Hochschule Offenburg
University of Applied Sciences

Algorithmen und Datenstrukturen
7. Baumsuche

Prof. Dr. Klaus Dorer

Problemlösen durch Suche

- Baumsuche
  - Beispiel Probleme
  - Uninformierte Suchverfahren
    - Breitensuche
    - Uniforme Kostensuche
    - Tiefensuche
    - Iterative Tiefensuche
  - Informierte Suchverfahren
    - Gierige Suche
    - A* Suche

Ziele

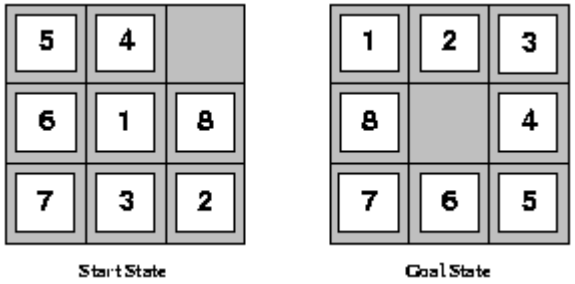
- Gemeinsamkeiten und Unterschiede der Verfahren kennen
- Behandelte Suchverfahren implementieren können

Quellen

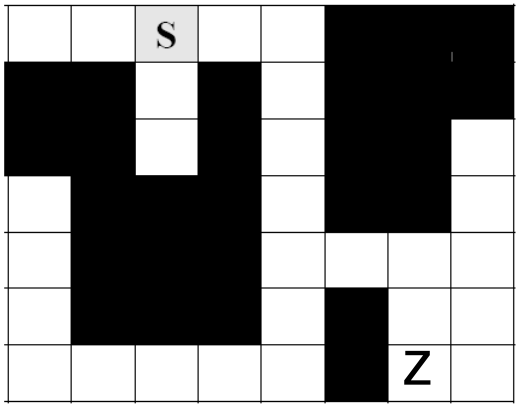
- Russel, Norvig (2002) Artificial Intelligence: A Modern Approach, Prentice Hall, ISBN 0137903952.
- Wrobel Stefan (2012) Optimierung von Transportaufträgen mit LKW Shifting. Bachelor Thesis, Hochschule Offenburg

Beispiele für Probleme

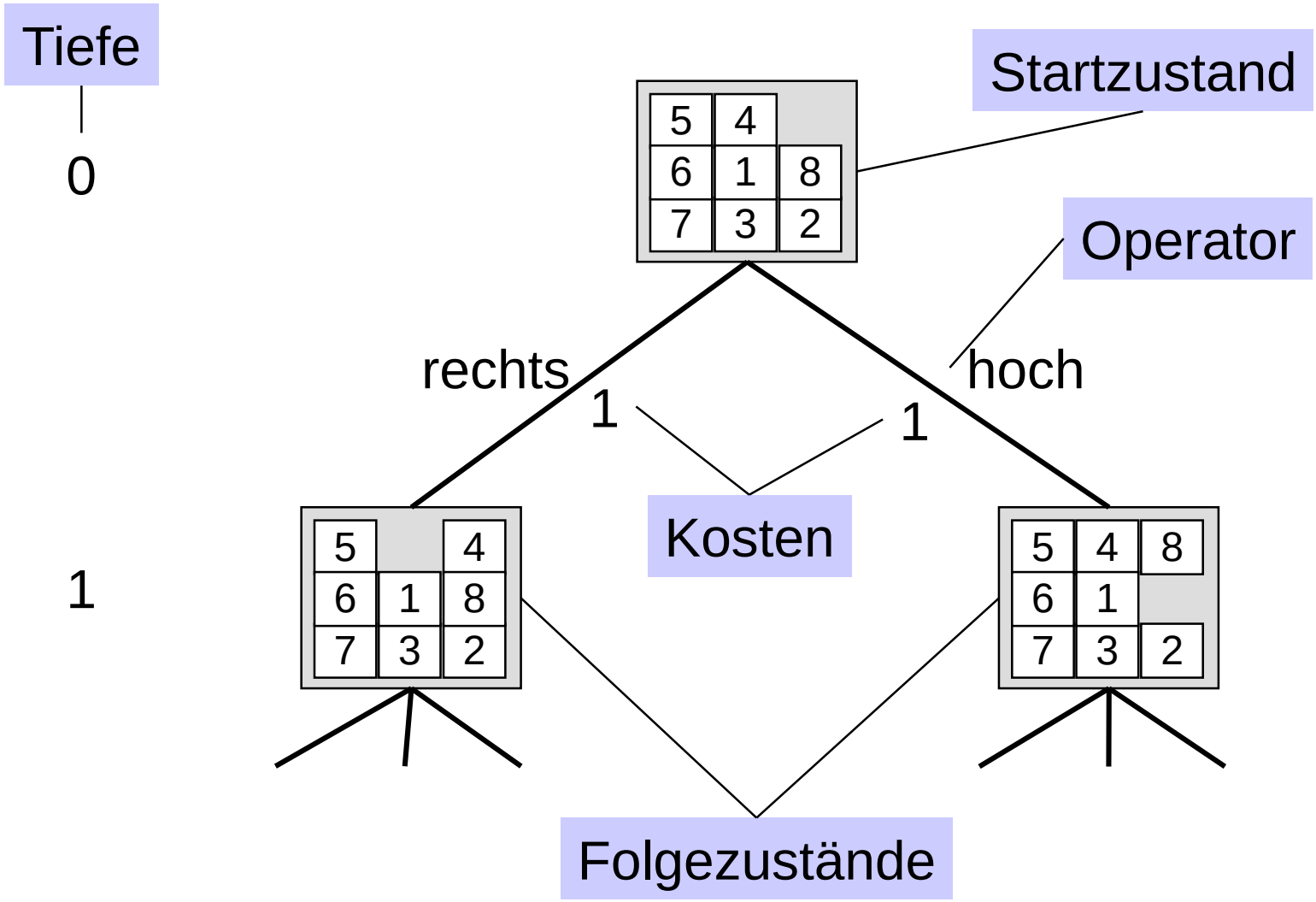
- Für manche Probleme ist nicht der Zielzustand selbst die Lösung, sondern der Weg vom Start- zum Zielzustand
- Schiebepuzzle



- Weg-Suche in einem Labyrinth



Suchbaum

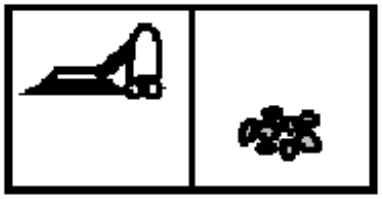

Tiefe
0
1
Startzustand
Operator
rechts
hoch
1
1
Kosten
Folgezustände
5 4
6 1 8
7 3 2
5 4
6 1 8
7 3 2
5 4 8
6 1
7 3 2

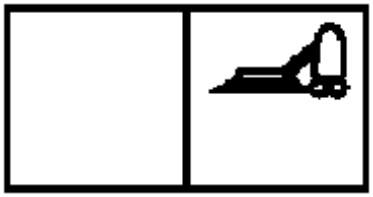
Baumsuche

- Um diesen Weg zu finden kann man einen Suchbaum erstellen, der die möglichen Wege vom Startzustand enthält
- Suchbäume bestehen aus Knoten, die definiert sind durch
  - den Zustand, den der Knoten repräsentiert
  - den übergeordneten Knoten (parent)
  - den Operator, der den Knoten generiert hat
  - die Tiefe des Knotens im Suchbaum
  - die Pfadkosten vom Anfangszustand bis zum Knoten
- Die wichtigsten Eigenschaften einer Suchstrategie sind
  - Vollständigkeit,
  - Optimalität,
  - Zeitkomplexität,
  - Speicherplatzkomplexität

