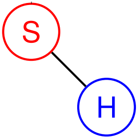
CS 170: Algorithms

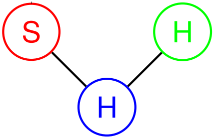
CS 170: Algorithms

CS 170: Algorithms

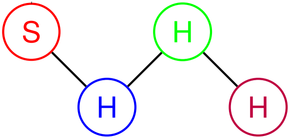
CS 170: Algorithms

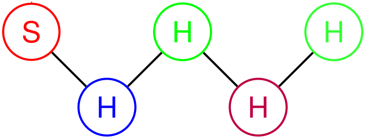


CS 170: Algorithms

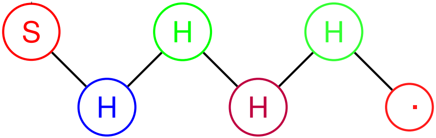


CS 170: Algorithms

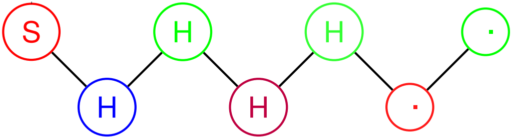



S
H
H
H
H

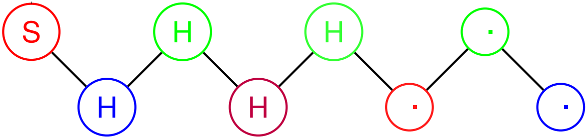
CS 170: Algorithms



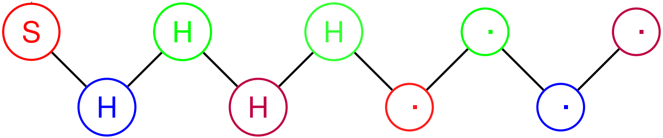
CS 170: Algorithms



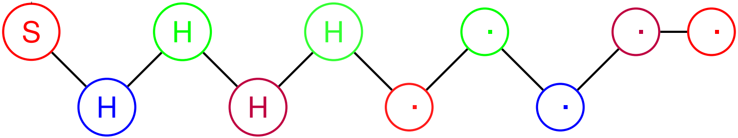
CS 170: Algorithms



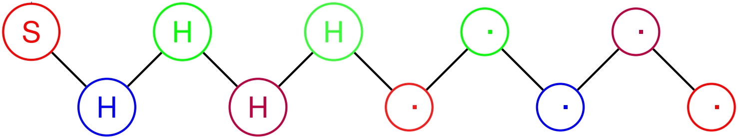
CS 170: Algorithms



CS 170: Algorithms



CS 170: Algorithms



Lecture in a minute

Maximum flow.
  "Greedy" augment path...
    Except reverse old decisions ..
    Reverse residual capacities.
  : Optimality?
    No augmenting path $\implies$
     $s-t$ cut size = flow value.
   Find flow and $s-t$ cut with equal value!

Lecture in a minute

Maximum flow.
- "Greedy" augment path...
  - Except reverse old decisions ..
  - Reverse residual capacities.
- : Optimality?
  - No augmenting path $\implies$
    - $s-t$ cut size = flow value.
  - Find flow and $s-t$ cut with equal value!

Maximum Matching.
- $G=(V,E)$, find subset of one-to-one matches.
- Reduction to max flow.
- Augmenting Alternating Path
- Algorithm $\equiv$ Simplex.

Maximum flow

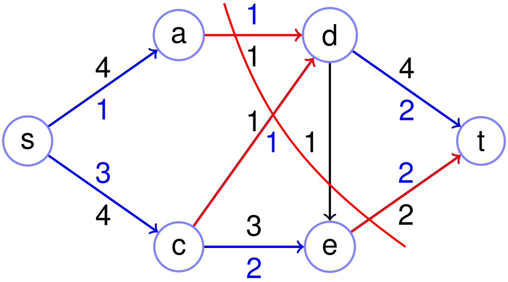
Flow network $G = (V, E)$, source s, sink $t \in V$, capacities $c_e > 0$.

Maximum flow

Flow network $G = (V, E)$, source s, sink $t \in V$, capacities $c_e > 0$.

Maximum flow

Flow network $G = (V, E)$, source s, sink $t \in V$, capacities $c_e > 0$.



Find Flow: f_e

1. $0 \le f_e \le c_e$. "Capacity constraints." $3 = f_{s,c} \le c_{s,c} = 4$.
2. If u is not s or t
   $\sum_{(w,u) \in E} f_{wu} = \sum_{(u,w) \in E} f_{uw}$. $3 = f_{s,c} = f_{c,d} + f_{c,e} = 2 + 1$.

Maximize: $\text{size}(f) = \sum_{(s,u) \in E} f_{su}$. $f_{sa} + f_{sc} = 1 + 3 = 4$

Optimal? $c_{ad} + c_{cd} + c_{et} = 1 + 1 + 2 = 4$.

S-T cut.

An $s - t$ cut is a partition of V into S and T where $s \in S$ and $t \in T$. Its capacity is the total capacity of edges from S to T.

Do you know the definition?

Find Flow: f_e

1. $0 \leq f_e \leq c_e$. "Capacity constraints."
2. If u is not s or t
   $\sum_{(w,u) \in E} f_{wu} = \sum_{(u,w) \in E} f_{uw}$.

Do you know the definition?

Find Flow: f_e

1. $0 \leq f_e \leq c_e$. "Capacity constraints."
2. If u is not s or t
   $\sum_{(w,u) \in E} f_{wu} = \sum_{(u,w) \in E} f_{uw}$.

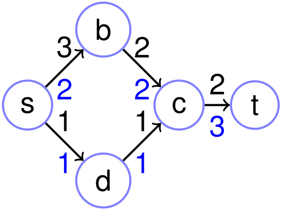
Valid or Invalid?

Do you know the definition?

Find Flow: f_e

1. $0 \leq f_e \leq c_e$. "Capacity constraints."
2. If u is not s or t
   $\sum_{(w,u)\in E} f_{wu} = \sum_{(u,w)\in E} f_{uw}$.

Valid or Invalid?

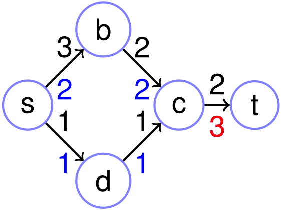


Do you know the definition?

Find Flow: f_e

1. $0 \leq f_e \leq c_e$. “Capacity constraints.”
2. If u is not s or t
   $\sum_{(w,u) \in E} f_{wu} = \sum_{(u,w) \in E} f_{uw}$.

Valid or Invalid?

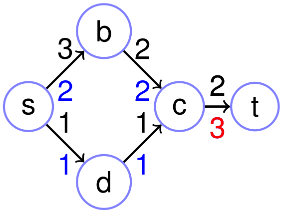


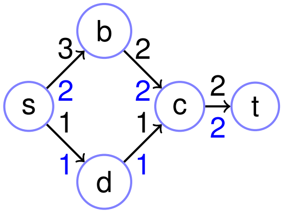
Do you know the definition?

Find Flow: f_e

1. $0 \leq f_e \leq c_e$. "Capacity constraints."
2. If u is not s or t
   $\sum_{(w,u) \in E} f_{wu} = \sum_{(u,w) \in E} f_{uw}$.

Valid or Invalid?



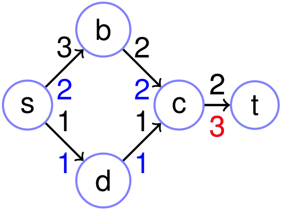


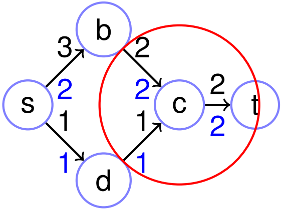
Do you know the definition?

Find Flow: f_e

1. $0 \le f_e \le c_e$. "Capacity constraints."
2. If u is not s or t
   $\sum_{(w,u)\in E} f_{wu} = \sum_{(u,w)\in E} f_{uw}$.

Valid or Invalid?





$2+1 \neq 2$

Algorithms.

FindFlow: f_e

1. $0 \leq f_e \leq c_e$. "Capacity constraints."
2. $\sum_{(w,u)\in E} f_{wu} = \sum_{(u,w)\in E} f_{uw}$.
3. maximize $\sum_{su} f_{su}$.

Algorithms.

FindFlow: f_e

1. $0 \leq f_e \leq c_e$. "Capacity constraints."
2. $\sum_{(w,u)\in E} f_{wu} = \sum_{(u,w)\in E} f_{uw}$.
3. maximize $\sum_{su} f_{su}$.

Linear program!

Algorithms.

FindFlow: f_e

1. $0 \leq f_e \leq c_e$. “Capacity constraints.”
2. $\sum_{(w,u)\in E} f_{wu} = \sum_{(u,w)\in E} f_{uw}$.
3. maximize $\sum_{su} f_{su}$.

Linear program!

Variables f_e, linear constraints, linear optimization function.

Algorithms.

FindFlow: f_e

1. $0 \leq f_e \leq c_e$. "Capacity constraints."
2. $\sum_{(w,u)\in E} f_{wu} = \sum_{(u,w)\in E} f_{uw}$.
3. maximize $\sum_{su} f_{su}$.

Linear program!

Variables f_e, linear constraints, linear optimization function.

Cool!

Algorithms.

FindFlow: f_e

1. $0 \leq f_e \leq c_e$. "Capacity constraints."
2. $\sum_{(w,u)\in E} f_{wu} = \sum_{(u,w)\in E} f_{uw}$.
3. maximize $\sum_{su} f_{su}$.

Linear program!

Variables f_e, linear constraints, linear optimization function.

Cool!

Note...

Algorithms.

FindFlow: f_e

1. $0 \leq f_e \leq c_e$. "Capacity constraints."
2. $\sum_{(w,u) \in E} f_{wu} = \sum_{(u,w) \in E} f_{uw}$.
3. maximize $\sum_{su} f_{su}$.

Linear program!

Variables f_e, linear constraints, linear optimization function.

Cool!

Note...

Integer? (Given integer capacities.)

Algorithms.

FindFlow: f_e

1. $0 \leq f_e \leq c_e$. "Capacity constraints."
2. $\sum_{(w,u)\in E} f_{wu} = \sum_{(u,w)\in E} f_{uw}$.
3. maximize $\sum_{su} f_{su}$.

Linear program!

Variables f_e, linear constraints, linear optimization function.

Cool!

Note...

Integer? (Given integer capacities.)

Yes. There is an integer vertex solution!

Algorithms.

FindFlow: f_e

1. $0 \leq f_e \leq c_e$. "Capacity constraints."
2. $\sum_{(w,u)\in E} f_{wu} = \sum_{(u,w)\in E} f_{uw}$.
3. maximize $\sum_{su} f_{su}$.

Linear program!

Variables f_e, linear constraints, linear optimization function.

Cool!

Note...

Integer? (Given integer capacities.)

Yes. There is an integer vertex solution!

Constraint matrix has every subdeterminant being 1, 0, -1.

Algorithms.

FindFlow: f_e

1. $0 \leq f_e \leq c_e$. "Capacity constraints."
2. $\sum_{(w,u)\in E} f_{wu} = \sum_{(u,w)\in E} f_{uw}$.
3. maximize $\sum_{su} f_{su}$.

Linear program!

Variables f_e, linear constraints, linear optimization function.

Cool!

Note...

Integer? (Given integer capacities.)

Yes. There is an integer vertex solution!

Constraint matrix has every subdeterminant being 1, 0, -1.

Vertex solution to linear program must be integral!

Ford-Fulkerson.

"Simplex" method.

Ford-Fulkerson.

"Simplex" method.

Find s to t path with remaining capacity.

Ford-Fulkerson.

"Simplex" method.

Find s to t path with remaining capacity.

Add to flow variables along path.

Ford-Fulkerson.

"Simplex" method.

Find s to t path with remaining capacity.

Add to flow variables along path.

Update remaining capacity.

Ford-Fulkerson.

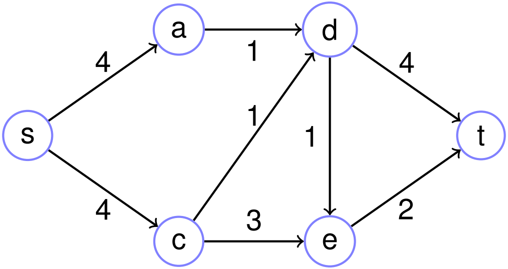
"Simplex" method.

Find *s* to *t* path with remaining capacity.

Add to flow variables along path.

Update remaining capacity.

Repeat.



Ford-Fulkerson.

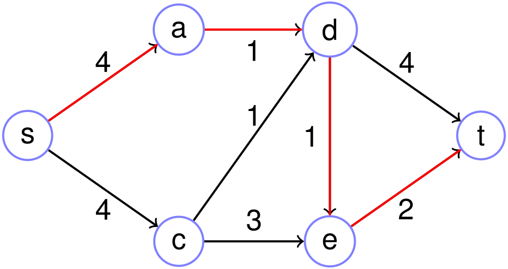
"Simplex" method.

Find s to t path with remaining capacity.

Add to flow variables along path.

Update remaining capacity.

Repeat.



Ford-Fulkerson.

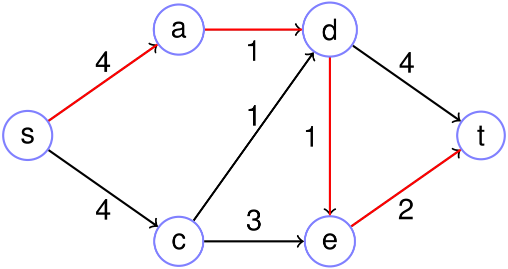
"Simplex" method.

Find s to t path with remaining capacity.

Add to flow variables along path.

Update remaining capacity.

Repeat.



Ford-Fulkerson.

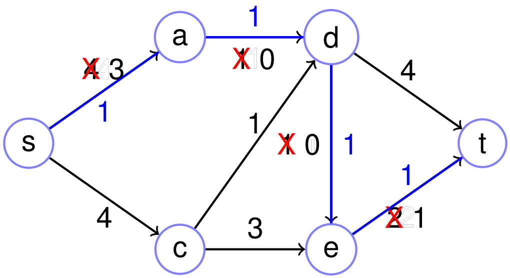
"Simplex" method.

Find s to t path with remaining capacity.

Add to flow variables along path.

Update remaining capacity.

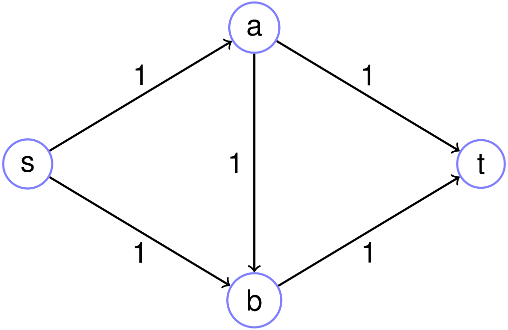
Repeat.



Residual Capacity.

Find s to t path with remaining capacity.
Add to flow along path.
Update remaining capacity.

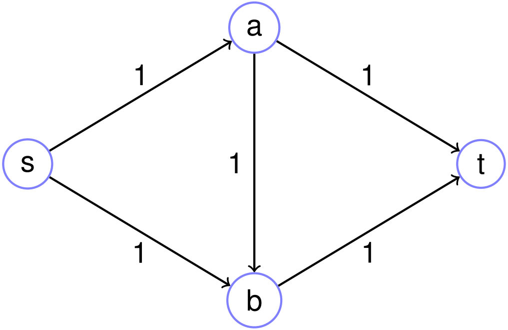
Repeat.



Residual Capacity.

Find s to t path with remaining capacity.
Add to flow along path.
Update remaining capacity.

