# Kapitel 4. Informierte Suche

## Übersicht

**Zeitplan:**

- insgesamt 1 Doppelstunde.

# 1 Einleitung

# 2 Intelligente Agenten

# 3 Problem-Lösen durch Suchen

# 4 Informierte Suche

**Idee:** Nutze Problem-spezifisches Wissen um die Suche intelligenter zu machen.

**General-Search** ist schon recht weit instantiiert. Einzige Freiheit: Queing-Fn.

## 4.1 Best-First-Suche

Angenommen wir haben eine Evaluierungsfunktion $f$ die jedem Knoten $n$ einen Wert $f(n)$ zuordnet.

Wir ändern nun die Queing-Fn so ab, daß die Knoten mit kleinstem $f$-Wert ganz vorne in der Queue liegen: die Queue wird also gemäss $f$ geordnet.

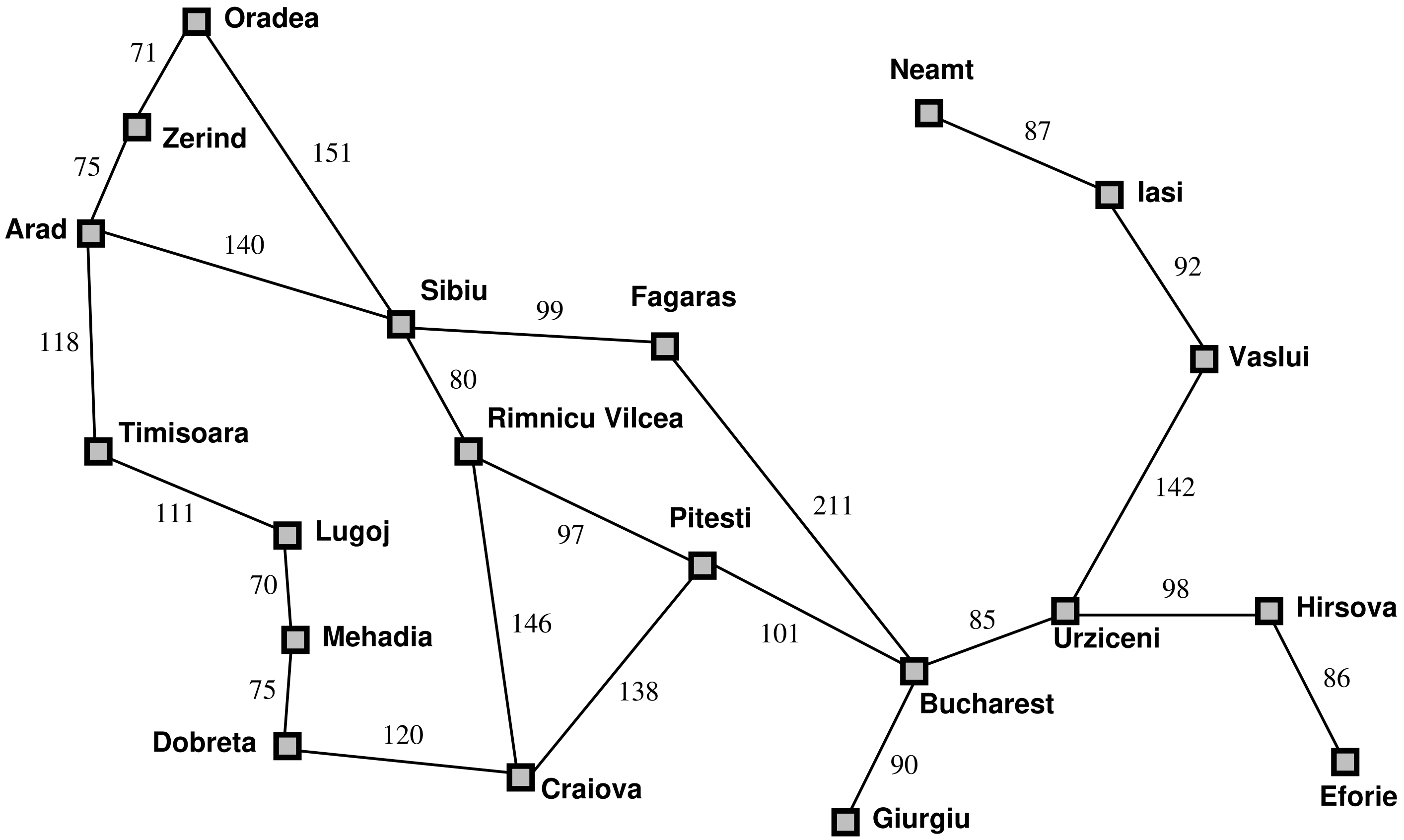
```
function BEST-FIRST-SEARCH(problem, EVAL-FN) returns a solution sequence
  inputs: problem, a problem
          Eval-Fn, an evaluation function

  Queueing-Fn ← a function that orders nodes by EVAL-FN
  return GENERAL-SEARCH(problem, Queueing-Fn)
```

