Eötvös Loránd Tudományegyetem

Természettudományi Kar

Csető Mária

Gráfkereső algoritmusok a mesterséges intelligenciában

BSc Szakdolgozat

Témavezető:

Dr. Fekete István

Számítógéptudományi Tanszék

Budapest, 2019

Tartalomjegyzék

1. Bevezetés **1**

2. Állapottér- és gráfreprezentáció **4**

2.1. Állapottér-reprezentáció 5

2.2. Reprezentációs gráf 6

2.3. Feladatok reprezentációi 7

2.3.1. Négy királynő probléma 7

2.3.2. Nyolcas játék 9

2.3.3. Hanoi tornyai 13

2.3.4. A kancsók problémája 14

2.3.5. A misszionárius-kannibál probléma 16

3. Visszalépéses keresés **19**

3.1. A visszalépéses keresés első változata 20

3.2. A visszalépéses keresés második változata 21

3.3. A visszalépéses keresés harmadik változata 24

4. Gráfkereső eljárások **28**

4.1. A gráfkeresés kiinduló algoritmusa 29

4.2. Általános gráfkereső algoritmus 30

4.3. Neminformált gráfkereső eljárások 34

4.3.1. Mélységi keresés 35

4.3.2. Szélességi keresés 36

4.3.3. Egyenletes keresés 36

5. Heurisztikus gráfkeresések **40**
5.1. Előretekintő keresés 42
5.2. Az A algoritmus 43
5.3. Az A* algoritmus 44
5.4. A monoton megszorításos algoritmus 46

Ábrák jegyzéke **50**

Felhasznált irodalom **51**

Köszönetnyilvánítás

Ezúton szeretnék köszönetet mondani témavezetőmnek, Dr. Fekete Istvánnak, aki hasznos tanácsaival hatalmas segítséget nyújtott szakdolgozatom elkészüléséhez. Köszönöm szüleimnek és húgaimnak, hogy támogattak egyetemi tanulmányaim alatt.

1. fejezet

Bevezetés

A *mesterséges intelligencia* (továbbiakban: az MI) olyan feladatok számítógépes megoldást tűzi ki célul, amelyeket – ha ember oldja meg őket – intelligenciát igényelnek. A problémamegoldó intelligencia fő jellemzője a feladatspecifikus heurisztikus ismeretek megjelenése és alkalmazása.

Az MI napjaink talán legdinamikusabban fejlődő tudományága. Elég azokra a szolgáltatásokra gondolni, amelyeket napi szinten használunk (pl. számítógépes játékok, sakkprogramok, az autós útvonaltervező GPS vagy fordítás egyik nyelvről a másikra), illetve a naponta hallott divatos irányzatok közül idézni néhányat (pl. intelligens otthon, önvezető autó).

Az újszerű eredmények mellett az MI elméleti alapjai, amint azt több bevezető könyv is bizonyítja, egyezményes módon kialakultak. Az alapok között meghatározó a keresési módszerek rendszerezett gyűjteménye.

Dolgozatomban a *gráfkereső eljárások* elméletének alapjait tekintem át. Ennek a témakörnek központi a jelentősége, eredményeit széles körben alkalmazzák, mivel az MI legtöbb programja tartalmaz keresést. Ez nem meglepő, hiszen az MI legtöbb feladatának a megoldásában általában nem egy direkt algoritmus, hanem nagyszámú szisztematikus próbálkozást végző eljárások vezetnek célhoz.

A dolgozat következő, *második fejezete* az állapottér- és a gráfreprezentációról szól. Ez az MI feladatainak gyakran alkalmazott specifikációs módszere. A problématér állapotai, az azok közötti átmeneteket megvalósító műveletek, valamint a kezdőállapot és a célállapotok halmazának megadása együtt alkotják az állapottér-reprezentációt.

Ezek az elemek együtt szemléletesen megjeleníthetők egy irányított gráf formájában. A probléma megoldását egy olyan élsorozat jelenti, amely a reprezentációs gráf startcsúcsából valamely célcsúcsába vezet. A műveletekhez és így az azokat rep-

rezentáló élekhez a gráfban költségeket is rendelhetünk. A fejezetben megadom öt tipikus MI-feladat reprezentációját, amelyekre a későbbi fejezetekben hivatkozom.

A *harmadik fejezet* az egyik leggyakrabban alkalmazható problémamegoldó módszert, a visszalépéses keresést mutatja be. Ez a jól ismert vezérlési stratégia egyetlen útvonalat tart nyilván, amelyet a lehetőség szerint egyre mélyít. Amennyiben a keresés akadályba ütközik, akkor az algoritmus visszalép az előző állapotba és új irányt keres a folytatáshoz. A visszalépéses keresés három változatát ismertetem, amelyek a reprezentációs gráf jellegzetességéhez, illetve a feladat követelményéhez illeszkednek (keresett célállapotok száma, a megoldó műveletsor meghatározása, állapotok ismétlésének ellenőrzése és kizárása, mélységi korlát alkalmazása).

A *negyedik fejezet* témája az gráfkeresés és annak speciális neminformált változatai. A gráfkeresés általános algoritmusa a reprezentációs gráfot a kezdőállapotból kezdi el építeni. A gráf minden csúcsánál nyilvántartjuk azt a legkisebb költséget, amellyel – a keresés aktuális állapotában – a kezdőcsúcsból hozzá eljuthatunk. Az általános gráfkereső algoritmus minden lépésben kiválaszt egy kiterjesztésre váró nyílt csúcsot egy függvény minimális értéke szerint. A kiválasztott csúcs minden szomszédjára ellenőrzi, hogy sikerült-e kisebb költségű utat találni hozzá. Ha igen, akkor a csúcshoz bejegyzésre kerül az új költségérték és feljegyezzük a szülő csúcsot is.

A neminformált gráfkereső eljárások úgy származtathatók az általános sémából, hogy a kiértékelő függvény definíciójában csak a műveletek, vagyis a reprezentációs gráf éleinek költségeit használjuk fel valamilyen formában. A költségek speciális esete az, amikor minden él költségét egységnyinek tekintjük. Egy csúcs elérésének költsége ekkor a mélységét fejezi ki a reprezentációs gráfban.

A szélességi keresésnél mindig egy legkisebb mélységű csúcsot választunk kiterjesztésre. A mélységi keresésnél minden lépésben egy legmélyebben fekvő csúcsból próbálunk tovább haladni. (A visszalépéses keresést nyilvánvalóan a mélységi kereséssel lehet közvetlen kapcsolatba hozni.) Az egyenletes keresésben (amely a szélességi keresés általános változatának is tekinthető) az élek valós nemnegatív költséggel rendelkeznek és az eljárás minden lépésben egy legkisebb költséggel elérhető csúcs kiterjesztésével halad tovább a megoldás felé.

Az *ötödik fejezet* a heurisztikus gráfkereső algoritmusokat tárgyalja. A heurisztika általában egy probléma megoldása során alkalmazott speciális, gyakran intuitív ismeret. A gráfkeresés elméletében a heurisztika mindig annak a költségnek a becslését jelenti, amellyel egy állapotból célállapotba juthatunk. A heurisztikus gráfkeresés kiértékelő függvénye ennek megfelelően (ritka kivételtől eltekintve) két

komponens összege: tartalmazza egy csúcs elérésének költségét és egy célcsúcs elérési költségének heurisztikusan becsült értékét.

A heurisztikus gráfkeresés menetét a kiértékelő függvény szabályozza, mivel annak minimum-értéke szerint választunk ki egy nyílt csúcsot kiterjesztésre. A heurisztikus tagra alkalmazott feltevések a keresések érdekes tulajdonságú osztályaihoz vezetnek, ezek az A, az $A*$, valamint a monoton megszorításos gráfkeresés. A három osztály között valódi tartalmazás áll fenn és egyre erősebb tulajdonságokkal rendelkeznek a megoldás megtalálása, annak optimalitása, illetve a csúcsok kiterjesztésének módja tekintetében. A tulajdonságokat megfogalmazó tételek bizonyításai dolgozatomban szerepelnek.

Dolgozatom megírásakor alapvetően témavezetőm és szerzőtársai által írott magyar nyelvű MI-könyvét vettem alapul, de tájékozódtam az MI szakirodalmának bevezető műveiben is. Dolgozatomban azért nem szerepelnek irodalmi hivatkozások, mert több hasonló tartalmú leírásban csak kis eltérések jelentek meg és az olvasottak végül egységes egésszé álltak össze.

A legtöbb forrás ugyanazt a néhány alapfeladatot ismerteti, ezeket kiegészítettem két további feladattal, a kancsók öntögetésének problémájával és a misszionárius-kannibál problémával. Önálló munkámnak tekinthető a visszalépéses keresés első változata is (amely nem szerepel a könyvekben), ez az összes célállapotot megkeresi egy reprezentációs gráfban.

Megemlítem azt is, hogy néhány tervezett tartalom megírására nem került sor. Ezek között szerepelt az „algoritmusok és adatstruktúrák” tantárgy, illetve tudományág által ismertetett gráfalgoritmusok és az MI megfelelő algoritmusainak az összehasonlító megfeleltetése (például szélességi és mélységi gráfbejárás, vagy Dijkstra algoritmusa). Szerettem volna egy kitekintést összeállítani, amely röviden kitért volna azt itt nem ismertetett kereső eljárásokra (pl kétirányú keresés, változó mélységű keresés vagy a szimulált hűtés).

Dolgozatomban összesen 22 ábra szerepel, ezek többségét magam rajzoltam meg, néhány összetettebb ábrát azonban – témavezetőm hozzájárulásával – a könyvükből vettem át.

2. fejezet

Állapottér- és gráfreprezentáció

A mesterséges intelligencia feladatainak megoldását is a specifikációval kezdjük. A „specifikáció” kifejezés helyett itt a „reprezentáció” szóhasználat terjedt el. A tapasztalatok szerin az MI feladatait az állapottér-reprezentáció szemléletével a legalkalmasabb specifikálni.

Először meghatározzuk a feladat állapotterét, amely az összes lényeges adatszerkezetet magában foglalja. A reprezentáció algoritmikus jellegű részét képezik az állapotokra alkalmazható műveletek. Ezek a feltételhez kötött, gyakran költséggel is rendelkező, állapotokon értelmezett eljárások újabb állapotokhoz vezetnek. A reprezentáció további komponense a feladat célállapotainak a halmaza, valamint a kezdőállapot, amely a megoldás keresésének kiindulását képezi.

Az állapottér-reprezentációt szemléletessé tehetjük egy irányított gráf alkalmazásával. A gráf csúcsai az egyes állapotoknak, az irányított élek az alkalmazható műveleteknek felelnek meg. A reprezentáció kezdőállapota a gráf kijelölt startcsúcsát határozza meg. A célállapotok a gráf csúcsainak elérendő részhalmazát képezik.

A gráffal ábrázolt feladatleírást gráfreprezentációnak nevezzük, amely mögött mindig egy gráfkereső eljárást tételezünk fel. A keresés algoritmusának a megválasztása vagy megtervezése – tágabb értelemben – a feladat megoldásának a meghatározó részét képezi.

Az MI-ben alkalmazott reprezentációs gráf jellemzője az, hogy a keresés megvalósításakor nem áll rendelkezésre explicit módon felépített módon a memóriában, hanem azt a kereső eljárás építi fel, általában csak részben. További jellemzője a gráfkeresésnek az, hogy a kereső eljárás működhet oly módon, hogy a csúcsok többszörözésével a gráfot speciálisan irányított, gyökeres fa formájában hozza létre, amely lehet elvben végtelen is.

Az állapottér- és gráfreprezentáció alkalmazását több példán is bemutatom ebben a fejezetben.

2.1. Állapottér-reprezentáció

A bevezető szerint a mesterséges intelligencia feladatainak specifikálására az állapottér-reprezentáció a legalkalmasabb és legelterjedtebb módszer. Az egyszerűbb, áttekinthető méretű feladatokra ez mindenképpen igaz. A feladatleírás ezen módszere mögött mindig egy kereső eljárást feltételezünk, annak kialakítását segíti elő a gondosan megfogalmazott specifikáció.

Az állapottér-reprezentáció egy $R = (A, M, T, s)$ négyes, amelynek komponensei a következők:

1. a feladat állapottere (A),
2. az állapottéren értelmezett műveletek (M),
3. a célállapotok halmaza (T),
4. a feladat kezdőállapota (s).

Az állapottér a feladat lényeges adatszerkezeteinek a lehetséges értékeit tartalmazza. Ha az állapotteret több egyszerűbb típus alkotja, akkor ezek típusértékhalmazai Descartes-szorzat formájában kapcsolódnak egybe. Az egyes értékek pedig n-eseket alkotnak ebben az esetben. Az állapottér leírásánál törekedni kell az absztrakció szintjének megfelelő megválasztására: az adatszerkezet elég kifejező legyen és ugyanakkor már mutasson az implementáció felé.

A műveleteket gyakran szabályoknak vagy operátoroknak nevezik az MI-ben. Ezek olyan transzformációk, amelyek az állapottéren vannak értelmezve. Az értelmezési tartományt az egyes műveletek végrehajthatósági feltételei jelölik ki. A műveletek algoritmusainak végrehajtása az egyik állapotból egy másik állapotba vezet. A műveletekhez általában költségeket is rendelünk. Ha ezt nem tesszük, akkor a műveleteket egységnyi költségűeknek szokás tekinteni.

A célállapotok az állapottérnek azok az elemei, amelyek közül egyiknek az elérésére törekszünk. Egy feladatnak több célállapota is lehet. Bizonyos esetekben a célállapotok explicit módon adottak, máskor viszont nem ismertek, hanem csak feltételekkel tudjuk azokat leírni.

A kezdőállapot az a konkrét és egyértelmű eleme az állapottérnek, amelyből kiindulva az aktuális esetben a megoldást keressük, azaz egy célállapothoz el szeretnénk jutni. (A feladat specifikálásának részét képezheti egy olyan elemzés is, amely meghatározza a lehetséges kezdőállapotok körét.)

A feladat megoldásának egy olyan műveletsorozatot tekintünk, amely a megadott kezdőállapotból elvezet egy célállapotba. A lehetséges megoldások közül gyakran egy

olyan műveletsorozat megkeresése a cél, amelynek minimális a költsége. Egy ilyet optimális megoldásnak nevezzünk. Olyan esetben, amikor a költségek egységnyiek, szokás legrövidebb megoldásról is beszélni.

Amint a megadott példákban látjuk majd, van olyan feladat, ahol csak a célállapotok előállítása fontos, az oda vezető műveletsorozatok mintegy a megoldás „melléktermékének" tekinthetők csupán. Máskor – éppen ellenkezőleg – a célállapotot már ismerjük és éppen az azt előállító műveletsorozatot keressük. Ezeken kívül, a célállapotok és a műveletsorozatok keresésére további egyedi előírások is előfordulhatnak.

2.2. Reprezentációs gráf

Az állapottér reprezentációt átláthatóbbá tehetjük egy irányított gráf lerajzolásával. A gráfreprezentáció segíthet a feladat jobb megértésében és a megfelelő kereső algoritmus kiválasztásában is. A gráfos specifikálással ugyanis mindig együtt jár egy gráfkereső algoritmussal, amely egy megadott kezdőállapotból indulva egy célcsúcs elérésére törekszik.

A reprezentációs gráf csúcsai a feladat egyes állapotainak felelnek meg, megjelölt startcsúcsa a keresés kiinduló állapota, a célállapotok pedig az állapotoknak az a részhalmaza, amelynek elérésére a keresés irányul. A gráf élei, amelyek a műveletek alkalmazásának felelnek meg, rendelkezhetnek költséggel is.

A feladat specifikálása során a reprezentációs gráfot általában felrajzoljuk (nagyobb méret esetén egy részét), de tudjuk, hogy a keresés számítógépes programja nem kapja készen a gráfot, hanem maga építi fel, legalább is a bejárt részét. Ezt hívják keresőgráfnak.

A feladat megoldását általában egy célcsúcs megtalálása jelenti. Ez egy olyan útvonal megkeresésével valósul meg, amely a reprezentációs gráf startcsúcsából valamely célcsúcsba vezet. Ha szükség van a célállapotba vezető műveletsorozatra is, akkor azok a feltárt útvonal éleiről leolvashatók.

Bizonyos feladatoknál az optimális, vagy minél kisebb költségű útvonal meghatározása a kitűzött cél.

Még azt is érdemes szem előtt tartani, hogy a reprezentációban szereplő gráfot a kereső eljárás – az állapotok többszörözésével – speciálisan fa formájában implementálja. Ekkor egy köröket tartalmazó reprezentációs gráfból (elvben) végtelen mélységű fa alakulhat ki.

2.3. Feladatok reprezentációi

Ebben az alfejezetben néhány (szám szerint öt) egyszerű feladaton mutatom be, hogy hogyan lehet megadni az állapottér-reprezentációt, illetve a gráfreprezentációt, mint az egyes problémák MI-ben szokásos specifikációit.

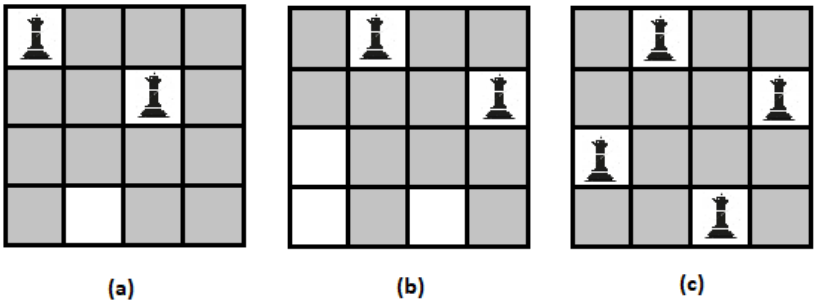
2.3.1. Négy királynő probléma

Első feladatunk a nyolc királynő probléma egyszerűbb, könnyebben áttekinthető változata, négy királynőre megfogalmazva. (A sakkozók hivatalosan vezért mondanak, de ennél a feladatnál ne változtassunk a kialakult szóhasználaton.)

