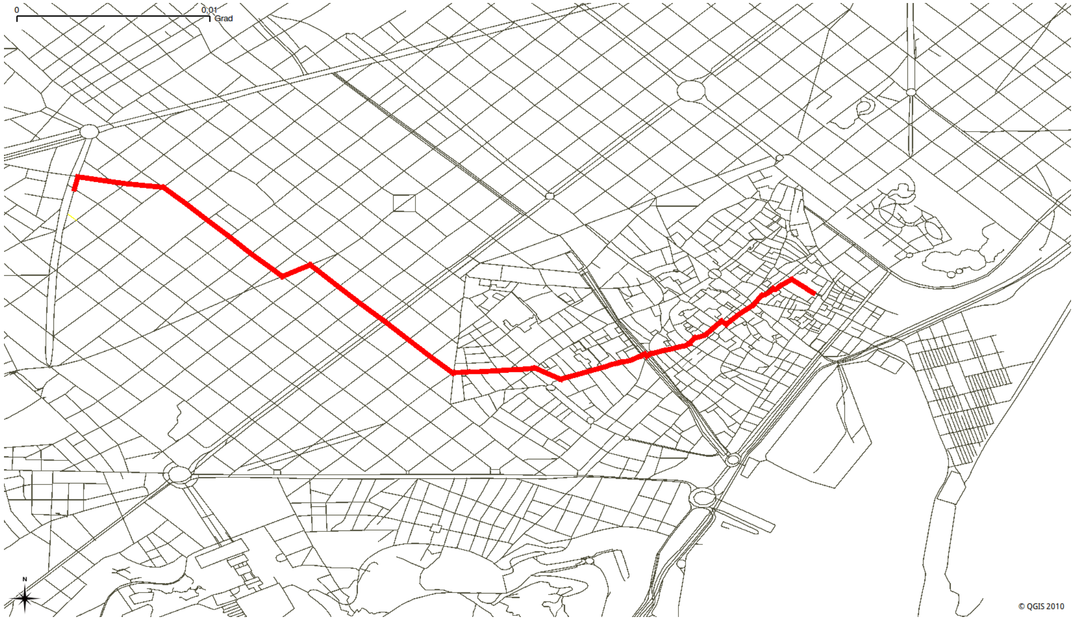
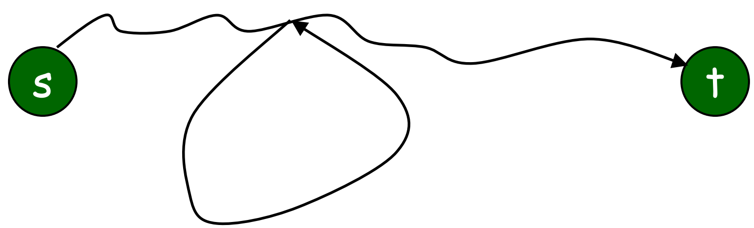
Lec. 1: Shortest Path & Min-Mean Circuit



Walk, Path, Circuits

- Walk: a traversal from vertex to vertex along edges.
- Trail: a walk without repeated edges
- Path: a walk without repeated internal vertices

- Cycle: closed trail
- Circuit: closed path

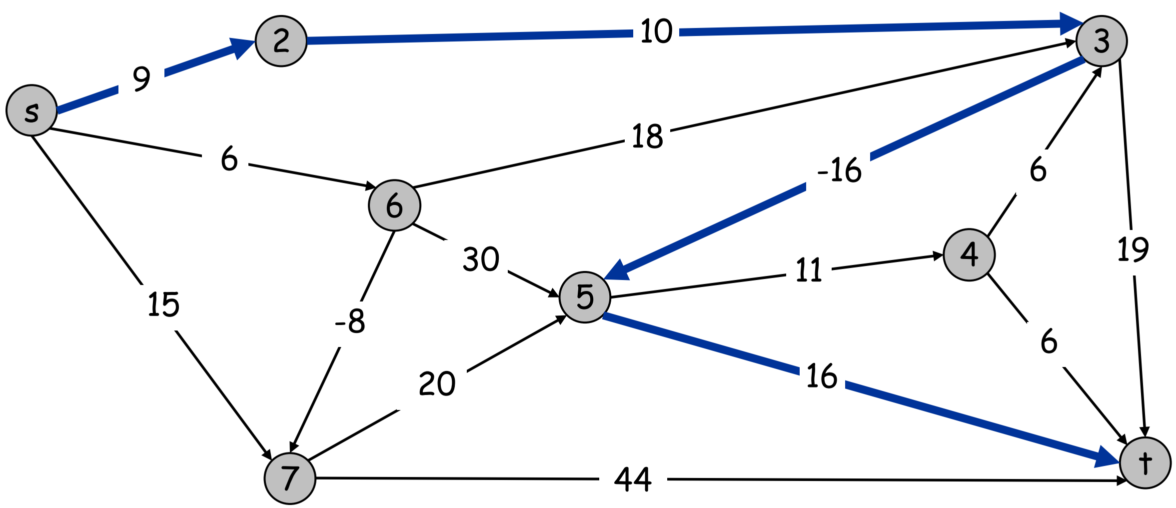


Shortest Paths

Shortest path problem. Given a directed graph $D = (V, A)$ with edge lengths (costs, weights) ℓ(u,v), find shortest path from node s to node t.

allow negative weights

Ex. Nodes represent agents in a financial setting and ℓ(u,v) is cost of transaction in which we buy from agent u and sell immediately to v.



NP-Completeness of Shortest Path

NP-complete: even if each arc has length -1. Equivalently, finding a longest path in a graph (with unit length arcs) is NP-complete.

Pf. Reduction from finding a Hamiltonian path

Remark: A shortest walk with at most (resp. exactly) k arcs can be computed in polynomial time.

NP-Completeness of Shortest Circuit

NP-complete: even if each arc has length –1. Equivalently, finding a a Hamiltonian circuit in a graph (with unit length arcs) is NP-complete.

Minimum-Mean Circuit: a circuit C with the least mean length $\ell(C)/|C|$. Solvable in polynomial time

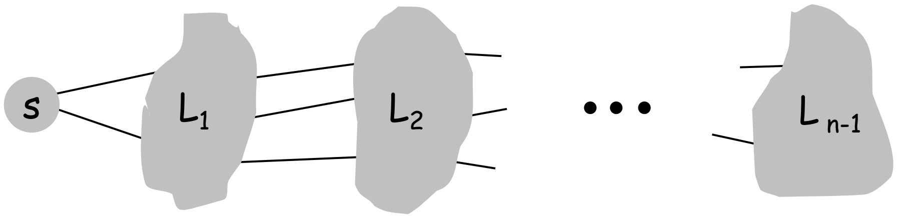
Outline

- Shortest path with unit lengths
- Shortest path with non-negative lengths
- Shortest Walk with arbitrary lengths
- All-pairs shortest paths
- Minimum-mean length directed circuit
- Elementary decomposition of circulations and transshipments

1. Shortest Path: Unit Lengths

SP with Unit Lengths: Breadth First Search

BFS intuition. Explore outward from s in all possible directions, adding nodes one "layer" at a time.



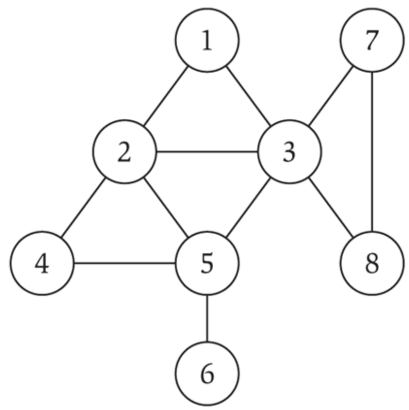
BFS algorithm.

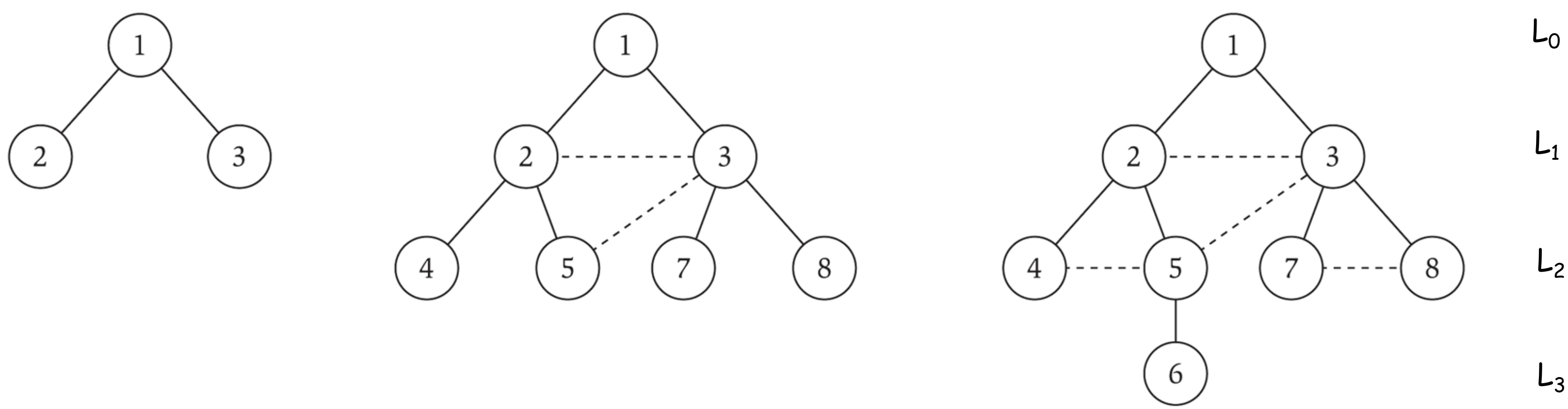
- $L_0 = \{ s \}$.
- L_1 = all neighbors of L_0.
- L_2 = all nodes that do not belong to L_0 or L_1, and that have an edge to a node in L_1.
- L_{i+1} = all nodes that do not belong to an earlier layer, and that have an edge to a node in L_i.

Theorem. For each i, L_i consists of all nodes at distance exactly i from s.

Shortest-Path Tree

Theorem. The A BFS tree is a shortest-path tree and can be computed in $O(m + n)$ time if the graph is given by its adjacency list.

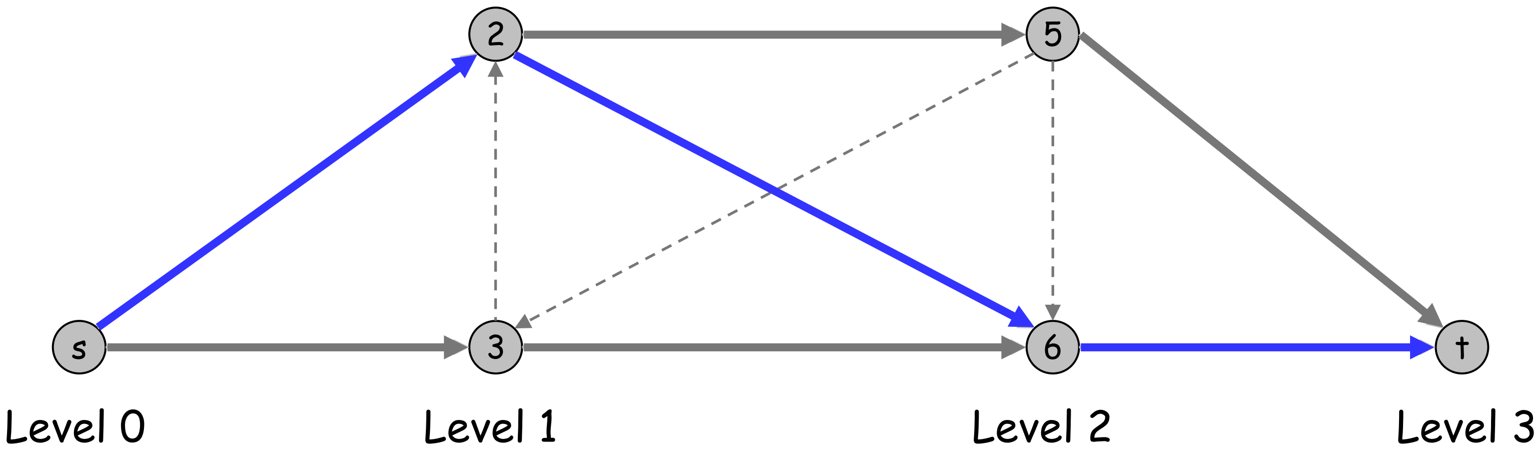




(a) (b) (c)

Level Graph

- Subgraph of D consisting of all vertices and edges appearing in some shortest s-t path in D
- Compute in $O(m+n)$ time using BFS by keeping only forward edges (deleting back and side edges).
- An inclusion-wise maximal collection of edge-disjoint (or vertex-disjoint) shortest s-t paths can be computed in $O(m+n)$ time (exercise).



