Grundlagen der Künstlichen Intelligenz

6. Klassische Suche: Datenstrukturen für Suchalgorithmen

Malte Helmert

Universität Basel

7. März 2014

Klassische Suche: Überblick

Kapitelüberblick klassische Suche:

- 3.–5. Einführung
- 6.–9. Basisalgorithmen
  - 6. Datenstrukturen für Suchalgorithmen
  - 7. Baumsuche und Graphensuche
  - 8. Breitensuche und uniforme Kostensuche
  - 9. Tiefensuche und iterative Tiefensuche
- folgende Kapitel: heuristische Algorithmen

Einführung

Suchalgorithmen

- Wir befassen und jetzt mit Suchalgorithmen.
- Wie überall in der Informatik sind geeignete Datenstrukturen ein Schlüssel zu guter Performance.
  - $\rightsquigarrow$ häufige Operationen müssen schnell sein
- gut implementierte Suchalgorithmen verarbeiten bis zu $\sim$30,000,000 Zustände/Sekunde auf einem CPU-Kern
  - $\rightsquigarrow$ Zusatzmaterial (Paper von Burns et al.)

dieses Kapitel: einige grundlegende Datenstrukturen für Suche

Vorschau: Suchalgorithmen

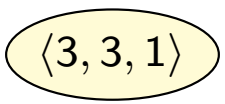
- Ab dem nächsten Kapitel betrachten wir Suchalgorithmen genauer.
- hier eine kurze Vorschau, um unsere Diskussion von Datenstrukturen zu motivieren

Beispiel: Suchalgorithmus

- Ausgehend vom Anfangszustand,
- expandiere wiederholt einen Zustand
  durch Erzeugen seiner Nachfolger.
- Höre auf, wenn ein Zielzustand expandiert wird
- oder alle erreichbaren Zustände betrachtet wurden.

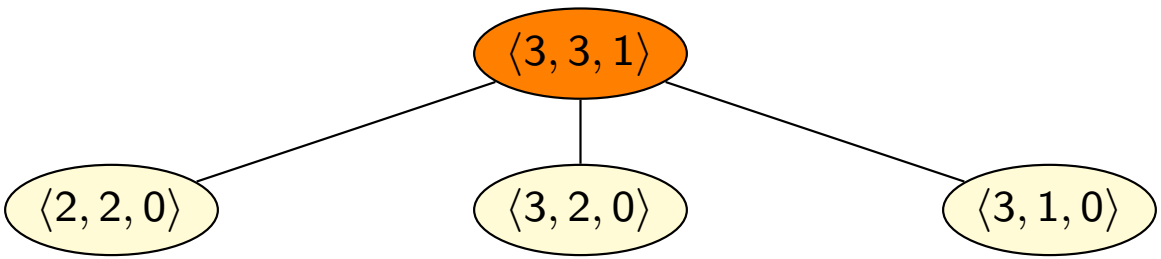
Beispiel: Suchalgorithmus

- Ausgehend vom Anfangszustand,
- expandiere wiederholt einen Zustand durch Erzeugen seiner Nachfolger.
- Höre auf, wenn ein Zielzustand expandiert wird
- oder alle erreichbaren Zustände betrachtet wurden.



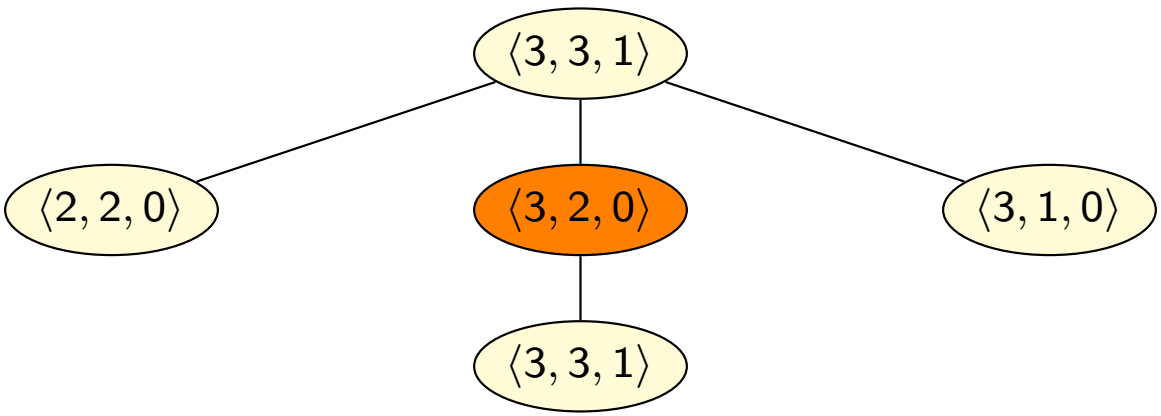
Beispiel: Suchalgorithmus

- Ausgehend vom Anfangszustand,
- expandiere wiederholt einen Zustand durch Erzeugen seiner Nachfolger.
- Höre auf, wenn ein Zielzustand expandiert wird
- oder alle erreichbaren Zustände betrachtet wurden.



