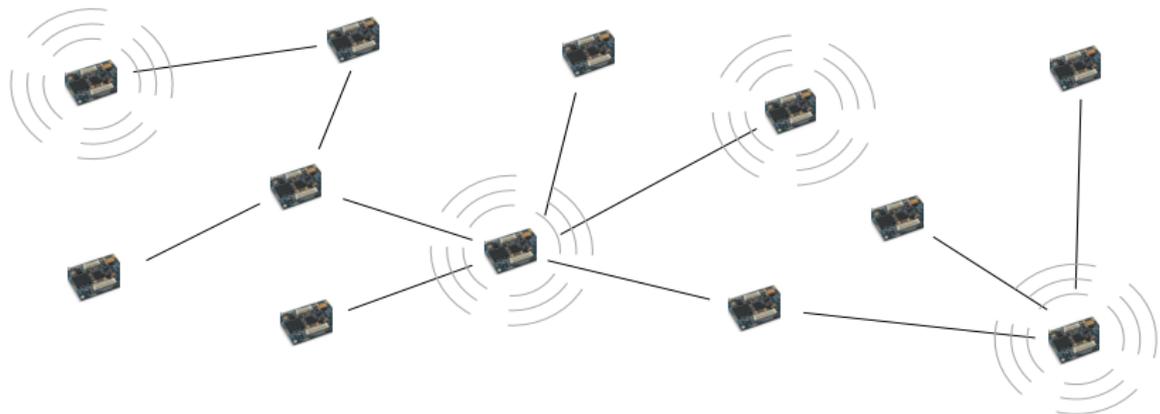


Algorithmen für Ad-hoc- und Sensornetze

VL 01 – Einführung und erste Schritte

Markus Völker | Version 1 vom 18. April 2012

INSTITUT FÜR THEORETISCHE INFORMATIK - LEHRSTUHL FÜR ALGORITHMIK (PROF. WAGNER)



- Organisatorisches
- Kurze Einführung zu Sensornetzen
 - Sensorknoten
 - Anwendungen
 - Herausforderungen
- Vorlesungsübersicht
- Erste Schritte
 - Definitionen
 - Broadcasts und Flooding
 - Leader Election
 - Routing: Link Reversal

- **Vorlesungstermine**
 - vom 18.04.2012 bis 18.07.2012 (14 Termine)
 - immer mittwochs von 14:00 Uhr bis 15:30 Uhr im SR 301 (2 SWS)
 - zusätzlich zum Stoff der Vorlesung werden vereinzelt Papers zur Lektüre zuhause empfohlen (1 SWS)
- **Webseite zur Vorlesung:**

<http://illwww.iti.uni-karlsruhe.de/teaching/sommer2012/sensornetze/index>

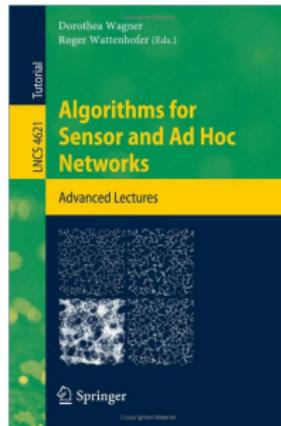
 - Folien (am Tag der Vorlesung vormittags)
 - korrigierte Foliensätze, Druckversionen
 - Referenzen und weiterführende Literatur
 - Ankündigungen zur Vorlesung
- **Sprechstunde**
 - jederzeit nach Vereinbarung, markus.voelker@kit.edu
 - Raum 306, Gebäude 50.34

- Prüfungsgebiete
 - Hauptdiplom/Master Informatik: 3 SWS, 5 LP
 - 01 Theoretische Grundlagen
 - 02 Algorithmentechnik
 - Diplom/Master Informationswirtschaft: 3 SWS, 5 LP
 - Modul „Advanced Algorithms“
 - Graduiertenkolleg GRK1194

- Prüfungsstoff
 - Fragestellungen, Motivationen
 - Algorithmische Ideen und Beweisskizzen
 - nur Stoff der Vorlesung ist prüfungsrelevant

- Grundbegriffe der Graphentheorie
 - Graph, Knoten, Kante, Grad, inzident, adjazent, Nachbarschaft, Baum, Wald, Kreis, planar, bipartit, Clique
- Algorithmen und Algorithmenanalyse
 - Asymptotische Laufzeit, $O(n)$, $o(n)$, $\Omega(n)$, $\Theta(n)$
 - **P**, **NP**, **NP**-schwer, **NP**-vollständig
 - Approximationsalgorithmen, Randomisierung
- Wenn etwas unklar ist, direkt nachfragen!
- Wenn ich zu schnell bin, bitte auch gleich Bescheid geben!

- Kurzschriften *Grundlagen*
 - *Begriffe der Graphentheorie*
 - *Algorithmen und ihre Laufzeit*
 - *Die Komplexitätsklassen P, NP und NPC*
 - <http://illwww.ira.uka.de/information/scripts>
- D. Wagner, R. Wattenhofer (Hrsg.):
Algorithms for Sensor and Ad Hoc Networks
 - Im Uninetz kostenlos verfügbar:
<http://www.springerlink.com/content/j17361060k67/>
- Literaturangaben zu einzelnen Themen in der Vorlesung (Webseite, Folien)



Algorithmen für Routenplanung

- Dozenten: Dr. Daniel Delling und Thomas Pajor
- Vorlesung: montags 14:00–15:30 Uhr (SR 301)
- Vorlesung: mittwochs 11:30–13:00 Uhr (SR 301, nicht wöchentl.)

Algorithmische Geometrie

- Dozenten: Dr. Martin Nöllenburg und Andreas Gemsa
- Vorlesung: dienstags 9:45–11:15 (SR 301)
- Übung: donnerstags 10:15–11:00 (SR 131)

Algorithmen für planare Graphen

- Dozent: Dr. Ignaz Rutter
- Vorlesung: mittwochs 15:45–17:15 Uhr (SR 301)
- Vorlesung: donnerstags 14:00–15:30 Uhr (SR 301)

