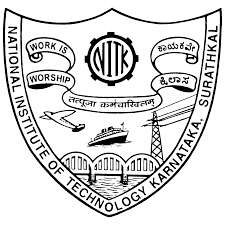
NATIONAL INSTITUTE OF TECHNOLOGY,

KARNATAKA

NH 66, Srinivas Nagar, Surathkal, Mangaluru, Karnataka 575025, India



Data Structures & Algorithms-I Lab

IT206 Odd 2022-23 Mini Project

GROUP NO: 6

Group Members:

Minank KP - 211IT040

Ekank Chhaparwal - 211IT019

Shuaib Jawid -211IT087

M Yoga Hiranmayi - 211IT038

Fahim Ahmed - 211IT020

**MATCHMAKING ALGORITHM**

| **What is the Problem?** |
| --- |
| The **Gale-Shapley algorithm**, also known as the deferred acceptance algorithm, is a method for finding a stable matching between two sets of participants, often referred to as the **"proposers"** and the **"acceptors"**. It is commonly used in college seat allocation to match students with their preferred colleges or programs.    The algorithm starts with the proposers (students) submitting a proposal to their most preferred college or program, while the acceptors (colleges or programs) review the proposals they receive and accept the proposal from the student they prefer the most. If the acceptor has already accepted a proposal from another student, they reject the new proposal. Rejected proposers then move on to their next preferred "acceptors"college or program and submit a new proposal. This process continues until all proposers have received a match.    One of the few drawbacks with this algorithm is that it is biased towards the proposers, as they are able to make multiple proposals while acceptors only have one opportunity to accept or reject.    Also, it assumes equal bargaining power between the parties, but this may not always be the case in reality. As a result, prestigious institutions have an advantage in the matching process which can lead to some lower quality applicants for less prestigious institutions.    Furthermore, it doesn't take into account the preferences of applicants who are not matched with their preferred institutions. This can lead to an applicant being matched with a less preferred institution, even though there may be other applicants who would prefer to be matched with that institution instead.    In conclusion, while the Gale-Shapley algorithm is a useful tool for creating stable matchings, it is important to consider its potential biases and limitations to ensure a fair and equitable allocation process. |
| **What is the Possible Solution?** |
| One potential solution to the biases and limitations of the Gale-Shapley algorithm would be to assign weights to each preference, and then match based on the higher preference weight first.  Additionally, matching the higher weights first would help to handle any collisions that arise between two proposals to the same acceptor. As a result, this would eliminate the bias towards the proposers, as the preferences would be matched based on weight rather than order.  Also, by implementing collision resolution, both the proposers and acceptors would have an equal opportunity to secure their preferred match and it would eliminate the problem of institutions having more bargaining power, and ensuring a fair allocation process.  **To make it easy to explain and understand we would consider the institutions and students as men and women. From here onwards, we would thus be referring everything in the form of the men-women example :**  (i)When the proposing side is men:    (ii)When proposing side is women:    If we allot weights to the entities of each side we would obtain different scores for both sides as the matchings are dependent on the proposing side.  Total points calculation in Gale Shapley Algorithm:  **Proposing Side-Men:498 Proposing Side-Women:588**   | MEN | WOMEN | WEIGHTS | MEN | WOMEN | WEIGHTS | | --- | --- | --- | --- | --- | --- | | M1 | W3 | 72 | M1 | W3 | 72 | | M2 | W9 | 80 | M2 | W9 | 80 | | M3 | W2 | 80 | M3 | W2 | 80 | | M4 | W4 | 40 | M4 | W1 | 56 | | M5 | W8 | 80 | M5 | W8 | 80 | | M6 | W6 | 8 | M6 | W10 | 16 | | M7 | W5 | 42 | M7 | W5 | 42 | | M8 | W1 | 24 | M8 | W6 | 50 | | M9 | W10 | 16 | M9 | W4 | 56 | | M10 | W7 | 56 | M10 | W7 | 56 |   As we observe there is a noticeable difference in the total points of both the matchings, it proves that there can be more than 1 stable matching and how favorable the matching is depends on the proposers side preference order.  Inorder to eliminate the biases ,the concepts of weights can be introduced where the individual with higher preference gets higher weight and vice-versa.   | **X** | y1 | y2 | y3 | | --- | --- | --- | --- | | x1 | x1\*y1 | x2\*y2 | x3\*y3 | | x2 | x2\*y1 | x2\*y2 | x2\*y3 | | x3 | x3\*y1 | x3\*y2 | x3\*y3 |   Weight based algorithm approach:  Total points =618   | MEN | WOMEN | WEIGHTS | | --- | --- | --- | | M1 | W2 | 64 | | M2 | W8 | 80 | | M3 | W6 | 80 | | M4 | W5 | 63 | | M5 | W7 | 81 | | M6 | W4 | 70 | | M7 | W3 | 24 | | M8 | W9 | 50 | | M9 | W1 | 56 | | M10 | W10 | 50 |   The weight based approach has higher po |