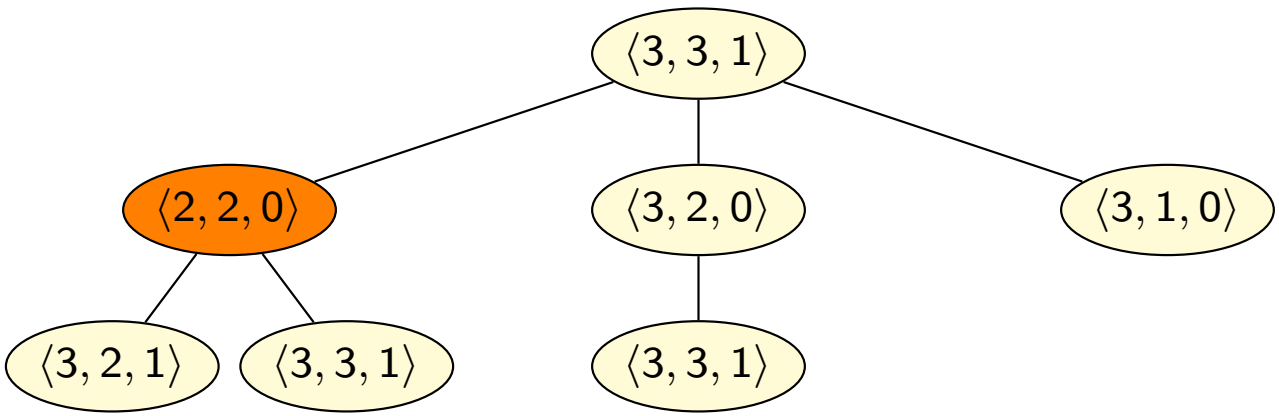
Beispiel: Suchalgorithmus

- Ausgehend vom Anfangszustand,
- expandiere wiederholt einen Zustand durch Erzeugen seiner Nachfolger.
- Höre auf, wenn ein Zielzustand expandiert wird
- oder alle erreichbaren Zustände betrachtet wurden.



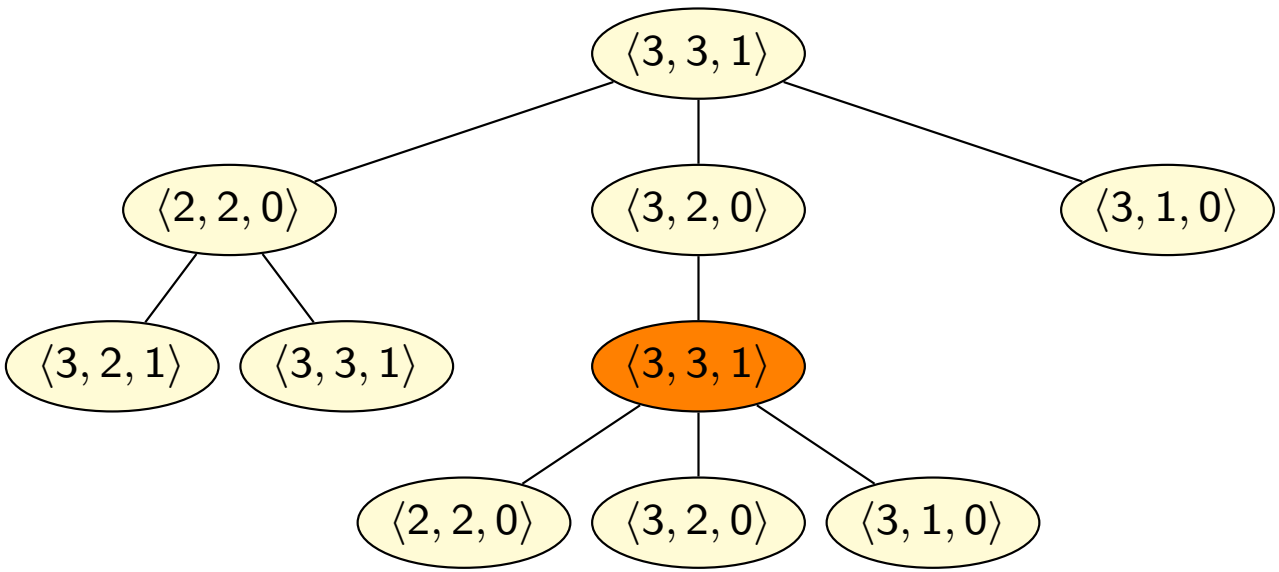
Beispiel: Suchalgorithmus

- Ausgehend vom Anfangszustand,
- expandiere wiederholt einen Zustand durch Erzeugen seiner Nachfolger.
- Höre auf, wenn ein Zielzustand expandiert wird
- oder alle erreichbaren Zustände betrachtet wurden.