1. Feladat Egy 4×4-es sakktáblán helyezzünk el négy királynőt úgy, hogy egyik se üsse a másikat!

A feladatot gyakran úgy tűzik ki, hogy a megoldásban az összes olyan állást adjuk meg, amely nem tartalmaz ütést. Az egyes célállapotokhoz vezető műveletsorozat nem kérdés, minthogy bármely állásról leolvashatók az azt kialakító műveletek.

Az állapottér-reprezentációhoz a probléma emberi kézzel történő megoldását vesszük alapul. A kezdőállapot esetünkben egyértelmű: az üres tábla. Az állapotteret azok az ütésmentes állások alkotják, melyekben a táblán $0, 1, 2, 3$ vagy 4 királynő áll, sorfolytonos elrendezésben, hiszen az ember sorról-sorra haladva próbálkozik a királynők elhelyezésével. A 2.1. ábrán három legális állapot látható, melyek egyike sem tartalmaz ütést.



2.1. ábra. A négy királynő probléma állapotterének három állása

Az 2.1.a és a 2.1.b ábrán egyaránt két királynő áll a táblán. Az első állásban nem tudunk egy újabb királynőt elhelyezni a tábla harmadik sorában. Bár a negyedik sorban van szabad mező, de előzetesen a királynők sorfolytonos elhelyezése mellett

döntöttünk. A második állásban ez lehetséges, a harmadik sor első mezője nem áll ütésben. A 2.1.c ábra a probléma egy célállapotát mutatja.

A kézzel történő megoldás során azt, hogy egy mező szabad-e, újra és újra szemmel ellenőrizzük. Így egy mezőnek kétféle állapota lehet: királynő áll rajta, vagy üres. A matematikai reprezentációban azonban célszerűbbnek látszik, ha háromféle státuszt különböztetünk meg:

1. királynő (a mezőn királynő áll)
2. szabad (a mező nem áll ütésben)
3. ütésben (a mező ütésben áll valamely a táblán már elhelyezett királynő által)

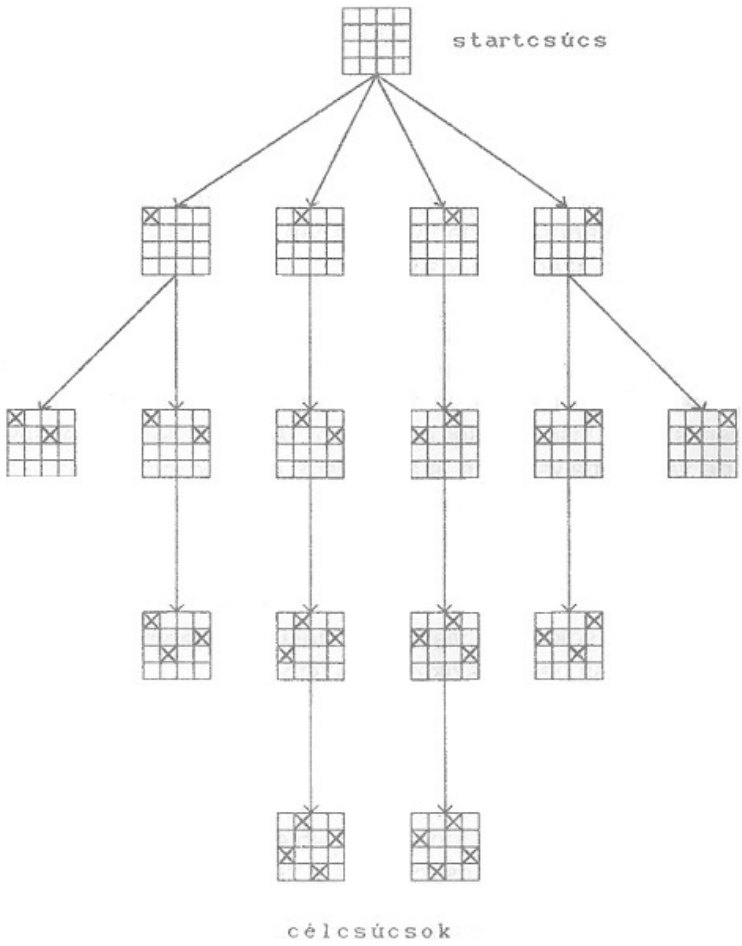
Ha a királynőt olyan mezőre helyezzük el a következő üres sorban, amely „szabad" státuszú, akkor legális álláshoz jutunk. Formálisan, az állapotokat egy $t[1..4, 1..4]$ mátrix megfelelő kitöltésével reprezentáljuk. A státuszokat meghagyhatjuk karakteres formában is, de kódolhatjuk rendre az 1, 0, -1 értékkel.

A kezdőállapot jellemzője az, hogy minden mezője „szabad" státuszú, azaz $t[i, j] = 0$ $(1 \leq i, j \leq 4)$. A céllállapotokra az igaz, hogy minden sorban pontosan egy királynő áll, az elhelyezés feltételének betartásával.

A műveletek halmazát legegyszerűbb úgy definiálni, hogy a királynőnek a tábla bármely mezőjére történő elhelyezését egy önálló műveletnek tekintjük. Jelöljük q_{ij}-vel $(1 \leq i, j \leq 4)$ azt a műveletet, amikor egy szabad mezőre királynőt helyezünk, azaz egy $t[i, j] = 0$ mátrixelem értékét $t[i, j] = 1$ értékre változtatjuk. Az elhelyezés, illetve az értékadás feltétele tehát egyrészt az, hogy ezt megelőzően a mező „szabad" státuszú, vagyis $t = [i, j] = 0$ legyen, másrészt az is, hogy a megelőző sorokban k $(1 \leq k \leq i - 1)$ már legyen egy-egy királynő elhelyezve. A q_{ij} művelethez hozzá tartozik az is, hogy $t[i, j]$-re elhelyezett királynő által „ütésbe"-be került mezők státuszát a -1 értékre állítsuk. Ez a következőket jelenti:

1. $t_{i,k} := -1$ $(1 \leq k \leq 4)$ a teljes i-edik sor lefoglalása
2. $t_{k,j} := -1$ $(1 \leq k \leq 4)$ a teljes j-edik oszlop lefoglalása
3. $t_{k,l} := -1$ $(1 \leq k, l \leq 4$ és $k - l = i - j)$ a $t[i, j]$ elemen átmenő teljes főátló lefoglalása
4. $t_{k,l} := -1$ $(1 \leq k, l \leq 4$ és $k + l = i + j)$ a $t[i, j]$ elemen átmenő mellékátló lefoglalása

A feladat természetéből következik, hogy a reprezentációs gráf ezúttal fa struktúrájú lesz, ahogyan az a 2.2. ábrán látható.



2.2. ábra. A négy királynő probléma reprezentációs gráfja

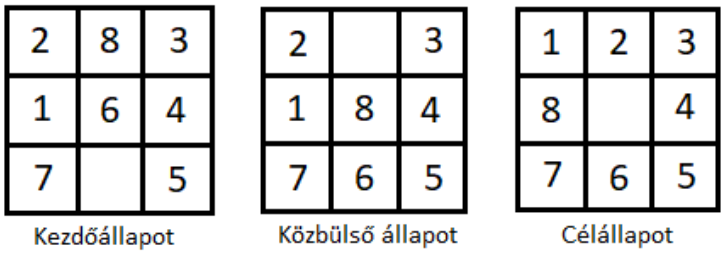
A fa gyökerében található a kezdőállás, leveleiben pedig vagy olyan nem-teljes kitöltések, amelyek nem folytathatók, vagy céllállapotok találhatók. A műveletek élei fentről lefelé mutatnak (a királynőket nem vesszük vissza a tábláról).

2.3.2. Nyolcas játék

Ez a feladat egyszerűsített változata egy létező játéknak, amely egy 4×4-es keretben 15 db számozott, tologatható négyzetet alakú lapocskát tartalmaz.

2. Feladat Adott 3×3-as elrendezésben 8 darab négyzet alakú számozott lapocska, valamint egy üres hely. Bármely lap elmozdítható, ha a szomszédos négyzet üres. A játék célja az, hogy a kezdőállapotból kiindulva az előre megadott célállapotot elérjük.

A gyakorlatban is használatos 4×4-es játék esetén célállapotnak a számok sorfolytonos elhelyezkedését tekintjük (így az üres hely a jobb alsó sarokba kerül). Esetünkben a számok körkörös elrendezése lesz a célállapot (középen az üres hellyel). Az elhelyezés az 1-es sorszámmal indul a bal felső sarokban és az óramutató járásával megegyező irányba halad. A 2.3. ábra egy lehetséges kezdőállapotot mutat, mellette egy közbülső állapot látható (amely két lépéssel érhető el a kezdőállapotból), végül a (három további húzással elérhető) célállapot következik.



2.3. ábra. A 8-as játék három állapota

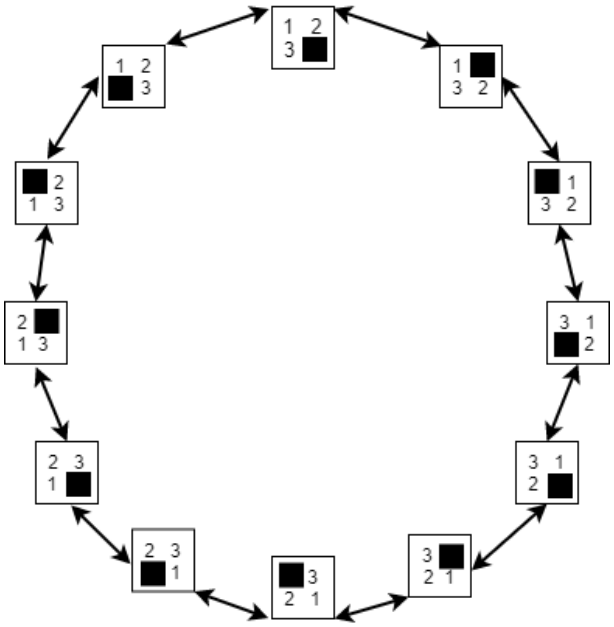
Egy állapotot az $m[1..3, 1..3]$ mátrix kitöltésével kapunk, az $1, 2...8$ számokat helyezzük el úgy, hogy egy mátrix elem „üresen” marad. Rendelhetjük ehhez az üres helyhez a 0 számot és akkor azt mondhatjuk, hogy egy állapot a mátrix kitöltése a $0, 1, 2...8$ számokkal. Így az állapotok száma $9! = 362,880$. A lehetséges kezdőállapotok feléből viszont nem érhető el a célállapot. Kezdőállapotnak bármely állás tekinthető. (A gyakorlatban nyilván nem lenne barátságos, ha olyan kezdőállapotot adnánk meg, amelyből a célállapot nem érhető el.)

A műveleteket úgy célszerű definiálni, hogy nem az egyes számozott lapok húzását jelentik az üres helyre, hanem az üres négyzet mozgatását rendre balra, felfelé, jobbra vagy lefelé. Minden mozgatásnak az a feltétele, hogy a mozgatás irányában ne a keret szélen legyen az üres hely, hanem legyen az adott irányban szomszédja. A formalizálás eredménye tehát az szemlélet, hogy az üres helyet mozgatjuk nem pedig az egyes számokat.

A feladat kitűzése bármely alkalmas kezdőállapotra kérheti annak a műveletsorozatnak a meghatározását, amellyel az ismert és egyértelmű célállapot elérhető. Itt most nem a célállapot megismerése a cél, hanem az odavezető út meghatározása a feladat. A kérdést feltehetjük úgy is, hogy a célhoz vezető legrövidebb utat szeretnénk megtudni.

A reprezentációs gráf lehetne egy meglehetősen terjedelmes gráf is, amelynek a csúcsai között élek vezetnek az alkalmazható műveleteknek (az üres hely lehetséges mozgatásainak) megfelelően.

A gráfreprezentáció műfaját érzékeltei az alábbi 2.4. ábra egy még egyszerűbb feladat esetén. Ebben egy 2×2-es mezőben három kis lapocska helyezkedik el az $1, 2, 3$ számokkal feliratozva, egy hely pedig itt is üresen marad. A célállapot legyen például a legfelül látható állapot. Ekkor ez (mint bármelyik másik állapot is) az összes lehetséges állapot feléből, azaz 12 kiinduló állásból érhető el.

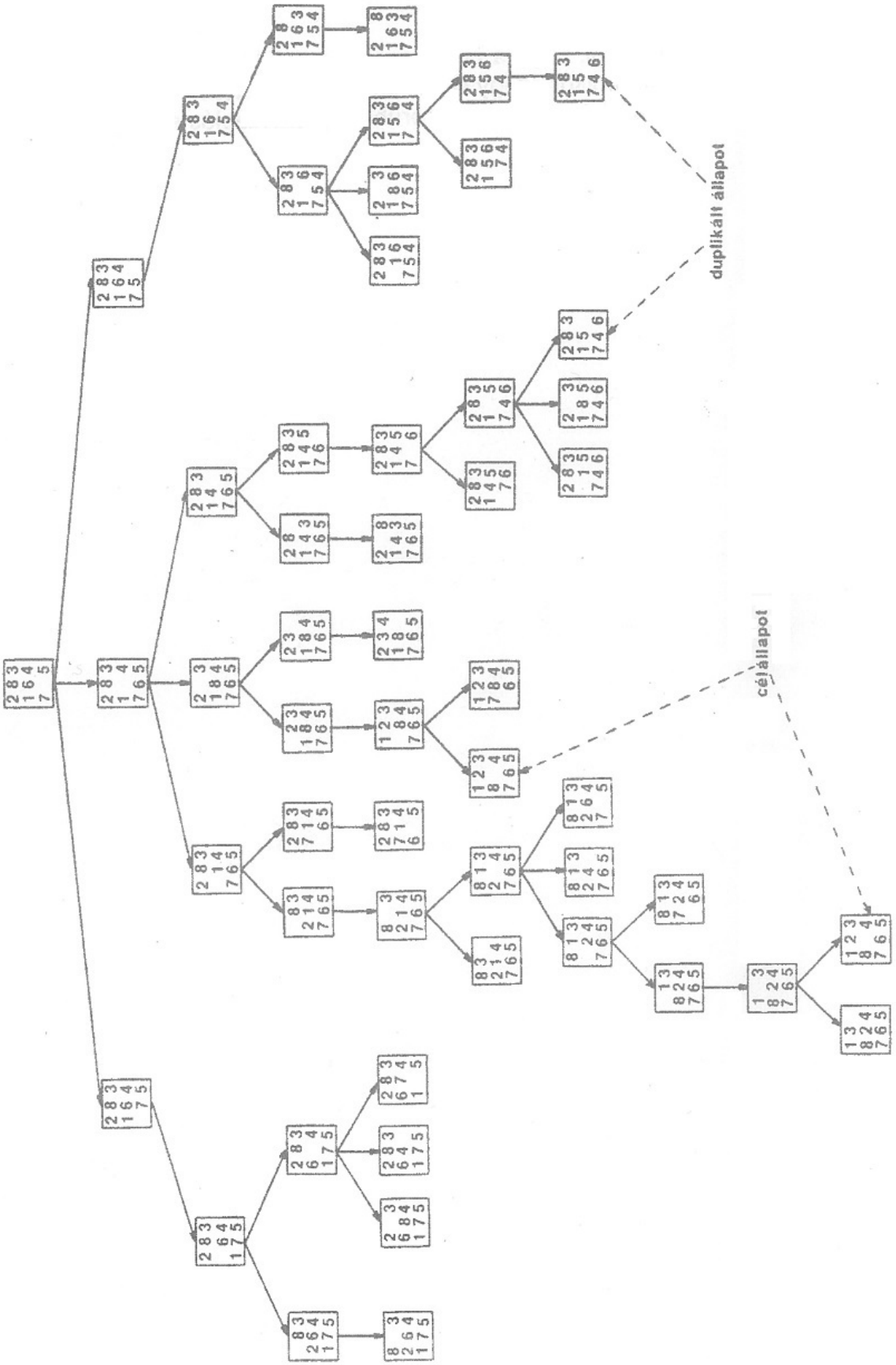


2.4. ábra. A 3-as játék reprezentációs gráfja

A nagyobb táblaméret esetén szövevényesebb gráf egy felrajzolása helyett kifejezőbbnek érezhetünk egy olyan – elvben végtelen méretű – reprezentációs fát, amely a kezdőcsúcsot tartalmazza a gyökerében és az állapotok többszörözésével jön létre.

A 2.5. ábrán láthatjuk a 8-as játék reprezentációs fájának részletét. A fa gyökércsúcsa a 2.3. ábrán megadott kezdőállapotot tartalmazza. Ahogyan az ábra leírásában szerepel, a célállapotba $2 + 3 = 5$ lépéssel juthatunk el. Az alábbi ábra ezt az útvonalat is bemutatja, de egy hosszabb műveletsorozat is látható a fában. Ezen kívül még egy megkettőzött közbülső állapot is felfedezhető az ábrán.

A reprezentációs fa a 8-as játék esetén kifejezőbbnek mondható, mint az eredeti gráf forma lenne. Következő feladatunknál majd éppen arra látunk példát, amikor ezt éppen fordítva gondolhatjuk. A reprezentációs formájáról hozott döntés összefüggésben áll a kereső eljárás megválasztásával is. Nem meglepő, hogy a reprezentációs fához egyszerűbb megoldó eljárások társíthatók, illetve beprogramozásuk fára egyszerűbb, mint általános gráfokra.

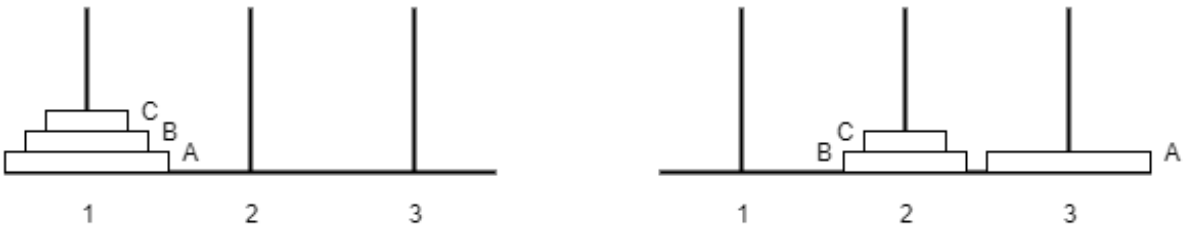


2.5. ábra. A 8-as játék reprezentációs fájának részlete

2.3.3. Hanoi tornyai

A következő feladatot egy legenda szerint eredetileg 64 korongra fogalmazták meg. A feladat közismert, amihez az is hozzájárul, hogy a programozás módszertan is egyik alapfeladatának tekinti (a rekurzió bevezetésekor), illetve az algoritmusokkal foglalkozó tantárgyakban hivatkoznak rá, mint exponenciális lépésszámú eljárásra.

