Alkalmazott mesterséges intelligencia (AMI)

http://www.mit.bme.hu/oktatas/targyak/vimibb01

2. ea.- (2023. ősz)

Problémamegoldás kereséssel – vakon

http://mialmanach.mit.bme.hu/aima/ch03s03
3. fejezet 3.4 alfejezet

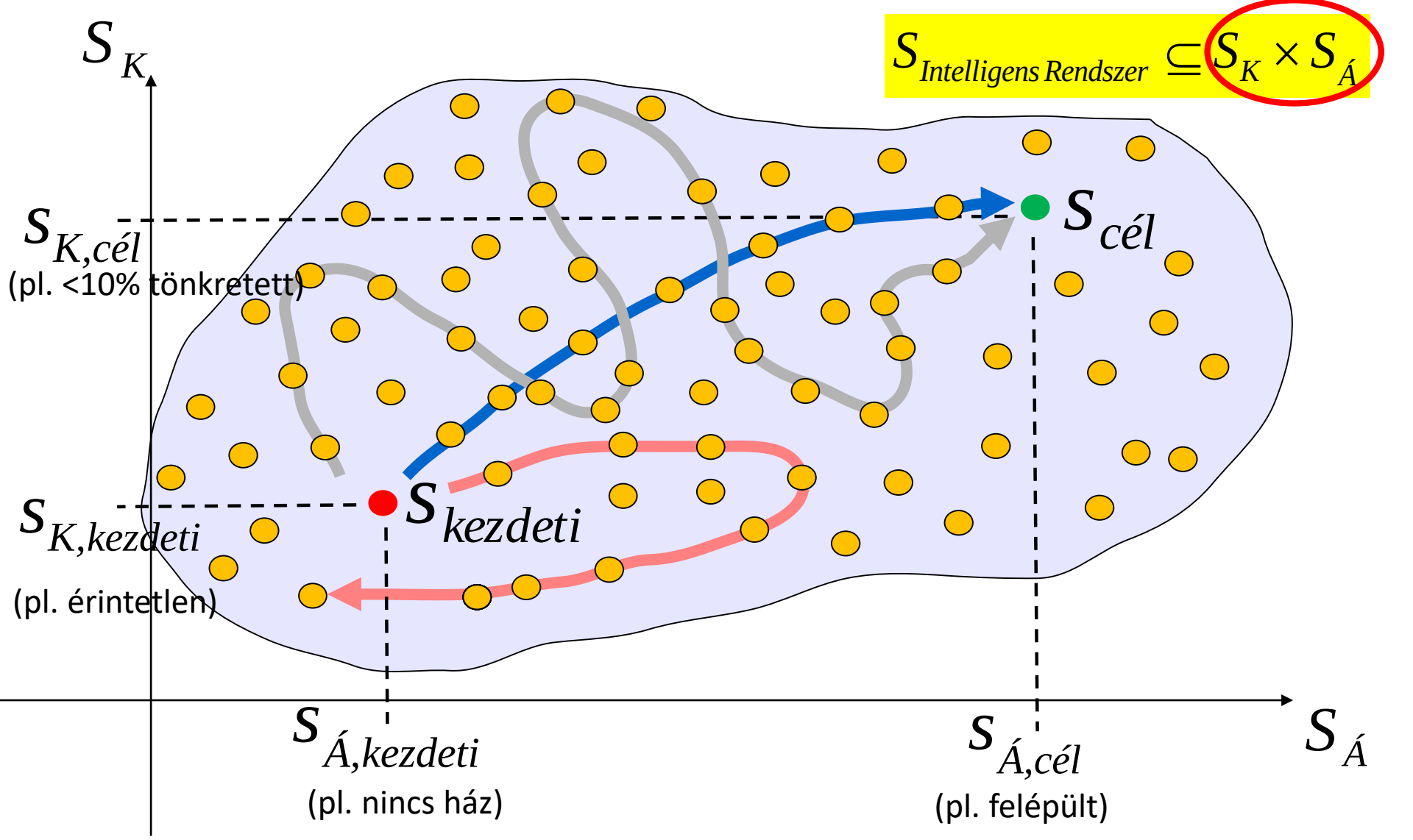
https://www.esrcheck.com/2023/06/05/artificial-intelligence-ai-experts-sign-statement-on-ai-risk/

