

ADS2 — Week “Network 2” Notes & Full Solutions

Field	Value
Title	ADS2 Network Flow — Week “Network 2”
Date	2025-10-18
Author	ADS2 Notes Copilot
Sources used	weekplan.pdf (pp.1–2); flow2-4x1.pdf (Edmonds–Karp slides); text_book_sections.pdf (Ch.7 excerpts: §§7.3, 7.4, 7.6–7.9); ex_7_14.pdf (KT 7.14); figures: Pasted image.png, Pasted image (2).png, Pasted image (3).png, Pasted image (4).png
Week plan filename	weekplan.pdf

General Methodology and Theory

- **Max flow / min cut** basics, augmenting paths, and **Edmonds–Karp (EK)** (BFS in residual graph → shortest-edge paths; $O(V \cdot E^2)$). **Capacity scaling**: only use residual edges with capacity $\geq \Delta$, halving Δ from the largest power of two \leq max capacity (total $O(E^2 \log C)$).
- **Circulations with demands / lower bounds**: reduce to a single $s^* \rightarrow t^*$ max-flow instance; also the right tool for multi-row/column capacity and “exactly- k ” constraints.
- **Bipartite matching**: unit capacities give $O(E\sqrt{V})$ (Hopcroft–Karp); with row/column bounds, use a small flow gadget (super-source/sink) or clone-rows trick.
- **Node capacities**: split each v into $v_{in} \rightarrow v_{out}$ with capacity $c(v)$.
- **Certificates**: for each computation give (S, T) cut with crossing capacities that sum to $|f|$.

Notes (quick references)

- EK path = any $s \rightarrow t$ path in the residual graph found by BFS; bottleneck $\delta = \min$ residual on its edges. After each augmentation update residual forward/backward edges.
- Scaling rule: start $\Delta = 2^{\lfloor \log_2 C_{\max} \rfloor}$; while $\Delta \geq 1$, keep augmenting with only edges of residual $\geq \Delta$; then $\Delta \leftarrow \Delta/2$.
- Matching/b-matching model: source \rightarrow Left side (row-nodes) with capacities (row quotas); edges for allowed placements with cap 1; right side (column-nodes) \rightarrow sink with column quotas.
- Mixed-graph Euler tour: pre-directed edges fix a **balance** $r(v) = \text{in_dir}(v) - \text{out_dir}(v)$. If $\text{udeg}(v)$ is the number of undirected edges at v , we must choose exactly $h(v) = (\text{udeg}(v) - r(v))/2$ of those to point **into** v . Existence reduces to a bipartite **b-matching** between **undirected edges** and **vertices**.

Coverage Table

Weekplan ID	Canonical ID	Title/Label (verbatim)	Assignment Source	Text Source	Status
1	—	The Edmonds–Karp algorithm and the scaling algorithm (two graphs)	weekplan.pdf p.1	Pasted image.png (both graphs)	Solved
2	KT 7.8	Blood Donations	weekplan.pdf p.1	Pasted image (4).png (table)	Solved
3.1	—	Christmas Trees — model as a graph	weekplan.pdf p.1	Pasted image (2).png (example grid)	Solved
3.2	—	Christmas Trees — algorithm, runtime, correctness	weekplan.pdf p.1	Pasted image (2).png	Solved
4	KT 7.14	Escape	weekplan.pdf p.1	ex_7_14.pdf p. 421	Solved
5	CSES 1696	School Dance	weekplan.pdf p.2	(external task; mapping + I/O)	Solved (method)
6	—	[*] Euler tours in mixed graphs	weekplan.pdf p.2	Pasted image (3).png (example pair)	Solved

Solutions

Exercise 1 — EK & Scaling on two graphs

Concept mapping. Compute a max flow and a min cut. Use EK (BFS) and then capacity-scaling. Tie-breaking: lexicographic by node name on adjacency lists.

Left graph (nodes $s, L, F, C, B, M, G, A, t$).

EK augmentation trace:

Step	Path	δ (delta)	Saturated edges	New residual facts
1	$s \rightarrow C \rightarrow G \rightarrow t$	2	$s \rightarrow C, G \rightarrow t$	back-edges $C \rightarrow s, t \rightarrow G$ get 2
2	$s \rightarrow L \rightarrow F \rightarrow A \rightarrow t$	3	$F \rightarrow A$	back-edges $A \rightarrow F$ get 3
3	$s \rightarrow L \rightarrow F \rightarrow G \rightarrow A \rightarrow t$	3	$F \rightarrow G$	back-edge $G \rightarrow F$ gets 3
4	$s \rightarrow L \rightarrow F \rightarrow G \rightarrow C \rightarrow B \rightarrow M \rightarrow t$	1	$M \rightarrow t$	back-edge $t \rightarrow M$ gets 1

Final value $|f| = 9$. Min-cut certificate: $S = \{s, L, F\}$; crossing edges and capacities: $s \rightarrow C$ (2), $F \rightarrow A$ (3), $F \rightarrow G$ (4); sum $2+3+4 = 9 = |f|$.

Scaling trace (Δ sequence 4,2,1):

- $\Delta=4$: $s \rightarrow L \rightarrow F \rightarrow G \rightarrow A \rightarrow t$, augment 4.
- $\Delta=2$: $s \rightarrow C \rightarrow G \rightarrow t$ (2); $s \rightarrow L \rightarrow F \rightarrow A \rightarrow t$ (2).
- $\Delta=1$: $s \rightarrow L \rightarrow F \rightarrow A \rightarrow G \rightarrow C \rightarrow B \rightarrow M \rightarrow t$ (1).

Same final $|f|=9$ and the cut above.

Right graph (A,B,C,D,E,F,G,H with s,t).

EK augmentation trace:

Step	Path	δ	Saturated edges
1	$s \rightarrow t$	2	$s \rightarrow t$
2	$s \rightarrow D \rightarrow E \rightarrow F \rightarrow t$	3	$D \rightarrow E$, $E \rightarrow F$
3	$s \rightarrow D \rightarrow E \rightarrow G \rightarrow H \rightarrow t$	1	$D \rightarrow E$ (now full)
4	$s \rightarrow D \rightarrow A \rightarrow B \rightarrow C \rightarrow F \rightarrow t$	2	$B \rightarrow C$ (now 3 left), $F \rightarrow t$ (now 0 left)
5	$s \rightarrow D \rightarrow A \rightarrow B \rightarrow C \rightarrow F \rightarrow E \rightarrow G \rightarrow H \rightarrow t$	3	$A \rightarrow B$ (now 0 left), back-edge $F \rightarrow E$ used

Final value $|f|=11$. Min-cut certificate: $S = \{s, D, A, B\}$, crossing edges $s \rightarrow t$ (2), $D \rightarrow E$ (4), $B \rightarrow C$ (5); sum $2+4+5 = 11$.

Scaling trace (Δ sequence 4,2,1):

- $\Delta=4$: $s \rightarrow D \rightarrow E \rightarrow G \rightarrow H \rightarrow t$, augment 4 (bottleneck $D \rightarrow E$).
- $\Delta=4$ (still): $s \rightarrow D \rightarrow A \rightarrow B \rightarrow C \rightarrow F \rightarrow t$, augment 5 (bottleneck $F \rightarrow t$).
- $\Delta=2$: $s \rightarrow t$, augment 2.

✓ **Answer:** Left graph max flow **9** with cut ($S=\{s,L,F\}$); Right graph max flow **11** with cut ($S=\{s,D,A,B\}$).

Pitfalls. Forgetting backward edges after non-unit δ ; treating scaling as “pick the fattest path” (it’s a Δ -filtered residual BFS).

Variant drill. Reduce $A \rightarrow t$ from 6 to 4 on the left graph: EK steps 2–3 would reduce to $\delta=2$ in step 2 and $\delta=2$ in step 3, total $|f|$ becomes **8**; cut remains $S=\{s,L,F\}$ but capacity drops to $2+4+2=8$.