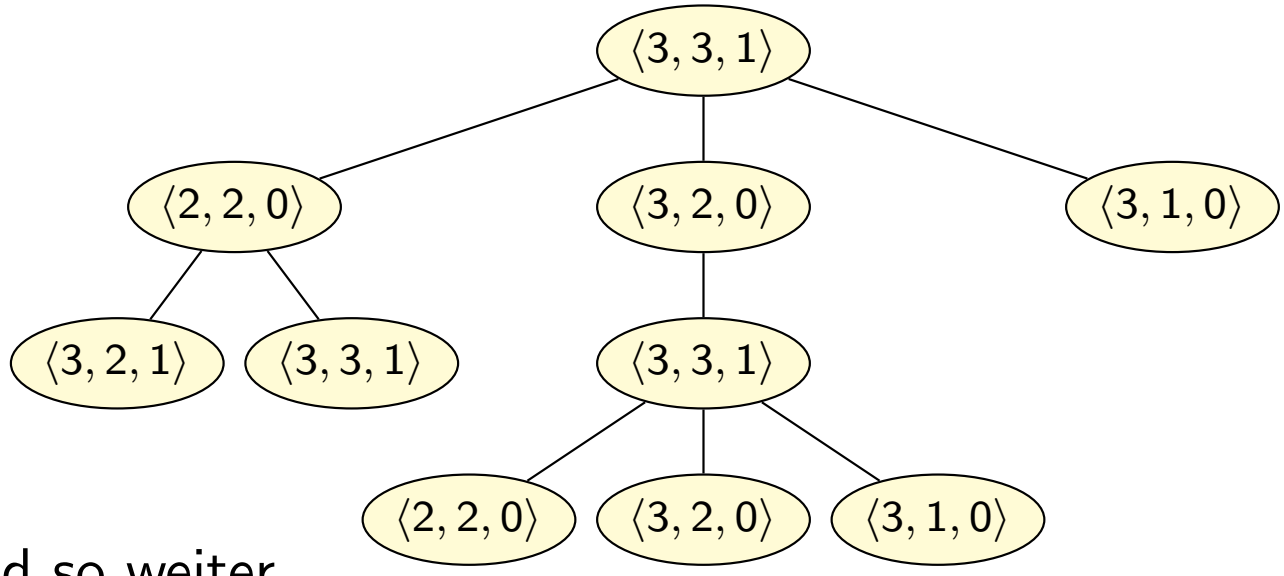
Beispiel: Suchalgorithmus

- Ausgehend vom Anfangszustand,
- expandiere wiederholt einen Zustand durch Erzeugen seiner Nachfolger.
- Höre auf, wenn ein Zielzustand expandiert wird
- oder alle erreichbaren Zustände betrachtet wurden.



Beispiel: Suchalgorithmus

- Ausgehend vom Anfangszustand,
- expandiere wiederholt einen Zustand durch Erzeugen seiner Nachfolger.
- Höre auf, wenn ein Zielzustand expandiert wird
- oder alle erreichbaren Zustände betrachtet wurden.



. . . und so weiter.

(Expansionsreihenfolge hängt vom gewählten Suchalgorithmus ab.)

Grundlegende Datenstrukturen für Suche

Wir betrachten drei abstrakte Datenstrukturen für Suche:

- Suchknoten: speichern welche Zustände erreicht werden können, wie, und unter welchen Kosten
  - $\rightsquigarrow$ Knoten des Beispielsuchbaums
- Open-Liste: organisiert die Blätter des Suchbaums effizient
  - $\rightsquigarrow$ Menge der Blätter des Beispielsuchbaums
- Closed-Liste: merkt expandierte Zustände, um mehrfache Expansion desselben Zustands zu vermeiden
  - $\rightsquigarrow$ innere Knoten eines Suchbaums

Nicht alle Algorithmen verwenden alle drei Datenstrukturen, und manchmal sind sie implizit (z. B. im CPU-Stack).

