AOS Project

(7) Problems on Process Co-ordination - 2

Team Name: RantProMax

Roll no.	Name
2022201006	Jaffrey Joy
2022201013	Amit Marathe
2022201034	Rohit Hooda
2022201046	Atharva Vaidya

GitHub Link: jaffreyjoy/IIITH_AOS_Project_7_RantProMax (github.com)

1: Problem 1 Life on a Chip

In future, water will be a scarce resource and most of the humans will have to generate water on their own. For this purpose, a chip was developed that will combine Hydrogen and Oxygen to make water. This chip has n sites where the reaction will take place, and each reaction will generate 1 E Mj of energy. For safety purpose, no two consecutive sites should be used for the reaction, and total energy generation at any moment should not exceed than some threshold value. As for reaction, each hydrogen atom must combine with another hydrogen atom and an oxygen atom at a site.

For the purpose of this problem, each atom can be assumed to be a thread, which will be in

sleep state and at every state of reaction you have to wake up the corresponding threads(atom) at particular site, and join them. Assume that a reaction take sleep(3) time to complete.

Your task is to write an algorithm using semaphores, which efficiently uses atoms, and gives a faster way to generate water.

Solution:

Hydrogen and oxygen atoms can be considered as resources that are shared among the sites. Herein, the concept of ***mutex locks*** and ***semaphores*** would be vital as we wouldn't want multiple sites accessing a single resource of the same type(since the site operations aren't atomic) and we can't let the resources count to go below 0.

The Algorithms design is as below:

