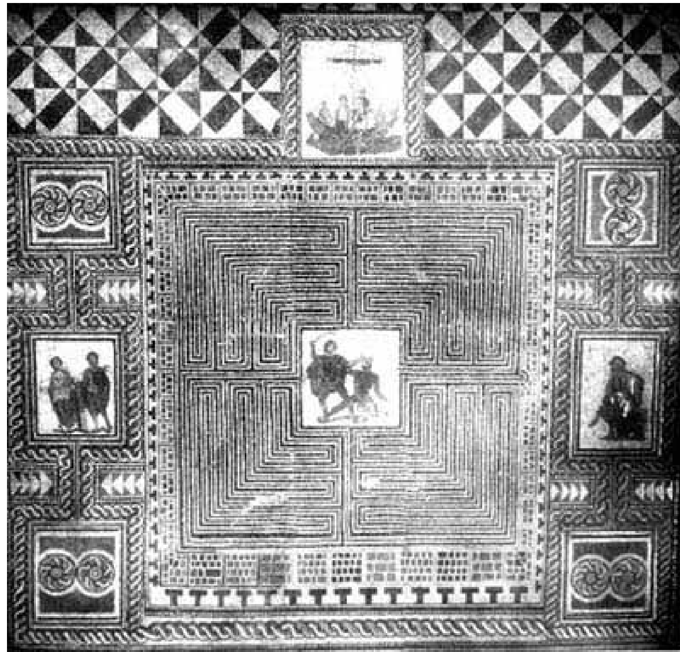
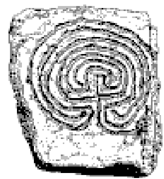
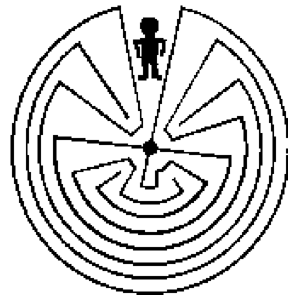
Labyrinthe



Römisches Mosaik (aus Loig bei Salzburg) illustriert
die Geschichte von Theseus und Minotaurus im Labyrinth



Arcera, Spanien



O'odham-Volk
(Papago-Indianer)
aus Süd-Arizona

Labyrinthe (2)

Der Algorithmus von Tarry

Traversieren *beliebiger* (zusammenhängender, unger.) Graphen
(Nachbaridentitäten der Knoten brauchen nicht bekannt zu sein!)

- Zwei Regeln zum Propagieren eines Tokens, die *wenn immer möglich* von einem Prozess angewendet werden:

R1: Ein Prozess schickt das Token niemals zwei Mal über die gleiche Kante *(Wo sagt Tarry das?)*

R2: Ein Prozess (≠ Initiator) schickt das Token erst dann an denjenigen Prozess zurück, von dem er es erstmalig erhielt, wenn er keine andere unbenutzte Kante mehr hat

--> Algorithmus ist nichtdeterministisch!

- Beh.: Algorithmus terminiert

Bew.: Max. 2e Mal wird das Token versendet...

- Beh.: Wenn der Algorithmus terminiert ist, ist das Token bei jedem Prozess vorbeigekommen und wieder zum Initiator zurückgekehrt

--> Tarry-Algorithmus ist ein Traversierungsalgorithmus
--> "Ziel" kann auch eine andere Stelle als der Eingang sein

Bew.: ...

Tarry's Verfahren ist ein Wellenalgorithmus

(1) Terminierung ==> Token ist beim Initiator

Beweis: Für jeden Nicht-Initiator p gilt: Wenn p das Token hat, dann hat p das Token k-Mal (auf jeweils unterschiedlichen Kanälen, Regel R1) erhalten und auf k-1 (jeweils unterschiedlichen Kanälen) gesendet
==> es gibt noch mindestens einen unbenutzten "Ausgangskanal"
==> das Token bleibt nicht bei p.

