Data Structures and Algorithms
Final Project
Team OG

Team Members: Hassan Abbasi, 05468

Project: Semester Schedule Builder

Objective

Prior to the beginning of every semester, students are required to manually formulate their schedules according to the courses they are required to take and their personal preferences. However, due to the poor design of the ERP system, it takes a long time to manually check all possible schedule combinations.

Moreover, coming up with alternate schedules in case of unavailability of classes makes the process even more tiresome. Taking all of this into consideration I have decided to build a WEB API which will allow students to select the courses they would like to take and receive a valid list of schedules in the form of a JSON response.

Outcomes

Send Get Request to get list of courses (1)

http://therovecompany.com:8080/MyHabibScheduler/HabibCS

Send Get Request with parameter codes to get a list of valid schedules (2)

http://therovecompany.com:8080/MyHabibScheduler/HabibCS?codes=

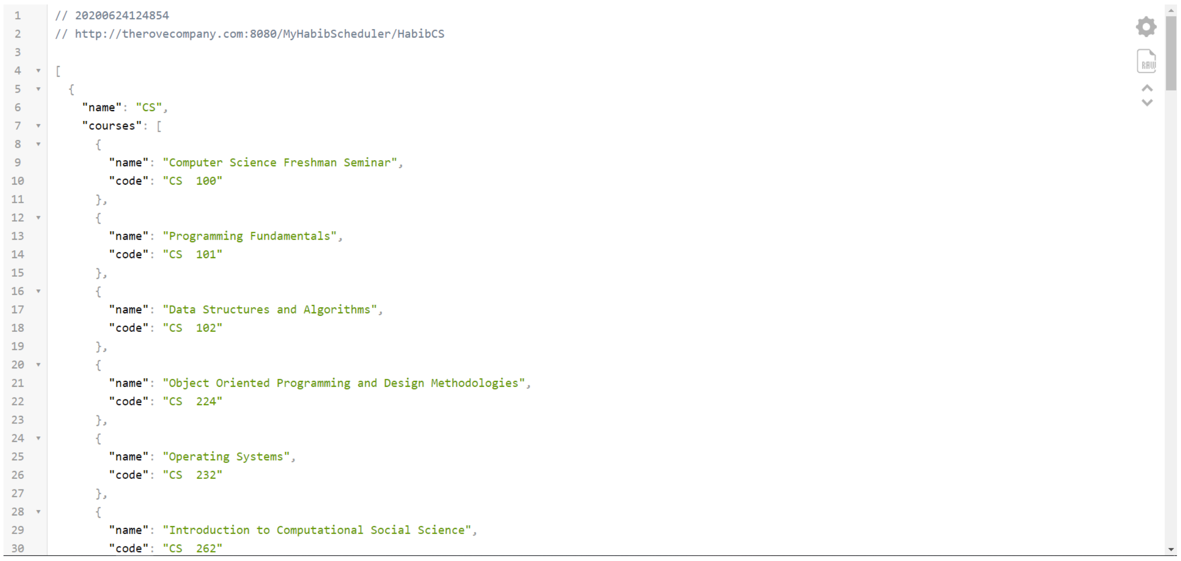
Sample Run:

User goes to (1) to retrieve the list of offered courses for the respective semester.

He then copies all the course codes (not the NBR codes but the course codes).

Finally, the user enters the copied codes as plain text after (2) and obtains all possible schedule combinations, containing the courses he has selected. Screenshots of the process have been attached below:

(1)

