CSC 591: Graph Data Mining Fall 2016

Unity ID: avshirod

HW#2: **Graph ADTs – Binary Heaps**

## Problem 1:

Min-Heap – Node weights correspond to degree of node in the graph

(1) Cost of finding node with smallest degree – ***O(1)***
Because it will be the root of the Min-Heap tree

(2) Cost of finding node with highest degree – ***O(n/2)***
And n/2 is the number of max possible leaves of the tree
By property of Min-Heap, every child is greater than its parent; so all the leaves must be greatest nodes along that branch. If we just compare the weights of the leaf nodes, we will find the node with maximum degree.

(3) One strategy might be to remove 'k' nodes with highest degree
For that, we will have to –
(a) Find node with highest degree
(b) Vaccinate it and remove it from the tree
(c) Update the Min-Heap
(d) Repeat above steps till 'k' nodes are vaccinated

For a graph with 'n' nodes and 'k' vaccines (k<<n) –
Finding node with highest degree = O(2^(h-1)) = O(n/2); as max number of leaf nodes is n/2
Removing the node with maximum degree and rebuilding the min-heap for the new graph = O(n log n)
So overall complexity = k times [ O(n/2) + O(n log n) ] = ***O(n log n)***
With the assumption that the data structure we are using for the graph, has an added field which updates the degree of a node with any operations performed.

So if we were to run the simulation for population of Raleigh (about 500,000 people), and we assume that processing one node would take one millisecond, it would take about 47 minutes for the simulation to complete.

Pros – Easy to implement
Cons – Not the best way to decide a vaccination program

<u>Problem 2:</u>

(1) Let's assume that we are using adjacency matrix to store the graph structure.
To find out the degree of one node, we have to look at all the incoming and outgoing edges; thus we have to look at the row and column for that particular node. If we have 'n' nodes in the graph, that would mean we have to look over 2n elements.
We would repeat this for 'n' nodes; so total time complexity = ***O(n^2)***

Alternatively, if we are using Adjacency list implementation, this would take ***O(V+E)*** time, where the graph contains V vertices and E edges. This is because we will go over every node and calculate the length of its adjacency list.

We can make this more efficient by calculating degree of a node when building the graph. For every edge we add, we update the degree. That method saves the above computation time.

(2) Let's assume that we have a graph with 'V' vertices and 'E' edges.

Our algorithm is –
(a) Build an adjacency list for the graph (via edgelist) = O(E log V)
(b) Find node with max degree = O(V+E)
(c) Remove node, and update graph
(d) Repeat above steps till there are no nodes in the graph

As there are 'V' nodes in the graph, total time complexity = *O(|V| * (|V| + |E|))*
Assuming that |V| > |E|, we can say that this brute-force algorithm would take ***O(V^2)*** time.

(3) If we use a min-heap to store the degree of nodes, we can find the node with max degree in O(V log V) time.
So overall complexity of the algorithm would drop from O(V^2) to ***O(V log V)***.
This is because we will still be updating the graph and building the adjacency list, which takes O(E log V). (E because going through E edges, log V because we need to see if a node is already in the graph)

(4) If we know the range of the degree (from 0 to N-1), then we can use ***bucket sort*** by creating N-1 buckets.
Or we can use ***Radix sort***. (We assume that the representation of degree in binary form is not comparable to number of vertices in graph; i.e. number of vertices is greater than no of bits it would take to represent the degree 'N-1')
Both these methods, with valid assumptions, will give us a O(N) time complexity. (and a much worse space complexity).