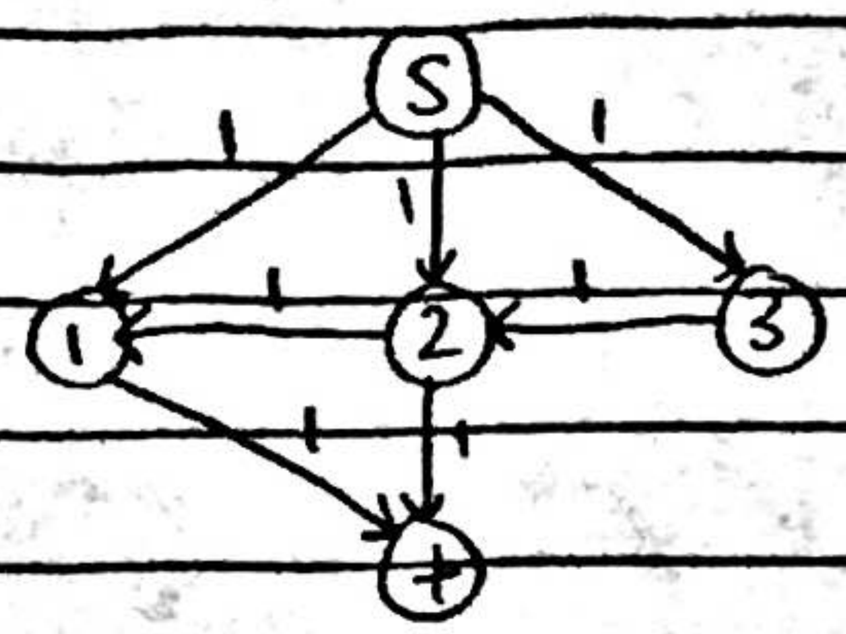
Assignment #2 Olivia Woodhouse

(1) Let G =



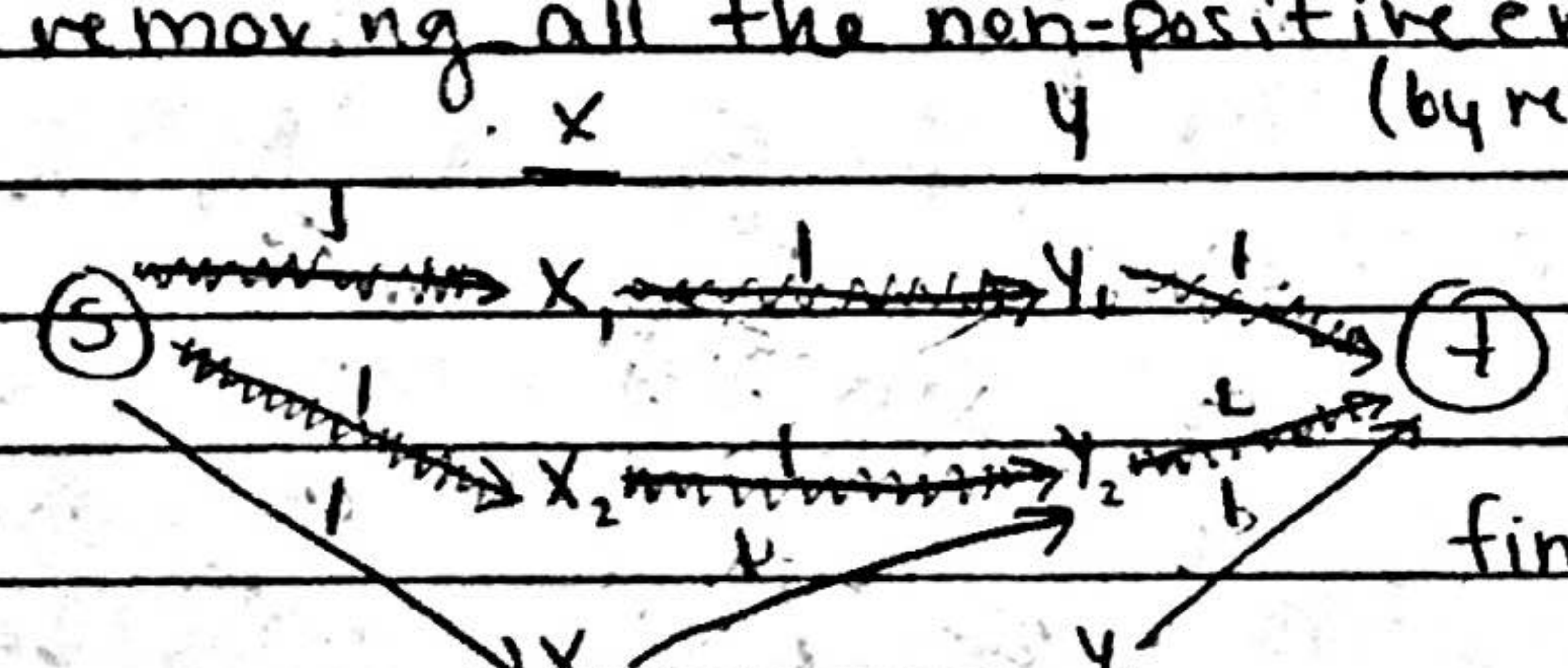
*note: In fact, no matter how you choose the paths to get a max-flow of 2, the min-cut will always be the same (in this case)

