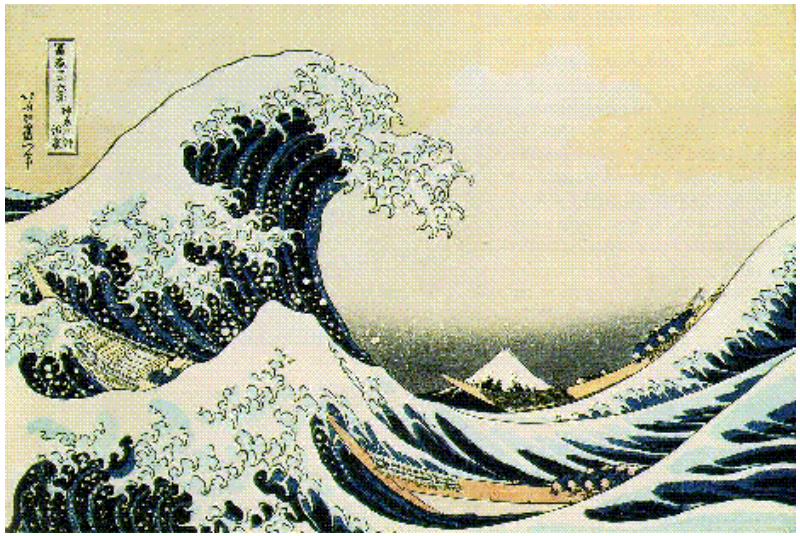
Wellenalgorithmen



Katsushika Hokusai (1760-1849): Die grosse Welle von Kanagawa, Metropolitan Museum of Art, New York

This well-known masterpiece shows Mt. Fuji behind raging waves off the seacoast. Hokusai created "Mt. Fuji Off Kanagawa" (popularly known in the West as "The Wave") as part of his subscription series, "Thirty-Six Views of Mt. Fuji," completed between 1826 and 1833. This is one of the best-known Japanese woodblock prints, and with others of this period inspired the entire French Impressionist school. By making Mt. Fujionly rather small in the background the artist expresses, in a novel way, the elemental powerof nature.

Wellenalgorithmen

Häufige Probleme bei verteilten Algorithmen / Systemen:

- *Broadcast* einer Information
- Globale *Synchronisation* zwischen Prozessen (Trennung von Phasen)
- *Triggern eines Ereignisses* in jedem Prozess
- *Einsammeln* von verteilten Daten

==> *Wellenalgorithmen*

- *Alle* Prozesse müssen sich beteiligen
- *Basisalgorithmen* ("Bausteine") für andere Algorithmen
  (wechselseitiger Ausschluss, Terminierungerkennung, Election...)

Abstraktere Definition:

(Algo. ist ggf. nicht-deterministisch!)

Wellenalgorithmus, wenn für jede seiner Berechnung gilt:

1. Berechnung enthält ein *init*-Ereignis
2. Alle Prozesse besitzen ein *visit*-Ereignis
3. Berechnung enthält ein *conclude*-Ereignis

Und es gilt folgendes für alle visit-Ereignisse:

1. init ≤ visit
2. visit ≤ conclude

} (==> init ≤ conclude)

Wellenalgorithmen (2)

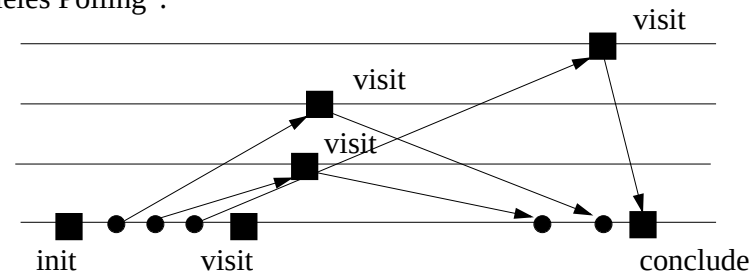
Aus 1: Über die Kausalketten lässt sich *Information* vom Initiator an alle Prozesse *verteilen*

Aus 2: - Nach conclude wurde jeder besucht (--> Terminierung!)
- Über die Kausalketten lässt sich *Information* von allen Prozessen *einsammeln*

