
# Algorithms: COMP3121/9101

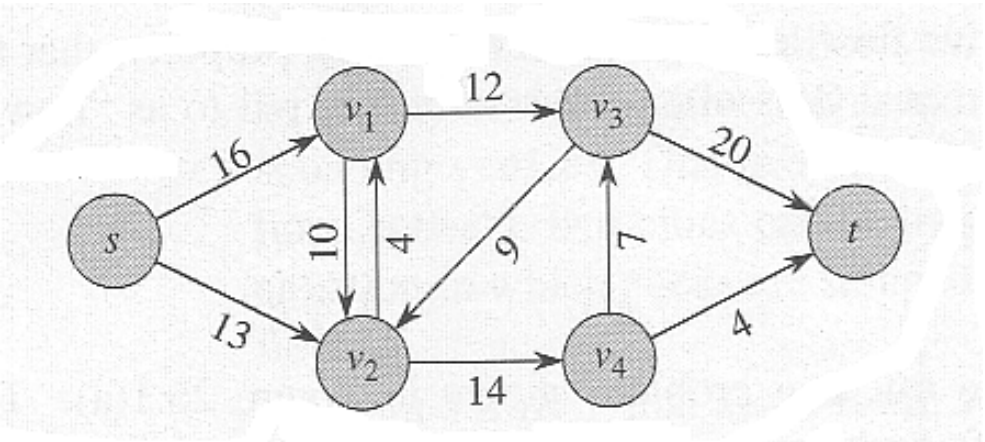
School of Computer Science and Engineering
University of New South Wales

8. MAXIMUM FLOW

# Flow Networks

- A **flow network** $G = (V, E)$ is a directed graph in which each edge $e = (u, v) \in E$ has a positive integer capacity $c(u, v) > 0$.

  There are two distinguished vertices: a **source** $s$ and a **sink** $t$; no edge leaves the sink and no edge enters the source.

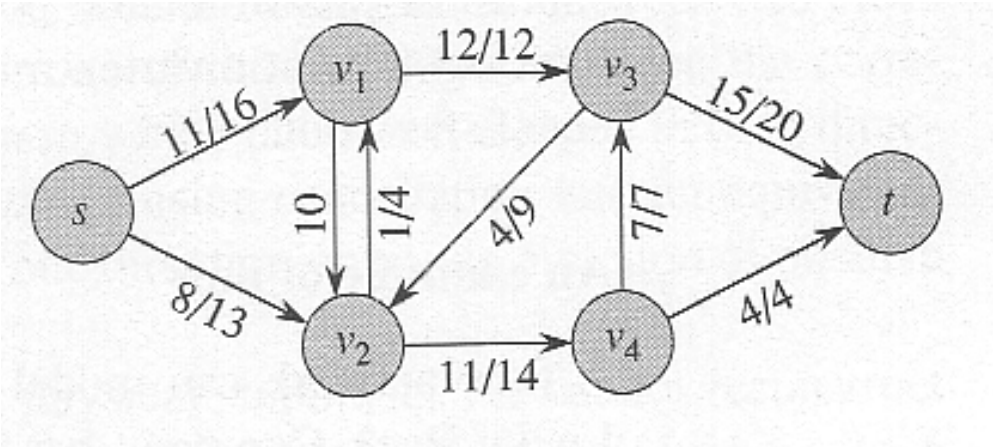



- A **flow** in $G$ is a function $f : E \to \mathbb{R}^+$, $f(u, v) \geq 0$, which satisfies
  1. **Capacity constraint:** for all edges $e(u, v) \in E$ we require $f(u, v) \leq c(u, v)$.
  2. **Flow conservation:** For all $v \in V - \{s, t\}$ we require

$$\sum_{(u,v)\in E} f(u, v) = \sum_{(v,w)\in E} f(v, w).$$

# Flow Networks

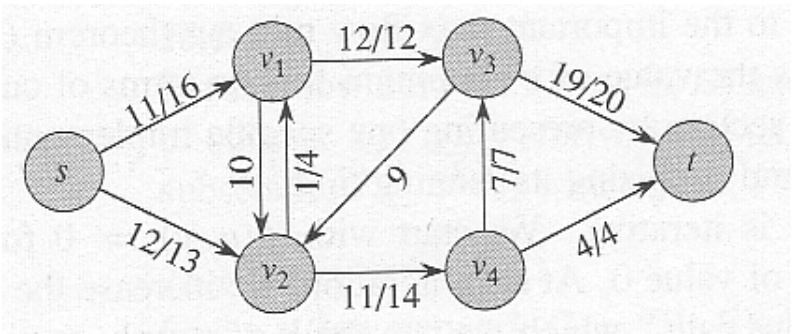
- The **value of the flow** is defined as $|f| = \sum_{v:(s,v)\in E} f(s,v)$.
- Clearly, also $|f| = \sum_{v:(v,t)\in E} f(v,t)$.
- Example of a flow network with some network flow in it:

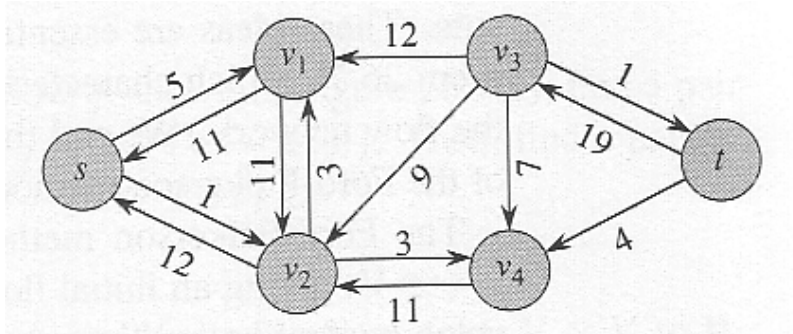



- The first number on an edge: the **flow** through that edge;
- The second number: the **capacity** of the edge.
- Examples of flow networks (possibly with several sources and many sinks): transportation networks, gas pipelines, computer networks...

# Flow Networks

- The **residual flow network** for a flow network with some flow in it: the network with the leftover capacities

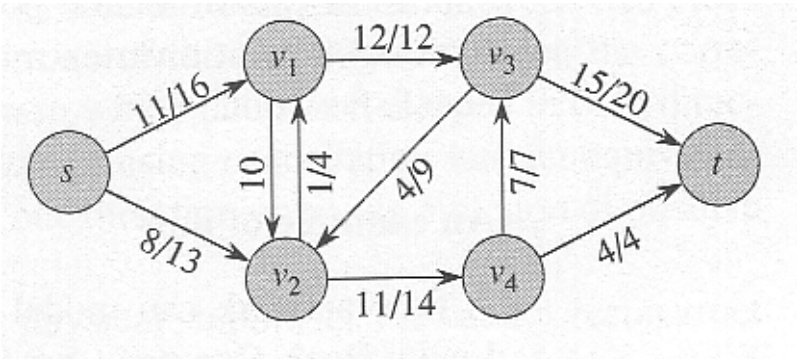


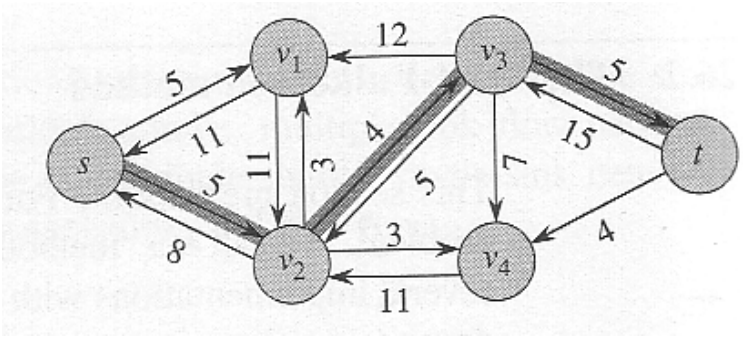




- Each edge of the original network has a leftover capacity for more flow which is equal to the capacity of the edge minus the flow through the edge.
- If the flow through an edge is equal to the capacity of the edge, this edge disappears in the residual network.
- New "virtual" edges appear in opposite direction of an original edge with some flow in it (unless there were already an edge in the opposite direction).
- They represent the possibility to **reduce/counteract** the flow through the original edge; thus their capacity is equal to the flow through the original edge (or, if there were already an edge in the opposite direction, the capacity of such an edge is increased for the amount of that flow; see vertices $v_1$ and $v_2$).