(2) Alle Kanäle des Initiators werden in beiden Richtungen genau 1 Mal vom Token durchlaufen

Beweis: Für jeden "Ausgangskanal" klar, sonst wäre der Algorithmus nicht terminiert: Nach (1) bleibt das Token nicht bei einem andereren Prozess stecken. Das Token muss genauso oft zum Initiator zurückgekehrt sein, wie es von dort weggeschickt wurde, und zwar über jeweils andere Kanäle (Regel R1) ==> Jeder "Eingangskanal" des Inititators wurde benutzt.

(3) Für jeden besuchten Prozess p gilt: Alle Kanäle von p wurden in beide Richtungen durchlaufen

Beweis durch Widerspruch: Betrachte den ersten (= "frühesten") besuchten Prozess p, für den dies nicht gilt. Nach (2) ist dies nicht der Initiator. Sei Vater(x) derjenige Prozess, von dem x erstmalig das Token erhielt. Für Vater(p) gilt nach Wahl von p, dass alle Kanäle in beide Richtungen durchlaufen wurden. ==> p hat das Token an Vater(p) gesendet. Wegen Regel R2 hat daher p das Token auf allen anderen (Ausgangs)kanälen gesendet. Das Token musste dazu aber genauso oft auf jeweils anderen Kanälen (R1) zu p zurückkommen. ==> Alle Eingangskanäle wurden ebenfalls benutzt.

(4) Alle Prozesse wurden besucht

Beweis: Andernfalls gäbe es eine Kante von einem besuchten Prozess q zu einem unbesuchten Prozess p, da der Graph zusammenhängend ist. Dies steht aber im Widerspruch zu (3), da diese Kante vom Token durchlaufen wurde.

Die Welleneigenschaft ergibt sich i.w. aus (1) und (4)

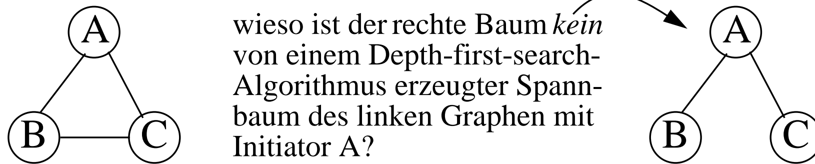
Depth-first-Algorithmen und Spannbäume

Klassischer Depth-first-search-Algorithmus:

- Token geht erst dann zurück, wenn alles andere "abgegrast" ist
- Token kehrt um, sobald es auf einen bereits besuchten Knoten trifft
- jede Kante wird in jede Richtung genau 1 Mal durchlaufen
- Tarry-Algorithmus lässt sich zu Depth-first-Traversierung spezialisieren

==> 2e Nachrichten, 2e Zeitkomplexität

- Depth-first-search-Algorithmen liefern beim Durchlaufen eines Graphen einen Spannbaum (Wurzel = Initiator)
  (wie jeder Wellenalgorithmus!)



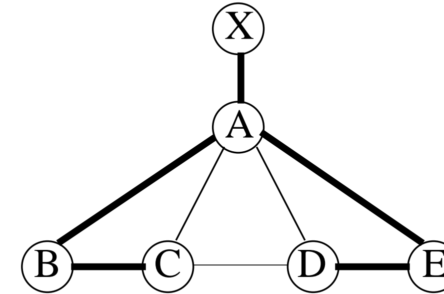
- Eine Charakterisierung solcher Spannbäume:

  Jede Kante des Graphen, die keine Kante des Spannbaumes ist, verbindet zwei Knoten, die auf dem gleichen Ast liegen

  Weg von der Wurzel zu einem Blatt

- wieso?
- kann der rechte Spannbaum vom Tarry-Algorithmus erzeugt werden?