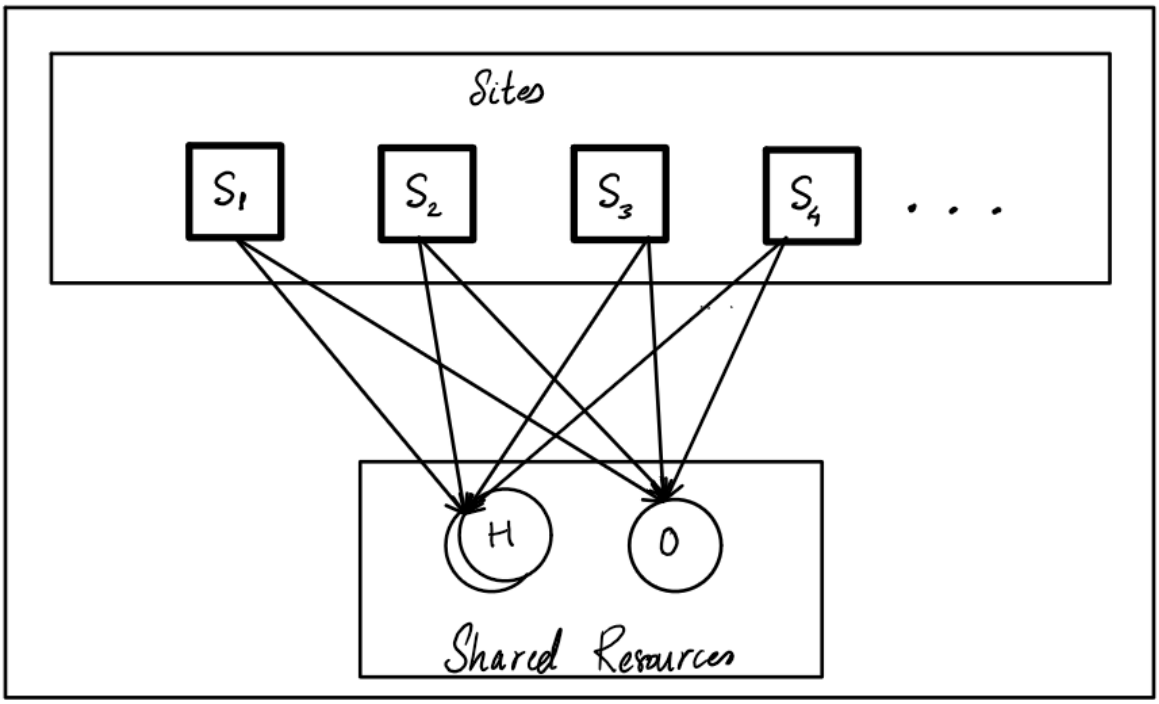


Fig. 1

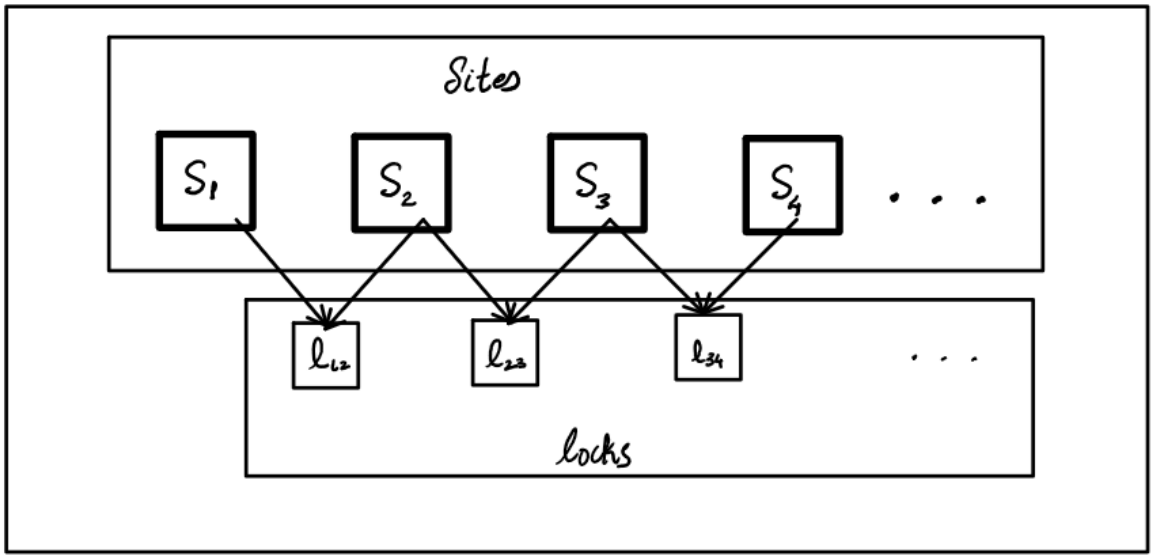


Fig. 2

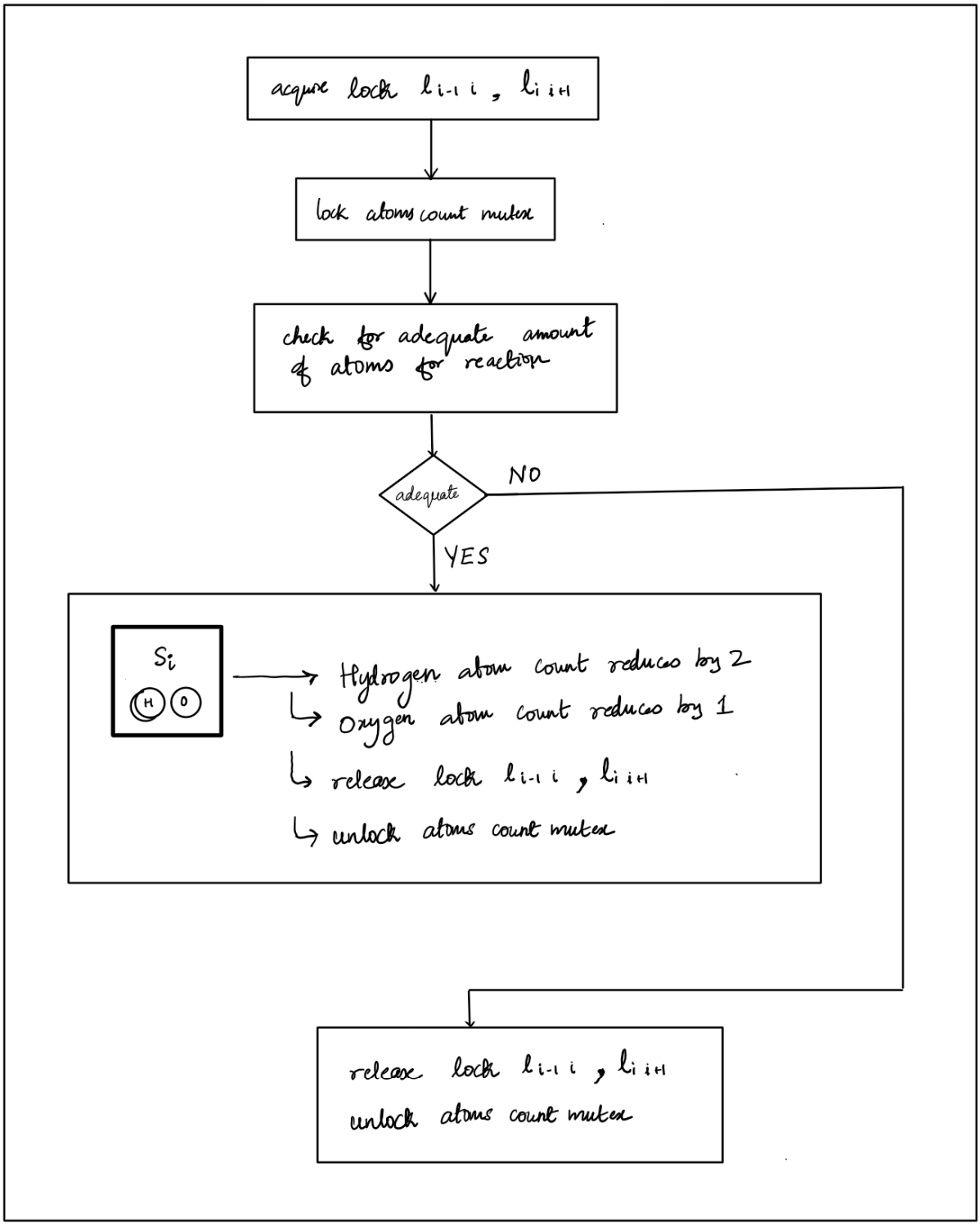


Fig. 3

Fig 1. signifies that sites require hydrogen and oxygen atoms for the reaction and the site has to acquire a lock over them in order to use them in a reaction.

