International Journal of Innovative Technology and Exploring Engineering (IJITEE)
ISSN: 2278-3075 (Online), Volume-9 Issue-6, April 2020

Invigilator Duty Allocation in Examination Hall

Vikas Verma, Apoorv Varshney, Ashish Mathur, Santanu Banerjee, Poorneshwar Anand

Abstract: ***The complexity in invigilator duty allocation in examination schedule has increased in last few years as number of examinations was increased. This complexity had some ambiguities as some invigilators got maximum number of duties while some got minimum number of duties. This paper proposed an optimize algorithm to allocate duties to invigilators. This algorithm provides optimized way to allocate duties of invigilators to reduce ambiguities such as unequal amount of duty allocation of invigilators. In this paper, an algorithm is proposed to allocate any numbers of invigilators in different examination halls in such a way that each invigilator will get equal amount of duties. This algorithm has written and implemented in java language.***

Keyword : Invigilator, Duty Allocation, Examination Schedule, Scheduling Algo.

I. INTRODUCTION

Invigilator duty allocation is an algorithm to allocate duties of each invigilator in different examination halls in such a way that each invigilator will get equal number of duties during whole examination session. This algorithm is developed because every year each school or college, each season of exams, the various departments or classes of an institution facing the complexity to draw a pattern to allocate the invigilators in different examinations halls. The invigilator allocation process must be optimized and simple. The main purpose of this algorithm is to demonstrate the possibility of allocating examination halls to each invigilator, automatically, using computers or other devices. This software consists front-end or user interface that will displayed on the screen and algorithm which is written in java where all the back-end coding, functionalities and activities linking related codes are written. The paper begins with the introduction of complexity in allocating invigilators to each examination halls and steps followed while creating algorithm to allocate invigilators to each examination hall.

Revised Manuscript Received on April 30, 2020.
***** Correspondence Author
Vikas Verma*, Assistant Professor,Department of Computer Science and Engineering, Jaipur National University, Jaipur, India.
Apoorv Varshney, Student Department of Computer Science and Engineering, Jaipur National University, Jaipur, India.
Ashish Mathur, Student Department of Computer Science and Engineering, Jaipur National University, Jaipur, India.
Santanu Banerjee, Student Department of Computer Science and Engineering, Jaipur National University, Jaipur, India.
Poorneshwar Anand Student Department of Computer Science and Engineering, Jaipur National University, Jaipur, India.

I. RELATED WORK

The Research paper that has been referenced here "A System of Automatic Construction of Exam Timetable Using Genetic Algorithm" illustrates that, to develop the scheduling algorithm the focus must be on two major components - Complexity and Optimality of algorithm; and it must consist user friendly interface. So, a great imagination power is required to develop an algorithm, so that allocation of invigilators can be done on all pros and cons and remove all bugs before deploy the software. Software must contain a responsive design, so that user interaction becomes easy. The UI of software should be able to attract user and easy to use because the number of user's uses the software is depended on the UI design and optimality of the software. Software must be updated periodically and all bugs should be resolved as soon as possible.The software must be tested among different type of users before deploy, so that a bug free software can be provided to all users.To build a successful software, all above guidelines must be followed to avoid the risk of losing users or customers and revenue can be increased by making algorithm more optimal and less complex. software is depended on the UI design and optimality of the software. Software must be updated periodically and all bugs should be resolved as soon as possible.The software must be tested among different type of users before deploy, so that a bug free software can be provided to all users.To build a successful software, all above guidelines must be followed to avoid the risk of losing users or customers and revenue can be increased by making algorithm more optimal and less complex.

II. PROPOSED WORK

As stated in the introduction of this paper, an optimal algorithm has been developed that covers approximately all the challenges faced while allocating invigilators to each examination halls. In this algorithm, the user (school/college/university) can manually decide the duty hours or number of duties of all invigilators. The user is also required to enter number of invigilators to be allocated and number of examination halls in which the invigilators will be allocated. The algorithm is also able to not allocate consecutive duties to the same invigilator. The working of algorithm will be shown as following.

A. Algorithm

Step 1: Start
Step 2: Declare a class Obj which implements Comparable interface, this class override compareTo() and priority() methods which has return type integer.

