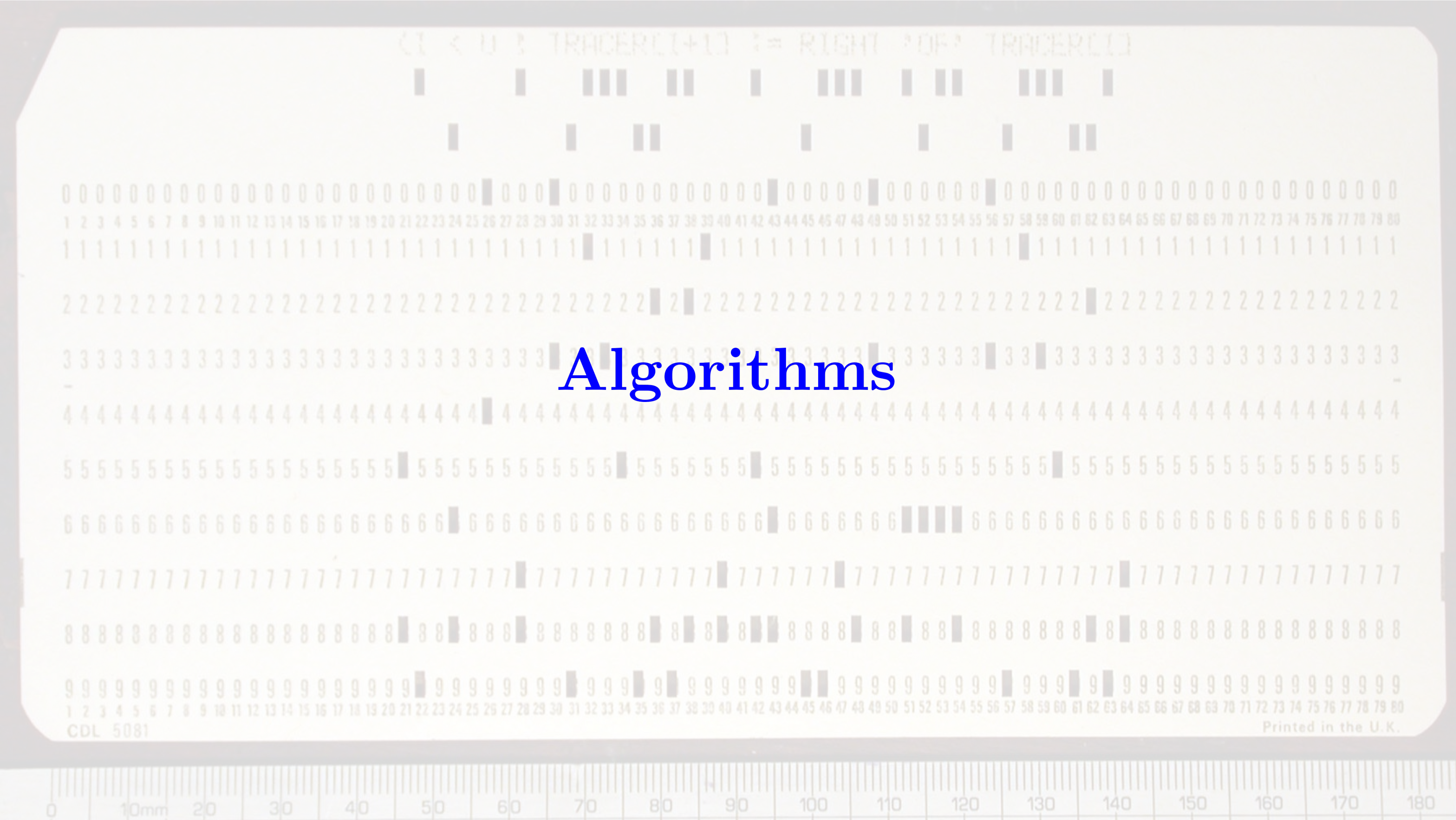
# Algorithms



# Reading

Network Flow: Sections 7.1-7.3
Network Flow Applications: Sections 7.5, 7.7
Randomized Algorithm: Section 12.2

**Suggested.** As many exercises as you can do from Chapter 7 of the textbook.

# Recitations

Thu, Apr 27 2023- Fida (OH by appt)
2:30 PM - 3:45 PM
Location Instructions
West Admin (A3-002)

