- Choose either project 1 or 2.
- **Deadline: May 9th, 2019.**
- **Refer to the project guidelines please.**
- Submit the reports and the source code and the reports on Blackboard by 11:59 PM (end of the day). For the source code and reports, I expect one submission per team, but mention the team members clearly in your submission.

Project 1 – Course Matching System

- Figure 1 shows a data model of the system.
- The courses table consists of basic information about each course.

Course ID	Course Name
303	Data Structures
490	Web Development
371	Database
550	Design and Analysis of Algorithms
201	Programming in Python

- The topics table consists of basic information about topics courses may cover.

Course ID	Course Name
1	Programming
2	HTML
3	PHP
4	C++
5	Statistics
..	…

- The CourseTopic table tells us what percentage of a course is a certain topic:

Course ID	TopicID	Percentage
303	1	20%
303	4	30%
371	2	5%
..	…	

(e.g. from the table we can tell that 20% of the data structures course is programming).

- The Professor table consists of professor information:

ID	Name
1	John Smith
2	Samir Khan
3	Ann Jackson
..	…

- The Expertise table tells us about how knowledgeable professors are at various topics. The level of expertise is a number from 0 to 5 where 0 means knows nothing and 5 means a mastery level.

Topic ID	Professor ID	Level of Expertise
1	1	5
2	1	1
2	2	5
..	…	

(e.g. from the table we can tell professor John Smith is an expert on programming, but hardly knows HTML).

