CS21120 Assignment – Wedding Planner

Zuzanna Ziolek – zuz2

Part 1: The Seating Plan

Implementation and Problems

To improve my understanding of the functions at my disposal when implementing functions like addGuestToTable() and removeGuestFromTable(), I began by implementing the smaller functions like getSeatsPerTable(). When implementing the addGuestToTable() function, I ensured that every edge case was accounted for and that it didn't contain redundant code. I encountered a few problems when implementing the Plan() constructor function because I lacked the complete understanding of the data structures I chose. After careful inspection of my code, I realised that I was only initialising the ArrayList of tables and not the set for each table. I created a for loop in the Plan() function and for every table, I initialised a set of size seatspertable.

Testing

Since I didn't add any extra functions to the Plan class, I was able to only use the provided test cases. I tested regularly to check the validity of my code and look through the test cases to ensure my code was robust and didn't just work due to technicalities.

Data Structures

I chose to use an ArrayList when implementing all the tables available at the wedding because of the included functions which allowed me to add and remove values easily. Since the tables can't contain the same person twice, I chose to implement each table as a set.

Time Complexity

addGuestToTable()	O(n)
removeGuestFromTable()	O(n)
isGuestPlaced()	O(n)

All the above functions have the time complexity of O(n) because both addGuestToTable() and removeGuestFromTable() call the function isGuestPlaced() which iterates over the ArrayList of tables that is dependent on the variable numoftables.

Part 2: The Rules

Implementation and Problems

Unfortunately, in the rules class I had to start by determining which data structures I would use instead of implementing the simplest functions first. To make this decision I had to clarify which data I would want to loop through, the rules or the guests. Since the getGuestsAtTable() function returns a set of guests, I chose to store a copy of that set in an array and loop through it. Since a rule has two factors person a and person b, I decided to use a dictionary to store each rule. This way I could utilise both the built-in functions of a set and an array.

When implementing the addMustBeTogether() and addMustBeApart() functions, I decided to create an extra function to check if a given rule exists. After trying to check both the together ArrayList and apart ArrayList in one function, I decided it made more sense to make two functions that check them separately because despite it being a virtually identical algorithm the size of the two data structures could differ. This meant I was risking not searching through some data or a stack overflow error if I chose the biggest size as the number of times to loop. Of course, all these factors could have been accounted for with the use of edge cases, but I think it also greatly improved readability.

Finally, I moved onto the isPlanOk() function which I struggled with considerably, I wrote pseudocode so I could wrap my head around the logic of what I needed to implement. The implementation that I created from this pseudocode worked for all the test cases except for the MustBeTogetherRulesPartialTable() test. I really struggled to figure out what was causing the error because it was a logic error which meant I had to read through all the code involved. I used a print statement to show what the table contained when it was

fetched in the function and found that table 1 contained A and B and table 2 contained D and C. This led me to believe there was a problem with the test cases but since I couldn't change them, I thought I had run out of options. As a last resort I checked the Assignment brief I was provided with, there I found the pseudocode for the isPlanOk() function and realised I was missing a vital condition in my if statements. The conditioned checked if the table was full and returned false as a result. After implementing this final functionality all the test cases passed

Testing

Since I added two extra functions, I needed to create Junit tests to test them and of course I used the provided tests to test the remaining functionality.

Data Structures

I chose to use an ArrayList to store all the rules and a dictionary to store the parameters of each rule, both the key and the value are strings. I chose to use a dictionary because it could only store two values which had a direct relationship with each other. This means that if I know the person a (key), I can easily look up person b (value). When adding each rule, I made sure to add two dictionaries to the ArrayList so that both person a and b where keys. This ensured that both values were able to be looked up and the value corresponding to it to be received. In hindsight I could have search through both the keys and the values instead of storing the same rule twice but in a different order. However, this would have been less efficient since I would have doubled the for loops in the isPlanOk() function.

