Birla Institute of Technology & Science, Pilani
Data Structures & Algorithms (CS F211)
Lab Assignment - 7 (Scheduling)

Instructions

- All input expressions should be read from stdin and output should be printed on stdout.
- For 1 hour 45 min, only a subset of test cases will be visible to students after submitting the code on the portal. After 1hour 45 min, all test cases will be made visible and they will have last 15 min to correct their code and resubmit.
- At the end of 2 hour period, the online system will stop accepting the submissions.
- Only the last submission by the student before end of lab will be considered for evaluation.
- Following messages by online portal will **tentatively** fetch these marks:
  - Correct → 4 marks
  - Wrong-answer (correct for more than half test cases) → 3 marks
  - Run-error/Compiler-error/Timelimit-error → 2 marks
- All submitted source code will be later checked manually by the instructor and final marks will be awarded, which will be posted on Nalanda after the lab assignment has been done by all lab sections.
- Solution must be implemented using the algorithm and data structures mentioned in the lab sheet only.

Online Pre-emptive Priority Scheduling on a Single Processor

Consider a processor which executes tasks. Each task is described by a 4-tuple (I, R, C, P), in which I is the task identifier. It is a unique integer ($I \geq 1$) which identifies the task. R is the release time of the task. R is an integer ($R \geq 0$) which specifies the time at which the task I is ready to run if it is allocated on the processor. The task I can be run at a time $T \geq R$. C is the computation time of the task I. C is an integer ($C \geq 1$) which specifies the duration for which the task I will be run. P specifies the priority of the task I. P is an integer ($P \geq 0$). At any given instance of time, only the task with highest priority can be run on the processor. Online scheduling means that the processor has knowledge of (at a time T) only the task with $R \leq T$. At any given instance of time T, the processor cannot know the "future" tasks with $R > T$. The scheduling decisions must be taken by considering the task with $R \leq T$. Pre-emptive scheduling means that if at any time a task arrives which is having a higher priority than the currently running task, then the new task is allocated on the processor.

Input

You will be given the 4-tuples (I, R, C, P) describing the tasks (one task per line) sorted in non-decreasing order of R (all values separated by blank space):

$$\begin{matrix} I_1 & R_1 & C_1 & P_1 \\ I_2 & R_2 & C_2 & P_2 \\ \vdots & \vdots & \vdots & \vdots \\ I_n & R_n & C_n & P_n \end{matrix}$$

Output

For each unit of time you have to print the task that is running on the processor (separated by blank space). If the processor is idle (no task is running on the processor), then you have to print "0" (zero) for the corresponding time unit.

$$T_1\, T_2\, T_3 \ldots T_m$$

Task T_i runs for $i-1 < T \leq i$, where $1 \leq i \leq m$. $T_i = 0$ means that the processor is idle for $i-1 < T \leq i$. We can have $T_i = T_j$ for $i \neq j$ (a task can run for more than one unit of time).

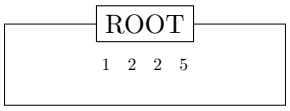
Sample Input

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

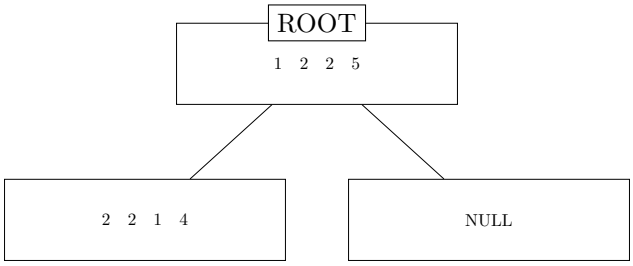
Procedure

You will have to implement the priority queue using a Binary Search Tree (BST). For inserting a node into the priority queue, implement the algorithm Tree-Insert for BST. For deleting a node from the priority queue modify the algorithm Tree-Delete for BST so that it always deletes the node with highest key. Scan the input one line at a time. The processor is idle for $0 \leq T \leq 2$. At $T = 2$, we have two tasks (1 and 2). Initialize the BST to be empty at the start. Insert task 1 and 2 in order into the BST.

$T = 2$: Processor → Empty

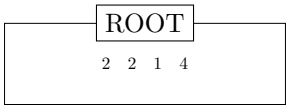


$T = 2$: Processor → Empty



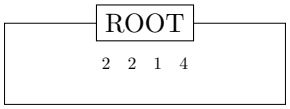
When all the tasks are inserted into the BST at $T = 2$, delete the task with highest priority and allocate it on the processor.

$T = 2$: Processor → 1 2 2 5



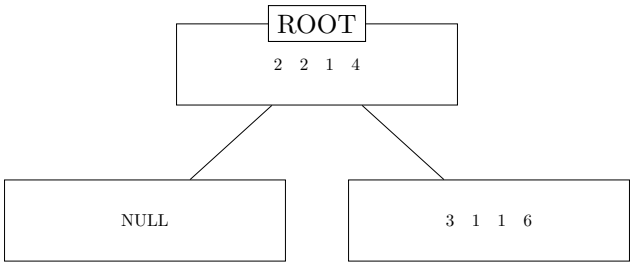
At $T = 3$, the task 1 has run for 1 unit of time. Update its C value:

