Jaewoo Park
CSCI 2300 – HW1
Jan 24, 2020

1. I will use inductive proof to show the relationship between T(n) and n.
   1) When n=0, T(0) = 1.
   2) When n=1, T(1) = 2.
   3) When n=2, T(2) = 4.
   4) When n=3, T(3) = 8.
   (Inductive Case) Therefore, we can conclude that T(n) = 2^n.
2. 1) When n=0, T(0) = 0.
   2) When n=1, T(1) = 1.
   3) When n=2, T(2) = 3.
   4) When n=3, T(3) = 6.
   (Inductive case) Therefore, we can conclude that T(n) = n(n+1)/2.
3. (a) Each edge (E) of a graph has two ends, both being connected to either one vertex(V) as a looping edge or two separate vertices as a regular edge. And, each end of an edge contributes one degree to the entire graph. Therefore,
   $\Sigma_{u \in V} d(u)$ = the number of *ends* from all the edges = twice the number of E's = 2|E|.

   (b) In a given graph G=(V,E), let's call the even-degreed vertices V_{even} and odd-degreed vertices V_{odd}. We proved above that $\Sigma_{u \in V} d(u)$ = 2|E|, which is always an even number.
   And, $\Sigma_{u \in V} d(u) = \Sigma_{u \in Veven} d(u) + \Sigma_{u \in Vodd} d(u)$. Also, we know that $\Sigma_{u \in Veven} d(u)$ is also an even number. Therefore $\Sigma_{u \in Vodd} d(u)$, which is a number given from (even) – (even), should also be even.
   (c) Yes. If there were odd number of indegrees in a given graph, there should be the same number of outdegrees, since a pair of indegree and outdegree compose of a directed edge. If there were odd number of indegrees in a directed graph, there will be an equivalent odd number of outdegrees, and $\Sigma_{u \in V} d(u)$ = 2|E| still holds true in this case.
4. Algorithm:
   1. Run BFS($V_1 \cup V_2$, s) until you find the shortest distance to t (s->t).
   2. Check which edge e $\in E'$ was used during the BFS execution. There will be only one e used out of all the edges E'.

   Step 1. will incur some computation time that is less than [$|V_1| + |V_2| + |E_1| + |E_2| + |E'|$] because not every edge of E_1 and E_2 is necessarily visited during the execution of BFS. Step 2. will incur |E'| or O(1) computation time depending on the specific implementation of the algorithm. Therefore the execution of my algorithm is in linear time.

5. Let the nodes of the graph V_1 to be $V_1 = \{v_1, v_2, v_3, \ldots\}$. In order to see if a given graph is bipartite, I will choose an arbitrary node v_1 from the graph V_1. Then, I run BFS on the node, BFS(v_1). Once that is done, then I check the distances between v_1 and v_i ($i = 2,3, \ldots$). For instance, if distance between v_1->v_2 was 2, but distance from v_1->v_3 is 3 (using the red line from v_2 to v_3), then it implies that there might be an edge connecting between v_2 & v_3, or some edge connected to v_3. In order for v_1 to be able to get to other nodes in V_1, it must transit from V_2, which constrains it to take 2n steps in traversing between V_1 and V_2. So, if dist(v_1 -> v_i) is an odd number, this could mean that our graph is not bipartite.