcetera. Last FM, in fact, can collect data from other music players like Spotify and Deezer.

This dataset is not directed nor temporal. The nodes in this graph represent each individual user from the asian country and edges represent if there is a relationship present between two users. The nodes also come with a label that specify each node into a specific class from multi classes. There is also a vertex feature denoted by the artists that the user likes.

Some possible analysis we can do on this large dataset include multinomial node classification, link prediction, community detection, and network visualization.

Data Structure:
A* search algorithm: Heaps
Dijkstra's Algorithm: Heaps
BFS : Queue

Algorithm:
1) Dijkstra's Algorithm:
Dijkstra algorithm is an algorithm aimed at finding the shortest path from the start to end node by comparing and finding the shortest path between two given nodes. The algorithm achieves this by first by considering all its adjacent nodes and calculating their distance or weights in order to find the smaller node. This iteration happens until the end node is marked as visited. While this algorithm is useful in finding the shortest path, it's sometimes not accurate as it does not consider the direction of where the end node might be.

Complexity:
The original algorithm uses a min-priority queue and runs in time $O(V + E)\log(V)$, where V is the number of nodes and E is the number of edges.

2) A* search algorithm:
A* search algorithm is an algorithm aimed to find a path between the start and end node with the shortest cost time or distance travelled. It works similarly to the Dijkstra's Algorithm where we follow the shortest path. However, the A* search algorithm addresses the problem with the Dijkstra's Algorithm: the Dijkstra's Algorithm only follows the shortest path of its adjacent nodes and doesn't look at the bigger picture.

The A* search algorithm utilizes the heuristic value when calculating which node would take the shortest route. The heuristic value can indicate that whichever node we choose would lead us closer to our end node.

What is the heuristic value? Each node's heuristic value determines an underestimated value that is taken from current node to the end node. Closest node at each iteration is determined by

$f(n) = g(n) + h(n)$

where n is the next node on the path, $g(n)$ is the cost of the path from the start node to n, and $h(n)$ is a heuristic function that estimates the cost of the cheapest path from n to the goal. This algorithm commonly uses the heap data structure where the node (f) with the lowest value is removed and its adjacent nodes are added.

Traversal:

BFS is a traversing algorithm which starts traversing from a selected node (source or starting node) and traverses a graph layerwise therefore exploring the neighbour nodes (nodes which are directly connected to the source node). BFS uses a queue (first in first out) to remember to get the next vertex to start a search, when a dead end occurs in any iteration.

BFS employs the following rules.

1. Rule 1 − Visit the adjacent unvisited vertex. Mark it as visited. Display it. Insert it in a queue.
2. Rule 2 − If no adjacent vertex is found, remove the first vertex from the queue.
3. Rule 3 − Repeat Rule 1 and Rule 2 until the queue is empty.

Complexity

The time complexity of BFS is O(V + E),

where V is the number of nodes and E is the number of edges.