# Flow Network: $F = (G, c, s, t)$, where

(directed graph; capacity; source; sink (target))

- $G = (V, E)$
- $c : E \to \mathbb{R}^+$
- $s, t \in V$
- $\nexists v \cdot (v, s) \in E$ or $(t, v) \in E$


---

**Flow**: an s-t flow in F is a function f satisfying

- $f : E \to \mathbb{R}^+$
- $\forall e \in E \cdot 0 \le f(e) \le c(e)$ (never break capacity)
- $\forall v \in V \setminus \{s,t\} \cdot \underbrace{\sum_{\{e=(v',v)\in E \mid v' \in V\}} f(e)}_{=: f^{in}(v)} = \underbrace{\sum_{\{e=(v,v')\in E \mid v' \in V\}} f(e)}_{=: f^{out}(v)}$ (no leak)

---

(Generalized to sets: $f^{in}(S) = \sum_{v \in S} f^{in}(v)$ $\quad f^{out}(S) = \sum_{v \in S} f^{out}(v)$)

---

**Value of flow f:**

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

---

**Max flow problem!** Given F,

Find s-t flow f in F with maximal $v(f)$. ($f_{max}$)

---

Ford-Fulkerson (FF) algorithm solves this problem in $\mathcal{O}(mC)$, with $m = |E|$ and $C = \sum_{\{e=(s,v)\in E \mid v \in V\}} c(e)$.

---

(if C is big but $c(e) \in \mathbb{Z}, \forall e$, then we can use an augmented version of FF which runs in $\mathcal{O}(m^2 \log C)$.)

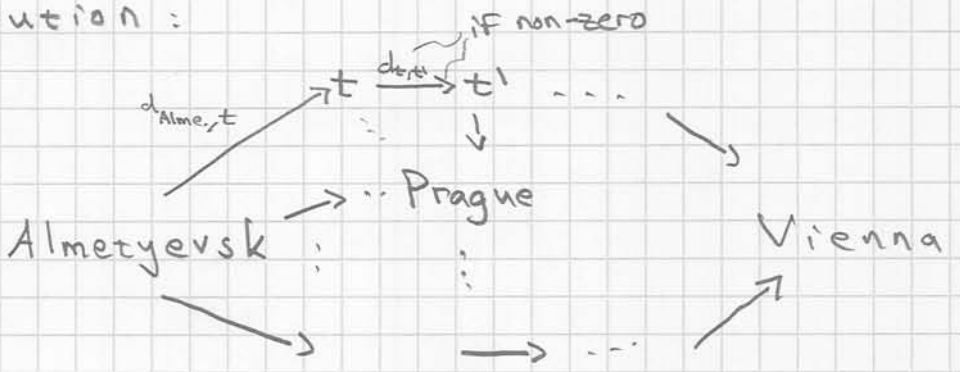
# Oil pipeline

$d_{t,t'}$ : Amount of oil which can be transported through the oil pipe from $t$ to $t'$.

How much oil can we transport from Almetyevsk to Vienna in 1 day?

—

Solution :




Fmax answers problem. ← computed with FF alg.

Fmax also says how to route oil to obtain max flow.

# Recipe (algorithm) for solving problems using network flow

1. Express constraints of your problem as a flow graph F.
2. Prove that a max flow $f_{max}$ in F yields the solution you are looking for. (existence or value)
3. Run the Ford-Fulkerson algorithm on F to compute $f_{max}$.

Note:

- if asked for efficient implementation, remember to include time it takes to construct F in 1..
- 2. usual pattern!

p : real-world property of interest
p'(f): property of flows

Claim: $p'(f_{max}) \iff p$

proof: Prove each direction of $\iff$

$\Rightarrow$: Assume $p'(f_{max})$.
To prove: p.

Use $f_{max}$ to construct evidence of truth of p.

$\Leftarrow$: Assume p.
To prove: $p'(f_{max})$.

p true $\Rightarrow$ p has evidence $\mathcal{E}$ of truth.
Use $\mathcal{E}$ to construct an f.
Argue why f is maximal. ∎

(Each direction a "proof by construction".

## Solved Exercise 2

You are helping the medical consulting firm Doctors Without Weekends set up the work schedules of doctors in a large hospital. They've got the regular daily schedules mainly worked out. Now, however, they need to deal with all the special cases and, in particular, make sure that they have at least one doctor covering each vacation day.