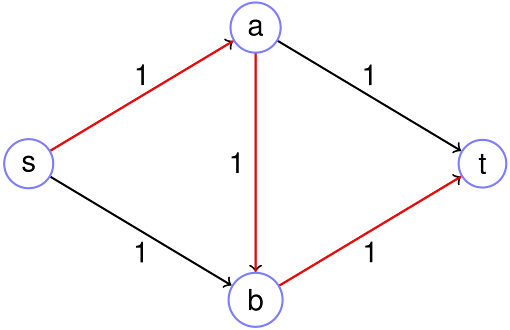
Repeat.



Residual Capacity.

Find s to t path with remaining capacity.
Add to flow along path.
Update remaining capacity.

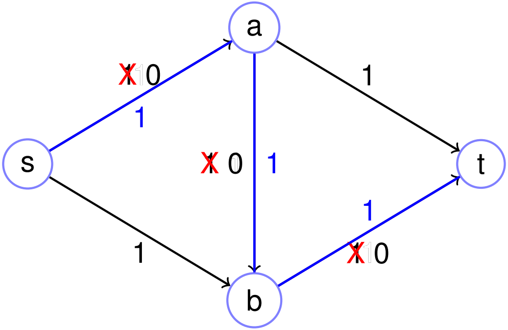
Repeat.



Residual Capacity.

Find s to t path with remaining capacity.
Add to flow along path.
Update remaining capacity.

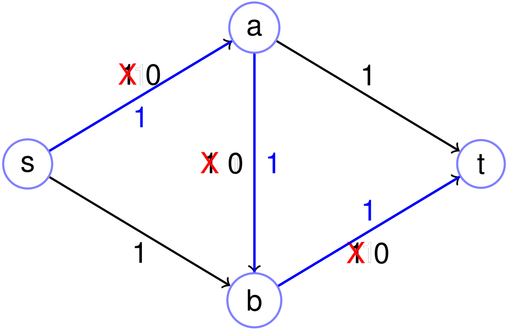
Repeat.



Residual Capacity.

Find s to t path with remaining capacity.
Add to flow along path.
Update remaining capacity.

Repeat.

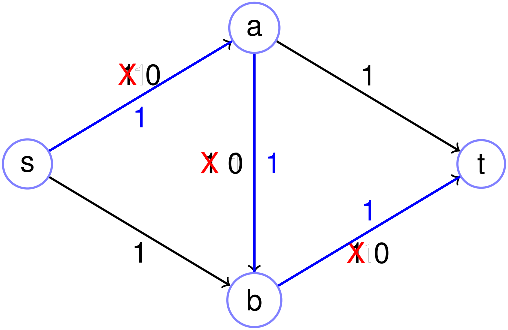


No remaining path.

Residual Capacity.

Find s to t path with remaining capacity.
Add to flow along path.
Update remaining capacity.

Repeat.

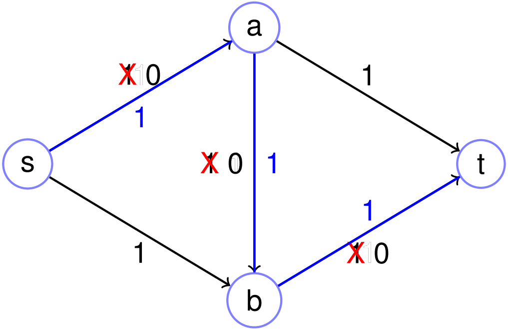


No remaining path. Uh oh! Optimal is 2! (At most 2 due to cut.)

Residual Capacity.

Find *s* to *t* path with remaining capacity.
Add to flow along path.
Update remaining capacity.

Repeat.

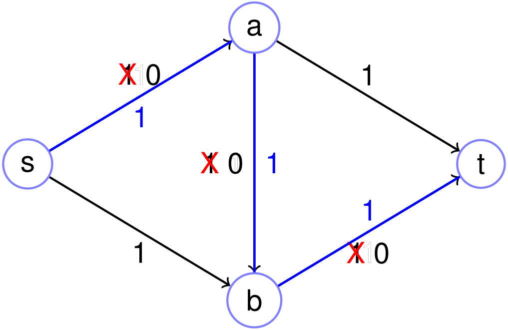


No remaining path. Uh oh! Optimal is 2! (At most 2 due to cut.)

Residual Capacity.

Find s to t path with remaining capacity.
Add to flow along path. Or reduce flow on reverse edge.
Update remaining capacity.

Repeat.

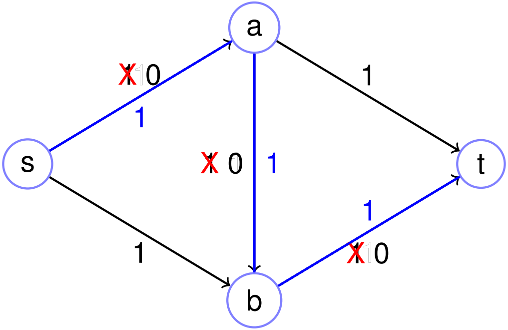


No remaining path. Uh oh! Optimal is 2! (At most 2 due to cut.)

Residual Capacity.

Find s to t path with remaining capacity.
Add to flow along path. Or reduce flow on reverse edge.
Update remaining capacity.
Reduce $r_e = c_e - f_e$

Repeat.

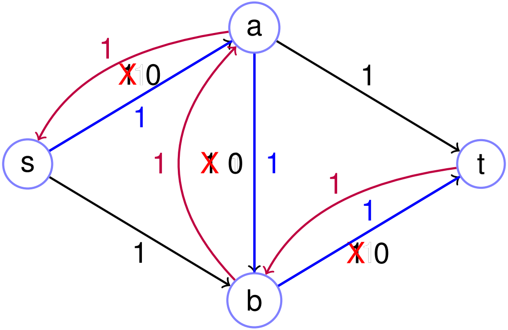


No remaining path. Uh oh! Optimal is 2! (At most 2 due to cut.)

Residual Capacity.

Find s to t path with remaining capacity.
Add to flow along path. Or reduce flow on reverse edge.
Update remaining capacity.
  Reduce $r_e = c_e - f_e$
  and add reverse $r_{uv} = f_{vu}$

Repeat.

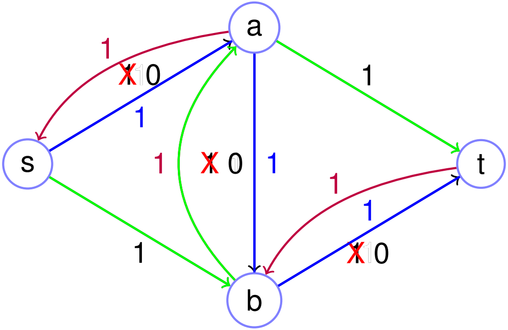


No remaining path. Uh oh! Optimal is 2! (At most 2 due to cut.)
Add reverse arcs to indicate "reverse" capacity.

Residual Capacity.

Find s to t path with remaining capacity.
Add to flow along path. Or reduce flow on reverse edge.
Update remaining capacity.
 Reduce $r_e = c_e - f_e$
 and add reverse $r_{uv} = f_{vu}$

Repeat.

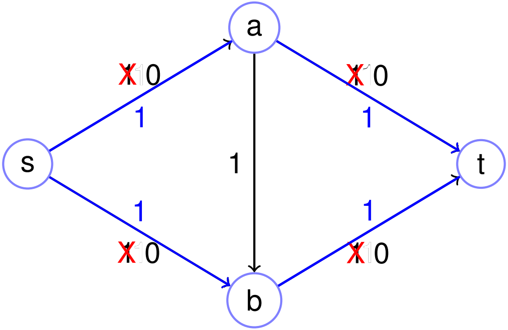


No remaining path. Uh oh! Optimal is 2! (At most 2 due to cut.)
Add reverse arcs to indicate “reverse” capacity.

Residual Capacity.

Find s to t path with remaining capacity.
Add to flow along path. Or reduce flow on reverse edge.
Update remaining capacity.
Reduce $r_e = c_e - f_e$
and add reverse $r_{uv} = f_{vu}$

Repeat.



No remaining path. Uh oh! Optimal is 2! (At most 2 due to cut.)
Add reverse arcs to indicate “reverse” capacity.

Bigger Example.

Find s to t path with remaining capacity.

Add to flow along path.

Update residual capacities: : $r_e = c_e - f_e; r_{uv} = f_{vu}$.

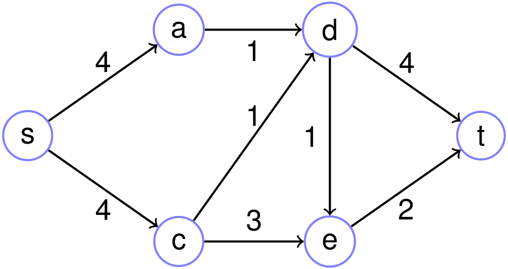
Bigger Example.

Find s to t path with remaining capacity.

Add to flow along path.

Update residual capacities: : $r_e = c_e - f_e; r_{uv} = f_{vu}$.

Repeat.



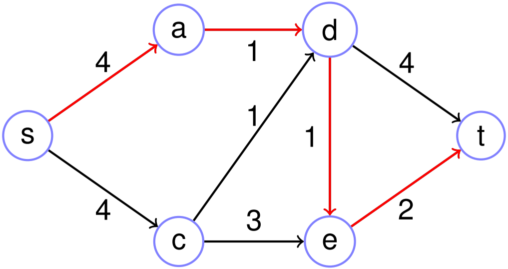
Bigger Example.

Find s to t path with remaining capacity.

Add to flow along path.

Update residual capacities: : $r_e = c_e - f_e; r_{uv} = f_{vu}$.

Repeat.



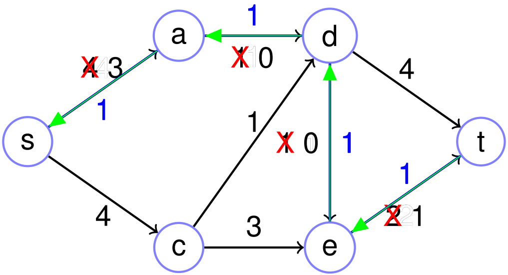
Bigger Example.

Find s to t path with remaining capacity.

Add to flow along path.

Update residual capacities: : $r_e = c_e - f_e; r_{uv} = f_{vu}$.

Repeat.



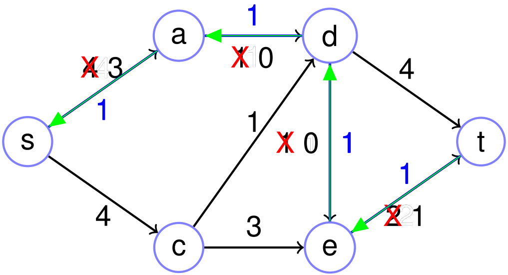
Bigger Example.

Find s to t path with remaining capacity.

Add to flow along path.

Update residual capacities: : $r_e = c_e - f_e; r_{uv} = f_{vu}$.

Repeat.



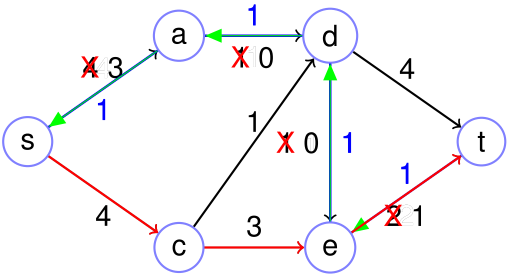
Bigger Example.

Find s to t path with remaining capacity.

Add to flow along path.

Update residual capacities: : $r_e = c_e - f_e; r_{uv} = f_{vu}$.

Repeat.



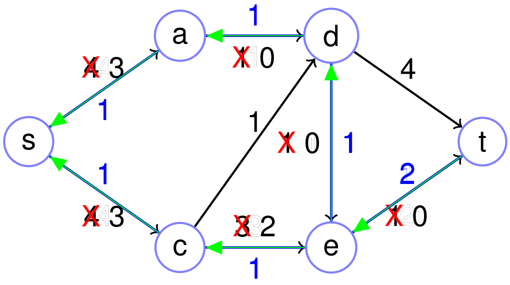
Bigger Example.

Find s to t path with remaining capacity.

Add to flow along path.

Update residual capacities: : $r_e = c_e - f_e; r_{uv} = f_{vu}$.

Repeat.



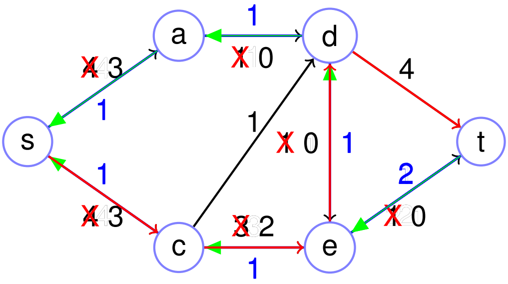
Bigger Example.

Find s to t path with remaining capacity.

Add to flow along path.

Update residual capacities: : $r_e = c_e - f_e; r_{uv} = f_{vu}$.

Repeat.



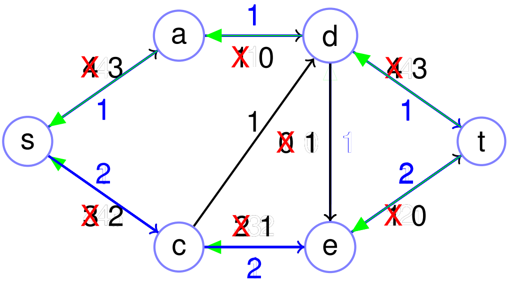
Bigger Example.

Find s to t path with remaining capacity.

Add to flow along path.

Update residual capacities: : $r_e = c_e - f_e; r_{uv} = f_{vu}$.

Repeat.



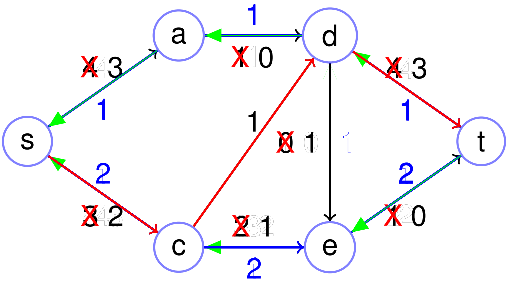
Bigger Example.

Find s to t path with remaining capacity.

Add to flow along path.

Update residual capacities: : $r_e = c_e - f_e; r_{uv} = f_{vu}$.

Repeat.



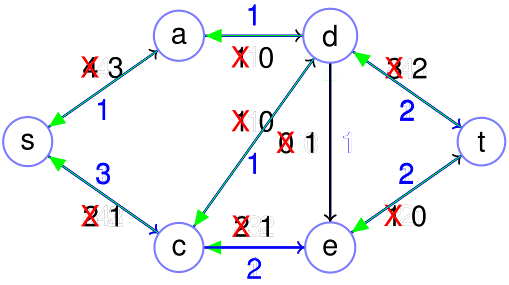
Bigger Example.

Find s to t path with remaining capacity.

Add to flow along path.

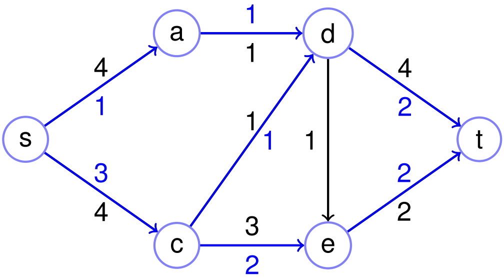
Update residual capacities: : $r_e = c_e - f_e; r_{uv} = f_{vu}$.

Repeat.



Check Result...

Check Result...

