

Maximum Bipartite Matching

by fady bassel

problem background:

bipartite graph is a graph whose vertices can be split into two separated groups for example X , Y. vertices from each groups are connected with edges with the other group, in other words all edges are connected between group X and Y as in figure 1.

In maximum bipartite matching our goal is to know the maximum possible pairs that can be produced from matching elements of the two different groups. For example matching readers to book , employees to jobs and chairs to desks.

In mathematics , a theorem was discovered by Philip hall called hall's theorem discussed the concept of maximum pairing and stated a condition for maximum pairing to occur . For a given subset of a bipartite graph their number of neighbors must be bigger or equal the number of elements in the subsets, for further illustration let's take the first and third blue vertices in figure 1 as a subset (X), their only neighbor (N) is red vertex number two, since $N[x] < X$ therefore maximum number of pairs is not applied in this case.

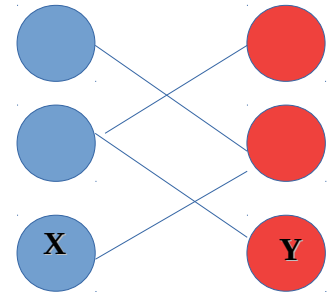


Figure 1

Techniques to solving the problem:

Hopcroft-Krap algorithm first proposed an algorithm to solving this problem using augmented paths, augmented paths are paths that starts and ends at an unassigned vertex and the edges between them alternate between assigned and unassigned. And also from the first to contribute in solving general pairing assignment problems was the Hungarian Method. Another approach is using maximum flow algorithm (Ford-Fulkerson) which is the technique used.

How the maximum flow approach is applied:

to apply Ford-Fulkerson algorithm the bipartite graph needs to be converted into a flow network with a source and destination if we convert figure 1 to a flow network it would appear as the graph in figure 2 , an edge is formed between the source S and every blue circle and an edge is formed between every red circle and the target T

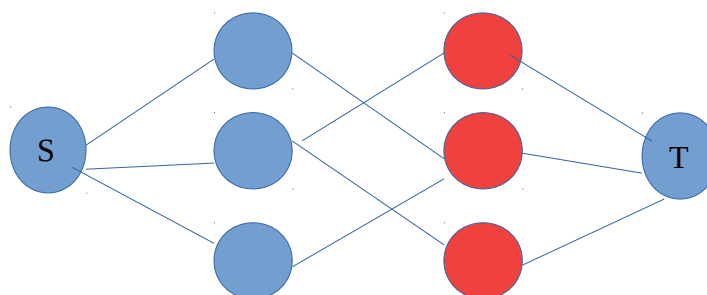
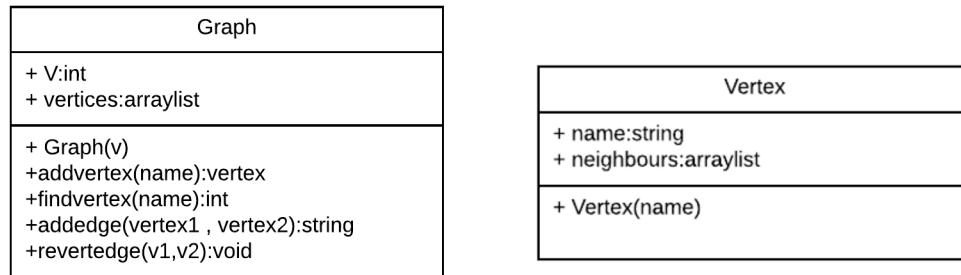


Figure 2

Program documentation:

in order to solve the problem first construction to the graph is required so two classes were designed class vertex and class graph



class vertex:

has attributes: name, neighbors which are the vertices the vertex are connected to (directly)

methods: constructor

class graph:

has attributes: V number of vertices , vertices arraylist of all the vertices in the graph

methods:

-constructor

-addvertex: adds vertex to the vertices of the graph.