In this case, if an algorithm is used to find the maxflow, it will equal 2. However there are two different ways in which one can achieve this max-flow. For example, paths $s \to 1 \to t$ and $s \to 2 \to t$ achieve a max-flow of 2 but paths $s \to 3 \to 2 \to 1 \to t$ and $s \to 2 \to t$ also achieve this maxflow of 2. Let these two different ways of achieving maxflow be indicated by f and f'. So $f = s \to 1 \to t$ and $s \to 2 \to t$, and $f' = s \to 3 \to 2 \to 1 \to t$ and $s \to 2 \to t$. If f is the max-flow path chosen, the min-cut would be (A,B) where $A = \{s,1,2,3\}$ and $B = \{t\}$. If f' is the max-flow path chosen, the min-cut would be (A',B') where $A' = \{s,1,2,3\}$ and $B' = \{t\}$. Since $(A,B) = (A',B')$, this disproves the contention that if G has at least two different integer-valued maximum flows f and f', then there are at least two different minimum cuts (A,B) and (A',B').

same for paths $s \to 3 \to 2 \to t$ and $s \to 1 \to t$

(2) $n \times n$ matrix A with integer entries... essentially we would like to find an efficient way for removing all the non-positive entries. (by removing smallest # of rows + cols)

$$A = \begin{bmatrix} 0 & 2 & 1 \\ 3 & -1 & 2 \\ 4 & 0 & 1 \end{bmatrix}$$



find min vertex cover!

the question is asking how to select the smallest number of rows and columns.. (not specific rows and columns)

1. Create a bipartite graph flownetwork where all vertices in X represent the rows in A ($x_1, x_2, x_3, \ldots$) and all vertices in Y represent the columns in A ($y_1, y_2, y_3, \ldots$). Draw an edge from x_m to y_n if $\exists$ a non-positive entry in A_{mn}. Set all edges (including $s \to x_1, x_2, x_3, \ldots$ and $y_1, y_2, y_3, \ldots \to t$) to value 1.

2. Using this graph, find the min-vertex cover value. We proved in class that the min-vertex cover = min-cut = cap(A,B) which = max-flow. This means that you can find this value using FF.

* Min-vertex cover = the minimum # of edges + columns to remove
→ This is because the min-vertex cover is the least # of vertices to delete that would also delete all edges of the bipartite graph (not including s and t). This means that we are finding the smallest number possible of rows and columns to be removed, that would also remove all non-positive entries (represented by the edges between X vertices and Y vertices in our graph.