Autonomer Staubsauger

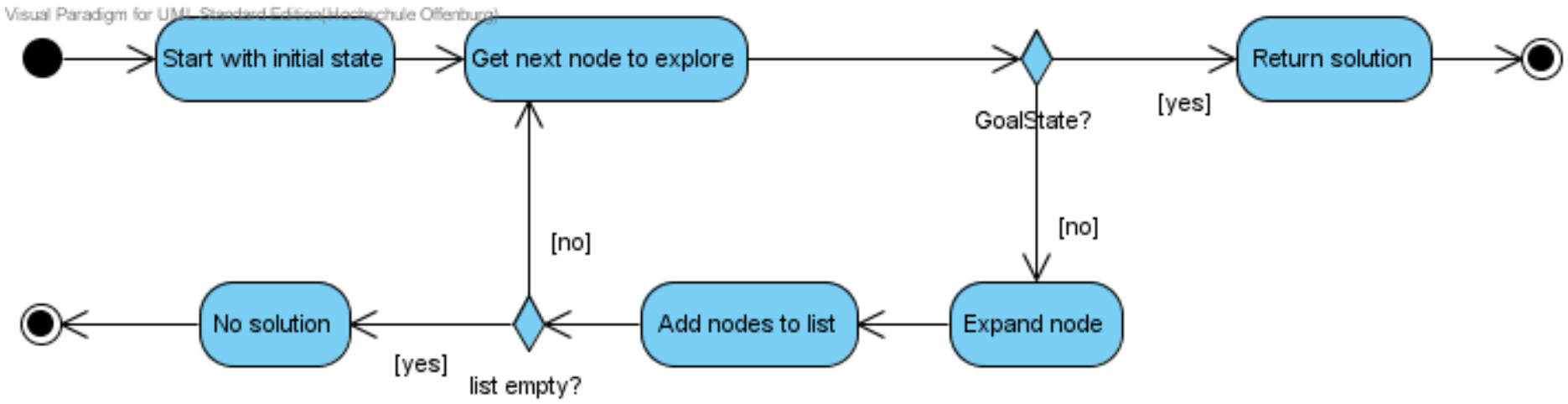
- Staubsauger soll alle Räume eines Hauses selbständig reinigen
- Aktionen
  - Links fahren
  - Saugen
  - Rechts fahren
- Ziel
  - Alle Räume sauber
- Aufgabe
  - Erstelle optimalen Plan von Aktionen





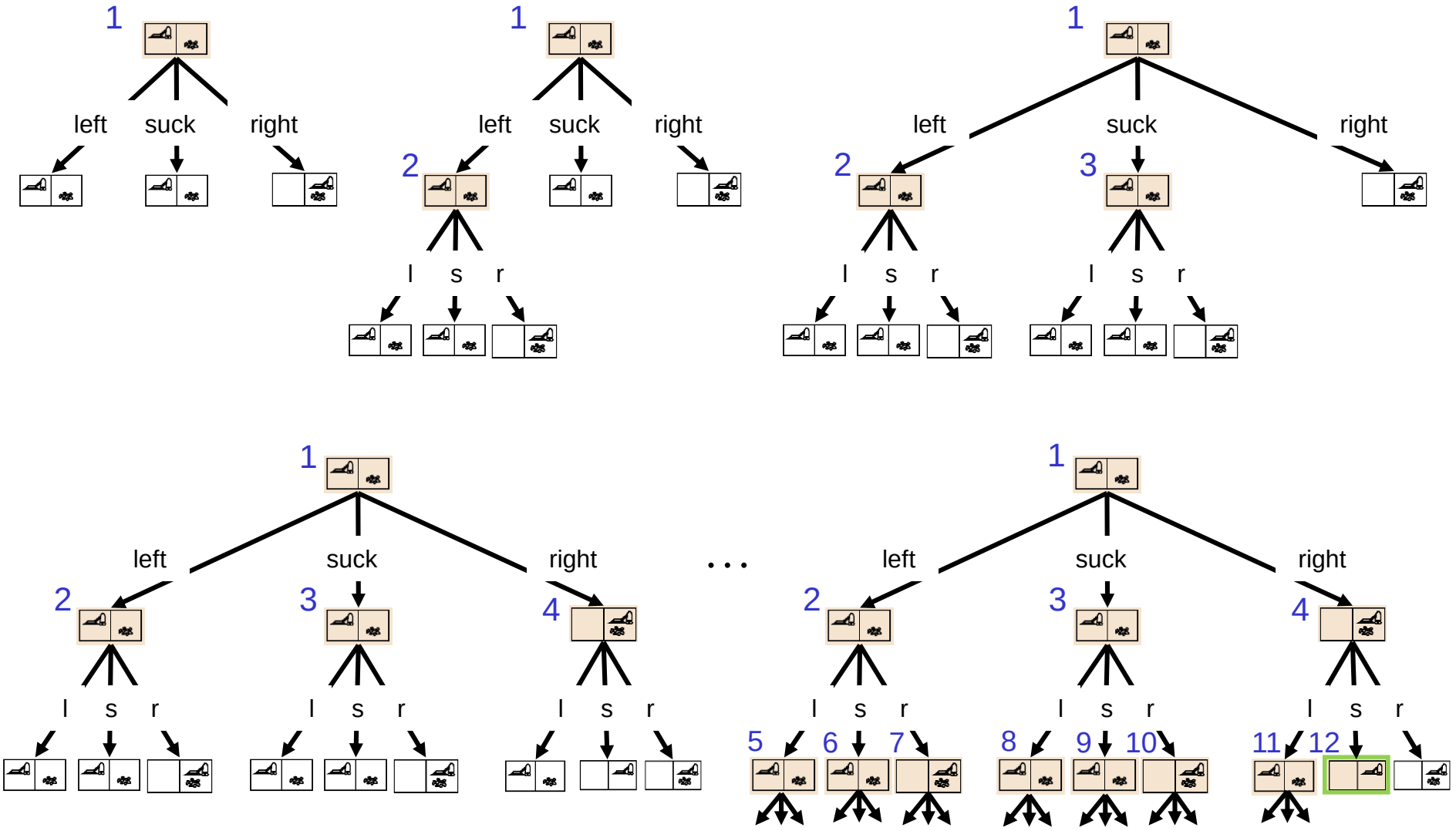
rechts - saugen

Baumsuche




```
// start with root node

// loop
    // get the next node to explore

    // in case of a solution state return solution

    // expand the current node and add child nodes to list of nodes

    // if no more nodes to explore no goal state was found
```

