

ADS2 Exam Notes – Week 4 Network Flows (Max-Flow/Min-Cut)

| Field | Value |
|--------------------|--|
| Title | ADS2 Exam Notes – Network Flows (Ford-Fulkerson, Min-Cut, Properties, Applications) |
| Date | 2025-10-18 (Europe/Copenhagen) |
| Author | ADS2 compact notes |
| Sources used | network1.pdf (weekplan, p.1-2), flow1-4x1.pdf (slides), week4_exer_7_1_and_7_2.pdf (KT 7.1-7.2 figures), week4_exer_7_4_and_7_5.pdf (KT 7.4-7.5 statements), figures Pasted image.png , Pasted image (2).png |
| Week plan filename | network1.pdf |

General Methodology and Theory

- **Flow network** on a directed graph with capacity $c(u, v) \geq 0$. A **flow** f obeys capacity and conservation except at s, t ; value $|f| = \sum_v f(s, v) = \sum_v f(v, t)$.
- **Residual graph** G_f with forward capacity $c(u, v) - f(u, v)$ and backward capacity $f(u, v)$.
- **Augmenting path** = any $s \rightarrow t$ path in G_f ; push by bottleneck $\delta = \min c_f(\cdot)$.
- **Ford-Fulkerson** (FF) repeats augmentation until no path in G_f . With BFS paths (**Edmonds-Karp**) the time is $O(nm^2)$.
- **Cut** (S, T) with $s \in S, t \in T$ has capacity $c(S, T) = \sum_{u \in S, v \in T} c(u, v)$. Always $|f| \leq c(S, T)$; equality holds for max flow and min cut (theorem).

Micro-pseudocode (slides-first):

```
Algorithm: edmonds_karp
Input: G=(V,E), c, s, t
Output: max flow f, value |f|
initialize f=0
while (parent ← bfs_augmenting_path(G,c,f,s,t)) ≠ ⊥:
    δ ← bottleneck along parent path
    augment f by δ along the path (use back-edges)
return f, |f|
// Time: O(n m^2); Space: O(m)
```

Notes

- When capacities are integers, FF terminates and increases $|f|$ by at least 1 per augmentation.
 - To **certify optimality fast** in small instances: after the last augmentation, let S be the vertices reachable from s in G_f ; then $c(S, T) = |f|$ is a min cut. This avoids redrawing the entire residual graph on paper.
 - Avoid transcribing edge lists; instead record a short **augmentation trace** and a **cut certificate**.
-

Coverage Table

| Weekplan ID | Canonical ID | Title/Label (verbatim) | Assignment Source | Text Source | Status |
|-------------|--------------|--|-----------------------|--|------------------|
| 1 | KT 7.1 & 7.2 | Max flow / min cut (two small graphs) | network1.pdf §1 p.1 | week4_exer_7_1_and_7_2.pdf (figures) | ✓ solved |
| 2 | — | Ford-Fulkerson on given graph | network1.pdf §2 p.1 | Pasted image.png (exercise figure) | ✓ solved |
| 3 | KT 7.4 | Properties of maximum flows | network1.pdf §3 p.1 | week4_exer_7_4_and_7_5.pdf (statement) | ✓ solved |
| 4 | KT 7.5 | Properties of minimum cuts | network1.pdf §4 p.1 | week4_exer_7_4_and_7_5.pdf (statement) | ✓ solved |
| 5.1 | — | Zombie breakout — Save the capital | network1.pdf §5.1 p.1 | weekplan text (p.1) | ✓ solved |
| 5.2 | — | Zombie breakout — Distributing the vaccine | network1.pdf §5.2 p.1 | weekplan text (p.1) | ✓ solved |
| 6 | CSES 1694 | Download Speed (max-flow) | network1.pdf §6 p.1 | CSES statement (external) | ✓ algorithm only |

No omissions; all weekplan items above have solution stubs completed.

Solutions

Exercise 1 — KT 7.1 (Fig. 7.24) and KT 7.2 (Fig. 7.25)

Assignment Source: network1.pdf §1 p.1. **Text Source:** week4_exer_7_1_and_7_2.pdf (figures in Pasted image (2).png).

- **KT 7.1 (all capacities 1):** enumerating valid cuts (S, T) with $s \in S, t \in T$ gives minimum capacity 2 . Representative minimum cuts: $S \in \{\{s\}, \{s, v\}, \{s, u, v\}\}$.

✓ **Answer:** min cut value 2 (any of the above).

- **KT 7.2 (weighted):** by direct cut enumeration the minimum capacity is 4 , attained by $S = \{s, v\}$.

✓ **Answer:** min cut value 4 with certificate $S = \{s, v\}$.

Pitfalls: forgetting that $u \rightarrow v$ contributes only when $u \in S, v \in T$.

Transfer Pattern (mapping guide):

- **Archetype name:** Min $s - t$ cut on a small directed graph.
- **Recognition cues:** “find the minimum cut”, few nodes/edges, capacities given on arcs, single s, t .
- **Mapping steps:** (1) Fix $s \in S, t \in T$. (2) Enumerate plausible S sets (on very small graphs) or argue via max-flow. (3) For each S , sum only edges crossing $S \rightarrow T$. (4) Choose the smallest sum.
- **Certificate form:** exhibit S and list its crossing edges with capacities and their sum.
- **Anti-cues / pitfalls:** counting edges entering S ; mixing undirected with directed; forgetting parallel edges' combined effect.

Exercise 2 — Ford-Fulkerson on the provided graph

Assignment Source: network1.pdf §2 p.1. **Text Source:** Pasted image.png.

Upper bounds: $\text{out}(s) = 14$, $\text{in}(t) = 10$. We exhibit a flow of value 9 and a matching min cut of capacity 9 .

Augmentation trace (one valid sequence):

| Step | Path | Bottleneck δ | Saturated edges after step | Residual note |
|------|---|---------------------|------------------------------|-----------------------------|
| 1 | $s \rightarrow L \rightarrow F \rightarrow A \rightarrow t$ | 3 | partial on $A \rightarrow t$ | back-edges appear |
| 2 | $s \rightarrow L \rightarrow F \rightarrow G \rightarrow A \rightarrow t$ | 3 | $F \rightarrow A$ now full | $G \rightarrow A$ carries 3 |
| 3 | $s \rightarrow L \rightarrow F \rightarrow G \rightarrow t$ | 1 | $G \rightarrow t$ full | — |

| Step | Path | Bottleneck δ | Saturated edges after step | Residual note |
|------|---|------------------------|-------------------------------|-------------------|
| 4 | $s \rightarrow C \rightarrow G \rightarrow t$ | 1 | — | feeds via C arm |
| 5 | $s \rightarrow C \rightarrow B \rightarrow M \rightarrow t$ | 1 | — | one unit via M |

Cut certificate: let $S = \{s, L, F\}$, $T = V \setminus S$. Crossing edges $\{s \rightarrow C, F \rightarrow A, F \rightarrow G\}$ sum to $2 + 3 + 4 = 9$.

Answer: max flow = 9 ; min cut $S = \{s, L, F\}$ with capacity 9 .

Checks: feasibility at each node; value into t equals 9 ; bound $|f| \leq c(S, T)$ tight.

Variant drill: raising $M \rightarrow t$ by 1 doesn't help unless its feeder $B \rightarrow M$ (and upstream $s \rightarrow C$) increase.

Transfer Pattern (mapping guide):

- **Archetype name:** Max-flow by augmenting paths (Ford-Fulkerson / Edmonds-Karp).
- **Recognition cues:** given a specific network with s, t and capacities; asked to "compute a maximum flow" or "show a min cut".
- **Mapping steps:** (1) Work in the residual graph; choose BFS (EK) for determinism. (2) Record an augmentation trace table: step, path, δ , saturations. (3) After no path remains, take $S =$ nodes reachable from s in G_f and sum crossing capacities.
- **Certificate form:** final flow value and a cut (S, T) with listed crossing edges summing to that value.
- **Anti-cues / pitfalls:** forgetting back-edges; violating conservation; writing full edge lists (not needed).

Exercise 3 — KT 7.4 (Properties of maximum flows)

Assignment Source: network1.pdf §3 p.1. **Text Source:** week4_exer_7_4_and_7_5.pdf.

Claim (KT 7.4 paraphrase): "In any network with positive integer capacities, every maximum $s \rightarrow t$ flow saturates all edges out of s ."

Verdict: False.

Counterexample (minimal): vertices $\{s, a, b, t\}$ with edges $s \rightarrow a$ (1), $s \rightarrow b$ (1), $a \rightarrow t$ (1), no arc from $b \rightarrow t$. The maximum flow has value 1 (via a) and **does not** saturate $s \rightarrow b$.

Why the claim fails: max flow only needs to saturate edges on all minimum cuts; other outgoing edges of s can remain slack if they do not lead to t in the residual sense.

Answer: False; counterexample above suffices.

Transfer Pattern (mapping guide):

- **Archetype name:** Counterexample builder for universal flow claims.