Suchknoten

Suchknoten

Suchknoten

Suchknoten (kurz: Knoten) speichern welche Zustände erreicht werden können, wie, und unter welchen Kosten

Gemeinsam bilden sie den so genannten Suchbaum.

Attribute von Suchknoten

Attribute eines Suchknoten n

n.state	Zustand, der zu diesem Knoten gehört
n.parent	Suchknoten, der diesen Knoten erzeugte (**none** für den Wurzelknoten)
n.action	Aktion, die von n.parent zu n führt (**none** für den Wurzelknoten)
n.path_cost	Kosten des Pfades vom Anfangszustand zu n.state, der aus Verfolgen der parent-Referenzen resultiert (traditionell mit $g(n)$ bezeichnet)

. . . und manchmal zusätzliche Felder (z. B., Tiefe im Baum)

Suchknoten: Java

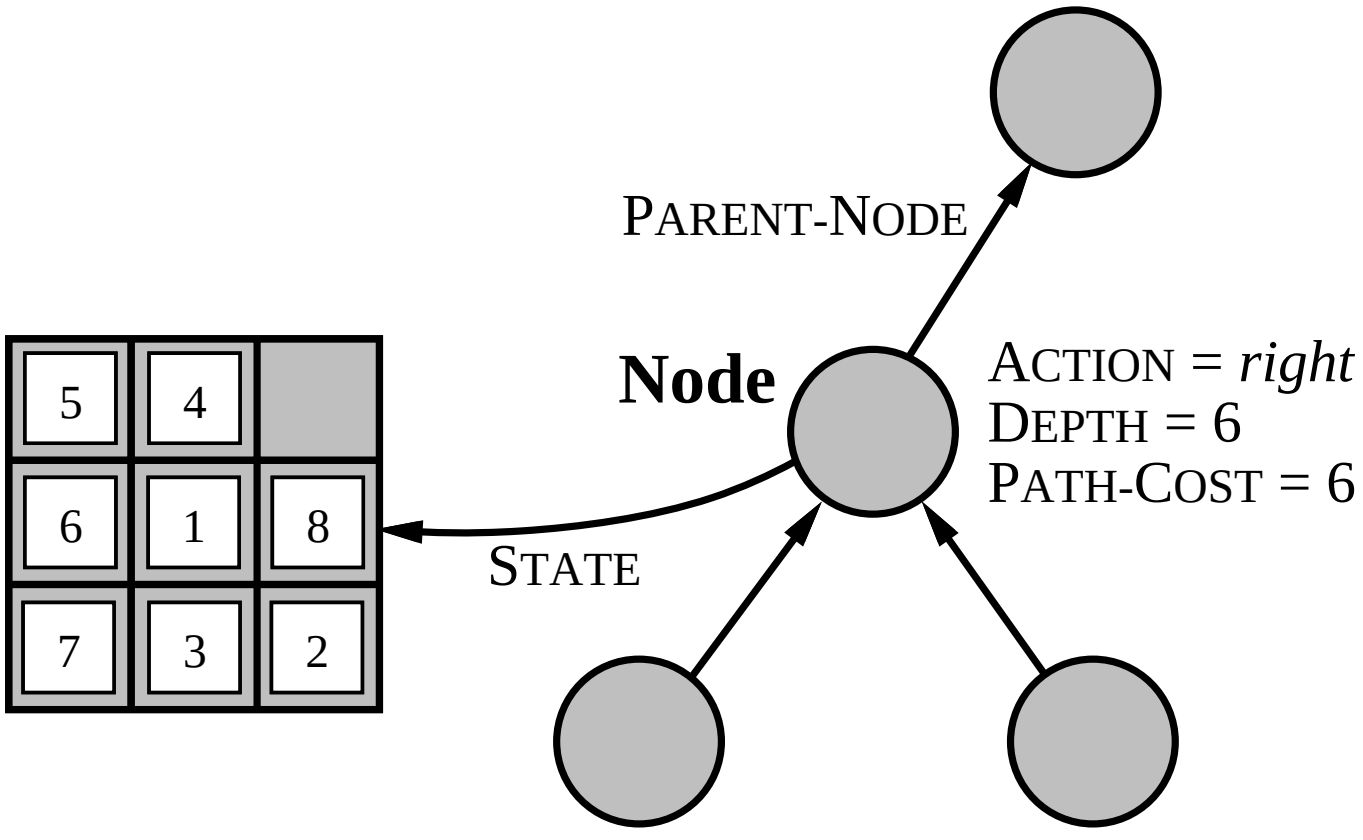
Suchknoten in Java-Syntax

```
public interface State {
}

public interface Action {
}

public class SearchNode {
    State state;
    SearchNode parent;
    Action action;
    int pathCost;
}
```

Knoten in einem Suchbaum



Implementierung von Suchknoten

- vernünftige Implementierung von Suchknoten ist einfach
- fortgeschrittene Aspekte:
  - Benötigen wir überhaupt explizite Knoten?
  - Können wir Lazy Evaluation verwenden?
  - Lohnt sich manuelle Speicherverwaltung?
  - Können wir Information komprimieren?