**Greedy-Suche:** Hier benutzt man als Evaluierungsfunktion

Eval_Fn($n$):= die erwarteten Kosten des *optimalen* Pfades vom Zustand in $n$ zum Ziel-Zustand. Optimal bedeutet dabei bezüglich der gegebenen Kostenfunktion $g$.

Diese Evaluierungsfunktion bezeichnet man auch als Heuristikfunktion uns schreibt sie als $h(n)$.

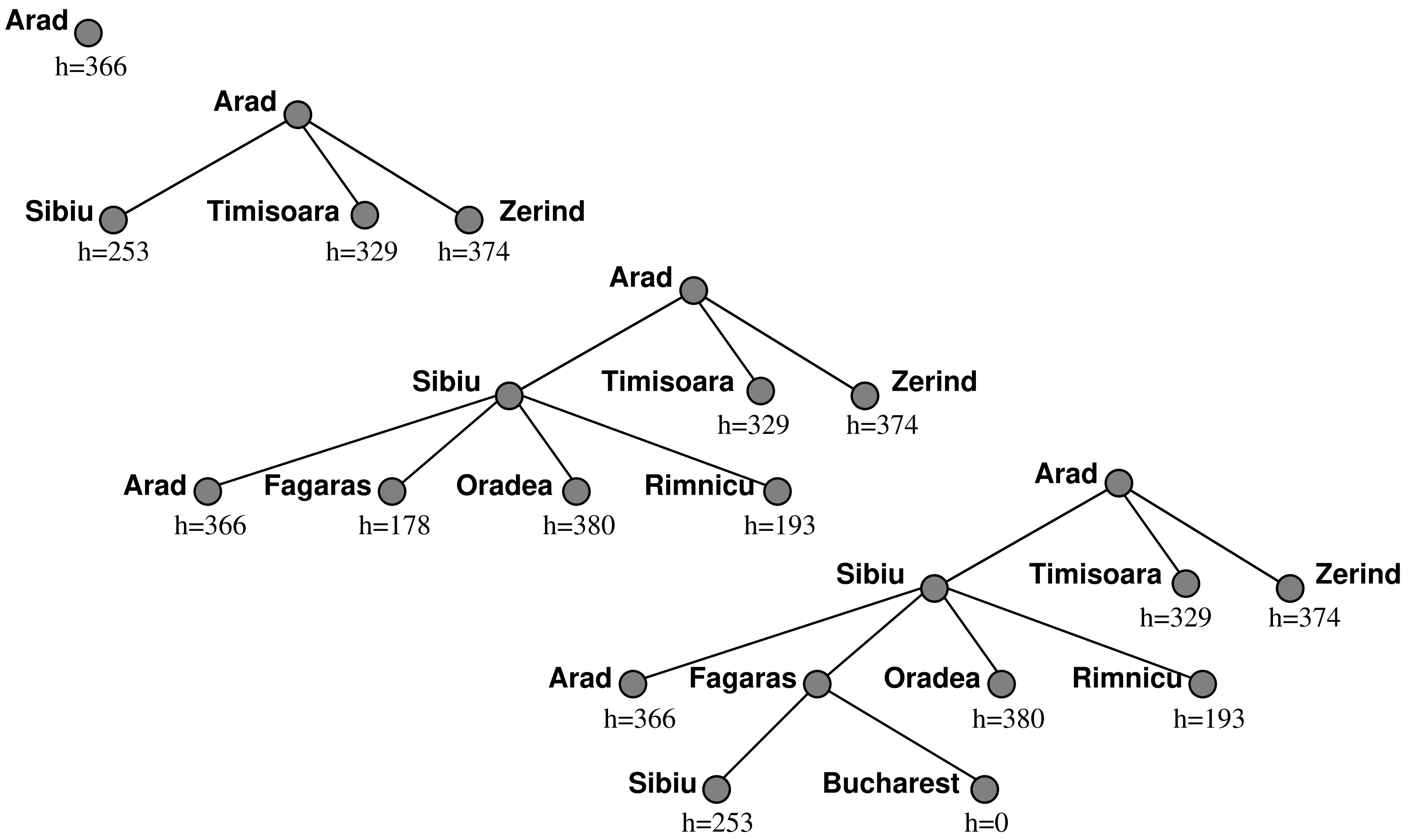
## Beispiel 4.1 (Weg-Suche)




