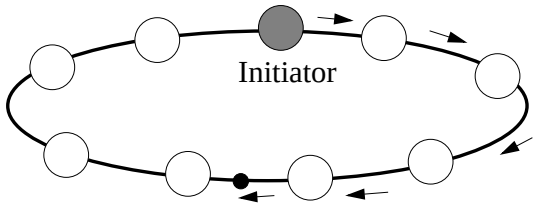
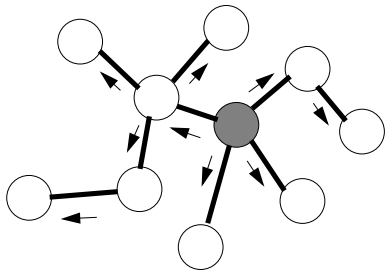
Broadcast auf speziellen Topologien

- Echo-Algorithmus realisiert einen Broadcast ← auf bel. zusammenh. Topologie
  - Verteilen von Information ausgehend von einem Initiator
  - für beliebige (zusammenhängende) Topologien
  - liefert sogar "Vollzugsmeldung" durch Echo-Nachrichten

- Auf *speziellen* Topologien lässt sich der Broadcast auch effizienter realisieren
  - Beispiel *Ring*: Ein "Token" zirkuliert mit der Information; alle sind informiert, wenn das Token wieder beim Initiator eingetroffen ist
  - ggf. kann einer anderen Topologie ein Ring überlagert werden



  - Beispiel *(Spann)baum* (tatsächlich Unterschied zum Echo-Algorithmus?)



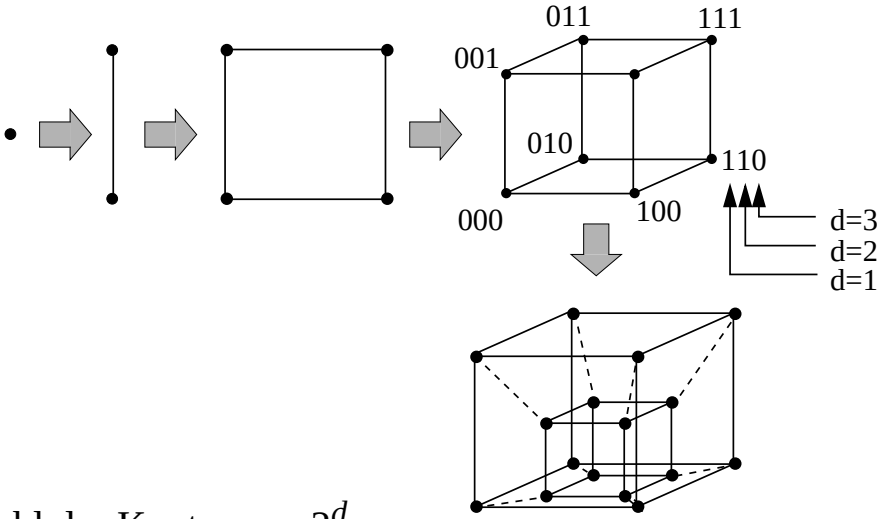
vorausgesetzt wird jeweils, dass der Algorithmus "weiss", dass eine spezifische Topologie vorliegt!

  - Beispiel *vollständiger Graph* (als Denkübung)

Hypercubes

- Hypercube = "Würfel der Dimension d"
- Rekursives Konstruktionsprinzip
  - Hypercube der Dimension 0: Einzelrechner
  - Hypercube der Dimension d+1:

„Nimm zwei Würfel der Dimension d und verbinde korrespondierende Ecken“



- Anzahl der Knoten $n = 2^d$
- Anzahl der Kanten = $d\ 2^{d-1}$ (Ordnung $O(n \log n)$)
  - viele Wegalternativen (Fehlertoleranz, Parallelität!)
  - maximale Weglänge: d = log n
  - mittlere Weglänge: d/2 (Beweis als Denkübung!)

wieviele verschiedene Wege der Länge k gibt es insgesamt?

- Knotengrad = d (nicht konstant bei Skalierung!)
- Einfaches Routing von einzelnen Nachrichten
  - xor von Absende- und Zieladresse...

Kombinatorische Aspekte (1)

- Wieviele verschiedene Wege der Länge k (ausgehend von Knoten 0) gibt es?
  - man wiederhole die Begriffe und Methoden der Kombinatorik...
  - für den Anfang: was ergibt sich für k=1, k=n?

