VÁRTERÉSZ MAGDA

Mesterséges intelligencia 1
előadások

2006/07-es tanév

Tartalomjegyzék

1. fejezet

A problémareprezentáció

A mesterséges intelligencia problémáinak megoldása a probléma megfogalmazásával kezdődik: a problémát leírjuk, **reprezentáljuk**.
Az egyik legelterjedtebb reprezentációs technika az

állapottér-reprezentáció

(***state space representation***).

1.1. Az állapottér-reprezentáció

Legyen adott egy probléma, amit jelöljünk p-vel.

- Megkeressük p világának legalább egy, de véges sok – a probléma megoldása során fontosnak vélt – meghatározóját.

  (pl. objektum, pozíció, méret, hőmérséklet, szín, stb)

  Tegyük fel, hogy m ilyen jellemzőt találtunk.

- Minden egyes jellemző p világát különböző értékekkel jellemzi.

  (pl. szín: fekete/fehér; hőmérséklet: $[-20°, 40°]$, stb)

Ha a megadott jellemzők épp rendre a $h_1, \ldots, h_m$ értékekkel rendelkeznek azt mondjuk, hogy p világa a $(h_1, \ldots, h_m)$ érték m-essel leírt **állapotban** (***state***) van.
A világunk állapotainak halmaza az **állapottér** (***state space***).

Jelölje az i-edik jellemző által felvehető értékek halmazát H_i ($i = 1, \ldots, m$). Ekkor p állapotai elemei a

$$H_1 \times \cdots \times H_m$$

halmaznak.
Azokat a feltételeket, amelyek meghatározzák, hogy ebből a halmazból mely érték m-esek állapotok, **kényszerfeltételeknek** nevezzük. Az állapottér tehát az értékhalmazok Descartes-szorzatának a kényszerfeltételekkel kijelölt részhalmaza:

$$\mathcal{A} = \{\, a \mid a \in H_1 \times \cdots \times H_m \text{ és } \textit{kényszerfeltétel}(a) \,\}$$

Az $\mathcal{A}$ állapottér azon állapotát, amit a probléma világa jellemzőinek kezdőértékei határoznak meg, **kezdőállapotnak** (***initial state***) nevezzük és *kezdő*-vel jelöljük.
A kezdőállapotból kiindulva a probléma világának sorban előálló állapotait rendre meg szeretnénk változtatni, míg végül valamely számunkra megfelelő ún. **célállapotba** (***goal state***) jutunk.
Jelölje $\mathcal{C} \subseteq \mathcal{A}$ a **célállapotok halmazát**. Megadása kétféleképpen történhet:

- felsorolással: $\mathcal{C} = \{\, c_1, \ldots, c_l \mid c_i \in \mathcal{A},\ i = 1, \ldots, \ell,\ \ell \geq 1 \,\}$
- **célfeltételek** megadásával: $\mathcal{C} = \{\, c \mid c \in \mathcal{A} \text{ és } \mathit{célfeltételek}(c) \,\}$

Általában $\mathcal{C} \subset \mathcal{A}$, hiszen $\mathit{kezdő} \notin \mathcal{C}$, különben nincs megoldandó feladat.

Hogy célállapotba juthassunk, meg kell tudnunk változtatni bizonyos állapotokat. Az állapotváltozásokat leíró leképezéseket **operátoroknak** (***operator***) nevezzük.
Nem minden operátor alkalmazható feltétlenül minden állapotra, ezért meg szoktuk adni az operátorok értelmezési tartományát az **operátoralkalmazási előfeltételek** segítségével.
Jelöljön az **operátorok** $\mathcal{O}$ véges **halmazából** o egy operátort. Ekkor

$$\mathrm{Dom}(o) = \{\, a \mid a \in \mathcal{A} \text{ és } o\text{-}\textit{alkalmazásának-előfeltétele}(a) \,\}$$

és

$$\mathrm{Rng}(o) = \{\, o(a) \mid a \in \mathrm{Dom}(o) \text{ és } o(a) \in \mathcal{A}\}.$$

1.1. DEFINÍCIÓ. Legyen p egy probléma. Azt mondjuk, hogy a p problémát **állapottér-reprezentáltuk**, ha megadtuk az

$$\langle \mathcal{A}, \mathit{kezdő}, \mathcal{C}, \mathcal{O} \rangle$$

négyest, azaz

- az $\mathcal{A} \neq \emptyset$ halmazt, a probléma állapotterét,
- a $\mathit{kezdő} \in \mathcal{A}$ kezdőállapotot,
- a célállapotok $\mathcal{C} \subset \mathcal{A}$ halmazát és
- az operátorok $\mathcal{O} \neq \emptyset$ véges halmazát.

Jelölése: $p = \langle \mathcal{A}, \mathit{kezdő}, \mathcal{C}, \mathcal{O} \rangle$.

Legyen p egy probléma, $\langle \mathcal{A}, \mathit{kezdő}, \mathcal{C}, \mathcal{O} \rangle$ p egy állapottér-reprezentációja és legyenek $a, a' \in \mathcal{A}$.

1.2. DEFINÍCIÓ. Az a állapotból az a' állapot **közvetlenül elérhető**, ha van olyan $o \in \mathcal{O}$ operátor, hogy

$$o\text{-}\mathit{alkalmazásának\text{-}előfeltétele}(a) \quad \text{és} \quad o(a) = a'.$$

Jelölése: $a \Rightarrow a'$,
ha fontos, hogy o állítja elő a-ból a'-t: $a \underset{o}{\Rightarrow} a'$.

1.3. DEFINÍCIÓ. Az a állapotból az a' állapot **elérhető**, ha vagy $a = a'$, vagy van olyan $a_1, \ldots, a_k \in \mathcal{A}$ $(k \geq 2)$ (véges) állapotsorozat, hogy

$$a = a_1, \quad a' = a_k \quad \text{és} \quad a_i \Rightarrow a_{i+1} \text{ minden } 1 \leq i \leq k-1 \text{ esetén.}$$

Jelölése: $a \overset{*}{\Rightarrow} a'$.

1.4. MEGJEGYZÉS. Ha $a_i \Rightarrow a_{i+1}$ minden $1 \leq i \leq k-1$ esetén, akkor van olyan $o_1, o_2, \ldots, o_{k-1}$ operátorsorozat, hogy $a_i \underset{o_i}{\Rightarrow} a_{i+1}$ $(1 \leq i \leq k-1)$. Ilyenkor azt mondjuk, hogy az a állapotból az a' állapot az $o_1, o_2, \ldots, o_{k-1}$ operátorsorozat segítségével elérhető. Jelölve: $a \underset{o_1,\ldots,o_{k-1}}{\overset{*}{\Rightarrow}} a'$.

1.5. DEFINÍCIÓ. Legyen $p = \langle \mathcal{A}, \mathit{kezdő}, \mathcal{C}, \mathcal{O} \rangle$. A p probléma **megoldható** ebben az állapottér-reprezentációban, ha van olyan $c \in \mathcal{C}$ célállapot, hogy $\mathit{kezdő} \overset{*}{\Rightarrow} c$. Ha $\mathit{kezdő} \underset{o_1,\ldots,o_r}{\overset{*}{\Rightarrow}} c$ $(r \geq 1)$, akkor az $o_1, \ldots, o_r$ operátorsorozat a probléma egy **megoldása**.

A feladatunk lehet

- annak eldöntése, hogy megoldható-e a probléma az adott állapottér-reprezentációban,
- egy (esetleg az összes) megoldás előállítása,
- egy (esetleg az összes) célállapot előállítása,
- valamilyen minősítés alapján jó megoldás előállítása (a megoldások között különbséget tehetünk, pl. a megoldás költsége alapján).

Jelölje *költség*(o, a) az o **operátor** a **állapotra alkalmazásának a költségét**.

1.6. DEFINÍCIÓ. Legyen $p = \langle \mathcal{A}, \mathit{kezdő}, \mathcal{C}, \mathcal{O} \rangle$ és $o_1, \ldots, o_r \in \mathcal{O}$ a probléma egy megoldása, azaz

$$\mathit{kezdő} \underset{o_1,\ldots,o_r}{\overset{*}{\Rightarrow}} c \quad (c \in \mathcal{C}).$$

Legyen rendre $a_i \underset{o_i}{\Rightarrow} a_{i+1} \quad (1 \leq i \leq r)$, ahol $a_1 = \mathit{kezdő}$ és $a_{r+1} = c$. Ekkor a **megoldás költsége**:

$$\sum_{i=1}^{r} \mathit{költség}(o_i, a_i).$$

Ha *költség*$(o, a) \equiv 1$, akkor a megoldás költsége az alkalmazott operátorok száma.

Ha az állapottér-reprezentációt valamilyen programozási nyelv segítségével szeretnénk leírni, akkor meg kell adni

- az állapotok típusát, továbbá a kényszerfeltételeket leíró, logikai értéket visszaadó függvényt;
- a kezdőállapotot egy állapot típusú konstanssal;
- a célállapotok halmazát:
  - állapot típusú konstansok felsorolásával vagy
  - a célfeltételt leíró logikai értéket visszaadó függvénnyel;
- az operátorokat: függvényekkel vagy eljárásokkal. Egy-egy operátorhoz az alkalmazásának előfeltételét leíró logikai függvény is tartozhat. Alkalmas függvénnyel ki lehet számolni (vagy táblázattal meg lehet adni) az operátorok alkalmazási költségeit is.

1.2. Példák állapottér-reprezentációra

1.2.1. A nyolcas kirakó játék

Adott nyolc számozott, négyzetalakú lapocska, melyek egy táblán 3×3*-as sémában helyezkednek el. Egy lapocskányi hely a táblán így nyilván üres. Az üres hellyel szomszédos bármelyik lapocskát az üres helyre tolhatjuk. A feladat az, hogy adott kezdőállásból kiindulva adott célállásnak megfelelő elrendezést alakítsunk ki.*

A probléma világa: a tábla a számozott lapocskákkal.

A világ leírása: a tábla egyes pozícióin épp mely számozott lapocskák vannak.

Egy-egy pozícióra hivatkozás: (sor,oszlop).

pozíció	$(1,1)$	$(1,2)$	$\ldots$	$(3,3)$
lapocska	$l_{1,1}$	$l_{1,2}$	$\ldots$	$l_{3,3}$

ahol $0 \leq l_{i,j} \leq 8$ minden $1 \leq i, j \leq 3$ esetén.

Tehát $H_{i,j} = H = \{0, 1, \ldots, 8\}$ minden $1 \leq i, j \leq 3$-ra.
A probléma világának egy-egy állapotát egy-egy olyan

$$\begin{pmatrix} l_{1,1} & l_{1,2} & l_{1,3} \\ l_{2,1} & l_{2,2} & l_{2,3} \\ l_{3,1} & l_{3,2} & l_{3,3} \end{pmatrix}$$

érték 9-es ((3×3)-as mátrix) határozza meg, melyben az értékek H-beli elemek, és az érték 9-esben minden egyes érték H-ból pontosan egyszer fordul elő:

$$\forall i \forall j \forall s \forall o (\neg(i = s) \vee \neg(j = o) \supset \neg(l_{i,j} = l_{s,o})).$$

A kezdőállapot és a célállapot:

$$\mathit{kezdő} = \begin{pmatrix} 1 & 2 & 0 \\ 4 & 6 & 3 \\ 7 & 5 & 8 \end{pmatrix} \quad c = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 0 \end{pmatrix}$$

A számozott lapocskák tologatásai során az üres pozíciója mindig változik, az eredetihez képest *fel, le, jobbra* vagy *balra* kerül egy-egy pozícióval. Így tehát négy operátor segítségével leírhatjuk az állapotváltozásokat. A

$$fel : \begin{pmatrix} l_{1,1} & l_{1,2} & l_{1,3} \\ l_{2,1} & l_{2,2} & l_{2,3} \\ l_{3,1} & l_{3,2} & l_{3,3} \end{pmatrix} \mapsto \begin{pmatrix} l'_{1,1} & l'_{1,2} & l'_{1,3} \\ l'_{2,1} & l'_{2,2} & l'_{2,3} \\ l'_{3,1} & l'_{3,2} & l'_{3,3} \end{pmatrix}$$

operátort akkor alkalmazhatjuk, ha

$$\neg\exists i \exists j((l_{i,j} = 0) \wedge (i = 1)).$$

Az alkalmazás eredménye, ha az éppen $l_{s,o} = 0$:

$$l'_{i,j} \rightleftharpoons \begin{cases} 0 & \text{ha } i = s-1, j = o \\ l_{s-1,o} & \text{ha } i = s, j = o \\ l_{i,j} & \text{egyébként.} \end{cases}$$

1.3. Az állapottérgráf

Legyen a p probléma az $\langle \mathcal{A}, \mathit{kezdő}, \mathcal{C}, \mathcal{O} \rangle$ állapottér-reprezentációval megadva. Ez a reprezentáció egy irányított gráfot határoz meg.

- Az $\mathcal{A}$ állapottér elemei (az állapotok) a gráf **csúcsai**. Vezessük be az $a \in \mathcal{A}$ állapot által definiált csúcsra az n_a jelölést. Ekkor a gráf csúcsainak halmaza

$$N = \{n_a \mid a \in \mathcal{A}\}.$$

- A gráf csúcsai közül kitüntetett szerepet játszanak a kezdőállapotot szemléltető ún. **startcsúcs** (jele: $n_{\mathit{kezdő}}$ vagy s)

- és a célállapotokat szemléltető **terminális csúcsok**. A terminális csúcsok halmaza tehát:

$$T = \{n_c \mid c \in \mathcal{C}\}.$$

- Minden $a \in \mathcal{A}$-t szemléltető csúcsból **irányított élt** húzunk az $a' \in \mathcal{A}$ -t szemléltető csúcsba, ha a-ból közvetlenül elérhető a', azaz a gráf irányított éleinek halmaza a következő:

$$E = \{(n_a, n_{a'}) \mid a, a' \in \mathcal{A} \text{ és } a \Rightarrow a'\}.$$

Az

$$\langle N, s, T, E \rangle$$

irányított gráfot a p probléma $\langle \mathcal{A}, \mathit{kezdő}, \mathcal{C}, \mathcal{O} \rangle$ állapottér-reprezentációjához tartozó **állapottérgráfjának** vagy **reprezentációs gráfjának** nevezzük.

1.7. LEMMA. *Legyen* $\langle N, s, T, E\rangle$ *a* p *probléma* $\langle \mathcal{A}, \textit{kezdő}, \mathcal{C}, \mathcal{O}\rangle$ *állapottér-reprezentációjához tartozó állapottérgráfja. Pontosan akkor vezet az állapottérgráf* n_a *csúcsából az ettől különböző* $n_{a'}$ *csúcsba irányított út, ha az* a *állapotból az* a' *állapot elérhető.*

BIZONYÍTÁS.

