Evan DePosit

Student ID: 916260075

3/12/19

New Beginnings

Capstone Project

# Reading Group Scheduler

## Introduction

Goal

My wife works in a first grade classroom in a school where most of the students come from a low socioeconomic background and are reading below grade level. In order to improve the literacy of her students, it is necessary for her and other support staff to be able to engage with the students in small groups lessons so that students can receive appropriate instruction at their reading level. Because there is a limited number of staff, not all students can participate in these targeted groups lessons at one time. This means that when students are not participating a small group lesson with a staff member, they must be engaged in reading activities that they can complete without help from a teacher or other staff member. The goal of this program is to match groups of students with a teacher so that they can complete a reading group lesson, and match individual students with other reading activities when they are not engaged in a reading lesson.

Design Requirements and Constraints

- Each staff member is not available during the whole time that reading lessons and individual reading activities are taking place. The program must be able to store at least two time periods that each staff member is available to conduct small group lessons.

- Certain staff members are only present to teach group lessons to certain groups; the reading specialist is only their to work with the lower reading level groups.  Therefore, the user must be able to specify which groups may be scheduled with each staff member.
- The program must match each reading group's daily small group lesson with the appropriate teacher based on the staff members time availability and the user's preference for which groups are allowed work with which staff members.
- Based on the maturity and behavior of the class, the teacher changes what individual reading activities are available to students when they are not participating in a small group lesson.  Therefore, the program must offer the user flexibility regarding what reading activities are available to students.  The program must allow the user to create new reading activities for each schedule.
- For each reading activity created by the user, the program must allow the user to specify when during the scheduled reading time these resources are available.  For example, the activity, "Writing About Reading", is not available at the start of the reading time because students would have nothing to write about if they have not read anything yet.
- Some reading activities that are created by the user may require a limited number of physical resources for students to use.  The user must be able to specify the maximum number of students that can use these resources at one time, so that each student has the materials needed to complete their reading activity.  For example, one activity is listening to a book on tape.  Only three students can participate in this activity at one time because there is a limited number of headphones.

- For each activity that the user creates, the program must create the appropriate number of events on the schedule, so that the activity is taking place at the correct time and so that no more events of that activity type are scheduled than physical resources allow.
- The program must match each student with one activity for each time period during the reading schedule that the student is not engaged in a small group lesson. After matching the students' free time periods to individual reading activities, the students' schedules should be sufficiently diverse, in other words they should be engaged in a variety of the reading activities that are created by the user.

Implementation Tools

- Comma Separated Value File (csv) - because spreadsheets are a common and easy way to manage lists or schedules. I chose to have the program read in the staff members time availability from a csv file because I thought that would be an easier interface than the command line for writing out several days of time availability for each staff member. The program also reads in the class list of students from a csv file. The final output of the program is a schedule of small group lessons for each teacher, and a personal schedule for each student that shows which activity or small group lesson the student is scheduled for at each time. Both of these schedules are printed to csv files because the resulting spreadsheet provides a logical formatting that make the schedule easy for the user to read.
- Python- The constraints of the program requires creating multiple instances of custom data objects, such as students, reading groups, staff, reading activities, and events. I chose Python because I believed its object oriented classes would allow me to created the required custom data types. I also chose python because the built in string operations allowed me to code more modules in a shorter time frame, which helped

ensure that the project would be completed within the appropriate time frame.  Finally, I chose python because I wanted to become a stronger coder in a language that is ubiquitous, so that I will be able to effectively collaborate with other students or colleagues in the future

## Data Structure Design and Implementation

The *Class List* object represents all the students that are part of the classroom.  It contains a list of *Student* objects, each of which contain information about a particular student.  The *Class List* also contains a list of *Reading Groups* objects that themselves contain a list of students that is a subset of all the students in the classroom.  *Reading Group* objects have a list of activities that they must complete.   Each activity represents a group lesson that must be completed with a teacher every day.  After the bipartite matching algorithm runs, each group has a list of scheduled *Events objects*, where each event is a reading group lesson with one of the staff members.  These reading group lesson *events* are also contained on a list for each "*Student*" Objects.  After the second matching algorithm runs, *events* for individual reading activities are added to the list of *events* for each student.

