CMSC 27230: Problem Set 5

Owen Fahey

February 2023

Problem 1

(a) Figure 1 shows the residual graph. The current flow is not maximal. The highlighted region shows a flow path from s to t greater than zero.

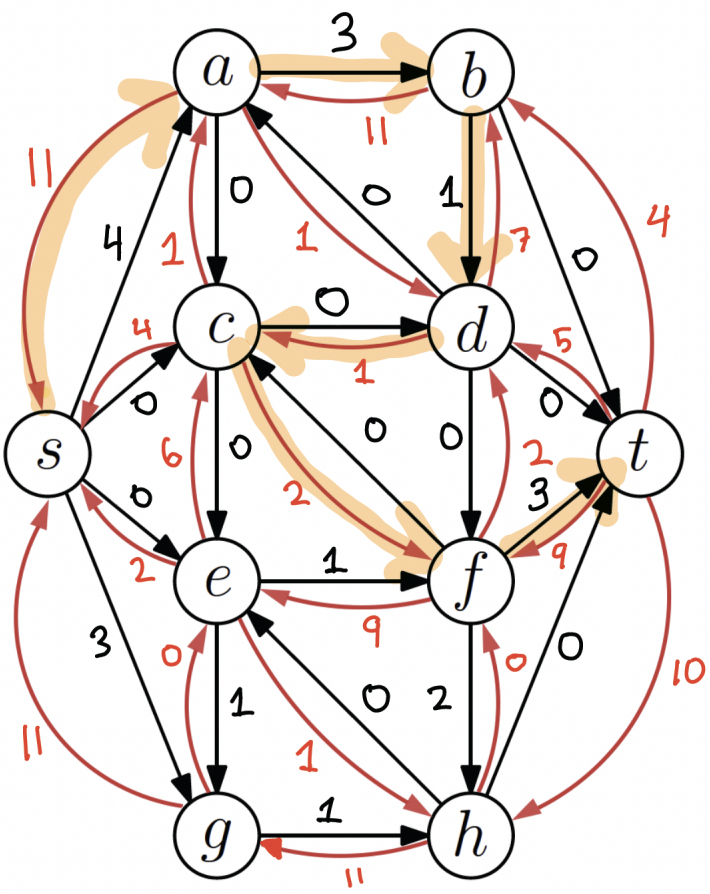


Figure 1: Residual Graph Not Maximized

We can direct a flow of 1 throw this highlighted path: s to a to b to d to c to f to t. This will give us the following residual graph:

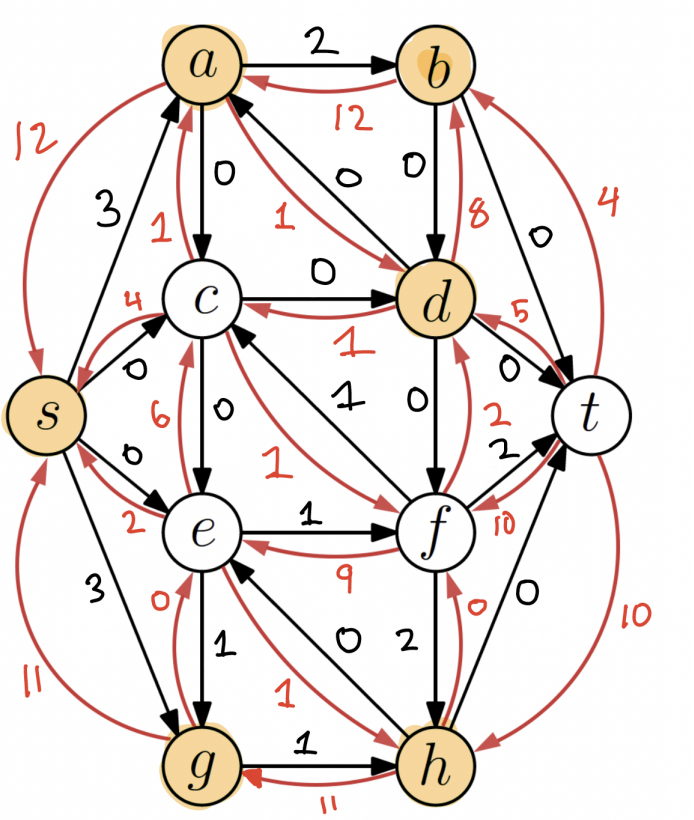


Figure 2: Residual Graph Maximized

I have highlighted all vertices able to reached from s. Since t is not reachable, the flow through this graph is maximal (and equal to 29).

