# Keresési módszerek

# Fogalmak, példák

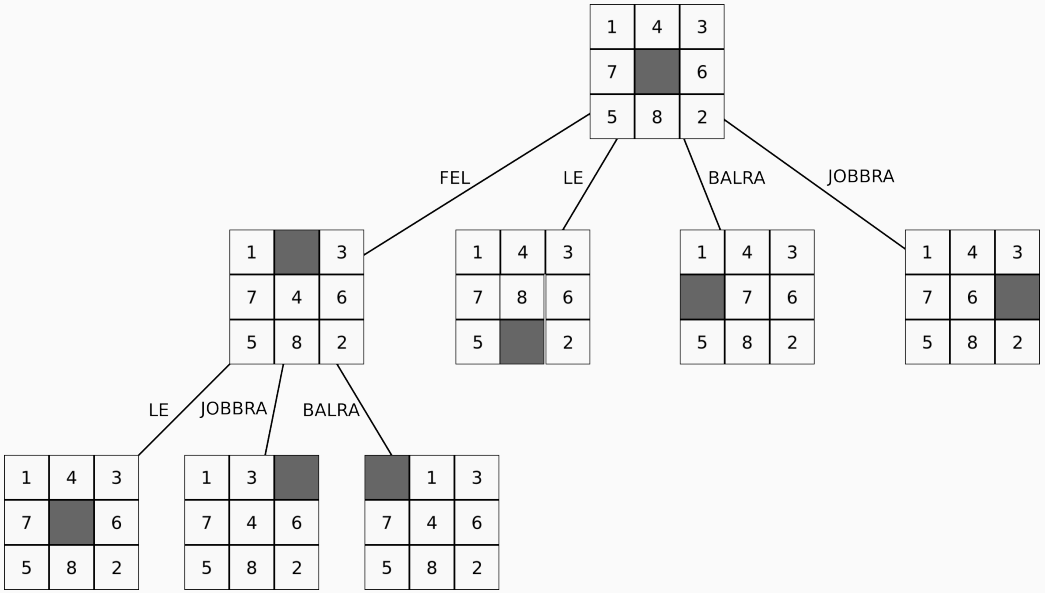
- word ladder: **table**-able-ale-all-hall-hail-hair-**chair**
- Knuth-féle sejtés: a gyökvonás–, egész érték– és faktoriális függvényeknek van az a sorozata, amellyel a 4-es számból bármely másik természetes szám előállítható, pl.

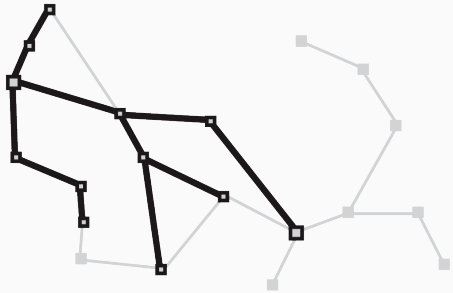
$$\left\lfloor \sqrt{\sqrt{\sqrt{\sqrt{\sqrt{(4!)!}}}}} \right\rfloor = 5$$

- kiinduló állapot, műveletek, végállapot (megoldás)

# Fogalmak, példák (2)

3. állapottér, keresési fa



4. útköltség (path cost)
5. stratégia: teljesség, időigény, tárigény, optimalitás
6. nem informált (vak) keresés: szélességi, egyenletes költségű, mélységi, mélységkorlátozott, iteratívan mélyülő, kétirányú
7. informált keresés: mohó, A*, IDA*

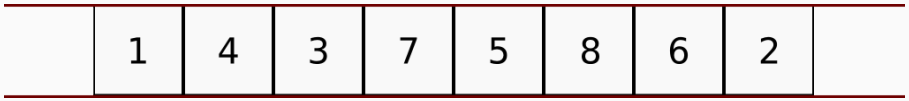
# Nem informált keresés

Ismert elemek:

- csomópont kibontása, műveletek, útköltség
- végállapot teszt

Stratégia lényege:

- a kibontásra váró csomópontok (perem — frontier) tárolására használt adatstruktúra (Q)
- Q-ba való beszúrás / törlés