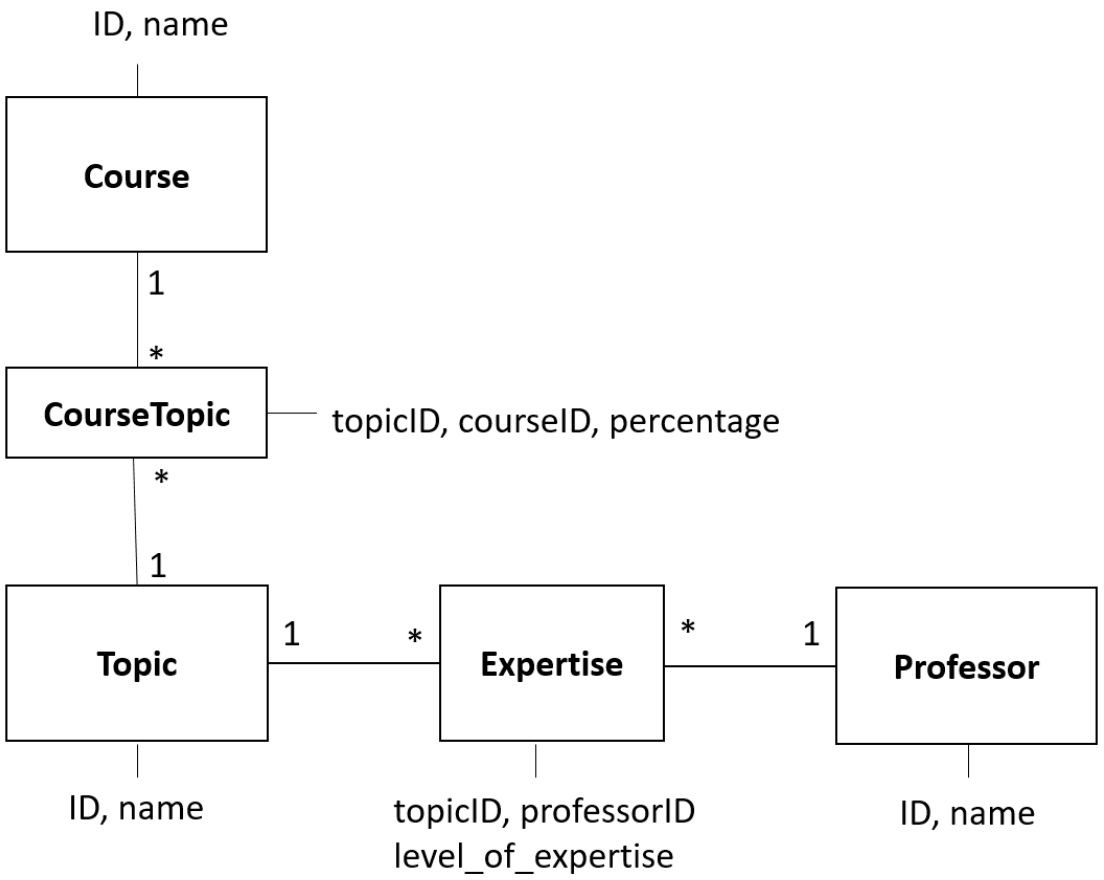

Figure 1

Technical Requirements

- Implement an algorithm that finds the best allocations for all professors and courses, i.e. the algorithm should find the best professors for courses based on the course content and the expertise of the professor. Please note that the number of professors and courses is the same. Hence, every professor must teach a course and every course must be taught by a professor. Obviously it would be ideal that every course is taught by a professor who masters its topics. However, this may not always be possible. Hence, we are interested in maximizing the totality of our course matching (meaning we want to find the best overall allocations of courses and teachers).
- Implement the algorithm in a language you are comfortable with.
- Make up the data for the project. It should be similar to the examples I gave. You can generate the files at run time, or have it stored in files, and you read it at run time.
- The algorithm should produce the course matching, e.g.:

Course ID	Professor ID	Match Score
303	1	60%
490	2	70%
371	3	40%
..	…	…

- How would you modify your algorithm if a professor must teach two courses instead?

Project 2 – Emergency Vehicle Dispatching System

- We are trying to design an emergency vehicle dispatching system. Let's assume that there are three different types of emergency vehicles: 1 (Ambulance), 2 (Fire Truck), 3 (Police car).
- Let's also assume that every request only needs one emergency vehicle.
- Here's an example of the table of EmergencyVehicles:

ID	Type	ZipCode
1	1	64151
2	1	64151
3	1	64151
4	2	64151
5	3	64151
6	3	64151
7	3	64149
…	…	…

(e.g. from the table above, we can see there are 3 ambulances, 1 fire truck, and 1 police car at 64151.

- Here's an example of the table Request

ID	VehicleType	ZipCode	VehicleID
1	1	64151	?
2	2	64149	?
3	1	51234	?
…	…	…	?

(e.g. from the table above, we can see there is a request for an ambulance at zip code 64151. The question mark means we still haven't assigned a vehicle for the request.

- Figure 3 shows the table distance, which consists of information telling us how far apart are neighboring zip codes.

ZipCode1	ZipCode2	Distance
64150	64151	4
64151	64152	2
64152	64153	3
…	…	…

(e.g. from the table above, we can see that the zip code 64150 is 4 miles away from 64151).

Note: the distance between zip codes that are not neighboring is not readily available.

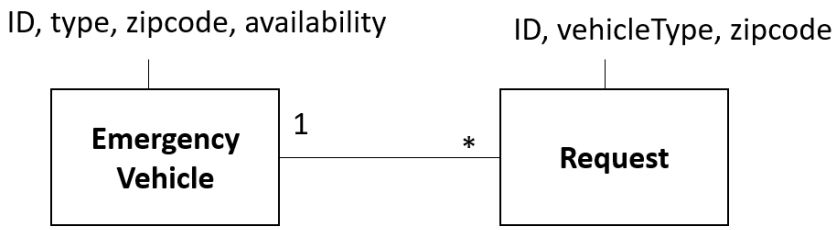

Figure 2

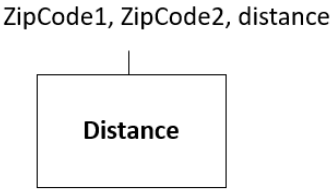

Figure 3

Technical Requirements

- Implement an algorithm that processes requests one by one. For each request, the algorithm should try to find the closest available emergency vehicle
- Implement the algorithm in a language you are comfortable with.
- Make up the data for the project. It should be similar to the examples I gave. You can generate the files at run time, or have it stored in files, and you read it at run time.
- The algorithm should produce the vehicle matching, e.g.:

ID	VehicleType	ZipCode	VehicleID	Distance
1	1	64151	1	2
2	2	64149	15	1
3	1	51234	20	3
…	…	…	…	…