INNOQ Technology Day / Das Internet / 13.11.2023

Von Dijkstra, Floyd und Elefanten in Höhlen

Wie man Graphenalgorithmen umsetzt und verbessert

INNOQ

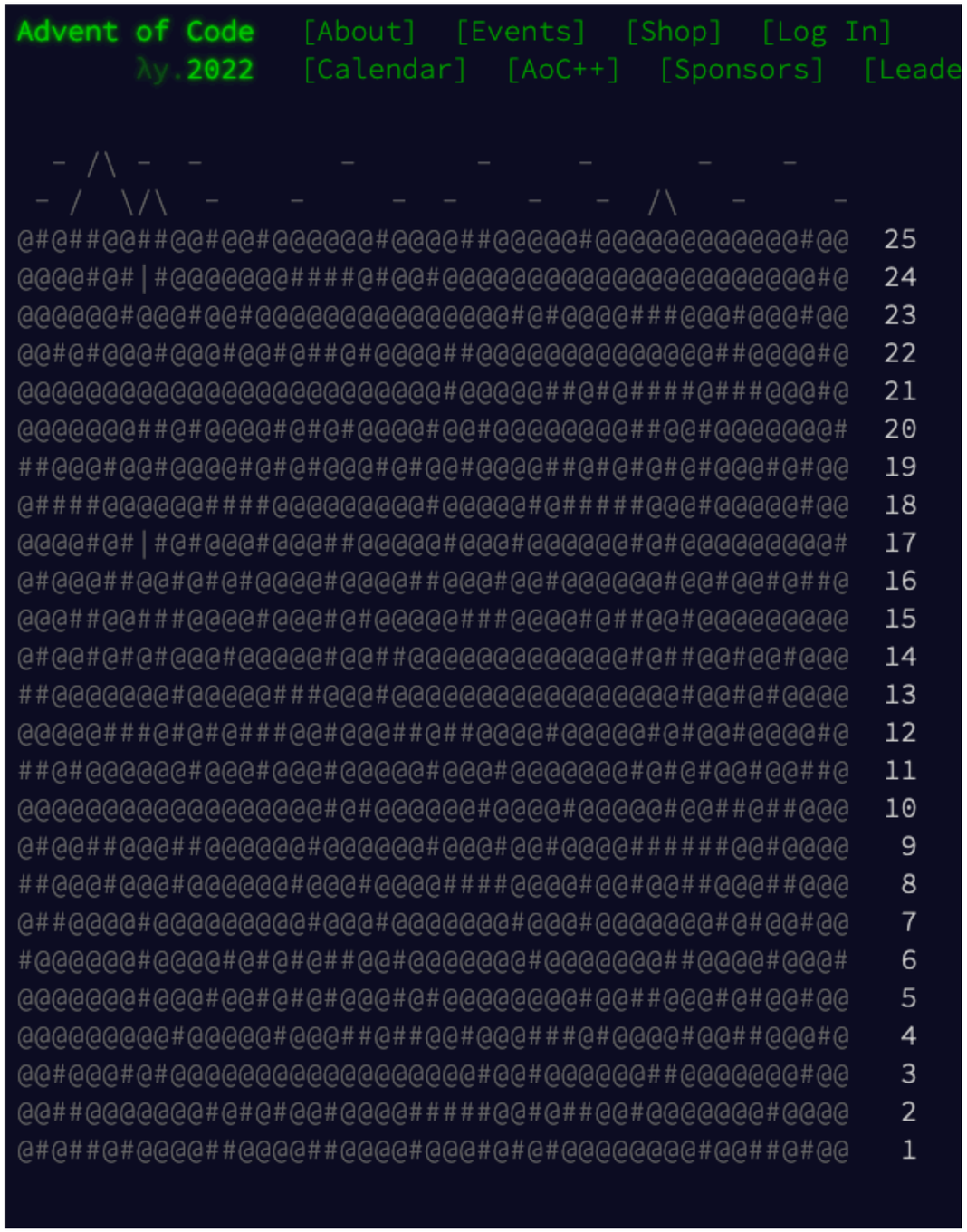
ISABEL WINGEN
CONSULTANT

Graphenalgorithmen

- Grundlegender Teil der Informatiker
- Begegnen einem im beruflichen Kontext selten

Advent of Code

- Von Eric Wastl gehostet und erstellt
- Programmierrätsel in der Adventszeit
  - 25 Rätsel mit zwei Teilen
- Eingebettet in absurde Story
- Interessante Problemstellungen



