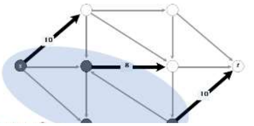
Minimum-cut problem

Def. An s-t cut (cut) is a partition (A, B) of the nodes with $s \in A$ and $t \in B$.

Def. Its capacity is the sum of the capacities of the edges from A to B.

$$cap(A, B) = \sum_{e \text{ out of } A} c(e)$$

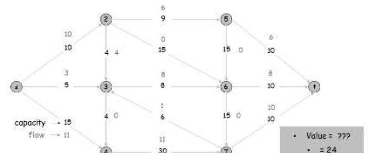
Min-cut problem. Find a cut of minimum capacity.



Flows and Cuts

Def. An s-t flow is a function that satisfies:
- For each $e \in E$: $0 \le f(e) \le c(e)$ [capacity]
- For each $v \in V - \{s, t\}$: $\sum_{e \text{ in to } v} f(e) = \sum_{e \text{ out of } v} f(e)$ [conservation]

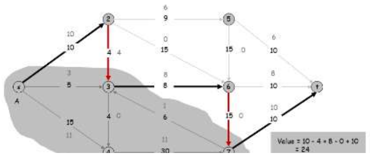
Def. The value of a flow f is: $v(f) = \sum_{e \text{ out of } s} f(e)$.



Certificate of Optimality

Flow value lemma. Let f be any flow, and let (A, B) be any s-t cut. Then, the net flow sent across the cut is equal to the amount leaving s.

$$\sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to } A} f(e) = v(f)$$



Certificate of optimality

Corollary. Let f be a flow and let (A, B) be any cut.
If $val(f) = cap(A, B)$, then f is a max flow and (A, B) is a min cut.

Pf.
- For any flow f: $val(f) \le cap(A, B) - val(f)$. weak duality
- For any cut (A_0, B_0): $cap(A_0, B_0) \ge val(f) = cap(A, B)$. ▪ weak duality

Flow value lemma. Let f be any flow, and let (A, B) be any s-t cut. Then

$$\sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to } A} f(e) = v(f).$$

Pf.

$$v(f) = \sum_{e \text{ out of } s} f(e)$$

by flow conservation, all terms except $v = s$ are 0 $\longrightarrow$ $= \sum_{v \in A} \left(\sum_{e \text{ out of } v} f(e) - \sum_{e \text{ in to } v} f(e) \right)$

$$= \sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to } A} f(e).$$

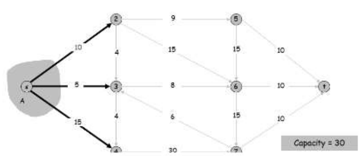
Why the greedy algorithm fails

Q. Why does the greedy algorithm fail?
A. Once greedy algorithm increases flow on an edge, it never decreases it.

Ex. Consider flow network G.
- The unique max flow f^* has $f^*(v, w) = 0$.
- Greedy algorithm could choose $s \to v \to w \to t$ as first path.

Weak duality. Let f be any flow, and let (A, B) be any s-t cut. Then the value of the flow is at most the capacity of the cut.

Cut capacity = 30 $\Rightarrow$ Flow value $\le$ 30

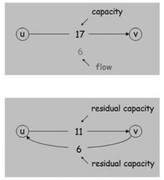


Residual Graph

Original edge: $e = (u, v) \in E$.
- Flow f(e), capacity c(e).

Residual edge.
- "Undo" flow sent.
- $e = (u, v)$ and $e^R = (v, u)$.
- Residual capacity:

$$c_f(e) = \begin{cases} c(e) - f(e) & \text{if } e \in E \\ f(e) & \text{if } e^R \in E \end{cases}$$

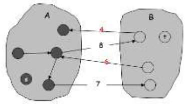


Residual graph: $G_f = (V, E_f)$.
- Residual edges with positive residual capacity.
- $E_f = \{e : f(e) < c(e)\} \cup \{e^R : f(e) > 0\}$.

Weak duality. Let f be any flow. Then, for any s-t cut (A, B) we have $v(f) \le cap(A, B)$.

