**SVKM’s NMIMS**

**Mukesh Patel School of Technology Management & Engineering**

**Computer Engineering Department**

Program: B.Tech/MBATech. Sem V

A.Y. 2020 - 21

**Course: Design and Analysis of Algorithms**

**Assignment Report**

|  |  |  |
| --- | --- | --- |
| Name of the Assignment: | InfoTech: Educational Website And Blog | |
|  | | |
| Details of Project Members |  |  |
| Batch | Roll No. | Name |
| E1 | E016 | Vaishnavi Rathod |
| E3 | E047 | Riya Tendulkar |
|  |  |  |
| Date of Submission: 13/10/2020 | | |
| Grades/Marks | | |

A.1 **AIM OF THE PROJECT/ PROBLEM STATEMENT**

The aim of this experiment was to display our knowledge learnt during the course of the semester in a creative way and we have done just that! We found two interesting problems and have created an informative/educational website and blog for everyone to learn about these algorithms and implementation of said algorithms in the interesting manner of challenging problems. All of the information is conveyed in an user friendly fashion of webpages which can be easily accessed by anyone. We have provided the basic explanation of the problem, its diagrammatic representation, the different approaches to solve the problem, the optimal solution, the algorithm used, its complexity and a detailed example explained for deeper understanding. Our goal was to present these algorithms in a visually pleasing and easily understandable manner in the form of a website and blog.

We have implemented 2 problems-

1. The Assignment Problem using Hungarian Algorithm

2. Stable Marriage Problem using Gale-Shapely Algorithm

Our project contains information and detailed description of the problems as well their code implementation.

A.2 **APPLICATION /USEFULNESS OF THE PROBLEM STATEMENT CHOSEN**

Anyone who is interested in learning about interesting problem statements and applications of algorithmic concepts can refer our website. It has been our observation that for a concept to make an impact in one's brain, they need to read it many times. What better way to learn than in simple graphic explanation and simple language content. We plan to extend this project to contain many such interesting and unique problems that will help students understand the algorithm behind it. It will also help them to implement such algorithms in different day to day life. We have also published blogs on our chosen assignment topics that is available for anyone to read and understand the concept better.

A.3 **DESCRIPTION OF ASSIGNMENT**

***The Assignment Problem***

**Problem Statement:**

Let there be n agents and n tasks. Any agent can be assigned to perform any task, incurring some cost that may vary depending on the agent-task assignment. It is required to perform all tasks by assigning exactly one agent to each task and exactly one task to each agent in such a way that the total cost of the assignment is minimized.  
We need to ensure the following things:

1. The number of assignments and agents should be the same.
2. Each agent should be assigned only one job and every job should be assigned only one agent
3. We want to find the feasible solution which will consume the minimum cost

**Approaches:**

* ***Approach 1: Brute Force***  
  Here we try all the combinations one by one to find the optimal solution. This is a tedious approach because as the number of tasks and cranes go on increasing (demonstrated in the figure), the number of calculations also increase alot. The complexity is n! which is very inefficient.
* ***Approach 2: Greedy Approach***  
  In this case, the algorithm will choose the lowest cost worker to be assigned to the task as the first assignment, then choose the next lowest cost worker to be assigned to the task, and so on until all tasks have been assigned. The algorithm repeats this procedure until all workers have at least one task. Greedy algorithms try to get close to the optimal solution by improving a candidate solution iteratively, with no guarantee that an optimal solution will actually be found.
* ***Approach 3: Graph Approach***  
  The algorithm is easier to describe if we formulate the problem using a bipartite graph. We have a complete bipartite graph G=(S, T; E) with n worker vertices S and n job vertices (T), and each edge has a nonnegative cost c(i,j). We want to find a perfect matching with a minimum total cost.
* ***Approach 4: Hungarian Algorithm***  
  The Hungarian Algorithm is a combinatorial optimization algorithm which is a faster approach which solves the problem in polynomial time complexity. We see the Hungarian approach ahead.

**Hungarian Algorithm:**

1. Find the minimum from each row and subtract that value from all the elements of the row.
2. Find the minimum from each column and subtract that value from all the elements of the column.
3. Let m=minimum number of lines required to cover all the zeroes in the table
4. while(m!=number of row/columns)
   1. Find the minimum element from the uncovered elements
   2. Subtract this element from all the other uncovered elements
   3. Add this element to the elements where the lines are intersecting
   4. Find new m
5. Use the zeroes to assign possible combinations- i.e wherever there is a zero present, task can be assigned.(Use of Backtracking)
6. Find the minimum cost
7. End

**Code Implementation:**

The first 2 steps of the algorithm can be implemented very easily by traversing the array. The real challenge is the step 3.