Breitensuche



Breitensuche

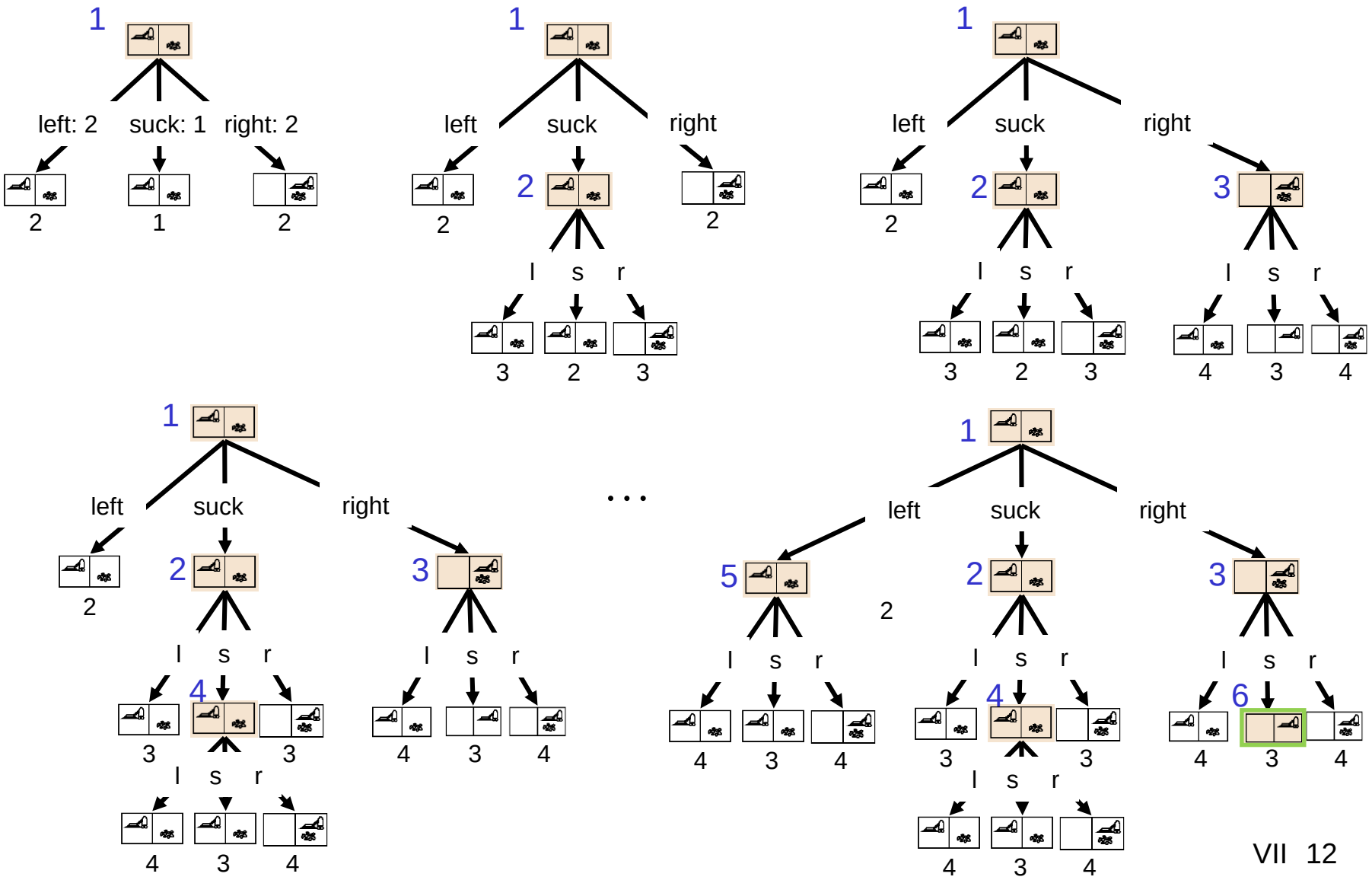
- Die Knoten des Suchbaums werden Ebene für Ebene vollständig untersucht
  - Add nodes: hinten anfügen an Liste der nicht explorierten Knoten
  - Get next node: wähle ältesten nicht explorierten Knoten
- Optimal und vollständig
  - bei gleichen Kosten aller Operatoren
- Speicherplatz- und Laufzeitkomplexität: b^d

Depth	Nodes	Time	Memory
0	1	1 millisecond	100 bytes
2	111	.1 seconds	11 kilobytes
4	11,111	11 seconds	1 megabyte
6	10^6	18 minutes	111 megabytes
8	10^8	31 hours	11 gigabytes
10	10^{10}	128 days	1 terabyte
12	10^{12}	35 years	111 terabytes
14	10^{14}	3500 years	11,111 terabytes

Uniforme Kostensuche

- Breitensuche, bei der die Knoten in der Reihenfolge ihrer Pfadkosten exploriert werden
  - Add nodes: hinten anfügen an Liste der nicht explorierten Knoten
  - Get next node: wähle Knoten mit kleinsten Pfadkosten
  - Oder
  - Add nodes: sortiert aufsteigend nach Pfadkosten einfügen
  - Get next node: wähle vordersten Knoten
- Optimal und vollständig
  - bei verschiedenen Kosten der Operatoren
- Speicherplatz- und Laufzeitkomplexität: b^d

