CPSC 320 2018W2: Assignment 2

Please follow the guidelines given in Assignment 1 for submission to Gradescope, and for group collaboration. Remember to provide short justifications for your answers. Submit by the deadline **Friday February 1, 2019 at 10PM**.

If you are using the .tex file to prepare your answers, then for questions where you need to select a circle you can change `\fillinMCmath` to `\fillinMCmathsoln` for your choice of answer.

All of the problems in this assignment concern graphs. Throughout, unless otherwise specified, graphs have n nodes and m edges. A node's *degree* is the number of edges incident on it. Assume that graphs have neither self-loops (edges from a node to itself), nor multiple edges between the same two nodes.

A *path* of length k is a list of $k+1$ nodes, such that there is an edge of the graph between each consecutive pair of nodes. The *shortest path* between two nodes is a path with the minimum number of edges. A *simple path* repeats no node. A *cycle* is a path that starts and ends at the same node. A *simple cycle* repeats no node other than the starting/end node (which only appears twice).

1 Graph Data Structures

Suppose that the edges of undirected graph $G = ([1..n], E)$ are represented using *adjacency lists*: for each i, there is a (not necessarily sorted) linked list $A[i]$ of the nodes j such that $(i, j) \in E$. Suppose furthermore that for each i, the degree of i is also stored, and so can be obtained in $\Theta(1)$ time. With these assumptions, what is the worst-case running time of the most efficient algorithm you can think of to solve each of the following problems, using basic graph algorithms familiar to you?

1. Given two nodes i and j of G, determine if they are adjacent.

   ◉ $O(n)$ ○ $O(nm)$
   ○ $O(n \log n)$ ○ $O(n^2 m)$
   ○ $O(n^2)$ ○ $O(nm^2)$
   ○ $O(n + m)$ ○ None of these

2. Determine if G is a star. An undirected graph is a *star* if for some node i (called the star centre), $E = \{(i, j) \mid i \neq j, 1 \leq j \leq n\}$. For example, if $n = 4$ then the graph with edges $\{(1, 2), (2, 3), 2, 4)\}$ is a star with centre 2. (Recall that for undirected graphs, the order of pair of nodes in an edge does not matter.)

   ○ $O(n)$ ○ $O(nm)$
   ○ $O(n \log n)$ ○ $O(n^2 m)$
   ◉ $O(n^2)$ ○ $O(nm^2)$
   ○ $O(n + m)$ ○ None of these

3. Determine if G contains a triangle, that is, three nodes i, j, k such that i and j are adjacent, i and k are adjacent, and j and k are adjacent.

   ○ $O(n)$ ◉ $O(nm)$
   ○ $O(n \log n)$ ○ $O(n^2 m)$
   ○ $O(n^2)$ ○ $O(nm^2)$
   ○ $O(n + m)$ ○ None of these

4. Find a node i of G that minimizes $d(i) = \max_j\{dist(i,j)\}$, where $dist(i,j)$ is the length of the shortest path connecting i to j. (Again, it's not obvious for this problem that there might not be a better algorithm, but this seems to be the best possible with simple graph algorithm techniques.)

○ $O(n)$	○ $O(nm)$
○ $O(n \log n)$	○ $O(n^2 m)$
○ $O(n^2)$	○ $O(nm^2)$
○ $O(n+m)$	● $None of these$

2 Graph Properties

Each of the following problems presents a scenario about a graph and a statement about that scenario. For each one, indicate by filling in the appropriate circle whether:

- The statement is **ALWAYS** true, i.e., true in *every* graph matching the scenario.
- The statement is **SOMETIMES** true, i.e., true in some graph matching the scenario but false in another graph.
- The statement is **NEVER** true, i.e., true in *none* of the graphs matching the scenario.