- **Recognition cues:** statements of the form “every maximum flow must ...” or “for all networks ...”.
 - **Mapping steps:** (1) Identify the asserted property. (2) Build a 3-4 node network that prevents the property while keeping a valid max flow. (3) Prove optimality of your flow (e.g., show a min cut).
 - **Certificate form:** explicit small graph and a max-flow/min-cut witness contradicting the claim.
 - **Anti-cues / pitfalls:** using zero-capacity edges when the claim assumes positive; failing to prove maximality.
-

Exercise 4 — KT 7.5 (Add 1 to every capacity)

Assignment Source: network1.pdf §4 p.1. **Text Source:** week4_exer_7_4_and_7_5.pdf.

Claim (KT 7.5 paraphrase): “If (A, B) is a minimum cut for capacities $\{c_e\}$, then after replacing every edge capacity by $1 + c_e$, the same cut (A, B) is still minimum.”

Verdict: False.

Counterexample (explicit): many small constructions work; here is one. Have five unit edges leaving s into a bottleneck node u that then connects to t with capacity 6. Initially $\{s\}$ is the min cut with value 5. After adding 1 to every capacity, that cut has value 10, while the cut enclosing u has value 7, so the minimum cut **changes**.

Reason: adding a constant to each edge changes every cut by “number of crossing edges,” which differs across cuts; order can flip.

✓ **Answer:** False; counterexample above.

Transfer Pattern (mapping guide):

- **Archetype name:** Cut-order instability under uniform capacity shifts.
 - **Recognition cues:** global operations applied to **all** edges (add k , multiply by α , etc.), question asks whether a specific min cut remains min.
 - **Mapping steps:** (1) Construct two near-optimal cuts with different numbers of crossing edges. (2) Apply the uniform change; the cut with fewer crossing edges becomes cheaper.
 - **Certificate form:** before/after cut capacities with sums exhibiting the flip.
 - **Anti-cues / pitfalls:** choosing cuts with equal crossing-edge counts (no flip possible).
-

Exercise 5.1 — Zombie breakout: Save the capital

Assignment Source: network1.pdf §5.1 p.1. **Text Source:** weekplan text.

Goal: minimum number of directed roads to remove so no infected city can reach the capital.

Reduction to min-cut: create a **super-source** s^* with arcs $s^* \rightarrow x$ to each infected city x (very large capacity). Let the **capital** be the sink t . Give every road capacity 1 and compute a minimum $s^* \rightarrow t$ cut. The cut edges are exactly the roads to destroy; their count equals the min-cut value.

Complexity: with $N = X + Y$ cities and R roads, Edmonds-Karp runs in $O(NR^2)$; Dinic faster in practice. Construction is linear in $N + R$.

Correctness sketch: any set of removed roads that separates all infected nodes from t is an $s^* \rightarrow t$ cut; the minimum such set is the minimum cut.

 **Answer:** remove exactly the edges in a minimum $s^* \rightarrow t$ cut of the unit-capacity road network.

Transfer Pattern (mapping guide):

- **Archetype name:** Minimum edge cut separating two sets (multi-source to sink).
 - **Recognition cues:** “fewest roads to destroy/block so group A cannot reach B”, unit costs or per-edge weights, directed map/roads.
 - **Mapping steps:** (1) Super-source into all sources in A; choose sink in B. (2) Unit capacity for cardinality; weights for costs. (3) Run min-cut; interpret crossing edges as removals.
 - **Certificate form:** cut (S, T) with listed roads and count (or cost).
 - **Anti-cues / pitfalls:** undirected roads need both directions or symmetric arcs; forgetting to include all infected sources.
-

Exercise 5.2 — Zombie breakout: Distributing the vaccine

Assignment Source: network1.pdf §5.2 p.1. **Text Source:** weekplan text.

Goal: send 10 doctors to each uninfected city in one day; infected cities allow at most 50 doctors to **pass through** per day (node capacity 50).

Reduction with node capacities: split each infected city v into $v_{in} \rightarrow v_{out}$ with arc capacity 50; redirect every incoming road to v_{in} and every outgoing road from v_{out} . Treat the capital as source with large out-capacity. Add arcs $u \rightarrow t^*$ of capacity 10 from each uninfected city u to a **super-sink** t^* . Compute a max flow to t^* and check if the value equals $10Y$.

 **Answer: Yes** iff the max flow from the capital to t^* equals $10Y$.

Complexity: graph grows by $O(X)$ due to splitting; run time as in 5.1.

Transfer Pattern (mapping guide):

- **Archetype name:** Node-capacity flow with per-destination quotas.
 - **Recognition cues:** transit-through limits on specific nodes; fixed demand q per destination; “can we route q to each city?”.
 - **Mapping steps:** (1) Node-split every capacity-limited node. (2) Add per-destination arcs to super-sink with capacity equal to required quota. (3) Solve a single max flow and compare value to total quota.
 - **Certificate form:** feasible flow of value total-quota; or a min-cut showing impossibility.
 - **Anti-cues / pitfalls:** confusing supply vs. transit constraints; forgetting to protect non-infected nodes with high capacity.
-

Exercise 6 — CSES 1694 “Download Speed” (max-flow)

Assignment Source: network1.pdf §6 p.1. **Text Source:** CSES problem statement.

I/O card: given n, m and directed edges with capacities, compute a maximum $s \rightarrow t$ flow and print its value.

Algorithm: Edmonds–Karp or Dinic. For contest constraints, **Dinic** is recommended.

Pseudocode (capacity-scaling Dinic, outline):

```
Algorithm: dinic_maxflow
Input: G=(V,E), c, s, t
Output: |f|
|f| ← 0; f ≡ 0
while bfs_layered_graph(G,c,f,s,t):
    work[ ] ← 0
    while ( $\delta \leftarrow \text{dfs\_blocking\_flow}(s,t,\infty)$ ) > 0:
        |f| ← |f| +  $\delta$ 
return |f|
// Typical:  $O(m n)$  general; faster on unit graphs and with scaling.
```

Checks: use 64-bit for sums; guard against parallel edges by summing or storing separately.

Transfer Pattern (mapping guide):

- **Archetype name:** Plain max-flow evaluation.
- **Recognition cues:** input is a graph with capacities and distinguished s, t ; output is a single number $|f|$.
- **Mapping steps:** (1) Build adjacency with residual edges. (2) Run Dinic/Edmonds–Karp. (3) Output $|f|$.
- **Certificate form:** not required by judge; locally verify with min-cut if needed.
- **Anti-cues / pitfalls:** 32-bit overflow on totals; mishandling parallel edges; forgetting to zero-index per statement.

Puzzle — Four Coins (from weekplan p.2)

Problem: after each of your flips (any subset of coins), the hangman may rotate the table by $0/90/180/270^\circ$. Win if all four are heads within 20 moves.

Strategy (sketch): use **orbit classes** under rotation and cycle a fixed sequence of subsets that breaks all rotational symmetries. One winning 10-move sequence is to flip, in order: $\{1\}, \{2\}, \{3\}, \{4\}, \{1, 2\}, \{1, 3\}, \{2, 3\}, \{1, 4\}, \{2, 4\}, \{3, 4\}$ where indices refer to positions relative to the table. After each move, the rotation cannot keep you in the same orbit indefinitely; the Hamming weight strictly increases at some step, and within 10 steps you reach all-heads. (Many equivalent sequences exist.)

Summary

- **Algorithms:** FF, Edmonds-Karp, Dinic; residual graphs; cut certificates.
- **Week results:** KT 7.1 \Rightarrow 2 ; KT 7.2 \Rightarrow 4 ; Ford-Fulkerson instance \Rightarrow max flow 9 with min cut $\{s, L, F\}$ (capacity 9) ; KT 7.4 \Rightarrow False (not all edges out of s need saturate); KT 7.5 \Rightarrow False (adding 1 can change the min cut); Zombies 5.1 \Rightarrow min edge cut from infected to capital; Zombies 5.2 \Rightarrow node-capacity flow with demand $10Y$; CSES 1694 \Rightarrow implement Dinic.
- **Transfer patterns:** min-cut enumeration; augmenting-path computation; node-splitting; super-sink quotas; counterexample builder for universal claims.
- **Notation:** $G = (V, E)$, c , f , G_f , $|f|$, $c(S, T)$, δ (delta).

Network Flows

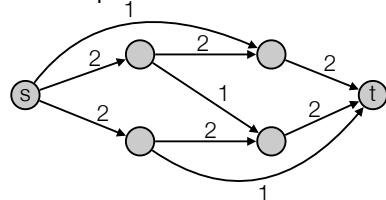
Inge Li Gørtz

Applications

- Matchings
- Job scheduling
- Image segmentation
- Baseball elimination
- Disjoint paths
- Survivable network design

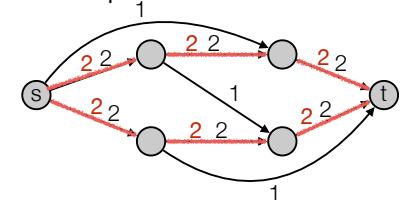
Network Flow

- Truck company: Wants to send as many trucks as possible from s to t . Limit of number of trucks on each road.



Network Flow

- Truck company: Wants to send as many trucks as possible from s to t . Limit of number of trucks on each road.
- Example 1:
 - Solution 1: 4 trucks

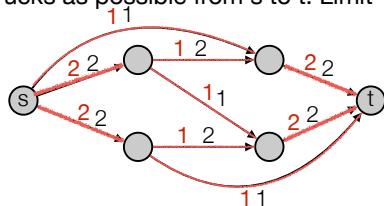


Network Flow

- Truck company: Wants to send as many trucks as possible from s to t . Limit of number of trucks on each road.

