Graph Data Mining HW2

Sai Kumar Goud Vengali (svengal@ncsu.edu)

11 September 2022

1. (a) For the given Binary Min-Heap, the root is always the minimum value. Hence, we have to retrieve the root value from the data structure to get the vertex with the smallest degree.
The cost requires is $\mathcal{O}(1)$

   (b) To get the maximum value from the min-heap we actually need to search for all the nodes. but in a binary min heap as the parent element will always be lesser than it's child, we can reduce our search elements from N to the number of leaf nodes.
let N be the number of nodes in a min-heap then,

   Number of leaf nodes=$\lfloor(n/2)\rfloor$ Therefore, the cost of finding the vertex with largest degree is $\mathcal{O}(n/2)$ which asymptotically equal to $\mathcal{O}(n)$.

   (c) The major requirements for the problem are:
To find and immunize the person who has more connections with his neighbor i.e the node with the highest degree.
Next, as we're considering the min heap we need to remove the maximum degree for each simulation and update the degree of the neighbors.

      i. Psuedocode for above scenario:

      ii. Here we assumed that the highest degree indicates the higher risk and the number of people is small.

      iii. The complexity of the algorithm is
time required to find the max element, deleting it and time to heapify.

$$O(N).O(1).O(logN) = O(N)$$

The above algorithm is run for k times until the immunizations are left. So, the total time required for this algorithm is

Total complexity = $\mathcal{O}(N)(k)(logN) = \mathcal{O}(NklogN)$

Algorithm 1

```
function D(e)leteMax(int A[],int k, int N):
    intitalize count=0, max = A[0],x=0
    for c = 1, 2, ... K do
        for i = 1, 2, ..., N do
            if A[i] > max then
                get this index and delete the maximum degree
            end if
            Set the pointer to the next maximum value
            Call the heapify function
        end for
    end for
    function H(e)apify(A[ ], int n, int i ):
        if A[i] < A[i/2] then
            swap the child and parent
            swap(A,i,i/2)
            heapify(A,i,i/2)
        end if
        function S(w)ap(A[], i, j):
            if A[i] < A[i/2] then
                int x
                x = A[i]
                A[i] = A[j]
                A[j] = x
            end if
```

iv. **Pros:**
This works well for the small values of n as we can easily find the person with highest degree and update the heap.

Cons: For higher value of n, the complexity increases in finding the maximum degree as the number of nodes increases.

2. (a) To find the highest degree we need to
find the degree of all vertices and then find the highest from them.
let's assume the given graph is stored in adjacency matrix then;
1. To find a degree of a vertex we need to find the total number of vertices it is connected to; This can be done in $\mathcal{O}(\mathcal{V})$.
2. This should be done for all the vertices of V. So, it takes $\mathcal{O}(\mathcal{V})$.

Total complexity = $\mathcal{O}(\mathcal{V}.\mathcal{V}) = \mathcal{O}(V^2)$

(b) Given that the graph is stored in adjacency list.
N= number of vertices in a graph.

1. Time required to find the highest degree is finding the length of each vertex's adjacency list and updating your max variable which can take upto $\mathcal{O}(V)$.

2. Removing the highest degree is done by remove the vertex with maximum degree and then updating all the other lists of its connected vertices which results in:

$$Time\ \ complexity = O(deg(V))$$

3. we have to repeat the above 2 steps till we run out of nodes:

$$Time\ \ complexity = O(V)$$

Therefore, total Time Complexity = $\mathcal{O}(V) * (V + deg(V)) = \mathcal{O}(V^2)$

(c) If we consider min-Heap, while implementing we need to store the negative values of degree in the heap. So, with this modification.
N n̄umber of vertices. 1. We can retrieve the maximum element from the min-heap by popping the root value and find absolute of it. This requires $\mathcal{O}(1)$.
2. Next, we have to heapify the remaining elements which takes $\mathcal{O}(logN)$.
3. Now we have to remove the maximum value out of the heap till v becomes zero.This takes $\mathcal{O}(n)$

$$Time\ \ Complexity = O(N.logN)$$

(d) If the degree of vertices are given between 0 and N-1, the Radix sort algorithm sorts the given vertices in increasing or decreasing order

with $\mathcal{O}(n)$.

Psuedocode:

Algorithm 2 Radix sort

```
d = number of digits
create d buckets of size 0-9
for i = 1, 2, ... d do
    sort the elements using counting Sort
end for


function (Count Sort):
    intitialize 0's for array of length N
    for j = 1, 2, ... N do
        calculate the count of each element and store in the array
    end for
    for i = 1, 2, ..., N do
        calculate the cumulative sum of the elements and d store it in count
array only
    end for
    for j = N, N − 1, ..., 1 do
        Now for each element in original array we find the index in Count
array
        Decrease the count by one.
    end for
```

Time complexity: O(N.d)