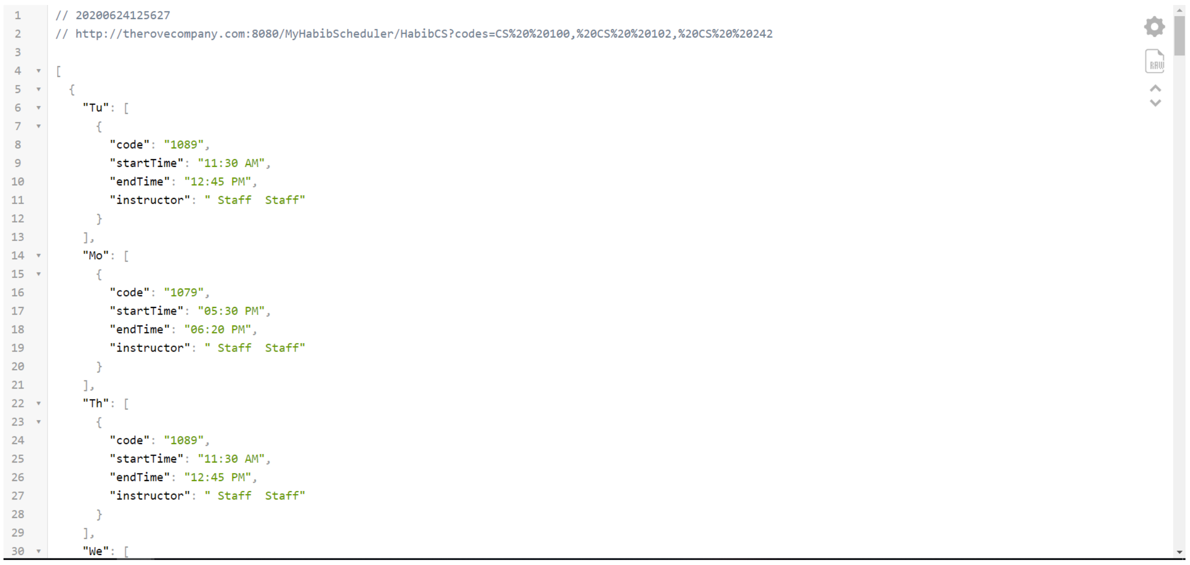


```
// 20200624124854
// http://therovecompany.com:8080/MyHabibScheduler/HabibCS

[
  {
    "name": "CS",
    "courses": [
      {
        "name": "Computer Science Freshman Seminar",
        "code": "CS  100"
      },
      {
        "name": "Programming Fundamentals",
        "code": "CS  101"
      },
      {
        "name": "Data Structures and Algorithms",
        "code": "CS  102"
      },
      {
        "name": "Object Oriented Programming and Design Methodologies",
        "code": "CS  224"
      },
      {
        "name": "Operating Systems",
        "code": "CS  232"
      },
      {
        "name": "Introduction to Computational Social Science",
        "code": "CS  262"
```

(2) URI: "http://therovecompany.com:8080/MyHabibScheduler/HabibCS?codes=CS 100, CS 102, CS 242"

As per the example, the courses selected are CS 100 (freshman seminar), CS 102 (Data Structures & Algorithms) and CS 242 (Object Oriented Programing and Design Methodologies)



```
// 20200624125627
// http://therovecompany.com:8080/MyHabibScheduler/HabibCS?codes=CS%20%20100,%20CS%20%20102,%20CS%20%20242

[
  {
    "Tu": [
      {
        "code": "1089",
        "startTime": "11:30 AM",
        "endTime": "12:45 PM",
        "instructor": " Staff  Staff"
      }
    ],
    "Mo": [
      {
        "code": "1079",
        "startTime": "05:30 PM",
        "endTime": "06:20 PM",
        "instructor": " Staff  Staff"
      }
    ],
    "Th": [
      {
        "code": "1089",
        "startTime": "11:30 AM",
        "endTime": "12:45 PM",
        "instructor": " Staff  Staff"
      }
    ],
    "We": [
```

The partially displayed list of schedules above can be accessed at the following URI: http://therovecompany.com:8080/MyHabibScheduler/HabibCS?codes=CS%20%20100,%20CS%20%20102,%20CS%20%20242

Code Documentation

The project contains a package named *com.roveapps.hScheduler*. This package contains all the relevant classes for creating a schedule. The project's main package is com.roveapps. It contains the sub package and relevant classes for creating a Java Servlet.