Tarry-Algorithmus und Spannbäume



- Der fett eingezeichnete Baum (mit Wurzel X) ist kein Depth-first-Spannbaum (wegen der Kante CD)
- Dennoch kann der Baum mit dem Tarry-Algorithmus über folgende Traversierung erzeugt werden:
  X, A, B, C, A, E, D, A, C, D, C, B, A, D, E, A, X

  bis hierhin auch mit depth-first!

  alternativ auch D oder C, aber nicht B oder X

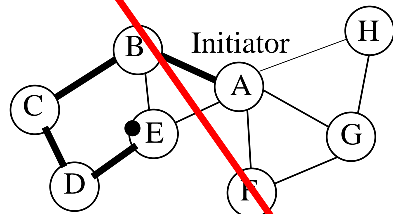
- Mit folgender Regel lässt sich der Nichtdeterminismus des Tarry-Algorithmus soweit einschränken, dass das klassische Depth-first-Traversierungsverfahren resultiert:

> **R3**: Ein Prozess schickt ein empfangenes Token sofort über die gleiche Kante zurück, wenn dies nach R1 und R2 gestattet ist

- damit ist in obigem Beispiel X, A, B, C, A, E... nicht mehr gestattet!
- Denkübung: *Wieso* wird durch R1-R3 die Depth-first-Traversierung realisiert?
- Denkübung: Kann mit dem Algorithmus *jeder* Spannbaum realisiert werden?

Depth-first mit Besuchslisten

Voraussetzung: Nachbaridentitäten der Knoten sind bekannt



Token bei E enthält die Namen der bereits besuchten Knoten A, B, C, D, E; Knoten E wird also das Token nicht an B oder A weitersenden (sondern zurück an D)

Idee: - Token merkt sich bereits besuchte Knoten und wird nicht dorthin propagiert

- Jeder Knoten (≠ Initiator) wird also nur 1 Mal besucht (und sendet Token 1 Mal zurück)

==> 2n-2 Nachrichten, 2n-2 Zeitkomplexität

Vorteil: Bei "dichten" Netzen (e ≫n) effizienter!

Nachteil: Hohe Bit-Komplexität (d.h. "lange" Nachrichten)

Bem.: Nachbarschaftswissen lässt sich immer mit 2e Nachrichten erreichen! (Algorithmus dafür haben wir früher kennengelernt)

Frage: Lässt sich vielleicht ein Wellenalgorithmus angeben, bei dem man immer oder zumindest manchmal mit n-1 Nachrichten auskommt (wenn Nachbarschaftswissen vorhanden ist; dann ist das eine untere Schranke, wie wir wissen), statt mit 2n-2?

Depth-first mit Auftragslisten

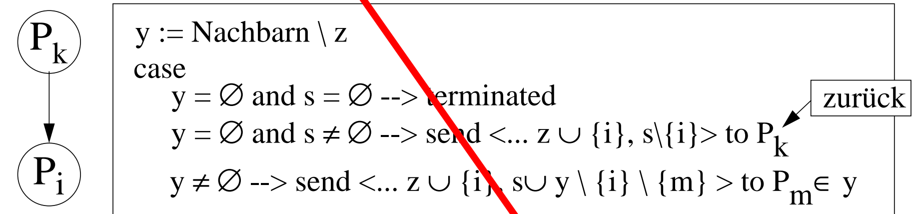
(Voraussetzung wieder: Nachbaridentitäten der Knoten sind bekannt)

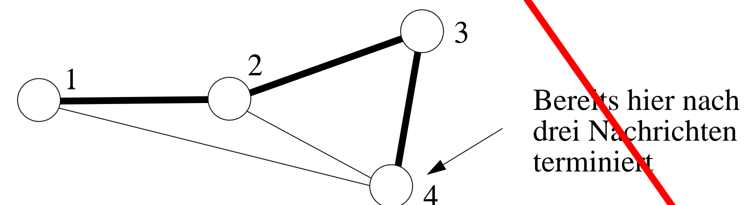
- Idee: Token enthält zwei Mengen <...,z,s>:

  z: Menge der bereits besuchten Knoten
  s: Menge noch zu besuchender Knoten ("Aufträge")

- Initial sendet Initiator P_i an einen Nachbarn P_j: <..., {i}, Nachbarn\{j}>

- Bei Empfang von <...,z,s> von P_k bei P_i:





- Beendet, wenn s leer wird (wieso? Beweis?)