2. Shortest-Path: Non-Negative Lengths

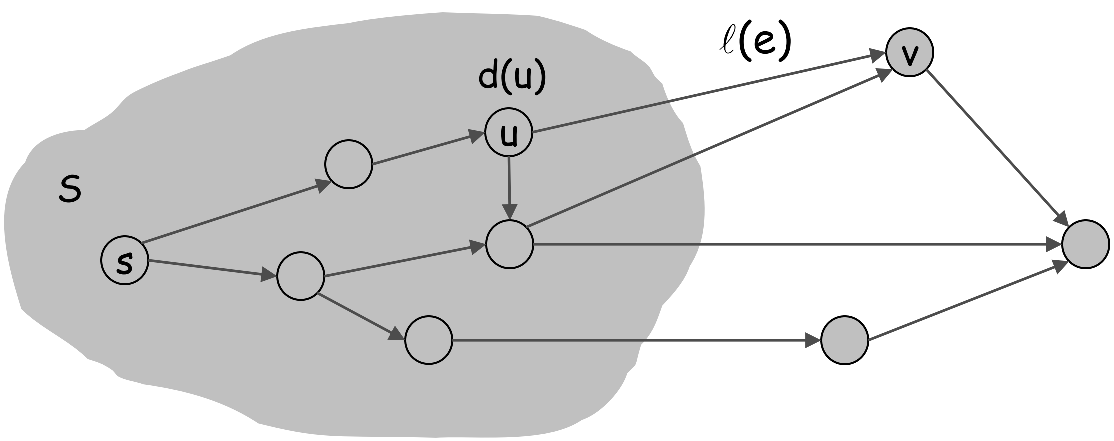
Dijkstra's (Greedy) Algorithm

- Maintain a set of explored nodes S for which we have determined the shortest path distance d(u) from s to u.
- Initialize S = { s }, d(s) = 0.
- Repeatedly choose unexplored node v which minimizes

$$\pi(v) = \min_{e = (u,v) : u \in S} d(u) + \ell(e),$$

shortest path to some u in explored part, followed by a single edge (u, v)

add v to S, and set d(v) = π(v).



Dijkstra's Algorithm: Push-Based Implementation

For each unexplored node, explicitly maintain $\pi(v) = \min_{e=(u,v):u \in S} d(u) + \ell(e)$.

- Next node to explore = node with minimum $\pi(v)$.
- When exploring v, for each e = (v, w) update

$$\pi(w) = \min \{ \pi(w), \pi(v) + \ell(e) \}.$$

Efficient implementation. Maintain a priority queue of unexplored nodes, prioritized by $\pi(v)$.

PQ Operation	Dijkstra	Array	Binary heap	d-way Heap	Fib heap †
Insert	n	n	log n	$d \log_d n$	1
ExtractMin	n	n	log n	$d \log_d n$	log n
ChangeKey	m	1	log n	$\log_d n$	1
IsEmpty	n	1	1	1	1
Total		n^2	m log n	$m \log_{m/n} n$	m + n log n

† Individual ops are amortized bounds

Reweighting Edges with Node Prices

p: a node price function

p-adjusted edge length: for each edge $a = (u, v)$

$$\ell_p(a) := \ell(a) - p(v) + p(u)$$

- purchase from u at price $p(u)$, ship to v at price $\ell(a)$, and sell to v at price $p(v)$

Invariant Properties:

- For each cycle C, $\ell_p(C) = \ell(C)$
- For each s-t walk/path P, $\ell_p(P) = \ell(P) - p(t) + p(s)$

Each cycle or s-t walk/path is shortest w.r.t. ℓ_p $\Leftrightarrow$ it is shortest w.r.t. ℓ.

Node Potential

Wish: ℓ_p is nonnegative s.t. Dijkstra's algorithm can be applied.

Potential p: $p(v) - p(u) \leq \ell(a)$ for each arc $a = (u, v)$;
$\Leftrightarrow$ ℓ_p is nonnegative.

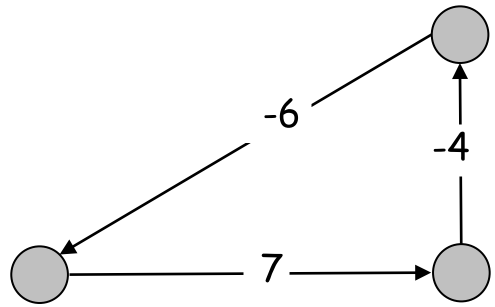
Theorem. There exists a potential $\Leftrightarrow$ each circuit is nonnegative.
If moreover ℓ is integer, the potential can be taken integer.

Pf. $\Leftarrow$ distance-based potential: $p(v)$:= the s-v distance.
$\Rightarrow$ trivial

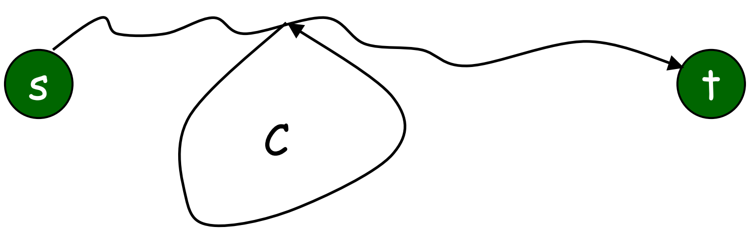
Remark: distance-based potential maximizes $p(t) - p(s)$ (string model)

3. Shortest Walks: Arbitrary Lengths

Negative Cycles



Observation. If some s-t walk contains a negative cycle, there does not exist a shortest s-t walk; otherwise, there exists one that is a s-t path.



$\ell(C) < 0$

Bellman-Ford Algorithm (DP)

Def. $d_i(v)$ = length of shortest v-t walk P using at most i edges.

$$d_0(v) = \begin{cases} 0 & \text{if } v = t \\ \infty & \text{otherwise} \end{cases}$$

For i>0

$$d_i(v) = \min\left\{ d_{i-1}(v), \min_{(v,w)\in A} \left\{ d_{i-1}(w) + \ell(vw) \right\} \right\}$$

Remark. By previous observation, if no negative cycles, then $d_{n-1}(v)$ = length of shortest v-t path.

Naïve Implementation

```
Shortest-Walk(D, t) {
   foreach node v ∈ V
      d[0, v] ← ∞
   d[0, t] ← 0

   for i = 1 to n-1
      foreach node v ∈ V
         d[i, v] ← d[i-1, v]
      foreach edge (v, w) ∈ A
         d[i, v] ← min { d[i, v], d[i-1, w] + ℓ(vw) }
}
```

Analysis. $\Theta(mn)$ time, $\Theta(n^2)$ space.

Finding the shortest walks. Maintain a "successor" for each table entry.

Space-Efficient Improvement with 1D-Table

- Maintain only 1D array d[v] = shortest v-t walk found so far.
- No need to check (v, w) unless d[w] changed in previous iteration.

```
Push-Based-Shortest-Walk(D, s, t) {
   foreach node v ∈ V
      d[v] ← ∞, successor[v] ← ϕ
   d[t] = 0

   for i = 1 to n-1
      foreach node w ∈ V
      if (d[w] has been updated in previous iteration)
         foreach node v such that (v, w) ∈ A
            if (d[v] > d[w] + ℓ(vw))
               d[v] ← d[w] + ℓ(vw) ,successor[v] ← w
      If no d[w] value changed in iteration i, stop.
```

Space-Efficient Improvement with 1D-Table

Theorem. Throughout the algorithm, each finite d[v] is length of some v-t walk; and after i rounds of updates, d[v] $\leq d_i(v)$.

Pf. By induction on i.

Overall impact.

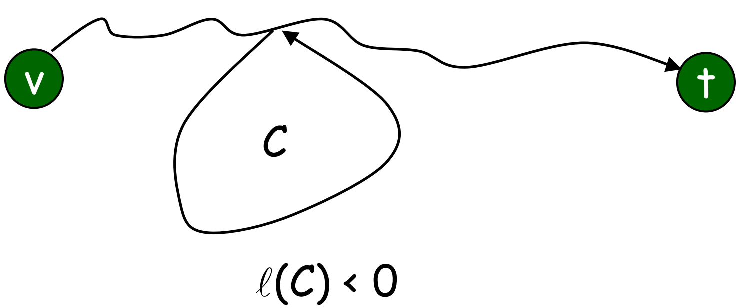
- Memory: O(m + n).
- Running time: O(mn) worst case, but substantially faster in practice

Detecting Negative Cycles

Lemma. If $d_n(v) = d_{n-1}(v)$ for all v, then no negative cycles.
Pf. By Bellman-Ford algorithm, $d_i(v) = d_{n-1}(v)$ for all $i \geq n-1$.

Lemma. If $d_n(v) < d_{n-1}(v)$ for some node v, then (any) shortest walk of n arcs from v to t contains a cycle C. Moreover, C has negative cost.

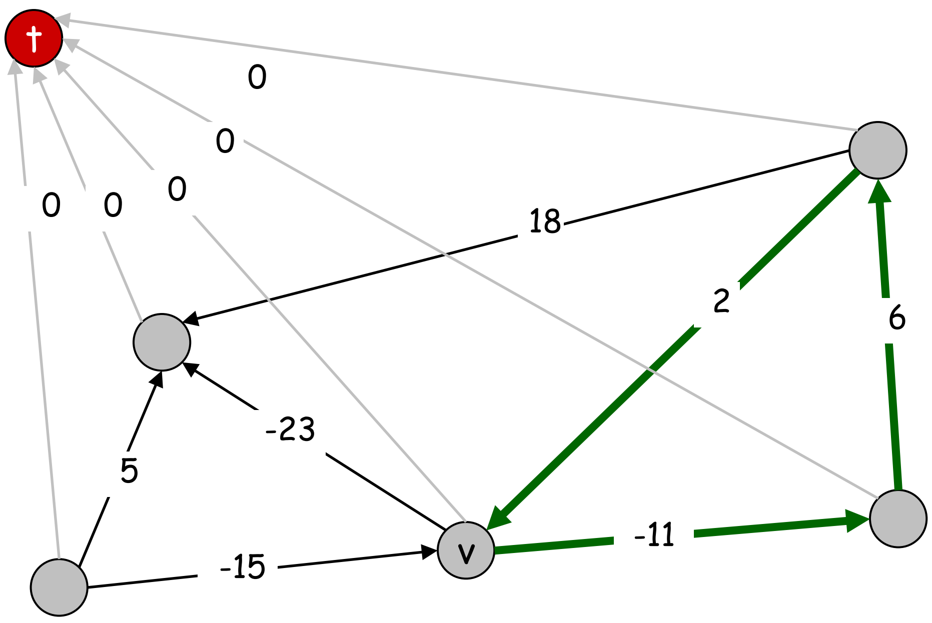
Pf. (by contradiction)
- Since $d_n(v) < d_{n-1}(v)$, we know P has exactly n edges.
- By pigeonhole principle, P must contain a directed cycle C.
- Deleting C yields a v-t path with < n edges $\Rightarrow$ C has negative cost.



