ANALYSIS OF ALGORITHMS 2

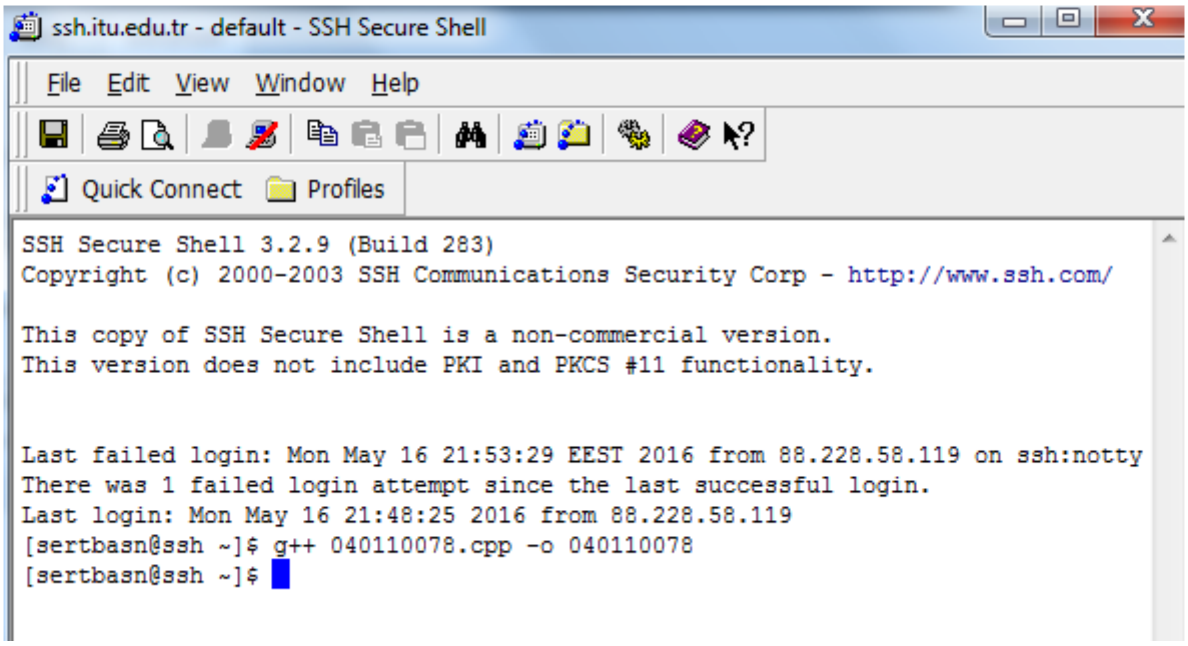
# PROJECT 3

Nurefşan Sertbaş
040110078
sertbasn@itu.edu.tr

A. INTRODUCTION

I implemented an algorithm that either finds a feasible allocation of all objects or determines that no such assignment exists. The given problem is solved as Maximum Bipartite Matching using Ford Fulkerson Algorithm in c++.

First I compiled and executed my code in SSH in order to see if there exist any error. The sample execution output is shown in below figure.


ssh.itu.edu.tr - default - SSH Secure Shell
File Edit View Window Help
Quick Connect
Profiles