9

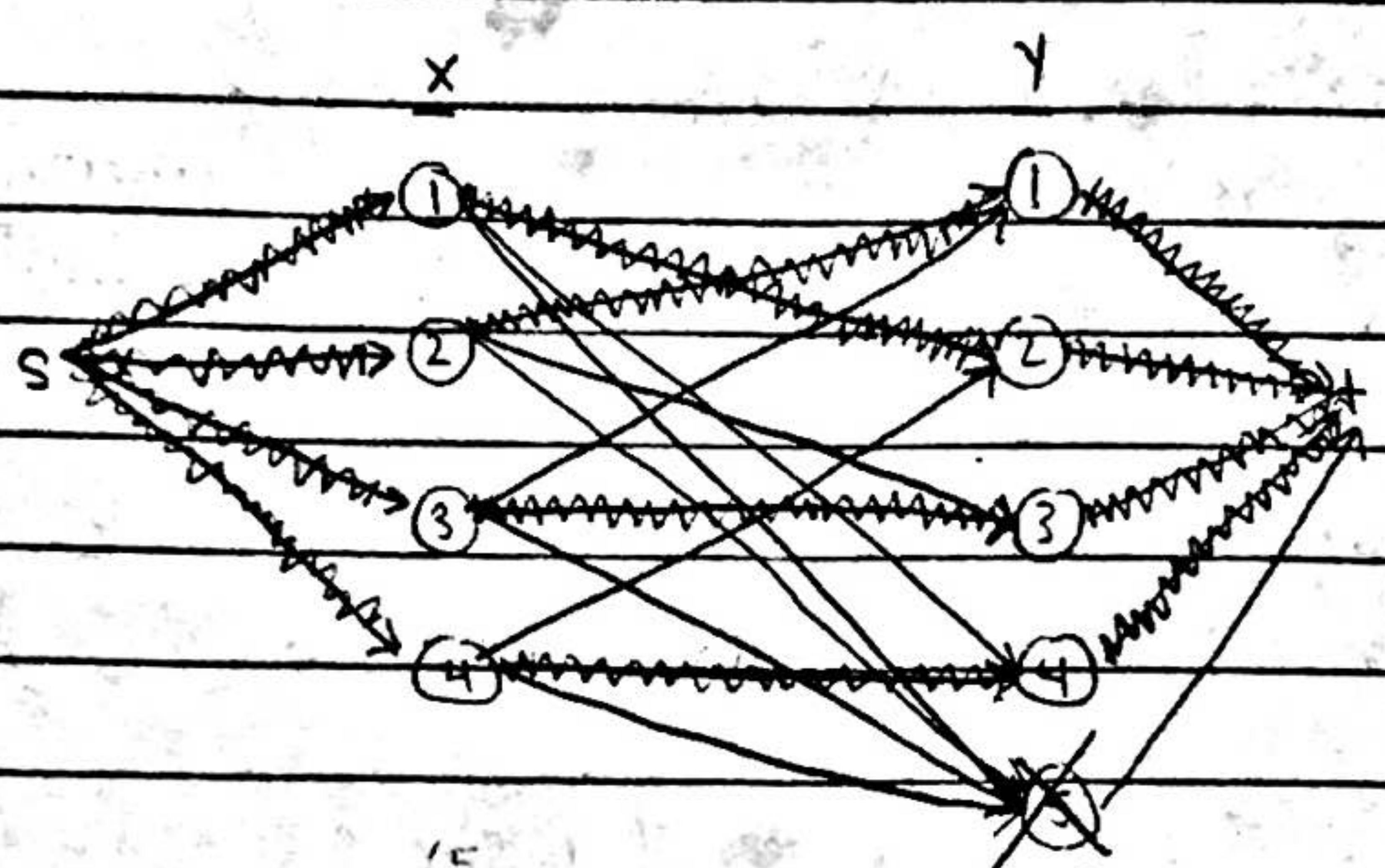
10

4. A = n×n matrix with ±1 entries

	1.	2.	3.	4.	5.
1.	-1	1	-1	1	1
2.	1	-1	1	-1	1
3.	1	-1	1	-1	1
4.	-1	1	-1	1	1

Y # of columns (5)

X # of rows (4)



*find maximum matching!
// max-flow (which we proved in class are equal!)

ignore because a 5th column will never be relavant ⇒ if |Y|>|X|, delete the extra vertices

...using this max-matching if X_x and Y_y have an edge between them, move row X to the row number that equals Y

...now we can reconstruct the matrix (into B)

	1.	2.	3.	4.	5.
2.	1	-1	1	-1	1
1.	-1	1	-1	1	1
3.	1	-1	1	-1	1
4.	-1	1	-1	1	1

= maximal diagonal matching

*note if |X|>|Y|, whichever rows were not matched with a Y can stay in their original position unless another row is moved to its spot in which case you can switch the row to the old spot of the one taking its place

*NOT relavant because it's an n×n matrix (ignore!)

So what does this look like in terms of an algorithm?

