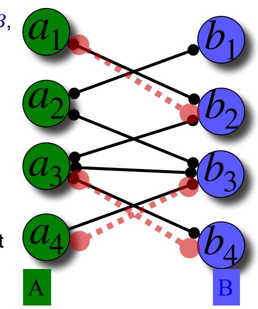
cs445

Bipartite Matching and Max-Flow in a Network

Alon Efrat

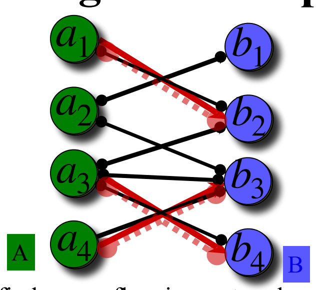
Application: Bipartite Matching.

- A graph $G(V,E)$ is called **bipartite** if V can be partitioned into two sets $V=A\cup B$, and each edge of E connects a vertex of A to a vertex of B. We sometimes denote these graphs by $G(A\cup B,E)$
- Example: The set $A=\{a_1 \ldots a_n\}$ is a set of instructors, the set $B=\{b_1 \ldots b_n\}$ is the set of courses. There is an edge $(\mathbf{a_i}, \mathbf{b_j}) \in \mathbf{E}$ iff instructor a_i could teach course b_j
- A **matching** is a set of edges M of E, where each vertex of A is adjacent to at most one vertex of B, and vice versa.
- *(in the example, each instructor will teach at most one course, and vice versa)*
- **Maximum-cardinality matching:** *Find a matching with as many edges as possible*
- This problem could be solved with in O(nm) time using Ford-Fulkerson algorithm. Faster algorithms exist as well. However, we will use it as an example to the ease of using ILP.



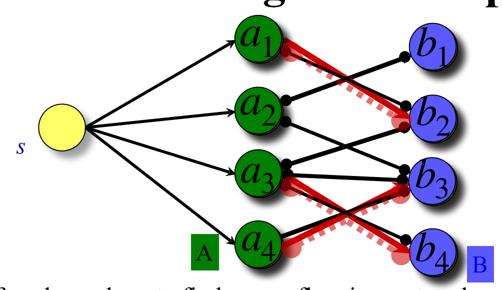
In red: Edge of the matching

Matching and flow problem



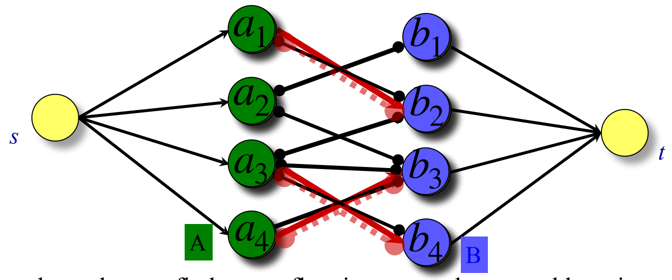
- If we know how to find a max-flow in a network, we could use it so solve a matching problem: For this, we need to express the matching problem as a flow problem:
  A. Add a vertex s, and connect it to each vertex of A.
  B. Add a vertex t, and connect each vertex of B to t.
  C. Assign capacity of 1 to each edge (u,v).
- Find max flow. Assume it is an **integer** flow, so the flow across each edge is either 0 or 1
- Each edge of G that carries flow is in the matching.
- Each edge of G that **does not** carry flow is **not in** the matching.
- <u>**Claim:**</u> The edge between A and B that carry flow form a matching M.
- Proof: We just need to show that no instructor a_i is matched to two courses b_j, b_k, and vice versa

Matching and flow problem



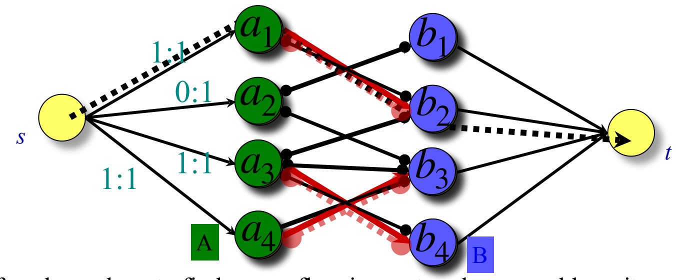
- If we know how to find a max-flow in a network, we could use it so solve a matching problem: For this, we need to express the matching problem as a flow problem:
  A. Add a vertex s, and connect it to each vertex of A.
  B. Add a vertex t, and connect each vertex of B to t.
  C. Assign capacity of 1 to each edge (u,v).
- Find max flow. Assume it is an **integer** flow, so the flow across each edge is either 0 or 1
- Each edge of G that carries flow is in the matching.
- Each edge of G that **does not** carry flow is **not in** the matching.
- <u>**Claim:**</u> The edge between A and B that carry flow form a matching M.
- Proof: We just need to show that no instructor a_i is matched to two courses b_j, b_k, and vice versa

Matching and flow problem



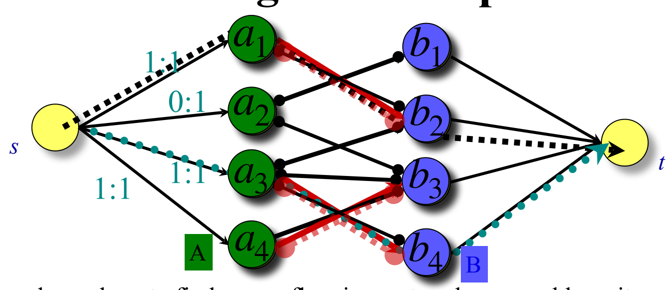
- If we know how to find a max-flow in a network, we could use it so solve a matching problem: For this, we need to express the matching problem as a flow problem:
  A. Add a vertex *s*, and connect it to each vertex of *A*.
  B. Add a vertex *t*, and connect each vertex of *B* to *t*.
  C. Assign capacity of 1 to each edge (u,v).
- Find max flow. Assume it is an **integer** flow, so the flow across each edge is either 0 or 1
- Each edge of *G* that carries flow is in the matching.
- Each edge of *G* that **does not** carry flow is **not in** the matching.
- **Claim:** The edge between *A* and *B* that carry flow form a matching M.
- Proof: We just need to show that no instructor a_i is matched to two courses b_j, b_k, and vice versa

Matching and flow problem



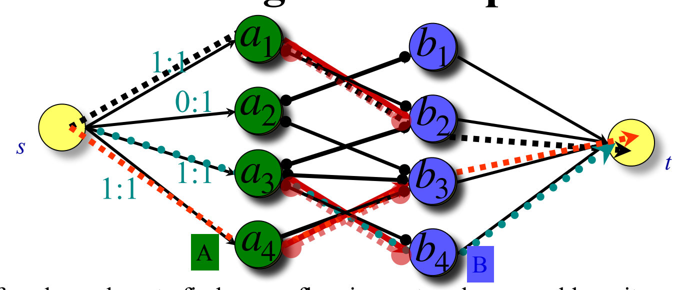
- If we know how to find a max-flow in a network, we could use it so solve a matching problem: For this, we need to express the matching problem as a flow problem:
  A. Add a vertex *s*, and connect it to each vertex of *A*.
  B. Add a vertex *t*, and connect each vertex of *B* to *t*.
  C. Assign capacity of 1 to each edge (u,v).
- Find max flow. Assume it is an **integer** flow, so the flow across each edge is either 0 or 1
- Each edge of *G* that carries flow is in the matching.
- Each edge of *G* that **does not** carry flow is **not in** the matching.
- **Claim:** The edge between *A* and *B* that carry flow form a matching M.
- Proof: We just need to show that no instructor a_i is matched to two courses b_j, b_k, and vice versa

Matching and flow problem



- If we know how to find a max-flow in a network, we could use it so solve a matching problem: For this, we need to express the matching problem as a flow problem:
  A. Add a vertex *s*, and connect it to each vertex of *A*.
  B. Add a vertex *t*, and connect each vertex of *B* to *t*.
  C. Assign capacity of 1 to each edge (u,v).
- Find max flow. Assume it is an **integer** flow, so the flow across each edge is either 0 or 1
- Each edge of *G* that carries flow is in the matching.
- Each edge of *G* that **does not** carry flow is **not in** the matching.
- **Claim:** The edge between *A* and *B* that carry flow form a matching M.
- Proof: We just need to show that no instructor a_i is matched to two courses b_j, b_k, and vice versa

Matching and flow problem



- If we know how to find a max-flow in a network, we could use it so solve a matching problem: For this, we need to express the matching problem as a flow problem:
  A. Add a vertex *s*, and connect it to each vertex of *A*.
  B. Add a vertex *t*, and connect each vertex of *B* to *t*.
  C. Assign capacity of 1 to each edge (u,v).
- Find max flow. Assume it is an **integer** flow, so the flow across each edge is either 0 or 1
- Each edge of *G* that carries flow is in the matching.
- Each edge of *G* that **does not** carry flow is **not in** the matching.
- **Claim:** The edge between *A* and *B* that carry flow form a matching M.
- Proof: We just need to show that no instructor a_i is matched to two courses b_j, b_k, and vice versa

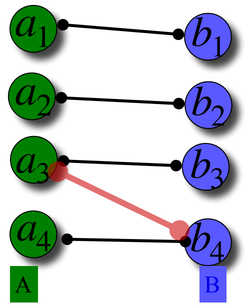
Ford-Fulkerson algorithm for finding max bipartite matching

- This algorithm is actually appropriate for any network flow problem, but notations and proofs are simpler if we concentrate on the matching directly.