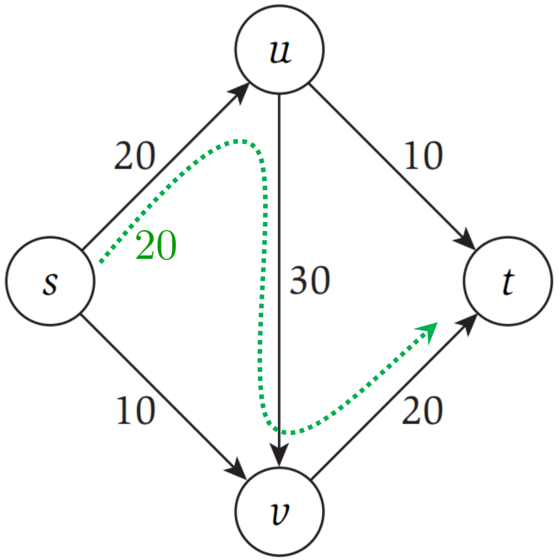
Tue, May 9 2023- Shan
2:30 PM - 3:45 PM
Location Instructions
West Admin (A3-002)

Thu, May 11 2023- Fida (send questions before May 8)
Class time
Class location

# Network Flows

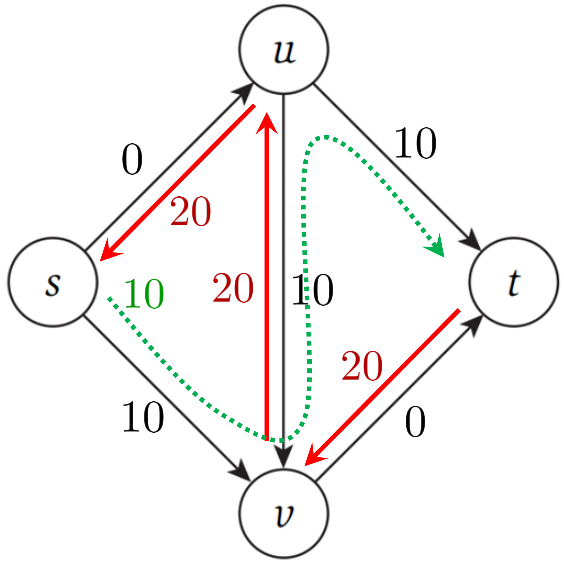
**Ford Fulkerson Algorithm.**

Repeatedly find a path in $G_f$ from $s$ to $t$ in which all edges have some leftover capacity and send flow equal to the capacity of the *bottleneck edge*.