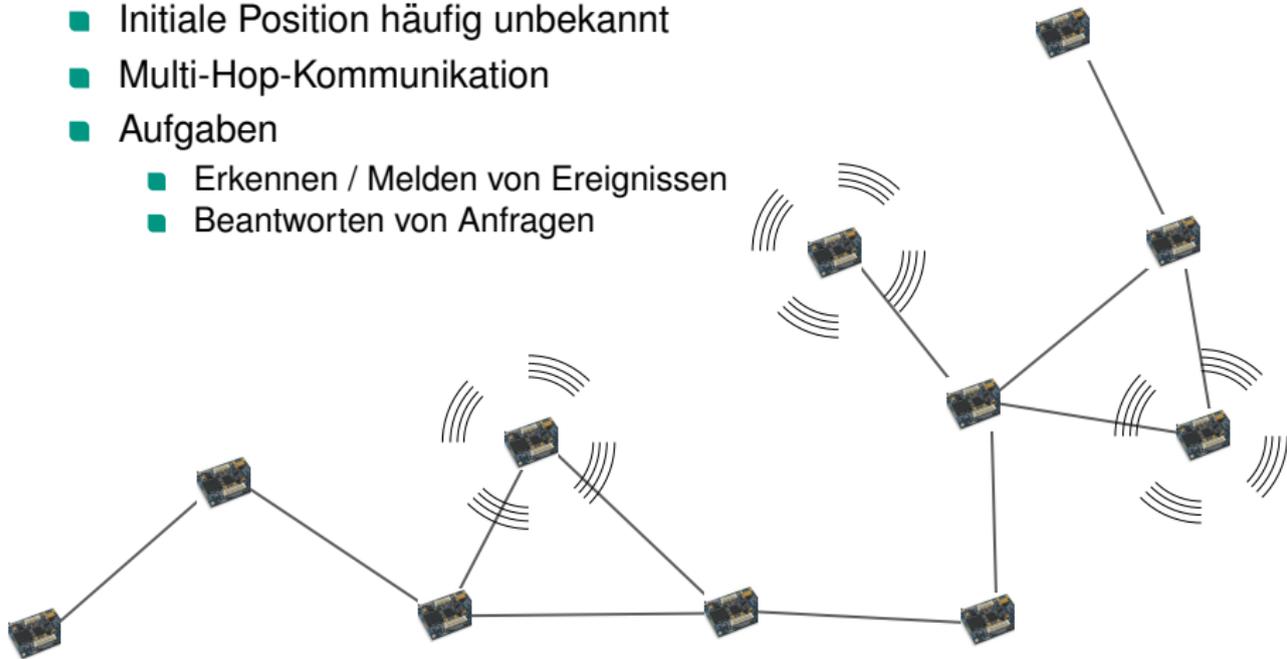
- Organisatorisches
- Kurze Einführung zu Sensornetzen
 - Sensorknoten
 - Anwendungen
 - Herausforderungen
- Vorlesungsübersicht
- Erste Schritte
 - Definitionen
 - Broadcasts und Flooding
 - Leader Election
 - Routing: Link Reversal

Sensorknoten (aka *motes*)

- Übliches Designziel: klein und billig
- Mikroprozessor
 - wenige MHz bis einige hundert MHz
 - energiesparende Zustände
- Speicher
 - häufig nur wenige KB Arbeitsspeicher
 - teils zusätzlich Flashspeicher für Code und Daten
- Sensorik
 - Temperatur, Licht, Druck, Beschleunigung, Feuchtigkeit uvm.
- Drahtlose Kommunikation
 - Reichweiten von wenigen Metern bis einige hundert Meter
 - Langsame Entwicklung von Standards (z. B. auf Basis von IEEE 802.15.4 wie ZigBee)
- Energieversorgung
 - heute: Batterien
 - in Zukunft: Solarzellen und anderes „Energy Harvesting“



- Keine zentrale Infrastruktur
- Initiale Position häufig unbekannt
- Multi-Hop-Kommunikation
- Aufgaben
 - Erkennen / Melden von Ereignissen
 - Beantworten von Anfragen



- Umweltüberwachung
 - Waldbrände, Gletscherbewegungen
 - Bewässerung von Agrarflächen
 - Schadstoffe (chemisch, radioaktiv)
- Beobachtung von Tieren
 - Überwachung von Brutstätten, Bewegungen
 - „Intelligente Zäune“
- Medizinische Überwachung
 - Patienten im Krankenhaus
- Strukturelle Überwachung
 - Strukturelle Integrität von Gebäuden
- Sicherheit
 - Grenzüberwachung
 - Überwachung von Gebäuden
- Verkehrsoptimierung
 - Stauwarnungen

- Geringes Energiebudget
 - geringe Leistung \leftrightarrow jahrelange Bereitschaft
 - Lastbalancierung, Aufteilung von Aufgaben
- Geringer Speicher, geringe Rechenleistung
 - verteilte Lösungen \leftrightarrow komplexe Selbstorganisation
- Viele Sensorknoten
 - Algorithmen müssen gut mit Größe des Netzes skalieren
- Geometrie
 - enger Zusammenhang zwischen Positionen und Daten/Aufgaben
 - gemeinsame Nutzung und Störung der Funkkanäle
- Redundanz, Unzuverlässigkeit und Dynamik
 - Knoten können ausfallen, sich bewegen
 - viele Knoten können Aufgaben redundant ausführen

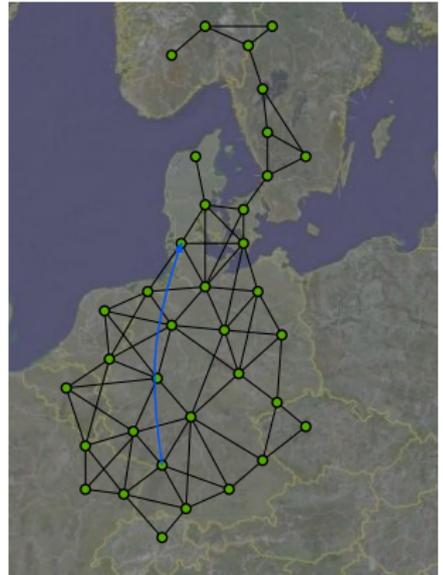
- Organisatorisches
- Kurze Einführung zu Sensornetzen
 - Sensorknoten
 - Anwendungen
 - Herausforderungen
- **Vorlesungsübersicht**
- Erste Schritte
 - Definitionen
 - Broadcasts und Flooding
 - Leader Election
 - Routing: Link Reversal



- Jede VL ein Problem in WSN
 - viele Themen algorithmisch isoliert
 - sehr unterschiedliche Sichtweisen
- Ziel: Zurechtfinden auf der Landkarte, Sichtweisen kennenlernen
- Vorlesung über algorithmische Ideen, nicht über Technologien!

