Social Network Analysis using Graph Algorithms

CY-D

Abdul Sami Qasim (22i-1725)
Ahmad Abdullah (22i-1609)

To:

Ma’am Amina Siddique

Submitted on (09/12/2024)

Introduction

In this project, we implement two graph-based algorithms to analyze a social network graph:

- Dijkstra's Algorithm for finding the shortest path between two nodes.
- Longest Increasing Path for identifying the longest sequence of nodes, where the influence score of each node is strictly increasing.

Algorithms

1. **Dijkstra's Algorithm**

Pseudocode:

```
function dijkstra(start, end):
  Initialize a priority queue pq
  Initialize distances map, set all distances to infinity
  Set distance[start] = 0
  Add (start, 0) to pq

  while pq is not empty:
    currentNode = extractMin(pq)
    if currentNode == end:
      break

    for each neighbor of currentNode:
      newDist = distance[currentNode] + weight(currentNode, neighbor)
      if newDist < distance[neighbor]:
        distance[neighbor] = newDist
        previous[neighbor] = currentNode
        add (neighbor, newDist) to pq

  path = reconstructPath(previous, start, end)
  return path
```

Explanation:

- The algorithm uses a priority queue to greedily explore the shortest path.
- Initially, all nodes have infinite distance except the start node.
- The algorithm explores neighbors, updating distances and pushing the updated nodes back into the queue.

- When the destination is reached, the algorithm stops and reconstructs the path.

2. Longest Increasing Path using DFS and Memoization

Pseudocode:

```
function longestIncreasingPath(startNode):
  Initialize visited set and dp map
  Sort users by influence score

  function dfs(node):
    if node is visited:
      return dp[node]

    visited[node] = true
    longestPath = [node]

    for each neighbor of node:
      if influence[neighbor] > influence[node]:
        neighborPath = dfs(neighbor)
        if length(neighborPath) + 1 > length(longestPath):
          longestPath = neighborPath
          prepend node to longestPath

    dp[node] = longestPath
    return longestPath

  maxLengthPath = []
  for each user in sortedUsers:
    path = dfs(user)
    if length(path) > length(maxLengthPath):
      maxLengthPath = path

  return maxLengthPath
```

Explanation:

- The algorithm uses Depth First Search (DFS) to explore paths starting from each user.
- Each user is processed in order of their influence score.

- The DFS is optimized using memoization to store the longest path from each node.
- The algorithm maintains a visited set to avoid re-processing nodes.

Time Complexity Analysis

1. **Dijkstra's Algorithm**
   - Priority Queue Operations: Extracting the minimum node from the priority queue takes O(logn), where n is the number of nodes.
   - Relaxation: For each node, we explore all its neighbors. If there are m edges, this operation takes O(m).

   Thus, the overall time complexity of Dijkstra's algorithm is:

   O((n+m)logn)

   Where:

   - n is the number of nodes.
   - mmm is the number of edges.

2. **Longest Increasing Path using DFS and Memoization**
   - DFS Traversal: The DFS visits each node once, making the base complexity O(n).
   - Memoization: Each node's longest path is calculated once and stored. Thus, the total number of operations for all nodes is O(n).
   - Sorting Users: Sorting the users based on their influence scores takes O(nlogn).

   Thus, the overall time complexity of the Longest Increasing Path algorithm is:

   O(nlogn+m)

   Where:

   - n is the number of nodes.
   - m is the number of edges.