Table of Contents

Section 1 - Cerebration Sessions

Unmet Needs

One challenge Dr. Perlman made clear is that it is extremely difficult to automatically schedule meetings for people in different time zones while also accounting for times of inconvenience. It is necessary to account for the most important people that need to be present for the meeting. This means prioritizing the meeting for guest speakers, members in high positions, and people with constrained time schedules. In addition, there are instances where certain time zone regions push to remain in a certain timeframe to schedule meetings. This is often the case on the east coast where the preferred times of hours are 9AM to 5PM to accommodate the west coast and Europe.

Thought Questions

- What are the exact pieces of information people need to provide in order to have a meeting effectively scheduled?
- How can we make sure that the tool is easy to use?
- How can we ensure that people will insert their information correctly?
- How will we be able to sort the information in an appropriate priority order, meaning distinguishing between those who are more or less flexible to meeting other than the originally scheduled time?
- As student coders, do we have the coding knowledge to put our idea into practice?
- Will people actually use this new software to schedule meetings, or will they stick to routine methods?
- How do we plan to display the information to the user in a user-friendly way?
- How are we going to convert all of the time zones into a single standard time?
- How do we take into account the weight of someone who is higher up in the company (i.e. Board Member, Vice President vs. Regular Employee)?

Resource Availability and Restrictions

- *Availability*
  - A team of Data Structures Teaching Assistants, Dr. Matthew Morrison, and Dr. Radia Perlman, who are all markedly great computer scientists
  - Plethora of Coding Challenge Homework assignments and In Class Code that we finished in our Data Structures course thus far to guide our team on how to implement the Data Structures we intend on using
  - Online resources, videos, and libraries on different data structures and algorithms
- *Restrictions*

- As Preternship Software Engineers, we have learned a lot about computer science, but we are in no way “fluent” in dealing with software. There are a lot of things that we do not know, and a potential restriction could be coordinating a time between four students, with someone that can help guide us in the right direction (Hopefully this scheduler can help alleviate this problem).
- Dr. Perlman mentioned to us in our first meeting that she deals mostly with assembly language code, and it may be difficult to help us implement data structures with Object Oriented Coding Languages
- Even over the first couple days of discussing the project, we have been growing the idea every day. Since we only have approximately a month to build this project, time and not being able to implement extra features that we would like to add, could get in the way
- We are still in the process of learning data structures, specifically regarding trees since we just started the unit. There may be a more ideal data structure that we can use for our project, but because we have not learned it yet, we will be unable to implement it
- Creating a user friendly, aesthetically pleasing program so everyday people can use it. Thus far, we have mainly been taking in data from files or the command line only. We may struggle to make the program look “pretty” (such as buttons HTML CSS styling etc.)

Identifying Risks and Alternatives

- These are some risks we have identified:
    - There is no automated system for scheduling meetings taking into account everyone’s availability, so there is a risk of time conflict for some of the people required at the meetings
    - Nonoptimal scheduled meetings for people in different timezones (e.g. 2am meetings)
    - People generally prefer to stick to routine, getting tenured jobholders to adapt to a new form of scheduling meeting may prove challenging
    - There may be no optimal time generated from our program for a meeting time, resulting in a failure of the program and even suggesting a time where some may need to be left out of the meeting
- And here are some solutions we are proposing:
    - Create a user-friendly interface for everyday people to schedule these meetings

    - Have the data in a pre-written text file and have the program read it in
    - Create a standard time to be able to compare availability amongst those who live in different time zones
    - If no “optimal” time is found, the best possible time will be generated by the use of the “convenience credits” that people have