Straight–line distance
to Bucharest

| | |
|---|---|
| **Arad** | 366 |
| **Bucharest** | 0 |
| **Craiova** | 160 |
| **Dobreta** | 242 |
| **Eforie** | 161 |
| **Fagaras** | 178 |
| **Giurgiu** | 77 |
| **Hirsova** | 151 |
| **Iasi** | 226 |
| **Lugoj** | 244 |
| **Mehadia** | 241 |
| **Neamt** | 234 |
| **Oradea** | 380 |
| **Pitesti** | 98 |
| **Rimnicu Vilcea** | 193 |
| **Sibiu** | 253 |
| **Timisoara** | 329 |
| **Urziceni** | 80 |
| **Vaslui** | 199 |
| **Zerind** | 374 |

$h(n)$ *kann als Luftlinien-Entfernung von Budapest bis zu der zu* $n$ *gehörenden Stadt definiert werden.*




Ist die Greedy-Suche optimal?

Ist die Greedy-Suche vollständig?

"Greedy" heisst gierig und ist eine der sieben Todsünden.

Greedy ist nicht optimal (in Beispiel wird ein nicht optimaler Weg gefunden).

Greedy ist auch nicht vollständig, denn es ist Oszillation möglich: betrachte die Aufgabe einen Weg von Iasi nach Fagaras zu finden.

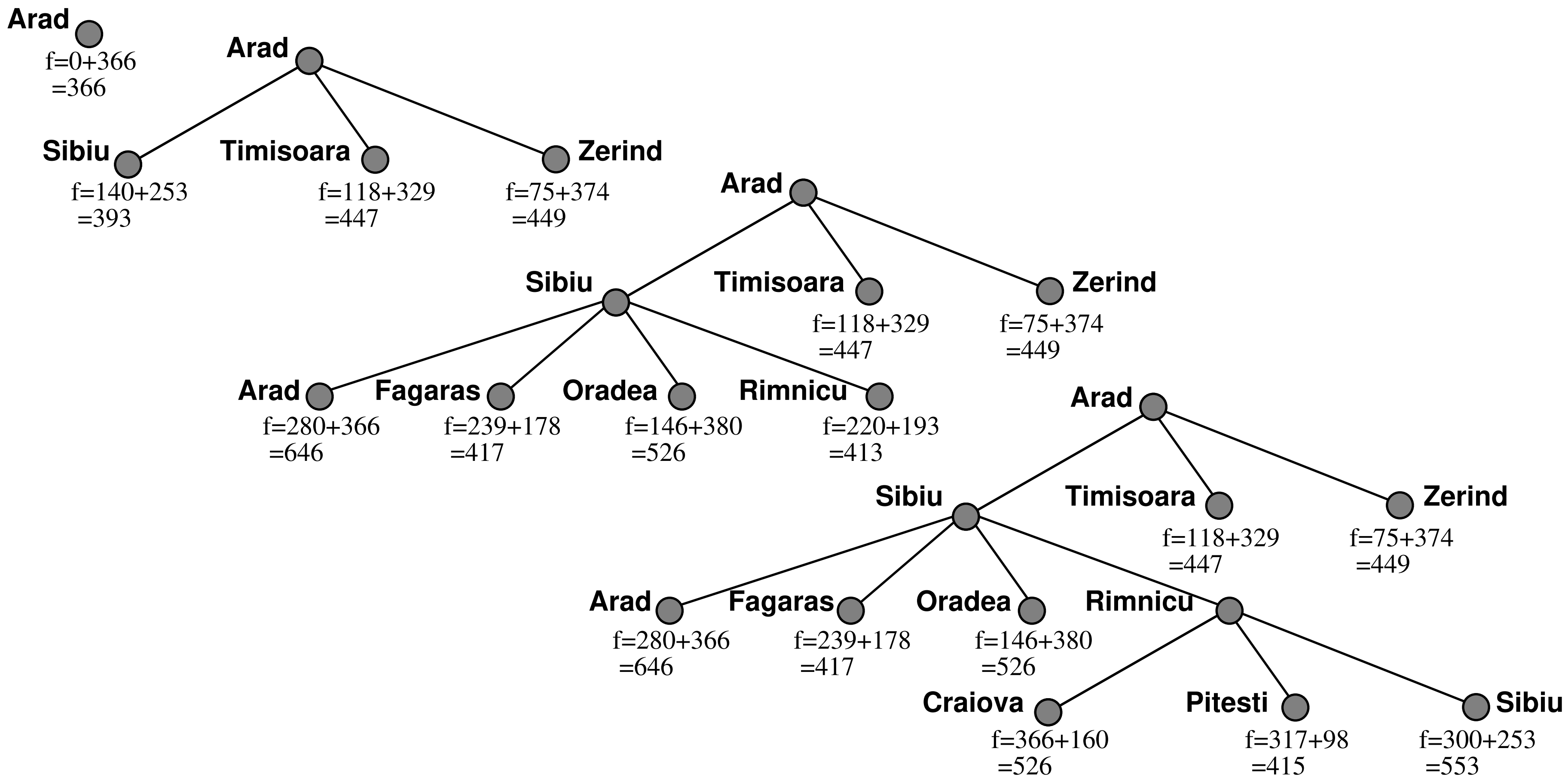
**A*-Suche:** Hier benutzt man als Evaluierungsfunktion Eval_Fn($n$) die Summe aus einer Heuristikfunktion $h(n)$ und den tatsächlichen Pfadkosten $g(n)$ :

