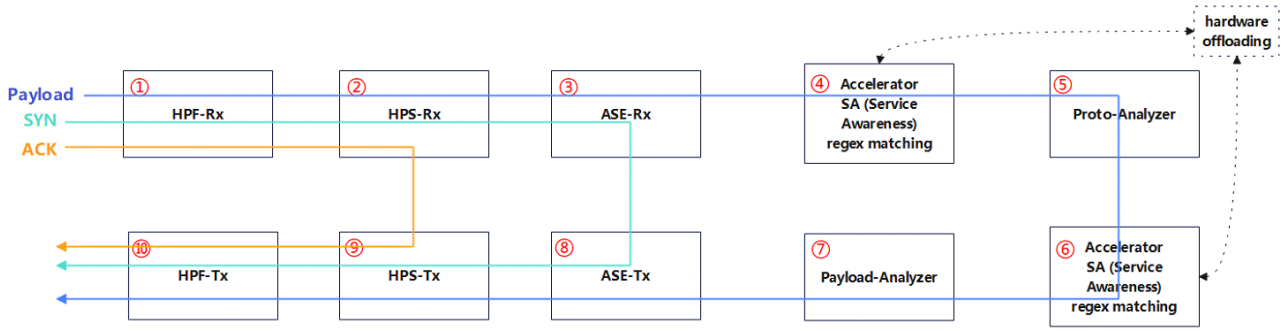
自适应安全引擎排程问题

Limits: 4 sec., 1024 MiB

The Adaptive Security Engine (ASE) is the next-generation content security processing engine for security gateways. It integrates proxy functionality, TLS decryption, and Next-Generation Engine (NGE) into a single engine, reducing scheduling overhead, and can additionally flexibly switch between stream and proxy modes in an adaptive manner, dynamically optimizing business processing.

The ASE computation graph can be abstracted as follows:



Incoming packets first arrive at node 1 (HPF-Rx), go through multiple stages of computation depending on their type, and finally are sent out after node 10 (HPF-Tx). The delay of a packet is the duration between when it arrives and when it is sent out.

There are 3 types of packets:

1. Payload packets, which contain the actual data being transmitted;
2. SYN packets, which are the first packet of a stream that will trigger stream creation;
3. ACK packets, which acknowledge the receipt of a message and do not contain data.

Their computation paths are shown in the above figure. A packet can be processed by a computation node only after it has been processed by the previous node on the path.

One key feature of ASE is batch processing. In a task of ASE, we select a batch of packets of the same type and perform their computation on a computation node. The computation time depends on the computation node, the batch size, and the packet type; the cost model of the ASE system will be specified later.

Another important feature of ASE is hardware offloading. In node 4 and node 6, after processing a batch of packets on the CPU, the packets will be sent to a queue (both node 4 and node 6 share the same queue), and a separate hardware accelerator will process the packets one by one in FIFO order. The hardware execution is asynchronous: after sending the batch of packets to the queue, the execution of the task on node 4/6 ends and the CPU is available for a new task, but a packet can be processed by node 5/7 only after its hardware execution completes.

In this problem, we assume ASE is deployed on a single CPU core, so only one task can be executed at the same time (except the execution on the hardware accelerator). Once a task starts, it will run to completion without interruption. When no task is running, the scheduler can form a new task and run it.

Your task is to implement a scheduler for ASE that optimizes the average delay of packets. You can find more information on scoring in the Scoring section of the problem statement.

Please note that this is an interactive problem. Your solution will communicate with the interactor through the stdin/stdout streams.

Interaction protocol

The first 20 lines of the input describe the costs of the computation nodes. Each line starts with three integers i, j and b_{ij} — the index of the computation node, the type of the packet, and the batch limit. Next, there are b_{ij} integers, the k-th integer c_{ijk} describes the CPU execution time for a batch of computation on the node i with packet type j and batch size k.

The 21-st line contains three integers c_4, c_6 and c_r — the time for processing a packet by the hardware accelerator at node 4, the time for processing a packet by the hardware accelerator at node 6, and the time for trying to receive incoming packets. Note that the computation graph and its cost model are fixed for this problem.

The next line contains a single integer n, which specifies the number of packets in this testcase. After reading the integer n you should start the interaction to receive further data.

You can perform two types of actions:

1. `R` t: Try to receive incoming packets. In order to make this action you should print a character `R` followed by an integer t denoting the start time of this call. Please note, that the value t should be strictly greater than the end time of the previous action (or greater than 0 for the first action).

   After that, you should first read an integer p — the number of packets that you received, and then p lines of data describing the packets. Each line contains three integers id, $type_{id}$ and $arrive_{id}$ — the ID of the packet, the type of the packet, and the arrival time of the packet. You will receive all packets that arrive after the previous call of action 1 and no later than time t. The incoming packets are ordered according to their arrival time.

