CS 6301 – Implementation of Advanced Data Structures and Algorithms

- by Dr. Balaji Raghavachari

Project Report for Performance Comparison of Prims algorithm using Priority Queue (non-index) and using Indexed Priority Queues

(Short Project 0 PQ)

by Bala Chandra Yadav (bxy140430)

Ravindhar Reddy Thallapureddy (rxt140930)

Mohammad Rafi Shaik (mxs146030)

Abstract

The project involves the implementation of Prim's algorithm using priority queues and using indexed priority queues. Analysing the performance of both the implementations on varied data set (in size) was the focus. Also involves the hands on implementation of Heap, Indexed heap.

Problem Statement

Implement Prim's Minimum spanning tree algorithm in java both by using Priority Queue (java.util.PriorityQueue) and by using Indexed Priority Queue (variation of Priority Queue where we maintain the index in array of each element). Compare its running time on various inputs with varying size.

Software Requirements (Used)

Java version: Java v1.8.0_31
JDK: Oracle Java SE Development Kit 8
Editor: Eclipse

Instructions to run the code (for detailed, please check readme.txt)

To compile the file

```
>javac Driver.java
```

To run the program use the following command

```
>java Driver PATH_TO_INPUT
```

Heap Size Setting (JVM ARGS Used)

-Xms512M – Minimum Heap Size
-Xmx512M – Maximum Heap Size

Analysis

	AVG RT*(in msec)		Memory		
input	MST1	MST2	MST1	MST2	Weight of MST
prim1.txt	0	4	0 MB / 16 MB	0 MB / 16 MB	84950
prim2.txt	0	3	1 MB / 16 MB	1 MB / 16 MB	110419
prim3.txt	3	4	2MB / 16 MB	2 MB / 16 MB	153534
sp0pq-big.txt	2511	1634	458 MB / 518 MB	414 MB / 518 MB	9999

RT - Running Time*

Conclusion:

From the above analysis, we understand that MST1 (Prim's Implementation using Priority Queues) performs better when the input size is small however when we have large input size (10000 nodes, ~9million edges) MST2 (Prim's implementation using Indexed Priority Queue) out performs MST1 with considerable margin.