Let L equal the vertices reachable from s: $L = \{s, a, b, d, g, h\}$.

Let R equal the vertices reachable from s: $R = \{c, e, f, t\}$.

Let C consist of edges from each vertex l in L to each vertex R that shares an edge with l.

Hence, $C = \{(s, c), (s, e), (a, c), (b, t), (d, f), (d, t), (h, e), (h, t)\}$

Let $c(x, y)$ be the capacity of the edge from x to y. The total capacity of the cut C in terms of the original capacity is equal to:

$$\sum_{(x,y)\in C} c(x, y)$$

$$(s, c) + (s, e) + (a, c) + (b, t) + (d, f) + (d, t) + (h, e) + (h, t)$$

$$4 + 2 + 1 + 4 + 2 + 5 + 1 + 10 = 29$$

Hence, the maximal flow does in fact equal this cut.

(b)

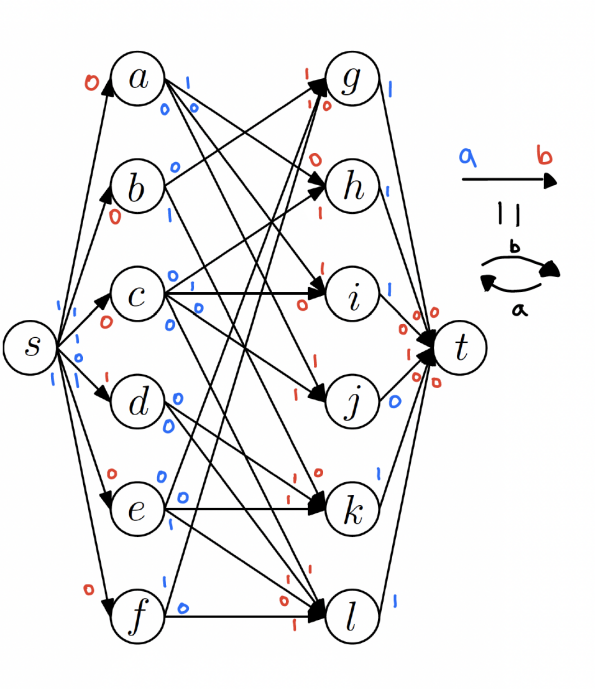


Figure 3: Residual Graph For First Graph

Please notice the trick I use where a red number means that much flow can be directed forward and a blue number means that much flow and can be directed backwards.

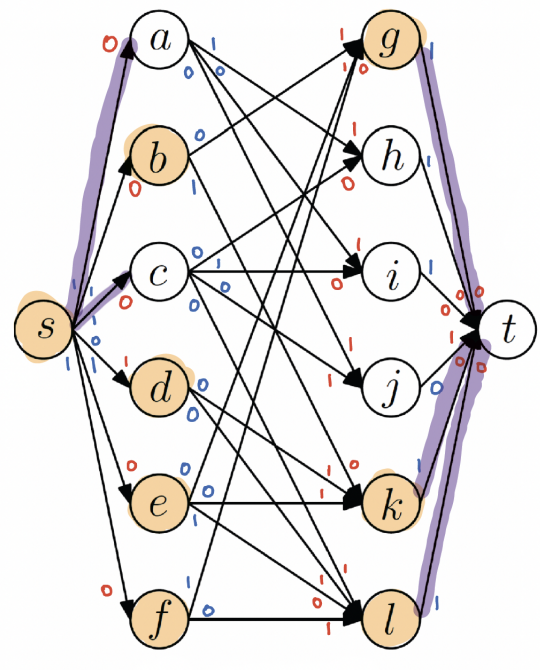


Figure 4: Reachable Vertices and Cut for First Graph

By examining all flow out of s, we can find the nodes that flow can reach. Since no flow reaches t, this flow is maximal (and equal to 5). The reachable vertices are highlighted as are the cut edges.