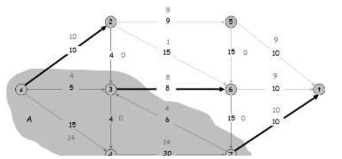
Pf.

$$\begin{aligned} v(f) &= \sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to } A} f(e) \\ &\le \sum_{e \text{ out of } A} f(e) \\ &\le \sum_{e \text{ out of } A} c(e) \\ &= cap(A, B) \quad ▪ \end{aligned}$$

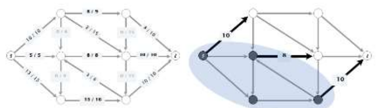


Corollary. Let f be any flow, and let (A, B) be any cut.
If v(f) = cap(A, B), then f is a max flow and (A, B) is a min cut.

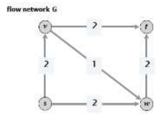
Value of flow = 28
Cut capacity = 28 $\Rightarrow$ Flow value $\le$ 28



Ford-Fulkerson Algorithm



Augmenting Path Algorithm



Max-Flow Min-Cut Theorem

Max-Flow Min-Cut Theorem

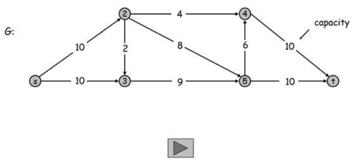
Augmenting path

Def. An augmenting path is a simple $s \to t$ path in the residual network G_f.

Def. The bottleneck capacity of an augmenting path P is the minimum residual capacity of any edge in P.

Key property. Let f be a flow and let P be an augmenting path in G_f. Then, after calling $f' \leftarrow$ AUGMENT(f, c, P), the resulting f' is a flow and $val(f') = val(f) + bottleneck(G_f, P)$.

AUGMENT(f, c, P)
$\delta \leftarrow$ bottleneck capacity of augmenting path P.
FOREACH edge $e \in P$:
IF $(e \in E)$ $f(e) \leftarrow f(e) + \delta$.
ELSE $f(e^{reverse}) \leftarrow f(e^{reverse}) - \delta$.
RETURN f.



f is a flow function that maps each edge e to a nonnegative number: E → R+
f(e): amount of flow carried by edge e

```
Augment(f, c, P) {
   b ← bottleneck(P)
   foreach e ∈ P {
      if (e ∈ E) f(e) ← f(e) + b        forward edge
      else       f(eR) ← f(eR) - b      reverse edge
   }
   return f
}
```

```
Ford-Fulkerson(G, s, t, c) {
   foreach e ∈ E  f(e) ← 0
   Gf ← residual graph

   while (there exists augmenting path P) {
      f ← Augment(f, c, P)
      update Gf
   }
   return f
```

Augmenting path theorem. Flow f is a max flow iff (if and only if) there are no augmenting paths.

Max-flow min-cut theorem. [Elias-Feinstein-Shannon 1956, Ford-Fulkerson 1956] The value of the max flow is equal to the value of the min cut.

Both can be proved simultaneously by showing TFAE (the following are equivalent):

(i) There exists a cut (A, B) such that v(f) = cap(A, B).

(ii) Flow f is a max flow.

(iii) There is no augmenting path relative to f.

Augmenting path theorem. Flow f is a max flow iff there are no augmenting paths.

Max-flow min-cut theorem. [Elias-Feinstein-Shannon 1956, Ford-Fulkerson 1956] The value of the max flow is equal to the value of the min cut.

Pf. We prove both simultaneously by showing TFAE (the following are equivalent):
(i) There exists a cut (A, B) such that v(f) = cap(A, B).
(ii) Flow f is a max flow.
(iii) There is no augmenting path relative to f.

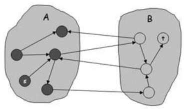
(i) $\Rightarrow$ (ii) This was the corollary to weak duality lemma.
Let f be any flow. Then, for any s-t cut (A, B) we have v(f) ≤ cap(A, B)

(ii) $\Rightarrow$ (iii) We show contrapositive.
- Let f be a flow. If there exists an augmenting path, then we can improve f by sending flow along path.