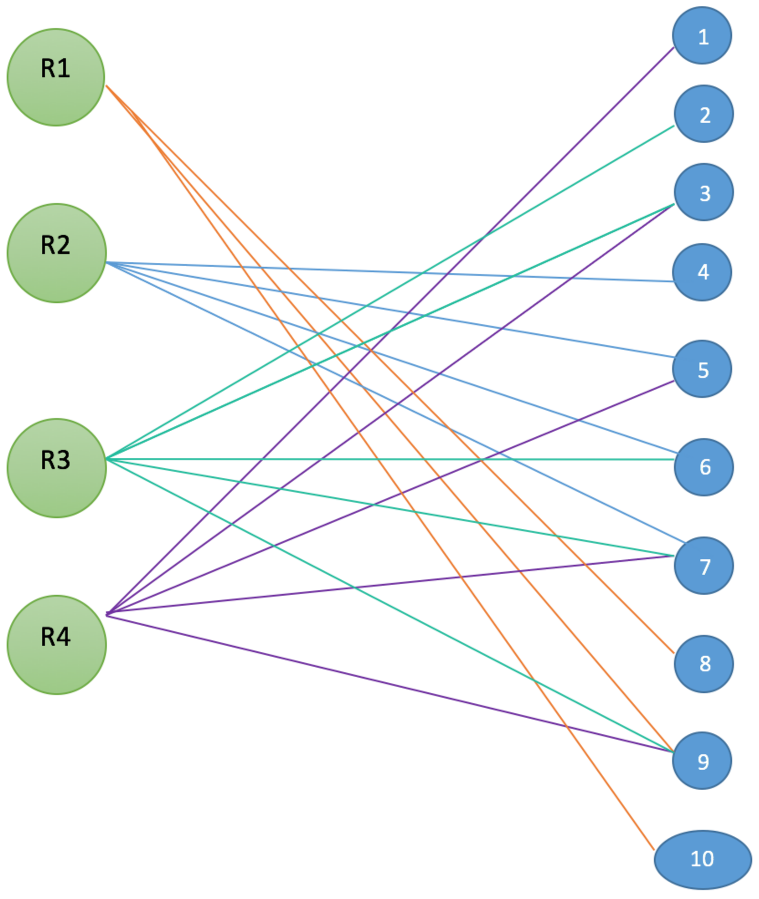
```
SSH Secure Shell 3.2.9 (Build 283)
Copyright (c) 2000-2003 SSH Communications Security Corp - http://www.ssh.com/

This copy of SSH Secure Shell is a non-commercial version.
This version does not include PKI and PKCS #11 functionality.


Last failed login: Mon May 16 21:53:29 EEST 2016 from 88.228.58.119 on ssh:notty
There was 1 failed login attempt since the last successful login.
Last login: Mon May 16 21:48:25 2016 from 88.228.58.119
[sertbasn@ssh ~]$ g++ 040110078.cpp -o 040110078
[sertbasn@ssh ~]$
```

B. BACKGROUND INFORMATION

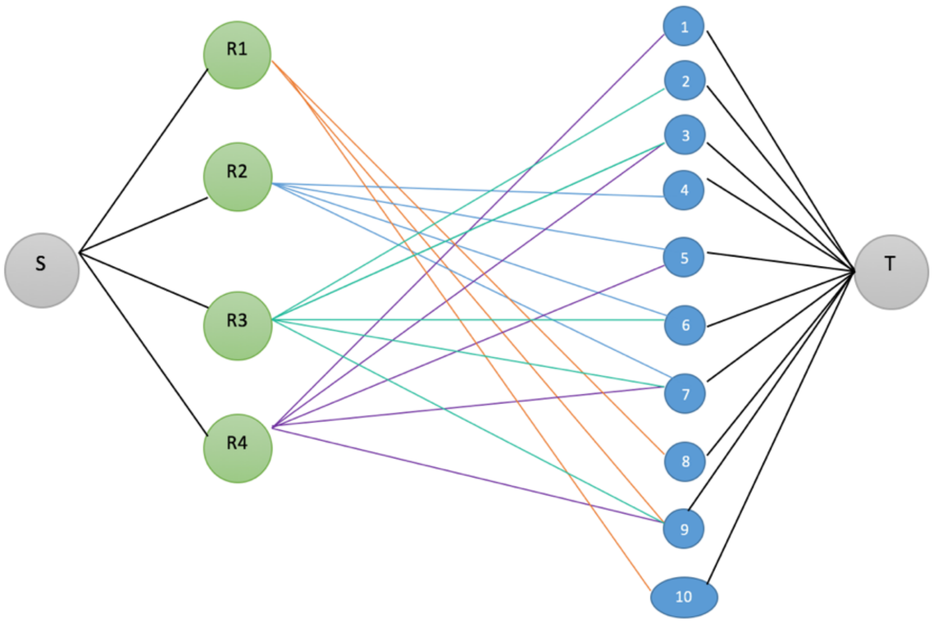
<u>Formulization</u>

Given problem is similar to max bipartite matching problem which gives an assignment of robots to objects for our scenario. Also, we know that max bipartite matching problem can be solved using Ford Fulkerson flow method but we need to convert the given bipartite graph as a flow network.