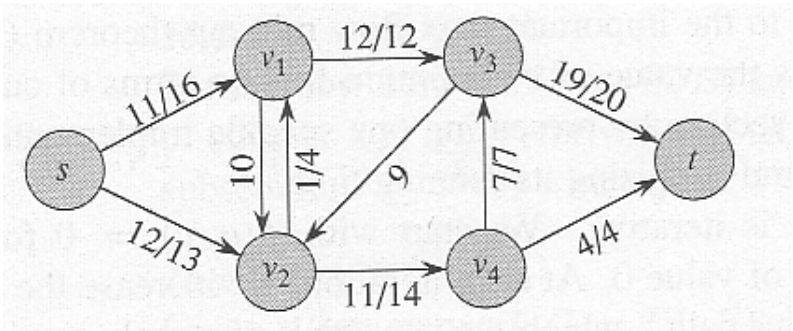
# Flow Networks

- Residual flow networks can be used to **increase** the total flow through the network by adding an **augmenting path**



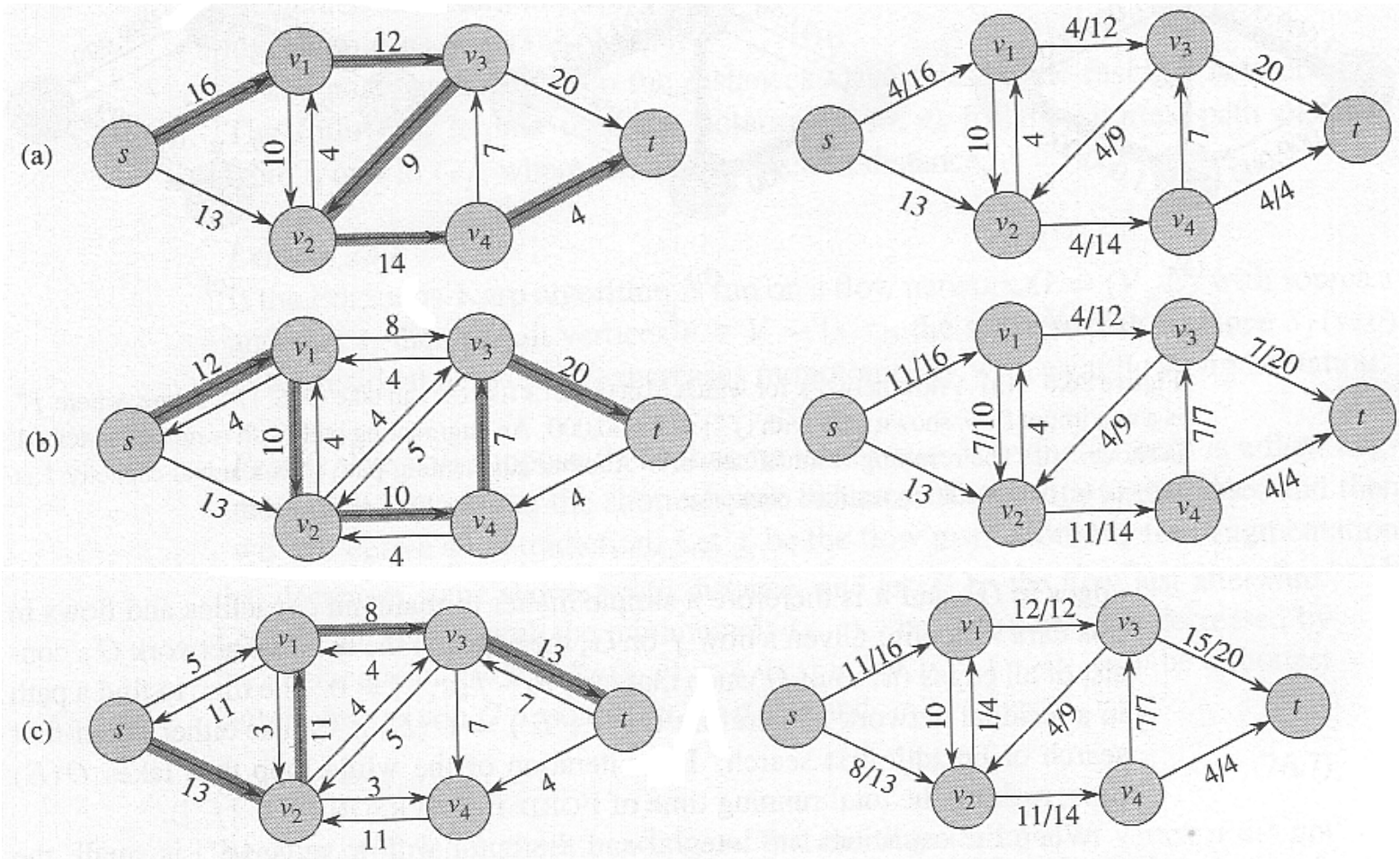

- The capacity of an augmenting path is the capacity of its "**bottleneck**" edge, i.e., the capacity of the smallest capacity edge on that path.
- We can now **recalculate** the flow through all edges along the augmenting path by adding the additional flow through the path if the flow through the augmenting path is in the same direction as the original flow, and subtracting if in opposite direction:



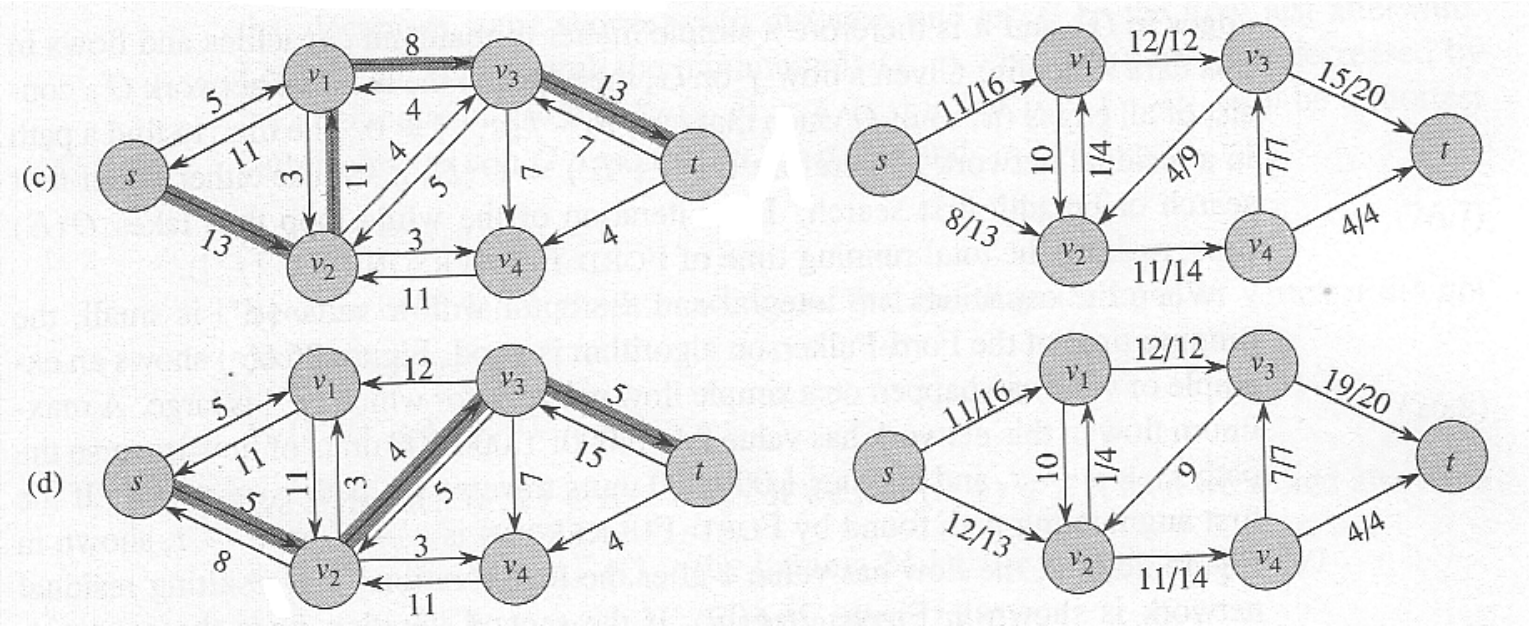
# Ford - Fulkerson method for finding maximal flow

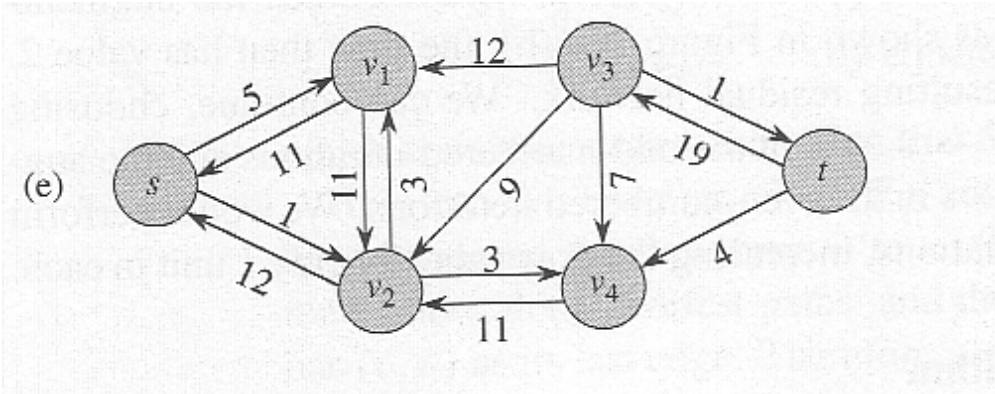
**Ford - Fulkerson algorithm for finding maximal flow in a flow network:**

- Keep adding flow through new augmenting paths for as long as it is possible.
- When there are no more augmenting paths, you have achieved the largest possible flow in the network.



