Design and Analysis of Algorithms : CS345A
Indian Institute of Technology Kanpur

Worked out Assignment on *DFS Traversal in Directed Graphs*

An Important guideline

- It is only through the assignments that one learns the most about the algorithms and data structures. For the current assignment, a sequence of hints are given. Try to look at a hint only after trying on your own for sufficiently long period of time.
- Ideally, one should look at one hint per day and then spend around 20-25 minutes to materialize the hint. The best way - go for a stroll after dinner pondering over the hint(s).
- You should judge your performance by the number of hints you indeed use – more hints you use, less is your performance.
- The onus of learning from a course lies first on you. So act wisely while working on this assignment.

An $O(n^2)$ time algorithm for the unique-paths problem on directed graphs.

Let $G = (V, E)$ be a directed graph on $n = |V|$ vertices and $m = |E|$ edges. As discussed in the class, G is said to be a unique-paths graph if for each $u, v \in V$, there is **at most** one path from u to v. We discussed an $O(mn)$ time algorithm for this problem in the 14th Lecture. The aim of this worked-out assignment is to design an $O(n^2)$ time algorithm for this problem. The biggest hint is the following:

You need to carefully study the $O(mn)$ time algorithm discussed in the class. This is because the $O(n^2)$ time algorithm is just a slightly *refined* version of this algorithm only. In particular, try to find the source of the redundant computation we are doing in the algorithm discussed in the class and then try to avoid it.

So try for sometime on your own, sincerely, patiently, and persistently. If you still do not succeed, do not feel discouraged. Go to the following page for more explicit hints.

In the $O(mn)$ time algorithm, we are doing n DFS traversals - one from each vertex. The aim of doing DFS traversal from a vertex, say u, is to determine whether there are multiple paths from u to any other vertex. Recall that if we encounter any forward edge or cross edge, we can stop immediately since that would imply that there are multiple paths from u. However, we totally *ignore* the backward edges because no backward edge during DFS(u) can be part of any path from u. Processing of these backward edges accounts for the $O(m)$ running time for the DFS procedure we carry out for a vertex u. Our aim is to improve it to $O(n)$ running time.

Observe that the presence of some backward edges encountered during DFS(u) may potentially be the source of multiple paths from a vertex other than u to some other vertex. It was illustrated in slide 18 of Lecture 14.

So how should we handle the backward edges ?

Ponder over it before going to the next page.

Some of you might now start thinking of ways to process the backward edges during DFS(u) with the aim to detect multiple paths from a vertex other than u to some other vertex. There are many combinations of backward edges that may lead to such multiple paths.

1. A pair of backward edges from a single vertex.
2. A pair of backward edges whose (source, destination) intervals overlap.
3. A pair of backward egdes from a subtree rooted at v that go to ancestors of v.
4. ...

It seems to require quite intensive analysis to find out if there is any pair of backward edges during DFS(u) that can lead to multiple paths from a vertex other than u.

But wait! We are distracting from our objective. Recall we just wish to speed up the processing carried out during DFS(u) from $O(m)$ to $O(n)$. We can accomplish this objective without carrying out the intensive analysis sketched above to design a complex algorithm for the problem.

Try to realize the message conveyed in the paragraph mentioned above. Think with calm mind ...

Go to the next page after sufficient time. You will be so amazed to see the simple algorithm that you will surely say ...

"Oh! it was just there. Why didn't I think of it ?".

Recall our aim is to avoid processing of *too many* backward edges that we might encounter during DFS(u).

Let $T(u)$ denote the DFS tree rooted at u. If there are two or more backward edges from any vertex v in $T(u)$, then there are multiple paths from v to some vertex (the destination of that back edge which is nearer to v).

So what if we stop as soon as we encounter 2 back edges emanating from any vertex during DFS(u) ?

Surely, we will not encounter more than $n-1$ backward edges during this *new* DFS(u) procedure. The running time of this procedure for u will be $O(n)$. And hence, the time complexity of the entire algorithm will be $O(n^2)$.

Are we sure that we are done ?

Indeed !

Think over it and get amazed by the simplicity and elegance of the $O(n^2)$ time algorithm.

Those whose interest is beyond A^* should go to the next page.

To the best of the knowledge of the instructor, $O(n^2)$ is the best bound known till date for the unique-path problem. Think of designing a faster, may be, $O(m+n)$ time algorithm for this problem.

A natural approach here would be to do DFS traversal in the graph and work on the resulting forest of DFS trees. This will take $O(m+n)$ time. But, the following is a hurdle.

There might be cross edges going from one DFS tree in the forest to another DFS tree. While some of them may be the source of multiple paths from a vertex to another vertex, it is not always the case that multiple cross edges lead to multiple paths from a vertex to another. You may draw simple example to verify this claim.

So, there is no obvious way to design an $O(m+n)$ time algorithm for this problem...