GIVEN:
An arbitrary workflow, i.e. directed acyclic graph (DAG) G=(V, E), where each node v is a job and each directed edge (u, v) shows that the output data of job u is transferred as the input data of job v, K homogeneous machines, the execution time t(v) of each job running individually on a single machine, and the communication time t(u, v) of each data transfer between jobs u and v running in different machines.

QUESTION:
Find the minimum execution time of the entire workflow along with a feasible schedule mapping all the jobs onto no more than K machines

REQUIREMENT:
Comment on the difficulty (i.e., computational complexity) of the above problem, design an efficient algorithm, implement the solution, (which may be an exact, approximation, or heuristic algorithm), and show the execution results. (C/C++ in Linux and Makefile are preferred; C/C++/Java is required.)

I do not expect a perfect solution from you. Based on your source code and algorithm description, I will evaluate
1) whether you have a correct, clear idea (algorithm) to solve this optimization problem.
2) whether your implementation is correct in source files.
3) your code readability (such as code format and necessary comments), algorithm efficiency, and self-learning capacity.

REPORT:

Algorithm Overview
- Represents workflow as a directed weighted graph
- Vertices are jobs, edges are dependencies
- Edge weights represent data transfer times

Steps to solve the algorithm:

1. **Find the order of job sequencing:**
   The algorithm employs topological sorting to determine the order of job execution. It ensures that a job is scheduled only after its dependencies have been completed. The topological sorting is performed using a modified Breadth-First Search (BFS) approach.

2. **Schedule the Jobs:**
   The algorithm schedules jobs onto machines based on their dependencies and execution times. It calculates the earliest available machine for a job, considering the completion times of its dependencies. The scheduling is done in a way that minimizes the makespan, i.e., the total execution time of the entire workflow.

3. **Dependency Consideration:**
   The algorithm appropriately considers dependencies when scheduling jobs. It calculates the finish time of each job based on the completion times of its dependent jobs and communication times.

Analysis time complexity:
1. Topological sorting of the jobs:

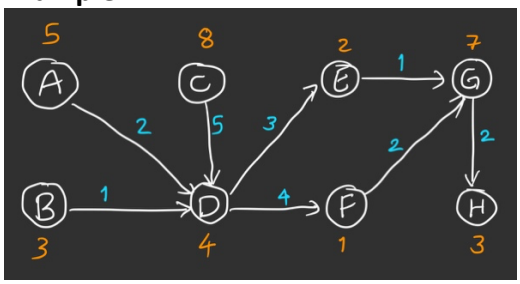
- This can be done in O(V+E) time, where V is the number of jobs (vertices) and E is the number of dependencies (edges).

2. Scheduling each job:
   - Find predecessors: O(V*E)
   - Finding the earliest machine takes O(K) time, where K is the number of machines.
   - Find and update finish time based on dependencies take maximum O(V)
   - Overall time complexity for scheduling each job is O(V*E + K + V) = O(V*E + K)

3. The overall algorithm does the following:
   - Topological sort: O(V+E)
   - Schedule V jobs: O(V(V*E+K))

So the overall time complexity is O(V+E + V(V*E+K)) = O(VVE + K)

Example:



Consider the above workflow graph.
A, B, and C are not dependent on any jobs. But D can only start the job when A, B, and C are completed. Similarly, E is dependent on D, F is dependent on D, G is dependent on E and F, and H is dependent on G.

Working of Algorithm:

1. Initialize an array of length equal to the number of machines (k) to store finishing times for each machine.

2. Visit each vertex in the topological order.

3. For each vertex, identify its predecessors (vertices that provide input to the current job). Since we use a topological order, the predecessors' jobs are guaranteed to be completed before the current job.

   For example, if we consider D, the predecessors of D are A, B, and C.

4. Determine the maximum time for computation and data transfer among all predecessor jobs.

   'A' is completed at 5 and can transfer data from A -> D in 2, so in total 'A' takes 7 units of time.
   'B' is completed at 3 and can transfer data from B -> D in 1, so in total 'B' takes 4 units of time.
   'C' is completed at 8 and can transfer data from C -> D in 5, so in total 'C' takes 13 units of time.

5. Add the determined maximum value to the current job's completion time, compare it with the finishing time of the earliest machine, and store the maximum of these two values.

D can complete 4 units of time, ‘D’ can only start at 13, so ‘D’ will complete its job at 17.

6. Repeat this process for all vertices.

7. Among all the machine completion times, select the maximum time. This represents the minimum time required to complete the workflow with K machines.