COMP 251 - Fall 2017 - Assignment 2

Due: 11:59pm Oct 13th

General rules: In solving these questions you may consult your book; You can discuss high level ideas with each other, but each student must find and write his/her own solution. You should upload the pdf file (either typed, or a clear scan) of your solution to mycourses.

1. (10 points) How can the number of strongly connected components of a directed graph change if we add a new edge? Explain your answer.

Solution: A strongly connected component is defined as a maximal subgraph of a directed graph where every node are mutually reachable. Trivially, a node which is not mutually reachable when paired with any other node is a strongly connected component by itself.

By adding an edge to a directed graph, the number of strongly connected components can either stay unchanged or be diminished. If the new edge introduces a cycle between multiple strongly connected components, those components will merge into one. Else, the number of strongly connected components stays unchanged.

2. (10 points) A celebrity among a group of \( n \) people is a person who knows nobody but is known by everyone else. The task is to identify a celebrity by only asking questions to people of the form: “Do you know him/her?” Design an efficient algorithm to identify a celebrity or determine that the group has no such person. How many questions does your algorithm need in the worst case?

Solution:

Suppose we have 2 people A & B; and if we ask the question to A if he/she knows B. Then we can have 2 possibilities:

Yes, A knows B which implies A cannot be the celebrity

No, A doesn’t know B which implies B cannot be the celebrity

Therefore with every question we can eliminate one person from the group as not being a celebrity. So we ask \( n-1 \) questions to eliminate n-1 people from the group. For the last person, let’s call him C we need to check if he/she is indeed the celebrity. To do this we ask every person apart from C if they know C; (\( n-1 \) questions) and we also ask C if he/she knows anyone else; (n-1 questions). If C knows nobody and everyone knows C, then we declare C as celebrity else return no celebrity exists.

In the worst case, we would require \( (n-1) + (n-1) + (n-1) = 3n-3 \) questions. The algorithm is linear with respect to the number of people.

3. (20 points) Consider a list of \( n \) airports, and a list of \( m \) flights. The information of the \( k \)-th flight is given to us as a quadruple \((a_k, b_k, t_k, d_k)\), where \( a_k \) is the name of the origin airport, \( b_k \) is the destination airport, \( t_k \) is the time of departure, and \( d_k \) is the time of arrival.

Given four parameters \( a, b, s, t \), we want to see that starting at time \( s \) at airport \( a \), we can take a sequence of flights to get to \( b \) no later than time \( t \). Here we assume that transferring from one flight to the next takes no time.
Give an algorithm that solves this problem in $O(n + m \log m)$.

**Solution:** We can assume that there are no flights whose arrival times are later than $t$ as they are irrelevant to the problem and can be easily removed from the input. Next we sort all the flights by their arrival time, so that $d_1 \leq d_2 \leq \ldots \leq d_m$. This can be achieved in $O(m \log(m))$.

For each airport $x$ we will keep a number $D[x]$ that is going to be the earliest time we can get to that airport.

We will then iterate through all flights and see if we could take them or not. Table initialization is done in $O(n)$. See algorithm 1 for details.

**Algorithm 1** Simple solution to problem 3

```plaintext
1: procedure ISAIRPORTREACHABLE(Flights, a,b,s,t)
2:    initialize $D[x] = \infty$ for every airport $x$
3:    $D[a] = s$
4:    for $k = 1, \ldots, m$ consider the flight $(a_k, b_k, t_k, d_k)$ do
5:        if $D[a_k] < t_k$ and $D[b_k] = \infty$ then $D[b_k] = d_k$.
6:    endfor
7:    if $D[b] \leq t$ then
8:        return “It is possible to reach $b$ before time $t”
9:    else
10:       return “Impossible to reach $b$ before time $t”
```

**Comment** : For this problem, other algorithms are also possibles.

4. (20 points) Suppose that a black stone is placed on a vertex $s$ and a white stone is placed on a different node $t$ of an undirected graph $G$, where there is no edge between $s$ and $t$. At every step, we have to move both stones simultaneously to two non-adjacent (different) vertices (the stones can visit a node several times during the algorithm if needed but they have to move at every step). A stone can only be moved from a node to a neighbouring node. The ultimate goal is to switch the places of the two stones. That is to have the black stone on $t$ and the white stone on $s$. Design an algorithm that takes as input $G$ and $s$ and $t$, and tells whether this is possible, and if it is, then what is the minimum number of steps required to achieve this.

**Solution:** For the graph $G$, we will construct a new graph $G'(V, E)$. The vertices of $G'$ are labelled as tuples $(x, y)$ where $x$ and $y$ are vertices of $G$. Two vertices $(x, y)$ and $(x', y')$ are joined by an edge in $G'$ if and only if it is a legal move to move the white stone to from vertex $x$ to $x'$ and the black stone from the vertex $y$ to $y'$ in $G$. Once we construct the new graph $G'$, we need to run BFS from the vertex labelled $(s, t)$ and check if we reach the target state $(t, s)$. If we reach $(t, s)$ in our traversal we output the depth of BFS traversal as the minimum number of moves required to switch the stones. If the target state $(t, s)$ is not reachable then we return that it is not possible to switch the stones in the given graph $G$.

