Karlsruher Institut für Technologie

Tutorium Algorithmen 1

05 · Hashing, Graphen · 27.5.2024

Peter Bohner Tutorium 3

Hashtabellen

Karlsruher Institut für Technologie

Hashtabelle

- Ungeordnete Datenstruktur (Kein index $A[i]$)
- Drei Operationen: **insert**, **remove**, **find** möglichst in konstanter Laufzeit

- Menge M von Elementen, jedem Element eindeutiger **Key** zugeordnet
- Speichern in **Hash-Tabelle** (Array) der Größe m
- Über **Hash-Funktion** wird jedem Key ein Index (Bucket) in der Tabelle zugeordnet
- Wir brauchen also:
  - $key : M \to \mathcal{U}$ (Menge der möglichen Keys (meistens $\mathbb{Z}_n$)) injektiv
  - Hashtabelle t: Array$[0 \ldots m-1]$
  - Hash-Funktion $h : \mathcal{U} \to \{0, \ldots, m-1\}$
    Element e landet in $t[h(key(e))]$

$\mathcal{U}$ wird auch Universum der möglichen Schlüssel genannt

Operationen

- **insert**(e: Element): Fügt Element e in Datenstruktur ein
  - Berechnet key(e) und fügt e in h(key(e)) in die Hash-Tabelle ein
- **remove**(k: Key): Entfernt Element e mit key(e) = k aus der Datenstruktur und gibt dieses zurück
- **find**(k: Key): Sucht Element e mit key(e) = k in der Hash-Tabelle an Index h(k) und gibt dieses falls vorhanden zurück

removeund **find** könnte man auch mit e: Element als input definieren, was müsste man da ändern?

Hashfunktionen

- Funktion die einen großen Definitionsbereich auf einen kleinen Wertebereich abbildet
- Am besten Einwegfunktion (Von x auf h(x) kommen ist einfach, umgekehrt (fast) unmöglich)

Motivation: Hashing von Tutanden

h(tutand) = (Zahlenrepräsentation des 1 Buchstaben des Namens (A=0) +
Letzte Ziffer der Mtrk. Nummer) mod 10

- Findet euren Hashwert
- Ordnet euch entsprechend eurem Hashwert an (oder meldet euch)

Was ist euch aufgefallen?

Kollisionen

Bei einer Hashfunktion redet man von einer Kollision wenn:

- $x \neq y \wedge h(x) = h(y)$
- Also: mehrere Elemente auf gleichen Index abgebildet $\Rightarrow$Kollision
- Mehrere Möglichkeiten der Kollisionsauflösung, Anzahl sollte trotzdem minimiert werden

Keine Hash-Funktion ist im worst case gut, es wird immer Eingaben mit vielen Kollisionen geben

Aufgabe

Seien $m, M \in \mathbb{N}$ und $m \ll M$.
Finde für jede dieser „Hashfunktionen“ eine Folge von m Eingaben aus dem Universum $\mathcal{U} := \mathbb{Z}_M$,
sodass für eine Hashtabelle der Größe m mit dieser Hashfunktion
das sukzessive Einfügen dieser Eingaben
pro Eingabe Kollisionen linear in der Anzahl bisher hinzugefügter Elemente hat.

- $h(x) = 3 \cdot x + 5 \mod m$ $\quad (m \cdot i)_{i \in \mathbb{Z}_m}$
- $h(x) = (42 \cdot x + 17) \mod 127 \mod m$ $\quad (127 \cdot i)_{i \in \mathbb{Z}_m}$
- $h(x) = \lfloor x \cdot \frac{m}{M} \rfloor$ $\quad (i)_{i \in \mathbb{Z}_m}$
- $h(x) = x \cdot \lfloor \frac{m}{2} \rfloor \mod m$ $\quad (m \cdot i)_{i \in \mathbb{Z}_m}$

Kollisionsauflösung

- An jedem Index der Hash-Tabelle wird einfach verkettete Liste angelegt
- Element immer am Anfang der Liste eingefügt
- Bei find/remove Liste durchlaufen, bis Element mit key k gefunden wird

