# Lange Pfade

Problem: Gibt es einen Pfad der Länge k in G?
→ NP-vollständiges Problem

## Reduktion auf Hamiltonkreise

**Satz 3.1.** Falls wir LONG-PATH für Graphen mit $n$ Knoten in $t(n)$ Zeit entscheiden können, dann können wir in $t(2n-2)+O(n^2)$ Zeit entscheiden, ob ein Graph mit $n$ Knoten einen Hamiltonkreis hat.


Beweis: Graphkonstruktion von G' mit $|V(G')| \leq 2n-2$ Knoten, sodass

G besitzt Hamiltonkreis $\Leftrightarrow$ G' hat einen Pfad der Länge n

"$\Rightarrow$": $\langle w_1, w_2, \dots, w_n, w_1 \rangle$ $\quad v = w_1$

o.B.d.A. $\langle \hat{w}_2, w_2, \dots, w_n, \hat{w}_n \rangle$ ein Pfad der Länge $n$

"$\Leftarrow$": $\langle u_0, u_1, \dots, u_{n-1}, u_n \rangle$ in G'

→ $u_1, \dots, u_{n-1}$ Grad $\geq 2$

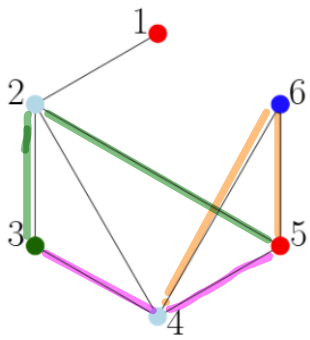
→ $u_0$ und $u_n$ sind neu hinzugefügte Knoten

→ $\langle v, u_1, \dots, u_{n-1}, v \rangle$ in G

# Bunte Pfade

Definiere Färbung $\gamma: V \to [k]$ für einen Graphen $G = (V,E)$

Pfad ist bunt $\Leftrightarrow$ alle Knoten des Pfades haben unterschiedliche Farben




Welche Länge hat der längste bunte Pfad?

$\rightarrow 3$

$P_3(6) = \{\{$grün, hellblau, rot, dunkelblau$\}\}$

$P_2(5) = \{\{$grün, hellblau, rot$\}, \{$hellblau, dunkelblau, rot$\}\}$

Gegeben: $G = (V,E)$, Färbung $\gamma: V \to [k]$
Ziel: Entscheide, ob es einen bunten Pfad der Länge $k-1$ gibt

$P_i(v) := \{S \subseteq [k] : |S| = i+1 \ \exists$ in $v$ endender, genau mit $S$ gefärbter bunter Pfad$\}$

$P_0(v) = \{\{\gamma(v)\}\}$

$P_1(v) = \{\{\gamma(x), \gamma(v)\} \mid x \in N(v), \gamma(x) \neq \gamma(v)\}$

$P_i(v) = \bigcup_{x \in N(v)} \{R \cup \{\gamma(v)\} \mid R \in P_{i-1}(x) \text{ und } \gamma(v) \notin R\}$

$\bigcup_{v \in V} P_{k-1}(v) \neq \emptyset \Leftrightarrow$ gibt es einen bunten Pfad der Länge $k-1$

$\rightarrow O(2^k km)$
$\rightarrow$ für $k \leq O(\log n)$ $O(n \log(n) m)$

Wir färben die Knoten von $G$ zufällig mit $k$ Farben.
Wie viele verschiedene Färbungen gibt es für einen Pfad der Länge $k-1$? $\rightarrow k^k$
Wie viele sind davon bunt? $\rightarrow k!$

Nehmen wir an es existiert ein Pfad $P$ der Länge $k-1$ in $G$.

$$\Pr[\exists \text{ ein bunter Pfad der Länge } k-1] \geq \Pr[P \text{ ist bunt}] = \frac{k!}{k^k} \geq e^{-k}$$

$$e^x = \sum_{i=0}^{\infty} \frac{x^i}{i!} \geq \frac{x^x}{x!}$$

Wie oft müssen wir $G$ im Erwartungswert färben, sodass $P$ bunt ist?