Proof of Max-Flow Min-Cut Theorem

(iii) $\Rightarrow$ (i)
- Let f be a flow with no augmenting paths.
- Let A be set of vertices reachable from s in residual graph.
- By definition of A, $s \in A$.
- By definition of f, $t \notin A$.

$$\begin{aligned} v(f) &= \sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to } A} f(e) \\ &= \sum_{e \text{ out of } A} c(e) \\ &= cap(A, B) \quad ▪ \end{aligned}$$



original network

How to Find Min-Cut

From Ford-Fulkerson, we get capacity of minimum cut.

How to find a minimum cut?
Use residual graph.

Following are steps to find a minimum cut:

1) Run Ford-Fulkerson algorithm and consider the final residual graph G_f

2) Perform BFS or DFS from source s to find set A that including all reachable vertices from s in the residual graph G_f.

3) Define set B = V- A, then return (A, B) as a min-cut.

Running Time for Ford-Fulkerson Algorithm

```
C times   while (there exists augmenting path P) {
O(m)         f ← Augment(f, c, P)
O(m)         update Gf
          }
```

Let C denotes the sum of capacities of all edges out of s. $C = \sum_{e \text{ out of } s} c(e)$.

Theorem. The algorithm terminates in at most $v(f^*) \le C$ iterations.
Pf. Each augmentation increase value by at least 1. ▪

- Finding an argument path takes O(m+n) = O(m) time, using BFS or DFS.
- Argument(f, P) takes O(n) < O(m) time since P contains n-1 edges.
- Update residual graph takes O(m) time since there are at most 2m edges in Gf.

Total running time is O(mC).

Choosing Good Augmenting Paths

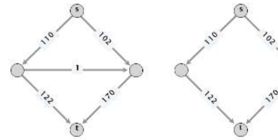
Use care when selecting augmenting paths.
- Some choices lead to exponential algorithms.
- Clever choices lead to polynomial algorithms.
- If capacities are irrational, algorithm not guaranteed to terminate!

Goal: choose augmenting paths so that:
- Can find augmenting paths efficiently.
- Few iterations.

Choose augmenting paths with: [Edmonds-Karp 1972, Dinitz 1970]
- Max bottleneck capacity.
- Sufficiently large bottleneck capacity.
- Fewest number of edges.

Capacity-scaling algorithm

Overview. Choosing augmenting paths with "large" bottleneck capacity.
- Maintain scaling parameter Δ. though not necessarily largest
- Let $G_f(\Delta)$ be the part of the residual network containing only those edges with capacity $\ge \Delta$.
- Any augmenting path in $G_f(\Delta)$ has bottleneck capacity $\ge \Delta$.



Capacity Scaling

$m^2 \log_2 C$

```
                        Scaling-Max-Flow(G, s, t, c) {
                           foreach e ∈ E  f(e) ← 0
                           Δ ← largest power of 2 no greater than C
                           Gf ← residual graph

1 + ⌈log2 C⌉ times         while (Δ ≥ 1) {
                              Gf(Δ) ← Δ-residual graph
<= 2m times                   while (there exists augmenting path P in Gf(Δ)) {
(Theorem 7.19)
                  O(m)           f ← augment(f, c, P)
                  O(m)           update Gf(Δ)
                              }
                              Δ ← Δ / 2
                           }
                           return f
                        }
```


Capacity-scaling algorithm

CAPACITY-SCALING(G)

FOREACH edge $e \in E$: $f(e) \leftarrow 0$.
$\Delta \leftarrow$ largest power of $2 \le C$.

WHILE $(\Delta \ge 1)$
$G_f(\Delta) \leftarrow \Delta$-residual network of G with respect to flow f.
WHILE (there exists an $s \to t$ path P in $G_f(\Delta)$)
$f \leftarrow$ AUGMENT(f, c, P).
Update $G_f(\Delta)$.
$\Delta \leftarrow \Delta / 2$.
RETURN f.

Δ-scaling phase

Capacity-scaling algorithm: proof of correctness

Assumption. All edge capacities are integers between 1 and C.