*send 20 units of flow*
*along* $s \to u \to v \to t$

Residual Graph

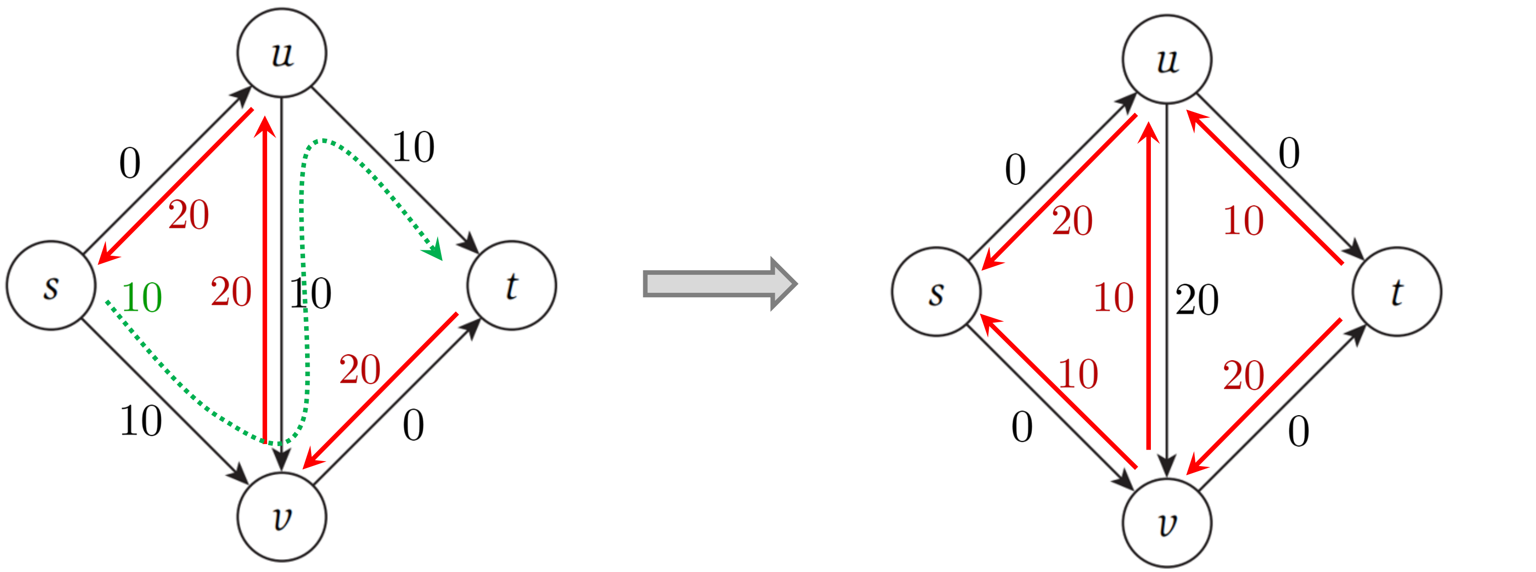



*send 10 units of flow*
*along* $s \to v \to u \to t$

# Network Flows

**Ford Fulkerson Algorithm.**

Repeatedly find a path in $G_f$ from $s$ to $t$ in which all edges have some leftover capacity and send flow equal to the capacity of the *bottleneck edge*.



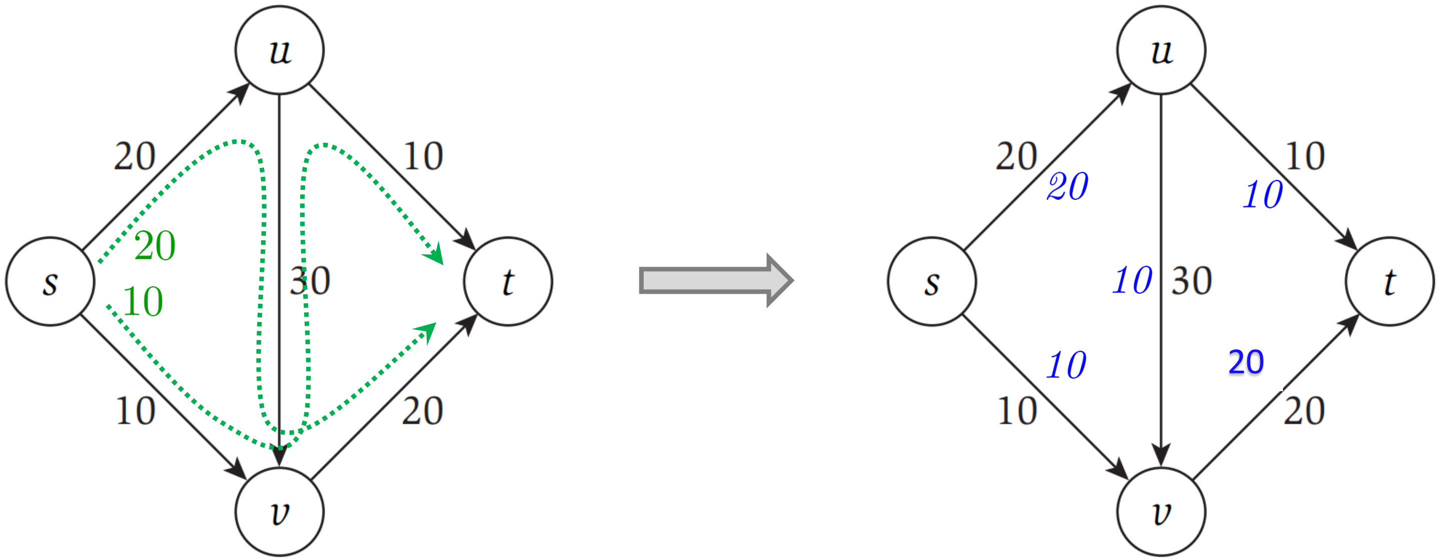

*send 10 units of flow along $s \to v \to u \to t$*

*No more $s \rightsquigarrow t$ paths of non-zero capacity!*

Total flow: 30 units.