Detecting Negative Cycles

Detect negative cycle in O(mn) time.

- Add new node t and connect all nodes to t with 0-length edge.
- Check if $d_n(v) = d_{n-1}(v)$ for all nodes v.
  - if yes, then no negative cycles
  - if no, then extract cycle from shortest walk from v to t



Detecting Negative Cycles: Summary

Bellman-Ford. O(mn) time, O(m + n) space.

- Run Bellman-Ford for n iterations (instead of n-1).
- Upon termination, Bellman-Ford successor variables trace a negative cycle if one exists.

Shortest k-Link Walks (DP)

Def. $d_i(v)$ = minimum length of a walk ending at v with exactly i arcs

$$d_i(v) = \begin{cases} 0 & \text{if } i = 0 \\ \min\limits_{(u,v)\in A} \{d_{i-1}(u) + \ell(uv)\} & \text{otherwise} \end{cases}$$

```
foreach v ∈ V
    d[0,v] ← 0
    predecessor[0,v] ← ϕ

for k = 1 to n
   foreach v ∈ V
      d[k,v] ← ∞
      foreach (u,v) ∈ A
         if d[k,v] > d[k-1,v] + ℓ(uv)) then
            d[k,v] ← d[k-1,v] + ℓ(uv)
            predecessor[k,v] ← u
```

4. All-Pairs Shortest Paths

Floyd-Warshall method (DP)

$v_1, v_2, \ldots, v_n$: an arbitrary vertex ordering

$d_k(s,t)$:= minimum length of an s-t walk using only vertices in $\{s, t, v_1, \ldots, v_k\}$.

$$d_0(s,t) = \begin{cases} \ell(s,t) & \text{if } (\mathrm{s},\mathrm{t}) \in \mathrm{A} \\ \infty & \text{otherwise} \end{cases}$$

$$d_k(s,t) = \min\{d_{k-1}(s,t), d_{k-1}(s,v_k) + d_{k-1}(v_k,t)\}$$

$$dist_\ell = d_n$$

Theorem. Under the condition of no negative-length circuit, all distances can be determined in time $O(n^3)$.

Space-Efficient Implementation with 2D-Table

```
FW-Shortest-Path(D)

   foreach (u,v) ∈ V x V
      if (u,v) ∈ A then
         d[u,v] ← ℓ(u,v), successor[u,v] ← v
      else
         d[u,v] ← ∞, successor[u,v] ← ϕ

   for k = 1 to n
      for i = 1 to n
         for j = 1 to n
            if d[i,j] > d[i,k] + d[k,j] then
               d[i,j] ← d[i,k] + d[k,j]
                successor[i,j] ← successor[i,k]
```

Theorem. Throughout the algorithm, each finite d[u,v] is length of some u-v walk; and after k rounds of updates, d[u,v] $\leq$ d_k(u,v).

Johnson's Algorithm (DP+Greedy)

Johnson's algorithm.

- Apply the Bellman-Ford method to find a distance-based potential p
- Reweight the lengths with p
- Apply Dijkstra's method to compute a shortest-path tree rooted at each other node

Analysis. $\Theta(n(m + n \log n))$ time.

5. Minimum-Mean Circuit

Minimum-Mean Circuit (MMC)

Def. mean length of a circuit $C := \ell(C)/|C|$.

Min-mean = the smallest value such that can be subtracted from each edge to ensure that each circuit becomes nonnegative.

MMC: invariant under uniform change on edge length

Assumption: D is strongly connected, for otherwise consider individual strong components.

Minimum Mean Length

Def. $d_i(v)$ = minimum length of a walk ending at v with exactly i arcs

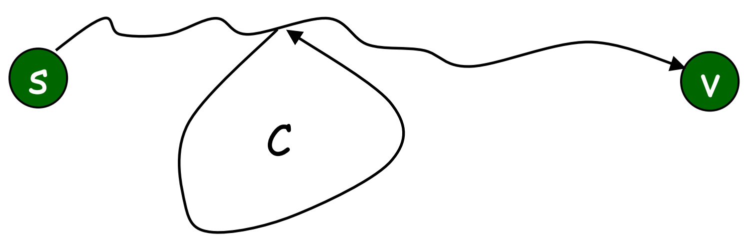
Theorem. The minimum mean length of a circuit is equal to

$$\min_{v \in V} \max_{0 \le i \le n-1} \frac{d_n(v) - d_i(v)}{n - i}.$$

Pf. W.l.o.g. assume MMC length = 0 (hence minimum circuit length =0).

$$\min_{v \in V} [d_n(v) - \min_{0 \le i \le n-1} d_n(v)] = 0.$$

- For all v, $d_n(v) \ge \min_{0 \le i \le n-1} d_n(v)$

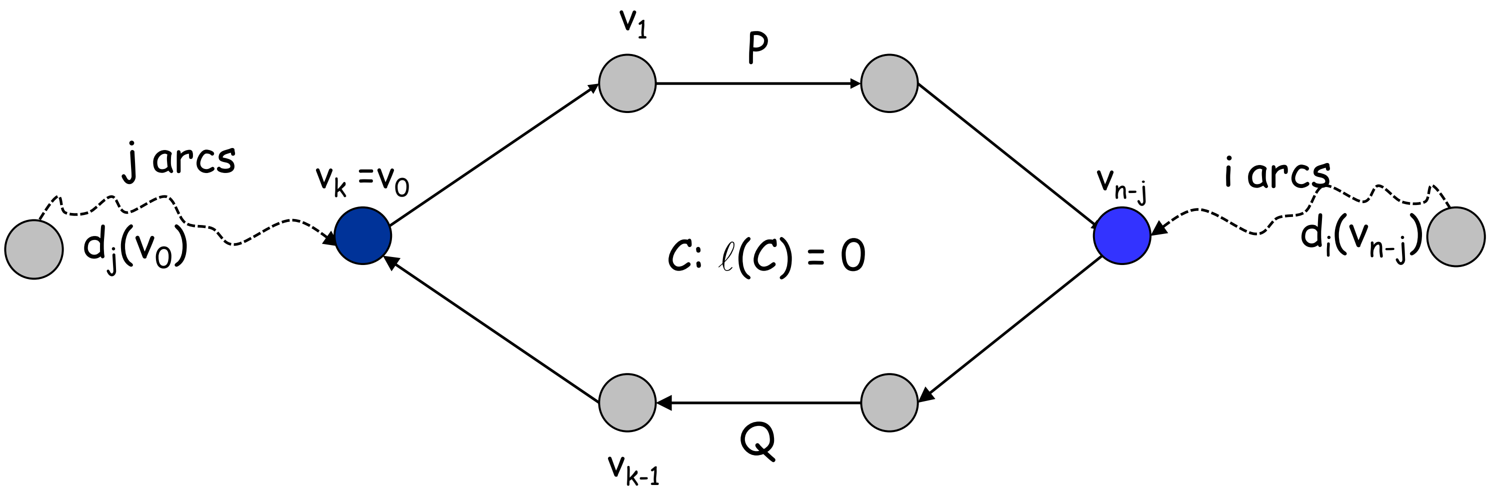


$\ell(C) \ge 0$

Minimum Mean Length

Pf. For some v, $d_n(v) \le \min_{0 \le i \le n-1} d_n(v)$

- $\min_j d_j(v_0)$ is attained by some j with $n-k \le j < n$.
- $\min_i d_i(v_{n-j})$ is attained by some i with $0 \le i < n$.



$$d_n(v_{n-j}) \le d_j(v_0) + \ell(P) \le d_i(v_{n-j}) + \ell(Q) + \ell(P) = d_i(v_{n-j})$$

Karp's Algorithm

- Compute $d_i(v)$ for all v and $0 \le i \le n$.
- Find v minimizing $\max\limits_{0 \le i \le n-1} \frac{d_n(v) - d_i(v)}{n-i}$

- Compute a shortest walk P ending at v with exactly n arcs,
- Find a circuit C in P and output C

P-C is a walk ending at v with i:= n-|C| arcs

$$\frac{\ell(C)}{|C|} = \frac{\ell(P) - \ell(P-C)}{n-i} \le \frac{d_n(v) - d_i(v)}{n-i}$$

Running time: O(mn)

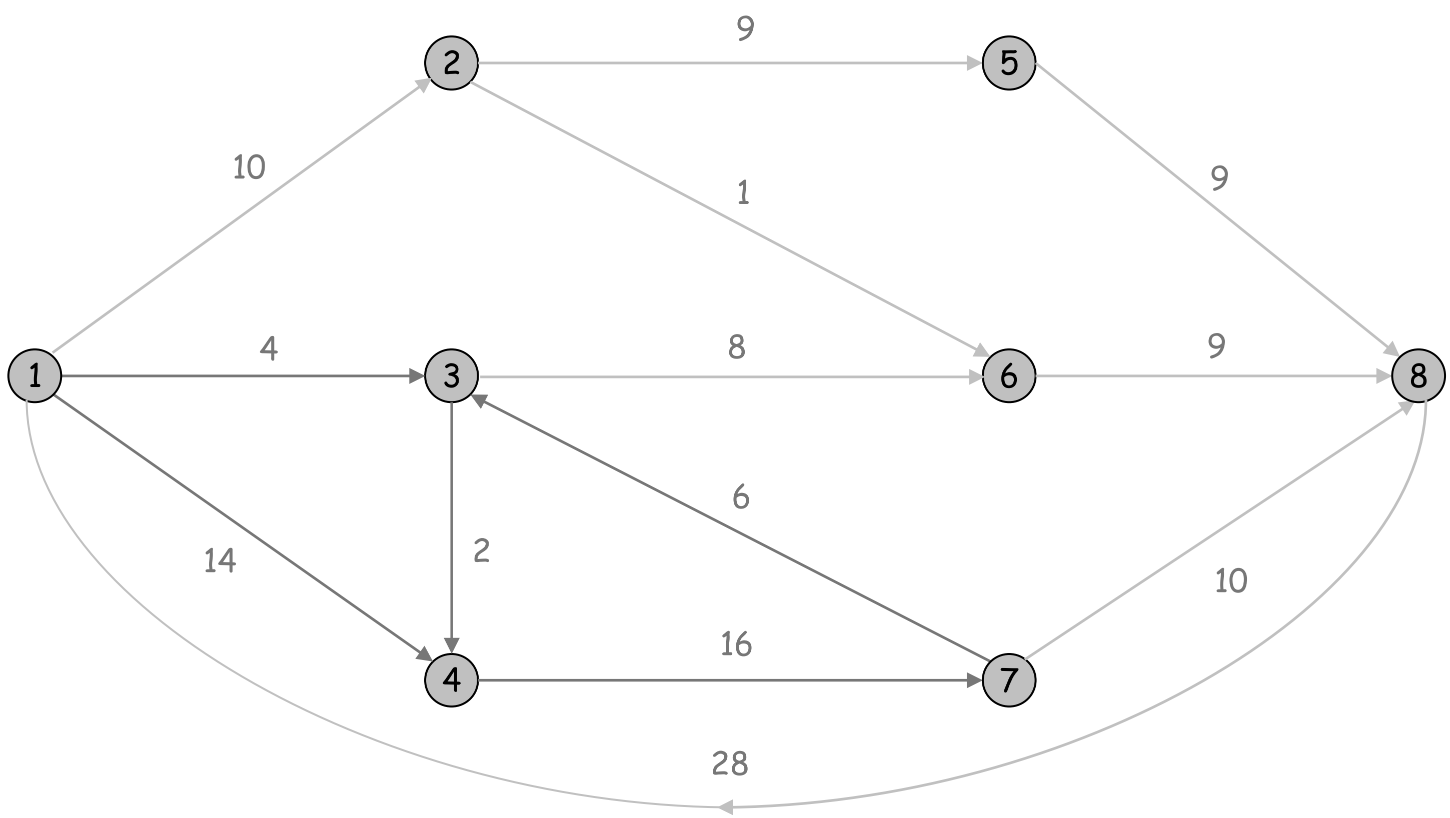
6. Decomposition of Circulations

Non-negative Circulation

$D = (V, A)$: a simple directed graph

x: a non-negative edge function

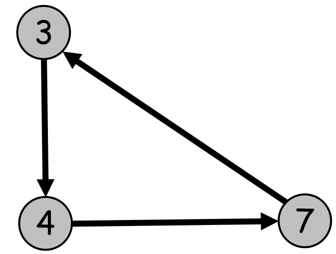
Def. x is a circulation if for any $v \in V$, $x\left(\delta^{in}(v)\right) = x\left(\delta^{out}(v)\right)$