- Ein ähnliches Problem: Wege in Manhatten...

Brian Hayes, American Scientist, January-February 1996

A Walk in Manhattan (Auszug)

When I was a commuter in New York, I walked a diagonal route across midtown Manhattan morning and evening. I used to wonder how many different paths I could find through the grid of east-west and north-south streets without taking extra steps. Eventually I grew curious enough to calculate some solutions. For a square lattice of n-by-n blocks, any minimum-length diagonal path is necessarily 2n blocks long. How many such paths are there? For the 1-by-1 lattice the answer is 2: You can go east and then north or you can go north and then east. In a two-block square there are six paths, and in a 3-by-3 block there are 20. The sequence of values I calculated begins like this: 2, 6, 20, 70, 252, 924, 3432, 12870, 48620....

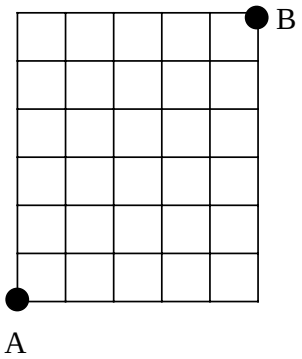
Do those numbers look familiar? They should. They are prominent members of one of the most ancient and famous families of numerical sequences. And yet I did not identify them until several years later, when I had a chance to submit them to Sloane's sequence server. The answer came back immediately: They were recognized as sequence M1645, the central binomial coefficients, the numbers that run down the middle of Pascal's triangle. My reaction was "of course! Why didn't I see it all along?" But I doubt that I would have made the connection without help.

Kombinatorische Aspekte (2)

The discovery was a productive one, which led not just to an answer but to an insight. I saw that counting the shortest diagonal paths for all rectangular lattices--not just the square ones--would fill in the rest of Pascal's triangle. Each row of the triangle consists of all the lattices with a given minimum diagonal path length. For example, the fifth row includes all the lattices with a path length of 4, namely the 0-by-4, 1-by-3, 2-by-2, 3-by-1 and 4-by-0 lattices. The corresponding counts of diagonal paths (and the corresponding entries in Pascal's triangle) are 1, 4, 6, 4 and 1.

http://www.sigmaxi.org/amsci/issues/comsci96/compsci96-01.html

---

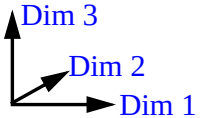


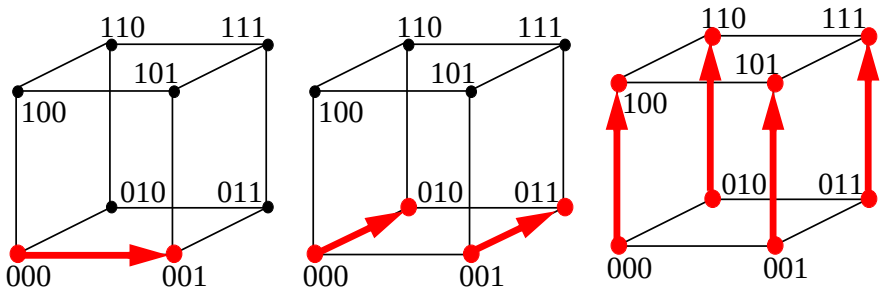
- Jeder kürzeste Weg von A nach B hat die Länge 11
- Repräsentation durch ein Wort der Länge 11
- 5 Nullen und 6 Einsen
- 0: gehe nach Osten
- 1: gehe nach Norden
- Wieviele solche Worte gibt es?

- Gegeben sei eine 11-elementige Menge
- Dem i-ten Element sei die Position i (in einem Wort) zugeordnet
- Anzahl der 6-elementigen Teilmengen = Anzahl der Worte der Länge 11 mit genau 5 Nullen und 6 Einsen
- ==> Binomialkoeffizienten

Broadcast in Hypercubes (1)

- Initiator habe die Nummer 00...00 (binär)
- Wir verzichten hier auf Vollzugsmeldung
  (also keine Acknowledgemnents oder Endeerkennung)