There are $n$ doctors at the hospital, and doctor $i$ has a set of vacation days $S_i$ when he or she is available to work. (This may include certain days from a given vacation period but not others; so, for example, a doctor may be able to work the Friday, Saturday, or Sunday of Thanksgiving weekend, but not the Thursday.)

Give a polynomial-time algorithm that takes this information and determines whether it is possible to select a single doctor to work on each vacation day, subject to the following constraint.

- For a given parameter $c$, each doctor should be assigned to work at most $c$ vacation days total, and only days when he or she is available.

The algorithm should either return an assignment of doctors satisfying these constraints or report (correctly) that no such assignment exists.

# Solved Exercise 2, KT7

(a)

$n$ : nr. of doctors

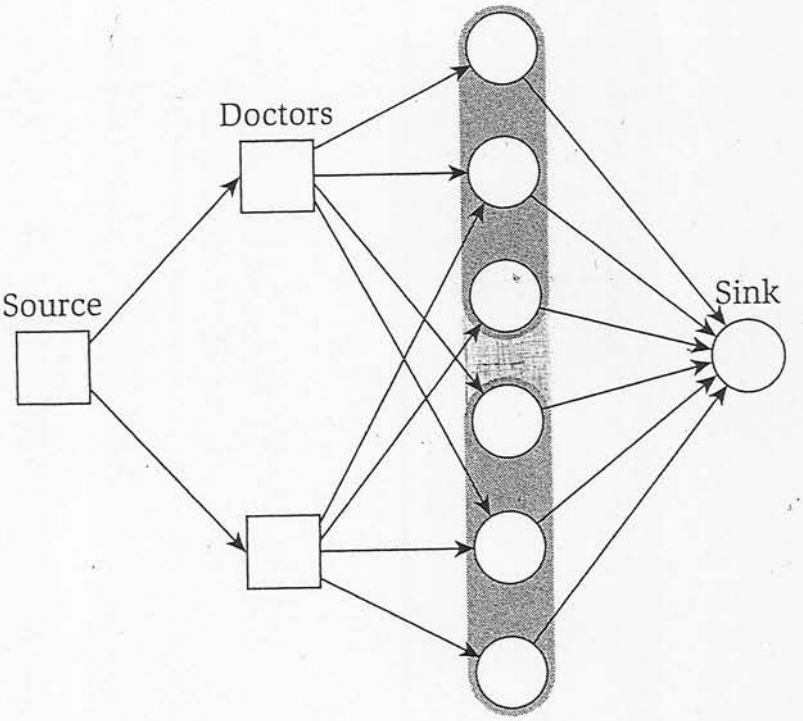
$D$ : set of vacation days ; $|D| = d$

$S_i \subseteq D$ : set of vacation days doctor $i$ can work.

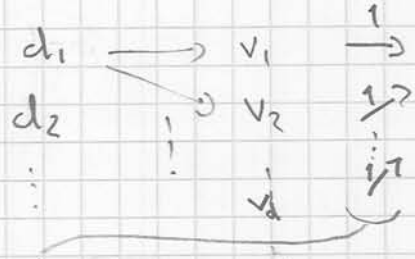
$c$ : max nr. of vacation days a doctor can work

Give polynomial algorithm which

- returns assignment of doctors to vacation days, or
- reports when no such assignment exists.


Doctors
Source
Sink

(a)



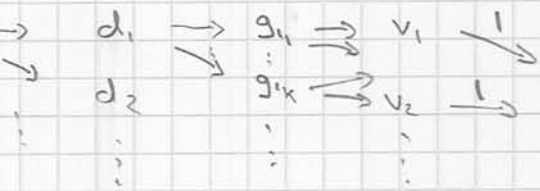

=: F

Corr: $v(f_{max}) = d \iff$ all vacation days coverable.

proof: $\Rightarrow$ : construct doctor-day map g by letting $v_j \in g(d_i)$ if there is flow on $(d_i, v_j)$.

$\Leftarrow$ : given g, construct $f_{max}$ by setting flow on all $(v_j, t)$ to 1, on $(d_i, v_j)$ to 1 if $v_j \in g(d_i)$, and on $(s, d_i)$ to the sum of $(d_i, v_j)$ flows. ▨

(b)




F is constructed with this "trick" in mind.

## Solved Exercise 2

You are helping the medical consulting firm Doctors Without Weekends set up the work schedules of doctors in a large hospital. They've got the regular daily schedules mainly worked out. Now, however, they need to deal with all the special cases and, in particular, make sure that they have at least one doctor covering each vacation day.