# Ford - Fulkerson method for finding maximal flow



# Ford - Fulkerson method for finding maximal flow

**Ford - Fulkerson algorithm for finding maximal flow in a flow network:**

- Keep adding flow through new augmenting paths for as long as it is possible;
- When there are no more augmenting paths, you have achieved the largest possible flow in the network.

- Why does this procedure **terminate**? Why can't we get stuck in a loop, which keeps adding augmenting paths forever?
- If all the capacities are integers, then each augmenting path increases the flow through the network for at least 1 unit;
- the total flow cannot be larger than the sum of all capacities of all edges leaving the source, so eventually the process must terminate.

# Ford - Fulkerson method for finding maximal flow

- Even if the procedure does terminate, why does it produce a flow of the **largest** possible value?
- Maybe we have created bottlenecks by choosing bad augmenting paths; maybe better choices of augmenting paths could produce a larger total flow through the network?
- This is not at all obvious, and to show that this is not the case we need a mathematical proof!
- The proof is based on the notion of a minimal cut in a flow network:

- A **cut** in a flow network is any partition of the vertices of the underlying graph into two subsets $S$ and $T$ such that:
  1. $S \cup T = V$
  2. $S \cap T = \emptyset$
  3. $s \in S$ and $t \in T$.

# Cuts in flow networks

- The **capacity** $c(S,T)$ **of a cut** $(S,T)$ is the sum of capacities of all edges leaving $S$ and entering $T$, i.e.

$$c(S,T) = \sum_{(u,v)\in E} \{c(u,v) : u \in S \ \& \ v \in T\}$$

- Note that the capacities of edges going in the opposite direction, i.e., from $T$ to $S$ do not count.

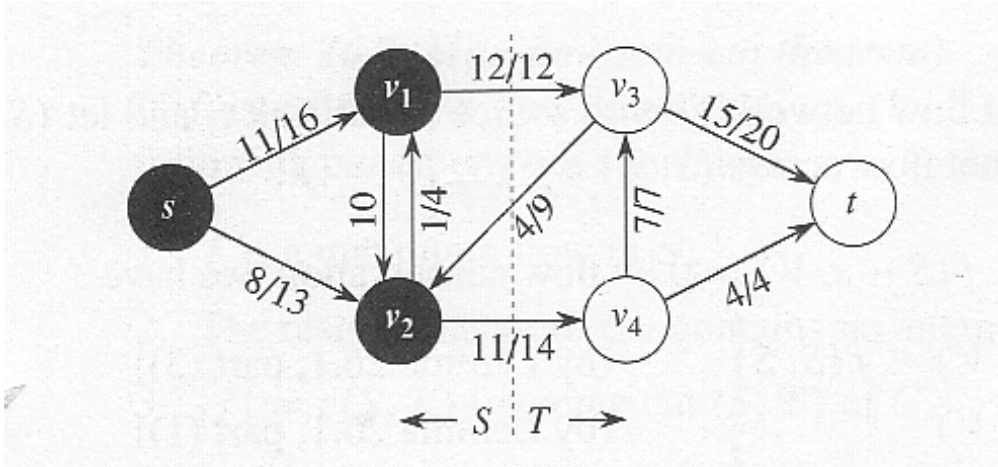
- The **flow through a cut** $f(S,T)$ is the total flow through edges from $S$ to $T$ minus the total flow through edges from $T$ to $S$:

$$f(S,T) = \sum_{(u,v)\in E} \{f(u,v) : u \in S \ \& \ v \in T\} - \sum_{(u,v)\in E} \{f(u,v) : u \in T \ \& \ v \in S\}$$

- Clearly, $f(S,T) \leq c(S,T)$ because for every edge $(u,v) \in E$ we assumed $f(u,v) \leq c(u,v)$, and $f(u,v) \geq 0$.

# Cuts in flow networks

- Example:




- In the above example the net flow across the cut is given by

$$f(S,T) = f(v_1,v_3) + f(v_2,v_4) - f(v_2,v_3) = 12 + 11 - 4 = 19$$

- Note that the flow in the opposite direction (from T to S) is subtracted.
- The capacity of the cut $c(S,T)$ is given by

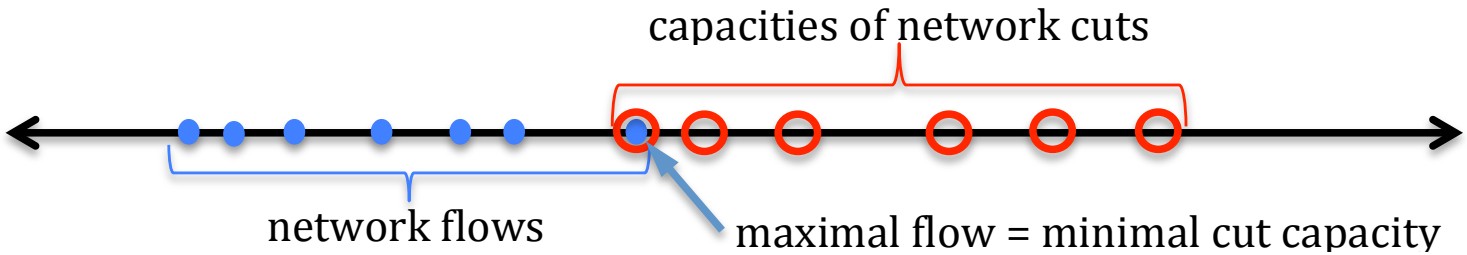
$$c(S,T) = c(v_1,v_3) + c(v_2,v_4) = 12 + 14 = 26$$

- As we have mentioned, we add only the capacities of vertices from S to T and not of vertices in the opposite direction.

# Cuts in flow networks

- **Theorem (max-flow min-cut):** The maximal amount of flow in a flow network is equal to the capacity of the cut of minimal capacity.

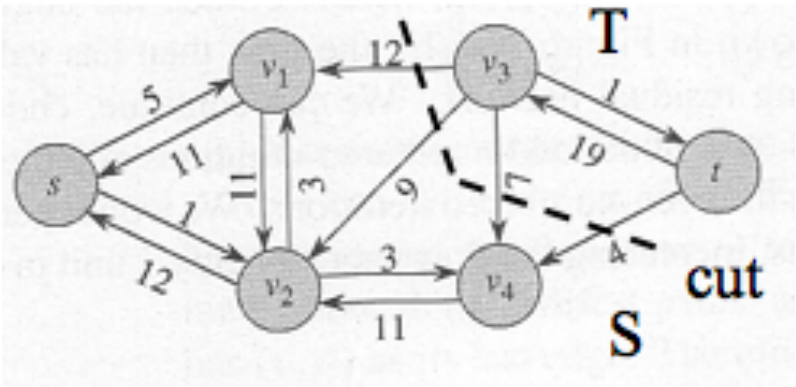
- Since any flow has to cross every cut, any flow must be smaller than the capacity of any cut: $f = f(S,T) \leq c(S,T)$.
- Thus, if we find a flow $f$ which equals the capacity of some cut $(S,T)$, then such flow must be maximal and the capacity of such a cut must be minimal.

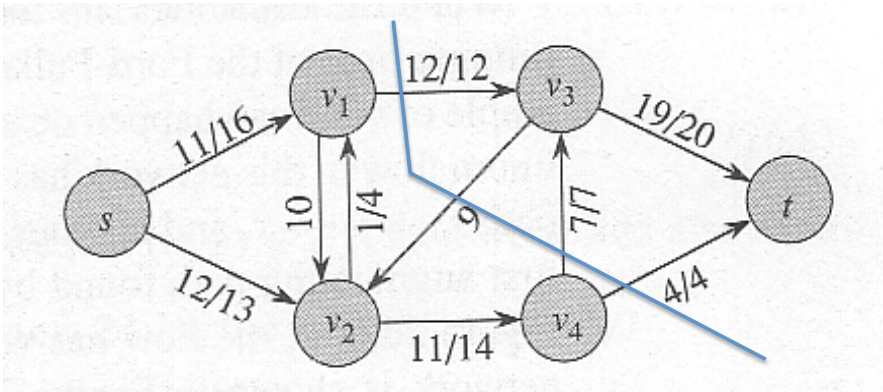



- We now show that when the Ford - Fulkerson algorithm terminates, it produces a flow equal to the capacity of an appropriately defined cut.