$L = \{s, b, d, e, f, g, k, l\}$

$R = \{a, c, h, i, j, h, t\}$

Hence, $C = \{(s, a), (s, c), (g, t), (k, t), (l, t)\}$

$$\sum_{(x,y)\in C} c(x, y) = |C| = 5$$

Hence, the maximal flow does in fact equal this cut. Moving on to the second problem:

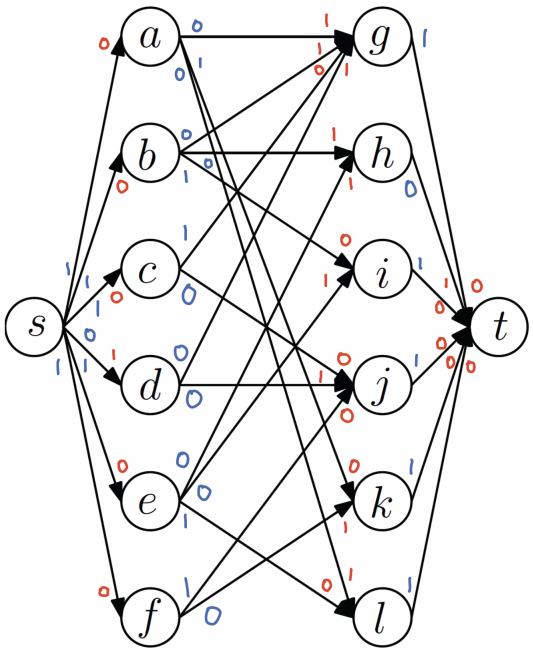


Figure 5: Not Maximal Residual Graph For The Second Graph

Notice that s to d to j to f to k to a to l to e to i to b to h to t is a path where we can send 1 flow from s to t. Doing so giving us the following residual graph where each highlighted edge has a flow of 1. Each color corresponds to a flow path. This graph has a total flow equal to the flow out of s so it is maximal (and equal to 6). In order to find the cut we are looking for, we take each edge going into t. Hence, $C = \{(g, t), (h, t), (i, t), (j, t), (k, t), (l, t)\}$ with has a capacity of 6. Hence, the maximal flow does in fact equal this cut.

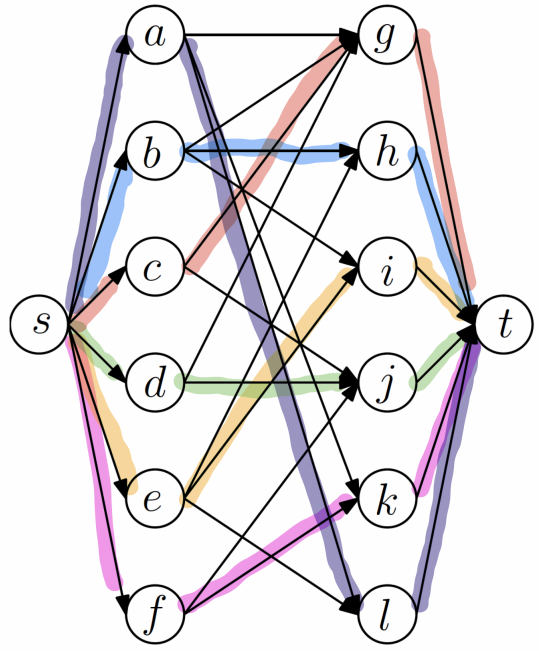


Figure 6: Maximal Residual Graph for The Second Graph

Discussion

Maximal Matching

We can represent maximal matching problems as flow problems. Say y_i are one group of vertices in a bipartite graph and x_j are the other group. We can set this up by adding in nodes s and t to the graph and creating edges of 1 from s to each y_i AND edges of 1 from each x_j to t. Connect each x_i and y_j that are connected in the bipartite graph with an edge of 1. As we showed in class, solving max flow on this graph is equivalent to solving maximal matching.