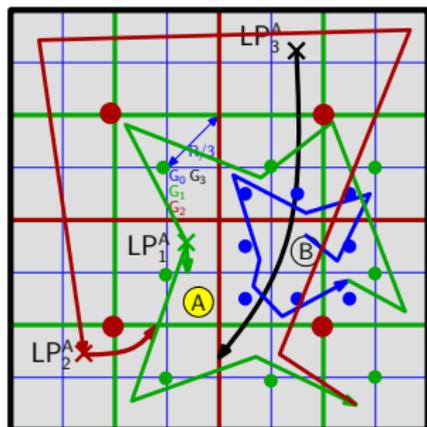
Auszug aus den behandelten Themen:

- Geographisches Routing
- Mobilität und Location Services
- Topologiekontrolle
- Lokalisierung
- Routing (ohne Geo-Koordinaten)
- Data Gathering
- Network Coding
- Clustering
- Medienzugriffskontrolle, Färbung
- Kapazität und Scheduling
- Synchronisation



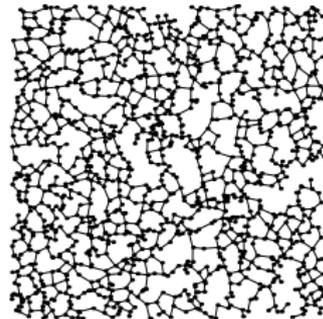
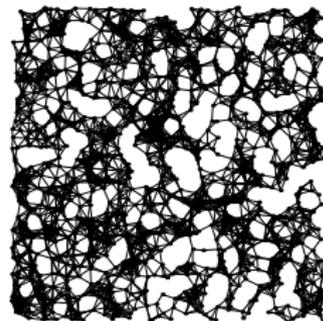
Auszug aus den behandelten Themen:

- Geographisches Routing
- Mobilität und Location Services
- Topologiekontrolle
- Lokalisierung
- Routing (ohne Geo-Koordinaten)
- Data Gathering
- Network Coding
- Clustering
- Medienzugriffskontrolle, Färbung
- Kapazität und Scheduling
- Synchronisation



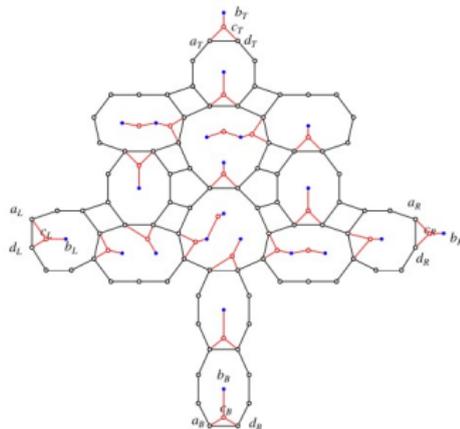
Auszug aus den behandelten Themen:

- Geographisches Routing
- Mobilität und Location Services
- Topologiekontrolle
- Lokalisierung
- Routing (ohne Geo-Koordinaten)
- Data Gathering
- Network Coding
- Clustering
- Medienzugriffskontrolle, Färbung
- Kapazität und Scheduling
- Synchronisation



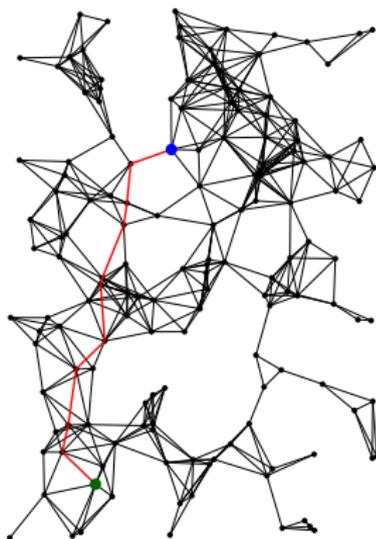
Auszug aus den behandelten Themen:

- Geographisches Routing
- Mobilität und Location Services
- Topologiekontrolle
- Lokalisierung
- Routing (ohne Geo-Koordinaten)
- Data Gathering
- Network Coding
- Clustering
- Medienzugriffskontrolle, Färbung
- Kapazität und Scheduling
- Synchronisation



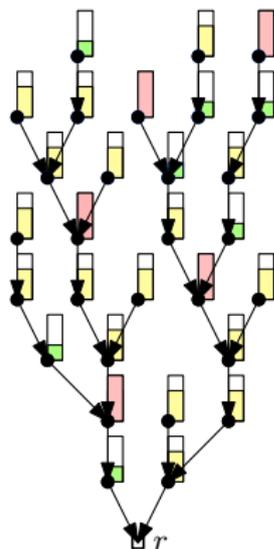
Auszug aus den behandelten Themen:

- Geographisches Routing
- Mobilität und Location Services
- Topologiekontrolle
- Lokalisierung
- Routing (ohne Geo-Koordinaten)
- Data Gathering
- Network Coding
- Clustering
- Medienzugriffskontrolle, Färbung
- Kapazität und Scheduling
- Synchronisation



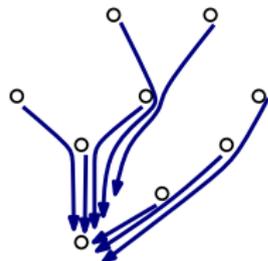
Auszug aus den behandelten Themen:

- Geographisches Routing
- Mobilität und Location Services
- Topologiekontrolle
- Lokalisierung
- Routing (ohne Geo-Koordinaten)
- Data Gathering
- Network Coding
- Clustering
- Medienzugriffskontrolle, Färbung
- Kapazität und Scheduling
- Synchronisation



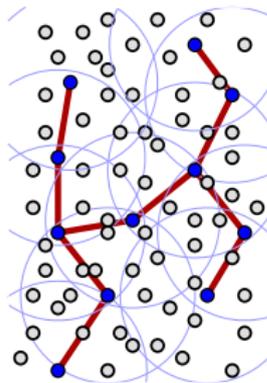
Auszug aus den behandelten Themen:

- Geographisches Routing
- Mobilität und Location Services
- Topologiekontrolle
- Lokalisierung
- Routing (ohne Geo-Koordinaten)
- Data Gathering
- Network Coding
- Clustering
- Medienzugriffskontrolle, Färbung
- Kapazität und Scheduling
- Synchronisation



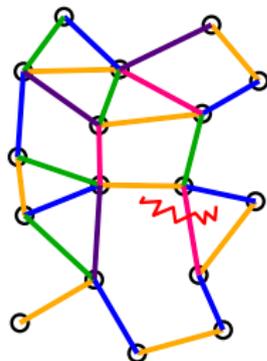
Auszug aus den behandelten Themen:

- Geographisches Routing
- Mobilität und Location Services
- Topologiekontrolle
- Lokalisierung
- Routing (ohne Geo-Koordinaten)
- Data Gathering
- Network Coding
- Clustering
- Medienzugriffskontrolle, Färbung
- Kapazität und Scheduling
- Synchronisation



