Student Project Allocation Problem: Three Sided Matching System

CS 254 Data Structures and Algorithms Lab Project

Made By- Aryan Verma (180001008), Srijan Saini (180001056)

Introduction

- In our day-to-day life, we usually come across situations which require two sided stable matching, a typical case of which was given by David Gale and Lloyd Shapley as Stable marriage problem.
- We consider an extension to this problem in which we try to find stable matching for three sides. It includes some constraints over some parameters, most of which are introduced in the research paper itself.
- The Student-Project Allocation problem (SPA) is a generalisation of the classical Hospitals/Residents problem (HR).
- An instance of SPA involves a set of students, projects and lecturers. Each project is offered by a unique lecturer, and both projects and lecturers have capacity constraints.

Introduction (contd.)

- The scope of the project includes two optimal linear-time algorithms for allocating students to projects, subject to the preference and capacity constraints.
- We try to implement the algorithms as efficiently as possible for finding a stable matching, given an instance of Student-Project Allocation Problem (SPA).
- Along with their implementation, we also aim to determine the time complexity and type of both the algorithms.
- The stable matching produced by the first algorithm is simultaneously best-possible for all students, whilst the one produced by the second algorithm is simultaneously best-possible for all lecturers.

SPA - Student Oriented

- This is the first instance of the SPA problem which is aligned towards students' preferences.
- It finds the stable matching in which each student obtains the best project that he/she could obtain in any stable matching.
- Initially, each student is assigned free, and each project and lecturer is assigned to be totally unsubscribed.
- Then, a Linear Time Algorithm (taking iteration of list of Lecturer, student & project together) will find the best permissible allocation of lecturers to students with a specified priority.

Algorithm (Pseudo Code)

```
SPA-student(I) {
    assign each student to be free;
    assign each project and lecturer to be totally unsubscribed;
    while (some student s_i is free and s_i has a non-empty list) {
        p_j = first project on s_i's list;
        l_k = lecturer who offers p_j;
        /* s_i applies to p_j */
        provisionally assign s_i to p_j;                    /* and to l_k */
        if (p_j is over-subscribed) {
            s_r = worst student assigned to p_j;            /* according to L_k^j */
            break provisional assignment between s_r and p_j;
        }
        else if (l_k is over-subscribed) {
            s_r = worst student assigned to l_k;
            p_t = project assigned s_r;
            break provisional assignment between s_r and p_t;
        }
        if (p_j is full) {
            s_r = worst student assigned to p_j;            /* according to L_k^j */
            for (each successor s_t of s_r on L_k^j)
                delete (s_t, p_j);
        }
        if (l_k is full) {
            s_r = worst student assigned to l_k;
            for (each successor s_t of s_r on L_k)
                for (each project p_u ∈ P_k ∩ A_t)
                    delete (s_t, p_u);
        }
    }
    return {(s_i, p_j) ∈ S × P : s_i is provisionally assigned to p_j};
}
```

Testcase

Student preferences	Lecturer preferences	
s_1 : p_1 p_7	l_1 : s_7 s_4 s_1 s_3 s_2 s_5 s_6	l_1 offers p_1, p_2, p_3
s_2 : p_1 p_2 p_3 p_4 p_5 p_6	l_2 : s_3 s_2 s_6 s_7 s_5	l_2 offers p_4, p_5, p_6
s_3 : p_2 p_1 p_4	l_3 : s_1 s_7	l_3 offers p_7, p_8
s_4 : p_2		
s_5 : p_1 p_2 p_3 p_4		
s_6 : p_2 p_3 p_4 p_5 p_6	Project capacities: $c_1 = 2$, $c_i = 1$ $(2 \leqslant i \leqslant 8)$	
s_7 : p_5 p_3 p_8	Lecturer capacities: $d_1 = 3$, $d_2 = 2$, $d_3 = 2$	

