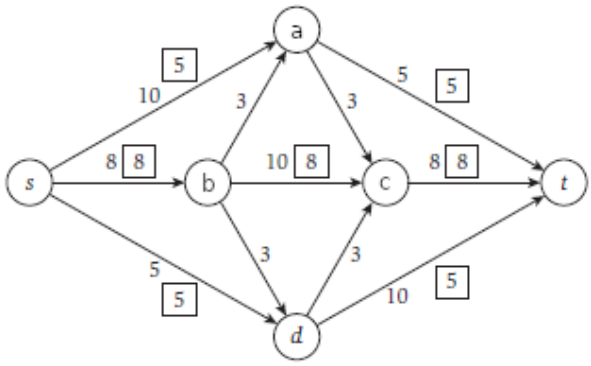
Answer the questions in the boxes provided on the question sheets. If you run out of room for an answer, add a page to the end of the document.

Name: Ki Min Kang Wisc id: 908-404-5062

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.

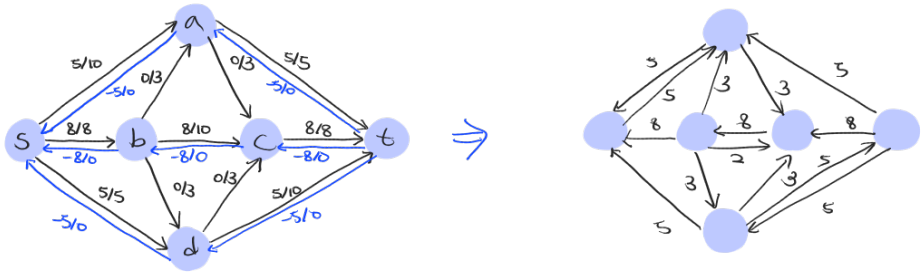


(a) What is the value of this flow?

Solution: 5 + 8 + 5 = 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 more flows in the path (s → a → c → b → d → 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:

Create the residual graph Gf in O(m+n) time, where capacities are integers.
If edge e* has a non-zero capacity in Gf, reducing it by 1 maintains the max flow f.
If e* has no capacity left in Gf, it was part of an augmenting path for s to t.
Use BFS twice to find an augmenting path: first from t to e*'s destination, then from e*'s source to s, in O(m+n) time.
Adjust flows along these paths by reducing each edge's flow by 1, creating a new valid flow f* in G* that is one less than the original flow
The new flow f* might be the maximum for G*, or another augmenting path can be found with BFS in O(m+n) time, resulting in the maximum flow for G*.

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, the friend's claim is incorrect. We can consider two scenarios to prove this.
1) A graph G has a source s, a sink t, and two columns of nodes a_1 to a_n and b_1 to b_n.
   Each node a_i is connected to b_i, and these to the sink, all with capacity 1.
   This setup allows a maximum flow of n.
2) A modified graph G' adds an edge from each b_i to a_{i+1}, except for the last b_n.
   If the first augmenting path chosen is $s \to a_1 \to b_1 \to a_2$ and so on to $b_n \to t$, it will only send a flow of 1 and then stop, not realizing the possible flow of n.

Therefore, the friend's algorithm might only achieve $\frac{1}{n}$ of the maximum flow, disproving their claim.

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:

Create a network with supply nodes for each blood type and a demand nodes for each patient blood type. Connect supply nodes to demand nodes, then connect a source to supply nodes to demand nodes to a sink, with edges' capacities equal to the supply or demand amount. Adequate supply is when maximum flow from the source to sink equals total demand. This network has 10 nodes and can be solved using the Ford-Fulkerson algorithm, with run time proportional to the number of edges and the maximum flow value.
For faster results, algorithms like Edmonds-Karp or Tin's method can be used since they depend only on edge count, providing an O(1) solution for this setup.

   (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:

We don't have enough blood for everyone who needs it. Everyone with type B or AB blood can get what they need, but when we add together all the type O and A blood we have, it's just 86 units, and the patients with these types need 87 units in total. Even though we're only short by one unit, it means at least one person won't get the blood they need.
Since type O blood can be given to anyone, we can use the extra type O blood for type A patients after all the type O patients get theirs. But there's still not quite enough to go around, leaving one type A patient without blood.

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 $m = |E|$ 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 $c(e)$. The nodes are not listed separately, but are numbered $\{1 \dots n\}$.

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

   Constraints:

   - $2 \leq n \leq 100$
   - $1 \leq m \leq \frac{n(n-1)}{2}$, the upper bound for m in an acyclic graph. For $n = 100$, $m \leq 4,950$.
   - $0 \leq c(e) \leq 100$

   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
```