$X \sim Geo(e^{-k}) \rightarrow E[X] = \frac{1}{e^{-k}} = e^k$

# Monte-Carlo-Algorithmus

1. Färbe $G$ mit $k = B+1$ Farben in $O(n)$
2. Suche nach bunten Pfaden der Länge $B$ in $O(2^k km)$ und gebe „JA" aus, falls es einen gibt und ansonsten „NEIN".

$\Pr[\text{Erfolg}] \geq e^{-k}$

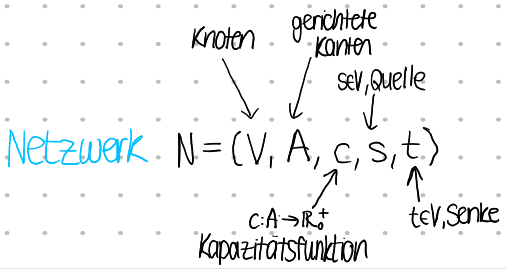
Wiederholen wir das ganze $\lambda e^k$ mal, was ist unsere Fehlerw'keit (w'keit, dass jeder Pfad der Länge $B$ bei allen Wiederholungen nicht bunt ist)?

$$\Pr[\text{Fehler}] \leq (1-e^{-k})^{\lambda e^k} \underset{\uparrow \; 1-x \leq e^{-x}}{\leq} (e^{-e^{-k}})^{\lambda e^k} = e^{-e^{-k} \cdot \lambda \cdot e^k} = e^{-\lambda}$$

Laufzeit: $O(\lambda e^k (2^k km + n)) = O(\lambda e^k 2^k km)$

$$\overset{k \leq O(\log n)}{\leq} O(\lambda n \cdot n \log(n) m) = O(\lambda n^2 \log(n) m)$$

# FLÜSSE




**Definition 3.5.** Gegeben sei ein Netzwerk $N = (V, A, c, s, t)$. Ein Fluss in $N$ ist eine Funktion $f : A \to \mathbb{R}$ mit den Bedingungen

$0 \le f(e) \le c(e)$ für alle $e \in A$, die *Zulässigkeit*, und

für alle $v \in V \setminus \{s, t\}$ gilt

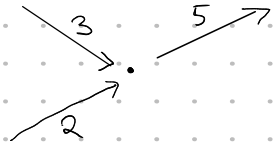
$$\sum_{u \in V:\,(u,v) \in A} f(u,v) = \sum_{u \in V:\,(v,u) \in A} f(v,u)$$

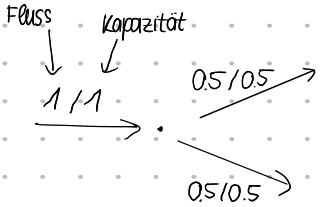
die *Flusserhaltung*.

Der *Wert* (engl.: value) eines Flusses $f$ ist durch

$$\mathrm{val}(f) := \mathrm{netoutflow}(s) := \sum_{u \in V:\,(s,u) \in A} f(s,u) - \sum_{u \in V:\,(u,s) \in A} f(u,s)$$

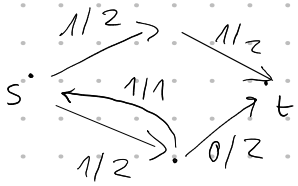
definiert. Wir nennen $f$ *ganzzahlig*, wenn $f(e) \in \mathbb{Z}\ \forall e \in A$.