**Ford Fulkerson Algorithm.**

Repeatedly find a path in $G_f$ from $s$ to $t$ in which all edges have some leftover capacity and send flow equal to the capacity of the *bottleneck edge*.




The paths along which we sent flows are called *augmenting paths*.

# Network Flows

```
Max-Flow
    Initially f(e) = 0 for all e in G
    While there is an s-t path in the residual graph G_f
        Let P be a simple s-t path in G_f
        f' = augment(f, P)
        Update f to f'
        Update the residual grapgh G_f to be G_f'
    EndWhile
    Return f
```

Flow value is 0

If a path is found, augment is called

Residual graph is updated

```
augment(f, P):
    Let b = bottleneck(P, f)
    For each edge (u,v) P
        If e = (u, v) is a forward edge then
            increase f(e) in G by b
        Else ((u, v) is a backward edge, and let e = (~~u,v~~ (v,u)))
            decrease f(e) in G by b
        Endif
    Endfor
    Return(f)
```

**Here we are calculating the flow being passed on each edge. So for backward edges u→v, we need to decrease the flow of v→u by b.

Calculates bottleneck

Sends flow equal to bottleneck on all path edges

Forward edges increase flow, backward edges decrease the flow.

Say a flow with value $f$ is produced

What is an upper bound for the number paths found?

# Network Flows

**Claim.** If all edge capacities are (non-negative) integers, then the Ford Fulkerson algorithm yields an optimal solution and runs in $O((m+n)\cdot f)$ time, where $f$ is the value of the flow returned.

*Proof.* In every iteration, the algorithm sends at least one additional unit of flow.

The algorithm therefore has $\leq f$ iterations.

In any iteration, we can find an augmenting path in $O(m+n)$ time using BFS/DFS on the residual graph.

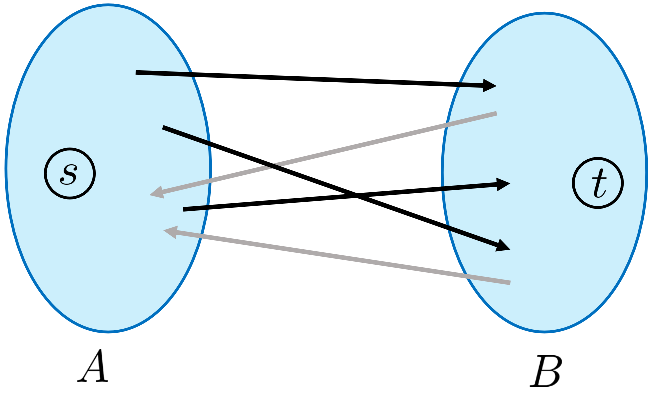
This implies that the running time is $O((m+n)\cdot f)$.

Why optimal?

# Two observations

# Network Flows

An **s-t cut** in the graph is a partition $(A, B)$ of the vertices into two sets $A$ and $B$ s.t. $s \in A$ and $t \in B$.




The capacity of the $s$-$t$ cut, denoted $c(A, B)$ is the sum of the capacities of the edges going *from $A$ to $B$.*

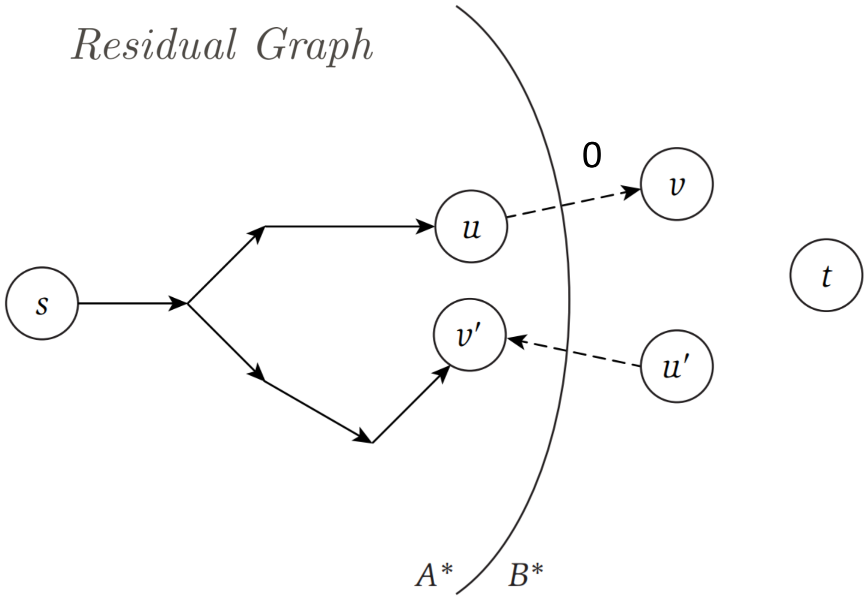
*Observation.* No flow can have value more than the capacity of an $s$-$t$ cut.