Laufzeiten:

insert: $\mathcal{O}(1)$

find: $\mathcal{O}$(Listenlänge) (worst-case $\mathcal{O}$(n), erwartet $\mathcal{O}(1)$)

remove: $\mathcal{O}$(Listenlänge) (worst-case $\mathcal{O}$(n), erwartet $\mathcal{O}(1)$)

Aufgabe

Gegeben ist eine Hash-Tabelle T der Größe 11, die intern verkettete Listen benutzt. Die Hash-Funktion h sei definiert als h(x) = x mod 11.
Fügt nacheinander 22, 18, 3, 80, 20, 47, 39, 55, 23, 41, 105 in die Hash-Tabelle ein (hierbei gilt key(e) = e).

55, 22	23		47, 80, 3			105, 39	18	41	20	

Simple Unfiorm Hashing Assumption

Simple Uniform Hashing Assumption

- jeder Schlüssel landet in jedem der Buckets mit der selben Wahrscheinlichkeit
- unabhängig von zuvor eingefügten Schlüsseln

$\Rightarrow$die erwartete Anzahl der Kollisionen einer Hashtabelle der größe m in die k Elemente eingefügt werden liegt in $\mathcal{O}(1)$ wenn $k \in \mathcal{O}(m)$

Sehr Optimistisch aber lässt sich gut in Beweisen verwenden

Wachsende Hashtabellen

- Was passiert wenn wir deutlich mehr als m Elemente einfügen? (z.B. Quadtratisch viele)
- Laufzeiten nicht mehr in $\mathcal{O}(1)$
- Wie bei dynamischen Arrays können wir die Größe der Hashtabelle verdoppeln
  - Dafür wird eine neue Hashfunktion gewählt und alle Elemente aus der alten Tabelle in die neue Eingefügt (rehashing)

$\Rightarrow$Laufzeiten erwartet und amortisiert in $\mathcal{O}(1)$

Universelle Familie

- bisher: Hashfunktion ist einfach gut (gleichverteilt)

Universelle Familie
Sei $\mathcal{H}$ eine Menge von Hashfunktionen. H ist eine **universelle Familie**, wenn für alle $x_1, x_2 \in U$ mit $x_1 \neq x_2$ und ein zufälliges $h \in H$ gilt, dass $\mathbb{P}(h(x_1) = h(x_2)) \leq \frac{1}{m}$.

$\Rightarrow$ „Nicht jede Funktion der Familie ist immer gut, aber im Schnitt sind sie für jedes Paar gut“

Zeige das $\mathcal{H} := \{h : x \mapsto \lfloor \sin(x + a) \cdot m \rfloor \mod m | a \in \mathcal{U}\}$ keine Universelle Familie ist.

Wähle $x_1 =$ beliebig, $x_2 = x_1 + 2\pi$, Sei $h \in H$ beliebig. Dann gilt:

$h(x_2) = \lfloor \sin(x_1 + 2\pi + a) \cdot m \rfloor \mod m = \lfloor \sin(x_1 + a) \cdot m \rfloor \mod m = h(x_1)$

$\Rightarrow \mathbb{P}(h(x_1) = h(x_2)) = 1 > \frac{1}{m}, m > 1 \Rightarrow$ nicht universell

$U = \mathbb{R}_+$(Universum) und m (Größe der Hashtabelle) beliebig, aber konstant

Aufgabe

Zeigt das die folgenden Menge keine Universellen Familie ist:

- $\mathcal{H} := \{h : 2x \mapsto x \mod m\}$

Wähle $x_1 =$ beliebig, $x_2 = x_1 + 2m$

$h(x_2) = 2(x_1 + 2m) \mod m = 2x_1 + 4m \mod m = 2x_1 \mod m = h(x_1)$

$\Rightarrow \mathbb{P}(h(x_1) = h(x_2)) = 1 > \frac{1}{m}, m > 1 \Rightarrow$ nicht universell