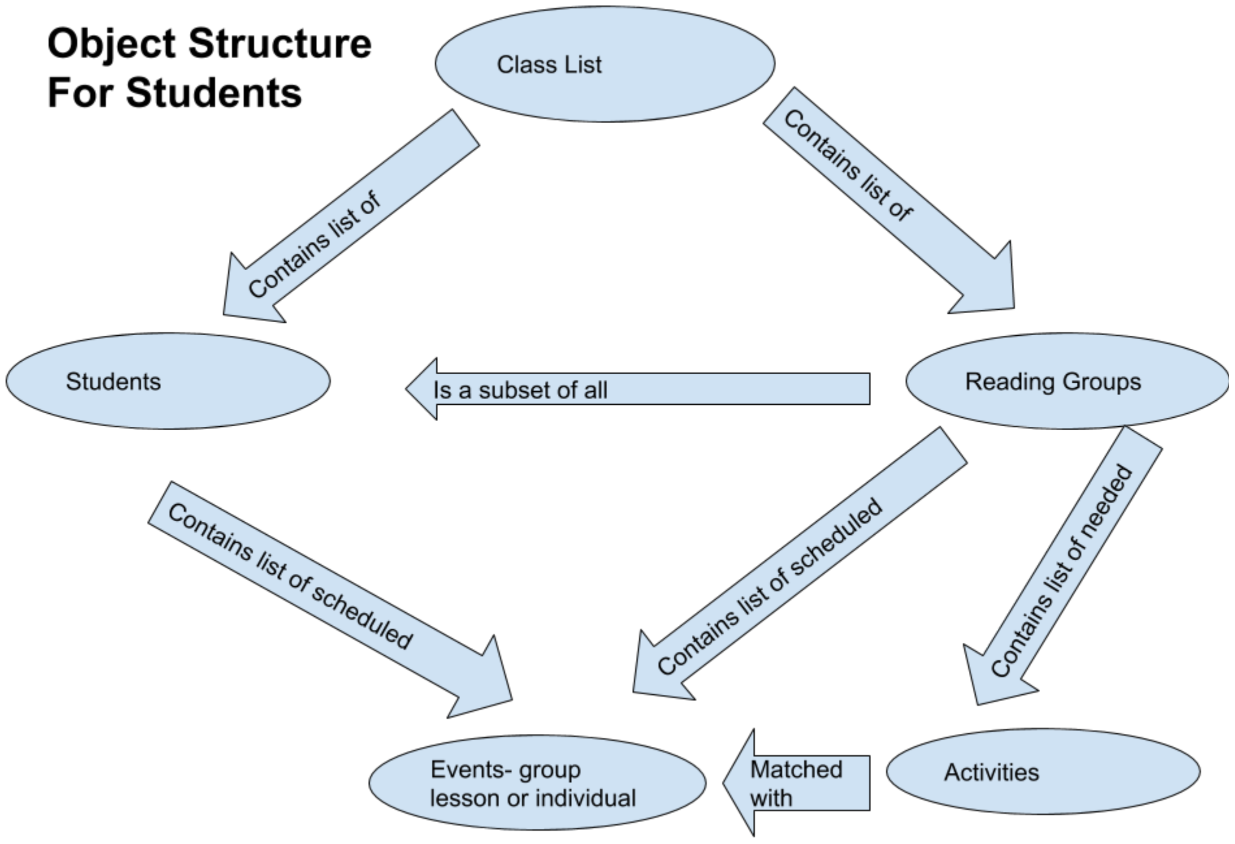
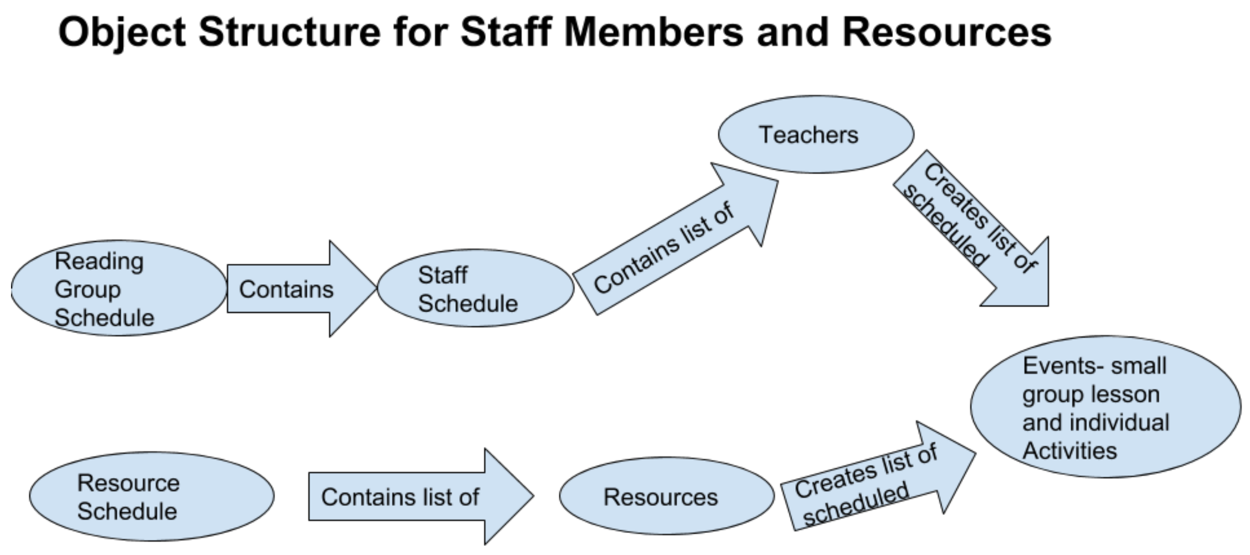