**Lemma 3.6.** Der Nettozufluss der Senke gleicht dem Wert des Flusses, d.h.

$$\mathrm{netinflow}(t) := \sum_{u \in V:\,(u,t) \in A} f(u,t) - \sum_{u \in V:\,(t,u) \in A} f(t,u) = \mathrm{val}(f)\ .$$




Beweis:

$$0 = \sum_{(u,v) \in A} f(u,v) - \sum_{(v,u) \in A} f(v,u)$$

$$= \sum_{v \in V} \Big( \sum_{\substack{u \in V \\ (u,v) \in A}} f(u,v) - \sum_{\substack{u \in V \\ (v,u) \in A}} f(v,u) \Big)$$

$= 0$ für $v \in V \setminus \{s,t\}$

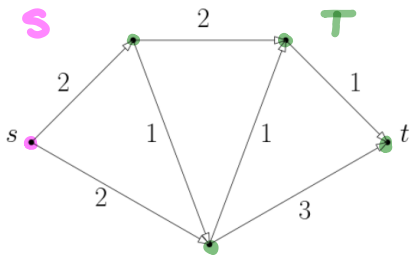
$$= \Big( \sum_{\substack{u \in V \\ (s,u) \in A}} f(s,u) - \sum_{\substack{u \in V \\ (u,s) \in A}} f(u,s) \Big) + \Big( \sum_{\substack{u \in V \\ (t,u) \in A}} f(t,u) - \sum_{\substack{u \in V \\ (u,t) \in A}} f(u,t) \Big)$$

$\mathrm{netoutflow}(s)$ $\quad$ $-\,\mathrm{netinflow}(t)$

## maximaler Fluss und Schnitte

s-t-Schnitt ist Partition von V in (S,T) ($S \cup T = V$ und $S \cap T = \emptyset$), wobei $s \in S$ und $t \in T$

$$\mathrm{cap}(S,T) := \sum_{(u,w) \in (S \times T) \cap A} c(u,w)$$



**Lemma:** Ist $f$ ein Fluss und $(S,T)$ ein $s$-$t$-Schnitt in einem Netzwerk $N$, so gilt $val(f) \leq cap(S,T)$

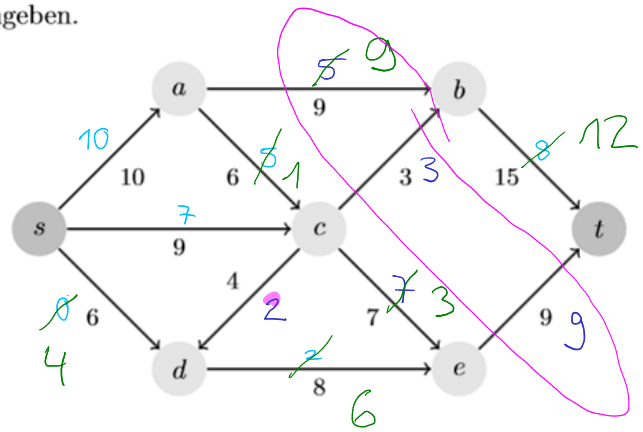
**Satz 3.9 („Maxflow-Mincut Theorem").** Jedes Netzwerk $N = (V, A, c, s, t)$ erfüllt

$$\max_{f \text{ Fluss in } N} val(f) = \min_{(S,T) \text{ s-t-Schnitt in } N} cap(S,T)$$

Wie finden wir nun einen Maxflow?

**Definition Restnetzwerk:** (1) Falls $e \in A$ mit $f(e) < c(e)$, dann $e \in A_f$ mit $r_f(e) := c(e) - f(e)$
(2) Falls $e \in A$ mit $f(e) > 0$, dann $e^{opp} \in A_f$ mit $r_f(e^{opp}) = f(e)$

Die folgende Abbildung zeigt ein Netzwerk mit Quelle $s$ und Senke $t$, wobei die Zahlen die Kapazitäten der Kanten angeben.