$$\text{Eval\_Fn}(n := h(n) + g(n)$$

Diese Evaluierungsfunktion schreiben wir auch einfach als $f(n)$.

A*-Suche ist also "Greedy + Uniform-Cost", denn für Zielzustände $n_z$ gilt ja $h(n_z) = 0$, also Eval_Fn($n_z$) $= g(n_z)$.

Wir stellen allerdings eine Bedingung an die Heuristikfunktion $h$: $h(n)$ soll immer kleiner sein als die optimalen Kosten von $n$ zu einem Ziel-Knoten ($h(n) < g(n_z) - g(n)$)


Arad
f=0+366
=366
Arad
Sibiu
f=140+253
=393
Timisoara
f=118+329
=447
Zerind
f=75+374
=449
Arad
Sibiu
Timisoara
f=118+329
=447
Zerind
f=75+374
=449
Arad
f=280+366
=646
Fagaras
f=239+178
=417
Oradea
f=146+380
=526
Rimnicu
f=220+193
=413
Arad
Sibiu
Timisoara
f=118+329
=447
Zerind
f=75+374
=449
Arad
f=280+366
=646
Fagaras
f=239+178
=417
Oradea
f=146+380
=526
Rimnicu
Craiova
f=366+160
=526
Pitesti
f=317+98
=415
Sibiu
f=300+253
=553

In unserem Beispiel ist $f :=$ Eval_Fn monoton. Im allgemeinen gilt das nicht.

Allerdings können wir $f$ leicht monoton wachsend machen:

**Definition 4.2 ($f \leadsto$ monotones$f$)**
*Wir definieren eine modifizierte Funktion $f^{mon}$ rekursiv nach der Knotentiefe*

$$f^{mon}(n) := max\{f^{mon}(n'), f(n)\}$$

*wobei $n'$ der Elter von $n$ ist.*

**Theorem 4.3 (Optimalität von A$^*$)**
*A$^*$ ist optimal.*

Hier ein Beispiel für nichtmonotones $f$. Sei $n'$ der Elter von $n$. Sei $g(n') = 3$ und $g(n) = 4$, und $h(n') = 4$ und $h(n) = 2$. Damit ist $f(n') = 7$ und $f(n) = 6$. Kurze Zeichnung dazu.

**Beweis:** Sei G ein optimaler Zielzustand, und $G'$ ein suboptimaler, d.h. $g(G') > g(G)$.

Angenommen, A* hat $G'$ aus der Queue ausgewählt. Dann hat A* irgendwann einmal einen Knoten $n_0$ der zum optimalen Pfad nach $G$ führt "verschmäht" (also nicht expandiert), d.h. $n_0$ kam in die Queue. Nun gilt aber

$$g(G) \geq f(n_0)$$

wegen der Bedingung an $h$, und es gilt auch

$$f(n_0) \geq f(G')$$

da ja $G'$ und nicht $n_0$ (der noch in der Queue sein muss) ausgewählt wurde.