Előadó: Pataki Béla

BME I.E. 414, 463-26-79
pataki@mit.bme.hu,
http://www.mit.bme.hu/general/staff/pataki

Egy sor gyakorlati probléma, aminek racionális (intelligens) megoldásához a most következő – keresési - módszerek segítséget nyújthatnak

- Térképen megtalálni az utat A-ból B-be (útvonaltervezés, pl. Kaszaperből Győrtelekre)
- Egy gyártás 20 technológiai lépésből áll, minden lépésben több lehetőség közül választhatunk, de a lehetőségeink függnek az előző döntéseinktől. A legolcsóbb technológiai lépéssort keressük.
- Hogyan tudjuk elérni Facebook ismerősökön és ismerősök ismerősein keresztül Joe Bident vagy Kylian Mbappét? Vagy azt a főhalljakendet, aki elintézi a méltányossági kérelmünket…. (5-6 kézfogásra vagyunk egymástól…)
- A mentést végző robot hogyan talál oda a hotel emeletének hátsó szobájába, amelyikben jelez a füstérzékelő?



Mindig nagyon fontos, hogy minél pontosabban meg tudjuk fogalmazni a problémát, azonosítani a fontos állapotjellemzőket, és látni fogjuk, hogy a megfogalmazás formája is nagyon fontos lehet!

Mindig nagyon fontos, hogy minél pontosabban meg tudjuk fogalmazni, hogy *mit tekintünk jónak*!

A megoldási út (állapotsorozat) keresésénél melyik a jó eljárás?

Részben az elért eredmény, részben annak megtalálási költsége minősíti. A szokásos értékelési szempontok:

1. Teljesség (*completeness*) - ha van megoldás, akkor biztosan megtalálja
2. Időigény (*time complexity*) – mennyi idő a megoldás megtalálása?
3. Tárigény (*space complexity*) – mennyi tárhely kell
4. Optimalitás (*optimality*) – ha több megoldás van, megtaláljuk-e a legjobbat? (megint: mi a legjobb?)

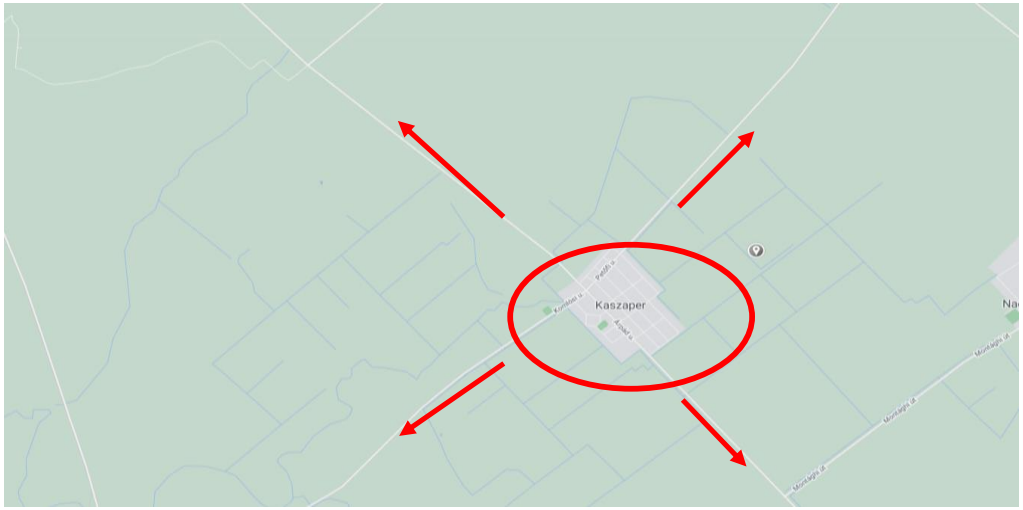
Keressük meg azt a jó trajektóriát (utat), ami a kezdeti állapotból a célállapotba visz, a lehető legkisebb költséggel!

