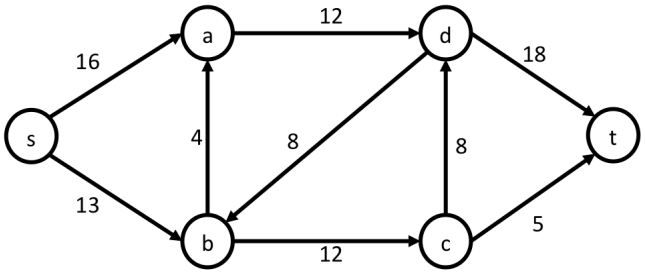
- The instructions are the same as in Homework-0, 1, 2.

There are 5 questions for a total of 100 points.

---

1. (20 points) Consider the network shown in the figure. Consider running the Ford-Fulkerson algorithm on this network.

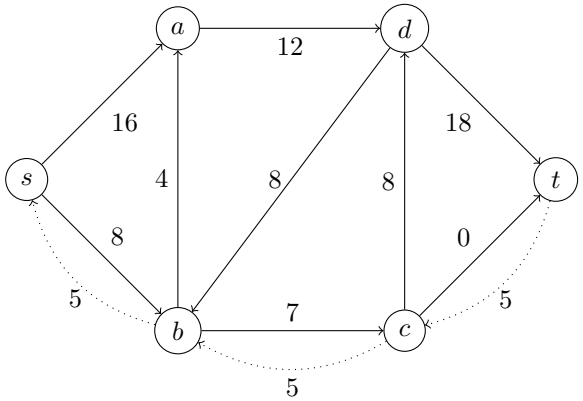


(a) We start with a zero s-t flow f. The algorithm then finds an augmenting path in G_f. Suppose the augmenting path is $s \to b \to c \to t$. Give the flow f' after augmenting flow along this path.

Solution:
A flow is a function from edges to non-negative real numbers, so it suffices to give the flow value for each edge.
$f'((s,b)) = 5$, $f'((s,a)) = 0$, $f'((b,a)) = 0$, $f'((a,d)) = 0$, $f'((d,b)) = 0$, $f'((b,c)) = 5$, $f'((c,d)) = 0$, $f'((c,t)) = 5$, $f'((d,t)) = 0$

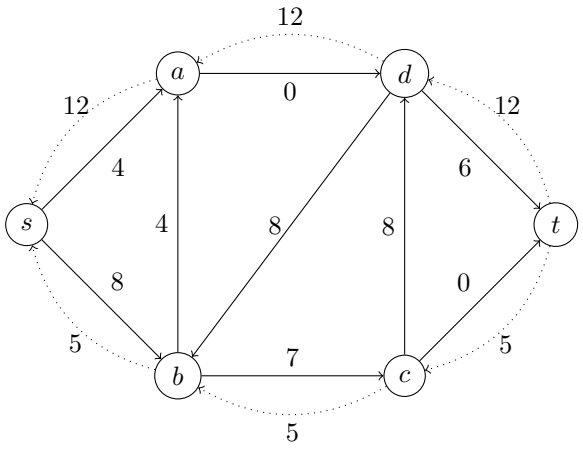
(b) Show the graph $G_{f'}$. That is, the residual graph with respect to s-t flow f'.

Solution:
In all the following parts, back edges that are not drawn should be treated to have zero weight. This has been done for the sake of a clear diagram (and as clarified on Piazza, it is valid too).



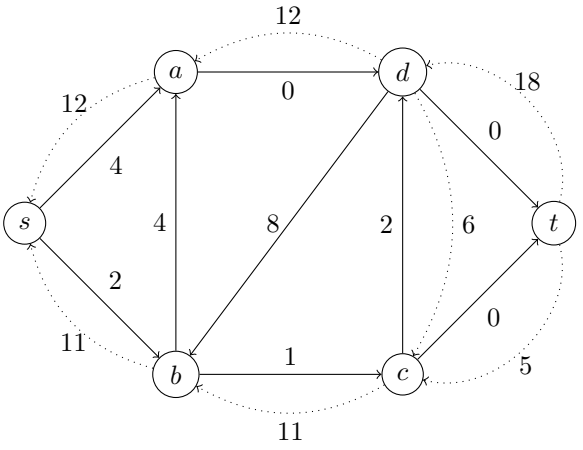
(c) The algorithm then sets f as f' and G_f as $G_{f'}$ and repeats. Suppose the augmenting path chosen in the next iteration of the while loop is $s \to a \to d \to t$. Give f' after augmenting flow along this path and show $G_{f'}$.

Solution: $f'((s,b)) = 5$, $f'((s,a)) = 12$, $f'((b,a)) = 0$, $f'((a,d)) = 12$, $f'((d,b)) = 0$, $f'((b,c)) = 5$, $f'((c,d)) = 0$, $f'((c,t)) = 5$, $f'((d,t)) = 12$



(d) Let f be the flow when the algorithm terminates. Give the flow f and draw the residual graph G_f.

Solution:
The residual graph has two augmenting paths: $s \to b \to d \to t$ and $s \to b \to c \to d \to t$, and after taking either of these two, the algorithm terminates. We suppose that the second augmenting path is chosen.
$f((s,b)) = 11$, $f((s,a)) = 12$, $f((b,a)) = 0$, $f((a,d)) = 12$, $f((d,b)) = 0$, $f((b,c)) = 11$, $f((c,d)) = 6$, $f'((c,t)) = 5$, $f((d,t)) = 18$



(e) Give the value of the flow f when the algorithm terminates. Let A^* be the vertices reachable (using edges of positive weight) from s in G_f and let B^* be the remaining vertices. Give A^* and B^*. Give the capacity of the cut (A^*, B^*).