- Algorithm: Start then $M = \emptyset$. No edge is in the matching.
- Output: $|M|$ is as large as possible
- At each step of the algorithm, we increase the cardinality of M by 1.

- General Step: Assume M is given. Terminology:
- A ***matched vertex*** is a vertex which is an edgepoint of an edge of M. Vertices that are not matched are called ***exposed vertices***.

- We will denote all matched edge $M \subseteq E$ by a thick red segments, and edge of $E \setminus M$ are depicted by a straight edge. Sometimes we will denote these edges by 

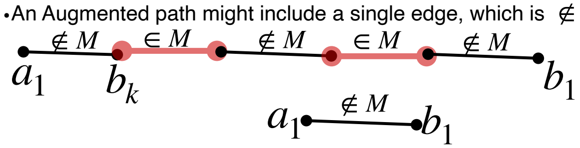


A **matching** is a set of edges M of E, where each vertex of A is adjacent to at most one vertex of B, and vice versa.

In red: Edge of the matching

- An **augmenting path** is a path that starts with an expose vertex of A, ends at an exposed vertex of B, and its edges alternates: An edge $\notin M$ followed by an edge $\in M$, followed by an edge $\notin M$ and so on.

- An Augmented path might include a single edge, which is $\notin M$



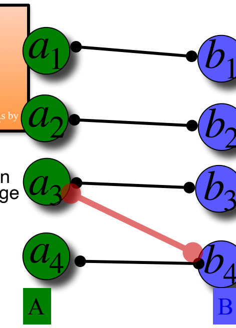
Ford-Fulkerson algorithm for finding max bipartite matching

This algorithm is actually appropriate for any network flow problem, but notations and proofs are simpler if we concentrate on the matching directly.

Algorithm: Start then $M = \emptyset$. No edge is in the matching.
Output: $|M|$ is as large as possible
At each step of the algorithm, we increase the cardinality of M by 1.

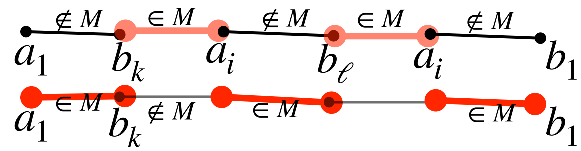
General Step: Assume M is given. Terminology:
A ***matched vertex*** is a vertex which is an edgepoint of an edge of M. Vertices that are not matched are called ***exposed vertices***.

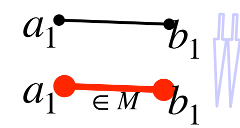
We will denote all matched edge $M \subseteq E$ by a thick red segments, and edge of $E \setminus M$ are depicted by a straight edge. Sometimes we will denote these edgeb_ks by



- An **augmenting path** is a path that starts with an expose vertex of A, ends at an exposed vertex of B, and its edges alternates: An edge $\notin M$ followed by an edge $\in M$, followed by an edge $\notin M$ and so on.

- Augmented path might include a single edge, which is $\notin M$
- Lets p be an augmenting path. The operation of **augmentation a path** consists of
  - Insert into M all edges of p which are $\notin M$, and remove from M all edges that originally were in M.



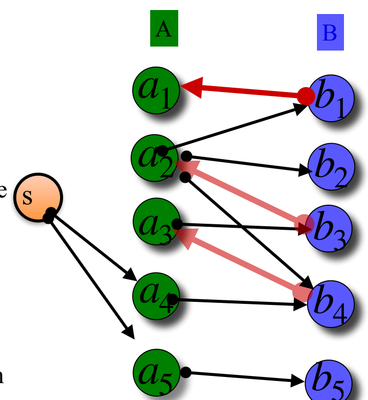


Claims:

1. A vertex that was matched before the augmentation, is matched after the augmentation
2. Matching is 1-1 (no course taught by two teaches, no teacher teaches two courses.
3. Augmentation increases the number of edges in the matching by 1.

How to find augmenting paths

- Makes the graph a directed graph:
  - Edges $\in M$ are directed from right to left
  - Edges $\notin M$ are directed from left to right
  - Add a vertex s, and connect it to every **exposed** $a_i \in A$
- Run DFS or BFS from s.
- Every path that leads to an exposed vertex must be an augmented path. And
- If there is an augmented path, this process will find this path.



Once an augmented bath is found, we augment its edges, and restart (re-bulding the directed graph).

If no augmented path is found, stop - M is maximum cardinality matching. (we will need to prove it)

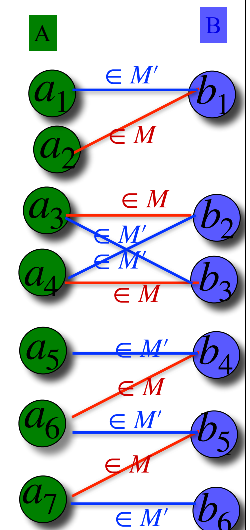
Running time: Each iteration, we increase |M| by 1, so the number of iterations is $\leq \min\{|A|,|B|\} \leq n$.

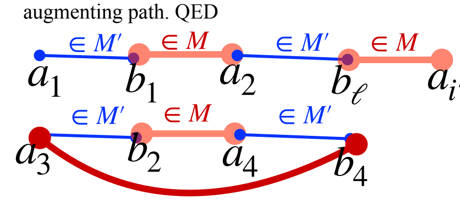
- Finding an augmented path is done via DFS or BFS, so its time is $O(|E|+|V|)$

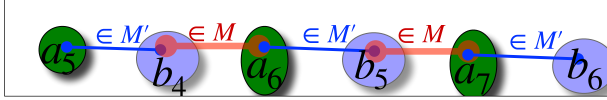
- Overall time $O(|E||V|) = O(mn)$

Optimality Theorem : M is maximum iff there is no augmenting path
Proof: One direction is trivial: If there is an augmenting path, then we could increase $|M|$, so M it is not optimum. Lets prove the second direction:

1. On the other hand, assume M is not optimum. Let M' be another matching such that $|M| < |M'|$.
2. Let think about $U \stackrel{def}{=} M \oplus M' \subseteq E$. These are the edges which are either in M or in M', but not in both. Some edges of E are in neither M nor in M'.
3. Each vertex $v \in V$ is on $\leq$ one edge of M and on $\leq$ one edge of M'.
4. Every path of U is an alternating path - an edge from M followed by an edge from M' and so on.
5. U might consists of several pathS and several cycleS.
6. Every cycle must have an even length (why?).
7. However, since |M|<|M'|, one of the alternating path contains more edges from M'. This must be a path whose first and last edge are from M'. This is an augmenting path. QED

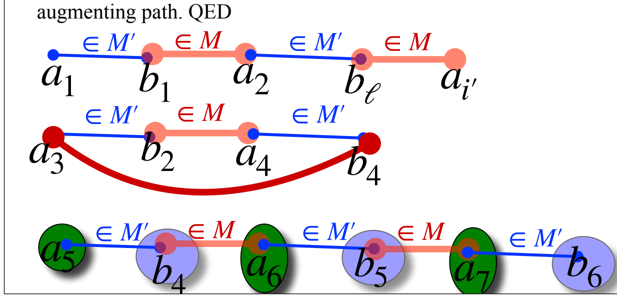


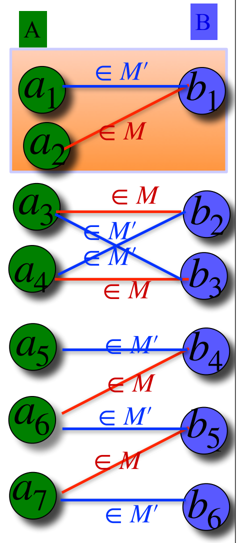




Optimality Theorem : M is maximum iff there is no augmenting path
Proof: One direction is trivial: If there is an augmenting path, then we could increase $|M|$, so M it is not optimum. Lets prove the second direction:

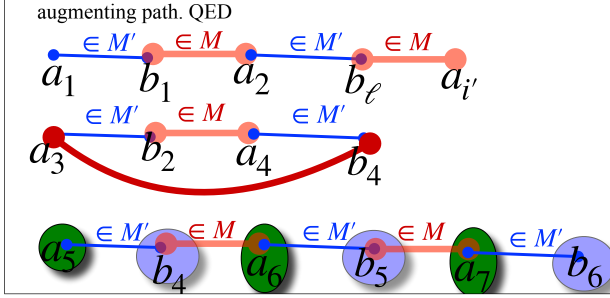
1. On the other hand, assume M is not optimum. Let M' be another matching such that $|M| < |M'|$.
2. Let think about $U \overset{def}{=} M \oplus M' \subseteq E$. These are the edges which are either in M or in M', but not in both. Some edges of E are in neither M nor in M'.
3. Each vertex $v \in V$ is on $\leq$ one edge of M and on $\leq$ one edge of M'.
4. Every path of U is an alternating path - an edge from M followed by an edge from M' and so on.
5. U might consists of several pathS and several cycleS.
6. Every cycle must have an even length (why?).
7. However, since |M|<|M'|, one of the alternating path contains more edges from M'. This must be a path whose first and last edge are from M'. This is an augmenting path. QED

