# 3. fejezet

# Informálatlan keresés

## Állapottér

Tekintsünk egy *diszkrét*, *statikus*, *determinisztikus* és *teljesen megfigyelhető* feladatkörnyezetet. Részletesebben: tegyük fel, hogy a világ tökélesen modellezhető a következőkkel:

- lehetséges állapotok halmaza
- egy kezdőállapot
- lehetséges cselekvések halmaza, és egy állapotátmenet függvény, amely minden állapothoz hozzárendel egy (cselekvés,állapot) típusú, rendezett párokból álló halmazt
- állapotátmenet költségfüggvénye, amely minden lehetséges állapot-cselekvés-állapot hármashoz egy $c(x, a, y)$ valós költségértéket rendel
- célállapotok halmaza (lehetséges állapotok részhalmaza)

A fenti modell egy súlyozott gráfot definiál, ahol a csúcsok az állapotok, az élek cselekvések, a súlyok pedig a költségek. Ez a gráf az *állapottér*.

A továbbiakban feltesszük hogy az állapotok számossága véges vagy megszámlálható. Egy állapotnak legfeljebb véges számú szomszédja lehet.

Úton állapotok cselekvésekkel összekötött sorozatát értjük (pl. $x_1, a_1, x_2, a_2, \ldots, x_n$, melynek költsége $\sum_{i=1}^{n-1} c(x_i, a_i, x_{i+1})$).

Ebben a feladatkörnyezetben ha az ágens ismeri a világ modelljét, akkor nem is kell neki szenzor!

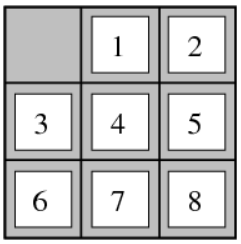
## Példák

*Térkép:* állapotok=városok; cselekvés=úttal összekötött két város közti áthaladás; költség=távolság; a kezdő és célállapot feladatfüggő. A probléma itt útvonaltervezés, a térkép pedig a világ modellje.

*Utazástervezési feladat:* útvonaltervezéshez hasonló. pl. állapotok=hely és időpont párok; cselekvés=közlekedési eszközök, amelyek onnan és azután indulnak mint az aktuális állapot; költség=idő és pénz függvénye; a kezdő és célállapot feladatfüggő. Ez már egy fokkal bonyolultabb.

*Porszívó világ:* illusztrációnak (nem realisztikus): a világ két pozíció, mindkettő lehet tiszta vagy poros, ill. pontosan az egyikben van a porszívó. Ez 8 állapot. cselekvés=szív, jobbra, balra; költség=konstans minden cselekvésre; a célállapot a tisztaság.

*8-kirakó (csúsztatós játék):* állapotok=célállapotból (ábra) csúsztatásokkal elérhető konfigurációk; cselekvés=üres hely mozgatása fel, le, jobbra, balra; költség=konstans minden cselekvésre; a célállapotot az ábra mutatja.




## Kereső algoritmusok: fakeresés

Adott kezdőállapotból találjunk egy minimális költségű utat egy célállapotba. Nem egészen a klasszikus legrövidebb út keresési probléma: az állapottér nem mindig adott explicit módon, és végtelen is lehet.

Ötlet: *keresőfa*, azaz a kezdőállapotból növesszünk egy fát a szomszédos állapotok hozzávételével, amíg célállapotot nem találunk. Ha ügyesek vagyunk, optimális is lesz.

Vigyázat: a keresőfa *nem* azonos a feladat állapotterével! Pl. az állapottér nem is biztosan fa, amely esetben a keresőfa nőhet végtelenre is, akkor is, ha az állapottér véges.

A keresőfa csúcsaiban a következő mezőket tároljuk: szülő, állapot, cselekvés (ami a szülőből ide vezetett), útköltség (eddigi költség a kezdőállapotból), mélység (a kezdőállapoté nulla).

```
fakeresés
1   perem <- {új-csúcs(kezdőállapot)}
2   if perem.üres() return failure
3   csúcs <- perem.elsőkivesz()
4   if csúcs.célállapot() return csúcs
5   else perem.beszúr(csúcs.kiterjeszt())
6   goto 2
```

A `csúcs.kiterjeszt()` metódus létrehozza a csúcsból elérhető összes állapothoz tartozó keresőfa csúcsot, a mezőket megfelelően inicializálva.

A `perem` egy prioritási sor, ez definiálja a bejárási stratégiát! Közelebbről a `perem.elsőkivesz` által feltételezett rendezést a csúcsok felett definiálhatjuk sokféleképpen, és ez adja meg a stratégiát (ld. később).

## Algoritmusok vizsgálata: teljesség, optimalitás, komplexitás

Egy adott konkrét keresési stratégia (perem prioritási sor implementáció) elemzésekor a következő tulajdonságokat fogjuk vizsgálni:

Egy algoritmus *teljes* akkor és csak akkor, ha minden esetben, amikor létezik véges számú állapot érintésével elérhető célállapot, az algoritmus meg is talál egyet.

