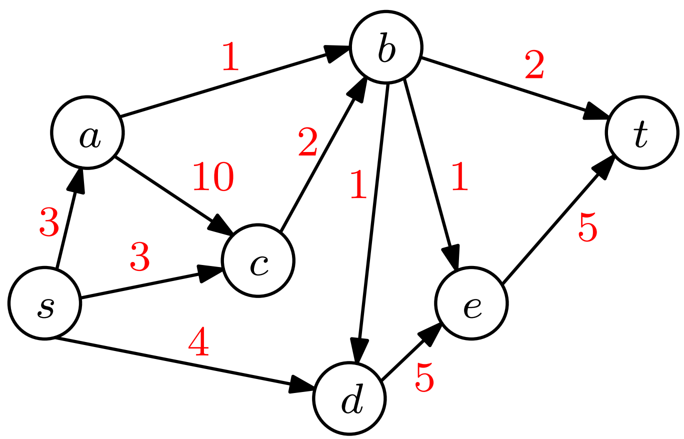
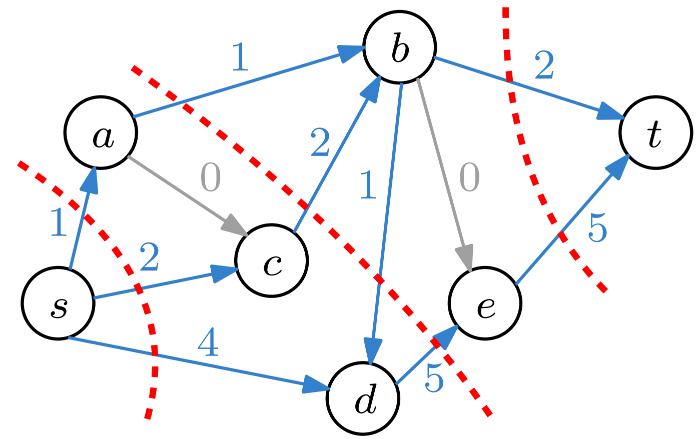
Algorithms

Network Flow

Hee-Kap Ahn
Graduate School of Artificial Intelligence
Dept. Computer Science and Engineering
Pohang University of Science and Technology (POSTECH)

Textbooks: Algorithms by Jeff Erickson /Algorithms by Sanjoy Dasgupta, Christos Papadimitriou, Umesh Vazirani.





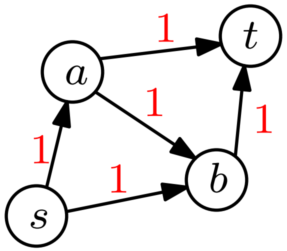
We are given
- a directed graph $G = (V, E)$;
- two special nodes $s, t \in V$, a source and a sink of G, respectively; and
- capacities $c_e > 0$.

We want to **send as much flow as possible** from s to t such that
- $0 \leq f_e \leq c_e$ for all $e \in E$.
- flow is conserved : $\sum_{(w,u)\in E} f_{wu} = \sum_{(u,z)\in E} f_{uz}$.
- size$(f) = \sum_{(s,u)\in E} f_{su} = \sum_{(v,t)\in E} f_{vt} =$ size(f).

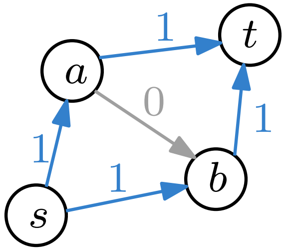
The maximum flow problem reduces to linear programming!

Residual Networks

The general simplex algorithm would
- start with zero flow
- repeat:
  choose an appropriate path from s to t, and increase flow along the edges of the path as much as possible.