1. Tegyük fel, hogy $a \overset{*}{\Rightarrow} a'$ $(a \neq a')$. Ez azt jelenti, hogy van olyan $a_1, \ldots, a_k \in \mathcal{A}$ $(k \geq 2)$ (véges) állapotsorozat, hogy

   $$a = a_1, a' = a_k \text{ és } a_i \Rightarrow a_{i+1} \text{ minden } 1 \leq i \leq k-1 \text{ esetén.}$$

   Tehát a reprezentációs gráfban minden $1 \leq i \leq k-1$-re az n_{a_i} csúcsból irányított él vezet az $n_{a_{i+1}}$ csúcsba, azaz

   $$(n_{a_i}, n_{a_{i+1}}) \in E.$$

   Ez viszont azt jelenti, hogy $n_{a_1} = n_a$ csúcsból irányított út vezet az $n_{a_k} = n_{a'}$ csúcsba.

2. Tegyük fel azt, hogy az állapottérgráf n_a csúcsából az ettől különböző $n_{a'}$ csúcsba irányított út vezet. Ez azt jelenti, hogy van olyan $(n_{a_1}, n_{a_2}), (n_{a_2}, n_{a_3}), \ldots, (n_{a_{k-1}}, n_{a_k}) \in E\,(k \geq 2)$ irányított élsorozat az állapottérgráfban, hogy $n_{a_1} = n_a$ és $n_{a_k} = n_{a'}$. Ekkor

   - egyrészt $a_1 = a$ és $a_k = a'$,
   - továbbá $(n_{a_{i-1}}, n_{a_i}) \in E$ csak akkor, ha $a_{i-1} \Rightarrow a_i$.

   Ez azt jelenti, hogy $a \overset{*}{\Rightarrow} a'$, tehát az a állapotból az a' állapot elérhető.

1.8. TÉTEL. *Legyen* $\langle N, s, T, E\rangle$ *a* p *probléma* $\langle \mathcal{A}, kezdő, \mathcal{C}, \mathcal{O}\rangle$ *állapottér-reprezentációjához tartozó állapottérgráfja. Pontosan akkor megoldható* p*, ha van az állapottérgráfban a startcsúcsból valamelyik terminális csúcsba vezető irányított út.*

BIZONYÍTÁS.

1. Egyrészt ha p megoldható, van olyan $c \in \mathcal{C}$ célállapot, hogy $kezdő \stackrel{*}{\Rightarrow} c$. De ekkor az állapottérgráfban irányított út vezet az $n_{kezdő}$ startcsúcsból az $n_c \in T$ terminális csúcsba.

2. Másrészt ha van az állapottérgáfban az $n_{kezdő}$ startcsúcsból valamely terminális csúcsba vezető irányított út, a *kezdő* állapotból elérhető ezen terminális csúcs által szemléltetett állapot. De a terminális csúcsok célállapotokat szemléltetnek. Tehát p megoldható.

A p probléma $\langle \mathcal{A}, \mathit{kezdő}, \mathcal{C}, \mathcal{O} \rangle$ állapottér-reprezentációjában vegyük figyelembe az operátorok alkalmazási költségeit. Rendeljünk ekkor minden $(n_a, n_{a'})$ élhez költséget: ha $a \underset{o}{\Rightarrow} a'$, akkor $\mathit{költség}(o, a)$ legyen ezen **él költsége** (jelölve: $\mathit{költség}(n_a, n_{a'})$). Egy

$$(n_{a_1}, n_{a_2}), (n_{a_2}, n_{a_3}), \ldots, (n_{a_{k-1}}, n_{a_k}) \in E \quad (k \geq 2)$$

irányított út költsége a benne szereplő élek költségösszege

$$\sum_{i=1}^{k-1} \mathit{költség}(n_{a_i}, n_{a_{i+1}}).$$

Ha minden él költsége egységnyi, az irányított út költsége éppen az út éleinek a száma.

Egy állapottér-reprezentált probléma megoldásának sikerét jelentősen befolyásolja a reprezentációs gráf bonyolultsága:

- a csúcsok száma,
- az egy csúcsból kiinduló élek száma,
- a hurkok és körök száma és hossza.

Ezért célszerű minden lehetséges egyszerűsítést végrehajtani. A lehetséges egyszerűsítések:

- csúcsok számának csökkentése – ügyes reprezentációval az állapottér kisebb méretű lehet;
- egy csúcsból kiinduló élek számának csökkentése – az operátorok értelmezési tartományának alkalmas megválasztásával;
- fává alakítás – a reprezentációs gráfban a hurkok, illetve körök „kiegyenesítésével”

2. fejezet

A megoldást kereső rendszerek

Az állapottérgráfban keressük a megoldást: a start csúcsból valamely terminális csúcsba vezető utat.

Az állapottérgráfot implicit módon – az állapottér-reprezentáció megadásával – adjuk meg a megoldást kereső rendszereknek. Ezek a keresés során addig és úgy építik a gráfot, amíg megoldást nem találnak, vagy amíg valamilyen ok miatt kudarcot nem vallanak a kereséssel.

A megoldást kereső rendszerek **felépítése**:

- **Az adatbázis**

  az állapottérgráfnak a keresés során előállított része, amit kiegészíthetünk a hatékony kereséshez szükséges bizonyos információkkal.

- **A műveletek**

  módosítják az adatbázist, azaz az állapottérgráf adatbázisbeli részéből az állapottérgráf egy újabb (további) részét állítják elő. A rendszer alkalmazhat

  – állapottér-reprezentációs operátorokból származtatott műveleteket,

  – „technikai” műveleteket (pl. visszalépést).

  A műveleteknek is vannak végrehajtási feltételeik.

- **A vezérlő**

  irányítja a keresést. Megmondja, hogy a megoldáskeresés folyamán az adatbázisra, annak mely részén, mikor, melyik a végrehajtási feltételeknek eleget tevő művelet hajtódjon végre. Figyeli azt is, hogy befejeződhet-e a keresés, azaz

  – megvan-e a probléma megoldása,
  – vagy kiderült, hogy nem megoldható a probléma.

```
1:  procedure KERESŐ(⟨A, kezdő, C, O⟩)
2:      adatbázis ← INICIALIZÁL(kezdő)
3:      while IGAZ do
4:          if MEGOLDÁS-TALÁL(adatbázis) then
5:              break
6:          end if
7:          if NEM-FOLYTAT(adatbázis) then
8:              break
9:          end if
10:         művelet ← VÁLASZT(adatbázis, műveletek)
11:         adatbázis ← ALKALMAZ(adatbázis, művelet)
12:     end while
13:     if MEGOLDÁS-TALÁL(adatbázis) then
14:         MEGOLDÁS-KIÍR(adatbázis)
15:     else
16:         print „Sikertelen keresés”
17:     end if
18: end procedure
```

A megoldást kereső rendszerek **osztályozhatók**:

- Módosítható-e valahogy egy már alkalmazott művelet hatása?
  - nem: nemmódosítható megoldáskeresők
  - igen: módosítható megoldáskeresők
    - visszalépéses (*backtracking*) keresők
    - keresőfával keresők
- Használunk-e valamiféle speciális tudást a keresés során?
  - nem: irányítatlan (vak, szisztematikus) megoldáskeresők
  - igen: heurisztikus megoldáskeresők
    „A **heurisztika** (heurisztikus szabály, módszer) olyan ökölszabály, stratégia, trükk, egyszerűsítés, vagy egyéb eszköz, amely drasztikusan korlátozza a megoldás keresését nagyméretű reprezentációs gráfokban.”[1]

[1] Feigenbaum és Feldman

- Milyen irányú a keresés?
  - **előrehaladó** (***forward***) vagy adatvezérelt kereső rendszer: a kezdő állapotból kiindulva keresünk célállapotba vezető utat.
  - **visszafelé haladó** (***backward***) vagy célvezérelt kereső rendszer: a célállapotból kiindulva – visszafelé haladva – próbáljuk rekonstruálni a kezdőállapotból odavezető utat.
  - **kétirányú** (***bidirectional***) kereső rendszer: mindkét irányból elindul, s valahol találkozik

A megoldáskereső rendszerek **értékelési szempontjai**:

Teljesség (***completeness***): A rendszer minden olyan esetben megtalálja-e a megoldást, amennyiben az létezik?

Optimalitás (***optimality***): Több megoldás létezése esetén a rendszer az optimális megoldást találja-e meg?

Időigény (***time complexity***): Mennyi ideig tart egy megoldás megtalálása?

Tárigény (***space complexity***): Mekkora tároló területre van szükség a megoldás megtalálásához?

2.1. Nemmódosítható megoldáskereső rendszerek

A MI módszereit nem használó, ún. hagyományos feladatmegoldási módoknál alkalmazzák.

A MI problémák megoldása során nem tudjuk, hogy a reprezentációs gráf megfelelő – a megoldást is tartalmazó – részét építjük-e, ezért ritkán alkalmazunk nemmódosítható keresést az MI területén.

Legyen $p = \langle \mathcal{A}, kezd, \mathcal{C}, \mathcal{O} \rangle$. Egy nemmódosítható megoldáskereső rendszer

- **adatbázisa** az állapottérgráf egyetlen csúcsa, az ún. aktuális csúcs;
- **műveletei** az állapottér-reprezentációs operátorok;
- **vezérlője**:

```
1: procedure NEMMÓDOSÍTHATÓ-KERESŐ(⟨A, kezdő, C, O⟩)
2:     aktuális ← kezdő
3:     while IGAZ do
4:         if aktuális ∈ C then
5:             break
6:         end if
7:         O' ← {o | o ∈ O ∧ ELŐFELTÉTEL(aktuális, o)}
8:         if O' ≠ ∅ then
9:             operátor ← VÁLASZT(O')
10:            aktuális ← ALKALMAZ(aktuális, operátor)
11:        else
12:            break
13:        end if
14:    end while
15:    if aktuális ∈ C then
16:        print aktuális
17:    else
18:        print „Sikertelen keresés”
19:    end if
20: end procedure
```

Csak olyan probléma megoldásánál alkalmazzuk, ahol egy a célfeltételeknek eleget tevő állapotot kell előállítani!
Ugyanazon probléma megoldásának keresése esetén a választás módjában lehet lényeges eltérés:

- irányítatlanul, szisztematikusan
  - előre rögzített operátorsorrend alapján
  - véletlenszerűen: **próba-hiba módszer**
- heurisztikusan: **hegymászó módszer**

Becsüljük meg a $h : \mathcal{A} \to R^+$ ún. **heurisztikus függvénnyel**, hogy az egyes állapotokból legkevesebb hány operátor alkalmazásával érhetünk célállapotba:

$$h(a) \rightleftharpoons \begin{cases} 0, & \text{ha } a \in \mathcal{C} \\ \infty, & \text{ha } \neg\exists c(c \in \mathcal{C} \wedge a \overset{*}{\Rightarrow} c), \end{cases}$$

egyébként $h(a) > 0$.

Ha az a állapot az aktuális, becsüljük meg h segítségével, hogy a milyen „távol” van a hozzá legközelebbi céltól: $h(a)$.

Legyen

$$\mathcal{O}' = \{o \mid o \in \mathcal{O} \wedge o\text{-}\mathit{alkalmazásának\text{-}előfeltétele}(a) \wedge h(o(a)) \leq h(a)\}.$$

Azt az $\mathcal{O}'$-beli *operátor*t fogjuk alkalmazni a-ra, amelyik a becslésünk szerint a legközelebb visz valamelyik terminálishoz:

$$h(\mathit{operátor}(a)) = \min\left\{h(o(a)) \mid o \in \mathcal{O}'\right\}$$

1: **procedure** HEGYMÁSZÓ-ALGORITMUS($\langle \mathcal{A}, \mathit{kezdő}, \mathcal{C}, \mathcal{O} \rangle$, h)
2: $\mathit{aktuális} \leftarrow \mathit{kezdő}$
3: **while** IGAZ **do**
4: **if** $\mathit{aktuális} \in \mathcal{C}$ **then**
5: **break**
6: **end if**
7: $\mathcal{O}' \leftarrow \{o \mid o \in \mathcal{O} \wedge \text{ELŐFELTÉTEL}(\mathit{aktuális}, o) \wedge h(\text{ALKALMAZ}(\mathit{aktuális}, o)) \leq h(\mathit{aktuális})\}$
8: **if** $\mathcal{O}' \neq \emptyset$ **then**
9: $\mathit{operátor} \leftarrow \text{VÁLASZT}(\{o \mid o \in \mathcal{O}' \wedge$
$\wedge \forall o'(o' \in \mathcal{O}' \supset h(\text{ALKALMAZ}(\mathit{aktuális}, o)) \leq h(\text{ALKALMAZ}(\mathit{aktuális}, o')))\})$
10: $\mathit{aktuális} \leftarrow \text{ALKALMAZ}(\mathit{aktuális}, \mathit{operátor})$
11: **else**
12: **break**
13: **end if**
14: **end while**
15: **if** $\mathit{aktuális} \in \mathcal{C}$ **then**
16: **print** *aktuális*
17: **else**
18: **print** „Sikertelen keresés”
19: **end if**
20: **end procedure**

A nemmódosítható megoldáskereső rendszerek értékelése:

Teljesség : Nem teljesek.

- Próba-hiba módszer: Ha olyan kört nem tartalmazó véges állapottérgráfokban keresünk, melyekben minden csúcsból vezet valamelyik terminális csúcsba irányított út, akkor előállít egy célállapotot.
- A hegymászó módszer esetén a heurisztika pontosságától függ, hogy a megoldást megtaláljuk-e vagy sem.

Tárigény : Rendkívül kis adatbázissal dolgozik.

2.2. Módosítható megoldáskereső rendszerek

2.2.1. Az alap visszalépéses megoldáskeresők

Legyen $p = \langle \mathcal{A}, kezd, \mathcal{C}, \mathcal{O} \rangle$. Az alap visszalépéses megoldáskereső

- **adatbázisa** egy a startcsúcsból induló az ún. aktuális csúcsba vezető utat, az aktuális utat tartalmazza,
  az út csúcsait és a csúccsal kapcsolatban lévő éleket nyilvántartó **csomópontokból** épül fel. Egy csomópont az alábbi információkat tartalmazza:
  - egy $a \in \mathcal{A}$ állapotot;
  - arra a csomópontra mutatót, mely a szülő állapotot (azt az állapotot, melyre operátort alkalmazva előállt a) tartalmazza;
  - azt az operátort, melyet a szülő állapotra alkalmazva előállt a;
  - a-ra a keresés során már alkalmazott (vagy még alkalmazható) operátorok halmazát.