1. create a bipartite graph flow network in which there exists a source (and edges connecting the source with every vertex in X) and a sink (and edges connecting the source with every vertex coming from Y). Create a vertex in X for every row in the matrix (name these $X_1, X_2, X_3, \ldots$). Create a vertex in Y for every column in the matrix (name these $Y_1, Y_2, Y_3, \ldots$). If ~~|X|<|Y| do NOT create vertices for any Y that is > |X|.~~ Add an edge between X_m and Y_n if the entry in A at $A_{mn} = 1$.

forgot it's an n×n matrix (ignore!)

2. Find the maximum matching in the graph. As proved in class, we can find this by using FF to find the max-flow.

↳ Once we have found max-matching and recorded the paths with flow in them, we can use this information to construct matrix B... ← initiate a matrix B $n \times n$ and copy into it the contents of A

3. For all e connecting X_m and Y_n, (from FF)

if $\exists$ flow in e AND $m == n$,

Keep the row where it is in B

else if $\exists$ flow in e AND $m =! n$

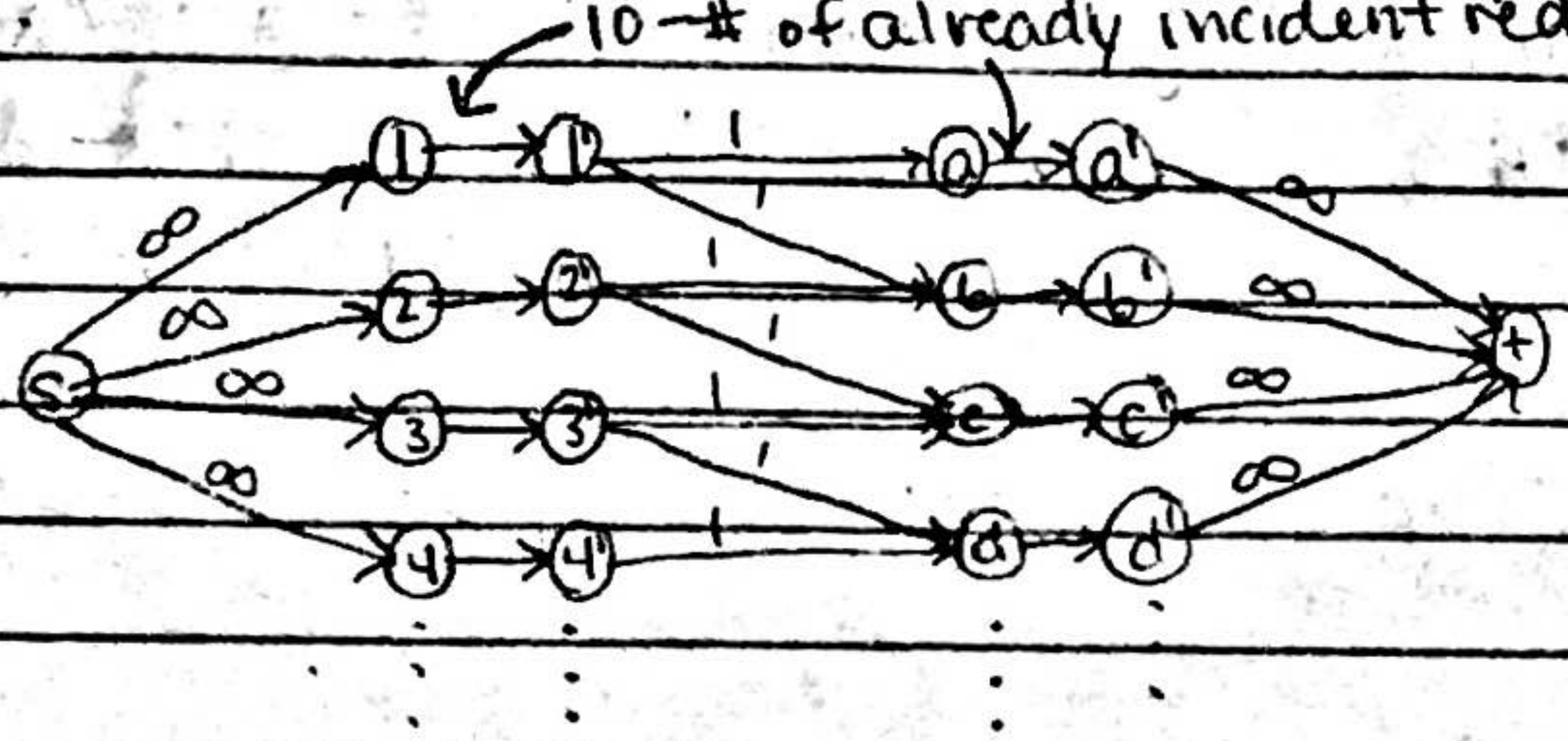
Switch the contents of row m and row n in B