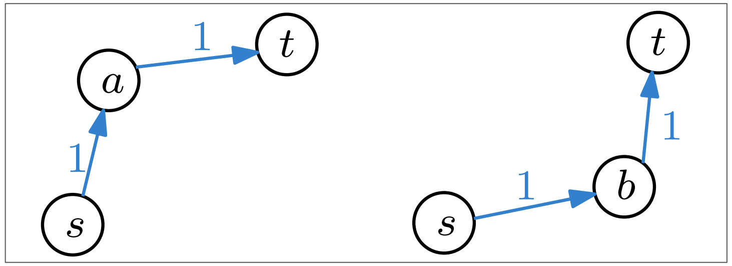
Given network

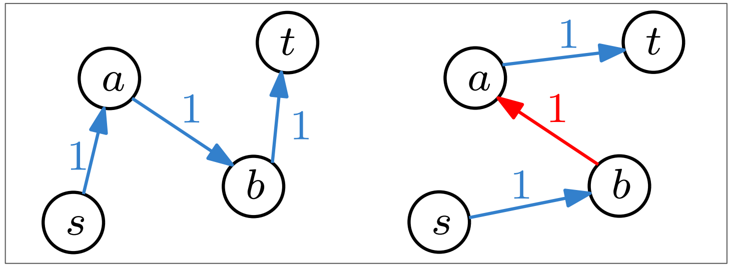


Final flow

Each iteration of simplex looks for an $s-t$ path whose edges (v, w) can be of two types:
- $(v, w) \in E$ and $f_{vw} < c_{vw}$.
- $(w, v) \in E$ and $f_{wv} > 0$.

$(a, b) \in E$ and $f_{ab} = 1$. Path $sbat$ is chosen.

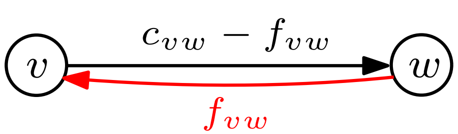




Input graph $G = (V, E)$ vs. Residual graph $G^f = (V, E^f)$.

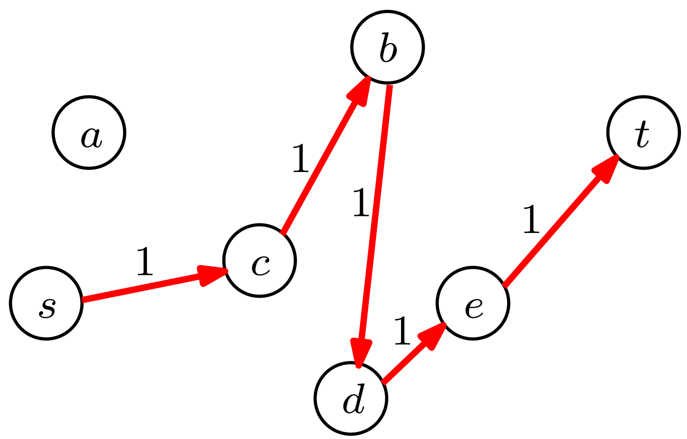
For an edge $(v, w) \in E$ with $f_{vw} \leq c_{vw}$, $G^f = (V, E^f)$ has two edges (v, w), (w, v) with residual capacities

$$\begin{aligned} c^f_{vw} &= c_{vw} - f_{vw}, \\ c^f_{wv} &= f_{vw}. \end{aligned}$$

