CS21120 – ASSIGNMENT

[By Jac105]

Jac105@aber.ac.uk

THE SEATING PLAN

For the seating plan, I declared two integers, one for the number of seats per table and the other for the number of tables. I then created an arrayList of sets of type String to hold each table in. I chose an arrayList over a linked list because it is better for storing and accessing data.

I then initialised these values, using a constructor, which took the parameters of numberOfTables and the amount of seatsPerTable. I needed to add the amount of tables specified, to the tables arrayList created. For this, I used a for loop to add a new hash set for the number of tables specified when the constructor is used. I used hash set instead of a tree set because the people on the table did not need to be ordered, and hash sets run faster.

The first two methods, getSeatsPerTable and getNumberOfTables were as simple as returning the integers we initialised in the constructor, seatsPerTable and numberOfTables.

I tried to code addGuestsToTable method but quickly realised I would need to check if the guest is already placed or not. I had to make isGuestPlaced method instead and I used a for loop to return true if a guest is present at each table there is and false if no tables contained this guest.

I could now deal with addGuestsToTable, using the isGuestPlaced to check whether this guest has been placed at any table yet. If they haven't and the table is not full then it will add them to the table. I then ran planTests to see if it had passed. It came up with an error about out of bounds and I realised I had to throw an out of bounds exception.

Finally I attempted to create the removeGuestFromTable method, however, I realised I needed another method to return the value of the table the guest is sat at. For this method, I used a do while loop to set a Boolean value called check to check if the current table contains the guest. After each iteration it would increase an integer counter for the current table. When the guest is found the method would return the counter value. I tested this by making my own test (included in files).

I could now easily do the removeGuestFromTable method by checking if the guest is placed, then using my new method to return the guest's table and then use this value to remove the guest from the hash set at that index of the arrayList.

Worst-Case Time Complexities:

addGuestsToTable = O(n); removeGuestFromTable = O(n^2); isGuestPlaced = O(n);

THE RULES

For the Rules class, I made two hash maps, one for friends and one for enemies. I used this because I could make each guest a string key and then their friends or enemies as a set of values.

The first method was addMustBeTogether, I had to check if either person was already a key in the hash Map, if they was then add each other to their set of friends. If they did not exist then it would create a new hash set (it did not need to be ordered) for the friends of this person and then add the other person to this set.

Afterwards, I created the addMustBeApart which was the exact same process as addMustBeTogether.

Finally, I made the isPlanOK method, which we had been given the pseudocode for. I used a for loop to iterate through every table and then a for each loop to go through each guest at that table. If the guest has enemies (has values in the hash map) then compare against other people on table using another for loop and return false if they are enemies. To make sure that all friends were at the table I had to create a new set and assign it to the set of friends of the current guest. I then used this new set to create a for each loop to check if the table contains the current friend in the loop. If the friend is not at the table return false. True will be returned if the plan is ok.

Worst-Case Time Complexities:

addMustBeTogether = O(1) constant time no matter the amount of rules

addMustBeApart = O(1) constant time no matter the amount of rules

isPlanOK = O(n^4) n being the number of rules

THE SOLVER

For this class, the constructor was given to us, so I declared the guests, plan and rules and then initialised them in the constructor.

I then had to complete the solve method, which was given to us in pseudocode. I used a for loop to go through every table in the plan and used another for loop with the initial increment counter as the size of the set of guests at the table and increment until table is full. Then, I used a for each loop to check if each guest is not placed on the current table, and if so then place them there and then using isPlanOK to check if the seating plan is valid. I then created a Boolean variable called result which was assigned to solve which attempts to recursively solve the plan with that guest added. If result is true, then true is returned and it has been solved. Else remove the guest because it can't be solved with that guest present or they would not fit there. False is then returned therefore we need to backtrack. If we get past this true is returned and we managed to fill all the seats. I had troubles following the pseudocode as I misplaced brackets and had an issue where my first for loop wasn't looping, but after some time, I found the misplaced brackets.

Worst-Case Time Complexities:

solver = O(n^8) in terms of number of guest.

