Rush-hour Hinweise

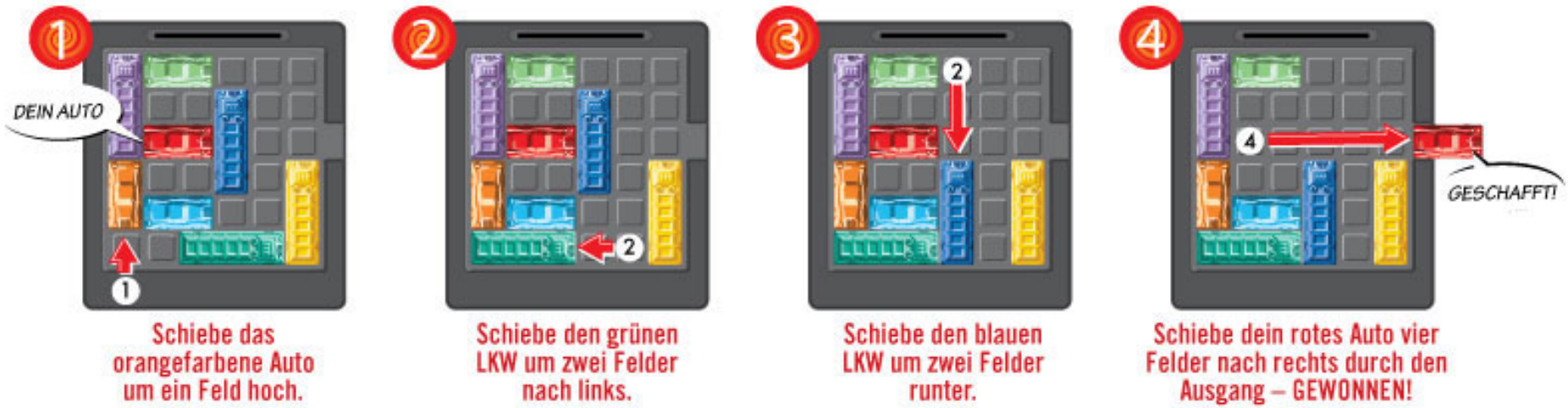
Christian Höner zu Siederdissen
christian.hoener.zu.siederdissen@uni-jena.de

Theoretische Bioinformatik, Bioinformatik Uni Jena

Dezember 2023

Rush-hour

https://www.thinkfun.de/products/rush-hour/



Rushhour ist *PSPACE-complete* und damit wahrscheinlich nur in exponentieller Zeit lösbar

Generische Puzzle-Spiele

Es handelt sich hierbei um Spiele für die Folgendes gilt:

- Spiele laufen in diskreten Zügen ab
- zu jedem beliebigen Zeitpunkt gibt es wohldefinierte Spielzustände
- aufeinander folgende Zustände lassen sich jeweils exakt beschreiben

Rush-hour ist so ein Spiel, aber auch Sudoko, Schach, go, Mensch ärgere dich nicht, und viele mehr. Wir interessieren uns nur für "1-Spieler" Spiele, da dann nicht "reagiert" werden muss.

Generische Puzzle-Spiele

Gegeben den aktuellen Zustand S eines Spiels, sind drei Probleme zu lösen (teilweise mittels Moves M):

solved $S \to \mathbb{B}$ is *wahr* wenn S ein Lösungszustand ist

move $S \to M \to S$, gegeben S und M wird ein neuer Zustand S generiert

moves $S \to \{M\}$, gegeben S, generiere alle legalen Moves $M_1 \ldots M_S$ für S

Allerdings hängen die Typen von S und M vom jeweiligen Spiel ab: wie organisieren wir das?

Listen

Listen können recht einfach missbraucht werden um mehrdimensionale Strukturen abzubilden. Der Index-Operator `(!!)` auf Listen ist 0-basierend.

```
1  -- Index-Operator
2  (l:_) !! 0 = l
3  (l:ls) !! k = ls !! (k-1)
4
5  let xs = [1..6] -- Eine Liste zum Spielen
6  let x = xs !! 3 == 4
7
8  -- 2-dimensionale Indizierung
9  let ys = [ [1], [1,2], [1,2,3] ]
10 let y = (ys !! 2) !! 2 == 3
```

Achtung: viele Operationen sind $O(n)$.

Maps

Endliches Mapping von beliebigen Schlüsseln zu Werten.

```
1  import qualified Data.Map.Strict as Map
2
3  let xs = Map.fromList $ zip [0..5] [1..6]
4  let x = xs Map.! 3 == 4
5  let y = xs Map.!? 10 == Nothing
6
7  let ys = Map.insert 10 99 xs
8  let z = ys Map.! 10 == 99
9  let n = xs Map.!? 10 == Nothing -- weil natuerlich xs
       nicht veraendert wird
```

Viele Operationen sind $O(\log n)$.

Maps nutzen

- Maps (und viele andere Datenstrukturen) sind *kein* Teil von Standard-Haskell
- es muss die `https://hackage.haskell.org/package/containers` library zur Verfügung stehen
- eventuell lässt sich diese über die Standard Linux Installation installieren, dann gerne
- ansonsten ist es nötig sich mit dem `cabal` Paketmanager zu beschäftigen
- Das Erstellen von Haskell-Paketen möchte ich gerne später in der Vorlesung machen
- Auch hier gilt wieder: wird bei Interessae gerne vorgezogen

Haskell-Arrays

- Haskell-Arrays sind ïmmutable", einmal gebaut, können sie nicht manipuliert werden
- Damit muss bei jeder Änderung das komplette Array neu gebaut werden
- Abhilfe schauen "mutableÄrrays in der ST Monade.
- Die ST Monade können wir gerne anschauen
- ich empfehle allerdings für den Moment nur Listen oder Maps zu nutzen

Hausaufgabe

`https://www.michaelfogleman.com/rush/`

- Download der Datenbank
- Parsen der Datenbank
- Konvertierung in das Format hier *oder* eigene Rush-hour Definition
- Ausprobieren
- Diese Version kennt auch "Mauern" die nicht beweglich sind!

Wer sich wirklich austoben möchte: generiert eigene Probleme für "beliebige" $N \times N$ große Instanzen.