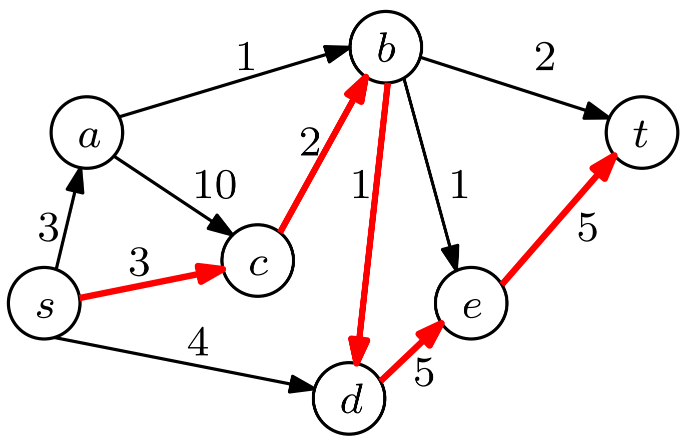


Simplex Algorithm in Residual Networks

Current flow

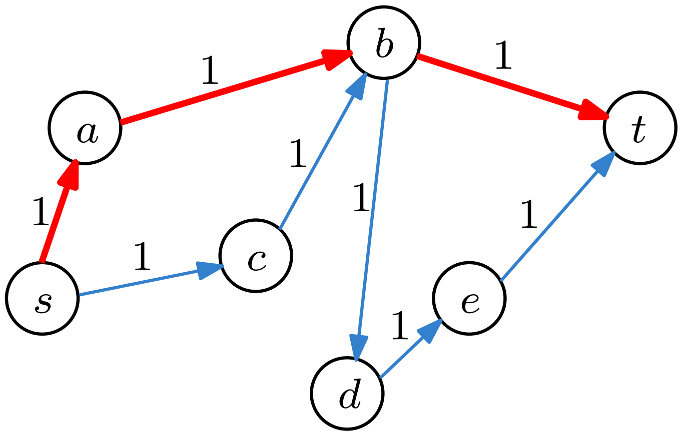


Residual graph

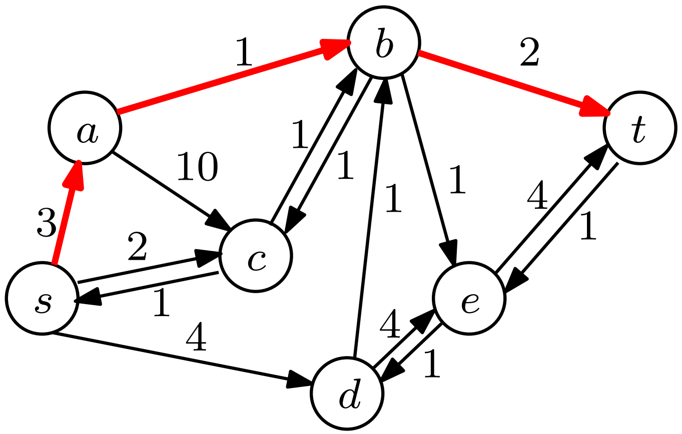


A $s-t$ path $scbdet$ of flow 1.

Current flow



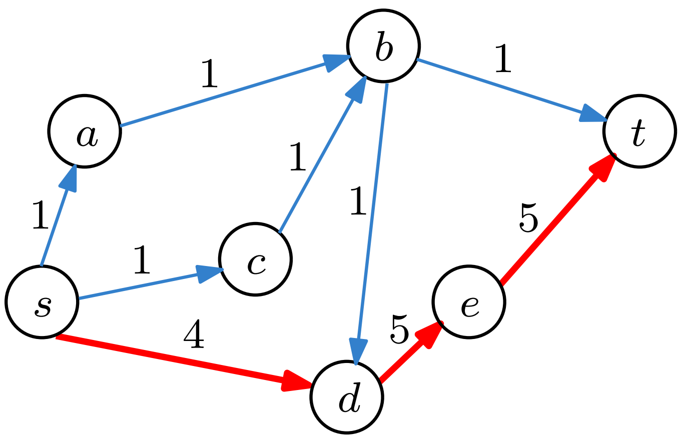
Residual graph



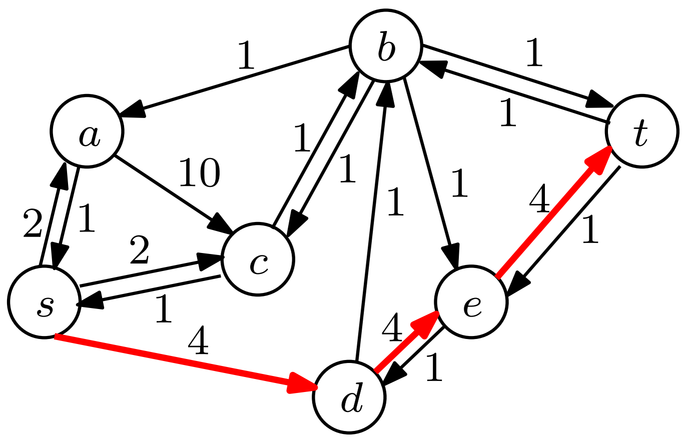
A $s - t$ path $sabt$ of flow 1.