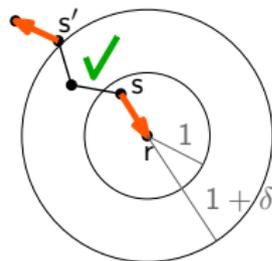
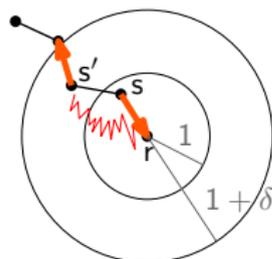
Auszug aus den behandelten Themen:

- Geographisches Routing
- Mobilität und Location Services
- Topologiekontrolle
- Lokalisierung
- Routing (ohne Geo-Koordinaten)
- Data Gathering
- Network Coding
- Clustering
- Medienzugriffskontrolle, Färbung
- Kapazität und Scheduling
- Synchronisation



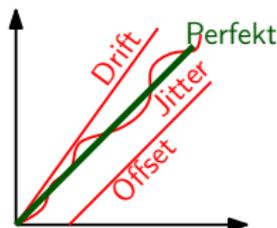
Auszug aus den behandelten Themen:

- Geographisches Routing
- Mobilität und Location Services
- Topologiekontrolle
- Lokalisierung
- Routing (ohne Geo-Koordinaten)
- Data Gathering
- Network Coding
- Clustering
- Medienzugriffskontrolle, Färbung
- Kapazität und Scheduling
- Synchronisation



Auszug aus den behandelten Themen:

- Geographisches Routing
- Mobilität und Location Services
- Topologiekontrolle
- Lokalisierung
- Routing (ohne Geo-Koordinaten)
- Data Gathering
- Network Coding
- Clustering
- Medienzugriffskontrolle, Färbung
- Kapazität und Scheduling
- Synchronisation



Hardwaredesign, Telematik, Robotik

Praxis

Realität

Testbed

Simulation

Modelle

Beweise

Theorie

Graphentheorie, Kombinatorik, Geometrie



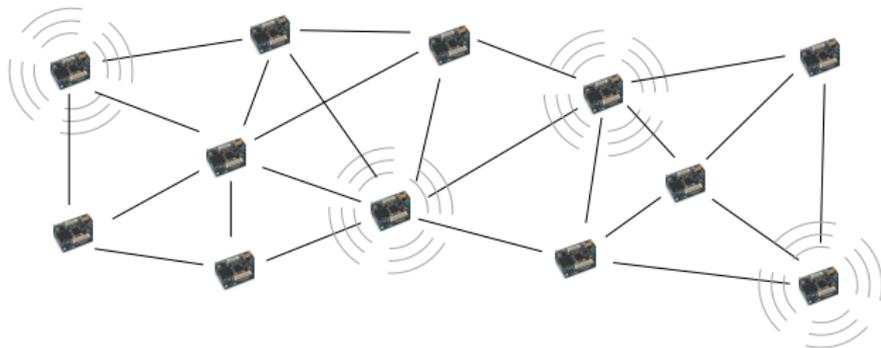
- Organisatorisches
- Kurze Einführung zu Sensornetzen
 - Sensorknoten
 - Anwendungen
 - Herausforderungen
- Vorlesungsübersicht
- Erste Schritte
 - Definitionen
 - Broadcasts und Flooding
 - Leader Election
 - Routing: Link Reversal

Definition

Ein *verteiltes System* ist eine Menge von selbständigen Recheneinheiten, die miteinander kommunizieren können.

- Kommunikationsparadigmen
 - synchron \leftrightarrow asynchron
 - nachrichtenbasiert \leftrightarrow shared memory
- Ad-hoc Netze sind von Natur aus nachrichtenbasiert!
- Synchrone Kommunikation deutlich einfacher zu analysieren (aber nur durch Synchronisation möglich: VL13)

- n Prozessoren p_1, \dots, p_n („Knoten“)
- Kommunikationsgraph („Topologie“, „Netz“)
 - Knoten: Prozessoren
 - Kanten: Mögliche Kommunikationspartner
 - nicht immer ungerichtet/symmetrisch
 - oft werden asymmetrische Links einfach ignoriert



- Jeder Prozessor führt *dasselbe* Programm aus
 - Programme starten gleichzeitig (meistens), ggf. in verschiedenen Zuständen (Eingabe)
 - Runde: Berechnungsschritt und Kommunikationsschritt
 - beliebige Berechnungen im Berechnungsschritt
 - *Verfassen* von Nachrichten an Nachbarn in Berechnungsschritt
 - *Zustellen* von Nachrichten in Kommunikationsschritt
- Nach jeder Runde hat jeder Prozessor p_i einen *Zustand* C_i
 - enthält zugestellte Nachrichten aus der letzten Runde
 - eine Teilmenge der Zustände sind *Endzustände*
 - aus Endzuständen können nur Endzustände erreicht werden
 - eine Ausführung ist beendet, wenn alle Prozessoren in Endzuständen sind

Definition

Ein verteilter Algorithmus ist *korrekt*, wenn seine Ausführung immer nach endlich vielen Schritten endet und danach eine korrekte Lösung des Problems vorliegt.

- Eingaben liegen (wenn überhaupt) verteilt vor
- Ausgabe/Lösung besteht aus Zustand der Knoten bei Terminierung
- Bis auf Weiteres: Nachrichten sind kurz, z. B. $O(\log n)$ Bits

Beispiel: Broadcast, Flooding

Gegeben: Symmetrischer, *zusammenhängender* Kommunikationsgraph $G = (V, E)$, Knoten $r \in V$.

Problem: Sende eine Nachricht M von r an alle Knoten.

Algorithmus 1: Flooding in Knoten p_i

wenn $p_i = r$ **dann**

| sende M an alle Nachbarn, nimm Endzustand an

sonst

| warte, bis Nachricht M empfangen wird

| sende M an alle Nachbarn, nimm Endzustand ein

Lemma

Der Flooding-Algorithmus auf einem symmetrischen und zusammenhängenden Graphen ist korrekt, d. h. nach endlich vielen Schritten ist die Ausführung beendet und alle Knoten haben M erhalten.

Beweis:

- Ein Prozessor ist in Endzustand \Leftrightarrow er hat M erhalten
- Annahme: Es gibt einen Prozessor p_i der nicht terminiert.
 - keiner seiner Nachbarn terminiert
 - kein Knoten, von dem es einen Weg zu p_i gibt, terminiert
 - Widerspruch: r terminiert sicher und G ist zusammenhängend!

- Laufzeit (Zeitkomplexität)
 - Maximale Anzahl der Runden bis alle Knoten einen Endzustand erreicht haben
- Nachrichtenkomplexität
 - Maximale Anzahl der Nachrichten, die insgesamt verschickt werden
 - Maximale Anzahl der Nachrichten, die ein einzelner Knoten verschicken muss
 - Manchmal in Sensornetzen: Dieselbe Nachricht an alle Nachbarn zählt nur einfach (*One-Hop-Broadcast*)

Lemma

Der Flooding-Algorithmus hat Nachrichtenkomplexität $2|E|$ und Zeitkomplexität D .