**Figure 1**: Object structure for Students shows the hierarchical relationship between class list, students, and reading groups objects and the relationship to reading events and activities.

The Staff Schedule object and the Resource Schedule object create the Event objects that become part of the schedule of the students. The Staff Schedule is part of the Reading Group schedule and contains a list of Teachers objects. For each available time in a teacher's schedule, an Event object is created for the purpose of a group lesson. Similarly, the Resource Schedule creates events for individual reading activities for each time period that the resource is available for use by students.



Object Structure for Staff Members and Resources
Teachers
Creates list of scheduled
Reading Group Schedule
Contains
Staff Schedule
Contains list of
Events- small group lesson and individual Activities
Resource Schedule
Contains list of
Resources
Creates list of scheduled

**Figure 2:** Object Structure for Staff Members and Resources shows hierarchical relationship between schedule objects and Teachers and Resource objects and how Teacher and Resource objects create Event objects.

When the program runs, the user the inputs the times associated with the schedule, such as how many days are in a week, the duration or a reading activity, and how many activities will be completed each day. The user then inputs what times the reading time starts and stops for the morning and afternoon session. The program uses this information to calculate at what time periods events will take place at for the reading activities.

Next the the program reads in the list of students to create Student objects and Reading Group objects from the students.csv file. Then the program reads in the list of teachers and their time availability from the teachers.csv file. Next, based on the time parameters entered by the user and the availability of each teacher, the program creates a list off reading group lesson Event objects for each teacher. These events are then matched to reading groups by the bipartite matching algorithm and the events are added to individual students' schedules

Next, the program creates a free list for each student that contains a list of times the student does not have any reading activities planned.  The Resource Schedule then creates individual reading activity events for each Resource object according to the times that it is available for use by the students. The second matching algorithm then matches each free period in the student's schedule with an event created by one of the classroom resources.