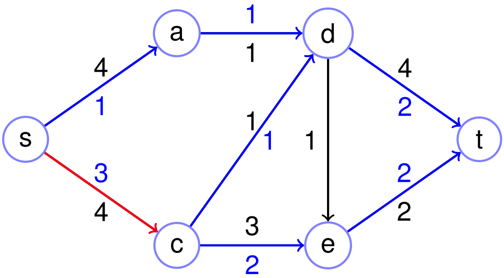


Find Flow: f_e

1. $0 \leq f_e \leq c_e$. "Capacity constraints."
2. If u is not s or t
   $\sum_{(w,u)\in E} f_{wu} = \sum_{(u,w)\in E} f_{uw}$.

Maximize: $\text{size}(f) = \sum_{(s,u)\in E} f_{su}$.

Check Result...

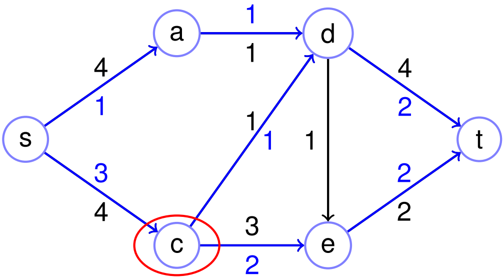


Find Flow: f_e

1. $0 \leq f_e \leq c_e$. "Capacity constraints." $3 = f_{s,c} \leq c_{s,c} = 4$.
2. If u is not s or t
   $\sum_{(w,u) \in E} f_{wu} = \sum_{(u,w) \in E} f_{uw}$.

Maximize: $\text{size}(f) = \sum_{(s,u) \in E} f_{su}$.

Check Result...

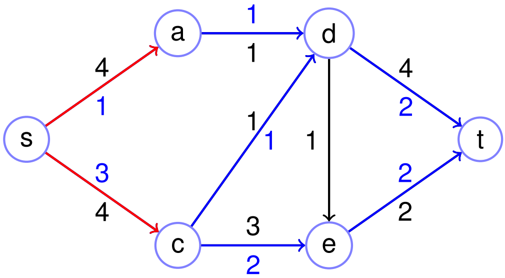


Find Flow: f_e

1. $0 \leq f_e \leq c_e$. "Capacity constraints." $3 = f_{s,c} \leq c_{s,c} = 4$.
2. If u is not s or t
   $\sum_{(w,u) \in E} f_{wu} = \sum_{(u,w) \in E} f_{uw}$. $3 = f_{s,c} = f_{c,d} + f_{c,e} = 2 + 1$.

Maximize: $\text{size}(f) = \sum_{(s,u) \in E} f_{su}$.

Check Result...

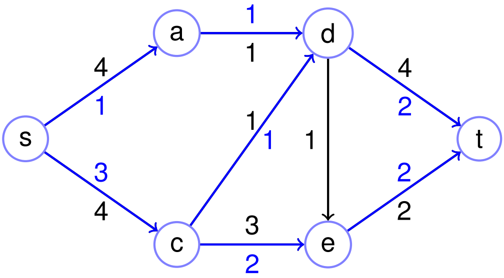


Find Flow: f_e

1. $0 \leq f_e \leq c_e$. "Capacity constraints." $3 = f_{s,c} \leq c_{s,c} = 4$.
2. If u is not s or t
   $\sum_{(w,u)\in E} f_{wu} = \sum_{(u,w)\in E} f_{uw}$. $3 = f_{s,c} = f_{c,d} + f_{c,e} = 2 + 1$.

Maximize: size$(f) = \sum_{(s,u)\in E} f_{su}$. $f_{sa} + f_{sc} = 1 + 3 = 4$

Check Result...



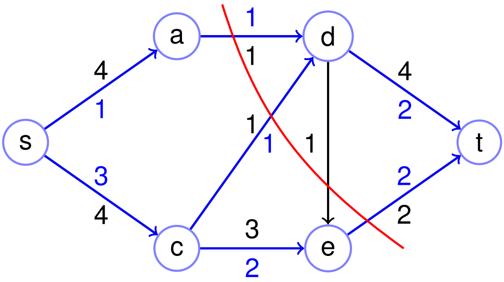
Find Flow: f_e

1. $0 \le f_e \le c_e$. "Capacity constraints." $3 = f_{s,c} \le c_{s,c} = 4$.
2. If u is not s or t
   $\sum_{(w,u)\in E} f_{wu} = \sum_{(u,w)\in E} f_{uw}$. $3 = f_{s,c} = f_{c,d} + f_{c,e} = 2 + 1$.

Maximize: $\text{size}(f) = \sum_{(s,u)\in E} f_{su}$. $f_{sa} + f_{sc} = 1 + 3 = 4$

Optimal?

Check Result...



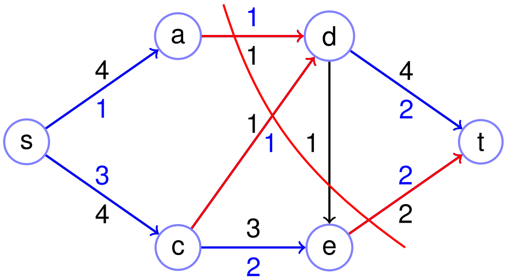
Find Flow: f_e

1. $0 \leq f_e \leq c_e$. "Capacity constraints." $3 = f_{s,c} \leq c_{s,c} = 4$.
2. If u is not s or t
   $\sum_{(w,u) \in E} f_{wu} = \sum_{(u,w) \in E} f_{uw}$. $3 = f_{s,c} = f_{c,d} + f_{c,e} = 2 + 1$.

Maximize: $\text{size}(f) = \sum_{(s,u) \in E} f_{su}$. $f_{sa} + f_{sc} = 1 + 3 = 4$

Optimal?

Check Result...



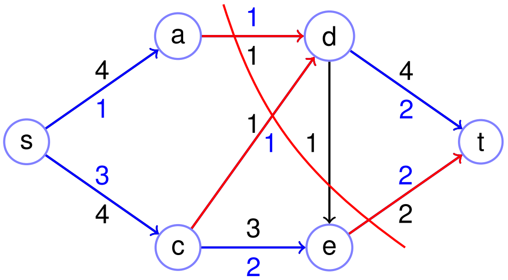
Find Flow: f_e

1. $0 \leq f_e \leq c_e$. "Capacity constraints." $3 = f_{s,c} \leq c_{s,c} = 4$.
2. If u is not s or t
   $\sum_{(w,u) \in E} f_{wu} = \sum_{(u,w) \in E} f_{uw}$. $3 = f_{s,c} = f_{c,d} + f_{c,e} = 2 + 1$.

Maximize: $\text{size}(f) = \sum_{(s,u) \in E} f_{su}$. $f_{sa} + f_{sc} = 1 + 3 = 4$

Optimal? $c_{ad} + c_{cd} + c_{et} = 1 + 1 + 2 = 4$.

Check Result...



Find Flow: f_e

1. $0 \leq f_e \leq c_e$. "Capacity constraints." $3 = f_{s,c} \leq c_{s,c} = 4$.
2. If u is not s or t
   $\sum_{(w,u)\in E} f_{wu} = \sum_{(u,w)\in E} f_{uw}$. $3 = f_{s,c} = f_{c,d} + f_{c,e} = 2 + 1$.

Maximize: $\text{size}(f) = \sum_{(s,u)\in E} f_{su}$. $f_{sa} + f_{sc} = 1 + 3 = 4$

Optimal? $c_{ad} + c_{cd} + c_{et} = 1 + 1 + 2 = 4$.

Any $s - t$ cut gives an upper bound.

Feasibility.

Feasibility.

1. Capacity Constraints: $0 \leq f_e \leq c_e$.

Feasibility.

1. Capacity Constraints: $0 \leq f_e \leq c_e$.
Only increase flow to c_e.

Feasibility.

1. Capacity Constraints: $0 \leq f_e \leq c_e$.
Only increase flow to c_e.
Or use reverse arcs decrease to 0.

Feasibility.

1. Capacity Constraints: $0 \leq f_e \leq c_e$.
Only increase flow to c_e.
Or use reverse arcs decrease to 0.
Flow values to be between 0 and c_e.

Feasibility.

1. Capacity Constraints: $0 \leq f_e \leq c_e$.
Only increase flow to c_e.
Or use reverse arcs decrease to 0.
Flow values to be between 0 and c_e.

2. Conservation Constraints:

Feasibility.

1. Capacity Constraints: $0 \leq f_e \leq c_e$.
Only increase flow to c_e.
Or use reverse arcs decrease to 0.
Flow values to be between 0 and c_e.

2. Conservation Constraints:
"flow into v" = "flow out of v" (if not s or t.)

Feasibility.

1. Capacity Constraints: $0 \leq f_e \leq c_e$.
Only increase flow to c_e.
Or use reverse arcs decrease to 0.
Flow values to be between 0 and c_e.

2. Conservation Constraints:
"flow into v" = "flow out of v" (if not s or t.)
Algorithm adds flow, say f, to path from s to t.

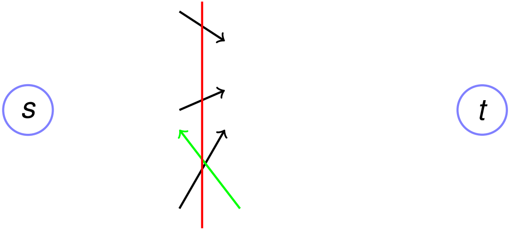
Feasibility.

1. Capacity Constraints: $0 \leq f_e \leq c_e$.
Only increase flow to c_e.
Or use reverse arcs decrease to 0.
Flow values to be between 0 and c_e.

2. Conservation Constraints:
"flow into v" = "flow out of v" (if not s or t.)
Algorithm adds flow, say f, to path from s to t.
Each internal node has f in, and f out.

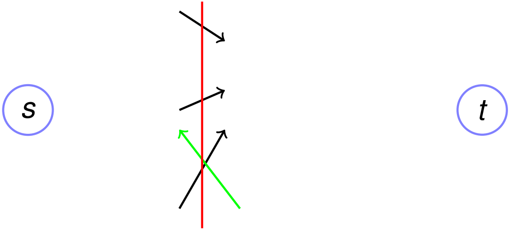
Optimality: upper bound.

s-t Cut: $V = S \cup T$ and $s \in S$ and $t \in T$.



Optimality: upper bound.

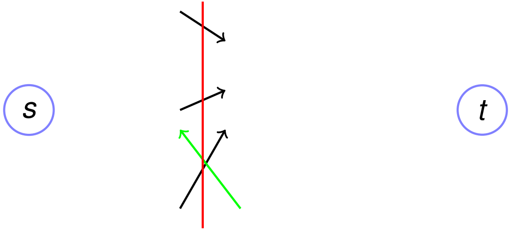
s-t Cut: $V = S \cup T$ and $s \in S$ and $t \in T$.



Lemma: Capacity of any $s - t$ cut is an upper bound on the flow.

Optimality: upper bound.

s-t Cut: $V = S \cup T$ and $s \in S$ and $t \in T$.

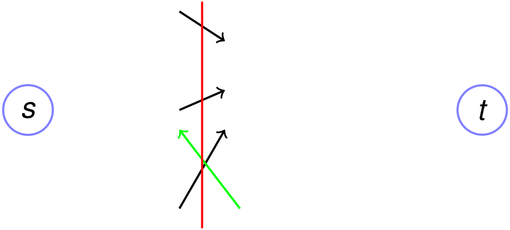


Lemma: Capacity of any $s - t$ cut is an upper bound on the flow.

$C(S, T)$ - sum of capacities of all arcs from S to T

Optimality: upper bound.

s-t Cut: $V = S \cup T$ and $s \in S$ and $t \in T$.



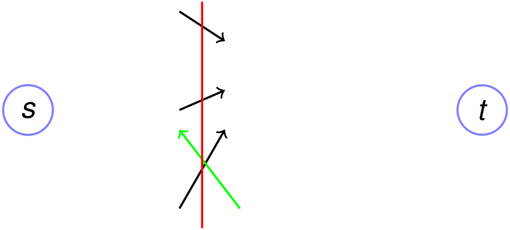
Lemma: Capacity of any $s - t$ cut is an upper bound on the flow.

$C(S,T)$ - sum of capacities of all arcs from S to T

$C(S,T) = \sum_{e=(u,v):u \in S, V \in T} c_e$

Optimality: upper bound.

s-*t* Cut: $V = S \cup T$ and $s \in S$ and $t \in T$.



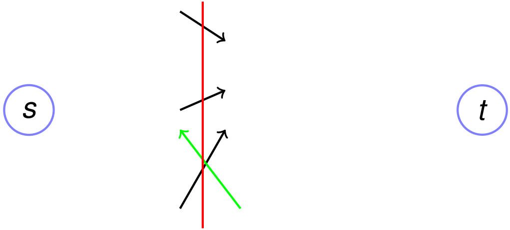
Lemma: Capacity of any $s - t$ cut is an upper bound on the flow.

$C(S,T)$ - sum of capacities of all arcs from S to T
$C(S,T) = \sum_{e=(u,v):u \in S, V \in T} c_e$

For valid flow:

Optimality: upper bound.

s-t Cut: $V = S \cup T$ and $s \in S$ and $t \in T$.



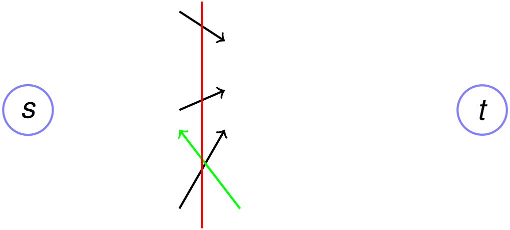
Lemma: Capacity of any $s - t$ cut is an upper bound on the flow.

$C(S,T)$ - sum of capacities of all arcs from S to T
$C(S,T) = \sum_{e=(u,v):u \in S, V \in T} c_e$

For valid flow:
Flow out of (S) = Flow out of s.

Optimality: upper bound.

s-t Cut: $V = S \cup T$ and $s \in S$ and $t \in T$.



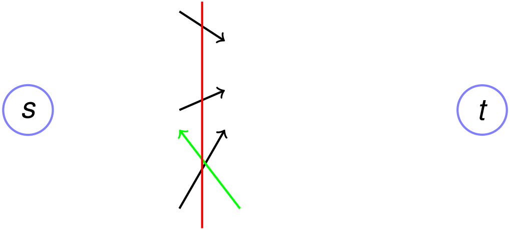
Lemma: Capacity of any $s - t$ cut is an upper bound on the flow.

$C(S,T)$ - sum of capacities of all arcs from S to T
$C(S,T) = \sum_{e=(u,v):u \in S, V \in T} c_e$

For valid flow:
Flow out of (S) = Flow out of s.
$\sum_{v \in S} \sum_{(v,u) \in E} f(v,u) - \sum_{(u,v) \in E} f(u,v) = \sum_{e \in (S,T)} f(e)$

Optimality: upper bound.

s-t Cut: $V = S \cup T$ and $s \in S$ and $t \in T$.



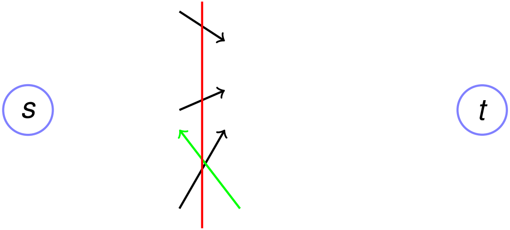
Lemma: Capacity of any $s - t$ cut is an upper bound on the flow.

$C(S,T)$ - sum of capacities of all arcs from S to T
$C(S,T) = \sum_{e=(u,v):u \in S, V \in T} c_e$

For valid flow:
Flow out of (S) = Flow out of s.
$\sum_{v \in S} \sum_{(v,u) \in E} f(v,u) - \sum_{(u,v) \in E} f(u,v) = \sum_{e \in (S,T)} f(e)$
Flow into (T) = Flow into t.

Optimality: upper bound.

s-t Cut: $V = S \cup T$ and $s \in S$ and $t \in T$.