5. (20 points) The distance between two nodes in an undirected graph $G$ is the number of edges in the shortest path between them. The diameter of a graph $G$ is the maximum distance between any pair of nodes in the graph. Give an $O(n)$ algorithm that computes the diameter of a tree $T$ on $n$ nodes. Analyze the running time of your algorithm.

**Solution 1:** Let $u$ be any node in the tree, and let $v$ be the furthest node from $u$.

Suppose the path from $u$ to $v$ shares a node with a path of maximum length. The furthest nodes from this shared node must be endpoints of paths of maximum length, as otherwise we could extend the path of maximum length. It follows that $u$ or $v$ is an endpoint of a path of maximum length.
Suppose the path from \( u \) to \( v \) does not share any nodes with any paths of maximum length, then we would like to show a contradiction. Select any path of maximum length and let \( a \) and \( b \) be the endpoints of this path.

Let \( w \) be the node on the path between \( u \) and \( v \) and \( c \) be the node on the path between \( a \) and \( b \) where the path between \( w \) and \( c \) is the shortest path between any pair of nodes on the two paths.

As \( v \) is the furthest node from \( u \) we have
\[
\text{dist}(u, b) \leq \text{dist}(u, v)
\]

We expand the inequality by considering the paths
\[
\text{dist}(u, w) + \text{dist}(w, c) + \text{dist}(c, b) \leq \text{dist}(u, w) + \text{dist}(w, v)
\]
\[
\text{dist}(w, c) + \text{dist}(c, b) \leq \text{dist}(w, v)
\]

Then since the paths are disjoint, we must have \( w \neq c \), and \( \text{dist}(w, c) \geq 1 \) and we can find a contradiction by constructing a path longer than the diameter.
\[
\text{dist}(c, b) < \text{dist}(c, w) + \text{dist}(w, v)
\]
\[
\text{dist}(a, c) + \text{dist}(c, b) < \text{dist}(a, c) + \text{dist}(c, w) + \text{dist}(w, v)
\]
\[
\text{dist}(a, b) < \text{dist}(a, v)
\]

This means we can find an endpoint of the diameter by finding the furthest node from any node in the tree via BFS. We compute the diameter by running BFS a second time starting from the endpoint.

BFS has a running time of \( O(n + m) \) where \( m = n - 1 \) for trees. As we only run BFS twice, we find the running time to be \( O(n) \).

**Solution 2:** This problem can be solved recursively. First we can pick an arbitrary vertex \( r \) as a root and run a BFS to construct a rooted tree where every node than \( r \) have a parent and possibly some children. Now note that the diameter of the subtree rooted at a vertex \( v \) can be computed in the following way: If the children of \( v \) are \( u_1, \ldots, u_k \), then either the diameter is the maximum of the diameters of the subtrees rooted at \( u_1, u_2, \ldots, u_k \), or it is 2 plus the sum of the two largest depths of the subtrees rooted at \( u_1, u_2, \ldots, u_k \) respectively. Hence recursively we can return two parameters: depth and the diameter of the subtree rooted at a node \( v \). They both can be computed recursively from the values on the children.

**Comment:** While a formal proof was not required, solutions without any form of justification were penalized.

6. (20 points) Consider a directed graph \( G \) with \( n \) vertices and \( m \) edges, where every node is labeled with a unique number from the set \( \{1, \ldots, n\} \). For every node \( u \), let \( \text{min}(u) \) be the smallest label among all the vertices that can be reached from \( u \). Give an \( O(m + n) \) algorithm that computes \( \text{min}(u) \) for all the vertices of \( G \).

**Solution:** This can be done by a modification of the DFS algorithm. If you consider the DFS tree, then the set of reachable vertices of a node is itself together with the set of the vertices that can be reached through its children. It follows that the smallest label of a node will be the minimum of its own label and the smallest label of its children. This can be done recursively.

For every node \( u \), we initialize \( \text{min}(u) \) as \( u \). Call \( \text{minDFS} \) on any unvisited node, until all nodes are visited.

Since this is a simple modification to DFS, the running time will remain \( O(n + m) \).
Algorithm 2

1: procedure minDFS(node \( v \))
2:   for each neighbour \( u \) of \( v \) do
3:     if \( u \) is not visited then
4:       Mark \( u \) as visited
5:     \( \min(u) := \text{minDFS}(u) \)
6:     if \( \min(u) \leq \min(v) \) then
7:       \( \min(v) := \min(u) \)
8:       return \( \min(v) \)