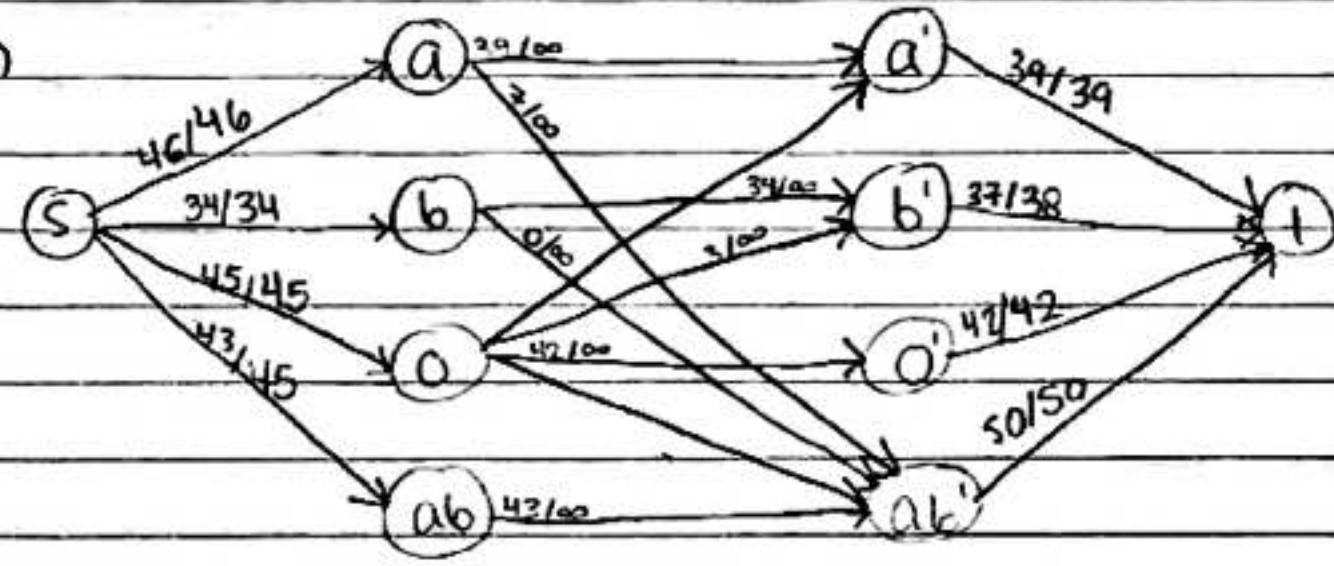
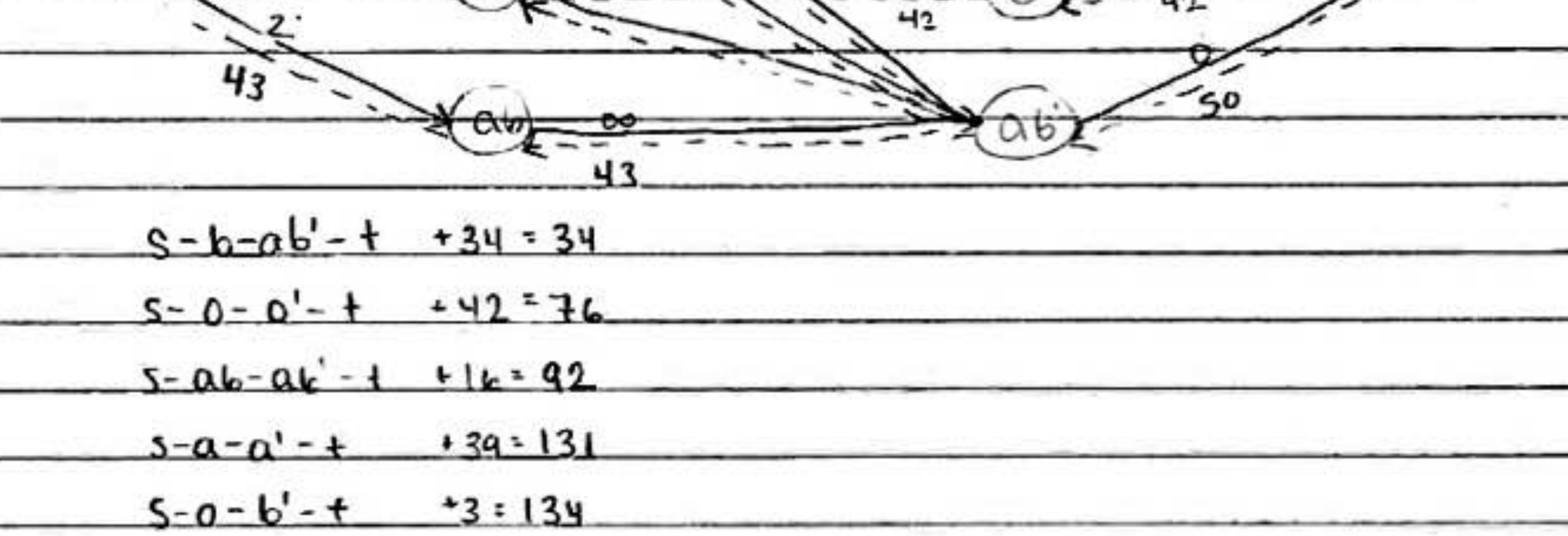
Assignment I: Olivia Woodhouse