- Example 1:

- Solution 1: 4 trucks
- Solution 2: 5 trucks



Network Flow

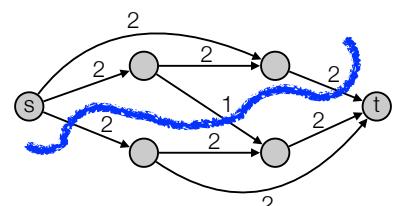
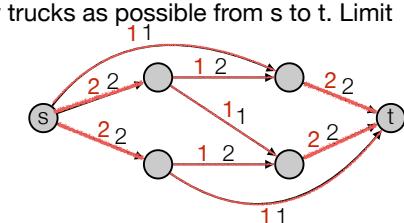
- Truck company: Wants to send as many trucks as possible from s to t . Limit of number of trucks on each road.

- Example 1:

- Solution 1: 4 trucks
- Solution 2: 5 trucks

- Example 2:

- 5 trucks (need to cross river).



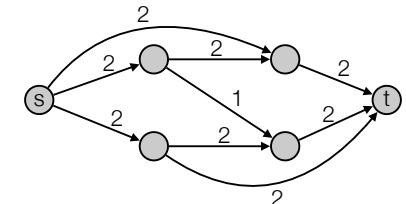
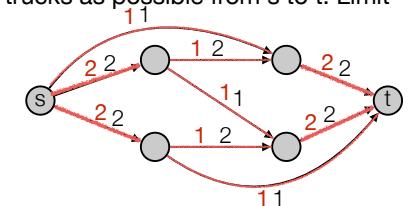
Network Flow

- Truck company: Wants to send as many trucks as possible from s to t . Limit of number of trucks on each road.

- Example 1:

- Solution 1: 4 trucks
- Solution 2: 5 trucks

- Example 2:



Network Flow

- Network flow:

- graph $G=(V,E)$.

- Special vertices s (source) and t (sink).

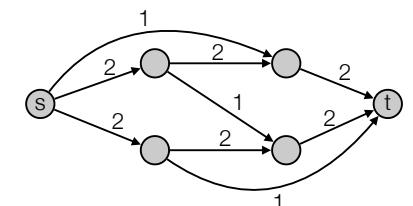
- s has no edges in and t has no edges out.

- Every edge (e) has a (integer) capacity $c(e) \geq 0$.

- Flow:

- capacity constraint:** every edge e has a flow $0 \leq f(e) \leq c(e)$.

- flow conservation:** for all $u \neq s, t$: flow into u equals flow out of u .

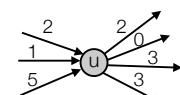


$$\sum_{v:(v,u) \in E} f(v, u) = \sum_{v:(u,v) \in E} f(u, v)$$

- Value of flow f is the sum of flows out of s :

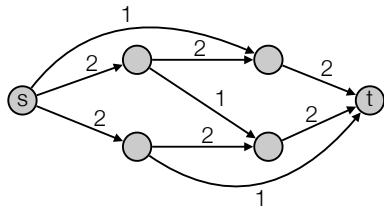
$$v(f) = \sum_{v:(s,v) \in E} f(e) = f^{out}(s)$$

- Maximum flow problem:** find $s-t$ flow of maximum value



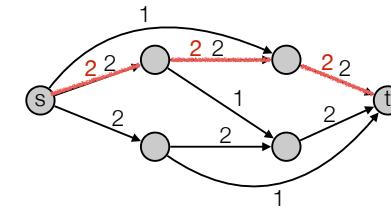
Algorithm

- Find path where we can send more flow.



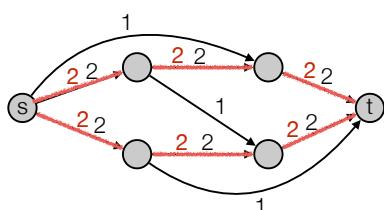
Algorithm

- Find path where we can send more flow.



Algorithm

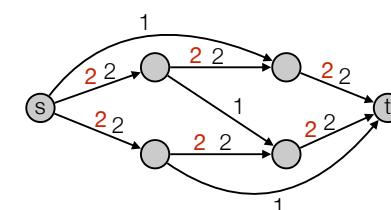
- Find path where we can send more flow.



Algorithm

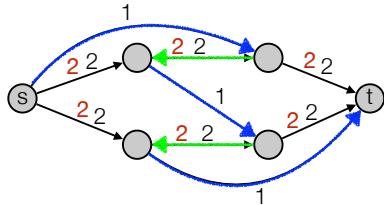
- Find path where we can send more flow.

- Send flow back (cancel flow).



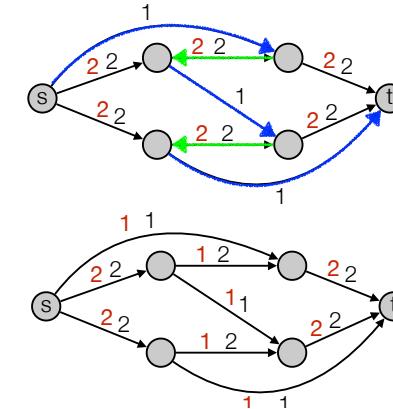
Algorithm

- Find path where we can send more flow.
- Send flow back (cancel flow).



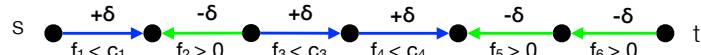
Algorithm

- Find path where we can send more flow.
- Send flow back (cancel flow).

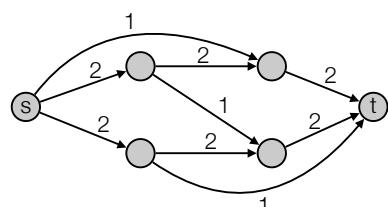


Augmenting Paths

- Augmenting path: s-t path P where
 - forward edges have leftover capacity
 - backwards edges have positive flow

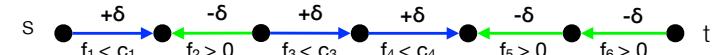


- Can add extra flow: $\min(c_1 - f_1, f_2, c_3 - f_3, c_4 - f_4, f_5, f_6) = \delta = \text{bottleneck}(P)$.

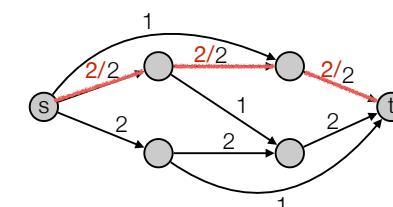


Augmenting Paths

- Augmenting path: s-t path P where
 - forward edges have leftover capacity
 - backwards edges have positive flow

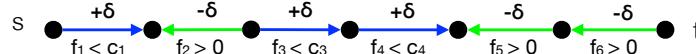


- Can add extra flow: $\min(c_1 - f_1, f_2, c_3 - f_3, c_4 - f_4, f_5, f_6) = \delta = \text{bottleneck}(P)$.

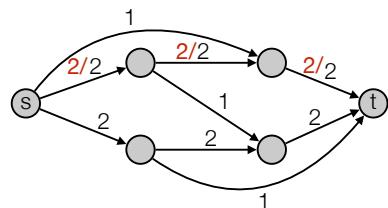


Augmenting Paths

- Augmenting path: s-t path P where
 - forward edges have leftover capacity
 - backwards edges have positive flow

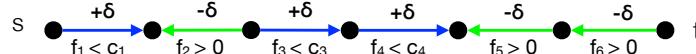


- Can add extra flow: $\min(c_1 - f_1, f_2, c_3 - f_3, c_4 - f_4, f_5, f_6) = \delta = \text{bottleneck}(P)$.

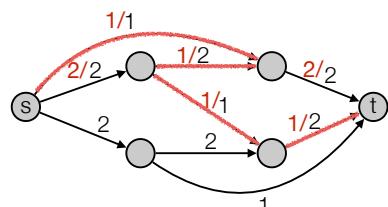


Augmenting Paths

- Augmenting path: s-t path P where
 - forward edges have leftover capacity
 - backwards edges have positive flow

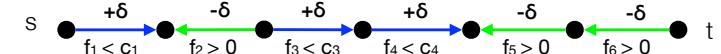


- Can add extra flow: $\min(c_1 - f_1, f_2, c_3 - f_3, c_4 - f_4, f_5, f_6) = \delta = \text{bottleneck}(P)$.

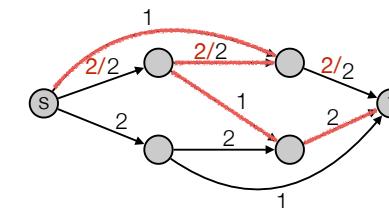


Augmenting Paths

- Augmenting path: s-t path P where
 - forward edges have leftover capacity
 - backwards edges have positive flow

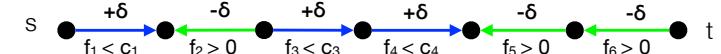


- Can add extra flow: $\min(c_1 - f_1, f_2, c_3 - f_3, c_4 - f_4, f_5, f_6) = \delta = \text{bottleneck}(P)$.

