**Indexed Priority Queues and Prim’s Algorithm for MST Design Document**

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Date | Modified By | Summary of Changes |
| 1.0 | 03/31/2015 | Madhao Wagh | Created |

Document History

##### Table of contents

[1. Project Overview 4](#_Toc415659529)

[2. Background: 4](#_Toc415659530)

[3. Classes 4](#_Toc415659531)

[4. Interfaces 4](#_Toc415659532)

[5. Functions for PriorityQueueIndexed: 5](#_Toc415659533)

[6. Citations: 5](#_Toc415659534)

# Project Overview

To implement Indexed Priority Queues and use them to calculate the Minimum Spanning Tree using Prim’s Algorithm

# Background:

**Prim's algorithm** is a **greedy algorithm** that finds a minimum spanning tree for a connected 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. There are many ways of finding minimum spanning tree. This project implements prim’s algorithm for finding minimum spanning tree using **priority queues.**

To talk about **priority queues**, we can say, it is a queue where each element has a priority associated with it and insertion and deletion depends on their priority. But when we need to update these priorities **on the fly**, we use an **Indexed Priority Queue.**

**Indexed Priority Queue implemented for this project** requires an interface PQIndex, that contains two functions getindex() and putindex() , to be implemented by the classes whose objects will be stored in the queue. These functions maintain the indexed positions of the elements in the queue. In this way, **on the** **fly**, when we find set of edges with minimum weights to reach certain set of vertices, we decrease the weights of these vertices and percolate them up in the queue. Operations that require us to keep track of **indexed positions of** the nodes are good candidates for **Indexed priority queues**.

**Data structure for Indexed Priority Queues (used in this project):**

* Used and Array to implement queue.
* Initially array is created of the size provided in the constructor.
* Because it is a priority queue and we are implementing minimum priority queue, Array [1] saves min element.
* It satisfies the Min heap property: Parent’s weight is always less than it’s children’s weight.
* ‘ith’ parent will have left child at ‘2i’ and right child at ‘2i + 1’

# Classes

**PriorityQueueIndexed:** Implements PQIndex and functions required for priority queue.

**Graph:** Defines Vertex class, Edge class and Graph class itself.

**MST:** Implements prim’s algorithm

# Interfaces

**PQIndex:** Interface that declares two functions putIndex and getIndex.

# Functions for PriorityQueueIndexed:

**Add(T x):** inserts element x in the min heap i.e indexed priority queue.

**Remove(T x):** same as deleteMin

**deleteMin():** returns queue[1]( the minimum element) , and deletes it from the queue

**min():** it just returns queue[1] ( the minimum element) , but does not delete the element

**decreaseKey(T x):** Reorders the heap such that x is placed appropriately as per its reduced value

**percolateUP(int i):** This function moves the element present at position i, up in the tree such that the queue satisfies min heap property

**percolateDown(int i) :** This function moves the element present at position i, down in the tree such that the queue satisfies min heap property

**buildHeap(T[] arr):** Given an array as input, it rearranges the elements such that, the array satisfies min heap property.

Results for following input files:

|  |  |  |
| --- | --- | --- |
| FILE | OUTPUT | TIME in milliseconds |
| prim1.txt | 84950 | 4 |
| prim2.txt | 110419 | 5 |
| prim3.txt | 153534 | 7 |
| primck.txt | 3384476 | 544 |

**Time Complexity:**

Every operation of PriorityQueueIndexed performs in O(log n) time.

# Citations:

<http://en.wikipedia.org/wiki/Prim%27s_algorithm>