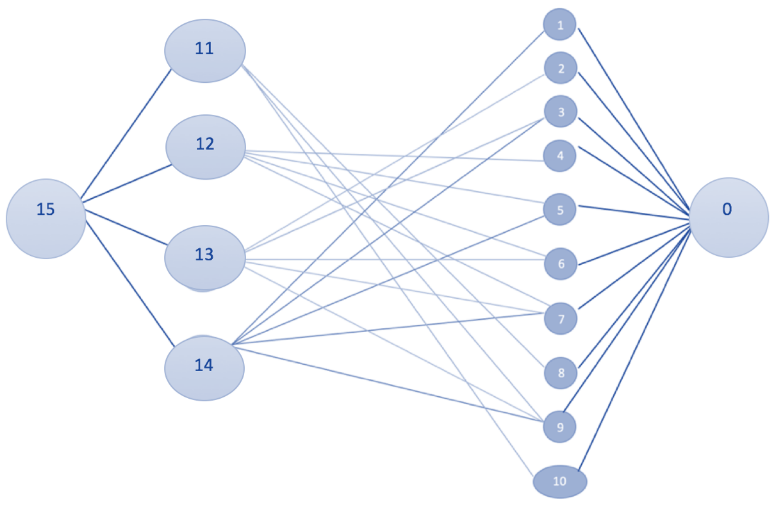
Let's start visualization with the given bipartite graph as follows



The above representation is obtained directly from the input.txt file. Then, we convert it to flow network as can be seen below.