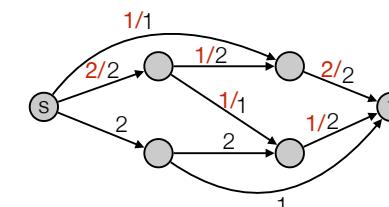


Augmenting Paths

- Augmenting path: s-t path P where
 - forward edges have leftover capacity
 - backwards edges have positive flow



- Can add extra flow: $\min(c_1 - f_1, f_2, c_3 - f_3, c_4 - f_4, f_5, f_6) = \delta = \text{bottleneck}(P)$.

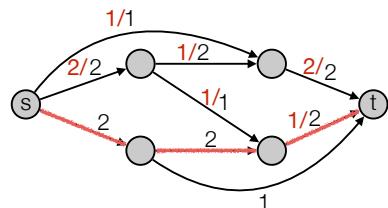


Augmenting Paths

- Augmenting path: s-t path P where
 - forward edges have leftover capacity
 - backwards edges have positive flow

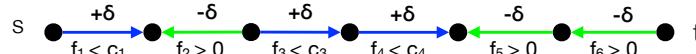


• Can add extra flow: $\min(c_1 - f_1, f_2, c_3 - f_3, c_4 - f_4, f_5, f_6) = \delta = \text{bottleneck}(P)$.

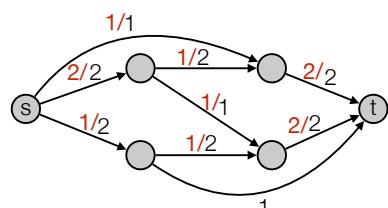


Augmenting Paths

- Augmenting path: s-t path P where
 - forward edges have leftover capacity
 - backwards edges have positive flow

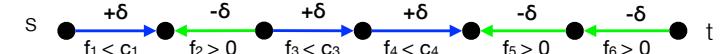


• Can add extra flow: $\min(c_1 - f_1, f_2, c_3 - f_3, c_4 - f_4, f_5, f_6) = \delta = \text{bottleneck}(P)$.

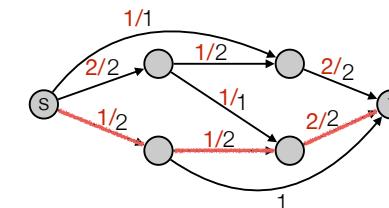


Augmenting Paths

- Augmenting path: s-t path P where
 - forward edges have leftover capacity
 - backwards edges have positive flow



• Can add extra flow: $\min(c_1 - f_1, f_2, c_3 - f_3, c_4 - f_4, f_5, f_6) = \delta = \text{bottleneck}(P)$.

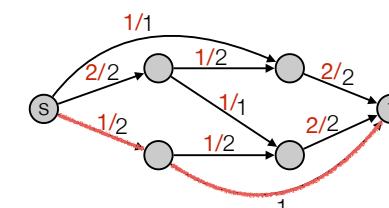


Augmenting Paths

- Augmenting path: s-t path P where
 - forward edges have leftover capacity
 - backwards edges have positive flow



• Can add extra flow: $\min(c_1 - f_1, f_2, c_3 - f_3, c_4 - f_4, f_5, f_6) = \delta = \text{bottleneck}(P)$.

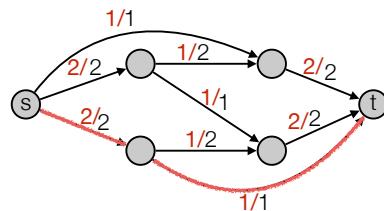


Augmenting Paths

- Augmenting path: s-t path P where
 - forward edges have leftover capacity
 - backwards edges have positive flow

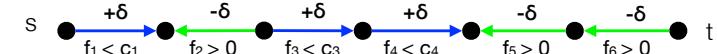


- Can add extra flow: $\min(c_1 - f_1, f_2, c_3 - f_3, c_4 - f_4, f_5, f_6) = \delta = \text{bottleneck}(P)$.

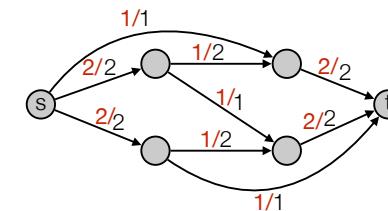


Augmenting Paths

- Augmenting path: s-t path P where
 - forward edges have leftover capacity
 - backwards edges have positive flow



- Can add extra flow: $\min(c_1 - f_1, f_2, c_3 - f_3, c_4 - f_4, f_5, f_6) = \delta = \text{bottleneck}(P)$.



Augmenting Paths

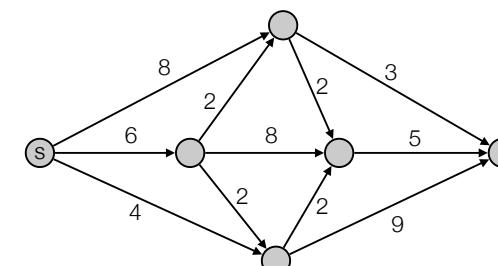
- Augmenting path (definition different than in CLRS): s-t path where
 - forward edges have leftover capacity
 - backwards edges have positive flow



- Can add extra flow: $\min(c_1 - f_1, f_2, c_3 - f_3, c_4 - f_4, f_5, f_6) = \delta = \text{bottleneck}(P)$.

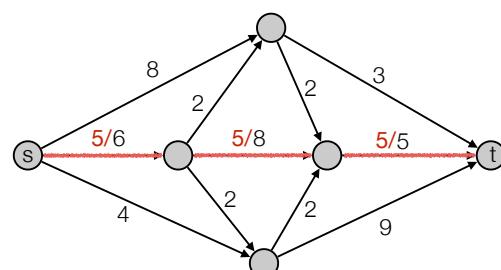
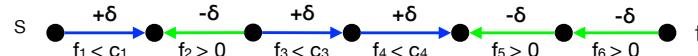
Ford Fulkerson

- Augmenting path: s-t path P where
 - forward edges have leftover capacity
 - backwards edges have positive flow



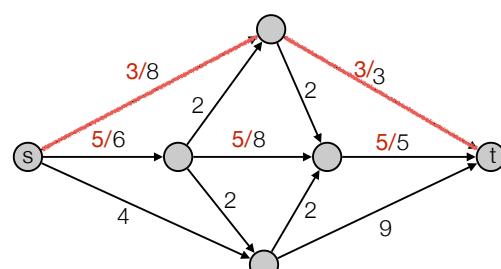
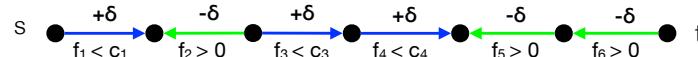
Ford Fulkerson

- Augmenting path: s-t path P where
 - forward edges have leftover capacity
 - backwards edges have positive flow



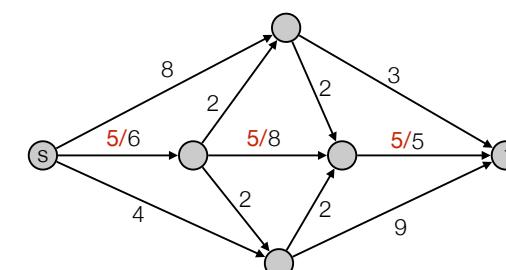
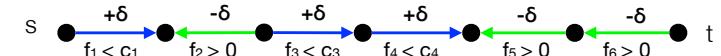
Ford Fulkerson

- Augmenting path: s-t path P where
 - forward edges have leftover capacity
 - backwards edges have positive flow



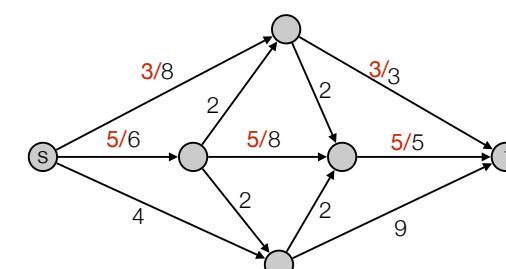
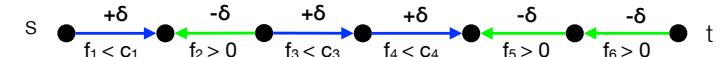
Ford Fulkerson

- Augmenting path: s-t path P where
 - forward edges have leftover capacity
 - backwards edges have positive flow



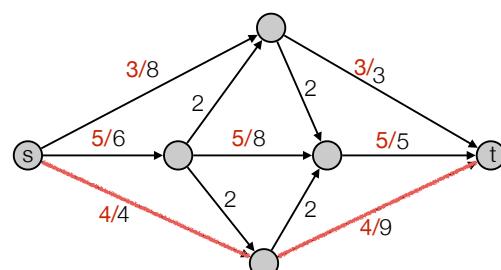
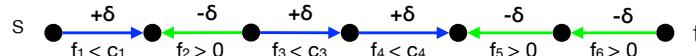
Ford Fulkerson

- Augmenting path: s-t path P where
 - forward edges have leftover capacity
 - backwards edges have positive flow