This class is used to check current value and its priority.

Step 3: Declare arrange_duty() method which return type is void and has an array list of Obj type as a parameter. This method arranges the duty of all invigilators in optimal manner and change priorities at each step accordingly.

Step 4: Declare method sort_priority which return type is void, parameters will be an array list of Obj type and two integer variables start and end for starting of priority and for changing priority after each step. This method is used to sort priorities of each invigilator after each arrangement.

Step 5: Declare method shift() which return type is void and has five parameters – one array list a1 of Obj type, integer sn(number of shifts), nf(number of faculties), nr(number of rooms) and t(exam duration). This method is used to show which invigilator allocated in which room and which shift, this also show remaining duty of invigilator, number of duties and priority of each invigilator after each arrangement.

Step 6: Declare method schedule which return type is void, parameters are integer variable days, Array List a1 and integer variables ns (number of shift), nf (number of faculties), nr (number of rooms), t (exam duration). Declare integer variable sum initialized with 0. Now we will check the failure cases that may occur if faculties are short in number or duty hours getting short to allot. If size of array list a1 is less than multiplication of nf and nr then print message insufficient faculties add more and exit the program. Create a loop from 0 to size of array list to calculate sum of value in sum, then check if sum is less then multiplication of nf, nr, ns, days, t then print message insufficient duty hours to allot and exit the program. If none of failure cases occurs then create a nested loop where outer loop will be from 1 to days, print day number, now run inner loop from 1 to ns and in inner loop run method arrange_duty with passing parameter a1, run method shift with passing parameter a1, j, nf, nr, t. End inner loop, end outer loop.

Step 7: Declare Obj type array list for faculty names list and pass the names in the list.

Step 8: Declare 5 integer variables ed(number of exam days), ns(number of shifts), nf(number of faculties per room), nr(number of rooms), t(exam duration).

Step 9: take value in those 5 integer variables from user.

Step 10: run the schedule method passing the parameters list, ed, ns, nf, nr, t.

Step 11: End

B. Pseudo Code

```
Class Object uses Comparable (type) {
    Object (integer argument value, String argument
string, integer argument p) {
        Set the parameters to current object
parameters;
    }
    Integer argument value, p;
    String argument string;
Override the method compareTo which will return
comparison of integer argument value of current
object;
Function priority (Object argument o):
      Return comparison of integer argument
p of current object;
 }
}
Declare static integer argument status;
Status equals 0;
    Function arrange_duty(ArrayList list) {
        Integer argument start;
        Start equals 0;
        Loop from 0 to list's size:
        If current index is less than size subtract 1 and
    current index object's value
of list is not equals to next index object's value of
list:
          Call function sort_priority(list, start,
current index);
          Start equals addition of current index
and 1;
       End if
      If current index is equals to size of list subtract 1:
          Call function sort_priority(list, start,
current index);
       End if
      End Loop
}
  Function sort_priority( ArrayList list, integer
argument start, integer argument end) {
      Integer argument k, l;
      String argument s1, s2;
      Outer Loop from start to end:
          Inner Loop from start to end:
          If comparison of value of p in list's object in
current inner loop index with value of p in list's object
in next inner loop index value using priority method is
less than 0:
                Call inbuilt function
Collections.swap(list, current inner loop index,
next inner loop index);
                End If
          End Inner Loop
      End Outer Loop
}
Function shift(ArrayList al, integer argument sn,
integer argument nf, integer argument nr, integer
argument t){
            Print shift number
            Integer argument p;
            P equals 0;
            Outer Loop from 1 to nr:
                Print room number;
                Inner Loop from 1 to nf:
                    Print value of string
    argument of object in al at p;
                    Integer argument k;
                    K equals value
    argument of object in al at p;
                    K equals subtraction
    of k and t;
                    Value argument of
    object in al at p equals k;
                    Value of p argument
    of object in al at p equals to pre increment
    of status;
                    Post increment of p;
                    End Inner Loop
                    Print Remaining;
                    Inner Loop from p
    to size of al:
```

```
                                    Print value
                                of string argument
                                of object in al at
                                current index;
                                End Inner Loop
                                Print Duties;
                                Inner Loop from 0
        to size of al:
                                    Print value
                                of string argument
                                of object in al at
                                current index;
                                End Inner Loop
                                Inner Loop from p
        to size of al:
                                                Print value
                                            argument of object

                                in al at current
                                index;
                                End Inner Loop
                                Print Priority;
                                Inner Loop from p
to size of al:
                                    Print value
                                of p argument of
                                object in al at
                                current index;
                                End Inner Loop
                        End Outer Loop
            }
Function schedule (ArrayList al, integer argument
days, integer argument ns, integer argument nf,
integer argument nr, integer argument t) {
            Integer argument sum;
            Sum equals 0;
            If al's size is less than
multiplication of nf and nr:
                        Print Insufficient faculties add
            more faculties;
                        Exit program;
            End If
            Loop from 0 to al's size:
                        Sum equals addition of sum and
            value argument of object at current index of
            al; End Loop
            If sum is less than multiplication of nf, nr, ns,
days and t
                        Print Insufficient faculty
                        duty hours add more
                        faculties or allocate more
                        duty hours; exit program;
            End If
            Outer Loop from 1 to days:
                                Print day number;
                                Inner Loop from 1
to ns:
                                    Call inbuilt

        Collections.sort(al);
                                    Call function

        arrange_duty(al);
                                    Call function

        shift(al, Current Inner Loop index, nf, nr,

        t);                                 }

                                End Inner Loop
                        End Outer Loop
}

In the main function {
            Declare ArrayList list of object
type;
            Add objects to list with duty hours, faculty
name, and priority;
            Print Enter Number of Exam Days to be
scheduled;
            Take input in ed from user;
            Print Enter Number of Shifts;
            Take input in ns from user;
            Print Enter Number of Invigilators
to allocate per room;
            Take input in nf from user; Print Enter
            Number of Rooms; Take input in nr from
            user;
            Print Enter Exam Duration (hours); Take
            input in t from user;
            Call function schedule (list, ed, ns, nf, nr, t);
            }
```