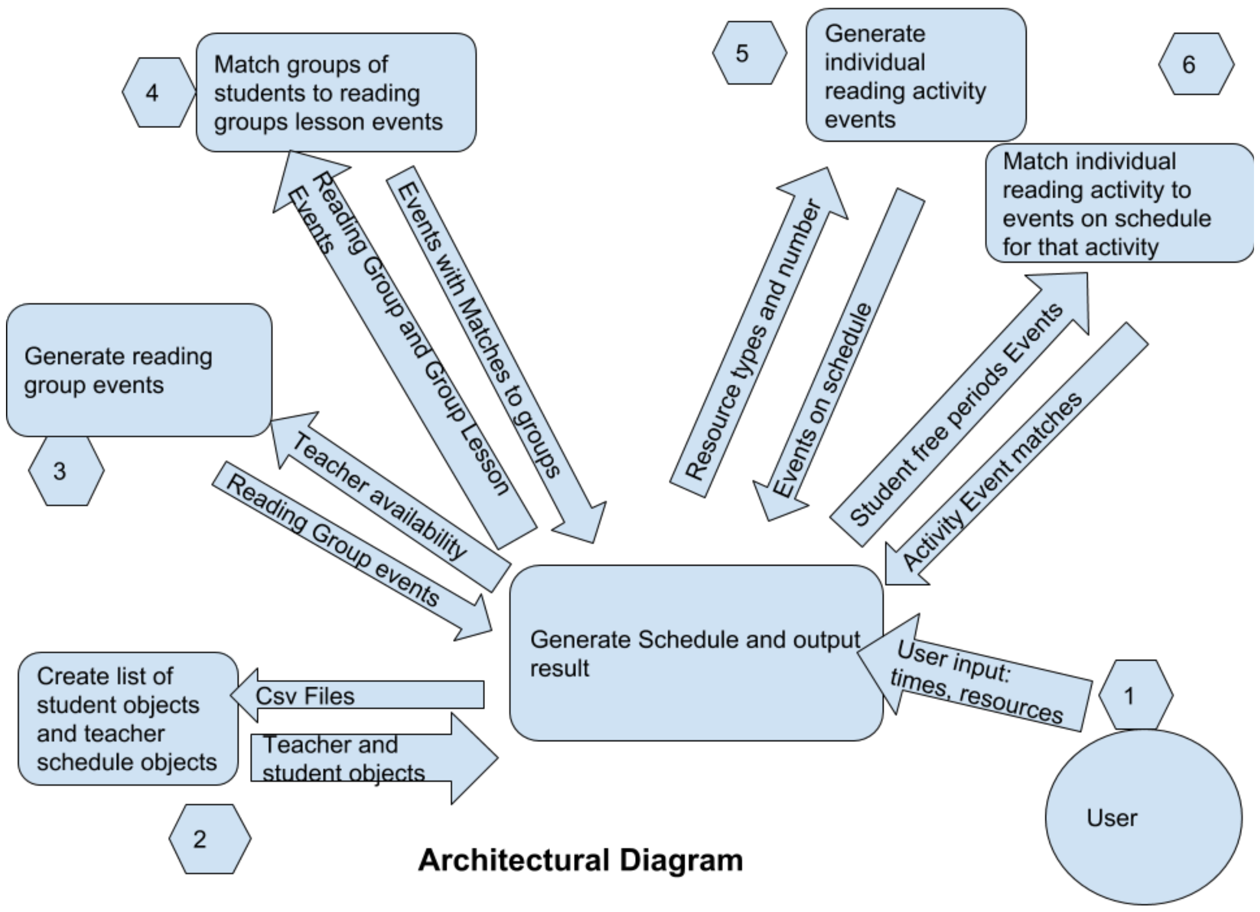

4
Match groups of students to reading groups lesson events
5
Generate individual reading activity events
6
Match individual reading activity to events on schedule for that activity
Reading Group and Group Lesson Events
Events with Matches to groups
Resource types and number
Events on schedule
Student free periods Events
Activity Event matches
Generate reading group events
3
Teacher availability
Reading Group events
Generate Schedule and output result
User input: times, resources
1
Create list of student objects and teacher schedule objects
Csv Files
Teacher and student objects
2
User
Architectural Diagram

**Figure 3:** Architectural Diagram shows the various steps during program execution

## Algorithms

<u>Maximum bipartite matching</u>

I used the Maximum Matching in Bipartite Graphs algorithm to match the required group lessons for each reading group to a group lesson being offered by a staff member. There is one set of reading group events that are offered at various times, but are unmatched to any reading group. The other set is composed of a list of required group lessons for each group. Each required group lesson has a day that it is it must take place on. Edges are constructed between group lesson events being offered by the teachers and all required group lessons belonging to a reading group that take place on the same day. Additionally, in order for an edge to occur, that teacher must be allowed to work with the reading group that the required group lesson belongs to as specified by the user.