Solution: $v(f) = 23$
$A^* = \{s, a, b, c, d\}$
$B^* = \{t\}$
$C(A^*, B^*) = \sum_{e \text{ out of } A^*} c(e) = 18 + 5 = 23$

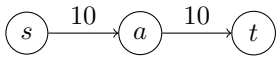
2. Answer the following:

(a) (1 point) State true or false: For every s-t network graph G, there is a unique s-t cut with minimum capacity.

Solution: False

(b) (4 points) Give reason for your answer to part (a).

Solution:



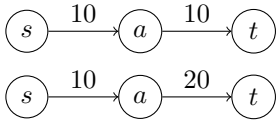
In the above figure if we consider sets
$A = \{s\}$
$B = \{a, t\}$
$A' = \{s, a\}$
$B' = \{t\}$
There are two cuts possible for the given flow: (A, B) and (A', B'). We can see that both have capacity 10. So the minimum capacity for any cut in the network graph is 10, and there are two possible cuts with that capacity. Therefore the given statement is false.

(c) (1 point) State true or false: For every s-t network graph G and any edge e in the graph G, increasing the capacity of e increases the value of maximum flow.

Solution: False

(d) (4 points) Give reason for your answer to part (c).

Solution: To prove that the statement is false, it is sufficient to provide a counterexample.
Consider the two graphs below. In the second one, the capacity of the edge $a \to t$ is changed from 10 to 20.
For the first graph the maximum flow will be 10.
For the second graph the maximum flow will also be 10.
So despite increasing the capacity of an edge, the maximum flow remains the same, and hence this is a valid counterexample.



(e) (1 point) State true or false: For every s-t network graph G for which an s-t flow with non-zero value exists, there exists an edge e in the graph such that decreasing the capacity of e decreases the value of maximum flow.

Solution: True

(f) (4 points) Give reason for your answer to part (e).

Solution: Let A^* be the set of vertices reachable from s in the residual graph G_f corresponding to the maximum flow, and let B^* be the set of the remaining vertices. Then (A^*, B^*) is an $s - t$ cut.

Note that there must be at least one edge from A^* to B^* since otherwise, there would be no edge from A^* to B^* implying that the capacity of this cut is 0, which means that the maximum flow is 0, which is false by the problem condition.

Note that the capacity of this cut equals the minimum flow as done in class, say c.

Consider any edge $e = (x, y)$ from A^* to B^*. Reducing the capacity of this edge reduces the capacity of this cut to some $v < c$. The maximum flow value of the new graph equals the minimum cut capacity of the new graph by max-flow min-cut theorem, which can be at most v since we have exhibited a cut of capacity v. Thus the maximum flow value of the new graph is at most $v < c$, so upon reducing the capacity of this edge, the value of the maximum flow in the graph decreases.

3. Suppose you are given a bipartite graph (L, R, E), where L denotes the vertices on the left, R denotes the vertices on the right and E denote the set of edges. Furthermore it is given that degree of every vertex is exactly d (you may assume that $d > 0$). We will construct a flow network G using this bipartite graph in the following manner: G has $|L| + |R| + 2$ vertices. There is a vertex corresponding to every vertex in L and R. There is also a source vertex s and a sink vertex t. There are directed edges with weight 1 from s to all vertices in L and directed edges of weight 1 from all vertices in R to t. For each edge $(u, v) \in E$, there is a directed edge from u to v with weight 1 in G.

(*The figure below shows an example of a bipartite graph and the construction of the network.*)

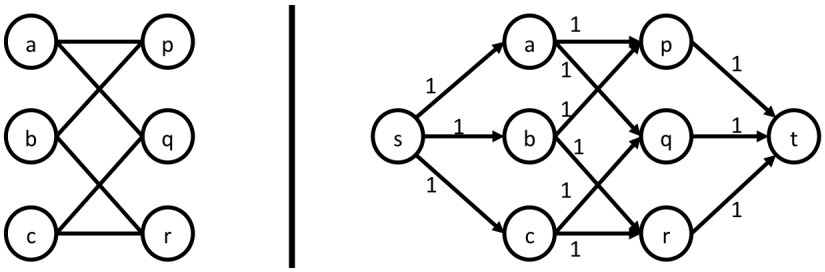


Figure 1: An example bipartite graph (with $d = 2$) and network construction.

(a) (5 points) Argue that for any such bipartite graph where the degree of every vertex is equal to d, $|L|$ is equal to $|R|$.

Solution: We count the number of edges in two ways.

Each edge of the graph has one endpoint in L and the other endpoint in R.

Hence, the total number of edges equals the total number of edges with exactly one endpoint in L, and that is precisely the sum of degrees of all vertices in the L (since there is no edge between vertices in L), which is $d|L|$.

More formally, we have:

$$\begin{aligned}
\sum_{\{u,v\}\in E} 1 &= \sum_{(u,v)\in E \wedge u\in L \wedge v\in R} 1 \\
&= \sum_{u\in L} \sum_{(u,v)\in E \wedge v\in R} 1 \\
&= \sum_{u\in L} deg(u) \\
&= \sum_{u\in L} d \\
&= d|L|
\end{aligned}$$

By a completely analogous argument, we have

$$\begin{aligned}\sum_{\{u,v\}\in E} 1 &= \sum_{(u,v)\in E\wedge u\in L\wedge v\in R} 1 \\ &= \sum_{v\in R}\sum_{(u,v)\in E\wedge u\in L} 1 \\ &= \sum_{v\in R} deg(v) \\ &= \sum_{v\in R} d \\ &= d|R|\end{aligned}$$

Hence, we have $d|L| = d|R|$, so since $d > 0$, we have $|L| = |R|$.