# Szélességi keresés

- adatstruktúra: sor

```
NODE breadth_first_search(NODE root) {
  QUEUE queue(root);
  while(!queue.is_empty()) {
    NODE act = queue.pop_front();
    if (is_solution(act)) return act;
    foreach(NODE child in act.children())
      queue.push_back(child);
  }
  return null;
}
```

- fa elágazási tényezője $b$, megoldás a $d$-ik szinten
- tulajdonságok: teljes, nem optimális, $O(b^d)$ memóriaigény, $O(b^d)$ időigény

# Egyenletes költségű keresés

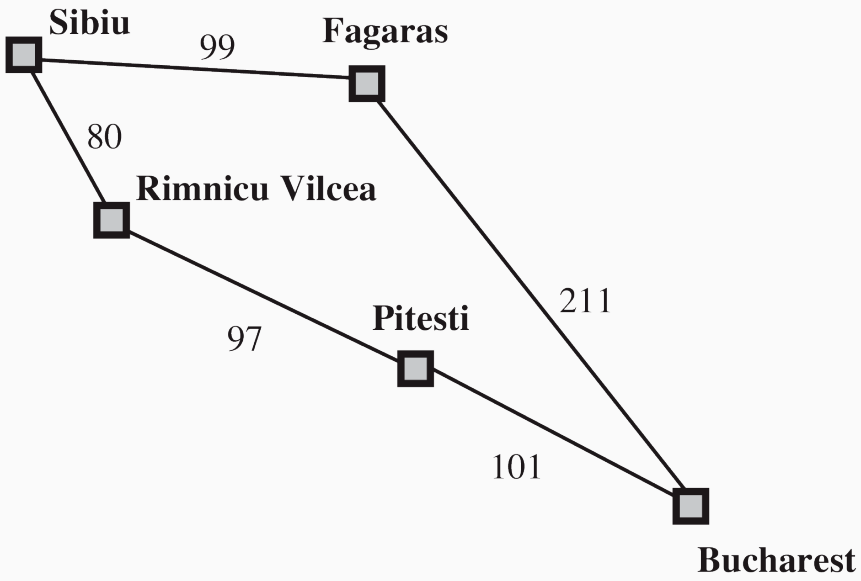

- **útköltség**, $g(n)$
- adatstruktúra: elsőbbségi sor, kritérium: $\arg\min_n g(n)$

```
NODE uniform_cost_search(NODE root, COSTFUN g) {
  QUEUE queue(root, g);
  while(!queue.is_empty()) {
    NODE act = queue.pop_min();
    if (is_solution(act)) return act;
    foreach(NODE child in act.children())
      queue.push_back(child);
  }
  return null;
}
```

- ha $\forall i,\ g(child_i(X)) \geq g(X)$, akkor **optimális**, exponenciális tárigény, exponenciális időigény

# Egyenletes költségű keresés (2)

- Szeben → Bukarest



# Mélységi keresés

- adatstruktúra: verem

```
NODE depth_first_search(NODE x) {
  if (is_solution(x)) return x;
  foreach(NODE child in x.children())
    if (depth_first_search(child) != null) return child;

  return null;
}
```

- tulajdonságok: lineáris memóriaigény, exponenciális időigény

# Mélységkorlátozott keresés

```
NODE depth_limited_search(NODE x, int depth, int max_depth) {
  if (is_solution(x)) return x;
  if (depth == max_depth) return null;

  foreach(NODE child in x.children())
    if (depth_limited_search(child,depth+1,max_depth) != null) return child;

  return null;
}
```