Advent of Code

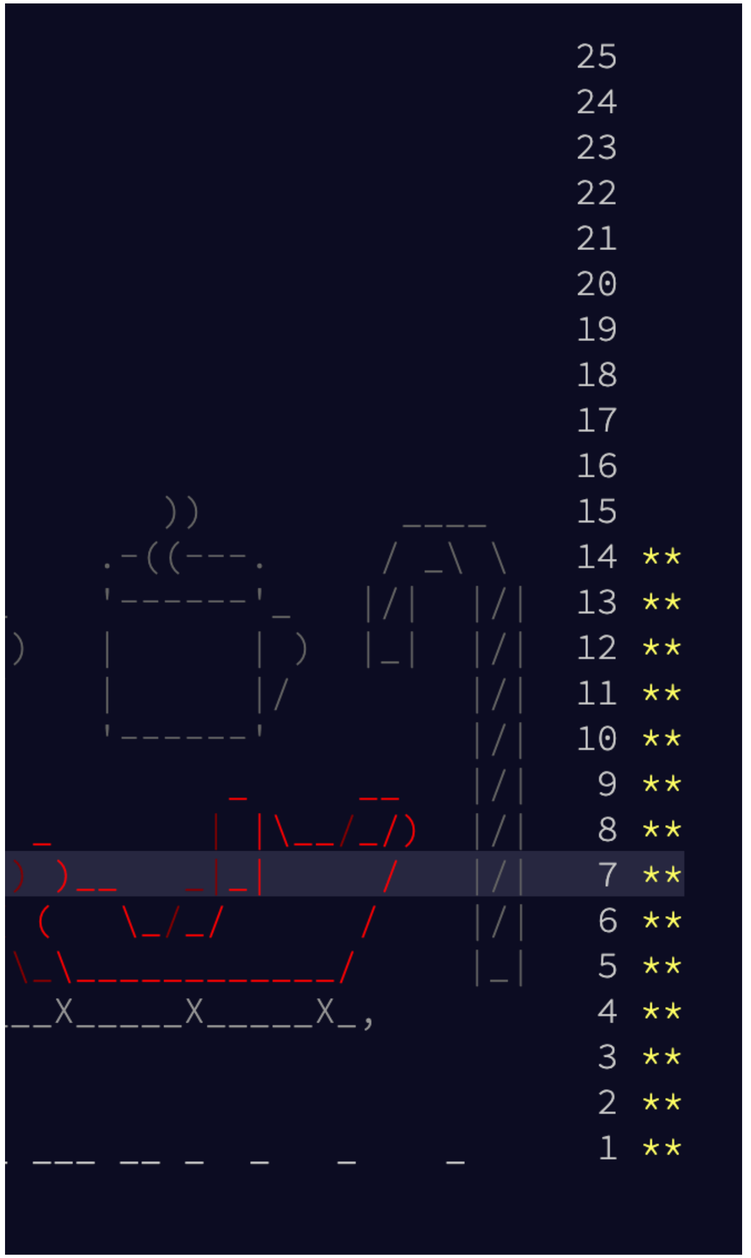
- Problem muss verstanden werden
- Algorithmus muss gefunden und implementieren
- Geschicktes Programmieren, Brute Force nicht möglich
- Dümmstmögliche Lösung muss weiterentwickelt werden
- Subreddit /r/adventofcode hilft sehr, wenn man nicht weiterkommt

SPOILERWARNUNG

Es werden die Lösungen von Advent of Code **Tag 19, 2022** und **Tag 16, 2022** besprochen

Advent Of Code

- Selber Challenges setzen
  - Als erstes lösen (Leaderboard)
  - Neue Sprache lernen
  - Algorithmen vertiefen
- Mein Ziel: Laufzeit reduzieren



https://timvisee.com/blog/solving-aoc-2020-in-under-a-second/

2022/Tag 19: Geoden aufbrechen

2022/Tag 19: Geoden aufbrechen

The wind has changed direction enough to stop sending lava droplets toward you, so you and the elephants exit the cave. As you do, you notice a collection of geodes around the pond. Perhaps you could use the obsidian to create some **geode-cracking robots** and break them open?

To collect the obsidian from the bottom of the pond, you'll need waterproof **obsidian-collecting robots**. Fortunately, there is an abundant amount of clay nearby that you can use to make them waterproof.

In order to harvest the clay, you'll need special-purpose **clay-collecting robots**. To make any type of robot, you'll need **ore**, which is also plentiful but in the opposite direction from the clay.

Collecting ore requires **ore-collecting robots** with big drills. Fortunately, **you have exactly one ore-collecting robot** in your pack that you can use to kickstart the whole operation.

Each robot can collect 1 of its resource type per minute. It also takes one minute for the robot factory (also conveniently from your pack) to construct any type of robot, although it consumes the necessary resources available when construction begins.

The robot factory has many **blueprints** (your puzzle input) you can choose from, but once you've configured it with a blueprint, you can't change it. You'll need to work out which blueprint is best.

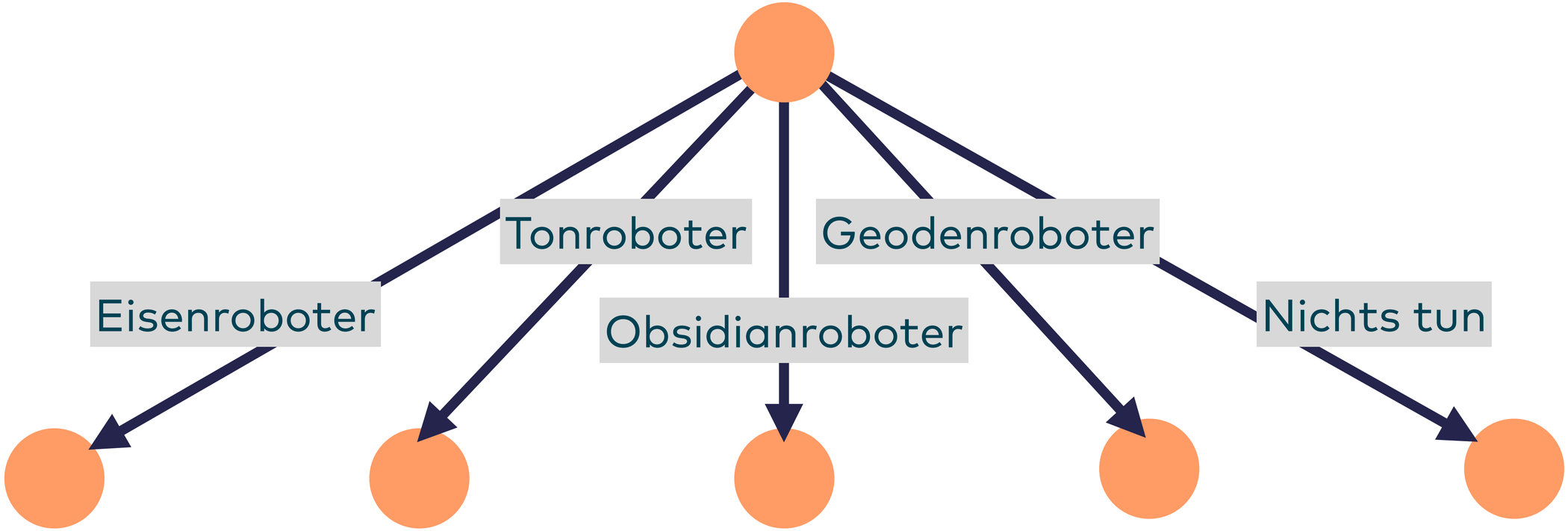
2022/Tag 19: Geoden aufbrechen

- Wir haben 30 verschiedene Baupläne
- Ein Bauplan gibt an, aus wie vielen Einheiten ein neuer Roboter gebaut werden kann
- Finde den besten Weg für einen Bauplan
- Finde aus allen Bauplänen den besten

Bauplan:
Eisenroboter kostet 4 Eisen
Tonroboter kostet 2 Eisen
Obsidianroboter kostet 3 Eisen und 14 Ton
Geodenroboter kostet 2 Eisen und 7 Obsidian

Was hat das mit Graphen zu tun?