Discussion and Debate

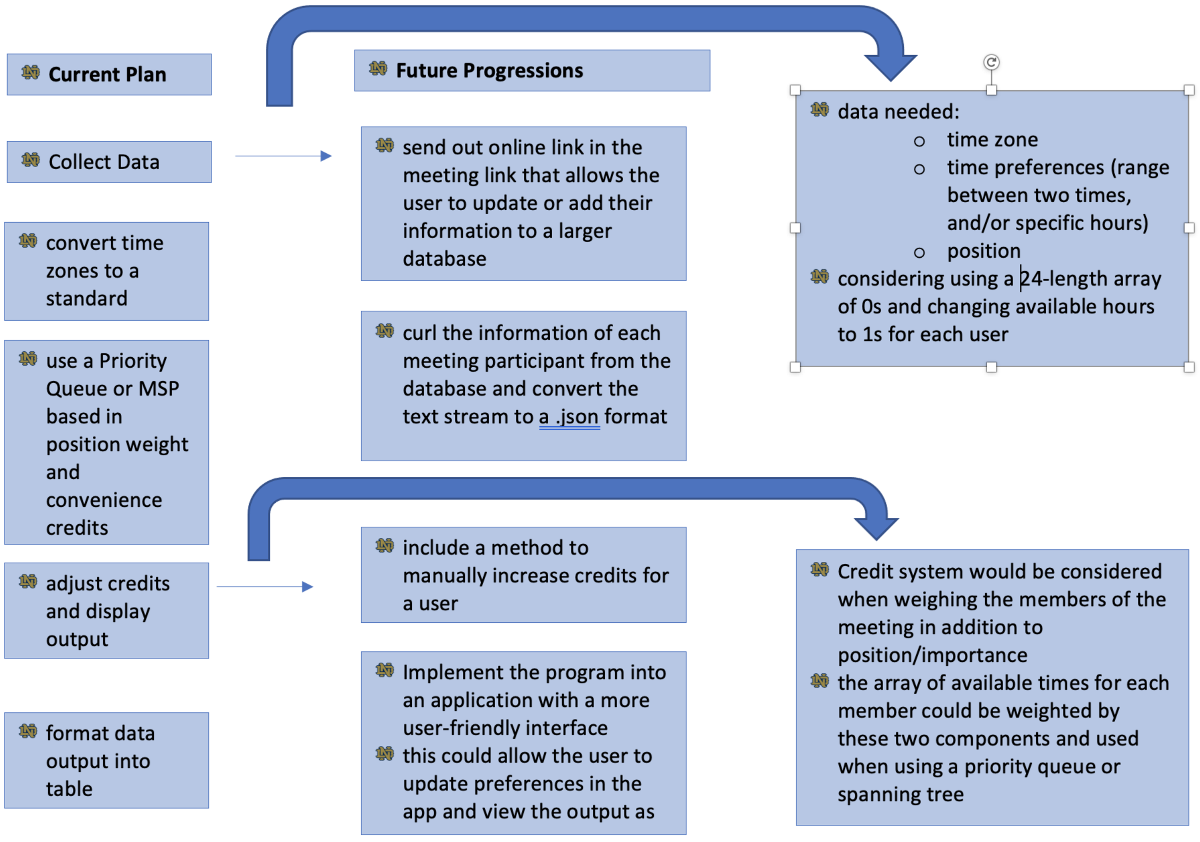
There were several concepts and ideas discussed before refining our Project Proposal. Prior to speaking with Dr. Perlman, we were also considering a project that could create chat rooms for coworkers to have an opportunity to organize informal times to converse with members in other divisions. With introverts and the current workplace pandemic environment, it can be difficult to branch out and have these conversations. While we liked the idea, there was not a clear method we could find to gather the ideal fields to connect people, and how to connect people in real time. Furthermore, in the given time for this project, it would be challenging to implement the ability to chat once the program found at least two other people. It was also noted that a more extroverted moderator might be needed to initiate conversation. Once we decided to move forward with the schedule optimizer, we discussed how to collect data and which language we would use. We decided that it would probably be good to use Python’s dictionary features to read a text stream into .json format. This series of dictionaries would be put into a priority queue or minimized spanning tree that could output the options to schedule the meeting. Another aspect discussed was the data needed for the .json included an opportunity to implement a credit feature to accommodate for meeting participants that received inconvenient meeting times in the past.