We can see the usefulness of this representation by thinking about these two max flow problems as matching problems. In the first example, the solution tells us that a maximum matching of 6 is impossible. The maximal matching is the following match of 5: $\{(a, h), (b, k), (c, i), (e, l), (f, g)\}$. In the second example, the solution gives us the matching $\{(a, l), (b, h), (c, g), (d, j), (e, i), (f, k)\}$ which would have been difficult to see without the algorithm.

Vertex Covers

Maximum matching connects to vertex covers as we showed in class when discussing Konig's Theorem. Once again, ignore s and t since we only care about the bipartite graph.

In the first example, we have that $\{a, c, g, k, l\}$ is a vertex cover. This vertex cover can be found by taking all the vertices touching edges in C. This makes a lot of sense. There are five paths each with three edges in the middle (ignoring edges involving s or t). Hence, if one of these edges is not touching any of the cut edges, then we can find a path from s to t. Hence, we cover all vertices that are part of cut edges, we guarantee that every edge touches at least one of these vertices. This also illuminates why the size of the minimum vertex covering is equal to the size of the minimum

cut is equal to the maximum flow. In the second example, we can just take all of $\{a, b, c, d, e, f\}$ or all of $\{g, h, i, j, k, l\}$.

Augmenting Paths

We look for augmenting paths when running Ford-Fulkerson in order to increase the flow from s to t. If there are no augmenting paths, then we have found the maximal flow.

Hence, in the first example, since there is no augmenting path, we can conclude we have found the maximal flow.

In the second example, we find the augmenting path s to d to j to f to k to a to l to e to i to b to$h$ to t. This augmenting path increases the total flow by 1. This is a great augmenting path as it uses the backwards edges from j to f.

Problem 2

Algorithm:

Setup: The inputs to our subproblems will be intervals of x values. Let us say $[g : h]$ represents the input $\{x_g, x_{g+1}, ..., x_h\}$. Let $c[g : h]$ give the minimum total number of queries to the nodes of any tree that could be constructed from $[g : h]$. Let s be difference between h and g for interval $[g : h]$.

Subproblems: For each $s \in [1, n-1]$, solve each subproblem $c[j, j+s]$ for each $j \in [1, n-s]$ in ascending order of s. As an observation, it is trivially the case that $c[g : g] = a_g$ as the there is only one possible tree with x_g at d_0. Similarly, $\sum_{i=g}^{g} a_i = a_g$. (This will matter later).

Recurrence Relationship: Let k be an integer where $g < k < h$,

$$c[g : h] = min \begin{cases} a_g + c[g+1 : h] + \sum_{i=g+1}^{h} a_i \\ c[g : k-1] + (\sum_{i=g}^{k-1} a_i) + a_k + c[k+1 : h] + (\sum_{i=k+1}^{h} a_i) \\ c[g : h-1] + (\sum_{i=g}^{h-1} a_1) + a_h \end{cases}$$

Solution: For each subproblem $c[g : h]$, record the tree that produces $c[g : h]$ as well as $\sum_{i=g}^{h} a_i$. To solve the problem, return the tree that produces $c[1, n]$.

Explanation of Correctness:

This recurrence relationship finds the minimum number of total queries by checking each possible root node. The first case accounts for when no left sub-tree exists and the third case accounts for when no right sub-tree exists.

If we choose x_k to be the root node for some interval, $[g : h]$. Then, we know all values $[g : k-1]$ will be on the left side of the tree and all values $[k+1 : h]$ will be on the right side of the tree. This follows from the definition of a binary tree and the fact $x_1 < x_2 < ... < x_n$.

We can use information about previous subproblems to solve the current subproblem. For tree with a given a root node x_k, the total number of queries is equal to the number of searches for the root node (a_k) plus the total number of queries involved in searches for nodes in the two sub-tress immediately to the left and to the right of the root node. The solution to previous subproblems allow us to easily find the minimize number of total queries involved in searches to these sub-trees.