Simplex Algorithm in Residual Networks

Current flow

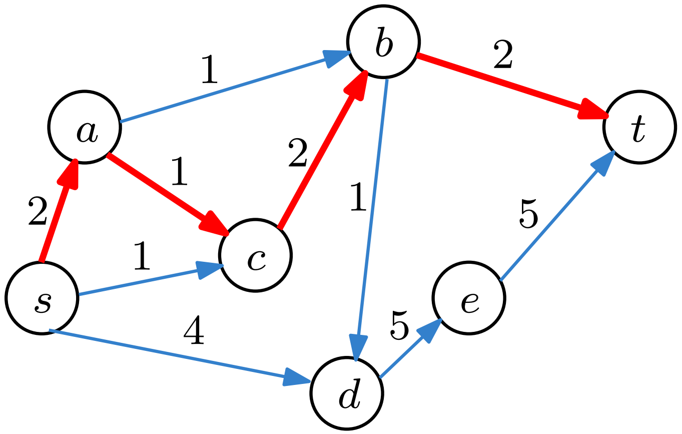


Residual graph

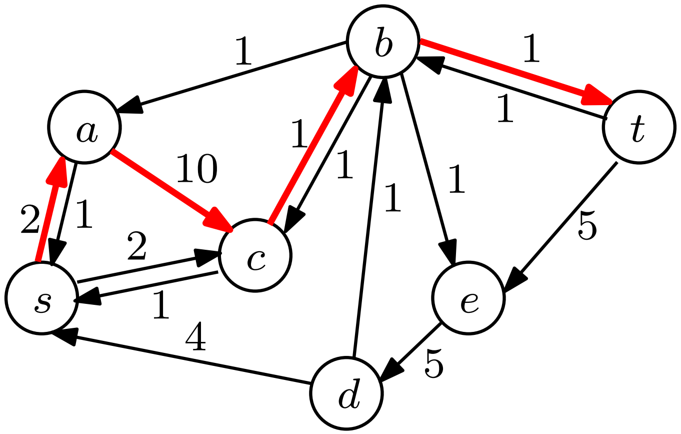


A $s-t$ path $sdet$ of flow 4.