- **műveletei**
  - **az operátorokból származtatott műveletek**: egy o operátor kiterjesztésével nyert művelet
    - alkalmazási előfeltétele: az aktuális csomópont állapotára alkalmazható o, de még a keresés során erre az állapotra (ezen az úton) még nem alkalmaztuk.
    - hatása:
  - **a visszalépés**
    - alkalmazási előfeltétele: van (aktuális) csomópont az aktuális úton.
    - hatása:
- **vezérlője** eldönti, hogy az adatbázisra mikor melyik műveletet kell végrehajtani, ha még nem teljesülnek a megállási feltételek.

```
1:  procedure ALAP-BACKTRACK-1(⟨𝒜, kezdő, 𝒞, 𝒪⟩)
2:      ÁLLAPOT[aktuális-csomópont] ← kezdő
3:      SZÜLŐ[aktuális-csomópont] ← NIL
4:      OPERÁTOR[aktuális-csomópont] ← ∗
5:      KIPRÓBÁLT[aktuális-csomópont] ← ∅
6:      while IGAZ do
7:          if aktuális-csomópont = NIL then
8:              break
9:          end if
10:         if ÁLLAPOT[aktuális-csomópont] ∈ 𝒞 then
11:             break
12:         end if
13:         𝒪′ ← {o | o ∈ 𝒪 ∧ ELŐFELTÉTEL(ÁLLAPOT[aktuális-csomópont], o) ∧
                                  ∧ o ∉ KIPRÓBÁLT[aktuális-csomópont]}
```

```
14:         if 𝒪′ ≠ ∅ then
15:             operátor ← VÁLASZT(𝒪′)
16:             KIPRÓBÁLT[aktuális-csomópont] ← KIPRÓBÁLT[aktuális-csomópont]∪{operátor}
17:             ÁLLAPOT[új] ← ALKALMAZ(ÁLLAPOT[aktuális-csomópont], operátor)
18:             SZÜLŐ[új] ← aktuális-csomópont
19:             OPERÁTOR[új] ← operátor
20:             KIPRÓBÁLT[új] ← ∅
21:             aktuális-csomópont ← új
22:         else
23:             aktuális-csomópont ← SZÜLŐ[aktuális-csomópont]
24:         end if
25:     end while
26:     if aktuális-csomópont ≠ NIL then
27:         MEGOLDÁS-KIÍR(aktuális-csomópont)
28:     else
29:         print „Nincs megoldás”
30:     end if
31: end procedure
```

```
1:  procedure ALAP-BACKTRACK-2(⟨𝒜, kezdő, 𝒞, 𝒪⟩)
2:      ÁLLAPOT[aktuális-csomópont] ← kezdő
3:      SZÜLŐ[aktuális-csomópont] ← NIL
4:      OPERÁTOR[aktuális-csomópont] ← ∗
5:      ALKALMAZHATÓ[aktuális-csomópont] ←
                                  {o | o ∈ 𝒪 ∧ ELŐFELTÉTEL(ÁLLAPOT[aktuális-csomópont], o)}
6:      while IGAZ do
7:          if aktuális-csomópont = NIL then
8:              break
9:          end if
10:         if ÁLLAPOT[aktuális-csomópont] ∈ 𝒞 then
11:             break
12:         end if
```

```
13:         if ALKALMAZHATÓ[aktuális-csomópont] ≠ ∅ then
14:             operátor ← VÁLASZT(ALKALMAZHATÓ[aktuális-csomópont])
15:             ALKALMAZHATÓ[aktuális-csomópont] ←
                                ALKALMAZHATÓ[aktuális-csomópont] \ {operátor}
16:             ÁLLAPOT[új] ← ALKALMAZ(ÁLLAPOT[aktuális-csomópont], operátor)
17:             SZÜLŐ[új] ← aktuális-csomópont
18:             OPERÁTOR[új] ← operátor
19:             ALKALMAZHATÓ[új] ← {o | o ∈ 𝒪 ∧ ELŐFELTÉTEL(ÁLLAPOT[új], o)}
20:             aktuális-csomópont ← új
21:         else
22:             aktuális-csomópont ← SZÜLŐ[aktuális-csomópont]
23:         end if
24:     end while
25:     if aktuális-csomópont ≠ NIL then
26:         MEGOLDÁS-KIÍR(aktuális-csomópont)
27:     else
28:         print „Nincs megoldás”
29:     end if
30: end procedure
```

Ugyanazon probléma megoldásának keresése esetén a választás módjában lehet lényeges eltérés:

- irányítatlanul, szisztematikusan
  - előre rögzített operátorsorrend alapján
  - véletlenszerűen
- heurisztikusan:
  Becsüljük meg a $h : \mathcal{A} \rightarrow R^+$ heurisztikával, hogy az egyes csúcsok milyen távol vannak a hozzájuk legközelebbi terminális csúcstól. Legyen

$$\mathcal{O}' = \{o \mid o \in \mathcal{O} \wedge o\text{-}\mathit{alkalmazásának\text{-}előfeltétele}(a)\} .$$

  Azt az $\mathcal{O}'$-beli *operátor*t fogjuk alkalmazni a-ra, amelyik a becslésünk szerint a legközelebb visz valamelyik terminálishoz:

$$h(\mathit{operátor}(a)) = \min \left\{ h(o(a)) \mid o \in \mathcal{O}' \right\} .$$

Az alap visszalépéses megoldáskeresők értékelése

Teljesség : Ha a reprezentációs gráf köröket nem tartalmazó véges gráf, akkor az alap visszalépéses megoldáskereső véges sok keresőlépés megtétele után befejezi a keresést,

- ha van megoldás, előállít egy lehetséges megoldást,
- ha nincs megoldás, azt felismeri.

Tárigény : Kis méretű az adatbázis.

2.2.2. Visszalépéses megoldáskeresés köröket is tartalmazó gráfokban

- **Visszalépéses megoldáskeresés körfigyeléssel:**
  Ha van megoldás, akkor van körmentes megoldás is. A vezérlő a visszalépést választja, ha az aktuális csúcs szerepelt már az aktuális úton.
- **Visszalépéses megoldáskeresés úthosszkorláttal:**
  Úthosszkorlátot vezetünk be, mely megakadályozza, hogy a köröket „végtelen sokszor” járjuk be. A vezérlő a visszalépést választja, ha az aktuális út hossza eléri, vagy meghaladja az úthosszkorlátot.

```
1: procedure KÖRFIGYELÉSES-BACKTRACK(⟨A, kezdő, C, O⟩)
2:     ÁLLAPOT[aktuális-csomópont] ← kezdő
3:     SZÜLŐ[aktuális-csomópont] ← NIL
4:     OPERÁTOR[aktuális-csomópont] ← ∗
5:     ALKALMAZHATÓ[aktuális-csomópont] ←
                          {o | o ∈ O ∧ ELŐFELTÉTEL(ÁLLAPOT[aktuális-csomópont], o)}
6:     while IGAZ do
7:         if aktuális-csomópont = NIL then
8:             break
9:         end if
10:        if ÁLLAPOT[aktuális-csomópont] ∈ C then
11:            break
12:        end if
13:        if VOLTMÁR(ÁLLAPOT[aktuális-csomópont]) then
14:            aktuális-csomópont ← SZÜLŐ[aktuális-csomópont]
15:        end if
```

```
16:         if ALKALMAZHATÓ[aktuális-csomópont] ≠ ∅ then
17:             operátor ← VÁLASZT(ALKALMAZHATÓ[aktuális-csomópont])
18:             ALKALMAZHATÓ[aktuális-csomópont] ←
                                ALKALMAZHATÓ[aktuális-csomópont] \ {operátor}
19:             ÁLLAPOT[új] ← ALKALMAZ(ÁLLAPOT[aktuális-csomópont], operátor)
20:             SZÜLŐ[új] ← aktuális-csomópont
21:             OPERÁTOR[új] ← operátor
22:             ALKALMAZHATÓ[új] ← {o | o ∈ 𝒪 ∧ ELŐFELTÉTEL(ÁLLAPOT[új], o)}
23:             aktuális-csomópont ← új
24:         else
25:             aktuális-csomópont ← SZÜLŐ[aktuális-csomópont]
26:         end if
27:     end while
28:     if aktuális-csomópont ≠ NIL then
29:         MEGOLDÁS-KIÍR(aktuális-csomópont)
30:     else
31:         print „Nincs megoldás”
32:     end if
33: end procedure
```

```
1:  procedure ÚthosszkorlátosBacktrack(⟨A, kezdő, C, O⟩, korlát)
2:      Állapot[aktuális-csomópont] ← kezdő
3:      Szülő[aktuális-csomópont] ← Nil
4:      Mélység[aktuális-csomópont] ← 0
5:      Operátor[aktuális-csomópont] ← ∗
6:      Alkalmazható[aktuális-csomópont] ←
                              {o | o ∈ O ∧ Előfeltétel(Állapot[aktuális-csomópont], o)}
7:      while Igaz do
8:          if aktuális-csomópont = Nil then
9:              break
10:         end if
11:         if Állapot[aktuális-csomópont] ∈ C then
12:             break
13:         end if
14:         if Mélység[aktuális-csomópont] = korlát then
15:             aktuális-csomópont ← Szülő[aktuális-csomópont]
16:         end if
```

```
17:         if ALKALMAZHATÓ[aktuális-csomópont] ≠ ∅ then
18:             operátor ← VÁLASZT(ALKALMAZHATÓ[aktuális-csomópont])
19:             ALKALMAZHATÓ[aktuális-csomópont] ←
                                ALKALMAZHATÓ[aktuális-csomópont] \ {operátor}
20:             ÁLLAPOT[új] ← ALKALMAZ(ÁLLAPOT[aktuális-csomópont], operátor)
21:             SZÜLŐ[új] ← aktuális-csomópont
22:             MÉLYSÉG[új] ← MÉLYSÉG[aktuális-csomópont] + 1
23:             OPERÁTOR[új] ← operátor
24:             ALKALMAZHATÓ[új] ← {o | o ∈ O ∧ ELŐFELTÉTEL(ÁLLAPOT[új], o)}
25:             aktuális-csomópont ← új
26:         else
27:             aktuális-csomópont ← SZÜLŐ[aktuális-csomópont]
28:         end if
29:     end while
30:     if aktuális-csomópont ≠ NIL then
31:         MEGOLDÁS-KIÍR(aktuális-csomópont)
32:     else
33:         print „Sikertelen keresés”
34:     end if
35: end procedure
```

Értékelés

Teljesség :

- A körfigyeléses backtrack ha a reprezentációs gráf véges, akkor véges sok keresőlépés megtétele után befejezi a keresést, és
  - ha van megoldás, előállít egy körmentes megoldást,
  - ha nincs megoldás, azt felismeri.
- Az úthosszkorlátos backtrack tetszőleges reprezentációs gráf esetén véges sok keresőlépés megtétele után befejezi a keresést,
  - ha van az úthosszkorlátnál nem hosszabb megoldás, előállít egy ilyen megoldást.
  - Ha keresés nem egy megoldás megtalálásával ér véget, akkor az úthosszkorlátnál csak hosszabb megoldás lehet a reprezentációs gráfban. (Vagy nincs megoldás, vagy az úthosszkorlát túl kicsi.)

Időigény :

- A körfigyeléses backtrack időigényes (főleg hosszú körök esetén).

Tárigény :

- Az úthosszkorlátos backtrack adatbázisa legfeljebb úthosszkorlátnyi elemet tartalmaz. A megtalált megoldás nem feltétlen körmentes.

2.2.3. Ág és korlát algoritmus

A backtrack alkalmas optimális (legrövidebb) megoldás keresésére is.

- Egy induló úthosszkorlátnál nem hosszabb megoldást keresünk.
- Ha találunk ilyet, tároljuk, majd ennek hosszát új úthosszkorlátnak választva folytatjuk a keresést.

2.1. MEGJEGYZÉS. Úthosszkorlát helyett költségkorlátot, csomópont mélysége helyett pedig az addig tartó út költségét is használhatjuk az algoritmusban. Ekkor legkisebb költségű megoldás előállítása lehet a cél.

```
1:  procedure Ág-és-Korlát(⟨A, kezdő, C, O⟩, korlát)
2:      talált ← Hamis
3:      Állapot[aktuális-csomópont] ← kezdő
4:      Szülő[aktuális-csomópont] ← Nil
5:      Mélység[aktuális-csomópont] ← 0
6:      Operátor[aktuális-csomópont] ← *
7:      Alkalmazható[aktuális-csomópont] ←
                              {o | o ∈ O ∧ Előfeltétel(Állapot[aktuális-csomópont], o)}
8:      while Igaz do
9:          if aktuális-csomópont = Nil then
10:             break
11:         end if
12:         if Állapot[aktuális-csomópont] ∈ C then
13:             talált ← Igaz
14:             Megoldás-Feljegyez(aktuális-csomópont)
15:             korlát ← Mélység[aktuális-csomópont]
16:         end if
17:         if Mélység[aktuális-csomópont] = korlát then
18:             aktuális-csomópont ← Szülő[aktuális-csomópont]
19:         end if
```

```
20:         if ALKALMAZHATÓ[aktuális-csomópont] ≠ ∅ then
21:             operátor ← VÁLASZT(ALKALMAZHATÓ[aktuális-csomópont])
22:             ALKALMAZHATÓ[aktuális-csomópont] ←
                                ALKALMAZHATÓ[aktuális-csomópont] \ {operátor}
23:             ÁLLAPOT[új] ← ALKALMAZ(ÁLLAPOT[aktuális-csomópont], operátor)
24:             SZÜLŐ[új] ← aktuális-csomópont
25:             MÉLYSÉG[új] ← MÉLYSÉG[aktuális-csomópont] + 1
26:             OPERÁTOR[új] ← operátor
27:             ALKALMAZHATÓ[új] ← {o | o ∈ 𝒪 ∧ ELŐFELTÉTEL(ÁLLAPOT[új], o)}
28:             aktuális-csomópont ← új
29:         else
30:             aktuális-csomópont ← SZÜLŐ[aktuális-csomópont]
31:         end if
32:     end while
33:     if talált then
34:         MEGOLDÁS-KIÍR
35:     else
36:         print „Sikertelen keresés”
37:     end if
38: end procedure
```

Értékelés

Optimalitás : Az ág és korlát algoritmus tetszőleges reprezentációs gráf esetén véges sok keresőlépés megtétele után befejezi a keresést,

- ha van az induló úthosszkorlátnál nem hosszabb megoldás, a legrövidebb megoldást állítja elő,
- ha a keresés nem megoldás megtalálásával ér véget, akkor az induló úthosszkorlátnál csak hosszabb megoldás lehet a reprezentációs gráfban. (Vagy nincs megoldás, vagy az induló úthosszkorlát túl kicsi.)

Tárigény :

- Az ág és korlát adatbázisa legfeljebb kétszer az induló úthosszkorlátnyi csomópontot tartalmaz.

2.2.4. Keresőfával keresők

Legyen $p = \langle \mathcal{A}, kezd, \mathcal{C}, \mathcal{O} \rangle$. A keresőfával kereső rendszerek

- **adatbázisa** a reprezentációs gráf már bejárt részét feszítő fa, az ún. **keresőfa**. A keresőfa csúcsait és a velük kapcsolatban lévő éleket (explicit vagy implicit módon) nyilvántartó csomópontok az alábbi információkat tartalmazzák:
  - egy $a \in \mathcal{A}$ állapotot;
  - arra a csomópontra mutatót, mely a szülő állapotot tartalmazza;
  - azt az operátort, melyet a szülő állapotra alkalmazva előállt a;
  - *státusz*:
    - **zárt**, ha a utódait tartalmazó csomópontokat a keresés során már előállítottuk;
    - **nyílt**, egyébként.