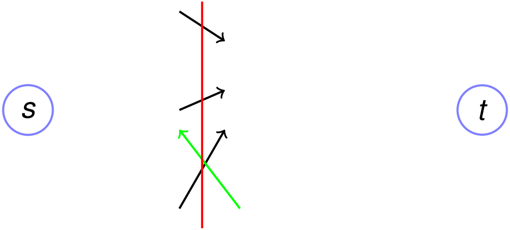
Lemma: Capacity of any $s - t$ cut is an upper bound on the flow.

$C(S,T)$ - sum of capacities of all arcs from S to T
$C(S,T) = \sum_{e=(u,v):u \in S, V \in T} c_e$

For valid flow:
Flow out of (S) = Flow out of s.
$\sum_{v \in S} \sum_{(v,u) \in E} f(v,u) - \sum_{(u,v) \in E} f(u,v) = \sum_{e \in (S,T)} f(e)$
Flow into (T) = Flow into t.

Optimality: upper bound.

s-t Cut: $V = S \cup T$ and $s \in S$ and $t \in T$.



Lemma: Capacity of any $s - t$ cut is an upper bound on the flow.

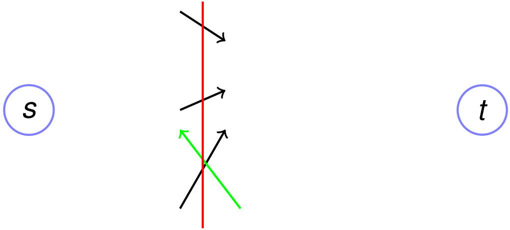
$C(S,T)$ - sum of capacities of all arcs from S to T
$C(S,T) = \sum_{e=(u,v):u \in S, V \in T} c_e$

For valid flow:
Flow out of (S) = Flow out of s.
$\sum_{v \in S} \sum_{(v,u) \in E} f(v,u) - \sum_{(u,v) \in E} f(u,v) = \sum_{e \in (S,T)} f(e)$
Flow into (T) = Flow into t.

For any valid flow, $f : E \to Z+$, the flow out of S (into T)

Optimality: upper bound.

s-t Cut: $V = S \cup T$ and $s \in S$ and $t \in T$.



Lemma: Capacity of any $s - t$ cut is an upper bound on the flow.

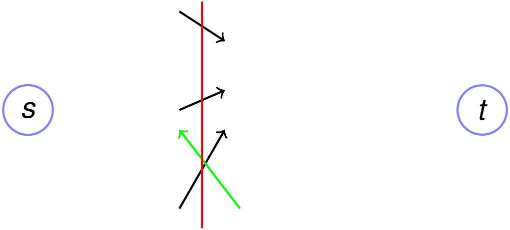
$C(S,T)$ - sum of capacities of all arcs from S to T
$C(S,T) = \sum_{e=(u,v):u \in S, V \in T} c_e$

For valid flow:
Flow out of (S) = Flow out of s.
$\sum_{v \in S} \sum_{(v,u) \in E} f(v,u) - \sum_{(u,v) \in E} f(u,v) = \sum_{e \in (S,T)} f(e)$
Flow into (T) = Flow into t.

For any valid flow, $f : E \rightarrow Z+$, the flow out of S (into T)

Optimality: upper bound.

s-t Cut: $V = S \cup T$ and $s \in S$ and $t \in T$.



Lemma: Capacity of any $s - t$ cut is an upper bound on the flow.

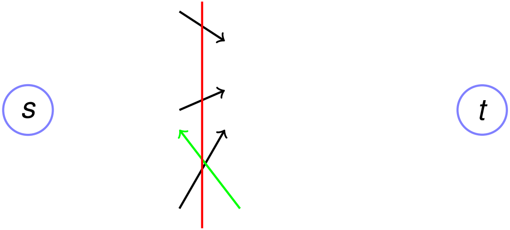
$C(S,T)$ - sum of capacities of all arcs from S to T
$C(S,T) = \sum_{e=(u,v):u \in S, V \in T} c_e$

For valid flow:
Flow out of (S) = Flow out of s.
$\sum_{v \in S} \sum_{(v,u) \in E} f(v,u) - \sum_{(u,v) \in E} f(u,v) = \sum_{e \in (S,T)} f(e)$
Flow into (T) = Flow into t.

For any valid flow, $f : E \rightarrow Z+$, the flow out of S (into T)
$\sum_{e \in S \times T} f_e$

Optimality: upper bound.

s-t Cut: $V = S \cup T$ and $s \in S$ and $t \in T$.



Lemma: Capacity of any $s - t$ cut is an upper bound on the flow.

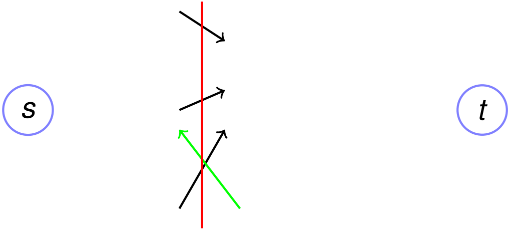
$C(S,T)$ - sum of capacities of all arcs from S to T
$C(S,T) = \sum_{e=(u,v):u \in S, V \in T} c_e$

For valid flow:
Flow out of (S) = Flow out of s.
$\sum_{v \in S} \sum_{(v,u) \in E} f(v,u) - \sum_{(u,v) \in E} f(u,v) = \sum_{e \in (S,T)} f(e)$
Flow into (T) = Flow into t.

For any valid flow, $f : E \rightarrow Z+$, the flow out of S (into T)
$\sum_{e \in S \times T} f_e - \sum_{e \in T \times S} f_e$

Optimality: upper bound.

s-t Cut: $V = S \cup T$ and $s \in S$ and $t \in T$.



Lemma: Capacity of any $s - t$ cut is an upper bound on the flow.

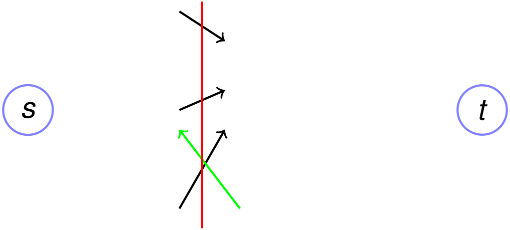
$C(S,T)$ - sum of capacities of all arcs from S to T
$C(S,T) = \sum_{e=(u,v):u \in S, V \in T} c_e$

For valid flow:
Flow out of (S) = Flow out of s.
$\sum_{v \in S} \sum_{(v,u) \in E} f(v,u) - \sum_{(u,v) \in E} f(u,v) = \sum_{e \in (S,T)} f(e)$
Flow into (T) = Flow into t.

For any valid flow, $f : E \rightarrow Z+$, the flow out of S (into T)
$\sum_{e \in S \times T} f_e - \sum_{e \in T \times S} f_e \leq \sum_{e \in S \times T} c_e - \sum_{e \in T \times S} 0$

Optimality: upper bound.

s-t Cut: $V = S \cup T$ and $s \in S$ and $t \in T$.



Lemma: Capacity of any $s - t$ cut is an upper bound on the flow.

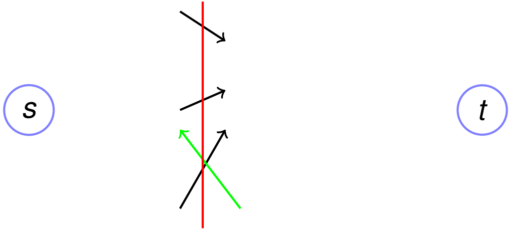
$C(S,T)$ - sum of capacities of all arcs from S to T
$C(S,T) = \sum_{e=(u,v):u \in S, V \in T} c_e$

For valid flow:
Flow out of (S) = Flow out of s.
$\sum_{v \in S} \sum_{(v,u) \in E} f(v,u) - \sum_{(u,v) \in E} f(u,v) = \sum_{e \in (S,T)} f(e)$
Flow into (T) = Flow into t.

For any valid flow, $f : E \rightarrow Z+$, the flow out of S (into T)
$\sum_{e \in S \times T} f_e - \sum_{e \in T \times S} f_e \leq \sum_{e \in S \times T} c_e - \sum_{e \in T \times S} 0 = C(S,T)$.

Optimality: upper bound.

s-t Cut: $V = S \cup T$ and $s \in S$ and $t \in T$.



Lemma: Capacity of any $s-t$ cut is an upper bound on the flow.

$C(S,T)$ - sum of capacities of all arcs from S to T
$C(S,T) = \sum_{e=(u,v):u \in S, V \in T} c_e$

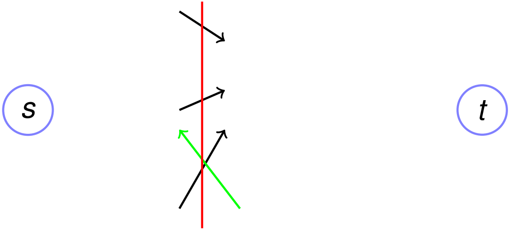
For valid flow:
Flow out of (S) = Flow out of s.
$\sum_{v \in S} \sum_{(v,u) \in E} f(v,u) - \sum_{(u,v) \in E} f(u,v) = \sum_{e \in (S,T)} f(e)$
Flow into (T) = Flow into t.

For any valid flow, $f : E \rightarrow Z+$, the flow out of S (into T)
$\sum_{e \in S \times T} f_e - \sum_{e \in T \times S} f_e \leq \sum_{e \in S \times T} c_e - \sum_{e \in T \times S} 0 = C(S,T)$.

$\rightarrow$ The value of any valid flow is at most $C(S,T)$!

Optimality: upper bound.

s-t Cut: $V = S \cup T$ and $s \in S$ and $t \in T$.



Lemma: Capacity of any $s - t$ cut is an upper bound on the flow.

$C(S,T)$ - sum of capacities of all arcs from S to T
$C(S,T) = \sum_{e=(u,v):u \in S, V \in T} c_e$

For valid flow:
Flow out of (S) = Flow out of s.
$\sum_{v \in S} \sum_{(v,u) \in E} f(v,u) - \sum_{(u,v) \in E} f(u,v) = \sum_{e \in (S,T)} f(e)$
Flow into (T) = Flow into t.

For any valid flow, $f : E \to Z+$, the flow out of S (into T)
$\sum_{e \in S \times T} f_e - \sum_{e \in T \times S} f_e \le \sum_{e \in S \times T} c_e - \sum_{e \in T \times S} 0 = C(S,T)$.

$\to$ The value of any valid flow is at most $C(S,T)$! $\square$

Optimality: max flow = min cut.

At termination of augmenting path algorithm.

Optimality: max flow = min cut.

At termination of augmenting path algorithm.

No path with residual capacity!

Optimality: max flow = min cut.

At termination of augmenting path algorithm.
  No path with residual capacity!

Depth first search only starting at s does not reach t.

Optimality: max flow = min cut.

At termination of augmenting path algorithm.
  No path with residual capacity!

Depth first search only starting at s does not reach t.

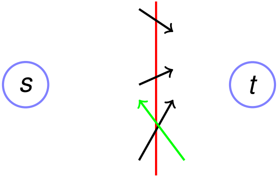
S be reachable nodes.

Optimality: max flow = min cut.

At termination of augmenting path algorithm.
No path with residual capacity!

Depth first search only starting at s does not reach t.

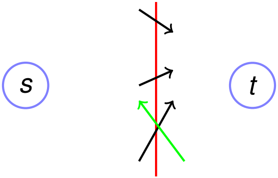
S be reachable nodes.



Optimality: max flow = min cut.

At termination of augmenting path algorithm.
  No path with residual capacity!

Depth first search only starting at *s* does not reach *t*.



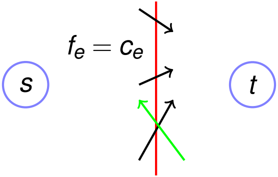
S be reachable nodes.

No arc with positive residual capacity leaving *S*

Optimality: max flow = min cut.

At termination of augmenting path algorithm.
No path with residual capacity!

Depth first search only starting at s does not reach t.



S be reachable nodes.

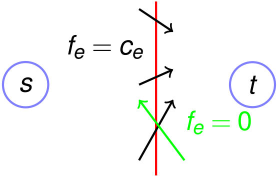
No arc with positive residual capacity leaving S

$\implies$ All arcs leaving S are full.

Optimality: max flow = min cut.

At termination of augmenting path algorithm.
No path with residual capacity!

Depth first search only starting at s does not reach t.



S be reachable nodes.

No arc with positive residual capacity leaving S

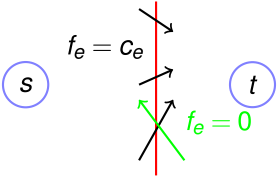
$\Longrightarrow$ All arcs leaving S are full.

$\Longrightarrow$ No arcs into S have flow.

Optimality: max flow = min cut.

At termination of augmenting path algorithm.
  No path with residual capacity!

Depth first search only starting at s does not reach t.



S be reachable nodes.

No arc with positive residual capacity leaving S

$\Longrightarrow$ All arcs leaving S are full.

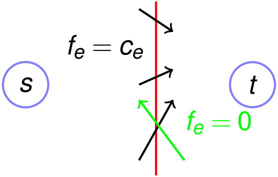
$\Longrightarrow$ No arcs into S have flow.

Total flow leaving S is $C(S,T)$.

Optimality: max flow = min cut.

At termination of augmenting path algorithm.
No path with residual capacity!

Depth first search only starting at s does not reach t.



S be reachable nodes.

No arc with positive residual capacity leaving S

$\implies$ All arcs leaving S are full.

$\implies$ No arcs into S have flow.

Total flow leaving S is $C(S, T)$.

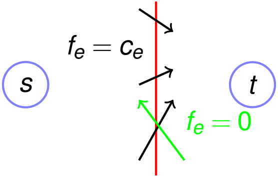
Valid flow $\implies$ all that flow from source.

Optimality: max flow = min cut.

At termination of augmenting path algorithm.

No path with residual capacity!

Depth first search only starting at s does not reach t.



S be reachable nodes.

No arc with positive residual capacity leaving S

$\implies$ All arcs leaving S are full.

$\implies$ No arcs into S have flow.

Total flow leaving S is $C(S, T)$.

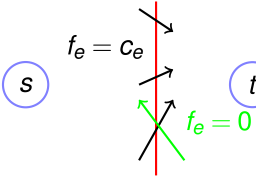
Valid flow $\implies$ all that flow from source.

Value of flow equals value of $C(S, T)$.

Optimality: max flow = min cut.

At termination of augmenting path algorithm.
No path with residual capacity!

Depth first search only starting at s does not reach t.



S be reachable nodes.

No arc with positive residual capacity leaving S

$\implies$ All arcs leaving S are full.

$\implies$ No arcs into S have flow.

Total flow leaving S is $C(S,T)$.

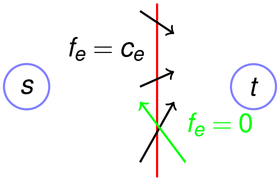
Valid flow $\implies$ all that flow from source.

Value of flow equals value of $C(S,T)$. and

Optimality: max flow = min cut.

At termination of augmenting path algorithm.
  No path with residual capacity!

Depth first search only starting at s does not reach t.



S be reachable nodes.

No arc with positive residual capacity leaving S

$\implies$ All arcs leaving S are full.

$\implies$ No arcs into S have flow.

Total flow leaving S is $C(S,T)$.

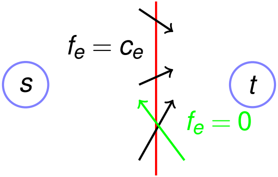
Valid flow $\implies$ all that flow from source.

Value of flow equals value of $C(S,T)$. and Optimal is $\leq C(S,T)$.

Optimality: max flow = min cut.

At termination of augmenting path algorithm.
No path with residual capacity!