- Aus den Entscheidungen, die wir jede Minute treffen, entsteht ein Baum
- Erinnerung Baum: gerichteter, kreisfreier Graph



Beispiel

Bauplan:
Eisenroboter kostet 4 Eisen
Tonroboter kostet 2 Eisen
Obsidianroboter kostet 3 Eisen und 14 Ton
Geodenroboter kostet 2 Eisen und 7 Obsidian

Minute	Eisen-roboter	Ton-roboter	Obisidian-roboter	Geoden-roboter	Eisen	Ton	Obsidian	Geknackte Geoden
1	1				1			

1 Eisenroboter baut 1 Eisen ab

Beispiel

Bauplan:
Eisenroboter kostet 4 Eisen
Tonroboter kostet 2 Eisen
Obsidianroboter kostet 3 Eisen und 14 Ton
Geodenroboter kostet 2 Eisen und 7 Obsidian

Minute	Eisen-roboter	Ton-roboter	Obisidian-roboter	Geoden-roboter	Eisen	Ton	Obsidian	Geknackte Geoden
2	1				2			

1 Eisenroboter baut 1 Eisen ab

Beispiel

Bauplan:
Eisenroboter kostet 4 Eisen
Tonroboter kostet 2 Eisen
Obsidianroboter kostet 3 Eisen und 14 Ton
Geodenroboter kostet 2 Eisen und 7 Obsidian

Minute	Eisen-roboter	Ton-roboter	Obisidian-roboter	Geoden-roboter	Eisen	Ton	Obsidian	Geknackte Geoden
3	1	1			1			

Baue einen Tonroboter aus 2 Eisen.
1 Eisenroboter baut 1 Eisen ab

Beispiel

Bauplan:
Eisenroboter kostet 4 Eisen
Tonroboter kostet 2 Eisen
Obsidianroboter kostet 3 Eisen und 14 Ton
Geodenroboter kostet 2 Eisen und 7 Obsidian

Minute	Eisen-roboter	Ton-roboter	Obisidian-roboter	Geoden-roboter	Eisen	Ton	Obsidian	Geknackte Geoden
4	1	1			2	1		

1 Eisenroboter baut 1 Eisen ab.
1 Tonroboter baut 1 Ton ab.

Beispiel

Bauplan:
Eisenroboter kostet 4 Eisen
Tonroboter kostet 2 Eisen
Obsidianroboter kostet 3 Eisen und 14 Ton
Geodenroboter kostet 2 Eisen und 7 Obsidian

Minute	Eisen-roboter	Ton-roboter	Obisidian-roboter	Geoden-roboter	Eisen	Ton	Obsidian	Geknackte Geoden
5	1	2			1	2		

Baue einen Tonroboter aus 2 Eisen.
1 Eisenroboter baut 1 Eisen ab.
1 Tonroboter baut 1 Ton ab

Beispiel

Bauplan:
Eisenroboter kostet 4 Eisen
Tonroboter kostet 2 Eisen
Obsidianroboter kostet 3 Eisen und 14 Ton
Geodenroboter kostet 2 Eisen und 7 Obsidian

Minute	Eisen-roboter	Ton-roboter	Obisidian-roboter	Geoden-roboter	Eisen	Ton	Obsidian	Geknackte Geoden
6	1	2			2	4		

1 Eisenroboter baut 1 Eisen ab.
2 Tonroboter baut 2 Ton ab.

Problem: Finde den besten Pfad

https://de.wikipedia.org/wiki/Breitensuche

https://de.wikipedia.org/wiki/Tiefensuche

Problem: Finde den besten Pfad

https://de.wikipedia.org/wiki/Breitensuche

https://de.wikipedia.org/wiki/Tiefensuche

Problem: Finde den besten Pfad

Breitensuche

```
private fun maxGeodes(blueprint: Blueprint): Int {
    val queue = LinkedList<State>()
    queue.add(State())
    var maxGeodes = 0
    while (queue.isNotEmpty()) {
        val state = queue.pollFirst()
        maxGeodes = maxOf(maxGeodes, score(state))
        if (!goal(state)) {
            getChildren(state, blueprint)
                .forEach { childState ->
                    queue.addLast(childState)
                }
        }
    }
    return maxGeodes
}
```

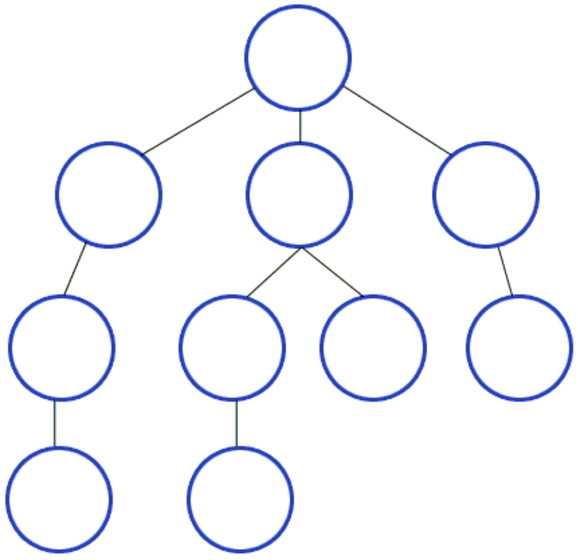
https://de.wikipedia.org/wiki/Breitensuche

https://de.wikipedia.org/wiki/Tiefensuche

Problem: Finde den besten Pfad

Breitensuche

```
private fun maxGeodes(blueprint: Blueprint): Int {
    val queue = LinkedList<State>()
    queue.add(State())
    var maxGeodes = 0
    while (queue.isNotEmpty()) {
        val state = queue.pollFirst()
        maxGeodes = maxOf(maxGeodes, score(state))
        if (!goal(state)) {
            getChildren(state, blueprint)
                .forEach { childState ->
                    queue.addLast(childState)
                }
        }
    }
    return maxGeodes
}
```