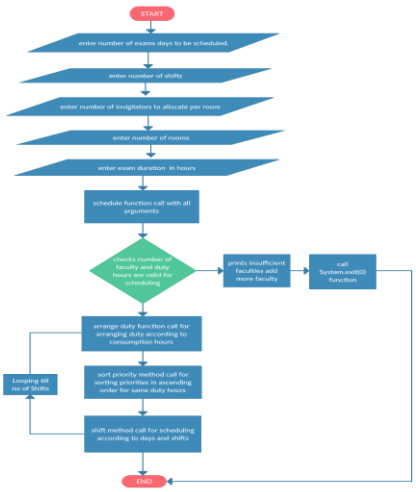

Figure-1: Flow Chart of Proposed Algorithm

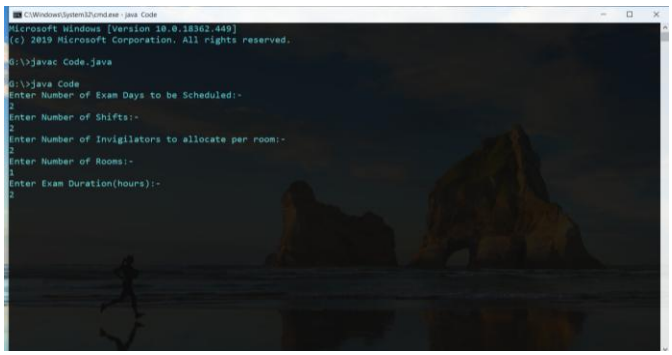

Figure 2: Screenshot of Input entered by the User

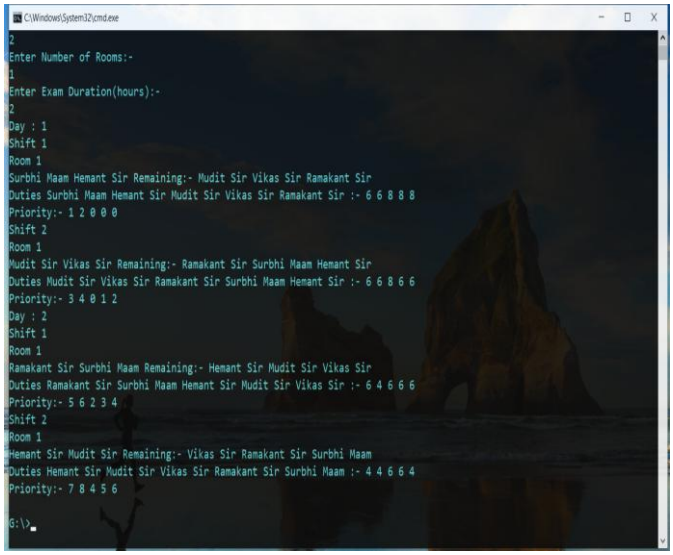

Figure 3: Screenshot of Scheduled Chart prepared by Algorithm

IV. RESULT ANALYSIS

This algorithm requires number of exam rooms, exam duration, number of days, number of shifts each day, number of invigilators etc. Priority of invigilators can be changed at the run time of the algorithm. This algorithm has running time complexity $O(n^2)$. We tested the algorithm using java code and provided the required arguments and the result was a scheduled list with faculty name along with duty hours and priority in which they were assigned the exam rooms

V. CONCLUSION

In this paper, description about that how efficiently an optimal and less complex algorithm can be developed. Here an algorithm is developed to automatically allocate invigilators to each examination rooms and all invigilators must get same number of duties. This algorithm is developed to solve an issue to allocate invigilators to each examination halls in such a way that no invigilator will get maximum or minimum duties and no invigilator will get consecutive duty in same examination hall and this allocation process is optimal and has less complexity.

This algorithm requires number of exam rooms, exam duration, number of days, number of shifts each day, number of invigilators etc. Priority of invigilators can be changed at the run time of the algorithm. This algorithm has less complexity as this algorithm's worst case complexity has been reduced from $O(n^5)$ to $O(n^2)$ since last update after November 2019.

VI. FUTURE WORK

In this paper, the proposed algorithm will be worked for only one department or branch or section of any college or university or school. So in future, this algorithm can be modified by adding multiple department's invigilator allotment facility, so that invigilators can be allocated to all department or branch or section of a school or college or university to more reduce the complexity in invigilator allocation and this algorithm can be fully modified by using machine learning, artificial intelligence and computer vision and the whole java code will be replaced by python code for that modification.