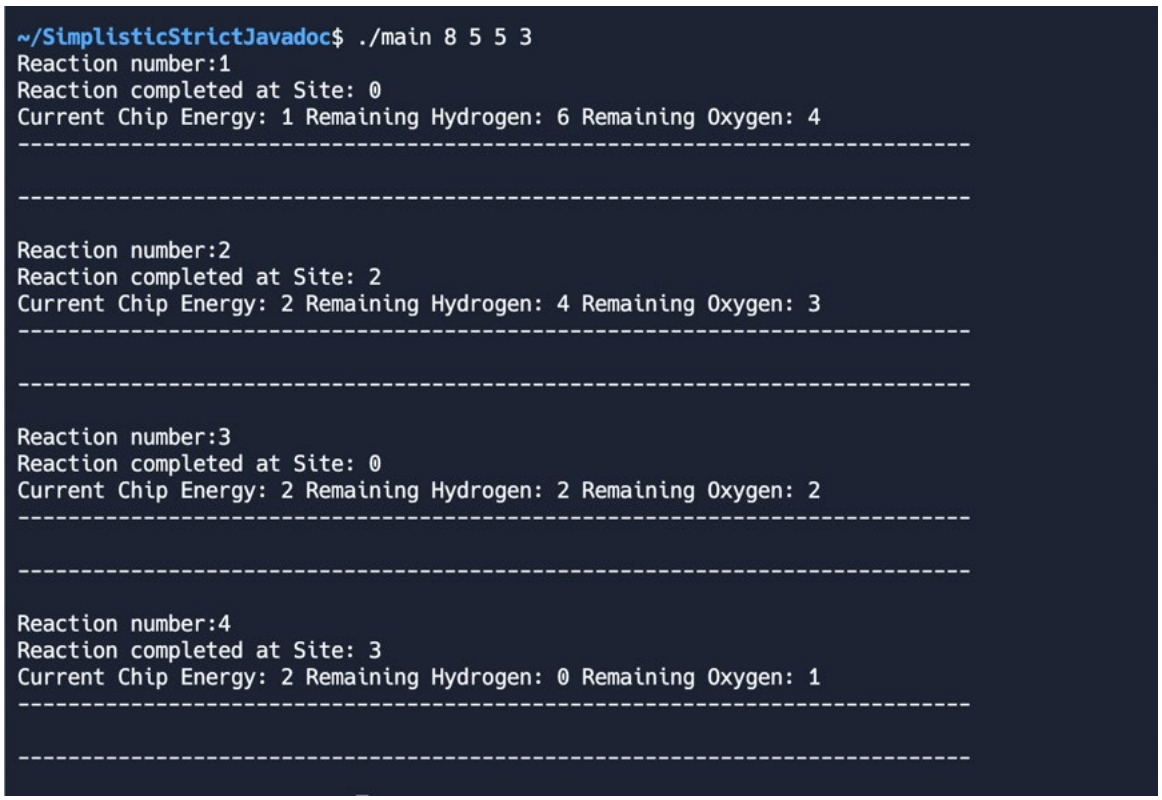
Fig 2. signifies that sites have a acquire 2 locks (except leftmost and rightmost sites (they need to acquire a single lock)) ($l_{i-1\ i}$) ($l_{i\ i+1}$). This is in order to ensure that no 2 consecutive sites are engaged in performing a reaction.

