Team:PSIX

Course Registration (Algorithm)

One of the most difficult parts in making this project was to write an algorithm which would distribute the courses optimally such that there would be no one unsatisfied. Of course it is almost unreal to make all students satisfied with their course list with limited number of courses and seats in classes, but we tried to make most efficient way of distribution. As our references we have researched some known algorithms (as stable marriage problem) and tried to generalize them to our situation with which we could have reached to the optimal solution. But it was hard to convert that idea to our problem's solution because these problems differ in many ways. For example, there was only one preferred man for a woman in SMP but in our case there is unlimited preferences for a student and in SMP, men also had a right of choice but in our case the course could not choose a student to register for. Though we did not use much of that known algorithms, we got a lot of ideas from them and also were inspired by them.

First of all, let us explain how it works. Assume that we have an array of courses where for each course we have a list of students' name who wants to take that class. OK! Then, we take any course from an array and we sort students in that list in such a way to give this course to these students who really wants to get or who really needs it. Here is a piece of our code where we sort that list with a comparator function:

```
for  (it = forEachCourseSuperClass.begin() ; it != forEachCourseSuperClass.end() ; it ++) {
                vector <SuperClass> &tmp = it->second;
                sort (tmp.begin() , tmp.end() , &cmp);
}
```

And, of course, some students of that list could not get that course because of lack of seats, let us denote that students as "thrown students". Thus, we have to give them another course which they prefer and their priority relative to others (non "thrown students") will increase, in other words, "thrown students" will have a greater chance to take a course in other course selections. That's all!!!

It was an explanation of our algorithm in few easy words. But now, let's get closer to an algorithm to see how it is all done and why it works optimally and also we will try to prove why it is good for a 'thrown student' in other course selections.

How it is all done? There are few variables in Students' class which plays an important role in this algorithm. We use them to compare two students and choose one of them to take an exact course.

```
public:
    int NumCourses;
    Course *StudentCourses;
    int StudentID;
    int wasThrownStudent , CreditsLeft , SemesterLeft;
    string FirstName , LastName;
    string Major , Minor;
```

To choose one of two, we first look at their **priorities** which were given to that course by them. If they are equal we look any of these statements in random arrangement and chose student by answering them:

1. Which one of them has this course as a major?

2. Which one of them was thrown more frequently than other?

If these two match, we go to another step of statements by random arrangement:

1. Which one of them has this course as a minor?
2. Which one of them has more semesters to finish?
3. Give to each of them 1 or 0 (numbers) by random. Which one of them has greater number?

If these two match, we go to the third step of statements by random arrangement:

1. Which one of them has more credits to finish?

Thus, we finish our comparison.

Results

Hopefuly, our program runned as we expected. We compared our algorithm with an algorithm which simulates current system by running randomly generated tests on both algorithms. Got following results:

Programs	1st test	2nd test	3rd test
Constructed algorithm	89.7112	93.2823	94.2864
Simulator of current system	78.5714	85.0614	84.8996

In all cases our algorithm worked 10% more optimal than simulated system and we believe these are good results. We calculated the level of satisfaction by formula *average* = (100 / 21) * cntOfCredits. We could have also checked the level of satisfaction by priorities of distributed classes but we thought that it would get us to same results as above.

Conclusion

It was a great for us in writing this algorithm with getting an important experience while struggling with new obstacles. Hope it will benefit our community.

Team:PSIX

Team Information

Narmamatov Kadyrbek (kadirpili@gmail.com)

Donets Vladislav (dontets.vladislav.s@gmail.com)

Matkaziev Erkin (matkaziev.erkin@gmail.com)