Transfer Pattern. Archetype: min s – t cut via EK/scaling. Cues: “augmenting path trace,” “compute max flow,” small labeled graph. Mapping: vertices as given; capacities on arrows; run EK; certify with (S,T) and sum. Certificate: list crossing edges with capacities.

Exercise 2 — KT 7.8 Blood Donations

Model. Donor types O, A, B, AB; patient types O, A, B, AB. Edges reflect compatibility ($O \rightarrow \text{all}$; $A \rightarrow A, AB$; $B \rightarrow B, AB$; $AB \rightarrow AB$). Capacities: source \rightarrow donor = supply; patient \rightarrow sink = demand.

Computation (integer max-flow). One feasible maximum:

- $O \rightarrow O$: 44, $O \rightarrow A$: 6.
- $A \rightarrow A$: 36.
- $B \rightarrow B$: 8.
- $AB \rightarrow AB$: 3.

Total treated = **97** patients.

Cut certificate & explanation. A must treat 42 but has only 36 type-A units; the extra **6** must come from type O. This leaves at most **50-6 = 44** O-units for O-patients, but O-demand is 45 \Rightarrow at least **one O-patient cannot be served**. This exhibits a min-cut of value 97 and proves optimality.

Plain-English rationale. Since type O is the only universal donor, any shortfall in A/B/AB must be compensated by O; here A is short by 6, so O can cover at most 44 of its own 45 patients.

Transfer Pattern. Archetype: circulation with multiple supplies/demands. Cues: compatibility table, “can this meet all demand?”. Mapping: donors as sources, patients as sinks; capacities as supplies/demands; run max flow; certificate is a small imbalance argument (O is the bottleneck).

✓ **Answer:** Maximum patients served **97**; exactly **1 O-patient** must go untreated.

Exercise 3.1 — Christmas Trees (model)

Graph model. Let rows $R_1 \dots R_n$ (capacity 2 each) and columns $C_1 \dots C_m$ (capacity 1 each). For each empty cell (i, j) (no tree), add edge $R_i \rightarrow C_j$ of capacity 1. Add source $s \rightarrow R_i$ (cap 2) and $C_j \rightarrow t$ (cap 1).

Example figure. The provided 4×8 instance yields a graph with row caps (2,2,2,2), column caps all 1, and edges only for non-tree cells.

Certificate form. A placement is a flow of value k ; its size is the total flow. A min cut proves optimality.

Exercise 3.2 — Christmas Trees (algorithm, runtime, correctness)

Algorithm (slides-first). Compute max flow in the bipartite network above.

```
Algorithm: place_tables
Input: n, m, forbidden  $\subseteq [n] \times [m]$ 
Output: maximum number of tables

build nodes s, {R_i}, {C_j}, t
add edges s  $\rightarrow$  R_i (cap 2) and C_j  $\rightarrow$  t (cap 1)
```

```

for each cell (i,j) not in forbidden: add R_i→C_j (cap 1)
run max_flow (EK or Dinic / Hopcroft-Karp via row-clones)
return |f|
// Time: O(E·√V) with Hopcroft-Karp; O(V·E²) with EK

```

Correctness.

- **Feasibility:** row/column caps enforce “ ≤ 2 per row, ≤ 1 per column”; unit edges enforce one table per chosen cell.
- **Optimality:** by max-flow/min-cut, any flow value equals the capacity of some cut; thus no placement can exceed $|f|$.

Runtime. $V \approx n+m+2$; $E \leq (\text{\#empties})+n+m$. With Hopcroft-Karp on a cloned-rows matching instance: $O(E\sqrt{V})$.

Example value. For the given grid the maximum is **7** (matches the figure’s claim); a cut taking R_2 ’s two units and the occupied columns certifies 7.

Pitfalls. Forgetting to remove edges for tree-cells; using greedy per row (can fail).

Variant drill. If each column allowed at most **2** tables, simply set $C_j \rightarrow t$ capacity = 2.

Transfer Pattern. Archetype: bipartite b-matching via flow.

Exercise 4 — KT 7.14 Escape

(a) Edge-disjoint escape routes.

- Build s^* connecting to each $x \in X$ with cap 1; keep original edges with cap 1; connect each safe node $u \in S$ to t^* with cap 1 (or ∞ —cap 1 suffices if each safe can receive one route; use ∞ if no per-safe bound).
- Run max flow; **feasible iff $|f| = |X|$** ; the unit $s^* \rightarrow x$ edges ensure one route out of every populated node; edge caps enforce edge-disjointness.

(b) Node-disjoint version.

- **Split every vertex v** into $v_{in} \rightarrow v_{out}$ of capacity 1 (except safe nodes if unlimited, set cap = ∞); replace each original (u,v) by $u_{out} \rightarrow v_{in}$ (cap 1). Keep $s^* \rightarrow x_{in}$ (cap 1) and s_{out} of x has cap 1 by the split.
- Run max flow; feasible iff $|f| = |X|$. Split-edges enforce at most one path through any vertex.

Why it works. **Integrality** gives $|f|$ vertex/edge-disjoint paths; the converse is immediate by sending 1 along each path. **Node-splitting** is the standard reduction for node capacities.

Complexity. $O(E\sqrt{V})$ with Dinic / HK on unit capacities, or EK in $O(V \cdot E^2)$ is fine.

Variant drill. If each safe node can absorb at most $c(u)$ evacuees, give $u_{in} \rightarrow u_{out}$ capacity $c(u)$ instead of ∞ .

Transfer Pattern. Archetype: disjoint paths via flow; cues: “routes do not share edges/nodes”. Mapping: unit capacities and super-source/sink; node-split for node constraints.

✓ **Answer:** Build the corresponding unit-capacity flow (with node-splits for (b)); **yes** iff the max flow equals $|X|$.

Exercise 5 — CSES 1696 School Dance (I/O micro-card + method)

- **Inputs.** n boys, m girls, E allowed pairs. Output: maximum set of pairs; list the pairs.
 - **Model.** Bipartite matching: $s \rightarrow \text{boys}$ (cap 1), allowed edges (cap 1), girls $\rightarrow t$ (cap 1).
 - **Algorithm.** Hopcroft-Karp: $O(E\sqrt{n+m})$. Extract matching edges (boy \rightarrow girl) from the flow.
 - **Pitfalls.** Ensure 1-based indices in output as required by CSES; avoid printing unmatched nodes.
-

Exercise 6 — [*] Euler tours in mixed graphs

Goal. Decide if we can orient all undirected edges so the final directed graph has an **Euler tour**.

Key facts (directed Euler). A directed graph has an Euler tour iff every vertex has $\text{in}(v) = \text{out}(v)$ and all non-isolated vertices lie in a single connected component when ignoring directions.

Balances. Let $r(v) = \text{in}_{\text{dir}}(v) - \text{out}_{\text{dir}}(v)$ from the already-directed edges, and let $\text{udeg}(v)$ be the number of incident undirected edges at v .

- If we orient an undirected edge $u-v$ as $u \rightarrow v$, then $r(u) \leftarrow r(u) - 1$ and $r(v) \leftarrow r(v) + 1$.
- Therefore each vertex must receive exactly $h(v) = (\text{udeg}(v) - r(v))/2$ incoming orientations from its incident undirected edges. Feasible only if $0 \leq h(v) \leq \text{udeg}(v)$ for all v and $\text{udeg}(v) \equiv r(v) \pmod{2}$.

Reduction (hint realized). Build a bipartite graph ($E_u \leftrightarrow V$): left nodes are undirected edges; right nodes are vertices; connect edge-node $e = \{u, v\}$ to u and v . Find a **b-matching** that matches each e exactly once and matches each vertex v exactly $h(v)$ times. Implement via flow:

- $s \rightarrow e$ edges (cap 1) for each e in E_u .
- $e \rightarrow u$ and $e \rightarrow v$ edges (cap 1).
- vertex $v \rightarrow t$ edge (cap $h(v)$).