# Cuts in flow networks

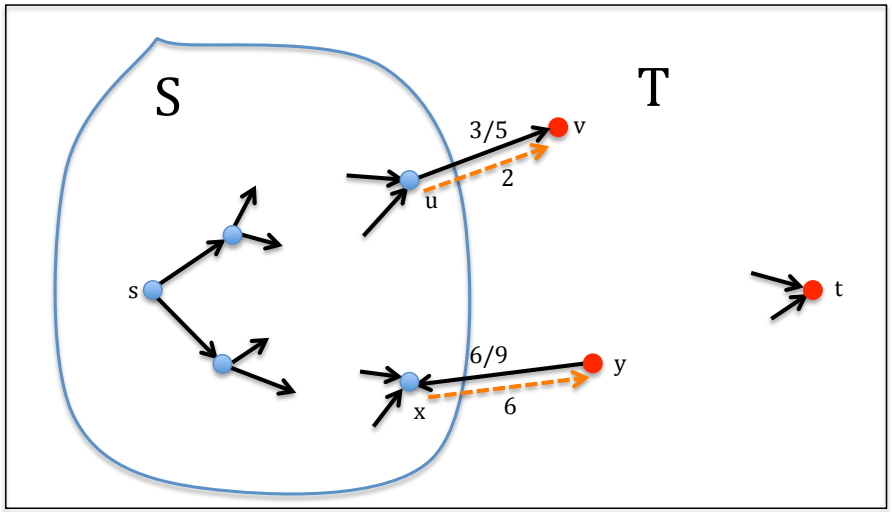
- Assume that the Ford - Fulkerson algorithm has terminated and that there are no more augmenting paths from the source $s$ to the sink $t$ in the last residual flow network.
- Define $S$ to be the source $s$ and all vertices $u$, such that there is a path in the residual flow network from the source $s$ to that vertex $u$.
- Define $T$ to be the set of all vertices for which there is no such path.
- Since there are no more augmenting paths from $s$ to $t$, clearly the sink $t$ belongs to $T$.



# Cuts in flow networks

- **Claim:** all the edges from $S$ to $T$ are **fully occupied** with flow, and all the edges from $T$ to $S$ are **empty**.



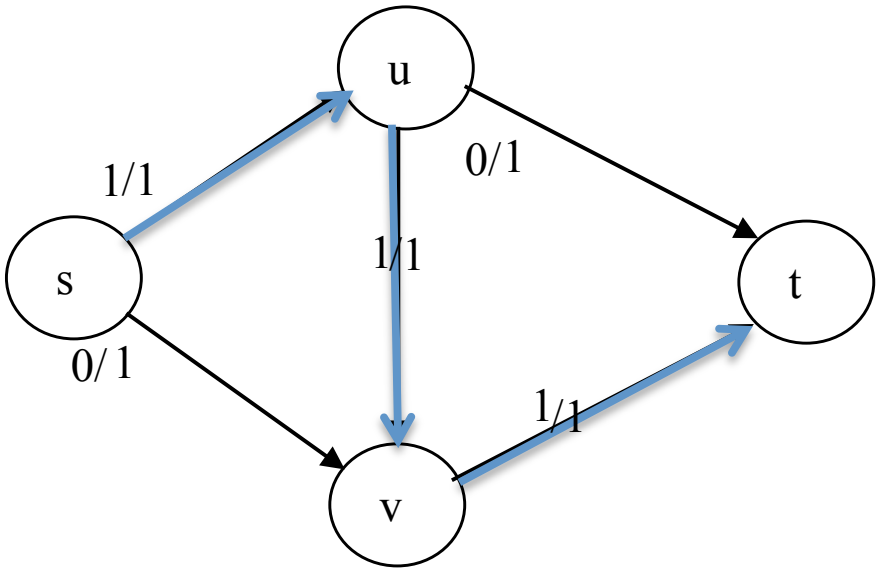

- **Proof:**
  - If an edge $(u, v)$ from $S$ to $T$ had some additional capacity left, then in the residual flow network the path from $s$ to $u$ could be extended to a path from $s$ to $v$ which contradict our assumption that $v \in T$.
  - If an edge $(y, x)$ from $T$ to $S$ had any flow in it, then in the residual flow network the path from $s$ to $x$ could be extended to a path from $s$ to $y$, which is again a contradiction with our assumption that $y \in T$.

# Cuts in flow networks

- Since all edges from $S$ to $T$ are occupied with flows to their full capacity and since there is no flow from $T$ to $S$, the flow across the cut $(S, T)$ is precisely equal to the capacity of this cut.

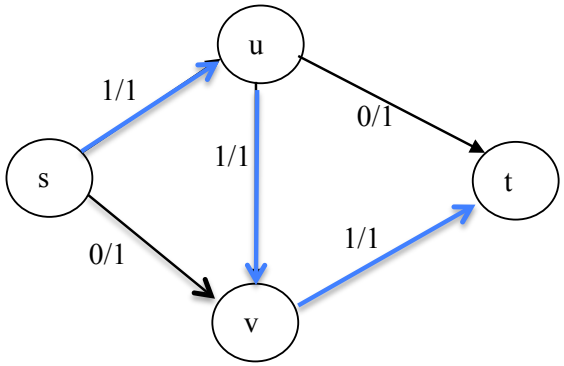
- Thus, such a flow is maximal and the corresponding cut is a minimal cut, regardless of the particular way how the augmenting paths were chosen.

- Trying to do the Ford Fulkerson algorithm without constructing the residual flow can make you miss augmenting paths: on the network flow diagram below it might appear that there is no augmenting paths:

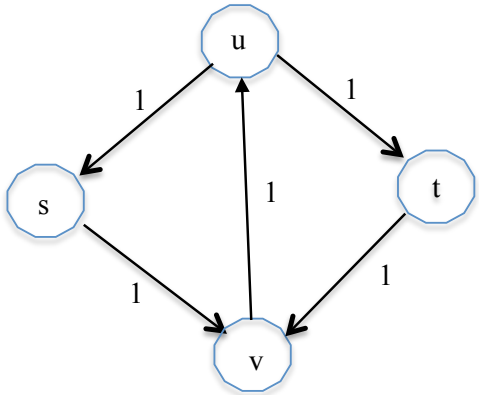


# Cuts in flow networks

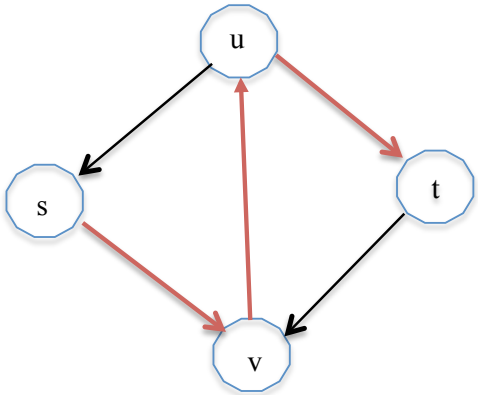
- But such a path is obvious in the residual network flow:



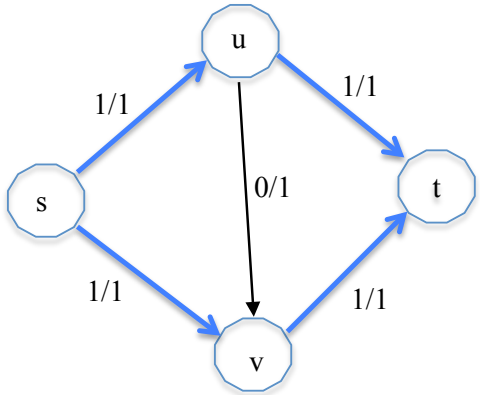

Network with flow




Residual network



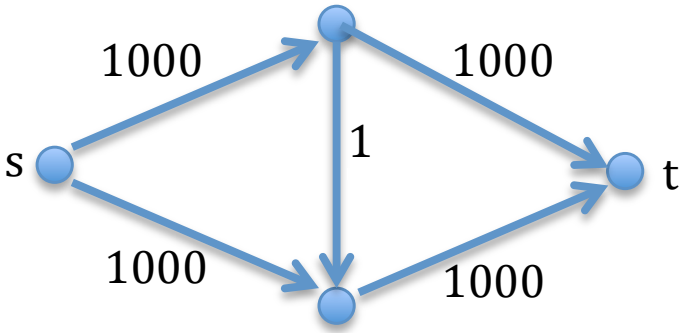

Augmenting path in the residual network

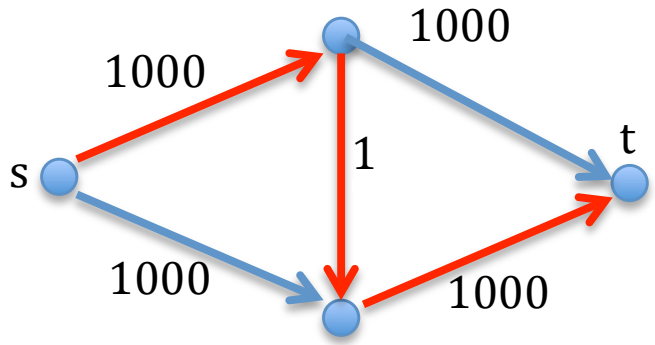


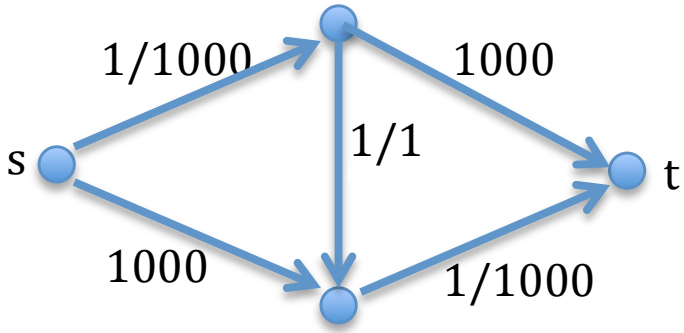

Final flow

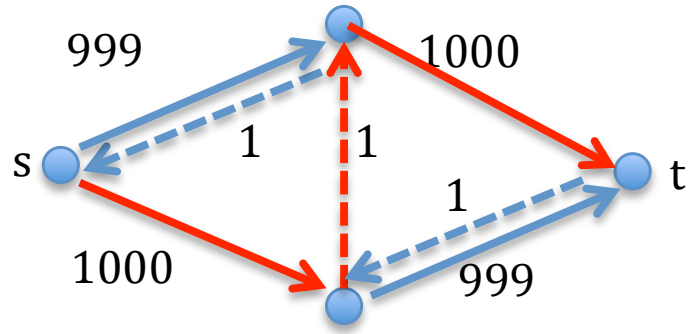
# Cuts in flow networks

- How efficient is the Ford-Fulkerson algorithm?