Testcase Output

```
aryan@Aryan-PC: ~/Desktop/Algo_Project
File  Edit  View  Search  Terminal  Help
(base) aryan@Aryan-PC:~/Desktop/Algo_Project$ g++ student_algo.cpp
(base) aryan@Aryan-PC:~/Desktop/Algo_Project$ ./a.out
Enter the number of students: 7
Select maximum number of projects allowed to be included in project preference list: 6
Enter project preference list (not more than maximum number of projects permissible) of student 0 : 0 6 -1
Enter project preference list (not more than maximum number of projects permissible) of student 1 : 0 1 2 3 4 5 -1
Enter project preference list (not more than maximum number of projects permissible) of student 2 : 1 0 3 -1
Enter project preference list (not more than maximum number of projects permissible) of student 3 : 1 -1
Enter project preference list (not more than maximum number of projects permissible) of student 4 : 0 1 2 3 -1
Enter project preference list (not more than maximum number of projects permissible) of student 5 : 1 2 3 4 5 -1
Enter project preference list (not more than maximum number of projects permissible) of student 6 : 4 2 7 -1
Enter the number of lecturers: 3
Enter student preference list of lecturer 0 : 6 3 0 2 1 4 5 -1
Enter student preference list of lecturer 1 : 2 1 5 6 4 -1
Enter student preference list of lecturer 2 : 0 6 -1
Enter the number of projects: 8
Enter the projects offered by lecturer 0 : 0 1 2 -1
Enter the projects offered by lecturer 1 : 3 4 5 -1
Enter the projects offered by lecturer 2 : 6 7 -1
Enter student capacity of each project: 2 1 1 1 1 1 1 1
Enter student capacity of each lecturer: 3 2 2


Project and lecturer assigned to each student are:
Student - Project - Lecturer
0         0         0
1         4         1
2         3         1
3         1         0
6         2         0
(base) aryan@Aryan-PC:~/Desktop/Algo_Project$
```

SPA -Lecturer Oriented

- This is the Lecturer -oriented counterpart of SPA Algorithm, here we will find a stable matching among Lecturers,Students & Projects in the order of Lecturers.
- Initially, all students are free, and every project and lecturer is totally unsubscribed.
- Then, a Linear Time Algorithm (taking iteration of list of Lecturer, student & project together) will find the best permissible allocation of lecturers to students with a specified priority.
- There will be some constraints over the lists for finding a stable matching.

Algorithm (Pseudo Code)