- $D := \max_{u,v \in V} d_G(u, v)$: Durchmesser des Graphen
- Beweis:
 - Jeder Knoten sendet genau einmal eine Nachricht zu allen Nachbarn: $\sum_{p_i \in V} \deg(p_i) = 2|E|$
 - Induktion: Nach k Runden sind alle Knoten in Abstand $\leq k - 1$ zu r im Endzustand.
 - IA: r nimmt Endzustand in Runde 1 ein.
 - IS: Jeder Knoten u mit Abstand k zu r hat einen Nachbarn v mit Abstand $k - 1$
 - $\stackrel{IV}{\Rightarrow} v$ nimmt Endzustand spätestens in Runde k ein und sendet M an u
 - $\Rightarrow u$ terminiert spätestens in Runde $k + 1$

Definition (Anonymität)

Ein verteilter Algorithmus heißt *anonym*, wenn die Prozessoren keine unterschiedlichen IDs besitzen.

Beispiel: Leader Election

Leader Election Problem:

Gegeben: Symmetrischer, zusammenhängender
Kommunikationsgraph G

Problem: Genau ein Prozessor soll als *Leader* ausgezeichnet
werden

Frage: *Gibt es einen anonymen Algorithmus für das Leader Election
Problem?*

Beispiel: Leader Election

Leader Election Problem:

Gegeben: Symmetrischer, zusammenhängender Kommunikationsgraph G

Problem: Genau ein Prozessor soll als *Leader* ausgezeichnet werden

Frage: *Gibt es einen anonymen Algorithmus für das Leader Election Problem?*

Satz

Es gibt **noch nichtmals einen** anonymen Algorithmus für das Leader Election Problem **in Ringen** und damit auch **keinen für beliebige Graphen**.

Satz

Es gibt keinen anonymen Algorithmus für das Leader Election Problem in Ringen.

Beweis:

- Vereinfachung: Jeder unterscheidet „linken“ und „rechten“ Nachbarn gleich
- Induktion: Knoten beginnen jede Runde im selben Zustand
 - IA: Runde 0: Keine IDs, keine Eingabe, Knoten beginnen im selben Zustand
 - IS: Runde $k \leadsto k + 1$:
 - Alle Knoten beginnen Runde k im selben Zustand
 - ⇒ Alle Knoten führen dieselbe Berechnung aus
 - ⇒ Alle Knoten enden im selben Zustand
 - ⇒ Alle Knoten senden dieselbe Nachricht nach links (rechts)
 - ⇒ Alle Knoten beginnen Runde $k + 1$ im selben Zustand
 - Kein Knoten kann alleine in Leader-Zustand enden.

Satz

Es gibt keinen anonymen Algorithmus für das Leader Election Problem in Ringen.

Beweis:

- Vereinfachung: Jeder unterscheidet „linken“ und „rechten“ Nachbarn gleich
- Induktion: Knoten beginnen jede Runde im selben Zustand
 - IA: Runde 0: Keine IDs, keine Eingabe, Knoten beginnen im selben Zustand
 - IS: Runde $k \leadsto k + 1$:
 - Alle Knoten beginnen Runde k im selben Zustand
 - ⇒ Alle Knoten führen dieselbe Berechnung aus
 - ⇒ Alle Knoten enden im selben Zustand
 - ⇒ Alle Knoten senden dieselbe Nachricht nach links (rechts)
 - ⇒ Alle Knoten beginnen Runde $k + 1$ im selben Zustand
 - Kein Knoten kann alleine in Leader-Zustand enden.

Algorithmus 2: Leader Election, Knoten p_i kennt ID_i

sende ID_i an linken Nachbarn

wenn *eine ID empfangen wird* **dann**

wenn $ID > ID_i$ **dann**

 sende ID an linken Nachbarn

wenn $ID = ID_i$ **dann**

 sende Nachricht „terminiere“ an linken Nachbarn

 terminiere als Leader

wenn Nachricht „terminiere“ empfangen wird **dann**

 sende Nachricht „terminiere“ an linken Nachbarn

 terminiere als Non-Leader

Zeit- und Nachrichtenkomplexität?

Algorithmus 2: Leader Election, Knoten p_i kennt ID_i

sende ID_i an linken Nachbarn

wenn *eine ID empfangen wird* **dann**

wenn $ID > ID_i$ **dann**

 | sende ID an linken Nachbarn

wenn $ID = ID_i$ **dann**

 | sende Nachricht „terminiere“ an linken Nachbarn
 | terminiere als Leader

wenn *Nachricht „terminiere“ empfangen wird* **dann**

 | sende Nachricht „terminiere“ an linken Nachbarn
 | terminiere als Non-Leader

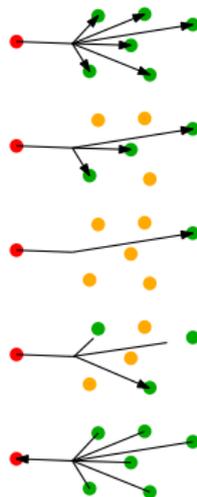
- Zeitkomplexität: $O(n)$
- Nachrichtenkomplexität: $O(n^2)$
 - *bessere Algorithmen:* $O(n \log n)$

Wie könnte ein Leader-Election-Algorithmus für allgemeine Graphen aussehen?



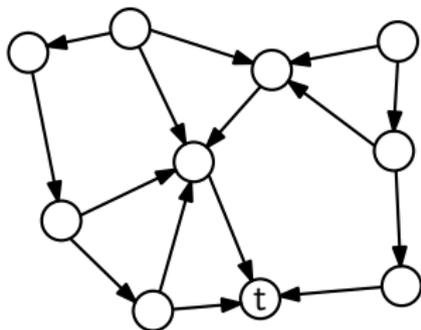
Routing is the act of moving information across a network from a source to a destination.