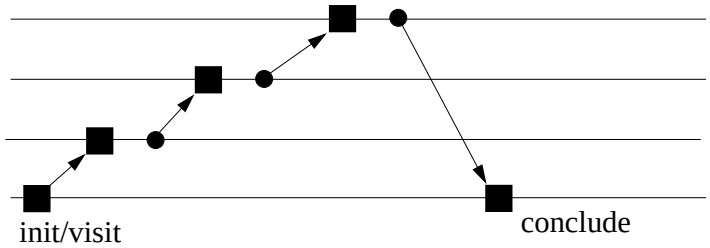
Bem.: a) init und conclude oft im selben Prozess ("Initiator")
b) init oder conclude kann mit visit verschmelzen (daher '≤' statt '<')

Beispiele:

"paralleles Polling":



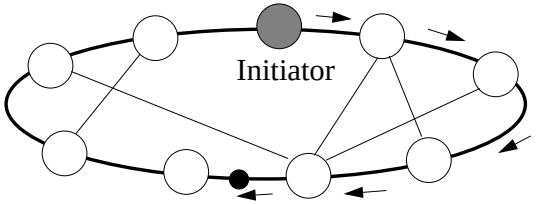
"Ring":



Einige Wellenalgorithmen

- init-, visit-, conclude-Ereignisse jeweils sinnvoll festlegen!

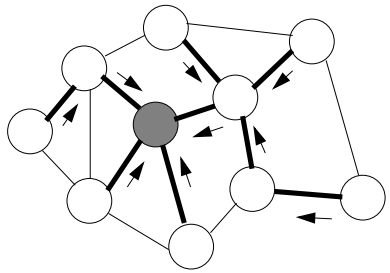
1.) *Ring* / *Hamiltonscher Zyklus* mit umlaufenden Token



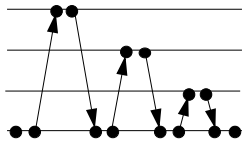
- "logischer" Ring genügt!
- in einem zusammenhängenden ungerichteten Graphen kann ein logischer Ring immer gefunden werden, indem man einen Spannbaum "umfährt"

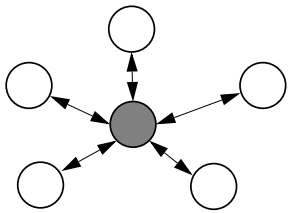
2.) *Spannbaum*

- an den Blättern reflektierte Welle (vereinfachter Echo-Algorithmus: flooding mit rek. acknowledgement)
- bei nicht-entartetem Spannbaum sind viele Nachrichten parallel unterwegs



3.) *(logischer) Stern*


sequentielles Polling



- "Polling" entweder sequentiell (jeweils höchstens eine Nachricht unterwegs) oder parallel

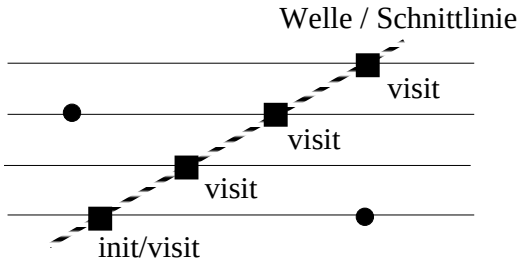
---

4.) *Echo-Algorithmus ist ein Wellenalgorithmus*

- visit-Ereignis entweder erster Erhalt eines Explorers oder Senden des Echos
- definiert so sogar zwei verschiedene "parallele" Wellen bzw. "Halbwellen"!

Wellen und Schnittlinien

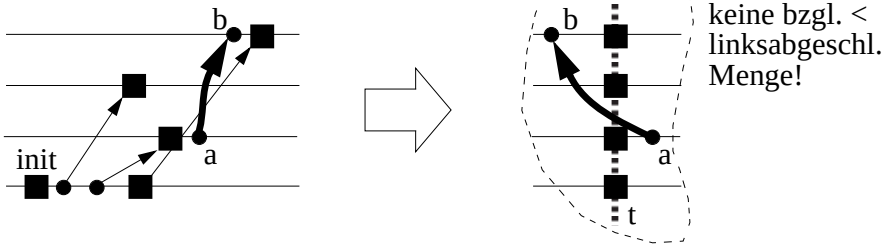
- Verbinden der visit-Ereignisse zu einer *Schnittlinie*:



Trennt die Menge der Ereignisse in *Vergangenheit* und *Zukunft*

oBdA: "gerade" Schnittlinie
(--> Gummibandtransf.)