(b) (10 points) Argue that for any given bipartite graph where the degree of every vertex is the same non-zero value d, there is an integer s-t flow (i,e., flow along any edge is an integer) in the corresponding network with value $|L|$.

Solution: Consider any $s-t$ cut (A, B). We claim that the capacity of this cut is at least $|L|$, and then exhibit a cut of capacity precisely $|L|$, to establish that the minimum capacity of a cut in the corresponding network is $|L|$, which by the max-flow-min-cut theorem equals the maximum value of a flow in the corresponding network.

Let $\ell = |L| = |R|$.

Note that since (A, B) is an $s-t$ cut, we have $s \in A, t \in B$.

Suppose A has n vertices from L and m vertices from R. Then B has $\ell - n$ vertices from L and $\ell - m$ vertices from R.

Define X_{IJ} to be the number of edges going from $L \cap I$ to $R \cap J$.

Then we have $C(A, B)$ to be the number of edges from s to $L \cap B$ + the number of edges from $R \cap A$ to t + the number of edges going from $L \cap A$ to $R \cap B$.

Hence we have

$$C(A, B) = \ell - n + m + X_{AB}$$

If $n \leq m$, then we are done, since then

$$C(A, B) = \ell + (m - n) + X_{AB} \geq \ell + 0 + 0 = \ell$$

Now suppose $n > m$. We claim that $X_{AB} \geq n - m$.

The total number of edges going from $A \cap L$ to R is $X_{AA} + X_{AB}$, and it is equal to the sum of degrees of vertices in $A \cap L$, which is dn.

Similarly, the total number of edges going from L to $A \cap R$ is $X_{AA} + X_{BA}$, and it is equal to the sum of degrees of vertices in $A \cap R$, which is dm. Hence we have

$$\begin{aligned}X_{AB} &= X_{BA} + (X_{AA} + X_{AB}) - (X_{AA} + X_{BA}) \\ &= X_{BA} + dn - dm \\ &= X_{BA} + d(n - m) \\ &\geq 0 + n - m \\ &= n - m\end{aligned}$$

as needed, where the inequality comes in since $d \geq 1$ and $n > m$. Hence in this case too, we have $C(A, B) \geq \ell$.

Thus we have shown that $C(A, B) \geq \ell$ for any $s - t$ cut.

Now consider the cut where $A = \{s\}$, and B is the set of the remaining vertices. Note that here the capacity of the cut is the sum of 1 over all edges outgoing from s, which is ℓ.

Hence for a particular cut, we have shown that $C(A, B) = \ell$, which when combined with the previous bound shows that the minimum capacity of a cut in the network is ℓ, and by the max-flow-min-cut theorem, this is equal to the maximum flow of the network.

Since the maximum flow of the network has value $|L|$, there exists a flow with value $|L|$, and since this flow is a maximum flow on a network with integer capacities, it can be computed using the Ford-Fulkerson algorithm, which always guarantees that the flow it returns is an integer flow. Hence there exists an integer $s - t$ flow in the corresponding network with value $|L|$, as desired.

(c) (5 points) A matching in a bipartite graph $G = (L, R, E)$ is a subset of edges $S \subseteq E$ such that for every vertex $v \in L \cup R$, v is present as an endpoint of at most one edge in S. A maximum matching is a matching of maximum cardinality. Show that the size of the maximum matching in any bipartite graph $G = (L, R, E)$ is the same as the maximum flow value in the corresponding network graph defined as above.

Solution: We shall

1. Construct a valid flow f of size $|f|$ from any valid matching of size $|f|$.
2. Construct a valid matching of size $|f|$ from any valid flow f.

Construction of flow from a maximum matching Suppose S is the set of all edges in the matching. By the bipartite-ness of the graph, each edge has exactly one endpoint in L and the other one in R. Hence, for each edge (u, v), where $u \in L, v \in R$, set $f((s, u)) = f((u, v)) = f((v, t)) = 1$ in the network. In the end, set f of all the remaining edges in the network to 0.

We need to show that this is a valid flow, and that the value of this flow is $|S|$.

For showing that it is a valid flow, we need to show flow conservation and the satisfaction of capacity constraints.

1. Flow conservation: Consider any vertex $u \in L$. If $f((s, u)) = 1$, then there must have been an edge in S with u as an endpoint (and exactly one such edge by the definition of a matching). Hence, $\sum_{e \text{ into } u} f(e) = 1$ and $\sum_{e \text{ out of } u} f(e) = 1$. For a vertex u which has $f((s, u)) = 0$, it is not an endpoint of any edge in S, so $\sum_{e \text{ into } u} f(e) = 0$ and $\sum_{e \text{ out of } u} f(e) = 0$. Now consider any vertex $v \in R$. If $f((v, t)) = 1$, then there must have been an edge in S with v as an endpoint (and exactly one such edge by the definition of a matching). Hence, $\sum_{e \text{ into } v} f(e) = 1$ and $\sum_{e \text{ out of } v} f(e) = 1$. For a vertex v which has $f((v, t)) = 0$, it is not an endpoint of any edge in S, so $\sum_{e \text{ into } v} f(e) = 0$ and $\sum_{e \text{ out of } v} f(e) = 0$. Hence for all internal vertices, flow conservation holds.
2. Capacity constraints: Since we assign either 0 or 1 to any edge, and the capacity of an edge is 1, the capacity constraints hold trivially.

Now to show that the value of this flow is $|S|$, note that any edge in S has exactly one endpoint in L, and no two edges in S share an endpoint. Hence, the number of vertices with $f((s, u)) = 1$ is precisely the number of edges in S, which is $|S|$. However, the value of the flow is the sum of

$f-$values assigned to edges emanating from s, which is equal to the number of edges emanating from s which we assigned 1 to, and that equals $|S|$.