- Anderer Knoten als Initiator stellt Terminierung fest (Nachteil? Vorteil, da schneller?)

- Nachrichten- / Zeitkomplexität: n-1 bis 2(n-1)

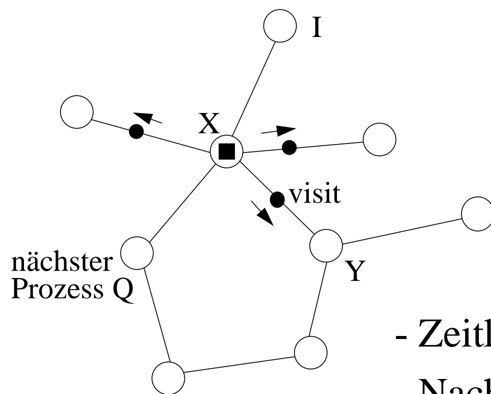
Verfahren von Awerbuch

- Voraussetzung: Nachbaridentitäten unbekannt
- Idee: Prozess weiss spätestens dann, wenn das Token ihn besucht, an welche Nachbarn es nicht weitergereicht werden soll, weil es dort schon war

- Wenn Token einen Prozess besucht:

Sende *visit-Nachrichten* an "alle" Nachbarn;
warte auf *Ack* bevor Token weitergereicht wird
--> alle Nachbarn wissen, wo Token bereits war

Ein Prozess schickt Token niemals an einen Prozess, von dem er eine visit-Nachricht erhalten hat
(Ausnahme: Vaterknoten beim Backtrack)



Keine visit-Nachricht notwendig an:
- Vaterprozess (hier: I)
- nächsten Prozess (hier: Q)

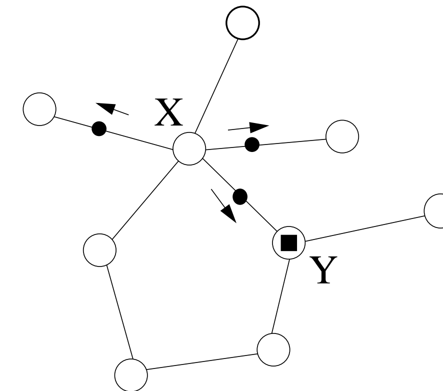
- Zeitkomplexität: 4n-2
- Nachrichtenkomplexität: 2 visit, 2 ack, bzw. Token... --> ca. 4e

- Frage: Muss Y eine visit-Nachricht an X schicken?
- Frage: Wozu überhaupt auf ack-Nachrichten warten?

Variante von Cidon

- Idee: Wie Awerbuch, aber keine acks senden!

- Was geschieht, wenn visit-Nachricht noch nicht da?
(also der Kanal XY langsam ist im Vergleich zum Umweg, den Token nahm)



- X erhält Token auf "falscher" Kante von Y --> ignorieren
(aber wie eine visit-Nachricht von Y behandeln; Y sendet kein visit mehr an X)

- Y erhält visit-Nachricht von X verspätet, nachdem bereits das Token an X gesendet wurde --> Token neu generieren und an einen anderen Nachbarn schicken

---

- Zeitkomplexität: 2n-2

Kann es (für beliebige Graphen) ein schnelleres Traversierungsverfahren geben?