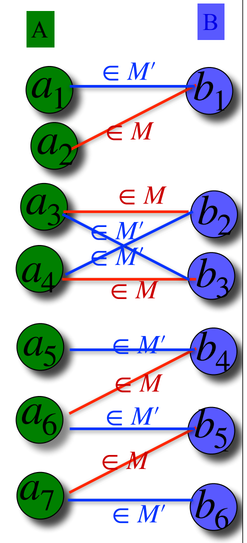




Optimality Theorem : M is maximum iff there is no augmenting path
Proof: One direction is trivial: If there is an augmenting path, then we could increase $|M|$, so M it is not optimum. Lets prove the second direction:

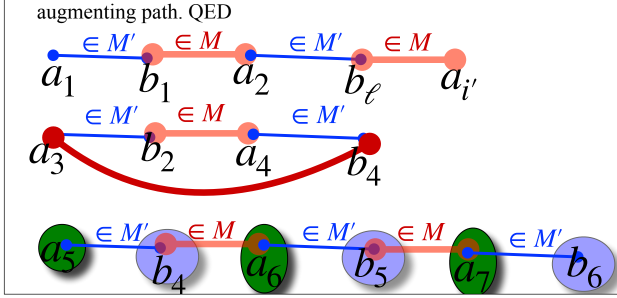
1. On the other hand, assume M is not optimum. Let M' be another matching such that $|M| < |M'|$.
2. Let think about $U \overset{def}{=} M \oplus M' \subseteq E$. These are the edges which are either in M or in M', but not in both. Some edges of E are in neither M nor in M'.
3. Each vertex $v \in V$ is on $\leq$ one edge of M and on $\leq$ one edge of M'.
4. Every path of U is an alternating path - an edge from M followed by an edge from M' and so on.
5. U might consists of several pathS and several cycleS.
6. Every cycle must have an even length (why?).
7. However, since |M|<|M'|, one of the alternating path contains more edges from M'. This must be a path whose first and last edge are from M'. This is an augmenting path. QED

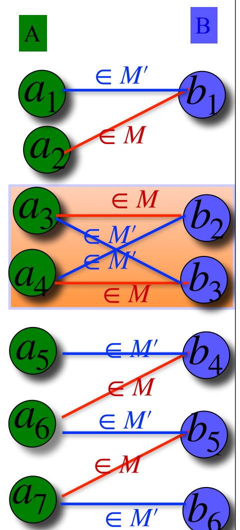




Optimality Theorem : M is maximum iff there is no augmenting path
Proof: One direction is trivial: If there is an augmenting path, then we could increase $|M|$, so M it is not optimum. Lets prove the second direction:

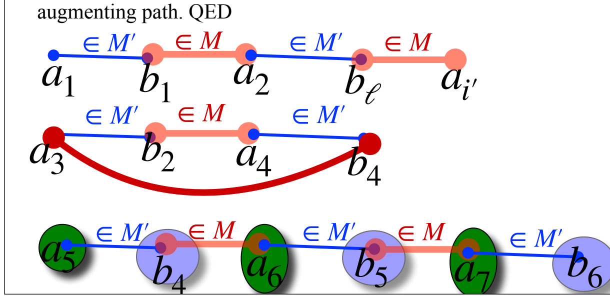
1. On the other hand, assume M is not optimum. Let M' be another matching such that $|M| < |M'|$.
2. Let think about $U \overset{def}{=} M \oplus M' \subseteq E$. These are the edges which are either in M or in M', but not in both. Some edges of E are in neither M nor in M'.
3. Each vertex $v \in V$ is on $\leq$ one edge of M and on $\leq$ one edge of M'.
4. Every path of U is an alternating path - an edge from M followed by an edge from M' and so on.
5. U might consists of several pathS and several cycleS.
6. Every cycle must have an even length (why?).
7. However, since |M|<|M'|, one of the alternating path contains more edges from M'. This must be a path whose first and last edge are from M'. This is an augmenting path. QED

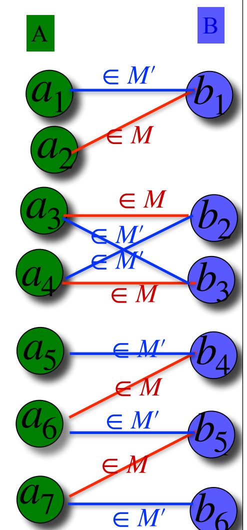




Optimality Theorem : M is maximum iff there is no augmenting path
Proof: One direction is trivial: If there is an augmenting path, then we could increase $|M|$, so M it is not optimum. Lets prove the second direction:

1. On the other hand, assume M is not optimum. Let M' be another matching such that $|M| < |M'|$.
2. Let think about $U \overset{def}{=} M \oplus M' \subseteq E$. These are the edges which are either in M or in M', but not in both. Some edges of E are in neither M nor in M'.
3. Each vertex $v \in V$ is on $\leq$ one edge of M and on $\leq$ one edge of M'.
4. Every path of U is an alternating path - an edge from M followed by an edge from M' and so on.
5. U might consists of several pathS and several cycleS.
6. Every cycle must have an even length (why?).
7. However, since |M|<|M'|, one of the alternating path contains more edges from M'. This must be a path whose first and last edge are from M'. This is an augmenting path. QED

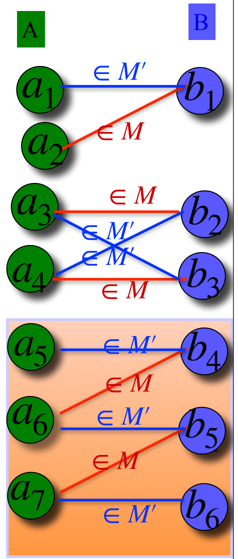


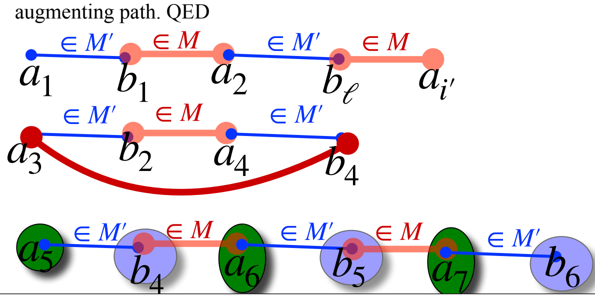


Optimality Theorem : M is maximum iff there is no augmenting path

Proof: One direction is trivial: If there is an augmenting path, then we could increase $|M|$, so M it is not optimum. Lets prove the second direction:

1. On the other hand, assume M is not optimum. Let M' be another matching such that $|M|<|M'|$.
2. Let think about $U \overset{def}{=} M \oplus M' \subseteq E$. These are the edges which are either in M or in M', but not in both. Some edges of E are in neither M nor in M'.
3. Each vertex $v \in V$ is on $\leq$ one edge of M and on $\leq$ one edge of M'.
4. Every path of U is an alternating path - an edge from M followed by an edge from M' and so on.
5. U might consists of several pathS and several cycleS.
6. Every cycle must have an even length (why?).
7. However, since |M|<|M'|, one of the alternating path contains more edges from M'. This must be a path whose first and last edge are from M'. This is an augmenting path. QED

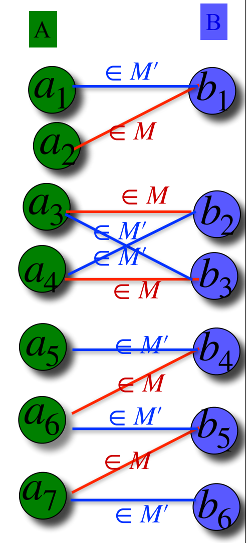


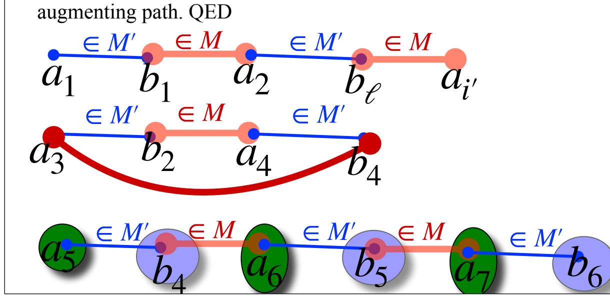


Optimality Theorem : M is maximum iff there is no augmenting path

Proof: One direction is trivial: If there is an augmenting path, then we could increase $|M|$, so M it is not optimum. Lets prove the second direction:

1. On the other hand, assume M is not optimum. Let M' be another matching such that $|M|<|M'|$.
2. Let think about $U \overset{def}{=} M \oplus M' \subseteq E$. These are the edges which are either in M or in M', but not in both. Some edges of E are in neither M nor in M'.
3. Each vertex $v \in V$ is on $\leq$ one edge of M and on $\leq$ one edge of M'.
4. Every path of U is an alternating path - an edge from M followed by an edge from M' and so on.
5. U might consists of several pathS and several cycleS.
6. Every cycle must have an even length (why?).
7. However, since |M|<|M'|, one of the alternating path contains more edges from M'. This must be a path whose first and last edge are from M'. This is an augmenting path. QED

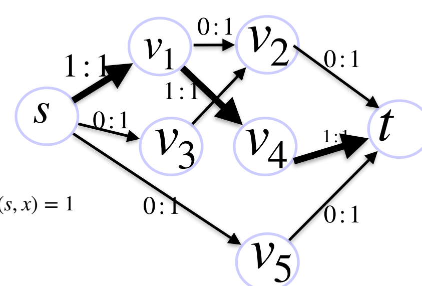




Max Flow in 0/1 Network

- A 0/1 network is a directed graph G(V,E), where there are given special vertices $s,t \in V$, and the capacity of every edge is 1. (instead of $c(u,v)$)
- A flow is **legal** if
  - for every edge $(u,v) \in E$ we are given the flow $f(u,v)$ across the edge (u,v).
  - $0 \leq f(u,v) \leq 1$. (capacity constrains)
  - For every vertex $v \in V - \{s,t\}$ we have
  - $\sum_{(w,v)\in E} f(w,v) = \sum_{(v,x)\in E} f(v,x)$ Flow conservation
- The value of the flow is $|f| := \sum_{(s,x)\in E} f(s,x)$ flow from s. This is the value we want to **maximize**.
- The gaol is to maximize the value of the flow.
- The matching problem is a special case of this problem.