Current flow

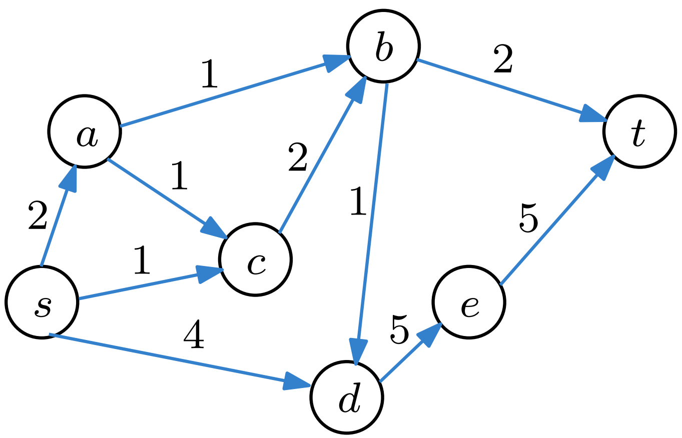


Residual graph

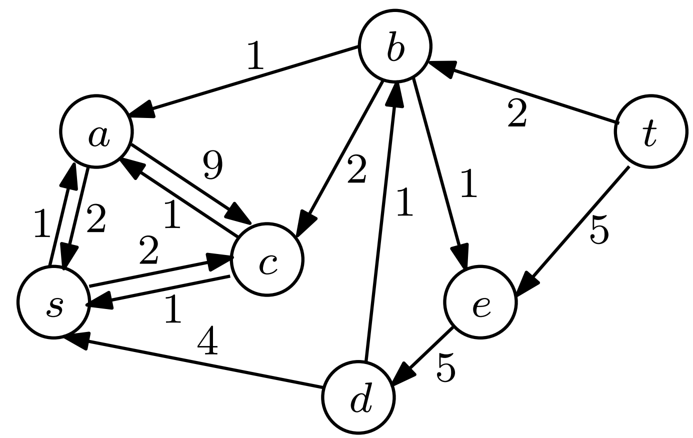


A $s-t$ path $sacbt$ of flow 1.

Current flow

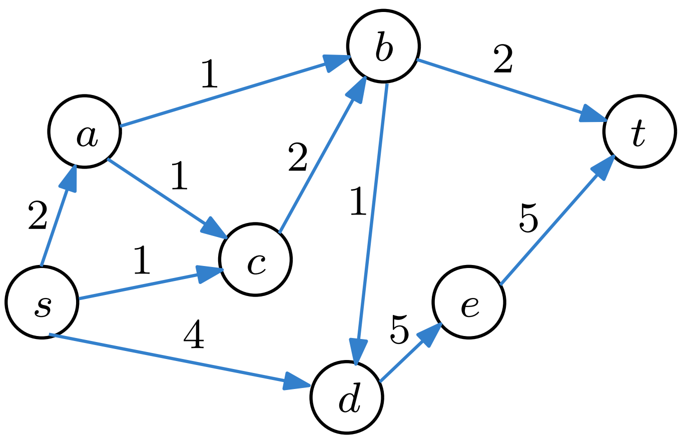


Residual graph

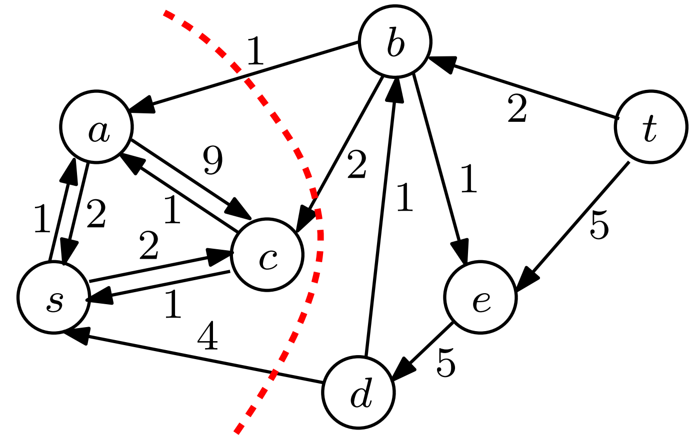


The final flow and the Residual graph.

Current flow



Residual graph

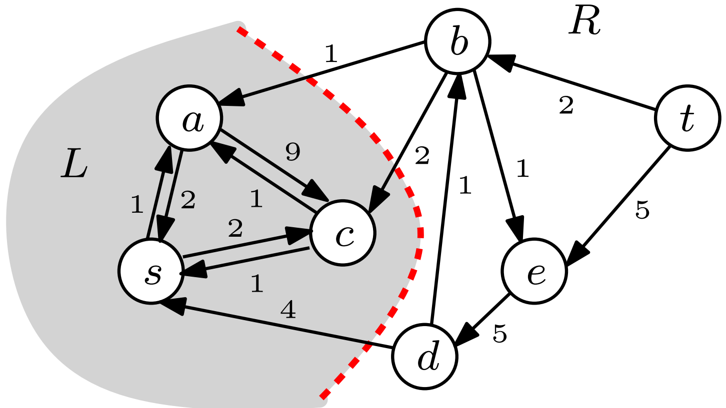


The final flow and the Residual graph.