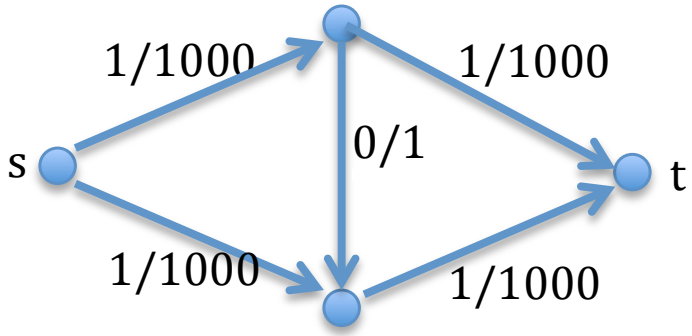


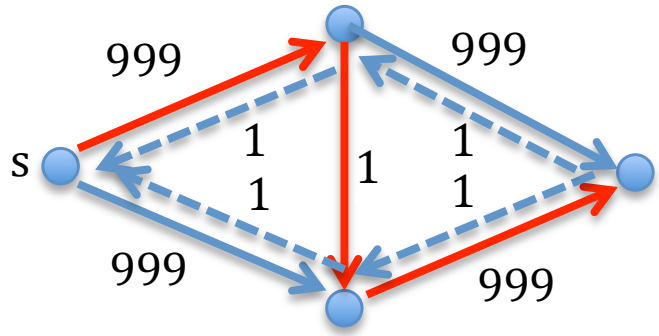



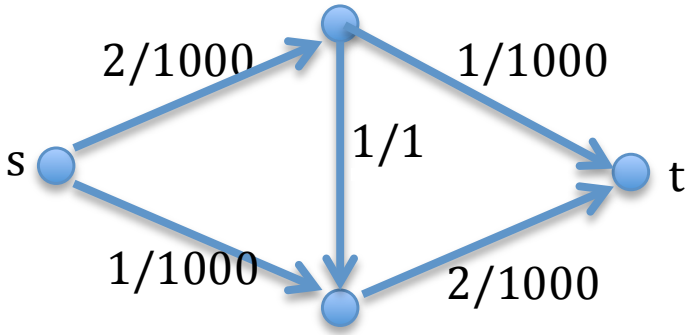



# Cuts in flow networks




- The Ford-Fulkerson algorithm can potentially run in time proportional to the value of max flow, which can be exponential in the size of the input.

# Edmonds-Karp Max Flow Algorithm

- The **Edmonds-Karp** algorithm improves the Ford Fulkerson algorithm in a simple way: always choose the *shortest path* from the source $s$ to the sink $t$, where the "shortest path" means the fewest number of edges, regardless of their capacities (i.e., each edge has the same unit weight).

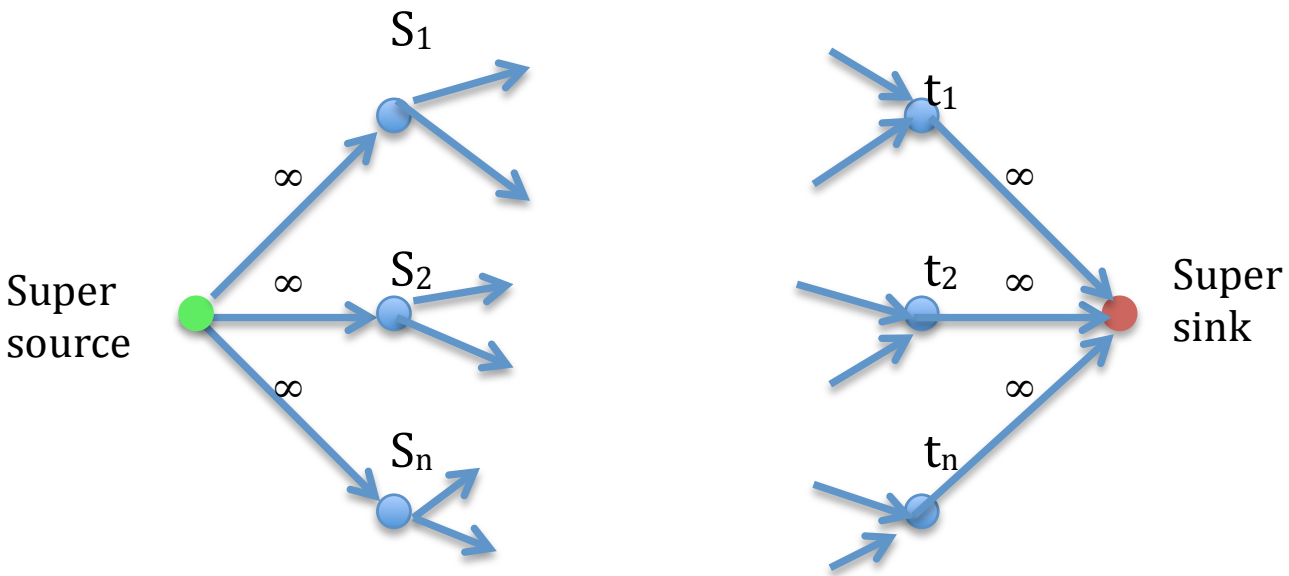
- Note that this choice is somewhat counter intuitive: we preferably take edges with small capacities over edges with large capacities, for as long as they are along a shortest path from $s$ to $t$.

- Why does such a choice speed up the Ford - Fulkerson algorithm? To see this, one needs a tricky mathematical proof, see the textbook. One can prove that such algorithm runs in time $O(|V|\ |E|^2)$.

- The fastest max flow algorithm to date, an extension of the PREFLOW-PUSH algorithm runs in time $O(|V|^3)$.

# Networks with multiple sources and sinks

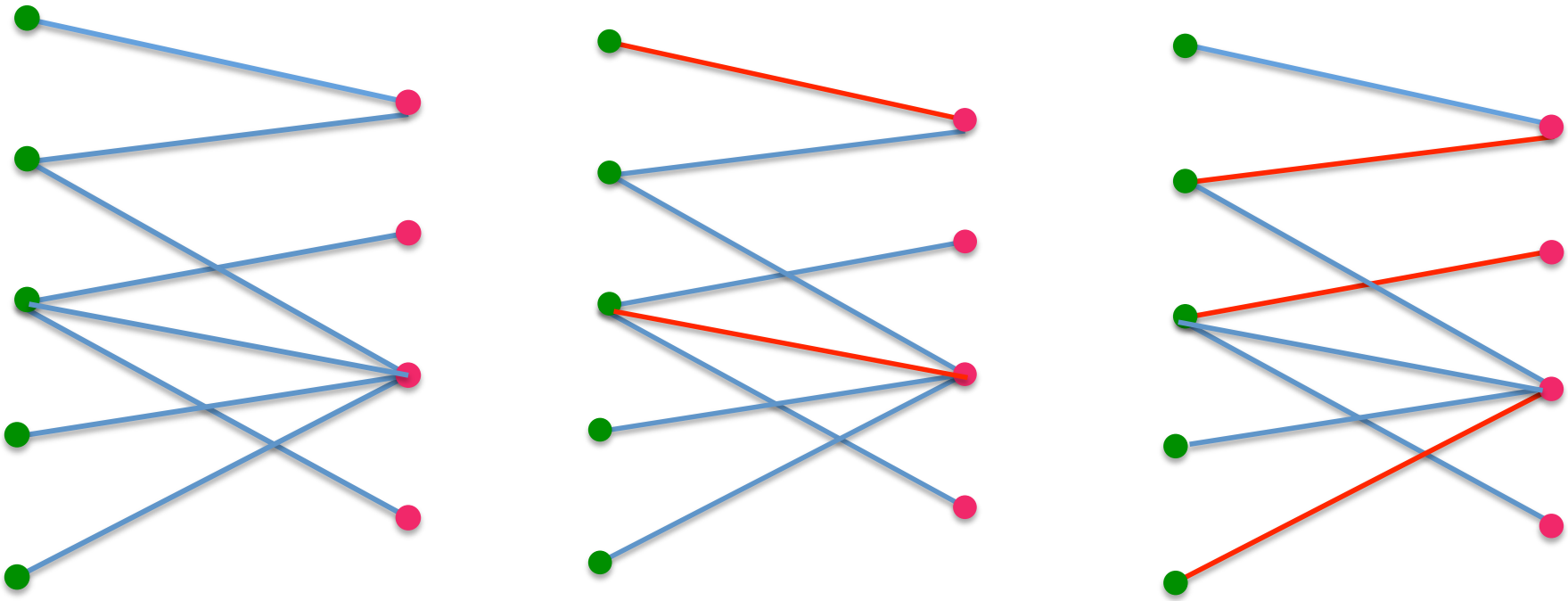
- Flow networks with multiple sources and sinks are reducible to networks with a single source and single sink by adding a "super-sink" and "super-source" and connecting them to all sources and sinks, respectively, by edges of infinite capacities.