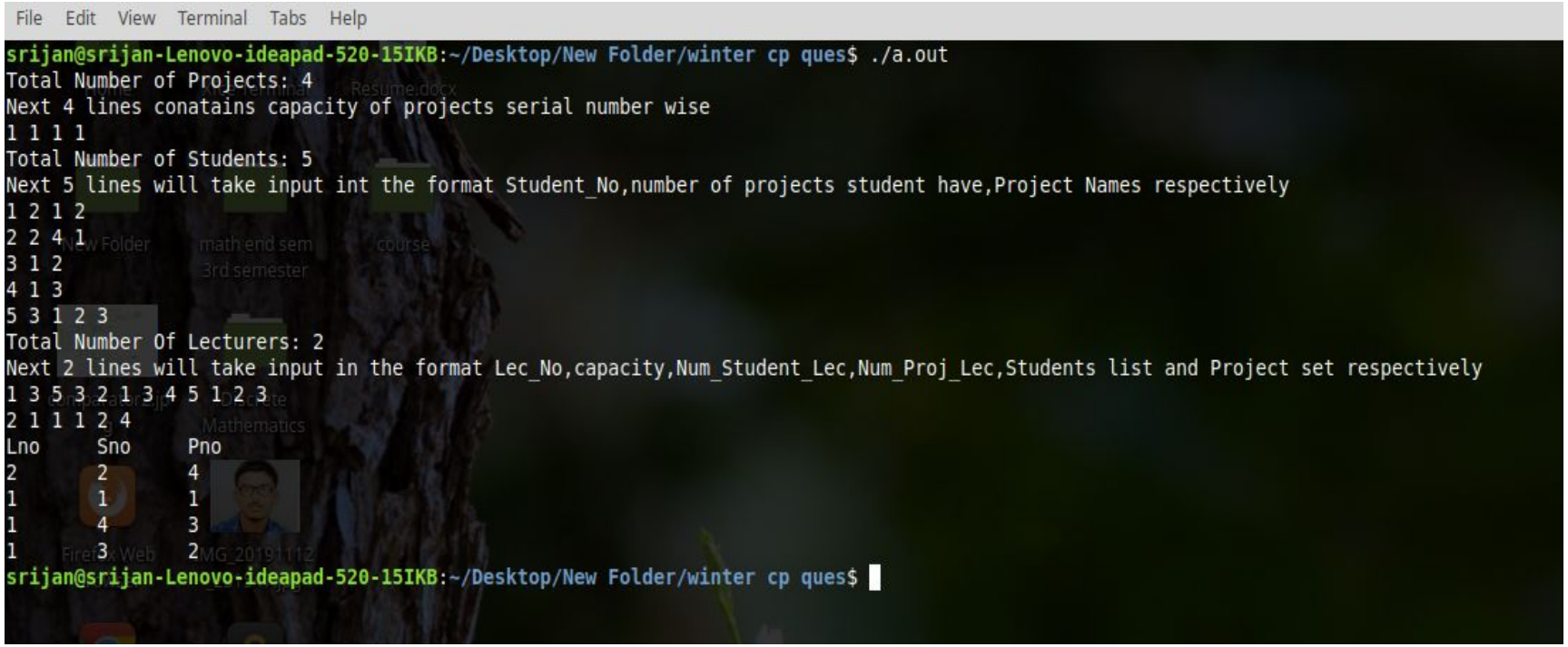
```
SPA-lecturer(I) {
    assign each student, project and lecturer to be free;
    while (some lecturer l_k is under-subscribed and
        there is some (student, project) pair (s_i, p_j) where
        s_i is not provisionally assigned to p_j and
        p_j ∈ P_k is under-subscribed and s_i ∈ L_k^j)
    {
        s_i = first such student on l_k's list;
        p_j = first such project on s_i's list;
        if (s_i is provisionally assigned to some project p)
            break the provisional assignment between s_i and p;
        /* l_k offers p_j to s_i */
        provisionally assign s_i to p_j; /* and to l_k */
        for each successor p of p_j on s_i's list
            delete (s_i, p);
    }
    return {(s_i, p_j) ∈ S × P : s_i is provisionally assigned to p_j};
}
```

Testcase

Student preferences	Lecturer preferences	
s_1 : p_1 p_2	l_1 : s_2 s_1 s_3 s_4 s_5	l_1 offers p_1, p_2, p_3
s_2 : p_4 p_1	l_2 : s_2	l_2 offers p_4
s_3 : p_2		
s_4 : p_3	Project capacities: $c_i = 1$ $(1 \leqslant i \leqslant 4)$	
s_5 : p_1 p_2 p_3	Lecturer capacities: $d_1 = 3$, $d_2 = 1$	

(i) l_1 offers p_1 to s_2; p_1 becomes full;
(ii) l_1 offers p_2 to s_1; p_2 becomes full;
(iii) l_1 offers p_3 to s_4; l_1 and p_3 become full;
(iv) l_2 offers p_4 to s_2; l_2 and p_4 become full; s_2 is freed from p_1; l_1 and p_1 become under-subscribed; (s_2, p_1) is deleted;
(v) l_1 offers p_1 to s_1; p_1 becomes full; s_1 is freed from p_2; p_2 becomes under-subscribed; (s_1, p_2) is deleted;
(vi) l_1 offers p_2 to s_3; l_1 and p_2 become full.

Testcase Output



Conclusions and Future Works

- The delete operations as described in the paper use virtual initialisation to reduce their time complexity. This concept is beyond the scope of this paper. Hence, the time complexity of deletion is also included in our code.
- As we are considering student-optimal and lecturer-optimal stable matchings, the optimal solutions of these two exist. Had we tried to seek a maximum cardinality stable matching, the problem would have been NP-hard, as explained in the paper.
- Many different formulations of the SPA model is possible. If only students supply preference lists, then the optimal matching may be constructed using network flow techniques.

References

- D.J. Abraham, R.W. Irving, D.F. Manlove

  The Student-Project Allocation problem Proceedings of ISAAC 2003: the 14th Annual International Symposium on Algorithms and Computation Lecture Notes in Computer Science, vol. 2906, Springer-Verlag (2003), pp. 474-484
- Introduction to Algorithms by Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein.