**LOVELY PROFESSIONAL UNIVERSITY**

**Project based test**

**Student Name:** Mohd Yusuf

**Student ID:** 11814160

**Email Address:** mohammadyusuf025@gmail.com

**GitHub Link: -** <https://github.com/techyusuf/Operating_System-jurassicPark-Project>

**Scenario:-**

1. Jurassic Park consists of a dinosaur museum and a park for safari riding. There are *m* passengers and *n* single-passenger cars. Passengers wander around the museum for a while, then line up to take a ride in a safari car. When a car is available, it loads the one passenger it can hold and rides around the park for a random amount of time. If the *n* cars are all out riding passengers around, then a passenger who wants to ride waits; if a car is ready to load but there are no waiting passengers, then the car waits.

Using synchronization tools like locks, semaphores and monitorssynchronize the *m* passenger processes and the *n* car processes.

**Description:**

Contemporary operating systems are built around the concept of processes or tasks. Jurassic Park is an inter-process communication and synchronization problem between multiple tasks. Visitors to the park want to take a ride in a safari car, for that they have to line up and wait for his time. When his time come they will ride according to his priority. A poorly implemented solution could lead to the inter-process communication problems of starvation and deadlock.

Jurassic Park is not unlike real world simulation and control problems such as found in airports, supply houses, traffic systems, oil tank farms, etc. Each requires multitasking roles involving inter-process coordination and communication. Like the classical dining philosophers and sleeping barber problems, the Jurassic Park exercise demonstrates what problems can occur in a multitasking, multi-user environment when many tasks are competing for the same resources.

**Algorithm:**

1. Visitors arrive at the Jurassic Park at random times.
2. Upon entering the park, a visitor gets in line

If

There is any priority for ticket. Then visitor gets in line to purchase a ticket.

Else

Visitors will wait for his ride.

1. After completing steps 2, the visitor gets in the museum line.
2. After visiting the museum, the visitor gets in the tour car line to wait until they start his tour.
3. When the touring car is filled with visitors and a driver, the car enters Jurassic Park and follows a guided tour path through the park.
4. When the tour car completes the ride and pulls into the unloading station then drivers left the visitors in previous place.
5. The driver goes back to sleep awaiting new duties, and the tour car pulls forward to be loaded again.
6. After visiting and completing park tour, the visitor exits the park.

**Code:**

|  |
| --- |
| #include <stdio.h>    #define min(A,B) ((A) > (B) ? (B) : (A))  #define max(A,B) ((A) > (B) ? (A) : (B))    **int** **main**(**void**){  **int** testCount;  scanf("%d", &testCount);    **while** (testCount--){  **int** cars, wander, ready, p, r, k;  **int** doneCount, ridingCount, carsWaiting;  **int** carArrives[**50**];  **int** becomeReady[**5100**];  **int** nextCar;  **int** totalPeople;  **int** i, time;    scanf("%d %d %d %d %d %d", &cars, &wander, &ready, &p, &r, &k);    **if** (cars == **0**){  **int** movedToReady = min(wander, k/r);  printf("0 0 %d %d**\n**", wander - movedToReady, ready + movedToReady);  **continue**;  }    doneCount = ridingCount = **0**;    **for** (i = **0**; i < cars; i++)  carArrives[i] = **0**;    totalPeople = wander+ready;  **for** (i = **0**; i < ready; i++)  becomeReady[i] = **0**;  **for** (i = ready; i < totalPeople; i++)  becomeReady[i] = (i-ready+**1**)\*r;    nextCar = **0**;  **for** (i = **0**; i < totalPeople; i++){  **int** readyTime = becomeReady[i];  **if** (readyTime > k)  **break**;    **if** (carArrives[nextCar] > readyTime)  readyTime = carArrives[nextCar];  carArrives[nextCar] = readyTime + p;  nextCar = (nextCar+**1**) % cars;      **if** (readyTime + p <= k)  doneCount++;  **else** **if** (readyTime <= k)  ridingCount++;  }    carsWaiting = **0**;  **for** (i = **0**; i < cars; i++)  **if** (carArrives[i] <= k)  carsWaiting++;    printf("%d %d %d %d**\n**", carsWaiting, doneCount, max(**0**, wander - k/r), ready + min(wander, k/r) - doneCount - ridingCount);  }    **return** **0**;  } |

**Explanation of Input Code:**

The input to this program is the total number of safari cars available in the park n (0 ≤ n ≤ 50). The number of passengers wandering in the museum m (0 ≤ m ≤ 5000) and the number of passengers q (0 ≤ q ≤ 100) in the park gate ready to take a ride in the park at time zero. The other inputs are the values of p (1 ≤ p ≤ 100), r (1 ≤ r ≤ 100) and k (r ≤ k ≤ 100000). Note that there can be no negative or fractional inputs.

First line contains an integer N (1 ≤ N ≤ 100) indicating number of test cases. Each of the following N lines contains input data, separated by single space, for different test cases in the given order:  
No. of Safari Cars, No. passengers in Museum at time zero, No. of passengers at park gate ready for ride at time zero, p, r, k

The output of this program is the number of cars waiting at the park gate, the number of passengers who have completed taking one ride in the park, the number of passengers still waiting at the park gate to take a ride and the number of passengers still wandering in the museum, respectively, after k units of time.

Output for each test case will be on separate lines. Each line will contain:  
No. of cars waiting at the park gate, No. of passengers completed the park ride, No. of passengers wandering in the museum, No. of passengers still waiting to take a ride, respectively.  
The output parameters should be in the above order with one space gap.