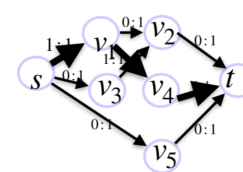
(in this example) $|f| := \sum_{(s,x)\in E} f(s,x) = 1$

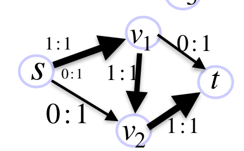


Max Flow in 0/1 Network

- A 0/1 network is a directed graph G(V,E), where there are given special vertices $s,t \in V$, and the capacity of every edge is 1.
- A flow is **legal** if
  - for every edge $(u,v) \in E$ we are given the flow $f(u,v)$ across the edge (u,v).
  - $0 \leq f(u,v) \leq 1$ Capacity constrains
  - For every vertex $v \in V - \{s,t\}$ we have
  - $\sum_{(w,v)\in E} f(w,v) = \sum_{(v,x)\in E} f(v,x)$ Flow conservation
- The value of the flow is $\sum_{(s,x)\in E} f(s,x)$ flow from s. This is the value we want to **maximize**.

- When we solve this problem using LP, we might find solutions that are non-integers.
- We can use ILP. Sometimes very efficient. Sometimes very slow.
- Ford-Fulkerson Algorithm: A sequence of iteration, at each, the value of the flow, |f| will be increased by 1.
- Under this algorithm, the flow across every edge is either 0 or 1. (but never 0.5)
- A greedy approach would be: Find a path $s \rightsquigarrow t$ of edges that carry zero flow. Increase the flow along this path, and repeat.
- This approach might not work (we saw a similar example in matching). Heavy edges carry flow.



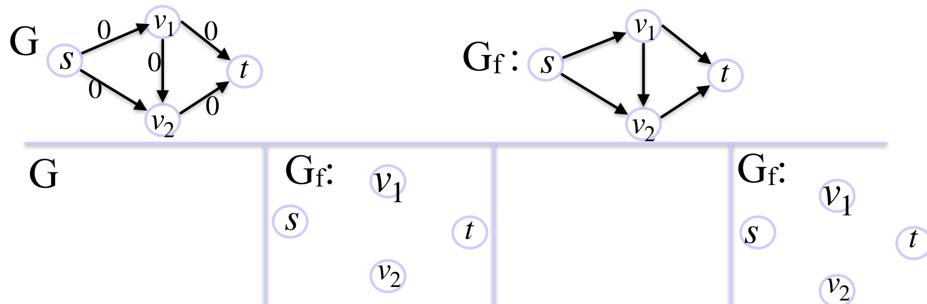


Ford-Fulkerson Algorithm: Assume that some (legit) flow f is given.
Create a new graph $G_f(V,E_f)$. In the textbook, it is called the residual graph.

- if f(u,v)=0 then $(u,v) \in E_f$.
- if f(u,v)=1 then $(v,u) \in E_f$. (that is, reverse the direction of the edges that carry flow.)

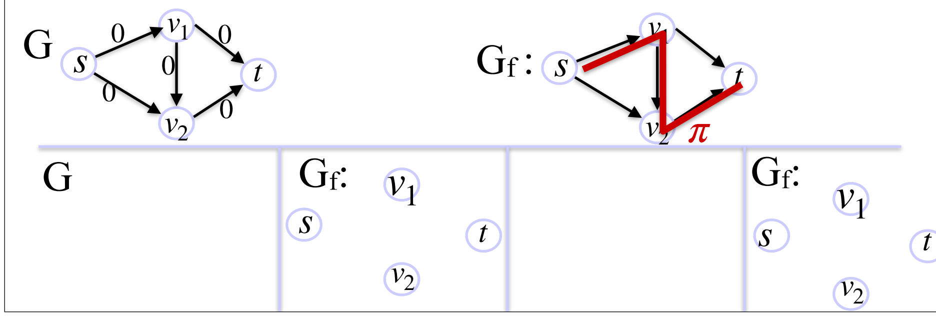
Ford-Fulkerson Algorithm:

1. Assume that some 0/1 (legit) flow f is given.
2. Create a new graph $G_f(V, E_f)$. In the textbook, it is called the **residual network**.
   For every edge $(u, v) \in E$
   - if f(u,v)=0 then insert (u, v) .
   - if f(u,v)=1 then insert $(\mathbf{v}, \mathbf{u})$ into $\mathbf{E_f}$. (that is, **reverse** the direction of the edges that carry flow.)
3. Find a path $\pi : s \rightsquigarrow t$ in the residual network G_f. If no path exists, |f| is maximum. **Exit**
4. Increase by 1 the flow along π as follows:
   For every edge $(u, v) \in \pi$
   - If $(u, v) \in E$ (edge not reversed) then $f(u, v) = f(u, v) + 1$
   - If $(\mathbf{v}, \mathbf{u}) \in E$ (edge reversed) then $f(u, v) = f(u, v) - \mathbf{1}$ // cancel the flow $1 \to 0$
5. The addition of the 1 to the flow along the edge of π increases $|f|$ by 1.



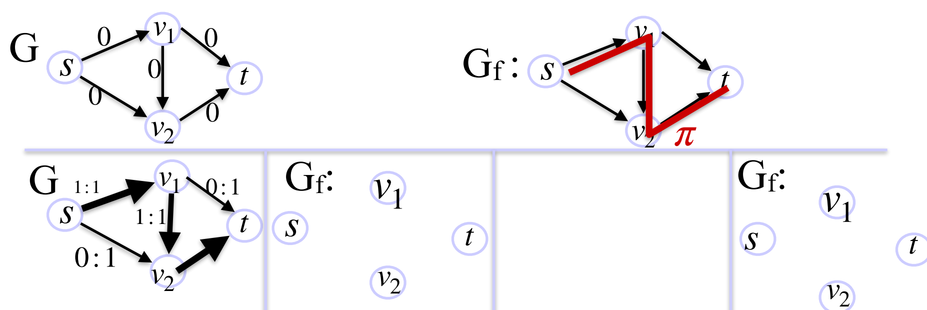
Ford-Fulkerson Algorithm:

1. Assume that some 0/1 (legit) flow f is given.
2. Create a new graph $G_f(V, E_f)$. In the textbook, it is called the **residual network**.
   For every edge $(u, v) \in E$
   - if f(u,v)=0 then insert (u, v) .
   - if f(u,v)=1 then insert $(\mathbf{v}, \mathbf{u})$ into $\mathbf{E_f}$. (that is, **reverse** the direction of the edges that carry flow.)
3. Find a path $\pi : s \rightsquigarrow t$ in the residual network G_f. If no path exists, |f| is maximum. **Exit**
4. Increase by 1 the flow along π as follows:
   For every edge $(u, v) \in \pi$
   - If $(u, v) \in E$ (edge not reversed) then $f(u, v) = f(u, v) + 1$
   - If $(\mathbf{v}, \mathbf{u}) \in E$ (edge reversed) then $f(u, v) = f(u, v) - \mathbf{1}$ // cancel the flow $1 \to 0$
5. The addition of the 1 to the flow along the edge of π increases $|f|$ by 1.



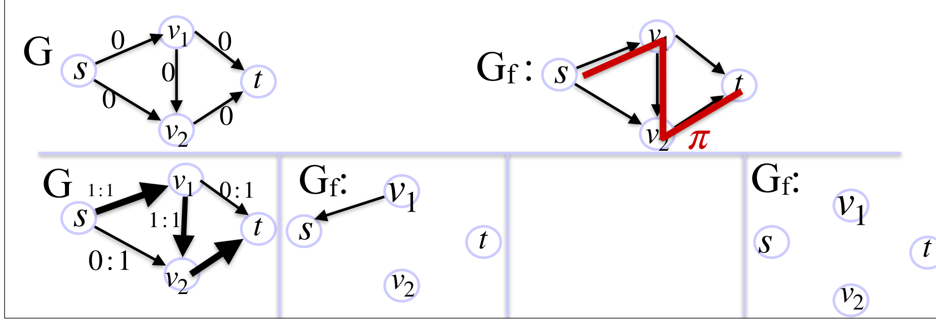
Ford-Fulkerson Algorithm:

1. Assume that some 0/1 (legit) flow f is given.
2. Create a new graph $G_f(V, E_f)$. In the textbook, it is called the **residual network**.
   For every edge $(u, v) \in E$
   - if f(u,v)=0 then insert (u, v) .
   - if f(u,v)=1 then insert $(\mathbf{v}, \mathbf{u})$ into $\mathbf{E_f}$. (that is, **reverse** the direction of the edges that carry flow.)
3. Find a path $\pi : s \rightsquigarrow t$ in the residual network G_f. If no path exists, |f| is maximum. **Exit**
4. Increase by 1 the flow along π as follows:
   For every edge $(u, v) \in \pi$
   - If $(u, v) \in E$ (edge not reversed) then $f(u, v) = f(u, v) + 1$
   - If $(\mathbf{v}, \mathbf{u}) \in E$ (edge reversed) then $f(u, v) = f(u, v) - \mathbf{1}$ // cancel the flow $1 \to 0$
5. The addition of the 1 to the flow along the edge of π increases $|f|$ by 1.



Ford-Fulkerson Algorithm:

1. Assume that some 0/1 (legit) flow f is given.
2. Create a new graph $G_f(V, E_f)$. In the textbook, it is called the **residual network**.
   For every edge $(u, v) \in E$
   - if f(u,v)=0 then insert (u, v) .
   - if f(u,v)=1 then insert $(\mathbf{v}, \mathbf{u})$ into $\mathbf{E_f}$. (that is, **reverse** the direction of the edges that carry flow.)
3. Find a path $\pi : s \rightsquigarrow t$ in the residual network G_f. If no path exists, |f| is maximum. **Exit**
4. Increase by 1 the flow along π as follows:
   For every edge $(u, v) \in \pi$
   - If $(u, v) \in E$ (edge not reversed) then $f(u, v) = f(u, v) + 1$
   - If $(\mathbf{v}, \mathbf{u}) \in E$ (edge reversed) then $f(u, v) = f(u, v) - \mathbf{1}$ // cancel the flow $1 \to 0$
5. The addition of the 1 to the flow along the edge of π increases $|f|$ by 1.

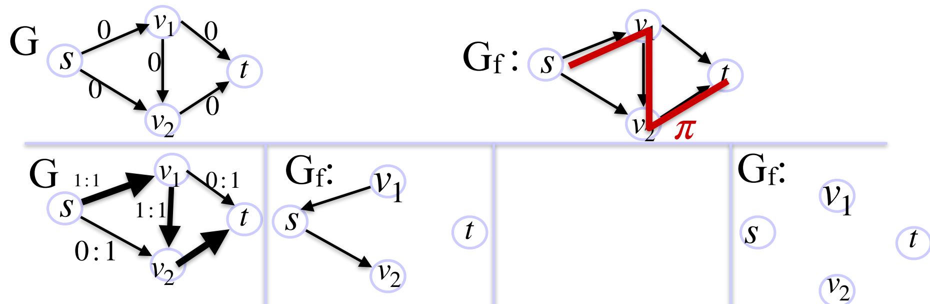


Ford-Fulkerson Algorithm:

1. Assume that some 0/1 (legit) flow f is given.
2. Create a new graph $G_f(V, E_f)$. In the textbook, it is called the **residual network**.

   For every edge $(u, v) \in E$
   - if f(u,v)=0 then insert (u, v) .
   - if f(u,v)=1 then insert $(\mathbf{v}, \mathbf{u})$ into $\mathbf{E_f}$. (that is, **reverse** the direction of the edges that carry flow.)
3. Find a path $\pi : s \rightsquigarrow t$ in the residual network G_f . If no path exists, |f| is maximum. **Exit**
4. Increase by 1 the flow along π as follows:

   For every edge $(u, v) \in \pi$
   - If $(u, v) \in E$ (edge not reversed) then $f(u, v) = f(u, v) + 1$
   - If $(\mathbf{v}, \mathbf{u}) \in E$ (edge reversed) then $f(u, v) = f(u, v) - \mathbf{1}$ // cancel the flow $1 \to 0$
5. The addition of the 1 to the flow along the edge of π increases $|f|$ by 1.

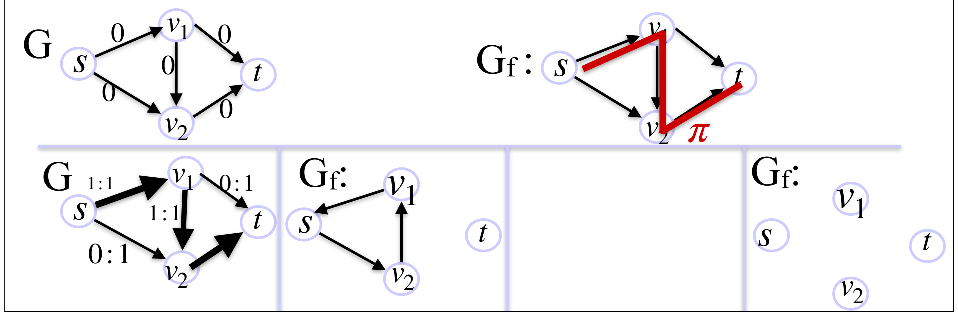


Ford-Fulkerson Algorithm:

1. Assume that some 0/1 (legit) flow f is given.
2. Create a new graph $G_f(V, E_f)$. In the textbook, it is called the **residual network**.

   For every edge $(u, v) \in E$
   - if f(u,v)=0 then insert (u, v) .
   - if f(u,v)=1 then insert $(\mathbf{v}, \mathbf{u})$ into $\mathbf{E_f}$. (that is, **reverse** the direction of the edges that carry flow.)
3. Find a path $\pi : s \rightsquigarrow t$ in the residual network G_f . If no path exists, |f| is maximum. **Exit**
4. Increase by 1 the flow along π as follows:

   For every edge $(u, v) \in \pi$
   - If $(u, v) \in E$ (edge not reversed) then $f(u, v) = f(u, v) + 1$
   - If $(\mathbf{v}, \mathbf{u}) \in E$ (edge reversed) then $f(u, v) = f(u, v) - \mathbf{1}$ // cancel the flow $1 \to 0$
5. The addition of the 1 to the flow along the edge of π increases $|f|$ by 1.

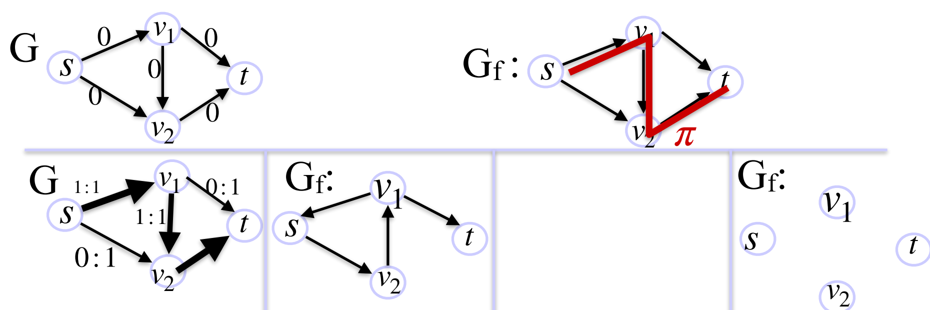


Ford-Fulkerson Algorithm:

1. Assume that some 0/1 (legit) flow f is given.
2. Create a new graph $G_f(V, E_f)$. In the textbook, it is called the **residual network**.

   For every edge $(u, v) \in E$
   - if f(u,v)=0 then insert (u, v) .
   - if f(u,v)=1 then insert $(\mathbf{v}, \mathbf{u})$ into $\mathbf{E_f}$. (that is, **reverse** the direction of the edges that carry flow.)
3. Find a path $\pi : s \rightsquigarrow t$ in the residual network G_f . If no path exists, |f| is maximum. **Exit**
4. Increase by 1 the flow along π as follows:

   For every edge $(u, v) \in \pi$
   - If $(u, v) \in E$ (edge not reversed) then $f(u, v) = f(u, v) + 1$
   - If $(\mathbf{v}, \mathbf{u}) \in E$ (edge reversed) then $f(u, v) = f(u, v) - \mathbf{1}$ // cancel the flow $1 \to 0$
5. The addition of the 1 to the flow along the edge of π increases $|f|$ by 1.

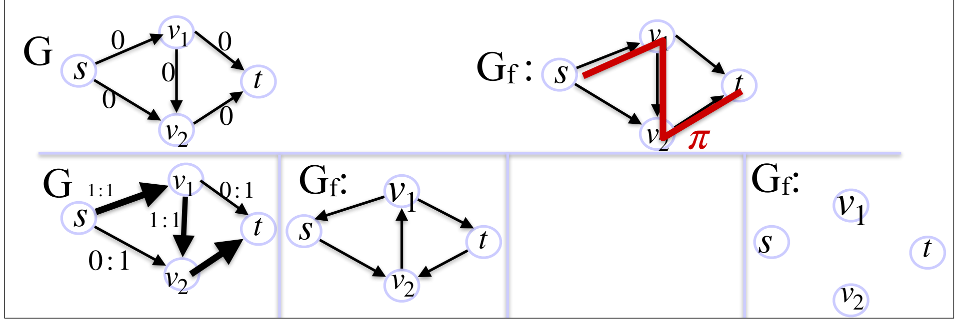


Ford-Fulkerson Algorithm:

1. Assume that some 0/1 (legit) flow f is given.
2. Create a new graph $G_f(V, E_f)$. In the textbook, it is called the **residual network**.

   For every edge $(u, v) \in E$
   - if f(u,v)=0 then insert (u, v) .
   - if f(u,v)=1 then insert $(\mathbf{v}, \mathbf{u})$ into $\mathbf{E_f}$. (that is, **reverse** the direction of the edges that carry flow.)
3. Find a path $\pi : s \rightsquigarrow t$ in the residual network G_f . If no path exists, |f| is maximum. **Exit**
4. Increase by 1 the flow along π as follows:

   For every edge $(u, v) \in \pi$
   - If $(u, v) \in E$ (edge not reversed) then $f(u, v) = f(u, v) + 1$
   - If $(\mathbf{v}, \mathbf{u}) \in E$ (edge reversed) then $f(u, v) = f(u, v) - \mathbf{1}$ // cancel the flow $1 \to 0$
5. The addition of the 1 to the flow along the edge of π increases $|f|$ by 1.

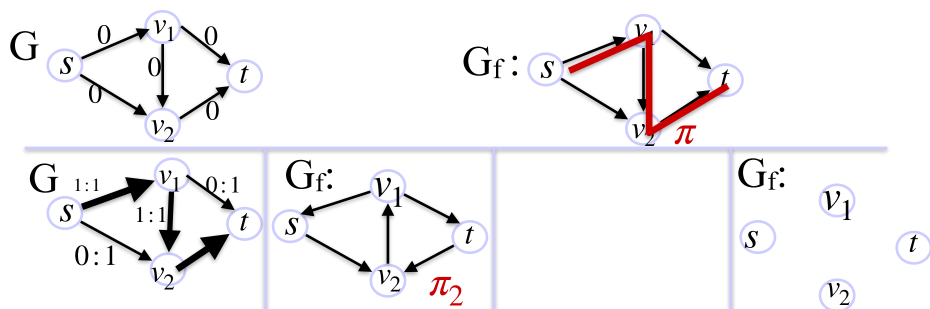


Ford-Fulkerson Algorithm:

1. Assume that some 0/1 (legit) flow f is given.
2. Create a new graph $G_f(V, E_f)$. In the textbook, it is called the **residual network**.

   For every edge $(u, v) \in E$
   - if f(u,v)=0 then insert (u, v) .
   - if f(u,v)=1 then insert $(\mathbf{v}, \mathbf{u})$ into $\mathbf{E_f}$. (that is, **reverse** the direction of the edges that carry flow.)
3. Find a path $\pi : s \rightsquigarrow t$ in the residual network G_f . If no path exists, |f| is maximum. **Exit**
4. Increase by 1 the flow along π as follows:

   For every edge $(u, v) \in \pi$
   - If $(u, v) \in E$ (edge not reversed) then $f(u, v) = f(u, v) + 1$
   - If $(\mathbf{v}, \mathbf{u}) \in E$ (edge reversed) then $f(u, v) = f(u, v) - \mathbf{1}$ // cancel the flow 1 → 0
5. The addition of the 1 to the flow along the edge of π increases $|f|$ by 1.

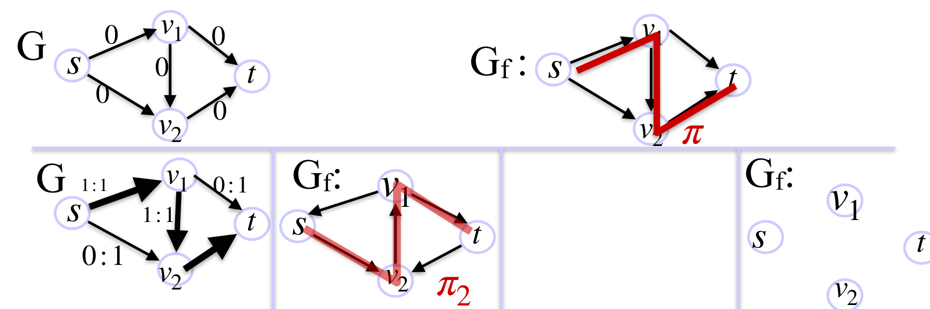


Ford-Fulkerson Algorithm:

1. Assume that some 0/1 (legit) flow f is given.
2. Create a new graph $G_f(V, E_f)$. In the textbook, it is called the **residual network**.

   For every edge $(u, v) \in E$
   - if f(u,v)=0 then insert (u, v) .
   - if f(u,v)=1 then insert $(\mathbf{v}, \mathbf{u})$ into $\mathbf{E_f}$. (that is, **reverse** the direction of the edges that carry flow.)
3. Find a path $\pi : s \rightsquigarrow t$ in the residual network G_f . If no path exists, |f| is maximum. **Exit**
4. Increase by 1 the flow along π as follows:

   For every edge $(u, v) \in \pi$
   - If $(u, v) \in E$ (edge not reversed) then $f(u, v) = f(u, v) + 1$
   - If $(\mathbf{v}, \mathbf{u}) \in E$ (edge reversed) then $f(u, v) = f(u, v) - \mathbf{1}$ // cancel the flow 1 → 0
5. The addition of the 1 to the flow along the edge of π increases $|f|$ by 1.

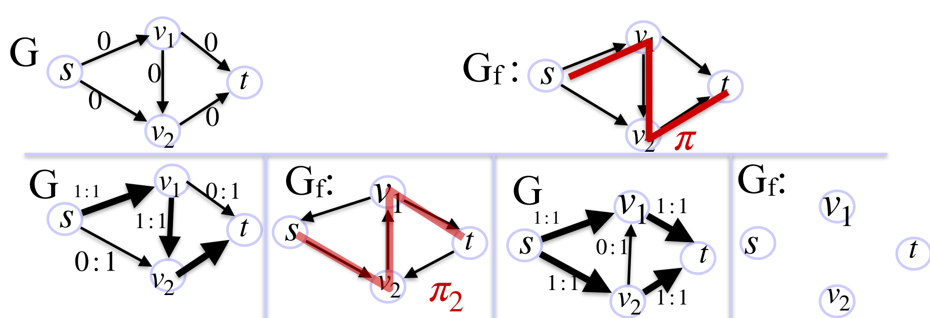


Ford-Fulkerson Algorithm:

1. Assume that some 0/1 (legit) flow f is given.
2. Create a new graph $G_f(V, E_f)$. In the textbook, it is called the **residual network**.

   For every edge $(u, v) \in E$
   - if f(u,v)=0 then insert (u, v) .
   - if f(u,v)=1 then insert $(\mathbf{v}, \mathbf{u})$ into $\mathbf{E_f}$. (that is, **reverse** the direction of the edges that carry flow.)
3. Find a path $\pi : s \rightsquigarrow t$ in the residual network G_f . If no path exists, |f| is maximum. **Exit**
4. Increase by 1 the flow along π as follows:

   For every edge $(u, v) \in \pi$
   - If $(u, v) \in E$ (edge not reversed) then $f(u, v) = f(u, v) + 1$
   - If $(\mathbf{v}, \mathbf{u}) \in E$ (edge reversed) then $f(u, v) = f(u, v) - \mathbf{1}$ // cancel the flow 1 → 0
5. The addition of the 1 to the flow along the edge of π increases $|f|$ by 1.

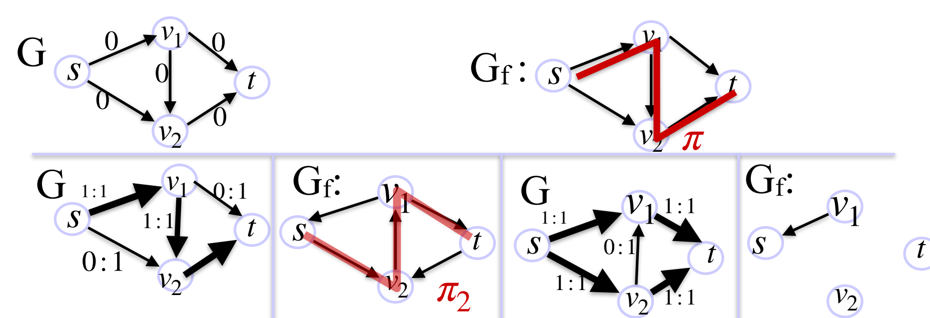


Ford-Fulkerson Algorithm:

1. Assume that some 0/1 (legit) flow f is given.
2. Create a new graph $G_f(V, E_f)$. In the textbook, it is called the **residual network**.

   For every edge $(u, v) \in E$
   - if f(u,v)=0 then insert (u, v) .
   - if f(u,v)=1 then insert $(\mathbf{v}, \mathbf{u})$ into $\mathbf{E_f}$. (that is, **reverse** the direction of the edges that carry flow.)
3. Find a path $\pi : s \rightsquigarrow t$ in the residual network G_f . If no path exists, |f| is maximum. **Exit**
4. Increase by 1 the flow along π as follows:

   For every edge $(u, v) \in \pi$
   - If $(u, v) \in E$ (edge not reversed) then $f(u, v) = f(u, v) + 1$
   - If $(\mathbf{v}, \mathbf{u}) \in E$ (edge reversed) then $f(u, v) = f(u, v) - \mathbf{1}$ // cancel the flow 1 → 0
5. The addition of the 1 to the flow along the edge of π increases $|f|$ by 1.