- Falls ein Wellenalgorithmus einer anderen verteilten Anwendung überlagert wird: Lässt sich dann die Schnittlinie immer *senkrecht* zeichnen (Gummibandtransformation), so dass keine Nachricht "echt" rückwärts läuft?



keine bzgl. < linksabgeschl. Menge!

- Beispiel: zu dem dadurch festgelegten "globalen Zeitpunkt" t wäre eine Anwendungsnachricht zwar angekommen, aber noch nicht abgesendet!

  ==> die Welle würde ein unmögliches ("inkonsistentes") Bild liefern

  ==> Visit-Ereignisse lassen sich (hier) nicht als "virtuell gleichzeitg" ansehen

- Wichtige Fragen: Unter welchen Umständen definieren die visit-Ereignisse einen "Schnitt", der als *senkrechte* Linie aufgefasst werden kann? Lässt sich das erzwingen?
  (damit hätten wir so etwas wie globale Zeit --> später)

Eigenschaften von Wellenalgorithmen

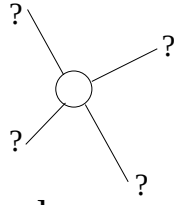
wodurch ist eigentlich der suggestive Name gerechtfertigt?

Satz: Es werden *mindestens n-1 Nachrichten* benötigt

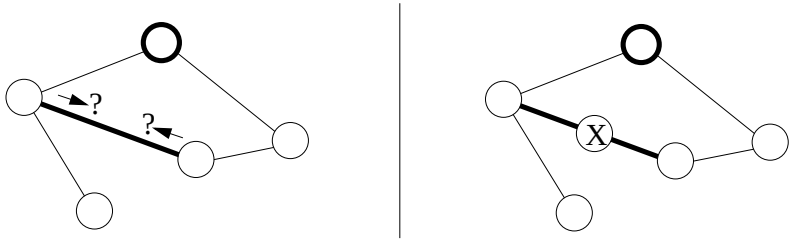
- Begründung: Da jedes visit-Ereignis vom init-Ereignis kausal abhängig ist, muss jeder andere Prozess mindestens eine Nachricht empfangen

- mindestens n Nachrichten, falls conclude und init auf gleichem Prozess stattfinden (dann muss auch der Initiator eine Nachricht empfangen)

Satz: Ohne Kenntnis der Nachbaridentitäten werden *mindestens e Nachrichten* benötigt



Bew.: Über jede Kante muss eine Nachricht gesendet werden, wenn die Identität der Nachbarn unbekannt ist:



- würde man im linken Szenario die Kante nicht durchlaufen, dann würde im rechten Szenario Knoten X nicht erreicht und er könnte kein von init abhängiges visit-Ereignis ausführen

Wellenalgorithmen und Spannbäume

Satz: Die Kanten, über die ein Nicht-Initiator erstmals eine Nachricht erhielt, bilden einen *spannenden Baum*

(Voraussetzung: der Algorithmus wird an einer einzigen Stelle initiiert)

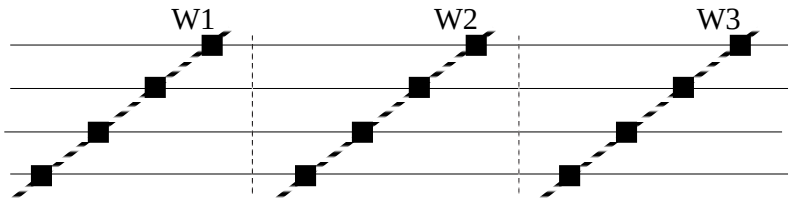
Beweis:

- Jeder Nicht-Initiator erhält mindestens eine Nachricht
- Es gibt also n-1 solche ersten Kanten
- Wir müssten noch einsehen:
  - die Menge der entsprechenden Kanten ist zyklenfrei,
  - *oder* die Menge dieser Kanten ist zusammenhängend

  } Denkübung: wieso?
- Damit und mit n-1 folgt die Baumeigenschaft

- man überlege sich, wieso der Beweis falsch wird, wenn ein Wellenalgorithmus an mehreren Stellen unabhängig initiiert wird!
- wir kennen solche Spannbäume bereits vom Echo-Algorithmus!

Folgen von Wellen



- Wellennummer j kann mit jeder Nachricht mitgeführt werden.