Feasible flow of value $|E_u| \Rightarrow$ choose head of e at the matched endpoint; tail is the other endpoint. Together with the original directed edges, the resulting graph has $\text{in} = \text{out}$ at each v ; connectivity was assumed in the statement when orientations are ignored, hence an Euler tour exists.

Complexity. $O(E\sqrt{V})$ for the b-matching flow; linear to read off orientations.

Pitfalls. Forgetting parity check; demanding strong connectivity beyond what Euler needs; giving some v negative $h(v)$.

Variant drill. If some undirected edges are **forced** to point a specific way, fix them first, update $r(\cdot)$ and $\text{udeg}(\cdot)$, then run the same flow.

Transfer Pattern. Archetype: circulation with vertex demands (exact balances) implemented as an edge-to-vertex b-matching.

✓ **Answer: Yes** iff (i) for all v , $h(v) = (u\deg(v) - r(v))/2$ is an integer in $[0, u\deg(v)]$, and (ii) the b-matching flow succeeds (value $|E_u|$). The b-matching also **constructs** a valid orientation.

Puzzle — 99 Cops (<299 questions)

Task. Identify all corrupt cops. Majority are honest. Query form: “Ask X whether Y is corrupt.”

Strategy.


1. **Find one honest cop (≤ 98 questions).** Boyer–Moore majority trick: keep a candidate c . For each other cop y , ask c about y .
2. If c says “ y is corrupt”, discard y .
3. If c says “ y is honest”, set $c \leftarrow y$. After 98 questions the candidate must be honest (corrupts can only cancel in pairs; honest are a strict majority).
4. **Classify everyone (≤ 98 questions).** Ask the honest cop H about each of the other 98 cops and trust the answers.

Total ≤ 196 questions < 299 .

Why it works. Pairwise cancellations cannot eliminate the strict majority of honest cops, so the survivor is honest; an honest witness then labels everyone correctly.

Summary

- **Patterns used:** EK & scaling; circulations with demands/lower bounds; b-matching; node-splitting; disjoint paths \leftrightarrow flow.
- **Certificates to remember:**
 - Min cut (S, T) with crossing capacities $= |f|$.
 - For blood: A-shortage consumes O-donor slack \rightarrow 1 O-patient must remain.
 - For trees: source row caps + column caps; value = #tables.
 - For mixed Euler: parity & b-matching feasibility.
- **Notational blurb:** $r(v) = \text{in} - \text{out}$ on directed part; $h(v)$ incoming heads required from undirected edges; δ bottleneck on an augmenting path; G_f residual graph; “clone” a row = split capacity into two unit nodes for HK.

Happy studying! 

Network Flow II

Inge Li Gørtz

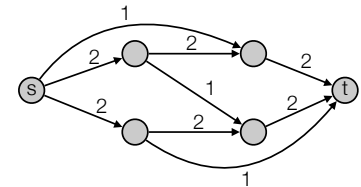
KT 7.3, 7.5, 7.6

1

Network Flow

• Network flow:

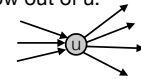
- graph $G=(V,E)$.
- Special vertices s (source) and t (sink).
- Every edge e has a capacity $c(e) \geq 0$.



• 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 minus sum of flows into s :

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

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

2

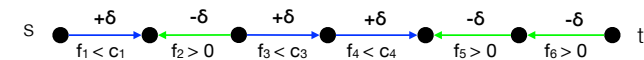
Today

- Applications
- Finding good augmenting paths. Edmonds-Karp and scaling algorithm.

3

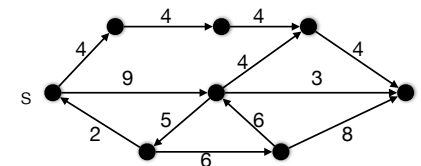
Ford-Fulkerson

- Find (any) augmenting path and use it.
- 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: $\min(c_1 - f_1, f_2, c_3 - f_3, c_4 - f_4, f_5, f_6) = \delta$

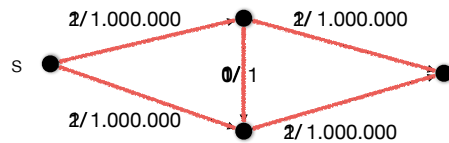
- To find augmenting path use DFS or BFS:



4

Ford-Fulkerson

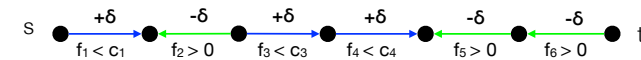
- Integral capacities:
 - Each augmenting path increases flow with at least 1.
 - At most $v(f)$ iterations
 - Find augmenting path via DFS/BFS: $O(m)$
 - Total running time: $O(m \cdot v(f))$
- Lemma.** If all the capacities are integers, then there is a maximum flow where the flow on every edge is an integer.
- Bad example for Ford-Fulkerson:



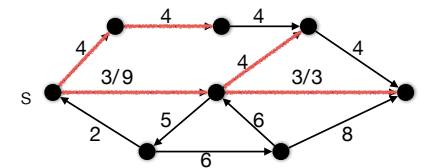
5

Edmonds-Karp

- Find **shortest augmenting path** and use it.
- Augmenting path (definition different than in CLRS): s-t path where
 - forward** edges have leftover capacity
 - backwards** edges have positive flow



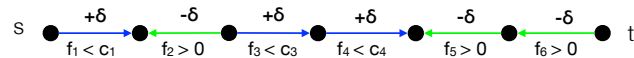
- Can add extra flow: $\min(c_1 - f_1, f_2, c_3 - f_3, c_4 - f_4, f_5, f_6) = \delta$
- To find augmenting path use **BFS**:



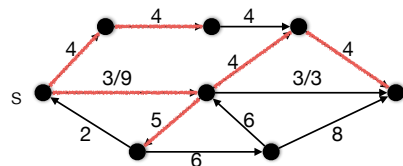
6

Edmonds-Karp

- Find **shortest augmenting path** and use it.
- 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: $\min(c_1 - f_1, f_2, c_3 - f_3, c_4 - f_4, f_5, f_6) = \delta$
- To find augmenting path use **BFS**:



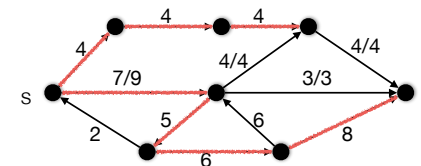
7

Edmonds-Karp

- Find **shortest augmenting path** and use it.
- 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: $\min(c_1 - f_1, f_2, c_3 - f_3, c_4 - f_4, f_5, f_6) = \delta$
- To find augmenting path use **BFS**:



8

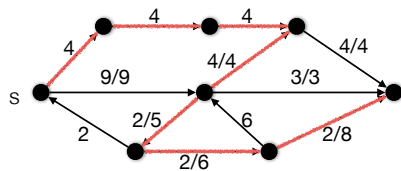
Edmonds-Karp

- Find *shortest augmenting path* and use it.
- 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: $\min(c_1 - f_1, f_2, c_3 - f_3, c_4 - f_4, f_5, f_6) = \delta$

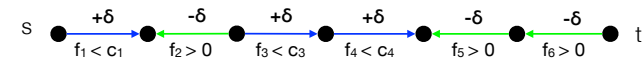
- To find augmenting path use *BFS*:



9

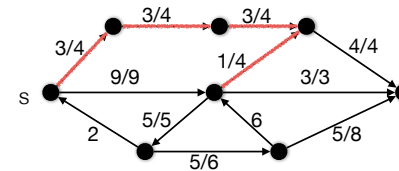
Edmonds-Karp

- Find *shortest augmenting path* and use it.
- 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: $\min(c_1 - f_1, f_2, c_3 - f_3, c_4 - f_4, f_5, f_6) = \delta$