Invariant. The scaling parameter Δ is a power of 2.
Pf. Initially a power of 2; each phase divides Δ by exactly 2. ▪

Integrality invariant. Throughout the algorithm, every edge flow $f(e)$ and residual capacity $c_f(e)$ is an integer.
Pf. Same as for generic Ford-Fulkerson. ▪

Theorem. If capacity-scaling algorithm terminates, then f is a max flow.
Pf.
- By integrality invariant, when $\Delta = 1 \Rightarrow G_f(\Delta) = G_f$.
- Upon termination of $\Delta = 1$ phase, there are no augmenting paths.
- Result follows augmenting path theorem ▪

Capacity-scaling algorithm: analysis of running time

Lemma 1. There are $1 + \lfloor \log_2 C \rfloor$ scaling phases.
Pf. Initially $C/2 < \Delta \le C$. Δ decreases by a factor of 2 in each iteration. ▪

Lemma 2. Let f be the flow at the end of a Δ-scaling phase. Then, the max-flow value $\le val(f) + m\,\Delta$.
Pf. Next slide.

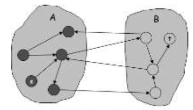
Lemma 3. There are $\le 2m$ augmentations per scaling phase. **Pf.**
- Let f be the flow at the beginning of a Δ-scaling phase. or equivalently, at the end of a 2Δ-scaling phase
- Lemma 2 $\Rightarrow$ max flow value $\le val(f) + m\,(2\Delta)$.
- Each augmentation in a Δ-phase increases $val(f)$ by at least Δ. ▪

Theorem. The capacity-scaling algorithm takes $O(m^2 \log C)$ time.
Pf.
- Lemma 1 + Lemma 3 $\Rightarrow$ $O(m \log C)$ augmentations.
- Finding an augmenting path takes $O(m)$ time. ▪

Capacity Scaling: Running Time

Lemma 2. Let f be the flow at the end of a Δ-scaling phase. Then value of the maximum flow is at most v(f) + m Δ.
Pf. (almost identical to proof of max-flow min-cut theorem)
- We show that at the end of a Δ-phase, there exists a cut (A, B) such that cap(A, B) ≤ v(f) + m Δ.
- Choose A to be the set of nodes reachable from s in $G_f(\Delta)$.
- By definition of A, $s \in A$.
- By definition of f, $t \notin A$.

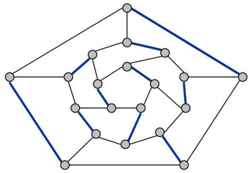
$$\begin{aligned} v(f) &= \sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to } A} f(e) \\ &\ge \sum_{e \text{ out of } A} (c(e) - \Delta) - \sum_{e \text{ in to } A} \Delta \\ &= \sum_{e \text{ out of } A} c(e) - \sum_{e \text{ out of } A} \Delta - \sum_{e \text{ in to } A} \Delta \\ &\ge cap(A, B) - m\Delta \quad ▪ \end{aligned}$$



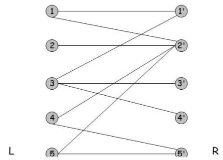
original network

- Generic augmenting path: O(m val(f*))
- Capacity scaling: O(m^2 log C).
  - C denotes the sum of capacities of all edges out of s.
- Shortest augmenting path: O(m^2n).
  - Proved by Dinitz (also by Edmonds and Karp)
- Preflow-Push algorithm: O(n^3)
  - Textbook 7.4
- Conclusion: Network Flow problem can be solved within polynomial time.

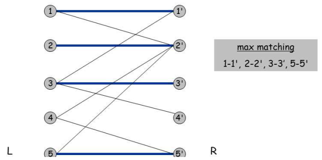
Matching.
- Input: undirected graph G = (V, E).
- M ⊆ E is a matching if each node appears in at most 1 edge in M.
- Max matching: find a max cardinality matching.



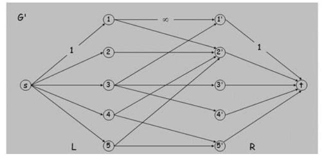
Bipartite matching.
- Input: undirected, bipartite graph G = (L ∪ R, E).
- M ⊆ E is a matching if each node appears in at most edge in M.
- Max matching: find a max cardinality matching.