Elementary Decomposition

Elementary circulation: χ_C along a circuit C in A

- $\chi_C(a) = 1$ for each $a \in C$;
- $\chi_C(a) = 0$ for each $a \notin C$.



$$A^+(x) := \{a \in A : x(a) > 0\}$$

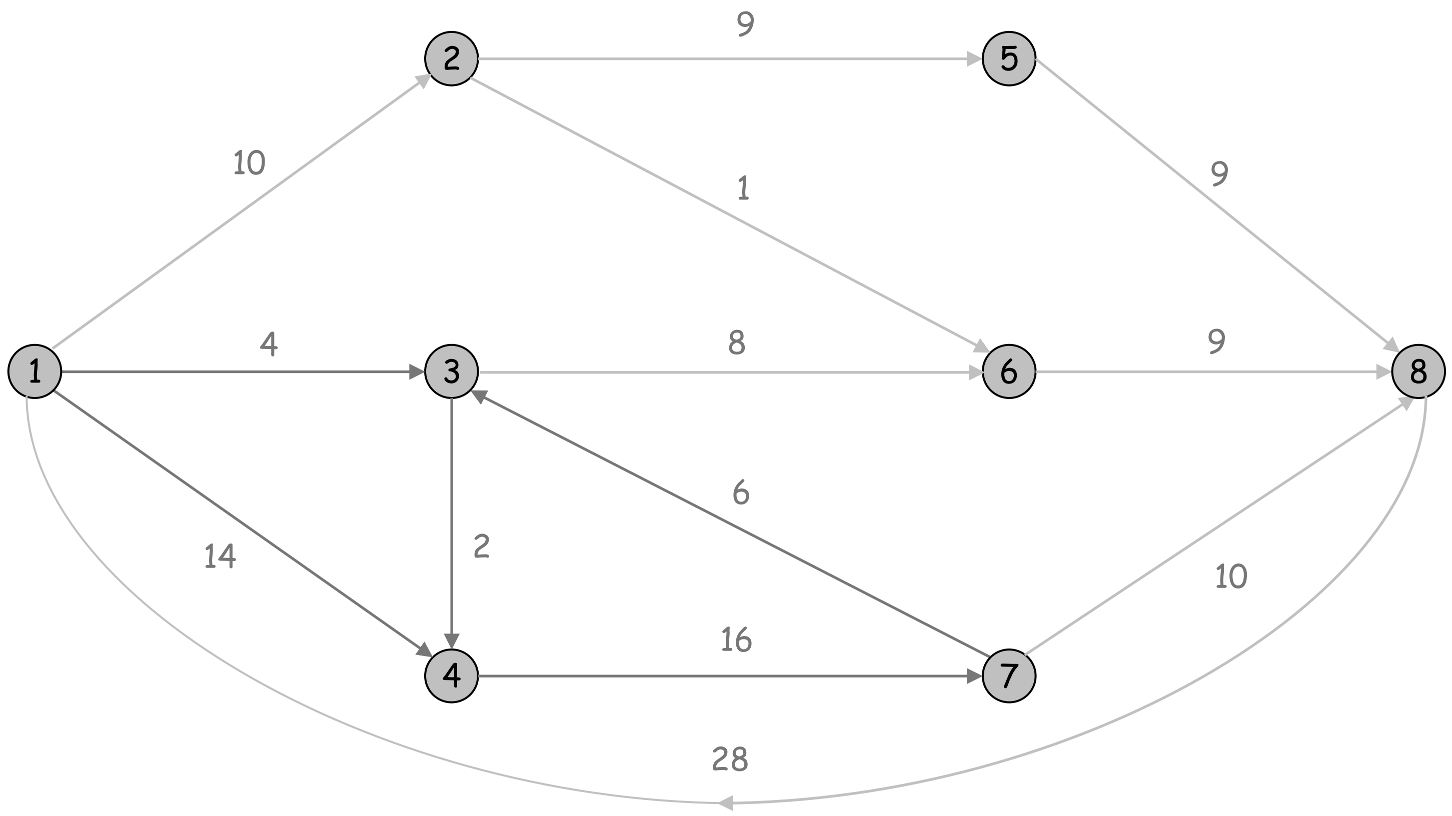
Theorem: There exist $k \leq |A^+(x)|$ circuits $C_1, \ldots, C_k$ in $A^+(x)$ together with k positive numbers $\varepsilon_1, \ldots, \varepsilon_k$ s.t.

$$x = \sum_{i=1}^{k} \varepsilon_i \chi_{C_i}.$$

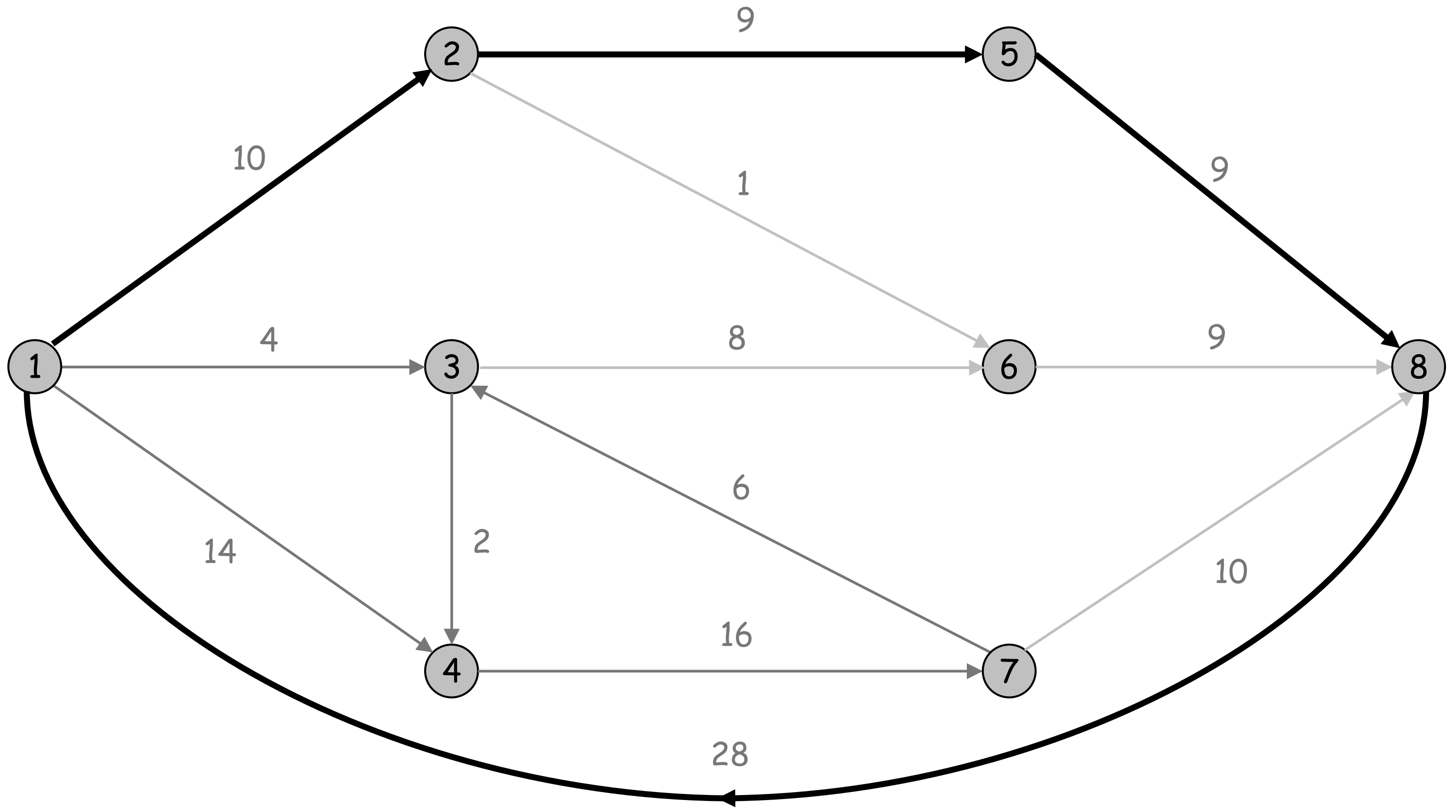
Moreover, if x is integer-valued, so are all ε-values.

The decomposition can be found in $O(nm)$ time.