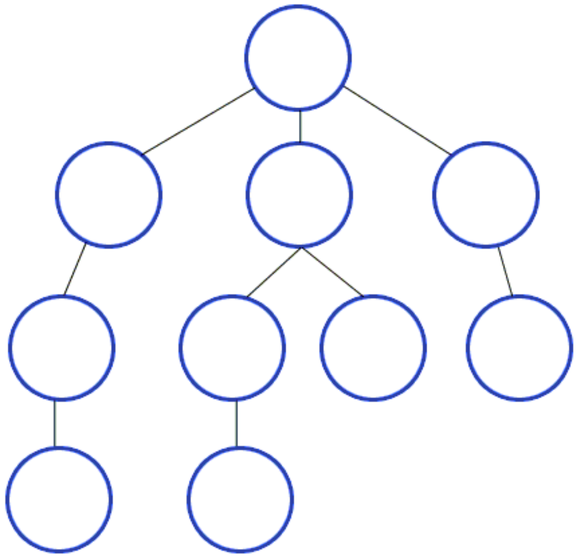
https://de.wikipedia.org/wiki/Breitensuche

https://de.wikipedia.org/wiki/Tiefensuche

Problem: Finde den besten Pfad

Breitensuche

```
private fun maxGeodes(blueprint: Blueprint): Int {
    val queue = LinkedList<State>()
    queue.add(State())
    var maxGeodes = 0
    while (queue.isNotEmpty()) {
        val state = queue.pollFirst()
        maxGeodes = maxOf(maxGeodes, score(state))
        if (!goal(state)) {
            getChildren(state, blueprint)
                .forEach { childState ->
                    queue.addLast(childState)
                }
        }
    }
    return maxGeodes
}
```



https://de.wikipedia.org/wiki/Breitensuche

Tiefensuche

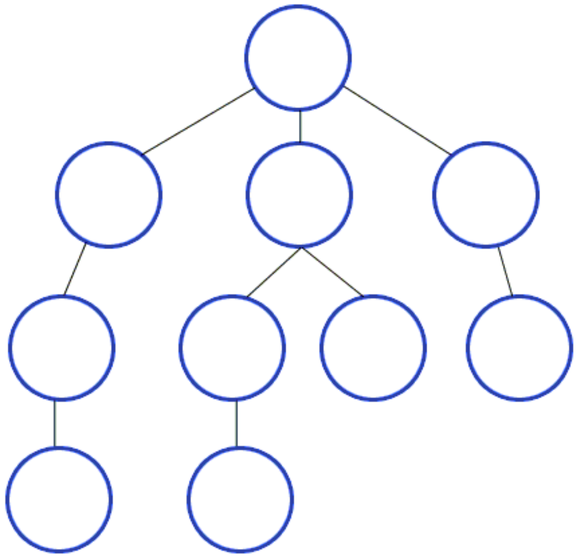
```
private fun maxGeodes(blueprint: Blueprint): Int {
    val stack = LinkedList<State>()
    stack.add(State())
    var maxGeodes = 0
    while (stack.isNotEmpty()) {
        val state = stack.pollFirst()
        maxGeodes = maxOf(maxGeodes, score(state))
       if (!goal(state)) {
            getChildren(state, blueprint)
                .forEach { childState ->
                    stack.addFirst(childState)
                }
        }
    }
    return maxGeodes
}
```

https://de.wikipedia.org/wiki/Tiefensuche

Problem: Finde den besten Pfad

Breitensuche

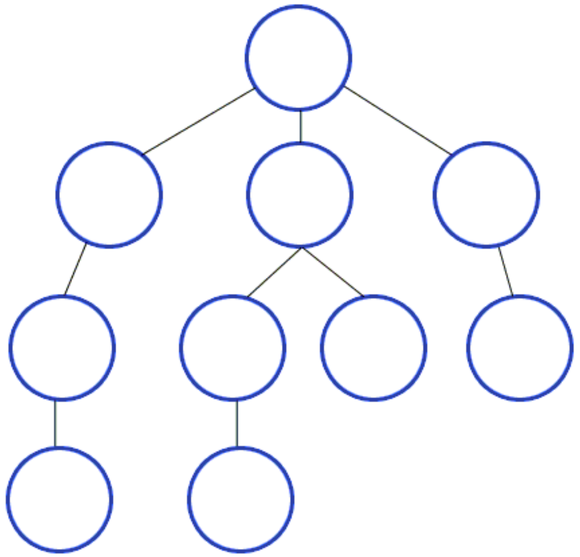
```
private fun maxGeodes(blueprint: Blueprint): Int {
    val queue = LinkedList<State>()
    queue.add(State())
    var maxGeodes = 0
    while (queue.isNotEmpty()) {
        val state = queue.pollFirst()
        maxGeodes = maxOf(maxGeodes, score(state))
        if (!goal(state)) {
            getChildren(state, blueprint)
                .forEach { childState ->
                    queue.addLast(childState)
                }
        }
    }
    return maxGeodes
}
```



https://de.wikipedia.org/wiki/Breitensuche

Tiefensuche

```
private fun maxGeodes(blueprint: Blueprint): Int {
    val stack = LinkedList<State>()
    stack.add(State())
    var maxGeodes = 0
    while (stack.isNotEmpty()) {
        val state = stack.pollFirst()
        maxGeodes = maxOf(maxGeodes, score(state))
       if (!goal(state)) {
            getChildren(state, blueprint)
                .forEach { childState ->
                    stack.addFirst(childState)
                }
        }
    }
    return maxGeodes
}
```