Hence, this is a valid flow, and the capacity of the flow is $|S|$ and we are done.

Construction of matching from a maximum flow Note that in a flow network with integer capacities, the maximum value of a flow is an integer, and there exists a flow with the maximum value, such that all $f-$values of edges are integers (which can be found using the Ford-Fulkerson algorithm).

Consider such a flow f, and the set of edges between internal nodes having $f-$values positive (and by combining the the discussion above and capacity constraints, the $f-$values must be 1).

More formally, let S be the set of edges (u, v) such that $f((u, v)) > 0$, and $\{s, t\} \cap \{u, v\} = \emptyset$. Then $f((u, v))$ is an integer and upper bounded by 1, so it is 1 for all such vertices. Note that we need to make each edge undirected before including it in S.

We claim that S is a matching with size equal to the value of the flow.

1. S is a matching: Note that S has edges incident only on vertices which are in G. Hence, S is a subset of E by the construction of the network. Now suppose there exists a vertex $v \in L \cup R$ which is incident to at least 2 edges in S. We make two cases:

   (a) $v \in L$: In that case, we have at least two edges emanating from v which have $f(e) = 1$. Hence by flow conservation, the sum of weights of all edges into v must be at least 2, which is a contradiction, since the only edge coming into v is (s, v), and it has capacity 1.

   (b) $v \in R$: In that case, we have at least two edges coming into v which have $f(e) = 1$. Hence by flow conservation, the sum of weights of all edges emanating from v must be at least 2, which is a contradiction, since the only edge emanating from v is (v, t), and it has capacity 1.

   Hence we have shown that S is a matching.

2. $|S| = v(f)$: Firstly note that since G is bipartite, all edges in S have exactly one endpoint each in L and R. Consider the set T of all vertices in L that are incident to an edge in S. Then, since each edge in S has an $f-$value of 1, and no two edges share a common endpoint (which implies at most 1 edge from S is incident on a given vertex in L), $\sum_{e \text{ out of } u} f(e) = 1$ for each u in T. By flow conservation, for each vertex v in T, we have $f((s, v)) = 1$. For the remaining vertices, using the same argument, we have $f((s, v)) = 0$. Summing this over all vertices in L, we have $v(f) = |T|$. Now since no two edges in S share an endpoint, we have $|S| = |T|$, and hence we have $|S| = v(f)$, as required.

From here, we can see that we have constructed a flow f with $v(f)$ being equal to the cardinality of the maximum matching, and also that we have constructed a matching with the cardinality being equal to the maximum value of any flow f.

Hence, the maximum value of a flow equals the cardinality of a certain valid matching, which is at most the cardinality of a maximum matching.

Also, the cardinality of a maximum matching equals the value of a certain valid flow, which is at most the maximum value of a flow.

Combining these inequalities, we can see that equality holds, and thus the cardinality of a maximum matching in the graph equals the maximum value of any flow in the corresponding network, as required.

4. (20 points) There are n stationary mobile-phones $c_1, ..., c_n$ and n stationary mobile-phone towers $t_1, ..., t_n$. The distances between mobile-phones and towers are given to you in an $n \times n$ matrix d, where $d[i, j]$ denotes the distance between phone c_i and tower t_j. It is possible for a mobile-phone c_i to connect to a tower t_j if and only if the distance between c_i and t_j is at most D, where D is the connecting radius. Furthermore, at one time, a mobile-phone can connect to at most one tower and a tower can allow at most one connection. Your goal as a Communications Engineer is to figure out whether all mobile-phones are usable simultaneously. That is, whether it is possible for all mobile-phones to connect simultaneously to distinct towers. Answer the following questions.

   (a) Consider a simple example with 5 mobile-phones and 5 towers. Let the connecting radius be $D = 2$ miles. The distance matrix for this example is as given in Figure 2.

d	1	2	3	4	5
1	1	2	3	4	7
2	4	1	1	5	12
3	5	7	2	1	11
4	4	3	6	1	1
5	1	21	8	9	13

Figure 2: Distance matrix d for part (a) of question 4.

Prove or disprove: It is possible for all 5 mobile-phones to simultaneously connect to distinct towers for this example.

Solution: Yes, it is possible. Here is a matching from cell-phones to towers that demonstrates how connections can be obtained.

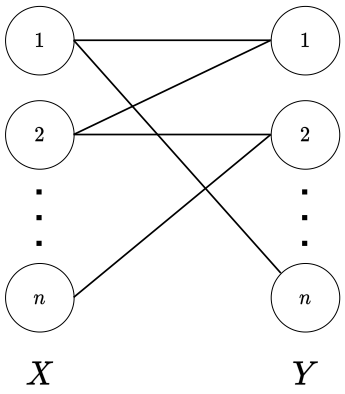
$1 \to 2$	$d[1, 2] = 2$
$2 \to 3$	$d[2, 3] = 1$
$3 \to 4$	$d[3, 4] = 1$
$4 \to 5$	$d[4, 5] = 1$
$5 \to 1$	$d[5, 1] = 1$

   (b) Design an algorithm that takes inputs n, D, and the distance matrix d, and outputs "yes" if it is possible for all mobile-phones to simultaneously connect to distinct towers (within the connecting radius D) and "no" otherwise. Analyze the running time of the algorithm and give proof of correctness.

Solution: We will show how the given problem may be reduced to the problem of determining if there exists perfect matching in a bipartite graph. This reduction will allow us to solve the given problem using any of the algorithms for the perfect matching.

Proof of Correctness Consider the bipartite graph G with 2 partitions X and Y of size n each. There is an edge between $X[i]$ and $Y[j]$ *iff* $d[i, j] \leq D$. An illustration of a possible graph G is as

shown below.