HScheduler:

Please find details of the classes utilized be the program to build and display schedules.

Class: Caching Service

- Functionality: Stores the document received from the Habib server and on requesting of document, provides a valid cached copy from storage, if one is available.

Class: HTML Parser

- Functionality: Parses the HTML document and extracts all data using JSOUP. The data is used to create objects of time Courses, Class etc.

Class: Networking Service

- Functionality: Connects to the Habib Server to authenticate the user and fetch authentication token. The authentication token is then persisted and used to fetch other information from the server such as the list of courses and the respective classes available.

Class: Subroutines

- Functionality: Contains global helper functions and properties such as conversion from String[] to String.

Class: Scheduler

- Functionality: Contains functions to create a graph, traverse it and evaluate If it is valid or not.

Class: User Service

- Functionality: Contains function to check if user has successfully authenticated on the server.

Object: Course

- Properties:
  - List of type Class
  - String name
  - String code

Object: Class

- Properties:
  - String Array of Instructors
  - String Array of Rooms
  - Dictionary of Timings
  - String code

Object: Timing

- Properties:
  - Date StartTime
  - Date EndTime

Main Package:

Class: HabibCS

- Functionality: Creates a servlet and checks GET request params

Class: JsonObjects

- Functionality: Contains all objects used in the JSON. Objects contains only relevant properties.

Class: Main

- Functionality:
  - Initializes the Networking Singleton
  - Fetches list of course
  - Initializes Scheduler with required courses and parses the data returned by the Scheduler into human readable schedules.

The Working

The Networking Class first connects to https://pscs.habib.edu.pk/ and attempts a login with the provided credentials and obtains an authentication token on success. It stores the token in cookies variable.

Once it obtains this authentication token, it is then allowed to access content available on the server. Therefore the Networking Class then fetches the html document containing all the list of courses by sending a POST request to the following URL https://pscs.habib.edu.pk/psc/ps/EMPLOYEE/SA/c/NUI_FRAMEWORK.PT_AGSTARTPAGE_NUI.GBL?CONTEXTIDPARAMS=TEMPLATE_ID%3aPTPPNAVCOL&scname=ADMN_CLASS_AND_COURSE_CATALOG&PanelCollapsible=Y&PTPPB_GROUPLET_ID=CLASS_CRSECATALOG_SRCH_STUDENT&CRefName=ADMN_NAVCOLL_23

After receiving the list of courses, the html data is parsed and cleaned by the HTML Parser. The data is also then structured into respective objects.

The object hierarchy of the data is as follows:-

Course: -

- Name
- Course Code
- List of object **Class**

  Each Class contains:-
  - List of Instructors
  - List of Rooms for the Class
  - Class Code
  - List of Object **Timing**

    Each Timing contains:
    - A start time
    - An end time

After creating this hierarchy of objects, the HTML Parser returns a list of objects Courses (Which contains all the sub classes stated above).