***Use of Greedy Approach:***

Here I need to find the minimum number of lines required to cover all the zeroes. Example:

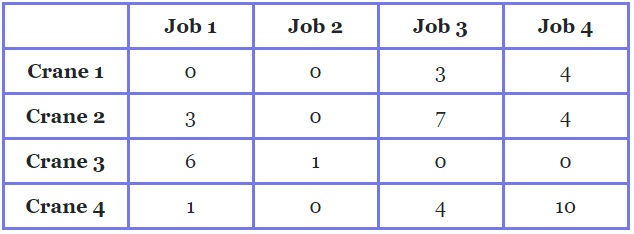
To solve this I have used the following algorithm using greedy approach. We are greedy for the maximum number of zeroes in each row/column. Algorithm used by me:

1. Count the number of zeroes in each row and column.
2. Store the index values for these zeroes.
3. Take the row or column which has the maximum number of zeroes and store this row/column
4. Remove all the zeroes having the above as row/column
5. Repeat steps 1 to 4 till the maximum number of zeroes becomes 0.

Representation of the algorithm:

Let us consider the following matrix: The row and column on the left and bottom represents the number of zeroes in each row and column.

|  |
| --- |
| Row |
| 2 |
| 1 |
| 2 |
| 2 |



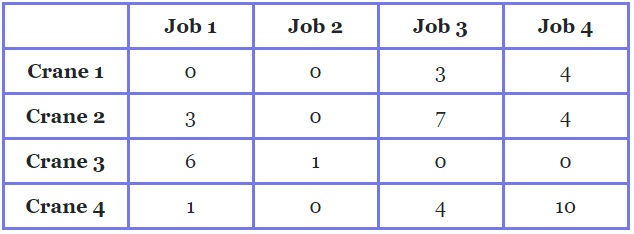
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Col | 1 | 3 | 1 | 1 |

Matrix Storing the index values of 0:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Zero Index | (0,0) | (0,1) | (1,1) | (3,1) | (2,2) | (2,3) |

* 1. Now we see that the maximum number of zeroes is in column 3(number of zeroes=3) and so now we remove all the zeroes in that row and also update the counters. Add column 1.

|  |
| --- |
| Row |
| 1 |
| 0 |
| 2 |
| 0 |

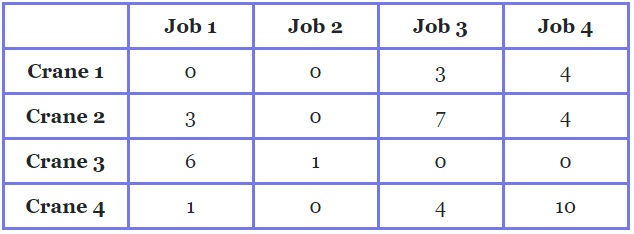


|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Col | 1 | 0 | 1 | 1 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Zero Index | (0,0) | - | - | - | (2,2) | (2,3) |

* 1. Now we see that the maximum number of zeroes is in row 2 and so we remove all the zeroes in that row and also update the counters. Add row 2.

|  |
| --- |
| Row |
| 1 |
| 0 |
| 0 |
| 0 |

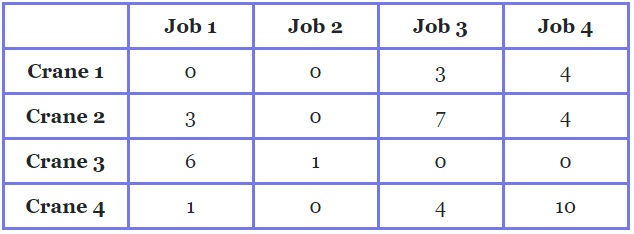


|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Col | 1 | 0 | 0 | 0 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Zero Index | (0,0) | - | - | - | - | - |

* 1. Now we see that the maximum is 1, we consider the row 1 and do the needful. Add column 0.

|  |
| --- |
| Row |
| 0 |
| 0 |
| 0 |
| 0 |



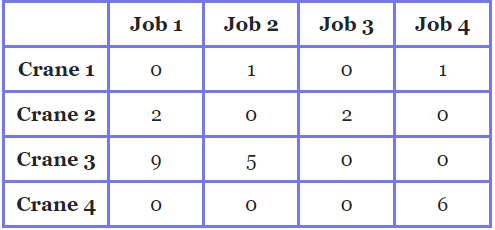
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Col | 0 | 0 | 0 | 0 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Zero Index | - | - | - | - | - | - |

* 1. Now we see that all zeroes have been covered and hence we have reached the end of the algorithm. The lines are row 2 and columns 0 and 1.