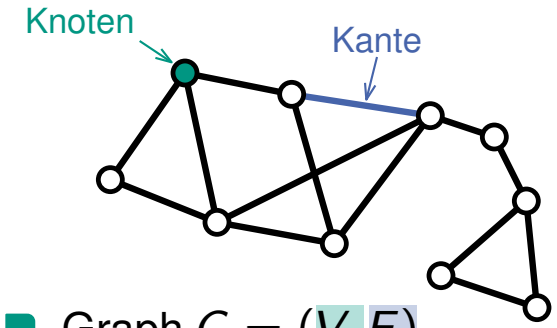
Aufgabe

Gegeben sei ein Array A mit n Ganzzahlen, in A sollen Duplikate gefunden und eliminiert werden. Die Ausgabe soll eine Liste B sein, das jedes Element aus A nur einmal enthält. Gebt einen Algorithmus mit erwarteter Laufzeit $\mathcal{O}(n)$ an.

- erwartet $\Rightarrow$Hashing
- Iteriere über alle Elemente aus A
- Prüfe ob Element bereits in der Hashtabelle (mit **find**)
  - Falls ja: überspringen
  - Falls nein: in die Hashtabelle und in B einfügen

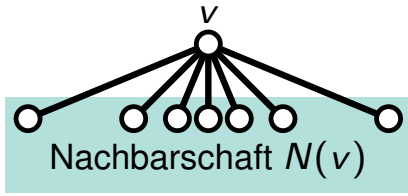
Graphen: Grundbegriffe

Knoten & Kanten

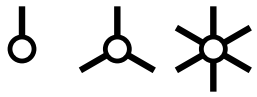


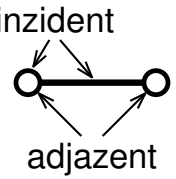
- Graph $G = (V, E)$
- $|V| = n$, $|E| = m$

Nachbarschaft

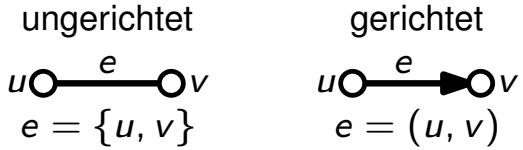


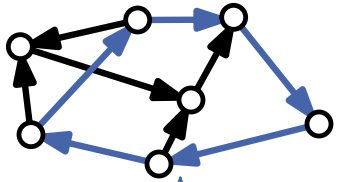
Knotengrad





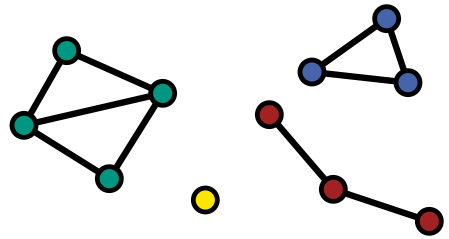
Gerichtete Graphen





gerichteter Kreis (Zyklus)

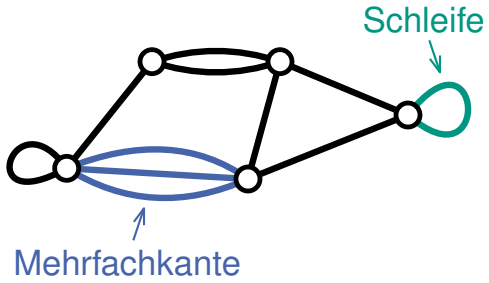
Komponenten



- unzusammenhängend
- 4 Zusammenhangskomp.