Egy algoritmus *optimális* akkor és csak akkor, ha teljes, és minden megtalált célállapot optimális költségű.

Az idő- és memóriaigényt *nem* az állapottér méretének függvényében vizsgáljuk, hanem a speciális alkalmazásunkra (MI) szabva a következő paraméterek függvényében: $b$: szomszédok maximális száma, $m$: keresőfa maximális mélysége, $d$: a legkisebb mélységű célállapot mélysége a keresőfában. $m$ és $d$ lehet megszámlálhatóan végtelen!

## Szélességi keresés

FIFO (first in first out) perem. Bizonyítsuk be, hogy

- Teljes, minden véges számú állapot érintésével elérhető állapotot véges időben elér.
- Általában nem optimális, de akkor pl. igen, ha a költség a mélység nem csökkenő függvénye.
- időigény = tárigény = $O(b^{d+1})$

A komplexitás exponenciális, tehát nem várható, hogy skálázódik: nagyon kis mélységek jönnek szóba, $d = 10$ körül. A memória egyébként előbb fogy el.

## Mélységi keresés

LIFO (last in first out) perem. Bizonyítsuk be, hogy

- Teljes, ha a keresési fa véges mélységű (azaz véges, hiszen $b$ véges). Egyébként nem.
- Nem optimális.
- időigény: a legrosszabb eset $O(b^m)$ (nagyon rossz, sőt, lehet végtelen), tárigény: legrosszabb esetben $O(bm)$ (ez viszont biztató, mert legalább nem exponenciális).

## Iteratívan mélyülő keresés

Mélységi keresések sorozata 1, 2, 3, stb., mélységre korlátozva, amíg célállapotot találunk. Bizonyítsuk be, hogy

- Teljesség és optimalitás a szélességi kereséssel egyezik meg.
- időigény = $O(b^d)$ (jobb, mint a szélességi, bár kis $b$ esetén ténylegesen nem feltétlenül jobb), tárigény = $O(bd)$ (jobb, mint a mélységi!).

Elsőre meglepő, de igaz, hogy annak ellenére, hogy az első szinteket újra és újra bejárjuk, mégis javítunk.

Ez a legjobb informálatlan (vak) kereső.

## Egyenletes költségű keresés

A peremben a rendezés költség alapú: először a legkisebb útköltségű csúcsot terjesztjük ki. Bizonyítsuk be, hogy

- Teljes és optimális, ha minden él költsége $\geq \epsilon > 0$.
- (Az idő- és tárigény nagyban függ a költségfüggvénytől, nem tárgyaljuk.)

## Ha nem fa az állapottér: gráfkeresés

Ha a kezdőállapotból több út is vezet egy állapotba, akkor a fakeresés végtelen ciklusba eshet, de legalább is a hatékonysága drasztikusan csökkenhet. Másrészt világos, hogy elég is a legjobb utat tárolni minden állapothoz.

Hogyan kerüljük el az ugyanazon állapotba vezető redundáns utak tárolását? *Zárt halmaz:* tárolni kell nem csak a peremet, de a peremből már egyszer kivett, kiterjesztett csúcsokat is. (A perem egy másik elnevezése *nyílt halmaz*)

A perembe helyezés előtt minden csúcsot letesztelünk, hogy a zárt halmazban van-e. Ha igen, nem tesszük a perembe. Másrészt minden peremből kivett csúcsot a zárt halmazba teszünk. Így minden állapothoz a legelső megtalált út lesz tárolva.

```
gráfkeresés
1   perem <- {új-csúcs(kezdőállapot)}
2   zárt <- {}
3   if perem.üres() return failure
4   csúcs <- perem.elsőkivesz()
5   if csúcs.célállapot() return csúcs
6   else perem.beszúr(csúcs.kiterjeszt() - zárt)
7   zárt.hozzáad(csúcs)
8   goto 3
```

*Probléma:* Mi van, ha egy adott állapothoz a később megtalált út a jobb?

- Egyenletes költségű keresésnél bizonyíthatóan az első megtalált útnál nincs jobb, ha minden él költsége nemnegatív. Ez ugyanis éppen a Dijkstra algoritmus az állapottérre alkalmazva. (Viszont a teljességhez továbbra is kell, hogy minden költség $\geq \epsilon > 0$ (nem csak nemnegatív), mert lehetnek végtelen állapotterek.)
- Mélységi keresésnél nem biztos, hogy az első megtalált út a legjobb, ekkor át kell linkelni a zárt halmazban tárolt csúcsot a jobb út felé. De a mélységi keresés itt már nem annyira vonzó, mert a zárt halmaz miatt sok memória kellhet neki.