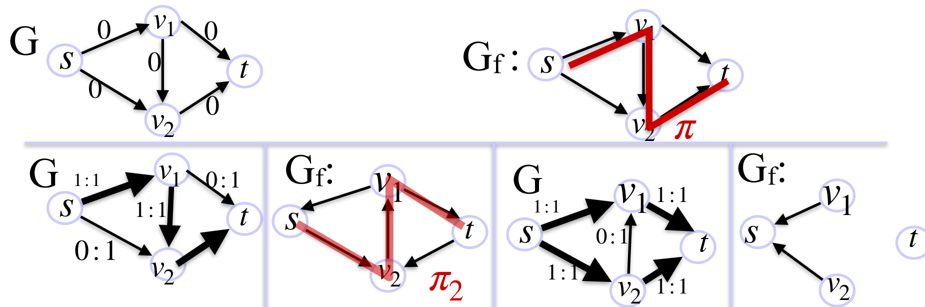


Ford-Fulkerson Algorithm:

1. Assume that some 0/1 (legit) flow f is given.
2. Create a new graph $G_f(V, E_f)$. In the textbook, it is called the **residual network**.

   For every edge $(u, v) \in E$
   - if f(u,v)=0 then insert (u, v) .
   - if f(u,v)=1 then insert $(\mathbf{v}, \mathbf{u})$ into $\mathbf{E_f}$. (that is, **reverse** the direction of the edges that carry flow.)
3. Find a path $\pi : s \rightsquigarrow t$ in the residual network G_f. If no path exists, |f| is maximum. **Exit**
4. Increase by 1 the flow along π as follows:

   For every edge $(u, v) \in \pi$
   - If $(u, v) \in E$ (edge not reversed) then $f(u, v) = f(u, v) + 1$
   - If $(\mathbf{v}, \mathbf{u}) \in E$ (edge reversed) then $f(u, v) = f(u, v) - \mathbf{1}$ // cancel the flow $1 \rightarrow 0$
5. The addition of the 1 to the flow along the edge of π increases $|f|$ by 1.

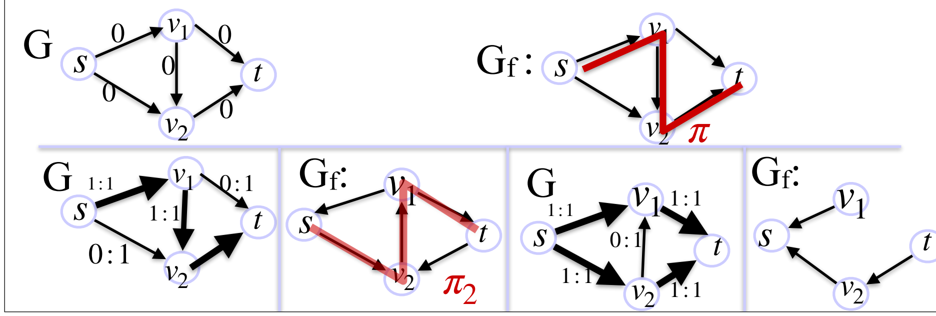


Ford-Fulkerson Algorithm:

1. Assume that some 0/1 (legit) flow f is given.
2. Create a new graph $G_f(V, E_f)$. In the textbook, it is called the **residual network**.

   For every edge $(u, v) \in E$
   - if f(u,v)=0 then insert (u, v) .
   - if f(u,v)=1 then insert $(\mathbf{v}, \mathbf{u})$ into $\mathbf{E_f}$. (that is, **reverse** the direction of the edges that carry flow.)
3. Find a path $\pi : s \rightsquigarrow t$ in the residual network G_f. If no path exists, |f| is maximum. **Exit**
4. Increase by 1 the flow along π as follows:

   For every edge $(u, v) \in \pi$
   - If $(u, v) \in E$ (edge not reversed) then $f(u, v) = f(u, v) + 1$
   - If $(\mathbf{v}, \mathbf{u}) \in E$ (edge reversed) then $f(u, v) = f(u, v) - \mathbf{1}$ // cancel the flow $1 \rightarrow 0$
5. The addition of the 1 to the flow along the edge of π increases $|f|$ by 1.

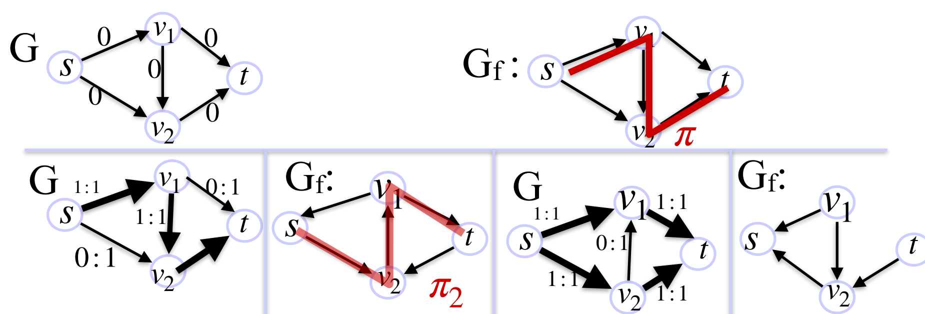


Ford-Fulkerson Algorithm:

1. Assume that some 0/1 (legit) flow f is given.
2. Create a new graph $G_f(V, E_f)$. In the textbook, it is called the **residual network**.

   For every edge $(u, v) \in E$
   - if f(u,v)=0 then insert (u, v) .
   - if f(u,v)=1 then insert $(\mathbf{v}, \mathbf{u})$ into $\mathbf{E_f}$. (that is, **reverse** the direction of the edges that carry flow.)
3. Find a path $\pi : s \rightsquigarrow t$ in the residual network G_f. If no path exists, |f| is maximum. **Exit**
4. Increase by 1 the flow along π as follows:

   For every edge $(u, v) \in \pi$
   - If $(u, v) \in E$ (edge not reversed) then $f(u, v) = f(u, v) + 1$
   - If $(\mathbf{v}, \mathbf{u}) \in E$ (edge reversed) then $f(u, v) = f(u, v) - \mathbf{1}$ // cancel the flow $1 \rightarrow 0$
5. The addition of the 1 to the flow along the edge of π increases $|f|$ by 1.

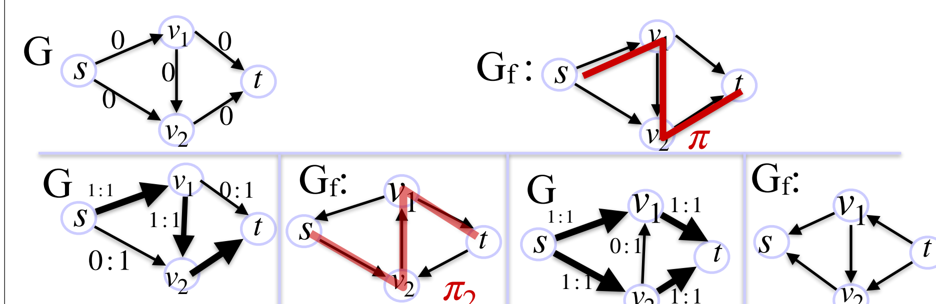


Ford-Fulkerson Algorithm:

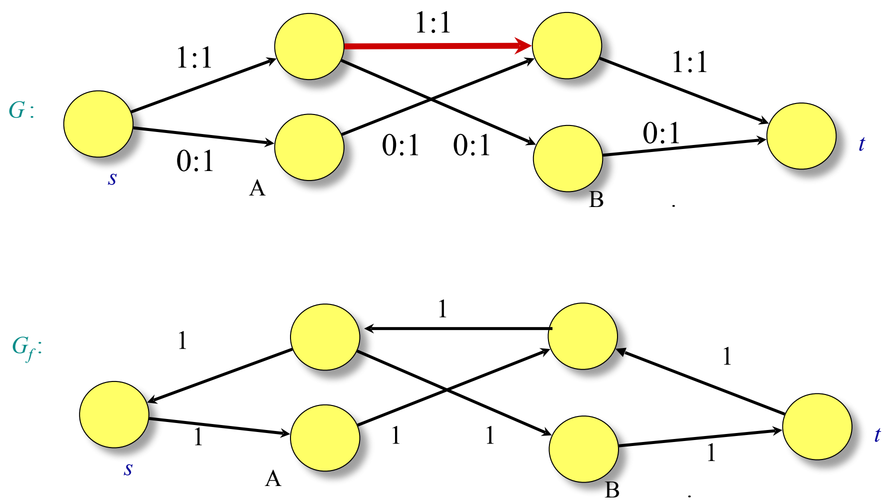
1. Assume that some 0/1 (legit) flow f is given.
2. Create a new graph $G_f(V, E_f)$. In the textbook, it is called the **residual network**.