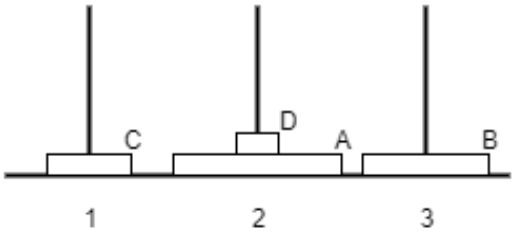
3. Feladat Adott n darab különböző nagyságú korong és három rúd. A játék célja,hogy az első rúdról az összes korong a harmadikra kerüljön úgy, hogy minden lépésben csak egy korongot tehetünk át egy másik rúdra, és csak kisebb korong kerülhet nagyobbra, illetve egy korongot üres rúdra is át szabad helyezni.



2.6. ábra. A Hanoi tornyai probléma két állapota

.

Az állapottér reprezentációt itt először az $n = 3$ esetre gondoljuk meg. Nevezzük a korongokat A, B, C-nek. Egy $v[1..3]$ vektor az egyes korongok helyét tartalmazza. A 2.6. ábrán látható kezdőállapotot $v = [1, 1, 1]$ írja le, míg a mellette elhelyezkedő közbülső állapot esetén $v = [3, 2, 2]$.

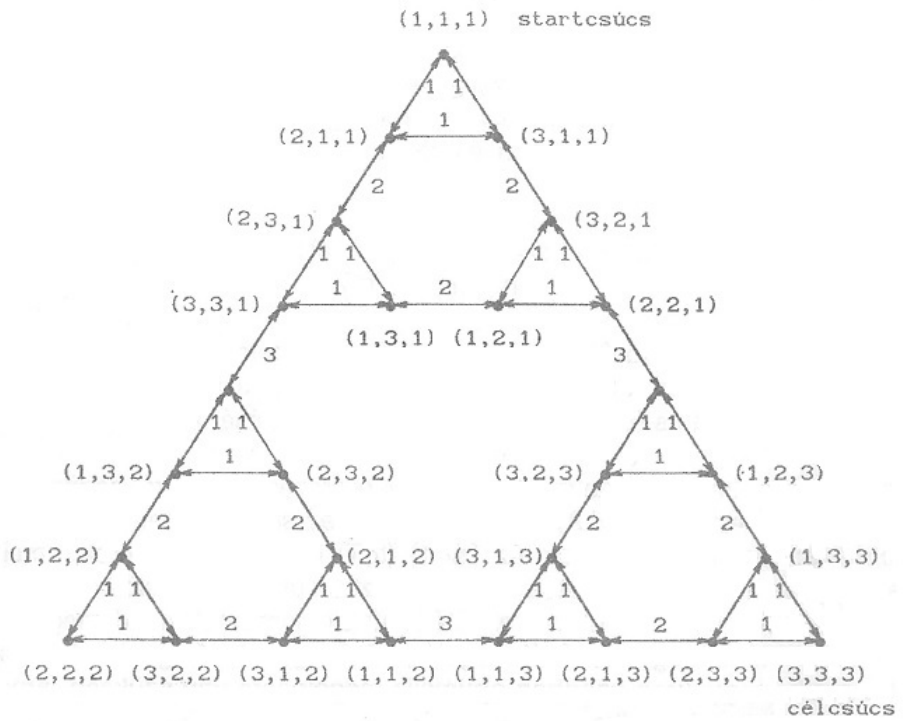


2.7. ábra. A Hanoi tornyai probléma egy állapota (4 korong)

A 2.7-as ábrán látható a feladat egy állapota $n = 4$ esetén, amelynek leírására hasonló vektor használunk: $v = [2, 3, 1, 2]$.

A kezdőállapot mindkét esetben az, amikor az összes korong az első rúdon van: $v = [1, 1, 1]$ és $v = [1, 1, 1, 1]$. A célállapotot pedig az jellemzi, hogy minden korong az utolsó rúdon található: $v = [3, 3, 3]$ és $v = [3, 3, 3, 3]$.

Általában egy lehetséges állapot a $v = [1..n]$ vektor az $1, 2, 3$ számokkal történő kitöltésével állítunk elő, így az állapotok száma 3^n. Egy műveletet m_{ij} jelöl és a felső korong áttételét jelenti az i-edik rúdról a j-edik rúdra. Ennek a lépésnek az a feltétele, hogy az i-edik rúd nem üres és a j-edik rúd üres, vagy a felső korong nagyobb mint az a korong, amelyet készülünk i-ről a j-re áthelyezni.



2.8. ábra. A Hanoi tornya probléma reprezentációs gráfja

A probléma reprezentációs gráfja a 2.8. ábrán látható. Ennél a feladatnál kifejezőbbnek gondolhatjuk a gráf alakot, mint egy abból képzett fát. Programozásmódszertani tanulmányainkból tudjuk, hogy a rekurzív algoritmus optimális megoldást ad, amelyet esetünkben az $(1, 1, 1)$ startcsúcsból a $(3, 3, 3)$ célcsúcsba vezető $2^3 - 1 = 7$ lépéses út reprezentál. Szemléletes azonban, hogy egy kereső eljárás – más elvek szerint haladva az állapotok között – találhat ennél hosszabb megoldó műveletsorozatot is.

2.3.4. A kancsók problémája

A különböző űrtartalmú kancsók öntögetésével kapcsolatos feladatok általában egy nem-teljes egyértelműséggel megadott állapot elérést tűzik ki célul, adott kezdő tartalom mellett. Ezekkel a problémákkal az MI-könyvek fejezetei végén, a feladatok között lehet találkozni. Dolgozatomban azért emeltem ki egy ilyet az alapfeladatok

közé, mert egyrészt a sem a célállapotok, sem az oda vezető műveletsorozatok nem ismertek, másrészt a reprezentációs gráf „igazi” szövevényes gráf-alakot ölt.

4. Feladat Adott három kancsó, amelyek 5, 3 és 2 literesek. Kezdetben az 5 literes kancsó tele van vízzel, a másik kettő üres. A cél az, hogy a 2 literes kancsóba pontosan 1 liter víz kerüljön. A öntések szabálya értelemszerű: az egyik kancsó tartalmát teljes egészében áttöltjük egy másikba, ha abba bele fér, egyébként pedig tele töltjük a másik kancsót.

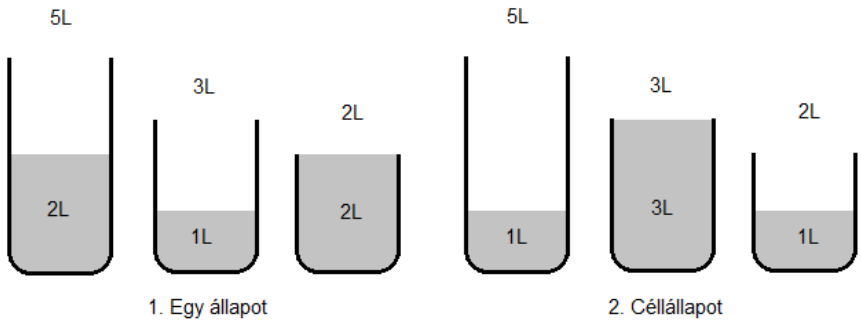
Az állapotok olyan nem-negatív egész számokból álló hármasokkal írhatók le, amelyekben a számok összege adott ($= 5$) és nem változik, továbbá egyik szám sem nagyobb a megfelelő kancsó űrtartalmánál.

A kezdőállapotot az $(5, 0, 0)$ számhármas írja le.

A célállapotok halmazát az $(x, y, 1)$ hármas írja le, azzal a feltétellel kiegészítve, hogy $x + y = 4$ és $y \leq 3$.

Egy tetszőleges (x, y, z) állapot eleget kell, hogy tegyen az $x+y+z = 5$, valamint az $y \leq 3$ és $z \leq 2$ feltételeknek.

A 2.9. ábrán a kitűzött feladat egy előállítható állapota, illetve egyik lehetséges célállapota látható.



2.9. ábra. A kancsók problémájának két állapota

Meg kell jegyezni, hogy nem minden olyan állapot áll elő az öntögetések során, amelyeket a fent megfogalmazott feltételek lehetővé tesznek. Ezt azonnal látni fogjuk, de előbb adjuk meg a műveletek definícióját!

A p_{ij} művelet azt jelenti, hogy az i-edik kancsóból a j-edikbe töltünk. A művelet feltétele:

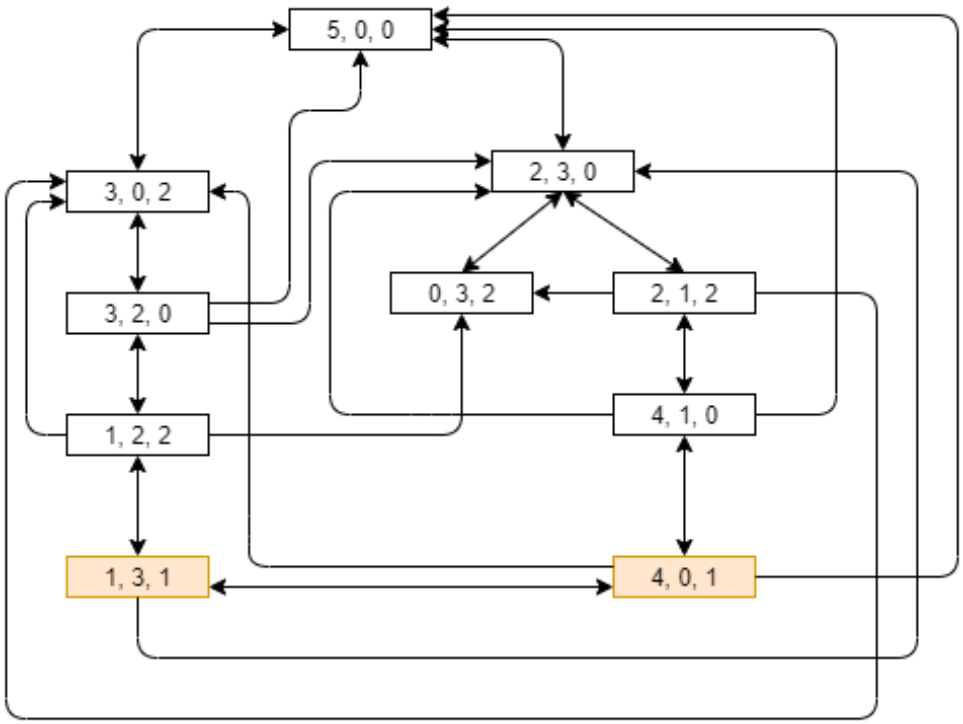
1. $a_j > 0$, tehát a i-edik kancsóban van víz

2. $a_j < max_j$, vagyis a j-edik kancsó nincs tele

Az áttöltött víz mennyisége:

$$v_{ij} = \begin{cases} a_i & \text{ha } a_i \leq max_j - a_j; \\ max_j - a_j & \text{ha } a_i > max_j - a_j. \end{cases}$$

Az alábbi 2.10. ábra a kancsós feladat reprezentációs gráfját illusztrálja. Látható, hogy a lehetséges 12 állapotból kettő, nevezetesen a $(3, 1, 1)$ és a $(2, 2, 1)$ tartalmak (mindkettő potenciális célállapot) nem állnak elő, az öntögetések ezeket nem hozzák létre.



2.10. ábra. A kancsók problémájának reprezentációs gráfja

2.3.5. A misszionárius-kannibál probléma

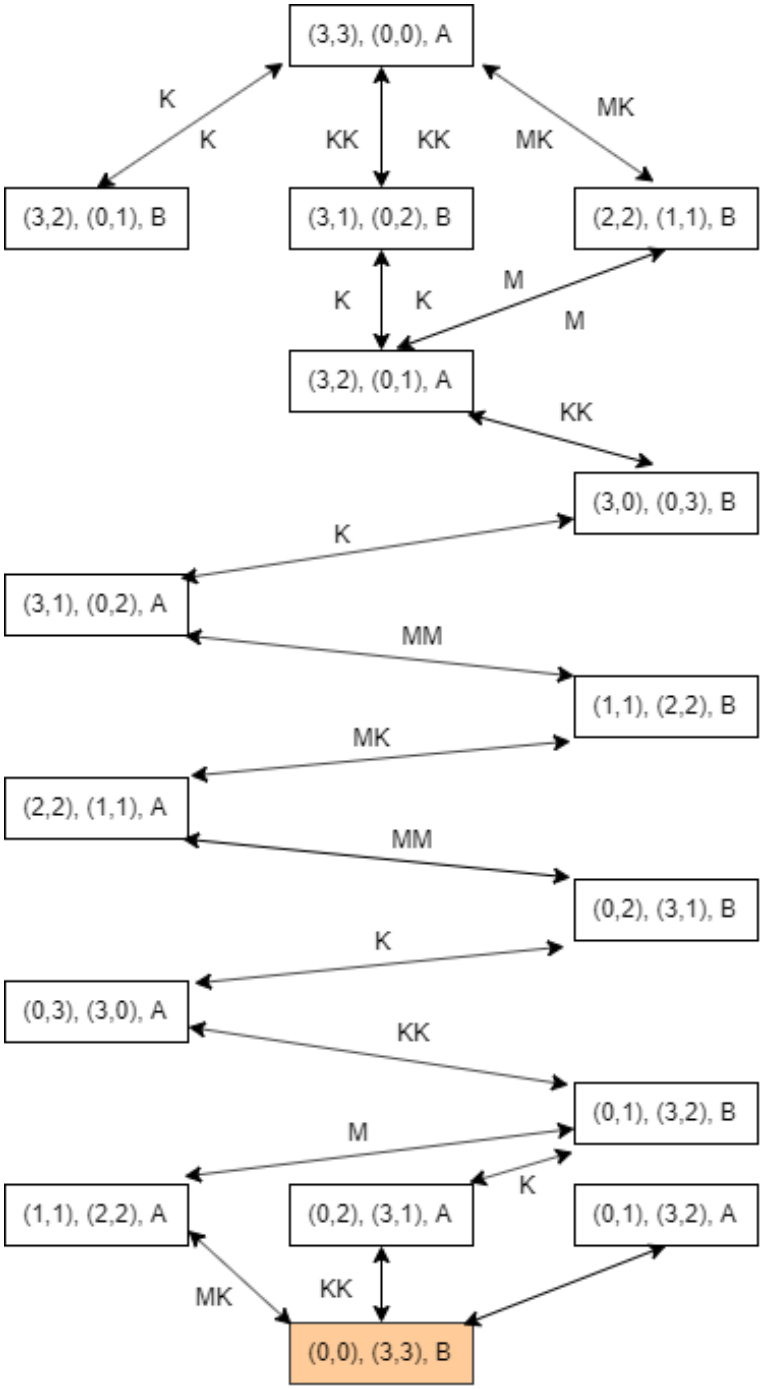
Az utolsóként bemutatott, ugyancsak jól ismert feladat az úgynevezett átkelési problémák közé tartozik és feltehetően a legrégebben megfogalmazott feladat közülük.

5. Feladat A folyó egyik partjáról három misszionárius és három kannibál szeretne átkelni egy csónak segítségével a folyó másik partjára. A csónak kétszemélyes, de egy embert is szállíthat (üresen nem képes átkelni a másik partra). Az átkelési folyamat során sosem lehet az egyik parton több kannibál mint misszionárius. Ez a szabály a csónakban mindig teljesül. Adjuk meg a békés átkelés lépéseit!

A feladat megoldásnak állapotait egy rendezett hármassal írhatjuk le. Nevezzük a folyó két partját A-nak és B-nek. A hármas első tagja megadja azt, hogy az A

parton hány misszionárius, illetve hány kannibál található. A második tag hasonló számokat közöl a B partról. A harmadik tag pedig az üres csónak helyzetét adja meg: ez az A vagy a B part lehet.

A kezdőállapotot a $((3,3),(0,0),A)$ hármas, míg a célállapotot a $((0,0),(3,3),B)$ hármas reprezentálja. A feladat reprezentációs gráfja a 2.11. ábrán látható.



2.11. ábra. A misszionárius-kannibál probléma reprezentációs gráfja

Egy állapot általános képlettel a következőképpen fejezhető ki: $((m_1, k_1), (m_2, k_2), x)$, ahol az alábbi feltételek teljesülnek:

1. $m_1 + m_2 = 3$, a misszionáriusok létszáma három
2. $k_1 + k_2 = 3$, a kannibálok is hárman vannak
3. $m_1 = 0$ vagy $m_1 \geq k_1$, a misszionáriusok nincsenek kisebbségben az A parton
4. $m_2 = 0$ vagy $m_2 \geq k_2$, a misszionáriusok nincsenek kisebbségben az B parton
5. $x = A$ vagy $x = B$, a csónak (üresen) az egyik parton várakozik

A műveleteket egy vagy két betűvel adjuk meg, annak megfelelően, hogy ki illetve kik szálltak be a csónakba az adott parton, pl. MK azt jelenti, hogy egy misszionárius és egy kannibál beszállt a csónakba. A műveletbe azt is bele értjük, hogy a csónak átmegy a másik partra és ott utasai kiszállnak.

3. fejezet

Visszalépéses keresés

Ebben a fejezetben az egyik leggyakrabban alkalmazható probléma megoldó módszert, a visszalépéses keresést ismertetem. A visszalépéses keresést is a feladatok reprezentációs gráfján értelmezzük. Amint azt láttuk, a mesterséges intelligencia egyszerű alapfeladatainak a leírása egy reprezentációs gráfot eredményez. Valójában ez a gráf nem áll a keresés rendelkezésére eleve adott formában, hanem azt a keresés algoritmusa maga építi fel, esetleg csak részben, a működése során.