- Broadcast
 - ein Knoten sendet (gleiche) Nachricht an alle
- Multicast
 - ein Knoten sendet (gleiche) Nachricht an viele andere
- Unicast
 - ein Knoten sendet Nachricht an einen anderen Knoten
- Anycast
 - ein Knoten sendet Nachricht an irgendeinen Knoten aus einer Zielmenge
- Convergecast
 - Alle (viele) Knoten senden Nachrichten an eine Senke



- proaktiv ↔ reaktiv
 - proaktiv: Routinginformationen werden im Voraus verteilt
 - + bei wenig Mobilität einmaliger Aufwand
 - in der Regel höherer Speicherbedarf
 - reaktiv: Routinginformationen werden bei Bedarf ermittelt
 - + bei viel Mobilität kein permanenter Overhead
 - Routen müssen bei Bedarf erst gefunden werden
- Beispiel "reaktiv": Flooding
 - meistens ungeeignet: alle bekommen Nachricht (mehrfach)
 - manchmal die einzige Lösung (hohe Mobilität)
- Beispiel "proaktiv": Link State Routing
 - Knoten tauschen regelmäßig gesamten Graphen aus
 - Pakete werden immer auf kürzeste Wege geschickt

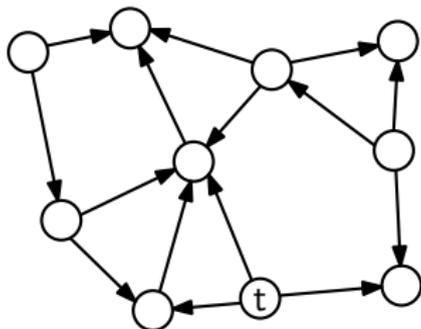
- Einfaches proaktives Protokoll für Convergecasts (eine Senke)
- Richte Kanten so, dass *jeder* gerichtete Pfad zur Senke führt
⇒ Schicke Pakete einfach über eine beliebige ausgehende Kante!



- Wie orientiert man Kanten so?

Definition

Ein gerichteter Graph ohne Zyklen heißt *DAG* (directed acyclic graph). Er heißt *t*-zielorientiert, wenn *t* der einzige Knoten ohne ausgehende Kante ist.



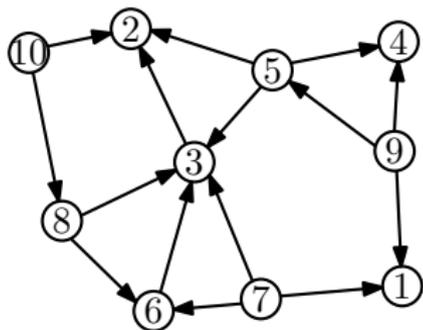
- DAG: Pakete können nicht im Kreis wandern
 - landen irgendwann bei Knoten ohne ausg. Kante
- *t*-orientiert: Pakete landen bei *t*!
- Das wollen wir (für gegebenes *t*)!

Gegeben: Ungerichteter Kommunikationsgraph $G = (V, E)$,
Senke $t \in V$

Gesucht: Orientierung der Kanten, so dass der entstehende gerichtete Graph G' ein t -zielorientierter DAG ist

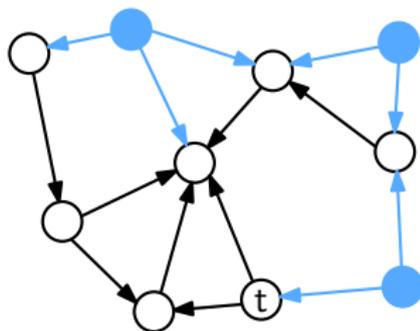
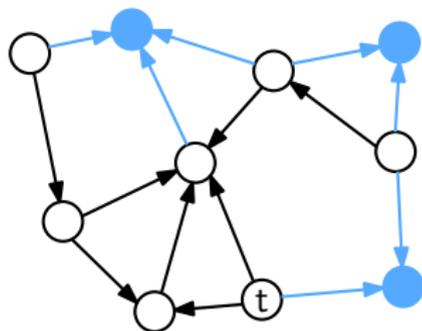
Richte alle Kanten von höherer ID zur niedrigeren

- jeder Knoten sendet seine ID an alle Nachbarn
- jeder Knoten merkt sich Richtungen inzidenter Kanten
- das ist ein DAG: Auf jedem Pfad nehmen IDs ab!



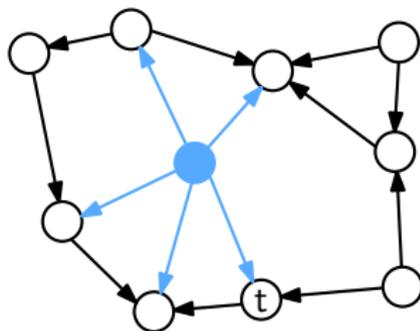
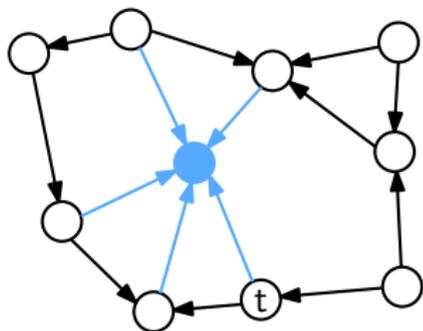
Falls Knoten $u \neq t$ keine ausgehende Kante hat, drehe alle inzidenten Kanten um!

- Knoten ohne ausgehende Kanten schicken „reverse“-Nachricht an alle Nachbarn
- das bleibt ein DAG: Gedrehte Kanten liegen auf keinem Zyklus!
- Irgendwann ist t die einzige Senke!?



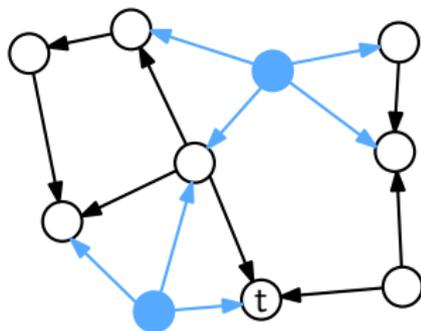
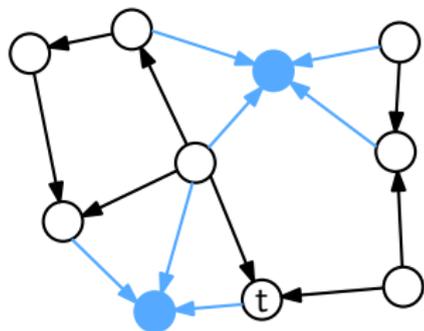
Falls Knoten $u \neq t$ keine ausgehende Kante hat, drehe alle inzidenten Kanten um!

- Knoten ohne ausgehende Kanten schicken „reverse“-Nachricht an alle Nachbarn
- das bleibt ein DAG: Gedrehte Kanten liegen auf keinem Zyklus!
- Irgendwann ist t die einzige Senke!?