# Network Flows

Consider the residual graph corresponding to a maximum flow $f$ .

Let $A^*$ be the set of all nodes $a$ in $G$ s.t. there is an $s \rightsquigarrow a$ path in $G_f$
i.e. all edges have non-zero (residual) capacity.

Let $B^*$ be the set of the remaining vertices. $B^* = V$ - $A^*$




*Observation.* Value of $f$ = Capacity of the $s$-$t$ cut $(A^*, B^*)$.

# Network Flows

We have made the following observations.

*Observation.* No flow can have value more than the capacity of an $s$-$t$ cut.

*Observation.* Corresponding to a maximum flow $f$, there is an $s$-$t$ cut $(A^*, B^*)$ s.t. $\nu(f) = c(A^*, B^*)$. *value of $f$ = capacity of $(A^*, B^*)$*

These two together imply:

**Max flow = Min Cut**

The maximum value of a flow is equal to the minimum capacity of an $s$-$t$ cut.

# Network Flows

**Claim.** If all edge capacities are (non-negative) integers, then the Ford Fulkerson algorithm yields an optimal solution and runs in $O((m+n)\cdot f)$ time, where $f$ is the value of the flow returned.

*Proof.* In every iteration, the algorithm sends at least one additional unit of flow.

The algorithm therefore has $\leq f$ iterations.

In any iteration, we can find an augmenting path in $O(m+n)$ time using BFS/DFS on the residual graph.

This implies that the running time is $O((m+n)\cdot f)$.

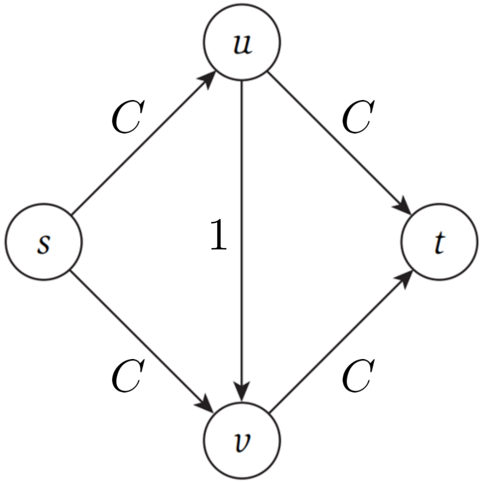
Why optimal?

The algorithm stops only when there is no augmenting path, implying that there is an $s$-$t$ cut whose capacity is equal to the value of the flow returned by the algorithm.

Thus, there cannot be a flow with larger value. ∎

# Network Flows

$O((m+n) \cdot f)$ is not a polynomial running time since $f$ can be large.

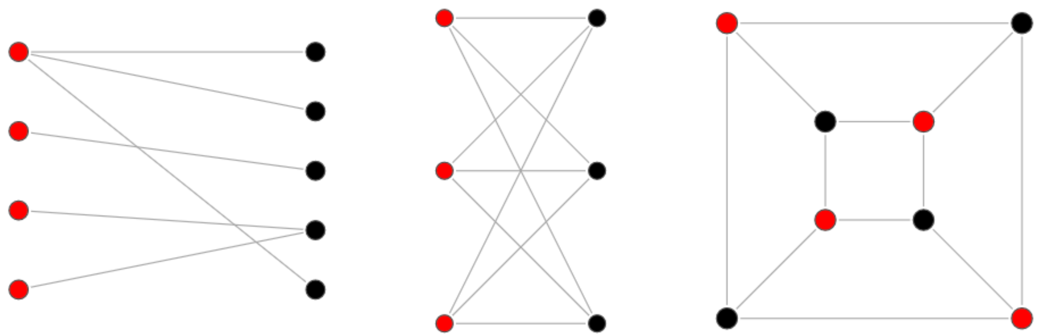



Exercise. The algorithm may take 2C steps (value of the maximum flow)
Describe how

# Bipartite Graphs

A bipartite graph, also called a bigraph, is a

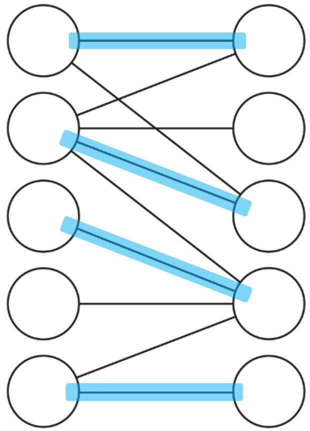
- set of graph vertices decomposed into two disjoint sets
- no two graph vertices within the same set are adjacent.