Here's how this works. There are $k$ vacation periods (e.g., the week of Christmas, the July 4th weekend, the Thanksgiving weekend, . . . ), each spanning several contiguous days. Let $D_j$ be the set of days included in the $j^{\text{th}}$ vacation period; we will refer to the union of all these days, $\cup_j D_j$, as the set of all *vacation days*.

There are $n$ doctors at the hospital, and doctor $i$ has a set of vacation days $S_i$ when he or she is available to work. (This may include certain days from a given vacation period but not others; so, for example, a doctor may be able to work the Friday, Saturday, or Sunday of Thanksgiving weekend, but not the Thursday.)

Give a polynomial-time algorithm that takes this information and determines whether it is possible to select a single doctor to work on each vacation day, subject to the following constraints.

- For a given parameter $c$, each doctor should be assigned to work at most $c$ vacation days total, and only days when he or she is available.
- For each vacation period $j$, each doctor should be assigned to work at most one of the days in the set $D_j$. (In other words, although a particular doctor may work on several vacation days over the course of a year, he or she should not be assigned to work two or more days of the Thanksgiving weekend, or two or more days of the July 4th weekend, etc.)

The algorithm should either return an assignment of doctors satisfying these constraints or report (correctly) that no such assignment exists.

# Solved Exercise 2, KT7

(a) $n$ : nr. of doctors

$D$ : set of vacation days ; $|D| = d$

$S_i \subseteq D$ : set of vacation days doctor $i$ can work.

$c$ : max nr. of vacation days a doctor can work

Give polynomial algorithm which

- returns assignment of doctors to vacation days, or
- reports when no such assignment exists.

---

(b) $k$ : vacation periods

$D_j \subseteq D$ : days in period $j$

Each doctor should be assigned to at most 1 day in each period.

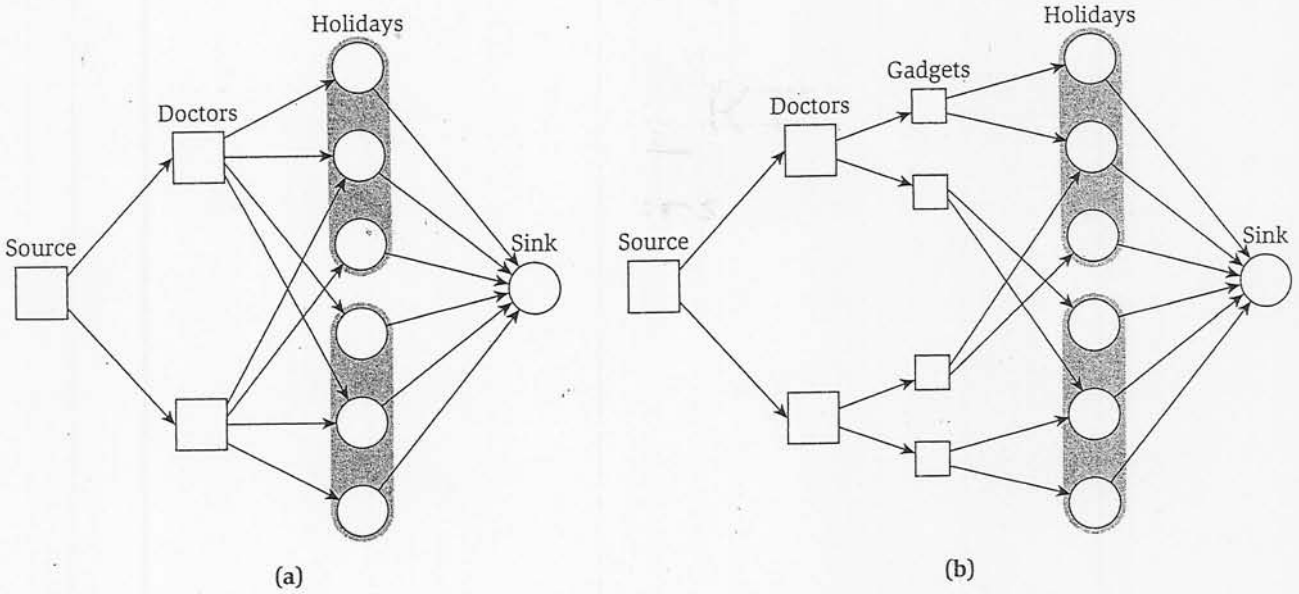


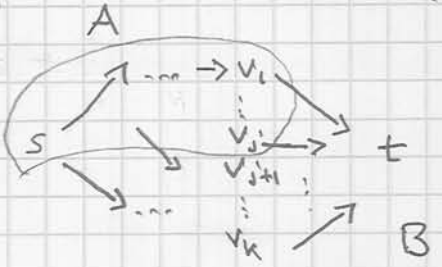

**Figure 7.23** (a) Doctors are assigned to holiday days without restricting how many days in one holiday a doctor can work. (b) The flow network is expanded with "gadgets" that prevent a doctor from working more than one day from each vacation period. The shaded sets correspond to the different vacation periods.