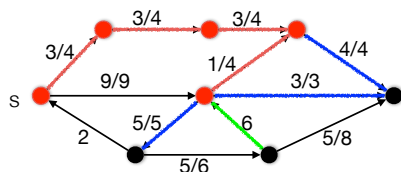
- To find augmenting path use *BFS*:



10

Find a minimum cut

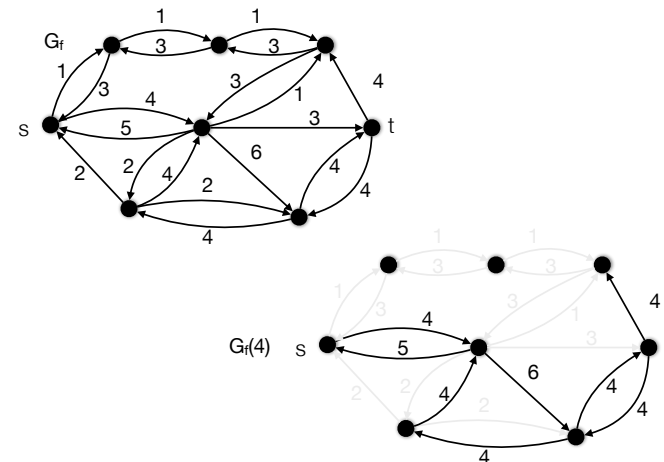
- When there are no more augmenting s-t paths:
- Find all augmenting paths from s.
- The nodes S that can be reached by these augmenting paths form the left side of a minimum cut.
 - edges out* of S have $c_e = f_e$.
 - edges into* S have $f_e = 0$.
 - Capacity of the cut equals the flow.



11

Scaling algorithm

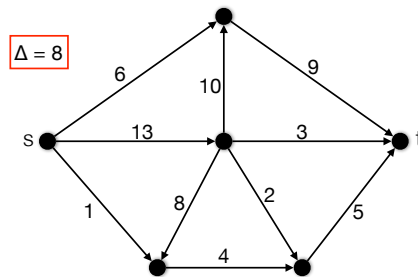
- Scaling parameter Δ
- Only consider edges with capacity at least Δ in residual graph $G_r(\Delta)$.
- Example: $\Delta = 4$



12

Scaling algorithm

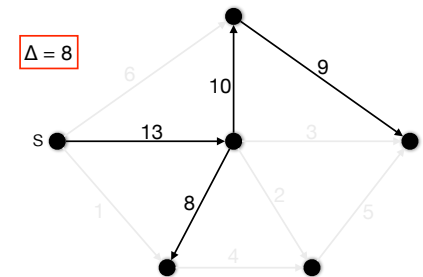
- Scaling parameter Δ
- Only consider edges with capacity at least Δ in residual graph $G_r(\Delta)$.
- Start with $\Delta = \text{"highest power of } 2 \leq \text{largest capacity out of } s\text{"}$



13

Scaling algorithm

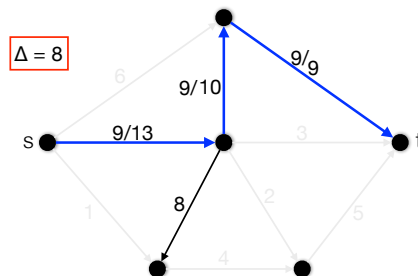
- Scaling parameter Δ
- Only consider edges with capacity at least Δ in residual graph $G_r(\Delta)$.
- Start with $\Delta = \text{"highest power of } 2 \leq \text{largest capacity out of } s\text{"}$



14

Scaling algorithm

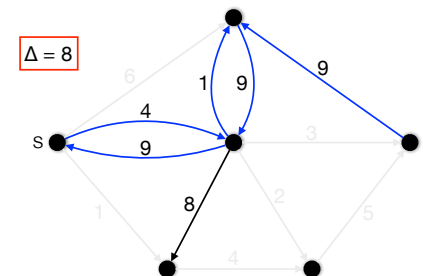
- Scaling parameter Δ
- Only consider edges with capacity at least Δ in residual graph $G_r(\Delta)$.
- Start with $\Delta = \text{"highest power of } 2 \leq \text{largest capacity out of } s\text{"}$



15

Scaling algorithm

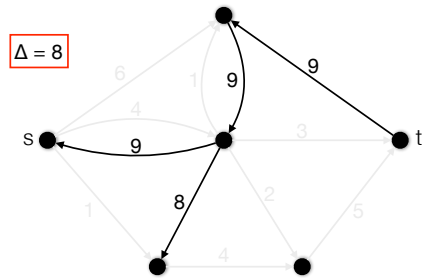
- Scaling parameter Δ
- Only consider edges with capacity at least Δ in residual graph $G_r(\Delta)$.
- Start with $\Delta = \text{"highest power of } 2 \leq \text{largest capacity out of } s\text{"}$



16

Scaling algorithm

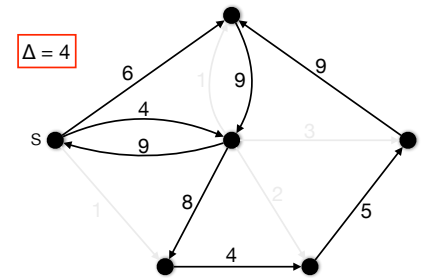
- Scaling parameter Δ
- Only consider edges with capacity at least Δ in residual graph $G_r(\Delta)$.
- Start with $\Delta = \text{"highest power of } 2 \leq \text{largest capacity out of } s\text{"}$
- When no more augmenting paths in $G_r(\Delta)$: $\Delta = \Delta/2$ (new phase).



17

Scaling algorithm

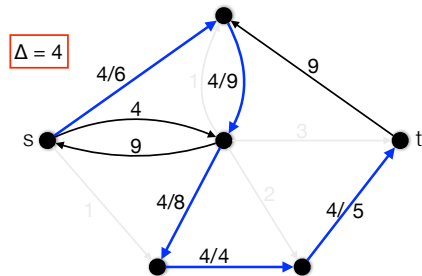
- Scaling parameter Δ
- Only consider edges with capacity at least Δ in residual graph $G_r(\Delta)$.
- Start with $\Delta = \text{"highest power of } 2 \leq \text{largest capacity out of } s\text{"}$
- When no more augmenting paths in $G_r(\Delta)$: $\Delta = \Delta/2$ (new phase).



18

Scaling algorithm

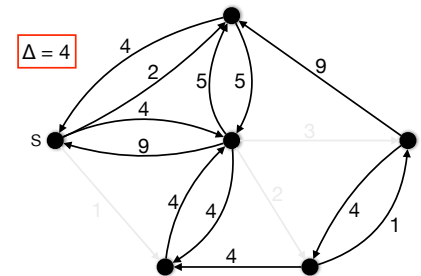
- Scaling parameter Δ
- Only consider edges with capacity at least Δ in residual graph $G_r(\Delta)$.
- Start with $\Delta = \text{"highest power of } 2 \leq \text{largest capacity out of } s\text{"}$
- When no more augmenting paths in $G_r(\Delta)$: $\Delta = \Delta/2$ (new phase).



19

Scaling algorithm

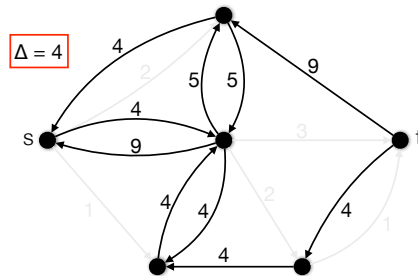
- Scaling parameter Δ
- Only consider edges with capacity at least Δ in residual graph $G_r(\Delta)$.
- Start with $\Delta = \text{"highest power of } 2 \leq \text{largest capacity out of } s\text{"}$
- When no more augmenting paths in $G_r(\Delta)$: $\Delta = \Delta/2$ (new phase).