1\. (a)



In this max flow formulation, the source represents the clinic and the sink represents the student populous that requires the transfusions. The capacities leaving the source represent the supply amount of each blood type and the capacities going to the sink represent the demand of each blood type. I chose the flow in the middle (between a,b,o,ab and a',b',o',ab') according to which patients can receive which blood types. I did not indicate capacities for these middle flows because we do not need to put a capacity on the amount of blood that is given to a group

(b) Residual Graph:



s-b-ab'-t +34 = 34

s-o-o'-t +42 = 76

s-ab-ab'-t +16 = 92

s-a-a'-t +39 = 131

s-o-b'-t +3 = 134

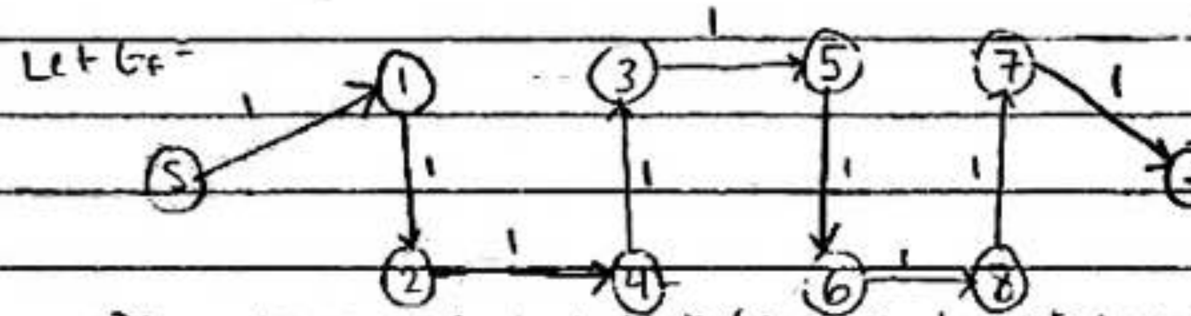
s-a-ab'-b-b'-t +7 = 141

s-ab-ab'-b-b'-t +27 = 168

no more augmenting paths at this point ∴ max flow = 168

(C) min cut: $\{S, ab, ab', a, a'\}$
(using DFS of Residual graph) 1+

2. Let G_F =



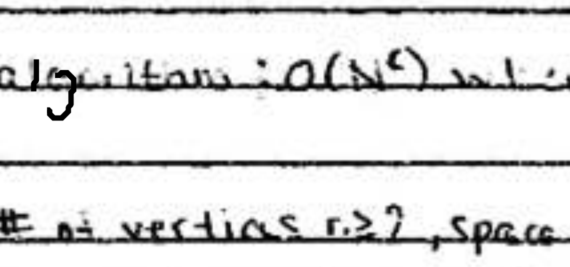
*let n = # of vertices in the graph
*remember cut is defined as a partition (A,B) where $s \in A$ and $t \in B$

a) How many cuts does this flow network have?
The key language for figuring out this problem is "single path starting at s and ending at t". Because $\exists$ only one path, this graph must be a line graph. Therefore there are $\boxed{2^{n-2}} = 256$ possible cuts because each edge (minus s and t) can either be in the (s) cut or not. However, there are $\boxed{n-1} = 9$ min-cuts.

b) How many of these cuts are min-cuts?
As defined in class, a min-cut is a cut of a graph with the smallest capacity. Therefore, $\exists$ 9 (in this graph) cuts in this graph are min-cuts because for all edges, $C_e = 1$, and because there is only 1 edge going from $A \to B$ for all possible cuts, the capacity of every possible cut = 1