A visszalépéses keresést az jellemzi, hogy – a kezdőcsúcsból indulva – az aktuális csúcsban kiválaszt egy műveletet és végrehajtja azt. Így egy útvonalon egyre mélyebbre hatol a reprezentációs gráfban. Ha olyan állapotba ér, ahonnan nem tud tovább lépni, akkor visszalép az azt megelőző csúcsba, és egy addig nem próbált művelettel próbálkozik. Ha tehát az algoritmus zsákutcába fut, akkor visszalépve új irányban próbálkozik.

Egy csúcsból akkor kell visszalépni, ha nem vezet belőle tovább út, vagy ha minden belőle kiinduló útról bebizonyosodott, hogy nem vezetnek célba. Azt, hogy a műveleteket milyen sorrendben alkalmazzuk, mindig az adott feladatnál adjuk meg, a visszalépéses keresés algoritmusa ezt nem definiálja.

Az algoritmus alkalmazása során csupán egyetlen utat tartunk nyilván, amely a kiinduló állapotból az aktuális állapotba vezet. Az útvonal csúcsainál fel kell jegyezni az alkalmazható, de még nem kipróbált műveleteket is.

Úgy lehet tekinteni, hogy a visszalépéses keresés algoritmusa alapvetően kétféle szabállyal dolgozik. Az első kategóriába tartoznak az állapotokon végzett műveletek, ezek meghosszabbítják a keresési utat. A másik fajta művelet a visszalépés, amely rövidíti a megtett útszakaszt. Ezt csak akkor hajthatjuk végre, ha az aktuális állapotban nincs alkalmazható művelet.

A visszalépéses keresés három változatát ismertetem. Az első változat (amelynek bevezetése saját eredményemnek mondható) véges és körmentes gráfokban minden

célcsúcsot megkeres, de a hozzájuk vezető műveleteket nem adja vissza. A második változat ugyanilyen gráfon egy célállapotot keres és az oda vezető műveletsorozatot is visszaadja egy lista formájában. A visszalépéses keresés harmadik változata olyan reprezentációs gráfokon is eredményes keresést képes végrehajtani, amelyek köröket, illetve – ettől függetlenül is – végtelen utakat tartalmaznak. A harmadik változat a mélységi korlát bevezetésével és az állapot-ismétlés ellenőrzésével dolgozik és egy célállapotot keres, az oda vezető műveletsorozattal együtt.

3.1. A visszalépéses keresés első változata

Gondoljunk a négy-királynő problémára, amelyben minden célállapot meg kell keresnünk. A műveletsorozatokat nem kell visszaadni, mert azok a célállapotokból egyszerűen leolvashatók.

A reprezentációs gráf ebben a feladatban eleve fastruktúrájú, mégpedig gyökeres véges irányított fa. Gráf esetében is megköveteljük azonban, hogy a gráf véges és körmentes legyen. (Ilyenkor az egyetlen különbség az, hogy egy csúcsnak több megelőzője is lehet, így egyértelmű szülő csúcsról nem beszélhetünk.)

A reprezentációs gráf nincs explicit módon megadva a memóriában. A keresés ebből egyetlen útvonalat tart nyilván, azaz épít és fogyaszt dinamikus módon. Ha a reprezentációs gráf nem fa, akkor újra elérhet és feldolgozhat egy olyan csúcsot, ahol már járt korábban. (A visszalépéses keresést alapvetően ez különbözteti meg a mélységi gráfkereséstől, amely a gráf bejárt részét nyilvántartja és színezi, így egy állapotot csak egyszer – az első elérés alkalmával – keres fel, feldolgozás és továbbhaladás céljával.)

```
VISSZLÉPÉSES-KERESÉS-1 (állapot)
1 if Céllállapot(állapot) then
2     Összefűz(állapot-lista, állapot)
3 művelet-lista := Alkalmazható-műveletek(állapot)
4 while not Üres(művelet-lista) do
5     művelet := Első(művelet-lista)
6     művelet-lista := Többi(művelet-lista)
7     következő-állapot := művelet(állapot)
8     VISSZALÉPÉSES-KERESÉS-1(következő-állapot)
9 return
```

A keresés első változata a preorder bejárás logikáját követi. A kezdőállapotot reprezentáló startcsúcsból indulva mindig megvizsgálja és feldolgozza az aktuális csúcsot, majd rekurzív módon bejárja annak összes leágazását. Közben egyetlen teendője az, hogy ha célcsúcshoz ér, akkor azt kiírja egy listába és az alatt már nem vizsgálódik, hanem onnan visszatér az aktuális megelőző csúcsba.

Az algoritmus leírásában alkalmazott jelölések önmagukért beszélnek. Az *állapot* változó értéke a reprezentációs gráf egy csúcsa, amely egy állapotot reprezentál. A *Céllállapot* függvény akkor ad vissza igaz értéket, ha az argumentuma egy célállapot. Ebben az esetben az elért (újabb) célcsúcs olyan formán kerül kiírásra, hogy a globális hatáskörrel deklarált *állapot-lista* változóhoz fűzi az eljárás.

A *művelet-lista* a vizsgált állapotra alkalmazható összes produkciós szabály listája, ezt az *Alkalmazható-műveletek* függvény tölti fel. (Itt befolyásolhatjuk a kiválasztott műveletek sorrendjét.) Az eljárás iterálva sorra veszi mindegyik műveletet, a soron lévőt leválasztja a listáról és képezi az aktuális állapot rákövetkezőjét, amire a visszalépéses keresés rekurzív módon meghívja önmagát.

Végül, a teljes gráf bejárása után – ami alól kivételt jelentenek a célcsúcs alatti részek, ha lennének ilyenek – az eljárás visszalép a külső hívás helyére és ezzel elhagyja a gráfot. A bejárás jellegű keresés végrehajtása után az *állapot-lista* tartalmazza az összes megtalált célállapotot.

3.2. A visszalépéses keresés második változata

A legtöbb feladat egyetlen célállapot megkeresését írja elő, az oda vezető műveletsorozat meghatározásával együtt. Általában ilyenkor csak egy célállapota van a feladatnak, amelyet legtöbbször ismerünk is, de a hozzá vezető műveletsorozatot már nem, így valójában annak meghatározása a feladat. Ilyen a nyolcas-játék, a Hanoi tornyai probléma vagy az átkelési feladat. A kancsók feladata több célállapottal rendelkezik (ezeket nem ismerjük), de dönthetünk úgy hogy elég egyet megadni az eléréséhez szükséges műveletsorozattal együtt.

A visszalépéses keresés második változata tehát egy célállapot elérésére törekszik. Ha megtalálta azt (vagy egy ilyet), akkor innen már a legrövidebb úton visszatér a vezérlés és a célcsúcsba vezető és közben számon tartott műveletsorozatot visszaadja a meghívás helyére.

Ennek a változatnak speciális jellemzője az, hogy nincs felkészülve a legáltalánosabb reprezentációs gráfra, nevezetesen a keresés két „kellemetlen” kísérő jelenségére: nem kezeli megfelelően a körök létrejöttét és a végtelen keresési útvonal jelenségét.

A reprezentációs gráfról fel kell tennünk tehát, hogy véges és körmentes, vagyis olyan tulajdonságú, mint az első változat esetében.

Mindenképpen érdemes azonban az eljárás ezen változatát is megadni, hiszen az előző változathoz képest néhány új algoritmikus elem szerepel benne.

```
VISSZLÉPÉSES-KERESÉS-2 (állapot)
if Céllállapot(állapot) then
    return (NIL)
művelet-lista := Alkalmazható-műveletek(állapot)
talált := FALSE
while not (Üres(szabály-lista) or talált) do
    művelet := Első(művelet-lista)
    művelet-lista := Többi(művelet-lista)
    következő-állapot := művelet(állapot)
    művelet-sorozat := VISSZALÉPÉSES-KERÉS-2(következő-állapot)
    if művelet-sorozat ≠ NONE then
        talált := TRUE
if talált then
    return (Összefűz(művelet, művelet-sorozat))
else return NONE
```

Az eljáráshoz fűzött magyarázatban nem ismételjük meg azokat az elemeket, amelyek változatlanul ugyanúgy érvényesek, mint az első változat esetén.

A függvények használatában nincs változás. A pszeudokódban két új alapszó jelenik meg. A NIL egy üres listát jelöl. A $NONE$ alapszó általában valamilyen hiányra vagy sikertelenségre utal, ami azonban nem számít hibának.

A visszalépéses keresés ezen változata jórészt ugyanazokat a változókat használja, mint az első verzió. Egyetlen különbség az output változóban mutatkozik. Míg az első változat a globális *állapot-lista* változóban állította elő a célállapotok listáját, addig a jelenlegi változat egy megtalált célállapothoz vezető műveletsorozatot ad vissza a *művelet-sorozat* változóban. Ez is egy lista típusú adatszerkezet, azonban nem globális hatáskörű, hanem a rekurzív hívásokból való visszalépések során épül a konkatenáció műveletével.

Az algoritmus most is rekurzív módon működik. Az eljárás első változata alapján érthető a keresés logikája, a következő kiegészítésekkel. Ha az eljárás megtalált egy célállapotot, akkor azt az üres lista (NIL) visszaadásával jelzi. Ha az aktuális állapot

(akár a kezdőállapot esetén, akár a későbbi rekurzív hívások során) nem célállapot, akkor az feldolgozásra kerül. Ez azt jelenti, hogy képezni kell a rá alkalmazható műveletek listáját és azokat rendre alkalmazva az aktuális állapotra, a következő állapotra rekurzív keresést kell végrehajtani.

Ha a *következő-állapot* változó tartalmából kiinduló keresés célállapotot talál, akkor a sorozatos visszatérések után a *művelet-sorozat* értéke egy kialakulóban lévő műveleti lista (tehát nem $NONE$). Ilyenkor a *talált* változó $IGAZ$-ra állításával elérjük azt, hogy az adott szinten termináljon a feldolgozási ciklus, hiszen ebben az esetben már csak a visszalépések sorozatára van szükség. Az iterációból kilépve kerül sor a *művelet-sorozat* lista bővítésére, konkatenáció műveletével és ezt visszalépésnek kell követnie a hívás magasabb szintjére.

Ha a *következő-állapot*-ból kiinduló keresés nem vezet célállapothoz, akkor azt az eljárás a $NONE$ szövegkonstans alapszó visszaadásával jelzi. Ilyenkor tehát a rekurzív hívás után a *művelet-sorozat* értéke $NONE$ lesz és ebben az esetben folytatni kell az aktuális állapot lehetséges rákövetkezőinek iterált feldolgozását.

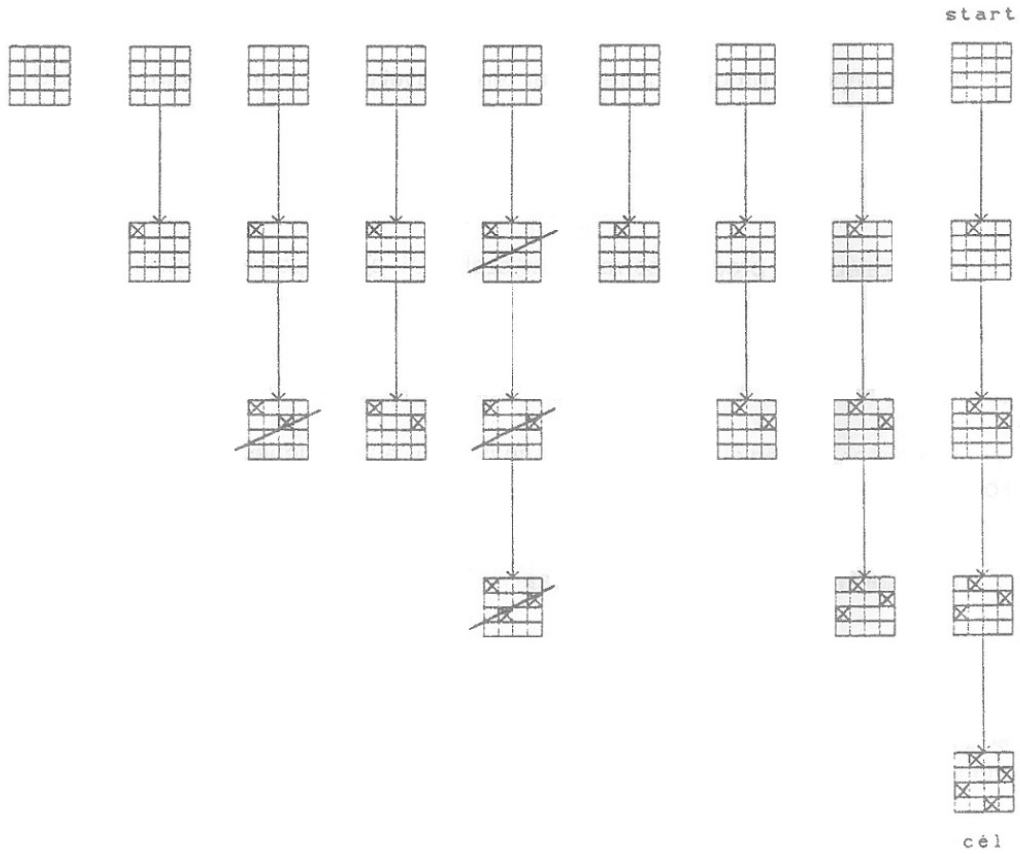
Ha az aktuális állapot minden rákövetkezőjével indított rekurzív keresés sikertelen, akkor a $NONE$ érték visszaadásával visszalépést kell kezdeményezni az eljáráshívás felső szintjére.

Végül, a külső hívás után rendelkezésünkre áll a megtalált célállapothoz vezető műveletsorozat, illetve $NONE$ érték jelzi azt, ha a reprezentációs gráf nem tartalmazna célállapotot.

Megismételjük, hogy a visszalépéses keresés második változata csak véges, körmentes gráfokon működik biztosan az elvárásoknak megfelelően. Ilyen esetekben az algoritmus futása véges időben befejeződik, és ha létezik célállapot, akkor ezek közül egyet biztosan megtalál.

A 3.1. ábrán a négy királynő probléma megoldásának azokat a lépéseit láthatjuk részletesen, amelyek az első célállapot megtalálásához vezetnek. Célszerű, ha a keresés útvonalát a 2.2. ábra reprezentációs fáján is követjük. Látható, hogy az első célállapothoz négy visszalépéssel jutunk el. Ez egyaránt igaz az algoritmus első és második változatára.

A visszalépéses keresés első változata még tovább működne ez után, a teljes fa bejárásával megtalálná a másik célállapotot is. A második változat csak eddig keres és a itt befejezi működését, amelynek során az alkalmazott szabályok listáját is feltöltötte a célállapothoz vezető királynő elhelyezésekkel.



3.1. ábra. A négy királynő probléma megoldása visszalépéses algoritmussal

3.3. A visszalépéses keresés harmadik változata

Az eddig látott két visszalépéses algoritmus – ahogyan az előző változatnál kitértem rá – nincs felkészülve a végtelen útvonal vagy kört tartalmazó keresési út létrejöttére. Ennek a két jelenségnek a kezelése valósul meg a visszalépéses keresés újabb, harmadik változatában.

A végtelen keresési út létrejöttét kizárhatjuk azzal, hogy egy *mélységi korlátot* vezetünk be. Így az algoritmus működése során nem hoz létre a megadott korlátnál hosszabb utakat. Ha az eljárás eléri az adott mélységi korlátot, akkor visszafordul és visszatér az eggyel magasabb szintre.

A kört tartalmazó gráfrészlet a másik olyan jelenség, amely nem-termináló keresést eredményezhet. Úgy tűnik első ránézésre, hogy a mélységi korlát ezt a problémát is megoldja. Ez a legtöbb gyakorlati esetben így lehet, azonban elméletileg elképzelhető, hogy a keresés „oda-vissza” lépeget a körön, miután elérte a mélységi korlátot.

A gráfbeli körök okozta problémát garantáltan megoldhatjuk azzal, ha az eljárásunk minden új lépésnél megnézi, hogy az aktuális csúcs szerepelt-e már a nyilván-

tartott úton. Így mindig olyan utat hozunk létre, amiben csak különböző állapot szerepel. Ennek az az implementációs következménye, hogy az eljárás paramétere nem egyetlen állapot, hanem az állapotok egy listája lesz.

A visszalépéses keresés harmadik változatában mindkét fenti módszer szerepel.

```
VISSZALÉPÉSES-KERESÉS-3(állapot-sorozat)
1  állapot := Első(állapot-sorozat)
2  if Eleme(állapot, Többi(állapot-sorozat) then
3      return (NONE)
4  if Céllállapot(állapot) then
5      return NIL
6  if Hossz(állapot-sorozat) > mélységi-korlát then
7      return (NONE)
8  művelet-lista := Alkalmazható-műveletek(állapot)
9  talált := HAMIS
10 while not (Üres(szabály-lista) or talált) do
11     művelet := Első(művelet-lista)
12     művelet-lista := Többi(művelet-lista)
13     következő-állapot := művelet(állapot)
14     új-állapot-sorozat := Összefűz(következő-állapot, állapot-sorozat)
15     művelet-sorozat := VISSZALÉPÉSES-KERESÉS-3(új-állapot-sorozat)
16     if művelet-sorozat ≠ NONE then
17         talált := IGAZ
18 if talált then
19     return (Összefűz(művelet, művelet-sorozat))
20 else return (NONE)
```

Az eljárás magyarázatában itt sem térek ki azokra a komponensekre, amelyek szerepeltek az előző két változatban. Az eddig is használt függvények mellet újként jelenik meg az *Eleme* függvény, amely azt dönti el, hogy egy változó értéke megtalálható-e egy lista elemei között, illetve a *Hossz* függvény, amely egy lista hosszát, azaz elemeinek a számát adja vissza.

A változó között megjelenő *mélységi-korlát* nem szerepel az eljárás paraméterei között, ezért tekintsük úgy, hogy globális úton kap értéket. Az *állapot-sorozat* az eljárás be- és kimenő paramétere, típusát tekintve állapotok listája, ebben tartja

nyilván a keresési utat. Az *új-állapot-sorozat* az előbbi listának az eljárás adott szintjén kiegészített változata.