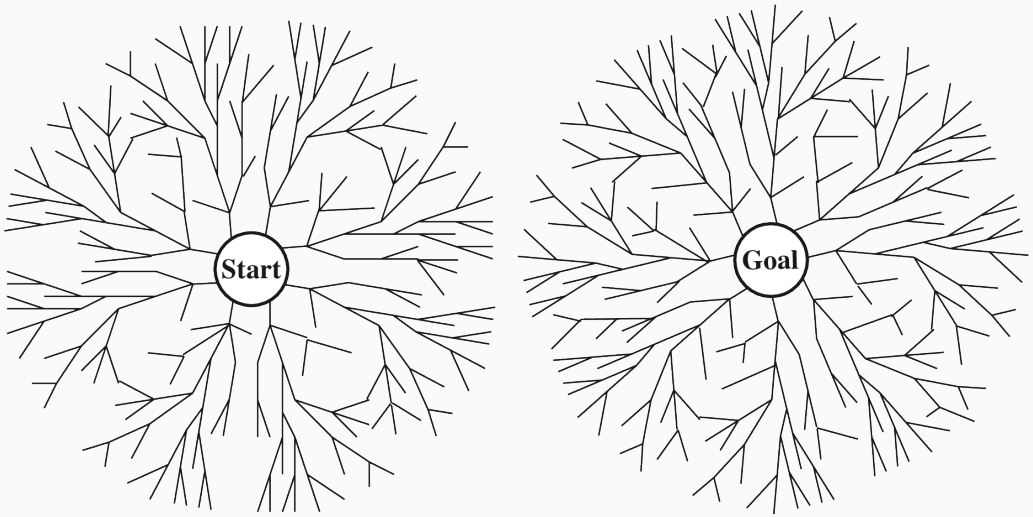
- tulajdonságok: nem teljes, nem optimális, lineáris memóriaigény, exponenciális időigény.

# Iteratívan mélyülő keresés

```
NODE iterative_deepening_search(NODE root) {
  for (int i = 0; i < INFINITY; i++) {
    NODE node = depth_limited_search(root, 0, i);
    if (node != null) return res;
  }
}
```

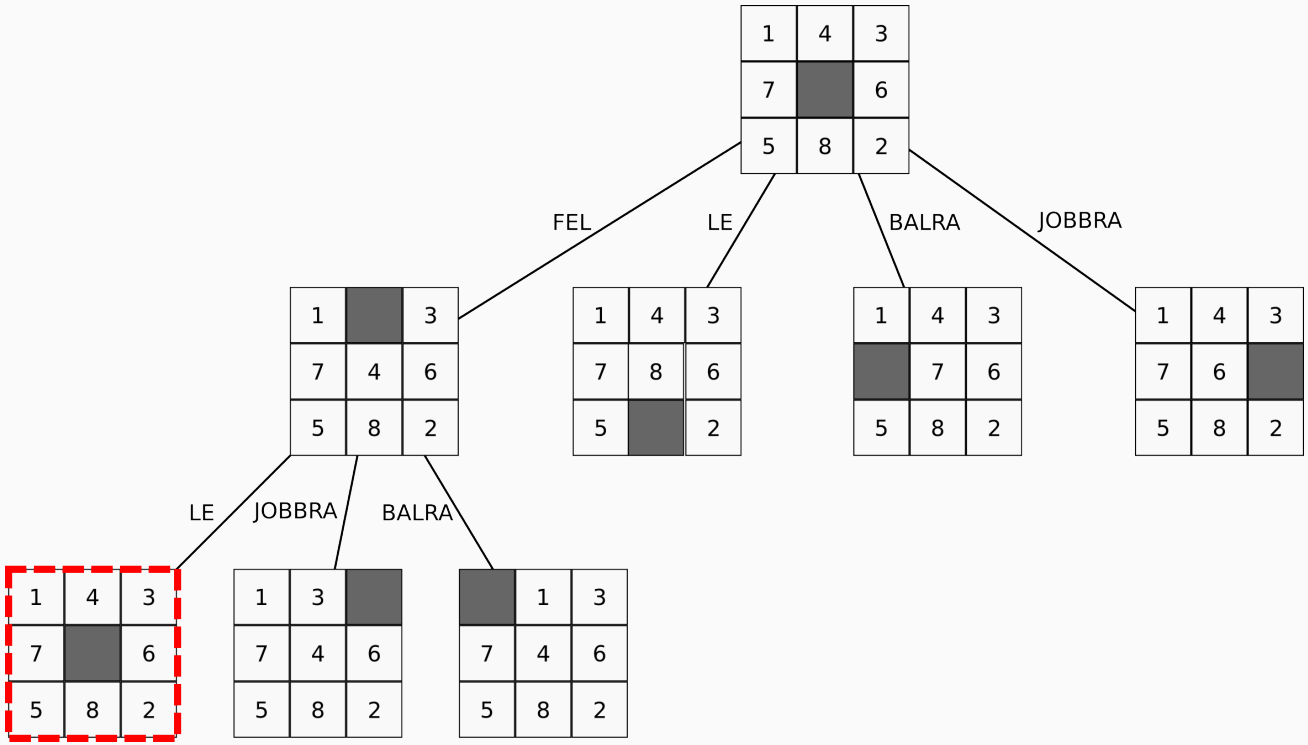
- nem optimális, bizonyos feltételek mellett teljes, lineáris memóriaigény, exponenciális időigény