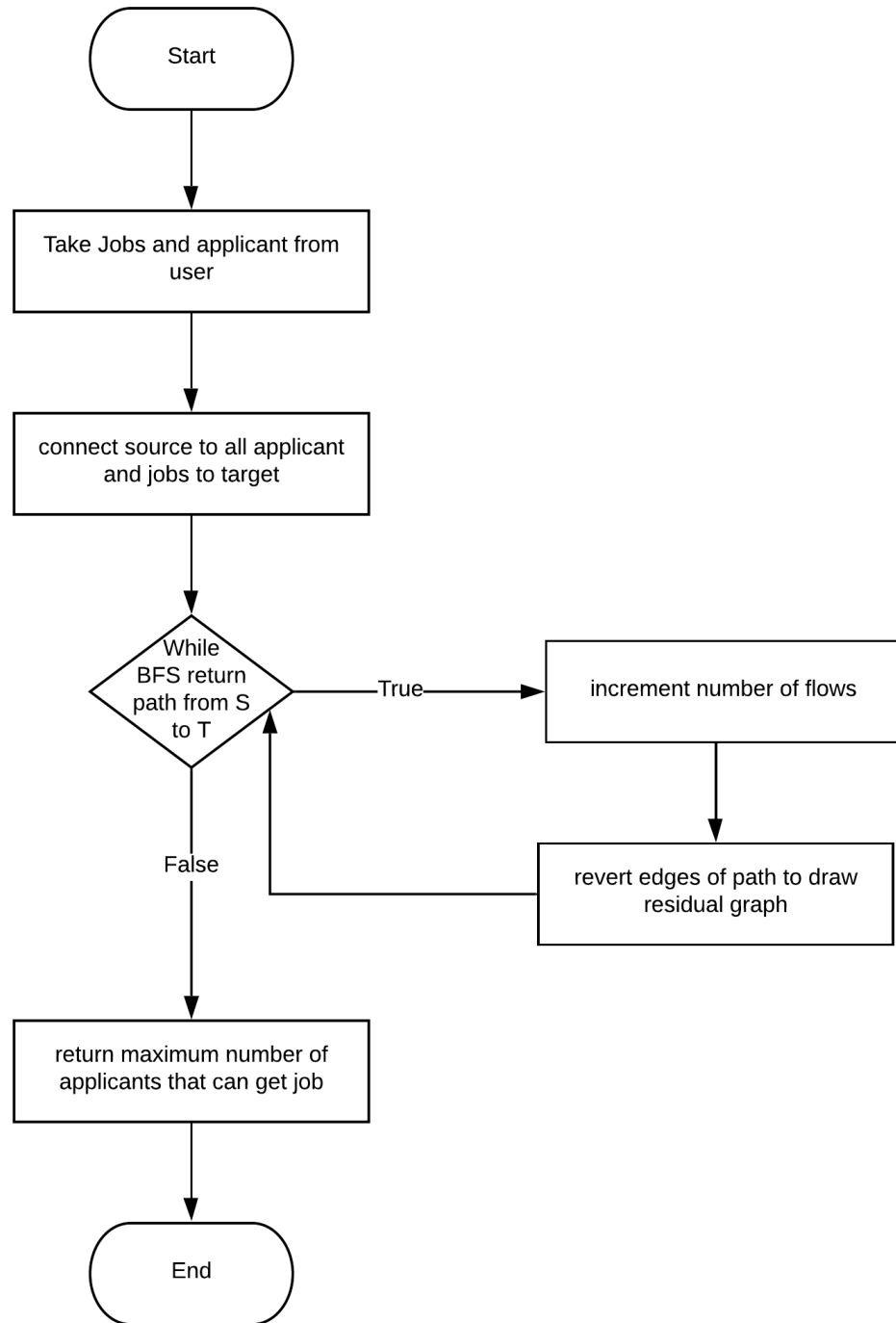
-find vertex: takes a string name and return the index of the vertex in the graph and returns -1 if not found.

