**Report- Implementation of Indexed heaps**

A **binary heap** is a [heap](https://en.wikipedia.org/wiki/Heap_(data_structure)) [data structure](https://en.wikipedia.org/wiki/Data_structure) that takes the form of a [binary tree](https://en.wikipedia.org/wiki/Binary_tree). Binary heaps are a common way of implementing [priority queues](https://en.wikipedia.org/wiki/Priority_queue). A binary heap is defined as a binary tree with two additional constraints:

* Shape property: a binary heap is a [*complete binary tree*](https://en.wikipedia.org/wiki/Complete_Binary_Tree); that is, all levels of the tree, except possibly the last one (deepest) are fully filled, and, if the last level of the tree is not complete, the nodes of that level are filled from left to right.
* Heap property: the key stored in each node is either greater than or equal to or less than or equal to the keys in the node's children, according to some [total order](https://en.wikipedia.org/wiki/Total_order).

**Prim's algorithm** is a [greedy algorithm](https://en.wikipedia.org/wiki/Greedy_algorithm) that finds a [minimum spanning tree](https://en.wikipedia.org/wiki/Minimum_spanning_tree) for a [weighted](https://en.wikipedia.org/wiki/Weighted_graph) [undirected graph](https://en.wikipedia.org/wiki/Undirected_graph). This means it finds a subset of the [edges](https://en.wikipedia.org/wiki/Edge_(graph_theory)) that forms a [tree](https://en.wikipedia.org/wiki/Tree_(graph_theory)) that includes every [vertex](https://en.wikipedia.org/wiki/Vertex_(graph_theory)), where the total weight of all the [edges](https://en.wikipedia.org/wiki/Graph_theory) in the tree is minimized. The algorithm operates by building this tree one vertex at a time, from an arbitrary starting vertex, at each step adding the cheapest possible connection from the tree to another vertex.

**Dijkstra's algorithm** is an [algorithm](https://en.wikipedia.org/wiki/Algorithm) for finding the [shortest paths](https://en.wikipedia.org/wiki/Shortest_path_problem) between [nodes](https://en.wikipedia.org/wiki/Vertex_(graph_theory)) in a [graph](https://en.wikipedia.org/wiki/Graph_(abstract_data_type)), which may represent, for example, road networks. For a given source node in the graph, the algorithm finds the shortest path between that node and every other. It can also be used for finding the shortest paths from a single node to a single destination node by stopping the algorithm once the shortest path to the destination node has been determined.

**Code base:**

Skeleton implementation: Java library, pq.zip

References: Pseudo code from Professor’s notes.

**Input format**

An undirected graph is given as input, in the format expected by readGraph method.

After the graph, two vertices, s and t are specified. Prim's algorithm is run with source as s.

Dijkstra's algorithm is run with s as source and distance to t is printed.

**Output format**

Count based on the type of operation performed and the time taken.

**The following have been implemented:**

(a) Indexed priority queues

(b) Prim1 (priority queue of edges; using Java's priority queues)

(c) Prim2 (priority queue of vertices, using indexed heaps)

(d) Dijkstra'a algorithm for shortest paths using indexed heaps

|  |  |
| --- | --- |
| **Sample input**  5 7  1 5 8  1 4 7  1 3 6  4 3 3  3 5 6  5 3 2  5 2 1  1 2 | **Sample output**  MST: 12  Distance: 9  Time: X msec.  Memory: X MB / XX MB |

**Results:**

|  |  |  |
| --- | --- | --- |
| Files | Prims1 | Prims2 and Dijkstra’s |
| G1 | 84950  Time: 5 msec.  Memory: 1 MB / 61 MB. | MST: 90029  Distance: 12020  Time: 17 msec.  Memory: 1 MB / 61 MB. |
| G2 | 110419  Time: 10 msec.  Memory: 1 MB / 61 MB. | MST: 120508  Distance: 9106  Time: 16 msec.  Memory: 1 MB / 61 MB. |
| G3 | 153534  Time: 15 msec.  Memory: 2 MB / 61 MB. | MST: 168284  Distance: -2147476783  Time: 13 msec.  Memory: 2 MB / 61 MB. |
| G4 | 10000  Time: 5245 msec.  Memory: 492 MB / 592 MB. | MST: -2147473650  Distance: -2147483646  Time: 1738 msec.  Memory: 391 MB / 639 MB. |
| in | 12  Time: 1 msec.  Memory: 1 MB / 61 MB. | MST: 12  Distance: 9  Time: 6 msec.  Memory: 1 MB / 61 MB. |

**Analysis:**

Space complexity of Prims1 algorithm is O(E)

Space complexity of Prims2 algorithm is O(V)

Since O(E) = O (V2), Prims 2 is much better than Prims 1 for large inputs for finding minimal spanning tree. For small inputs, both the algorithms perform similarly. But as the size of the tree increases, time taken by Prims 2 is much lesser than Prims 1. Prims 2 is approximately 3 times faster than Prims 1 for large inputs. Hence proves that Prims 2 is better than Prims 1.