Lab Assignment #10

Prim’s MST

Full Name: Shreyas Srinivasa
BlazerId: SSRINIVA

CS 303 Algorithms and Data Structures

Spring 2023

1. Problem Specification

The purpose of this project was to implement the Prim's algorithm and derive a minimum spanning tree from a weighted graph. The Prim's algorithm was implemented to work with a Graph implementation that stored weight information of edges. It was further tested with small datasets to verify its correctness. The algorithm was further executed on large datasets to measure real time machine specific performance and perform our time complexity analysis.

2. Program Design

```
public Graph(int size) {
    this.graph = new ArrayList<>(size);
    this.weights = new ArrayList<>(size);
    for (int i = 0; i < size; i++) {
        graph.add(new LinkedList<>());
        weights.add(new HashMap<>());
    }
    this.vertices = new Node[size];
}

public Node get(int i) {
    return vertices[i];
}

public void insert(int v1, int v2, double weight, boolean directed) {

    if (vertices[v1] == null)
        vertices[v1] = new Node((long) v1);
    if (vertices[v2] == null)
        vertices[v2] = new Node((long) v2);

    graph.get(v1).add(vertices[v2]);
    if(!directed) graph.get(v2).add(vertices[v1]);
    weights.get(v1).put((long) v2, weight);
    weights.get(v2).put((long) v1, weight);

}
```

The functions for retrieving a single vertex from the list of vertices and inserting a new vertex by specifying its neighbor. The insertion function has been now modified to work with directed weighted graphs. A list of map data structures is used to store weights for all the edges associated to each vertex.

```
public void print() {
    for (int i = 0; i < graph.size(); i++) {
        System.out.format(format:"%d -> ", i);
        for (Node v : graph.get(i)) {
            System.out.format(format:"%d %d -> ", v.getValue().intValue(),
                    weights.get(i).get(v.getValue()).intValue());
        }
        System.out.print(s:"/ \n");
    }
}

public void mstPrim(Node r) {
    for (Node u : vertices) {
        u.setKey(Double.MAX_VALUE);
        u.setP(p:null);
    }
    r.setKey(key:0.0);
    PriorityQueue<Node> Q = new PriorityQueue<>(Arrays.asList(vertices));
    while (Q.size() > 0) {
        Node u = Q.poll();
        for (Node v : graph.get(u.getValue().intValue())) {
            Double weight = weights.get(u.getValue().intValue()).get(v.getValue());
            if (Q.contains(v) && weight < v.getKey()) {
                Q.remove(v);
                v.setP(u);
                v.setKey(weight);
                Q.add(v);
            }
        }
    }
}
```

The implementation of the print function to display the contents of the graph, which is the same as used in the DFS experiment. The mstPrim() is the implementation of Prim's algorithm which is used to derive the minimum spanning tree. The pseudo code provided in the assignment document has been used to produce this. The information required to retrieve the said tree is stored in a property present in every Node object that represents a vertex.

```
public void printMst() {
    for (Node v : vertices) {
        if (v.getP() != null) {
            System.out.format(format:"%-10d %-10d %-10f\n", v.getValue().intValue(), v.getP().getValue().intValue(),
                    weights.get(v.getValue().intValue()).get(v.getP().getValue()));
        }
    }
}

public Node[] getAllVertices() {
    return vertices;
}
```

The printMst() function which actually lets us view the computed spanning tree. Printing each and every vertex and its 'p' property gives us all the edges that belong to our minimum spanning tree.

```java
public static Graph getGraphFromFile(String filename, boolean directed) {
    Graph g = null;
    try (Scanner fileScanner = new Scanner(Paths.get(filename))) {
        Integer vertices = fileScanner.nextInt();
        g = new Graph(vertices);
        Integer edges = fileScanner.nextInt();
        fileScanner.nextLine();
        while (fileScanner.hasNextLine()) {
            String line = fileScanner.nextLine().strip();
            String elements[] = line.split(regex:"\\s+");
            g.insert(Integer.valueOf(elements[0]), Integer.valueOf(elements[1]), Double.valueOf(elements[2]), directed);
        }
        fileScanner.close();
    } catch (Exception e) {
        System.out.println("Error: " + e.toString());
        System.exit(status:1);
    }
    return g;
}
```

The function used to construct a Graph object from datasets present in text files.

```java
public static void driver1(String filename, boolean directed) {

    Graph g = getGraphFromFile(filename, directed);
    //g.print();

    g.mstPrim(g.get(i:0));

    g.printMst();

}

public static void driver2(String filename, boolean directed) {
    Instant start, finish;
    long timeElapsed;

    Graph g = getGraphFromFile(filename, directed);

    start = Instant.now();
    g.mstPrim(g.get(i:0));
    finish = Instant.now();
    timeElapsed = Duration.between(start, finish).toNanos();

    System.out.format(format:"%s: Time taken for completion:= %d ns\n", filename, timeElapsed);


}
```