$T = 3$: Processor → 1 2 1 5



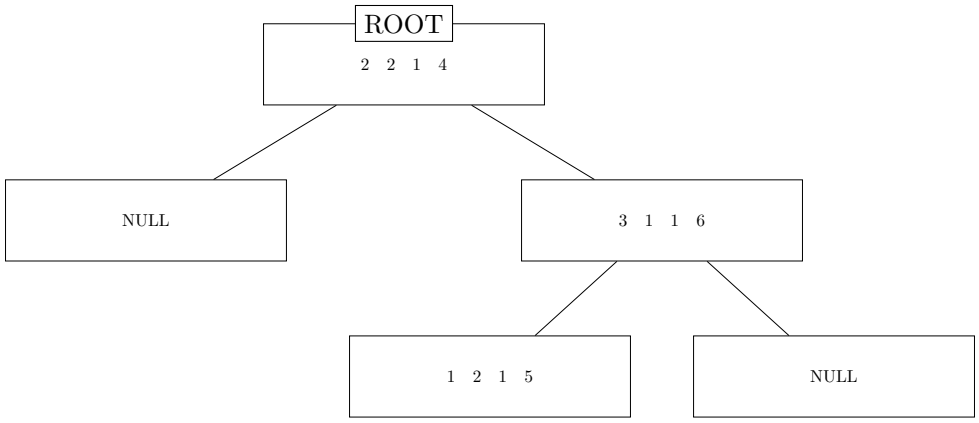
Now the task 3 arrives with priority 6. It is having higher priority than the currently running task. First insert 3 1 1 6 in the BST:

$T = 3$: Processor → 1 2 1 5



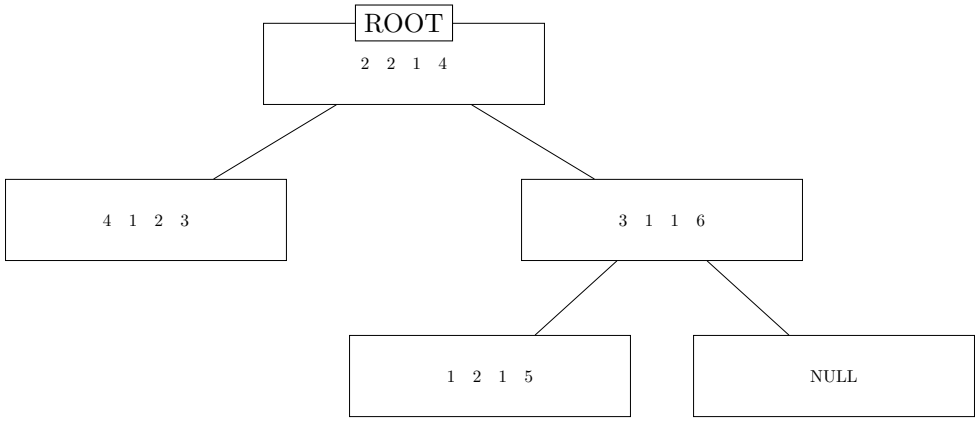
Now insert 1 2 1 5 back in the BST because a new task with higher priority has arrived. If the new task is not having higher priority than the currently running task, then we will not put back the currently running task.

$T = 3$: Processor → Empty



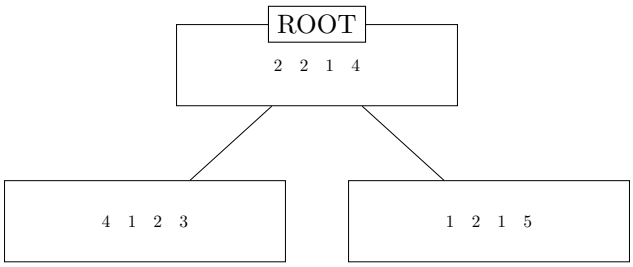
Now another task (4 1 2 3) arrives at $T = 3$. Insert it into the BST:

$T = 3$: Processor → Empty



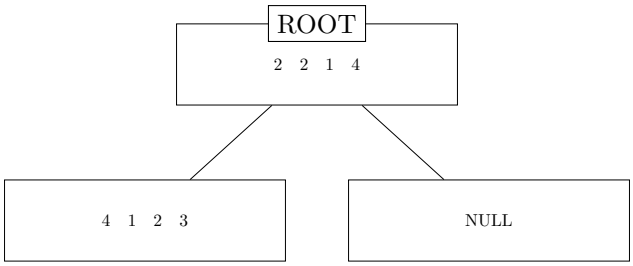
Now delete the highest priority task from the BST, and allocate it on the processor:

$T = 3$: Processor → 3 1 1 6



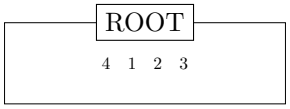
The task 3 will run for 1 unit of time. At $T = 4$, the task 3 will finish. Now delete the highest priority task from BST and allocate it on the processor:

$T = 4$: Processor → 1 2 1 5



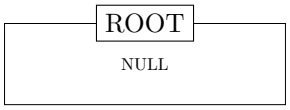
The task 1 will run for 1 unit of time. At $T = 5$, the task 1 will finish. Now delete the highest priority task from BST and allocate it on the processor:

$T = 5$: Processor $\rightarrow$ $\boxed{2 \quad 2 \quad 1 \quad 4}$



The task 2 will run for 1 unit of time. At $T = 6$, the task 2 will finish. Now delete the highest priority task from BST and allocate it on the processor:

$T = 6$: Processor $\rightarrow$ $\boxed{4 \quad 1 \quad 2 \quad 3}$



The task 4 will run for 2 units of time. At $T = 8$, all tasks are finished and the BST is empty.

Sample Output

0 0 1 3 1 2 4 4