Nimmt man beide Gleichungen zusammen erhält man

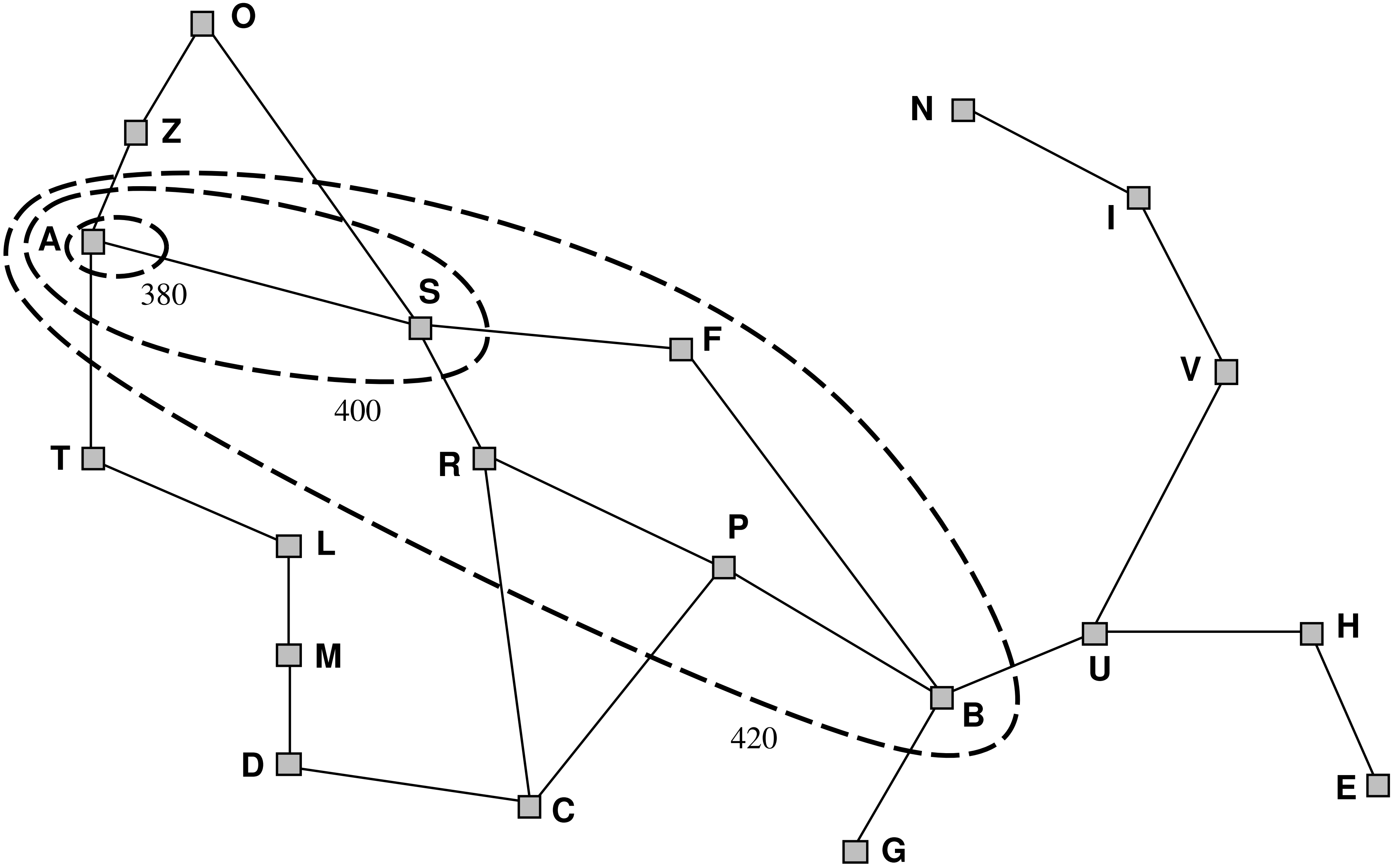
$$g(G) \geq f(G') = g(G')$$

was ein Widerspruch zu unserer Annahme ist. ∎

Für die Vollständigkeit müssen wir 2 Fälle ausschliessen:

1. Es werden auf einen Schlag unendlich viele Knoten erzeugt.
2. Es werden Knoten erzeugt deren Evaluierungsfunktion zwar wächst, aber unter einem bestimmten Wert ($g(Ziel)$) bleibt.