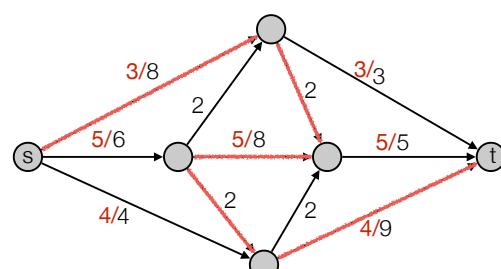
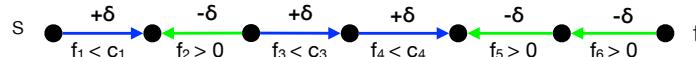
Ford Fulkerson

- Augmenting path: s-t path P where
 - forward edges have leftover capacity
 - backwards edges have positive flow



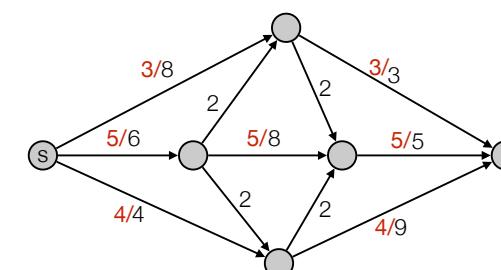
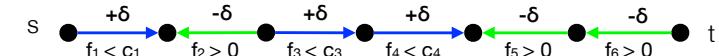
Ford Fulkerson

- Augmenting path: s-t path P where
 - forward edges have leftover capacity
 - backwards edges have positive flow



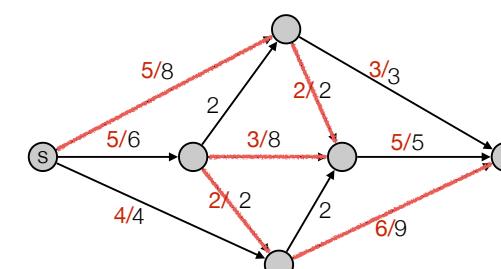
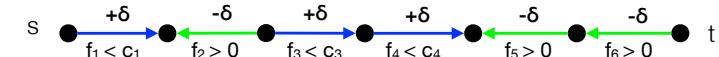
Ford Fulkerson

- Augmenting path: s-t path P where
 - forward edges have leftover capacity
 - backwards edges have positive flow



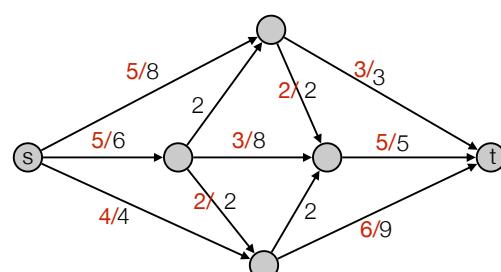
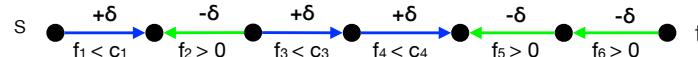
Ford Fulkerson

- Augmenting path: s-t path P where
 - forward edges have leftover capacity
 - backwards edges have positive flow

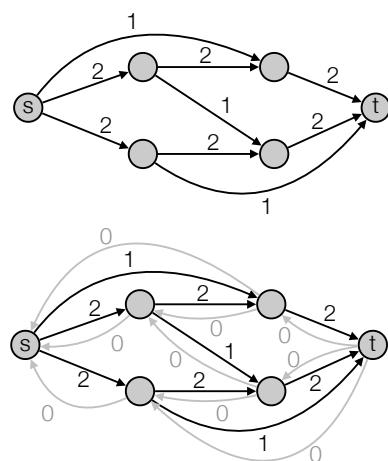


Ford Fulkerson

- Augmenting path: s-t path P where
 - forward edges have leftover capacity
 - backwards edges have positive flow



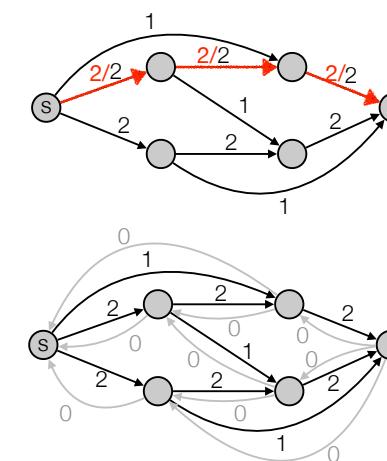
Residual networks



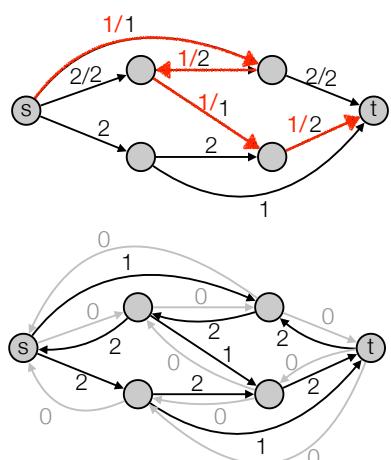
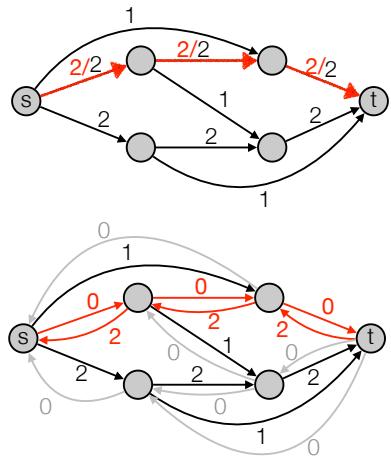
Analysis of Ford-Fulkerson

- Integral capacities implies there's a maximum flow where all flow values $f(e)$ are integers.
- Number of iterations:
 - Always increment flow by at least 1: #iterations \leq max flow value f^*
- Time for one iteration:
 - Can find augmenting path in linear time: One iteration takes $O(m)$ time.
- Total running time = $O(|f^*| m)$.

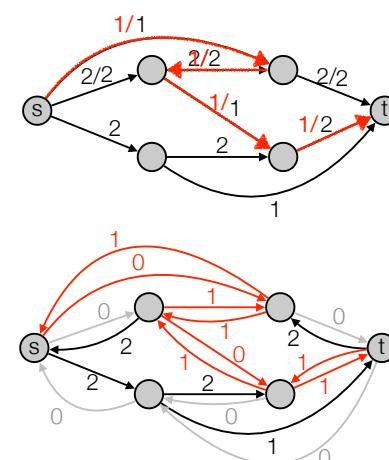
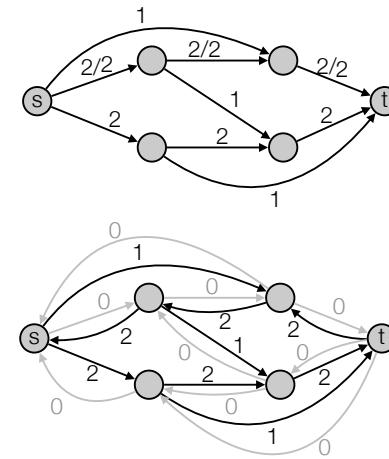
Residual networks



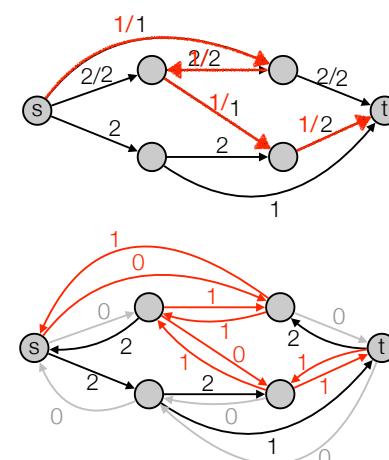
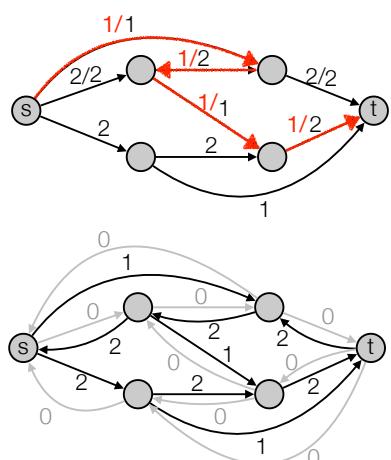
Residual networks



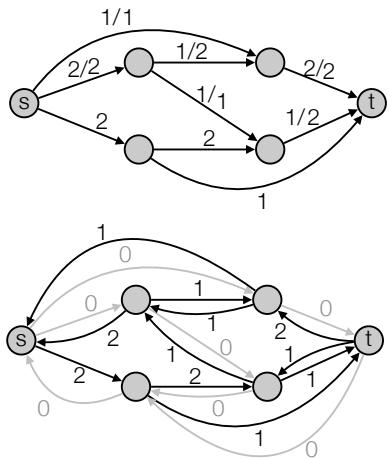
Residual networks



Residual networks



Residual networks



Implementation

```

adj[0..n-1]          # adjacency list
cap                # capacity dictionary

for each edge (u,v,c):
    adj[u].append(v)
    adj[v].append(u)
    cap[(u,v)] = c
    cap[(v,u)] = 0

# Graph search algorithm that searches for an augmenting path from u->v      (e.g. BFS or DFS)
AugPath():
    visited[0..n-1]        # visited list initialized to False
    pred[0..n-1]           # predecessor list
    stack S                # initialize stack S

    push(S,s) and set visited[s] = True
    while S not empty and not visited[t]:
        u = pop(S)
        for v in adj[u]:
            if visited[v] or cap[(u,v)] = 0:
                continue
            visited[v] = True
            pred[v] = u
            push(S,v)

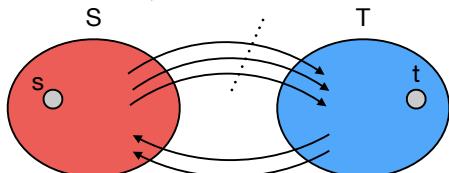
    if visited[t]:           # found augmenting path
        follow pred pointers back from t to s to find delta
        follow pred pointers back from t to s to update capacities
        return delta
    return 0                 # no augmenting path found

```

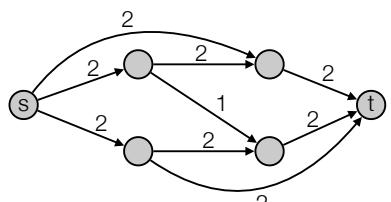
(fill out details yourself)
(fill out details yourself)

s-t Cuts

- **Cut:** Partition of vertices into S and T , such that $s \in S$ and $t \in T$.