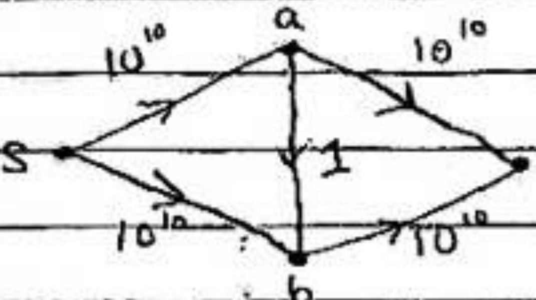
3. Polynomial algorithm: $O(N^c)$ where c is a fixed constant, N = # of bits required to represent the input

FF file: 1. # of vertices $n \geq 2$, space
2. $n \times n$ = 1 number each seperated by spaces
↳ represent in $n \times n$ matrix A. A_{ij} = capacity of the edge from $i \to j$ (base 10) ($0 \Rightarrow \nexists e_{i \to j}$)
↳ represented in decimal → 7-bit ASCII characters (for each decimal digit)
$N \geq 1000$ (divisible by 100)
so each digit is 7-bits



1. Show that the input size² (input file) is smaller than N bits.
2. Show that the FF run time can be longer than $2^{N/100}$ (not efficient)

let $N=1000 \rightarrow N/100 = 1000/100 = 10 \rightarrow 10^{10} = 10000000000$



each digit = 7 bits
(assume each space is also 7 bits)

input file: 1: # of vertices (4), space $\Rightarrow$ 14 bits (7 bits × 2)

2. $n \times n$ matrix $= n^2$, $n^2 - 2$ spaces

$\rightarrow n^2 = 16$, however $16 - 5 = 11$ spots in the matrix $= 0$
and 1 spot in the matrix $= 1$
$\rightarrow 12$ spots will be 7 bits each $= 84$ bits

the remaining 4 spots will be 11×7 bits each $= 77$ bits each (×4) $= 308$

and last but not least, the $4^2 - 2 = 14$ spaces taking up 7 bits of space each $(14 \times 7) = 98$

$\Rightarrow$ the whole file space $= 14 + 84 + 308 + 98 = 504$ bits total

so input file $= 504 \leq 1000$ (our N)

$\therefore$ input size^2 (or input file) $<$ N bits

As shown in class Ford-Fulkerson run time $= O(mnk)$
where m = total number of edges in the flow network, n = number of edges leaving the source, and k = largest capacity
we must show that when $N=1000$, FF can be longer than 2^{10}
$\rightarrow$ the naive algorithm of this graph (as shown in class) can take $10^{10} \times 2$ iterations $\times \underset{(\#\text{ of edges} = m)}{5}$

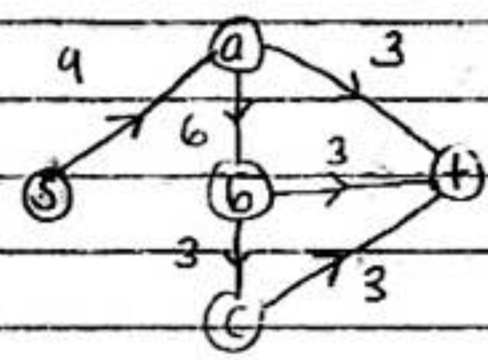
AND $(10^{10} \times 2) \times 5 > 2^{10}$ $\therefore$ the running time of the FF on this input can be longer than $2^{N/100}$ which is greater than $O(N^c)$ mean FF is <u>not</u> a polynomial-time algorithm

4. Remember: Ω (big omega) is the notation for asymptotic lower bounds

Must show that for every even m, $\exists$ a flow network that requires $m/2$ augmentations no matter how we choose the augmenting path.

m = # of edges of G

Let $m = 6$ and $n = \frac{m}{2} = \frac{6}{2} = 3$



In this case there are $\frac{m}{2} = 3$ augmentations. For every even m, we can just add another vertex with the flow coming from $s \to a \to b \to c \to$ 'new vertex' and going directly to t. We would then add 3 to each edge along this path and set the edge going from the new vertex $\to t$ to 3. For example, to add vertex d, we would create edge $c \to d$ and edge $d \to t$. Then we'd set $s \to a = 12$, $a \to b = 9$, $b \to c = 6$, $c \to d = 3$, and $d \to t = 3$. This would ensure a flow network that requires $\frac{m}{2}$ augmentations no matter how we choose the augmenting path.

7. remember: $val(f) = f^{out}(A) - f^{in}(A)$, must prove that $val(f)$ is equal to the sum of the flow on the edges going into the sink

→ As we proved in class, $val(f) = f^{out}(A) - f^{in}(A)$ ∴

we defined capacity of a cut as the sum of all the capacities going from A to B.

