SAT ADT Justification

Lists were used to separate each booking since each separate booking will be addressed to convert it to a graph. This eliminates the advantage of a dictionary, also the bookings don’t need to be ordered till after the graph colouring algorithm reducing the advantages of a queue or stack. Lists are also able to grow and shrink making them better than an array - room numbers will be added to the end of each booking then displayed to the user it shows which meeting is allocated to each room.

Each element in the list contains four sub-elements. The first and last element satisfy the design brief requirements of having meeting priorities and identification of the person respectively. For simplicity, the priorities range from 0-3, 0 for approved special events, 1 for the principal, 2 for being a Head of department and 3 for those who fall outside these classifications. The name field is left for the initials of the person booking the room. The middle two elements represent the start and end times of the meeting - meeting length could be used in place of end time, but I decided that it would be easier for the user to interpret with start and end times. The user inputs the time in 24-hour form and the minute input is converted to decimal form. This is done to simplify the comparison process between time values and the addition involved in determining overlap between meetings as well as calculating length.

The graph ADT was used as it is an easy way to represent meeting overlap. Each node represents a meeting and edges between nodes represents an overlap between meetings. As shown below, the meetings 1 and 2 have an overlap from 9am-9:30am and meetings 2 and 3 have an overlap from 11am to about 11:10am. This can be represented on a graph (left) with meeting 2 having 2 edges, the first connecting to meeting one and the other to meeting 3.

Chart, line chart

Description automatically generatedDiagram

Description automatically generated

Within the makeGraph() function, each booking is stored in a dictionary with the key being the node representing the booking and the value being that booking without the priority value - this was removed as the user won’t require the priority in the final output. The dictionary will be used at the end to report all the room allocations for each booking at the end of the algorithm.

This function also leverages the inbuilt attributes and storage functions in PyNode. Each node in PyNode has a name, value, label and priority field. Name could not be used since nodes with the same name can’t exist. The value field contained the start time and the label field the length of the meeting – label could have contained the end time, but meeting length was favoured as it was easier for comparison. Putting the priority in the priority field was a way to keep naming consistent.

A form of Depth First Search (DFS) will be used for the node colouring algorithm. This was preferred as it has the ability to identify circuits in the graph. To achieve this, a stack will be used in combination with a dictionary for traversal and identifying visited nodes respectively. The circuits will then be put into a list to determine which one will be deleted.