# Maximum matching in bipartite graphs

- We will consider bipartite graphs; i.e., graphs whose vertices can be split into two subsets, $L$ and $R$ such that every edge $e \in E$ has one end in the set $L$ and the other in the set $R$.

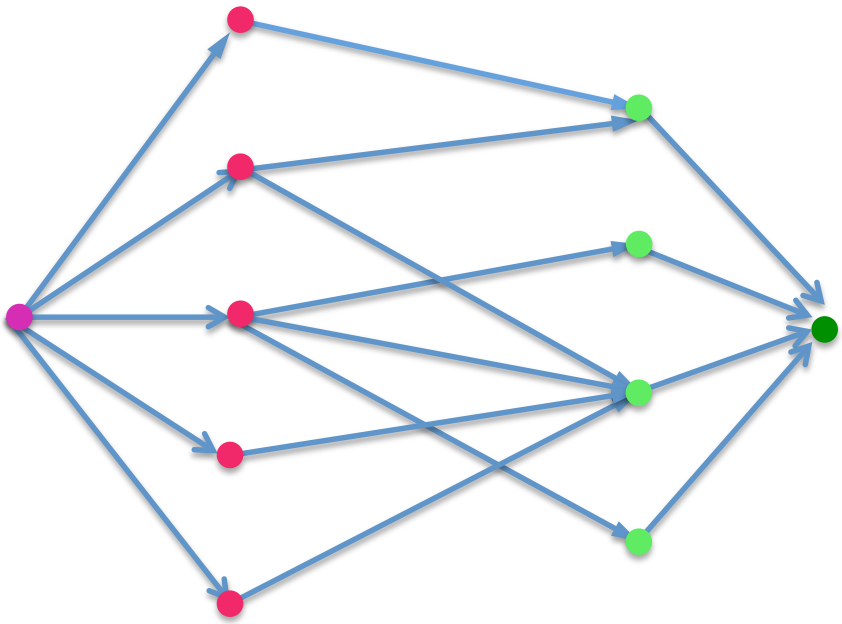
- A matching in a graph $G$ is a subset $M$ of edges in $E$ such that each vertex of the graph belongs to at most one of the edges in the matching $M$.

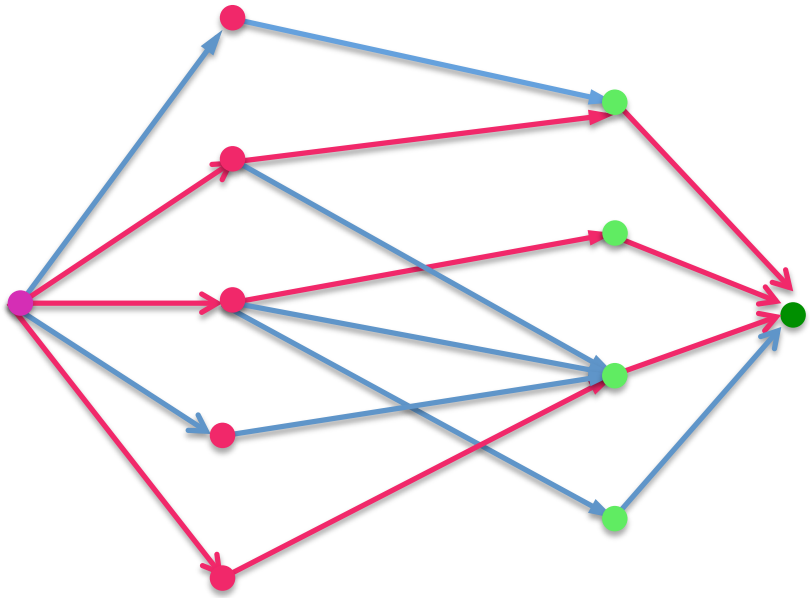


# Maximum matching in bipartite graphs

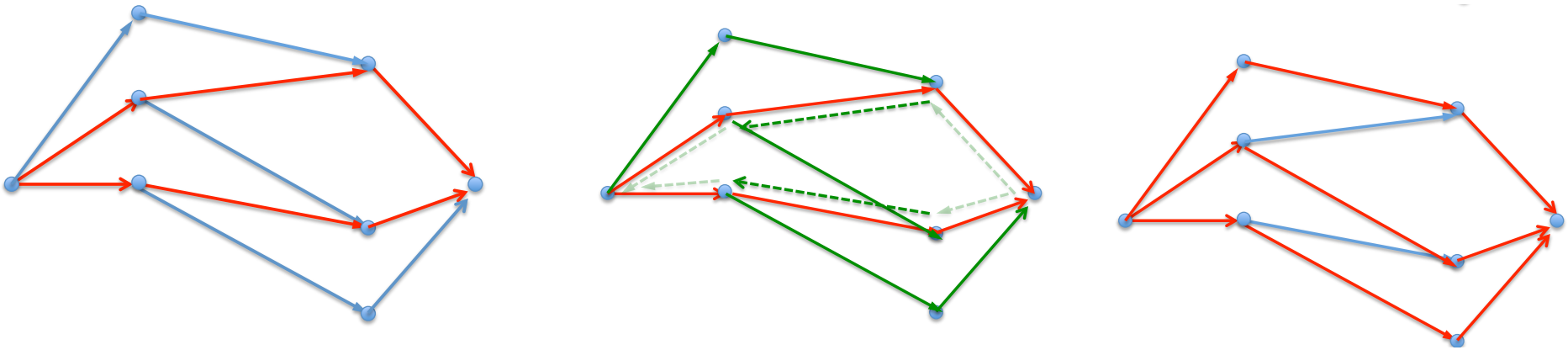
- A maximum matching in a bipartite graph $G$ is a matching containing the largest possible number of edges.

- We turn a Maximum Matching problem into a Max Flow problem by adding a super source and a super sink, and by giving all edges a capacity of 1





# Maximum matching in bipartite graphs



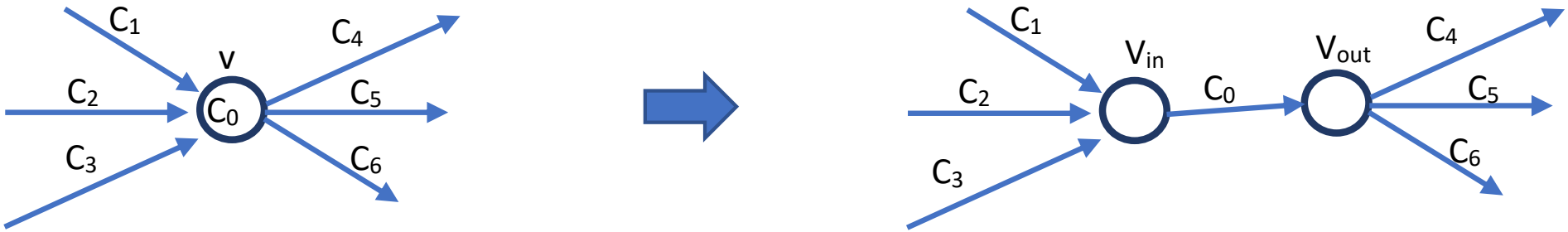
- Note how the residual flow network allows rerouting the flow in order to increase the total throughput.