Claim 1. *All mobile phones will be usable simultaneously* iff *there exists a perfect matching, i.e. matching of size n, in the bipartite graph G as constructed above.*

Proof. We will prove this claim by proving the implication from both the sides, as follows:

Claim 2. *Suppose that it is possible to obtain a pairing P of mobile phones to the towers such that all mobile phones are usable. Then, the matching $M = \{(X[i], Y[j]) \mid (i, j) \in P\}$ is a valid matching of size n in G.*

Proof. First we will show that $M \subseteq E(G)$. For this, note that if $(X[i], Y[j]) \in M$ then $(i, j) \in P$. And since P is a valid pairing by assumption, we have that $d[i, j] \leq D$. Therefore $(X[i], Y[j]) \in E(G)$ by construction of G. Hence, $M \subseteq E(G)$.

Note that P has size n by definition, since each phone is used. Now since $|M| = |P|$, we have $|M| = n$. We now only need to show that M is a valid matching.

Now, suppose M is not a valid matching. Then, there must be a vertex $v \in G$ such that it has more than one incident edge in M. Now, $v \in X$ or $v \in Y$. If $v = X[i]$ for some i then it means that mobile phone i is connected to more than one tower. Similarly if $v = Y[j]$ for some j, then it means that tower j is allowing more than one connection at a time. Both of these cases violate the problem constraints and contradict the fact that P is a valid pairing. Hence we must have that no such $v \in G$ can exist. Therefore, M is a valid matching. □

Claim 3. *Now, suppose there exists a matching M of size n in G. Then, pairing $P = \{(i, j) \mid (X[i], Y[j]) \in M\}$ of mobile phones to towers is a valid pairing such that all mobile phones are usable.*

Proof. Note that by definition $|P| = |M| = n$. So P pairs each mobile phone to a tower. To show that P is a valid pairing we need to show that if mobile phone i is paired with tower j then $d[i, j] \leq D$, and, each mobile phone is paired with atmost one tower and vice versa.

By definition of P, $(i, j) \in P$ iff $(X[i], Y[j]) \in M$. Also, by construction of G if there is an edge between $X[i]$ and $Y[j]$ it implies that $d[i, j] \leq D$.

Now, if possible, let i be a mobile phone that is paired with more than one tower, say j_1, j_2 in P. Then we have that $(X[i], Y[j_1]), (X[i], Y[j_2]) \in M$. But this is a contradiction because by definition a matching includes only vertex disjoint edges. *Mutatis mutandis*, we can argue that all towers j in P will be paired with exactly one mobile phone. □

Hence we have proved our original claim. □

Using the above claim we have reduced the given problem into the problem of existence of perfect-matching in a bipartite graph G. So, to obtain the answer to original problem we will construct the graph G as defined and then calculate the size of maximum matching in G. The answer will be "yes" *iff* the size of maximum matching is n. The algorithm for calculating size of max-matching in a bipartite graph G has already been discussed in the class using the application of network flows. So in the algorithm presented below we will assume the existence of subroutine MaxMatching which takes a bipartite graph G and returns size of maximum matching.

Pseudocode

```
1: function ConnectionPossible(n, D, d)
2:     let X ← set of n nodes.
3:     let Y ← set of n nodes.
4:     let G ← (X ∪ Y, ∅)
5:     for i ∈ [1...n] do
6:         for j ∈ [1...n] do
7:             if d[i, j] ≤ D then
8:                 add edge (X[i], Y[j]) to G
9:             end if
10:        end for
11:    end for
12:    if MaxMatching(G) = n then
13:        return "yes"
14:    else
15:        return "no"
16:    end if
17: end function
```

Running Time Analysis First initialization of G with no edges and $2n$ nodes will take $O(n)$ time in adjacency list representation. Then, we have a nested loop with $O(n^2)$ iterations and in each iteration we potentially add an edge to G. In the adjacency list representation adding an edge is equivalent to appending to the end of a linked list (or two). Therefore it will only take $O(1)$ time per iteration. So G will be a graph with $2n$ vertices and at most n^2 edges in the worst case. MaxMatching algorithm routine is as discussed in the class, internally it will transform G to a directed graph with $2n + 2$ vertices (source and sink included) and $n^2 + 2n$ edges in the worst case. Then it will apply the ford-fulkerson algorithm to obtain the max flow in this transformed graph which will therefore take $O(n \cdot (n + n^2)) = O(n^3)$ time (since the max flow is upper bounded by n, and the number of vertices + edges is in $O(n + n^2)$). So overall worst case time complexity of the algorithm is $O(n^3)$.

5. (25 points) Town authorities of a certain town are planning for an impending virus outbreak. They want to plan for panic buying and taking cues from some other towns, they know that one of the items that the town may run out of is toilet paper. They have asked your help to figure out whether the toilet paper demand of all n residents can be met. They provide you with the following information:

   - There are n residents $r_1, ..., r_n$, m stores $s_1, ..., s_m$, and p toilet paper suppliers $x_1, ..., x_p$.
   - The demand of each of the residents in terms of the number of rolls required.
   - The list of stores that each of the residents can visit and purchase rolls from. A store cannot put any restriction on the number of rolls a customer can purchase given that those many rolls are available at the store.
   - The number of rolls that supplier x_j can supply to store s_i for all $i \in \{1, ..., m\}$ and $j \in \{1, ..., p\}$.

   The above information is provided in the following data structures:

   - A 1-dimensional integer array $D[1...n]$ of size n, where $D[i]$ is the demand of resident r_i.
   - A 2-dimensional 0/1 array $V[1...n, 1...m]$ of size $n \times m$, where $V[i, j] = 1$ if resident r_i can visit store s_j and $V[i, j] = 0$ otherwise.
   - A 2-dimensional integer array $W[1...m, 1...p]$ of size $m \times p$, where $W[i, j]$ is the number of rolls of toilet paper that the supplier x_j can supply to store s_i.