- Analog zum rekursiven Aufbau des Hypercube:
  - zunächst in Dimension 1 senden: Teil-Hypercube der Dimension 1 ist damit informiert
  - dann senden alle Knoten der Dimension 1 in Dimension 2
  - dann Dimension 3 etc.
- Nach d "Takten" sind alle Knoten informiert
  - Zeitkomplexität ist daher d (unter welchem Zeitmass?)
  - Nachrichtenkomplexität: $1 + 2 + 4 + ... + 2^{d-1} = 2^{d-1} - 1 + 2^{d-1} = n-1$
    (jeder Knoten, ausgenommen der Initiator, erhält genau eine Nachricht)
- Welche Komplexität hat ein *optimaler* Broadcast-Algo.?
- Geht es besser? ← was heisst überhaupt "besser"?
  - Algorithmus startet ziemlich "langsam": am Anfang geschieht wenig parallel!
  - Kann man dies durch *gleichzeitiges* Versenden "in alle Richtungen" beschleunigen?

Broadcast in Hypercubes (2)

- Ein anderes Verfahren (Vergleich als Denkübung!)
- Initiator sendet an alle seine Nachbarn: (am besten gleichzeitig, wenn dies technisch geht!)

  0...01, 0...010, 0...100, ... , 10...0

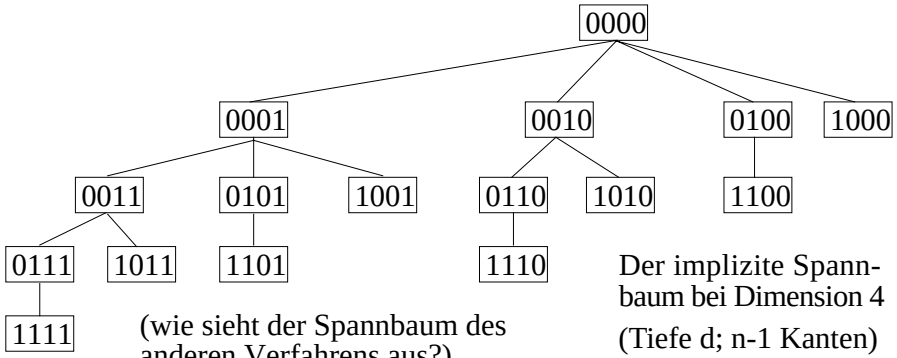
  in "kanonischer" Numerierung

- Ein Knoten mit der Nummer 0...01x...y...z leitet die Information an alle seine "höheren" Nachbarn weiter:

  (linkeste 1, beliebiges Restmuster)

  0...00**1**1x...y...z
  0...0**1**01x...y...z
  0...**1**001x...y...z
  ...
  10...001x...y...z

  Von welchem (eindeutigen) Knoten A wird Knoten B informiert?
  Setze *vorderste 1* von B auf 0
  --> = Nummer von A



(wie sieht der Spannbaum des anderen Verfahrens aus?)

Der implizite Spannbaum bei Dimension 4
(Tiefe d; n-1 Kanten)

- Der Algorithmus wird z.B. in Mehrprozessorsystemen verwendet
- Wie effizient ist der Algorithmus? (Geht es besser?)
- Denkübung: Formuliere Algorithmus für einen beliebigen Initiator (schliesslich sind Hypercubes symmetrisch...)
- Denkübung: Vergleich mit Flooding bzw. Echo-Algorithmus

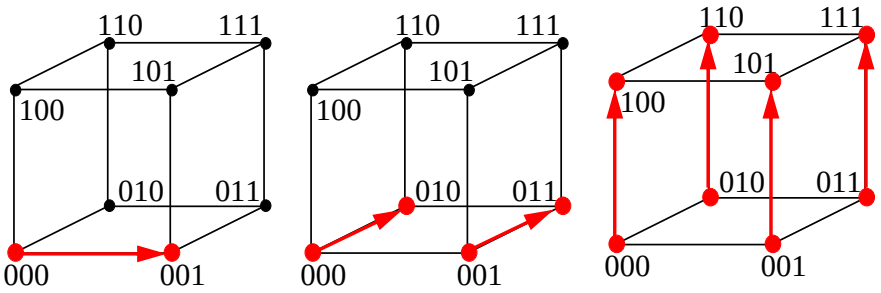
Noch ein anderer (besserer?) Algorithmus

- Beobachtungen:
  - Ein Baum verwendet im Hypercube relativ wenig Kanten --> schlechte Ausnutzung potentielle Parallelität
  - Es gibt *mehrere* Spannbäume in Hypercubes --> diese nutzen?