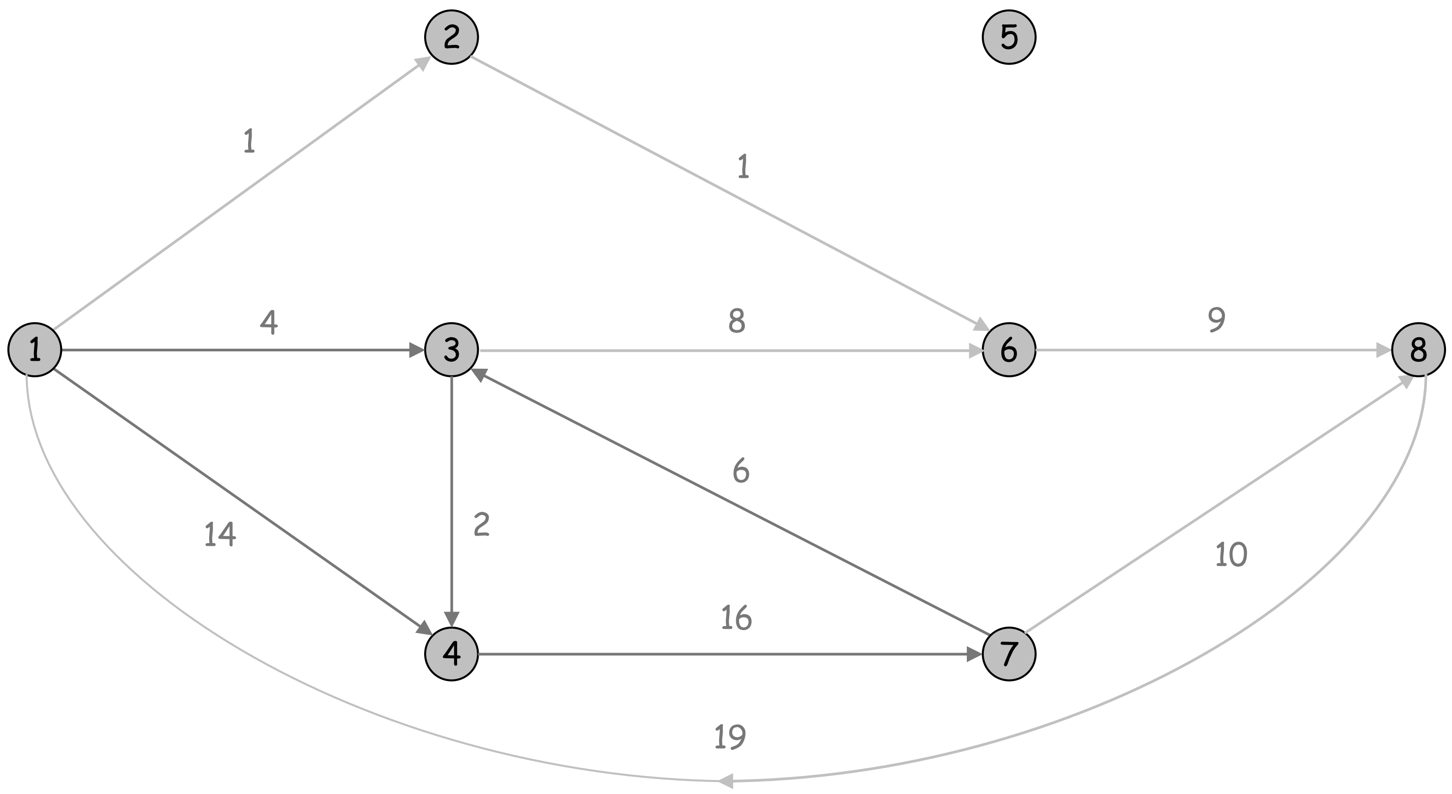
Elementary Decomposition of Circulation



Elementary Decomposition of Circulation

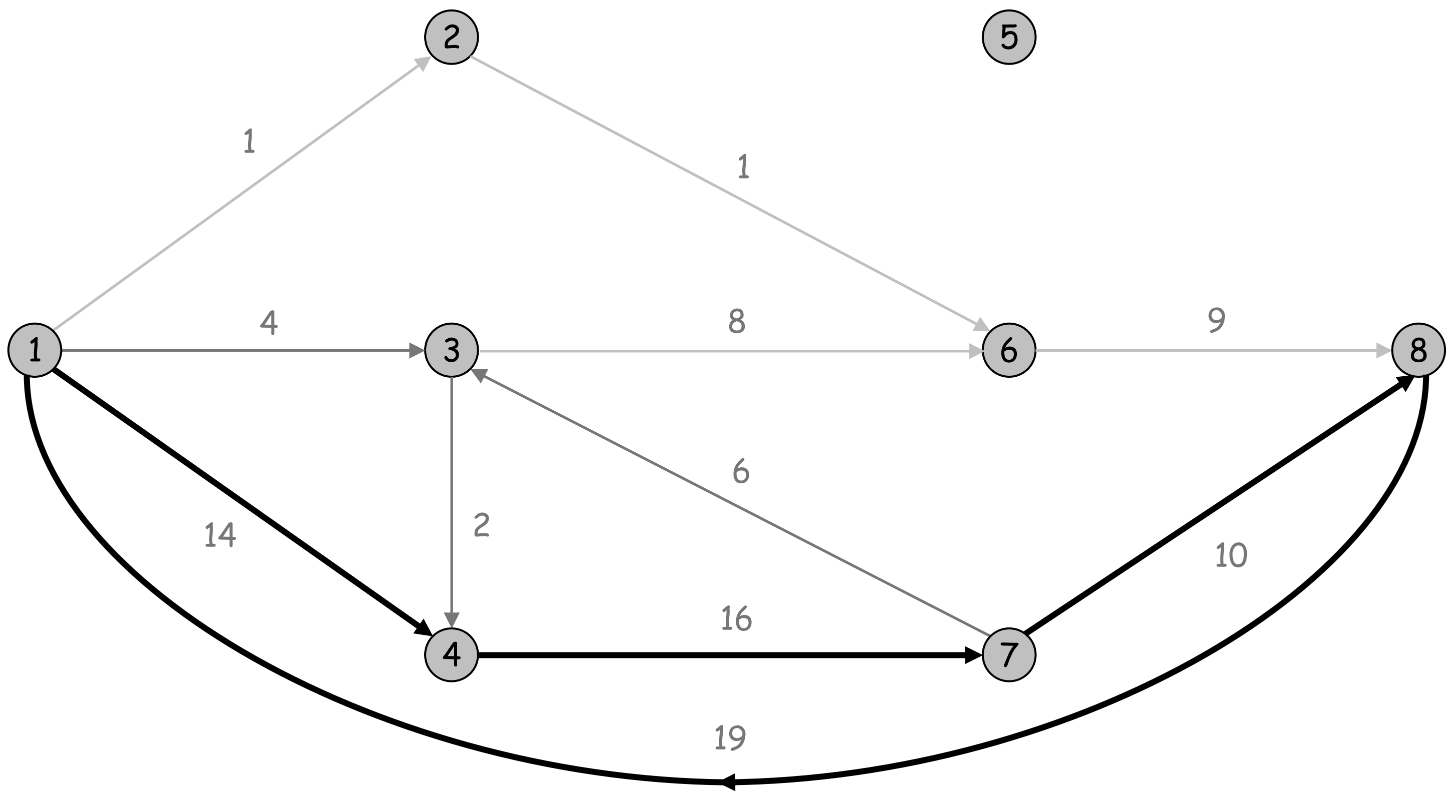


Elementary Decomposition of Circulation



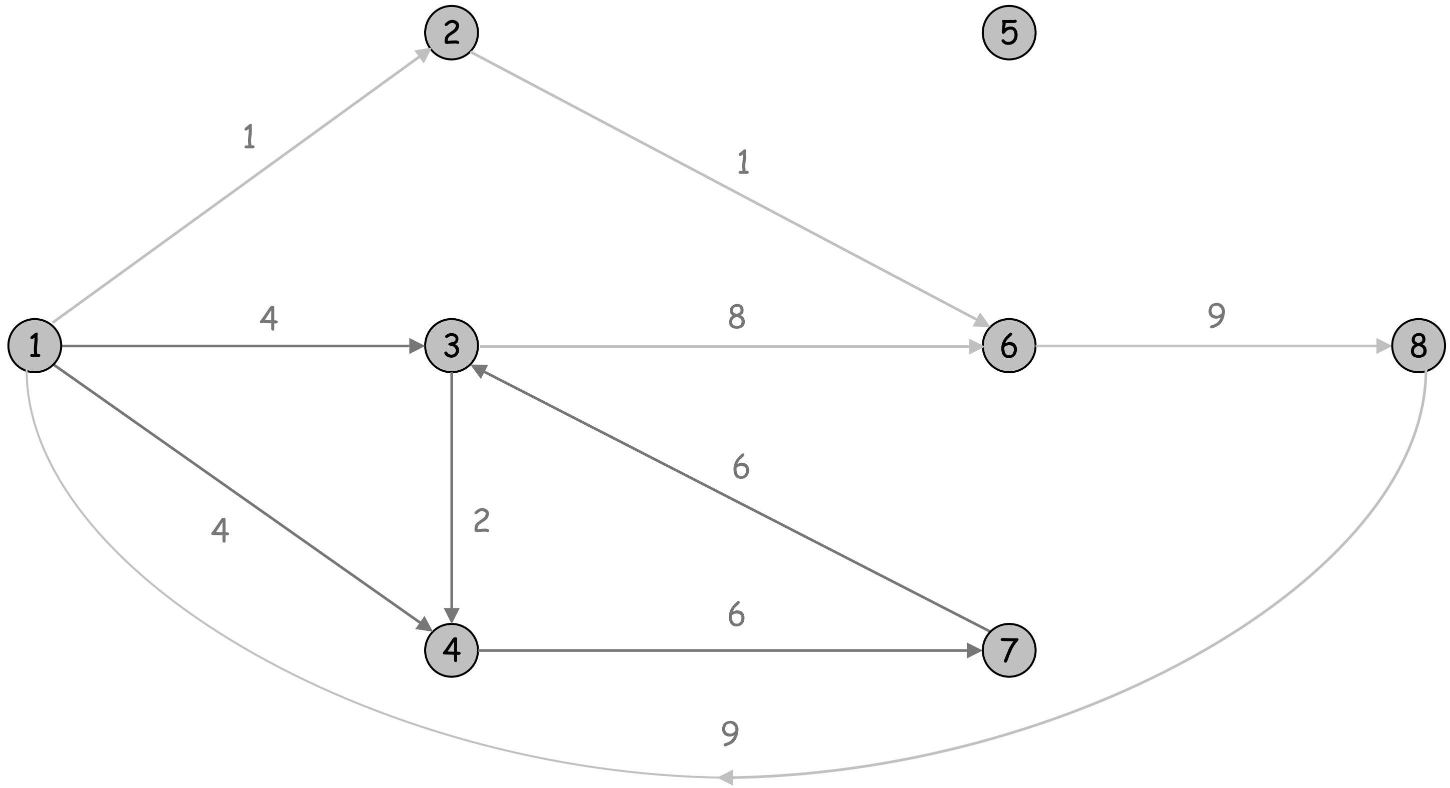
<1,2,5,8> : 9

Elementary Decomposition of Circulation