Start:

```
j := 1 (* Wellennummer *)
x := lokale Berechnung
init(j)  (* ggf. broadcast von x *)
```

j-te Iteration

Ersetze conclude durch:

```
j := j+1
if (* noch nicht fertig *)
   x := lokale Berechnung
   init(j)  (* ggf. broadcast von x *)
fi
```

Satz: Zu jedem globalen Zeitpunkt unterscheiden sich die Wellennummern zweier Prozesse um höchstens 1

Beachte zum Beweis den Zusammenhang zwischen Zeit und Kausalität

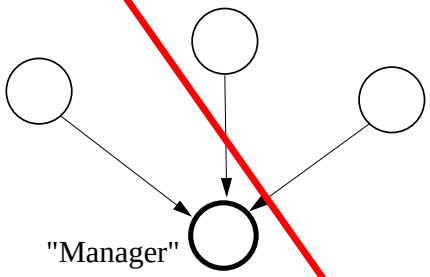
==> Phasensynchronisation der Prozesse möglich

Bem.: Verschiedene unabhängige Wellen (ggf. verschiedener Initiatoren) können sich in transparenter Weise "parallel" überlagern, wenn eindeutige Identitäten mitgeführt werden

Halbwellen

- Abschwächung des Wellenbegriffs, z.B. *kein init*:

--> *kontrahierende* Halbwelle



- Bsp.: Prozesse melden unaufgefordert ihre einzusammelnden Werte einem zentralen Prozess (*Stern*) bzw. allen Prozessen (*vollst. Graph*)

- Analog auch für *Bäume*, wo die Blätter "spontan" ihr Ergebnis weiter nach innen melden

- Manager bzw. Wurzel des Baumes kann conclude durchführen, wenn alle Nachbarknoten geantwortet haben (also ihr visit ausgeführt haben)

- Andere Auffassung: es gibt kein *eindeutiges* init, sondern i.a. mehrere unabhängige (--> "multisource-Algorithmus")

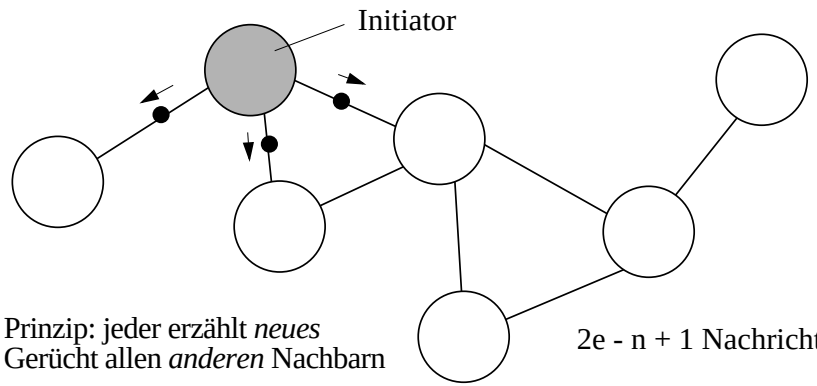
- Entsprechend: *kein conclude*: --> *expandierende* Halbwelle

- Es gibt keinen Prozess, der von der *Terminierung* des Algorithmus erfährt
- Beispiel: *flooding*-Algorithmus (entspricht erster Halbwelle des Echo-Algorithmus)

Der flooding-Algorithmus als Halbwelle

- Algorithmus kennen wir bereits!

Welle ohne conclude



Prinzip: jeder erzählt *neues* Gerücht allen *anderen* Nachbarn

2e - n + 1 Nachrichten

visit:

```
V:  {Eine Informationsnachricht kommt an}
    if not informed then
      informed := true;
      send <info> to all other neighbors;
    fi
```

informed = false

init:

```
I:  {not informed}
    informed := true;
    send <info> to all neighbors;
```

Aktion I nur beim einzigen Initiator

- auch für solche expandierenden Halbwellen gilt (bei einem einzigen Initiator), dass die "ersten Kanten" einen *spannenden Wurzelbaum* bilden
- diesen Baum kann man benutzen, um (entsprechend einer kontrahierenden Halbwelle) *Acknowledgements* zum Initiator zu senden (dazu müssen die Blätter des Baumes aber erkennen, dass sie Blätter sind - wie geht das?)
- man erhält nach dieser Idee den *Echo-Algorithmus*!