Additionally, we keep going back and forth on what language we want to use: Python or C++. The advantages of using Python are that we can use its dictionary features to read a text stream into a .json file. We also do not have to worry as much about memory, pointers, etc., as Python manages this automatically. The advantages of using C++ is that we have been using it all semester long, and we have been implementing the data structures that we plan on using. Therefore, we already have experience and examples to refer back to. We are planning on implementing the priority queue and meeting generator while using a C++ library and program to gather the data and format it into a JSON.

Finally, we discussed the best way to display the output of the program. Once reading the file, we want to print around 3 of the best times to schedule the meeting and the names of the individuals invited. The meeting times will also be displayed in a table format including the times displayed with the timezones of the members in the meeting.

Opportunities for Proofs of Concept

While we are limited by time, this project would serve as a base proof of concept to expand from. The scheduler could implement an online form, like a Google Form, for meeting attendees to add or update their information. This project could also be designed so that there is an updatable database of meeting information and the program pulls the information from the users requested in the meeting before running it through the dictionary. One final note is that the program could eventually be converted into an application user interface for ease of use and widespread usage.



- **Current Plan**
- Collect Data
- convert time zones to a standard
- use a Priority Queue or MSP based in position weight and convenience credits
- adjust credits and display output
- format data output into table

- **Future Progressions**
- send out online link in the meeting link that allows the user to update or add their information to a larger database
- curl the information of each meeting participant from the database and convert the text stream to a .json format
- include a method to manually increase credits for a user
- Implement the program into an application with a more user-friendly interface
- this could allow the user to update preferences in the app and view the output as

- data needed:
  - time zone
  - time preferences (range between two times, and/or specific hours)
  - position
- considering using a 24-length array of 0s and changing available hours to 1s for each user

- Credit system would be considered when weighing the members of the meeting in addition to position/importance
- the array of available times for each member could be weighted by these two components and used when using a priority queue or spanning tree

Section 2 - Key Terms and Definitions

- Object oriented programming: An object's own methods (functions) can access and modify the data fields of itself.
- Python: An interpreted, object-oriented, high-level programming language
- C++: general-purpose and object-oriented programming language.
- Dictionaries (key,value): A dictionary is a general-purpose data structure for storing a group of objects. It has a set of keys and each key has a single associated value. It is also known as a hash, a map, or a hashmap.
- Doubly Linked List (see “Priority Queue”): Collection of self-referential nodes, connected by pointer links, accessed by keeping a pointer to the head and tail node node, with a previous and next node.
- Priority queue: A data structure aggregated upon a doubly linked list where elements are stored and accessed in a desired sorting order.
- Minimized Spanning tree: A minimum spanning tree is a special kind of tree that minimizes the lengths (or “weights”) of the edges of the tree to find the most ideal path. One can look at it as finding the most “cost-efficient” path.
- Json format: A text-based format for representing structured data based on JavaScript object syntax.
- Data structure: A data organization, management, and storage format that enables efficient access and modification.
- Array: A homogeneous collection of data in a statically allocated memory space on the Data Heap.
- Vector: A collection of data in a dynamically allocated memory space, may be directly accessed, and is reallocated on the Heap when the memory is full.