**Theorem 4.4 (Vollständigkeit von A$^*$)**
*A$^*$ ist vollständig für lokal endliche Graphen mit:* $\exists \delta > 0 :$ *jeder Operator erhöht* $g(n)$ *um mindestens* $\delta$*.*


O
Z
A
380
S
F
400
T
R
L
P
M
D
C
N
I
V
U
H
B
420
G
E

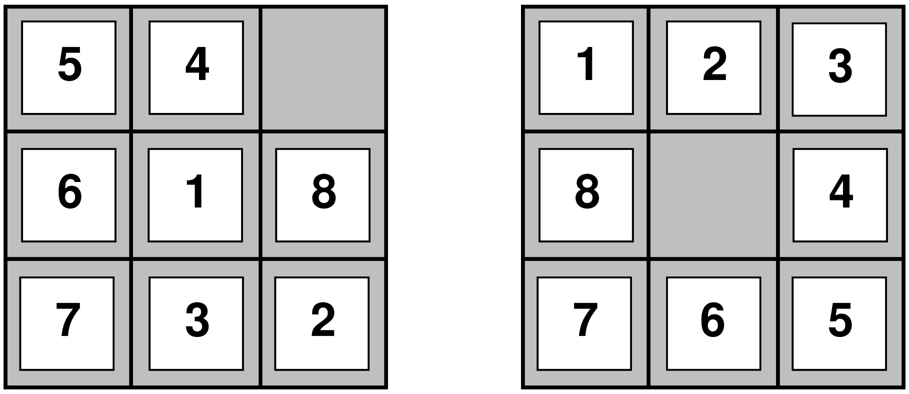
$A^*$ zieht Konturen die zielgerichtet sind. Was macht Uniform-Cost? Uniform-Cost zieht konzentrische Kreise!

Die Anzahl der Knoten im Speicher ist exponentiell in der Länge der Lösung. Auch bei der A*-Suche ist der Speicher das Problem, nicht die Zeit.

Dies führt zu

- **IDA*:** A* + Iterative Deepening,
- **SMA*:** Simplified Memory Bounded A* ($\leadsto$ Abschnitt 4.3)

## 4.2 Beispiele für Heuristiken




Start State Goal State

Was ist eigentlich der Verzweigungsfaktor in diesem Problem?

**Ungefähr 3.**

Wieviele Zustände müssen schlimmstenfalls betrachtet werden?

$$3^g \approx 10^{10}$$

wobei $g$ die Anzahl der Züge ist, um zu einer Lösung zu gelangen. $g$ ist ungefähr 20.

Aber: Es gibt ja nur $9\mathbf{!} \approx 10^5$ Zustände! Mit anderen Worten: Testen auf Schleifen hilft sehr viel.

Welche Heuristikfunktionen bieten sich an?

**Hamming-Distanz:** $h_1$ ist die Anzahl der Zahlen in falscher Position. Z.B. $h_1(Anfang) = 8$.

**Manhattan-Distanz:** Wir rechnen für jedes Teil die Entfernung zur richtigen Position aus und summieren diese:

$$h_2 := \sum_{i=1}^{8} \text{Entfernung von } i \text{ zur richtigen Position}$$

Z.B. $h_2(Anfang) = 2 + 3 + 2 + 1 + 2 + 2 + 1 + 2 = 15$.

Strenggenommen gibt es keinen einheitlichen Verzweigungsfaktor. 1 Zustand hat VF 4 (das Kästchen in der Mitte), die 4 Eckzustände haben VF 2, und die 4 übrigen Zustände haben 3.

Wie kann man die Qualität einer Heuristik messen? Möglichst in einem einzigen Wert!

**Definition 4.5 (Effektiver Verzweigungsfaktor $b^*$)**
*Angenommen, $A^*$ findet die optimale Lösung für eine Problem Instanz in der Tiefe $d$ und musste $N$ Knoten expandieren. Dann heisst $b^*$ definiert durch*

$$N = 1 + b^* + (b^*)^2) + \ldots + (b^*)^d)$$

*der effektive Verzweigungsfaktor von $A^*$.*

Man beachte, daß $b^*$ nicht nur von $h$ sondern auch von der speziellen Problem-Instanz abhängt.