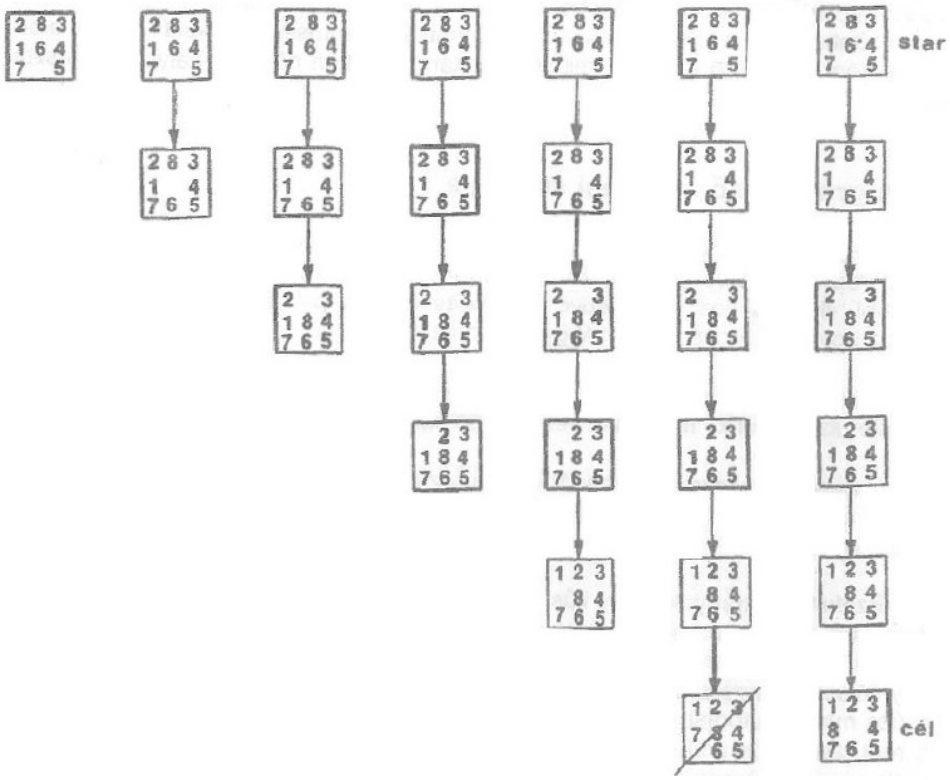
A visszalépéses keresés – szintén rekurzív – harmadik változata vezérlési szerkezetében nagyban hasonlít a második változatra.

A meghívást követő azonnali pozitív visszatérés mellett (célállapot elérése) két újabb esetet kell megvizsgálni. Azonnali sikertelen visszatérés indokolt abban az esetben, ha a keresés elérte a mélységi korlátot, illetve akkor is, ha aktuális elem szerepel a keresés nyilvántartott útvonalán, vagyis ha kör valósult meg a keresés során.

A keresés ciklusa egyetlen új utasítással bővül (14. sor), amely előállítja a rekurzív hívás (15. sor) bemenő paraméterét.

Ez a változat minden feladatra terminál, de ez nem feltétlenül jelenti azt, hogy sikeresen. A *mélységi korlát* jó megválasztása fontos, hiszen csak az annál hosszabb utakat nem vizsgálja meg az eljárás, így az annál mélyebben fekvő célcsúcsokig nem jut el a keresés.

A 3.2. ábrán egy újabb példát láthatunk a visszalépéses keresésre.



3.2. ábra. A 8-as játék megoldása visszalépéses algoritmussal

Ezúttal a 8-as játék célállapotát találja meg az eljárás – szerencsés paraméterezés mellett – mindössze egy visszalépés alkalmazásával. Az üres hely cseréjének a sorrendjét úgy határoztuk meg, hogy a próbálkozások sorrendje (*föl, balra, le, jobbra*) legyen. Tegyük fel, hogy legalább a 2-hosszúságú körök figyelését és kizárását elvégzi az algoritmusunk, azaz soha nem próbálkozik visszalépéssel. A mélységi korlátot 5-nek választva valóban adódik a gyors eljutás a célállapothoz.

A visszalépéses keresés előnye (1) a könnyű megvalósíthatóság, valamint az, hogy (2) működése kevés tárhelyet igényel, hiszen mindig csak az aktuális utat kell megjegyezni.

Hátránya az, hogy (1) egy magas szinten elkövetett nem célirányos lépésre esetleg csak sok lépés után derül fény, illetve, (2) ha van egy csúcs, amely sok másikból elérhető, de belőle nem vezet út a célcsúcsba, akkor az algoritmus újra és újra rálép erre a csúcsra és onnan mindig tovább is megy, hiszen nem tárolja a memóriában a már bejárt és „téves választásnak” bizonyult csúcsokat.

4. fejezet

Gráfkereső eljárások

Az előző fejezetet, amely a visszalépéses keresésről szólt, az eljárás előnyeinek és hátrányainak a számbavételével fejeztem be. Láttuk, hogy ennek az egyszerű stratégiának két gyenge pontja van:

- Egy „rossz” irányválasztásra, esetleg csak hosszú út megtétele után derül ki, hogy nem vezet megoldáshoz.
- Az eljárás nem jegyzi meg a célállapothoz nem vezető, már bejárt csúcsokat, így ezekhez újra és újra visszatérhet.

Mindkét problémára megoldást adhat, ha a kereső eljárás minden olyan, a kezdőcsúcsból kiinduló utat nyilvántart, amelyeket már bejárt. Az a probléma, amelyet egy korábban felkeresett csúcs újbóli bejárása okoz, ezzel a stratégiával megelőzhető. A kedvezőtlen csúcsválasztás problémáját pedig azzal lehet feloldani, hogy egy lépésben a nyilvántartott utak közül bármelyiknek a végpontja választható és onnan lehet tovább folytatni a keresést. Így lehetségessé válik, hogy minden lépésben azt az utat válasszuk, amely a „legjobbnak” tűnik.

Ez a keresési stratégia jóval nagyobb szabadságot enged, mint a visszalépéses keresés. Még egy könnyítést vezetünk be: az aktuális csúcsnak nem egyetlen rákövetkezőjét képezzük, hanem mindet, amely egy művelet alkalmazásával létre hozható. Ezt a lépést nevezzük egy csúcs kiterjesztésének.

Azt a keresési stratégiát, amely a startcsúcsból indulva a reprezentációs gráf bejárt részgráfját nyilvántartja, gráfkereső eljárásnak nevezzük. Az eljárás által felépített részgráfot pedig keresőgráfnak nevezzük.

Megjegyzem, hogy a gráfra ebben a fejezetben egy absztrakt adatstruktúraként gondolunk, amelyen lényegében megegyezik azzal a képpel, amelyet a matematika használ a gráfok elméletében. A mesterséges intelligencia feladataira tekintettel a

„gráf csúcsa” és az „állapot” kifejezéseket szinonimaként használom ugyanúgy, ahogy a visszalépéses keresésnél is tettem.

A fejezet további részeiben először megfogalmazom a gráfkeresés alapalgoritmusát, amely – még bizonyos részletek figyelmen kívül hagyásával – célcsúcsot keres a feladat reprezentációs gráfjában.

Utána az általános gráfkereső eljárás leírását adom meg, amely az alkalmazható műveletek költségeit, ezáltal a keresési utak költségeit is figyelembe veszi a következő lépés megválasztásában.

A gráfkeresés általános algoritmusából két kereső eljárás lehet származtatni: a szélességi és a mélységi keresést. Ezeket olyan gyakran alkalmazzák, hogy célszerű őket külön is ismertetni.

4.1. A gráfkeresés kiinduló algoritmusa

A következő eljárás a gráfkeresés alap-algoritmusának tekinthető. Az algoritmus – egyelőre bizonyos részletek kidolgozásától eltekintve – célcsúcsot (célállapotot) keres egy feladat reprezentációs gráfjában, amelyet az alkalmazható műveletek függvényei implicit módon határoznak meg.

GRÁFKERESÉS-0

1 $G := \{s\}$; *nyílt* $:= \{s\}$
2 $n := s$
3 **while not** *Célállapot(n)* **or** *Üres(nyílt)* **do**
4 $\quad$ *nyílt* $:=$ *nyílt* $\setminus \{n\}$
5 $\quad G := G \cup \Gamma(n)$
6 $\quad$ *nyílt* $:=$ *nyílt* $\cup\ \Gamma(n)$
7 $\quad n :=$ *Elem(nyílt)*

Az eljárás a reprezentációs gráf kezdőcsúcsából, vagyis a feladat kezdőállapotából kiindulva felépít egy G keresőgráfot. Ez a gráf tartalmaz minden olyan, a kezdőpontból kiinduló utat, amelyet már bejárt a gráfkereső algoritmus. Nyílt csúcsoknak nevezzük azokat a csúcsokat, amelyek utódai még nem szerepelnek a keresőgráfban. Ezeket a nyílt halmazban külön is tároljuk. A G gráf többi csúcsát, amelyek már feldolgozásra kerültek, zártnak nevezzük.

A keresés a megfelelő inicializálásokkal kezdődik (1-2. sor). A kezdőállapot lesz a keresőgráf első csúcsa, egyúttal az aktuális csúcs, amely nyílt tulajdonságú, hiszen még nem került kiterjesztésre.

Az algoritmus ezt követően iterálva keres egy célcsúcsot a gráfban. Az eljárás működése két okból érhet végét: vagy talál egy célcsúcsot, vagy pedig kiürül a *nyílt-csúcsok* halmaz, azaz nem folytatható tovább a gráf kiterjesztése (3. sor).

Nézzük, mi történik egy iterációs lépésben (4-7. sor)! Az eljárás törli az előzőleg kiválasztott csúcsot a nyílt halmazból, utána pedig a Γ operátor alkalmazásával elvégzi az aktuális csúcs kiterjesztését. Az alkalmazható műveletek által meghatározott rákövetkező csúcsok csatolásra kerülnek a keresőgráfhoz, továbbá a nyílt csúcsok halmazában is megjelennek új elemként. A ciklusmag utolsó lépésében a nyílt csúcsok közül kiválasztásra kerül az új aktuális csúcs, a következő iterációs lépés számára.

A gráfkeresés alapalgoritmusa nyitva hagyja azt a kérdést, hogy milyen szempontok alapján kerül kiválasztásra a következő nyílt csúcs (7. sor). Ezt részletesen vizsgáljuk majd a következő alfejezetben.

Az algoritmus „gyengéje” az a stratégia, hogy az aktuális n csúcs $\Gamma(n)$ rákövetkezői mind bekerülnek a nyílt halmazba, még akkor is, ha egy rákövetkező csúcs már szerepel a keresőgráfban és feldolgozásra is került. Eljárásunk azonban csak egy elvi, algoritmikus „vázlat”, amelyet felhasználunk a következő, már implementálható kereső eljárás kialakításában.

4.2. Általános gráfkereső algoritmus

Az előző rész végén rámutattam arra, hogy a gráfkereső algoritmus kidolgozása során tisztázni kell néhány kérdést. Ezt úgy tudjuk megtenni, ha előbb bevezetjük műveletek költségének fogalmát. A feladatok megoldása során az állapotokra műveleteket alkalmazunk, így próbálunk egy célállapothoz eljutni. A műveleteknek általában nemnegatív költségük van, ami absztrakt szinten a reprezentációs gráf éleihez rendelt költségértékek formájában jelentkezik.

Legyen $g(n)$ ez a költségfüggvény, nyilvánvalóan additív számítási móddal: egy útvonal költsége az élek költségének összegeként áll elő. A kiterjesztés előre haladásának megfelelően, ha az utolsó lépésben az m csúcs kiterjesztése az n csúcsot is létrehozza, akkor $g(n) = g(m) + c(m, n)$, ahol $c(m, n) \geq 0$ az (m, n) él költségét jelenti.

Természetes gondolat, hogy a reprezentációs gráf minden csúcsához egy utat tartunk nyilván, méghozzá a legkisebb költségű utat azok közül, amelyeket a keresés

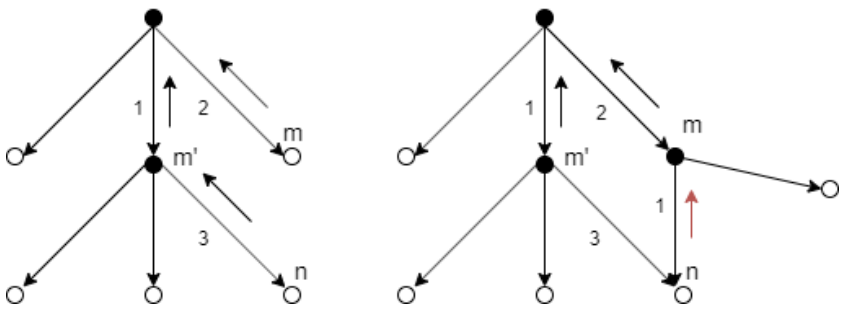
egy adott időpontban már bejárt. Ha a keresés által létrehozott gráfban minden csúcshoz egyetlen utat tüntetünk ki, akkor reprezentációs gráf (bejárt részének) feszítőfájához jutunk.

Az egyes utak megjelenítésére visszafelé mutató pointereket használunk. A startcsúcsot kivéve minden csúcs pointere a szülő csúcsra mutat. Ha például az n csúcs szülője a feszítőfában az m csúcs, akkor $p(n) = m$, illetve az s kezdőcsúcsra $p(s) = NIL$.

Most már megvizsgálhatjuk a nyitva hagyott kérdéseket. Először is azt, hogy milyen kritérium alapján válasszunk ki egy nyílt csúcsot kiterjesztésre. A gyakorlati alkalmazásokban általában bevezetünk egy $f(n)$ kiértékelő függvényt és a nyílt csúcsok közül szokás szerint egy minimális f értékkel rendelkező csúcsot választunk kiterjesztésre.

Nagyon lényeges tisztázni az $f(n)$ és a $g(n)$ függvények kapcsolatát! Aki az algoritmusok és adatstruktúrák világa felől közelít a mesterséges intelligenciához, annak számára nyilvánvalónak látszik az az összefüggés, amely a két függvény közé egyenlőséget tesz. Erre a szemléletre még visszatérek, de előbb rögzíteni szeretném azt, hogy a mesterséges intelligenciában vannak ugyan ésszerű megfeleltetések a két függvény között, de az általános gráfkeresés szintjén nem tételezünk fel semmilyen összefüggést közöttük.

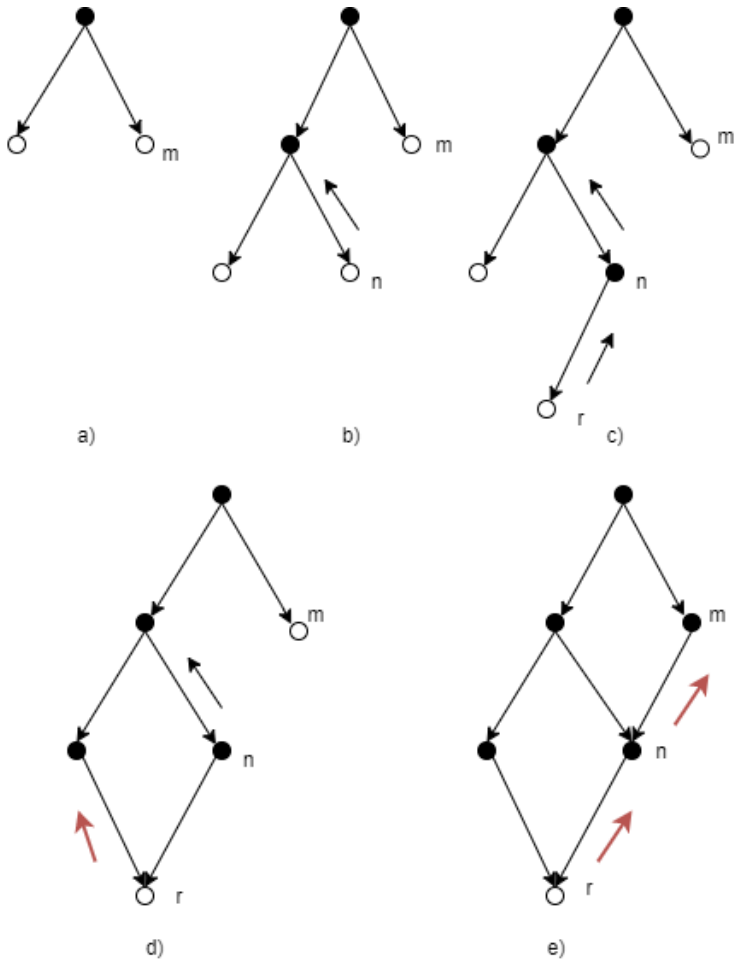
Az előző alfejezet végén jelzett problémák között szerepelt annak a kérdése is, hogy mi történjen akkor, ha egy már megtalált csúcshoz nem először jut el a keresés. Egyszerűbb az az eset, ha egy nyílt csúcshoz jut el újból a keresés.



4.1. ábra. Nyílt csúcs pointerének beállítása

A 4.1. ábrán láthatjuk, hogy hogyan állítja át az algoritmus egy csúcs pointerét ha talál egy olcsóbb hozzá vezető utat. Az első ábrán, ahol a kezdőcsúcsból n-be az m'-n keresztül jutottunk el, $g(n)$ értéke 4 volt. Mivel az m-en keresztül már csak 3 lesz ez az érték, a $p(n)$ pointer pedig ezután már m-re mutat. (Az n csúcs természetesen továbbra is nyílt marad, hiszen egy csúcs akkor válik zárttá, amikor az algoritmus kiválasztja kiterjesztésre.)

Meg kell vizsgálnunk azt a lehetőséget is, ha az m csúcs kiterjesztésekor egy zárt n csúcshoz jutunk el és kisebb költségű utat találunk hozzá. Ekkor az n leszármazottjainak az elérési költsége is csökkenhet. Erre látunk példát a 4.2. ábrán, ahol az m csúcs kiterjesztése után nem csak az n csúcs elérési költsége csökken, hanem az r csúcsé is, amely az n leszármazottja. Az aktuális állapotot az tükrözi konzisztens módon, hogy az n és az r csúcsnál egyaránt átállítjuk a költségértéket és a pointert.