1. **Scenario:** An undirected graph containing two simple paths that share no edges, each of length k for some integer $k \geq 2$. **Statement:** $n > k + 1$.
   - ● $ALWAYS$
   - ○ SOMETIMES
   - ○ NEVER
2. **Scenario:** A rooted tree found by running DFS from some node of a connected, undirected graph with $n \geq 2$. **Statement:** Every node in the tree has at most two children.
   - ● $ALWAYS$
   - ○ SOMETIMES
   - ○ NEVER
3. **Scenario:** A weakly-connected but **not** strongly-connected, directed graph with $n \geq 2$. **Statement:** A simple cycle of length $n + 1$ exists.

   SOLUTION
   - ○ ALWAYS
   - ○ SOMETIMES
   - ● $NEVER$

3 Graph Application: Tutorial Assignments

We give definitions of two problems here; your task is to reduce the first to the second. (After reading and understanding the two problem definitions, and before continuing on to read more, make some notes for yourself of what is needed to do a reduction and analyze its running time, and then what is needed to show that the reduction is correct. The rest of this problem breaks these parts down for you, but it's good to get practice at breaking the parts down for yourself too.)

- **Tutorial Assignment Problem (TAP)**. An instance of TAP is given by

- A list of tutorials $t_1, t_2, \ldots, t_k$ and their capacities $c_1, c_2, \ldots, c_k$, such that $\sum_{j=1}^{k} c_j = n$.
- A set S_i of the tutorials that student i can attend, where the total number of students equals the total capacity n.

A **valid** solution is an assignment of students to tutorials such that

- the total number of students assigned to tutorial t_j is at most c_j, $1 \leq j \leq k$,
- each student i is either unassigned or is assigned to a tutorial in S_i, $1 \leq i \leq n$, and
- no student is assigned to more than one tutorial.

We can write a valid solution as a set A of items of the form t_j:$\{i_1, i_2, ...\}$, where i_1, i_2, etc. are the students assigned to tutorial t_j. Not all students need be assigned to tutorials. A **good** solution is a valid solution that maximizes the number of assigned students.

- **Bipartite Matching Problem (BMP).** An instance of BMP is an undirected bipartite graph $G = (X, Y, E)$ where $E \subseteq X \times Y$. That is, the nodes of G are partitioned into two sets X and Y, and all edges of G connect a node of X with a node of Y. A **valid solution** is a matching, i.e., a subset M of E in which each node appears at most once. A **good** solution is a maximum matching.

1. Describe a reduction from TAP to BMP.

   Part (i): Transform an instance I of TAP to an instance I' of BMP.
   To transform the instance I into I' we need to create an undirected bipartite graph G. In this graph, one partition will be filled with n students nodes. The other partition will be n nodes that each one is the representation of a 'seat' in each tutorial, so for every tutorial t_j it will be c_j nodes that we will denote as $seat_{t_j x}, 1 <= x <= c_j$. In this way we ensure that there will be no more than c_j students assigned to the tutorial t_j. So the graph will have n students and n 'seats' of all the tutorials.

   The graph also need to have edges between students and 'seats', this edges will represent that a student i is capable of attend to a tutorial t_j. So to create this edges, we loop through the set S_i of each student i and for every tutorial t_j, that he can attend, we create an edge between i and every 'seat' node of the tutorial t_j in G.
   This process will give us the graph G that we can use to solve BMP.

   Part (ii): Transform a solution M' for instance I' of BMP to a solution A for instance I of TAP.
   The solution M' will be a set containing matches between students and 'seats' on tutorials. To transform M' in to the set A, that will contain a set of students for each tutorial t_j, we need to loop through M' and for each match $(i, seat_{t_j x})$ we will add i to the set t_j in A. This process will give us a valid solution for the instance I on the TAP problem.

2. Give the worst-case running time of part (i) of your reduction, as a big-O function of n.
   The worst-case running time, will be in the case that each student is capable of attending to all the tutorials. This will give us a big-O $O(n^2)$ because for each student i we need to add an edge to all the tutorial seats which in the worst case will be n seats for each i student.