Fig 3. shows the flow of actions done by a site for a reaction to be performed or not based on the criteria that needs to be fulfilled by the site.

Caveat(s):

The (no 2 consecutive sites should engage in a reaction at a time) restriction raises an issue of deadlock where, if every site from S_1 to $S_{(n-1)}$ acquires the right lock i.e. ($l_{i\ i+1}$), none of the sites could proceed to perform a reaction, even though the required atoms are available.

Screenshot(s):



2: GHAC Old Bridge Trip

In Hyderabad, GHAC organizes Trek and other adventurous stuff for people. However, recently there are many geek people who have tried on these adventure trips. As a result, there had been two major classification of people, geeks and non-geeks. In a particular trek, where they have to cross an old bridge which can allow maximum 4 people, with the condition that bridge can only hold weight at one point i.e. all 4 of them have to be together on bridge and it can only be used one at a time; a next pass on bridge should only happen when previous pass is complete. All other combination are safe. Furthermore, this bridge is located near to the village of singers. So Trip Leader decided to invite some singers on the trip to bridge. In order to maintain the spirit of adventure, you cannot put three geeks with one non-geek or vice-versa.
But if boarding group has a singer, then he can calm down even unbalanced group. But you can never allow two singers on same pass, as they have jealousy issues.
Two procedures are needed, `GeekArrives()` and `NonGeekArrives()`, called by a geek or non-geek when he/she arrives at the end of bridge. The procedures arrange the arriving geeks and nongeeks into safe bridgeloads.
To get on the bridge, a thread calls `BoardBridge()`; once the bridge is at its full capacity, one thread calls `GoBridge()`. `GoBridge()` does not return until the members have left the end. Assume `BoardBridge()` and `GoBridge()` are already written. Implement `GeekArrives()` and `NonGeekArrives()`. These methods should not return until after `GoBridge()` has been called for the bridgeload. Any order is acceptable (again, don't worry about starvation), and there should be no busy-waiting and no undue waiting (geeks and nongeeks should not wait if there are enough of them for a safe bridgeload)

Solution:

The main objective of this problem is to ensure that `BoardBridge()` and `GoBridge()` are executed only when 4 people are available based on the given criteria.

To check whether the criteria is fulfilled on every `GeekArrives()` or `NonGeekArrives()` we maintain a lock to perform operations on the `Bridge` object whose members are `geeks_on_bridge` and `nonGeeks_on_bridge` which are incremented based on who arrives. This lock ensures that the checking of the criteria and incrementing of the various counts which are shared variables are collectively atomic in nature which prevents the anomaly produced in race conditions.