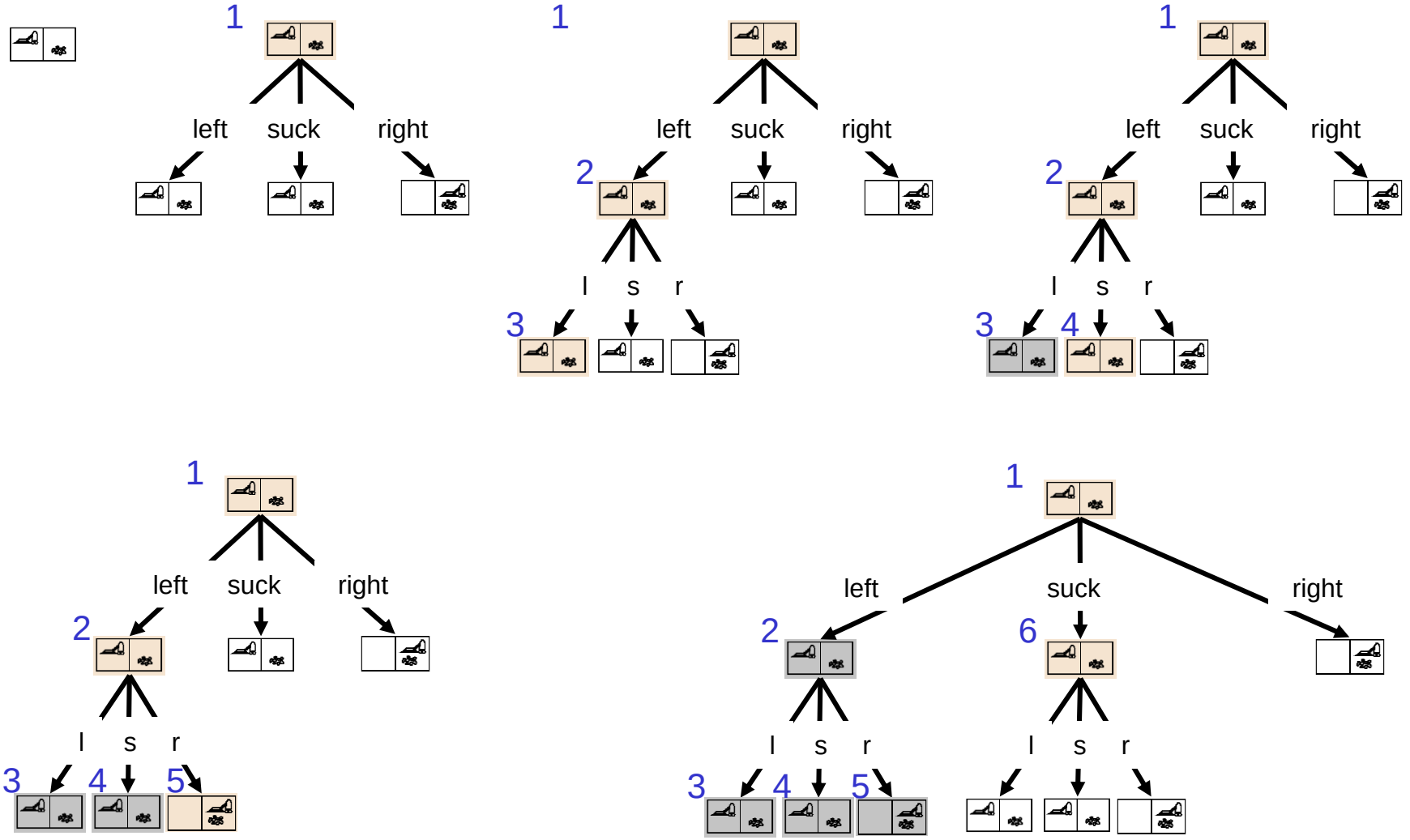
Uniforme Kostensuche



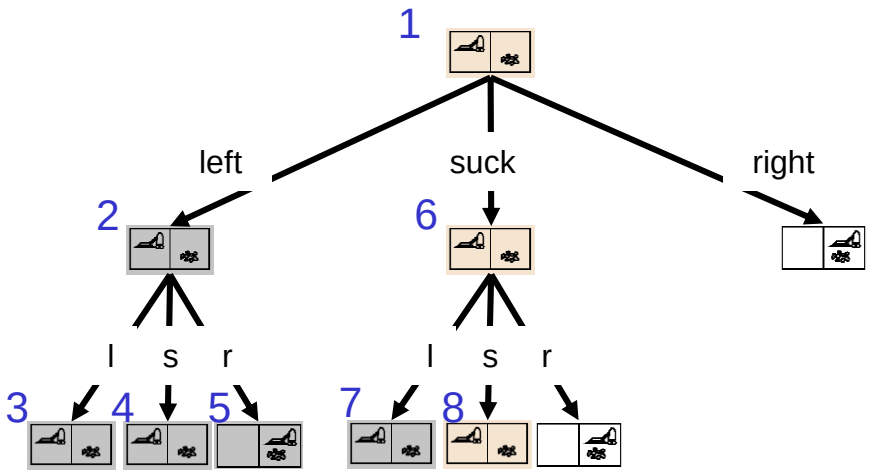
Tiefensuche

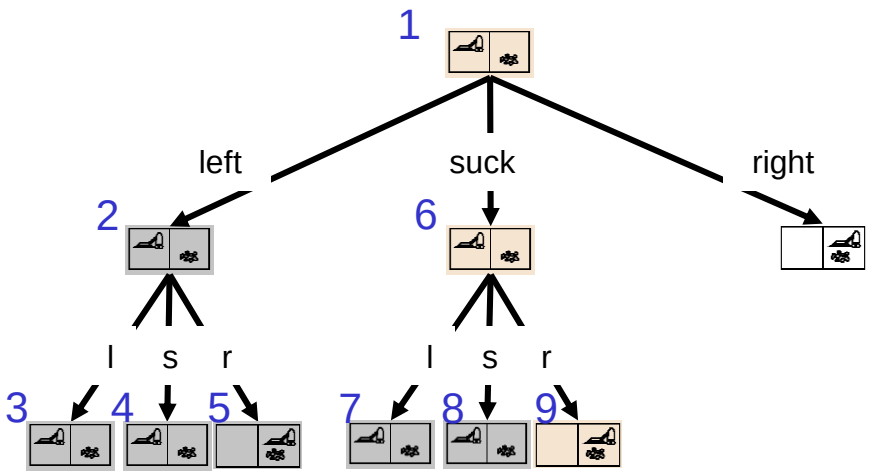
- Ein Pfad wird bis zum Ende verfolgt, bevor der nächste Ast exploriert wird
  - Add nodes: hinten anfügen an Liste der nicht explorierten Knoten
  - Get next node: wähle jüngsten nicht explorierten Knoten
- Ohne Tiefenbeschränkung
  - Kann unendlich tief absteigen
- Mit Tiefenbeschränkung
  - Ignoriere alle Knoten unterhalb einer Tiefe l
- Nicht vollständig
- Nicht optimal
- Laufzeit: b^d
- Speicher: bd

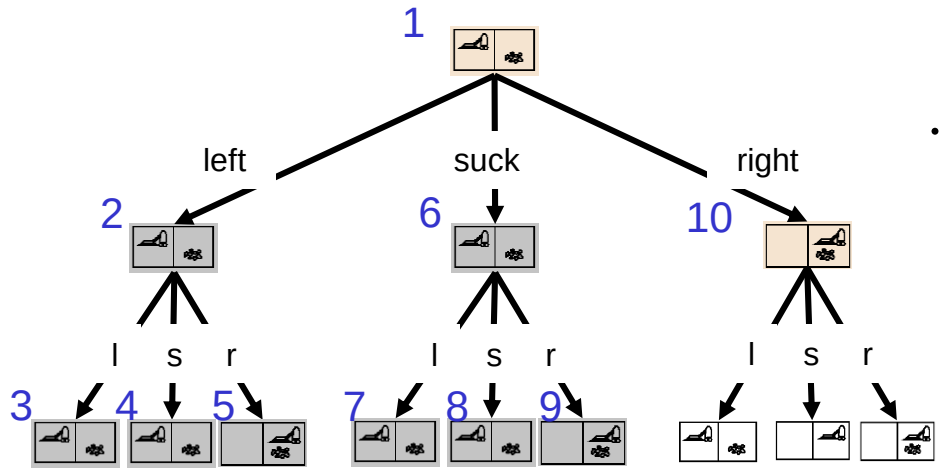
Tiefensuche (mit Tiefenbeschränkung)



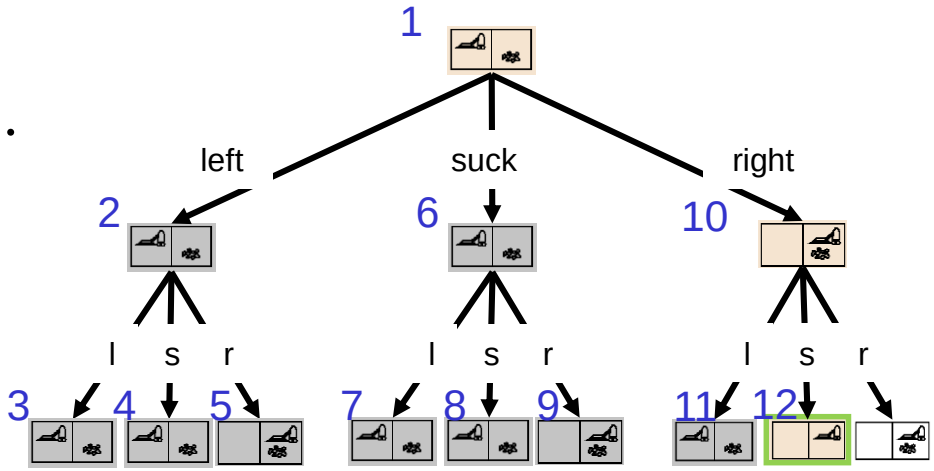
Tiefensuche (mit Tiefenbeschränkung)