Section 3 - Project Introduction Statement

The objective of the Dr. Perlman Preternship Project is to build a Scheduler Program that will take in user data from questions about their time zone, meeting preference times, position, and if they are presenting. It will then take this data and store it in a .json file, which we can use to store the data in a dictionary in Python. Next, we will write a function to convert all of the time zones into one standard time zone, so that we can compare what times people are available to meet and then return the overlapping times in an array where 1s are “Available” and 0s are “Busy”. Members in higher positions will have the array multiplied by a factor to weight their importance so a VP in the meeting may appear as more “people”, when comparing overlap times (eg. a VP weight of 4 would equate to 4 employees). If a common time is found amongst all meeting participants, the overlapping time will be sent to everyone in the meeting in his or her home timezone. If there is no optimal time, then a priority queue will be created to find the most optimal time. It will organize the members by position so the most important people are prioritized when the meeting time is generated. Then, the top person has their times compared with the other participants and a meeting will be created if X% of member participation is reached within ±30 minutes of their preferred time availability. If a meeting is reached, “convenience credits” will be applied to inconvenienced participants based on their position and displacement each set of 30 minutes from their preferred times. If the required meeting participant percentage is not reached, the next highest member will be compared and the top member will have their preferred available hours adjusted by 30 minutes before comparing the overlap again. This will continue down to a specified position level and if a match is not found, the meeting will not generate a schedule (this should be unlikely). Once a schedule is generated, the output will be displayed in a table including the times in all of the time zones of the members (eg. EST, CST, PST as column headers with the times below them). Lastly, there will be an override option for the higher members that will force the program to find the most overlaps for that one member without moving on to the next participant in the priority queue.

Section 4 - Design Considerations

Assumptions

- Users enter all of their information in a Google Form and it is already converted for us to JSON
- The data is correctly in JSON format prior to the program reading it in (i.e. a dictionary of dictionaries)
- The program should run correctly and efficiently on a Intel(R) Xeon(R) Silver 4208 CPU, 2.10 GHz machine using Python 3
- The output of the optimal time and the different time zones will be presented in a readable table

Project Requirements

- The program will parse the JSON file into the dictionaries (1.0) of people, their available times, and priority (position in company), converting the times to a standard time zone and creating the 64-bit bitsets for time availability (1.1).
- Using a priority queue (2.0) and weights, the optimal time will be determined based on overlapping availability times (2.1)
- If no overlapping times are found, the priority queue will pop the first member in the queue, then expand the availability times of the rest by plus and minus thirty minutes. Then, overlaps will be checked for again
- The data will be presented in a readable table (3.0) with the different time zone times for each person (3.1). Multiple options, if possible, for scheduling will be displayed (3.2)

Specifications

- (1.0) The program will rely on the JSON file created by the user availability form
    - The JSON file contains the people, available times in half hours (0-48), priority category (position in company), timezone
        - Positions include Speaker, Board Member; Vice President/Fellow; Tenured Employee; Employee
    - (1.1) Dictionaries of the people will be created with their available times in a 64 bit bitset
        - Last 48 bits: each available time counts as a 1 or true
        - First 4 bits: specify the weight or importance of the person to be used in calculating the optimal time
- The optimal time will be calculated using the people data
    - (2.0) The people will be entered into the priority queue based on the importance of them being at the meeting (position)

    - Using the first person in the priority queue, their available times will be used to find available times for the rest of the people in the queue
        - (2.1) The sum of the values in the time bitsets will be maximized
        - If there is no 100% available time, a threshold of 80% will be used to find an optimal time, otherwise, it is recommended to schedule the meeting another day
        - If there are conflicts, the first person on the queue will be popped and optimal times will be calculated based on the next most important person
- (3.0) From the determined optimal schedule time, the results will be displayed in a student machine terminal
    - (3.1) Time will be converted to each of the timezones for the people involved in the meeting in a table
    - Threshold/percentage of people able to make the meeting
    - (3.2) Suboptimal times will be displayed

Initial Risks and Alternatives

- The risks we currently see at this point are the following:
    - There is no automated system for scheduling meetings taking into account everyone's availability, so there is a risk of time conflict for some of the people required to be at the meetings
    - Non-optimal scheduled meetings for people in different timezones (e.g. 2am meetings)
    - People generally prefer to stick to routine, getting tenured jobholders to adapt to a new form of scheduling meeting may prove challenging
    - There may be no optimal time generated from our program for a meeting time, resulting in a failure of the program and even suggesting a time where some may need to be left out of the meeting
- And here is how we are currently planning on addressing them:
    - Create a user-friendly Unix command line interface for everyday people to schedule these meetings
    - Have the data in a pre-formatted .json file and have the program read it in
    - Use EST as our base, converting all other timezones into EST in order to more easily find overlaps
    - If no "optimal" time is found, the best possible time for a given threshold of participants (e.g. at least 80% of people can make it at this time) will be presented. Inconvenience credits will be given to the person(s) who are being asked to meet outside of their specified meeting times based on the number of half hour intervals.