Depth first search only starting at s does not reach t.



S be reachable nodes.

No arc with positive residual capacity leaving S

$\implies$ All arcs leaving S are full.

$\implies$ No arcs into S have flow.

Total flow leaving S is $C(S, T)$.

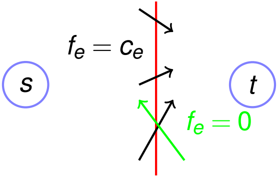
Valid flow $\implies$ all that flow from source.

Value of flow equals value of $C(S, T)$. and Optimal is $\leq C(S, T)$.
$\rightarrow$

Optimality: max flow = min cut.

At termination of augmenting path algorithm.
No path with residual capacity!

Depth first search only starting at s does not reach t.



S be reachable nodes.

No arc with positive residual capacity leaving S

$\implies$ All arcs leaving S are full.

$\implies$ No arcs into S have flow.

Total flow leaving S is $C(S, T)$.

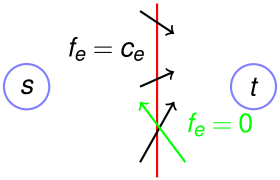
Valid flow $\implies$ all that flow from source.

Value of flow equals value of $C(S, T)$. and Optimal is $\leq C(S, T)$.
$\rightarrow$ Flow is maximum!!

Optimality: max flow = min cut.

At termination of augmenting path algorithm.
  No path with residual capacity!

Depth first search only starting at s does not reach t.



S be reachable nodes.

No arc with positive residual capacity leaving S

$\implies$ All arcs leaving S are full.

$\implies$ No arcs into S have flow.

Total flow leaving S is $C(S, T)$.

Valid flow $\implies$ all that flow from source.

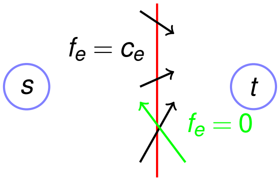
Value of flow equals value of $C(S, T)$. and Optimal is $\leq C(S, T)$.
$\rightarrow$ Flow is maximum!!

Cut is minimum $s - t$ cut too!

Optimality: max flow = min cut.

At termination of augmenting path algorithm.
  No path with residual capacity!

Depth first search only starting at s does not reach t.



S be reachable nodes.

No arc with positive residual capacity leaving S

$\implies$ All arcs leaving S are full.

$\implies$ No arcs into S have flow.

Total flow leaving S is $C(S, T)$.

Valid flow $\implies$ all that flow from source.

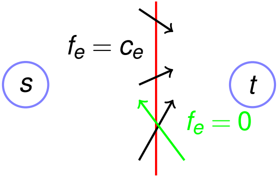
Value of flow equals value of $C(S, T)$. and Optimal is $\leq C(S, T)$.
$\rightarrow$ Flow is maximum!!

Cut is minimum $s - t$ cut too!
"any flow" $\leq$ "any cut"

Optimality: max flow = min cut.

At termination of augmenting path algorithm.
No path with residual capacity!

Depth first search only starting at s does not reach t.



S be reachable nodes.

No arc with positive residual capacity leaving S

$\implies$ All arcs leaving S are full.

$\implies$ No arcs into S have flow.

Total flow leaving S is $C(S, T)$.

Valid flow $\implies$ all that flow from source.

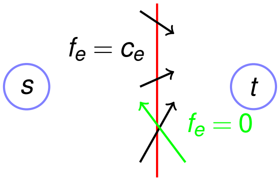
Value of flow equals value of $C(S, T)$. and Optimal is $\leq C(S, T)$.
$\rightarrow$ Flow is maximum!!

Cut is minimum $s - t$ cut too!
"any flow" $\leq$ "any cut" and

Optimality: max flow = min cut.

At termination of augmenting path algorithm.
No path with residual capacity!

Depth first search only starting at s does not reach t.



S be reachable nodes.

No arc with positive residual capacity leaving S

$\implies$ All arcs leaving S are full.

$\implies$ No arcs into S have flow.

Total flow leaving S is $C(S, T)$.

Valid flow $\implies$ all that flow from source.

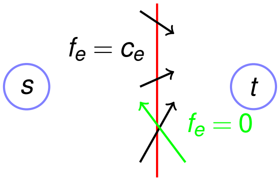
Value of flow equals value of $C(S, T)$. and Optimal is $\leq C(S, T)$.
$\rightarrow$ Flow is maximum!!

Cut is minimum $s - t$ cut too!
"any flow" $\leq$ "any cut" and this flow $=$ this cut.

Optimality: max flow = min cut.

At termination of augmenting path algorithm.
No path with residual capacity!

Depth first search only starting at s does not reach t.



S be reachable nodes.

No arc with positive residual capacity leaving S

$\implies$ All arcs leaving S are full.

$\implies$ No arcs into S have flow.

Total flow leaving S is $C(S,T)$.

Valid flow $\implies$ all that flow from source.

Value of flow equals value of $C(S,T)$. and Optimal is $\leq C(S,T)$.
$\rightarrow$ Flow is maximum!!

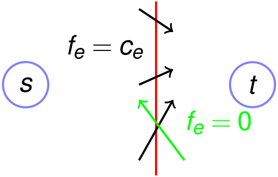
Cut is minimum $s-t$ cut too!
"any flow" $\leq$ "any cut" and this flow $=$ this cut.

$\rightarrow$

Optimality: max flow = min cut.

At termination of augmenting path algorithm.
No path with residual capacity!

Depth first search only starting at s does not reach t.



S be reachable nodes.

No arc with positive residual capacity leaving S

$\implies$ All arcs leaving S are full.

$\implies$ No arcs into S have flow.

Total flow leaving S is $C(S, T)$.

Valid flow $\implies$ all that flow from source.

Value of flow equals value of $C(S, T)$. and Optimal is $\leq C(S, T)$.
$\rightarrow$ Flow is maximum!!

Cut is minimum $s - t$ cut too!
"any flow" $\leq$ "any cut" and this flow $=$ this cut.
$\rightarrow$ Maximum flow and minimum $s - t$ cut!

Celebrated max flow -minimum cut theorem.

Theorem: In any flow network, the maximum s-t flow is equal to the minimum cut.

Celebrated max flow -minimum cut theorem.

Theorem: In any flow network, the maximum *s*-*t* flow is equal to the minimum cut.

Celebrate!

Back to business: Algorithm Terminates?

Back to business: Algorithm Terminates?

It will!!

Back to business: Algorithm Terminates?

It will!!

Flow keeps increasing.

Back to business: Algorithm Terminates?

It will!!

Flow keeps increasing.

How long?

Back to business: Algorithm Terminates?

It will!!

Flow keeps increasing.

How long?

One more unit every step!

Back to business: Algorithm Terminates?

It will!!

Flow keeps increasing.

How long?

One more unit every step!

$O(mF)$ time

Back to business: Algorithm Terminates?

It will!!

Flow keeps increasing.

How long?

One more unit every step!

$O(mF)$ time where F is size of flow.

Back to business: Algorithm Terminates?

It will!!

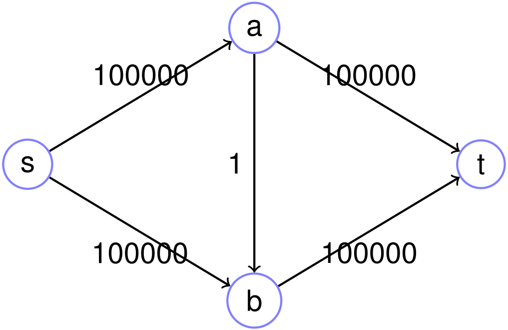
Flow keeps increasing.

How long?

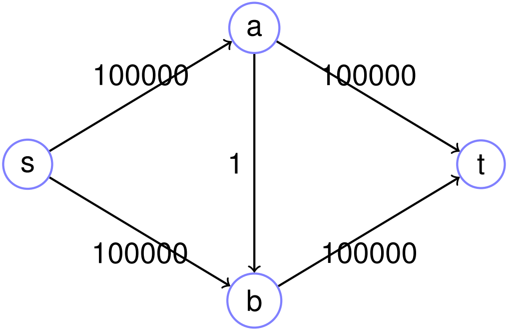
One more unit every step!

$O(mF)$ time where F is size of flow.

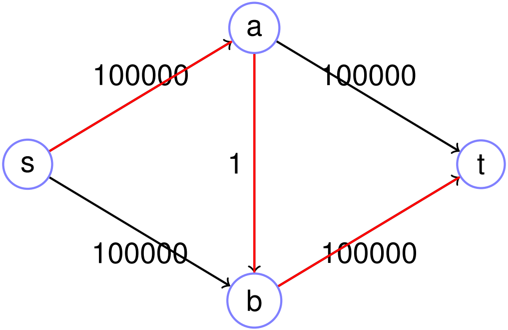
Efficiency.



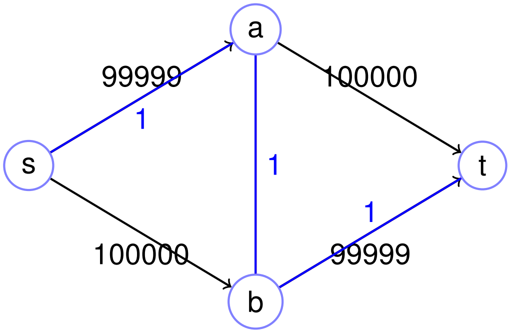
Efficiency.



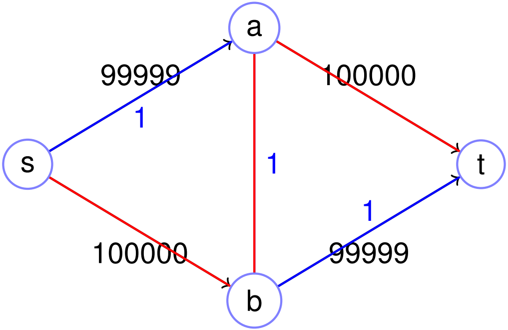
Efficiency.



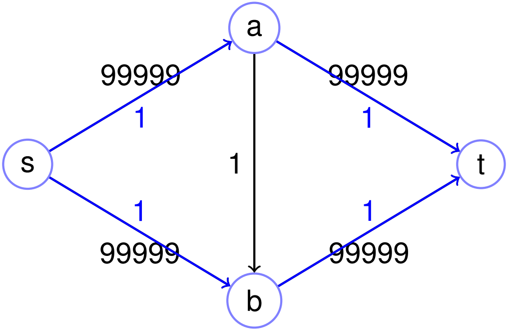
Efficiency.



Efficiency.



Efficiency.



Edmonds-Karp

Augment along shortest path.

Edmonds-Karp

Augment along shortest path.

Breadth first search!

Edmonds-Karp

Augment along shortest path.

Breadth first search!

$O(|V||E|)$ augmentations.

Edmonds-Karp

Augment along shortest path.

Breadth first search!

$O(|V||E|)$ augmentations.

Analysis idea.

Edmonds-Karp

Augment along shortest path.

Breadth first search!

$O(|V||E|)$ augmentations.

Analysis idea.

$d(v)$ is distance to sink.

Edmonds-Karp

Augment along shortest path.

Breadth first search!

$O(|V||E|)$ augmentations.

Analysis idea.

$d(v)$ is distance to sink.

Only route flow on (u,v) if $d(u) \geq d(v)$.

Edmonds-Karp

Augment along shortest path.

Breadth first search!

$O(|V||E|)$ augmentations.

Analysis idea.

$d(v)$ is distance to sink.

Only route flow on (u,v) if $d(u) \geq d(v)$.

Only reverse flow on (u,v) if $d(v) \leq d(u)$.

Edmonds-Karp

Augment along shortest path.

Breadth first search!

$O(|V||E|)$ augmentations.

Analysis idea.

$d(v)$ is distance to sink.

Only route flow on (u,v) if $d(u) \geq d(v)$.

Only reverse flow on (u,v) if $d(v) \leq d(u)$.

Maximum $d(v)$ is $|V|$.

Edmonds-Karp

Augment along shortest path.

Breadth first search!

$O(|V||E|)$ augmentations.

Analysis idea.

$d(v)$ is distance to sink.

Only route flow on (u,v) if $d(u) \geq d(v)$.

Only reverse flow on (u,v) if $d(v) \leq d(u)$.

Maximum $d(v)$ is $|V|$.

Distances only go up. (To prove!)

Edmonds-Karp

Augment along shortest path.

Breadth first search!

$O(|V||E|)$ augmentations.

Analysis idea.

$d(v)$ is distance to sink.

Only route flow on (u,v) if $d(u) \geq d(v)$.

Only reverse flow on (u,v) if $d(v) \leq d(u)$.

Maximum $d(v)$ is $|V|$.

Distances only go up. (To prove!)

Every $O(|E|)$ time augment removes edge “at” a distance.

Edmonds-Karp

Augment along shortest path.

Breadth first search!

$O(|V||E|)$ augmentations.

Analysis idea.

$d(v)$ is distance to sink.

Only route flow on (u,v) if $d(u) \geq d(v)$.

Only reverse flow on (u,v) if $d(v) \leq d(u)$.

Maximum $d(v)$ is $|V|$.

Distances only go up. (To prove!)

Every $O(|E|)$ time augment removes edge “at” a distance.

$O(|V||E|)$ removals.

Edmonds-Karp

Augment along shortest path.

Breadth first search!

$O(|V||E|)$ augmentations.

Analysis idea.

$d(v)$ is distance to sink.

Only route flow on (u,v) if $d(u) \geq d(v)$.

Only reverse flow on (u,v) if $d(v) \leq d(u)$.

Maximum $d(v)$ is $|V|$.

Distances only go up. (To prove!)

Every $O(|E|)$ time augment removes edge "at" a distance.

$O(|V||E|)$ removals.

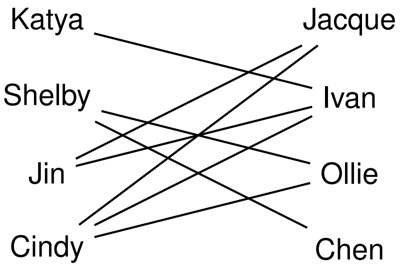
$O(|V||E|^2)$ time.

Bipartite Matching

Given a bipartite graph: $B = (L, R, E)$ where $E \subseteq L \times R$.

Bipartite Matching

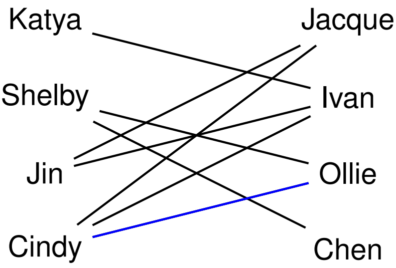
Given a bipartite graph: $B = (L, R, E)$ where $E \subseteq L \times R$.



Find largest subset of edges ("matches") which are one to one.

Bipartite Matching

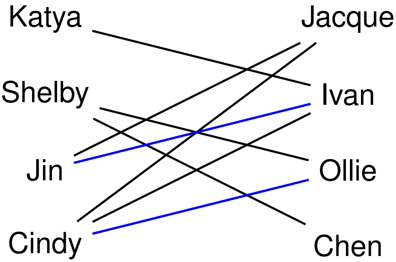
Given a bipartite graph: $B = (L, R, E)$ where $E \subseteq L \times R$.



Find largest subset of edges (“matches”) which are one to one.

Bipartite Matching

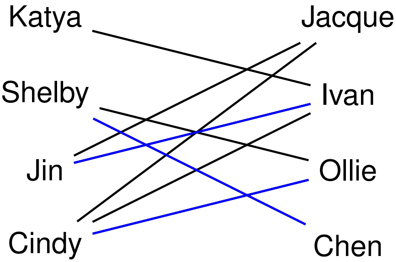
Given a bipartite graph: $B = (L, R, E)$ where $E \subseteq L \times R$.