20

Scaling algorithm

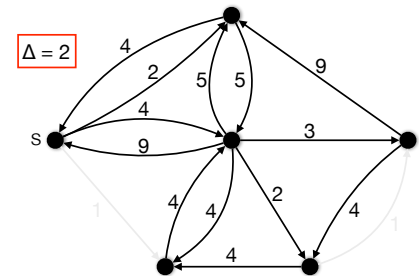
- Scaling parameter Δ
- Only consider edges with capacity at least Δ in residual graph $G_r(\Delta)$.
- Start with $\Delta = \text{"highest power of 2} \leq \text{largest capacity out of } s\text{"}$
- When no more augmenting paths in $G_r(\Delta)$: $\Delta = \Delta/2$ (new phase).



21

Scaling algorithm

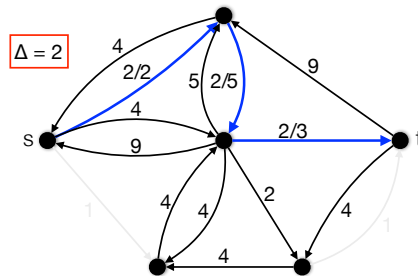
- Scaling parameter Δ
- Only consider edges with capacity at least Δ in residual graph $G_r(\Delta)$.
- Start with $\Delta = \text{"highest power of 2} \leq \text{largest capacity out of } s\text{"}$
- When no more augmenting paths in $G_r(\Delta)$: $\Delta = \Delta/2$ (new phase).



22

Scaling algorithm

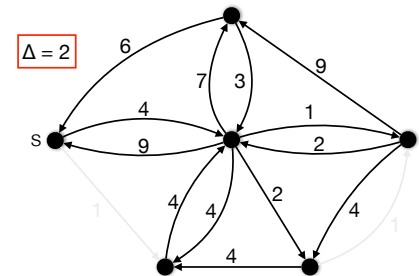
- Scaling parameter Δ
- Only consider edges with capacity at least Δ in residual graph $G_r(\Delta)$.
- Start with $\Delta = \text{"highest power of 2} \leq \text{largest capacity out of } s\text{"}$
- When no more augmenting paths in $G_r(\Delta)$: $\Delta = \Delta/2$ (new phase).



23

Scaling algorithm

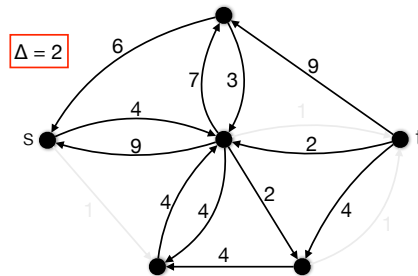
- Scaling parameter Δ
- Only consider edges with capacity at least Δ in residual graph $G_r(\Delta)$.
- Start with $\Delta = \text{"highest power of 2} \leq \text{largest capacity out of } s\text{"}$
- When no more augmenting paths in $G_r(\Delta)$: $\Delta = \Delta/2$ (new phase).



24

Scaling algorithm

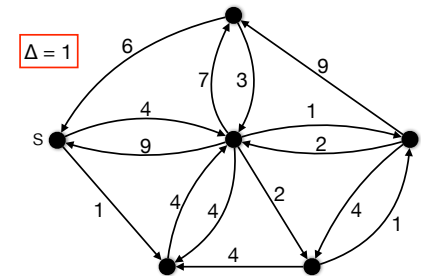
- Scaling parameter Δ
- Only consider edges with capacity at least Δ in residual graph $G_r(\Delta)$.
- Start with $\Delta = \text{"highest power of 2} \leq \text{largest capacity out of } s\text{"}$
- When no more augmenting paths in $G_r(\Delta)$: $\Delta = \Delta/2$ (new phase).



25

Scaling algorithm

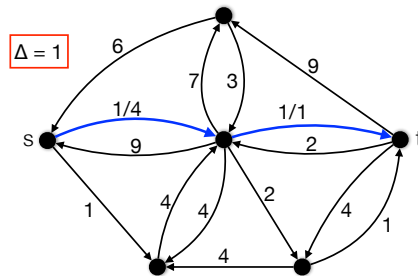
- Scaling parameter Δ
- Only consider edges with capacity at least Δ in residual graph $G_r(\Delta)$.
- Start with $\Delta = \text{"highest power of 2} \leq \text{largest capacity out of } s\text{"}$
- When no more augmenting paths in $G_r(\Delta)$: $\Delta = \Delta/2$ (new phase).



26

Scaling algorithm

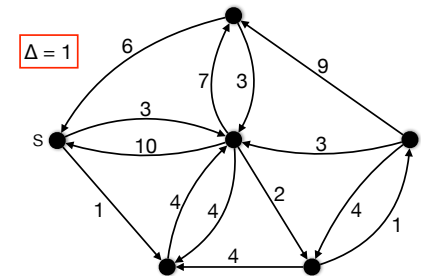
- Scaling parameter Δ
- Only consider edges with capacity at least Δ in residual graph $G_r(\Delta)$.
- Start with $\Delta = \text{"highest power of 2} \leq \text{largest capacity out of } s\text{"}$
- When no more augmenting paths in $G_r(\Delta)$: $\Delta = \Delta/2$ (new phase).



27

Scaling algorithm

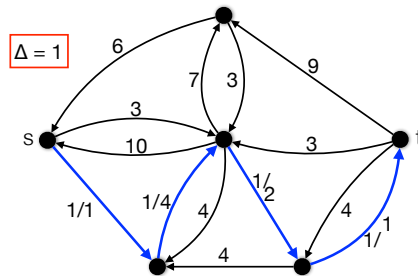
- Scaling parameter Δ
- Only consider edges with capacity at least Δ in residual graph $G_r(\Delta)$.
- Start with $\Delta = \text{"highest power of 2} \leq \text{largest capacity out of } s\text{"}$
- When no more augmenting paths in $G_r(\Delta)$: $\Delta = \Delta/2$ (new phase).



28

Scaling algorithm

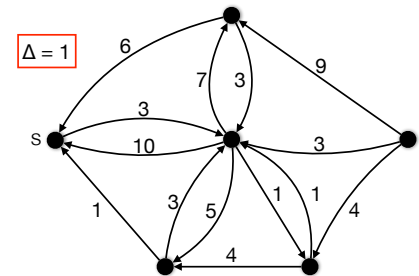
- Scaling parameter Δ
- Only consider edges with capacity at least Δ in residual graph $G_r(\Delta)$.
- Start with $\Delta = \text{"highest power of 2} \leq \text{largest capacity out of } s\text{"}$
- When no more augmenting paths in $G_r(\Delta)$: $\Delta = \Delta/2$ (new phase).



29

Scaling algorithm

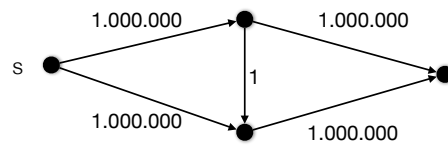
- Scaling parameter Δ
- Only consider edges with capacity at least Δ in residual graph $G_r(\Delta)$.
- Start with $\Delta = \text{"highest power of 2} \leq \text{largest capacity out of } s\text{"}$
- When no more augmenting paths in $G_r(\Delta)$: $\Delta = \Delta/2$ (new phase).



- Stop when no more augmenting paths in $G_r(1)$.