- **művelete a kiterjesztés**: a keresőfát annak egy nyílt csomópontján keresztül kibővíti.
  - alkalmazási előfeltétele: a keresőfában van nyílt csomópont.
  - hatása:
    - alkalmazzuk az összes alkalmazható operátort a nyílt csomópont állapotára,
    - az előálló állapotok közül
      - amelyek még nem szerepeltek a keresőfa egyetlen csomópontjában sem, azokból a keresőfába felfűzött új nyílt csomópont készül,
      - amelyek már szerepeltek a keresőfa valamely csomópontjában, azok sorsa keresőfüggő.
    - a kiterjesztett csomópont zárttá válik.

- **vezérlő** megmondja, hogy melyik nyílt csomópont legyen a következő lépésben kiterjesztve.
  - Ha a kiválasztott nyílt csomópont állapota teljesíti a célfeltételeket, a keresőfában a szülőre mutatók mentén elő tudunk állítani egy megoldást is.
  - Nincs megoldás, ha egyetlenegy nyílt csomópont sincs a keresőfában.

```
1:  procedure KERESŐFÁVAL-KERESŐ(⟨𝒜, kezdő, 𝒞, 𝒪⟩)
2:      ÁLLAPOT[csomópont] ← kezdő
3:      SZÜLŐ[csomópont] ← NIL
4:      OPERÁTOR[csomópont] ← ∗
5:      nyíltak ← {csomópont}; zártak ← ∅
6:      while IGAZ do
7:          if nyíltak = ∅ then
8:              break
9:          end if
10:         csomópont ← VÁLASZT(nyíltak)
11:         if ÁLLAPOT[csomópont] ∈ 𝒞 then
12:             break
13:         end if
14:         KITERJESZT(csomópont, nyíltak, zártak)
15:     end while
16:     if nyíltak ≠ ∅ then
17:         MEGOLDÁS-KIÍR(csomópont)
18:     else
19:         print „Nincs megoldás”
20:     end if
21: end procedure
```

Ugyanazon probléma megoldásának keresése esetén lényeges eltérés lehet

1. a választás módjában. A vezérlő választhat
   - irányítatlanul, szisztematikusan
     - a csomópontok keresőgráfbeli mélysége alapján: szélességi és mélységi keresők;
     - a csomópontok állapotait előállító költség alapján: optimális kereső;
   - heurisztikusan:
     - best-first algoritmus;
     - **A** algoritmusok.
2. abban, hogy mi történik, ha a keresőgráf egy csúcsához a keresés során újabb odavezető utat tár fel a vezérlő.
3. a célfeltételek vizsgálatának időpontjában.

2.2.5. Szélességi és mélységi keresők

1. Egy csomópont előállításakor követjük, hogy a csomópontban nyilvántartott csúcs a keresőfában milyen „mélyen” van:

$$\mathit{mélység}(m) \rightleftharpoons \begin{cases} 0 & \text{ha } m = s \\ \mathit{mélység}(n) + 1 & (n, m) \in E. \end{cases}$$

Kiterjesztésre

- a szélességi kereső vezérlője a **legkisebb mélységi számú**
- a mélységi kereső vezérlője a **legnagyobb mélységi számú**

nyílt csomópontot választja ki.

2. Ha a vezérlő a keresőgráf egy csúcsához a keresés során **újabb odavezető utat** tár fel, ezt **nem tárolja**, „elfelejti”.

3. A tesztelést **előre hozhatjuk**.

```
1: procedure KITERJESZT(⟨A, kezdő, C, O⟩, csomópont, nyíltak, zártak)
2:     for all o ∈ O do
3:         if ELŐFELTÉTEL(ÁLLAPOT[csomópont], o) then
4:             állapot ← ALKALMAZ(ÁLLAPOT[csomópont],o)
5:             ny ← KERES(nyíltak, állapot)
6:             z ← KERES(zártak, állapot)
7:             if ny = NIL and z = NIL then
8:                 ÁLLAPOT[új-csomópont] ← állapot
9:                 SZÜLŐ[új-csomópont] ← csomópont
10:                OPERÁTOR[új-csomópont] ← o
11:                MÉLYSÉG[új-csomópont] ← MÉLYSÉG[csomópont] + 1
12:                nyíltak ← nyíltak ∪ {új-csomópont}
13:            end if
14:        end if
15:    end for
16:    nyíltak ← nyíltak \ {csomópont}
17:    zártak ← zártak ∪ {csomópont}
18: end procedure
```

19: **procedure** SZÉLESSÉGI-KERESŐ($\langle \mathcal{A}, \mathit{kezdő}, \mathcal{C}, \mathcal{O} \rangle$)
20: ÁLLAPOT[*új-csomópont*] $\leftarrow$ *kezdő*
21: SZÜLŐ[*új-csomópont*] $\leftarrow$ NIL
22: OPERÁTOR[*új-csomópont*] $\leftarrow *$
23: MÉLYSÉG[*új-csomópont*] $\leftarrow 0$
24: *nyíltak* $\leftarrow \{\mathit{új\text{-}csomópont}\}$
25: *zártak* $\leftarrow \emptyset$
26: **while** IGAZ **do**
27: **if** *nyíltak* $= \emptyset$ **then**
28: **break**
29: **end if**
30: *csomópont* $\leftarrow$ VÁLASZT($\{cs \mid cs \in \mathit{nyíltak} \wedge$
$\wedge \forall cs'(cs' \in \mathit{nyíltak} \supset$ MÉLYSÉG$[cs] \leq$ MÉLYSÉG$[cs'])\}$)
31: **if** ÁLLAPOT[*csomópont*] $\in \mathcal{C}$ **then**
32: **break**
33: **end if**
34: KITERJESZT($\langle \mathcal{A}, \mathit{kezdő}, \mathcal{C}, \mathcal{O} \rangle$, *csomópont*, *nyíltak*, *zártak*)
35: **end while**

```
36:     if nyíltak ≠ ∅ then
37:         MEGOLDÁS-KIÍR(csomópont)
38:     else
39:         print „Nincs megoldás”
40:     end if
41: end procedure
```

```
1:  procedure MÉLYSÉGI-KERESŐ(⟨𝒜, kezdő, 𝒞, 𝒪⟩)
2:      ÁLLAPOT[új-csomópont] ← kezdő
3:      SZÜLŐ[új-csomópont] ← NIL
4:      OPERÁTOR[új-csomópont] ← *
5:      MÉLYSÉG[új-csomópont] ← 0
6:      nyíltak ← {új-csomópont}
7:      zártak ← ∅
8:      while IGAZ do
9:          if nyíltak = ∅ then
10:             break
11:         end if
12:         csomópont ← Választ({cs | cs ∈ nyíltak ∧
                                ∧ ∀cs′(cs′ ∈ nyíltak ⊃ MÉLYSÉG[cs] ≥ MÉLYSÉG[cs′])})
13:         if ÁLLAPOT[csomópont] ∈ 𝒞 then
14:             break
15:         end if
16:         KITERJESZT(⟨𝒜, kezdő, 𝒞, 𝒪⟩, csomópont, nyíltak, zártak)
17:     end while
```

```
18:     if nyíltak ≠ ∅ then
19:         MEGOLDÁS-KIÍR(csomópont)
20:     else
21:         print „Nincs megoldás”
22:     end if
23: end procedure
```

A szélességi kereső értékelése

Teljesség : A vezérlő,

- ha van megoldás, tetszőleges reprezentációs gráfban véges sok keresőlépés után előállít egy megoldást,
- ha nincs az adott reprezentációban megoldás, akkor véges gráf esetén azt a nyílt csomópontok elfogyásával felismeri.

Optimalitás : Ha van megoldás, tetszőleges reprezentációs gráfban a vezérlő a legrövidebb megoldást állítja elő.

Tárigény : Nagy az adatbázis. Legyen a reprezentációs fa minden csúcsának d gyermeke, és l hosszúságú a legrövidebb megoldás. Ekkor a keresőgráf csomópontjainak száma a keresés végére (a legrosszabb esetben):

$$1 + d + d^2 + d^3 + \ldots + d^{l+1} - d \approx O(d^{l+1}).$$

A mélységi kereső értékelése

Teljesség : A vezérlő véges reprezentációs gráfban,

- ha van megoldás, véges sok keresőlépés után előállít egy megoldást,
- ha nincs a problémának az adott reprezentációban megoldása, akkor azt a nyílt csomópontok elfogyásával felismeri.

2.2. MEGJEGYZÉS. A nyílt csomópontokat gyakran

- a szélességi kereső **sorban**,
- a mélységi kereső **veremben**

tartja nyilván, melyből mélységi szám szerint ezek épp megfelelő sorrendben kerülnek ki.

2.2.6. Optimális kereső

1. Minden csomópontnál számon tartjuk az odavezető nyilvántartott út költségét:

$$\textit{útköltség}(m) \rightleftharpoons \begin{cases} 0 & \text{ha } m = s \\ \textit{útköltség}(n) + \textit{költség}(n, m) & (n, m) \in E. \end{cases}$$

$\textit{útköltség}^*(n)$ jelölje a startcsúcsból n-be jutás optimális költségét. Ekkor

$$\textit{útköltség}(n) \geq \textit{útköltség}^*(n), \text{ minden } n \in N\text{-re.}$$

Kiterjesztésre az optimális kereső vezérlője a **legkisebb költségű** nyílt csomópontot választja ki.

2. Ha a vezérlő a keresőgráf egy csúcsához a keresés során **újabb odavezető utat** tár fel, azaz az n csomópont kiterjesztéskor előállt állapot szerepel már a keresőgráf m csomópontjában

- nyíltként: ha az

$$\textit{útköltség}(n) + \textit{költség}(n, m) < \textit{útköltség}(m),$$

ekkor az új kisebb költségű utat tároljuk, a régit „elfelejtjük”.

- zártként: az m csomópontot már n kiterjesztése előtt kiterjesztette a vezérlő, azaz $\textit{útköltség}(m) \leq \textit{útköltség}(n)$ volt, így

$$\textit{útköltség}(n) + \textit{költség}(n, m) > \textit{útköltség}(m).$$

Az m-hez feltárt új út költségesebb.

3. A tesztelést **nem hozhatjuk előre**.

```
1: procedure KITERJESZT(⟨A, kezdő, C, O⟩, költség, csomópont, nyíltak, zártak)
2:     for all o ∈ O do
3:         if ELŐFELTÉTEL(ÁLLAPOT[csomópont], o) then
4:             állapot ← ALKALMAZ(ÁLLAPOT[csomópont],o)
5:             ny ← KERES(nyíltak, állapot)
6:             z ← KERES(zártak, állapot)
7:             if ny = NIL and z = NIL then
8:                 ÁLLAPOT[új-csomópont] ← állapot
9:                 SZÜLŐ[új-csomópont] ← csomópont
10:                OPERÁTOR[új-csomópont] ← o
11:                ÚTKÖLTSÉG[új-csomópont] ← ÚTKÖLTSÉG[csomópont] + költség(o, ÁLLAPOT[csomópont])
12:                nyíltak ← nyíltak ∪ {új-csomópont}
13:            else if ny ≠ NIL then
14:                új-út-költség ← ÚTKÖLTSÉG[csomópont] + költség(o, ÁLLAPOT[csomópont])
15:                if új-út-költség < ÚTKÖLTSÉG[ny] then
16:                    SZÜLŐ[ny] ← csomópont
17:                    OPERÁTOR[ny] ← o
18:                    ÚTKÖLTSÉG[ny] ← új-út-költség
19:                end if
20:            end if
21:        end if
22:    end for
23:    nyíltak ← nyíltak \ {csomópont}
24:    zártak ← zártak ∪ {csomópont}
25: end procedure
```

```
26: procedure OPTIMÁLIS-KERESŐ(⟨A, kezdő, C, O⟩, költség)
27:     ÁLLAPOT[új-csomópont] ← kezdő
28:     SZÜLŐ[új-csomópont] ← NIL
29:     OPERÁTOR[új-csomópont] ← ∗
30:     ÚTKÖLTSÉG[új-csomópont] ← 0
31:     nyíltak ← {új-csomópont}
32:     zártak ← ∅
33:     while IGAZ do
34:         if nyíltak = ∅ then
35:             break
36:         end if
37:         csomópont ← VÁLASZT({cs | cs ∈ nyíltak∧∀cs′(cs′ ∈ nyíltak ⊃ ÚTKÖLTSÉG[cs] ≤ ÚTKÖLTSÉG[cs′])})
38:         if ÁLLAPOT[csomópont] ∈ C then
39:             break
40:         end if
41:         KITERJESZT(⟨A, kezdő, C, O⟩, költség, csomópont, nyíltak, zártak)
42:     end while
43:     if nyíltak ≠ ∅ then
44:         MEGOLDÁS-KIÍR(csomópont)
45:     else
46:         print „Nincs megoldás”
47:     end if
48: end procedure
```

Az optimális kereső értékelése

Teljesség : A vezérlő,

- ha van megoldás, tetszőleges reprezentációs gráfban véges sok keresőlépés után előállít egy megoldást,
- ha nincs a problémának az adott reprezentációban megoldása, akkor véges gráf esetén azt a nyílt csomópontok elfogyásával felismeri.

Optimalitás : Ha van megoldás, tetszőleges reprezentációs gráfban a vezérlő véges sok keresőlépés után az optimális megoldást állítja elő.

2.2.7. Best-first algoritmus

1. A keresőfa minden csomópontjánál nyilvántartjuk, hogy a csomópontbeli állapot a h heurisztikus függvény szerint milyen „távol” van a hozzá legközelebbi céltól.
Kiterjesztésre a best-first vezérlője a **legkisebb heurisztikájú** nyílt csomópontot választja ki (legjobb irányban haladó keresés).
2. Ha a vezérlő a keresőgráf egy csúcsához a keresés során **újabb odavezető utat** tár fel, ezt **nem tárolja**, „elfelejti”.
3. A tesztelést **előre hozhatjuk**.

```
1:  procedure KITERJESZT(⟨A, kezdő, C, O⟩, h, csomópont, nyíltak, zártak)
2:      for all o ∈ O do
3:          if ELŐFELTÉTEL(ÁLLAPOT[csomópont], o) then
4:              állapot ← ALKALMAZ(ÁLLAPOT[csomópont],o)
5:              ny ← KERES(nyíltak, állapot)
6:              z ← KERES(zártak, állapot)
7:              if ny = NIL and z = NIL then
8:                  ÁLLAPOT[új-csomópont] ← állapot
9:                  SZÜLŐ[új-csomópont] ← csomópont
10:                 OPERÁTOR[új-csomópont] ← o
11:                 HEURISZTIKA[új-csomópont] ← h(állapot)
12:                 nyíltak ← nyíltak ∪ {új-csomópont}
13:             end if
14:         end if
15:     end for
16:     nyíltak ← nyíltak \ {csomópont}
17:     zártak ← zártak ∪ {csomópont}
18: end procedure
```

```
19: procedure BEST-FIRST(⟨A, kezdő, C, O⟩, h)
20:     ÁLLAPOT[új-csomópont] ← kezdő
21:     SZÜLŐ[új-csomópont] ← NIL
22:     OPERÁTOR[új-csomópont] ← *
23:     HEURISZTIKA[új-csomópont] ← h(kezdő)
24:     nyíltak ← {új-csomópont}
25:     zártak ← ∅
26:     while IGAZ do
27:         if nyíltak = ∅ then
28:             break
29:         end if
30:         csomópont ← VÁLASZT({cs | cs ∈ nyíltak ∧
                                ∧∀cs′(cs′ ∈ nyíltak ⊃ HEURISZTIKA[cs] ≤ HEURISZTIKA[cs′])})
31:         if ÁLLAPOT[csomópont] ∈ C then
32:             break
33:         end if
34:         KITERJESZT(⟨A, kezdő, C, O⟩, h, csomópont, nyíltak, zártak)
35:     end while
```

```
36:     if nyíltak ≠ ∅ then
37:         MEGOLDÁS-KIÍR(csomópont)
38:     else
39:         print „Nincs megoldás”
40:     end if
41: end procedure
```

A best-first algoritmus értékelése

Teljesség : A vezérlő tetszőleges véges reprezentációs gráfban,

- ha van megoldás, véges sok keresőlépés után előállít egy megoldást,
- ha nincs a problémának az adott reprezentációban megoldása, akkor azt a nyílt csomópontok elfogyásával felismeri.