Bipartite matching.
- Input: undirected, bipartite graph G = (L ∪ R, E).
- M ⊆ E is a matching if each node appears in at most one edge in M.
- Max matching: find a max cardinality matching.

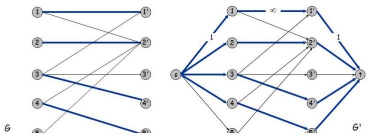


Max flow formulation.
- Create digraph G' = (L ∪ R ∪ {s, t}, E').
- Direct all edges from L to R, and assign infinite (or unit) capacity.
- Add source s, and unit capacity edges from s to each node in L.
- Add sink t, and unit capacity edges from each node in R to t.



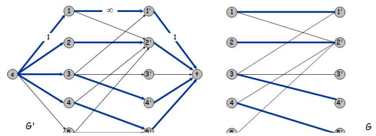
Bipartite Matching: Proof of Correctness

Theorem. Max cardinality matching in G = value of max flow in G'.
Pf. ≤
- Given max matching M of cardinality k.
- Consider flow f that sends 1 unit along each of k paths.
- f is a flow, and has cardinality k. ▪



Bipartite Matching: Proof of Correctness

Theorem. Max cardinality matching in G = value of max flow in G'.
Pf. ≥
- Let f be a max flow in G' of value k.
- Integrality theorem ⇒ k is integral and can assume f is 0-1.
- Consider M = set of edges from L to R with f(e) = 1.
  - each node in L and R participates in at most one edge in M
  - |M| = k: consider cut (L ∪ s, R ∪ t) ▪



Perfect Matching

Def. A matching M ⊆ E is perfect if each node appears in exactly one edge in M.

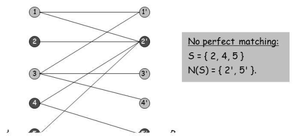
Q. When does a bipartite graph have a perfect matching?

Structure of bipartite graphs with perfect matchings.
- Clearly we must have |L| = |R|.
- What other conditions are necessary?
- What conditions are sufficient?

Perfect Matching

Notation. Let S be a subset of nodes, and let N(S) be the set of nodes adjacent to nodes in S.

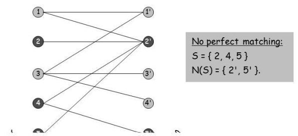
Observation. If a bipartite graph G = (L ∪ R, E), has a perfect matching, then |N(S)| ≥ |S| for all subsets S ⊆ L.
Pf. Each node in S has to be matched to a different node in N(S).



Marriage Theorem

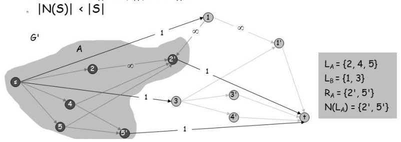
Marriage Theorem. [Frobenius 1917, Hall 1935] Let G = (L ∪ R, E) be a bipartite graph with |L| = |R|. Then, G has a perfect matching iff |N(S)| ≥ |S| for all subsets S ⊆ L.

Pf. ⇒ This was the previous observation.



Proof of Marriage Theorem

Pf. ⇐ Suppose G does not have a perfect matching.
- Formulate as a max flow problem and let (A, B) be min cut in G'.
- By max-flow min-cut, cap(A, B) < | L |.
- Define $L_A = L \cap A$, $L_B = L \cap B$, $R_A = R \cap A$.
- cap(A, B) = $|L_B| + |R_A|$.
- Since min cut can't use ∞ edges: $N(L_A) \subseteq R_A$.
- $|N(L_A)| \le |R_A| = \text{cap}(A, B) - |L_B| < |L| - |L_B| = |L_A|$.
- Choose $S = L_A$, $|N(L_A)| < |L_A|$,
- |N(S)| < |S|



Bipartite Matching: Running Time

Which max flow algorithm to use for bipartite matching?

- Generic augmenting path: O(m val(f*))
- Capacity scaling: O(m^2 log C).
  - C denotes the sum of capacities of all edges out of s.
- Shortest augmenting path: O(m^2n).
  - Proved by Dinitz (also by Edmonds and Karp)
- Preflow-Push algorithm: O(n^3)

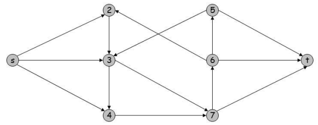
Edge Disjoint Paths

Disjoint path problem. Given a digraph G = (V, E) and two nodes s and t, find the max number of edge-disjoint s-t paths.