# Max Flow with vertex capacities

- Sometimes not only the edges but also the vertices $v_i$ of the flow graph might have capacities $C(v_i)$, which limit the total throughput of the flow coming to the vertex (and, consequently, also leaving the vertex):

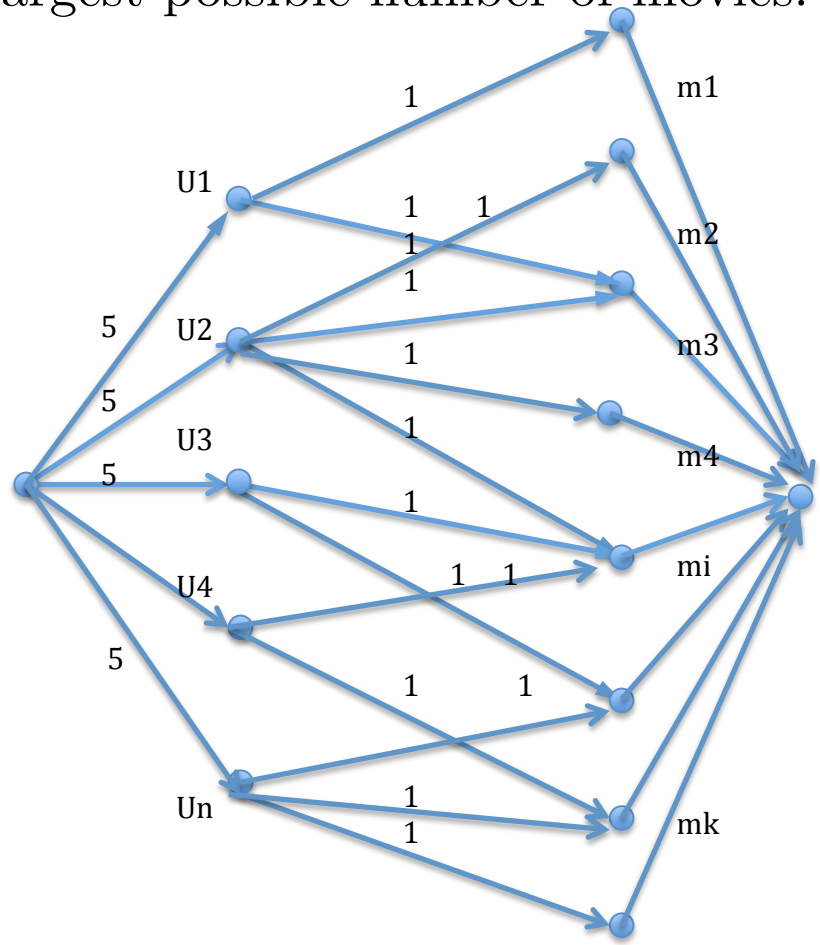
$$\sum_{e(u,v)\in E} f(u,v) = \sum_{e(v,w)\in E} f(v,w) \leq C(v)$$

- Such case is reduced to the case where only edges have capacities by splitting each vertex $v$ with limited capacity $C(v)$ into two vertices $v_{in}$ and $v_{out}$ so that all edges coming into $v$ go into $v_{in}$, all edges leaving $v$ now leave $v_{out}$ and by connecting the new vertices $v_{in}$ and $v_{out}$ with an edge $e^* = (v_{in}, v_{out})$ with capacity equal to the capacity of the original vertex $v$:



# Applications of Max Flow algorithm

- Assume you have a movie rental agency. At the moment you have $k$ movies in stock, with $m_i$ copies of movie $i$. Each of $n$ customers can rent out at most 5 movies at a time. The customers have sent you their preferences which are lists of movies they would like to see. Your goal is to dispatch the largest possible number of movies.



# Example problem: Movie Rental

Construct a flow network with:

- source $s$ and sink $t$,
- a vertex $u_i$ for each customer $i$ and a vertex $v_j$ for each movie $j$,
- for each $i$, an edge from $s$ to $u_i$ with capacity 5,
- for each customer $i$, for each movie $j$ that they are willing to see, an edge from $u_i$ to $v_j$ with capacity 1.
- for each $j$, an edge from $v_j$ to $t$ with capacity $m_j$.

# Example problem: Movie Rental

Consider a flow in this graph:

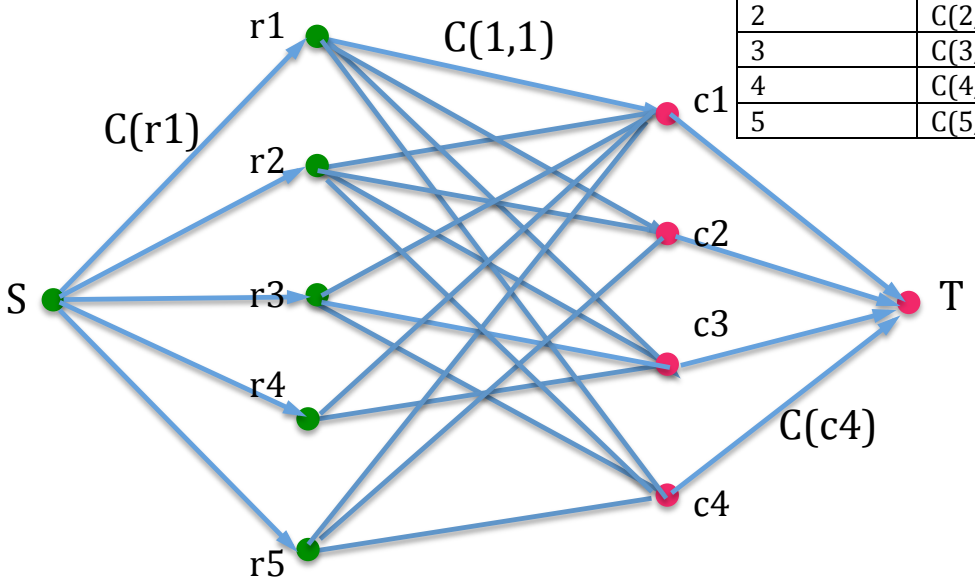
- Each customer-movie edge has capacity 1, so we will interpret a flow of 1 from $u_i$ to $v_j$ as assigning movie $j$ to customer $i$.
- Each customer only receives movies that they want to see.
- By *flow conservation*, the amount of flow sent along the edge from $s$ to $u_i$ is equal to the total flow sent from $u_i$ to all movie vertices $v_j$, so it represents the number of movies received by customer $i$. Again, the *capacity constraint* ensures that this does not exceed 5 as required.
- Similarly, the movie stock levels $m_j$ are also respected.

# Example problem: Movie Rental

- Therefore, any flow in this graph corresponds to a valid allocation of movies to customers.
- To maximise the movies dispatched, we find the maximum flow using the Edmonds-Karp algorithm.
- There are $n + k + 2$ vertices and up to $nk + n + k$ edges, so the time complexity is $O((n + k + 2)(nk + n + k)^2)$, which is polynomial in $n$ and $k$ as required.
- Since the value of any flow is constrained by the total capacity from $s$, which in this case is $5n$, we can achieve a tighter bound of $O(E|f|) = O(n(nk + n + k)) = O(n^2k)$.

# Applications of Max Flow algorithm

- The storage space of a ship is in the form of a rectangular grid of cells with $n$ rows and $m$ columns. Some of the cells are taken by support pillars and cannot be used for storage, so they have 0 capacity. You are given the capacity of every cell; cell in row $r_i$ and column $c_j$ has capacity $C(i, j)$. To ensure the stability of the ship, the total weight in each row $r_i$ must not exceed $C(r_i)$ and the total weight in each column $c_j$ must not exceed $C(c_j)$. Find how to allocate the cargo weight to each cell to maximise to total load without exceeding the limits per column, limits per row and limits per available cell.




| row\column \ | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | C(1,1) | C(1,2) | C(1,3) | C(1,4) |
| 2 | C(2,1) | C(2,2) | C(2,3) | C(2,4) |
| 3 | C(3,1) | C(3,2) | 0 | C(3,4) |
| 4 | C(4,1) | 0 | C(4,3) | 0 |
| 5 | C(5,1) | C(5,2) | 0 | C(5,4) |

| row | capacity |
|---|---|
| r1 | C(r1) |
| r2 | C(r2) |
| r3 | C(r3) |
| r4 | C(r4) |
| r5 | C(r5) |

| column | capacity |
|---|---|
| c1 | C(c1) |
| c2 | C(c2) |
| c3 | C(c3) |
| c4 | C(c4) |

# Example problem: Cargo Allocation

Construct a flow network with:

- source $s$ and sink $t$,
- a vertex $r_i$ for each row $i$ and a vertex $c_j$ for each column $j$,
- for each $i$, an edge from $s$ to $r_i$ with capacity $C_r(i)$,
- for each cell $(i, j)$ which is not a pillar, an edge from $r_i$ to $c_j$ with capacity $C(i, j)$.
- for each $j$, an edge from $c_j$ to $t$ with capacity $C_c(j)$.

# Example problem: Cargo Allocation

- Consider a flow in this graph.
- We will interpret the amount of flow sent along the edge from $r_i$ to $c_j$ as the weight of cargo stored in cell $(i, j)$.
- By the *capacity constraint*, this weight cannot exceed the edge capacity $C(i, j)$, so the weight limit of each cell is satisfied.
- By *flow conservation*, the amount of flow sent along the edge from $s$ to $r_i$ is equal to the total flow sent from $r_i$ to all column vertices $c_j$, so it represents the total weight stored in row $i$. Again, the *capacity constraint* ensures that this does not exceed $C_r(i)$ as required.
- Similarly, the column capacities $C_c(j)$ are also respected.