| **Which Data Structure to Use for the Solution (Details of the Data Structure)** |
| **The various data structures used in the implementation are as follows:**  **Vectors:**  Vectors are sequence containers representing arrays that can change in size.  Just like arrays, vectors use contiguous storage locations for their elements  Compared to arrays,vectors are very efficient accessing its elements (just like arrays) and relatively efficient adding or removing elements from its end.    **Pairs:**  Data structures that are used to combine together two values that may be of different data types. Pair provides a way to store two heterogeneous objects as a single unit. It is basically used if we want to store tuples.  Pair can be assigned, copied, and compared or even nested.    The array of objects allocated in a map or hash\_map is of type ‘pair’ by default in which all the ‘first’ elements are unique keys associated with their ‘second’ value objects.  **Maps:**  Map is the part of the STL library that stores key value pairs in it and no two values have the same keys but the different keys can store similar values. The map stores keys in sorted order.  **Queue:**  A Queue is defined as a linear data structure that is open at both ends and the operations are performed in First In First Out (FIFO) order. We define a queue to be a list in which all additions to the list are made at one end, and all deletions from the list are made at the other end.    **Priority Queue:**  A priority queue is a special type of queue in which each element is associated with a priority value. And, elements are served on the basis of their priority. That is, higher priority elements are served first.  · Every item has a priority associated with it.  · An element with high priority is dequeued before an element with low priority.  · If two elements have the same priority, they are served according to their order in the queue.    **The nested form of above data structures are used in our program implementation.** |
| **Justify Why This is the Best Data Structure for this Application?** |
| We have used Data Structures like **vector**, **map**, **pair**, **priority** **queue**, and **string**. The use of common Data Structures like vector, pair and string is very self explanatory. However, it is felt that the use of Priority Queue and Map need to be explained properly as they form the backbone of our algorithm.  We chose to use **map** in our algorithm as it stores elements in a sorted order, with each element consisting of a key-value pair. Though, this could have been implemented using some other data structure as well like array/vector but that would require us to remember the index or otherwise we had to search it, which would increase time complexity. Meanwhile, with Map we could access the element directly using the key in O(1) time complexity. We have used it to store each individual's priority of choices, which we later on use to find their best match and also help in collision resolution.  Other than that we use **priority** **queue** which allows us to implement algorithms that can process elements in a specific order, based on their priority. In our case, once our matrix is formed, we put them in our priority queue. So, the great thing about priority queue is that we can insert and delete elements in logarithmic time complexity and for accessing the top element, the time complexity is constant which greatly helps in keeping our code optimized.  We have chosen all these Data Structures keeping in mind our exact requirements for the problem and for keeping our code as optimized as possible. |
| ***Implementation Details (Experimental Setup)*** |
| We have approached our code in a function driven method i.e. everything has been segmented as different functions to maintain a clean and understandable code.  We have defined the following functions in our code and they are explained as follows:  **createPriorityForMales(n,malesList,malesPriority) :**   * The priorities of N number of Males are stored in the vector of map and also in the 2D Dynamic Matrix i.e. vector<vector<int>> X. * The priority Order of each Male is being generated Randomly using rand() function.       **createPriorityForFemales(n,femaleList,femalesPriority) :**   * The priorities of N number of Females is being stored in the vector of map and also in the 2D Dynamic Matrix i.e. vector<vector<int>> X. * The priority Order of each Female is being generated Randomly using rand() function.   **printPriorityofMen(n, malesPriority) and printPriorityofWomen(n, femalesPriority)** :  These two functions are used to print the priorities of Male and Female on the display screen.  **calculateWeights(n, femaleList, malesList, finalWeights) :**  The weights are calculated from the KEY-VALUE pairs stored in vectors of maps of both males and females and stored in the 2D vector matrix.   * At the beginning of the program the priorities of male and female are allotted using **createPriorityForMales() and createPriorityForFemales()** functions. * The priorities that have been allotted are displayed on the screen using **printPriorityofMen and printPriorityofWomen** functions. * The weights of each possible are calculated using **calculateWeights()** in order to store all the weights in a priority queue. * At the end we define the **priority queue** and store all the finalWeights and their respective pairs in that queue using the **pairs** Data structure in C++. It helps in getting the weights in decreasing order along with their respective pairs. * We run a while loop until we get the **best N number of pairs** with maximum possible weights.   Inside the while loop we have some if-else statements that are used to resolve the collisions that occurred due to same weights and two different persons wanted to match with the same person. |
| ***Complexity/Performance Analysis*** |
| The space complexity of each individual data structure is the main function is as follows:   * **malesList** and **femaleList**:   The space complexity for each of these vectors is O(n) and the space complexity of the map is O(m), so the overall space complexity for each of these vectors of maps is O(n\*m).   * **malesPriority, femalesPriority** and **finalWeights:**   The space complexity for each of these 2D vectors is O(n^2)   * **priority\_queue(pq)** :   Space complexity of priority queue is O(n)   * **male\_match** and **female\_match**:   Space complexity of vector is O(n)   * **queue(q);** : Space complexity of queue is O(n)   So overall **Space complexity** is **O(n^2 + nm +n)**    **Time and space analysis of each function**   * **createPriorityForMales** and **createPriorityForFemale**   + The time complexity of the function is O(n^2).   + The space complexity of both functions is O(n^2). * **printPriorityofMen** and **printPriorityofWomen**    + The time complexity of the function is O(n^2).   + The space complexity of both functions is O(1). * **calculateWeights**    + The time complexity of the function is O(n^2).   + The space complexity of both functions is O(1).   **Time and space complexity analysis of the main function:**  The Time complexity of the matching algorithm in the main function is O(n^2 log n). The outer loop runs in O(n) time, and the inner loop runs in O(n) time as well. In each iteration of the inner loop, a new pair is inserted into the priority queue, which has a time complexity of O(log n) due to the implementation of the underlying data structure (typically a binary heap) used by the STL's priority\_queue class. Therefore, the total time complexity of the nested loop that pushes pairs into the priority queue is O(n^2 log n). The while loop that processes the priority queue has a time complexity of O(n^2) because for each element of the priority queue, the algorithm does a constant amount of work that includes checking for matches and making new matches. The time complexity of the priority\_queue.pop() operation is O(log n) but since it's used n\*n times, it becomes O(n^2 log n). Adding up all these gives **O(n^2 log n)** for the entire algorithm. |
| ***Observations and Conclusions*** |
| The goal has been to investigate the different available concepts of pair matching algorithm and develop an improved pair matching algorithm. This goal has been achieved through developing a system using a weight-based Pair matching algorithm that is capable of solving any pair- matching problem.  The whole idea of the weight-based Pair matching algorithm is matching the preferences of n number of one party with n number of other party (where n is any counting number). In the implementation, The Preferences of n number of males are matched with the preference of n number of females.  In general, pair-matching problems like student-seat-allocation and stable-marriage-problem are tackled using variations of Gale Shapley algorithm. One of the main drawbacks of this algorithm is that it may not always find the "best" matching. For instance, it may not always lead to matches that are optimal in terms of the preferences of all the participants as it is biased to the proposing side. The weight-based Pair matching algorithm overcomes this drawback to produce the "best" matching by giving weights to preferences and then matching from higher to lower preference. It also uses the above explained collision resolution scheme to deal with any collisions and since there is no particular proposing side, no individual side is given any priority. |