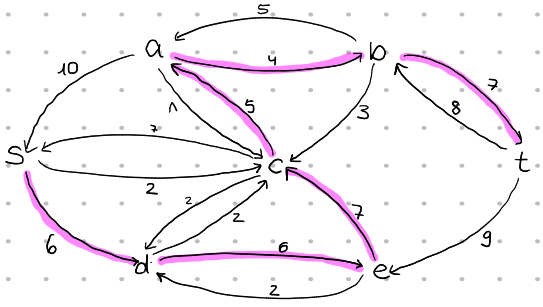

Auf einigen Netzwerkkanten ist eine nichtnegative Funktion $f$ gegeben durch

| $(x,y)$ | $(s,a)$ | $(s,c)$ | $(s,d)$ | $(a,c)$ | $(d,e)$ | $(b,t)$ |
|---|---|---|---|---|---|---|
| $f(x,y)$ | 10 | 7 | 0 | 5 | 2 | 8 |

(a) Wie ist $f$ auf alle übrigen Netzwerkkanten fortzusetzen, so dass $f$ ein Fluss ist? Was ist der Wert dieses Flusses?

(b) Zeichnen Sie das Residualnetzwerk.

(c) Finden Sie einen augmentierenden Pfad von $s$ nach $t$ und erhöhen Sie den Fluss entlang dieses Pfades um den maximal möglichen Betrag. Falls nötig, dann iterieren Sie diesen Schritt, bis der so gefundene Fluss maximal ist.



**Satz 3.11.** Ein Fluss $f$ in einem Netzwerk $N$ ist ein maximaler Fluss gdw. es im Restnetzwerk $N_f$ keinen gerichteten Pfad von der Quelle $s$ zur Senke $t$ gibt. Für jeden solchen maximalen Fluss gibt es einen $s$-$t$-Schnitt $(S, T)$ mit $\mathrm{val}(f) = \mathrm{cap}(S, T)$.

---

FORD-FULKERSON$(V, A, c, s, t)$

---

1: $f \leftarrow \mathbf{0}$ ▷ Fluss konstant 0
2: **while** $\exists$ $s$-$t$-Pfad $P$ in $(V, A_f)$ **do** ▷ augmentierender Pfad
3: Erhöhe den Fluss entlang $P$ ▷ wie in Beweis zu Satz 3.11
4: **return** $f$ ▷ maximaler Fluss

---

**Satz 3.12.** Sind in einem Netzwerk ohne entgegen gerichtete Kanten alle Kapazitäten ganzzahlig und höchstens $U$, so gibt es einen ganzzahligen maximalen Fluss, der in Zeit $O(mnU)$ berechnet werden kann ($m$ ist die Anzahl Kanten, $n$ die Anzahl Knoten im Netzwerk).

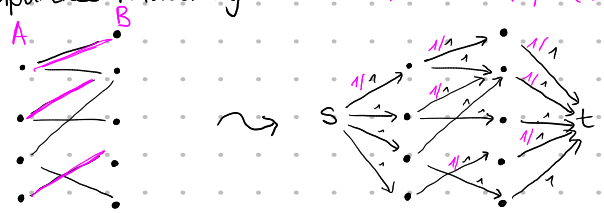
Capacity Scaling: Kapazitäten ganzzahlig und höchstens $U$ $O(mn(1+\log n))$

Dynamic Trees: $O(mn\log n)$

# ANWENDUNGEN VON FLÜSSEN

## ① Bipartites Matching

M  f, val(f) = |M|



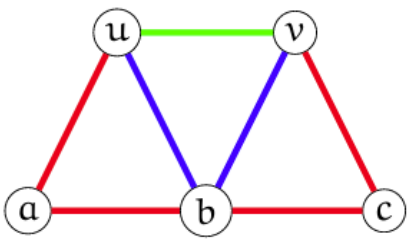

**Lemma 3.15.** Die maximale Grösse eines Matchings im bipartiten Graph G ist gleich dem Wert eines maximalen Flusses im Netzwerk N.