Suppose we are considering a tree constructed from $[g : k-1]$. The total number of queries to this tree is equal to $\sum_{i=g}^{k-1} a_i(d_i + 1)$ which is the value that is minimized by $c[g : k-1]$. If this tree were to becomes a sub-tree by moving down a level, the depth of each node would increase by 1. Hence, the number of total queries involved in searching for nodes in this this sub-tree increases by the sum of the number of searches for each node $\sum_{i=g}^{k-1} a_i$ (it is worth pointing out that this value doesn't depend on how the tree is constructed). This is because for each search to each node in this sub-tree, we must first query the root node. Hence, the minimum number of queries involved in searching for nodes of this sub-tree is $c[g : k-1] + (\sum_{i=g}^{k-1} a_i)$.

Hence, if we know $c[g : h]$ and $(\sum_{i=g}^{h} a_i)$ for all possible values of g and h such that $1 \leq g \leq h \leq n$, we can solve $c[1 : n]$. Every interval of size z will depend on subproblems with intervals smaller than z. Hence, it is important we solve the subproblems in ascending order of interval size.

Runtime Analysis:

For any interval of size z, there will be z possible root nodes to check. The time it takes get the total number of queries for some root node is constant since at most we are summing 6 stored values. We check all intervals of size 2 to n. There will be $\lfloor \frac{n}{2} \rfloor$ intervals with 2 elements, $\lfloor \frac{n}{3} \rfloor$ intervals with 3 elements and so on.

Putting this all together, we have that we have to check the follow number of root nodes:

$$\sum_{i=2}^{n} \lfloor \frac{i}{2} \rfloor \cdot i \rightarrow O(n^3)$$

Since, each root node takes constant time to check, the whole algorithm takes $O(n^3)$.

Problem 3

Answer: This conjecture is false.

For clarity, lets say NB refers to Ford-Fulkerson with no backwards edges.

Countering all $c > 1$:

Suppose $c > 1$. Assume there is also a possible flow greater than cF through a graph. Let us say the value of this flow is kF where $k > c$. This contradicts the definition of F since F is supposed to be the maximum possible flow through the graph but kF is a possible flow through the graph with $kF > cF > F$. Hence, our assumption must be false and there is no possible flow greater than cF through any graph.

Countering all c where $0 < c \leq 1$:

Suppose we are given $c = \frac{1}{a}$ where $a \geq 1$. We can always construct a graph such that a flow returned by NB is less than or equal to cF, thus contradicting our friend's conjecture. For the sake of simplicity, let us say that $b = \lfloor a \rfloor$. If we can show that it is possible for a flow less than $\frac{1}{b}F$ to be returned then it is possible for a flow less than or equal to $\frac{1}{a}F$ to be returned by since $b \leq a \rightarrow \frac{1}{b}F \leq \frac{1}{a}F = cF$.

In order to construct such a graph, create the following nodes and edges:

1. Edges of 1 between node s and nodes $\{v_1, v_2, ...v_{b+1}\}$.
2. Edges of 1 between each node v_k to u_k for every $k \in [1, b+1]$
3. Edges of 1 between each node u_k to v_{k+1} for every $k \in [1, b]$

The max flow through this graph is achieved by

$$s \text{ to } v_k \text{ to } u_k \text{ to } t \text{ for every } k \in [1, b+1].$$

Since, there are b_1 of these paths and each routes a flow of 1 from s to t, the max flow will be equal to $b + 1$. We can be confident that this is the max flow since capacity out of s is equal to $b + 1$.

It is always possible to find a path with flow 1 through this graph such that there are no other possible paths with available capacity if we ignore backwards flows. Such a path could always be returned by NB for the graph we constructed and is as follows:

$$s \text{ to } v_1 \text{ to } u_1 \text{ to } v_2 \text{ to ... to } v_k \text{ to } u_k \text{ to } v_{k+1} \text{ to } u_{k+1} \text{ to ... } v_{b+1} \text{ to } u_{b+1} \text{ to } t.$$

The flow of this path will be equal to 1. Since $1 < \frac{1}{b} \cdot F = 1 + \frac{1}{b}$.

