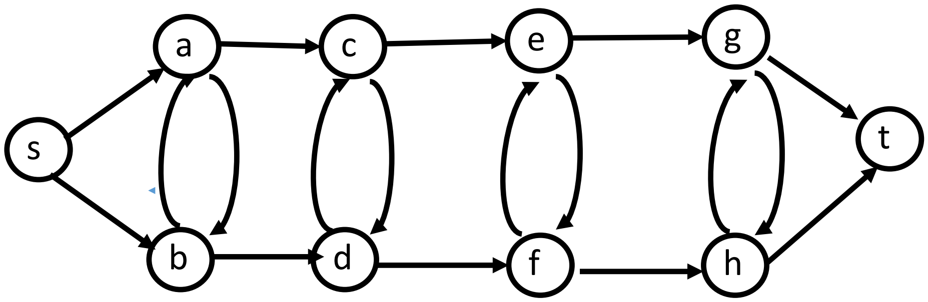
**(1) Finding minimum cut closest to** $t$ Consider a maximum flow problem with directed graph $G = (V, E)$ with $m$ edges and $n \leq m$ nodes, a source $s \in V$, a sink $t \in V$ and integer capacities $c_e \geq 0$ on each edge. We note that some graphs have many minimum capacity $(s,t)$-cuts. For example, in the graph below, if all edge capacities are $c_e = 1$, this graph has lots of minimum capacity cuts: any of $\{s\}, \{s, a, b\}, \{s, a, b, c, d\}$, etc. are all $s$-sides of minimum capacity cuts (cuts of capacity 2).




Assume that you are given a maximum value flow $f$ that was already computed. Give an $O(m)$ time algorithm that finds a minimum capacity $(s,t)$-cut, $(A, B)$ with $s \in A$ and $t \in B$, with as many nodes in $A$ as possible. In the example above, this would be the cut $A = \{s, a, b, c, d, e, f, g, h\}$ and $B = \{t\}$.

**(2) Scheduling Interviews**

You are scheduling initial phone interviews for $n$ job candidates that have applied to different jobs at the same company (each candidate has applied to just one job). The company has $k \leq n$ recruiters, and each recruiter is qualified to interview candidates only for some of the jobs. Each candidate needs to be assigned to a recruiter to be interviewed.

Candidates will be labeled $0, 1, \ldots, n-1$, and recruiters will be labeled $n, n+1, \ldots, n+k-1$. We will let $\mathsf{neighbors}$ hold the adjacency lists of the bipartite graph representing compatible candidate-recruiter pairs. That is, for each candidate $i$, $\mathsf{neighbors}[i]$ holds the recruiters that they can be interviewed by. For each recruiter $j$, $\mathsf{neighbors}[j]$ holds the candidates that recruiter $j$ can interview. You also want to make sure that recruiters are not overloaded: so, we have an array $\mathsf{recruiter\_capacities}$ where $\mathsf{recruiter\_capacities}[j]$ is the maximum number of interviews that recruiter $j$ can do (note that because of our indexing scheme, entries 0 through $n-1$ of this array are empty).

Someone has already tried to find an assignment of candidates to interviewers, but they are having trouble. They have an array $\mathsf{preliminary\_assignment}$ that assigns each candidate $i$ to recruiter $\mathsf{preliminary\_assignment}[i]$ (without overloading any recruiter). Unfortunately, $\mathsf{preliminary\_assignment}[n-1]$ is blank, and they are having trouble filling this last entry.

You must code an efficient algorithm that has the following behavior:

- If there exists a valid assignment that assigns all job candidates to recruiters, output it. Note that **there may be more than one such valid assignment**. We will accept any, as long as it is assigns all job candidates to recruiters in a valid way (i.e., no recruiter is overbooked).
- If no such assignment exists, you plan to ask one of the recruiters $j$ to increase their capacity $\mathsf{recruiter\_capacities}[j]$. Output the list of recruiters $j$ such that if their capacity is increased by 1 (while the other capacities remain the same), then a solution will exist.

Your algorithm must run in $O(m)$ time, where $m = \sum_{i=0}^{n-1} |\mathsf{neighbors}[i]|$ is the number of edges in the graph. Solutions that take longer (e.g., $O(nk)$) will only get partial credit.

We illustrate the problem with the following **example**:

- Suppose we have $n = 3$ candidates, $k = 2$ recruiters, candidate neighbors $\mathsf{neighbors}[0] = (3, 4)$, $\mathsf{neighbors}[1] = (3)$, and $\mathsf{neighbors}[2] = (3)$ (the rest of the array can be inferred from these entries), and recruiter capacities $\mathsf{recruiter\_capacities}[3] = 2, \mathsf{recruiter\_capacities}[4] = 1$. If we are given $\mathsf{preliminary\_assignment} = (3, 3, \cdot)$, then a fully satisfying assignment does exist: it is $\mathsf{valid\_assignment} = (4, 3, 3)$.
- For the (otherwise) same input, if $\mathsf{recruiter\_capacities}[3] = 1, \mathsf{recruiter\_capacities}[4] = 2$, and $\mathsf{preliminary\_assignment} = (4, 3, \cdot)$, then no fully satisfying assignment exists, and the only way to create one is to increase $\mathsf{recruiter\_capacities}[3]$.