- Nachrichtenkomplexität: max. 4e (max. 2 visit, 2 Token pro Kante)

Der Phasenalgorithmus

- Voraussetzung (hier): bidirektionale Nachrichtenkanäle

- *Grundprinzip (Initiator):*

Prinzip geht auch auf *gerichteten* Graphen

```
{noch nicht terminiert}
- Nachricht an alle Nachbarn senden
- Lokale Berechnung durchführen
- Phase := Phase + 1  /* lok. Variable; initial 0 */
- Eine Nachricht von allen Nachbarn empfangen
```

- Nicht-Initiatoren steigen erst nach Erhalt einer ersten Nachricht in einen solchen Zyklus ein

---

Eigenschaften:

Aufgabe: Hiermit die *Wellen-eigenschaft* nachweisen!

Was sind visit- und conclude-Ereignise?

- Falls ein Prozess in der i-ten Phase ist, befinden sich seine Nachbarn in der i-ten, i-1-ten, oder i+1-ten (wieso?)
- Zwei Prozesse der Entfernung q haben zu jedem Zeitpunkt einen "Phasenunterschied" von max. q (wieso?)
  - Denkübung: können alle Prozesse gleichzeitig in Phase 1 (oder allg.: k) sein?
- Mit D = Durchmesser --> Ist der (einzige) Initiator am Ende von Phase D, wurde jeder vom Initiator (indirekt) erreicht
  - Denkübung: hat der Initiator dann bereits von allen Prozessen indirekt gehört?
  - wenn ich weiss, dass jeder in einer Phase > 0 ist, gibt es dann eine effektiv nutzbare Nachrichtenkette von jedem zu mir?
- Eine obere Schranke von D (z.B. n) muss bekannt sein, damit die Terminierung festgestellt werden kann

Algorithmus von Finn

- Wellenalgorithmus für (stark zusammenhängende) *gerichtete* Graphen (unterscheide daher *in-neighbors* und *out-neighbors*)

- Prozesse müssen eindeutige Identitäten besitzen

- *Idee des Algorithmus*: Solange es Prozesse gibt, die man kennt, aber deren Nachbarn man noch nicht alle kennt, ist man noch nicht fertig
  --> Hüllenbildung; dadurch Kennenlernen des gesamten Graphen

---

- Jeder Prozess P_i besitzt zwei Mengenvariablen A, B, die so initialisiert sind: $A = \{i\}$; $B = \emptyset$

R_i

Bei Empfang von <A',B'>:
$A := A \cup A'$; $B := B \cup B'$;
if {über alle Eingangskanäle irgendwann mind. eine Nachricht erhalten} **then** $B := B \cup \{i\}$; **fi**;
if {A oder B hat sich geändert} **then**
send <A,B> **to all** out-neighbors; **fi**

I_1

{noch nicht gestartet}
send <A,B> **to all** out-neighbors

init-Ereignis (was geschieht eigentlich bei *mehreren* Initiatoren?)

C_i

{A = B}
conclude

jeder Prozess kann conclude ausführen, insbesondere aber der Initiator

Finns Verfahren ist ein Wellenalgorithmus

(1) $x \in B ==> \textit{in-neighbors}(x) \subseteq A$
ist eine Invariante für alle Paare (A,B)

- Aktionen C und I verändert (A,B) nicht.
- Aktion I erzeugt ein neues Paar ($\{i\},\emptyset$).
- Die beiden update-Aktionen $A := A \cup A'$; $B := B \cup B'$ in Aktion R erhalten ebenfalls die Invariante: Für ein $x \in B$ nach dem update war $x \in B$ vor dem update oder $x \in B'$. Folglich war auch bereits vorher in-neighbors(x) $\subseteq$ A oder in-neighbors(x) $\subseteq$ A', und damit auch noch nach dem update.
- Falls in R das Statement $B := B \cup \{i\}$ ausgeführt wird, sind alle $y \in$ in-neighbors(i) bereits in A, da Prozess i laut Algorithmus von allen Nachbarn eine Nachricht erhalten hat, in deren A-Menge die Identität des Senders (also desNachbarn) enthalten ist.
- Man beachte ferner, dass dies nicht nur eine Invariante ist, sondern sogar *immer gilt,* da schon initial (A,B) = ($\{i\},\emptyset$) in jedem Prozess ist. □