Hence, given $c > 0$, it is always possible to construct a graph where the flow returned by Ford-Fulkerson with no backwards edges is less than cF and therefore, my friend's conjecture is false.

An example:

Consider the following example for $c = \frac{1}{2}$ where all the edges are equal to 1.

Figure 1:

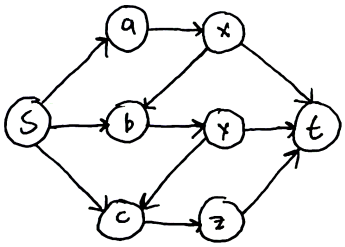


Figure 2:

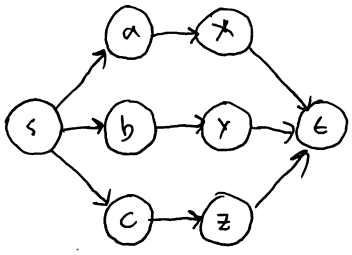
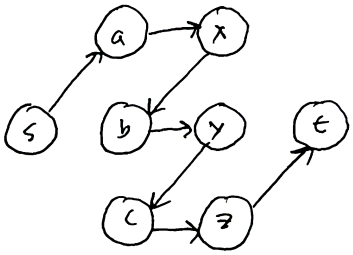


Figure 3:



Let the G be graph in figure 1. The maximum flow through G is equal to 3 and is shown by figure 2. However, figure 3 shows a possible flow that could be returned by NB with a value of 1. Hence, we have a possible flow returned by NB of 1 where $1 < cF = \frac{1}{2} \cdot 3 = \frac{3}{2}$

Problem 4

Setup: This problem can be represented as a max flow problem. To set it up, create a directed graph with the following edges:

An edge of 1 from s to every man y_k in $\{y_1, ...y_n\}$

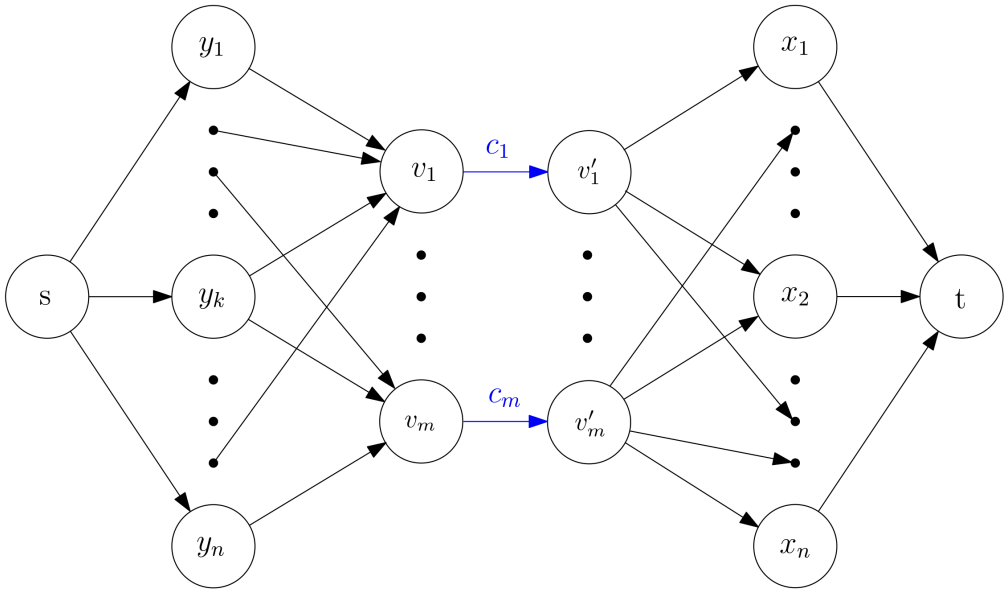
An edge of 1 from every man y_k in $\{y_1, ...y_n\}$ to every venue in each man's list, Y_k

An edge of c_k from every venue v_k in $\{v_1, ...v_m\}$ to a clone venue v'_k in $\{v'_1, ...v'_m\}$

An edge of 1 from every clone venue $\{v'_1, ...v'_m\}$ to every woman $\{x_1, ...x_n\}$

An edge of 1 from every woman x_k in $\{x_1, ...x_n\}$ to t

The graph should looks something like as follows:



In order to solve this problem, run Ford–Fulkerson on this resulting graph. The flow graph given by Ford-Fulkerson will tell us how to how match men and women to venues. Specifically, any resulting edges between y_i and v_k with flow of 1 will tell us to assign y_i to v_k and any resulting edges between x_j and v'_k with flow of 1 will tell us to y_i to v'_k.