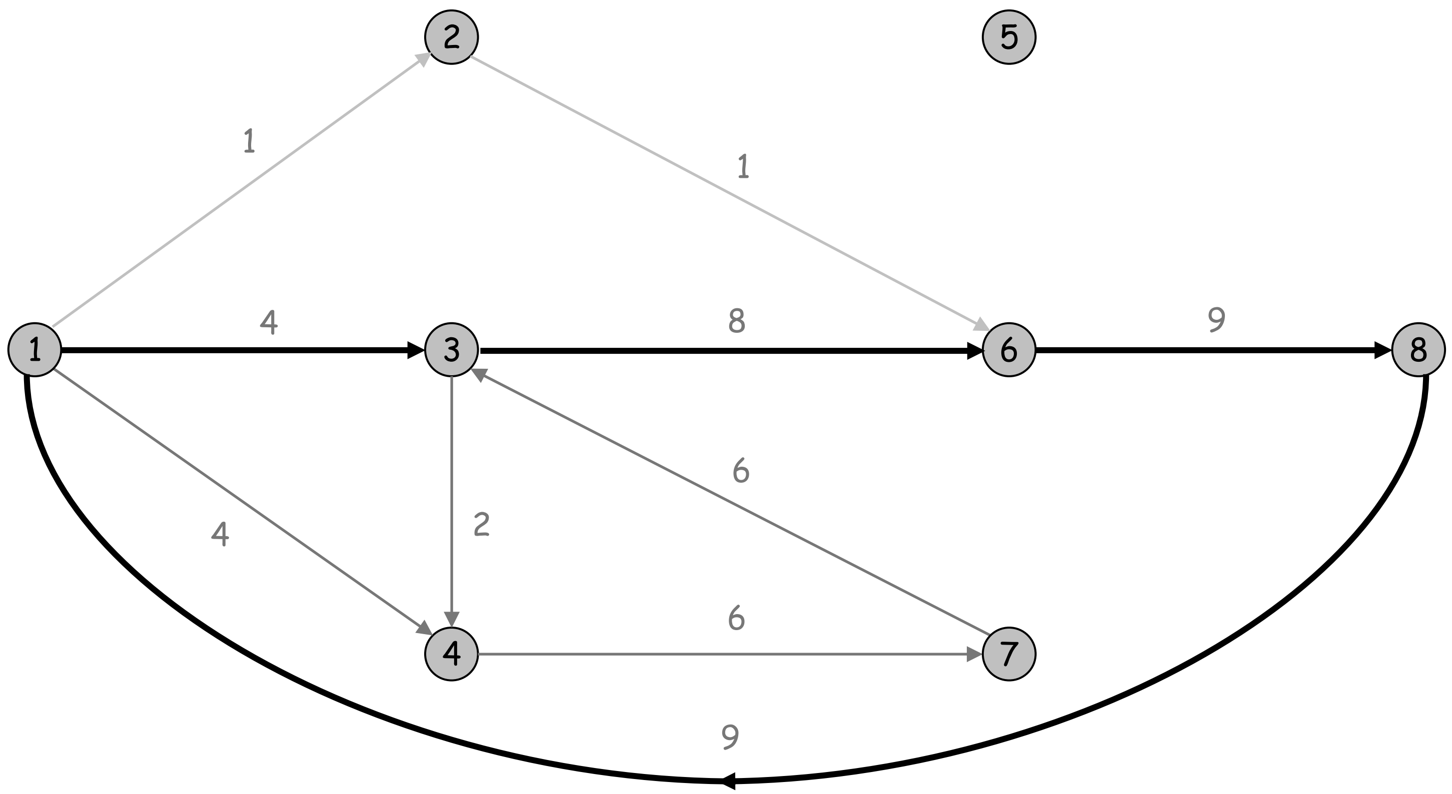
<1,2,5,8> : 9

Elementary Decomposition of Circulation



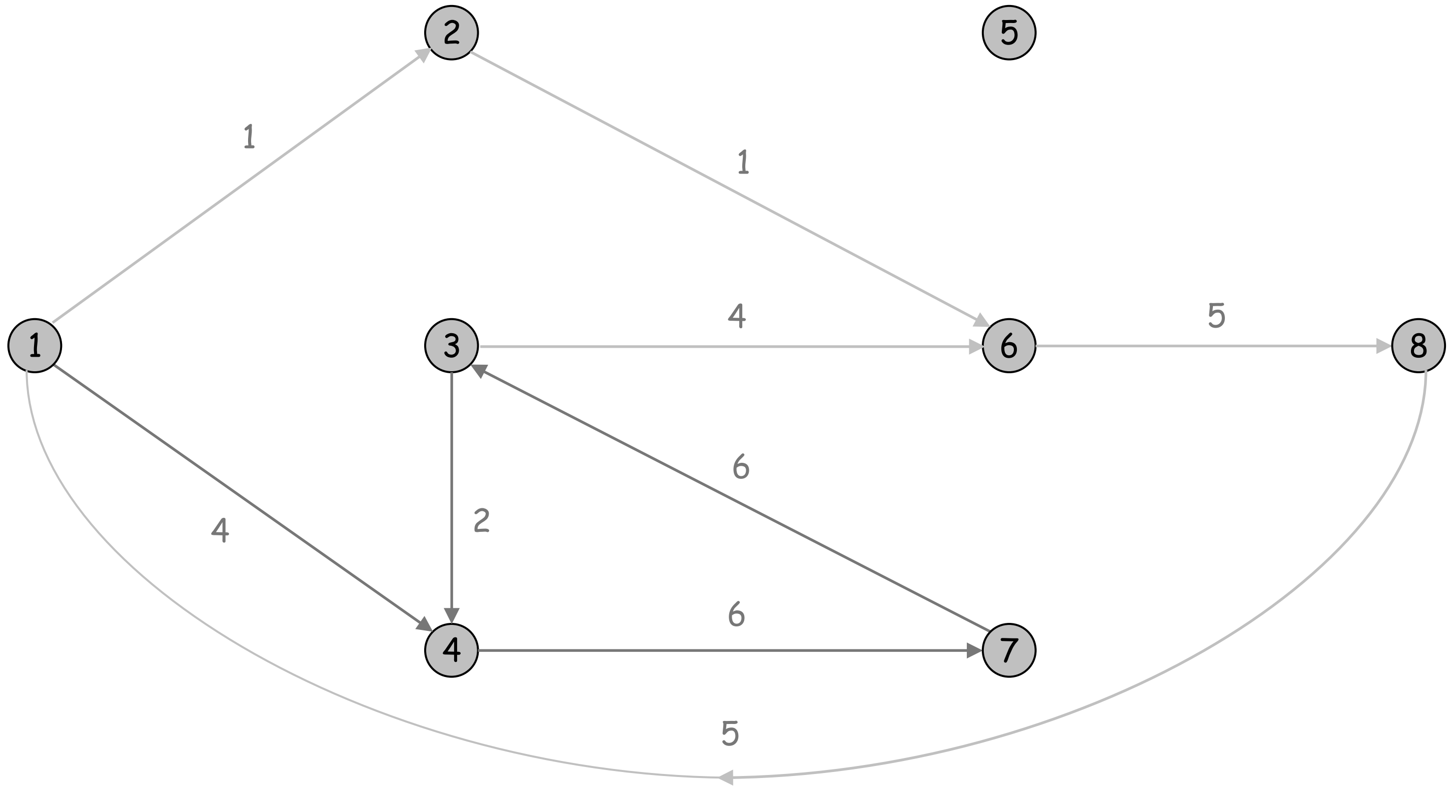
1,2,5,8 : 9
1,4,7,8 : 10

Elementary Decomposition of Circulation



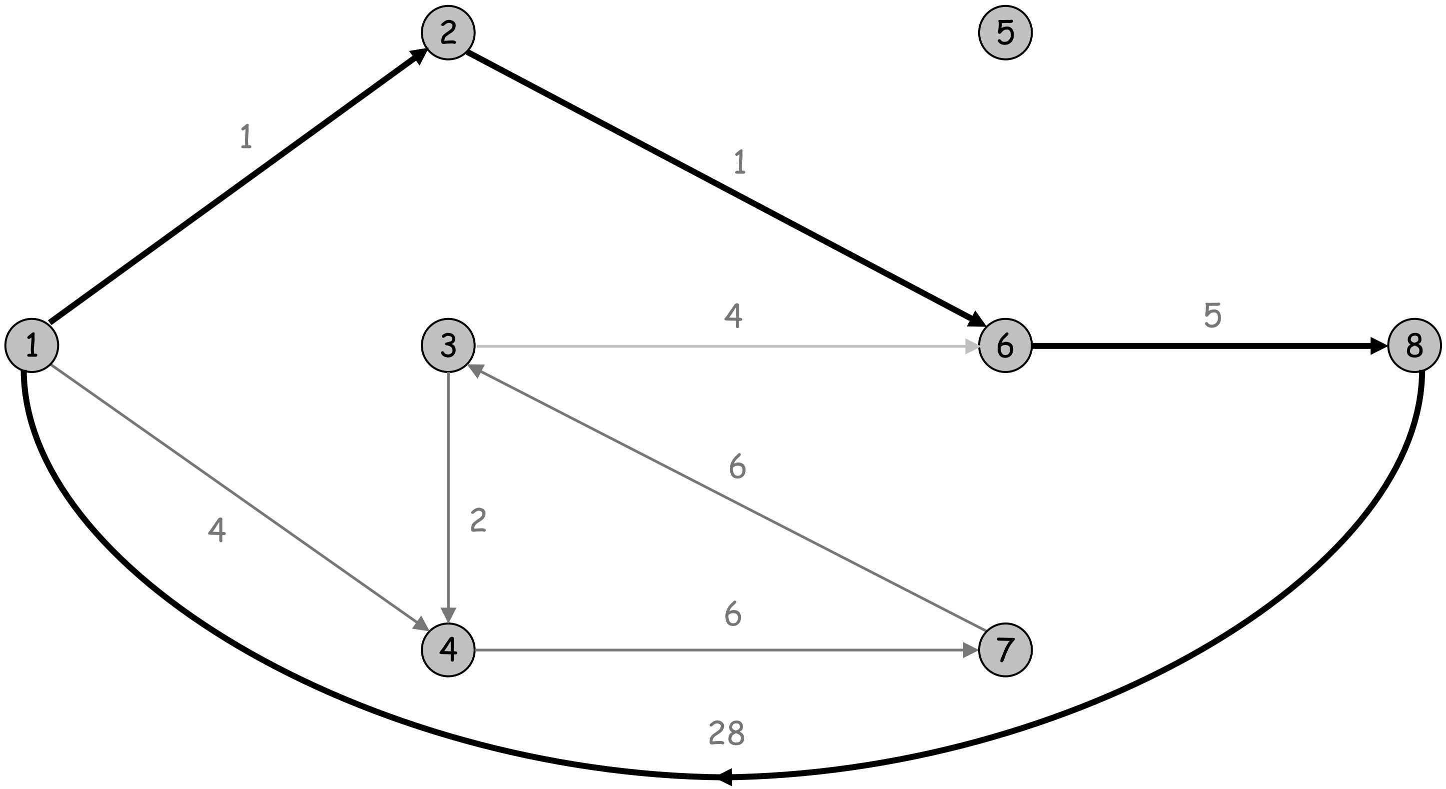
1,2,5,8 : 9
1,4,7,8 : 10

Elementary Decomposition of Circulation



