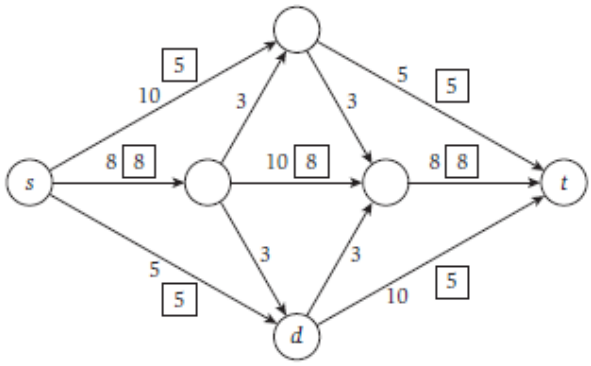
Answer the questions in the boxes provided on the question sheets. If you run out of room for an answer, add a page to the end of the document.

Name: _____ Wisc id: _____

Network Flow

1. *Kleinberg, Jon. Algorithm Design (p. 415, q. 3a)* The figure below shows a flow network on which an $s-t$ flow has been computed. The capacity of each edge appears as a label next to the edge, and the flow is shown in boxes next to each edge. An edge with no box has no flow being sent down it.

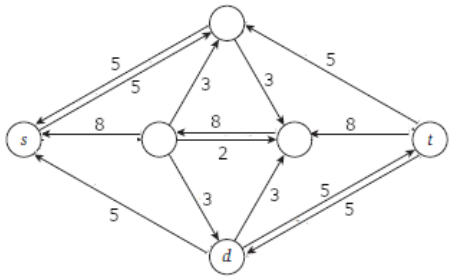


(a) What is the value of this flow?

Solution: 18

(b) Please draw the **residual graph** associated with this flow.

Solution:



(c) Is this a maximum $s-t$ flow in this graph? If not, describe an augmenting path that would increase the total flow.

Solution: No. We can add 3 flow along $s \to (up) \to (right) \to (left) \to (d) \to t$

2. *Kleinberg, Jon. Algorithm Design (p. 419, q. 10)* Suppose you are given a directed graph $G = (V, E)$. This graph has a positive integer capacity c_e on each edge, a source $s \in V$, a sink $t \in V$. You are also given a maximum $s - t$ flow through G: f. You know that this flow is *acyclic* (no cycles with positive flow all the way around the cycle), and every flow $f_e \in f$ has an integer value.

   Now suppose we pick an edge e^* and reduce its capacity by 1 unit. Show how to find a maximum flow in the resulting graph G^* in time $O(m + n)$, where $n = |V|$ and $m = |E|$.

   Solution: First, generate the residual graph G_f in $O(m+n)$ time. Since all capacities are integers, if e^* has nonzero capacity in G_f, then reducing its capacity by 1 will still allow us to retain the maximum flow f.

   Suppose e^* has no remaining capacity in G_f. It must have been part of some augmenting path $s \to t$. We can find such an augmenting path using BFS twice in G_f. Once to find a path from the source of e^* back to s, and once to find a path from t to the destination of e^*. (BFS runs in $O(m+n)$ time.) Reduce the flow through each edge by 1, increasing the capacities of the back edges to match. The result is a valid flow f^* in G^* with 1 less total flow.

   Either f^* is a maximum flow in G^*, or we can find the maximum flow using one more augmenting path found in $O(m + n)$ time with BFS.

3. *Kleinberg, Jon. Algorithm Design (p. 420, q. 11)* A friend of yours has written a very fast piece of code to calculate the maximum flow based on repeatedly finding augmenting paths. However, you realize that it's not always finding the maximum flow. Your friend never wrote the part of the algorithm that uses backward edges! So their program finds only augmenting paths that include all forward edges, and halts when no more such augmenting paths remain. (Note: We haven't specified *how* the algorithm selects forward-only augmenting paths.)

   When confronted, your friend claims that their algorithm may not produce the maximum flow every time, but it is guaranteed to produce flow which is within a factor of b of maximum. That is, there is some constant b such that no matter what input you come up with, their algorithm will produce flow at least $1/b$ times the maximum possible on that input.

   Is your friend right? Provide a proof supporting your choice.

   Solution: No. Imagine a graph G set up with a source node on the left, a sink node on the right, and two columns of nodes in the middle $a_1 \dots a_n$ and $b_1 \dots b_n$. There is an edge with capacity 1 from the source, to each node a_i, from each node a_i to its matching b_i, and from each b_i to the sink. Thus, there is a trivial flow of n available in this graph.

