Summary of
Edge-Disjoint Spanning Trees and Depth First Search by Robert Endre Tarjan

Nayan Deshmukh(14418)

October 24, 2016

This paper presents an algorithm for finding two edge-disjoint spanning trees
rooted at a fixed vertex of a directed graph. The algorithm uses depth first
search and an efficient method for computing disjoint set unions. It requires
\( O(m\alpha(m,n)) \) time and \( O(m) \) space to analyze a graph with \( n \) vertices and \( e \)
edges, where \( \alpha(m,n) \) is a very slowly growing function related to a functional
inverse of Ackermann’s function.

The author first presents a trivial algorithm to solve the problem. It first
calculates a tree \( T_1 \) rooted at \( r \) using DFS and then finds another tree \( T_2 \) in
\( G-T_1 \), then starts growing \( T_2 \) by swapping edges from \( T_1 \) to \( T_2 \). We keep
on growing \( T_2 \) until it becomes a spanning tree. This algorithm has a time
complexity \( O(mn) \) which can be further improved using data structures which
results in \( O(n^2) \) algorithm, but DFS gives an even faster algorithm.

The algorithm can be divided into steps:-

- **Step 1:** Perform a depth-first search of the problem graph. Determine
  LCA \((v,w)\) for all edges \((v,w)\). Time: \( O(m\alpha(m,n)) \).
- **Step 2:** Compute edges which will be in both the spanning tress called
  bridges. Duplicate all bridges. Time: \( O(m\alpha(m,n)) \).
- **Step 3:** Find paths needed for spanning tree construction. Time: \( O(m) \).
- **Step 4:** Build spanning trees using fastspan2. Time: \( O(n) \).

The method requires \( O(m\alpha(m,n)) \) total time and \( O(m) \) storage space.
This paper has presented a simple \( O(nm) \) algorithm and a more sophisticated
\( O(m\alpha(m,n)) \) algorithm for finding two spanning trees with fewest common
edges in a directed graph. Though the \( O(m\alpha(m,n)) \) algorithm uses some power-
ful techniques, it would be quite easy to program. Computational experience
with similar algorithms suggests that the \( O(m\alpha(m,n)) \) algorithm would be com-
petitive with the simple algorithm for small-to-medium-size problems (0–100
vertices) and much faster for large problems (0–1000 vertices). Both algorithms
can be generalized to find two minimally intersecting spanning trees with pos-
sibly different roots.