***Use of Backtracking***

The below image is a case where we have come to the optimal solution and now we need to assign the jobs. Where ever there is a 0, the task can be assigned to the respective job. We need to do this such that every job can be assigned only one task and every task can be assigned only one job. This in coding language translates as every row and column can have only one zero assigned. As we have reached the optimal solution, we know that there is atleast one zero in each row and column. To solve this I have used backtracking. If you observe the problem, it is similar to the N Queens Problem: every row and column can have only one queen and they should be placed such that the queens don’t attack each other. The only difference is that instead of considering the attack, we see that a zero is assigned.



Backtracking Algorithm:

* 1. Start from first index in every row
  2. If all zeroes are assigned, stop
  3. Try all rows in current column:
     1. If task can be assigned safely, then mark this zero as a part of solution and then go forward to check if solution is possible.
     2. If assigning the zero leads to a solution then return true.
     3. If assignment is not possible then remove this assignment and backtrack and try other rows.

A row/column will be marked if its value is 0 and it has no other marked zero in its row/column.

**Complexity Analysis:**

The complexity is O(n^3) because of 3 nested for loops. The code snippet below shows 3 nested loops.

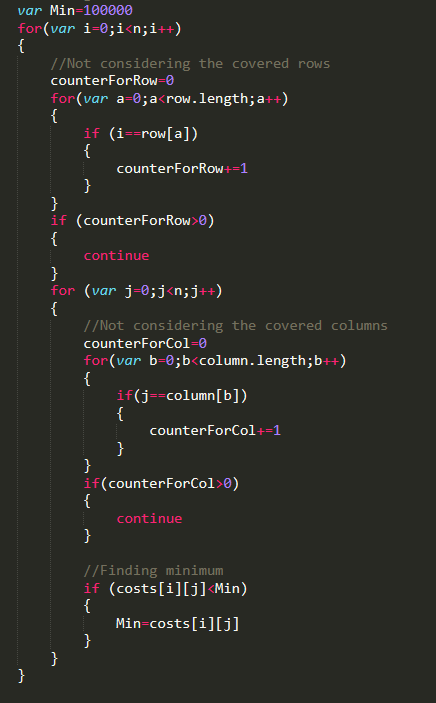


Figure 1: A part of my code having 3 nested loops.

**Code Functions:**

|  |  |
| --- | --- |
| **Function Name** | **Description** |
| Safe(n,costs,row,col) | Part of Backtracking- checks if it is safe to assign the agent to the job. |
| solveBackTrack(n,costs,col) | Backtracking to assign tasks and agents such that each task has one agent and vice-versa. |
| inputCosts(n) | Function to take the dynamic input for the costs. |
| display(n,costs,text) | Function to display each step of the output in the form of a table |
| calculation(n) | Function to do the steps of the algorithm |

***Stable Marriage Problem***

**Problem Statement:**

The Stable Marriage Problem states that given N men and N women, where each person has ranked all members of the opposite sex in order of preference, marry the men and women together such that there are no two people of opposite sex who would both rather have each other than their current partners. If there are no such people, all the marriages are “stable”.  
We need to ensure the following things:

1. The number of men and women should be the same.
2. Stable marriage exists only between a man and a woman. So, each man must give the preference of only women and vice versa.
3. We want to find the stable marriage where no two people of opposite sex that would prefer each other over their assigned partners.

**Approaches:**

1. ***Approach 1: Brute Force***

The first approach is the brute force approach where we try all the combinations one by one to find the optimal solution. This is a tedious approach because as the number of men and women will go on increasing and the matching combinations will be a lot to get through.

1. ***Approach 2: Gale Shapely Algorithm***

The algorithm is much faster and yields better results. The idea is to iterate through all free men while there is any free man available. Every free man goes to all women in his preference list according to the order. For every woman he goes to, he checks if the woman is free, if yes, they both become engaged. If the woman is not free, then the woman chooses either says no to him or dumps her current engagement according to her preference list. So an engagement done once can be broken if a woman gets better option. Time Complexity of Gale-Shapley Algorithm is O(n2).