# Matchings in Bipartite Graphs

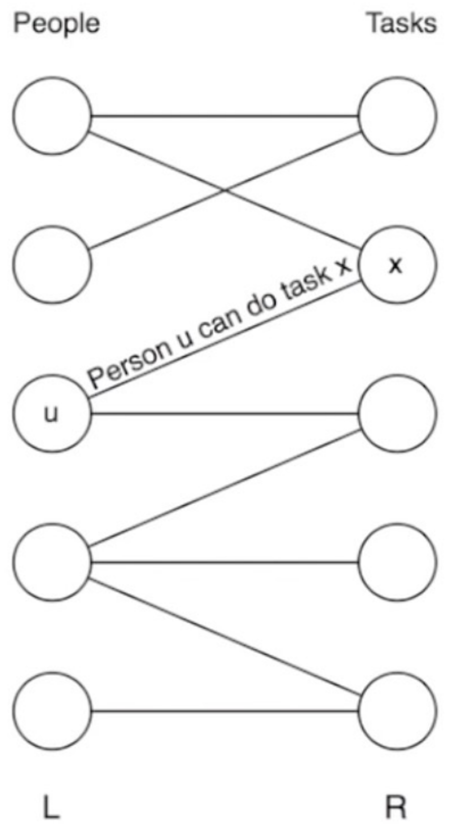
A matching in a graph is a subset of edges s.t. each vertex appears in at most one edge.

Find a matching



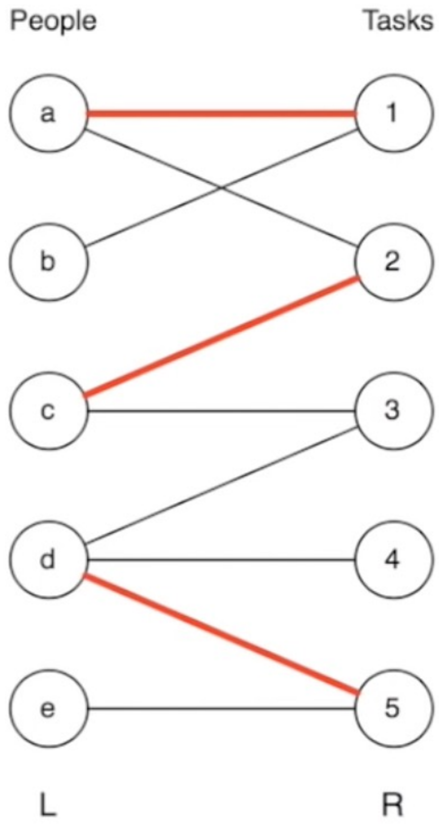
# Matchings in Bipartite Graphs

- Suppose we have a set of people $L$ and set of jobs $R$.
- Each person can do only some of the jobs.
- Can model this as a bipartite graph $\rightarrow$



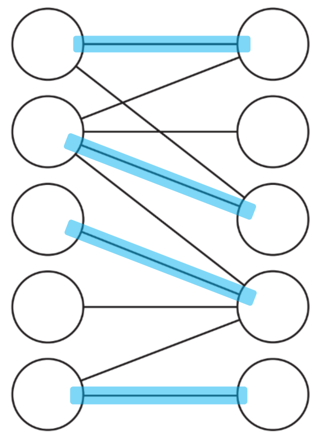
# Matchings in Bipartite Graphs

- A **matching** gives an assignment of people to tasks.
- Want to get as many tasks done as possible.
- So, want a **maximum matching**: one that contains as many edges as possible.
- (This one is not maximum.)



# Matchings in Bipartite Graphs

A matching in a graph is a subset of edges s.t. each vertex appears in at most one edge.



**Task.** Find a matching with the maximum number of edges in a bipartite graph.