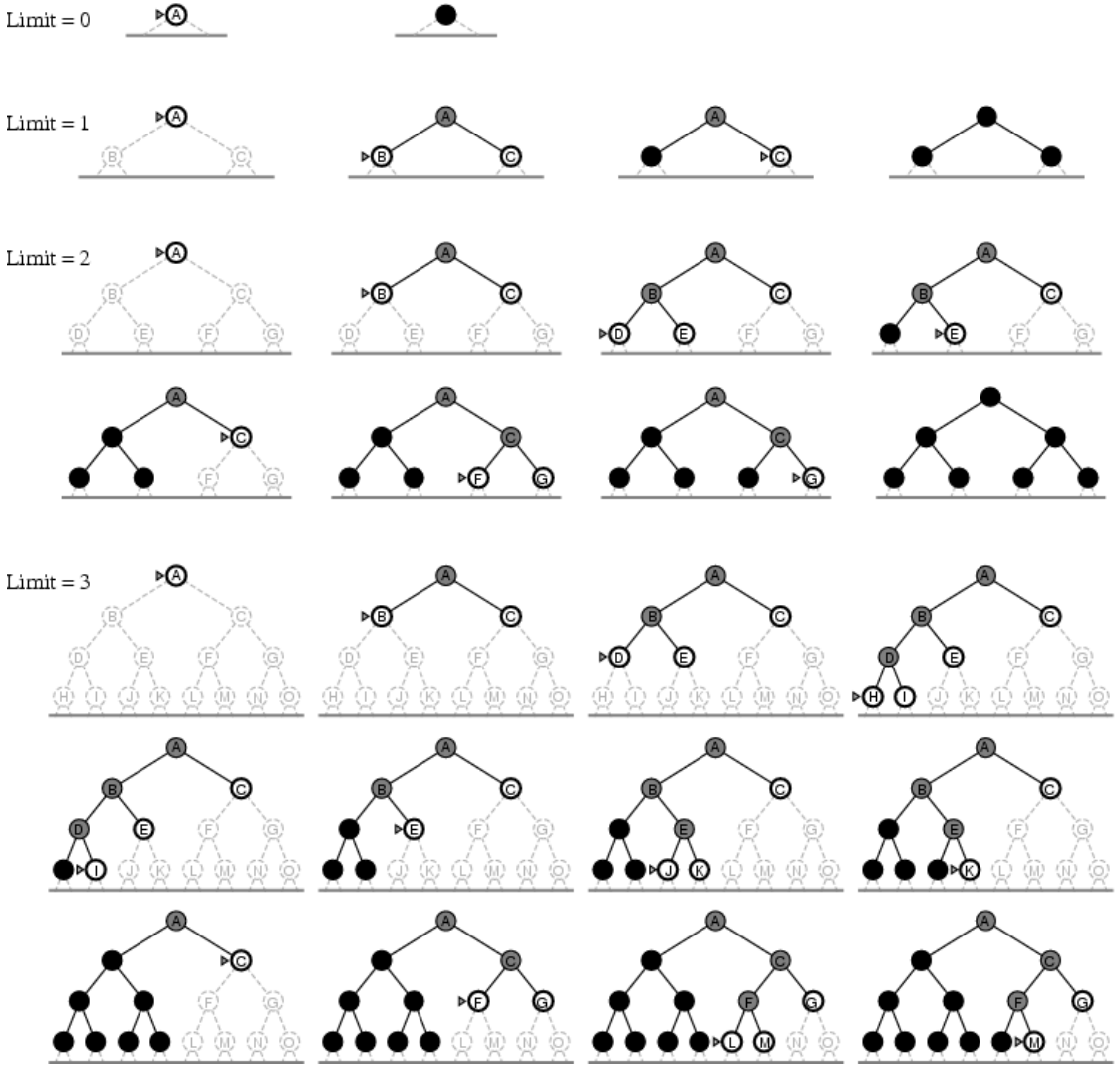


...



Iterative Tiefensuche

- Tiefensuche mit Tiefenbeschränkung, die schrittweise erhöht wird



Zusammenfassung

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening
Complete?	Yes	Yes	No	No	Yes
Time	$O(b^{d+1})$	$O(b^{\lceil C^*/\epsilon \rceil})$	$O(b^m)$	$O(b^l)$	$O(b^d)$
Space	$O(b^{d+1})$	$O(b^{\lceil C^*/\epsilon \rceil})$	$O(bm)$	$O(bl)$	$O(bd)$
Optimal?	Yes	Yes	No	No	Yes

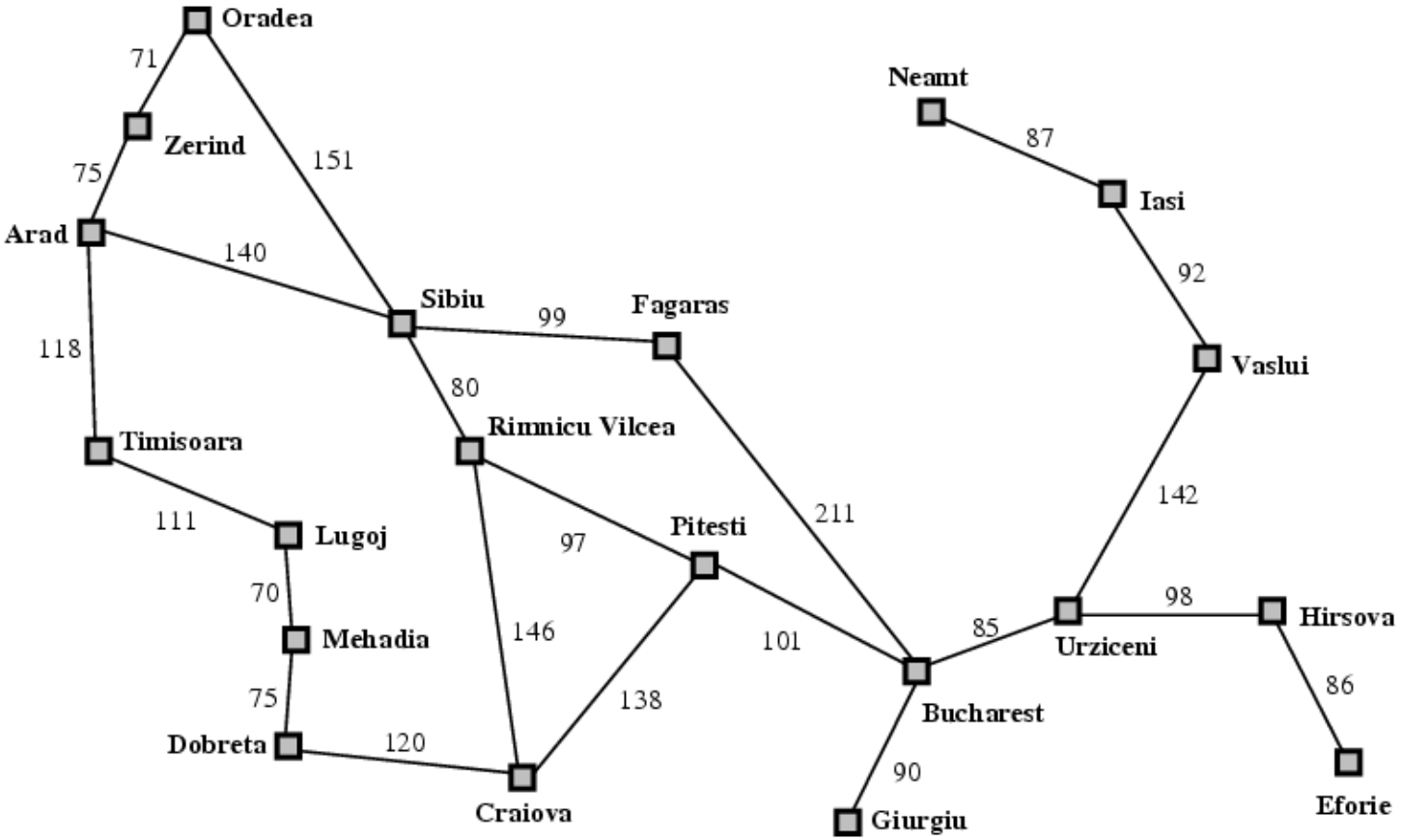
- b = Verzweigungsfaktor
- d = Tiefe
- m = maximale Tiefe
- l = Tiefenlimit