→ there is no need to also switch colums because the wording of the problem clearly states that the rearranging must be row then columns and if the rows are rearranged first then the matrix is already maximized to full potential

③ can we color more edges red so that every vertex is incident to 10 red edges

10 − # of already incident red edges ← (let this number = x)

$f^{(in)} \leq X \ (\forall v \in V)$



In order to solve this problem using max flow, create a v' for $\forall v \in V$. v' will have 1 edge coming from v and all the original edges leaving from v will now be leaving v' (instead of v). This will almost act as a gate for flow to go through. If we set this edge between v and v' equal to 10 − the number of red-edges already incident to v (and now v'), this edge will represent the number of edges that still need to be colored red that are incident to v (and v') in order for it to be incident to exactly 10 edges. If we use a flow network setting all edges leaving s to ∞ and all edges entering t to ∞ and then all other edges (except edges between v and v' for $\forall v \in V$) to 1, then if the max flow is equal to the sum of all capacities (not final flow) of the edges between v and v' $\forall v \in X$ ← (1st partition of bipartite graph), then such a coloring is possible.

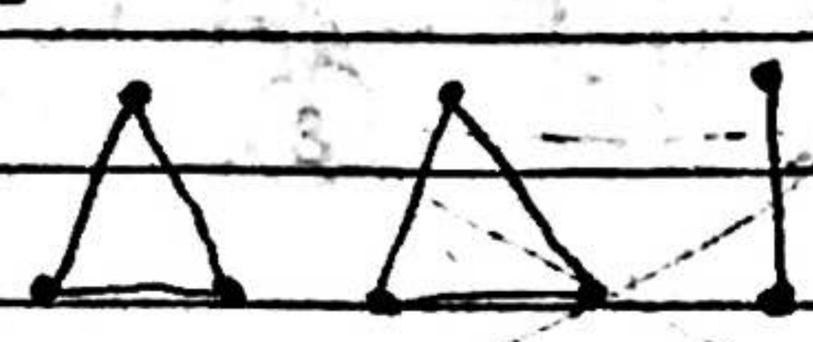
(5) $a_1 + \dots + a_k = b_1 + \dots + b_t = n$

G = n vertices that is formed by taking disjoint union of K cliques on $a_1, \dots, a_k$ vertices

let $a_1 = 3$, $a_2 = 3$, $a_3 = 2$

$b_1 = 1$, $b_2 = 4$, $b_3 = 3$ n=8

G = k=3 (cliques!)