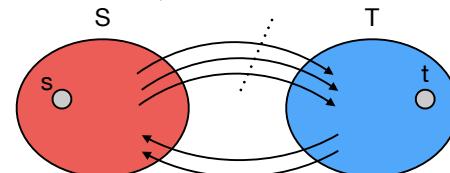


- Capacity of cut: total capacity of edges going from S to T .

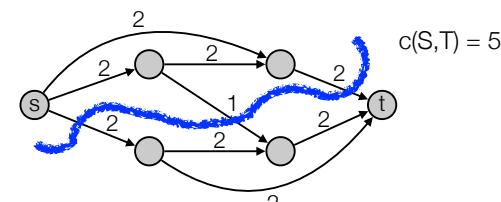


s-t Cuts

- **Cut:** Partition of vertices into S and T , such that $s \in S$ and $t \in T$.

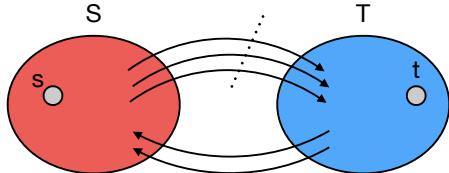


- Capacity of cut: total capacity of edges going from S to T .

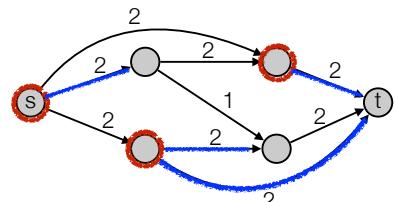


s-t Cuts

- **Cut:** Partition of vertices into S and T, such that $s \in S$ and $t \in T$.

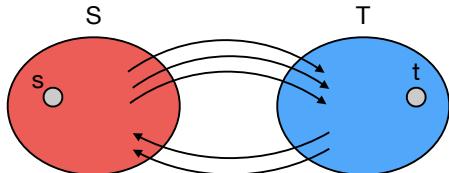


- Capacity of cut: total capacity of edges going from S to T.

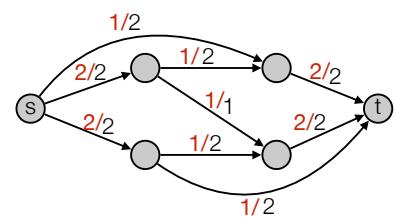


s-t Cuts

- **Cut:** Partition of vertices into S and T, such that $s \in S$ and $t \in T$.

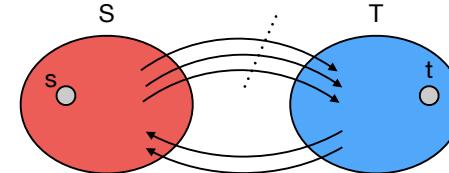


- Flow across cut: = flow from S to T minus flow from T to S.

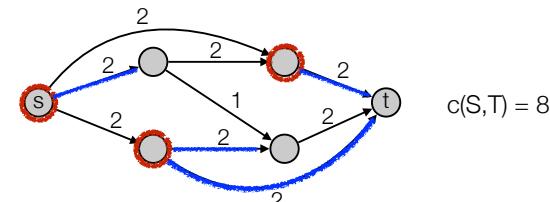


s-t Cuts

- **Cut:** Partition of vertices into S and T, such that $s \in S$ and $t \in T$.

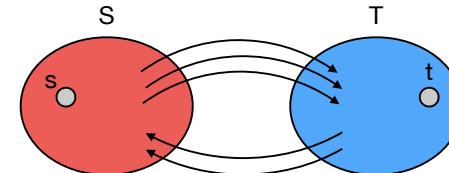


- Capacity of cut: total capacity of edges going from S to T.

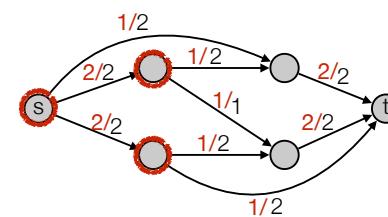


s-t Cuts

- **Cut:** Partition of vertices into S and T, such that $s \in S$ and $t \in T$.

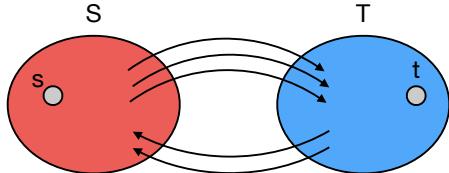


- Flow across cut: = flow from S to T minus flow from T to S.

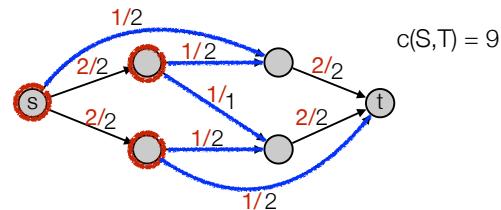


s-t Cuts

- **Cut:** Partition of vertices into S and T, such that $s \in S$ and $t \in T$.

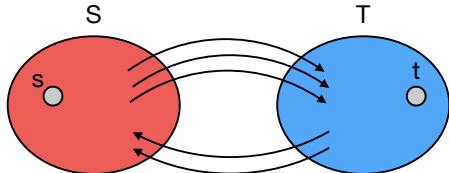


• Flow across cut: = flow from S to T minus flow from T to S.

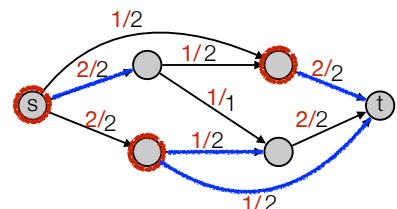


s-t Cuts

- **Cut:** Partition of vertices into S and T, such that $s \in S$ and $t \in T$.

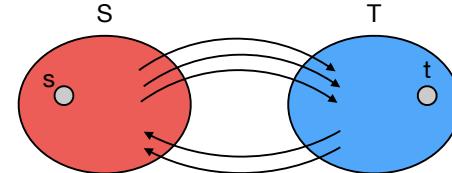


• Flow across cut: = flow from S to T minus flow from T to S.

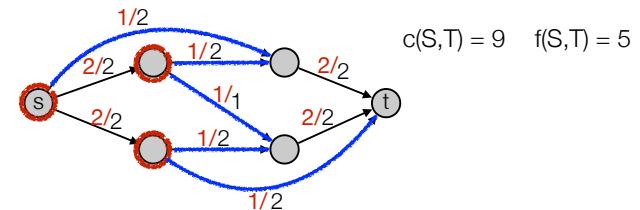


s-t Cuts

- **Cut:** Partition of vertices into S and T, such that $s \in S$ and $t \in T$.

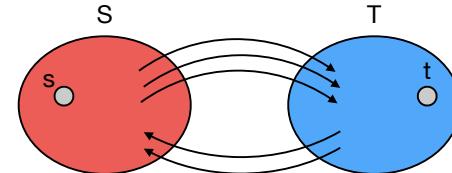


• Flow across cut: = flow from S to T minus flow from T to S.

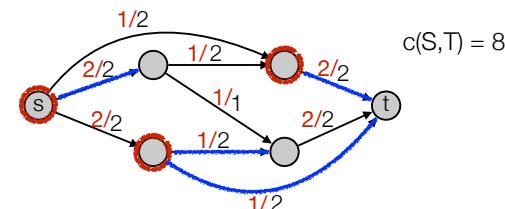


s-t Cuts

- **Cut:** Partition of vertices into S and T, such that $s \in S$ and $t \in T$.

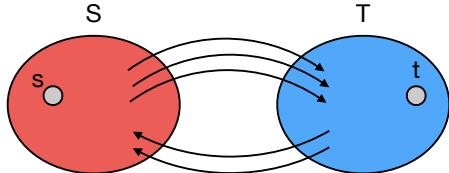


• Flow across cut: = flow from S to T minus flow from T to S.