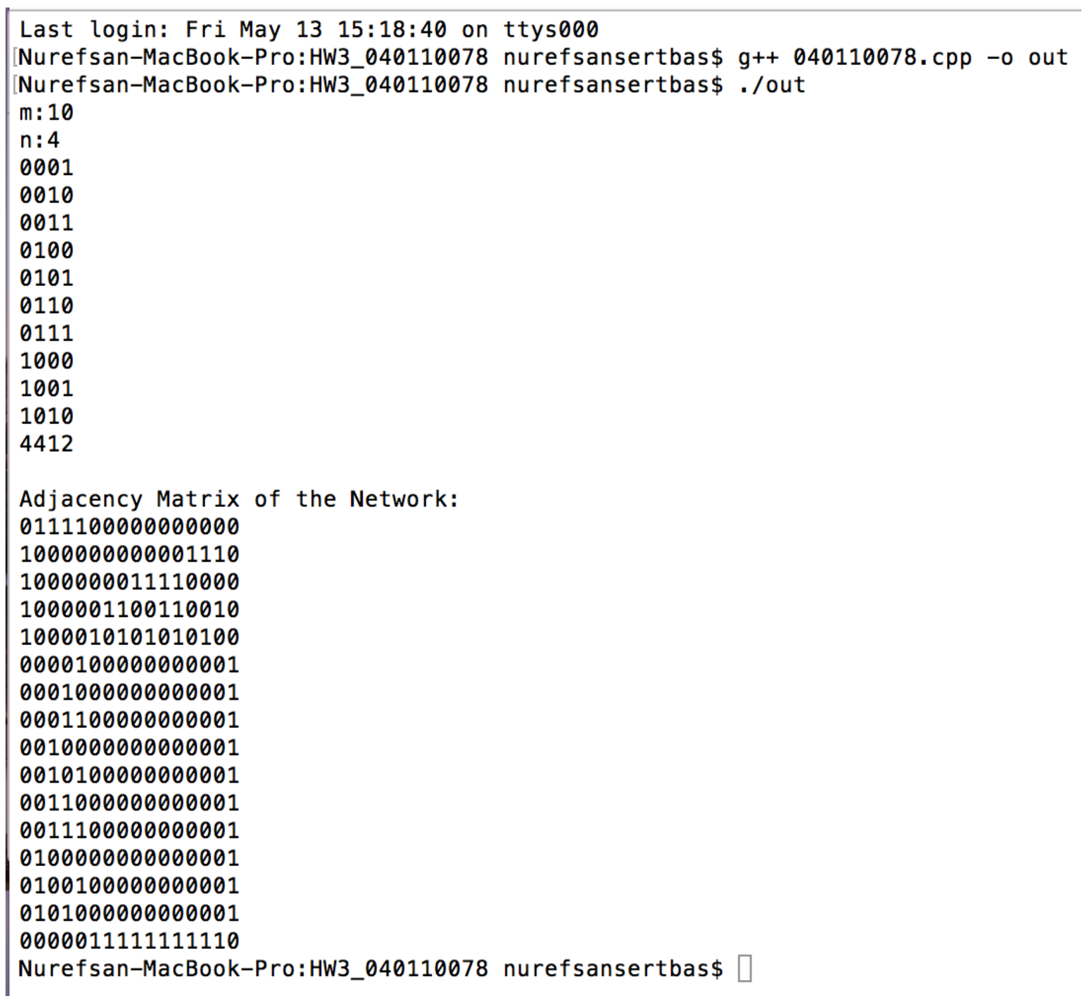


Now our flow network is ready to execute algorithm. In code, we hold the related network with an adjacency matrix after enumerating each node. Here, below screenshot can be given as an example. Also, it is possible to enumerate starting from any selected node.



Note: All undirected edges between left to right vertices are now directed edges but I forgot to show in representation.

Note that in our scenario adjacency matrix has 16 rows and 16 columns. I begin enumeration from sink node as 0 to termination node as 15. The adjacency matrix generated after execution of the code is given below.



```
Last login: Fri May 13 15:18:40 on ttys000
Nurefsan-MacBook-Pro:HW3_040110078 nurefsansertbas$ g++ 040110078.cpp -o out
Nurefsan-MacBook-Pro:HW3_040110078 nurefsansertbas$ ./out
m:10
n:4
0001
0010
0011
0100
0101
0110
0111
1000
1001
1010
4412

Adjacency Matrix of the Network:
0111100000000000
1000000000001110
1000000011110000
1000001100110010
1000010101010100
0000100000000001
0001000000000001
0001100000000001
0010000000000001
0010100000000001
0011000000000001
0011100000000001
0100000000000001
0100100000000001
0101000000000001
0000011111111110
Nurefsan-MacBook-Pro:HW3_040110078 nurefsansertbas$ 
```

Then, we assign flow capacity 1 to all edges so that it is available to apply Ford Fulkerson algorithm to find max flow (max bipartite matching in the scenario).