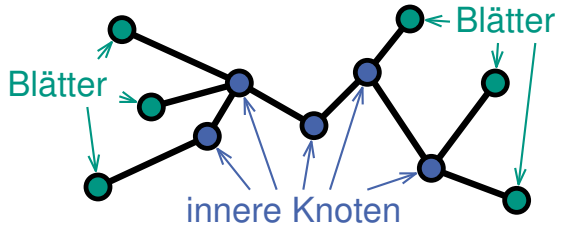
Einfache Graphen

- keine Schleifen
- keine Mehrfachkanten



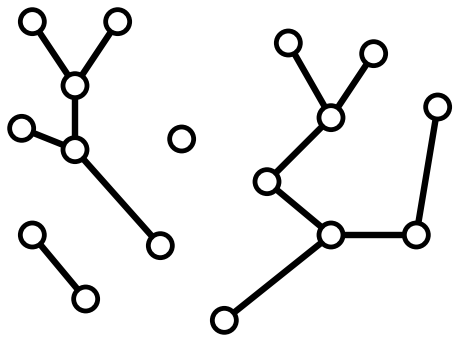
Baum

- kreisfrei
- zusammenhängend

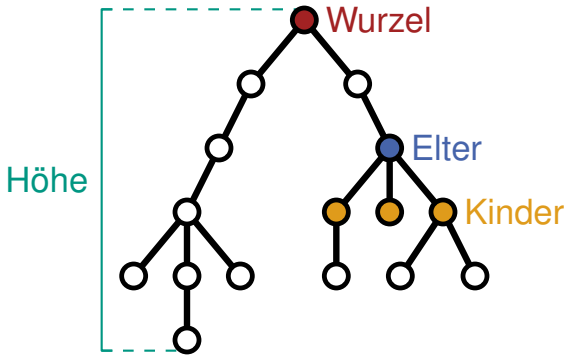


Wald

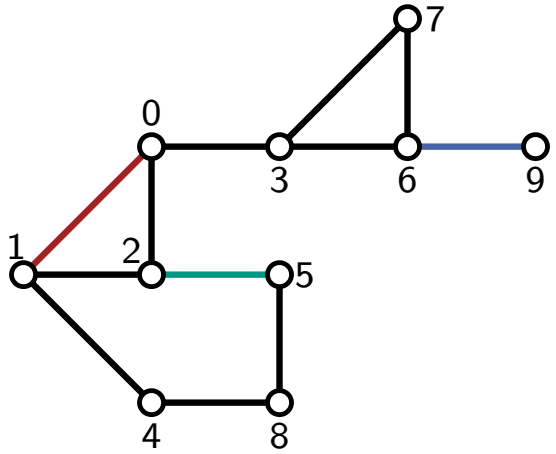
- kreisfrei



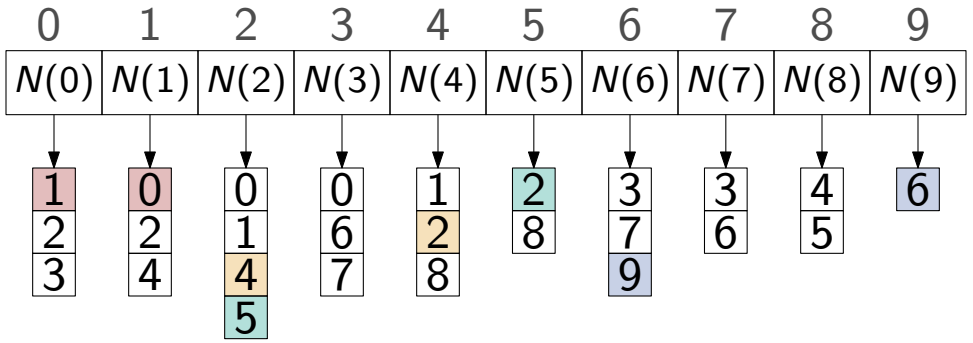
Gewurzelter Baum



Graphrepräsentationen



Adjazenzliste



Adjazenzmatrix

	0	1	2	3	4	5	6	7	8	9
0	0	1	1	1	0	0	0	0	0	0
1	1	0	1	0	1	0	0	0	0	0
2	1	1	0	0	1	1	0	0	0	0
3	1	0	0	0	0	0	1	1	0	0
4	0	1	1	0	0	0	0	0	1	0
5	0	0	1	0	0	0	0	0	1	0
6	0	0	0	1	0	0	0	1	0	1
7	0	0	0	1	0	0	1	0	0	0
8	0	0	0	0	1	1	0	0	0	0
9	0	0	0	0	0	0	1	0	0	0

Graphrepräsentationen

Wie (schnell) überprüft man in einer Adjazenzliste und wie (schnell) in einer Adjazenzmatrix:

- ob der Graph eine bestimmte Kante enthält?
- ob der Graph ungerichtet ist?
- ob der Graph schleifenfrei ist?
- ob für gegebenes $c \in \mathbb{N}$ der maximale (Ausgangs-)Grad $\leq c$ ist?

Graphenbeweise

Karlsruher Institut für Technologie

Tipps:

Beweise, dass..