4.2. ábra. Zárt csúcs és utódja pointerének a beállítása

Arra gondolhatunk, hogy az n csúcsból egy lokális gráfkereső eljárást indítunk, ennek megvalósítása azonban nem tűnik egyszerűnek. A probléma egyszerűbb módon megoldható úgy is, hogy a zárt n csúcsnál átállítjuk a $g(n)$ költségértéket és a $p(n)$ pointert, majd az n csúcsot újból a nyílt halmazba helyezzük. Ekkor átmenetileg elveszíthetjük a konzisztenciát a keresőgráfban, de az n csúcs későbbi, újbóli kiterjesztése javít ezen.

Ezek után megadom az általános gráfkeresés algoritmusát.

Gráfkeresés

1 $G := \{s\}$; *nyílt* $:= \{s\}$
2 $n := s$; $g(s) := 0$; $p(s) := NIL$
3 **while not** *Célállapot(n)* **or** *Üres(nyílt)* **do**
4 $\quad$ *nyílt* := *nyílt* $\setminus \{n\}$
5 $\quad$ $M := \Gamma(n)$
6 $\quad$ **while not** *Üres(M)* **do**
7 $\quad\quad$ m := *Elem(M)*; $M := M \setminus \{m\}$
8 $\quad\quad$ **if** $m \notin G$ **or** $g(m) > g(n) + c(n, m)$ **then**
9 $\quad\quad\quad$ $g(m) := g(n) + c(n, m)$; $p(m) := n$
10 $\quad$ $G := G \cup \Gamma(n)$
11 $\quad$ $n := min_f$ *(nyílt)*

A gráfkeresés kiinduló algoritmusánál szereplő magyarázatot nem ismétlem, csak az új algoritmikus lépésekre térek ki. Bevezetjük a $g(n)$ költségfüggvényt és a $p(n)$ pointereket, amelyek megjelennek a kezdőértékek kialakításában is (1-2. sor).

Az eljárás ciklusfeltétele ugyanaz, mint a kiinduló algoritmus esetében (3.sor). Az is változatlan, hogy az eljárás a kiválasztott csúcsot kiveszi a nyílt halmazból és a Γ operátor alkalmazásával kiterjeszti azt (4-5. sor).

Új algoritmikus finomítás az, hogy az n csúcs kiterjesztésének minden egyes $m \in \Gamma(n)$ csúcsát külön kezeli az eljárás (6-7. sor). Ha az m csúcs újként jelenik meg a G keresőgráfban, akkor a nyílt halmazba kerül, mint ahogy akkor is, ha m nem új csúcs, de a keresés kisebb költségű utat talált hozzá (8-10. sor).

Az n csúcs $\Gamma(n)$ kiterjesztését a G keresőgráfhoz csatolja az eljárás (13. sor). Új elem a nyílt halmazból való választás elve is: ez az f függvény minimális értéke szerint történik (14. sor).

Kimondjuk a bemutatott algoritmus működéséről szóló tételt.

4.1. Tétel. Az általános gráfkereső eljárás véges reprezentációs gráfra mindig terminál. Ha a reprezentációs gráf tartalmaz célcsúcsot, akkor az eljárás mindig megtalál ezek közül egyet.

Bizonyítás. Ha a gráf körmentes, akkor minden csúcshoz véges sok különböző út vezet a startcsúcsból. Így egy csúcs véges sokszor kerülhet a nyílt halmazba, ahonnan véges sok lépésben eltávolítható. Ha a reprezentációs gráfban nincs célcsúcs, akkor a nyílt halmaz véges lépésben kiürül és az algoritmus terminál.

Ha a véges reprezentációs gráf tartalmaz köröket, akkor is igaz, hogy minden csúcshoz véges sok körmentes út vezet és ezeknek mindig kisebb a költsége, mint

a megfelelő, köröket tartalmazó utaknak, így azok feltárásának nincs semmilyen módosító hatása a nyílt halmazra.

Ha a véges reprezentációs gráf tartalmaz célcsúcsot, akkor két érvet kell figyelembe vennünk. Először is, ha célcsúcsot választ az eljárás, akkor az a nyílt halmazban marad (a ciklus 4. sora nem kerül végrehajtásra).

Másodszor, az eljárás egyetlen elérhető célcsúcsot sem „veszít el", ugyanis bármely csúcsra, így minden célcsúcsokra is igaz az, hogy a startcsúcsból hozzá vezető (optimális) utat valameddig már feltárta. Az útvonal (esetleg üres) első fele zárt, míg az (esetleg csak a célcsúcsra korlátozódó) második fele nyílt csúcsokat tartalmaz. Egy nyílt csúcs kiterjesztése során megjelenik a következő csúcs ezen az útvonalon, így egy éllel közelebb kerülünk az adott célcsúcshoz.

Egy célcsúcs kiválasztását csak egy másik célcsúcs elérése akadályozhatja meg, de a nyílt halmaz kiürülése semmiképpen sem, mivel az nem jöhet létre. □

4.3. Neminformált gráfkereső eljárások

A gráfkeresés általános algoritmusából több olyan kereső eljárás is származtatható, amelyek nem tartalmaznak az adott feladatra jellemző speciális ismereteket, vagyis az f kiértékelő függvény a feladattól független, előre rögzített stratégia szerint szabályozza a nyílt csúcsok kiválasztásának a sorrendjét. Ezeket nevezzük neminformált eljárásoknak, amelyek közül hármat ismertetek ebben az alfejezetben.

A gráf minden n csúcsánál nyilvántartjuk azt a legkisebb $g(n)$ költséget, amellyel a keresés aktuális állapotában az s kezdőcsúcsból eljuthatunk hozzá. Ezen kívül, még egy pointer is mutat a megelőző csúcsra, amely az eddigi legjobb úton helyezkedik el. A gráf csúcsainak kétféle státusza lehet: egy csúcs nyílt, ha még kiválasztásra kerülhet és zárt akkor, ha egyelőre nem számolunk a kiterjesztésével.

Az általános gráfkereső algoritmus minden lépésben kiválaszt egy nyílt csúcsot kiterjesztésre egy közelebbről nem definiált f kiértékelő függvény minimális értéke szerint. A kiválasztott csúcs minden m szomszédjára ellenőrzi, hogy sikerült-e kisebb költségű utat találni hozzá. Ha igen, akkor az m csúcshoz bejegyzésre kerül a $g(n) + c(n, m)$ költségérték és m-ből az n-be állítjuk a $p(m)$ pointert.

Ha a kisebb költséggel elérhető m szomszédos csúcs már zárt volt, akkor újra nyíltra állítjuk. Ezzel megteremtjük annak a lehetőségét, hogy a későbbi kiválasztás során a kisebb költségű út megtalálása „tovább gyűrűzzön" a leszármazott csúcsokra.

A neminformált gráfkereső eljárások úgy származtathatók az általános sémából, hogy az f kiértékelő függvény definíciójában csak a $g(n)$ költségfüggvényt használjuk fel valamilyen formában. Meg kell jegyezni, hogy a költségfüggvény speciális esete

az, amikor minden él költsége egységnyi és ekkor $g(n)$ a csúcs mélységét jelent egy adott elérési útvonalon, amelyet $d(n)$-nel jelölünk.

A szélességi keresésnél mindig egy legkisebb mélységű csúcsot választunk kiterjesztésre, vagyis $f(n) = d(n)$. A mélységi keresésnél minden lépésben egy legmélyebben fekvő csúcsból próbálunk tovább haladni, aminek megfelel az $f(n) = -d(n)$ választás. A szélességi keresés általános változata az egyenletes keresés, amelyben az élek költsége bármely nemnegatív (gyakorlatban pozitív) érték lehet. A kiválasztás ekkor az $f(n) = g(n)$ függvény minimuma szerint történik. (A korábban bevezetett és tárgyalt visszalépéses keresést ebből a nézőpontból nyilvánvalóan a mélységi kereséssel lehet közvetlen kapcsolatba hozni.)

A neminformált gráfkeresések esetében is mód van arra, növeljük hogy a keresés hatékonyságát. Erre ebben az esetben csupán a csúcsok kiválasztási sorrendjének befolyásolása teremt lehetőséget, azonban a visszalépéses, illetve a mélységi keresésnél ennek is jelentős hatása lehet (lásd a 3.2. ábra példáját).

4.3.1. Mélységi keresés

A mélységi keresés olyan feladatok megoldásra alkalmazható, amelyeknél a műveletek költségének nincs szerepe, vagyis a reprezentációs gráf minden éle egységnyi költségűnek vehető.

A követett stratégia hasonlít a visszalépéses kereséshez. A mélységi keresés is minden lépésben egy legmélyebb nyitott csúcsot választ kiterjesztésre. Ha reprezentációs fában keresünk, akkor a csúcs mélysége egyértelmű. Ha gráfban helyezkedik el a csúcs, akkor a mélységét az eddig feltárt legrövidebb útvonal hossza határozza meg, amely a startcsúcsból oda vezet.

Mivel az f kiértékelő függvény minimális értéke szerint szeretnénk kiterjesztésre egy csúcsot kiválasztani, ezt a következő definícióval tudjuk teljesíteni:

$$f(n) := -d(n)$$

A mélységi keresés a reprezentációs gráf bejárt részét létre hozza a memóriában, szemben a visszalépéses kereséssel, amely mindig csak a keresés aktuális útvonalát tartja nyilván. A mélységi keresés nem „lép rá” egy olyan csúcsra, amelyet korábban már más úton elért (ezt a csúcs lezárásával vagy színezéssel megoldja), míg a visszalépéses algoritmus újra rátévedhet egy zsákutcát megvalósító útvonalra.

A mélységi keresésnek nincs szüksége a visszalépés műveletére, mivel ez az eljárás, egyszerre több utat tart nyilván. A mélységi keresés nem feltétlenül terminál. Ha fennáll annak a lehetősége, hogy a keresés „elmegy” a megoldást jelentő célcsúcsok mellett, akkor célszerű mélységi korlátot alkalmazni.

A 4.3. ábrán a 8-as játék megoldása látható mélységi kereséssel. Az eljárás a *mélységi-korlát* $= 5$ paraméterezéssel működik és az üres hely mozgatásával a $(balra, fel, jobbra, le)$ sorrendben próbálkozik. A keresés egy 5 mélységű célcsúcs megtalálásával terminál.

4.3.2. Szélességi keresés

A szélességi keresés szintén olyan feladatoknál jön szóba, ahol a műveletek költsége egységnyinek tekinthető. A kiértékelő függvényt most a mélységi függvénnyel azonosnak vesszük:

$$f(n) := d(n)$$

A szélességi keresés minden lépésben kiválaszt egy nyílt csúcsot a legkisebb mélységűek közül. A keresés ennélfogva egy adott mélységben minden csúcsot kiválaszt kiterjesztésre és csak azután tér rá az egy szinttel mélyebben fekvő csúcsokra.

A szélességi keresés mindig talál megoldást, ha az létezik, sőt legrövidebb megoldási utat talál. Ezt szemléletesen is elfogadhatjuk, de állításunk következik az 5. fejezet általános tételeiből is.

A szélességi keresés működését a 8-as játékra a 4.4. ábra szemlélteti.

4.3.3. Egyenletes keresés

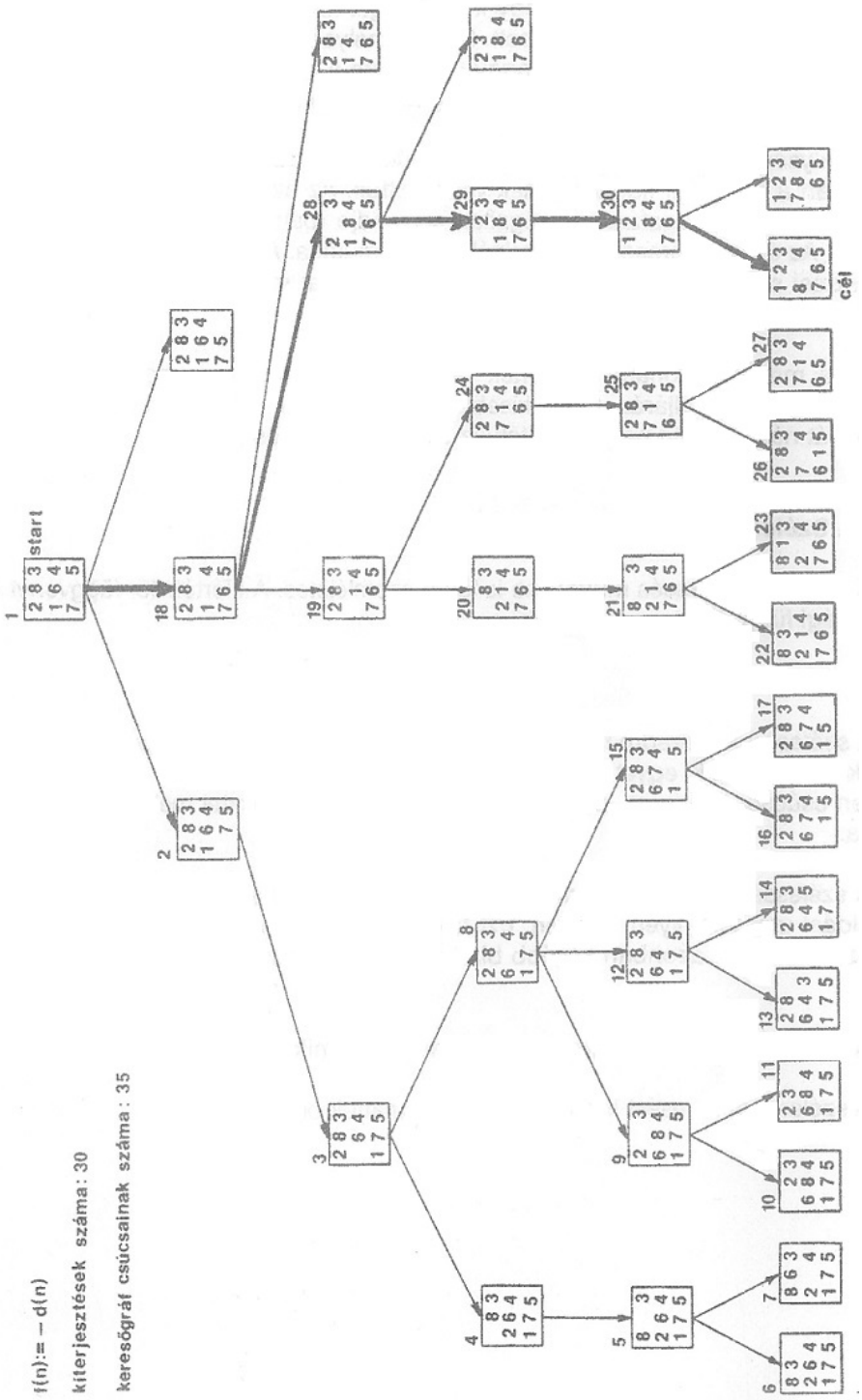
Az egyenletes keresést olyan feladatokra alkalmazzuk, amelyeknél a műveletek valós költségértékekkel rendelkeznek és a reprezentációs gráfban. A kiértékelő függvényt ebben esetben az

$$f(n) := g(n)$$

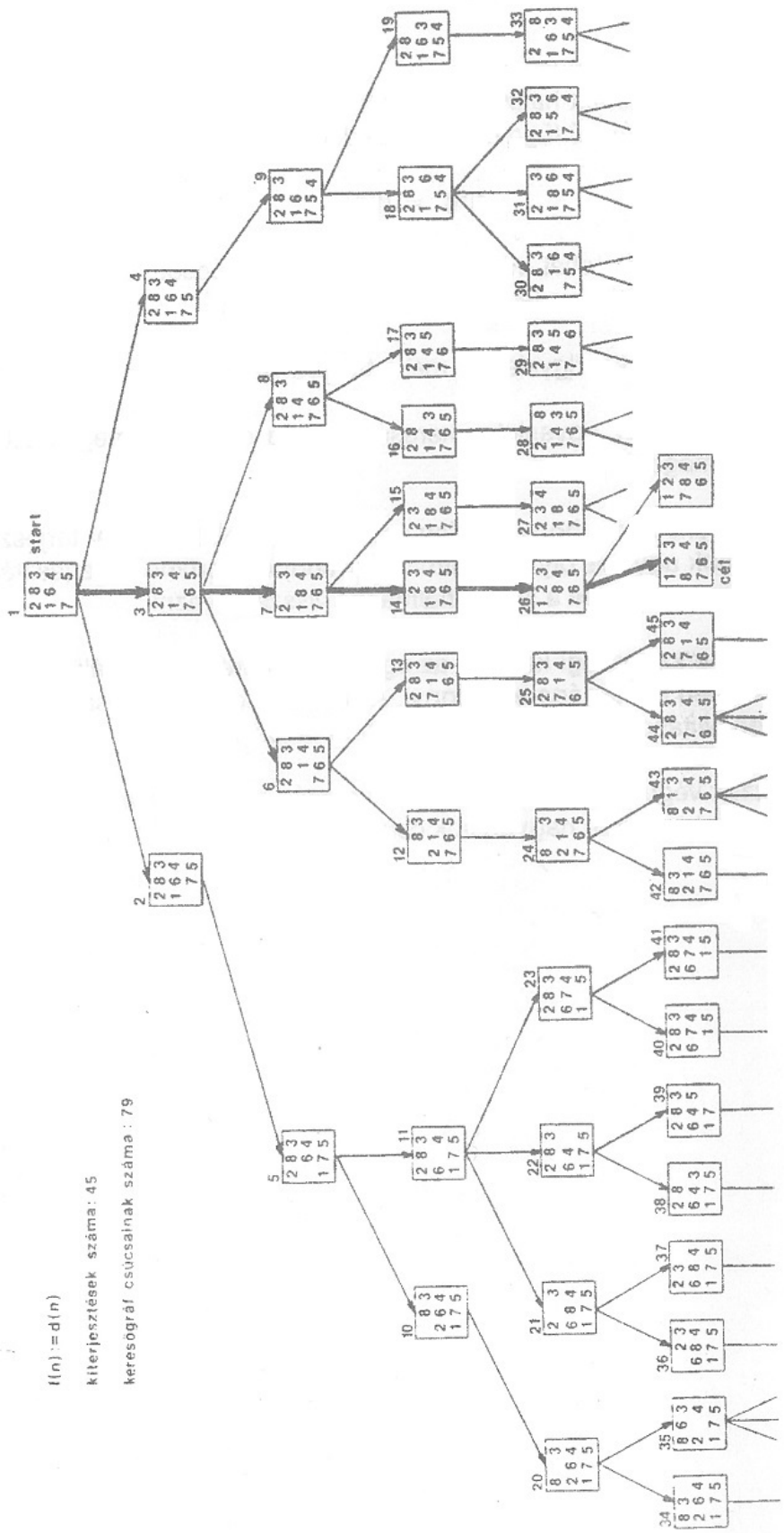
képlettel definiáljuk. Az keresés mindig olyan nyílt csúcsot választ kiterjesztésre, amelyhez a keresőgráfban a legrövidebb út vezet.