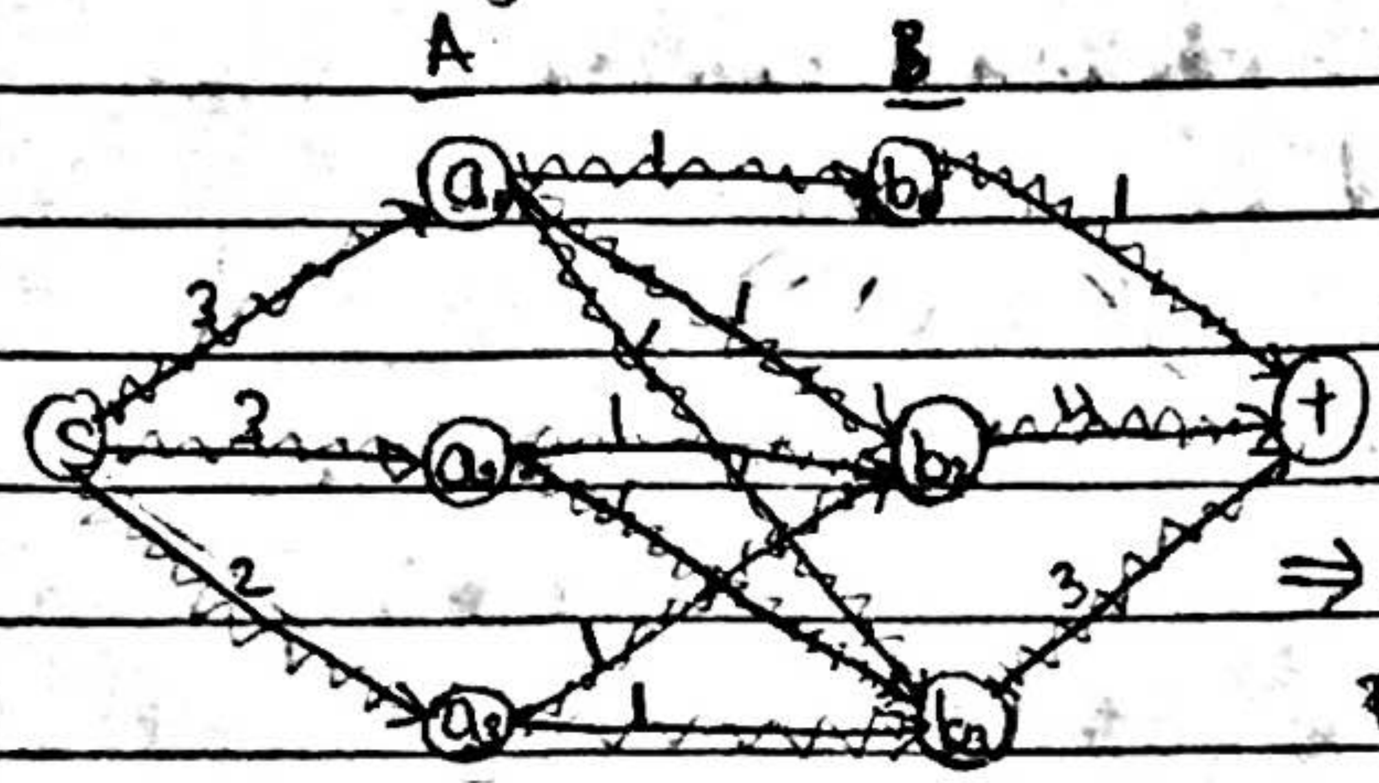
t=3 (colors!)

*note clique definition

*note disjoint union ($A \sqcup B$)

↳ union of 2 disjoint sets

can we color these vertices such that 1st color used b_1 times, 2nd color used b_2 times, ... (no 2 adj vertices are colored with the same color)



maxFlow = 7

7<8

⇒ coloring not possible

let 3 colors = blue, red, yellow
(1 time) (4 times) (3 times)

→ and each color can only be used ONCE per clique (because all vertices in each clique are adjacent)

- start by connecting (s) to nodes representing group A (a_1, a_2, a_3) and giving them cap of the value of each int represented by a_k (eg edge $S \to a_1$ has cap 3 because $a_1 = 3$)
- then connect edges representing group B (b_1, b_2, b_3) to t and giving them cap of the value of each in represented by b_t
- now it's time to connect A vertices to B vertices ← Set all edges between A's and B's to 1!
start with the vertex with the biggest cap in A and draw as many edges as its capacity will allow (eg a_1 can have max 3 edges to B vertices because its cap $s \to a_1$ is 3)... an edge can be drawn to a vertex in B iff the degree-in of edges to b_t does NOT exceed cap $b_t \to t$ (eg a_2 cannot have an edge connecting it to b_1 because b_1 already has edge $a_1 \to b_1$ and cap $b_1 \to t = 1$)
- now solve for max flow... if max flow is < n ⇒ such a coloring is not possible (however, if max-flow = n ⇒ coloring is possible!)

(6.) Input: coords of n atennas, coord of i-th antenna is of the form (x_i, y_i) where x_i and y_i are integers.

Output: For antenna i, select a backup set B_i of five other antennas such that the antennas in B_i are all within distances of at most 100 from the antenna i, and in total, no antenna belongs to more than 10 backup sets.

Algorithm: 1. Create a bipartite graph for all antennas in this plane, group X vertices will be called $X_1, X_2, X_3, \ldots, X_n$ for every antenna on the plane. Group Y vertices will be called $Y_1, Y_2, Y_3, \ldots, Y_n$ also for every antenna on the plane (so each antenna on the plane will be represented by 2 vertices in the bipartite graph, one in X and one in Y).

2. Draw an edge from X_i to Y_j if Y_j is within 100 distance from X_i. However, do NOT draw an edge from X_i to Y_i (an edge from vertex in X to the same numbered vertex in Y). The weight of each of these edges will equal 1.

