CS 170

1. Study Group

None

2. True Source

Algorithm Description:
First we derive an algorithm only works for directed acyclic graph: count the number of vertex whose in degree equals 0, if there is only one, then there is a vertex v from which every other vertex can be reached. Otherwise, the answer is no. For a more general case, we first find the strong connected component(SCC) of the graph, then see each SCC as a vertex. In this way, we derive a new graph which is acyclic. Then we can run the same algorithm on this new graph.
Correctness:
For the acyclic graph case, we claim that :
there is only one vertex whose in degree equals 0$\Longleftrightarrow$there exists one vertex which every other vertex can be reached
$\Rightarrow$:
Run DFS from the zero-in-degree vertex v, we can derive a DFS tree (since graph is acyclic). Since v is the root the tree, it is obvious that v can reach every other vertex in the graph.$\blacksquare$
$\Leftarrow$:
Since the graph is acyclic, there is at least one vertex whose in degree equals 0. Suppose there are more than one vertex whose in degree equals 0, call them u, v, since only u can reach itself, so if there exists a so-called true source, it can only be u, but u can not reach v, which contradicts.$\blacksquare$
For the general case, if there exists a true source in that derived new graph, it can reach every other vertex in the new graph, remember that each vertex in the new graph represents a SCC, where each vertex can reach each other freely. So the general algorithm also works.
Runtime Analysis:
First the algorithm to find SCCs runs in O($|V| + |E|$), then counting the number of vertex whose in degree equals 0 runs in O($|E|$). So the total cost is O($|V| + |E|$).

3. Finding Clusters

Algorithm Description:
During the DFS process, after vertex i has explored all of its child vertices, it can determine $m(i)$ as the smallest $m(j)$, where $j \in i's$child vertices $\cup\{i\}$. The pseudocode is as follows:

```
function Find_Clusters():
    $m$ := [$i$ for $i$ in range(1, $n+1$)]
    $marked$ := $[0] * n$
    function Explore($v$):
        $marked[v] = 1$
        for $child$ in $V[v]$:
            if not $marked[v]$:
                Explore($child$)
            $m[v] = min(m[v], m[child])$
    for $i$ in range(1, $n+1$):
        if not $marked[v]$:
            Explore($i$)
    return $m$
```

Correctness:
It is obvious that $m(i)$ is $\min_j m(j)$, where $j \in i's$child vertices $\cup\{i\}$. The base case is that vertex i has no child, then $m(i) = i$. After one vertex has explored all of its children, it has all the information to calculate $m(i)$.

Runtime Analysis:
The algorithm only needs one pass of DFS, so the runtime is $\mathcal{O}(|V| + |E|)$.

4. All Roads Lead to Rome

(a) **Algorithm Description:**
First we run Dijkstra's algorithm on G from v_0 to get $disto$ array and $prev1$ array. Then we run Dijkstra's algorithm on G^{inv} from v_0 to get $disfrom$ array and $prev2$ array. Finally, for each pair (u, v), the shortest path from u to v which passes v_0 equals $disfrom[u] + disto[v]$.
Correctness:
It is obvious that $disto[v]$ gives the shortest path from v_0 to v, and $disfrom[v]$ gives the shortest path from v to v_0. Then the shortest path from u to v which passes through v_0 equals $disfrom[u] + disto[v]$.
Runtime Analysis
Two Dijkstra cost time $O(|V| + |E|)log(|V|)$.

(b) For the *distancequery*, we only need the two array $disto$ and $disfrom$, which contains $|V|$ vertices each. For a query (u, v), the answer is $disfrom[u] + disto[v]$, which can be computed in O(1).

(c) For the *pathquery*, we only need $prev1$ and $prev2$ array. The way to compute the answer is straight forward.

5. The Greatest Roads in America

Algorithm Description:
Get $k+1$ copies of the vertex of graph G, call them $G_0, G_1, G_2, ..., G_k$. If the traveler is in G_i, he has visited i "amazing" roads. Then for each road (u, v) in G, if it is a normal road, then connect u_i and v_i in each G_i with weight $d(u, v)$, if it is a "amazing" road, then connect u_i and v_{i+1} with weight $d(u, v)$, $i \in \{0, 1, ..., k-1\}$, also connect u_k and v_k with weight $d(u, v)$.
Now we get a new graph G', we run Dijkstra's algorithm from h_0 on G' and find the shortest path from h_0 to h_k. This distance is the length of the shortest path that hits at least k of these amazing roads.
Correctness:
From the construction of G', it is straightforward to see that each path in G' from h_0 to h_k has a one-on-one relation to a path in G which hits at least k amazing roads and starts and ends at h.
Runtime Analysis:
O$(k(n+m)log(km))$.

6. Vertex Cut

Proof:
Define $S_i = \{j | distance(u, j) = i\}$, i.e. all the vertices whose distance from u equals i.
Since $distance(u, v) = d$, we know $v \in S_d$. So to split u, v into two cut, we only need to remove one of S_i, where $i \in \{1, 2, ..., d-1\}$.
Let $s = \min_{1 \leq i \leq d-1} |S_i|$, we have $n - 2 \geq \sum_{i=1}^{d-1} |S_i| \geq (d-1)s$
So $s \leq \frac{n-2}{d-1}$. ■