A feladat könnyűnek látszhat, DE

- nagyon sok állapot van, (….exponenciális….. ☹)
- sokféle átmenet lehet egyikből a másikba,
- *nem látjuk át „felülről” az állapotteret*

Például Kaszaper ⇒

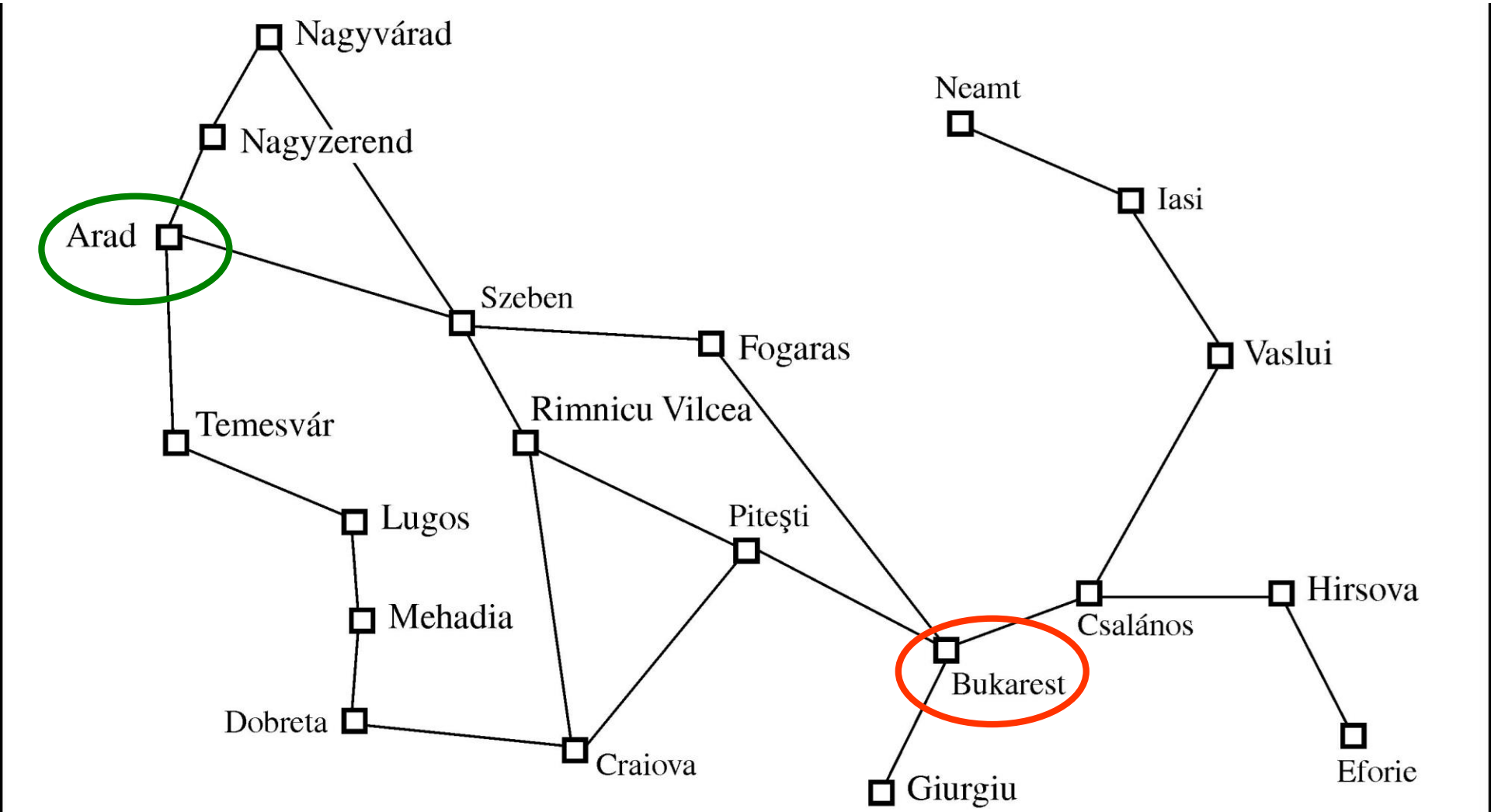
⇒ Győrtelek útvonal….