# Kétirányú keresés




- megfontolás: $b^{\frac{d}{2}} + b^{\frac{d}{2}} \ll b^d$
- végállapot teszt: a két keresés pereme metszi egymást?
- optimalitás?
- **megelőző** csomópontok kiszámítása?

# Ismételt állapotok elkerülése




- kifejtett csomópontokat tartalmazó lista: **zárt lista**
- a kifejtésre váró csomópontokat tartalmazó lista: **nyitott lista**
- ha az aktuális csomópont benne van a zárt listában, akkor eldobható

# Informált keresés

- problémaspecifikus információkat is figyelembe veszünk

# Mohó keresés

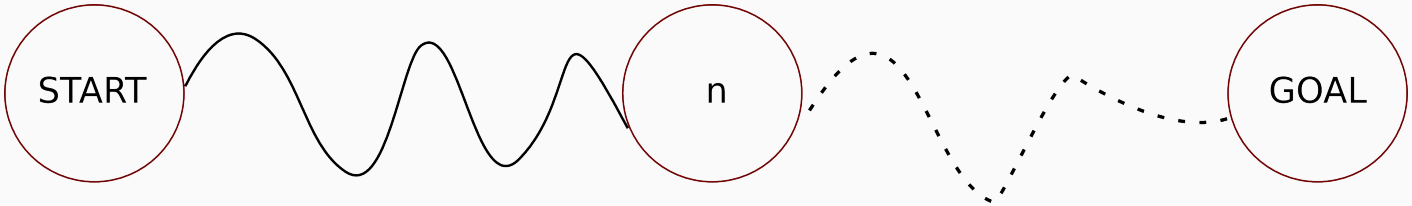
- $h(n)$ **megbecsüli** az $n$ állapotnak a célállapotba vezető út költségét; ha $n$ célállapot, akkor $h(n) = 0$


- adatstruktúra: elsőbbségi sor, kritérium: $\arg\min_n h(n)$

```
NODE greedy_best_first_search(NODE root, HEURISTIC h) {
  QUEUE queue(root, h);
  while(!queue.is_empty()) {
    NODE act = queue.pop_min();
    if (is_solution(act)) return act;
    foreach(NODE child in act.children())
      queue.push_back(child);
  }
  return null;
}
```

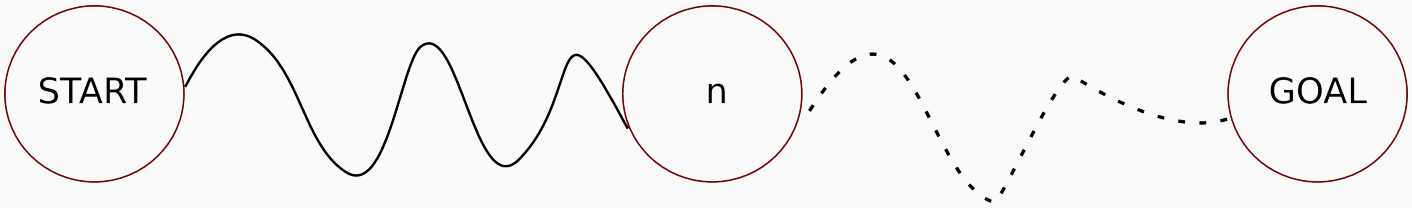
- heurisztika a legrövidebb útra, tologatós játékra?