---

(2) *Wenn* A = B in einem Prozess P_i gilt, dann ist dort A = B = {Menge *aller* Prozessidentitäten}

- Es gilt schon initial $i \in A$, folglich (da nun A=B) auch $i \in B$. Da die Mengen nie vermindert werden, bleibt i in beiden Mengen.
- Wegen obiger Invarianten (1) ist in-neighbors(i) $\subseteq$ A.
- Wegen A = B also auch in-neighbors(i) $\subseteq$ B.
- Aus der Invarianten (1) folgt entsprechend in-neighbors(in-neighbors(i)) $\subseteq$ A.
- Damit gilt induktiv $\forall$k: in-neighbors^k(i) $\subseteq$ A = B.
- Wegen des starken Zusammenhangs ist jeder Prozess in in-neighbors^j(i) für ein gewisses j. □

(3) Sei *visit* = Aktion R bei erstem Versenden der eigenen Identität in der Menge B
Bei conclude (d.h. A=B) im Initiator gilt:

a) Jeder Prozess $P_j \neq$Initiator hat $visit_j$ ausgeführt und $visit_j$ < conclude,

b) init < $visit_j$

a): Wegen (2) "kennt" der Initiator bei conclude P_j (d.h. $j \in A = B$); er kann davon nur über eine Nachrichtenkette erfahren haben, an deren Anfang P_j seine Identität j versendet hat.

b): P_j versendet seine Identität nicht spontan, sondern nur nach (indirekter) Aufforderung durch den Initiator. □

Es ist nun noch zu zeigen, dass der Initiator tatsächlich nach endlicher Zeit conclude ausführt, d.h. dass bei ihm A = B wird (liveness)!

(4) Beim Initiator wird schliesslich A=B

(Voraussetzungen: stark zusammenhängender Graph, endliche Nachrichtenlaufzeiten und Aktionsdauern etc.)

- Über jede Kante des Graphen läuft schliesslich mindestens eine Nachricht ("flooding"): Das erste Empfangen eine Nachricht von einem Nachbarn vergrössert echt die eigene Menge A --> Welle wird an alle Nachbarn weiterverteilt.
- Jeder Prozess P_i verbreitet schliesslich seine eigene Identität i an alle Nachbarn über die B-Mengen, da $B := B \cup \{i\}$ ausgeführt wird.
- Jede Identität erreicht schliesslich mittels Flooding den Initiator über die B-Mengen.
- Beim Initiator gilt so schliesslich: B = {Menge aller Prozessidentitäten}.
- $A \supseteq B$ ist eine Invariante, wie man leicht überprüft.
- Aus den letzten beiden Eigenschaften folgt, dass schliesslich A = B gilt.

□

Mit (3) und (4) ist alles gezeigt:
Finns Algorithmus ist ein Wellenalgorithmus!

- obere Schranke für Durchmesser etc. braucht nicht bekannt zu sein
- Nachrichtenkomplexität $\leq 2\,n\,e$ (wieso?)
- höhere Bit-Komplexität als z.B. der Echo-Algorithmus

Eingesetzte Beweistechniken:

- atomare Aktionen
- Invarianten
- monotone Approximation

Es gibt verschiedene Wellenalgorithmen

- Topologiespezifische, z.B. für
  - Ring
  - Baum
  - allg. Graph

  (Ring, Baum:) hierfür spezialisierte Verfahren u.U. besonders effizient

- Voraussetzungen bzgl. Knotenidentitäten
  - eindeutig oder
  - anonym

- Voraussetzungen bzgl. notwendigem "Wissen", z.B.
  - Nachbaridentitäten
  - Anzahl der Knoten (bzw. obere Schranke)
  - ...

- Voraussetzungen bzgl. Kommunikationssemantik
  - synchron, asynchron, FIFO-Kanäle, bidirektionale Kanäle...?