Az egyenletes keresés két kedvező tulajdonsággal is rendelkezik. Az egyik az, hogy ha létezik megoldása a feladatnak, akkor mindig megtalál egyet, méghozzá optimális megoldással terminál. A másik nevezetes tulajdonsága az, hogy minden csúcsot legfeljebb egyszer terjeszt ki, azaz nem fordulhat elő az, hogy egy zárt csúcsot vissza kell helyezni a nyílt halmazba. Ezen tulajdonságok bizonyítását az általános esetre az 5. fejezetben adom meg.

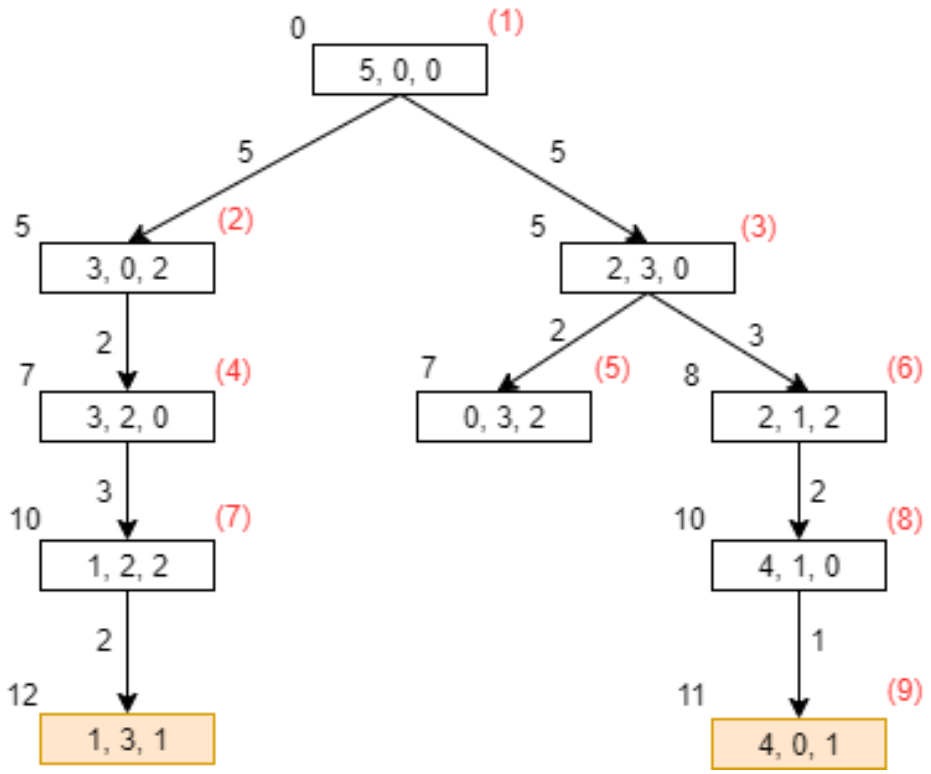
A 4.5. ábrán példát láthatunk az egyenletes keresés működésére. A kancsók feladatában egy áttöltési művelethez a felemelt kancsó súlyát rendeljük költségként. A feltüntetett költségértékek és a kiterjesztések sorszámai alapján világos, hogy a reprezentációs gráf 10 csúcsa közül 9 került kiválasztásra, utolsó lépésben a kisebb $g(n) = 11$ költséggel elérhető célcsúcs (a másik célcsúcs 12 költséggel érhető el).



4.3. ábra. A 8-as játék megoldása mélységi kereséssel



4.4. ábra. A 8-as játék megoldása szélességi kereséssel



4.5. ábra. A kancsók problémájának megoldása egyenletes kereséssel

5. fejezet

Heurisztikus gráfkeresések

Az előző fejezetben megismertük a gráfkeresés általános algoritmusát majd az ebből származtatható néhány neminformált keresést.

Az általános sémában a gráf minden n csúcsánál nyilvántartjuk azt a legkisebb $g(n)$ költséget, amellyel a keresés aktuális állapotában az s kezdőcsúcsból eljuthatunk n-hez. Ezen kívül, még egy pointer is mutat a megelőző csúcsra, amely egy optimális úton helyezkedik el. A gráf csúcsainak kétféle státusza lehet: egy csúcs nyílt, ha még kiválasztásra kerülhet és zárt, akkor, ha egyelőre nem számolunk a kiterjesztésével.

Az általános gráfkereső algoritmus minden lépésben kiválaszt egy nyílt csúcsot kiterjesztésre egy – az általánosság szintjén – közelebbről nem definiált f kiértékelő függvény minimális értéke szerint. A kiválasztott csúcs minden m szomszédjára ellenőrzi, hogy sikerült-e kisebb költségű utat találni hozzá. Ha igen, akkor az m csúcshoz bejegyzésre kerül a $g(n) + c(n, m)$ költségérték és m-ből az n-be állítjuk a $p(m)$ pointert.

Ha a kisebb költséggel elérhető m szomszédos csúcs már zárt volt, akkor újra nyíltra állítjuk. Ezzel megteremtjük annak a lehetőségét, hogy a későbbi kiválasztás során a kisebb költségű út megtalálása „tovább gyűrűzzön” a leszármazott csúcsokra.

A neminformált gráfkereső eljárások úgy származtathatók az általános sémából, hogy az f kiértékelő függvény definíciójában csak a $g(n)$ költségfüggvényt használjuk fel valamilyen formában. Meg kell jegyezni, hogy a költségfüggvény speciális esete az, amikor minden él költsége egységnyi (erre mondják, hogy „nincs költség”) és ekkor $g(n)$ a csúcs mélységét jelent egy adott elérési útvonalon, amelyet $d(n)$-nel jelölünk, ha egy legrövidebb út szerint mélységről van szó.

A szélességi keresésnél mindig egy legkisebb mélységű csúcsot választunk kiterjesztésre, vagyis $f(n) = d(n)$. A mélységi keresésnél minden lépésben egy legmélyebben fekvő csúcsból próbálunk tovább haladni, aminek megfelel az $f(n) = -d(n)$

választás. A szélességi keresés általános változata az egyenletes keresés, amelyben az élek költsége bármely nemnegatív (gyakorlatban pozitív) érték lehet. A kiválasztás ekkor az $f(n) = g(n)$ függvény minimuma szerint történik.

A korábban bevezetett és tárgyalt visszalépéses keresést ebből a nézőpontból nyilvánvalóan a mélységi kereséssel lehet közvetlen kapcsolatba hozni.

A neminformált gráfkeresések esetében is mód van arra, hogy valamilyen megfontolás beépítésével befolyásoljuk a keresés hatékonyságát. Erre ebben az esetben csupán a csúcsok kiválasztási sorrendjének befolyásolása teremt lehetőséget, azonban a visszalépéses, illetve a mélységi keresésnél ennek is jelentős hatása lehet (lásd: 3.2. ábra példája).

Ebben a fejezetben olyan gráfkereső algoritmusokat mutatok be, melyekben magába az f kiértékelő függvénybe építünk be valamilyen speciális tudást. Ennek a feladatról szóló ismeretnek kell egzaktnak lenni, elég, ha a gyakorlatban (az esetek nagyobb részében) beválik és javítja az algoritmus hatékonyságát, a megoldást kisebb költséggel sikerül megtalálni.

Az ilyen fajta ismereteket heurisztikának nevezzük. Azokat a gráfkereső algoritmusokat pedig, amelyek f kiértékelő függvényébe ilyen, a hatékonyságot növelő tudáselemek kerülnek beépítésre, heurisztikus gráfkereséseknek nevezzük.

A heurisztikus gráfkereső algoritmusok elég általános és elméletileg jó tárgyalható osztályához jutunk egy ésszerű feltételezéssel. Tekintsük úgy, hogy a heurisztikus ismeretünk a gráf minden csúcsából (az egyes állapotokból) becslést ad arra, hogy mekkora az a legkisebb költség, amellyel onnan célcsúcsba (célállapotba) lehet jutni.

Formálisan, vezessük be a $h^*(n)$ függvényt, amely megadja azt a minimális költséget, amellyel az n csúcsból bármely $t \in T$ célcsúcsba el lehet jutni:

$$h^*(n) := \min_{t \in T} k^*(n, t)$$

ahol $k^*(n)$ jelöli azt a legkisebb költségét, amellyel n-ből a t célcsúcsba juthatunk. Természetesen előfordulhat, hogy több olyan csúcs is létezik a gráfban, melyekre $h^*(n)$ megegyezik. Illetve ha nem vezet az adott csúcsból út egy célcsúcsba sem, akkor nem definiáljuk a függvényt.

A $h(n)$ heurisztikus függvény a $h^*(n)$ költségfüggvény becslése, amely azért is hasznos, mert egy kiterjesztésre váró n csúcsnál nem ismerjük a $h^*(n)$ értéket, ezért csak becsülni tudjuk azt.

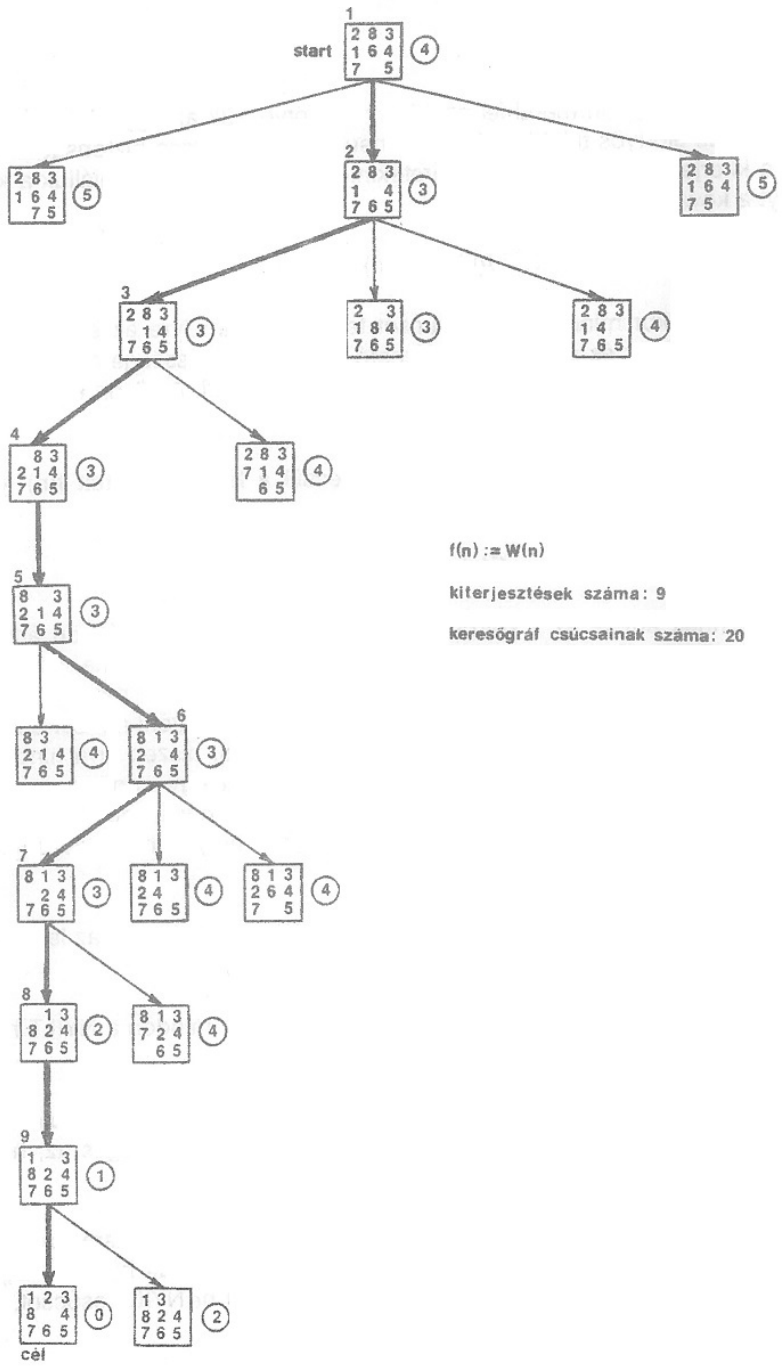
Az $f(n)$ kiértékelő függvény alakjára, illetve a $h(n)$ heurisztikus becslés tulajdonságára kimondott feltevésekkel különböző, jellegzetes tulajdonságú algoritmusosztályokhoz jutunk.

5.1. Előretekintő keresés

Az eddigi algoritmusokkal szemben az előretekintő keresés a csúcsok "előéletével" nem foglalkozik, a kiértékelő függvény megegyezik a heurisztikus függvénnyel:

$$f(n) := h(n)$$

Az algoritmus olyan csúcsot választ kiterjesztésre, melyre a heurisztikus függvény értéke a legkisebb. Egy ilyen keresést láthatunk az 5.1. ábrán.



5.1. ábra. A 8-as játék megoldása előretekintő kereséssel

Ez a módszer nem feltétlenül talál megoldást, az eredmény nagyban függ $h(n)$ megválasztásától, hiszen a csúcsok elérésének tényleges költségei figyelmen kívül maradnak.

A következőkben szeretnénk ötvözni az egyenletes keresés „óvatos” biztonságát és az előretekintő keresés „kalandorjellegű” célratörését. A keresés biztonságát megtartjuk, de a heurisztika segítségével csökkentjük a kiterjesztések számát.

5.2. Az A algoritmus

Egy gráfkeresést A algoritmusnak nevezzük, ha a kiértékelő függvénye a következő:

$$f(n) = g(n) + h(n)$$

$$h(n) \geq 0$$

Ekkor mindig olyan csúcsot választ kiterjesztésre az algoritmus, amelyre az eddig talált legolcsóbb út költségének és a célhalmazba jutás várható költségének az összege a legkisebb.

Belátjuk, hogy az A algoritmus mindig talál megoldást, ha az létezik. Ehhez szükségünk van a következő lemmára.

5.1. Lemma. Ha az A algoritmus nem terminál, akkor minden nyílt csúcs véges sok lépés után kiterjesztésre kerül.

Bizonyítás. Az m nyílt csúcsot biztosan kiterjesztjük, ha már nem létezik olyan $n \in$ *nyílt* csúcs, amelyre $f(n) \leq f(m)$. Be kell látnunk, hogy csak véges sok esetben fordulhat elő, hogy az eljárás olyan csúcsot talál, amelyre $n \in$ *nyílt* és $f(n) \leq f(m)$. Az $f(n)$ értéke az n csúcs mélységével arányos korlát felett van, mivel igaz, hogy

$$f(n) = g(n) + h(n) \geq g^*(n) \geq d^*(n)\delta$$

ahol $d^*(n)$ legyen a n csúcsnak a reprezentációs gráfban felvett optimális feszítőfa szerinti mélysége, δ pedig alsó korlát az élek költségére.

Rögzítsük az $f(m)$ költséget. Ehhez meg tudunk adni egy olyan mélységét a gráfnak, melyre $f(n) \leq f(m)$. Válasszuk d-t $f(m)/\delta$ felső egészrészének, ekkor a d szintnél mélyebben fekvő r csúcsokra igaz, hogy $f(r) > f(m)$.

Tudjuk, hogy a d mélységű szint feletti csúcsok száma véges, és ezen szint felett mindegyikhez véges sok körmentes út vezethet s-ből. Egy csúcsot legfeljebb annyiszor terjeszthet ki az algoritmus, ahány körmentes út vezet hozzá a kezdőcsúcsból. Ezért véges sok esetben választhatunk olyan $n \in$ *nyílt* csúcsot, melyre $f(n) \leq f(m)$ teljesül. Ezzel beláttuk, hogy az $m \in$ *nyílt* csúcs véges sok lépés után kiterjesztésre kerül. □

5.2. Tétel. Az A algoritmus mindig talál megoldási utat, feltéve, hogy létezik ilyen.

Bizonyítás. Tudjuk, hogy ha a véges reprezentációs gráfban létezik célcsúcs, akkor az általános gráfkereső algoritmus megtalál egyet közülük (lásd: 3.1. Tétel). Tehát véges reprezentációs gráfokra igaz a fenti állítás, így már csak olyan végtelen reprezentációs gráfokkal kell foglalkoznunk, amelyekben vezet út s-ből valamely $t \in T$ csúcsba.

Vegyünk egy $s = n_0, n_1 \dots n_k = t$ optimális utat (s legyen a startcsúcs, t pedig egy célcsúcs.) Tudjuk, hogy mindig van ezen az úton nyílt csúcs, vagyis létezik $n_i \in$ *nyílt* $(0 \leq i \leq k)$ (a 3.1. Tétel szerint). Már beláttuk, hogy ha algoritmus nem terminál, akkor minden *nyílt* halmazba került csúcs véges sok lépés után kiterjesztésre kerül, ezért n_i $(0 \leq i \leq k-1)$ is kiterjesztésre kerülne. Miután n_i-t kiterjesztettük, n_{i+1} bekerül a *nyílt* halmazba. Ezt így folytatva véges sok lépés alatt elérünk t célcsúcsba. Ez csak akkor nem valósul meg, ha egy másik célcsúcs megtalálásával már korábban terminált az algoritmus. □

A most bizonyított tétel mindössze egy megoldási út megtalálását állítja, de azt nem mondja ki, hogy ez az útvonal optimális lenne.