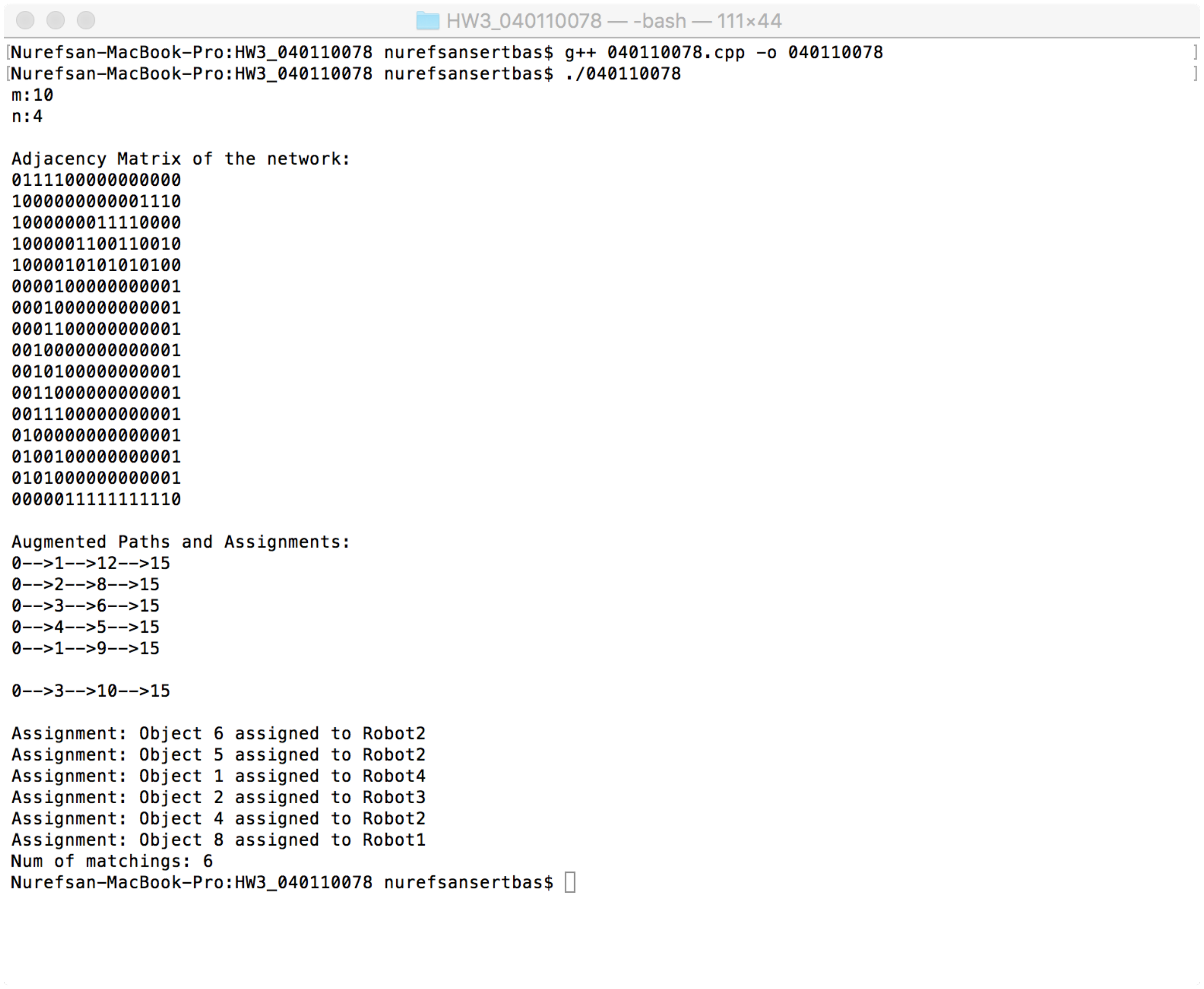
How it works

First we read the file and hold it into a matrix named as adjmat. It has m+n+2 rows and columns (m+n+2 vertices graph). If i'th node has an edge with j'th node value is 1 otherwise 0. Then we call the algorithm with additional parameters.

The algorithm continues until there is no augmenting path. Here, we use DFS traversal to find augmenting paths. DFS based function named as findpath tries all possibilities to make assignment between robots and objects. It also checks the capacities of the robots I mean max number of objects that can be carried by a robot. The function calls itself as a recursive way and returns true or false.

In each iteration (for each object), if there is no assignment we assigned object I to robot s (See the code for detail). Otherwise, if it has already assigned, we check is it possible to make assignment to other one. Here, we used matching array to avoid repetitions. According to the return value function updates the flow of the network.

Sample execution is given below.



```
Nurefsan-MacBook-Pro:HW3_040110078 nurefsansertbas$ g++ 040110078.cpp -o 040110078
Nurefsan-MacBook-Pro:HW3_040110078 nurefsansertbas$ ./040110078
m:10
n:4

Adjacency Matrix of the network:
0111100000000000
1000000000001110
1000000011110000
1000001100110010
1000010101010100
0000100000000001
0001000000000001
0001100000000001
0010000000000001
0010100000000001
0011000000000001
0011100000000001
0100000000000001
0100100000000001
0101000000000001
0000011111111110

Augmented Paths and Assignments:
0-->1-->12-->15
0-->2-->8-->15
0-->3-->6-->15
0-->4-->5-->15
0-->1-->9-->15

0-->3-->10-->15

Assignment: Object 6 assigned to Robot2
Assignment: Object 5 assigned to Robot2
Assignment: Object 1 assigned to Robot4
Assignment: Object 2 assigned to Robot3
Assignment: Object 4 assigned to Robot2
Assignment: Object 8 assigned to Robot1
Num of matchings: 6
Nurefsan-MacBook-Pro:HW3_040110078 nurefsansertbas$
```

The code works with no error but there is something missing but I could not find it. For example, for the above scenario it should do 3 more assignments as follows: Object 10 to Robot 1, Object7 to Robot3 and Object 9 to Robot4.

Complexity and Space Requirements

This implementation requires O((M+N)*(M+N)) space.

	Line in code	**Complexity**
Reading File and assigning to related data structures	Line 52-95, Nested for loop	O(m*n)
Assignment of initial values to function parameters	Line 109-193, Nested for loop	$O(N^2)$ where N=m+n
Applying the algorithm	Line 196-246, While loop and function call	O(m/n)* O(m*n!)
Printing/organizing results	While loop for matching vector operations	O(k) where k is the number of assignments