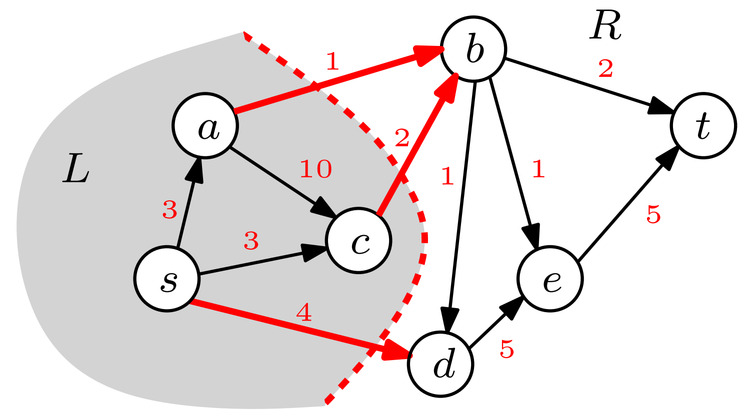
Flow and (s,t)-cut

A certificate of optimality Partition the nodes of the network into two groups. In the following residual graph, let $L = \{s, a, c\}$ and $R = \{b, d, e, t\}$, for example.

Residual graph G^f



(L, R) of G



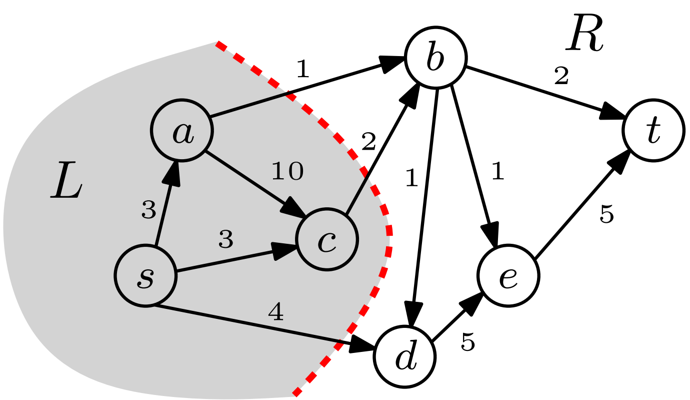
An (s,t)-cut partitions the vertices into two disjoint groups L and R s.t. $s \in L$ and $t \in R$. Its capacity is the total capacity of edges $\in G$ **from** L **to** R.

Pick any flow f and any (s,t)-cut.
Then size$(f) \leq$ capacity(L, R).

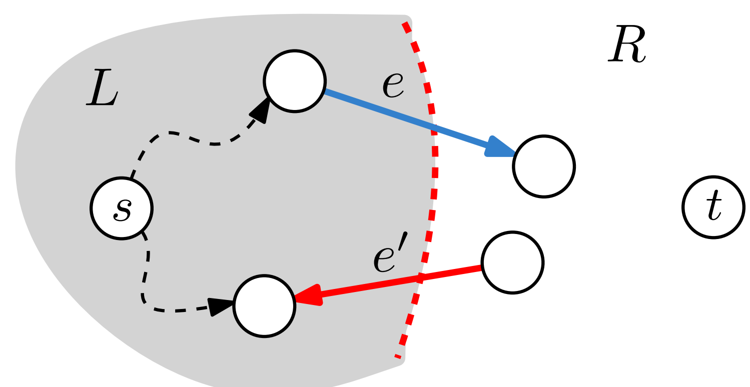
Max-flow Min-cut Theorem

> The size of the maximum flow in a network equals
> the capacity of the smallest (s,t)-cut.

Suppose f is the final flow when the algorithm terminates. Then t is no longer reachable from s in G^f.
Let L be the set of nodes reachable from s in G^f, and $R = V - L$.
We claim that size(f) = capacity(L, R).