- There are four loops in isPlanOK
- One loop in isGuestPlaced
- Three loops in top level

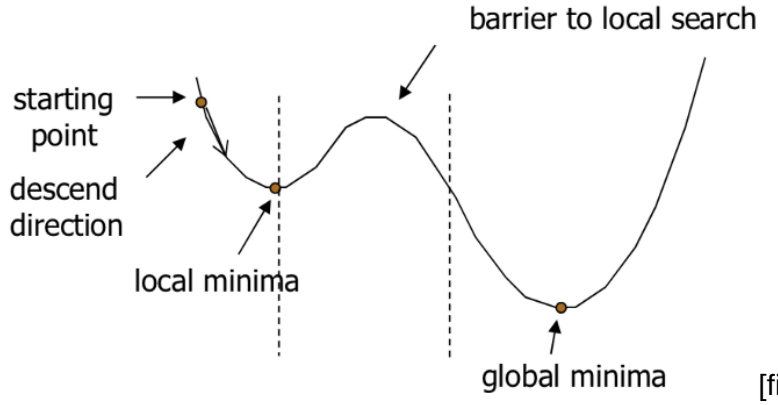
ANOTHER TECHNIQUE

HEURISTIC ALGORITHMS

A heuristic algorithm is one that is intended to solve a problem faster and more effectively than conventional approaches, but it does so at the expense of accuracy, completeness or optimality. The wedding planner is an NP problem and heuristic algorithms are used to solve NP-complete problems, a class of decision problems. [1] It is important that heuristic algorithm finds every existent solution and will only find optimal solutions even if there are many valid solutions. The algorithm must be understandable to the user and should be designed in way they can be taught quickly to non-experienced users. [2] An example how a heuristic algorithm would work for a wedding seating plan is Pick an unseated guest with a maximal number of incident edges (conflicts) If the guest has a conflict with someone seated at each table, then fail, else assign the guest at random to an available table and repeat until all guests are seated.[3]

Simulated Annealing

Simulated annealing is a form of heuristic technique that can offer a logical approximation of a global optimum for a function with a large search space. [2] Surprisingly simple to implement. It chooses a random move, and it is always approved if the chosen move makes the solution better. If not, the algorithm still makes the moves with a probability of less than 1. The probability of the move occurring decreases exponentially with the “badness” of the move (i.e. energy is increased). [4] A straightforward optimization algorithm compares the results of the objective functions running at the current and neighbouring points in the domain iteratively, saving the neighbouring point's best result as the base solution for the following iteration if it produces a better result than the current point. Otherwise, the algorithm stops the process without checking a wider range of possibilities for better outcomes. The algorithm is hence vulnerable to becoming stuck in local minima or maxima.[5]



[figure 1] showing local minima and global minima [6]

SELF EVALUATION

At first, I struggled with out of bound exceptions but they were easily tackled. Rules was the hardest class to complete as it required better understanding of the data structures to create the methods. I think I done well passing all the tests and managing to convert pseudocode into real code. Therefore, due to this and my documentation and research, I believe I will get between 75-85% because there is always ways to improve code.

REFERENCES

[1] Kenny, VK and Nathal MN, (2014 May 25) 'Heuristic Algorithms' https://optimization.mccormick.northwestern.edu/index.php/Heuristic_algorithms#Simulated_Annealing

[2] Mishra, S., Tripathy, H. K., Mallick, P. K., Sangaiah, A. K., & Chae, G. S. (Eds.). (2021). Cognitive Big Data Intelligence with a Metaheuristic Approach. Academic Press.

[3] mrip (2013 october 2) 'Which algorithm could solve my wedding table issue' https://stackoverflow.com/a/19145099

[4] Press, W., Teukolsky, S., Vetterling, W., and Flannery, B. (1992). Numerical Recipes in C, Second Edition.

[5] Erdinc, O. (2017). Optimization in renewable energy systems: recent perspectives.

[6] Liang LF. (2020 April 21) Optimization Techniques - Simulated Annealing. https://miro.medium.com/max/828/1*iXV2btukAUcn5lfd-ZjU7A.webp