We ALSO proved that $val(f) = f^{out}(s) = \sum_{v \in A} f^{out}(v) - f^{in}(v)$

Lastly we proved that for all vertices (except source and sink), $f^{out}(v) = f^{in}(v)$ because of the conservation nature of the FF algorithm (vd sert)

So if $val(f) = f^{out}(s)$ AND $f^{out}(v) = f^{in}(v)$ the flow remaining (that it intends to) in the graph after reaching all of the vertices that are NOT source or sink will $= f^{out}(s)$ because this flow will be conserved as it goes through all the middle vertices.

After the flow has reached all intended vertices but the sink, it MUST go to (end at) the sink (because of the definition of a flow network claiming all flow leaves from the source and ENDS AT THE SINK).

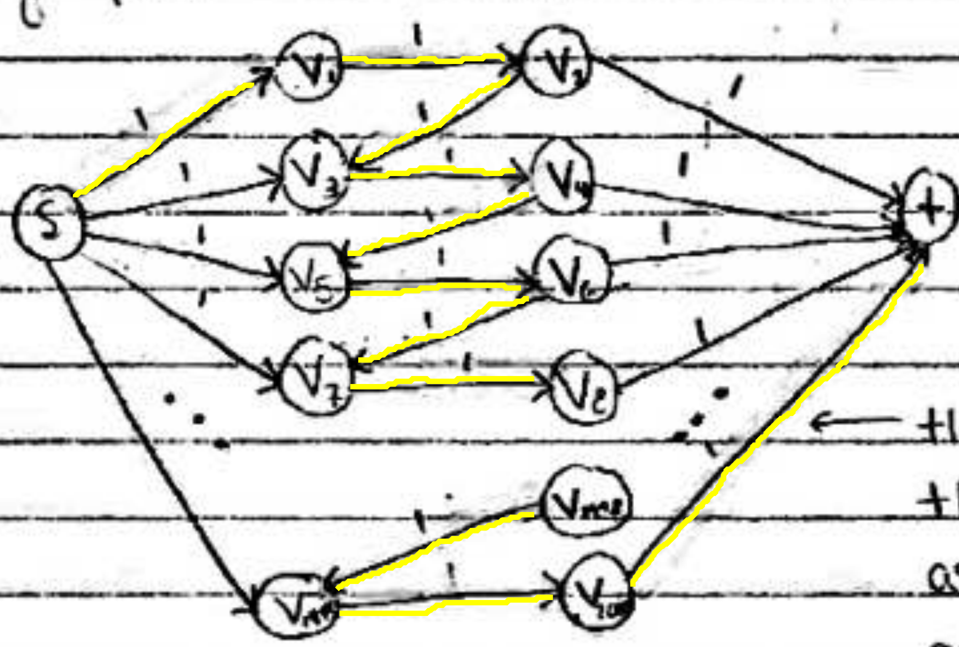
⇒ this flow going into the sink $= f^{out}(s)$ and because we already proved $f^{out}(s) = val(f)$ we know that $f^{in}(t) = val(f)$ (the sum of the flow on the edges going into the sink $= val(f)$)

8. Since we have already found the Max-Flow of this graph, suppose we used the Ford-Fulkerson algorithm. This means that we already have all the final flow values of each edge. In order to find such an edge (one that, increasing it by 1 would also increase the max flow) we would simply find a path that only includes ONE edge that is at max capacity. To do this in an efficient amount of time, we could create a graph from the residual graph where every (forward) edge with residual capacity = 0, set to 1. For all other (forward) edges set to 0. So to test if a path includes one of these edges, use our newly constructed graph. If there does NOT exist a path from $s \to t$

for example, a graph where all edges are at max capacity

with $\sum_{e \in path} value(e) = 1$, then this would be an example of a graph that does not have any edges like this (ie an example where such an edge does not exist). However, if the graph has a path with this summation of all the used edges = 1, then the edge in that path that has value set as 1 would be an edge that can increase the max-flow if it itself is increased. Using our newly constructed graph, we can find this edge using DFS which takes O(m) time.

5. Let our graph=



← the dot represent $V_9 - V_{999}$ that are in the same layout as the ones shown here. (a path going from $S \to V_9 \to V_{10} \to t$ (all edges of weight 1) as well as an edge coming from $V_8 \to V_9$ and so on.

If the normal ford-fulkerson is run on this graph max-capacity = 1,000. However, with this modified FF, if we follow the highlighted path, it is not possible to augment the flow any further.

6. No, if we pick the shortest augmenting path with this algorithm we do not necessarily find the max flow... an example of this would be