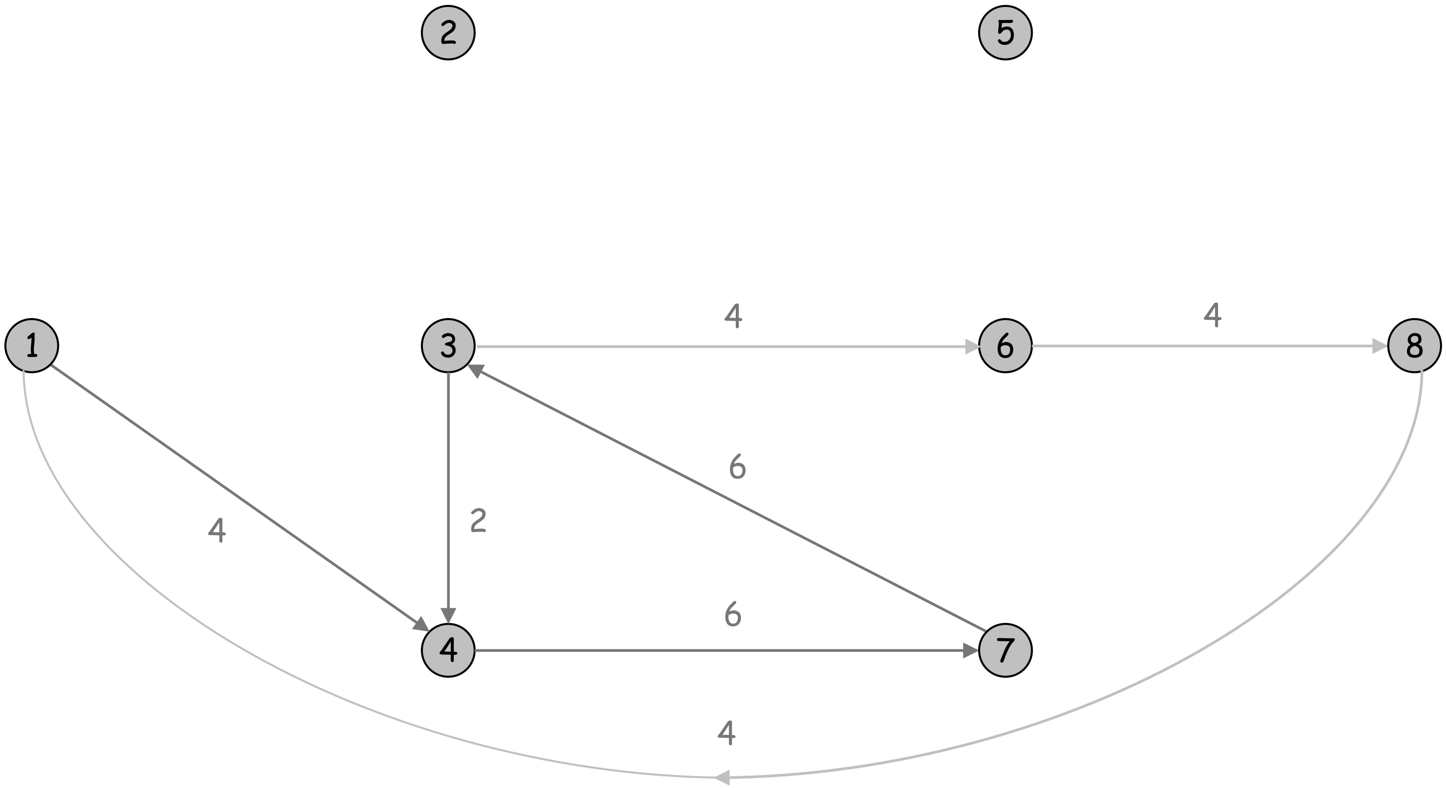
1,2,5,8 : 9
1,4,7,8 : 10
1,3,6,8 : 4

Elementary Decomposition of Circulation



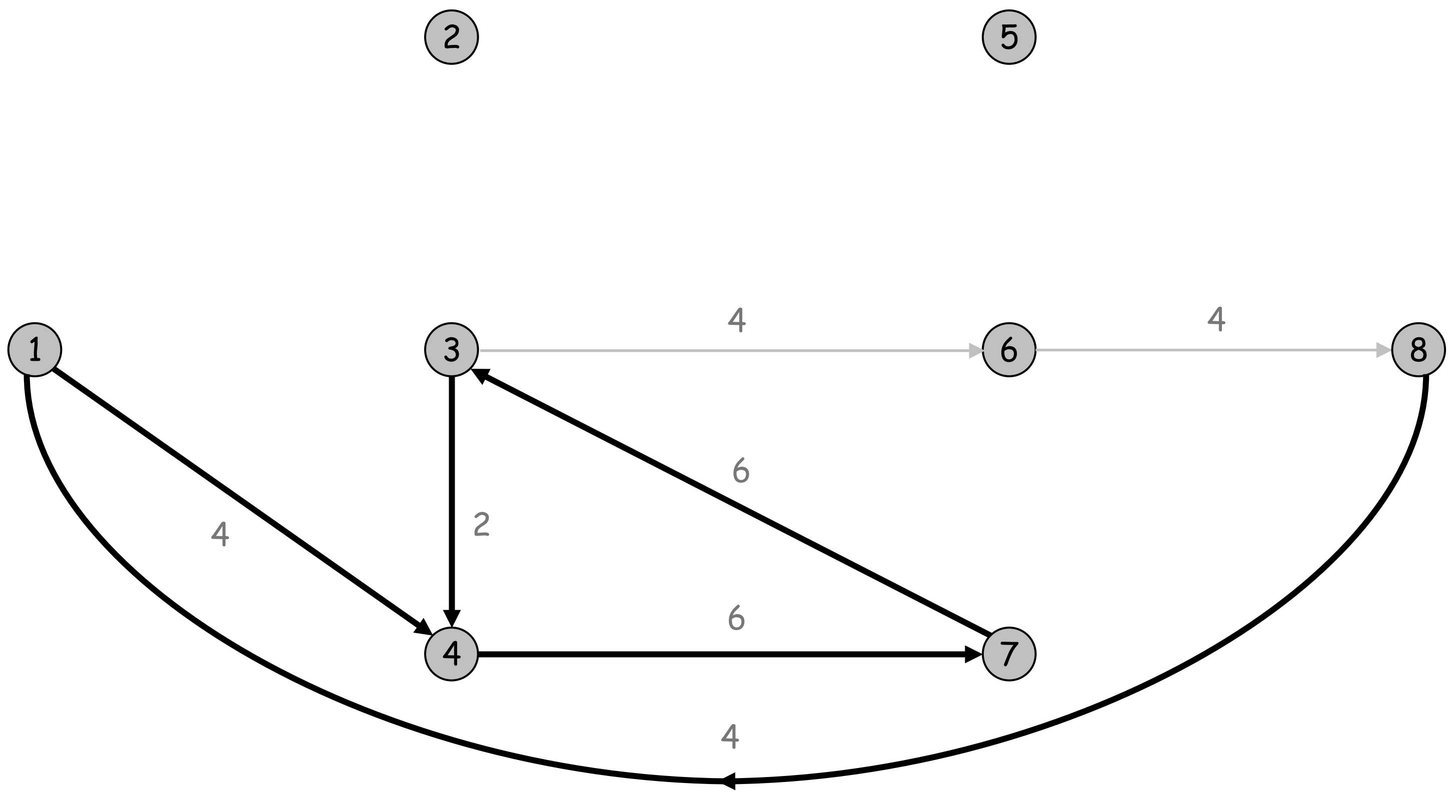
1,2,5,8 : 9
1,4,7,8 : 10
1,3,6,8 : 4

Elementary Decomposition of Circulation



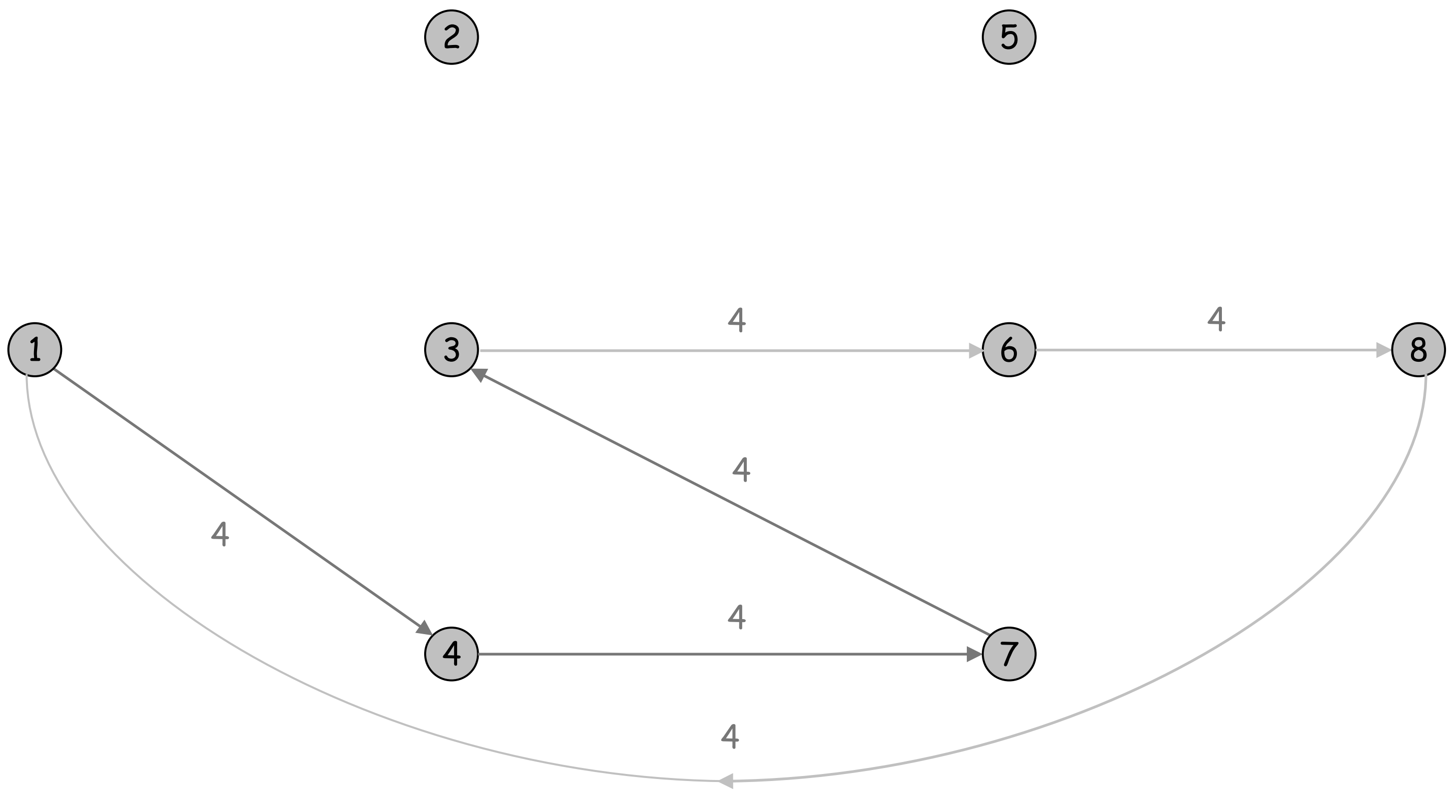
1,2,5,8 : 9
1,4,7,8 : 10
1,3,6,8 : 4
1,2,6,8 : 1

