Job Scheduling and Data Assignment Problem

Background

With the rapid development of chip hardware computing capabilities in cloud computing, 5G communications, and artificial intelligence, hardware architectures are evolving towards more complex multi-core and heterogeneous architectures. The coupling problem of job scheduling and memory allocation exist widely in operating systems, distributed systems and compilers, and play a key role in software-hardware co-optimization. In this problem, you need to design an efficient task scheduling algorithm together with a memory allocation policy.

Description:

There are l tasks $J_1, J_2, \dots J_l$ which need to be scheduled on n machines $M_1, M_2, \dots, M_n$ with varying processing powers. There are also m disks $D_1, D_2, \dots, D_m$ with varying transmission speeds and storage quotas.

- The processing power machine of M_i is $power(M_i)$.
- Each disk D_j has a read/write transmission rate of $rate(D_j)$ and a storage quota of $quota(D_j)$.
- Each task J_i has a size of $size(J_i)$, which means it will take $size(J_i)/power(M_j)$ amount of time to execute if it is ran on machine M_j.
- When a task J_i finishes its execution, it will output a piece of data of size $data(J_i)$ to some disk. This will take an additional time of $output(J_i)/rate(D_k)$ if this data is stored on D_k.
- Before a task J_i starts its execution, it needs to read the output data of some other tasks. We let $PRED_{data}(J_i)$ denote the set of tasks whose output data is needs by J_i. This will take a total time of $\sum_{J_j \in PRED_{data}(J_i)} {output(J_j)}/{speed(D_{z_j})}$ and this must be done before task J_i starts its execution.

Task schedule. You goal is to find a task schedule which is an assignment of tasks to machines and disks. More specifically, for each task J_i, you needs to specify three parameters (x_i, y_i, z_i),

meaning that J_i starts at time x_i on machine M_{y_i} and stores its data to disk D_{z_i}. The earliest starting time of any task is 0.

When a task starts, it goes through three phases: reading data from disk, executing the task, and writing its data to the disk. For convenience, we denote the starting time of three phases as a_i, b_i, c_i and the finishing time of the last phase as d_i. These values are determined as following.

$$a_i = x_i$$

$$b_i = a_i + \sum_{J_j \in PRED_{data}(J_i)} output(J_j) / speed(D_{z_j})$$

$$c_i = b_i + size(J_i)/power(M_{y_i})$$

$$d_i = c_i + output(J_i)/speed(D_{z_i})$$

All the above division results need to be rounded up.
In addition, a feasible schedule should satisfy the following requirements:

1. **Environment dependencies**: we say a job J_i is environment-dependent on job J_j, which means J_i cannot start until J_j finishes its execution. i.e., $a_i \geq c_j$. Note that if J_i is scheduled on same machine with J_j, then it still needs to wait for J_j to complete storing its data.
2. **Data dependencies**: if $J_j \in PRED_{data}(J_i)$, then we say a job J_i is data-dependent on job J_j, which means J_i cannot start until J_j finishes storing its data. i.e., $a_i \geq d_i$.
3. **Affinity of machines**: each task J_i has a list A_i of affinitive machines that it can be assigned to. It must be scheduled on one of its affinitive machines, i.e., $y_i \in A_i$.
4. **No preemption**: Each machine executes only one task or transmits one data at a time, and no interruption is allowed during its lifecycle. In other words, for each pair of tasks J_i and J_j such that $y_i = y_j$, the two intervals $[a_i, d_i]$ and $[a_j, d_j]$ cannot have any overlaps.
5. **Quota of disk**: the total size of all data stored in same disk cannot exceed that disk's quota. That is, we must have $\sum_{J_i : z_i = j} data(J_i) \leq quota(D_j)$ for all $1 \leq j \leq m$.

Objectives:

You objective is to find an assignment that minimizes the *makespan*, which is the finishing time of the last task. In other words, the objective is minimize $\max_{1 \leq i \leq l} \{d_i\}$.