   If you read a value $p = -1$, this means that the time t that you printed is invalid. In this case, you have to stop the execution of your program immediately, and you will receive a `Wrong Answer` verdict for the current test case.

   This action takes time c_r: the start time of the action is t and the end time is $t + c_r - 1$. You can start the next action at the time $t + c_r$ or later.

2. `E` t i s id_1 id_2 ... id_s: Execute a task. In order to make this action you should print a character `E`, followed by an integer t denoting the start time of this task, an integer i denoting the index of the computation node, an integer s denoting the number of packets in the batch, and s integers id_k indicating the IDs of the packets in this batch. The value t should be strictly greater than the end time of the previous action (or greater than 0 for the first action). The packets in the batch must be of the same type j.

   If $i = 4$ or $i = 6$, the batch of packets will be added to the end of the hardware queue in this order. The hardware accelerator will start processing the packet id_1 at time $max(t + c_{ijs}, Q)$ and finish at time $max(t + c_{ijs}, Q) + c_i - 1$. The packet id_1 will be available for further processing at node $i+1$ at time $max(t+c_{ijs}, Q)+c_i$. The hardware accelerator will start processing the packet id_k at time $max(t + c_{ijs}, Q) + (k - 1) * c_i$ and finish at time $max(t + c_{ijs}, Q) + k * c_i - 1$. The packet id_k will be available for further processing at node $i + 1$ at time $max(t + c_{ijs}, Q) + k * c_i$.

   The value Q indicates a time unit when the hardware accelerator becomes available to process the next packet. After this action, the value Q becomes equal $min(t + c_{ijs}, Q) + s * c_i$. Please note that there is a single queue for both notes 4 and 6. Initially (before the first action) the value Q is equal to 1.

   This action takes time c_{ijs}: the start time of the action is t and the end time is $t + c_{ijs} - 1$. You can start the next action at the time $t + c_{ijs}$ or later.

   After that, you should read a value p. If $p = 0$ then the action was valid, and you can continue with the interaction. If $p = -1$ then the action was invalid. In this case, you have to stop the execution of your program immediately, and you will receive a `Wrong Answer` verdict for the current test case.

After the execution of tasks on the last node ($i = 10$), all packets within this action are considered to be processed at time $t+c_{ijs}$: $processed_{id_k} = t+c_{ijs}$. When all n packets have completed processing,

the interaction is over. Your solution must stop its execution. You will receive an `Accepted` verdict for this test case and your score will be calculated.

Input constraints

$1 \le i \le 10$,
$1 \le j \le 3$,
$1 \le b_{ij} \le 16$,
$1 \le c_{ijk} \le 1000$,
$c_4 = 31$,
$c_7 = 107$,
$c_r = 20$,
$1 \le n \le 10^4$,
$1 \le id \le n$,
all id values are unique,
$1 \le arrive_{id} \le 5 \cdot 10^6$,
$1 \le type_{id} \le 3$,
$-1 \le p \le n$.

Output constraints

$1 \le i \le 10$,
$1 \le s \le b_{i,type_{id_1}}$,
$1 \le id_k \le n$,
$1 \le t \le 10^7$,
value t should be greater than the end time of the previous action (or greater than 0 for the first action),
all packets in the batch should be ready for execution on the node i at the time t,
all packets in the batch should have the same type.

Interaction example

Please follow the attached example to view. There, you can find input and output examples, a detailed explanation of each step, as well as the location and progress of each packet.

Scoring

If your solution performs an invalid action or violates one of the output constraints, it is considered to be incorrect, and for such a test case you receive 0 points. Otherwise, your score is calculated as follows:

$$score = \max\left(0, \left\lfloor \left(10^4 - d_{ave}\right) \cdot 10^3 \right\rfloor\right)$$

Where d_{ave} is the average delay of packet processing:

$$d_{ave} = \frac{\sum_{k=1}^{n} \left(processed_k - arrive_k\right)}{n}$$

Your overall score is the sum of your scores over all test cases.

Submissions

- The execution time limit is 4 seconds per test case and the memory limit is 1024 mebibytes.
- The code size limit is 64 kibibytes.
- The compilation time limit is 1 minute.

- There are 50 provisional test cases. Your submissions will be evaluated on the provisional set during the submission phase.
- You can submit your code once 2 times / day, and you will receive feedback with your score for each of the provisional tests.
- There will be 500 test cases in the final testing after the submission phase is over. Please note that provisional tests are **not included** in the final testing.

Quick start

Check the sample solution, which implements a very simple algorithm: iterate through the nodes, look for packets available for processing, and process them in batches of one packet; if no packets are found, receive new ones.

- C++
- Python