$v(f_{max}) \leq \sum_{\{e=(s,v) \mid v \in V\}} c(e)$ ; at best, all capacity out of $s$ used. Better upper bound?

## Cut in $F = (G, c, s, t)$: (s-t cut)

"a partition $(A, B)$ of $V$ s.t.

$s \in A$, and
$t \in B$."




## Capacity

- $c(A, B) = \sum_{\{e=(v,w) \mid v \in A\}} c(e)$ — at most $c(A,B)$ flow can leave $A$

## Properties: For any s-t flow $f$ and cut $(A, B)$,

- $v(f) = f^{out}(A) - f^{in}(A)$
  - $f^{out}(A)$: flow leaving $A$; $f^{in}(A)$: flow re-entering $A$
  - $= f^{in}(B) - f^{out}(B)$
    - $f^{in}(B)$: flow entering $B$; $f^{out}(B)$: flow leaving $B$
  - = flow which stays in $B$
  - = flow which ends up in $t$ (no leak)
- $v(f) \leq c(A, B)$

## Max flow = Min cut

- $v(f_{max}) = \min_{(A,B)} \{c(A,B)\}$ — $(A_{min}, B_{min})$
  - $(A,B)$ ← ranges over all s-t cuts

$(A_{min}, B_{min})$ can be computed in $O(m)$ from $f_{max}$.

7.35

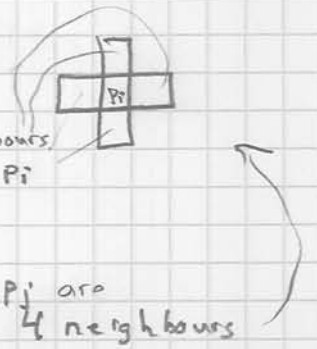



Image segmentation.

| | |
|---|---|
| $V$ | pixel set |
| $E$ | set of pairs of pixels : $(p_i,p_j),(p_j,p_i) \in E$ iff $p_i, p_j$ are neighbours in image. Each pixel has 4 neighbours |
| $n$ | pixels : $\lvert V\rvert = n$ $\lvert E\rvert = m$ |
| $a_i$ | likelihood weight of $p_i$ being in foreground . $a_i \in [0,d] \subseteq \mathbb{Z}$ |
| $b_i$ | " " " " background . $b_i \in [0,d]$ |
| $p_i$ | pixel $i$ ; $1 \le i \le n$ |

If $p_i$ in background, chances of all $p_j$ s.t. $(p_i,p_j) \in E$ being also in background increases.

| | |
|---|---|
| $p_{ij}$ | separation penalty for placing $p_i$ in foreground and $p_j$ in background (or vice versa). ($p_i,p_j$ neighbours) |

Segmentation problem: maximise

$$q(A,B) = \sum_{i \in A} a_i + \sum_{j \in B} b_j - \sum_{\substack{(i,j) \in E \\ |A \cap \{i,j\}| = 1}} p_{ij} \quad ; A,B \text{ disjoint}, A \cup B = V$$

( same as minimise

$$q'(A,B) = \sum_{i \in A} b_i + \sum_{j \in B} a_j + \text{---}''\text{---} \quad \text{---}''\text{---} \quad )$$

$(A_0, B_0)$ initial segmentation
$v_k$ mouse-clicked pixels $1 \le k \le t$
$(A_k, B_k)$ segmentation after mouseclick $k$.

Each $(A_k,B_k)$ maximises $q(A_k,B_k)$, provided pixel $v_k$ is in foreground.

Goal:

Algorithm which computes :
- $(A_0,B_0)$ in constant-factor time of time to compute max flow once,
- $(A_k,B_k)$ in $O(dn)$ time, for each $v_k$.

value of min-cut $(A,B) = f^{out}(A) - f^{in}(A)$