- Qualitätseigenschaften
  - Sequentiell oder parallel (bzw. "Parallelitätsgrad")
  - Anzahl möglicher Initiatoren (mehr als einer?)
  - Zeitkomplexität
  - Nachrichtenkomplexität (worst/average case)
  - Bitkomplexität (Länge der Nachrichten)
  - Dezentralität (kein Engpass?)
  - Symmetrie (alle lok. Algorithmen identisch?)
  - Fehlertoleranz (Fehlermodell? Grad and Fehlertoleranz?)
  - Einfachheit (--> Verifizierbar, einsichtig...)
  - Praktikabilität, Implementierbarkeit
  - Skalierbarkeit (auch für grosse Systeme geeignet?)
  - ...

Wellenalgorithmen: Zusammenfassung

- Es gibt viele Wellenalgorithmen, wir kennen u.a.:
  - Echo-Algorithmus ("Flooding mit indirektem Acknowledge")
  - Traversierung von Ringen, Gittern, Hypercubes, Sterntopologien,...
  - Paralleles Durchlaufen von (Spann)bäumen
  - Paralleles Polling auf Sternen
  - Tarry-Algorithmus, Depth-first-Traversierungen
  - Verfahren von Awerbuch und Variante von Cidon
  - Phasenalgorithmus
  - Algorithmus von Finn

- Anwendung von Wellenalgorithmen (u.a.):
  - *Broadcast*
  - *Einsammeln* von verteilten Daten ("gather")
  - Konstruktion eines *Spannbaumes*
  - *Phasensynchronisation* von Prozessen
  - *Triggern eines Ereignisses* in jedem Prozess
  - Implementierung von *Schnittlinien* (--> Schnappschuss etc.)
  - *Basisalgorithmus* für andere Verfahren (Deadlock, Terminierung,...)
  - Bestimmung des *glob. Infimums* (z.B.: "ist ein flag gesetzt?")

- Es gibt viele Wellenalgorithmen ==> welcher ist der beste?
  - Es gibt sicherlich keinen "allgemein besten" - je nach Voraussetzungen wird man nur eine Teilmenge davon in Betracht ziehen können, ferner gibt es sehr unterschiedliche Qualitätskriterien (vgl. frühere Aufzählung)!
  - *Aufgabe*: Diesbezüglicher Vergleich aller Wellenalgorithmen!

Übungen (5)

Man *vergleiche* die in der Vorlesung und den Übungen betrachteten *Wellenalgorithmen*. Dazu gehören insbesondere der Echo-Algorithmus, der Phasenalgorithmus, Finns Algorithmus, Depth-first-Traversierung (bzw. Verfahren von Tarry), Awerbuchs Verfahren, Variante von Cidon, sowie Verfahren für Graphen mit bekanntem spannendem Baum, bekanntem Hamilton'schen Zyklus und Verfahren auf Sterntopologien (bzw. auch vollständigen Graphen) mit dem Initiator als Sternmittelpunkt.

Die zu untersuchenden *Kriterien* sollen u.a. sein:
-- Nachrichtenkomplexität (worst case, average case),
-- Zeitkomplexität (worst case, average case),
-- Bitkomplexität,
-- Anwendbarkeit auf gerichtete / ungerichtete Graphen,
-- Anwendbarkeit, falls Nachrichten sich überholen können,
-- Anwendbarkeit bei anonymen Graphen,
-- Berechnung von nicht-idempotenten (assoziative, kommutativen) Operationen,
-- Kenntnis von Parametern wie Durchmesser, Nachbaridentitäten etc.

Die wichtigsten Angaben ordne man zweckmässigerweise in *Tabellenform* an. Man *diskutiere* die relativen Vor- und Nachteile. Gibt es einen besten oder schlechtesten Wellenalgorithmus? Aussagen zu den Kriterien sollten *begründet* werden, sofern sie nicht offensichtlich sind oder in der Vorlesung bewiesen wurden.