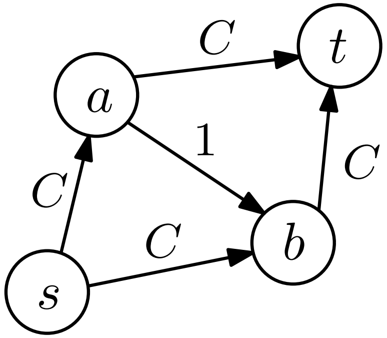
(L, R) of G



Consider any edge $e \in E$ from a node in L to a node in R. Then $f_e = c_e$. Consider now any edge $e' \in E$ from a node in R to a node in L. Then $f_{e'} = 0$. Do you see why?

Therefore, (L, R) is the smallest (s,t)-cut!

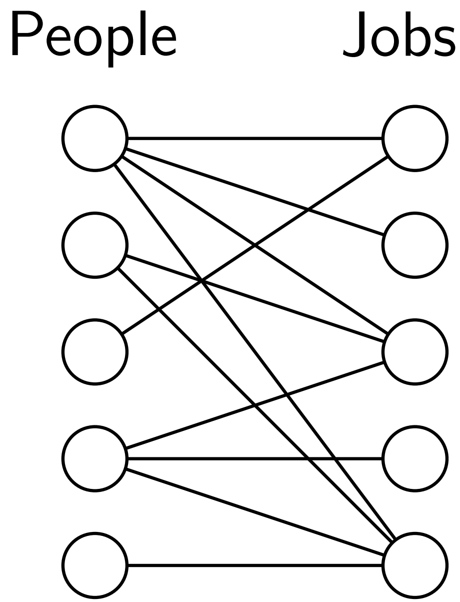
Efficiency Each iteration of our maximum-flow algorithm is efficient, requiring $O(|E|)$ time, if a DFS or BFS is used to find an $s-t$ path.
But how many iterations are there?



Is there a way to avoid $C|E|$ iterations, when C is a huge number?

Bipartite Matching

We have a set of people P and a set of jobs J.
Each person can do some of the jobs.
We can model this as a bipartite graph.



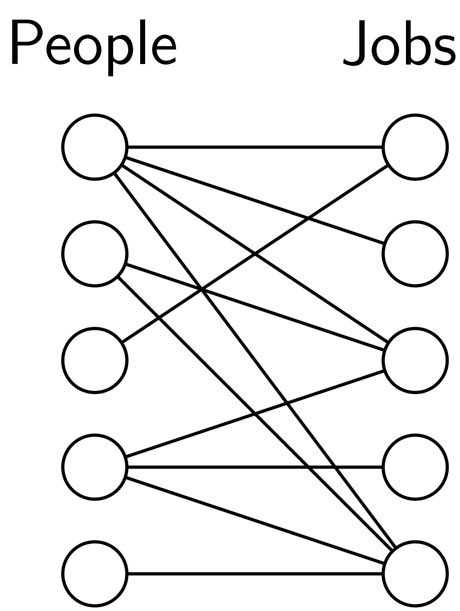
A **matching** gives an assignment of people to jobs such that

- a person is assigned to at most one job and
- at most one person is assigned to a job.

Bipartite Matching

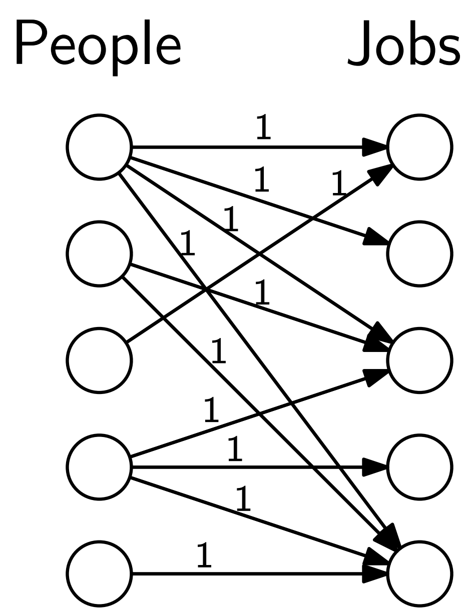
Given an instance of bipartite matching, we can find a maximum matching by the following reduction.