Time Complexity

addMustBeTogether()	$O(n)$
addMustBeApart()	$O(n)$
isPlanOK()	$O(n^3)$

The functions addMustBeTogether() and addMustBeApart() call on a function which loops over the rules, once the condition is satisfied the loop is broken which means it's time complexity is O(n). isPlanOk consists of three levels of nested for loops, at the final level if a problem with the rules is found the loops are broken out of and return a boolean. This means that the time complexity is O(n3).

Part 3: The Solver

Implementation and Problems

I began by implementing the pseudocode algorithm in the brief. I some issues with it because I didn't quite understand how it needed to recurse. I decided to go through the PowerPoint which showed a step-by-step process of how the solver was supposed to work, which inspired me to create my own test cases with the same rules and guest list as the example. I used this alongside print statements which printed out the value of each important variable. In the end I went back to the provided pseudocode and noticed that I

incorrectly assumed that the code followed by an if statement was supposed to be in an else statement. After removing that I was able to get all the tests to run correctly.

Time Complexity

solve()	$O(n^n)$

The time complexity for this algorithm was slightly more complicated this is because it consisted of three nested for loops as well as recursing on itself. This meant that my drafted time complexity came to $(n^3 \cdot n^n)$. This simplifies to n^n.

Part 4: Another Technique – Simulated Annealing

What is Simulated Annealing?

The annealing procedure was used to compute the process of growing crystals to reach its absolute minimum internal energy configuration. It works by generating random points in the range of the current best point and evaluating the problems' functions there.

How is it applied in this case?

The implementation of the cost function is made by refining the "strength of the constraints" better. Instead of looping over each of the pairs of guests who need to be kept apart and guests who need to be kept together, the apart and together constraints are stored as negative and positive numbers with the larger the score the larger the effect. The imported data is converted into dictionary pairs and those pairs are interpreted by the imported library NetworkX into a graph with edges. Finally, this graph is translated using Numpy into a matrix which is filtered to show only guests with seating constraints.

The simulated annealing approach randomly generates multiple solutions and compares them to find the best one. The 'cost' of each solution is what determines the final choice. These steps are repeated until an acceptable solution is found or the maximum number of iterations is reached.

Advantages and Disadvantages

The advantage of this implementation compared to the backtracking algorithm is that it doesn't need to repeat steps since it stores the 'cost' of each solution. The biggest advantage of using the matrix reformulation is that it's possible to trace the resulting seating arrangement matrix and return a single value defining how 'good' the seating arrangement is. This implementation also allows the user to identify inconsistencies better and it is more likely to find the true global optimum.

The shortcoming of this method is the unknown rate of reduction and the uncertainty in the total number of trails. This means that, it still takes a considerable amount of time to run when the number of guests or conditions increase. At times the solution generated may not be guaranteed to exactly meet all the seating constraints. Simulated annealing is complex which means a lot of knowledge is required to understand how to adjust the constraints of the cost function. As well as this by its' nature the algorithm cannot be certain that the final solution is the most optimal.

Part 5: Self-Evaluation

What were my most notable difficulties?

The things I consecutively struggled with the most was understanding which data structures would be best to use for the implementation. I overcame these difficulties by further researching how each of my proposed data structures were used specifically in Java and what their best use cases were. I also often struggled with debugging logic errors in my code which were failing the test cases. I managed to solve all the by looking through the code of the failed tests and revisiting the brief to ensure I was implementing all the required functionality.

Which parts were implemented successfully?

I implemented all parts successfully in order of the tasks, due to my experience with programming my functions and classes are clear and well commented. There is no redundancy, and my code has optimised readability.

What mark do I expect to get?

I think due to the quality of the work I provided I deserve to get a first because code is well commented and functional. My write up is presented in a professional and concise manner and my work is well researched and explained.

Citations

Zhijing Eu - Building A Wedding Seating Plan Using Probabilistic Methods & Simulated Annealing

ScienceDirect – Simulated Annealing Algorithm