**Gale-Shapely Algorithm:**

1. Begin with every man and woman being free.
2. While there exists a free woman w, she proposes to her most prefered man m whom she hasn't proposed to yet as follows:
   1. w proposes to m
   2. if m is free: m and w get engaged and (m,w) is added to S
   3. if m is currently engaged to another woman, w' and he prefers w' over w i.e. M(w')>M(w), then w remains free.
   4. if m is currently engaged to another woman, w' and he prefers w over w' i.e. M(w)>M(w'), then m breaks off engagement with w' and gets engaged to w.(m,w') is removed from S and (m,w) added to S. w' becomes free
3. Return S as stable marriage.
4. End

**Code Implementation:**

***Logic-***

We take preferences input from N men and N women for this problem. The idea is to iterate through all free men while there is any free man available. Every free man goes to all women in his preference list according to the order. For every woman he goes to, he checks if the woman is free, if yes, they both become engaged. If the woman is not free, then the woman chooses either says no to him or dumps her current engagement according to her preference list. So an engagement done once can be broken if a woman gets better option.

***Javascript-***

First we take user input for the no of men and women. In function people(), we dynamically create fields for them to enter their choices. After entering everything needed, we move on. In function preferences we store all the choices in array. This will help us in matching later. Then we go to function marriage() which is the main part of the problem. We iterate to find any free men. Once we find them we go over their preferences to match them with a woman. This way the matching takes place. But the marriage isn’t stable yet. We have to also check if the woman is happy with her marriage. If she prefers some other men who is free over her partner, then she can go to him. All of this happens in function wPrefersM1OverM(). When we finally achieve the stable marriage needed, we display it to the user dynamically using html in javascript.

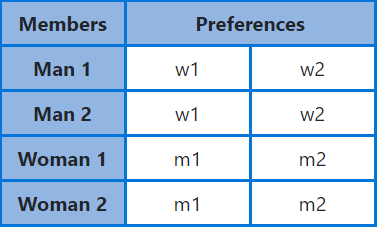
**Algorithm applied:**

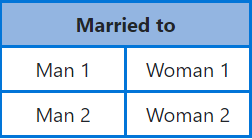
1. Start
2. Initialize all men and women to free
3. while there exist a free man m who still has a woman w to propose to

* {
* w = m's highest ranked such woman to whom he has not yet proposed
* if w is free
  + (m, w) become engaged
* else some pair (m', w) already exists
* if w prefers m to m'
  + (m, w) become engaged
  + m' becomes free
* else
* }

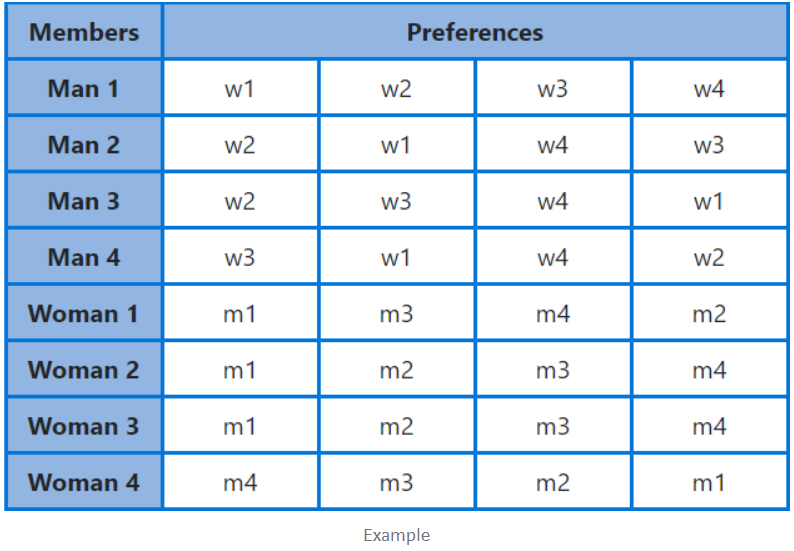
1. End

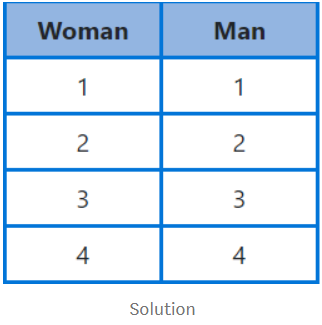
***Example 1:***





***Example 2:***





**Complexity Analysis:**

The run time complexity of Gale Shapely's algorithm is given as **O(N^2)**

**There are no more that n^2 proposals during a run of the Gale-Shapely algorithm**

No woman proposes to a man more than once. If the all n women propose to all n men, there would be n^2 proposals. Since no woman proposes to a man more than once, there can’t be any additional proposals.