Elementary Decomposition of Circulation



1,2,5,8 : 9
1,4,7,8 : 10
1,3,6,8 : 4
1,2,6,8 : 1

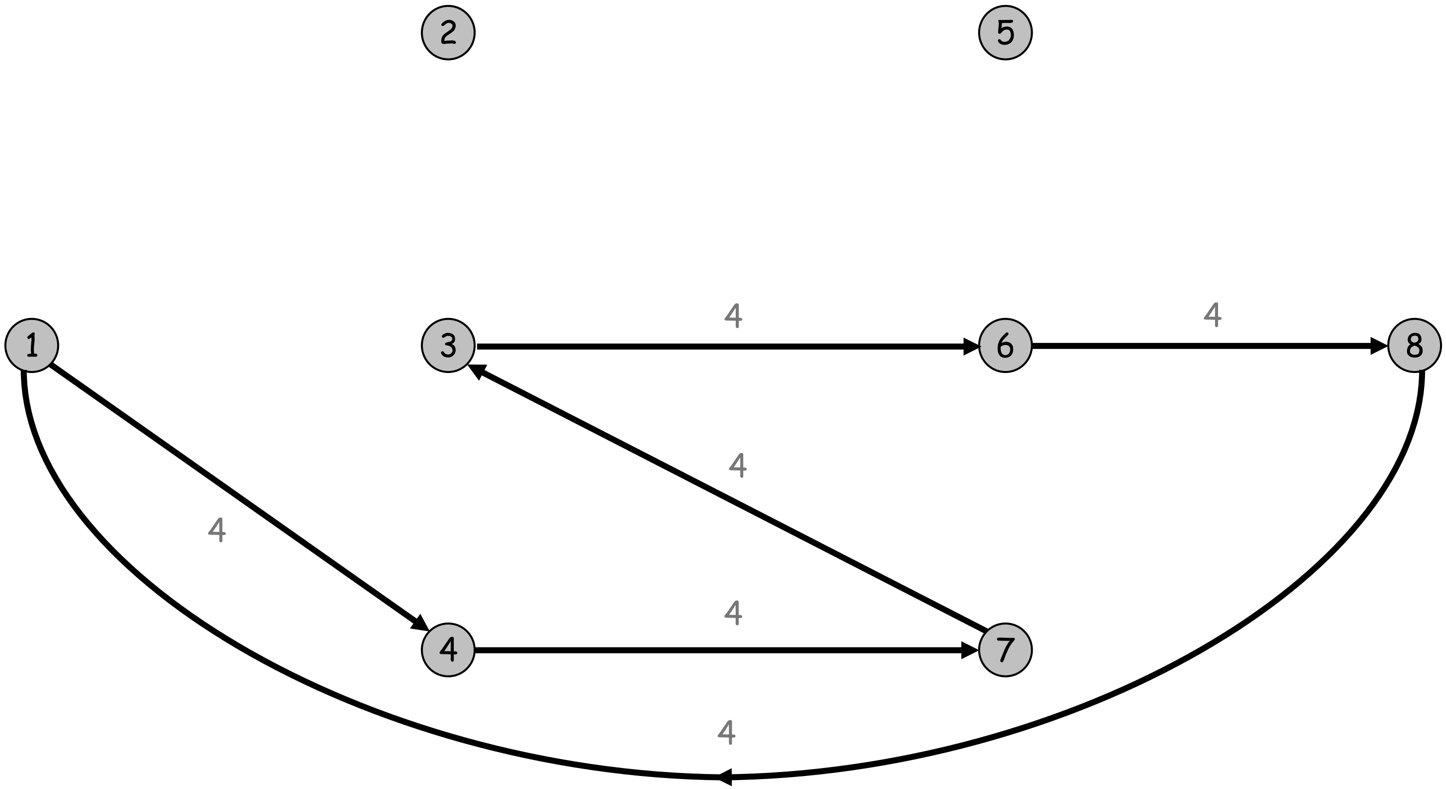
Elementary Decomposition of Circulation



1,2,5,8 : 9
1,4,7,8 : 10
1,3,6,8 : 4
1,2,6,8 : 1

3,4,7 : 2

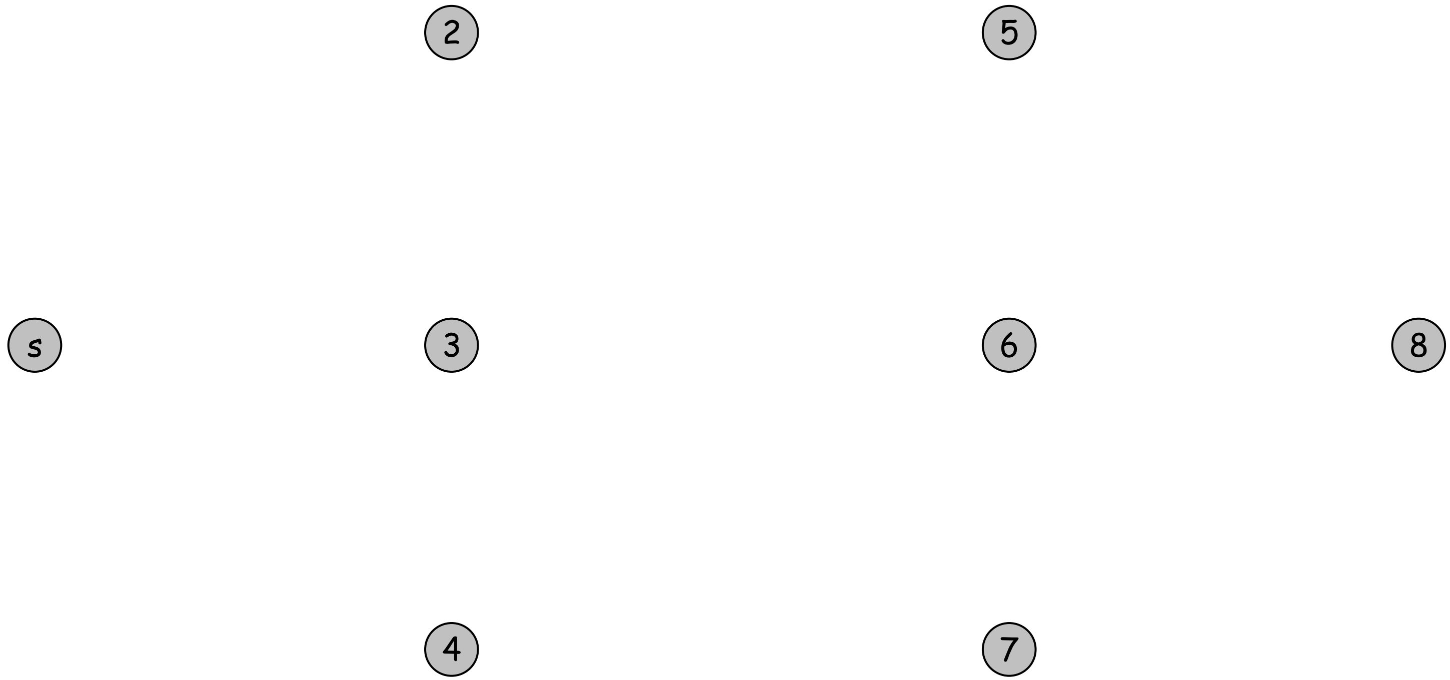
Elementary Decomposition of Circulation



1,2,5,8 : 9
1,4,7,8 : 10
1,3,6,8 : 4
1,2,6,8 : 1

3,4,7 : 2

Elementary Decomposition of Circulation



1,2,5,8 : 9
1,4,7,8 : 10
1,3,6,8 : 4
1,2,6,8 : 1
1,4,7,3,6,8 : 4

3,4,7 : 2

Non-negative Transshipment

$D = (V, A)$: a simple directed graph
x: a non-negative edge function

Def. $\mathrm{excess}(v) := x\left(\delta^{in}(v)\right) - x\left(\delta^{out}(v)\right)$ is called the x-excess of v; and its additive inverse $\mathrm{deficit}(v)$ is called the x-deficit of v;

Def. A node $v \in V$ is said to be excessive (resp, deficient, balanced) if it has a positive (resp, negative, 0) excess.

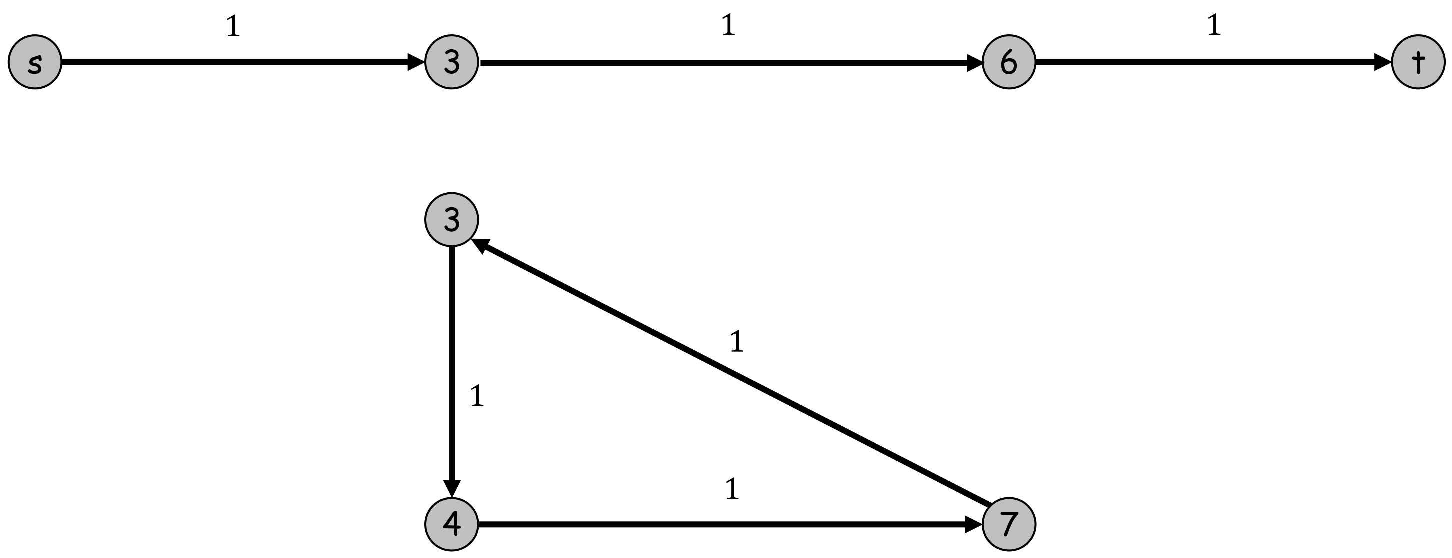
Def. If $\mathrm{excess}(v) = b(v)$ for $v \in V$, then x is called a b-TS

Def. The value of a b-TS x: $|x| := (1/2)\|b\|_1$
= total excesses of excessive nodes,
= total deficits of deficient nodes.

Elementary Transshipment

Elementary TS: χ_P along a path or circuit P in A

- $\chi_P(a) = 1$ for each $a \in P$;
- $\chi_P(a) = 0$ for each $a \notin P$.



Elementary Decomposition of TS

$$A^+(x) \coloneqq \{a \in A : x(a) > 0\}$$

Theorem: There exist a collection $\mathcal{P}$ of paths and a collection $\mathcal{C}$ of circuits in $A^+(x)$, with positive amounts ε s.t.

- $x = \sum_{P \in \mathcal{P} \cup \mathcal{C}} \varepsilon(P) \chi_P$;
- each path $P \in \mathcal{P}$ is from a deficient node to an excessive node;
- $|\mathcal{P}| + |\mathcal{C}| \leq |A^+(x)| \leq m$;
- moreover, if x is integer-valued, so are all $\varepsilon(P)$.

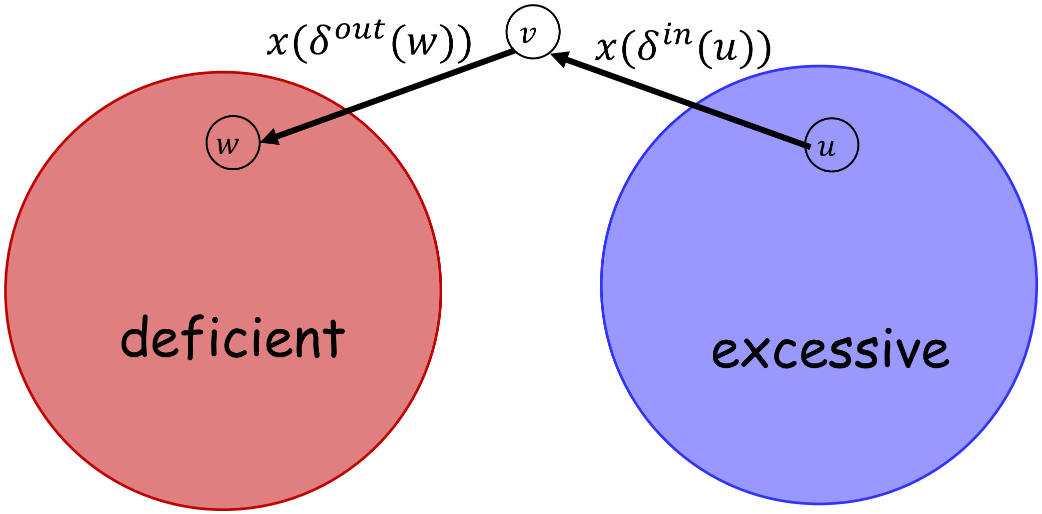
The decomposition can be found in $O(nm)$ time. Furthermore,

$$|x| = \sum_{P \in \mathcal{P}} \varepsilon(P)$$

Convert to Circulation

If not a circulation,

- add a new vertex v;
- add an arc (u, v) for each excessive u, put $x(u, v) \coloneqq x(\delta^{in}(u))$;
- add an arc (v, w) for each deficient w, put $x(v, w) \coloneqq x(\delta^{out}(w))$.



The new x is a circulation, and a decomposition of the new x gives a decomposition of the original x.

Summary

Single-source shortest paths

- Unit lengths: BFS in O(m+n) time
- Non-negative lengths: Dijkstra's algorithm in O(m+nlogn) time
- No-negative circuits: Bellman-Ford algorithm in O(mn) time, or Dijkstra's algorithm with a given potential

All-to-all shortest paths

- Floyd-Washall algorithm: $O(n^3)$ time
- Johnson's algorithm: O(n(m+nlogn)) time.

Minimum mean circuit

- Karp's algorithm: O (mn) time.

Elementary decomposition of circulations and transshipments

Further Topics

k shortest paths

https://en.wikipedia.org/wiki/K_shortest_path_routing

Shortest trail

- Special case of min-cost flow

Shortest path in Monge DAG: linear time and space

- Shortest k-link path: $O(\sqrt{k(n-k)}\sqrt{n\log(n-k)})$ time and linear space

Shortest path in undirected graphs without negative circuits

- Reduction to weighted non-bipartite matching