ADS Analysis for Prim

Introduction

Prim's Algorithm is a greedy algorithm that finds a minimum spanning tree (MST) for a weighted, undirected graph. It works by starting from an arbitrary node and iteratively adds new nodes and edges while ensuring that the acquired subgraph remains a tree. This report discusses the key data structures and algorithmic developments implemented to solve the MST problem using Prim's Algorithm.

> ⚠️ System Configuration
>
> Please be advised that the code presented in this report was executed on a 16-inch, 2021 MacBook Pro with an Apple M1 Pro processor and 16 GB of Memory, running macOS Sonoma 14. Performance results may vary based on system and configurations.

Data Structures

Several data structures are used to implement the algorithm: `Graph`, `NodeKey`, and `PriorityQueue`.

Graph

A Graph data structure is utilised to represent the input graph and the resulting MST. The underlying implementation is an adjacency map that stores a mapping of nodes to their neighbours along with edge labels (weights). This data structure includes supporting methods for manipulating and querying graphs, such as adding/removing nodes or edges, retrieving neighbours, or accessing edge labels.

NodeKey

The `NodeKey` data structure represents the information of nodes associated with the growing MST during the execution of Prim's Algorithm. It holds the node identifier, key (which represents the minimum distance to the existing MST), and the parent node in the MST. The `NodeKey` allows tracking of the construction process and determining the next node to be added to the MST.

Properties

1. **Node**: A unique identifier for the node in the graph. In this implementation, it is a `String` representing the name of a location.
2. **Key**: A `double` value representing the minimum distance (weight) from the node to the existing MST. The key is initialised to positive infinity for all nodes except the starting node, which is assigned a key of 0.
3. **Parent**: The identifier of a node's immediate parent. The parent is initially null for all nodes and is updated throughout the iteration of Prim's Algorithm.

Use in the Algorithm

Each node in the graph corresponds to a `NodeKey` object, which stores vital information on its position and connectivity within the MST during execution. In Prim's Algorithm, `NodeKey` objects are primarily used to:

1. Keep track of each node's current key and parent in the MST.
2. Facilitate the update of keys and parents as the MST grows.
3. Determine the next node to be added to the MST, as the algorithm selects the node with the smallest key not yet included in the MST.

By utilising the `NodeKey` data structure, one can seamlessly determine the next node to add to the MST and maintain the MST's properties.

Interaction with other Data Structures

`NodeKey` interacts with other data structures in the following ways:

- During the algorithm initialisation, each graph node is assigned a corresponding `NodeKey` object, with keys set to positive infinity (except for the starting node) and parents set to null. These NodeKey objects are stored in a `HashMap` for quick access.
- `NodeKey` objects are also aded to the `PriorityQueue`, which maintains a heap for efficient extraction of nodes with minimum keys.
- The algorithm updates the keys and parents of `NodeKey` objects by accessing their respective getter and setter methods, based on the graph's edge weights. Upon selecting the next node for the MST, the algorithm evaluates the `NodeKEy` values associated with the nodes' neighbours and consequentially updates their corresponding properties.

PriorityQueue

A `PriorityQueue` data structure is employed to maintain a priority queue of `NodeKey` objects. This data structure is required to efficiently identify the node with the minimum key at each step of the algorithm. The implementation uses an `ArrayList` to store elements as a heap and a `HashMap` to quickly access elements' indices. The `PriorityQueue` offers methods to add, remove or update elements and their priorities.

Algorithm Development

Prim's algorithm implementation in this project consists of the following main steps:

1. Read the input graph from a CSV file and store it in a Graph data structure.
2. Initialise the `NodeKey` objects and the `PriorityQueue` for all nodes in the graph.
3. Set the key for the starting node to 0 and populate the priority queue.
4. Iteratively perform the following operations until the priority queue is empty:
    a. Extract the node with the minimum key and mark it as visited.
    b. Add this node to the growing MST, connecting it with the parent specified in `NodeKey`.
    c. Update the keys for all neighbours that are not visited yet, and adjust their priority in the priority queue accordingly.
5. Calculate and print the results, including execution time, the number of nodes, and the total distance of the MST.

Conclusion

Through the thoughtful implementation of the data structures and algorithmic components, this project demonstrates an efficient and effective solution to the minimum spanning tree problem using Prim's Algorithm. By leveraging the Graph, NodeKey, and PriorityQueue data structures, we optimize the algorithm for time efficiency while maintaining a clear and concise codebase. Such an approach provides a solid foundation for further exploration and application of graph algorithms in various domains.