   Design an algorithm to determine if the demand of all residents can be met. That is, given (n, m, p, D, V, W) as input, your algorithm should output "yes" if it is possible for all residents to obtain the required number of rolls and "no" otherwise. Argue correctness and discuss running time.

   (*For example, consider that the town has two residents, one store and two suppliers. If* $D = [2, 2], V = \left[\begin{smallmatrix}1\\1\end{smallmatrix}\right]$, *and* $W = [2, 2]$, *then the demand can be met. However, if* $D = [2, 2], V = \left[\begin{smallmatrix}1\\1\end{smallmatrix}\right]$, *and* $W = [2, 1]$, *then the demand cannot be met.*)

Solution:

Pseudocode

```
1:  function IsRequirementMet(n, m, p, D, V, W)
2:      Create source and sink vertices s, t
3:      Create vertices S[1...p] corresponding to the suppliers
4:      Create vertices T[1...m] corresponding to the stores
5:      Create vertices R[1...n] corresponding to the residents
6:      Let G be a graph with all the above vertices
7:      for i ∈ [1...p] do
8:          Add an edge (s, S[i]) with capacity Σ_{j=1}^{m} W[j, i] to G
9:          for j ∈ [1...m] do
10:             Add an edge (S[i], T[j]) with capacity W[j, i] to G
11:         end for
12:     end for
13:     for i ∈ [1...n] do
14:         Add an edge (R[i], t) with capacity D[i] to G
15:         for j ∈ [1...m] do
16:             if V[i, j] = 1 then
17:                 Add an edge (T[j], R[i]) with capacity D[i] to G
18:             end if
19:         end for
20:     end for
```

```
    if Value(MaxFlow(G, s, t)) = ∑_{i=1}^{m} D[i] then
        return "yes"
    else
        return "no"
    end if
end function
```

Proof of Correctness By a flow, we shall always refer to an integer flow. The max-flow returned by the function MaxFlow is always an integer flow for a flow problem with integer capacities (if we use an algorithm like the Ford-Fulkerson algorithm).

By an *assignment*, we denote a tuple (A', B') of arrays of size $m \times p$, $n \times m$, where $A'[i, j]$ is the number of rolls supplier x_j supplies to store s_i, and $B'[i, j]$ is the number of rolls resident r_i gets from store s_j, and A', B' are both subject to the constraints of the problem, i.e., $0 \leq A'[i, j] \leq W[i, j]$, $B'[i, j] \geq 0$, and $\sum_{j=1}^{n} B'[j, i] \leq \sum_{j=1}^{p} A'[i, j] \quad \forall i \in [1 \dots m]$.

By an *efficient assignment*, we denote an assignment where the following conditions are satisfied:

$$\sum_{j=1}^{m} B'[i, j] \leq D[i] \quad \forall i \in [1 \dots n] \tag{1}$$

$$\sum_{j=1}^{n} B'[j, i] = \sum_{j=1}^{p} A'[i, j] \quad \forall i \in [1 \dots m] \tag{2}$$

In other words, an efficient assignment is one in which we have nothing left over at a store after the residents buy rolls, and all residents take no more rolls than their requirements.

We shall first show a bijection between valid flows in the graph and efficient assignments, where if flow f is mapped to the assignment a, then $v(f) = \sum_{i=1}^{n} \sum_{j=1}^{m} B'[i, j]$ (total rolls bought by residents).

Proof. Consider an efficient assignment (A', B').

Consider a flow defined as follows:

1. For an edge $(s, S[i])$, let the flow associated to this edge be $\sum_{j=1}^{m} A'[j, i]$.
2. For an edge $(S[i], T[j])$, let the flow associated to this edge be $A'[j, i]$.
3. For an edge $(T[j], R[i])$, let the flow associated to this edge be $B'[i, j]$.
4. For an edge $(R[i], t)$, let the flow associated to this edge be $\sum_{j=1}^{m} B'[i, j]$.

First we show that this is a valid flow.

1. Flow conservation:
   (a) At vertex $S[i]$: Note that the only incoming edge is $(s, S[i])$, and the only outgoing edges are $(S[i], T[j])$ for $j \in \{1 \dots m\}$. From the definition of the flow values of these edges, we are done.
   (b) At vertex $T[i]$: Note that since (A', B') is an efficient assignment, we have $\sum_{i=1}^{n} B'[i, j] = \sum_{j=1}^{p} A'[i, j]$. The only incoming edges are $(S[j], T[i])$ for $j \in \{1 \dots p\}$, and the only outgoing edges are $(T[i], R[j])$ for $jin\{1 \dots n\}$. By noting the definitions of A', B' and the equality mentioned above, we are done.

(c) At vertex $R[i]$: Note that the only incoming edges are $(T[j], R[i])$ for $j \in \{1 \dots m\}$, and the only outgoing edge is $(R[i], t)$. From the definition of the flow values of these edges, we are done.

2. Capacity constraints: Firstly note that all weights are non-negative integers. Now for the capacity constraints, note the following:

   (a) For an edge $(S[i], T[j])$, the flow associated to this edge is $A'[j, i]$. By the conditions of the problem, since an efficient assignment is a valid assignment, we have $A'[j, i] \leq W[j, i]$.

   (b) For an edge $(s, S[i])$, the flow associated to this edge is $\sum_{j=1}^{m} A'[j, i]$. By summing the result of the previous case over j, this is $\leq \sum_{j=1}^{m} W[j, i]$.