-addege:takes two strings of vertices name and add vertex 2 in the neighbors of vertex 1 (directed edge) .

-revert edge: takes two two vertices index and reverse the direction of the edge.

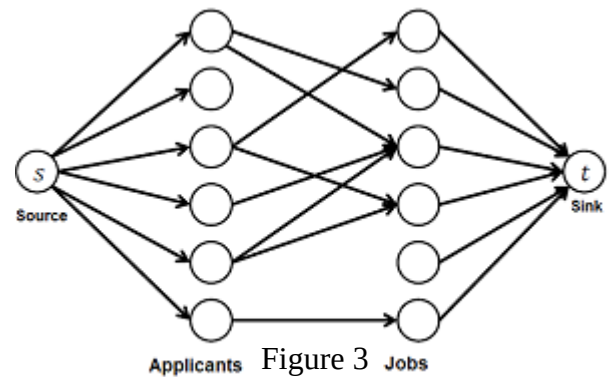
Implementing Ford-Fulkerson :

in our problem the flow always increases by one each time we find a path from the source to target so it is not necessary to add a flow value to the edges. In order to find a path a searching algorithm must be applied to find the target so a BFS is used to find the path and store it in order to be used to draw the residual graph. In our problem the residual graph can be represented by just flipping the directions of the edges of the path found using the function revertedge() in the graph. So in order to find the maximum pairs we keep calling the BFS and while there are still paths found to target we increment the flow by 1 each time and revert the edges direction in order to draw the residual graph. This is done by implementing two functions BFS and ford.



Program Run Test:

testing the program with the given problem in figure 3.



I) taking from user number of applicants and number of jobs ,with maximum of 20 for each of them

```
enter number of applicants.  
6  
enter number of jobs.  
6
```

II) connecting vertex S with an edge to all the applicants and connecting an edge between all jobs and Vertex T automatically

```
0 1  
edge added successfully  
0 2  
edge added successfully  
0 3  
edge added successfully  
0 4  
edge added successfully  
0 5  
edge added successfully  
0 6  
edge added successfully
```

0 is the index of vertex S and the numbers are the indices of the applicants.

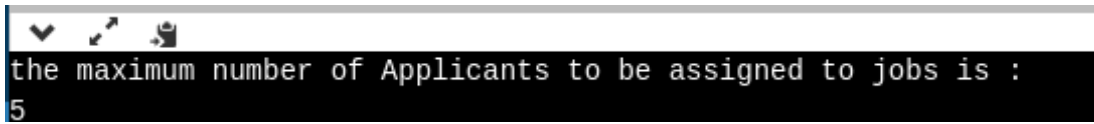
```
7 13  
edge added successfully  
8 13  
edge added successfully  
9 13  
edge added successfully  
10 13  
edge added successfully  
11 13  
edge added successfully  
12 13  
edge added successfully
```

13 is the index of vertex T and the numbers are the indices of the jobs.

III) Adding edges between applicants and jobs.

```
enter jobs for applicant no. 1 -1 to stop  
2  
1 8  
edge added successfully  
3  
1 9  
edge added successfully  
-1  
enter jobs for applicant no. 2 -1 to stop  
-1  
enter jobs for applicant no. 3 -1 to stop  
1  
3 7  
edge added successfully  
4  
3 10  
edge added successfully  
-1
```

IV) applying Ford- Fulkerson and displaying output.

A terminal window with a dark background and light gray text. The text reads: "the maximum number of Applicants to be assigned to jobs is : 5". The terminal has a standard toolbar at the top with icons for window management and a cursor icon.

```
the maximum number of Applicants to be assigned to jobs is :  
5
```

v) according to the time complexity of Ford-Fulkerson algorithm since there is a while loop finding paths and the loop continues until there are no more paths available so the big O is $O(F \cdot E)$ where F is the max flow number and E is the number of edges.