3. Make the bipartite graph into a flow network by adding a source and a sink.

4. Connect the source to (meaning arrow is leaving the source) each vertex in X with an edge. Set each of these edges equal to 5.

5. Connect each edge in Y to the sink with an edge. Set each of these edges equal to 10.

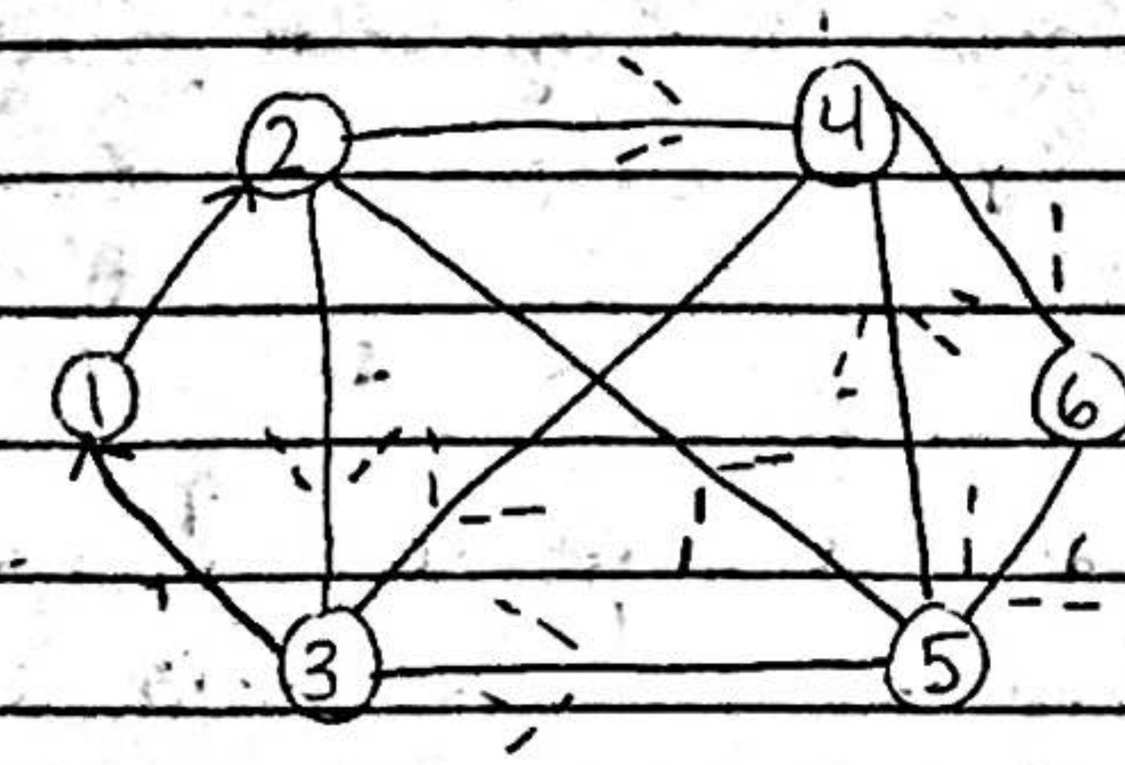
6. Solve for the max-matching with FF (as previously discussed).

7. For each antenna i, B_i will include the five antennas (each) indicated by its respective vertex in Y that has flow between X_i and Y_j.