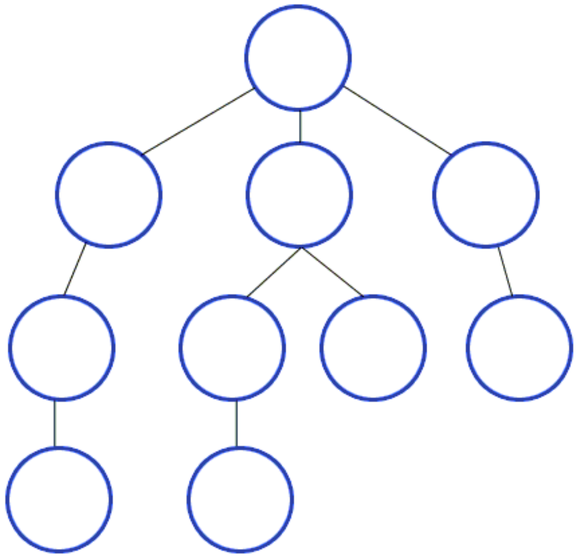


https://de.wikipedia.org/wiki/Tiefensuche

Problem: Finde den besten Pfad

Breitensuche

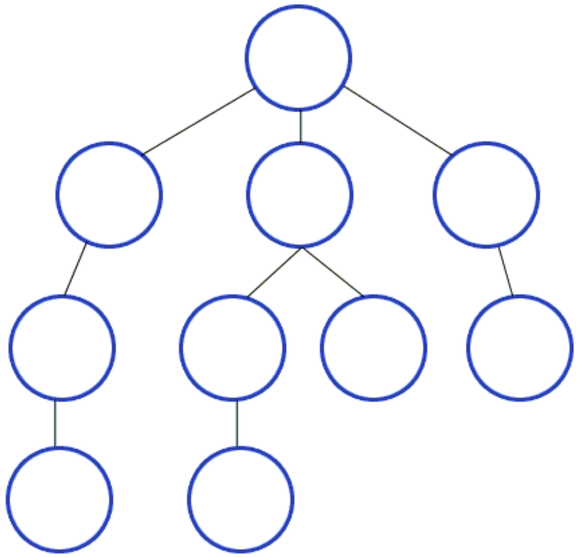
```
private fun maxGeodes(blueprint: Blueprint): Int {
    val queue = LinkedList<State>()
    queue.add(State())
    var maxGeodes = 0
    while (queue.isNotEmpty()) {
        val state = queue.pollFirst()
        maxGeodes = maxOf(maxGeodes, score(state))
        if (!goal(state)) {
            getChildren(state, blueprint)
                .forEach { childState ->
                    queue.addLast(childState)
                }
        }
    }
    return maxGeodes
}
```



https://de.wikipedia.org/wiki/Breitensuche

Tiefensuche

```
private fun maxGeodes(blueprint: Blueprint): Int {
    val stack = LinkedList<State>()
    stack.add(State())
    var maxGeodes = 0
    while (stack.isNotEmpty()) {
        val state = stack.pollFirst()
        maxGeodes = maxOf(maxGeodes, score(state))
       if (!goal(state)) {
            getChildren(state, blueprint)
                .forEach { childState ->
                    stack.addFirst(childState)
                }
        }
    }
    return maxGeodes
}
```

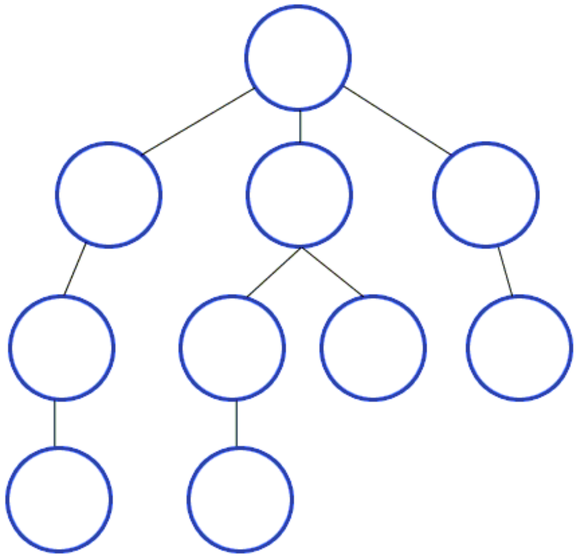


https://de.wikipedia.org/wiki/Tiefensuche

Problem: Finde den besten Pfad

Breitensuche

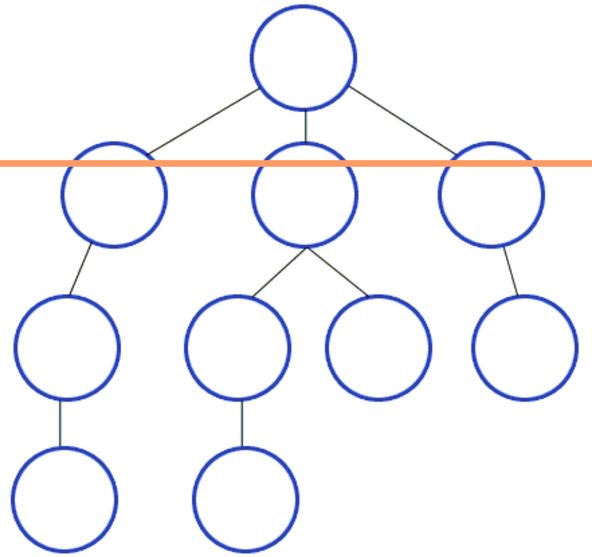
```
private fun maxGeodes(blueprint: Blueprint): Int {
    val queue = LinkedList<State>()
    queue.add(State())
    var maxGeodes = 0
    while (queue.isNotEmpty()) {
        val state = queue.pollFirst()
        maxGeodes = maxOf(maxGeodes, score(state))
        if (!goal(state)) {
            getChildren(state, blueprint)
                .forEach { childState ->
                    queue.addLast(childState)
                }
        }
    }
    return maxGeodes
}
```



https://de.wikipedia.org/wiki/Breitensuche

Tiefensuche

```
private fun maxGeodes(blueprint: Blueprint): Int {
    val stack = LinkedList<State>()
    stack.add(State())
    var maxGeodes = 0
    while (stack.isNotEmpty()) {
        val state = stack.pollFirst()
        maxGeodes = maxOf(maxGeodes, score(state))
       if (!goal(state)) {
            getChildren(state, blueprint)
                .forEach { childState ->
                    stack.addFirst(childState)
                }
        }
    }
    return maxGeodes
}
```



https://de.wikipedia.org/wiki/Tiefensuche

Problem: Finde den besten Pfad

- Pro Bauplan müssen wir im worst case 5^24 = 59.604.644.775.390.625 Pfade durchgehen
  - Bei 1 ms pro Pfad entspricht das das 1.8 Millionen Jahren
- Wir müssen optimieren