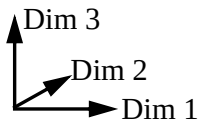
- Sender 0...0 teilt die Nachricht in d Pakete
- Sender startet für jedes Paket eine eigene “Welle”:
- *1. Paket* in *Dimension 1* senden --> 0...01
- Dann: Alle informierten Knoten (also 0...0 und 0...01) senden dieses Paket in Dimension 2
- Etc. Welle für Paket 1 breitet sich analog zur rekursiven Definition des Hypercubes in einer jeweils zusätzlichen Dimension aus
- Das *2. Paket* wird erst in *Dimension 2*, dann 3,..., d und erst zuletzt in Dimension 1 gesendet
- Das *3. Paket*: Dimensionsreihenfolge 3, 4, ..., d,1, 2
- Etc.: das *d.-Paket* in Dimensionsreihenfolge d, 1, 2,..., d-1

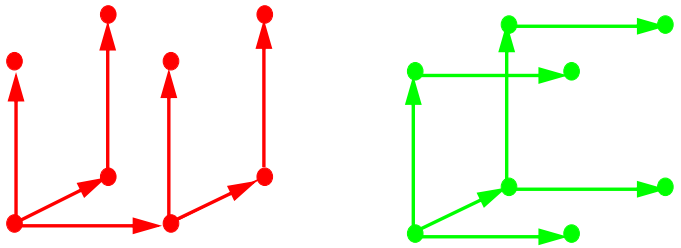
Denkübungen:
- Können so die Pakete gleichzeitig verschickt werden?
- Ist dann in jedem “Takt” pro Kante nur eine Nachricht unterwegs?
- Wieviele (kantendisjunkte ?) Spannbäume gibt es in einem Hypercube?

Veranschaulichung des Algorithmus



Die drei “Takte” der Welle von Paket 1

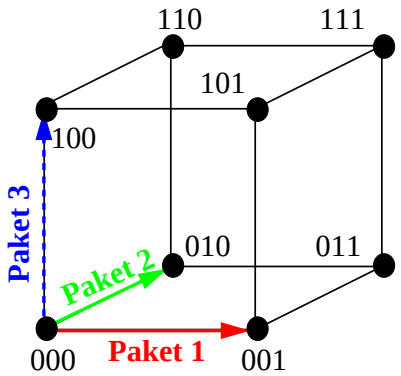




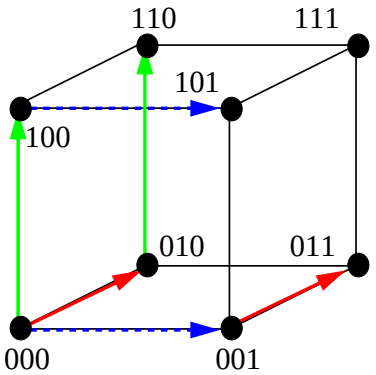
Die Spannbäume bzgl. Paket 1 und Paket 2

Parallelausführung der drei Wellen

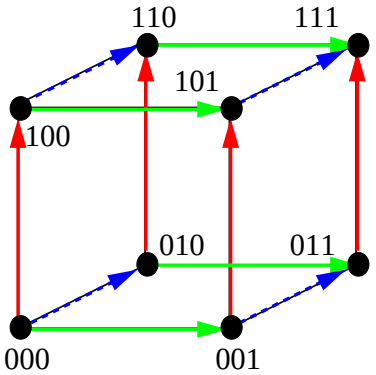
Takt 1



Takt 2



Takt 3



	Takt 1	2	3	
Paket 1:	1.	2.	3.	Dim.
Paket 2:	2.	3.	1.	Dim.
Paket 3:	3.	1.	2.	Dim.

Pro Takt laufen die Pakete in jeweils unterschiedlicher Dimension!