REFERENCES

1. Oluwasefunmi T. Arogundade, et. al., A Genetic Algorithm Approach for a Real World University Examination Timetabling Problem", International Journal of Computer Applications (0975 – 8887), Volume 12– No.5, December 2010.
2. Omar Al Jadaan,et. al., "Improved Selection Operator For GA", Journal of Theoretical and Applied Information Technology, 2005 - 2008 JATIT.
3. Loganathan R, Smitha Kurian, "Automated Allocation of Resources for Examination System using Genetic Algorithm", International Journal of Engineering and Advanced Technology (IJEAT), ISSN: 2249 – 8958, Volume-9 Issue-2, December, 2019.
4. Sheikh Ramzan et. al., "Intelligent Examination Staff Allotment System", International Journal of Engineering Research & Technology (IJERT), ISSN: 2278-0181, RTESIT - 2019 Conference Proceedings.
5. Introduction to Algorithms, 3rd Edition by Thomas H. Cormen , Charles E. Leiserson, Ronald L. Rivest, Clifford Stein.
6. Vikas Verma "Automatic mood classification of Indian Popular music", International Journal for research in applied science and Engineering Technology (IJRASET), ISSN : 2321-9653, Volume-5, Issue-VI, 2017.

AUTHORS PROFILE

Mr. Vikas Verma is an Assistant Professor in the Department of Computer Science and Engineering, Jaipur National University, Jaipur, India. His research areas are Machine Learning, Data Science and Automation.

Mr. Apoorv Varshney is an Student in the Department of Computer Science and Engineering, Jaipur National University, Jaipur, India. Currently He pursues his graduation. He is working in the areas Automation and Cyber security.

Mr. Santanu Banrejee is an Student in the Department of Computer Science and Engineering, Jaipur National University, Jaipur, India. Currently He pursues his graduation. He is working in the areas Automation and Machine Learning.