# Example problem: Cargo Allocation

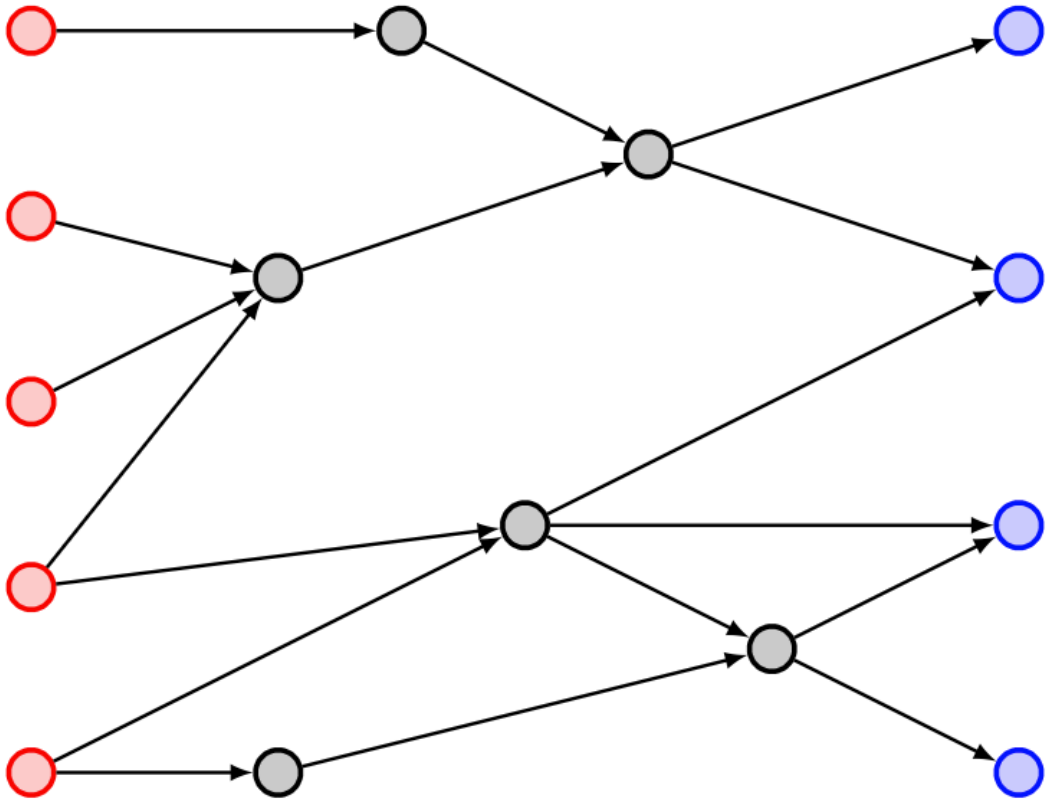
- Therefore, any flow in this graph corresponds to a valid allocation of cargo to cells.
- To maximise the cargo allocated, we find the maximum flow using the Edmonds-Karp algorithm.
- There are $n + m + 2$ vertices and up to $nm + n + m$ edges, so the time complexity is

$$\begin{aligned} &O((n + m + 2)(nm + n + m)^2) \\ &= O((n + m)(nm)^2), \end{aligned}$$

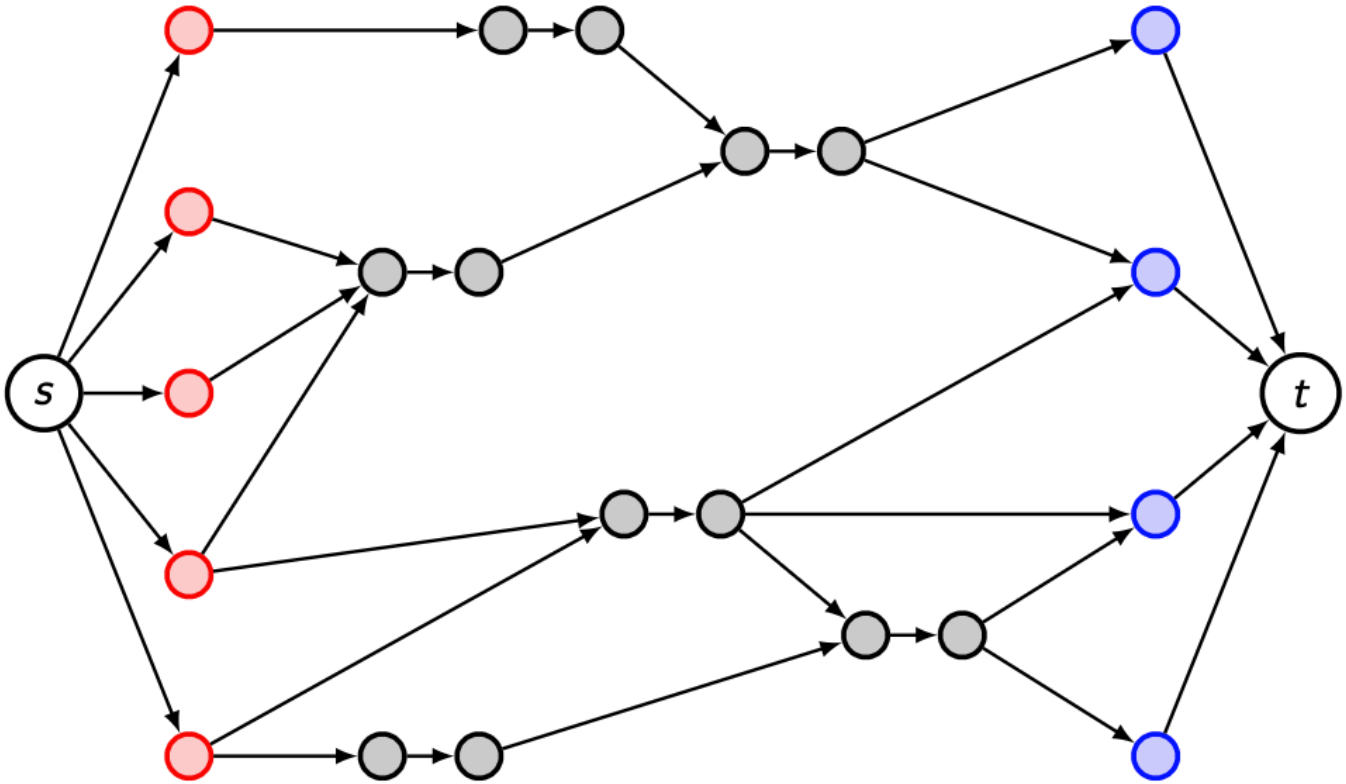
which is polynomial in $n$ and $m$ as required.

# Applications of Max Flow algorithm

- You are given a connected, directed graph $G$ with $N$ vertices. Out of these $N$ vertices $k$ are painted red, $m$ are painted blue, and the remaining $N - k - m > 0$ of the vertices are black. Red vertices have only outgoing edges and blue vertices have only incoming edges. Your task is to determine the largest possible number of vertex-disjoint (i.e., non-intersecting) paths in this graph, each of which starts at a red vertex and finishes at a blue vertex.



# Example problem: Vertex-Disjoint Paths



# Example problem: Vertex-Disjoint Paths

Construct a flow network with:

- super-source $s$ joined to each red vertex,
- each blue vertex joined to super-sink $t$,
- for each black vertex, two vertices $v_{in}$ and $v_{out}$, joined by an edge,
- each edge of the original graph, with edges from a black vertex drawn from the corresponding out-vertex, and edges to a black vertex drawn to the corresponding in-vertex.

All edges have capacity 1.

# Example problem: Vertex-Disjoint Paths

- Consider a flow in this graph.
- We will interpret each flowed edge as contributing to one of the red-blue paths.
- *Flow conservation* ensures that each unit of flow travels from $s$ to a red vertex, then to zero or more black vertices, before arriving at a blue vertex and terminating at $t$.

# Example problem: Vertex-Disjoint Paths

- By the *capacity constraint*, each red vertex receives at most one unit of flow from $s$, so *flow conservation* ensures that at most one edge from that vertex is flowed, i.e. we use this vertex in at most one path.
- Similarly, each blue vertex is used in at most one path also.
- Likewise, each in-vertex and out-vertex pair contributes to at most one path, so the paths are indeed vertex-disjoint.

# Example problem: Vertex-Disjoint Paths

- Therefore, any flow in this graph corresponds to a selection of vertex-disjoint red-blue paths in the original graph.
- To maximise the number of paths, we find the maximum flow using the Edmonds-Karp algorithm.
- There are at most $2n$ vertices and exactly $n+m$ edges, so the time complexity is $O(n(n+m)^2)$, which is polynomial in $n$ and $m$ as required.
- Since the value of any flow is constrained by the total capacities from $s$ and to $t$, which in this case are $r$ and $b$ respectively, we can achieve a tighter bound of $O(E|f|) = O((n+m)\min(r,b)) = O(n(n+m))$.

# PUZZLE!!

You are taking two kinds of medicines, $A$ and $B$; pills of $A$ are completely indistinguishable from pills of $B$. You take one of each every day and they both come in supply of 30 pills. One day you drop both bottles on the floor and you see that 3 pills have fallen on the floor but you do not know how many from each bottle and you cannot tell which ones (if any) are of type $A$ and which are of type $B$. How can you solve the problem of continuing to take 1 of each every day without throwing away any pills?