Tárigény : Perfekt heurisztika esetén, ha a reprezentációs fa minden csúcsának d gyermeke van, s l hosszú a legrövidebb megoldás, a keresőgráf csomópontjainak száma a keresés végére:

$$1 + d + d + ... + d = O(l \cdot d).$$

2.2.8. Az A algoritmus

1. A keresőgráf minden csomópontjában megbecsüljük a rajta keresztülhaladó megoldás költségét. Ez egyrészt a csomópontig vezető nyilvántartott út költsége, amihez hozzászámítjuk a célig hátralevő út becsült költségét:

$$\textit{összköltség}(m) = \textit{útköltség}(m) + \textit{heurisztika}(m),$$

azaz

$$\textit{összköltség}(m) = \textit{útköltség}(n) + \textit{költség}(n, m) + \textit{heurisztika}(m),$$

ha $(n, m) \in E$. Ha $\textit{összköltség}^*(m)$-mel jelöljük az m csúcson keresztül célba jutás optimális költségét, akkor minden m csúcsra

$$\textit{összköltség}^*(m) \approx \textit{összköltség}(m)$$

Kiterjesztésre az **A** algoritmus vezérlője a **legkisebb összköltségű** nyílt csomópontot választja ki.

2. Ha a vezérlő a keresőgráf egy csúcsához a keresés során **újabb odavezető utat** tár fel, azaz az n csomópont kiterjesztéskor előállt állapot szerepel már a keresőgráf m csomópontjában, és az

$$\textit{útköltség}(n) + \textit{költség}(n, m) < \textit{útköltség}(m),$$

ekkor az új kisebb költségű utat tároljuk, a régit „elfelejtjük”.

- Ha m nyílt volt, más teendő nincs.
- Ha m zárt volt, a keresőfa m-ből induló részének csomópontjaiban az *útköltség*-et frissíteni kell, ami problémát okoz:
  - külön eljárást írunk a frissítésre;
  - az **A** algoritmussal frissíttetjük a részgráfot;
  - megelőzzük a probléma kialakulását.

3. A tesztelést **nem hozhatjuk előre**.

```
1: procedure KITERJESZT(⟨A, kezdő, C, O⟩, költség, h, csomópont, nyíltak, zártak)
2:     for all o ∈ O do
3:         if ELŐFELTÉTEL(ÁLLAPOT[csomópont], o) then
4:             állapot ← ALKALMAZ(ÁLLAPOT[csomópont],o)
5:             ny ← KERES(nyíltak, állapot)
6:             z ← KERES(zártak, állapot)
7:             if ny = NIL and z = NIL then
8:                 ÁLLAPOT[új-csomópont] ← állapot
9:                 SZÜLŐ[új-csomópont] ← csomópont
10:                OPERÁTOR[új-csomópont] ← o
11:                ÚTKÖLTSÉG[új-csomópont] ← ÚTKÖLTSÉG[csomópont] + költség(o, ÁLLAPOT[csomópont])
12:                HEURISZTIKA[új-csomópont] ← h(állapot)
13:                nyíltak ← nyíltak ∪ {új-csomópont}
14:            else
15:                új-út-költség ← ÚTKÖLTSÉG[csomópont] + költség(o, ÁLLAPOT[csomópont])
```

```
16:                 if ny ≠ NIL then
17:                     if új-út-költség < ÚTKÖLTSÉG[ny] then
18:                         SZÜLŐ[ny] ← csomópont
19:                         OPERÁTOR[ny] ← o
20:                         ÚTKÖLTSÉG[ny] ← új-út-költség
21:                     end if
22:                 else
23:                     if új-út-költség < ÚTKÖLTSÉG[z] then
24:                         SZÜLŐ[z] ← csomópont
25:                         OPERÁTOR[z] ← o
26:                         ÚTKÖLTSÉG[z] ← új-út-költség
27:                         zártak ← zártak \ {z}
28:                         nyíltak ← nyíltak ∪ {z}
29:                     end if
30:                 end if
31:             end if
32:         end if
33:     end for
34:     nyíltak ← nyíltak \ {csomópont}
35:     zártak ← zártak ∪ {csomópont}
36: end procedure
```

```
37: procedure A-ALGORITMUS(⟨A, kezdő, C, O⟩, költség, h)
38:     ÁLLAPOT[új-csomópont] ← kezdő
39:     SZÜLŐ[új-csomópont] ← NIL
40:     OPERÁTOR[új-csomópont] ← ∗
41:     ÚTKÖLTSÉG[új-csomópont] ← 0
42:     HEURISZTIKA[új-csomópont] ← h(kezdő)
43:     nyíltak ← {új-csomópont}
44:     zártak ← ∅
45:     while IGAZ do
46:         if nyíltak = ∅ then
47:             break
48:         end if
49:         csomópont ← VÁLASZT({cs | cs ∈ nyíltak ∧
                ∧∀cs′(cs′ ∈ nyíltak ⊃ (ÚTKÖLTSÉG[cs]+HEURISZTIKA[cs]) ≤ (ÚTKÖLTSÉG[cs′]+HEURISZTIKA[cs′]))})
50:         if ÁLLAPOT[csomópont] ∈ C then
51:             break
52:         end if
53:         KITERJESZT(⟨A, kezdő, C, O⟩, költség, h, csomópont, nyíltak, zártak)
54:     end while
55:     if nyíltak ≠ ∅ then
56:         MEGOLDÁS-KIÍR(csomópont)
57:     else
58:         print „Nincs megoldás”
59:     end if
60: end procedure
```

Az A algoritmus értékelése

Teljesség : A vezérlő,

- ha van megoldás, tetszőleges reprezentációs gráfban véges sok keresőlépés után előállít egy megoldást,
- ha nincs a problémának az adott reprezentációban megoldása, akkor véges gráf esetén azt a nyílt csomópontok elfogyásával felismeri.

Optimalitás : Nincs garancia az optimális megoldás előállítására. De ha minden $a \in \mathcal{A}$ esetén

$$h(a) \leq h^*(a),$$

ahol $h^*(a)$ az a állapotból célba jutás optimális költsége, akkor az **A** algoritmus az optimális megoldást állítja elő, ha van megoldás. Ez az **A* algoritmus**.

2.3. LEMMA. *Az A algoritmus a működése során egy csúcsot legföljebb véges sokszor terjeszt ki.*

BIZONYÍTÁS. Egy csúcsot csak akkor terjeszthetünk ki, ha nyílt. A nyílt csúcsok közé legfeljebb annyiszor kerülhet, ahányszor egy minden addiginál olcsóbb utat találunk hozzá. Belátjuk, hogy véges sok ilyen út van.

Jelölje δ az élek költségének pozitív alsó korlátját, vagyis minden (n, m) esetén

$$\mathit{költség}(n, m) \geq \delta > 0.$$

Tegyük föl, hogy a p csúcsba először egy $\mathit{útköltség}(p)$ költségű úton jutunk el. Ez az út legfeljebb $\mathit{útköltség}(p)/\delta$ hosszú lehet. Az ennél olcsóbb p-be vezető utak $\mathit{útköltség}(p)/\delta$-nál biztosan rövidebbek. A $\mathit{útköltség}(p)/\delta$ korlátnál rövidebb p-be vezető utak száma viszont véges. □

2.4. LEMMA. *Az A algoritmus, hacsak közben nem fejezi be sikeresen a keresést, minden a nyíltak halmazba bekerülő csomópontot véges sok lépés után kiterjeszt.*

BIZONYÍTÁS. Legyen $p \in \mathit{nyíltak}$. Megmutatjuk, hogy p kiválasztása előtt kiterjesztendő (nála kisebb összköltséggel rendelkező) csomópontok száma véges és a 2.3 lemma miatt egy ilyen csak véges sokszor kerülhet vissza a nyílt csomópontok közé.

- Először belátjuk, hogy egy csomópont összköltsége arányos a csomópont mélységével. Amikor egy n csomópont bekerül a *nyíltak* halmazba, akkor

$$\begin{aligned} \mathit{összköltség}(n) = \mathit{útköltség}(n) + \mathit{heurisztika}(n) \geq \mathit{útköltség}(n) \geq \\ \geq \mathit{útköltség}^*(n) \geq \mathit{úthossz}^*(n) \cdot \delta, \end{aligned}$$

  ahol $\mathit{úthossz}^*(n)$ az s-ből n-be vezető optimális költségű út hossza.

- Most egy mélységi korlátot adunk meg. Legyen

$$K = \left[\frac{\textit{összköltség}(p)}{\delta}\right] + 1. \quad \text{(Nyilván } K \geq \textit{összköltség}(p)/\delta\text{).}$$

- Ennél a korlátnál mélyebben fekvő csomópontokra az összköltség nagyobb, mint $\textit{összköltség}(p)$. Ugyanis ha egy k csomópontra $\textit{úthossz}^*(k) > K$, akkor

$$\textit{összköltség}(k) \geq \textit{úthossz}^*(k) \cdot \delta > K \cdot \delta \geq \textit{összköltség}(p).$$

  Tehát $\textit{összköltség}(k) > \textit{összköltség}(p)$, azaz az $\textit{összköltség}(p)$-nél nem nagyobb összköltséggel rendelkező csomópontok a K mélységi korlátnál magasabban helyezkednek el.

- Mivel az egyes csomópontokból kiinduló élek száma fölülről korlátos, így egy adott mélységi korlátnál magasabban fekvő csomópontok száma véges.

□

2.5. TÉTEL. *Az A algoritmus véges reprezentációs gráfban véges sok lépés után befejezi a keresést.*

BIZONYÍTÁS. A 2.3. lemma értelmében az A algoritmus a véges sok lehetséges csomópont mindegyikét legfeljebb véges sokszor terjesztheti ki. Ez azt jelenti, hogy véges sok lépésen belül az összes csomópontot végleg ki is terjeszti, ha előbb nem áll le sikeresen a keresés. Az algoritmus tehát

- vagy talál célállapotot tartalmazó csomópontot,
- vagy pedig elfogynak a nyílt csomópontok,

és befejeződik a keresés. □

2.6. LEMMA. *Ha van megoldás, az A algoritmus adatbázisában a nyílt csomópontok között mindig van az optimális úton fekvő csúcs.*

BIZONYÍTÁS. Legyen $s = n_0, n_1, \ldots, n_k = t$ optimális út.

1. kiválasztás előtt: $s \in$ *nyíltak.*

k. kiválasztás előtt: indukciós feltevésünk szerint $\exists j (n_j \in$ *nyíltak*$)$. Legyen i a legkisebb ilyen index.

k+1. kiválasztás előtt:
- Ha nem n_i-t terjesztjük ki, akkor n_i nyílt marad.
- Ha n_i-t terjesztjük ki, akkor akár új csomópont, akár már szerepelt a keresőfában: n_{i+1} nyílt lesz.

□

2.7. TÉTEL. *Tetszőleges reprezentációs gráf esetén, ha van megoldás, az A algoritmus véges sok lépésben megoldással fejezi be a keresést.*

2.8. LEMMA. *Az A^* algoritmus által kiterjesztésre kiválasztott tetszőleges n csomópontra*

$$\text{összköltség}(n) \leq \text{összköltség}^*(s).$$

BIZONYÍTÁS. Ha a gráfban nincs megoldás, $\mathit{összköltség}^*(s) = \infty$, egyébként $\mathit{összköltség}^*(s) = \mathit{útköltség}^*(s) + \mathit{heurisztika}^*(s) < \infty$ az optimális megoldás költsége.
A 2.6. lemma szerint van n kiterjesztése előtt a nyíltak között az optimális úton fekvő csomópont. Legyen m az első ilyen. Ekkor $\mathit{útköltség}(m) = \mathit{útköltség}^*(m)$. Az algoritmus az n csúcsot választotta kiterjesztésre, tehát $\mathit{összköltség}(n) \leq \mathit{összköltség}(m)$. De

$$\begin{aligned}&\mathit{összköltség}(m) = \mathit{útköltség}^*(m) + \mathit{heurisztika}(m) \leq \\ &\leq \mathit{útköltség}^*(m) + \mathit{heurisztika}^*(m) = \mathit{összköltség}^*(m) = \mathit{összköltség}^*(s),\end{aligned}$$

amiből $\mathit{összköltség}(n) \leq \mathit{összköltség}^*(s)$ következik. □

A 2.8. lemmát a következőképpen is megfogalmazhatjuk:
Annak szükséges feltétele, hogy az A^* algoritmus egy n csomópontot kiterjesztésre kiválasszon:

$$\textit{összköltség}(n) \leq \textit{összköltség}^*(s).$$

Tehát az

$$S = \{n \mid \textit{összköltség}(n) > \textit{összköltség}^*(s)\}$$

csomópontok nem kerülnek soha kiterjesztésre, nem is kell őket a keresőfában őrizni. Nem ismerjük ugyanakkor az $\textit{összköltség}^*(s)$ értéket. Becsüljük meg: legyen $K \geq \textit{összköltség}^*(s)$. Ekkor az

$$S' = \{n \mid \textit{összköltség}(n) > K\} \subseteq S$$

csomópontokat a keresőfából elhagyhatjuk.
Annak elegendő feltétele, hogy az A^* algoritmus egy n csomópontot kiterjesztésre kiválasszon:

$$\textit{összköltség}(n) < \textit{összköltség}^*(s).$$

2.9. DEFINÍCIÓ. Legyen P és Q két A^* algoritmus! Azt mondjuk, hogy P *jobban informált*, mint Q, ha célállapotot tartalmazó csomópontok kivételével bármely n csomópontra

$$heurisztika_P(n) > heurisztika_Q(n)$$

teljesül, ahol $heurisztika_P$ és $heurisztika_Q$ a P és Q algoritmusok heurisztikus függvényei. (Más szóval: a P algoritmus alulról pontosabban becsli a hátralévő út költségét bármely csúcsban.)