- Replace each edge to a directed edge from a person to a job of capacity 1.
- Add a special node s and connect s to every node in People by directed edge of capacity 1.
- Add a special node t and connect every node in Jobs to t by directed edge of capacity 1.



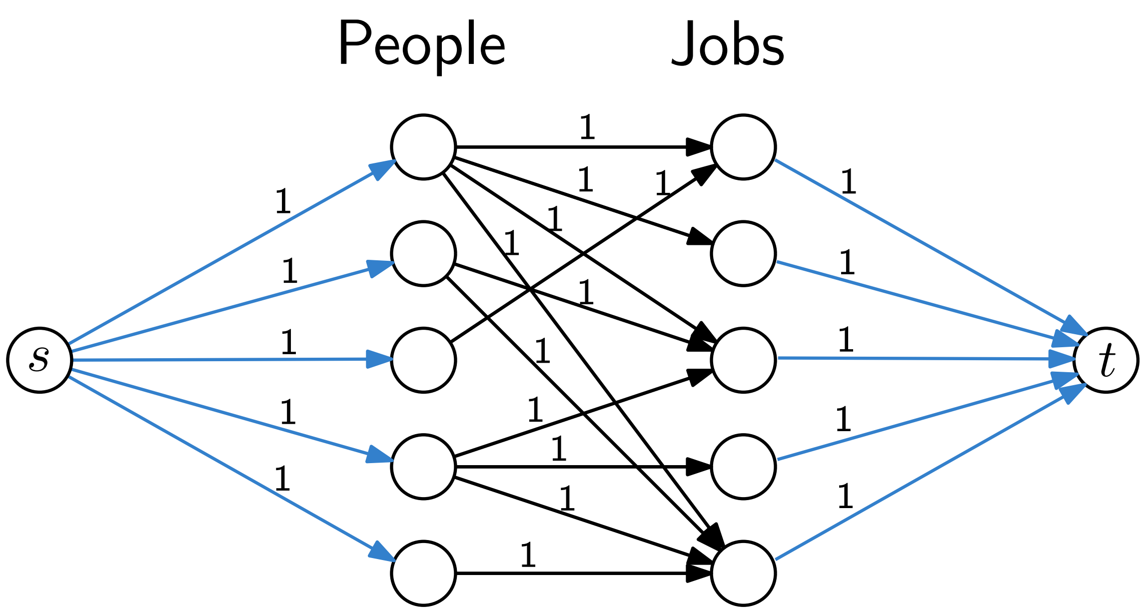
Given an instance of bipartite matching, we can find a maximum matching by the following reduction.

- Replace each edge to a directed edge from a person to a job of capacity 1.
- Add a special node s and connect s to every node in People by directed edge of capacity 1.
- Add a special node t and connect every node in Jobs to t by directed edge of capacity 1.



Given an instance of bipartite matching, we can find a maximum matching by the following reduction.

- Replace each edge to a directed edge from a person to a job of capacity 1.
- Add a special node s and connect s to every node in People by directed edge of capacity 1.
- Add a special node t and connect every node in Jobs to t by directed edge of capacity 1.



The edges used in the maximum flow correspond to the maximum matching.
- all capacities are integers, so the maximum flow is integral.
- all capacities are 1, so an edge is used (flow 1) or not.

Let M be the set of edges used in the maximum flow of value k. Then M is a matching because there is at most one edge in M leaving any person and entering any job. Moreover, M consists of k edges.

If there is a matching consisting of more than k edges, there must be a flow of value larger than k, which is a contradiction.

For n people and m edges, the running time is?