# COMP3121 Assignment 3

## Question 3

3.1 Suppose we have a cycle $(1,2,..,k)$ in $G$, suggesting that we must finish task 1 before doing 2, and finish 2 before doing 3, …, and finish k before doing 1. We see that is impossible since we can start from nowhere (we must do finish task 1 before starting task 1, contradiction.) Thus, only if $G$ is acyclic could we do this job.

3.2 We can use topological sort to finish this job. Since $G$ is acyclic (as we showed in 3.1), we may find a topological sort, so we need create a linear ordering of the vertices of a directed acyclic graph such that each vertex represents a calculation, and each edge represents a dependency between calculations.

Hence this gives a proper order of doing the jobs. Note that we can always do multiple tasks at a time, so for each task in the order, we may start doing it once we reach the corresponding position in the ordering we find.

To find the global minimum time cost, we first compute the $t_i$ of each task $i$ using the equation given in the problem. This would cost $O(m+n)$ time. Then we may use a min-heap to keep track of the finishing time of the tasks that are currently running. Each time we have finished a task with the earliest finishing time, we can look at the sequence of tasks and see if we can add more tasks to the heap (since their pred tasks are finished). When we finally saturate all tasks, the current time is the minimum time we need to finish all tasks, and our time cost is $O(m+n)$.

3.3 For this question, we can modify the algorithm from question 3.2. First, we have at most $s$ calculation, after we use the supercomputer then we reduce the value of $s$. When $s$ reduce to $0$ we mark it as an ineligible calculation, if the supercomputer has not yet been used s times and all the dependencies required for calculation $i$ have already been computed, we can use the supercomputer to compile the results and store them in parallel with any calculations that depend on it.
We keep track of how many times the supercomputer has been used by decrementing the count after each use. If the supercomputer has not been used $s$ times and at least one dependent calculation has not been computed, then we use the parallel computer to perform the computation and simultaneously save the result and any calculations that rely on it.
Therefore, we iterate over the sorted order requires $O(n+m)$ time, and we can use the supercomputer to execute up to $s$ computations, which takes $O(s(n+m))$ time in total.