Def. Two paths are edge-disjoint if they have no edge in common.

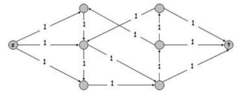
Ex: communication networks.

Class Exercise: Find the max number of edge-disjoint s-t paths!



Edge Disjoint Paths

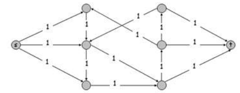
Max flow formulation: assign unit capacity to every edge.



Theorem. Max number edge-disjoint s-t paths equals max flow value.
Pf. ≤
- Suppose there are k edge-disjoint paths $P_1, \dots, P_k$.
- Set f(e) = 1 if e participates in some path P_i ; else set f(e) = 0.
- Since paths are edge-disjoint, f is a flow of value k. ▪

Edge Disjoint Paths

Max flow formulation: assign unit capacity to every edge.



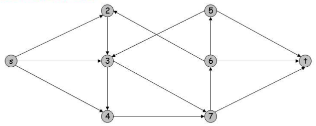
Theorem. Max number edge-disjoint s-t paths equals max flow value.
Pf. ≥
- Suppose max flow value is k.
- Integrality theorem ⇒ there exists 0-1 flow f of value k.
- Consider edge (s, u) with f(s, u) = 1.
  - by conservation, there exists an edge (u, v) with f(u, v) = 1
  - continue until reach t, always choosing a new edge
- Produces k (not necessarily simple) edge-disjoint paths. ▪

Network Connectivity

Network connectivity. Given a digraph G = (V, E) and two nodes s and t, find min number of edges whose removal disconnects t from s.

Def. A set of edges F ⊆ E disconnects t from s if every s-t path uses at least one edge in F.

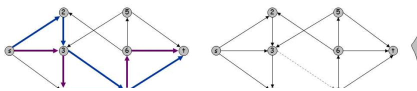
Class Exercise: Find the min number of edges whose removal disconnects t from s!



Edge Disjoint Paths and Network Connectivity

Theorem. [Menger 1927] The max number of edge-disjoint s-t paths is equal to the min number of edges whose removal disconnects t from s.

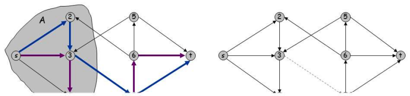
Pf. ≤
- Suppose the removal of F ⊆ E disconnects t from s, and |F| = k.
- Every s-t path uses at least one edge in F.
  Hence, the number of edge-disjoint paths is at most k. ▪



Disjoint Paths and Network Connectivity

Theorem. [Menger 1927] The max number of edge-disjoint s-t paths is equal to the min number of edges whose removal disconnects t from s.

Pf. ≥
- Suppose max number of edge-disjoint paths is k.
- Then max flow value is k.
- Max-flow min-cut ⇒ cut (A, B) of capacity k.
- Let F be set of edges going from A to B.
- |F| = k and disconnects t from s. ▪



Circulation with Demands

Circulation with demands.
- Directed graph G = (V, E).
- Edge capacities c(e), e ∈ E.
- Node supply and demands d(v), v ∈ V.

demand if d(v) > 0; supply if d(v) < 0; transshipment if d(v) = 0

Def. A circulation is a function that satisfies:
- For each e ∈ E: $0 \le f(e) \le c(e)$ (capacity)
- For each v ∈ V: $\sum_{e \text{ in to } v} f(e) - \sum_{e \text{ out of } v} f(e) = d(v)$ (conservation)

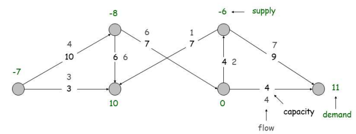
Circulation problem: given (V, E, c, d), does there exist a circulation?

Circulation with Demands

Necessary condition: sum of supplies = sum of demands.

$$\sum_{v : d(v) > 0} d(v) = \sum_{v : d(v) < 0} -d(v) =: D$$

Pf. Sum conservation constraints for every demand node v.