**Each proposal can be performed in O(1) time**

This can be achieved by using certain data structures to maintain the needed information. Comparing preferences could take O(n) time if we only use the preference lists, but by storing an array of preferences indexed by people this can be performed in constant time. Setting up this array took Θ(n^2) time during setup, but results in a faster runtime within this loop. It is common in algorithm design to take more time on setup which is executed once to save time on steps that are executed many times such as the body of a loop..

**Code Functions:**

|  |  |
| --- | --- |
| **Function Name** | **Description** |
| Persons() | Creating HTML elements on the screen for dynamic input of couples and taking their preference inputs. |
| Preferences() | Creating a double dimensional array to store the preferences inputted by men and women. |
| Marriage(prefer) | Here we try to match the free man with the women of his choice, if she is available. |
| wPrefersM1OverM(prefer, woman, man, man1) | Function to check if the woman prefers some other man than the one she is paired with. |

A.5 **CONTRIBUTION OF EACH PROJECT MEMBERS:**

|  |  |  |
| --- | --- | --- |
| **Roll No.** | **Name** | **Contribution** |
| E016 | Vaishnavi Rathod | Stable Marriage Problem |
| E047 | Riya Tendulkar | The Assignment Problem |

A.6 **MODE OF PRESENTATION**

Our presentation was in the form of a website where we have explained the problems as well as have coded the algorithm. We had a total of 6 pages, details of which are:

|  |  |  |
| --- | --- | --- |
| Page Name | Language | Description |
| Home Page | HTML/CSS | A home page giving information about the basic paradigms of Algorithms and links to individual pages |
| Assignment Problem- Information | HTML/CSS | Informative website with information of the problem, example, applications, approaches, algorithm, example using algorithm. |
| Assignment Problem- Implementation | HTML/CSS/JS | Implementation of the algorithm with dynamic input and output and display of each step. |
| Stable Marriage Problem-Information | HTML/CSS | Informative website with information of the problem, example, applications, approaches, algorithm, example using algorithm. |
| Stable Marriage ProblemImplementation | HTML/CSS/JS | Implementation of the algorithm with dynamic input and output. |
| About Us | HTML/CSS | A little bit about us, the creators of this website |

**We have also *published* our individual algorithms information in the form of blog.** Links are:

1. Stable Marriage problem: <https://medium.com/@vaishnavirathod311/gale-shapely-algorithm-stable-marriage-problem-d96eba632f8e>
2. Assignment Problem: <https://medium.com/@riya.tendulkar16/the-assignment-problem-using-hungarian-algorithm-4f105729af18>

A.7 **LEARNING FROM THE ASSIGNMENT**

This assignment has been an eye opener as to how the information learnt in class can be implemented in real day to day life. Often we learn only technical things in the college curriculum but we had the freedom to select whatever we want for this experiment and so it was refreshing to code and learn about fun and interesting problems. We have learnt the following things as well:

1. Understanding of real-life problems
2. Applying the concepts learnt in the class to problems
3. Analyzing the complexity of the algorithms
4. Choosing appropriate algorithms and approaches to solve problems
5. Coding the appropriate logics such that the complexity is minimum

A.8 **CHALLENGES YOU FACED WHILE DOING THE ASSIGNMENT**

Assignment Problem (Riya) - The implementation of my chosen algorithm was not available on the internet at all, so I had no idea where to start. I have written the entire code using logic of my own and it was challenging to code the part where I had to find the minimum number of lines and the lines themselves required to cover zeros in each row and column. At first I was stuck, but then I used the “Greedy Approach” to solve the problem and was finally able to resolve the issue.

Second challenge was coding complex algorithms in Java Script, a language which was very new to me as it was taught to us in this semester itself. Using JavaScript I have dynamically taken the inputs as well as displayed every step of the output, which changes for each problem.

Stable Marriage Problem (Vaishnavi): Since I have been mostly coding in C++, Java and python it was difficult to try to implement the logic in a new language. Coding on Javascript meant taking dynamic inputs and creating that many html elements to display the outputs. So even though I understood the logic of the program, I had to code from scratch for WebD implementation of the problem.

A.9 **CONCLUSION**

**Working on such an assignment was very refreshing and educative. There were absolutely no limitations for us to soar to new heights with our creativity. This assignment helped us learn many things like user interface, implementing logic in front end and learn in depth about the chosen algorithms. It also gave us an opportunity to give back to the society and share our knowledge with any and everyone.**

Note:

1. Create a readme file if you have multiple files
2. All files must be properly renamed
3. All functions and variable should have proper names
4. The code if any must be properly commented.
5. Submit all relevant files of your work.

**Focus on uniqueness, plagiarism is highly discouraged**