3. Give the worst-case running time of part (ii) of your reduction, as a big-O function of n. We will need to loop through all of the matchings in the solution M' and for each one add each student to each corresponding tutorial, the length of M' will be at worst of length n. So the worst-case running time will be $O(n)$

4. There is a well-known algorithm for solving BMP with running time $O(nm)$. (We won't cover the algorithm, but it can be found in Chapter 7 of K&T's textbook.) Combining this algorithm with your reduction, what is worst-case running time of your TAP algorithm, as a function of n?
   On the worst case the instance generated instance I' will have $2n$ nodes and n^2 edges, so if the

algorithm that computes the solution for BMP in $O(nm)$ then the runtime of computing I' would be n^3, if we consider this in the reduction algorithm the whole running time will be $n^2 + n^3 + n$, that will give us $O(n^3)$.

5. Explain why your reduction is correct. Suppose that M'_{good} is a good solution (maximum matching) for I' and let A_{good} be the transformed solution (assignment) obtained by part (ii) of your reduction applied to M_{good}.

   (a) Show that A_{good} is a valid solution for I.
   The instance I' is a completely valid instance for BMP, because we created an undirected bipartite graph with the instance I for the TAP. Therefore M'_{good} must be a valid solution for BMP.
   Then M'_{good} must have a set of matchings between students and tutorial seats, in which each student i appearsat most once and every tutorial t_j is link to at most c_j students. So by visiting each matching in the set M'_{good} we can create a set A_{good} and for each student in M'_{good} we add it to the corresponding tutorial in A_{good}. Giving us at the end a valid solution with the instance I for TAP.

   (b) Show that A_{good} is a good solution for I, i.e., maximizes the number of assigned students.
   If M'_{good} is a good solution to BMP, meaning that it haves the maximum matching in I' which represents students and tutorials seats, then A_{good} will be a good solution for TAP, because it will have the most number of students matched for a tutorial, while still being a valid solution for TAP (proved on point 3.5.a).

4 Graph Application: Network Connectivity

(Adapted from Problem 9, Chapter 3 of K&T) Think of a communications network as a connected, undirected graph, where messages from one node s to another node t are sent along paths from s to t. Nodes can sometimes fail. If a node v fails then no messages can be sent along edges incident on v. A network is particularly vulnerable if failure of a single node v can cause two nodes, say s and t, to become disconnected. That is, when v and its incident edges are removed from the network, there is no path from s to t. We call such a node v a *vulnerability*.

1. Suppose that connected, undirected graph G contains two nodes s and t such that the shortest path between s and t is strictly greater than $n/2$. Show that G must contain a vulnerability.
   In order to have a path between two nodes (u and v) that is strictly greater than $n/2$ in G, there must be an articulation node w in the graph. We can see w as a bottleneck through which the path s must pass through in order to reach v from u there won't be another way to reach v from u. Because G is complete there is no other way for it to have a path greater than $n/2$, it must contain a bottleneck, which will increase the path length.

   So if w is disconnected from the graph it will create at least two separate connected components, in one of them will be u and in the other one v so the path s will no longer exists. For this application problem we consider w as a vulnerability because if it doesn't exists there would not be a way to reach v from u.

2. Give an algorithm that, given as input G and a node s such that there is some node t for which the distance between s and t is guaranteed to be greater than $n/2$, finds a vulnerability. Your algorithm should have running time $O(m + n)$.

   Algorithm *Find-Vulnerability* $(V, E, s, depth = 0)$
   // Initialize scores with the substring score of the first column and row
   mark s as visited

```
For all neighbours v of s:
    if v is not marked:
        if depth > n/2:
                return s
        return Find-Vulnerability(V, E, v, depth + 1)
return null
```

The algorithm above has a running time $O(m + n)$ because it simply does a BFS from the node s, keeping track of the depth and when the depth reaches a level greater than $n/2$ then the node that creates that path in that depth, must be a vulnerability.