Allerdings stellt man fest, daß $b^*$ ziemlich konstant für grosse Klassen von Problem-Instanzen ist.

| | Search Cost | | | Effective Branching Factor | | |
|---|---|---|---|---|---|---|
| $d$ | IDS | A*($h_1$) | A*($h_2$) | IDS | A*($h_1$) | A*($h_2$) |
| 2 | 10 | 6 | 6 | 2.45 | 1.79 | 1.79 |
| 4 | 112 | 13 | 12 | 2.87 | 1.48 | 1.45 |
| 6 | 680 | 20 | 18 | 2.73 | 1.34 | 1.30 |
| 8 | 6384 | 39 | 25 | 2.80 | 1.33 | 1.24 |
| 10 | 47127 | 93 | 39 | 2.79 | 1.38 | 1.22 |
| 12 | 364404 | 227 | 73 | 2.78 | 1.42 | 1.24 |
| 14 | 3473941 | 539 | 113 | 2.83 | 1.44 | 1.23 |
| 16 | – | 1301 | 211 | – | 1.45 | 1.25 |
| 18 | – | 3056 | 363 | – | 1.46 | 1.26 |
| 20 | – | 7276 | 676 | – | 1.47 | 1.27 |
| 22 | – | 18094 | 1219 | – | 1.48 | 1.28 |
| 24 | – | 39135 | 1641 | – | 1.48 | 1.26 |

## 4.3 SMA*-Suche

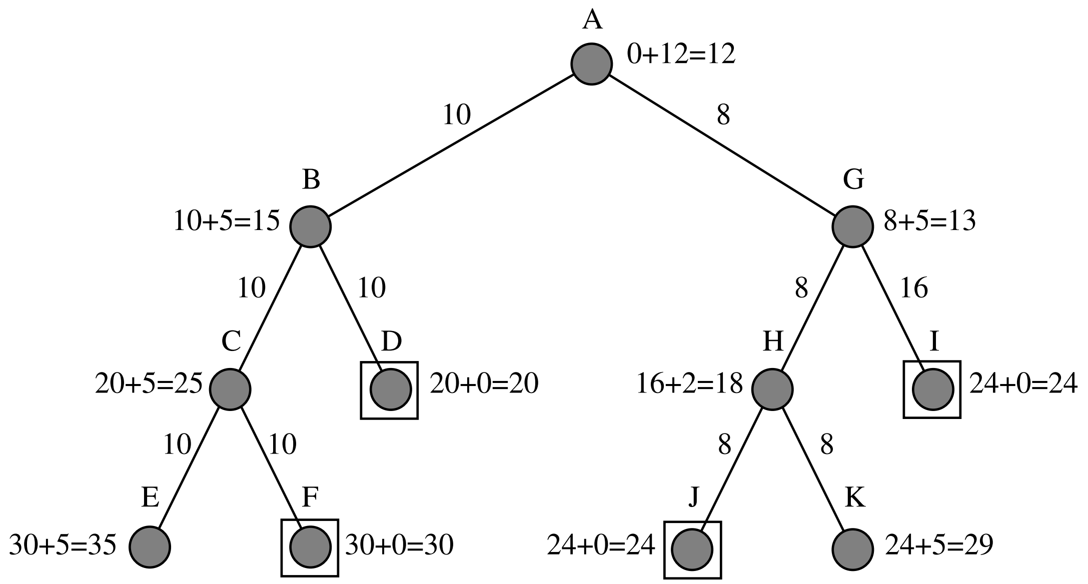
A* braucht viel Speicher. Was, wenn nur beschränkter Speicher vorhanden ist? Wenn also die Queue nur eine maximale Größe haben kann?