Informierte Suchverfahren

- Uninformierte Suchverfahren sind häufig brutal ineffizient
- Durch die Verwendung von Problem-spezifischem Wissen kann die Effizienz deutlich gesteigert werden
- Heuristiken sollen helfen, die interessantesten Knoten zuerst zu explorieren
- Dazu muss jeder Knoten bewertet werden

Beispiel Problem

- Rumänien

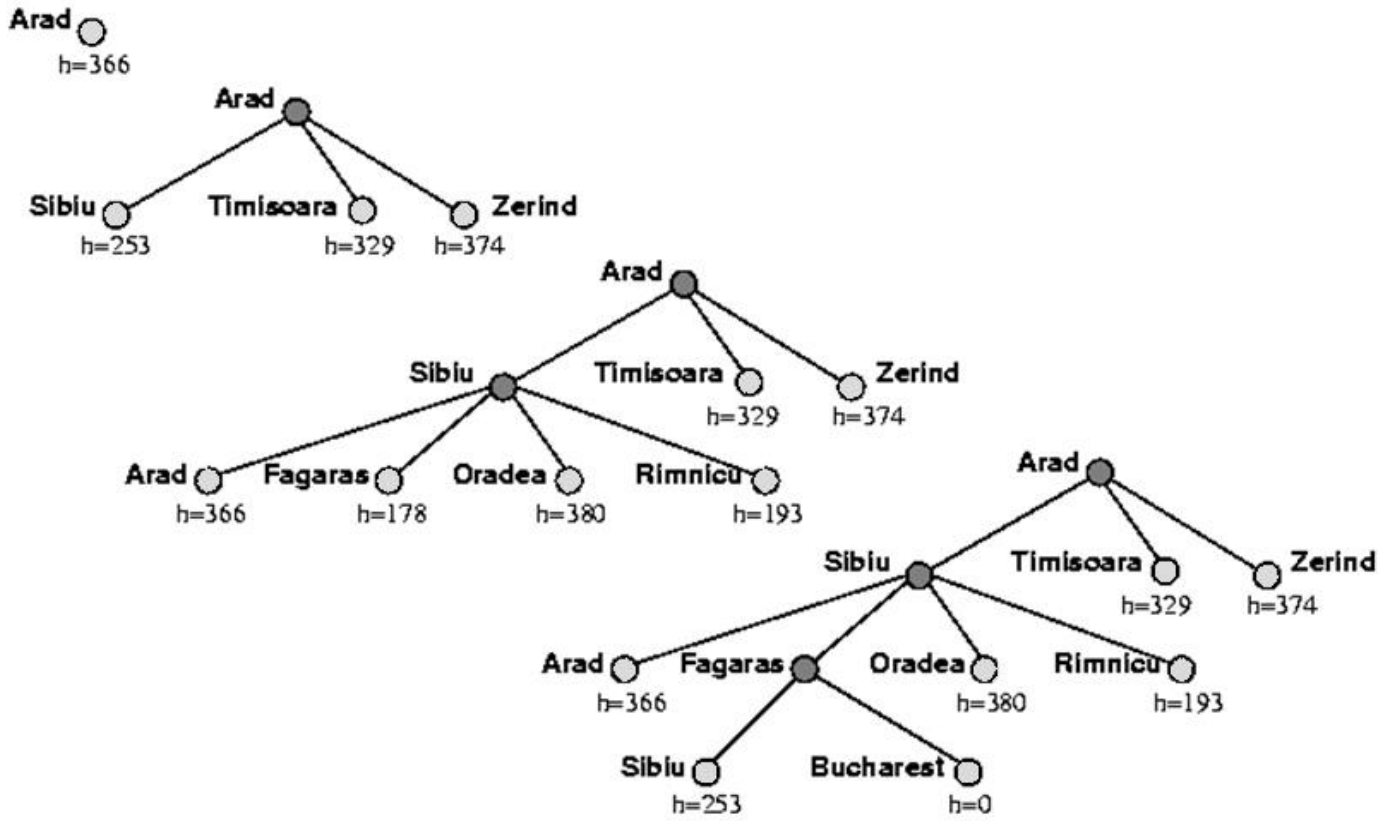


Straight-line-distance nach Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Gierige Suche (Greedy Search)

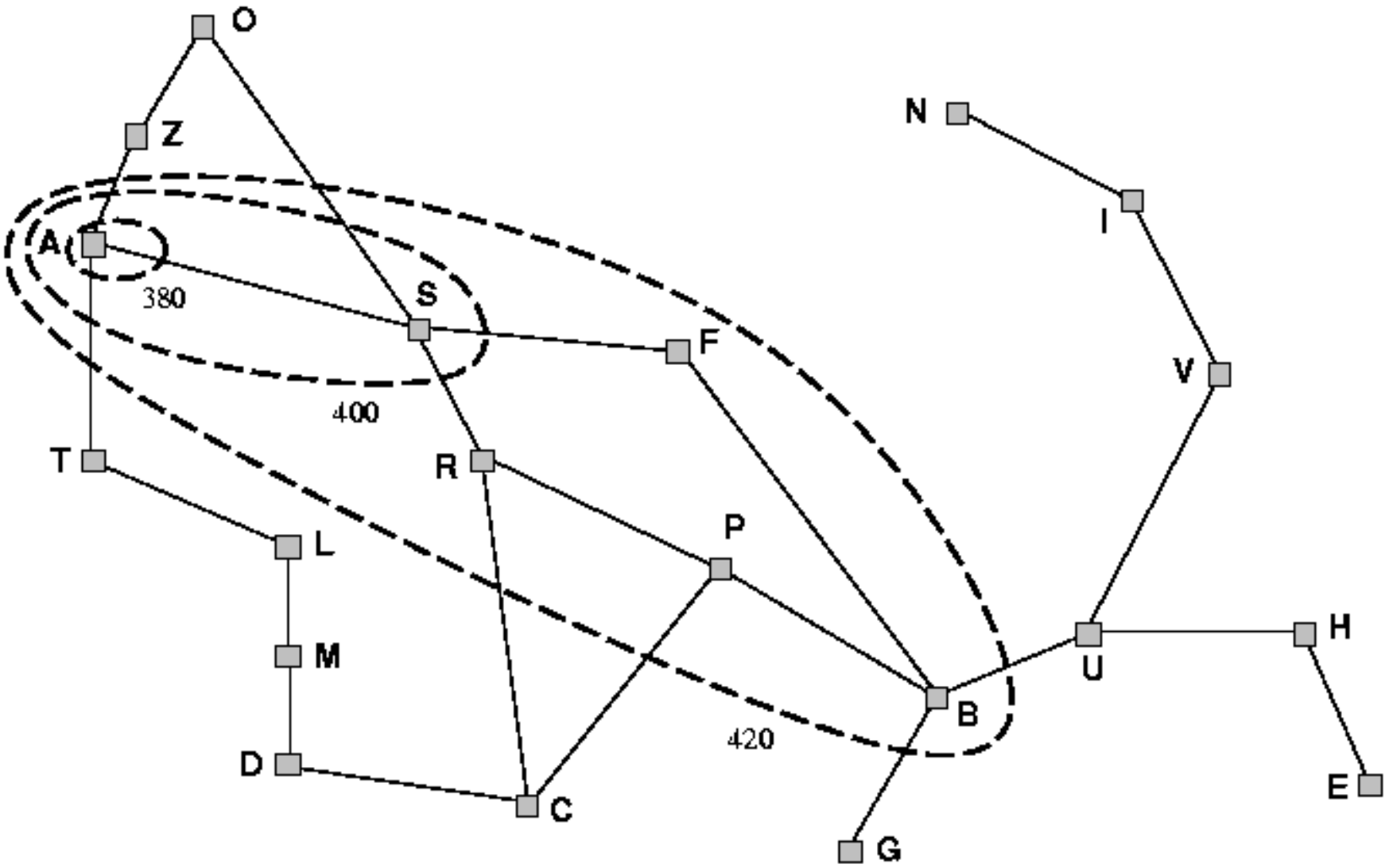
- h(n) ist eine Heuristikfunktion, die die geschätzten Kosten des billigsten Pfads vom Zustand n zum Ziel angibt
- Bei der Greedy Suche wird der Knoten zuerst expandiert, der die geringsten geschätzten Kosten zum Ziel aufweist