2.10. TÉTEL. *Ha P jobban informált A^* algoritmus Q-nál, akkor minden olyan csomópontot, amelyet P kiterjeszt, kiterjeszt Q is.*

BIZONYÍTÁS. Legyen n egy olyan nyílt csomópont, melyet P éppen kiválaszt kiterjesztésre! Ekkor a 2.8. lemma értelmében

$$\mathit{összköltség}_P(n) \leq \mathit{összköltség}^*(s).$$

A P keresőgráfjában az s-ből n-be vezető út tetszőleges n' csúcsára szintén teljesül az

$$\mathit{összköltség}_P(n') \leq \mathit{összköltség}^*(s)$$

összefüggés. Mit csinál ezzel az n' csúccsal a Q algoritmus? Mivel

$$\mathit{összköltség}_Q(n') = \mathit{útköltség}(n') + \mathit{heurisztika}_Q(n') <$$
$$< \mathit{útköltség}(n') + \mathit{heurisztika}_P(n') = \mathit{összköltség}_P(n') \leq \mathit{összköltség}^*(s),$$

azaz $\mathit{összköltség}_Q(n') < \mathit{összköltség}^*(s)$, ezért n'-t a Q algoritmus is kiterjeszti. Az n' tetszőleges volt, így a Q algoritmus az s-ből n-be vezető út minden csúcsát kiterjeszti, beleértve n-et is. □

A monoton A algoritmus

2.11. DEFINÍCIÓ. Azt mondjuk, hogy egy h *heurisztikus függvény* kielégíti a *monoton* megszorítás feltételét, ha értéke bármely él mentén legföljebb az illető él költségével csökken, azaz minden $(n, m) \in E$ él esetén

$$h(n) - h(m) \leq \textit{költség}(n, m).$$

2.12. TÉTEL. *Ha egy heurisztikus függvény kielégíti a monoton megszorítás feltételét, akkor*

$$h(n) \leq h^*(n)$$

teljesül minden $n \in N$-re.

BIZONYÍTÁS. A bizonyítás két részből áll:

1. Ha az n csúcsból nem vezet út terminálisba, akkor $h^*(n) = \infty$.

2. Ha van ilyen út, akkor legyen $n = n_0, n_1, n_2, \ldots, n_\ell = t \in T$ optimális út. Ennek éleire

$$\begin{aligned} h(n) - h(n_1) &\leq \mathit{költség}(n, n_1) \\ h(n_1) - h(n_2) &\leq \mathit{költség}(n_1, n_2) \\ &\vdots \\ h(n_{\ell-1}) - h(t) &\leq \mathit{költség}(n_{\ell-1}, t) \end{aligned}$$

teljesül. Az egyenlőtlenségeket összeadva

$$h(n) - h(t) \leq \sum_{i=1}^{\ell} \mathit{költség}(n_{i-1}, n_i) = h^*(n)$$

adódik, ahol a bal oldal $h(n)$, mivel $h(t) = 0$, lévén t terminális csúcs. Így $h(n) \leq h^*(n)$.

□

2.13. DEFINÍCIÓ. *Monoton A* algoritmusnak nevezzük azt az A algoritmust, amelynek heurisztikus függvénye monoton megszorításos.

2.14. TÉTEL. *Amikor a monoton A algoritmus egy nyílt n csomópontot kiterjesztésre kiválaszt, akkor n-be már optimális utat talált, azaz* útköltség(n) = útköltség$^*(n)$.

BIZONYÍTÁS. Tegyük föl indirekt módon, hogy amikor az n csúcsot kiterjesztésre kiválasztja az algoritmus, *útköltség*$(n) >$ *útköltség*$^*(n)$. Legyen m egy olyan nyílt csúcs, amely egy s-ből n-be vezető optimális úton van és amelyre *útköltség*$(m) =$ *útköltség*$^*(m)$ teljesül. Az indirekt föltevés miatt n és m nem lehet ugyanaz a csúcs. Mivel azonban az algoritmus n-et választotta m helyett, ez azt jelenti, hogy *összköltség*$(n) \leq$ *összköltség*(m).

Ugyanakkor az m-ből n-be vezető
$m = m_0, m_1, m_2, \ldots, m_{\ell-1}, m_\ell = n$ optimális útvonalra felírható
a következő összefüggés:

$$\begin{aligned}
\textit{összköltség}(m) &= \textit{útköltség}(m) + \textit{heurisztika}(m) = \\
&= \textit{útköltség}^*(m) + \textit{heurisztika}(m) \leq \\
&\leq \textit{útköltség}^*(m) + \textit{költség}(m, m_1) + \textit{heurisztika}(m_1) = \\
&= \textit{útköltség}^*(m_1) + \textit{heurisztika}(m_1) \leq \\
&\leq \textit{útköltség}^*(m_1) + \textit{költség}(m_1, m_2) + \textit{heurisztika}(m_2) = \\
&= \textit{útköltség}^*(m_2) + \textit{heurisztika}(m_2) \leq \ldots \leq \\
&\leq \textit{útköltség}^*(m_{\ell-1}) + \textit{költség}(m_{\ell-1}, n) + \textit{heurisztika}(n) = \\
&= \textit{útköltség}^*(n) + \textit{heurisztika}(n) < \\
&< \textit{útköltség}(n) + \textit{heurisztika}(n) = \textit{összköltség}(n).
\end{aligned}$$

A levezetésből azt kaptuk, hogy $\textit{összköltség}(m) < \textit{összköltség}(n)$,
ami ellentmond annak, hogy az n csomópontot választjuk ki. $\square$

3. fejezet

Kétszemélyes stratégiai játékok és lépésajánló algoritmusok

Stratégiai játékok azok a játékok, melyekben játékosoknak a játék kimenetelére (ellenőrizhető módon) van befolyásuk. Ilyen játékok pl. a sakk, a bridzs, a póker, az üzleti „játékok” mint két vállalat konkurrencia harca, harci „játékok”.
1921 E. Borel
1928 Neumann János
1944 Neumann János és O. Morgenstein
1994 Harsányi János (közgazdasági Nobel-díj)

Egy játék leírásához meg kell adni

- a játék lehetséges állásait (helyzeteit),
- a játékosok számát,
- hogyan következnek lépni az egyes játékosok (pl. egy időben vagy felváltva egymás után),
- egy-egy állásban a játékosoknak milyen lehetséges lépései (lehetőségei) vannak,
- a játékosok milyen – a játékkal kapcsolatos – információval rendelkeznek a játék folyamán,
- van-e a véletlennek szerepe a játékban és hol,
- milyen állásban kezdődik és mikor ér véget a játék,
- és az egyes játékosok mikor, mennyit nyernek, illetve veszítenek.

Osztályozás

- a játékosok száma szerint: pl. *kétszemélyes játékok*;
- a játszma állásból állásba vivő lépések sorozata: *diszkrét játékok*;
- az állásokban véges sok lehetséges lépése van minden játékosnak és a játszmák véges sok lépés után véget érnek: *véges játékok*;
- a játékosok a játékkal kapcsolatos összes információval rendelkeznek a játék folyamán: *teljes információjú játékok*;
- nincs a véletlennek szerepe a játékban: *determinisztikus játékok*;
- a játékosok nyereségeinek és veszteségeinek összege 0: *zérusösszegű játékok*.

A továbbiakban játék alatt kétszemélyes, diszkrét, véges, teljes információjú, determinisztikus, zérusösszegű stratégiai játékot fogunk érteni.

3.1. A játékok reprezentációja

Jelölje a két játékost $\mathtt{A}$ és $\mathtt{B}$, a játékállások halmazát H. A játékot az $a_0 \in H$ kezdőállásban kezdje $\mathtt{J}_0 \in \{\mathtt{A},\mathtt{B}\}$. Tegyük fel, hogy a játékosok a játék során felváltva lépnek, és ismerjük az egyes állásokban megtehető lépéseket: $\{l \mid l : H \to H\}$. Az l lépés egy a állásban akkor tehető meg, ha *l-megtételének-előfeltétele*(a). A játék az a állásban véget ér, ha *végállás*(a). A szabályok leírják, itt ki a nyerő játékos: $\mathit{nyer} : \{a \mid \mathit{végállás}(a)\} \to \{\mathtt{jön}_a, \mathtt{volt}_a\}$, ahol $\mathit{jön}_a$ lenne az a állásban soron következő játékos ($\mathtt{jön}_a \neq \mathtt{volt}_a$). E játék állapottér-reprezentációja az az $\langle \mathcal{A}, \mathit{kezdő}, \mathcal{V}, \mathcal{O} \rangle$ négyes, ahol

- $\mathcal{A} = \{(a, \mathtt{J}) \mid a \in H, \mathtt{J} \in \{\mathtt{A},\mathtt{B}\}, \mathtt{J}$ következik lépni$\}$,
- $\mathit{kezdő} = (a_0, \mathtt{J}_0)$
- $\mathcal{V} = \{(a, \mathtt{J}) \mid \mathit{végállás}(a), \mathtt{J}$ nyer, ha $\mathit{nyer}(a) = \mathtt{jön}_a\}$
- $\mathcal{O} = \{o_l \mid o_l(a, \mathtt{J}) = (l(a), \mathtt{I}), \mathtt{I}, \mathtt{J} \in \{\mathtt{A},\mathtt{B}\}, \mathtt{I} \neq \mathtt{J}\}$

A játék állapottér-repezentációját szemléltető gráf a **játékgráf**. „Egyenesítsük ki” a játékgráfot fává. A **játékfában**

- páros szinteken lévő állásokban a kezdő játékos, páratlan szinteken lévőkben pedig az ellenfele léphet;
- egy állást annyi különböző csúcs szemléltet, ahány különböző módon a játék során a kezdőállásból eljuthatunk hozzá;
- véges hosszúságúak az utak, hisz véges játékokkal foglalkozunk.

Ha a játék során *kezdő* állapotból a játékosok valamelyik $v \in \mathcal{V}$ állapotba érnek, azaz $\textit{kezdő} \overset{*}{\underset{o_1,\ldots,o_r}{\Rightarrow}} v \quad (r \geq 1)$, azt mondjuk lejátszottak egy **játszmát**. A játszmákat a játékfában a startcsúcsból a levélelemekbe vezető utak szemléltetik.

Egy játék játékfája a játék összes lehetséges játszmáját szemlélteti a startcsúcsból induló, a különböző levelekben végződő útjaival.

3.1. DEFINÍCIÓ. Az $\langle N, HE \rangle$ párt **ÉS/VAGY gráfnak** nevezzük:

- N nemüres halmaz, a gráf csúcsainak halmaza,
- $HE \subseteq \left\{(n, M) \in N \times 2^N \mid 0 \neq |M| < \infty\right\}$ pedig az irányított **hiperélek** halmaza.

3.2. DEFINÍCIÓ. Az ÉS/VAGY gráfban a gráf hiperéleinek egy olyan

$$\begin{gathered}(n_1, \left\{n_{11}, n_{12}, \ldots, n_{1k_1}\right\}), \\ (n_2, \left\{n_{21}, n_{22}, \ldots, n_{2k_2}\right\}), \\ \vdots \\ (n_r, \left\{n_{r1}, n_{r2}, \ldots, n_{rk_r}\right\})\end{gathered}$$

sorozata, ahol

$$\forall i \forall j \left(\neg(i = j) \supset \neg(n_i = n_j)\right) \wedge$$
$$\wedge \forall i \left((i > 1) \supset \exists j \left((i > j) \wedge (n_i \in \left\{n_{j1}, n_{j2}, \ldots, n_{jk_j}\right\})\right)\right),$$

a gráf egy **hiperútja**.

3.2. A stratégia

3.3. DEFINÍCIÓ. A J játékos **stratégiája** egy olyan

$$\mathcal{S}_{\mathtt{J}} : \{(a, \mathtt{J}) \mid (a, \mathtt{J}) \in \mathcal{A}\} \to \mathcal{O}$$

döntési terv, amely J számára előírja, hogy a játék során előforduló azon állásokban, melyekben J következik lépni, a megtehető lépései közül melyiket lépje meg.

A J játékos stratégiáinak szemléltetése a játékfában

Alakítsuk át a játékfát ÉS/VAGY fává J játékos szempontjából: J lépéseit szemléltető élek mindegyike egy élből álló hiperél marad (VAGY élek), ellenfelének egy-egy állásból megtehető lépéseit szemléltető élköteg egy-egy hiperél lesz (ÉS élek). Ebben az ÉS/VAGY gráfban J stratégiáit a startcsúcsból kinduló olyan hiperutak szemléltethetik, melyek levelei az eredeti játékgráfnak is levelei.

3.4. LEMMA. *Tegyük fel, hogy a J játékos az $\mathcal{S}_J$ stratégiájával játszik. Ekkor csak az $\mathcal{S}_J$-t szemléltető hiperutat alkotó közönséges utak által szemléltetett játszmák játszhatók le.*

3.5. LEMMA. *Tegyük fel, hogy az A játékos az $\mathcal{S}_A$, a B játékos pedig az $\mathcal{S}_B$ stratégiájával játszik. A két stratégia egyértelműen meghatározza a lejátszható játszmát.*

3.6. DEFINÍCIÓ. A J játékos stratégiáját J **nyerő stratégiájának** nevezzük, ha (az ellenfelének stratégia-választásától függetlenül) minden a stratégia alkalmazása mellett lejátszható játszmában J nyer.

3.7. MEGJEGYZÉS. A J szempontjából átalakított ÉS/VAGY fában a J nyerő stratégiát szemléltető hiperút levélelemei mind J-nyerő állások.

3.8. TÉTEL. *(Az általunk vizsgált) minden játék esetén valamelyik (de nyilván csak az egyik) játékos számára van nyerő stratégia.*

BIZONYÍTÁS. Rögzítsünk tetszőlegesen egy játékot, és tegyük fel, hogy adott a teljes játékfa.

- Ekkor a fa leveleit címkézzük `A`-val, ha a levél olyan végállást szemléltet, ahol `A` nyer, illetve `B`-vel, ha a levél olyan végállást szemléltet, ahol `B` nyer.
- Címkézzük szintenként csökkenő sorrendben a nem levél csúcsokat is: ha a csúcsban $\mathtt{J} \in \{\mathtt{A},\mathtt{B}\}$ következik lépni és van `J` címkéjű gyermeke, `J` címkét kap, egyébként az ellenfél címkéjét.

Mivel a játékfa véges, végül a startcsúcs is címkét kap. A teljes indukció elve alapján ez a címke egyértelmű. A startcsúcs címkéje pedig megmutatja, melyik játékosnak van nyerő stratégiája, és magát a nyerő stratégiá(ka)t is le lehet olvasni a felcímkézett játékfáról.