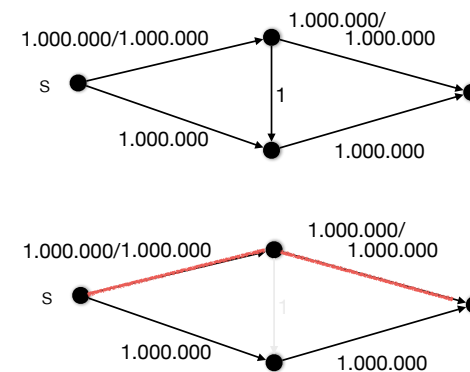
30

Scaling algorithm



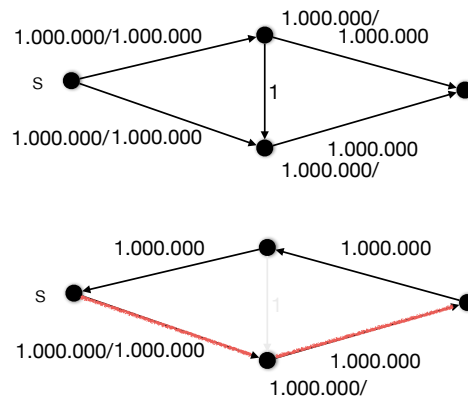
31

Scaling algorithm



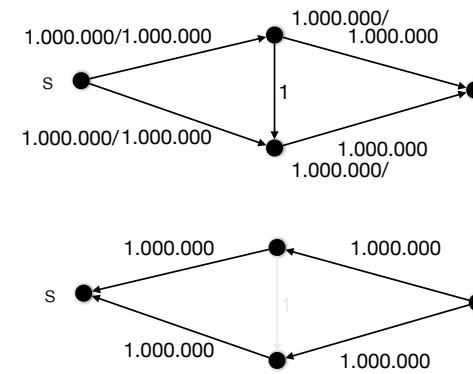
32

Scaling algorithm



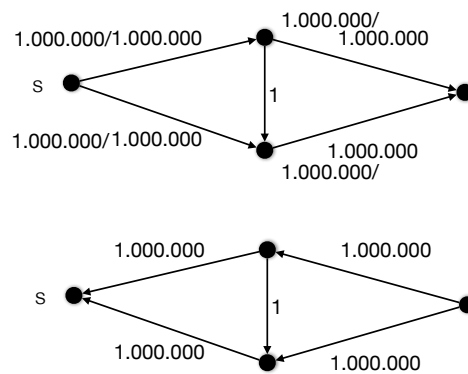
33

Scaling algorithm



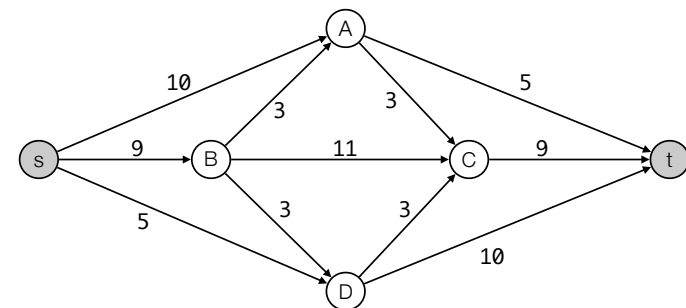
34

Scaling algorithm



35

Exercise



Scaling algorithm

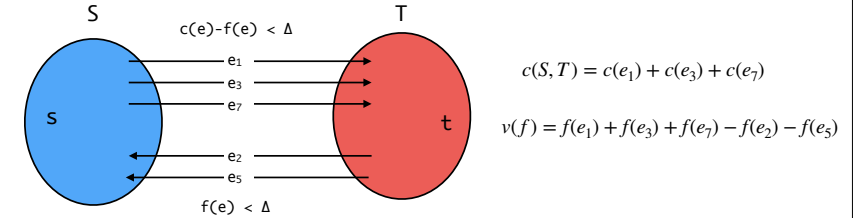
- **Running time:** $O(m^2 \log C)$, where C is the largest capacity out of s .
- **Lemma 1.** Number of scaling phases: $1 + \lceil \lg C \rceil$
- Prove: number of augmentation paths found in a scaling phase is at most $2m$.
- **Lemma 2.** Let f be the flow when Δ -scaling phase ends, and let f^* be the maximum flow. Then $v(f^*) \leq v(f) + m\Delta$.

37

Scaling algorithm

- **Lemma 2.** Let f be the flow when Δ -scaling phase ends, and let f^* be the maximum flow. Then $v(f^*) \leq v(f) + m\Delta$.

- By the end of the phase there is a cut $c(S, T) \leq v(f) + m\Delta$.



$$\begin{aligned} c(S, T) - v(f) &= c(e_1) + c(e_3) + c(e_7) - f(e_1) - f(e_3) - f(e_7) + f(e_2) + f(e_5) \\ &= c(e_1) - f(e_1) + c(e_3) - f(e_3) + c(e_7) - f(e_7) + f(e_2) + f(e_5) \\ &< \Delta + \Delta + \Delta + \Delta + \Delta = 5\Delta \end{aligned}$$

38

Scaling algorithm

- **Running time:** $O(m^2 \log C)$, where C is the largest capacity out of s .
- **Lemma 1.** Number of scaling phases: $1 + \lceil \lg C \rceil$
- **Lemma 2.** Let f be the flow when Δ -scaling phase ends, and let f^* be the maximum flow. Then $v(f^*) \leq v(f) + m\Delta$.
- **Lemma 3.** The number of augmentations in a scaling phase is at most $2m$.
 - First phase: can use each edge out of s in at most one augmenting path.
 - f flow at the end of previous phase.
 - Used $\Delta' = 2\Delta$ in last round.
 - Lemma 2: $v(f^*) \leq v(f) + m\Delta' = v(f) + 2m\Delta$.
 - “Leftover flow” to be found $\leq 2m\Delta$.
 - Each augmentation in a Δ -scaling phase augments flow with at least Δ .

39

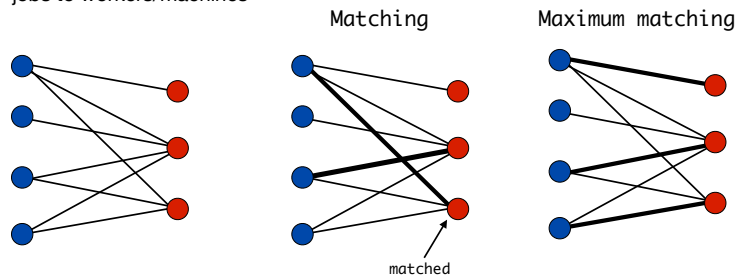
Maximum flow algorithms

- **Edmonds-Karp:** $O(m^2n)$
- **Scaling:** $O(m^2 \log C)$
- **Ford-Fulkerson:** $O(m v(f))$.
- Preflow-push $O(n^3)$
- Other algorithms: $O(mn \log n)$ or $O(\min(n^{2/3}, m^{1/2})m \log n \log U)$.

40

Maximum Bipartite Matching

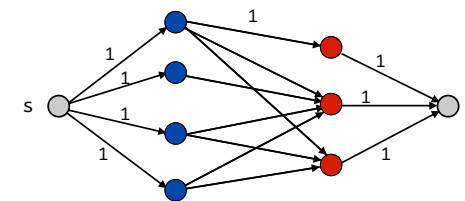
- **Bipartite graph:** Can color vertices red and blue such that all edges have a red and a blue endpoint.
- **Matching:** Subset of edges $M \subseteq E$ such that no edges in M share an endpoint.
- **Maximum matching:** matching of maximum cardinality.
- **Applications:**
 - planes to routes
 - jobs to workers/machines



41

Maximum Bipartite Matching

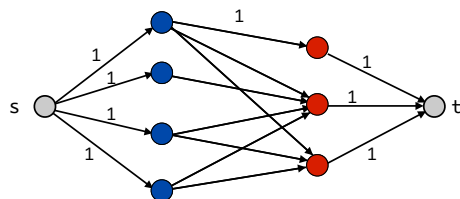
- **Bipartite graph:** Can color vertices red and blue such that all edges have a red and a blue endpoint.
- **Matching:** Subset of edges $M \subseteq E$ such that no edges in M share an endpoint.
- **Maximum matching:** matching of maximum cardinality.
- Solve via flow:



42

Maximum Bipartite Matching

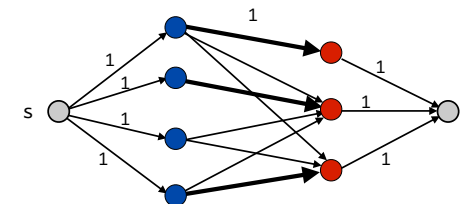
- **Bipartite graph:** Can color vertices red and blue such that all edges have a red and a blue endpoint.
- **Matching:** Subset of edges $M \subseteq E$ such that no edges in M share an endpoint.
- **Maximum matching:** matching of maximum cardinality.
- Solve via flow:
 - Matching $M \Rightarrow$ flow of value $|M|$



43

Maximum Bipartite Matching

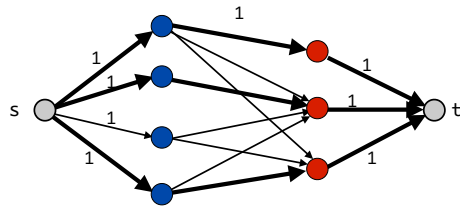
- **Bipartite graph:** Can color vertices red and blue such that all edges have a red and a blue endpoint.
- **Matching:** Subset of edges $M \subseteq E$ such that no edges in M share an endpoint.
- **Maximum matching:** matching of maximum cardinality.
- Solve via flow:
 - Matching $M \Rightarrow$ flow of value $|M|$



44

Maximum Bipartite Matching

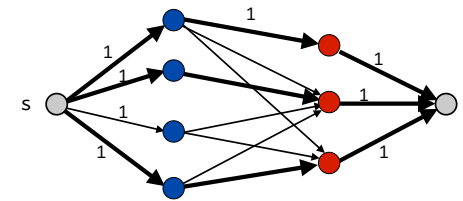
- **Bipartite graph:** Can color vertices red and blue such that all edges have a red and a blue endpoint.
- **Matching:** Subset of edges $M \subseteq E$ such that no edges in M share an endpoint.
- **Maximum matching:** matching of maximum cardinality.
- Solve via flow:
 - Matching $M \Rightarrow$ flow of value $|M|$



45

Maximum Bipartite Matching

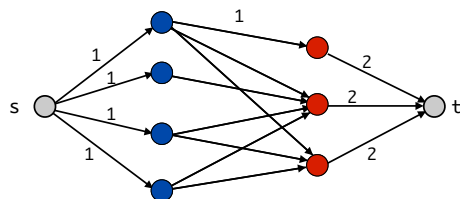
- **Bipartite graph:** Can color vertices red and blue such that all edges have a red and a blue endpoint.
- **Matching:** Subset of edges $M \subseteq E$ such that no edges in M share an endpoint.
- **Maximum matching:** matching of maximum cardinality.
- Solve via flow:
 - Matching $M \Rightarrow$ flow of value $|M|$
 - Flow of value $v(f) \Rightarrow$ matching of size $v(f)$



46

Maximum Bipartite Matching

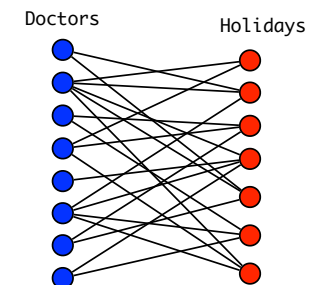
- **Bipartite graph:** Can color vertices red and blue such that all edges have a red and a blue endpoint.
- **Matching:** Subset of edges $M \subseteq E$ such that no edges in M share an endpoint.
- **Maximum matching:** matching of maximum cardinality.
- Solve via flow:
 - Can generalize to general matchings



47

Scheduling of doctors

- X doctors, Y holidays, each doctor should work at at most 1 holiday, each doctor is available at some of the holidays.

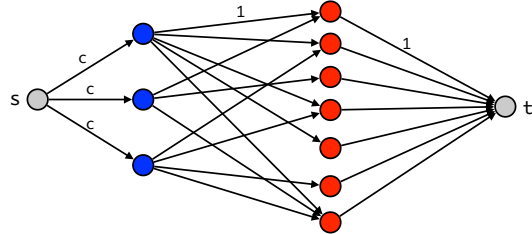


- Same problem, but each doctor should work at most c holidays?

48

Scheduling of doctors

- X doctors, Y holidays, each doctor should work at at most c holidays, each doctor is available at some of the holidays.

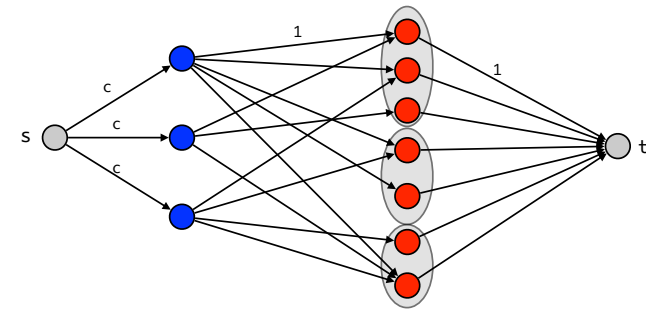


- Same problem, but each doctor should work at most one day in each vacation period?

49

Scheduling of doctors

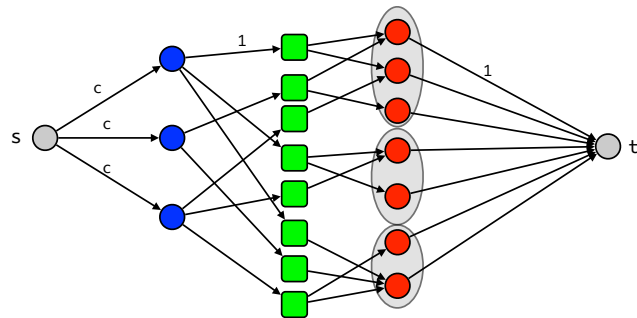
- X doctors, Y holidays, each doctor should work at at most c holidays, each doctor is available at some of the holidays.
- Same problem, but each doctor should work at most one day in each vacation period?



50

Scheduling of doctors

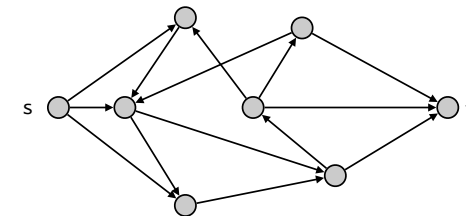
- X doctors, Y holidays, each doctor should work at at most c holidays, each doctor is available at some of the holidays.
- Same problem, but each doctor should work at most one day in each vacation period?



51

Edge Disjoint paths

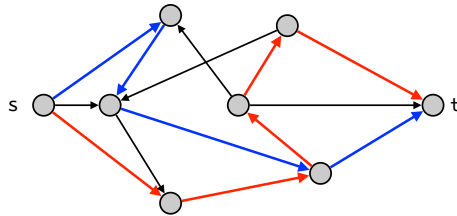
- Problem: Find maximum number of edge-disjoint paths from s to t.
- Two paths are edge-disjoint if they have no edge in common.



52

Edge Disjoint paths

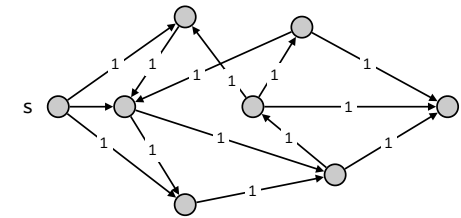
- **Edge-disjoint path problem.** Find the maximum number of edge-disjoint paths from s to t .
- Two paths are edge-disjoint if they have no edge in common.



53

Edge Disjoint Paths

- Reduction to max flow: assign capacity 1 to each edge.

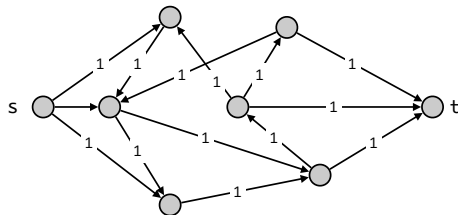


- **Thm.** Max number of edge-disjoint s - t paths is equal to the value of a maximum flow.
 - Suppose there are k edge-disjoint paths: then there is a flow of k (let all edges on the paths have flow 1).
 - Other way (graph theory course).
- Ford-Fulkerson: $v(f) \leq n$ (no multiple edges and therefore at most n edges out of s)
 \Rightarrow running time $O(nm)$.

