Analysis of Algorithms and Data Structures: Problem set 8

Giosue Castellano

Problem 5

To answer this problem, let us start with a key observation for it in the context of DAGs.

Observation: If the graph is a DAG, the problem of finding the longest walk simply becomes a problem of finding the longest path.

Why? Consider any node v. Then, since the graph is acyclic, once we leave v we may never return to it and explore different subgraphs. This means that we can never "turn back" in the graph, so the longest (not necessary simple) walk will be the longest path in the DAG. We already know how to solve this problem in linear time, using topological sort ($O(n+m)$) and then building a dp-array which records the longest path starting from a given node ($O(n)$).

The problem is that we do not know if the graph is acyclic and if there is a cycle, simply finding the longest path is not a correct solution since we may explore more than 1 subgraph for a given node v.

To make it work in the general case, we can consider strongly connected components (SCC). They are very useful here, since as we have seen in class, the metagraph of all SCCs is a DAG and constructing them takes linear time $O(n+m)$. To find a correct solution we follow these steps:

- Find the strongly connected components of the graph and construct a metagraph where every node stores the number of nodes in its strongly connected component ($size(node)$). $O(n+m)$.
- Use topological sort to find the topological order in the metagraph. $O(n+m)$.
- Reverse the topological order. $O(n)$.
- Create a dp-array of size n. For every $i \in \{1, ..., n\}$ initialize $dp[i]$ as $size[i]$. This takes care of the fact that if there were no edges in the metagraph, we still return the correct solution. Loop through the reversed topological order of the metagraph (so starting from the leaves) and update the dp array. Say you are currently at node i, then the update will be $dp[i] = max_{j \in Nbr(i)}(dp[j] + size(i))$. The update step is a modification of finding the longest path, since it does not increment the count by 1, but rather by all elements in the SCC of i. This takes $O(n)$.
- The answer is the maximum of the dp-array $max(dp[n])$.

This procedure respects the required time complexity of $O(n+m)$ and finds the longest not necessary simple walk in a directed graph.