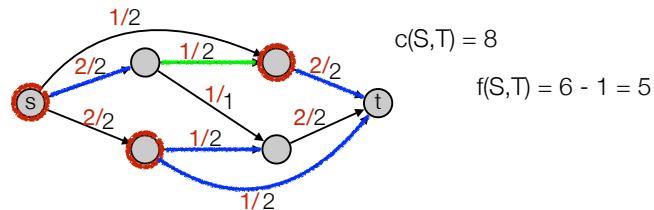


s-t Cuts

- **Cut:** Partition of vertices into S and T, such that $s \in S$ and $t \in T$.

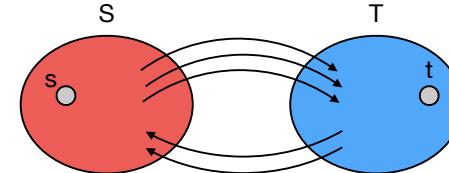


• Flow across cut: = flow from S to T minus flow from T to S.

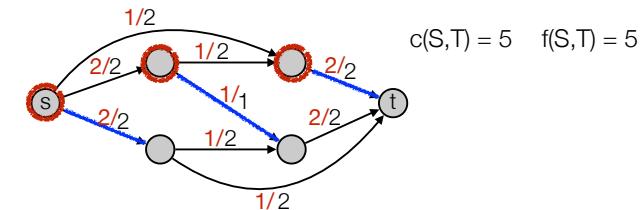


s-t Cuts

- **Cut:** Partition of vertices into S and T, such that $s \in S$ and $t \in T$.

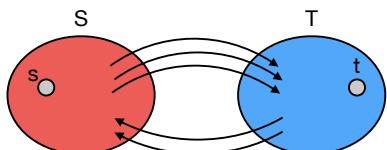


• Flow across cut: = flow from S to T minus flow from T to S.

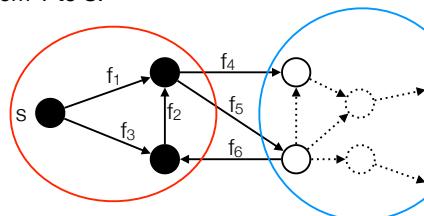


s-t Cuts

- **Cut:** Partition of vertices into S and T, such that $s \in S$ and $t \in T$.

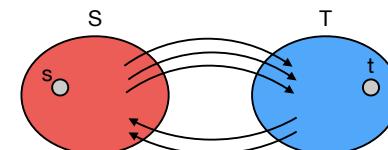


• Flow across cut = flow from S to T minus flow from T to S.



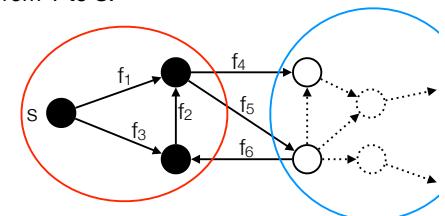
s-t Cuts

- **Cut:** Partition of vertices into S and T, such that $s \in S$ and $t \in T$.



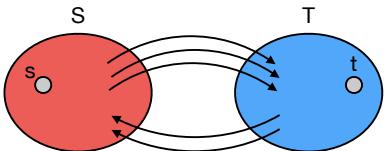
• Flow across cut = flow from S to T minus flow from T to S.

• Flow across cut: $f_4 + f_5 - f_6 = ?$



s-t Cuts

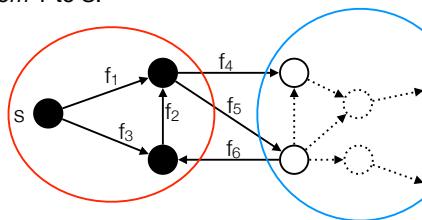
- **Cut:** Partition of vertices into S and T, such that $s \in S$ and $t \in T$.



• Flow across cut = flow from S to T minus flow from T to S.

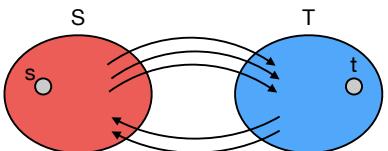
• Flow across cut: $f_4 + f_5 - f_6 = ?$

- $f_4 + f_5 - f_1 - f_2 = 0$
- $f_2 - f_6 - f_3 = 0$
- $f_1 + f_3 = |f|$
- $(f_4 + f_5 - f_1 - f_2) + (f_2 - f_6 - f_3) + (f_1 + f_3) = |f|$



s-t Cuts

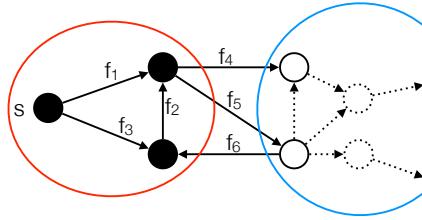
- **Cut:** Partition of vertices into S and T, such that $s \in S$ and $t \in T$.



• Flow across cut = flow from S to T minus flow from T to S.

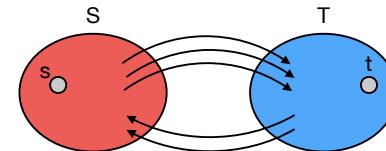
• Flow across cut: $f_4 + f_5 - f_6 = ?$

- $f_4 + f_5 - f_1 - f_2 = 0$
- $f_2 - f_6 - f_3 = 0$
- $f_1 + f_3 = |f|$
- $(f_4 + f_5 - f_1 - f_2) + (f_2 - f_6 - f_3) + (f_1 + f_3) = |f|$
- $f_4 + f_5 - f_6 = |f|$



s-t Cuts

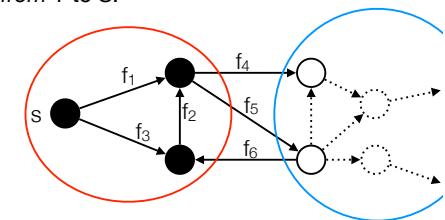
- **Cut:** Partition of vertices into S and T, such that $s \in S$ and $t \in T$.



• Flow across cut = flow from S to T minus flow from T to S.

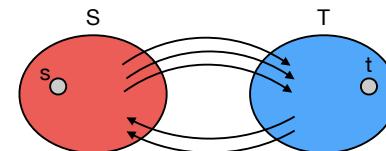
• Flow across cut: $f_4 + f_5 - f_6 = ?$

- $f_4 + f_5 - f_1 - f_2 = 0$
- $f_2 - f_6 - f_3 = 0$
- $f_1 + f_3 = |f|$
- $(f_4 + f_5 - f_1 - f_2) + (f_2 - f_6 - f_3) + (f_1 + f_3) = |f|$



s-t Cuts

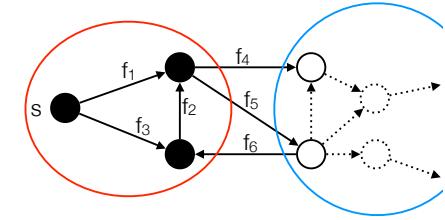
- **Cut:** Partition of vertices into S and T, such that $s \in S$ and $t \in T$.



• Flow across cut = flow from S to T minus flow from T to S.

• Flow across cut: $f_4 + f_5 - f_6 = ?$

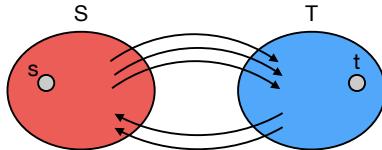
- $f_4 + f_5 - f_1 - f_2 = 0$
- $f_2 - f_6 - f_3 = 0$
- $f_1 + f_3 = |f|$
- $(f_4 + f_5 - f_1 - f_2) + (f_2 - f_6 - f_3) + (f_1 + f_3) = |f|$
- $f_4 + f_5 - f_6 = |f|$



• Flow across cut is $|f|$ for all cuts => flow out of s = flow into t.

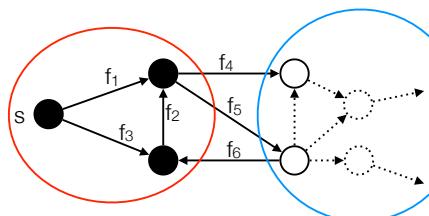
s-t Cuts

- **Cut:** Partition of vertices into S and T , such that $s \in S$ and $t \in T$.



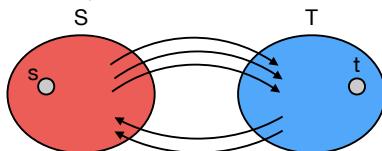
- Flow across cut is $|f|$ for all cuts \Rightarrow flow out of s = flow into t .
- $|f| \leq c(S, T)$:

$$\bullet |f| = f_4 + f_5 - f_6 \leq f_4 + f_5 \leq c_4 + c_5 = c(S, T)$$



s-t Cuts

- **Cut:** Partition of vertices into S and T , such that $s \in S$ and $t \in T$.



- Suppose we have found flow f and cut (S, T) such that $|f| = c(S, T)$. Then f is a maximum flow and (S, T) is a minimum cut.

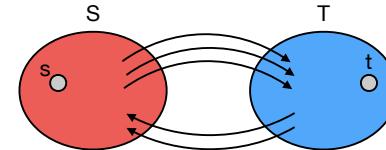
• Let f^* be the maximum flow and the (S^*, T^*) minimum cut:

$$\bullet |f| \leq |f^*| \leq c(S^*, T^*) \leq c(S, T).$$

• Since $|f| = c(S, T)$ this implies $|f| = |f^*|$ and $c(S, T) = c(S^*, T^*)$.

s-t Cuts

- **Cut:** Partition of vertices into S and T , such that $s \in S$ and $t \in T$.