3.3. Minimax algoritmus

Cél: a támogatott játékosnak, J-nek, egy adott állásban „elég jó" lépést ajánlani.
Az algoritmus számára át kell adni

- a játék $\langle \mathcal{A}, \mathit{kezdő}, \mathcal{V}, \mathcal{O} \rangle$ reprezentációját,
- J azon a állását, ahol lépni következik,
- az állások „jóságát" J szempontjából becslő $h_{\mathtt{J}} : \mathcal{A} \to R$ heurisztikát
- és egy mélységi *korlát*-ot.

Az algoritmus fő lépései:

1. A játékfa $(a, \mathtt{J})$ állapotot szemléltető csúcsából kiinduló részének előállítása *korlát* mélységig.
2. A részfa leveleiben található állások jóságainak becslése a heurisztika segítségével: $jóság(n_b) = h_{\mathtt{J}}(b)$.
3. Szintenként csökkenő sorrendben a részfa nem levél csúcsai jóságainak számítása: ha az n csúcs gyermekei rendre $n_1, \ldots, n_k$, akkor

$$jóság(n) \rightleftharpoons \begin{cases} \max\{jóság(n_1), \ldots, jóság(n_k)\}, & \text{ha } n \text{ szintje páros,} \\ \min\{jóság(n_1), \ldots, jóság(n_k)\}, & \text{ha } n \text{ szintje páratlan.} \end{cases}$$

Javaslat: az a állásból egy olyan lépést tegyen meg J, amelyik az n_a csúcs „jóság” értékével megegyező értékű gyermekébe vezet.

```
1: function MINIMAX-LÉPÉS(⟨A, kezdő, V, O⟩, állapot, korlát, h_J)
2:     max ← −∞
3:     operátor ← NIL
4:     for all o ∈ O do
5:         if ELŐFELTÉTEL(állapot, o) then
6:             új-állapot ← ALKALMAZ(állapot, o)
7:             v ← MINIMAX-ÉRTÉK(⟨A, kezdő, V, O⟩, új-állapot, korlát − 1, h)
8:             if v > max then
9:                 max ← v
10:                operátor ← o
11:            end if
12:        end if
13:    end for
14:    return operátor
15: end function
```

```
16: function MINIMAX-ÉRTÉK(⟨A, kezdő, V, O⟩, állapot, mélység, h_J)
17:     if állapot ∈ V or mélység = 0 then
18:         return h_J(állapot)
19:     else if JÁTÉKOS[állapot] = J then
20:         max ← −∞
21:         for all o ∈ O do
22:             if ELŐFELTÉTEL(állapot, o) then
23:                 új-állapot ← ALKALMAZ(állapot, o)
24:                 v ← MINIMAX-ÉRTÉK(⟨A, kezdő, V, O⟩, új-állapot, mélység − 1, h_J)
25:                 if v > max then
26:                     max ← v
27:                 end if
28:             end if
29:         end for
30:         return max
```

```
    else
        min ← ∞
        for all o ∈ 𝒪 do
            if Előfeltétel(állapot, o) then
                új-állapot ← Alkalmaz(állapot, o)
                v ← Minimax-Érték(⟨𝒜, kezdő, 𝒱, 𝒪⟩, új-állapot, mélység − 1, h_J)
                if v < min then
                    min ← v
                end if
            end if
        end for
        return min
    end if
end function
```

3.4. Negamax algoritmus

Cél: a támogatott játékosnak, J-nek, egy adott állásban „elég jó" lépést ajánlani.
Az algoritmus számára át kell adni

- a játék $\langle \mathcal{A}, \mathit{kezdő}, \mathcal{V}, \mathcal{O} \rangle$ reprezentációját,
- J azon a állását, ahol lépni következik,
- az állások „jóságát" a soron következő játékos szempontjából becslő $h : \mathcal{A} \to R$ heurisztikát
- és egy mélységi *korlát*-ot.

Az algoritmus fő lépései:

1. A játékfa $(a, \mathtt{J})$ állapotot szemléltető csúcsából kiinduló részének előállítása *korlát* mélységig.
2. A részfa leveleiben található állások jóságainak becslése a heurisztika segítségével: $jóság(n_b) = h(b)$.
3. Szintenként csökkenő sorrendben a részfa nem levél csúcsai jóságainak számítása: ha az n csúcs gyermekei rendre $n_1, \ldots, n_k$, akkor

$$jóság(n) \rightleftharpoons \max\{-jóság(n_1), \ldots, -jóság(n_k)\},$$

Javaslat: az a állásból egy olyan lépést tegyen meg $\mathtt{J}$, amelyik az n_a csúcs „jóság" értékének -1-szeresével megegyező értékű gyermekébe vezet.

```
1:  function NEGAMAX-LÉPÉS(⟨A, kezdő, V, O⟩, állapot, korlát, h)
2:      max ← −∞
3:      operátor ← NIL
4:      for all o ∈ O do
5:          if ELŐFELTÉTEL(állapot, o) then
6:              új-állapot ← ALKALMAZ(állapot, o)
7:              v ← −NEGAMAX-ÉRTÉK(⟨A, kezdő, V, O⟩, új-állapot, korlát − 1, h)
8:              if v > max then
9:                  max ← v
10:                 operátor ← o
11:             end if
12:         end if
13:     end for
14:     return operátor
15: end function
```

```
16: function NEGAMAX-ÉRTÉK(⟨A, kezdő, V, O⟩, állapot, mélység, h)
17:     if állapot ∈ V or mélység = 0 then
18:         return h(állapot)
19:     else
20:         max ← −∞
21:         for all o ∈ O do
22:             if ELŐFELTÉTEL(állapot, o) then
23:                 új-állapot ← ALKALMAZ(állapot, o)
24:                 v ← −NEGAMAX-ÉRTÉK(⟨A, kezdő, V, O⟩, új-állapot, mélység − 1, h)
25:                 if v > max then
26:                     max ← v
27:                 end if
28:             end if
29:         end for
30:         return max
31:     end if
32: end function
```

4. fejezet

Problémamegoldás redukcióval

Gyakran előfordul, hogy egy problémát úgy próbálunk megoldani, hogy több külön-külön megoldandó részproblémára bontjuk. Ha a részproblémákat megoldjuk, az eredeti probléma megoldását is megkapjuk. A részproblémák megoldását további részek megoldására vezetjük vissza, egészen addig, amíg csupa olyan problémához nem jutunk, amelyeket egyszerűségüknél fogva már könnyedén meg tudunk oldani. A probléma megoldásnak ezt a módját

problémaredukciónak

nevezzük.

4.1. Problémaredukciós reprezentáció

- Először is le kell írni az eredeti problémát, jelöljük ezt most p-vel.
- Egy probléma részproblémákra bontása során a nyert részek az eredeti problémához hasonló, de annál egyszerűbb problémák. Jelöljük az így nyert problémahalmazt $\mathcal{P}$-vel.
  Természetesen $p \in \mathcal{P}$.
- $\mathcal{P}$ problémáinak összegyűjtése során törekszünk arra, hogy legyenek közöttük olyanok, melyeket meg tudunk oldani, vagy ismerjük a megoldásukat. Ezek a problémák az ún. **egyszerű problémák**. Az egyszerű problémák halmazát $\mathcal{E}$-vel jelöljük.
  $\mathcal{E} \subset \mathcal{P}$, hiszen $p \notin \mathcal{E}$, különben nincs megoldandó feladat.

- Meg kell még adni a problémákat egyszerűsítő, illetve részekre bontó **redukciós operátorokat**. Egy redukciós operátor egy problémához azokat a (rész)problémákat rendeli hozzá, melyek egyenkénti megoldásával a probléma megoldása is előáll. Jelöljön a **redukciós operátorok** $\mathcal{R}$ véges **halmazából** r egy operátort. Ekkor

$$\mathrm{Dom}(r) = \{\, q \mid q \in \mathcal{P} \setminus \mathcal{E} \text{ és } r\text{-}\textit{alkalmazásának-előfeltétele}(q) \,\}$$

és

$$\mathrm{Rng}(r) = \{\, r(q) = \{q_1, \ldots, q_m\} \mid q \in \mathrm{Dom}(r) \text{ és } q_1, \ldots, q_m \in \mathcal{P}\}.$$

Tehát egy redukciós operátor egy-egy problémához $\mathcal{P}$ egy-egy részhalmazát rendeli, így értékkészlete $\mathcal{P}$ hatványhalmazának valamely részhalmaza.

4.1. DEFINÍCIÓ. Legyen p egy probléma. Azt mondjuk, hogy a p problémát **problémaredukciós reprezentációval** írtuk le, ha megadtuk a $\langle \mathcal{P}, p, \mathcal{E}, \mathcal{R} \rangle$ négyest, azaz

- a megoldandó $p \in \mathcal{P}$ problémát,
- a $\mathcal{P} \neq \emptyset$ halmazt, a p problémához hasonló problémák halmazát,
- az egyszerű problémák $\mathcal{E} \subset \mathcal{P}$ halmazát és
- a redukciós operátorok $\mathcal{R} \neq \emptyset$ véges halmazát.

Jelölése: $\langle \mathcal{P}, p, \mathcal{E}, \mathcal{R} \rangle$.

4.2. DEFINÍCIÓ. Legyen a p probléma a $\langle \mathcal{P}, p, \mathcal{E}, \mathcal{R} \rangle$ reprezentációval leírva és legyenek

$$Q = \{p_1, p_2, \ldots, p_i, \ldots, p_n\} \subseteq \mathcal{P}$$

$$Q' = \{p_1, p_2, \ldots, p_{i-1}, q_1, q_2, \ldots, q_m, p_{i+1}, \ldots, p_n\} \subseteq \mathcal{P}$$

egy-egy problémahalmaz ($n \geq 1, m \geq 1$). Azt mondjuk, hogy a Q problémahalmaz **egy lépésben** vagy **közvetlenül redukálható** a Q' problémahalmazzá, ha van olyan $r \in \mathcal{R}$ redukciós operátor, melyre $p_i \in \mathrm{Dom}(r)$, és

$$r(p_i) = \{q_1, q_2, \ldots, q_m\}.$$

Ennek jelölése: $Q \quad \lhd \quad Q'$, illetve ha fontos, hogy az r redukciós operátor segítségével állítottul elő Q-ból a Q'-t, akkor $Q \quad \lhd_r \quad Q'$.

4.3. DEFINÍCIÓ. Legyen a p probléma reprezentációja $\langle \mathcal{P}, p, \mathcal{E}, \mathcal{R} \rangle$, és $Q, Q' \subseteq \mathcal{P}$. A Q-ból a Q' **redukálható**, ha van olyan $P_1, \ldots, P_k \subseteq \mathcal{P} \quad (k \geq 2)$ véges problémahalmaz-sorozat, hogy

$$P_1 = Q, \quad P_k = Q'$$

$$\text{és } P_i \quad \lhd \quad P_{i+1} \text{ minden } 1 \leq i \leq k-1 \text{ esetén.}$$

Jelölése: $Q \quad \lhd^* \quad Q'$.

4.4. MEGJEGYZÉS. Nyilvánvaló, hogy ha $P_i \quad \lhd \quad P_{i+1}$ minden $1 \leq i \leq k-1$ esetén, akkor van olyan $r_1, r_2, \ldots, r_{k-1}$ redukciós operátorsorozat, hogy $P_i \quad \lhd_{r_i} \quad P_{i+1} \quad (1 \leq i \leq k-1)$. Ilyenkor azt mondjuk, hogy a Q problémahalmazt a Q' problémahalmazzá az $r_1, r_2, \ldots, r_{k-1}$ redukciós operátorsorozat segítségével redukáltuk. Jelölve: $Q \quad \lhd^*_{r_1,\ldots,r_{k-1}} \quad Q'$.

4.5. DEFINÍCIÓ. Legyen a p probléma problémaredukciós reprezentációja $\langle \mathcal{P}, p, \mathcal{E}, \mathcal{R} \rangle$. A p probléma **megoldható** ebben a reprezentációban, ha $\{p\}$ csupa egyszerű problémából álló problémahalmazzá redukálható , azaz

$$\{p\} \quad \lhd^*_{r_1,\ldots,r_l} \quad Q \subseteq \mathcal{E}.$$

Ekkor az $r_1, \ldots, r_l$ redukciós operátorsorozatot tekinthetjük a **probléma megoldásának**.

A feladatunk lehet

- annak eldöntése, hogy megoldható-e a probléma az adott problémaredukciós reprezentációban,
- egy (esetleg az összes) megoldás előállítása,
- valamilyen minősítés alapján jó megoldás előállítása (a megoldások között különbséget tehetünk, pl. a megoldás költsége alapján).

Jelölje $költség(r, q) \geq 0$ az r **redukciós operátor** q problémára való **alkalmazásának a költségét**, és ha $q \in \mathcal{E}$, akkor $c(q) \geq 0$ pedig a q egyszerű probléma közvetlen megoldásának költségét.

4.6. DEFINÍCIÓ. A $\langle \mathcal{P}, p, \mathcal{E}, \mathcal{R} \rangle$ problémaredukciós reprezentációban $g(q)$ a $q \in \mathcal{P}$ probléma **megoldásának minimális költsége**, ha

$$g(q) = \begin{cases} c(q) & \text{ha } q \in \mathcal{E}, \\ \infty & \text{ha } q \notin \mathcal{E}, \text{ de} \\ & \neg\exists r(r \in \mathcal{R} \wedge q \in \mathrm{Dom}(r)), \\ \min\{költség(r, q) + \sum_{q_i \in r(q)} g(q_i)\} & \text{egyébként.} \end{cases}$$

A részproblémák párhuzamos megoldása esetén lehetőségünk van a legrövidebb idő alatt előállítható megoldás megkeresésére. Ekkor *költség*(r, q) az r **redukciós operátor** q problémára való **alkalmazásának a végrehajtási idejét**, $c(q)$ pedig a q egyszerű probléma közvetlen megoldásának idejét jelenti.

4.7. DEFINÍCIÓ. A $\langle \mathcal{P}, p, \mathcal{E}, \mathcal{R} \rangle$ problémaredukciós reprezentációban $t(q)$ a $q \in \mathcal{P}$ probléma **megoldásának minimális ideje**, ha

$$t(q) = \begin{cases} c(q) & \text{ha } q \in \mathcal{E}, \\ \infty & \text{ha } q \notin \mathcal{E}, \text{ de} \\ & \neg\exists r(r \in \mathcal{R} \wedge q \in \mathrm{Dom}(r)), \\ \min\{\mathit{költség}(r, q) + \max_{q_i \in r(q)} t(q_i)\} & \text{egyébként.} \end{cases}$$

4.2. Példák problémaredukciós reprezentációra

4.2.1. Hanoi tornyai

A legenda szerint egy szerzetesek lakta távol-keleti kolostor udvarán áll három rúd, amelyeken 64 különböző átmérőjű aranykorong található. Eredetileg mind a 64 korong egyetlen rúdra volt rárakva úgy, hogy minden korong alatt egy nála nagyobb volt. A szerzeteseknek az a feladatuk, hogy a korongokat helyezzék át az első rúdról a harmadik rúdra, egyszerre mindig csak egyet mozgatva úgy, hogy sohase rakjanak nagyobb korongot kisebbre. Amint mind a 64 korongot átpakolják a harmadik rúdra, eljön majd a világvége.