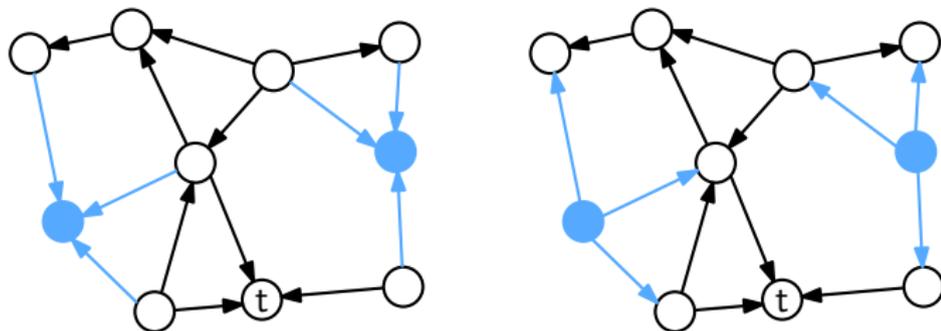
Falls Knoten $u \neq t$ keine ausgehende Kante hat, drehe alle inzidenten Kanten um!

- Knoten ohne ausgehende Kanten schicken „reverse“-Nachricht an alle Nachbarn
- das bleibt ein DAG: Gedrehte Kanten liegen auf keinem Zyklus!
- Irgendwann ist t die einzige Senke!?



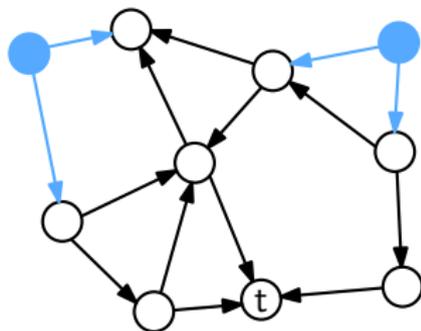
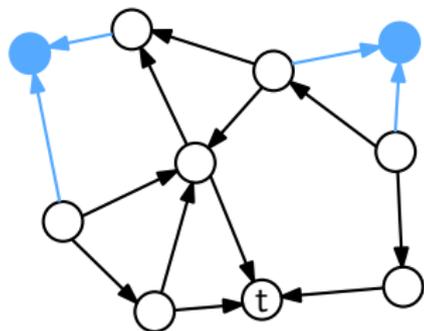
Falls Knoten $u \neq t$ keine ausgehende Kante hat, drehe alle inzidenten Kanten um!

- Knoten ohne ausgehende Kanten schicken „reverse“-Nachricht an alle Nachbarn
- das bleibt ein DAG: Gedrehte Kanten liegen auf keinem Zyklus!
- Irgendwann ist t die einzige Senke!?



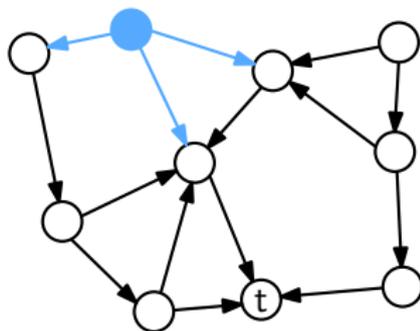
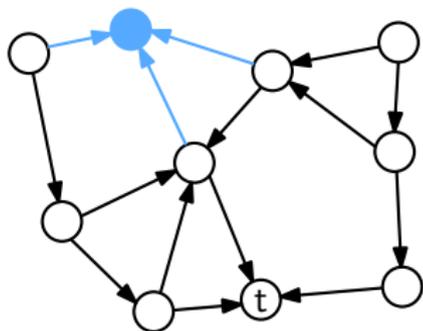
Falls Knoten $u \neq t$ keine ausgehende Kante hat, drehe alle inzidenten Kanten um!

- Knoten ohne ausgehende Kanten schicken „reverse“-Nachricht an alle Nachbarn
- das bleibt ein DAG: Gedrehte Kanten liegen auf keinem Zyklus!
- Irgendwann ist t die einzige Senke!?



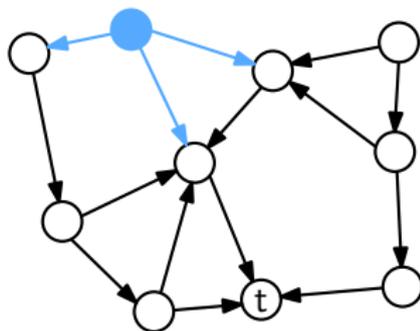
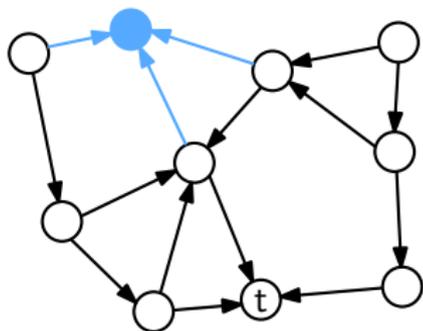
Falls Knoten $u \neq t$ keine ausgehende Kante hat, drehe alle inzidenten Kanten um!

- Knoten ohne ausgehende Kanten schicken „reverse“-Nachricht an alle Nachbarn
- das bleibt ein DAG: Gedrehte Kanten liegen auf keinem Zyklus!
- Irgendwann ist t die einzige Senke!?



Falls Knoten $u \neq t$ keine ausgehende Kante hat, drehe alle inzidenten Kanten um!

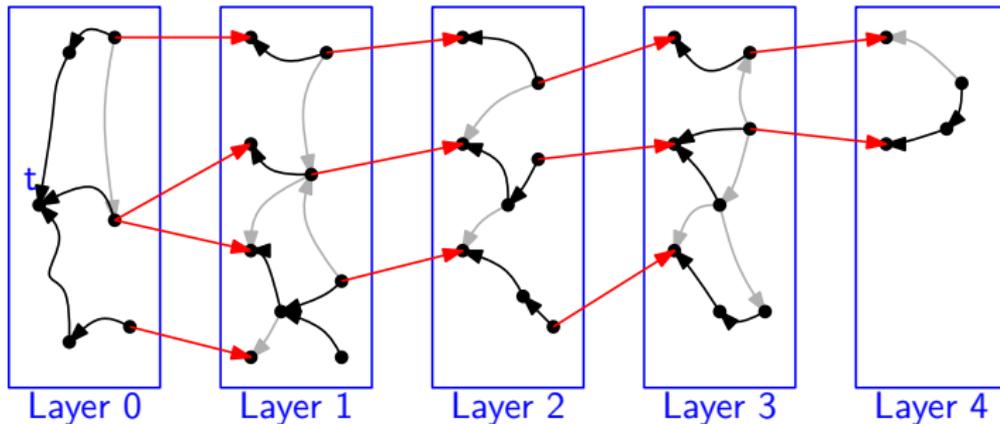
- Knoten ohne ausgehende Kanten schicken „reverse“-Nachricht an alle Nachbarn
- das bleibt ein DAG: Gedrehte Kanten liegen auf keinem Zyklus!
- Irgendwann ist t die einzige Senke!?! **Terminiert das immer?**



Initial- und Zwischenlösungen induzieren *Layer*.