## Matching individual reading activities to individual students

To complete each students schedule, the program must match events for individual reading activities to places in the students' schedule where no group lesson is occuring. The bipartite matching algorithm was not suited for this purpose because it would simply maximize the highest number of matches, meaning that individual reading activity events could be unequally distributed among the students. For example, one student could end up with all the "ipad' activities and another student could end up with all the "Write About Reading" activities. I needed an algorithm that would take into account how much the student needed the activity.

Initially I was going to use a modified version of Prim's algorithm for minimum spanning tree, where the weight of each edge would represent the inverse of how much the student needed the reading activity. Edges would have been constructed between the events created between physical resources and "Free Events" that existed in the students' schedule where no reading group lesson had been planned. The algorithm would then add the most needed reading activities first, and then the check for cycles would have eliminated the possibility that any student would be double booked for two events at one time. However, it occured to me that in order to implement this algorithm I would have to create low weighted edges to connect the events created by the physical resources, and I would have to add an additional check to make sure that the event or resource was not already being used by the maximum number of students.

In the end, it seemed easier to diverge from Prim's algorithms and write my own that was simply influenced by Prim's algorithm.   Each resource creates the number of reading activity events equal to the number of students that could use that resource at one time.  These events are placed in a list.  There is another list of all students' "Free Events".  These events take place at the times where no group lesson is planned for a student.  The free events are in an unordered list.  The algorithm removes a "Free Event" from the list and then finds the reading activity event created by the resource that has the lowest weight and takes place at the same time as the free event.  The weight is equal to the percent of that reading activity of that type that are currently on the student's schedule.  Because the weight changes, it is recalculated each time, unlike Prim's algorithm.  When the lowest weight reading activity is found it is placed on the student's schedule and removed from the list of events created by the resources.  The algorithm stops when all the Free events have been matched to individual reading activity events created by the resources.

## Code Statistics

Total Number of lines of code: 1206

Number of lines of borrowed code: 0

Number of lines of comments: 328

## Conclusion

The result of my project is a working program that adheres to the basic constraints of planning reading group lessons in my wife's classroom.  The program could be improved by more evenly distributing the different reading groups to different staff members.  I achieved this somewhat by shuffling the set of reading group' activities and the lesson events.  I assume this could be more effectively done by using a weighted bipartite matching algorithm.   Another shortcoming is that my second matching algorithm matches students to events based on what percent of events of that type is held by the student.  This ensures that students have an even distribution of events of different types; however, there may be times where a teacher wants students to engage in some activities more than others.  For example, a teacher may not want a student to use the ipads if they have recently used them.  I would need to modify how the weights are calculated to allow the user to dictate how often students should use various classroom resources for reading activities.

The biggest challenge I encountered when constructing my program was understanding and implementing the appropriate algorithms.  It took me a few days of rereading the same section of Levetin and watching youtube videos before I had an appropriate conceptual grasp of the Maximum Matching in Bipartite Graphs algorithm.  Once I converted the pseudo code into python, it took even longer and required  more careful study of the algorithm to work out several logical bugs in my code.  The second hurdle I faced was deciding on an algorithm to match free periods in students' schedules to individual reading activities.  I read through algorithm textbooks to look for a scheduling algorithms that seemed similar to my problem, but I could not find any algorithm that matched the unique constraints of my problem.  In the end, I was able to incorporate elements other algorithms to create my own.  I borrowed from Prim the idea of simply looking for lowest edge for each vertex in a set to expand my schedule.  To calculate the weights, I was inspired by the simple ways the scheduler in an operating system decides which process to run next.

My personal goal for this project was to become more fluent in python. I had the opportunity last term to code a wide variety of small assignments in python. However, because the assignments were completed mainly within the confines of lab time, I still felt shaky and unsure about basic python syntax and data structures. I wanted to make sure that by the time I a started my first class in the graduate program that I was confident in both python and C. This project gave me the opportunity I needed to delve deeper into python and become comfortable with its use.