A legenda szerint a szerzetesek a munka elvégzésével a legidősebb társukat bízták meg. Sokat törte a fejét, gondolkodott, meditált, majd hirtelen világosság töltötte el: a feladatot három lépésben meg tudja oldani!

1. lépés: Át kell vinni az első rúdon lévő felső 63 korongból álló tornyot a második rúdra.
2. lépés: Át kell vinni az első rúdon lévő utolsó, legnagyobb korongot a harmadik rúdra.
3. lépés: Át kell vinni a második rúdon lévő 63 korongból álló tornyot a harmadik rúdra.

A szerzetes másnap kiszögezte a templom kapujára az algoritmus leírását:

Módszer és út arra vonatkozóan hogy hogyan vigyünk át egy n korongból álló tornyot az x rúdról az y-ra a z felhasználásával:

1. *Abban az esetben, ha a torony egynél több korongból áll, bízd meg a legöregebb tanítványodat, hogy a szóban forgó torony felső $n-1$ korongját vigye át az x rúdról a z-re, miközben az y-t használhatja.*
2. *Vidd át magad az x rúdon maradt egyetlen korongot az y-ra.*
3. *Abban az esetben, ha a torony egynél több korongból állt, bízd meg a legöregebb tanítványodat, hogy a szóban forgó torony felső $n-1$ korongját vigye át a z rúdról az y-ra, miközben az x-t használhatja.*

A legenda szerint tehát a hanoi szerzetesek problémaredukcióval próbálták megoldani az előttük álló feladatot. Adjuk meg most az elképzelésüknek megfelelő reprezentációt a módosított feladatra.

A megoldandó p feladat tehát: mindhárom az A rúdon levő korong átvitele a C rúdra (B felhasználásával). Ezt jelölhetjük a következőképpen:

$$p = (3;\ A \to C)$$

A megoldandó feladathoz hasonló feladatok a következők: az x rúdon levő felső valahány korong átvitele a y rúdra (z felhasználásával):

$$\mathcal{P} = \{(n;\ x \to y) \mid n \geq 1, x, y \in \{A, B, C\}, x \neq y\}$$

Ezek közül egyszerűen megoldhatók, ha a felső korongot kell áthelyezni az x rúdról egy olyan rúdra, amelyiken nincs ennél kisebb átmérőjű:

$$\mathcal{E} = \{(1;\ x \to y) \mid x, y \in \{A, B, C\}, x \neq y \\ x\text{-en van korong, } y\text{-on nincs ennél kisebb átmérőjű}\}$$

Minden nem egyszerű problémát három, az eredetinél egyszerűbb részre bonthatunk:

$$\mathcal{R} = \{r(n;\ x \to y) = \begin{cases} (m;\ x \to z) \\ (n-m;\ x \to y) \ \mid \\ (m;\ z \to y) \end{cases} \\ \mid 1 \leq m \leq n-1, x, y, z \in \{A, B, C\}, x \neq y, x \neq z, y \neq z\}$$

4.3. A problémaredukciós reprezentációt szemléltető gráf

Legyen a p probléma a $\langle \mathcal{P}, p, \mathcal{E}, \mathcal{R} \rangle$ reprezentációval megadva. Ez a reprezentáció is egy irányított gráfot, ún. **ÉS/VAGY gráfot** határoz meg.

- A $\mathcal{P}$ problémahalmaz elemei (a problémák) a gráf **csúcsai**. Vezessük be az $q \in \mathcal{P}$ probléma által definiált csúcsra az n_q jelölést. Ekkor a gráf csúcsainak halmaza

$$N = \{n_q \mid q \in \mathcal{P}\}.$$

- A gráf csúcsai közül kitüntetett szerepet játszanak a p problémát szemléltető ún. **startcsúcs** (jele: n_p vagy s)
- és az egyszerű problémákat szemléltető **terminális csúcsok**. A terminális csúcsok halmaza tehát:

$$T = \{n_e \mid e \in \mathcal{E}\}.$$

- Egy $q \in \mathcal{P}$ problémát szemléltető csúcsból **irányított éleket** húzunk az $q_1, \ldots, q_m \in \mathcal{P}$ problémákat szemléltető $n_{q_1}, \ldots, n_{q_m}$ csúcsokba, amikor $\{q\} \quad \lhd \quad \{q_1, q_2, \ldots, q_m\}$. Ezek az élek összetartozónak tekinthetők: egy **ÉS élköteget** vagy **hiperélt** alkotnak. A gráf hiperéleinek halmaza tehát a következő:

$$E = \{(n_q, \{n_{q_1}, n_{q_2}, \ldots, n_{q_m}\}) \mid q, q_1, q_2, \ldots, q_m \in \mathcal{P} \text{ és } \{q\} \quad \lhd \quad \{n_{q_1}, n_{q_2}, \ldots, n_{q_m}\}\}.$$

Azt mondjuk, hogy az $\langle N, s, T, E\rangle$ irányított ÉS/VAGY gráf a p probléma $\langle \mathcal{P}, p, \mathcal{E}, \mathcal{R}\rangle$ problémaredukciós reprezentációjához tartozó **reprezentációs gráfja**.

4.8. LEMMA. *Legyen* $\langle N, s, T, E\rangle$ *a* p *probléma* $\langle \mathcal{P}, p, \mathcal{E}, \mathcal{R}\rangle$ *problémaredukciós reprezentációjához tartozó reprezentációs gráfja. Pontosan akkor áll fenn a* $\{q\} \quad \lhd^* \quad \{q_1, q_2, \ldots, q_m\}$ *reláció, ha a reprezentációs gráfban van az* n_q *csúcsából induló olyan hiperút, melynek levelei éppen az* $\{n_{q_1}, n_{q_2}, \ldots, n_{q_m}\}$ *csúcsok.*

BIZONYÍTÁS.

1. Tegyük fel, hogy $\{q\} \quad \lhd^* \quad \{q_1, q_2, \ldots, q_m\}$. Ez a 4.3. definíció miatt azt jelenti, hogy létezik olyan $P_1, P_2, \ldots, P_k \subseteq \mathcal{P} \quad (k \geq 2)$ (véges) problémahalmaz-sorozat úgy, hogy

$$P_1 = \{q\}, \quad P_k = \{q_1, q_2, \ldots, q_m\}$$

és

$$P_i \quad \lhd \quad P_{i+1}$$

minden $1 \leq i \leq k-1$ esetén.

- Tehát minden $1 \leq i \leq k-1$-re P_i valamelyik problémája, mondjuk p_{ij} P_{i+1}-ben már részekre van bontva, azaz van olyan redukciós operátor, amelyik p_{ij}-t épp ezekre a részproblémákra bontja, így a reprezentációs gráfban a p_{ij}-t szemléltető csúcsból ÉS élköteg indul a részproblémákat szemléltető csúcsokba.
- Továbbá p_{ij} P_{i+1}-ben már nem szerepel, tehát újabb hiperél már nem indul belőle.

Azaz a reprezentációs gráfunkban egy $k-1$ hiperélből álló sorozatunk van, melyben az első hiperél a q-t szemléltető n_q csúcsból indul, minden következő hiperél kezdőcsúcsa valamely előző hiperél végcsúcsa, és minden csúcsból legfeljebb egy hiperél indul. Tehát a szemléltető részgráf egy hiperút.

Továbbá a sorozat utolsó halmazának, P_k-nak a problémái azok, amiket nem bontottunk tovább, tehát az ezeket szemléltető csúcsok a a hiperút levelei.

2. Most tegyük fel azt, hogy a reprezentációs gráf n_q csúcsából indul olyan hiperút, melynek levelei az $\{n_{q_1}, n_{q_2}, \ldots, n_{q_m}\}$ csúcsok. Ez azt jelenti, hogy van olyan

$$\begin{gathered}(n_1, \{n_{11}, n_{12}, \ldots, n_{1m_1}\}),\\ (n_2, \{n_{21}, n_{22}, \ldots, n_{2m_2}\}),\\ \vdots\\ (n_k, \{n_{k1}, n_{k2}, \ldots, n_{km_k}\})\end{gathered}$$

hiperélsorozat a reprezentációs gráfban, hogy

$$n_q = n_1,$$

továbbá

$$\forall i \forall j((1 \leq i, j \leq k) \wedge (i \neq j) \supset (n_i \neq n_j))$$

és

$$\forall i((1 < i \leq k) \supset \exists j((i > j \leq k) \wedge (n_i \in \{n_{j1}, n_{j2}, \ldots, n_{jm_j}\})).$$

A sorozat minden $(n_i, \{n_{i1}, n_{i2}, \ldots, n_{im_i}\})$ hiperéle egy redukciós operátoralkalmazást szemléltet: az n_i által szepléltetett problémát bontja a redukciós operátor az $\{n_{i1}, n_{i2}, \ldots, n_{im_i}\}$ csúcsok által szemléltetett problémákká. Tehát a hiperélsorozat egy redukciós operátorsorozat, mely első operátorát q-ra alkalmaztuk, az összes többit pedig, valamely megelőző operátor eredményeképpen előállt problémára.

4.9. TÉTEL. *Legyen $\langle N, s, T, E\rangle$ a p probléma $\langle \mathcal{P}, p, \mathcal{E}, \mathcal{R}\rangle$ problémaredukciós reprezentációjához tartozó reprezentációs gráfja. Pontosan akkor oldható meg p, ha van a reprezentációs gráfban a startcsúcsból induló olyan hiperút, melynek levelei terminális csúcsok.*

4.4. Problémaredukcióval reprezentált feladatok megoldáskereső módszerei

4.4.1. Visszalépéses megoldáskeresés ÉS/VAGY fák esetén

Legyen $\langle \mathcal{P}, p, \mathcal{E}, \mathcal{R} \rangle$a p probléma problémaredukciós reprezentációja. Egy visszalépéses megoldáskereső

- **adatbázisa** a reprezentációs gráf egy a startcsúcsból induló hiperútját tartalmazza. Ezt az utat **aktuális hiperútnak** nevezzük.
  Az adatbázis az aktuális hiperút csúcsait és e csúcsokból kiinduló bizonyos hiperéleket (explicit vagy implicit módon) nyilvántartó csomópontokból épül fel.
  A keresés megkezdésekor az adatbázis egyetlen egy - a p kezdőproblémát tartalmazó - csomópontból áll.

Egy csomópont az alábbi információkat tartalmazza:

- egy $q \in \mathcal{P}$ problémát;
- arra a csomópontra mutatót, mely a szülő problémát (azt a problémát, melyre redukciós operátort alkalmazva előállt q) tartalmazza;
- q első részproblémáját (q első ÉS gyermekét) nyilvántartó csomópontra mutatót;
- q szülőjének q-t követő részproblémáját (q következő ÉS testvérét) nyilvántartó csomópontra mutatót;
- azt a redukciós operátort, melyet q-ra aktuálisan alkalmaztunk;
- q-ra a keresés során már alkalmazott (vagy még alkalmazható) redukciós operátorok halmazát.

- A visszalépéses megoldáskeresők műveleteit egyrészt a redukciós operátorokból származtatjuk, továbbá alkalmazhatjuk a **visszalépést**.
  - Az $r \in \mathcal{R}$ redukciós operátorból nyert művelet
    - alkalmazási előfeltétele: a kiválasztott levél csomópontban található problémára alkalmazható r, de még sikertelenül nem alkalmaztuk rá.
    - hatása:
  - A visszalépés
    - alkalmazási előfeltétele: van csomópont az adatbázisban.
    - hatása:

- Az induló adatbázis létrehozása után kezdi el a vezérlő a keresést.

  – Ha elfogytak a csomópontok az adatbázisból
  $\longrightarrow$ az adott reprezentációban nincs megoldás.

  – Ha van nem egyszerű problémát tartalmazó levélcsomópont az adatbázisban, akkor a vezérlő választ egyet.

    * A kiválasztott problémára választ egy még sikertelenül ki nem próbált redukciós operátort, és alkalmazza.
    * Ha ilyen nincs, visszalép.

  – Ha a hiperút minden levél csomópontja egyszerű problémát tartalmaz
  $\longrightarrow$ előállt egy megoldás.

4.4.2. Keresőfával megoldáskeresés ÉS/VAGY fák esetén

Legyen $\langle \mathcal{P}, p, \mathcal{E}, \mathcal{R} \rangle$a p probléma problémaredukciós reprezentációja. A reprezentációs gráfot alakítsuk át olyan ÉS/VAGY gráffá, melyben minden csúcsból vagy csak VAGY élek, vagy csak egy ÉS élköteg indul ki.
Keresőfával megoldást keresés esetén az

- **adatbázis** a reprezentációs gráf startcsúcsból induló felderített hiperútjai.
  Az adatbázis a hiperutak csúcsait és e csúcsokból kiinduló hiperéleket (explicit vagy implicit módon) nyilvántartó csomópontokból épül fel.
  A keresés megkezdésekor az adatbázis egyetlen egy - a p kezdőproblémát tartalmazó - csomópontból áll.

Egy csomópont az alábbi információkat tartalmazza:

- egy $q \in \mathcal{P}$ problémát;
- ha q VAGY gyermek:
  - a szülő csomópontra mutatót;
  - azt a redukciós operátort, mellyel q-t redukáljuk;
  - q következő VAGY testvérét tartalmazó csúcsra mutatót;
  - q első ÉS gyermekét nyilvántartó csomópontra mutatót;
- ha q ÉS gyermek
  - a szülő csomópontra mutatót;
  - q következő ÉS testvérét nyilvántartó csomópontra mutatót;
  - q első VAGY gyermekét nyilvántartó csomópontra mutatót;
- címkét: *megoldott* / *megoldhatatlan* / *folyamatban*

- **művelete a kiterjesztés**: a keresőfát annak egy *folyamatban* címkéjű levélcsomópontján keresztül kibővíti.
  - alkalmazási előfeltétele: a keresőfában van *folyamatban* címkéjű levélcsomópont.
  - hatása:
    * alkalmazzuk az összes alkalmazható redukciós operátort a *folyamatban* címkéjű levélcsomópont problémájára,
    * az előálló problémákat új csomópontokként felfűzzük a keresőfába megfelelő címkékkel:
      · *megoldott*, ha az előállt probléma egyszerű;
      · *folyamatban*, ha az előállt probléma nem egyszerű;
    * módosítjuk a keresőfa csúcsainak címkéit.

- – Ha a gyökér csomópont címkéje *megoldott*, előállt az adatbázisban egy megoldás.
  – Ha a gyökér csomópont címkéje *megoldhatatlan*, nincs a reprezentáció mellett a problémának megoldása.
  – Ha a gyökér csomópont címkéje *folyamatban*, a **vezérlő** megmondja, hogy melyik *folyamatban* címkéjű levélcsomópont legyen a következő lépésben kiterjesztve.