Definition

Ein Knoten u liegt im kleinsten Layer k , so dass es einen (ungerichteten) Pfad von u zu t gibt, der nur k Kanten entgegen der Orientierung nutzt.



Betrachte parallele Änderungen in beliebiger Reihenfolge:

Lemma

Wenn ein Knoten u seine Kanten umorientiert, verringert sich u 's Layer um 1, alle anderen Knoten bleiben in ihrem Layer.

Beweis (Teil 1)

- Sei u aus Layer k .
 - ⇒ es gibt einen Pfad von u nach t , der k Kanten entgegen der (alten) Orientierung nutzt. Darunter ist genau eine zu u inzidente Kante.
 - ⇒ dieser Pfad nutzt $k - 1$ Kanten entgegen der neuen Orientierung
 - ⇒ nach der Umorientierung ist u in Layer $k - 1$

Betrachte parallele Änderungen in beliebiger Reihenfolge:

Lemma

Wenn ein Knoten u seine Kanten umorientiert, verringert sich u 's Layer um 1, alle anderen Knoten bleiben in ihrem Layer.

Beweis (Teil 2)

- Sei $v \neq u$,
 - Sei P ein ungerichteter Pfad von v zu t
 - Anzahl der Kanten, die P entgegen der Orientierung nutzt, ändert sich durch das Umorientieren nicht:
 - Fall 1: P enthält u nicht (trivial)
 - Fall 2: P enthält u , dann ändern sich zwei Kanten unterschiedlicher Richtung auf P
- ⇒ v 's Layer ändert sich nicht.

Betrachte parallele Änderungen in beliebiger Reihenfolge:

Lemma

Wenn ein Knoten u seine Kanten umorientiert, verringert sich u 's Layer um 1, alle anderen Knoten bleiben in ihrem Layer.

Beweis (Teil 2)

- Sei $v \neq u$,
 - Sei P ein ungerichteter Pfad von v zu t
 - Anzahl der Kanten, die P entgegen der Orientierung nutzt, ändert sich durch das Umorientieren nicht:
 - Fall 1: P enthält u nicht (trivial)
 - Fall 2: P enthält u , dann ändern sich zwei Kanten unterschiedlicher Richtung auf P
- ⇒ v 's Layer ändert sich nicht.

Satz

Full Link Reversal führt $O(n^2)$ Umorientierungen durch und terminiert nach $O(n^2)$ Schritten in einem t -zielorientierten DAG.

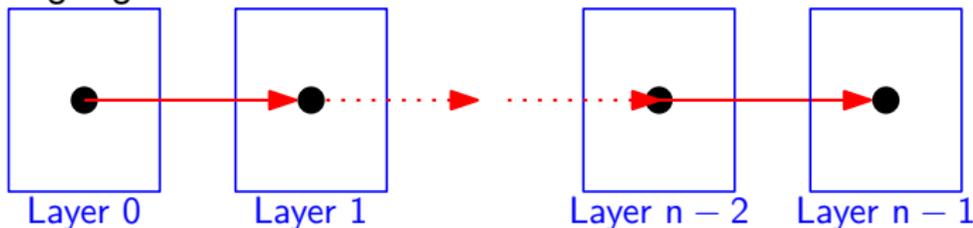
Beweis:

- in jedem Schritt ist der Graph ein DAG
- kein Knoten kann in einem Layer $> n$ starten
- in jedem Schritt verringert mindestens ein Knoten sein Layer
- nach spätestens n^2 Schritten sind alle Knoten in Layer 0
- wenn alle Knoten in Layer 0 sind, ist der DAG t -zielorientiert

Satz

Es gibt Initiallösungen auf Graphen bei denen der Full-Reversal-Algorithmus insgesamt $\Theta(n^2)$ Umorientierungen ausführt.

- sogar ganz einfache: falsch orientierte Pfade

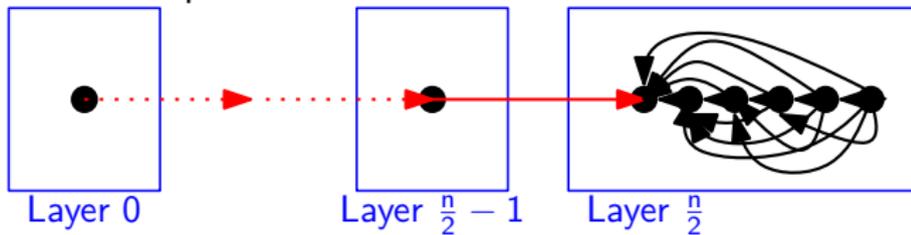


- $\sum_{i=1}^n (i-1) \in \Theta(n^2)$ Umorientierungen, da ein Knoten in Layer i exakt i mal umorientiert werden muss
- ⇒ Schranke für die Zahl der Umorientierungen scharf!
- Und die Laufzeitschranke?
 - In diesem Beispiel nur $\Theta(n)$ Schritte

Satz

Es gibt Initiallösungen auf Graphen bei denen der Full-Reversal-Algorithmus insgesamt $\Theta(n^2)$ Schritte benötigt.

- etwas komplexere:



- vollständiger DAG auf $n/2$ Knoten in Layer $n/2$
 - jeder davon braucht $n/2$ Umorientierungen
 - immer nur einer kann zur Zeit Senke sein
- ⇒ Laufzeit $\Omega(n^2)$, Laufzeitschranke $O(n^2)$ damit auch scharf!

- Hervorragend geeignet, um auf Änderungen in der Topologie zu reagieren
- Ausfälle die Senken erzeugen werden mit FLR gelöst
 - Im schlimmsten Fall sehr aufwendig!
- Neue Knoten, neue Kanten einfach!?

- Vorlesungsübersicht
 - Jede Vorlesung ein Thema, algorithmische Perlen
- Sensor- und Ad-hoc-Netze
 - Sensorknoten
 - Anwendungen
 - Herausforderungen
- Verteilte Systeme
 - Algorithmus, Endzustände, Anonymität/IDs
 - Broadcast, Leader Election in Ringen
- Erster Sensornetzalgorithmus: Full Link Reversal
 - Einfache Idee, nichttriviale Analyse
 - Leicht umzusetzen, aber im Worst-Case *sehr* schlecht

- H. Attiya, J. Welch: *Distributed Computing*. Wiley, 2004.
- C. Busch, S. Surapaneni, S. Tirthapura: *Analysis of link reversal routing algorithms for mobile ad hoc networks*. In SPAA '03: Proceedings of the Fifteenth Annual ACM Symposium on Parallel Algorithms and Architectures, 2003, ACM