   For every edge $(u, v) \in E$
   - if f(u,v)=0 then insert (u, v) .
   - if f(u,v)=1 then insert $(\mathbf{v}, \mathbf{u})$ into $\mathbf{E_f}$. (that is, **reverse** the direction of the edges that carry flow.)
3. Find a path $\pi : s \rightsquigarrow t$ in the residual network G_f. If no path exists, |f| is maximum. **Exit**
4. Increase by 1 the flow along π as follows:

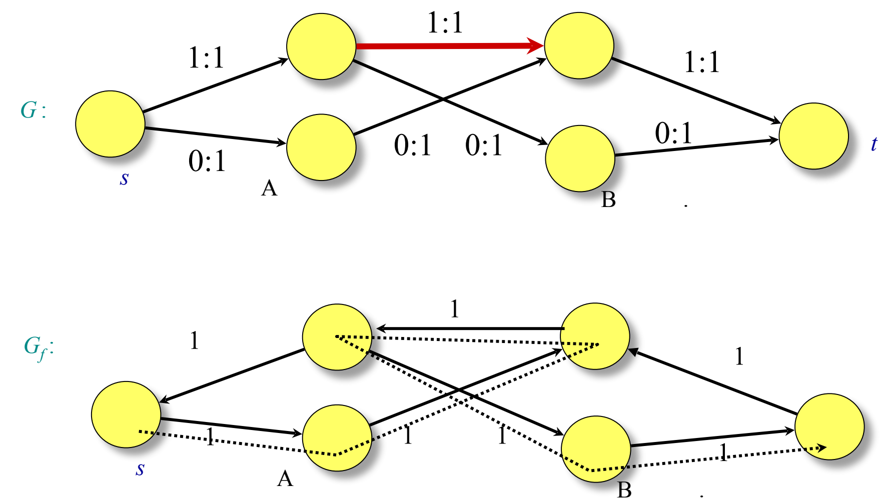
   For every edge $(u, v) \in \pi$
   - If $(u, v) \in E$ (edge not reversed) then $f(u, v) = f(u, v) + 1$
   - If $(\mathbf{v}, \mathbf{u}) \in E$ (edge reversed) then $f(u, v) = f(u, v) - \mathbf{1}$ // cancel the flow $1 \rightarrow 0$
5. The addition of the 1 to the flow along the edge of π increases $|f|$ by 1.



Example – maximum matching



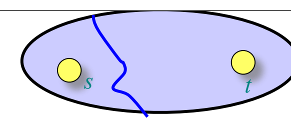
Example – maximum matching



Ford-Fulkerson max-flow algorithm

- Start: $f[u, v] \leftarrow 0$ for all $(u, v) \in E$
- While **(1) {**
  - construct G_f
  - if an augmenting path p in G_f exists then
    augment f //Any path would do
  - else exit }

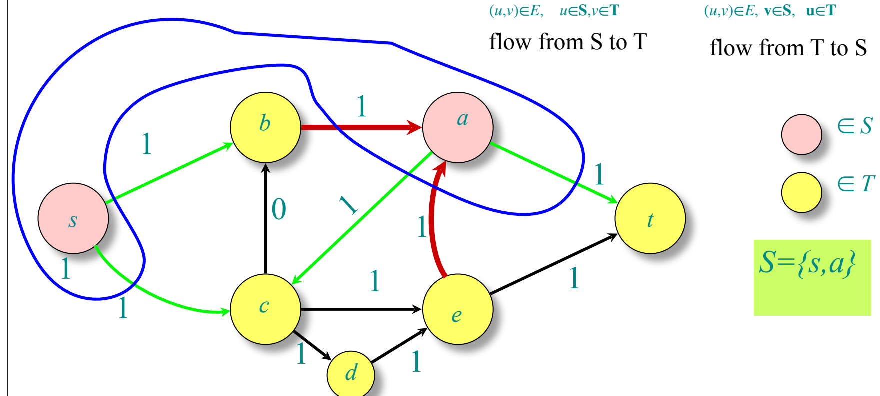
Cuts



Definitions. A ***cut*** $(\mathbf{S}, \mathbf{T})$ of a flow network $G=(V, E)$ is a partition of V such that $s \in \boldsymbol{S}$ and $t \in \boldsymbol{T}$.

If f is a flow on G, then the ***flow across the cut*** denoted $f(\mathbf{S}, \mathbf{T})$ is

$$f(\mathbf{S},\mathbf{T}) := \underbrace{\sum_{(u,v)\in E,\ u\in \mathbf{S}, v\in \mathbf{T}} f(u,v)}_{\text{flow from S to T}} - \underbrace{\sum_{(u,v)\in E,\ v\in \mathbf{S},\ u\in \mathbf{T}} f(u,v)}_{\text{flow from T to S}}$$



$S=\{s,a\}$

$f(\mathbf{S},\mathbf{T}) = f(s,b) + f(s,c) + f(a,t) + f(a,c) - f(b,a) - f(e,a)$

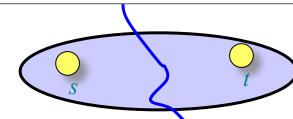
Lemma: Flow across the cut

Remember $|f| := \sum_{(s,v)\in E} f(s,v)$ That is, it is the flow leaving s

Lemma. For any flow f and any cut $(\mathbf{S}, \mathbf{T})$, we have $|f| = f(\mathbf{S},\mathbf{T})$ (*flow from T to S*).

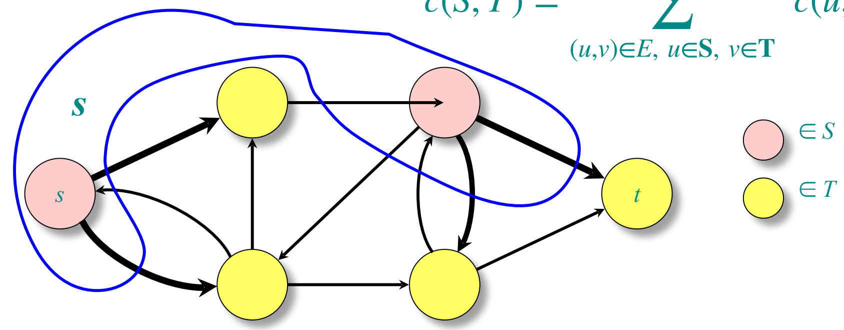
Proof: On whiteboard

Capacity of a cut



Definition. The ***capacity of a cut*** (S, T) is the number of edges across the cut

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



Upper bound on the maximum flow value

Theorem. The value of any flow no larger than the capacity of any cut: $|f| \le c(S,T)$.

$$
\begin{aligned}
|f| &= f(\mathbf{S},\mathbf{T}) \\
&= \sum_{(u,v)\in E,\ u\in\mathbf{S}, v\in\mathbf{T}} f(u,v) - \sum_{(u,v)\in E,\ v\in\mathbf{S},\ u\in\mathbf{T}} f(u,v) \\
&\le \sum_{(u,v)\in E,\ u\in\mathbf{S}, v\in\mathbf{T}} f(u,v) \\
&\le \sum_{(u,v)\in E,\ u\in\mathbf{S}, v\in\mathbf{T}} c(u,v) \qquad = \mathbf{C}(\mathbf{S},\mathbf{T})
\end{aligned}
$$

.

Max-flow, min-cut theorem

Theorem. The following are equivalent:

1. $|f| = c(S, T)$ for some cut (S, T).
2. f is a maximum flow.
3. f admits no augmenting paths.

Max-flow, min-cut theorem

Theorem. The following are equivalent:

1. $|f| = c(S, T)$ for some cut (S, T).
2. f is a maximum flow.
3. f admits no augmenting paths.

Proof.

(1) ⇒ (2): Since $|f| \le c(S, T)$ for any cut (S, T) (by the theorem from a few slides back), the assumption that $|f| = c(S, T)$ implies that f is a maximum flow.

(2) ⇒ (3): If there were an augmenting path, the flow value could be increased, contradicting the maximality of f.

Max-flow, min-cut theorem

Theorem. The following are equivalent:

1. $|f| = c(S, T)$ for some cut (S, T).

min-cut

2. f is a maximum flow.
3. f admits no augmenting paths.

Proof.

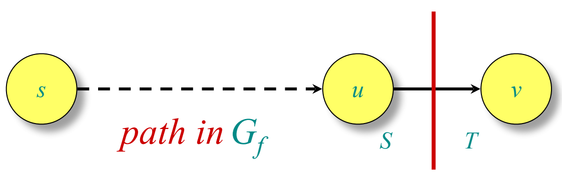
(1) ⇒ (2): Since $|f| \le c(S, T)$ for any cut (S, T) (by the theorem from a few slides back), the assumption that $|f| = c(S, T)$ implies that f is a maximum flow.

(2) ⇒ (3): If there were an augmenting path, the flow value could be increased, contradicting the maximality of f.

(3) ⇒ (1): Define $S = \{v \in V \mid$ there exists a path in G_f from s to $v\}$,

Let $T = V - S$. Since f admits no augmenting paths, there is no path from s to t in G_f.
Hence, $s \in S$ and $t \notin S$, So $t \in T$.

Thus (S, T) is a cut.



Consider edge (u,v) $u \in S$, $v \in T$. Observe that f(u,v)=1, since if it was zero, we would add (u,v) to the path.

Thus, $f(u, v) = c(u, v)$

Summing over all $u \in S$ and $v \in T$ yields $f(S, T) = c(S, T)$, and since $|f| = f(S, T)$, the theorem follows.