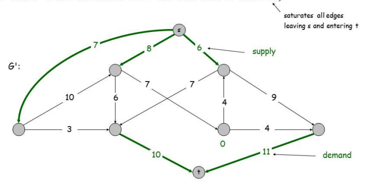
Circulation with Demands

Max flow formulation.

$$\sum_{v : d(v) > 0} d(v) = \sum_{v : d(v) < 0} -d(v) =: D$$

- Add new source s and sink t.
- For each v with d(v) < 0, add edge (s, v) with capacity -d(v).
- For each v with d(v) > 0, add edge (v, t) with capacity d(v).
- Claim: G has circulation iff G' has max flow of value D.

saturates all edges leaving s and entering t



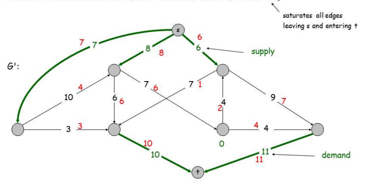
Circulation with Demands

Max flow formulation.

$$\sum_{v : d(v) > 0} d(v) = \sum_{v : d(v) < 0} -d(v) =: D$$

- Add new source s and sink t.
- For each v with d(v) < 0, add edge (s, v) with capacity -d(v).
- For each v with d(v) > 0, add edge (v, t) with capacity d(v).
- Claim: G has circulation iff G' has max flow of value D.

saturates all edges leaving s and entering t



Circulation with Demands

Characterization. Given (V, E, c, d), there does not exists a circulation iff there exists a node partition (A, B) such that $\Sigma_{v \in B} d_v > \text{cap}(A, B)$

Pf idea. Look at min cut in G'.

demand by nodes in B exceeds supply of nodes in B plus max capacity of edges going from A to B

Circulation with Demands and Lower Bounds

Feasible circulation.
- Directed graph G = (V, E).
- Edge capacities c(e) and lower bounds ℓ (e), e ∈ E.
- Node supply and demands d(v), v ∈ V.

Def. A circulation is a function that satisfies:
- For each e ∈ E: $\ell(e) \le f(e) \le c(e)$ (capacity)
- For each v ∈ V: $\sum_{e \text{ in to } v} f(e) - \sum_{e \text{ out of } v} f(e) = d(v)$ (conservation)

Circulation problem with lower bounds. Given (V, E, ℓ, c, d), does there exist an circulation?

Max-flow and Circulation Comparison

Max-flow
- G = (V, E) = directed graph,
- Two distinguished nodes:
  - s = source, t = sink.
- c(e) = capacity of edge e.
- $0 \le f(e) \le c(e)$
  $\sum_{e \text{ in to } v} f(e) = \sum_{e \text{ out of } v} f(e)$
- max flow = min cut
- Algorithms:
- Generic augmenting path:
  - O(m val(f*)).
- Capacity scaling:
  - O(m^2 log C)
- *Shortest augmenting path:
  - O(m^2n).
- * Preflow-Push:
  - O(m n^2) or O(n^3) .

Circulation with demands
- Node supply and demands d(v), v ∈ V.
  - demand if d(v) > 0;
  - supply if d(v) < 0;
  - transshipment if d(v) = 0
- Conservation $\sum_{e \text{ in to } v} f(e) - \sum_{e \text{ out of } v} f(e) = d(v)$
- Necessary condition to have a circulation
  $$\sum_{v : d(v) > 0} d(v) = \sum_{v : d(v) < 0} -d(v) =: D$$
- Convert to network flow:
  - Add new source s and sink t.
  - For each v with d(v) < 0, add edge (s, v) with capacity -d(v).
  - For each v with d(v) > 0, add edge (v, t) with capacity d(v).
  - Claim: G has circulation iff G' has max flow of value D (saturates all edges leaving s and entering t)
  - with Demands and Lower Bound:
    $\ell(e) \le f(e) \le c(e)$
- Transfer each edge e: (v, w):
- d(v)=d(v)+l(e); d(w)=d(w)-l(e); c(e)=c(e)-l(e)