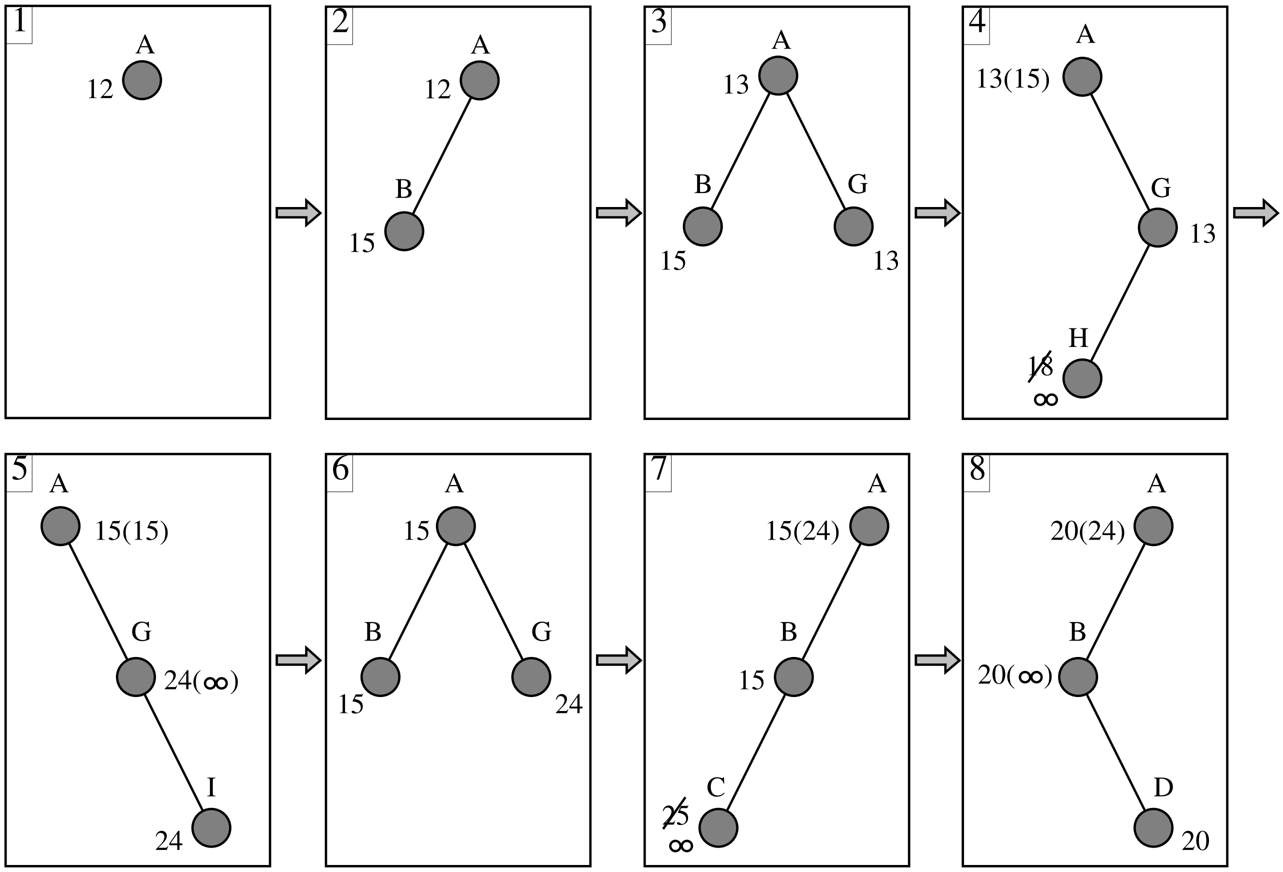
SMA* ist eine Erweiterung von A* die mit beschränktem Speicher auskommt. Falls kein Platz mehr da ist, aber Knoten expandiert werden müssen, werden Knoten aus der Queue gestrichen:

- und zwar solche mit möglichst großen $f$-Kosten (**forgotten nodes**). Deren $f$-Kosten merkt man sich aber. Später werden diese Knoten nur dann wieder betrachtet, wenn alle anderen Pfade zu höheren Kosten führen.

**Theorem 4.6 (Eigenschaften von SMA*)**
*SMA* ist vollständig, falls genügend Speicher da ist um den* **kürzesten** *Lösungspfad zu speichern.*

*SMA* ist optimal, falls genügend Speicher da ist um den* **optimalen** *Lösungspfad zu speichern.*


A
0+12=12
10
8
B
10+5=15
G
8+5=13
10
10
8
16
C
20+5=25
D
20+0=20
H
16+2=18
I
24+0=24
10
10
8
8
E
30+5=35
F
30+0=30
J
24+0=24
K
24+5=29


1
A
12
2
A
12
B
15
3
A
13
B
15
G
13
4
A
13(15)
G
13
H
18
∞
5
A
15(15)
G
24(∞)
I
24
6
A
15
B
15
G
24
7
A
15(24)
B
15
C
25
∞
8
A
20(24)
B
20(∞)
D
20

```
function SMA*(problem) returns a solution sequence
  inputs: problem, a problem
  static: Queue, a queue of nodes ordered by f-cost

  Queue ← MAKE-QUEUE({MAKE-NODE(INITIAL-STATE[problem])})
  loop do
     if Queue is empty then return failure
     n ← deepest least-f-cost node in Queue
     if GOAL-TEST(n) then return success
     s ← NEXT-SUCCESSOR(n)
     if s is not a goal and is at maximum depth then
        f(s) ← ∞
     else
        f(s) ← MAX(f(n), g(s)+h(s))
     if all of n's successors have been generated then
        update n's f-cost and those of its ancestors if necessary
     if SUCCESSORS(n) all in memory then remove n from Queue
     if memory is full then
        delete shallowest, highest-f-cost node in Queue
        remove it from its parent's successor list
        insert its parent on Queue if necessary
     insert s on Queue
  end
```