Operationen für Suchknoten: make_root_node

Erzeuge Wurzelknoten eines Suchbaums:

function make_root_node()

node := **new** SearchNode
node.state := init()
node.parent := **none**
node.action := **none**
node.path_cost := 0
return *node*

Operationen für Suchknoten: make_node

Erzeuge Kindknoten in einem Suchbaum:

function make_node(*parent*, *action*, *state*)

node := **new** SearchNode
node.state := *state*
node.parent := *parent*
node.action := *action*
node.path_cost := *parent*.path_cost + cost(*action*)
return *node*

Operationen für Suchknoten: extract_path

Extrahiere den Pfad zu einem Suchknoten:

function extract_path(*node*)

```
path := ⟨⟩
while node.parent ≠ none:
    path.append(node.action)
    node := node.parent
path.reverse()
return path
```

Open-Liste

Open-Listen

Open-Liste

Die Open-Liste (auch: Frontier) organisiert die Blätter des Suchbaums.

Sie muss zwei Operationen effizient unterstützen:

- bestimme und entferne den nächsten zu expandieren Knoten
- füge ein neuen Knoten ein, der Kandidat für Expansion ist

Anmerkung: trotz des Namens ist es meist eine sehr schlechte Idee, die Open-Liste als einfache Liste zu implementieren.

Open-Listen: Einträge modifizieren

- Manche Implementierungen unterstützen zusätzlich die Modifikation eines Eintrags in der Open-Liste, wenn ein billigerer Pfad zum zugehörigen Zustand gefunden wird.
- Dies macht die Implementierung komplizierter.

$\rightsquigarrow$ wir betrachten solche Modifikationen nicht und verwenden stattdessen verzögerte Duplikateliminierung ($\rightsquigarrow$ später)

Interface von Open-Listen

Methoden einer Open-Liste *open*

open.is_empty()	testet, ob die Open-Liste leer ist
open.pop()	entfernt den nächsten zu expandieren Knoten und liefert ihn zurück
open.insert(*n*)	fügt Knoten node *n* in die Open-Liste ein

- Unterschiedliche Suchalgorithmen verwenden unterschiedliche Strategien für die Entscheidung, welcher Knoten in *open*.pop zurückgeliefert wird.
- Die Wahl der passenden Datenstruktur hängt von dieser Strategie ab (z. B.: Stack, Deque, Min-Heap).

Closed-Liste

Closed-Listen

Closed-Liste

Die Closed-Liste merkt sich die expandierten Zustände,
um mehrfache Expansion desselben Zustands zu vermeiden

Sie muss zwei Operationen effizient unterstützen:

- füge einen Knoten ein, dessen Zustand
  noch nicht in der Closed-Liste ist
- teste, ob ein Knoten mit einem gegebenen Zustand
  in der Closed-Liste ist; falls ja, liefere ihn zurück

Anmerkung: trotz des Namens ist es meist eine sehr schlechte Idee,
die Closed-Liste als einfache Liste zu implementieren. (Warum?)

Interface und Implementierung von Closed-Listen

Methoden einer Closed-Liste *closed*

closed.insert(*n*)	füge Knoten *n* in *closed* ein; falls ein Knoten mit demselben Zustand bereits in *closed* existiert, ersetze ihn
closed.lookup(*s*)	teste, ob ein Knoten mit Zustand *s* in der Closed-Liste vorhanden ist; falls ja, liefere ihn zurück; sonst, liefere **none** zurück

- Hash-Tabellen mit Zuständen als Schlüssel können als effiziente Implementierung einer Closed-Liste dienen.

Zusammenfassung

Zusammenfassung

- Suchknoten:
  repräsentieren während der Suche erreichte Knoten
  und zugehörige Informationen
- Knotenexpansion:
  Erzeugen der Nachfolgeknoten eines Knoten durch Anwenden der Aktionen, die im Zustand des Knoten anwendbar sind
- Open-Liste oder Frontier:
  Menge der Knoten, die derzeit Kandidaten für Expansion sind
- Closed-Liste:
  Menge der bereits expandierten Knoten (und deren Zustände)