Theory Assignment-4: ADA Winter-2024

Himanshu Raj (2022216) Tanish Verma (2022532)

March 28, 2024

1 Preprocessing

Given a directed acyclic graph G = (V, E) and two vertices s and t, |V|=n and |E|=m.
Initialize a boolean array isCutVertex of size n with all values set to false. This array contains the result of the algorithm where the vertices marked as true are the output or the s-t cut vertices.
We will follow 1-based indexing.

2 Algorithm Description

1. Perform Topological Sort using DFS (as done in lectures).
If after sort t does not appear after s, this means that there is no path from s to t, simply return.
Else copy the sorted vertices to an array named TopoArr such that first vertex is s and last vertex is t, i.e. copy a subarray from s to t from the original sorted result so that first vertex is s and last vertex is t.
TopoArr contains a path from s to t, so if there exists any cut vertex it should be one of these vertices except s and t. This statement holds true due to the graph being directed and acyclic.

2. Now we need to check whether every vertex in TopoArr is a cut vertex.
A vertex v in TopoArr is a cut vertex iff there doesn't exist an edge u -> w, where u occurs before v and w occurs after v in TopoArr. This means that removing v will disconnect the path between s and t as there will be no other way to connect vertices before and after v.
Initialize two integer arrays InDegree and OutDegree of size of TopoArr, with all values set as number of inward edges from vertices in TopoArr to a given vertex in InDegree and number of outward edges to vertices in TopoArr from a given vertex in OutDegree. For this, take a vertex from TopoArr and check remaining vertices in TopoArr for inward and outward edge from adjacency matrix and set values accordingly.

3. To check the above condition, we initialize a counter with out-degree of s.
Then we iterate over sorted vertices in TopoArr except(s and t), i.e. 2 to |TopoArr|-1 where |TopoArr| means size of TopoArr.
In each iteration for vertex v, decreament the counter by InDegree[v]. If counter equals 0, this means v is a cut vertex, set isCutVertex[v] to true. Then increament the counter by OutDegree[v] and proceed to next iteration.

3 Proof of Correctness

The proof of correctness involves proving that all s-t cut vertices are present in TopoArr (step 1 according to description) and the FindCutVertex function (step 3 according to description) works correctly.

The topological sort on a directed acyclic graph returns vertices in an order, where if there exists an edge u->v then u will occur before (not necessary immediate before) v in the topological sort.
Let u be any vertex before s which has an edge to t, for u to be a cut vertex there should exist an edge s->u which will then form a path outside vertices of TopoArr, but this cannot happen since u occurs before s there is no edge s->u, otherwise it contradicts property of topological sort.
Let u be any vertex after t which has an edge from s, for u to be a cut vertex there should exist an edge u->t which will then form a path outside vertices of TopoArr, but this cannot happen since u occurs before t there is no edge u->t, otherwise it contradicts property of topological sort.

In any iteration in FindCutVertex for vertex v, if counter equals 0 this means that the remaining outward edges of vertices before v (which were not balanced by an inward edge) are balanced by inward edges of v, signifying all vertices

before v have no edge with a vertex after v in the TopoArr. This implies that all of the possible paths from s to t must pass through v and hence v is a s-t cut vertex.

4 Pseudocode

```
1: function FindCutVertex(TopoArr, InDegree, OutDegree, s, t, isCutVertex)
2:     counter = OutDegree[s]
3:     for (i : 2 to |TopoArr|-1):
4:         v = TopoArr[i]
5:         counter = counter - InDegree[v]
6:         if(counter == 0):
7:             isCutVertex[v] = true
8:             print(v)
9:         counter = counter + OutDegree[v]
10: end function
```

5 Complexity Analysis

5.1 Time complexity

Initializing isCutVertex are linear time operations, i.e. O(n). Topological sort using DFS takes O(n+m) time as each vertex and edge is checked once. Copying a subarray of above result takes O(n) time in worst case. Initializing InDegree and OutDegree is also $O(n^2)$ in worst case. Checking whether every vertex is a cut-vertex involves iterating TopoArr once resulting in O(n) time operation.
So the overall run time complexity of the algorithm is O(max (n^2, n+m)).

5.2 Space complexity

The algorithm uses two additional arrays of size n to solve the problem and one array to return the result.
So the overall space complexity of the algorithm is O(n).