- Für alle Graphen/Bäume gilt
  - Widerspruchsbeweis
  - Vollständige oder strukturelle Induktion
- Alle Aussagen äquivalent sind
  - Ringschluss

Beweise, dass nicht..

- Gegenbeispiel
- Angenommen diese Eigenschaft gilt ... Widerspruch!

Aufgabe

Sei $G = (V, E)$ ein ungerichteter Graph
Zeige, dass die Anzahl der Knoten mit ungeradem Knotengrad gerade ist.

- Hinweis 1: Was weißt du über die Summe aller Knotengrade? ist gerade
- Hinweis 2: . . . und über die Summe aller geraden Knotengrade? ist gerade
- Hinweis 3: Was folgt daraus für die Summe aller ungeraden Knotengrade? ist gerade

$\Rightarrow$Also muss die Anzahl an ungeraden Knoten gerade sein.

Aufgabe

Zeige: Ein Graph $G = (V, E)$ ist ein Baum gdw. für alle $(u, v) \in V^2$ genau ein Pfad von u nach v existiert.

Hinweis: Definition Baum: zusammenhängender und kreisfreier Graph

„$\Rightarrow$“

- Gibt es keinen Pfad zwischen zwei Knoten $u, v \in V$, so ist G nicht zusammenhängend $\Rightarrow$ kein Baum
- Gibt es zwei Pfade zwischen zwei Knoten $u, v \in V$, so ist G nicht kreisfrei $\Rightarrow$ kein Baum

„$\Leftarrow$“

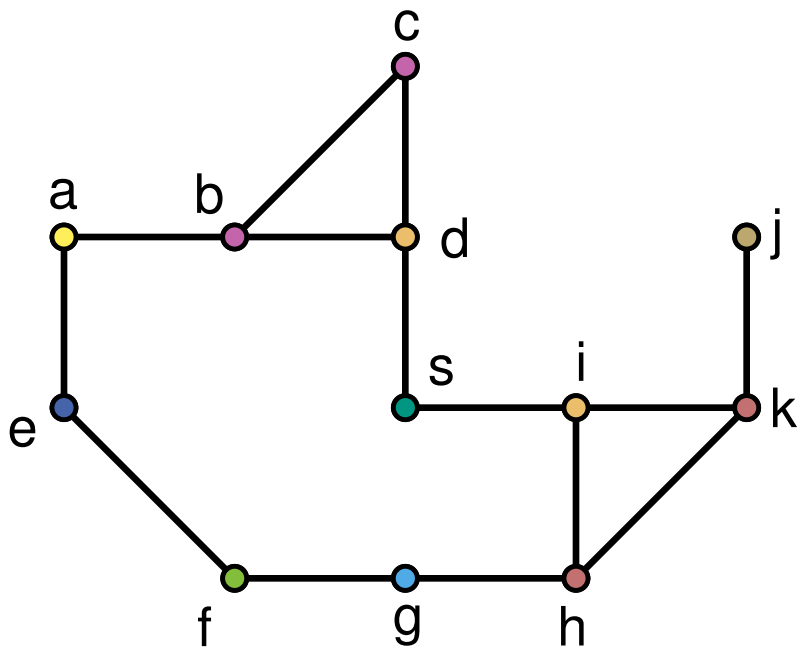
- G ist zusammenhängend
- Angenommen G enthält Kreis. Dann ex. $(u, v) \in V^2, u \neq v$ mit zwei Pfaden von u nach v. Widerspruch $\Rightarrow$ G ist kreisfrei.

$\Rightarrow G$ ist Baum

BFS

- Erkunde den Graphen Schicht für Schicht
- Startknoten erste Schicht, adjazente Knoten zweite usw.

