Wendy Wang (www2105)
Raymond Li (rwl2117)

Parallelizing Topological Sort in a Dependency Graph

Problem: Dependency Graph Ordering

The topological sort algorithm performed on a directed acyclic graph, where V is the set of vertices and E is the set of directed edges, produces a linear ordering of vertices such that, for every $e_{ab} \in E$ pointing from vertex *a* to vertex *b*, vertex *a* appears before vertex *b* in the ordering. We can say that vertex *b* "depends on" vertex *a* because *b* must come after *a* is completed or visited.

We use topological sort to resolve a dependency graph and find a valid ordering of nodes. We will incorporate monadic components to allow us to efficiently and succinctly apply the functions we need to the nodes in the graph.

Parallelization Component:

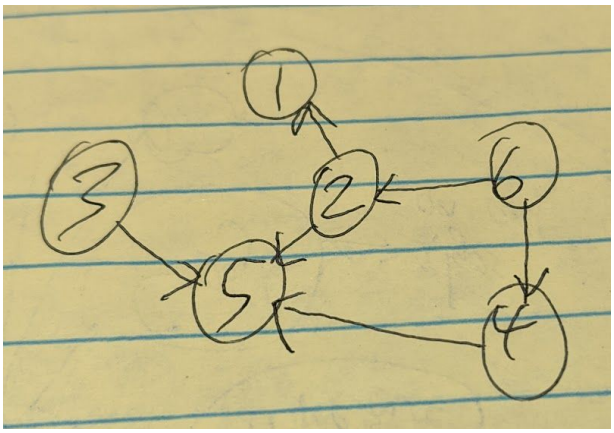
Our topological sort algorithm will be parallelized by starting a separate topological sort from every node of in-degree 0. This will allow the algorithm to determine a valid topological ordering for each subtree. The alternative parallelization tactic we considered involves first determining each disjoint tree; the preprocessing necessitated, however, would likely take as long as a non-parallel topological sort.

Expected Result:

Each parallelized run will give us the resulting topologically-sorted list of nodes such that each node in the list comes before nodes that depend on it. Then, the entire set of input vertices and arcs, possibly representing several disjoint graphs, will be merged in order such that the feasible topological order is still preserved in the final list of all vertices. This is possible if we perform the merge by iterating from the back of the list and inserting nodes while ignoring duplicates, so that all vertices are inserted in a position in front of their dependent vertices.

Example:

The input.txt file will have an integer *n* in its first line representing *n* nodes in the graph (we assume the nodes span the range from 1 to *n*, inclusive). The next *n* lines will have two integers, a and b, separated by a space. This represents $e_{ab} \in E$ pointing from vertex *a* to vertex *b*.



input.txt:

6

3 5

2 1

2 5

6 2

6 4

4 5

results from parallel computations:

[3, 5]

[6, 2, 4, 1, 5]

output (after merging lists):

[3, 6, 2, 4, 1, 5]