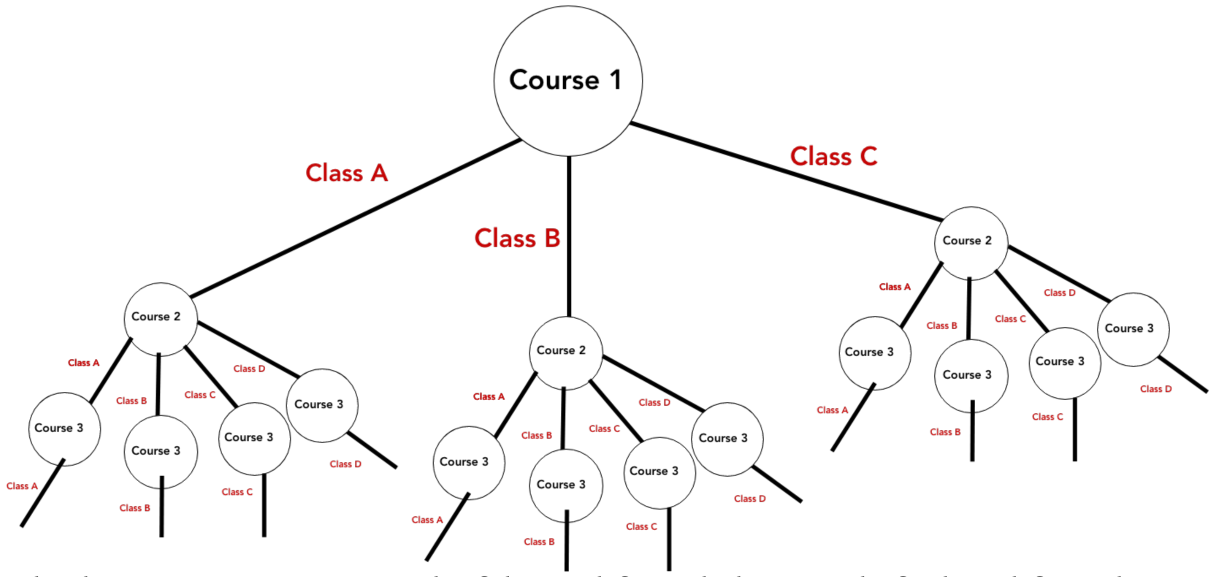
The list of courses are then used by Scheduler Class, for DFS traversal.

As the program utilizes a DFS algorithm, the final graph traversed would have looked something like this.

As can be seen, such a traversal would explore many individual classes and run code for each final schedule to determine if it contains and clashes. In such a case as each course contains n number of classes and there are m courses.

For the first course there will be a total of n traversals,
Then for each n traversal there would be further n traversals,
This would continue till the last course.

Hence the total traversals would be n^m.

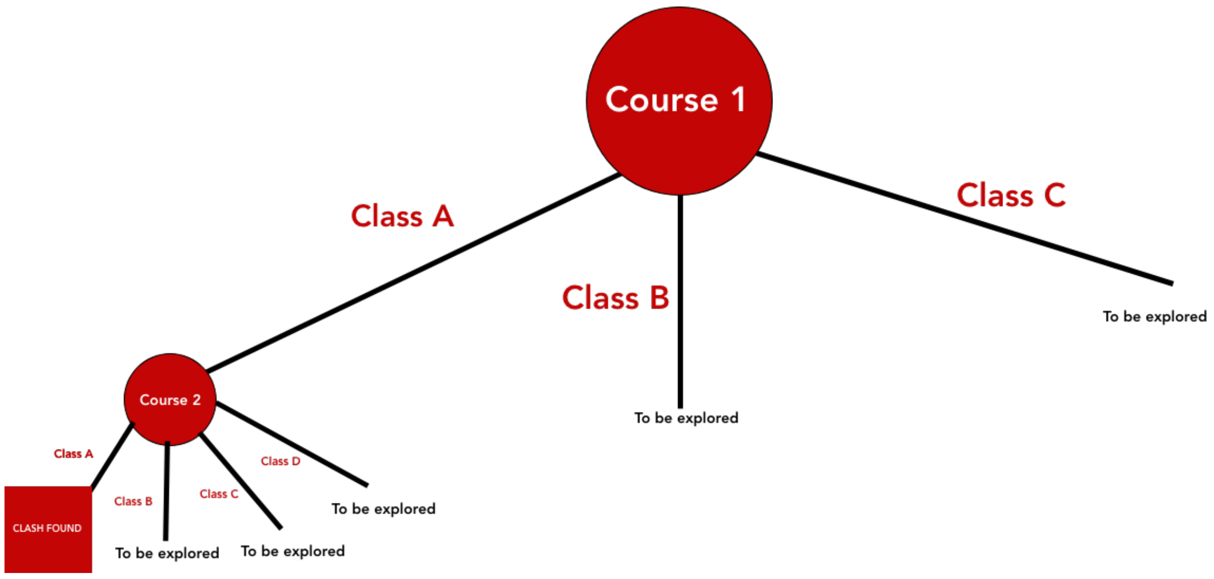


This diagram represents an example of the graph formed. This is not the final graph formed in our case.

