**Team Member:**

Mao Zheng

Aanchal Tiwari

Ciara Jones

Melkam Getachew

**Project 1 report**

**GitHub link:** https://github.com/CiaraJones97/CS-404-Project

**Explanation of the ideas:**

Our team chose project 1 option , the course matching system. The algorithm tries to find the best assignment between instructor and course by using a greedy approach. It is assumed that a course contents can be grouped into five general categories. The categories are: algorithms, math, software architecture, HTML and programing. The course information is randomly generated dynamically during run time. The data is generated in terms of the percentage of contribution for each category and it is stored as a course object. An example of the course information generated is presented in the table below.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| CourseID | Algorithms | Math | Architecture | Html | Programming |
| 305 | 20 | 25 | 5 | 10 | 40 |
| 401 | 40 | 5 | 10 | 15 | 30 |
|  |  |  |  |  |  |

Table: Percentage content of a course

An instructor’s expertise is also measured based on the above five categories. For each category the instructor is rated out of 5. The randomly generated data is stored by creating an instructor object. An example of the data generated for rating an instructor is shown below.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| InstructorID | Algorithms | Math | Architecture | Html | Programming |
| 57 | 3 | 1 | 2 | 3 | 4 |
| 69 | 2 | 4 | 3 | 0 | 2 |
|  |  |  |  |  |  |

Table: Instructor rating for each category out of five.

The instructor ratings are converted into a percentage before storing it. For example, the above table will be stored as follows:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| InstructorID | Algorithms | Math | Architecture | Html | Programming |
| 57 | 60 | 20 | 40 | 60 | 80 |
| 69 | 40 | 80 | 60 | 0 | 40 |
|  |  |  |  |  |  |

Table: Percentage of instructor rating for each category.

The following assumptions were made in determining a match score between an instructor and a course.

- Instructor rating and course content percentage scales are equivalent.

- Each category will contribute a maximum of 20 % to the total score.

- The minimum score an instructor can get from a certain category is 0.

- If the instructor expertise is below course content and greater than 0, the instructor will get a value between 0 and 20, based on his level of expertise. See the attachment for detailed analysis.

It is assumed that the number of instructors and courses are equal, and one instructor teaches one course only. The optimal allocation between an instructor and a course was determined by using a greedy approach as follows.

Course objects stored in a vector

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| C1 | C2 | C3 | C4 | …… |

Instructor object stored in a vector

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| I 1 | I 2 | I 3 | I 4 | …… |

Pick the first element from the course vector. Compare the attributes, that is c1.math, c1.programming, c1.architectre, c1.html,c1.algo and find the maximum. Let say c1.algo was found to be the maximum. From the instructors vector, find an instructor that has algorithm expertise greater than or equal to c1.algo. If no instructor is found randomly choose one. Assign that instructor for c1 and remove the instructor and the course from the vector. Repeat this until the size of the vectors is equal to zero. Based on the assignments calculate the match score.

To find the optimal allocation we must repeat the above iteration many times. For each iteration we will first randomly reshuffle the course vector. After reshuffling we might get the course vector to be as follows:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| C3 | C4 | C1 | C2 | …… |

In this iteration we will do our analysis by picking C3 first. Each iteration will give a match score and finally the assignment with maximum match score will be chosen.

**Classes**

For this project, we decided to create three classes in total. These classes are:

1. Assignment
2. Course
3. Instructor

For each, this report will explain the unique functions, and how they work together.

**Assignment Class**

To start, start by explaining the functions used within the Assignment class. This class contents a instructor ID, Course ID as well as Match Score. This class includes the standard getters and setters for the variables stored within. Through this class, we can use functions to get back all these variables and set the value for each variable.

**Course Class**

Within Course class we have the variables to hold course information such as course id, course types. This class includes the standard getters and setters for the variables stored within.

**Instructor Class**

Within Course class we have the variables to hold course information such as course id, course types. This class includes the standard getters and setters for the variables stored within.

**Complexity**

The complexity of the algorithm depends on the “assign” function. This function is called n times by the main function. The function has a nested while and for loop. This makes the complexity of the algorithm to be n2. It is possible to improve the complexity of this algorithm by just reading the data from the file instead of creating it in the run time that will be able to improve the complexity to O(n).

**Functions.cpp**

We use three classes beyond to implement some future developed functions such as CalculateMathScore(..), instructorInput(...). …

* **instructorInput(vector<instructor>)**

In this function we ask the user to input the number of instructor they want and then assign the expertise level to each of them randomly. Then assign the random value to each course type.

O(n)

* **displayInstructorInfo(vector <instructor> v)**

This function will display the instructors information

O(n)

* **courseInput(vector<Course> & v, int numberOfCourses)**

This function use random function and assign course information to each one.

O(n)

* **displayCourseInfo(vector <Course> v)**

This function is used to display this course information table

O(n)

* **assign(vector<instructor> i, vector<Course> c, vector<Assignment> &a)**

This function takes 3 parameters and assign information to instrutor, course and assignments.

O(n)

* **CalculateMatchScore(instructor i, Course c, vector <Assignment> & a)**

This function is designed to be used to calculate the match score for instructor.

O(1)

**Source.cpp**

* main()

The main functions include four vectors which contains the information of courses, instructors, assignment as well as optimal assignment. Using a for loop to go through each course and then be able to use the functions above to calculate the match score for each teacher

O(n)

**Algorithm Modification**

If a professor can teach two courses and the total number of courses are 2 \* the number of teachers. To modify the algorithm, we first need to change the assignment object class. It must be changed in a way that it can store two courses.

Course objects stored in a vector

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| C1 | C2 | C3 | C4 | …… |

Instructor object stored in a vector

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| I 1 | I 2 | I 3 | I 4 | …… |

Pick the first element from the course vector. Compare the attributes, that is c1.math, c1.programming, c1.architectre, c1.html,c1.algo and find the maximum. Let say c1.algo was found to be the maximum. From the instructors vector, find an instructor that has algorithm expertise greater than or equal to c1.algo. If no instructor is found randomly choose one. Let say I 2 was chosen. Assign C1 to I 2 and remove C2 from course vector. Compare attributes of I 2 (math, programming, architecture and html) and choose the maximum. Let say I 2.math was chosen, from the course vector find a course that has math content less than I 2.math and assign it to I 2. Remove the course and the instructor from the vectors. Repeat this until the size of the vectors is equal to zero. Based on the assignments calculate the match score.