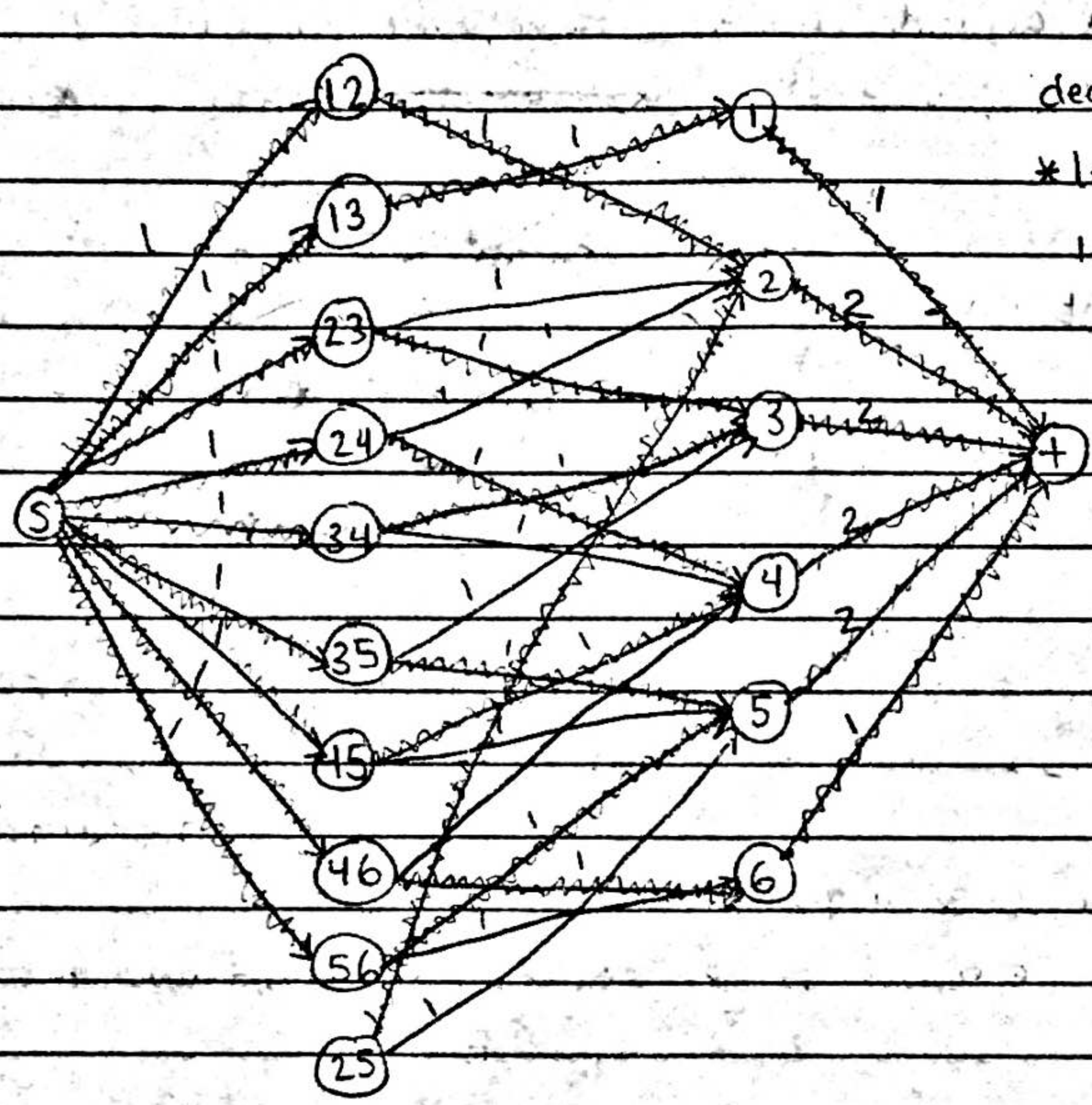
*Conclusion: Such a matching is possible if the max-matching (or max-flow) equals 5 * the # of antennas

This algorithm works because it finds the maximized/most efficient way to select B_i for antenna i with adhering to the constraints that B_i must contain five other antennas AND no antenna can belong to more than 10 backup sets.

7. graph w both directed and undirected edges, assign direction to undirected edges such that the resulting directed has the $f^{(in)} = f^{(out)}$ for $\forall e \in E$



← need to find a way of determining the dotted arrows.



degree of v / 2 ← integer division (roundup!)

*let the cap of the edge from 1, 2, 3, ... to t = $\lceil \text{degree of vertex}/2 \rceil$

THEN, solve for max flow... the paths you get when solving for max flow will indicate which direction each edge should go in the original graph (eg node 35 has a path to 5 in the bipartite representation ⇒ e_{35} has an arrow pointing to 5 in the original graph).

(8) Show that for every network flow, there is always a sequence of augmenting paths that leads to max flow, where none of these paths decrease the flow on any of the edges.

- Find a path in the flow network... however do NOT use backward edges
- Let the flow sent through the path equal the bottleneck of that path (the capacity of the edge with the smallest weight)... as the flow is sent through the path, decrease the flow of each edge by this bottleneck
- Repeat the 1st two steps until there ∄ st-path in the flow network

*Let the max-flow equal the sum of the flow of each path.