Input:

The first line contains an integer l representing the number of tasks. In the following l lines, each line represents a task: the first three integers represent task id, task size, the size of task's output data. The fourth integer k is the number of affinitive machines for this task, and the remaining k integers are the IDs of affinitive machines.

Next, a line contains an integer n representing the number of machines. In the following n lines, the two integers in each line represent machine ID and its power, respectively.

Next again, a line contains integer m representing the number of disks. In the following m lines, the three integers in each line represent disk ID, its speed and quota, respectively.

The remaining part describes the two types of dependencies among tasks.

The first line of this part is an integer N meaning the number of data-dependencies. In the following N lines, each line contains two integers i, j, which means task j is data-dependent on task i.

Then there is a new line with integer M meaning the number of environment-dependencies. In the following M lines, each line contains two integers i, j, which means task j is environment-dependent on task i.

Output:

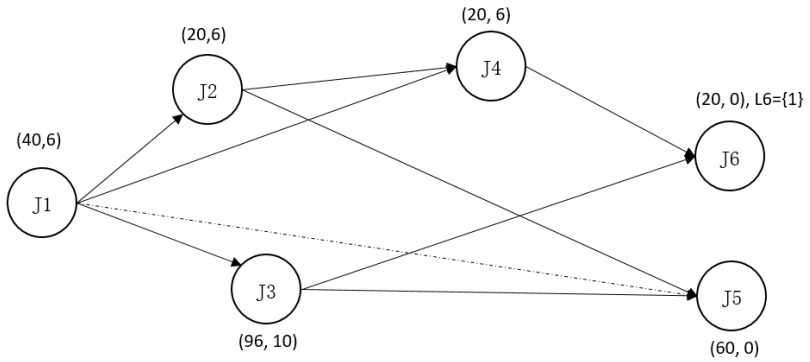
You should output l lines, each line i contains four integers i, x, y, z means the task J_i will start at time x on machine M_y and store the result on disk D_z.

Sample:

Input	Output
6 1 40 6 2 1 2 2 20 6 2 1 2 3 96 10 2 1 2 4 20 6 2 1 2 5 60 0 2 1 2 6 20 0 1 1 2 1 1 2 2 2 1 1 30 2 2 17 8 1 2	1 0 2 2 2 23 1 1 3 23 2 2 4 52 1 1 5 79 2 1 6 87 1 2

1 3 1 4 2 4 2 5 3 5 3 6 4 6 1 1 5	

Notes:



At time 0, J_1 is scheduled on M_2 and stores its output to D_2, it costs 40/2 and 6/2 unit time in reading phase and executing phase, respectively. So, it finishes at time 23. Note that the time cost in reading phase is zero, since it does not data-dependent on any task.

At time 23, J_2 is scheduled on M_1 and reads its data from D_2 and stores its output to D_1. Finally, It costs 6/2 + 20 + 6 = 29 unit time and it finishes at time 52.

At the same time, J_3 is scheduled on M_2 and reads its data from D_2. And it costs 6/2+96/2+10/2 and finishes at 79. Note though J_3 and J_1 are running on same machine, J_3 still needs to read the J_1's outputted data from disk.

At time 52, J_4 starts on M_1 and reads data of J_1 and J_2, which costs 6/2 + 6/1 unit time. So the final time cost is (3+6)+20+6 and finishes at time 87.

At time 79, J_5 starts on M_2 and costs (10+6)/2 unit time to read data and another 60/2 unit time to execute, does not output any data. So, it finishes at time 120.

At time 87, J_6 starts on M_1 and finishes at time 118.

So, the final makespan is 120 and it is a feasible solution. Let's verify the feasibility of above solution.

Dependency constraints: In the solution, J_2, J_3 start after J_1 and J_4 starts after J_2, J_3, and so on.

Affinity of machines: In this case, the only restriction of affinity is that task 6 can only run in machine 1. And in this solution, it DOES.

No preemption: it is obvious.

Quota of disk: (1) The tasks storing data in the disk 1 are J_2, J_4, J_6, their total data size is 6+6+0<30; (2) The tasks storing data in the disk 2 are J_1, J_3, J_5, their total data size is 6+10<17