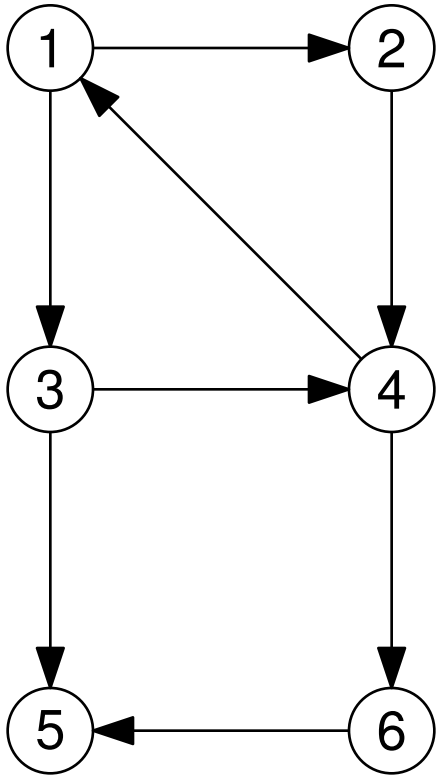
```
BFS(Graph G, Node s):
    Queue Q := {s}
    färbe s
    while ¬ Q.empty() do
        Node u := Q.dequeue()
        for v ∈ N(u) do
            if v ungefärbt then
                Q.enqueue(v)
                färbe v
```



Q: s d i b c h k a g j e f

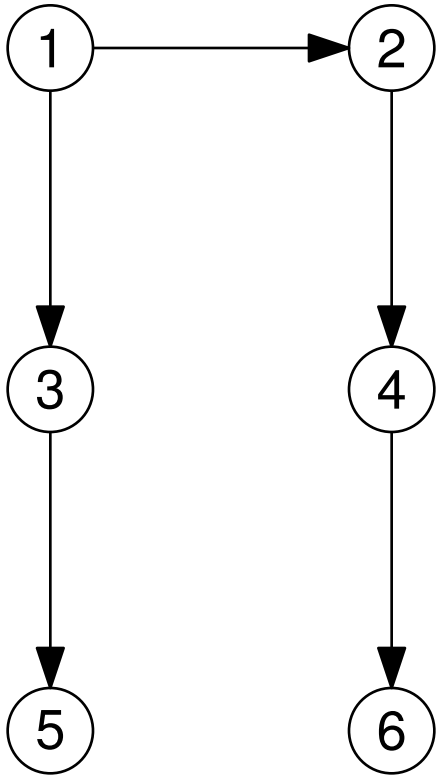
- Laufzeit in $\Theta(m)$
- Wenn wir uns in jedem Schritt merken von welchem Knoten wir einen anderen entdecken bekommen wir den sog. BFS Baum.
  Der BFS Baum enthält die kürzesten Wege zwischen s und allen anderen Knoten

Aufgabe



Führt auf dem Graphen Breitensuche ausgehend von Knoten 1 aus und gebt den Graph an der ensteht wenn alle Kanten die in der BFS nicht genommen werden entfernt werden.

Lösung



Aufgabe

Sei G = (V,E) ein ungerichteter Graph
Beschreibt einen Algorithmus der erkennt ob:

- G Zusammenhängend ist
- ein gegebener zusammenhängender Graph kreisfrei ist
- G ein Baum ist

Zusammenhängend:

- Führe von beliebigem Knoten BFS aus
- G ist zusammenhängend $\Leftrightarrow$ am Ende sind alle Knoten markiert

Kreisfrei:

- Führe von beliebigem Knoten BFS aus
- Für jeden Knoten: speichere Vorgänger (den Knoten, von dem aus er gefunden wurde)
- Falls ein schon gefärbter Knoten gefunden wird und dies nicht der Vorgänger ist: Kreis gefunden

Baum:

- Erst Zusammenhang prüfen, dann Kreisfreiheit

Aufgabe

Gegeben ein Labyrinth aus n Zimmern, die durch m Gänge verbunden sind. In den Zimmern $a_i, \ldots, a_k$ befindet sich je eine Person, die alle gleich schnell von einem Zimmer zum nächsten gehen können. Der einzige Ausgang befindet sich im Zimmer 1.

Beschreibe einen Algo, der in $\mathcal{O}(n+m)$ die Person mit dem kürzesten Weg zum Ausgang findet.

- Führe eine BFS startend im Knoten 1 durch
- Die gesuchte Person befindet sich im ersten Knoten $a_1, \ldots, a_k$, der von der BFS besucht wird.