Find largest subset of edges ("matches") which are one to one.

Bipartite Matching

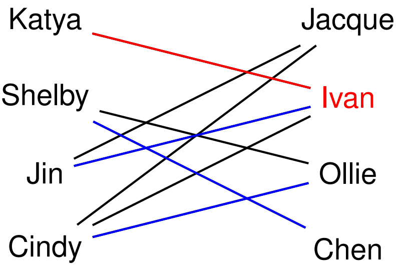
Given a bipartite graph: $B = (L, R, E)$ where $E \subseteq L \times R$.



Find largest subset of edges ("matches") which are one to one.

Bipartite Matching

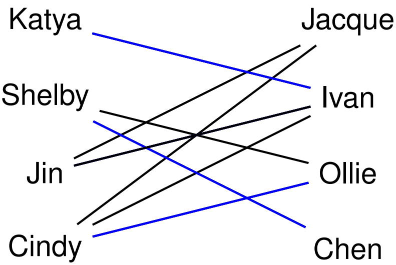
Given a bipartite graph: $B = (L, R, E)$ where $E \subseteq L \times R$.



Find largest subset of edges (“matches”) which are one to one.

Bipartite Matching

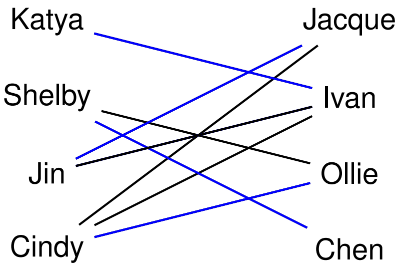
Given a bipartite graph: $B = (L, R, E)$ where $E \subseteq L \times R$.



Find largest subset of edges (“matches”) which are one to one.

Bipartite Matching

Given a bipartite graph: $B = (L, R, E)$ where $E \subseteq L \times R$.



Find largest subset of edges (“matches”) which are one to one.

Bipartite Matching

Algorithm by “Reduction.”:

From matching problem produce flow problem.

Bipartite Matching

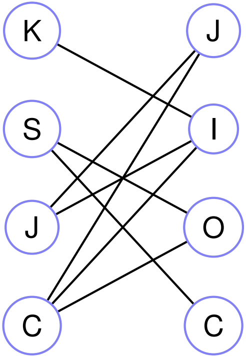
Algorithm by “Reduction.”:
From matching problem produce flow problem.
From flow solution produce matching solution.

Bipartite Matching

Algorithm by "Reduction.":

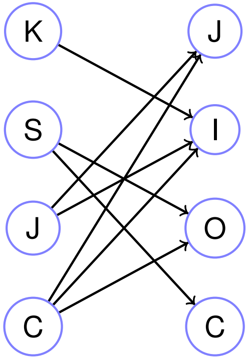
From matching problem produce flow problem.

From flow solution produce matching solution.



Bipartite Matching

Algorithm by “Reduction.”:
  From matching problem produce flow problem.
  From flow solution produce matching solution.

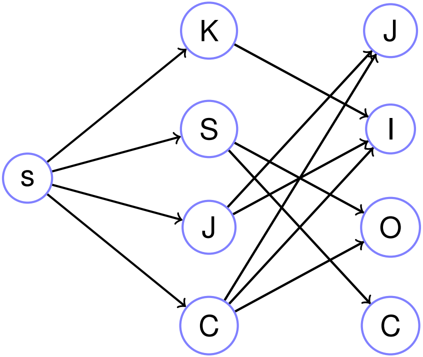


Bipartite Matching

Algorithm by "Reduction.":

From matching problem produce flow problem.

From flow solution produce matching solution.

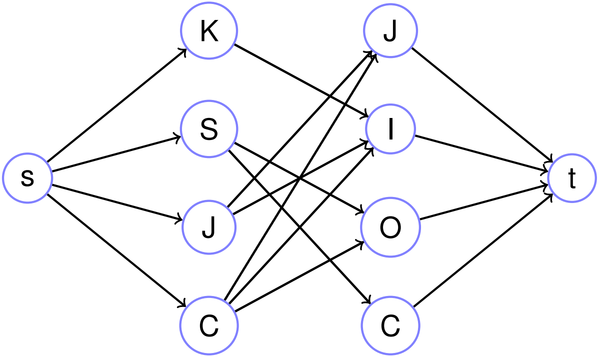


Bipartite Matching

Algorithm by "Reduction.":

From matching problem produce flow problem.

From flow solution produce matching solution.

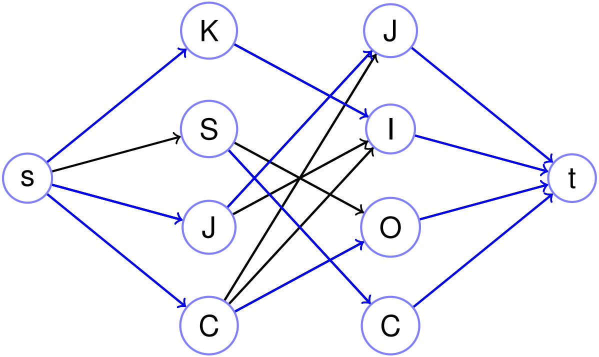


Bipartite Matching

Algorithm by “Reduction.”:

From matching problem produce flow problem.

From flow solution produce matching solution.

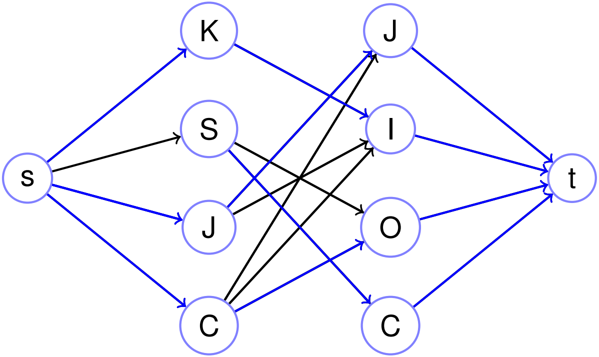


Bipartite Matching

Algorithm by "Reduction.":

From matching problem produce flow problem.

From flow solution produce matching solution.



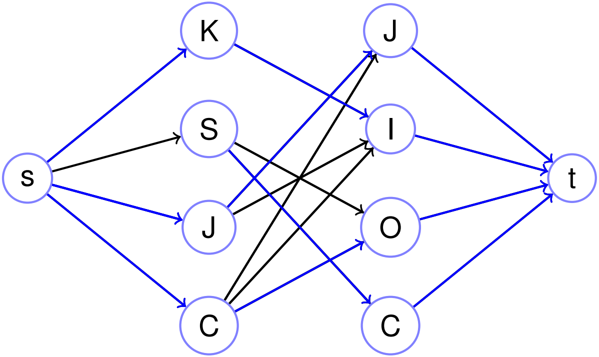
Max flow = Max Matching Size.

Bipartite Matching

Algorithm by “Reduction.”:

From matching problem produce flow problem.

From flow solution produce matching solution.



Max flow = Max Matching Size.

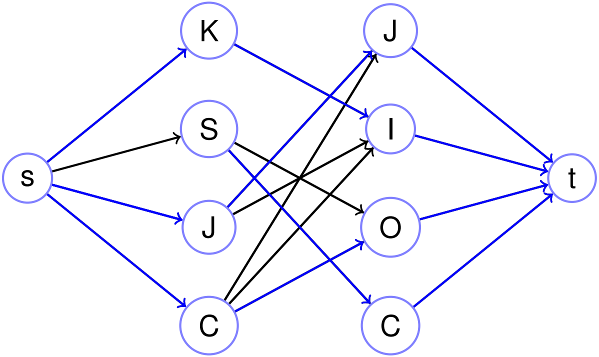
Flow is not integer necessarily....

Bipartite Matching

Algorithm by "Reduction.":

From matching problem produce flow problem.

From flow solution produce matching solution.



Max flow = Max Matching Size.

Flow is not integer necessarily....

Augmenting path algorithm gives integer flow.

Maximum Matching Problem.

Maximum Matching

Given a bipartite graph, $G = (U, V, E)$, find a maximum sized matching.

Maximum Matching

Given a bipartite graph, $G = (U, V, E)$, find a maximum sized matching.

Key Idea: Augmenting Alternating Paths.

Maximum Matching

Given a bipartite graph, $G = (U, V, E)$, find a maximum sized matching.

Key Idea: Augmenting Alternating Paths.

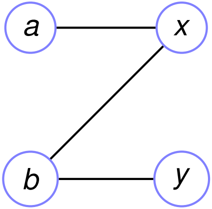
Example:

Maximum Matching

Given a bipartite graph, $G = (U, V, E)$, find a maximum sized matching.

Key Idea: Augmenting Alternating Paths.

Example:

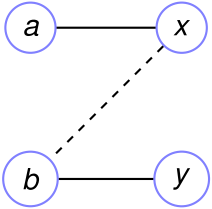


Maximum Matching

Given a bipartite graph, $G = (U, V, E)$, find a maximum sized matching.

Key Idea: Augmenting Alternating Paths.

Example:

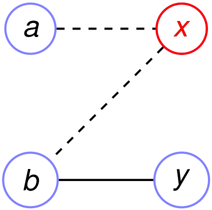


Maximum Matching

Given a bipartite graph, $G = (U, V, E)$, find a maximum sized matching.

Key Idea: Augmenting Alternating Paths.

Example:

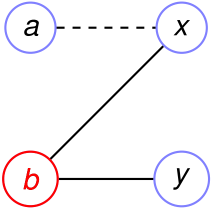


Maximum Matching

Given a bipartite graph, $G = (U, V, E)$, find a maximum sized matching.

Key Idea: Augmenting Alternating Paths.

Example:



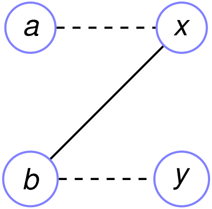
Start at unmatched node(s),
follow unmatched edge(s),
follow matched.
Repeat until an unmatched node.

Maximum Matching

Given a bipartite graph, $G = (U, V, E)$, find a maximum sized matching.

Key Idea: Augmenting Alternating Paths.

Example:

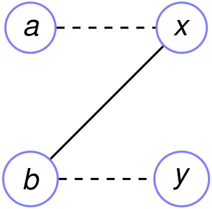


Maximum Matching

Given a bipartite graph, $G = (U, V, E)$, find a maximum sized matching.

Key Idea: Augmenting Alternating Paths.

Example:



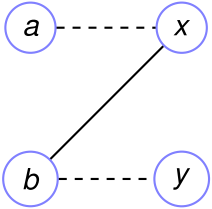
Start at unmatched node(s),

Maximum Matching

Given a bipartite graph, $G = (U, V, E)$, find a maximum sized matching.

Key Idea: Augmenting Alternating Paths.

Example:



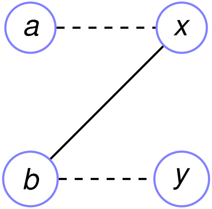
Start at unmatched node(s),
follow unmatched edge(s),

Maximum Matching

Given a bipartite graph, $G = (U, V, E)$, find a maximum sized matching.

Key Idea: Augmenting Alternating Paths.

Example:



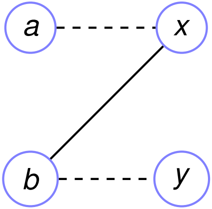
Start at unmatched node(s),
follow unmatched edge(s),
follow matched.

Maximum Matching

Given a bipartite graph, $G = (U, V, E)$, find a maximum sized matching.

Key Idea: Augmenting Alternating Paths.

Example:

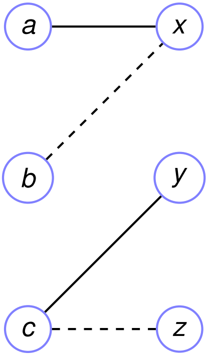


Start at unmatched node(s),
follow unmatched edge(s),
follow matched.
Repeat until an unmatched node.

No perfect matching

Algorithm:
Given matching.
Direct unmatched edges U to V, matched V to U.
Find path between unmatched nodes on left to right. (BFS, DFS).
Until everything matched ... or output a cut.

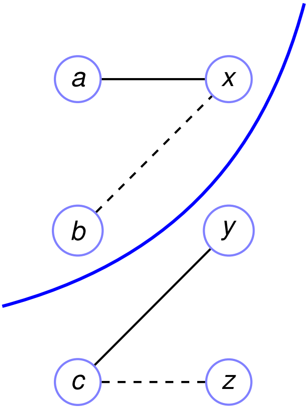
No perfect matching



Can't increase matching size.
No alternating path from (a) to (y).

Algorithm:
Given matching.
Direct unmatched edges U to V, matched V to U.
Find path between unmatched nodes on left to right. (BFS, DFS).
Until everything matched ... or output a cut.

No perfect matching

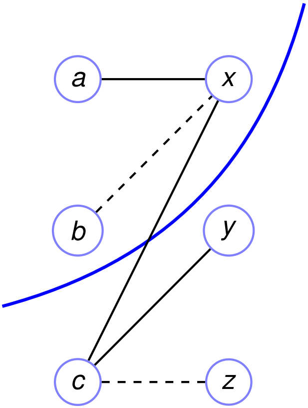


Can't increase matching size.
No alternating path from (a) to (y).

Cut!

Algorithm:
Given matching.
Direct unmatched edges U to V, matched V to U.
Find path between unmatched nodes on left to right. (BFS, DFS).
Until everything matched ... or output a cut.

No perfect matching



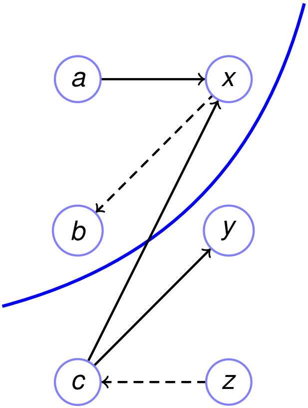
Can't increase matching size.
No alternating path from (a) to (y).

Cut!

Still no augmenting path.
Still Cut?

Algorithm:
Given matching.
Direct unmatched edges U to V, matched V to U.
Find path between unmatched nodes on left to right. (BFS, DFS).
Until everything matched ... or output a cut.

No perfect matching



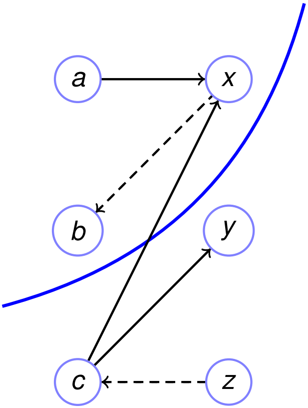
Can't increase matching size.
No alternating path from (a) to (y).

Cut!

Still no augmenting path.
Still Cut?

Use directed graph!
Cut in this graph.

No perfect matching



Can't increase matching size.
No alternating path from (a) to (y).

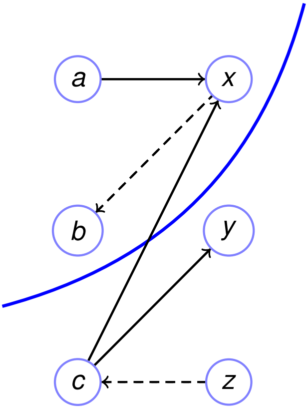
Cut!

Still no augmenting path.
Still Cut?

Use directed graph!
Cut in this graph.

Algorithm:

No perfect matching



Can't increase matching size.
No alternating path from (a) to (y).

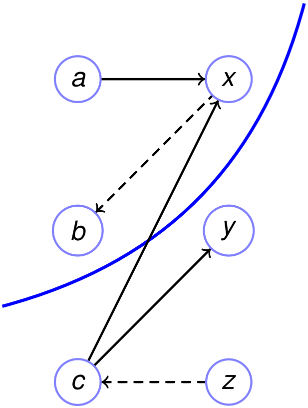
Cut!