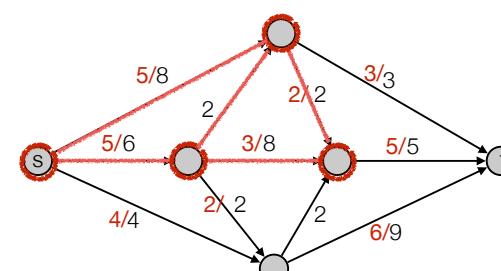
- Suppose we have found flow f and cut (S, T) such that $|f| = c(S, T)$. Then f is a maximum flow and (S, T) is a minimum cut.

Finding minimum cuts

- Use Ford-Fulkerson to find a max-flow (finding augmenting paths).

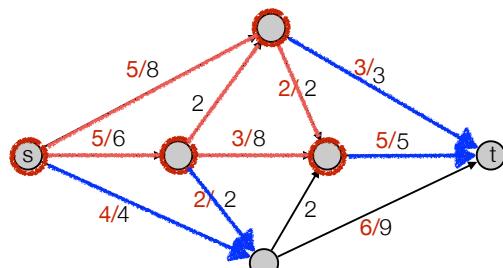
- When no augmenting s-t path:

• Let S be all vertices to which there exists an augmenting path from s .



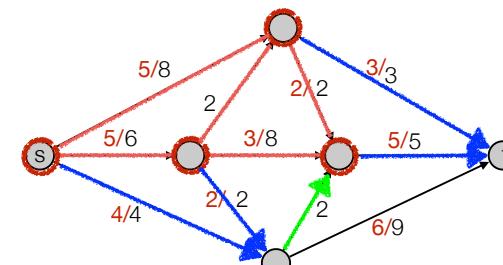
Finding minimum cuts

- Use Ford-Fulkerson to find a max-flow (finding augmenting paths).
- When no augmenting s-t path:
 - Let S be all vertices to which there exists an augmenting path from s.
 - value of flow (S,T) = capacity of the cut:
 - All **forward** edges in the minimum cut are “full” (flow = capacity).



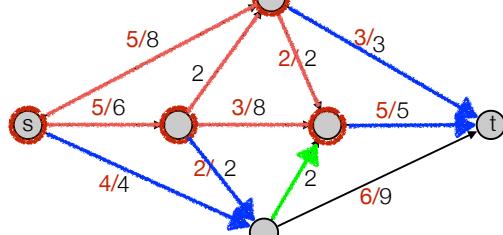
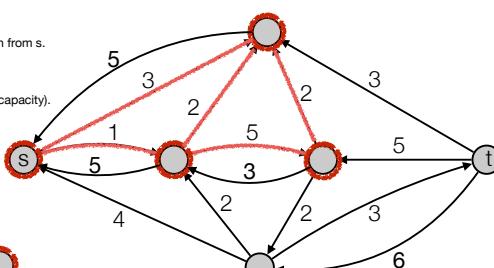
Finding minimum cuts

- Use Ford-Fulkerson to find a max-flow (finding augmenting paths).
- When no augmenting s-t path:
 - Let S be all vertices to which there exists an augmenting path from s.
 - value of flow (S,T) = capacity of the cut:
 - All **forward** edges in the minimum cut are “full” (flow = capacity).
 - All **backwards** edges in minimum cut have 0 flow.



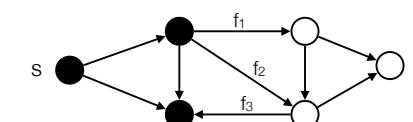
Finding minimum cuts (with residual network).

- Use Ford-Fulkerson to find a max-flow (finding augmenting paths).
- When no augmenting s-t path:
 - Let S be all vertices to which there exists an augmenting path from s.
 - value of flow (S,T) = capacity of the cut:
 - All **forward** edges in the minimum cut are “full” (flow = capacity).
 - All **backwards** edges in minimum cut have 0 flow.



Use of Max-flow min-cut theorem

- There is no augmenting path $\Leftrightarrow f$ is a maximum flow.
- f maximum flow \Rightarrow no augmenting path:
 - Show that exists augmenting path $\Rightarrow f$ not maximum flow.
- no augmenting path $\Rightarrow f$ maximum flow
 - no augmenting path \Rightarrow exists cut (S,T) where $|f| = c(S,T)$:
 - Let S be all vertices to which there exists an augmenting path from s.
 - t not in S (since there is no augmenting s-t path).
 - Edges from S to T: $f_1 = c_1$ and $f_2 = c_2$.
 - Edges from T to S: $f_3 = 0$.
 - $\Rightarrow |f| = f_1 + f_2 - f_3 = f_1 + f_2 = c_1 + c_2 = c(S,T)$.
 - $\Rightarrow f$ a maximum flow and (S,T) a minimum cut.



Removing assumptions

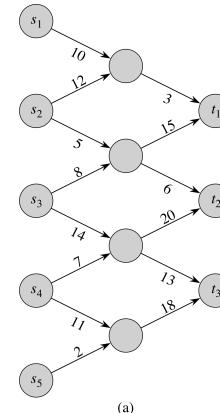
- Edges into s and out of t :

$$v(f) = f^{out}(s) - f^{in}(t)$$

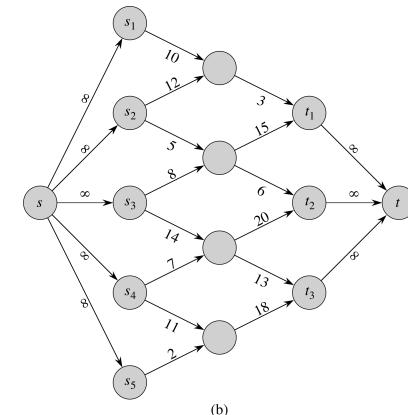
- Capacities not integers.

Network Flow

- Multiple sources and sinks:



(a)



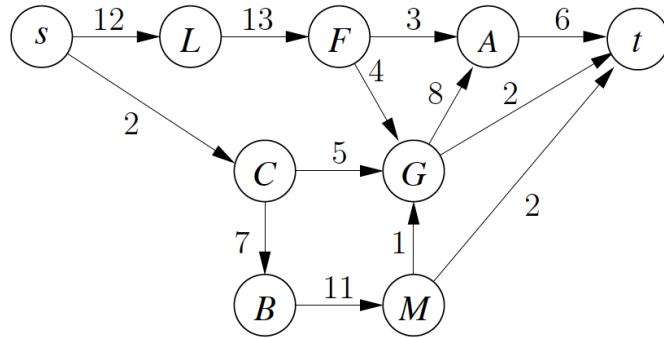
(b)

Reading material

At the lecture we will talk about network flows. For reading material see the webpage.

Exercises

- 1 [w] **Max flow/min cut** Solve exercises KT 7.1 and 7.2.
- 2 [w] **Ford-Fulkersons algorithm** Compute a maximum flow and minimum cut on graph below.



- 3 **Properties of maximum flows** Solve exercise KT 7.4.

- 4 **Properties of minimum cuts** Solve exercise KT 7.5.

5 Zombie breakout (from the Exam 2014) A zombie breakout has occurred in the country 1D and you have been asked by the prime minister to help find solutions to stop the infection. You are given a map of the country, with coordinates of all cities. There are X infected cities and Y uninfected cities. The map also contains information about all roads between cities. A road goes directly to from one city to another and all roads are directed. There are R roads. A city is reachable from the capital if you can follow one or more roads from the capital to the city. You can assume that the travel time from the capital to any reachable city is no more than a day.

5.1 Save the capital The zombies only travel on the roads. The capital is still uninfected and the prime minister wants to know how many roads that must be destroyed to keep the capital uninfected. Give an algorithm to compute the minimum number of roads that has to be destroyed to cut off the capital from the infected cities. Analyze the running time of your algorithm in terms of X , Y , and R . Remember to argue that your algorithm is correct.

5.2 Distributing the vaccine Researchers in the center in the capital have found a vaccine. To get the vaccine to the people in the uninfected cities we need to send 10 doctors to each uninfected city. The doctors can only travel on the roads. They can go through infected cities but at most 50 doctors can go through each infected city per day. Give an algorithm to decide whether we can get the vaccine out to all uninfected cities in one day. Analyze the running time of your algorithm in terms of X , Y , and R . Remember to argue that your algorithm is correct.

- 6 **CSES** Solve the exercise Download Speed (under Graph Algorithms): <https://cses.fi/problemset/task/1694> First give an algorithm and analyse its correctness and running time. Then implement it afterwards.

Puzzle of the week: Four Coins You have to win a game against the hangman. Before the game starts you are blindfolded. There are four coins placed on a square table, one coin at each corner. The initial configuration of the coins is chosen by the hangman, arbitrarily and unknown to you. Your goal is to have all four coins heads up. In each move you can select any subsets of the four coins, which are then flipped simultaneously by the hangman. After your move, if all four coins are heads up, you win. If not, the hangman may rotate the table by an amount of his choice (90, 180, 270, or 360 degrees). If you don't manage to have all four coins heads up in 20 moves or less, you lose and the hangman executes his job. What's your strategy?