A discussion of why this works:

This process finds the maximum number of matchings between men, women, and venues such that venues do not exceed capacity, people only get matched with one their preferred venues and there is gender parity at each venue.

Recall that the flow through a node can not exceed the sum of the capacity of all edges into the node nor the sum of the capacity of all edges out from the node. Hence, in the flow graph resulting from running Ford-Fulkerson, the following facts will be true.

1. Venues won't exceed capacity.

Assume that v_k had a flow greater than c_k in. Since, the maximum capacity out of v_k is at most c_k, this would violate the condition that the flow in must be the same as the flow out. Hence, this assumption must be false. This same reasoning allows us to conclude that the maximum number of edges going out of v'_k is c_k. It is also quite easy to see that the flow into v_k must be the same as the flow out of v'_k.

2. People will only get assigned one their preferred venues

No man or woman has an edge between themselves and any venue they do not prefer. Hence, it is only possible to match people to a venue in their preference list.

A match only occurs between an y_i and a v_k if there is a flow of 1 out of y_i into v_k. Likewise, a match only occurs between x_i and a v_k if there is a flow of 1 out of v'_k into x_i. Since each y_i has a capacity of 1 in, they can have at most a flow of 1 out. Likewise, since each x_i has a capacity of 1 out, they can have at most a flow of 1 in. This allows us to conclude that each x_i and each y_i get matched with at most 1 of their preferred venues.

3. The number of men and women assigned to each venue will be the same.

As we stated earlier, the flow into v_k must be the same as the flow out v'_k. The number of men matched with v_k will be equal to the flow into v_k. Likewise, the number of women matched with v_k will be equal to the flow out of v'_k. Hence, the number of men and women matched with v_k will be the same.

As a final note, it is possible that the max flow does not include all people. In such a cases, it is impossible for all 3 of the aforementioned conditions. Hence, a solution exists to the problem iff the max flow is equal to n.

Proof this takes polynomial time:

The time complexity of this algorithm can be broken down into:

1. Constructing the graph
2. Running Ford-Fulkerson

Constructing a graph with edges and vertices will take O($|V| + |E|$). The number of vertices is as follows:

$$|V| = |\{s\}| + |\{y_1, ...y_n\}| + |\{v_1, ...v_m\}| + |\{v'_1, ...v'_m\}| + |\{x_1, ...x_n\}| + |\{t\}|$$

$$= 1 + n + m + m + n + 1$$

$$= 2 + 2n + 2m$$

The number of edges involves $|Y_k|$ and $|X_k|$ for each $k \in [0...n]$. We know that $max|X_k| = max|Y_k| = m$ since the most venues that could be in someone's list is the maximum number of venues, m. Hence, the number of edges is as follows:

$$|E| = |\{y_1, ...y_n\}| + \sum_{k=1}^{n} |Y_k| + |\{v_1, ...v_m\}| + \sum_{k=1}^{n} |X_k| + |\{x_1, ...x_n\}|$$

$$= n + n \cdot m + m + n \cdot m + n$$

$$= 2n + m + 2nm$$

Putting this altogether, the time to construct the graph is $4n + 3m + 2nm + 1 \rightarrow O(nm)$.

The time complexity of running Ford-Fulkerson depends on the max flow, F. The maximum possible max flow is equal to n and occurs when every couple is assigned to a venue. Hence, the run time of Ford-Fulkerson is $O(F \cdot |E(G)| = O(n^2 m)$.

Together, we have the time complexity is $O(n^2 m)$.