Still no augmenting path.
Still Cut?

Use directed graph!
Cut in this graph.

Algorithm:
Given matching.

No perfect matching



Can't increase matching size.
No alternating path from (a) to (y).

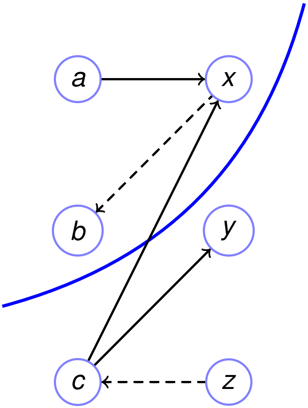
Cut!

Still no augmenting path.
Still Cut?

Use directed graph!
Cut in this graph.

Algorithm:
Given matching.
Direct unmatched edges U to V, matched V to U.

No perfect matching



Can't increase matching size.
No alternating path from (a) to (y).

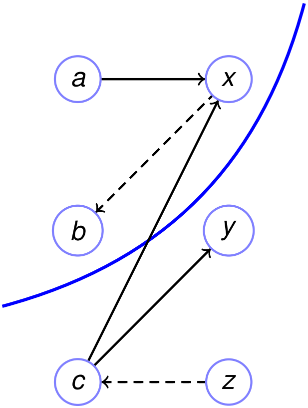
Cut!

Still no augmenting path.
Still Cut?

Use directed graph!
Cut in this graph.

Algorithm:
Given matching.
Direct unmatched edges U to V, matched V to U.
Find path between unmatched nodes on left to right. (BFS, DFS).

No perfect matching



Can’t increase matching size.
No alternating path from (a) to (y).

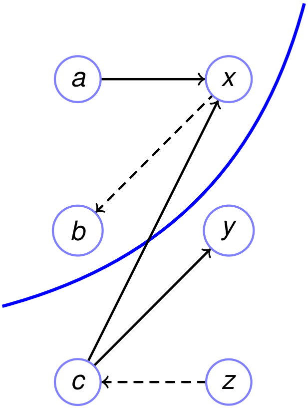
Cut!

Still no augmenting path.
Still Cut?

Use directed graph!
Cut in this graph.

Algorithm:
Given matching.
Direct unmatched edges U to V, matched V to U.
Find path between unmatched nodes on left to right. (BFS, DFS).
Until everything matched

No perfect matching



Can’t increase matching size.
No alternating path from (a) to (y).

Cut!

Still no augmenting path.
Still Cut?

Use directed graph!
Cut in this graph.

Algorithm:
Given matching.
Direct unmatched edges U to V, matched V to U.
Find path between unmatched nodes on left to right. (BFS, DFS).
Until everything matched ... or output a cut.

Simplex Algorithm

$$\max c \cdot x.$$

Simplex Algorithm

$$\max c \cdot x.$$

$$\begin{aligned} Ax &\leq b \\ x &\geq 0 \end{aligned}$$

Simplex Algorithm

$$\max c \cdot x.$$

$$\begin{aligned} Ax &\le b \\ x &\ge 0 \end{aligned}$$

Start at feasible point where n equations are satisfied.

Simplex Algorithm

$$\max c \cdot x.$$

$$\begin{aligned} Ax &\le b \\ x &\ge 0 \end{aligned}$$

Start at feasible point where n equations are satisfied.

E.g., $x = 0$.

Simplex Algorithm

$$\max c \cdot x.$$

$$\begin{aligned} Ax &\leq b \\ x &\geq 0 \end{aligned}$$

Start at feasible point where n equations are satisfied.

E.g., $x = 0$.
This is a point.

Simplex Algorithm

$$\max c \cdot x.$$

$$\begin{aligned} Ax &\leq b \\ x &\geq 0 \end{aligned}$$

Start at feasible point where n equations are satisfied.

E.g., $x = 0$.
This is a point.
Another view: intersection of n hyperplanes.

Simplex Algorithm

$$\max c \cdot x.$$

$$Ax \leq b$$
$$x \geq 0$$

Start at feasible point where n equations are satisfied.

E.g., $x = 0$.
This is a point.
Another view: intersection of n hyperplanes.

Simplex Algorithm

$$\max c \cdot x.$$

$$Ax \le b$$
$$x \ge 0$$

Start at feasible point where n equations are satisfied.

E.g., $x = 0$.
This is a point.
Another view: intersection of n hyperplanes.

Drop one equation:

Simplex Algorithm

$$\max c \cdot x.$$

$$Ax \leq b$$
$$x \geq 0$$

Start at feasible point where n equations are satisfied.

E.g., $x = 0$.
This is a point.
Another view: intersection of n hyperplanes.

Drop one equation:
Points on line satisfy $n-1$ ind. equations.

Simplex Algorithm

$$\max c \cdot x.$$

$$Ax \leq b$$
$$x \geq 0$$

Start at feasible point where n equations are satisfied.

E.g., $x = 0$.
This is a point.
Another view: intersection of n hyperplanes.

Drop one equation:
Points on line satisfy $n-1$ ind. equations.
Intersection of $n-1$ hyperplanes.

Simplex Algorithm

$$\max c \cdot x.$$

$$\begin{aligned} Ax &\leq b \\ x &\geq 0 \end{aligned}$$

Start at feasible point where n equations are satisfied.

E.g., $x = 0$.
This is a point.
Another view: intersection of n hyperplanes.

Drop one equation:
Points on line satisfy $n-1$ ind. equations.
Intersection of $n-1$ hyperplanes.

Move in direction that increases objective.

Simplex Algorithm

$$\max c \cdot x.$$

$$\begin{aligned} Ax &\leq b \\ x &\geq 0 \end{aligned}$$

Start at feasible point where n equations are satisfied.

E.g., $x = 0$.
This is a point.
Another view: intersection of n hyperplanes.

Drop one equation:
Points on line satisfy $n-1$ ind. equations.
Intersection of $n-1$ hyperplanes.

Move in direction that increases objective.

Until new tight constraint.

Simplex Algorithm

$$\max c \cdot x.$$

$$Ax \leq b$$
$$x \geq 0$$

Start at feasible point where n equations are satisfied.

E.g., $x = 0$.
This is a point.
Another view: intersection of n hyperplanes.

Drop one equation:
Points on line satisfy $n-1$ ind. equations.
Intersection of $n-1$ hyperplanes.

Move in direction that increases objective.

Until new tight constraint.

No direction increases objective.

Hyperplane View

$x + y + z \leq 1$

Hyperplane View

$x + y + z \leq 1$

On one side of hyperplane.

Hyperplane View

$x + y + z \leq 1$

On one side of hyperplane.

Normal to hyperplane?

Hyperplane View

$x + y + z \leq 1$

On one side of hyperplane.

Normal to hyperplane? $(1,1,1)$.

Hyperplane View

$x + y + z \leq 1$

On one side of hyperplane.

Normal to hyperplane? $(1,1,1)$.

Why?

Hyperplane View

$x + y + z \leq 1$

On one side of hyperplane.

Normal to hyperplane? $(1,1,1)$.

Why?
(a,b,c) where $a+b+c=1$.

Hyperplane View

$x + y + z \leq 1$

On one side of hyperplane.

Normal to hyperplane? $(1,1,1)$.

Why?
  (a,b,c) where $a+b+c = 1$.
  (a',b',c') where $a'+b'+c' = 1$.

Hyperplane View

$x + y + z \leq 1$

On one side of hyperplane.

Normal to hyperplane? $(1,1,1)$.

Why?
(a,b,c) where $a+b+c=1$.
(a',b',c') where $a'+b'+c'=1$.
$(a+1,b+1,c+1)-(a,b,c)=(1,1,1)$.

Hyperplane View

$x + y + z \leq 1$

On one side of hyperplane.

Normal to hyperplane? $(1,1,1)$.

Why?

(a,b,c) where $a+b+c = 1$.
(a',b',c') where $a'+b'+c' = 1$.
$(a+1,b+1,c+1) - (a,b,c) = (1,1,1)$.
$(a'-a,b'-b,c'-c) \cdot (1,1,1) = (a'+b'+c'-(a+b+c)) = 0$.

Hyperplane View

$x+y+z \leq 1$

On one side of hyperplane.

Normal to hyperplane? $(1,1,1)$.

Why?
(a,b,c) where $a+b+c=1$.
(a',b',c') where $a'+b'+c'=1$.
$(a+1,b+1,c+1)-(a,b,c)=(1,1,1)$.
$(a'-a,b'-b,c'-c)\cdot(1,1,1)=(a'+b'+c'-(a+b+c))=0$.

Hyperplane View

$x+y+z \leq 1$

On one side of hyperplane.

Normal to hyperplane? $(1,1,1)$.

Why?

(a,b,c) where $a+b+c = 1$.
(a',b',c') where $a'+b'+c' = 1$.
$(a+1,b+1,c+1)-(a,b,c) = (1,1,1)$.
$(a'-a,b'-b,c'-c)\cdot(1,1,1) = (a'+b'+c'-(a+b+c)) = 0$.

Normal to $mx+ny+pz = C$?

Hyperplane View

$x+y+z \leq 1$

On one side of hyperplane.

Normal to hyperplane? $(1,1,1)$.

Why?
(a,b,c) where $a+b+c=1$.
(a',b',c') where $a'+b'+c'=1$.
$(a+1,b+1,c+1)-(a,b,c)=(1,1,1)$.
$(a'-a,b'-b,c'-c)\cdot(1,1,1)=(a'+b'+c'-(a+b+c))=0$.

Normal to $mx+ny+pz=C$? (m,n,p)

Hyperplane View

$x + y + z \leq 1$

On one side of hyperplane.

Normal to hyperplane? $(1,1,1)$.

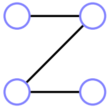
Why?

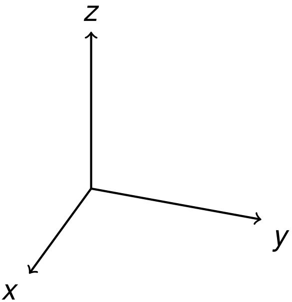
(a,b,c) where $a+b+c = 1$.
(a',b',c') where $a'+b'+c' = 1$.
$(a+1,b+1,c+1)-(a,b,c) = (1,1,1)$.
$(a'-a,b'-b,c'-c)\cdot(1,1,1) = (a'+b'+c'-(a+b+c)) = 0$.

Normal to $mx+ny+pz = C$? (m,n,p)

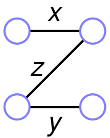
Points in hyperplane are related by nullspace of row.

Maximum matching and simplex.

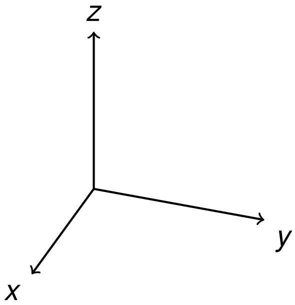




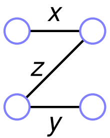
Maximum matching and simplex.



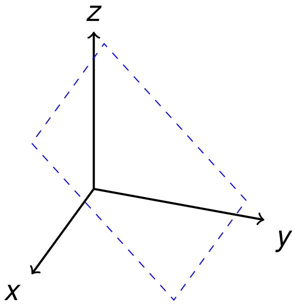
$$\begin{aligned} \max x+y+z \\ x \leq 1 \\ x+z \leq 1 \\ z+y \leq 1 \\ y \leq 1 \\ x \geq 0 \\ y \geq 0 \\ z \geq 0 \end{aligned}$$



Maximum matching and simplex.

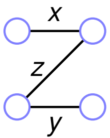


$$\max x + y + z$$
$$x \leq 1$$
$$x + z \leq 1$$
$$z + y \leq 1$$
$$y \leq 1$$
$$x \geq 0$$
$$y \geq 0$$
$$z \geq 0$$

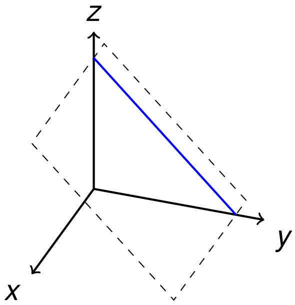


Blue constraints intersect.

Maximum matching and simplex.

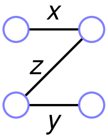


$$\begin{aligned} \max x+y+z \\ x \leq 1 \\ x+z \leq 1 \\ z+y \leq 1 \\ y \leq 1 \\ x \geq 0 \\ y \geq 0 \\ z \geq 0 \end{aligned}$$

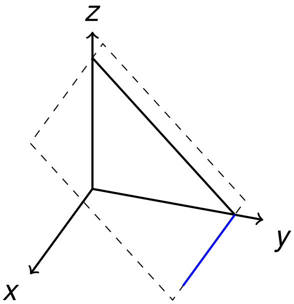


Blue constraints intersect.

Maximum matching and simplex.

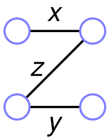


$$
\begin{aligned}
\max x+y+z \\
x \leq 1 \\
x+z \leq 1 \\
z+y \leq 1 \\
y \leq 1 \\
x \geq 0 \\
y \geq 0 \\
z \geq 0
\end{aligned}
$$

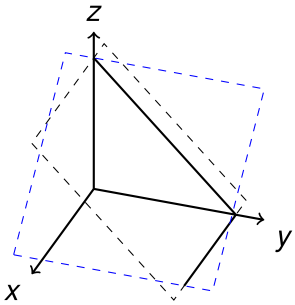


Blue constraints intersect.

Maximum matching and simplex.

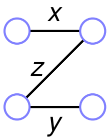


$$\max x + y + z$$
$$x \le 1$$
$$x + z \le 1$$
$$z + y \le 1$$
$$y \le 1$$
$$x \ge 0$$
$$y \ge 0$$
$$z \ge 0$$

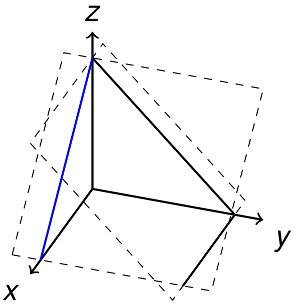


Blue constraints intersect.

Maximum matching and simplex.

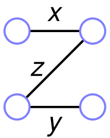


$$
\begin{aligned}
\max x+y+z \\
x \le 1 \\
{\color{blue} x+z \le 1} \\
z+y \le 1 \\
y \le 1 \\
x \ge 0 \\
{\color{blue} y \ge 0} \\
z \ge 0
\end{aligned}
$$

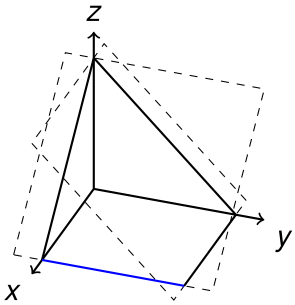


Blue constraints intersect.

Maximum matching and simplex.

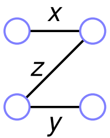


$$
\begin{aligned}
\max x+y+z \\
x \le 1 \\
x+z \le 1 \\
z+y \le 1 \\
y \le 1 \\
x \ge 0 \\
y \ge 0 \\
z \ge 0
\end{aligned}
$$

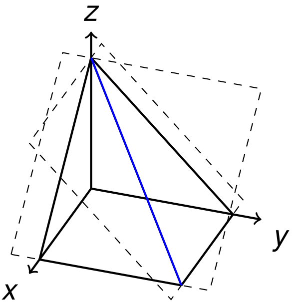


Blue constraints intersect.

Maximum matching and simplex.

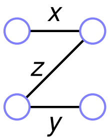


$$\max x + y + z$$
$$x \leq 1$$
$$x + z \leq 1$$
$$z + y \leq 1$$
$$y \leq 1$$
$$x \geq 0$$
$$y \geq 0$$
$$z \geq 0$$

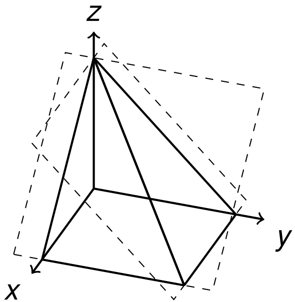


Blue constraints intersect.