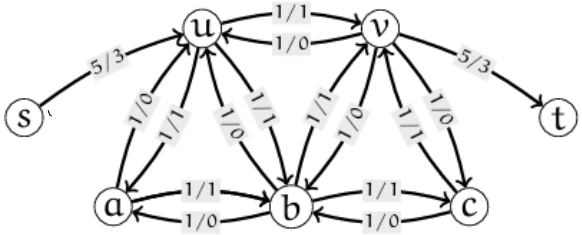
## ② Kanten- und knotendisjunkte Pfade

Wie viele intern kanten- (oder knoten-) disjunkte u-v-Pfade gibt es?

Netzwerkkonstruktion: für alle $\{u,v\} \in E$ füge (u,v) und (v,u) mit jeweils Kapazität 1 zu A hinzu
$(s,u), (v,t) \in A$, wobei Kapazität dieser Kanten $= |V|$

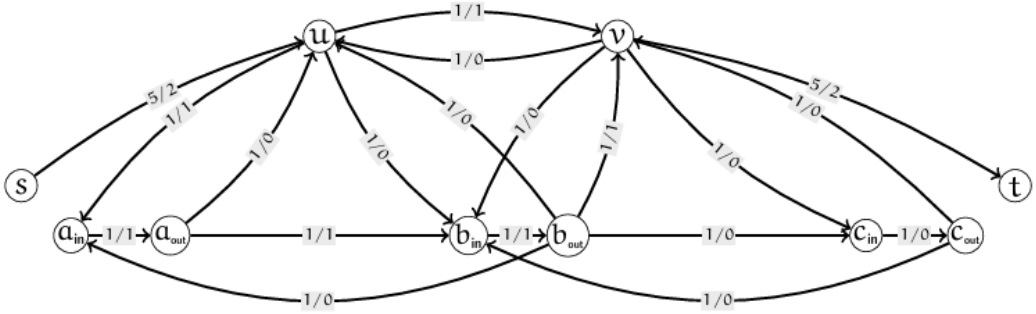
u  v




#kantendisjunkter Pfade = val(f)

für knotendisjunkte Pfade: $\forall x \in V \setminus \{u,v\}$ zwei neue Knoten $x_{in}$ und $x_{out}$
alle Eingangsknoten zu $x_{in}$, Ausgangskanten aus $x_{out}$
$(x_{in}, x_{out}) \in A$ mit $c(x_{in}, x_{out}) = 1$
→ wir können x nur einmal nutzen



# ③ Bildsegmentierung

Ziel: Unterteilung eines Bildes in Vordergrund (A) und Hintergrund (B)

Qualitätsfunktion $q(A,B) = \sum_{p\in A} \alpha_p + \sum_{p\in B} \beta_p - \sum_{\substack{e\in E \\ |e\cap A|=1}} \gamma_e$

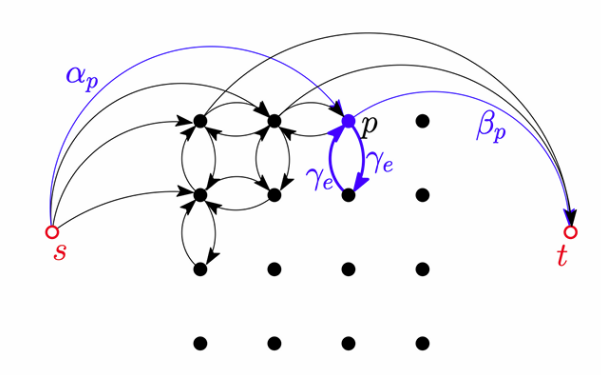
(Vordergrund, Hintergrund, gleicher Teil)

wenn wir eine Partition finden wollen, die $q(A,B)$ maximiert ist das equivalent zur Minimierung von

$$q'(A,B) = \sum_{p\in A} \beta_p + \sum_{p\in B} \alpha_p + \sum_{\substack{e\in E \\ |e\cap A|=1}} \gamma_e = cap(S,T)$$