   (c) For an edge $(T[j], R[i])$, the flow associated to this edge is $B'[i, j]$. Since we have $B'[i, j] \leq \sum_{j=1}^{n} \leq D[i]$ (as it is a efficient assignment), we are done for this case.

   (d) For an edge $(R[i], t)$, the flow associated to this edge is $\sum_{j=1}^{m} B'[i, j]$. Since we have $\sum_{j=1}^{n} \leq D[i]$ (as it is a efficient assignment), we are done for this case.

Now we show that the value of this flow equals $\sum_{i=1}^{n} \sum_{j=1}^{m} B'[i, j]$. Note that by considering the $s - t$ cut $(A, B) = (V(G) \setminus \{t\}, \{t\})$, since there are no edges from B to A, we have $f^{in}(A) = 0$, so the value of the flow is $f^{out}(A) = \sum_{i=1}^{n} \sum_{j=1}^{m} B'[i, j]$, as needed.

Hence we have shown that for any valid assignment, there exists a valid flow in the network with value being equal to the total number of rolls bought by residents.

Now consider any flow f.

Consider the following assignment:

$$A'[i, j] = f((S[j], T[i])) \tag{3}$$
$$B'[i, j] = f((T[j], R[i])) \tag{4}$$

Now we need to show that this is an efficient assignment.

1. It is an assignment:

   For this we need to show that $0 \leq A'[i, j] \leq W[i, j]$, $0 \leq B'[i, j]$ and $\sum_{j=1}^{n} B'[j, i] \leq \sum_{j=1}^{p} A'[i, j] \quad \forall i \in [1 \dots m]$. However, the first of these follows from the fact that $f((S[j], T[i]))$ is flow associated to an edge with capacity $W[i, j]$ in a valid flow f. The second of these follows from $0 \leq B'[i, j]$ since it is flow associated to an edge and must be non-negative. The third of these follows from the fact we will show below.

2. It is an efficient assignment:

   (a) Consider flow conservation applied at vertex $R[i]$. Then the total incoming flow is $\sum_{j=1}^{m} f((T[j], R[i])) = \sum_{j=1}^{m} B'[i, j]$, which must equal the outgoing flow. And the only outgoing edge is $(R[i], t)$ which has a capacity of $D[i]$. Hence, we must have $\sum_{j=1}^{m} B'[i, j] \leq D[i]$, which verifies the first condition for an assignment to be an efficient assignment.

   (b) Consider flow conservation applied at vertex $T[i]$. The total incoming flow is $\sum_{j=1}^{p} f((S[j], T[i])) = \sum_{j=1}^{p} A'[i, j]$. The total outgoing flow is $\sum_{j=1}^{n} f((T[i], R[j])) = \sum_{j=1}^{n} B'[j, i]$. By flow conservation, this verifies the second condition for an assignment to be an efficient assignment.

Now we need to show that $v(f) = \sum_{i=1}^{n} \sum_{j=1}^{m} B'[i,j]$.

Consider the $s-t$ cut $(A, B) = (\{s\} \cup S[1 \dots p] \cup T[1 \dots m], R[1 \dots n] \cup \{t\})$.

Note that there is no edge from B to A, so $f^{in}(A) = 0$, hence we have $v(f) = f^{out}(A)$. The only edges from A to B are the edges between vertices of the form $T[j], R[i]$. Hence we have $v(f) = f^{out}(A) = \sum_{i=1}^{n} \sum_{j=1}^{m} f((T[j], R[i])) = \sum_{i=1}^{n} \sum_{j=1}^{m} B'[i,j]$, as needed.

From here we see that there is a bijection between efficient assignments and valid flows in the network which have the value of the flow being equal to the total number of rolls bought by the residents in the efficient assignment.

□

Now we shall show that if there exists an assignment which satisfies all the residents' needs, then there also exists an efficient assignment which satisfies all the residents' needs.

Proof. Consider any assignment (A', B') which satisfies all the residents' needs. We shall construct an efficient assignment that also satisfies all the residents' needs using the following algorithm (note that this algorithm is never run in the final algorithm, and we aim to use this as an aid in a proof of the claim above):

Require: $\sum_{j=1}^{m} B'[i,j] \geq D[i] \quad \forall i \in [1 \dots n]$
Require: $0 \leq B'[i,j] \quad \forall (i,j) \in [1 \dots n] \times [1 \dots m]$
Require: $0 \leq A'[i,j] \leq W[i,j]$
Require: $\sum_{j=1}^{n} B'[j,i] \leq \sum_{j=1}^{p} A'[i,j] \quad \forall i \in [1 \dots m]$

function GetEfficientAssignment(A', B')
  let $A'' \leftarrow A, B'' \leftarrow B$
  for $i \in [1 \dots n]$ **do**
    while $\sum_{j=1}^{m} B''[i,j] > D[i]$ **do**
      $j = \arg\max_j B''[i,j]$
      $B''[i,j] \leftarrow B''[i,j] - 1$
    end while
  end for
  for $i \in [1 \dots n]$ **do**
    while $\sum_{j=1}^{n} B''[j,i] < \sum_{j=1}^{p} A''[i,j]$ **do**
      $j = \arg\max_j A''[i,j]$
      $A''[i,j] \leftarrow A''[i,j] - 1$
    end while
  end for
  return (A'', B'')
end function

We claim that the above algorithm returns an efficient assignment that also satisfies the demands of all residents.

1. It is an assignment

   (a) $B''[i,j] \geq 0$: Note that we have $D[i] \geq 0$, and hence we decrement some $B''[i,j]$ if the sum of $B''[i,j']$ over all j' is positive. Note that the maximum $B''[i,j']$ must be always positive (if not, then the whole sum is non-positive). Hence we decrement a $B''[i,j]$ by 1 in a step only if it is positive. This implies that once we reach zero (which we reach since B'' is an integer matrix and the element we decrement decreases by exactly 1 at each decrement), we never decrement it further, proving this part.