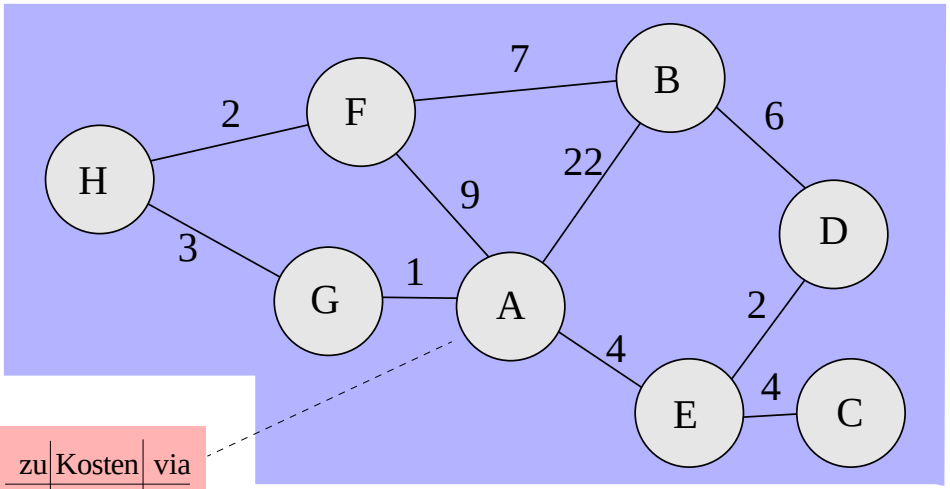
Es können also tatsächlich die drei Wellen parallel ausgeführt werden, ohne dass diese sich gegenseitig stören!

--> Dies ist das (im Prinzip) schnellere Verfahren!

Beachte: Ein globaler Takt ist gar nicht nötig!

Verteilte Berechnung von Routingtabellen für kürzeste Wege

Gegeben: ungerichteter zusammenhängender Graph mit bewerteten Kanten (Kosten, Länge...)



zu	Kosten	via
A	0	-
B	22	B
C	∞	?
D	∞	?
E	4	E
F	9	F
G	1	G
H	∞	?

Anfangstabelle für Knoten A

- Jeder kennt anfangs die Kosten zu seinen Nachbarn
- "Spontanstart": Sende eigene Tabelle an Nachbarn
- Bei Empfang einer Tabelle über Verbindung mit Kosten g:
  Für alle Zeilen i der Tabelle:
  Falls Nachricht.Kosten[i]+g < Knoten.Kosten[i]:
  ersetze Zeile (Kosten := Kosten+g; via := Absender)
- Falls sich Tabelle verändert hat:
  Neue Tabelle an alle Nachbarn (Ausnahme: Sender)
- Wie Terminierung feststellen?

- Ist eine verteilte Version des Bellman-Ford-Algorithmus
  - ähnlich dem bekannten Dijkstra-Algorithmus für kürzeste Wege
  - " Relexationsprinzip" (Bellman 1958, Dijkstra 1959, Ford 1962)

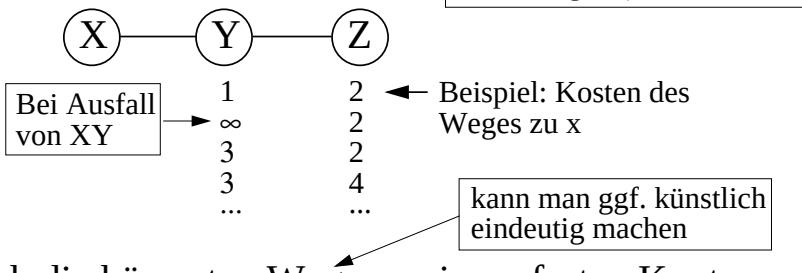
Kürzeste Wege in Rechnernetzen

- Algorithmus wird oft als dynamisches (“adaptives”) Routing-Verfahren verwendet, wo in regelmässigen Zeitabständen die Tabellen neu berechnet und ausgetauscht werden
  - bzw. dann, wenn sich etwas ändert (Kosten einer Verbindung, z.B. Ausfall einer Leitung oder Änderung der Lastsituation)

- Metrik für die Kosten z.B.:
  - (gewichtete)Anzahl der hops
  - Bitrate einer Verbindung
  - Verzögerung einer Verbindung (z.B. gemessen mit Testpaketen)
  - Länge der Warteschlange vor einer Verbindung