$A = S\setminus\{s\}$
$B = T\setminus\{t\}$

Netzwerkkonstruktion:



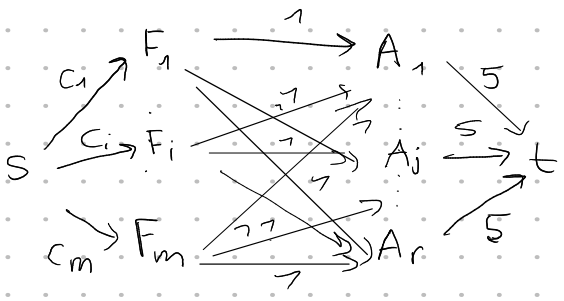

$(s,p)$, $p\in B$, $\alpha_p$
$(p,t)$, $p\in A$, $\beta_p$
$(p,p')$, $p\in A, p'\in B$, $\gamma_{(p,p')}$

## Aufgabe

Zu einem Kongress werden Mitarbeiter von $m$ Firmen gesandt. Dabei sendet die $i$-te Firma $c_i$ viele Mitarbeiter. Während des Kongresses sollen die anwesenden Personen in bis zu $r$ verschiedenen Arbeitsgruppen mit jeweils höchstens 5 Mitgliedern aufgeteilt werden, wobei keine zwei Mitarbeiter derselben Firma in derselben Arbeitsgruppe sein sollen.

Wir sollen eine solche Aufteilung mit Hilfe von Fluss-Algorithmen finden.

(a) Definieren Sie dazu ein geeignetes Netzwerk $N = (V, A, c, s, t)$ und zeigen Sie, dass es gibt eine mögliche Aufteilung wie oben genau dann wenn maxflow$(N)$ hat eine gewisse (welche?) Eigenschaft.




$\text{maxflow}(N) = \sum_{i=1}^{m} c_i$

$V = \{s,t\} \cup A \cup F$

$A = \{(s,f) \mid f\in F\}$

# Surprise food!

A friend wants to open an online food company called 'Surprise food!'. The idea is that clients do not decide for a single dish, but instead select a list of all dishes they like from the menu. Then the company delivers *one* dish out of the list. However, planning is not one of your friend's top skills and he has just prepared some meals without actually checking the request list. That is, he has prepared $d_i$ meals of the $i$-th dish, hoping that this will be enough.

The grand opening is in two hours and your friend wants to know if *it is possible to distribute the prepared meals in such a way that every client gets a meal from her list*. The goal is to decide whether such a distribution of meals is possible.

## Input

The first line of the input file contains an integer $1 \leq t \leq 30$ denoting the number of test cases that follow. Each of the $t$ test cases is described as follows.

- It starts with a line consisting of two integers $\mathtt{n}$ $\mathtt{m}$, separated by a space, denoting the number of different dishes in the menu ($1 \leq n \leq 100$) and the number of clients ($1 \leq m \leq 100$).
- The next line contains $n$ integers $\mathtt{d}_0 \ \ldots \ \mathtt{d}_{n-1}$, separated by a space, and such that $0 \leq d_i \leq 10^3$. Each $d_i$ denotes the number of prepared meals of dish $i$.
- The following $m$ lines define the lists of dishes of the $m$ clients. The $\ell$-th line describes the list of the $\ell$-th client. It starts with an integer $\mathtt{k}$ which specifies the length of this list, followed by $k$ different integers $\mathtt{f}_0 \ \ldots \ \mathtt{f}_{k-1}$, separated by a space, and such that $0 \leq f_j < n$, for all $j \in \{0, \ldots, k-1\}$. Each $f_j$ denotes a dish on the list of this client.

## Output

For each test case output one line containing `yes` if it is possible to distribute the meals such that every client gets a meal from her list and `no` otherwise. You can serve at most $d_i$ meals from the $i$-th dish.

## Points

This exercise is worth $100$ points.