Idee A: Größere Schritte

- Nicht jede Minute durchlaufen
- 4 Auswahlmöglichkeiten: Eisen-, Ton-, Obsidian- oder Geoden-Roboter
  - Entweder gar nicht möglich: Überspringen
  - Sofort möglich: Neuen Zustand berechnen und zur Queue hinzufügen
  - In N Schritten möglich: Zustand in N Schritten berechnen und zur Queue hinzufügen
- Reduziert die Anzahl der Zustände

Idee B: Keine unnötigen Roboter bauen

- Wir können im Bauplan nachgucken, wie viele Einheiten von einem Material pro Runde maximal benötigt werden
- Mehr Roboter von diesem Typ werden wir nicht benötigen
- Wenn mir mehr als genug Material haben, als wir jemals brauchen, können wir auch abbrechen

```
Blueprint 1:
  Each ore robot costs 4 ore.
  Each clay robot costs 2 ore.
  Each obsidian robot costs 3 ore and 14 clay.
  Each geode robot costs 2 ore and 7 obsidian.

Blueprint 2:
  Each ore robot costs 2 ore.
  Each clay robot costs 3 ore.
  Each obsidian robot costs 3 ore and 8 clay.
  Each geode robot costs 3 ore and 12 obsidian.
```

Idee C: Pfade abbrechen

- Ab dem ersten durchlaufenen Pfad haben wir ein vorläufiges Ergebnis
- Pfade, die auf jeden Fall schlechter als dieses Ergebnis sind, müssen nicht weiter untersucht werden
- Sehr grobe Heuristik: Ab jetzt jede Minute ein Geodenroboter (egal ob möglich)

Welches war die beste Maßnahme?

Welches war die beste Maßnahme?

C: 52 Minuten

Welches war die beste Maßnahme?

B: 55 Sekunden

C: 52 Minuten

Welches war die beste Maßnahme?

A: 10 Sekunden

B: 55 Sekunden

C: 52 Minuten

Welches war die beste Maßnahme?

A: 10 Sekunden

B: 55 Sekunden

C: 52 Minuten

Alle zusammen: 112 ms

Weitere kleine Ideen

- Im letzten Zug nur noch Geoden Roboter bauen
- Bei Bau eines Geoden-Roboter sofort Beitrag zum Score ausrechnen und nur das speichern
- Gleiche Zustände zusammenführen

Ideen, die nicht funktioniert haben

- Falsch: Immer den hochwertigsten Roboter bauen
- Falsch: Schon zwei Schritte vor Schluss nur noch Geoden-Roboter bauen
- Schwer messbar: Reihenfolge der zu bauenden Roboter

Fazit

- Falls Breiten- oder Tiefensuche verwendet wird, bewusst für eines entscheiden
- Laufzeit-optimiertes Programmieren spart signifikant Rechenzeit bei großen Datenmengen

2022/Tag 16: Elefanten retten

The ground rumbles again, much stronger this time. What kind of cave is this, exactly? You scan the cave with your handheld device; it reports mostly igneous rock, some ash, pockets of pressurized gas, magma... this isn't just a cave, it's a volcano!

You need to get the elephants out of here, quickly. Your device estimates that you have **30 minutes** before the volcano erupts, so you don't have time to go back out the way you came in.

You scan the cave for other options and discover a network of pipes and pressure-release **valves**. You aren't sure how such a system got into a volcano, but you don't have time to complain; your device produces a report (your puzzle input) of each valve's **flow rate** if it were opened (in pressure per minute) and the tunnels you could use to move between the valves.

There's even a valve in the room you and the elephants are currently standing in labeled `AA`. You estimate it will take you one minute to open a single valve and one minute to follow any tunnel from one valve to another. What is the most pressure you could release?

2022/Tag 16: Elefanten retten

- 60 verschiedene Höhlen mit 2 bis 5 Wegen zu anderen Höhlen
- Alle haben Ventile mit verschiedenen Flowrates, viele davon 0
- Wichtig: Pro Minute die das Ventil auf ist, wird der angegeben Druck abgegeben

```
Valve HN has flow rate=19; tunnels lead to valves UW, UH, GL, WZ, GC
Valve VT has flow rate=0; tunnels lead to valves ZD, PE
Valve VI has flow rate=0; tunnels lead to valves JS, AA
Valve YL has flow rate=12; tunnels lead to valves PM, MH, RM, CM
Valve LA has flow rate=0; tunnels lead to valves SN, IY
Valve CM has flow rate=0; tunnels lead to valves YL, UL
Valve JI has flow rate=24; tunnels lead to valves UV, WH, XW, OJ
Valve ZD has flow rate=25; tunnels lead to valves VB, XW, VT
```

Grundidee

- Wir berechnen alle möglichen Wege und Ventil-Öffnungs-Kombinationen
  - Finde das Maximum
- Die Höhlen bilden einen ungerichteten Graphen
  - Die Höhlen sind die Knoten
  - Die Wege die Kanten
- Dijkstra?
- Schwierigkeit: Der Wert eines Knoten hängt von dem Zeitpunkt, an dem das Ventil geöffnet wird, ab