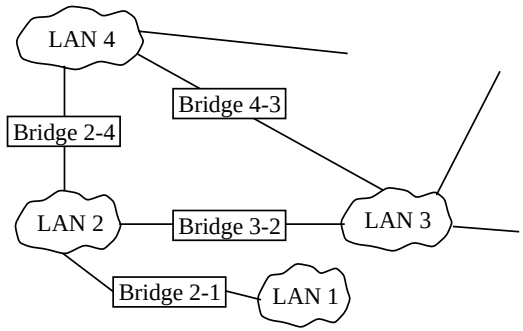
- "Count to infinity-Problem"

mehr zu diesen Dingen in anderen Vorlesungen ("Rechnernetze")



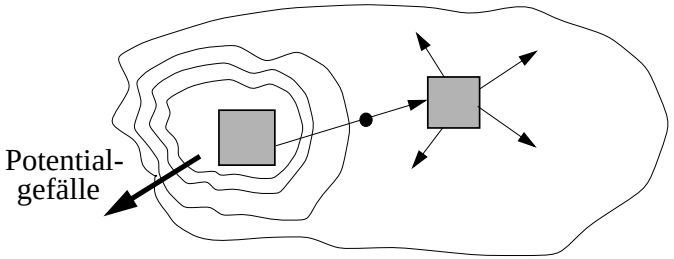
- Durch die kürzesten Wege zu einem festen Knoten ("Wurzel") ist ein kostenminimaler Baum gegeben

  - Algorithmus wird in LANs eingesetzt, um einen Spannbaum zu bestimmen (Knoten = Teil-LAN; Kante = Bridge)
  - Zyklenfreiheit ist wichtig, da kein Routing in LANs
  - Ende wird heuristisch durch Abwarten einer Zeitspanne festgestellt



Das Paradigma der vert. Approximation

Prinzip:
- Anfang: Informiere alle Nachbarn spontan
- Bei Empfang einer Nachricht:
  - berechne neue Approximation
  - falls diese "besser": informiere Nachbarn



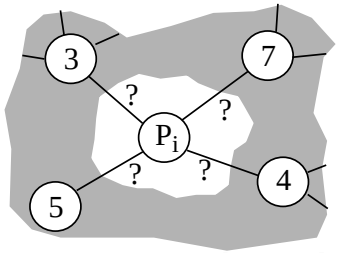
- *Nachrichtengesteuert* (aber "Spontanstart")
- Alle Prozesse arbeiten gleich, alle sind beteiligt
- *Nichtdeterministischer* Ablauf, determin. Ergebnis
- Beliebige stark zusammenhängende Topologie
- Assoziative Operatoren (min, max, $\cap$, $\cup$, +, and, or, ...)
- *Stagnation* bei globalem Gleichgewicht ("Optimum")
  --> Potentialunterschiede ausgeglichen
  --> Terminierungsproblem

Beispiele ("Instanzen der Algorithmenklasse"):
- ggT
- Zahlenrätsel
- Verteilen von Information ("Wissensausgleich")
- Routingmatrizen (inkl. Spannbaum)
- Maximale Identität ("election") } (noch nicht behandelt)
- Lastausgleich (Approx. eines dyn. Optimums) } (noch nicht behandelt)
- Relaxationsverfahren (Lösen von DGL) } (noch nicht behandelt)

Nachbarschaftswissen

- Zweck: Lerne Identitäten Deiner Nachbarn kennen
  (Idee: "Ich heisse i, wie heisst Du?")

- Hier nachrichtengesteuert für P_i:
  (vgl. verteiltes Approximationsschema)



Netz zusammenhängend
mit bidirektionalen Kanten

```
{Nachricht <j> kommt an}
if not vorgestellt then
    send <i> to all neighbors;
fi;
Nachbarn := Nachbarn ∪ {j};
vorgestellt := true;
```

mehrere?

Einer muss "spontan" mit
der Vorstellung beginnen

- *global terminiert*, wenn
  - vorgstellt = true bei allen Prozessen und keine Nachricht unterwegs
  - *oder*: alle kennen alle

- *lokal terminiert* (genügt meistens!), wenn zu jeder inzidenten Kante die zugehörigen Nachbarknoten bekannt sind

- 2e Nachrichten (bei Ring also 2n); Zeitkomplexität?

Verwendung als:

- vorgeschalteter eigener Algorithmus
- "piggybacking" der Vorstellungsnachrichten mit eigentlichen (jew. "ersten") Anwendungsnachrichten