Prim's Algorithm: A Greedy Approach

Sumanth

June 18, 2025

Abstract

Prim's Algorithm is a **greedy algorithm** designed to find the Minimum Spanning Tree (MST) of a connected, undirected, weighted graph. It incrementally builds the MST by selecting the smallest edge connecting a visited node to an unvisited one, ensuring no cycles. This document details the algorithm's formulation, pseudocode, complexity analysis, practical applications, and implementation considerations, including adjacency matrix and priority queue implementations. Limitations, extensions, and comparisons with Kruskal's algorithm are also discussed.

1 Introduction

Prim's Algorithm computes the **Minimum Spanning Tree (MST)** of a connected, undirected, weighted graph, ensuring the total edge weight is minimized while connecting all vertices without cycles. Using a greedy approach, it starts with a single vertex and grows the MST by adding the minimum-weight edge to an unvisited vertex. The algorithm is widely used in network design, clustering, and approximation algorithms due to its efficiency, particularly for dense graphs.

2 Problem Formulation

Given a connected, undirected graph $G = (V, E)$ with $|V| = n$ vertices, $|E| = m$ edges, and edge weights $w(e)$, find a Minimum Spanning Tree $T \subseteq E$ that connects all vertices with minimum total weight. The output is a set of $n-1$ edges forming the MST and the total weight of these edges.

2.1 Key Idea

The algorithm maintains a set of visited vertices and greedily selects the minimum-weight edge connecting a visited vertex to an unvisited one. This process repeats until all vertices are included in the MST, ensuring no cycles are formed.

3 Prim's Algorithm

The algorithm can be implemented using an adjacency matrix for dense graphs or an adjacency list with a priority queue for sparse graphs. Below is the pseudocode for the adjacency matrix implementation:

Algorithm 1 Prim's Algorithm (Adjacency Matrix)

Input: Graph $G = (V, E)$, adjacency matrix w, start vertex s
Output: MST edges T, total weight
Initialize visited$[v] \leftarrow$ false for all $v \in V$
Initialize key$[v] \leftarrow \infty$ for all $v \in V$, key$[s] \leftarrow 0$
Initialize parent$[v] \leftarrow -1$ for all $v \in V$
Initialize empty set T for MST edges
visited$[s] \leftarrow$ true
for each vertex $v \neq s$ **do**
    key$[v] \leftarrow w[s][v]$
    if $w[s][v] \neq \infty$ **then**
        parent$[v] \leftarrow s$
    end if
end for
for $i = 1$ to $n - 1$ **do**
    Let u be the unvisited vertex with minimum key$[u]$
    if u does not exist **then**
        break
    end if
    visited$[u] \leftarrow$ true
    if thenparent$[u] \neq -1$
        Add edge $(\text{parent}[u], u)$ to T
    end if
    for each vertex v where visited$[v] =$ false **do**
        if $w[u][v] <$ key$[v]$ **then**
            key$[v] \leftarrow w[u][v]$
            parent$[v] \leftarrow u$
        end if
    end for
end for
Compute total weight as sum of $w(u, v)$ for $(u, v) \in T$
Return T, total weight

3.1 MST Reconstruction

The MST edges are stored in set T, with each edge represented by $(\text{parent}[v], v)$. The `parent` array tracks the parent of each vertex in the MST, allowing reconstruction of the tree structure.

4 Complexity Analysis

- **Time Complexity**:
  - Adjacency matrix: $O(n^2)$, suitable for dense graphs.
  - Adjacency list with min-heap: $O(m \log n)$, efficient for sparse graphs.
  - Fibonacci heap (theoretical): $O(m + n \log n)$.
- **Space Complexity**:
  - $O(n)$ for visited, key, and parent arrays.
  - $O(n^2)$ for the adjacency matrix.
  - $O(n)$ additional space for a priority queue in heap-based implementations.
  - $O(m+n)$ for adjacency list representation.

5 Practical Applications

Prim's Algorithm is applied in:

- **Network Infrastructure**: Designing LANs, WANs, or road/cable networks with minimal cost.
- **Approximation Algorithms**: Providing bounds for problems like the Travelling Salesman Problem.
- **Clustering**: Grouping data points in machine learning by connectivity.
- **Circuit Design**: Minimizing wiring costs in electronic circuits.

6 Example

Consider a graph with 5 vertices and the following adjacency matrix (∞ denotes no edge):

	1	2	3	4	5
1	0	2	3	∞	∞
2	2	0	4	1	∞
3	3	4	0	5	6
4	∞	1	5	0	3
5	∞	∞	6	3	0

Starting from vertex 1, Prim's Algorithm yields the MST:

Edge	$(1,2)$	$(2,4)$	$(4,5)$	$(1,3)$
Weight	2	1	3	3

Total weight: $2+1+3+3=9$.

7 Limitations and Extensions

- **Limitations**:
  - Requires a connected graph; fails for disconnected graphs.
  - Adjacency matrix implementation is inefficient for sparse graphs.
- **Extensions**:
  - **Dense Graph Optimization**: Use adjacency matrix for faster edge lookups in dense graphs.
  - **Parallel Prim's**: Distribute computation for large graphs using parallel processing.
  - **Dynamic MST**: Adapt the algorithm for graphs with changing edges or weights.

8 Implementation Considerations

- **Priority Queue**: For sparse graphs, use a min-heap (e.g., `std::priority`$_q$*ueueinC*++)*toachieve*$O(m\log n)$.
- **`Adjacency Matrix`**: `The provided implementation uses an adjacency matrix for simplicity (`$O(n^2)$`), ideal for dense graphs.`
- **`Numeric Stability`**: `Use appropriate data types (e.g., long long) for edge weights to avoid overflow.`
- **`Graph Connectivity`**: `Validate that the graph is connected before running the algorithm.`
- **`Edge Storage`**: `Store MST edges efficiently, e.g., as a list of pairs, to minimize memory usage.`

9 References

- GeeksForGeeks: Prim's Minimum Spanning Tree
- Wikipedia: Prim's Algorithm
- Cormen, T. H., et al., *Introduction to Algorithms*, Chapter on Minimum Spanning Trees.