Virtuell gleichzeitiges Markieren

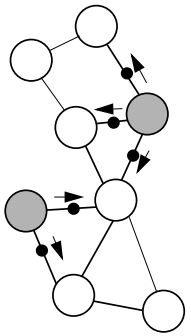
- Als Anwendung eines Halbwellenalgorithmus (≈ flooding)
- Voraussetzung hier: FIFO-Kanäle (d.h. keine Überholungen)

```
I:  {not marked}
    marked := true; (* = init = visit *)
    send <marker> to all neighbors;
```

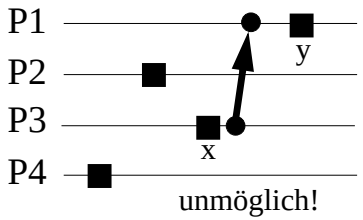
```
V:  {Eine Marker-Nachricht kommt an}
    if not marked then
       marked := true; (* = visit *)
       send <marker> to all neighbors;
    fi
```

!

mehrere Initiatoren sind zulässig!

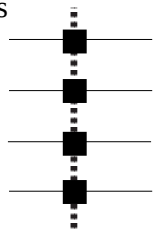


- Über *jede* Kante läuft in beide Richtungen ein Marker
- Eine andere Nachricht ("Basisnachricht"), die von einem markierten Knoten gesendet wird, kommt erst dann an, wenn der Empfänger auch bereits markiert ist



unmöglich!

Es ist daher stets rechts nebenstehende *Sicht* als Gummibandtransformation *möglich*, wo Basisnachrichten nicht rückwärts verlaufen und alle visits simultan sind (*wieso?*)

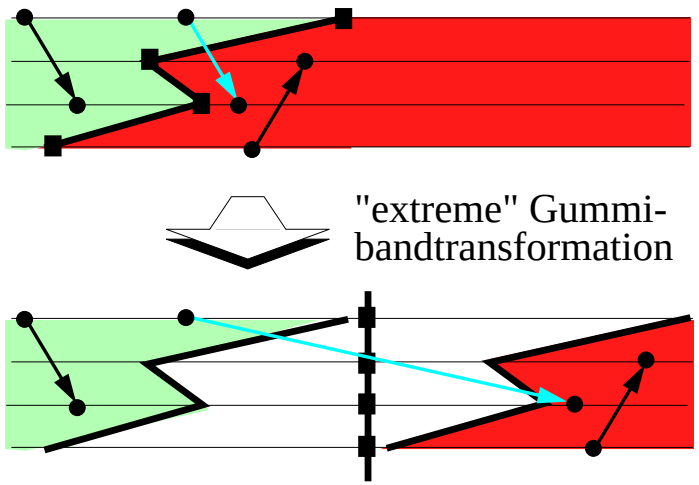


Eine "rückwärts" über den Schnitt laufende Basisnachricht kann es nicht geben: Wenn P3 und P1 benachbart sind, dann wurde bei x ein Marker an P1 gesendet, der nicht (wie von der gezeichneten Nachricht) überholt werden kann

Fragen:
- geht virtuell gleichzeitiges Markieren auch ohne FIFO-Kanäle? (FIFO abschwächen, FIFO erzwingen, FIFO "simulieren"...)
- geht das auch mit weniger als 2e Nachrichten?

Vertikale Schnittlinien?

- Voraussetzung: Keine Nachricht läuft von der "Zukunft" in die "Vergangenheit" einer Schnittlinie
  - solche Schnittlinien heissen *konsistent* (linke Hälfte ist dann linksabgeschlossen bzgl. der Kausalrelation, d.h. linke Hälfte ist eine Präfixberechnung)
- Dann lässt sich diese Schnittlinie *vertikal* zeichnen, ohne dass dabei Nachrichten von rechts nach links laufen
  - als hätte die zugehörige Welle alle Prozesse gleichzeitig besucht!
  - offenbar nützlich für Terminierungserkennung, Sicherungspunkte...!
- "Konstruktiver" Beweis: Auseinanderschneiden entlang der Schnittlinie und den rechten Teil "ganz" über den weitesten rechts liegenden Punkt des linken Teils hinausschieben



"extreme" Gummibandtransformation

- Zerschnittene Nachrichten reparieren ("verlängern")
- Schnittereignisse in die Lücke senkrecht untereinander legen

Anwendung von Wellenalgorithmen

- Als "Unterprogramm" innerhalb anderer Anwendungen, z.B.
  - Broadcast einer Information an alle Prozesse
  - Einsammeln verteilter Daten
  - Traversieren aller Prozese eines Prozessgraphen
  - Berechnung einer globalen Funktion, deren Parameter auf alle Prozesse verteilt sind (z.B. globales Minimum)
  - ...