A* Suche

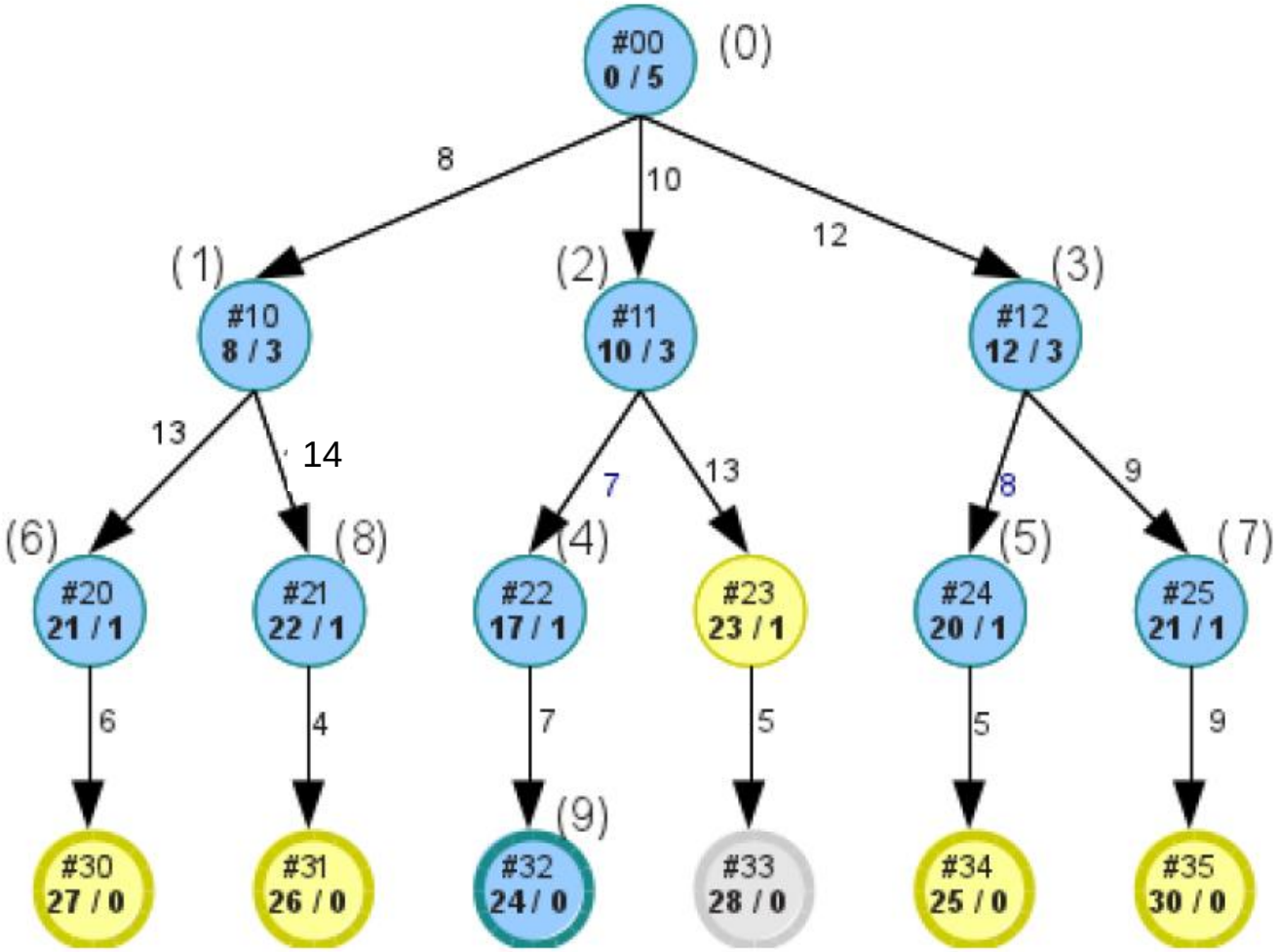
- Uniforme Kostensuche optimiert aufgrund von g(n), ist aber häufig ineffizient
- Greedy Search optimiert aufgrund von h(n), ist aber weder vollständig noch optimal
- A* Suche kombiniert beide und verwendet die Heuristik Funktion f(n) = g(n) + h(n)
- A* Suche ist optimal und vollständig!, wenn h eine unterschätzende Funktion (admissible heuristic) ist, d.h. die erwarteten Kosten zum Ziel sind immer niedriger als die tatsächlichen Kosten
- Dadurch steigt f monoton entlang eines Pfades
- A* ist außerdem optimal effizient, d.h. jeder Algorithmus der weniger Knoten expandiert ist nicht optimal