- Compatibility:
  This project should be compatible with all stakeholders' expectations, including users, customers, developers, maintainers, and investors because this project will be very inexpensive, and easy to use. We are not infringing on any privacy, and we are not asking users to for much time at all to input data. Therefore, I see no reason for our program to be incompatible.

- Understanding:
  The architecture we are using can all be done on the Notre Dame Student Machines, and we have all of the Data Structures that we need to use at our disposal.

- Missing items:
  As of April 12, 2021, besides asking questions that arise along the way, there is nothing more that we need for the completion of this project

Section 5 - Implementation of Data Structures Course Concepts

- Dictionary
    - The use of a dictionary can effectively be used to increase the efficiency of our program. We will use a dictionary to store and access the data of names, meeting times, time zones, and company position. We will then read in the information from our JSON file using requests. This will in turn create our dictionary of dictionaries. Due to our keys being the different users of our program, searching for users will be O(1) run time. The nested dictionaries will then have keys of any of our different fields coming from the form that we send out. This makes it very straightforward to access the values of the different fields, that we will need to find possible meeting times and build our priority queue.
- Priority Queue
    - A priority queue is a data structure aggregated upon a doubly linked list where elements are stored and accessed in a desired sorting order.The priority queue will be used to search for overlaps while focusing on the most important members when trying to find the best time for a meeting. It will use the positions specified from the JSON to build the list and then pop off each member to compare it with other participants. This data structure's function will be to organize/sort the order of how the meeting will check and generate meeting times
- Array
    - An array is a homogeneous collection of data in a statically allocated memory space on the Data Heap. We will be implementing an array for each member of the meeting. We know the size of memory prior to creating and updating the array, so it does not need to be dynamically allocated. From the JSON dictionary information, each member will get an array initialized with a size of 47 (0-48) that will be updated based on their available hours. The array will be 48 instead of 24 so we can account for times each half hour. Any available time will change the initialized 0 to a 1 at those hour indexes. The arrays created for each member will be compared with other members of the meeting when searching for an overlap.
- Other
    - If we were to use C++ as the language for our code, a way to save memory would be to use bitsets when representing the availability arrays. This could be accomplished by using std::bitset, where we would be able to pack 48-bits into a 64-bit word. Alternatively, we could use a std::vector of bools, and the compiler will recognize it as a collection of bool data values and bitpack it.

Section 6 - Goals and Timeline

Each member will be spending an hour each week out of the 8 total individual hours working on the weekly coding review and memorandum assignments. This leaves 7 hours each week to work on the project coding and member goals.

- Week 1 (28 hours)
  - By the end of Week 1, we would love to have all of the data read in by the user and stored into a JSON file, formatted as a dictionary of dictionaries
  - Convert times to a standard comparison time, likely eastern time
- Week 2 (24 hours)
  - Set up the availability bitsets of each member in the meeting
  - Building the priority queue and searching through the first few members to find an optimal meeting
  - Practice with different data “popping” and seeing the potential outcomes
- Week 3 (32 hours)
  - Adding in extra conditionals:
    - Higher position meeting time override
    - Distributing “convenience credits” to impacted members
    - Priority queue limit to how far down it will check. We plan to only only two pops from the queue because we anticipate this will meet our goal of getting at least 80% of people in the meetings
  - Display output
    - Convert times to necessary time zone
    - Setup and display table
- Week 4 (28 hours)
  - Finalizing project and considering potential additions from the proof of concept into the design
    - Implementing “convenience credits” into the meeting generator when looking for overlaps
    - More user-friendly interface
    - Creating an employee database for the program to pull information from