Funktioniert das auch, wenn Gänge nur in einer Richtung passierbar sind?

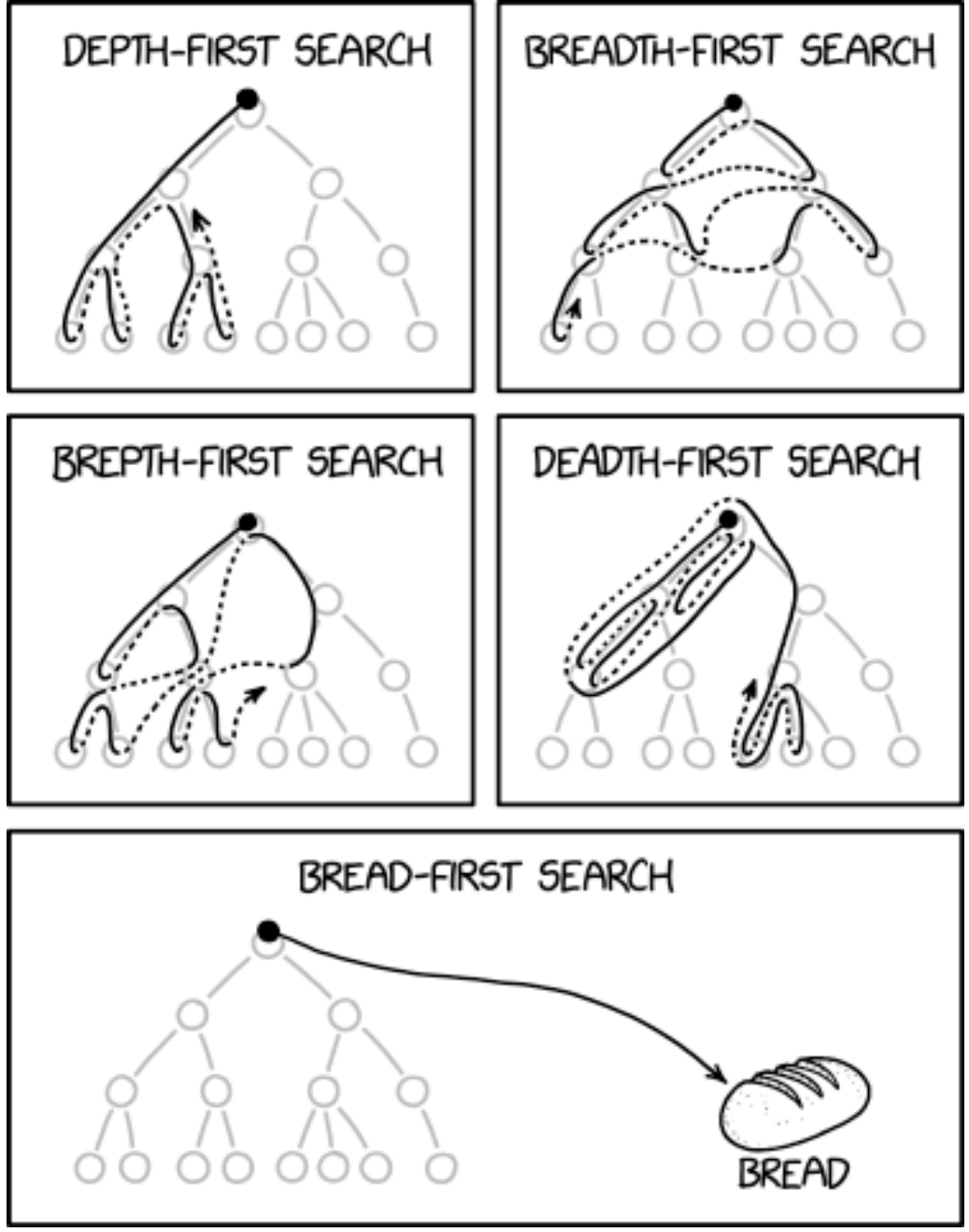
- Ja, wir müssen nur erst alle Kanten umdrehen

Fragen?

Karlsruher Institut für Technologie

Fragen!

Ende



https://xkcd.com/2407/