Maximum matching and simplex.

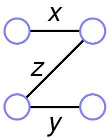


$$
\begin{aligned}
\max x+y+z & \\
x &\le 1\\
x+z &\le 1\\
z+y &\le 1\\
y &\le 1\\
x &\ge 0\\
y &\ge 0\\
z &\ge 0
\end{aligned}
$$

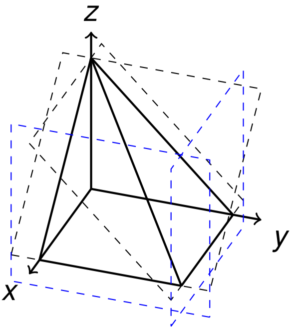


Blue constraints intersect.

Maximum matching and simplex.

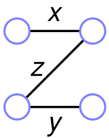


$$\max x + y + z$$
$$x \le 1$$
$$x + z \le 1$$
$$z + y \le 1$$
$$y \le 1$$
$$x \ge 0$$
$$y \ge 0$$
$$z \ge 0$$



Blue constraints redundant.

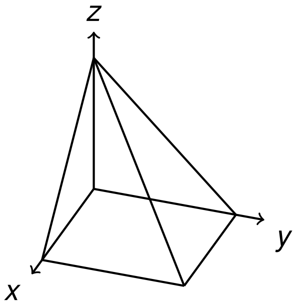
Maximum matching and simplex.



$$\max x + y + z$$

$$x + z \le 1$$
$$z + y \le 1$$

$$x \ge 0$$
$$y \ge 0$$
$$z \ge 0$$

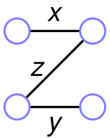


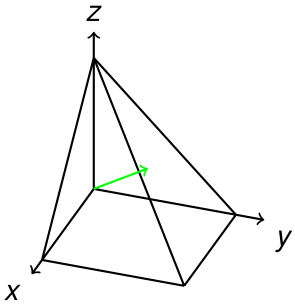
Maximum matching and simplex.

$$\max x + y + z$$

$$x + z \leq 1$$
$$z + y \leq 1$$

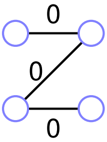
$$x \geq 0$$
$$y \geq 0$$
$$z \geq 0$$





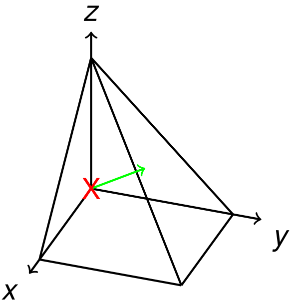
Maximum matching and simplex.

$$\max x + y + z$$



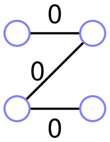
$$x + z \leq 1$$
$$z + y \leq 1$$

$$x \geq 0$$
$$y \geq 0$$
$$z \geq 0$$



Blue constraints tight.

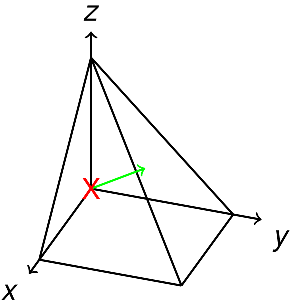
Maximum matching and simplex.



$$\max x + y + z$$

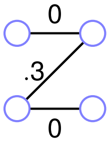
$$x + z \leq 1$$
$$z + y \leq 1$$

$$x \geq 0$$
$$y \geq 0$$
$$z \geq 0$$



Blue constraints tight.

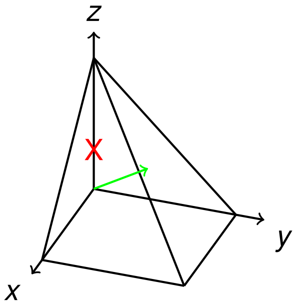
Maximum matching and simplex.



$$\max x + y + z$$

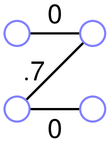
$$x + z \leq 1$$
$$z + y \leq 1$$

$$x \geq 0$$
$$y \geq 0$$
$$z \geq 0$$



Blue constraints tight.

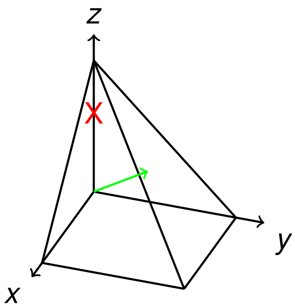
Maximum matching and simplex.



$$\max x + y + z$$

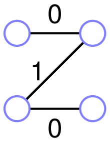
$$x + z \leq 1$$
$$z + y \leq 1$$

$$x \geq 0$$
$$y \geq 0$$
$$z \geq 0$$



Blue constraints tight.

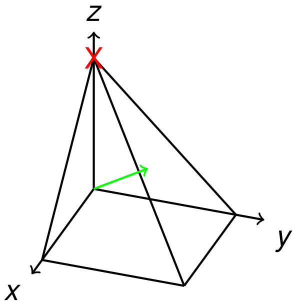
Maximum matching and simplex.



$$\max x + y + z$$

$$x + z \leq 1$$
$$z + y \leq 1$$

$$x \geq 0$$
$$y \geq 0$$
$$z \geq 0$$

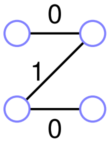


Blue constraints tight.

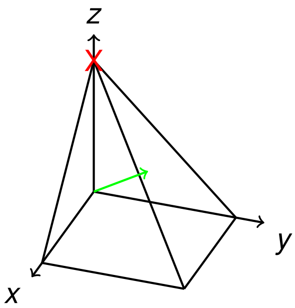
Maximum matching and simplex.

$$\max x + y + z$$

$$x + z \leq 1$$
$$z + y \leq 1$$

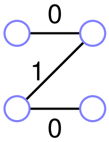


$$x \geq 0$$
$$y \geq 0$$
$$z \geq 0$$



Blue constraints tight.

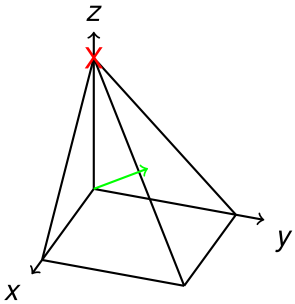
Maximum matching and simplex.



$$\max x + y + z$$

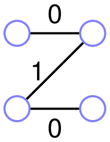
$$x + z \leq 1$$
$$z + y \leq 1$$

$$x \geq 0$$
$$y \geq 0$$
$$z \geq 0$$



Blue constraints tight.

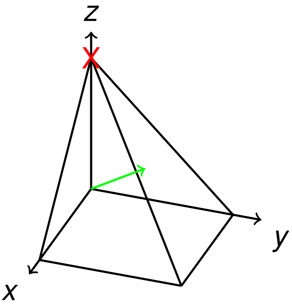
Maximum matching and simplex.



$$\max x + y + z$$

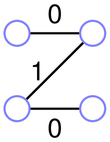
$$x + z \leq 1$$
$$z + y \leq 1$$

$$x \geq 0$$
$$y \geq 0$$
$$z \geq 0$$



Blue constraints tight.

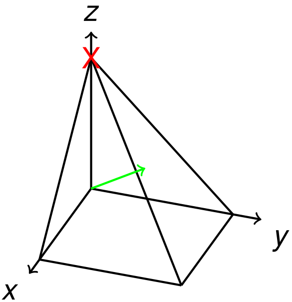
Maximum matching and simplex.



$$\max x + y + z$$

$$x + z \leq 1$$
$$z + y \leq 1$$

$$x \geq 0$$
$$y \geq 0$$
$$z \geq 0$$

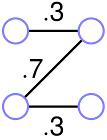


Blue constraints tight.

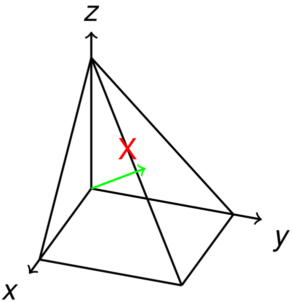
Maximum matching and simplex.

$$\max x + y + z$$

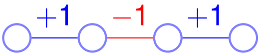
$$x + z \le 1$$
$$z + y \le 1$$



$$x \ge 0$$
$$y \ge 0$$
$$z \ge 0$$

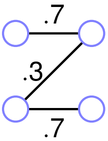


Blue constraints tight.



Augmenting Path.

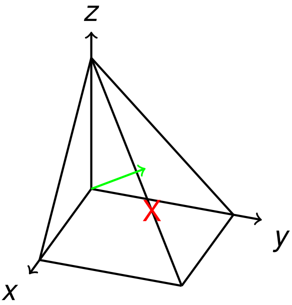
Maximum matching and simplex.



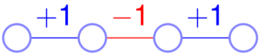
$$\max x + y + z$$

$$x + z \le 1$$
$$z + y \le 1$$

$$x \ge 0$$
$$y \ge 0$$
$$z \ge 0$$



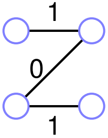
Blue constraints tight.



Augmenting Path. Via Gaussian Elimination!

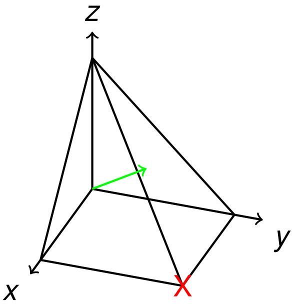
Maximum matching and simplex.

$$\max x + y + z$$

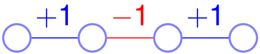


$$x + z \leq 1$$
$$z + y \leq 1$$

$$x \geq 0$$
$$y \geq 0$$
$$z \geq 0$$



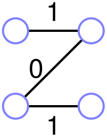
Blue constraints tight.



Augmenting Path. Via Gaussian Elimination!

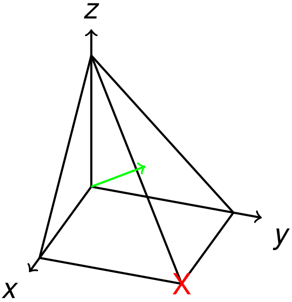
Maximum matching and simplex.

$$\max x + y + z$$

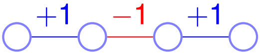


$$x + z \le 1$$
$$z + y \le 1$$

$$x \ge 0$$
$$y \ge 0$$
$$z \ge 0$$



Blue constraints tight.



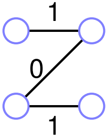
Augmenting Path. Via Gaussian Elimination!

Maximum matching and simplex.

$$\max x + y + z$$

$$x + z \leq 1$$

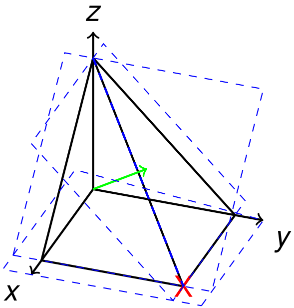
$$z + y \leq 1$$



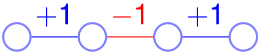
$$x \geq 0$$

$$y \geq 0$$

$$z \geq 0$$

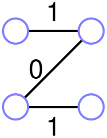


Blue constraints tight.



Augmenting Path. Via Gaussian Elimination!

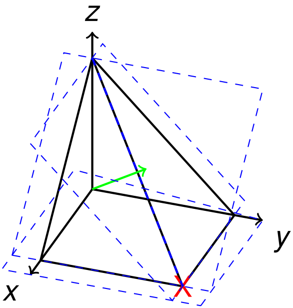
Maximum matching and simplex.



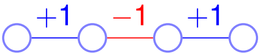
$$\max x + y + z$$

$$x + z \leq 1 \quad a$$
$$z + y \leq 1 \quad b$$

$$x \geq 0$$
$$y \geq 0$$
$$z \geq 0 \quad c$$

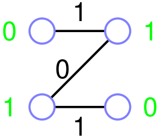


Blue constraints tight.



Augmenting Path. Via Gaussian Elimination!

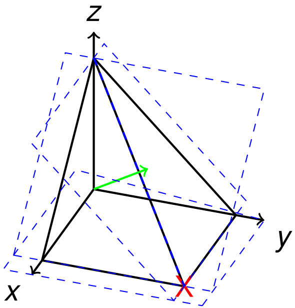
Maximum matching and simplex.



$$\max x + y + z$$

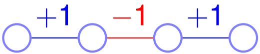
$$x + z \le 1 \quad a = 1$$
$$z + y \le 1 \quad b = 1$$

$$x \ge 0$$
$$y \ge 0$$
$$z \ge 0 \quad c = 1$$



Blue constraints tight.

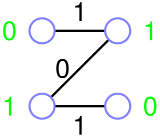
Sum: $x + z + y$.



Augmenting Path. Via Gaussian Elimination!

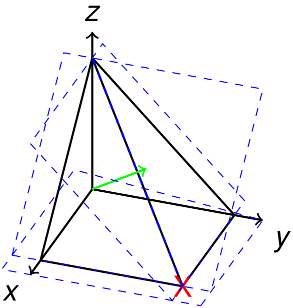
Maximum matching and simplex.

$$\max x + y + z$$



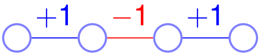
$$x + z \leq 1 \quad a = 1$$
$$z + y \leq 1 \quad b = 1$$

$$x \geq 0$$
$$y \geq 0$$
$$z \geq 0 \quad c = 1$$



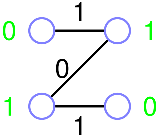
Blue constraints tight.

Sum: $x + z + y$.



Augmenting Path. Via Gaussian Elimination!

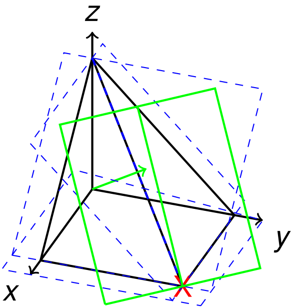
Maximum matching and simplex.



$$\max x + y + z$$

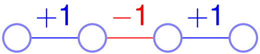
$$x + z \leq 1 \quad a = 1$$
$$z + y \leq 1 \quad b = 1$$

$$x \geq 0$$
$$y \geq 0$$
$$z \geq 0 \quad c = 1$$



Blue constraints tight.

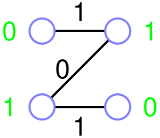
Sum: $x + z + y$.



Augmenting Path. Via Gaussian Elimination!

Maximum matching and simplex.

$$\max x + y + z$$



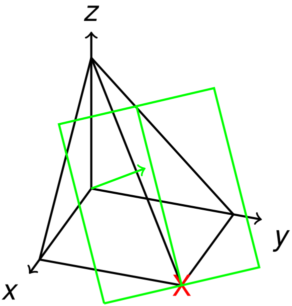
$$x + z \le 1 \quad a = 1$$

$$z + y \le 1 \quad b = 1$$

$$x \ge 0$$

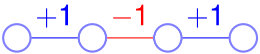
$$y \ge 0$$

$$z \ge 0 \quad c = 1$$



Blue constraints tight.

Sum: $x + z + y$.



Augmenting Path. Via Gaussian Elimination!

Lecture in a minute

Maximum flow.

"Greedy" augment path...

Except reverse old decisions ..

Reverse residual capacities.

: Optimality?

No augmenting path $\implies$

$s-t$ cut size = flow value.

Find flow and $s-t$ cut with equal value!

Lecture in a minute

Maximum flow.

“Greedy” augment path...

Except reverse old decisions ..

Reverse residual capacities.

: Optimality?

No augmenting path $\implies$

$s-t$ cut size = flow value.

Find flow and $s-t$ cut with equal value!

Maximum Matching.

$G=(V,E)$, find subset of one-to-one matches.

Reduction to max flow.

Augmenting Alternating Path

Algorithm $\equiv$ Simplex.