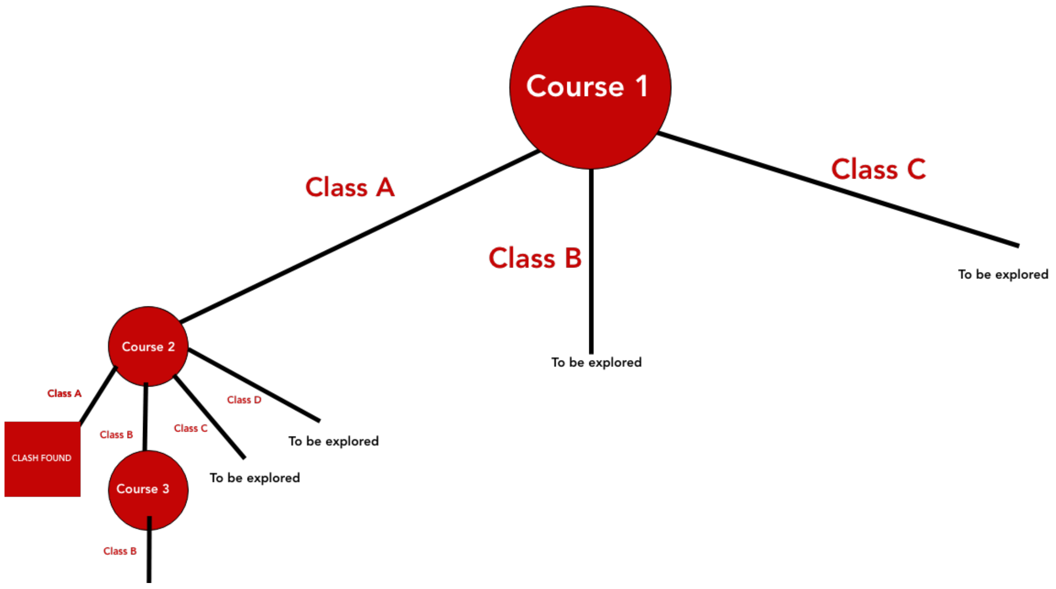
As this can be a lot of traversals, to reduce this the Scheduler class makes use of a backtracking algorithm.

Explanation of BackTracking Algorithm:-
https://www.geeksforgeeks.org/backtracking-algorithms/
https://www.cis.upenn.edu/~matuszek/cit594-2012/Pages/backtracking.html

In the case of our algorithim, the program picks the first course's first class, then proceeds to the second course and picks it's first class. However, unlike a normal DFS it does not progress further without checking for a clash of Course 2's class A with the already added class(s) (Class A of Course 1 in this case).



As a clash is found, the program does not go any further down that path and returns to Course 2's Class B and does the same execution. The program therefore does not explore as many paths, as some paths are rejected because they contain a clash early on. This reduces the number of traversals and therefore the complexity. However, it is difficult to determine the exact number of traversals skipped as it requires some use of probability to determine the likelihood of encountering a clash at an earlier stage.



The rejection of paths during exploration uses a bounding function or a constraint function/condition. In our case the constraint is provided by the method

- checkClashWithSchedule (Checks Clash with Already Added Courses)

Future Improvements:-

One short coming of the program is that it does not consider interdependent classes, such as a required lab class. Hence it does not include the timings of a required class with a particular class and treats both classes independently. Consequently, the dependency for classes is often not me. To eliminate this problem, a future release could contain a property in each class called ‘Dependency’ which could be a list of classes, that the particular class is dependent on. For example, for PFUN Class 101, the dependency list would could contain 101L