The driver functions which fulfill the objectives laid down by the assignment

3. **Output**

```
1          5          0.320000
2          6          0.400000
3          7          0.390000
4          5          0.350000
5          7          0.280000
```

Output 1 The output of the MST formed from tinyDG.txt

```
mediumDG.txt: Time taken for completion:= 13537669 ns
largeDG.txt: Time taken for completion:= 60488114 ns
XtraLargeDG.txt: Time taken for completion:= 1316503703 ns
```

Output 2. The execution times in nanoseconds recorded for all the other datasets provided

4. Analysis and Conclusions

The Prim's Algorithm is designed to find the minimum spanning tree in a graph. The widely used algorithm uses various methods to address the same issue. The prim algorithm first chooses the root vertex before moving along surrounding vertex paths.

Prim's algorithm, which is frequently used as a Greedy approach to find the minimum spanning tree for a weighted undirected graph. This method tends to look for edges that can build trees, and the total weight of all the edges in the tree should be as small as possible.

The spanning tree used by the technique is empty at first. Maintaining two sets of vertices is the goal. Vertices that have previously been included in the MST are in the first set, while those that have not yet been included are in the second set. It selects the minimum weight edge from among all the edges that join the two sets at each stage. It selects the edge and then adds the opposite vertex to the set containing MST.

Working on Prim's Algorithm
Prim's algorithm finds the subset of edges that includes every vertex of the graph such that the sum of the weights of the edges can be minimized. Prim's algorithm starts with the single node and explores all the adjacent nodes with all the connecting edges at every step. The working of Prim's algorithm is as follows:

1. Initially, the set A of nodes contains a single arbitrary node (starting vertex), and the set T of edges are empty.
2. Prim's algorithm searches for the shortest possible edge (u, v) at each step such that u $\in$ V-A and V $\in$
3. In this way, the edges in T form a minimal spanning tree for the nodes in A. Repeat this process until A $\neq$ V.

Analysis of Prim's Algorithm
Prim's algorithm is a greedy algorithm that finds a minimum spanning tree for a weighted undirected graph. This means it finds a subset of the edges that forms a tree that includes every vertex, where the total weight of all the edges in the tree is minimized. The analysis of Prim's algorithm by using min-heap (The Java implementation of priority queue uses a heap implementation under the hood, so we are including that in our time complexity analysis) is as follows.

For constructing heap: **O(V)**

The loop executes **|V|** times, and extracting the minimum element from the min-heap takes log V time, so while loop takes **VlogV**.

Total 'E' decrease key operations. Hence takes **ElogV**
(Vlog V + Elog V) = (V + E)logV ≈ ElogV

Time Complexity of Prim's Algorithm
Assume we are given V vertices and E edges in a graph and need to find an MST. To complete one iteration, we delete the min node from the Min-Heap and add several edge weights to the Min-Heap.

We delete the V vertex from the Min-Heap because we have V vertices in the graph, and each iteration deletes 1 edge, for a total of V-1 edges in MST, with a complexity of **O(log(V))**. And we add up to E edges altogether, with each addition having a complexity of **O(log(V))**. As a result, the total complexity is **O((V+E)Log(V))**.

Prim's Algorithm vs Kruskal's Algorithm

- The advantage of Prim's algorithm is its complexity, which is better than Kruskal's algorithm. Therefore, Prim's algorithm is helpful when dealing with dense graphs that have lots of edges. However, Prim's algorithm doesn't allow us much control over the chosen edges when multiple edges with the same weight occur.

- The time complexity of Kruskal's algorithm is O(E log V). In Prim's algorithm, all the graph elements must be connected. As a result, Kruskal's algorithm may have disconnected graphs. However, Prim's algorithm runs faster when it comes to dense graphs.

Observations based on the lab experiment

- The execution time increased greatly with the increase in the number of edges. The usage of the PriorityQueue enabled us to obtain the most efficient implementation of the algorithms which use an adjacency list :- the one which uses a binary heap. We also update the key property to increase efficiency.

5. References

- https://docs.oracle.com/javase/7/docs/api/java/util/PriorityQueue.html
- https://www.freecodecamp.org/news/priority-queue-implementation-in-java/
- https://stackoverflow.com/questions/1871253/updating-java-priorityqueue-when-its-elements-change-priority
- https://www.javatpoint.com/prim-algorithm
- https://www.geeksforgeeks.org/prims-mst-for-adjacency-list-representation-greedy-algo-6/
- https://www.geeksforgeeks.org/prims-minimum-spanning-tree-mst-greedy-algo-5/