(b) $0 \leq A''[i,j] \leq W[i,j]$: The second part follows from the fact that $A'[i,j] \leq W[i,j]$ and we either keep the value the same or decrease it. For the first part, note that from the previous point, we have $\sum_{j=1}^{n} B''[j,i] \geq 0$. By a completely analogous argument as in the previous point, we decrement $A''[i,j]$ only if it is positive, and hence we are done for this part.

(c) $\sum_{j=1}^{n} B''[j,i] \leq \sum_{j=1}^{p} A''[i,j]$: We claim that equality in fact holds, and that is done in the next part.

2. It is an efficient assignment

   (a) $\sum_{j=1}^{m} B''[i,j] \leq D[i] \quad \forall i \in [1 \dots n]$: There is a stronger assertion in the next point about satisfaction of all residents, and it is that equality holds, so we defer this to the next point.

   (b) $\sum_{j=1}^{n} B''[j,i] = \sum_{j=1}^{p} A''[i,j] \quad \forall i \in [1 \dots m]$: Either the loop runs, or the first condition is false. Since we never increment $B''[i,j]$, we have $\sum_{j=1}^{n} B''[j,i] \leq \sum_{j=1}^{n} B'[j,i]$. By the condition that stores can't supply more than the suppliers supply them, we have $\sum_{j=1}^{n} B''[j,i] \leq \sum_{j=1}^{n} B'[j,i] \leq \sum_{j=1}^{p} A'[i,j]$. Hence if the loop doesn't run, we have $\sum_{j=1}^{n} B''[j,i] = \sum_{j=1}^{p} A'[i,j] = \sum_{j=1}^{p} A''[i,j]$. In the case that the loop runs, we keep decrementing the value of $\sum_{j=1}^{p} A''[i,j]$ by precisely 1 in each step, till we reach a value $\leq \sum_{j=1}^{n} B''[j,i]$. Since A'', B'' are integer matrices, the value we reach is precisely $\sum_{j=1}^{n} B''[j,i]$, and we are done.

3. It satisfies the demands of all residents, i.e., $\sum_{j=1}^{m} B''[i,j] = D[i] \quad \forall i \in [1 \dots n]$: Note that by the condition that the input assignment must satisfy the demands of all the residents, the loop must either run, or we have $\sum_{j=1}^{m} B''[i,j] = D[i]$ already. If the loop runs, we keep decrementing the value of $\sum_{j=1}^{m} B''[i,j]$ by 1 in each step, till we reach a value $\leq D[i]$. Since B'' is an integer matrix, the value we reach is precisely $D[i]$.

From the above analysis, we have proven the claim, as required. □

Hence from here, we get that we only need to search for an efficient assignment that satisfies the needs of all residents, and by the bijection, this is equivalent to searching for a flow which satisfies all the demands.

Using the cut $(A, B) = (V(G) \setminus \{t\}, \{t\})$, noting that $f^{in}(A) = 0$ (since there is no edge from B to A), the value of a flow f in the network is $f^{out}(A)$. The capacity of this cut is $\sum_{i=1}^{n} D[i]$, so we have $\textsc{Value}(\textsc{MaxFlow}(G, s, t)) = f^{out}(A) \leq \sum_{i=1}^{n} D[i]$, where equality holds iff all the edges $(R[i], t)$ have flow through them equal to the capacity of that edge. By considering the efficient assignment corresponding to the maximum flow, we have the fact that there exists an efficient assignment that can satisfy all residents if and only if $\textsc{Value}(\textsc{MaxFlow}(G, s, t)) = \sum_{i=1}^{n} D[i]$.

Now we argue as follows:

1. If there exists an assignment satisfying all residents, then there exists an efficient assignment satisfying all residents, and such an assignment is detected by the algorithm and we return "yes".

2. If we return "yes", then there is an efficient assignment satisfying all residents, and since an efficient assignment is an assignment, there is an answer to the problem.

Hence, we return "yes" if and only if there is a solution to the problem, and "no" in the other case (i.e., when there is no solution to the problem).

Time Complexity Analysis Consider the cuts $(\{s\}, V(G)\setminus\{s\})$, $(V(G)\setminus\{t\}, \{t\})$. Both of these cuts have capacities at least the maximum flow in the graph, from the max-flow-min-cut theorem. Hence the maximum flow is upper bounded by

$$C = \min\left(\sum_{i=1}^{n} D[i], \sum_{i=1}^{p}\sum_{j=1}^{m} W[j,i]\right)$$

The number of vertices in the graph is $2+p+m+n$, and the number of edges is $p+pm+mn+n$.

Hence the call to MaxFlow takes $O((1+p+m+n+pm+mn)C)$ time, where C is defined as above. The call to the value function takes $O(1+p+m+n+pm+mn)$ time (the total number of edges and vertices in the graph).

Creating the whole graph takes $O(1+p+m+n+pm+mn)$ time as well (apart from the computations used for computing the capacities of edges $(s, S[i])$, and checking if $(T[j], R[i])$ needs to be added or not).

For computing the capacities of the edges, the sum can be computed in $O(pm)$ time and the checks for $V[i,j]$ happen in $O(mn)$ time, so the total graph construction time is again in $O(1+p+m+n+pm+mn)$.

Computing the total sum of demands takes time $O(m)$.

Hence the overall time complexity is $O((1+p+m+n+pm+mn)(C+1))$, where C is defined as $C = \min\left(\sum_{i=1}^{n} D[i], \sum_{i=1}^{p}\sum_{j=1}^{m} W[j,i]\right)$.