- Abstrakt: um Schnitte durch ein Zeitdiagramm zu legen
  (Problem jedoch ggf.: Konsistenz des Schnittes feststellen / erzwingen)
  - Terminierungserkennung
  - Schnappschuss
  - ...

- Wellenalgorithmen sind typische *Basisalgorithmen*
  - verrichten *Dienste* einer niedrigeren Schicht
  - sind *Bausteine* für komplexere Verfahren
  - bestimmen oft qualitative Eigenschaften wie Nachrichten- oder Zeitkomplexität der aufgesetzten Verfahren entscheidend mit
  - lassen sich (sofern die Rahmenbedingungen stimmen) gegeneinander austauschen

- Es gibt viele verschiedene Wellenalgorithmen
  - für die verschiedensten Topologien
  - mit unterschiedlichen Voraussetzungen
  - mit unterschiedlichen Qualitätseigenschaften

Sequentielle Traversierungsverfahren

- Unterklasse der Wellenalgorithmen (auf ungerichteten und zusammenhängenden Graphen) mit:
  - einem einzigen Initiator (bei dem init und conclude stattfindet)
  - *totaler Ordnung* auf allen visit-Ereignissen

- Interpretation: Ein *Token* wandert durch alle Prozesse und kehrt zum Initiator zurück
  - Visit-Ereignis: erstmalige Ankunft (oder Weiterreichen) des Tokens

- Relativ hohe Zeitkomplexität, da keine wesentliche Parallelität

---

- Bekannte einfache Verfahren für *spezielle Topologien*:
  - *Stern* (bzw. vollst. Graph): sequentielles "polling"
  - *Ring*
  - *Baum* (bzw. Spannbaum)
  - n-dimensionales *Gitter*: Verallgemeinerung des "ebenenweise" Durchlaufens eines 3-dimensionale Gitters (einen Schritt in der Dimension k+1, wenn k-dimensionales Untergitter durchlaufen...)
  - n-dimensionale *Hyperwürfel*: Traversiere ("rekursiv") einen n-1-dim. Hyperwürfel bis auf den letzten Schritt, gehe in Richtung der n-ten Dimension und traversiere dort den n-1-dim. Hyperwürfel...
  - Hamiltonsche Graphen (z.B. auch Ring und Hyperwürfel als Sonderfall)

NOUVELLES ANNALES
DE
MATHÉMATIQUES
JOURNAL DES CANDIDATS
AUX ÉCOLES SPÉCIALES, A LA LICENCE ET A L'AGRÉGATION,
RÉDIGÉ PAR
M. Ch. BRISSE,
ET
M. E. ROUCHÉ,
Publication fondée en 1842 par MM. Gerono et Terquem,
et continuée par MM. Gerono, Prouhet, Bourget et Brisse.
TROISIÈME SÉRIE.
TOME QUATORZIÈME.
PARIS,
GAUTHIER-VILLARS ET FILS, IMPRIMEURS-LIBRAIRES
DU BUREAU DES LONGITUDES, DE L'ÉCOLE POLYTECHNIQUE,
Quai des Grands-Augustins, 55.
1895

LE PROBLÈME DES LABYRINTHES;

PAR M. G. TARRY.

Tout labyrinthe peut être parcouru en une seule course, en passant deux fois en sens contraire par chacune des allées, sans qu'il soit nécessaire d'en connaître le plan.

Pour résoudre ce problème, il suffit d'observer cette règle unique :

Ne reprendre l'allée initiale qui a conduit à un carrefour pour la première fois que lorsqu'on ne peut pas faire autrement.

. . .

En suivant cette marche pratique, un voyageur perdu dans un labyrinthe ou dans des catacombes, retrouvera forcément l'entrée avant d'avoir parcouru toutes les allées et sans passer plus de deux fois par la même allée.

Ce qui démontre qu'un labyrinthe n'est jamais inextricable, et que le meilleur fil d'Ariane est le fil du raisonnement.