Dijkstra

- Jeder Knoten initial Distanz unendlich
- Es gibt einen Startknoten mit Distanz=0
- Knoten u mit kleinster Distanz auswählen
- Für alle (unbesuchten) Nachbarn v berechnen, ob der Weg über u kürzer ist als der bisheriger Weg
- Ergebnis: Kürzester Weg zu jedem Knoten

```
 1  function Dijkstra(Graph, source):
 2
 3      for each vertex v in Graph.Vertices:
 4          dist[v] ← INFINITY
 5          prev[v] ← UNDEFINED
 6          add v to Q
 7      dist[source] ← 0
 8
 9      while Q is not empty:
10          u ← vertex in Q with min dist[u]
11          remove u from Q
12
13          for each neighbor v of u still in Q:
14              alt ← dist[u] + Graph.Edges(u, v)
15              if alt < dist[v]:
16                  dist[v] ← alt
17                  prev[v] ← u
18
19      return dist[], prev[]
```

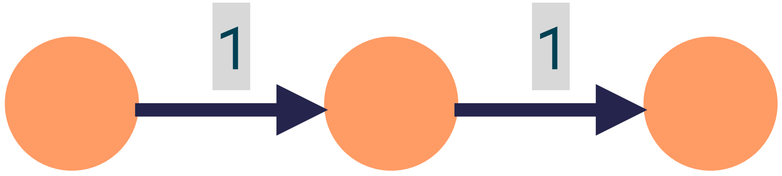
https://de.wikipedia.org/wiki/Dijkstra-Algorithmus

Geeignet?

- Aber was wäre hier überhaupt das Kantengewicht?
  - Veränderliche Werte durch Zeit
- Null-Ventile verkomplizieren den Weg
  - Ebenso das man das Ventil nicht öffnen muss
- Wann gilt eine Höhle im Dijkstra Sinne als „besucht“?

Erster Schritt: Leere Knoten entfernen

- Höhlen mit Ventilen mit Druck=0 bringen keinen Mehrwert und vergrößern nur die Anzahl der Wege
- Ähnlich wie beim ersten Problem
- Alle nacheinander entfernen
- Neuer Graph mit weniger Knoten (15!) und Kanten mit unterschiedlichen Werten



Zweiter Schritt: Das Ziel ist das Ziel

- Entscheidend ist die Reihenfolge in der die Ventile geöffnet werden und zu welchem Zeitpunkt
- Graph umdefinieren:
  - Paarweise Weglänge zwischen allen Höhlen berechnen
  - Von 60 Knoten auf 15 Knoten

Floyd-Warshall

- Eigentlich für gerichtete Graphen
- Findet für alle Knotenpaaren den kürzesten Weg
- Zeitkomponente erstmal ausklammern, nur Weglänge betrachen

```
let dist be a |V| × |V| array of minimum distances initialized to ∞
for each edge (u, v) do
    dist[u][v] ← w(u, v)  // The weight of the edge (u, v)
for each vertex v do
    dist[v][v] ← 0
for k from 1 to |V|
    for i from 1 to |V|
        for j from 1 to |V|
            if dist[i][j] > dist[i][k] + dist[k][j]
                dist[i][j] ← dist[i][k] + dist[k][j]
            end if
```

https://en.wikipedia.org/wiki/Floyd%E2%80%93Warshall_algorithm

Dritter Schritt: Dijkstra???

```
private fun findBestPath(dist: Map<String, Map<String, Int>>, values: Map<String, Int>, nodes: Set<String> = dist.keys, time: Int = 30): Long {
    val stack = MutableList( size: 1) { Point( current: "AA", time, setOf("AA"),  units: 0) }
    var max = 0L
    while (stack.isNotEmpty()) {
        val p = stack.removeAt( index: 0)
        val (current, remaining, open, units) = p
        val candidates = nodes.filter { !open.contains(it) }.filter { dist[current]!![it]!! + 1 < remaining }
        if (candidates.isEmpty()) {
            if (units > max) {
                max = p.units
            }
        } else {
            val potential = nodes.filter { !open.contains(it) } List<String>
                .sortedBy { values[it] }.reversed()
                .take( n: maxOf( a: remaining-2,  b: 0)/2)
                .mapIndexed { i, s -> (remaining-2-2*i) * values[s]!! } List<Int>
                .sum()
            if (potential+units < max) {
                continue
            }
            candidates.forEach {  it: String
                val timeToOpen = dist[current]!![it]!! + 1
                val unitsProduced = (remaining - timeToOpen) * values[it]!!
                stack.add(
                     index: 0,
                    Point(it,  remainingMinutes: remaining - timeToOpen,  openValves: open + it,  units: units + unitsProduced))
            }
        }
    }
    return max
}
```

Dritter Schritt: Dijkstra???

```
private fun findBestPath(dist: Map<String, Map<String, Int>>, values: Map<String, Int>, nodes: Set<String> = dist.keys, time: Int = 30): Long {
    val stack = MutableList( size: 1) { Point( current: "AA", time, setOf("AA"),  units: 0) }
    var max = 0L
    while (stack.isNotEmpty()) {
        val p = stack.removeAt( index: 0)
        val (current, remaining, open, units) = p
        val candidates = nodes.filter { !open.contains(it) }.filter { dist[current]!![it]!! + 1 < remaining }
        if (candidates.isEmpty()) {
            if (units > max) {
                max = p.units
            }
        } else {
            val potential = nodes.filter { !open.contains(it) } List<String>
                .sortedBy { values[it] }.reversed()
                .take( n: maxOf( a: remaining-2,  b: 0)/2)
                .mapIndexed { i, s -> (remaining-2-2*i) * values[s]!! } List<Int>
                .sum()
            if (potential+units < max) {
                continue
            }
            candidates.forEach {  it: String
                val timeToOpen = dist[current]!![it]!! + 1
                val unitsProduced = (remaining - timeToOpen) * values[it]!!
                stack.add(
                     index: 0,
                    Point(it,  remainingMinutes: remaining - timeToOpen,  openValves: open + it,  units: units + unitsProduced))
            }
        }
    }
    return max
}
```