5.3. Az A* algoritmus

Az olyan A algoritmust, melynek a heurisztikus függvénye minden csúcsra alulról becsüli a hátralevő optimális költséget, A^* algoritmusnak nevezzük (vagyis amelyre $h(n) \leq h^*(n)$).

5.3. Lemma. Az A^* algoritmus által kiterjesztésre kiválasztott bármely n csúcsra igaz a következő egyenlőtlenség:

$$f(n) \leq f^*(s)$$

Bizonyítás. Feltesszük, hogy a reprezentációs gráfban létezik megoldási út, és ekkor $f^*(s) = g^*(s) + h^*(s)$ jelenti egy optimális út költségét. Ha nem érhető el egyetlen célcsúcs sem n-ből akkor $f^*(s)$-t végtelennek tekintjük.

A 3.1-es tételben láttuk, hogy van a *nyílt* halmazban egy olyan m csúcs, mely raja van az s-ből t-be vezető úton és az elődei már zártak (tehát $g(m) = g^*(m)$). Az algoritmus m helyett az n csúcsot választotta kiterjesztésre, ezért

$$f(n) \leq f(m) = g^*(m) + h(m) \leq g^*(m) + h^*(m) = f^*(m) = f^*(s)$$

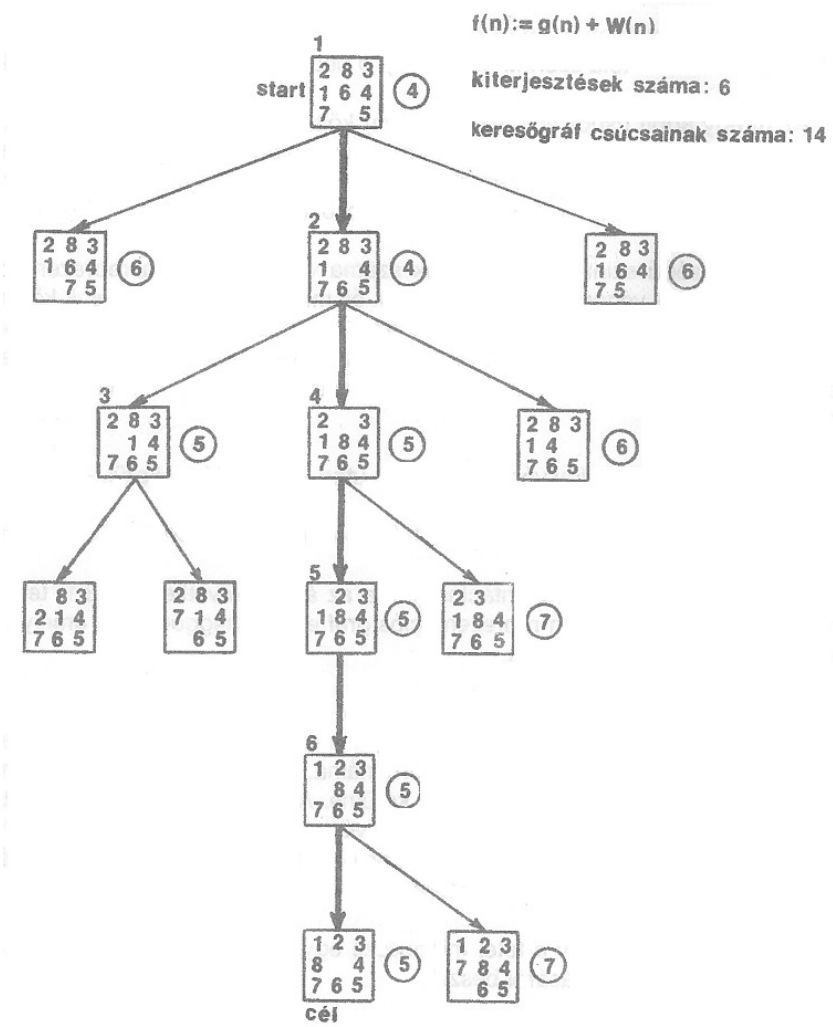
$f^*(m) = f^*(s)$ egyenlőség azért áll fenn, mert f^* értéke egy optimális út bármely csúcsán egyenlő az optimális megoldási költséggel. □

5.4. Tétel. Az A^* algoritmus mindig megtalálja az optimális megoldást, feltéve, hogy létezik megoldás.

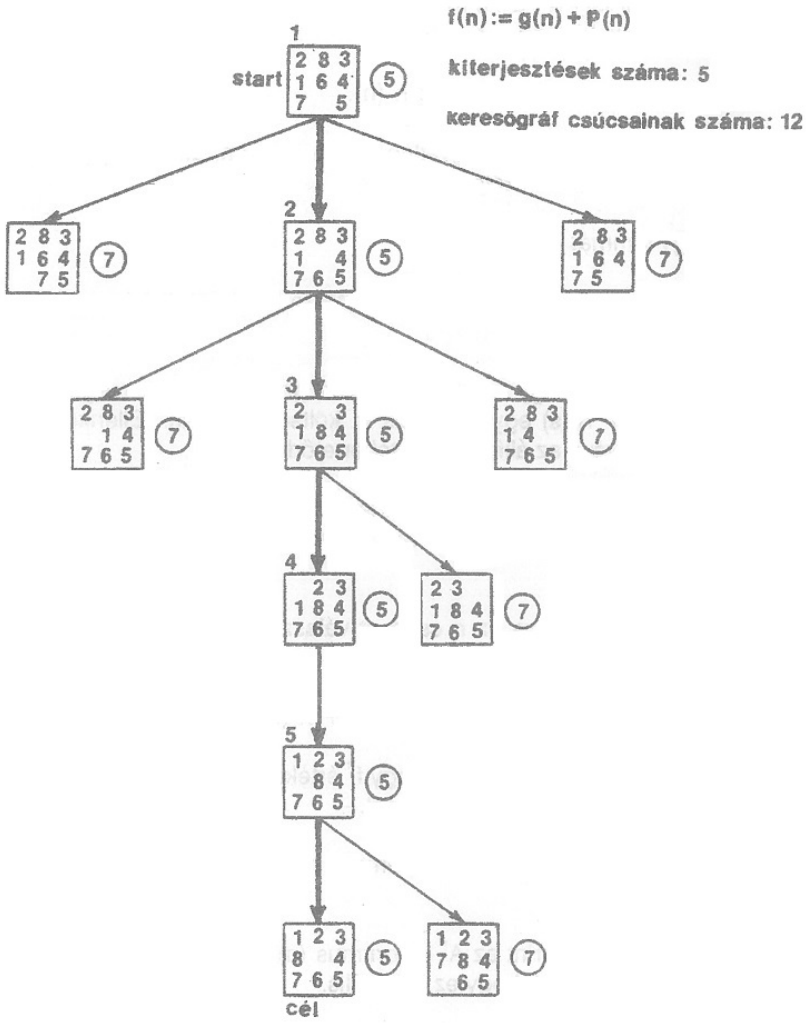
Bizonyítás. Mivel az A^* egy speciális A algoritmus, ezért biztosan talál megoldást. Azt kell csak belátnunk, hogy ez a megoldás optimális is. Tegyük fel, hogy az algoritmus segítségével egy olyan úton jutottunk el a $t \in T$ célcsúcsba, mely nem minimális költségű. Ekkor $f(t) > f^*(s)$, de ez ellentmond az 5.2.Lemmának, amely szerint $f(n) \leq f^*(s)$. □

Azt az eljárást, mely az optimális megoldást garantálja, megengedhetőnek is szokták nevezni.

Az alábbi 5.2. és 5.3. ábrákon a 8-as játék megoldása látható kétféle heurisztikával, amelyek eleget tesznek az A^* algoritmus követelményének.



5.2. ábra. A 8-as játék megoldása A (és A*) algoritmussal a W heurisztika mellett



5.3. ábra. A 8-as játék megoldása A (és A*) algoritmussal a P heurisztika mellett

Az 5.2. ábrán látható keresés a $W(n)$ heurisztikát alkalmazza, amely az n állapotban a nem saját helyükön található lapocskák számát adja meg. Ennél jobbnak mondható $P(n)$ heurisztika, amelyet az 5.3. ábrán bemutatott keresés használ, ez a lapocskák távolság-összegét adja meg, amely a célállapotbeli helyüktől fennáll.

Nyilvánvalóan teljesül a $W(n) \leq P(n) \leq g^*(n)$ egyenlőtlenség, ahol $g^*(n)$ a célállapot kirakásához szükséges lépések száma. Általánosan is belátható az, amit a két ábra mutat: ilyen esetben a jobb heurisztika legfeljebb annyi kiterjesztést hajt végre, mint a kevésbé jó másik.

5.4. A monoton megszorításos algoritmus

Egy A algoritmust monoton megszorításos algoritmusnak nevezünk, ha a heurisztikus függvénye bármely él mentén legfeljebb az él költségével csökken.

Megköveteljük tehát azt, hogy

$$h(n) - h(m) \leq c(m, n)$$

$$h(t) = 0 \ (t \in T)$$

legyen igaz, minden (m, n) élre, illetve $t \in T$ célcsúcsra a reprezentációs gráfban.

Belátjuk az algoritmus nevezetes tulajdonságát, amely szerint minden csúcsot legfeljebb egyszer terjeszt ki, mivel ha egy $n \in$ *nyílt* csúcsot kiterjesztésre kiválaszt az eljárás, akkor már optimális utat talált n-be. Ehhez előbb egy lemmát mondunk ki és bizonyítunk.

5.5. Lemma. Ha a h heurisztikára teljesül a monoton megszorítás, akkor egy tetszőleges n csúcsba vezető optimális út mentén a $g^* + h$ összeg értéke nem csökken.

Bizonyítás. Legyen $s = n_0, n_1 \ldots n_k = n$ egy optimális út. Ekkor a monoton megszorítás miatt bármely i-re $(0 < i < k)$:

$$g^*(n_i) + h(n_i) \leq g^*(n_i) + c(n_i, n_{i+1}) + h(n_{i+1}) = g^*(n_{i+1}) + h(n_{i+1})$$

5.6. Tétel. Ha a monoton megszorításos algoritmus egy n nyílt csúcsot kiterjesztésre kiválaszt, akkor n-be már optimális utat talált, azaz $g(n) = g^*(n)$.

Bizonyítás. Tegyük fel indirekt, hogy az $n \in$ *nyílt* csúcsot úgy terjesztünk ki, hogy $g(n) > g^*(n)$. Tudjuk, hogy van olyan $m \in$ *nyílt* csúcs, mely rajta van egy (s, m) optimális úton és $g(m) = g^*(m)$ (lásd: 3.1. Tétel). Az m csúcs az n csúcsot megelőzi, ezért az előző lemmát figyelembe véve:

$$f(m) = g^*(m) + h(m) \leq g^*(n) + h(n) < g(n) + h(n) = f(n)$$

Tehát $f(m) < f(n)$ teljesül, viszont ez ellentmond annak, hogy n csúcs kiválasztása miatt $f(n) \leq f(m)$. □

A monoton megszorításos algoritmus, ugyanúgy mint az A^* algoritmus, mindig optimális megoldást talál. Az analógia kedvéért itt egy lemma áll elől.

5.7. Lemma. A monoton megszorításos algoritmus által bármely, kiterjesztésre kiválasztott n csúcsra $f(n) \leq f^*(s)$.

Bizonyítás. Hasonlóan, mint az A^* algoritmusnál itt is feltesszük, hogy a reprezentációs gráfban létezik megoldási út. Ha nem érthető el egyetlen célcsúcs sem n-ből akkor $f^*(s)$-et végtelennek tekintjük. A 3.1. Tétel szerint a *nyílt* halmazban van egy olyan m csúcs, mely rajta van az s-ből t-be vezető úton és az elődei már zártak (tehát $g(m) = g^*(m)$).

A vezérlés a feltevés alapján n csúcsot választja kiterjesztésre, ezért:

$$f(n) \leq f(m) = g^*(m) + h(m)$$

Az m csúcs rajta van az optimális s-ből t-be vezető optimális úton, ezért az előző 5.5. lemma alapján:

$$g^*(m) + h(m) \leq g^*(t) + h(t)$$

Mivel t célcsúcs, ezért $h(t) = 0$, illetve tudjuk, hogy $g^*(t) = f^*(s)$, ezért teljesül:

$$g^*(t) + h^*(t) = f^*(s)$$

Az eddigieket összevetve megkapjuk az

$$f(n) \leq f^*(s)$$

állítást, amely bizonyítandó volt. □

5.8. Tétel. A monoton megszorításos algoritmus optimális megoldás megtalálásával terminál, feltéve, hogy létezik megoldás.

Bizonyítás. Megegyezik az 5.4. tétel bizonyításával, azzal a különbséggel, hogy itt az 5.7. lemmára hivatkozunk. □

Megmutatjuk, hogy minden monoton megszorításos gráfkeresés egyben A algoritmus is.

5.9. Tétel. Ha egy h heurisztikus függvény kielégíti a monoton megszorítás feltételét, akkor egyben alsó becslése a h^* költségfüggvénynek minden n csúcsra.

Bizonyítás. Ha az n csúcsból nem vezet út a célcsúcsba, akkor h^* értékét végtelennek tekintjük, így igaz lesz az állítás. Ellenkező esetben vegyünk egy $n = n_0, n_1 \ldots n_{k-1}, n_k = t$ utat, és minden éllére írjuk fel a monoton megszorítást.

$$h(n) - h(n_1) \leq c(n, n_1)$$

$$h(n_1) - h(n_2) \leq c(n_1, n_2)$$

$$\ldots$$

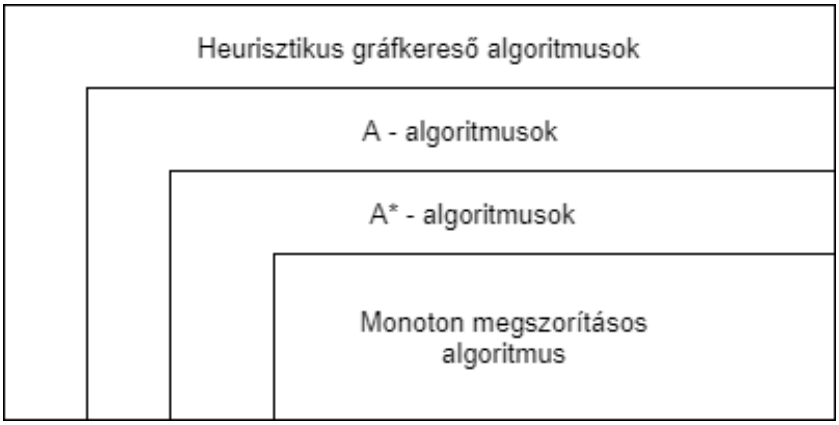
$$h(n_{k-1}) - h(t) \leq c(n_{k-1}, t)$$

Ha összeadjuk ezeket az egyenlőtlenségeket, akkor az kapjuk, hogy:

$$h(n) - h(t) \leq \sum_{i=0}^{k-1} c(n_i, n_{i+1})$$

ami $h(t) = 0$ miatt ugyanaz, mint $h(n) \leq h^*(n)$ □

Az A^* és a monoton algoritmusok osztálya között valódi tartalmazás áll fenn, ugyanis könnyű olyan gráfot és élköltségeket mutatni, amely ezt bizonyítja.



5.4. ábra. A heurisztikus gráfkereső algoritmusok osztályai

A fejezet eredményeinek összefoglalásaként az 5.4. ábrán megadom a heurisztikus gráfkereső algoritmusok egymáshoz való viszonyát.

Ábrák jegyzéke

2.1. A négy királynő probléma állapotterének három állása 7
2.2. A négy királynő probléma reprezentációs gráfja 9
2.3. A 8-as játék három állapota 10
2.4. A 3-as játék reprezentációs gráfja 11
2.5. A 8-as játék reprezentációs fájának részlete 12
2.6. A Hanoi tornyai probléma két állapota 13
2.7. A Hanoi tornyai probléma egy állapota (4 korong) 13
2.8. A Hanoi tornya probléma reprezentációs gráfja 14
2.9. A kancsók problémájának két állapota 15
2.10. A kancsók problémájának reprezentációs gráfja 16
2.11. A misszionárius-kannibál probléma reprezentációs gráfja 17

3.1. A négy királynő probléma megoldása visszalépéses algoritmussal . . . 24
3.2. A 8-as játék megoldása visszalépéses algoritmussal 26

4.1. Nyílt csúcs pointerének beállítása 31
4.2. Zárt csúcs és utódja pointerének a beállítása 32
4.3. A 8-as játék megoldása mélységi kereséssel 37
4.4. A 8-as játék megoldása szélességi kereséssel 38
4.5. A kancsók problémájának megoldása egyenletes kereséssel 39

5.1. A 8-as játék megoldása előretekintő kereséssel 42
5.2. A 8-as játék megoldása A (és A*) algoritmussal a W heurisztika mellett 45
5.3. A 8-as játék megoldása A (és A*) algoritmussal a P heurisztika mellett 46
5.4. A heurisztikus gráfkereső algoritmusok osztályai 49

Felhasznált irodalom

[1] Fekete I. - Gregorics T. - Nagy S.:
Bevezetés a mesterséges intelligenciába.
ELTE Eötvös Kiadó, 2006.

[2] Nilsson, N. J.:
Artificial Intelligence.
Springer, 1982.

[3] Russell, S. - Norvig, P.:
Mesterséges Intelligencia modern megközelítésben.
Panem Kiadó, 2005.

[4] Zhang, W.:
State-Space Search.
Springer, 1999.

[5] Poole, D. L. - Mackworth A. K.:
Artificial Intelligence, Foundations of Computational Agents.
Cambridge University Press, 2010.

* * *

[6] Fekete I.- Hunyadvári L.:
Algoritmusok és adatszerkezetek.
Digitális egyetemi tankönyv:
`https://www.tankonyvtar.hu/hu/tartalom/tamop412A/2011-0052_28_algoritmusok_es_adatszerkezetek/adatok.html`