# A*




- A* algoritmus $\approx$ egyenletes költségű keresés + mohó keresés + ismételt állapotok elkerülése
- $g(n)$: tényleges költség a kiinduló állaptból $n$-ig
- $h(n)$: becsült költség $n$-ből a végállapotig
- adatstruktúra: elsőbbségi sor, kritérium: $\arg\min_n f(n) = g(n) + h(n)$
- demo

# A*, heurisztika




- legyen $O(n)$ az optimális költség $n$-ből a végállapotig

- ha $h(n) = 0 \rightarrow$ A* visszaalakul egyenletes költségű kereséssé (Dijkstra)
- ha $h(n) \leq O(n)$ ($h$ elfogadható heurisztika) $\rightarrow$ A* **optimális**, garantáltan megtalálja a legrövidebb utat
- ha $h(n) = O(n) \rightarrow$ A* csak a legrövidebb úton levő csomópontokat fogja kibontani
- ha $h(n) > O(n) \rightarrow$ A* nem biztos, hogy a legrövidebb utat találja meg
- ha $h(n) \gg g(n) \rightarrow$ A* visszaalakul mohó kereséssé

# A*, pszeudokód

```
initialize containers OPEN and CLOSED
create nodes GOAL and START
add node START to OPEN
while OPEN is not empty {
  get node N off the OPEN container with the lowest f(N) (*)
  add N to CLOSED
  if state(N) == state(GOAL) return solution(N)
  for each successor node N' of N {
    parent(N') = N
    g(N') = cost(N, N')
    h(N') = heuristic_cost(N', GOAL)
    f(N') = g(N') + h(N')
    if (X = find(OPEN, state(N'))) not null {
      if f(X) <= f(N') continue
      else remove X from OPEN
    }
    if (X = find(CLOSED, state(N'))) not null {
      if f(X) <= f(N') continue
      else remove X from CLOSED
    }
    add N' to OPEN
  }
}
return failure
```

# A*, gyakorlati megfontolások

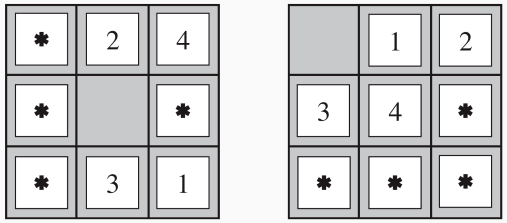
- milyen adatstruktúrát érdemes használni az OPEN, illetve a CLOSED listának?
- (*) mi történjen, ha a két különböző állapot ugyanazzal az f értékkel rendelkezik?
- törekedni kell az állapotok kompakt ábrázolására?

# Heurisztikus függvények

- cél: **optimistán** és minél pontosabban megbecsülni a tényleges költséget
- a heurisztikus függvényt rendszerint a **relaxált** problémából származtatjuk
- kirakós játékhoz, ha $s$ egy állapot
  - $h_1(s)$ - a rossz helyen levő csempék száma
  - $h_2(s)$ - a csempéknek a saját célhelyeiktől mért Manhattan távolságaik összege
  - NB: az üres négyzet távolságát nem kell beleszámolni
- több heurisztikus függvény kombinálása:

$$h(s) = \max\{h_1(s), h_2(s), \ldots, h_n(s)\}$$

# Heurisztikus függvények (2)




- a **részfeladatok** megoldása szintén képezheti heurisztika alapját; pl az 1-2-3-4-es csempe optimális elmozgatása egy részfeladata a tologatós problémának
- $h_{1234}$: minta-adatbázis, ami tartalmazza az 1-2-3-4-es csempék **összes** lehetséges kiinduló pozíciójára a végállapotba mozgatás költségét
- elfogaható heurisztika $h_{1234}$?
- hogy lehet egy kombinált heurisztikát készíteni $h_{1234}$ és $h_{5678}$-ból?

# Összefoglaló

- vak keresés: szélességi, egyenletes költségű, mélységi, mélységkorlátozott, iteratívan mélyülő, kétirányú
- informált keresés: mohó, A*, IDA*

# Könyvészet, ábrák

- Russel & Norvig: Artificial Intelligence, a Modern Approach (2. és 3. kiadás)
- Reinfeld: Complete solution of the eight puzzle and the benefit of node ordering in IDA*