ICCS 313: Assignment 3

Tawan Chaeyklinthes u5980963

Date: 07 Oct 2018

Problem 1

A tree is an acyclic and connected undirected graph which infer that it must have $n-1$ edges.

```
def isTree(V,E):
   n = |V|, m = |E|
   if(m != n-1):
      return False
   else:
      Run bfs on the graph.
      Keep track of all visited nodes in a set, S.
      if len(S) == n:
         return True
      else:
       return False
```

Therefor, the running time of this algorithm is $= O(m+n)$.

Problem 2

Lemma : If G is a DAG, then G has a node with no entering edges.

Proof :

For the sake of contradiction, suppose that G is a DAG and every node has at least one incoming edges. Pick a node v and follow an edge backward to u. Repeat this step for node u and follow an edge backwards to some node x. Keep doing this until we meet the same node twice, say w. Now, we have a sequence $w \to w$ which is a cycle. Thus, graph G is not a DAG .Hence we have our contradiction. Therefore, a DAG has at least one node with no incoming edges.

Lemma : If G is a DAG, then G has a node with no outgoing edges.

Proof :

For the sake of contradiction, suppose that G is a DAG and every node has at least one outgoing edges. Pick a node v and follow an edge forwards to u. Repeat this step for node u and follow an edge forwards to some node x. Keep doing this until we meet the same node twice, say w. Now, we have a sequence $w \to w$ which is a cycle. Thus, graph G is not a DAG .Hence we have our contradiction. Therefore, a DAG has at least one node with no outgoing edges.

Using the two lemmas, we can conclude that if G is a DAG, G has a node with no outgoing edges and a node with no incoming edges.

Problem 3

Obs : For some node v, if $deg(v) = n/2$, v must be connected to $n/2 + 1$ nodes.

Lemma : Let G be a graph of n nodes, where n is an even number. If every node of G has degree at least $n/2$, then $G$is connected.

Proof:

For the sake of contradiction, let assume that G is a disconnected graph comprise of graphs G_1 and G_2 and every node has degree at least $n/2$. In G_1, since every node has to has a degree at least $n/2$, $|V_1| \geqslant n/2 + 1$ and therefore $|V_2| = |V| - |V_1|$ which is $\leqslant n/2 - 1$. With $\leqslant n/2 - 1$ nodes in G_2, the maximum degree of the nodes in G_2 is $(n/2 - 1) - 1 = n/2 - 2$. Hence we have a contradiction. Therefore if every node has degree at least $n/2$, the graph has to be connected.

Problem 4

We can find the order in which to serve the customer by sorting their service time, t_i so that $t_1 \leqslant t_2 \leqslant t_3 ... \leqslant t_n$ whereby customer with t_1 will be served first and customer with t_n will be served last.

Problem 5

(a) Let's look at this problem as a graph where each node represent a person and each edge shows the friendship between any 2 persons.

- Firstly, looping through the list edges ,we find out the degree of each node.
- Keep the degree and name of each person as a pair.
- Sort these pairs according to their degrees in ascending order.
- Iterate if the first pair has degree < 5, we removed it and update its neighbors.
- Sort these pairs.
- Repeat the previous 2 steps until first pair have degree $\geqslant 5$.
- Now invite all remaining pairs.

Let S be the set that is produced by greedy and T be the set of optimal solution. And let the set that is removed from S and T be $q_1, q_2, ..., q_n$ and $r_1, r_2, ..., r_m$ respectively, where q_n and r_m are just some pairs.

Lemma 1 : At the end of each iteration degree of $q_1, q_2, ..., q_k < 5$.

Proof :

Initialized: The removed set is empty. Thus the claim is trivially true.

Maintenance: Assuming that the in iteration k , the claim is true. In iteration $k + 1$, if the degree of first element in the remaining set is $\geqslant 5$ then the loop will terminate.

However , if degree > 5, then it will be removed from S. Thus the q_k will be add to removed set and now degree of $q_1, q_2, ..., q_{k-1}, q_k < 5$.

Terminate: The loop will terminate when the degree first pair element $\geqslant 5$. Thus, $q_1, q_2, ..., q_n < 5$.

Lemma 2 : The greedy will returns an optimal set S .

Proof :

For the sake of contradiction, let assume that S is not optimal which means that the number of pairs removed from S is more than that of T. In other word lets assume that $m = k$, the $n = k + 1$. From the lemma above, we know that $deg(i_{k+1}) < 5$. Thus it can be said that in T there are still pairs with degree < 5 left. Thus we have a contradiction since T is supposed to be an optimal solution.

(b) Let E be the list of edges:

```
def invite(E):
    Make adjTable ,map name to neighbours.
    Make degTable ,map name to degree.
    Make removedSet ,store removed pairs.
    Make PriorityQueue<Pair>, pq , higher priority for lower degree pairs.
    for name in degTable.key():
        p=(name,degTable.get(name))
        pq.add(p)
    while (!pq.isEmpty && pq.peek < 5):
        name, deg = pq.poll()
        removedSet.add(name)
        for neighbour in adjTable.get(name):
            if !(removedSet.contains(neighbour)):
                Update degree of neighbour in degTable.
                Create pair (neighbour,degTable.get(neightbour))
                Add new pair to pq.
    invited = []
    for name in adjTable.key():
        if(!(removed.contains(name))):
            invited.append(name)
    return invited
```

Runtime:

The first for-loop take $O(n \log n)$ to make first priority queue. In the while-loop, the maximum number of poll pq has to make is n times. This is because of the if-condition in the for-loop which make sure that a removed node is never added back to pq. Therefore, polling will take a total of $O(n \log n)$. In the second for-loop, the maximum number of iteration is $\sum deg(neighbour) = O(m)$ and $pq.add()$ takes $O(\log n)$. Thus, the cost of this for-loop is $O(m \log n)$.Lastly, the last for-loop will cost $O(n)$. Thus, the total running time is : $O(n \log n) + O(n \log n) + O(m \log n) = 2O(n \log n) + O(m \log n) = O((n + m) \log n)$.