A* Suche



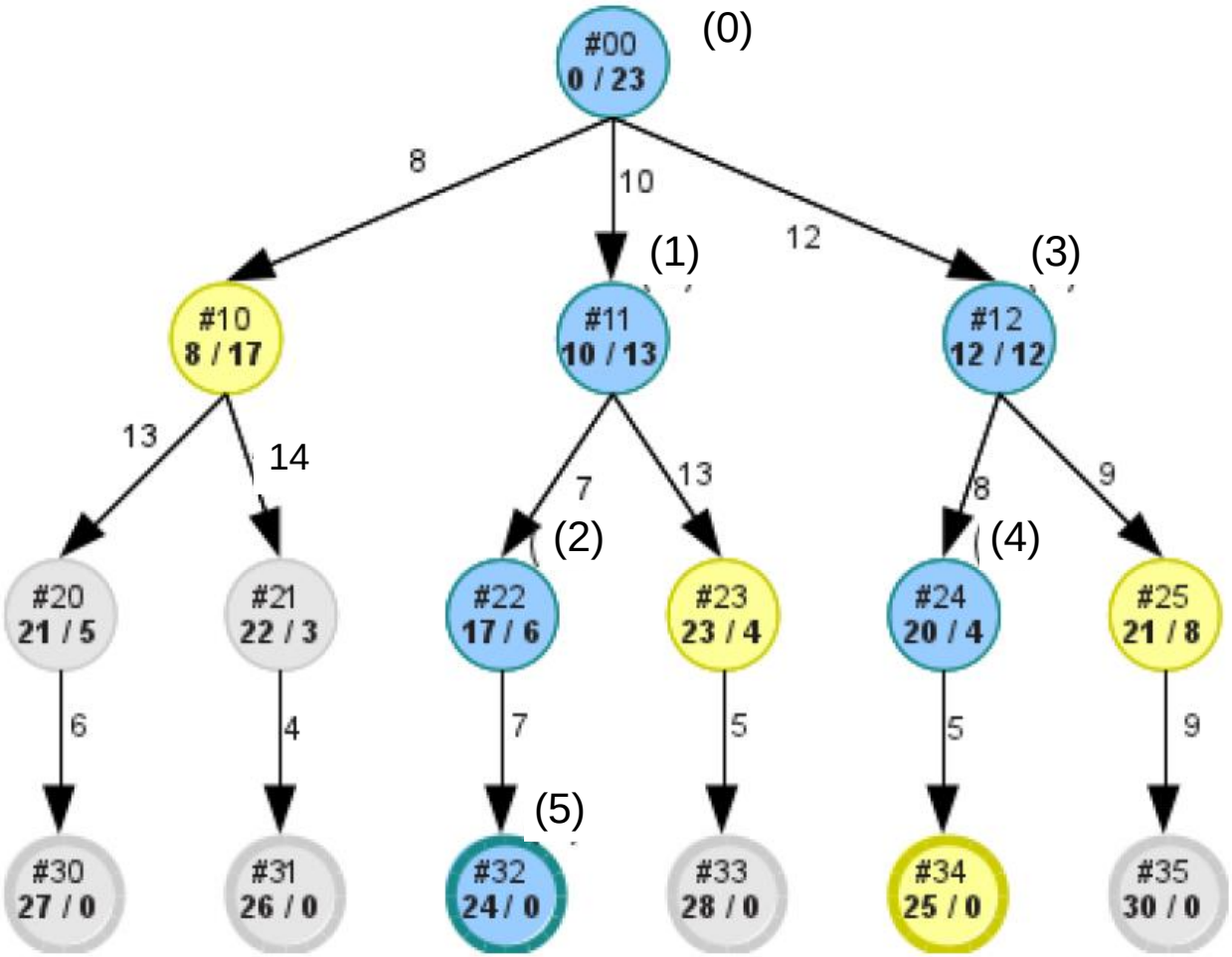
A* Suche

- Stark unterschätzende Heuristik



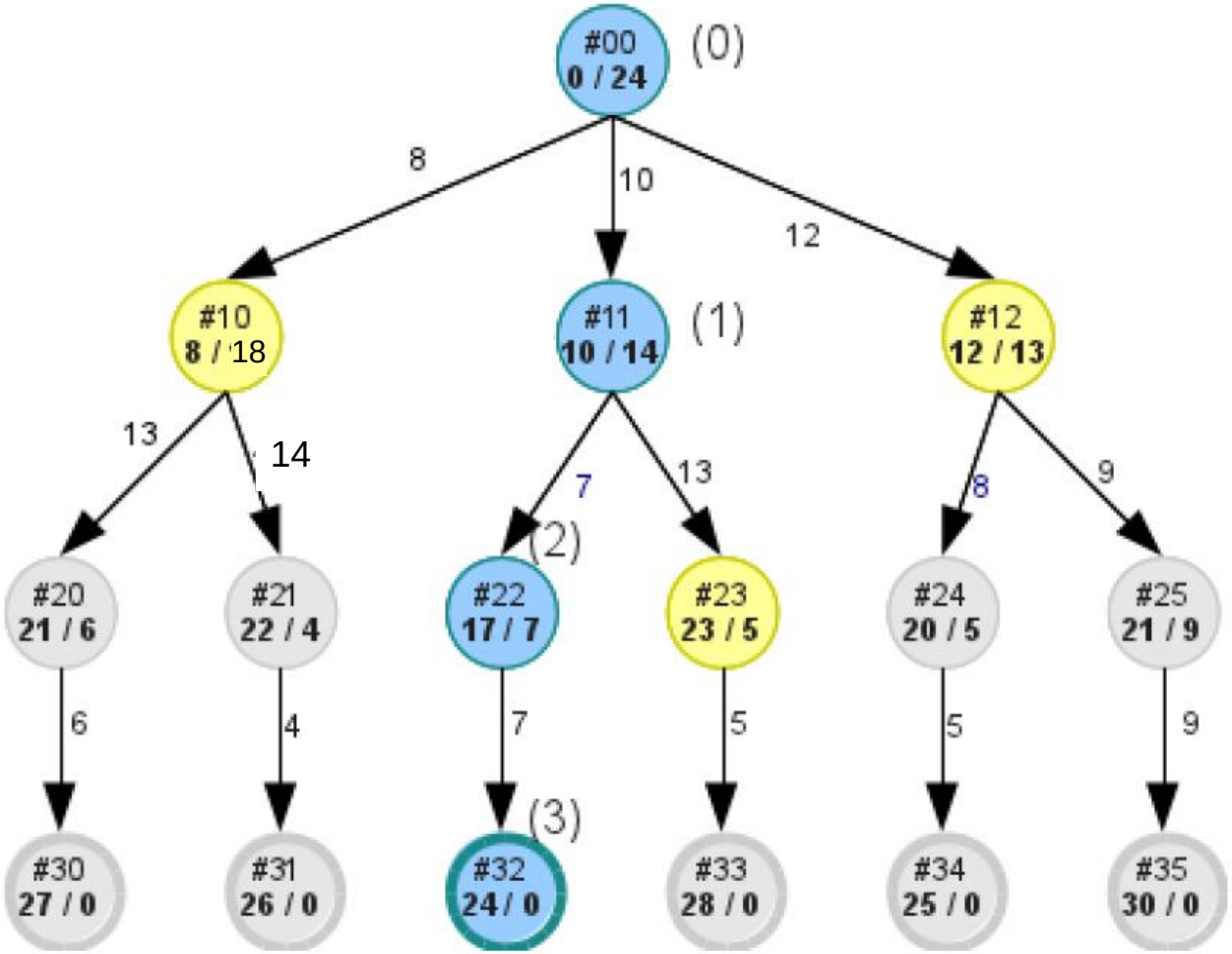
A* Suche

- Bessere Heuristik



A* Suche

- Exakte Heuristik



Vergleich

- Vergleich von iterativer Tiefensuche und A* Suche mit Heuristik h1 und h2 beim 8 puzzle
  - h1: Anzahl Plättchen, die nicht in der Zielposition liegen
  - h2: Summe der Distanzen der Plättchen von der Zielposition

	Search Cost			Effective Branching Factor		
d	IDS	A*(h_1)	A*(h_2)	IDS	A*(h_1)	A*(h_2)
2	l0	6	6	2.45	l.79	l.79
4	l l2	l3	l2	2.87	l.48	l.45
6	680	20	l8	2.73	l.34	l.30
8	6384	39	25	2.80	l.33	l.24
l0	47l27	93	39	2.79	l.38	l.22
l2	364404	227	73	2.78	l.42	l.24
l4	3473941	539	l l3	2.83	l.44	l.23
l6	–	l30l	2l l	–	l.45	l.25
l8	–	3056	363	–	l.46	l.26
20	–	7276	676	–	l.47	l.27
22	–	l8094	l2l9	–	l.48	l.28
24	–	39l35	l64l	–	l.48	l.26