Screenshot(s):

```
D:\mtech\acad\sem_1\aos\project\q2>ob.exe 9 7 2
Trip Started with :: Geek Geek Geek Geek
Geek count :: 5
Non geek count :: 7
==================================================
Trip Started with :: Geek Geek Geek Geek
Geek count :: 1
Non geek count :: 7
==================================================
Trip Started with :: Non-Geek Non-Geek Non-Geek Singer
Geek count :: 0
Non geek count :: 4
==================================================
Trip Started with :: Non-Geek Non-Geek Non-Geek Non-Geek
Geek count :: 0
Non geek count :: 1
==================================================
```

3: Courses Allocation Algorithm

LSR College offers a great deal of varied courses to its students. However, allocation, this time was a mayhem, and for that reason their administrative staff approached ICS231 to get this problem solved. Before we get into details, here is the structure of any course in LSR.

- Each course allows upto 60 students to enroll.
- Each course belongs to one of 4 group, also known as Knowledge Spectrum. For eg.
  Commerce, Humanities, Management, Arts etc.
- Each course allows a certain quota to respective branch of students. For eg. M.Com, PHD, B.Com etc.

(Here Quota defines maximum number of people from each branch that can enroll to any
course)
Students too have a structure.

- Each student belongs to one of four branch, i.e M.Com, PHD, B.Com and Arts.
- Each student can take exactly 4 courses in one semester.
- Each student is allowed to make a list of preferences to the courses they like, such that
  size of list is 8.

Now Administrative head of LSR have asked us to impose certain criteria

1. Each student must pick at least one course from each Knowledge spectrum, i.e Commerce,Arts etc.
2. Each student should get 4 courses from the preference of 8.
3. Quota for each branch is in ratio 1:1:2:1 i.e out of 60 studnets, 12 belong to M.com, 12 to PHD, 24 to B.com and 12 to Arts

Your task is to write an suitable algorithm, which maintains certain data-structure for the process of allocation.
For eg. Queues. You can think of a distributed or centralized apporach to this problem. Both
approaches will work, and have their pits and falls. In case, of starvation, i.e when some student doesn't get their 4 opted courses, you have to report that student.
You can generate preference on random, and #of courses can be equally divided into 4 knowledge spectrum. Branch capacity can be determined by the quota mentioned above.

Solution:

Preferences are generated based on the below logic:
Since a student needs to choose a course from every knowledge stream, we first generate a `preference_count_per_knowledge_stream` array of length 4 where each index corresponds to a knowledge stream. If pc_i is an element in the above mentioned array then:

$$1 <= pc_i <= 5 \quad \text{and} \quad \sum_{i=0}^{3} pc_i = 8$$

This array is then used to generate pc_i distinct random course-ids (cid) where:

$$0 <= cid < (\text{total\_course\_count}/4)$$

After randomly generating the preference list of student, we come to the part of allocation.
To maintain the information of every knowledge spectrum, courses in it and branch of students enrolled in it, we maintain a 3-d vector(1st dimension corresponds to the knowledge spectrum, 2nd dimension corresponds to the courses in a knowledge spectrum and 3rd dimension corresponds to the branch of student enrolled).
For example:-
The 3d vector used is named `seats`. `seats[1][2][0]` would contain the value of the no. of seats available in the branch 0 quota of courseid 2 of knowledge spectrum 1.

In the allocation procedure, as we have to assign students to course in 1:1:2:1 ratio of quota and the maximum students in any course is 60, the number of students in branches 0, 1, 2, 3 will be 12, 12, 24, 12 respectively. To facilitate this we maintain 4 variables which are initialized to 12, 12, 24, 12 respectively which act as counting semaphores.
As and when a student is allocated courses, the quota of courses which are allocated to student's branch is decremented. Every student is a thread which executes independently.
If a student cannot be allocated any course of a knowledge spectrum then he will not be allotted any course of other knowledge spectrums too and are printed to `stdout` as mentioned in the output requirements. If student is allocated a course from every set of knowledge spectrum then the student's allocation is printed in file allocation.txt.

Every student thread acquires the mutex lock before requesting the course allocation and releases the lock when the student thread is either allocated all 4 courses or allocation fails for any knowledge spectrum.

