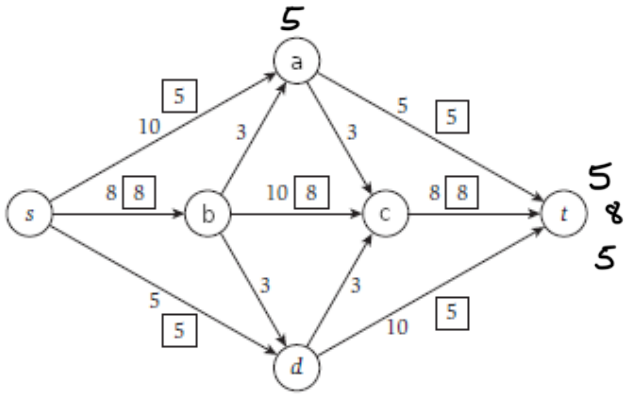
Answer the questions in the boxes provided on the question sheets. If you run out of room for an answer, add a page to the end of the document.

Name: Mahir Wisc id: mhkhan9

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.

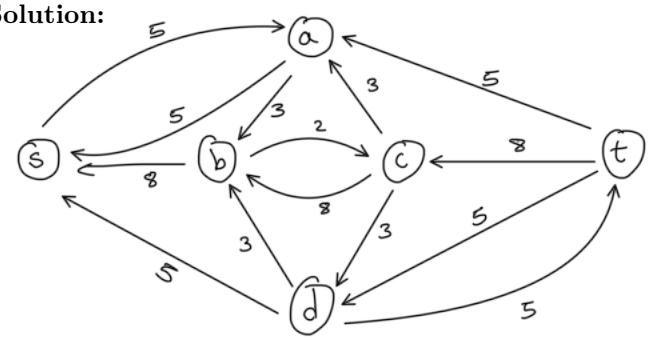


(a) What is the value of this flow?

Solution: The value of this flow is 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, it's not the max s-t flow in this graph. The augmenting path s → a → b → c → d → t would increase the flow by 2, to 20.

18

s → a → b → c → 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 $m = |E|$.

*can't build a new max flow from scratch, as that is pseudopolynomial and we must do it in O(n+m) time.

Solution: Since we are reducing the capacity of 1 edge by only 1 unit, then there must only be two possible scenarios: the value of f is the same, OR, the value of f has reduced by 1. Also, there are 2 cases for e^*, either the original e was saturated with flow, or it wasn't. If e wasn't saturated with flow, then reducing its capacity by 1 makes no difference to the flow, thus f would be the max flow for G^*. However, if e was saturated with flow, then that means the flow to e^* must be reduced by 1 to account for the capacity change. Assume e^* is an edge between (u,v), thus all flows from s to u and v to t should reduce in flow by 1. This will create the flow f', where f' = f-1
Now, run f' on G^*, and either no augmenting path will be found, thus f' = f-1 = max flow, or an augmenting path is found, and f' = f = max flow. —> Thus done in O(m+n) time.

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: Assume a ^bipartite^ graph ^G^ with a source node s, sink node t, and nodes $a_1 \dots a_n$ on side A and $b_1 \dots b_m$ on side B, with s connected to A nodes, and B nodes connected to t. There are edges from a_i to b_i. Assume a max flow f_{friend}.
Now assume a identical graph G^{**} except it also has edges from b_i to a_j $(j \neq i)$. Assume it also has flow f_{friend} available,

—> I don't know how to do this problem, it makes no sense to me.

—> what is b? How can I show b? I understand that a backward edge can lead to a larger flow, but i don't understand how a constant b is relevant to this, or what it might equate to in the graph's context.

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:

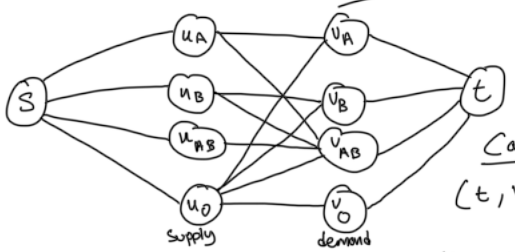
   In a (simplified) model, the patients each have blood of one of four types: A, B, AB, or O. Blood type A has the A antigen, type B has the B antigen, AB has both, and O has neither. Patients with blood type A can receive either A or O blood. Likewise patients with type B can receive either B or O type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

Type	Antigen	Receive?
A	A	A, O
B	B	B, O
AB	A, B	A, B, AB, O
O	∅	O

Solution: $S = \{s_O, s_A, s_B, s_{AB}\}$ = hospital supplies for each blood type.

$T = \{d_A, d_B, d_O, d_{AB}\}$ = projected demand for each blood type.



Node Set (V) = $\{s, t, u_A, u_B, u_{AB}, u_O, v_A, v_B, v_{AB}, v_O\}$

Edge set (E) = $\{(s, (u_A, u_B, u_{AB}, u_O)), ((v_A, v_B, v_{AB}, v_O), t), (u_i, v_j)$: if j can receive blood type i $\}$

Capacities: $(s, u_i) = s_i \in S$, $(u_i, v_j) = d_j$, $i \in$ blood types, $i, j \in$ blood types, $(t, v_i) = v_i \in T$ → The capacity of edges between u and v is the demand v needs, incase u can supply all of it.

Now we have successfully reduced the problem into a network flow problem, thus we run the FF algorithm to find the max flow on our Graph $G = (V, E)$, denoted by f_{max}

Then, if $f_{max} = \sum_{i \in T} d_i$, then the supply meets the projected demand

This is all done in polynomial time, as it takes $O(nm)$ time as $n = |s| = |T|$, and $m = |E|$, and we know the FF method takes $O(mC)$ time to run, where $C = n$, thus $O(mn)$. Changing the NF reduction problem to the original problem also takes $O(mn)$. Thus, a polynomial time algo has been made.

   (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: So, the current blood supplies the hospital has available is not sufficient to meet the projected demand for the blood types. This is because, for example, the demand for blood type A cannot be met. A can only take blood types A and O. Giving all of A supply would still leave a demand of 6, and there is only 5 of O available to supply A (after supplying all O blood type demands). Thus, one patient of blood type A or O will have to go without the blood donor.

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
```