Dritter Schritt: ~~Dijkstra???~~ TS!

```
private fun findBestPath(dist: Map<String, Map<String, Int>>, values: Map<String, Int>, nodes: Set<String> = dist.keys, time: Int = 30): Long {
    val stack = MutableList( size: 1) { Point( current: "AA", time, setOf("AA"),  units: 0) }
    var max = 0L
    while (stack.isNotEmpty()) {
        val p = stack.removeAt( index: 0)
        val (current, remaining, open, units) = p
        val candidates = nodes.filter { !open.contains(it) }.filter { dist[current]!![it]!! + 1 < remaining }
        if (candidates.isEmpty()) {
            if (units > max) {
                max = p.units
            }
        } else {
            val potential = nodes.filter { !open.contains(it) } List<String>
                .sortedBy { values[it] }.reversed()
                .take( n: maxOf( a: remaining-2,  b: 0)/2)
                .mapIndexed { i, s -> (remaining-2-2*i) * values[s]!! } List<Int>
                .sum()
            if (potential+units < max) {
                continue
            }
            candidates.forEach {  it: String
                val timeToOpen = dist[current]!![it]!! + 1
                val unitsProduced = (remaining - timeToOpen) * values[it]!!
                stack.add(
                     index: 0,
                    Point(it,  remainingMinutes: remaining - timeToOpen,  openValves: open + it,  units: units + unitsProduced))
            }
        }
    }
    return max
}
```

In Einfach(er)

```
data class State(val current: String, val remainingMinutes: Int, val openValves: Set<String>, val units: Long)

private fun findBestPath(dist: Map<String, Map<String, Int>>, values: Map<String, Int>): Long {
    val stack = LinkedList<State>()
    stack.add(State( current: "AA",  remainingMinutes: 30, setOf("AA"),  units: 0))
    var max = 0L
    while (stack.isNotEmpty()) {
        val state = stack.pollFirst()
        val candidates = dist.keys Set<String>
            .filter { !state.openValves.contains(it) } List<String>
            .filter { dist[state.current]!![it]!! + 1 < state.remainingMinutes }
        if (candidates.isEmpty()) {
            if (state.units > max) {
                max = state.units
            }
        } else {
            candidates.forEach {  it: String
                val timeToOpen = dist[state.current]!![it]!! + 1
                val unitsProduced = (state.remainingMinutes - timeToOpen) * values[it]!!
                stack.add(
                    index: 0,
                    State(it,
                        remainingMinutes: state.remainingMinutes - timeToOpen,
                        openValves: state.openValves + it,
                        units: state.units + unitsProduced))
            }
        }
    }
    return max
}
```

In Einfach(er)

```
data class State(val current: String, val remainingMinutes: Int, val openValves: Set<String>, val units: Long)

private fun findBestPath(dist: Map<String, Map<String, Int>>, values: Map<String, Int>): Long {
    val stack = LinkedList<State>()
    stack.add(State( current: "AA",  remainingMinutes: 30, setOf("AA"),  units: 0))
    var max = 0L
    while (stack.isNotEmpty()) {
        val state = stack.pollFirst()
        val candidates = dist.keys Set<String>
            .filter { !state.openValves.contains(it) } List<String>
            .filter { dist[state.current]!![it]!! + 1 < state.remainingMinutes }
        if (candidates.isEmpty()) {
            if (state.units > max) {
                max = state.units
            }
        } else {
            candidates.forEach {  it: String
                val timeToOpen = dist[state.current]!![it]!! + 1
                val unitsProduced = (state.remainingMinutes - timeToOpen) * values[it]!!
                stack.add(
                    index: 0,
                    State(it,
                        remainingMinutes: state.remainingMinutes - timeToOpen,
                        openValves: state.openValves + it,
                        units: state.units + unitsProduced))
            }
        }
    }
    return max
}
```

642ms

Mit Optimierung

```
} else {
    val potential = dist.keys Set<String>
        .filter { !state.openValves.contains(it) } List<String>
        .sortedBy { values[it] }.reversed()
        .take( n: maxOf( a: state.remainingMinutes - 2,  b: 0)/2)
        .mapIndexed { i, s -> (state.remainingMinutes - 2 - 2*i) * values[s]!! } List<Int>
        .sum()
    if (potential + state.units < max) {
        continue
    }
    candidates.forEach {  it: String
        val timeToOpen = dist[state.current]!![it]!! + 1
        val unitsProduced = (state.remainingMinutes - timeToOpen) * values[it]!!
        stack.add(
            index: 0,
            State(it, remainingMinutes: state.remainingMinutes - timeToOpen,  openValves: state.openValves + it,  units: state.units + unitsProduced))
    }
}
```

Mit Optimierung

```
} else {
    val potential = dist.keys Set<String>
        .filter { !state.openValves.contains(it) } List<String>
        .sortedBy { values[it] }.reversed()
        .take( n: maxOf( a: state.remainingMinutes - 2,  b: 0)/2)
        .mapIndexed { i, s -> (state.remainingMinutes - 2 - 2*i) * values[s]!! } List<Int>
        .sum()
    if (potential + state.units < max) {
        continue
    }
    candidates.forEach {  it: String
        val timeToOpen = dist[state.current]!![it]!! + 1
        val unitsProduced = (state.remainingMinutes - timeToOpen) * values[it]!!
        stack.add(
            index: 0,
            State(it, remainingMinutes: state.remainingMinutes - timeToOpen,  openValves: state.openValves + it,  units: state.units + unitsProduced))
    }
}
```

150ms

Fazit

- Keine Zeitmessung vor Optimierung möglich
- Zeitlicher Verlauf und Zuständen —> Tiefensuche/Breitensuche

Danke! Fragen?

Isabel Wingen
Isabel.Wingen@innoq.com

INNOQ
www.innoq.com

innoQ Deutschland GmbH

Krischerstr. 100
40789 Monheim
+49 2173 3366-0

Ohlauer Str. 43
10999 Berlin

Ludwigstr. 180E
63067 Offenbach

Kreuzstr. 16
80331 München

Wendenstraße 130
20537 Hamburg

Spichernstraße 44
50672 Köln

Königstorgraben 11
90402 Nürnberg

Welches war die beste Maßnahme?

Welches war die beste Maßnahme?

Optimierungen	Laufzeit (Durchschnitt 3-mal)
A	10 s
B	55 s
C	52 m
A+B	137 ms
A+C	5 s
B+C	36 s
A+B+C	112 ms