Screenshot(s):

```
atharva@atharva-HP-ENVY-Laptop-13-ba1xxx:~/assignments/aos/2022201046$ ./a.out 300 12
Student 121 of branch 1 is not allocated
Student 125 of branch 1 is not allocated
Student 127 of branch 3 is not allocated
Student 128 of branch 0 is not allocated
Student 129 of branch 1 is not allocated
Student 132 of branch 0 is not allocated
Student 133 of branch 1 is not allocated
Student 135 of branch 3 is not allocated
Student 136 of branch 0 is not allocated
Student 137 of branch 1 is not allocated
Student 141 of branch 1 is not allocated
Student 144 of branch 0 is not allocated
Student 145 of branch 1 is not allocated
Student 147 of branch 3 is not allocated
Student 149 of branch 1 is not allocated
Student 151 of branch 3 is not allocated
Student 152 of branch 0 is not allocated
Student 155 of branch 3 is not allocated
Student 156 of branch 0 is not allocated
Student 157 of branch 1 is not allocated
Student 159 of branch 3 is not allocated
Student 160 of branch 0 is not allocated
Student 163 of branch 3 is not allocated
Student 164 of branch 0 is not allocated
Student 168 of branch 0 is not allocated
```

allocation.txt

```
Student 3 of branch 3 got allocated:0 0 2 0
Student 0 of branch 0 got allocated:2 1 1 2
Student 5 of branch 1 got allocated:0 1 2 2
Student 7 of branch 3 got allocated:0 2 0 2
Student 8 of branch 0 got allocated:1 1 2 0
Student 1 of branch 1 got allocated:1 1 0 1
Student 4 of branch 0 got allocated:1 2 0 0
Student 10 of branch 2 got allocated:1 2 0 0
Student 9 of branch 1 got allocated:2 2 1 2
Student 6 of branch 2 got allocated:0 0 0 1
Student 12 of branch 0 got allocated:0 2 2 2
Student 2 of branch 2 got allocated:2 2 1 2
Student 11 of branch 3 got allocated:2 1 2 2
Student 13 of branch 1 got allocated:0 1 1 2
Student 14 of branch 2 got allocated:0 2 2 0
Student 17 of branch 1 got allocated:2 0 1 2
Student 15 of branch 3 got allocated:1 2 0 1
Student 18 of branch 2 got allocated:0 2 2 0
Student 16 of branch 0 got allocated:0 0 1 0
Student 19 of branch 3 got allocated:2 2 1 1
Student 20 of branch 0 got allocated:1 1 0 1
Student 21 of branch 1 got allocated:1 0 0 0
Student 22 of branch 2 got allocated:1 1 2 0
Student 23 of branch 3 got allocated:0 0 2 2
Student 24 of branch 0 got allocated:2 0 2 1
Student 25 of branch 1 got allocated:1 0 2 2
Student 26 of branch 2 got allocated:0 1 0 1
Student 28 of branch 0 got allocated:0 2 0 0
```

allocation.txt

```
Student 242 of branch 2 got allocated:0 0 0 2
Student 243 of branch 2 got allocated:2 2 0 2
Student 245 of branch 2 got allocated:2 0 0 2
Student 246 of branch 2 got allocated:2 2 0 2
Student 247 of branch 2 got allocated:0 1 2 2
Student 249 of branch 2 got allocated:0 1 0 2
Student 252 of branch 2 got allocated:0 1 2 1
Student 253 of branch 2 got allocated:0 1 2 1
Student 262 of branch 2 got allocated:2 1 2 1
Student 271 of branch 2 got allocated:2 1 2 1
Student 292 of branch 2 got allocated:2 1 2 1
Knowledge spectrum 0 has following students count in its courses:
Course 0 has 58 students
Course 1 has 60 students
Course 2 has 60 students
Knowledge spectrum 1 has following students count in its courses:
Course 0 has 60 students
Course 1 has 60 students
Course 2 has 58 students
Knowledge spectrum 2 has following students count in its courses:
Course 0 has 60 students
Course 1 has 60 students
Course 2 has 58 students
Knowledge spectrum 3 has following students count in its courses:
Course 0 has 58 students
Course 1 has 60 students
Course 2 has 60 students
```