Q 3.1)

When a graph is not acyclic, meaning there’s at least one (directed) cycle. Since some calculations depend on the result of others, if there’s a cycle, meaning that a calculation needs to wait for calculation to be computed but is also waiting for to be computed. Causing it to be impossible to compute any of them because they are forever waiting for each other.

So, it is impossible to perform all calculations in a cyclic graph (it is impossible to compute a group of calculations in a cycle).

Q 3.2)

Assuming the graph is an adjacency list. First having , a set of computations that depends on, and , the amount of time vertex () takes to collate the result of computations that depends on, all initialize as 0. By iterating through all edges (all dependencies represent as ), adding the amount of into representing , and adding index into representing , both take constant time.

Then adding a sink vertex into the graph and adding edges from all vertices with no outgoing edges to this sink vertex (no other calculations depend on these calculations). This represents the time requirement at the end which need to wait for the last among those vertices with no outgoing edges to finish computed. Set to be 0 since this sink vertex isn’t part of the calculations.

**Subproblem:** for all , let be the problem of determining , the minimum amount of time required to finish performing calculations up to on the parallel computer.

**Recurrence:** for all ,

itself takes seconds to be computed (include collating the results of ), however, it still needs to wait until all are computed, before could start being computed.

By using the parallel computer to compute all those calculations depends on at the same time in order to get the minimum amount of time require to perform up to . Then only the maximum time require among those calculations is the time require for to wait (since all other calculations depends on would already be done computed because they take less time).

Since depends on all for vertices with outgoing edges to , so we need to solve for each such before solving . This could be done by solving the vertices (calculations to and the sink vertex) in topological order, from left to right. All edges point from left to right, so any vertex with an outgoing edge to is solved before is.

And the sink vertex () itself is not a calculation that need to be performed.

**Base cases:** for all that then .

Interpreting those calculations which depend on no others need to be computed first. Since we are using DP in topological order, those who are on the left most need to be computed first, so that those who are waiting for them to be computed could be computed next. This takes at most time for iterating through vertices.

(Sink vertex), is the minimum amount of time required to perform all calculations on the parallel computer, because we need to wait until the last calculation is computed, and since all of its left vertices from topological order need to be done computing first.

Overall time complexity is . Iterating through edges in time. By using topological sort in time having vertices (calculations) and edges. Then adding a sink vertex and edges in times for at most edges from calculations (vertices). Initializing base cases in time. For the recurrence, each edge (dependency) is only considered once (at its end point) costing time (tighter bound). Costing in total.

Q 3.3)

Setting up , sink node and do a topological sort just like in Q3.2 (with the same reasons) in time.

**Subproblems:** for all and , let be the problem of determining , the minimum amount of time require to finish performing calculations up to using the supercomputer for at most calculations, and the parallel computer for all other calculations.

**Recurrence:** for and ,

This is exactly like Q3.2 for the case of not using any super computers.

Interpret choosing a minimal choice between using super computer at current calculation or had used in the past of current one’s dependencies.

And the sink vertex () itself is not a calculation that need to be performed.

Since depends on all for vertices with outgoing edges to , so we need to solve for each such before solving . This could be done by solving the vertices (calculations to and the sink vertex) in topological order, from left to right. All edges point from left to right, so any vertex with an outgoing edge to is solved before is. Then increasing .

**Base cases:** for all that then .

With same reasons as Q3.2.

(Sink vertex), is the minimum amount of time required to perform all calculations on the parallel computer using at most times of super computer, because we need to wait until the last calculation is computed, and since all of its left vertices from topological order need to be done computing first.

Overall time complexity is . Taking for setting up, and using similar logic as Q3.2 but repeating it times causing it to be . in total.