   Now imagine a graph G' which is identical to G except it also has an edge from each b_i to a_{i+1}. (Note: b_n does not get a new edge, nor does a_1.) The same flow from before is still available, so the maximum flow is at least n. Yet if my first augmenting path runs $s \to a_1 \to b_1 \to a_2 \to b_2 \dots a_n \to b_n t$, my friend's algorithm will halt immediately after completing this 1-flow path.

   No matter what n I might pick, there is a graph with $2n+2$ nodes where it is possible for my friend's algorithm to produce flow only $1/n$ times the maximum possible.

4. *Kleinberg, Jon. Algorithm Design (p. 418, q. 8)* Consider this problem faced by a hospital that is trying to evaluate whether its blood supply is sufficient:

   In a (simplified) model, the patients each have blood of one of four types: A, B, AB, or O. Blood type A has the A antigen, type B has the B antigen, AB has both, and O has neither. Patients with blood type A can receive either A or O blood. Likewise patients with type B can receive either B or O type blood. Patients with type O can only receive type O blood, and patients with type AB can receive any of the four types.

   (a) Let integers s_O, s_A, s_B, s_{AB} denote the hospital's blood supply on hand, and let integers d_A, d_B, d_O, d_{AB} denote their projected demand for the coming week. Give a polynomial time algorithm to evaluate whether the blood supply is enough to cover the projected need.

   Solution: This is a bipartite matching problem.

   Create a graph G with a node for each type of blood supply, and a node for each type of blood demand. Draw an edge with unlimited capacity between each compatible supply-demand pair. (For example, s_A has outgoing edges to d_A and d_{AB}.) Then add a source node s with an edge to each supply node whose capacity is the amount of supply for that type of blood. Create a sink node t with an edge from each demand node whose capacity is the amount of demand for that type of blood.

   The supply is sufficient for the projected demand if and only if the maximum flow through G is equal to the sum of the demand$\rightarrow t$ edge capacities.

   To show that this algorithm takes polynomial time, observe that there are 10 nodes in this graph: s, t, the 4 supply nodes, and the 4 demand nodes. Thus, both $|V| = O(1)$ and $|E| = O(1)$. The Ford-Fulkerson max-flow method runs in $O(|E||f^*|)$ time (with $|E|$ being the number of edges in the graph and $|f^*|$ being the value of the max flow) in settings with integer capacities. Since the max-flow equals the min-cut, we know that the max-flow is $O(d_A + d_B + d_O + d_{AB})$ (a cut of the graph). Thus, our algorithm runs in polynomial time (linear) with respect to the sum of the demands (and the supplies by similar logic). Additionally, we could use a max-flow algorithm such as Edmonds-Karp or Orlin's whose runtimes only rely on the number of edges and vertices in the graph to get an $O(1)$ solution.

   (b) Network flow is one of the most powerful and versatile tools in the algorithms toolbox, but it can be difficult to explain to people who don't know algorithms. Consider the following instance. Show that the supply is **insufficient** in this case, and provide an explanation for this fact that would be understandable to a non-computer scientist. (For example: to a hospital administrator.) Your explanation should not involve the words *flow*, *cut*, or *graph*.

blood type	supply	demand
O	50	45
A	36	42
B	11	8
AB	8	3

   Solution: At least one patient of blood type O or A will be unable to receive blood according to this projection.

   Patients of these blood types cannot receive blood from type B or type AB suppliers. There are 86 units of blood in the combined O+A supply, and there is demand for 87 units between those two types.

   Extra: Properly applied, only one patient will go without: Supply patients of each blood type from the blood supplies of the matching type. This leaves only type A patients, and we can reroute the remaining 5 units of O type blood to 5 of the 6 remaining A type patients.

5. Implement the Ford-Fulkerson method for finding maximum flow in graphs with only integer edge capacities, in either C, C++, C#, Java, or Python. Be efficient and implement it in $O(mF)$ time, where m is the number of edges in the graph and F is the value of the maximum flow in the graph. We suggest using BFS or DFS to find augmenting paths. (You may be able to do better than this.)

   The input will start with a positive integer, giving the number of instances that follow. For each instance, there will be two positive integers, indicating the number of nodes $n = |V|$ in the graph and the number of edges $|E|$ in the graph. Following this, there will be $|E|$ additional lines describing the edges. Each edge line consists of a number indicating the source node, a number indicating the destination node, and a capacity. The nodes are not listed separately, but are numbered $\{1 \ldots n\}$.

   Your program should compute the maximum flow value from node 1 to node n in each given graph.

   A sample input is the following:

```
2
3 2
2 3 4
1 2 5
6 9
1 2 9
1 3 4
2 4 1
2 5 6
3 4 4
3 5 5
4 6 8
5 6 5
5 6 3
```

   The sample input has two instances. For each instance, your program should output the maximum flow on a separate line. Each output line should be terminated by a newline. The correct output for the sample input would be:

```
4
11
```