54

Network Connectivity

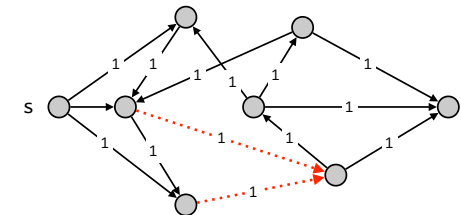
- **Network connectivity.** Find minimum number of edges whose removal disconnects t from s (destroys all s - t paths).



55

Network Connectivity

- **Network connectivity.** Find minimum number of edges whose removal disconnects t from s (destroys all s - t paths).



- Set all capacities to 1 and find minimum cut.
- **Thm. (Menger)** The maximum number of edge-disjoint s - t paths is equal to the minimum number of edges whose removal disconnects t from s .

56

Baseball elimination

Team	Wins	Games left	Against			
			NY	Bal	Tor	Bos
New York	92	2	-	1	1	0
Baltimore	91	3	1	-	1	1
Toronto	91	3	1	1	-	1
Boston	90	2	0	1	1	-

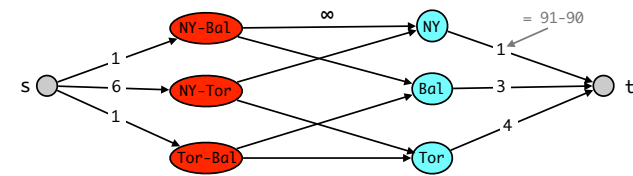
- Question: Can Boston finish in first place (or in tie of first place)?
- No: Boston must win both its remaining 2 and NY must loose. But then Baltimore and Toronto both beat NY so winner of Baltimore-Toronto will get 93 points.
- Other argument: Boston can finish with at most 92. Cumulatively the other three teams have 274 wins currently and their 3 games against each other will give another 3 points => 277. $277/3 = 92,33333$ => one of them must win > 92 .

57

Baseball elimination

Team	Wins	Games left	Against			
			NY	Bal	Tor	Bos
New York	90	11	-	1	6	4
Baltimore	88	6	1	-	1	4
Toronto	87	11	6	1	-	4
Boston	79	12	4	4	4	-

- Question: Can Boston finish in first place (or in tie of first place)?



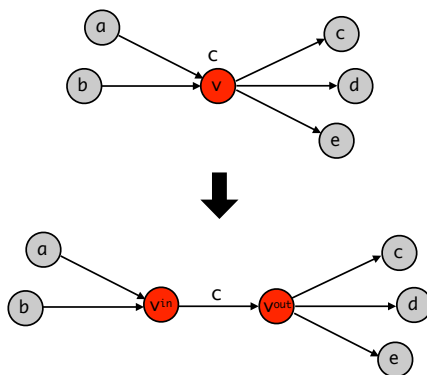
Boston can get at most $79 + 12 = 91$ points

- Boston cannot win (or tie) \Leftrightarrow max s-t flow < 8 .

58

Node capacities

- Capacities on nodes.



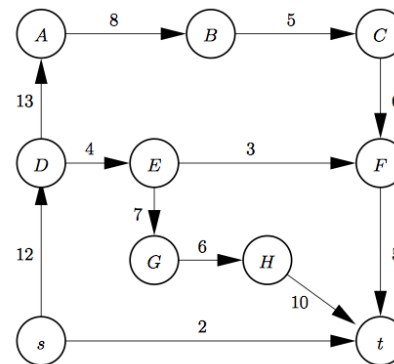
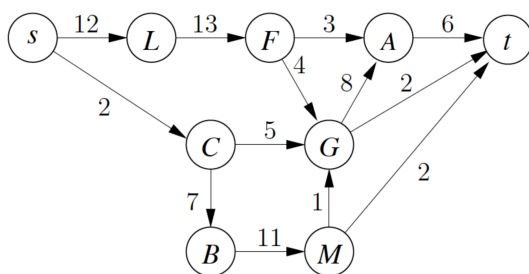
59

Reading Material

At the lecture we will talk more about network flow. We will consider two applications: bipartite matching and finding disjoint paths. We will also discuss how to find good augmenting paths. For reading material see the webpage.

Exercises

1 [w] The Edmonds-Karp algorithm and the scaling algorithm Use both the Edmonds-Karp's algorithm and the scaling algorithm to compute a maximum flow and minimum cut on the two graphs below. For each augmenting path write the nodes on the path and the value you augment the path with.



2 Blood Donations Solve exercise KT 7.8.

3 Christmas Trees (from the Exam E15) The Dean has asked you to arrange the annual Christmas party for the students at DTU. You have to make a plan for how to place the tables in the hall. The local fire department has divided the hall up into an $n \times m$ grid of subsquares and declared that you can place at most two tables in each row and at most one in each column. Unfortunately, the Dean who loves Christmas has put up Christmas trees in many of the subsquares. You cannot place a table in a subsquare with a Christmas tree.

Example Here $n = 4$ and $m = 8$. The $*$ are Christmas trees and T are tables. In the example the maximum number of tables that can be placed is 7.

*	T						T
T	*	*	T	*	*		*
	*	*		*	*	T	*
	*	T	*		T	*	

3.1 Model the problem as a graph problem. Explain how you model the problem as a graph problem and draw the graph corresponding to the example above.

3.2 Describe an algorithm that given n , m , and the placement of the Christmas trees computes the maximum number of tables you can place in the hall. Analyze the asymptotic running time of your algorithm. Remember to argue that your algorithm is correct.

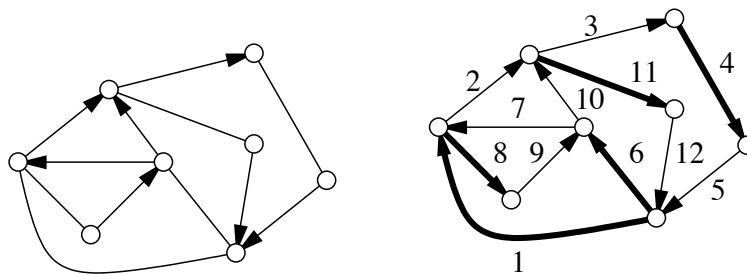
4 Escape Solve exercise KT 7.14.

5 CSES School dance Solve <https://cses.fi/problemset/task/1696>

6 [*] Euler tours in mixed graphs It is well-known that a strongly connected graph has an Euler tour (a cycle that contains each edge exactly once) if and only if any vertex has the same number of ingoing and outgoing edges.

In the following we consider mixed graphs, which are graphs where some edges are directed and some are not. We assume that the graphs are connected, i.e., if we ignore the orientation on all directed edges, then the graph is connected.

We want an algorithm to decide whether it is possible to assign directions to all undirected edges, such that the graph has an Euler tour. In the example below the mixed graph on the left can be assigned directions (graph on right) such that there is an Euler tour (the numbers on the edges denotes the order the edges can be visited in).



Give an algorithm to decide whether it is possible to assign directions to all undirected edges, such that a graph G has an Euler tour.

Hint: Consider a bipartite graph, where the nodes on one side are edges in G and the nodes on the other side are the vertices in G .

Puzzle of the week: 99 Cops A town has 99 cops. A cop is either honest or corrupt, the majority of the cops is honest. You need to figure out all the corrupt cops, with less than 299 questions. All cops know who is honest and who is corrupt, but only honest cops will answer truthfully. Corrupt cops may lie arbitrarily. For security reasons you can only ask one type of question: You may ask cop X whether cop Y is corrupt. This question will be answered by X with either "Y is corrupt" or "Y is honest".