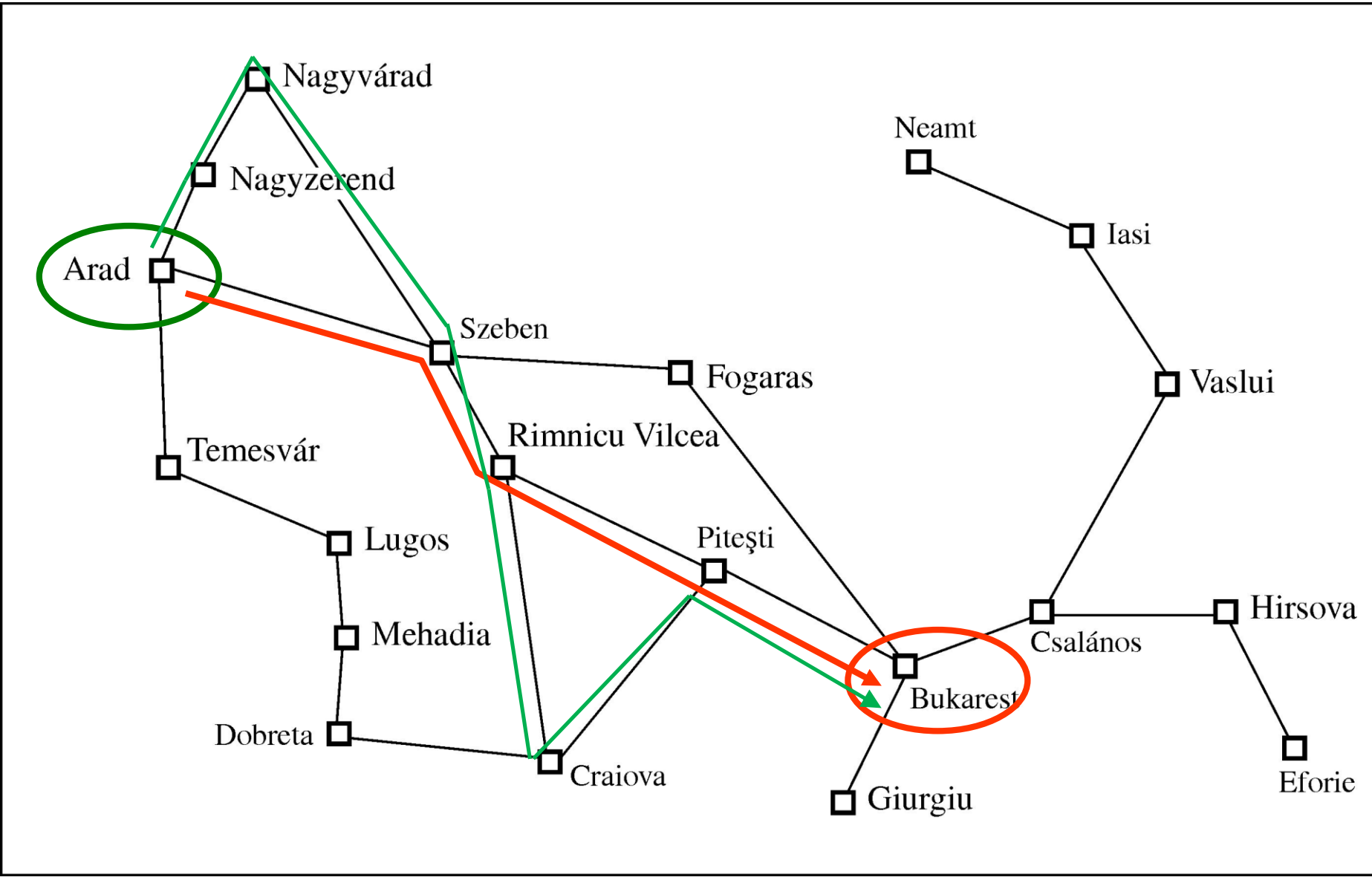
Mindig csak azt tudjuk, hogy az adott helységnek melyek a közvetlen, autóúton elérhető szomszédai. (az adatbázisunkban ez van tárolva, vagy a régi lapozgatós térképek, vagy a helyiek infói: hát tudja kedveském, a görbe fánál lehet menni jobbra is, balra is…)

A Russel-Norvig könyv példája: el akarunk jutni Aradról Bukarestbe...

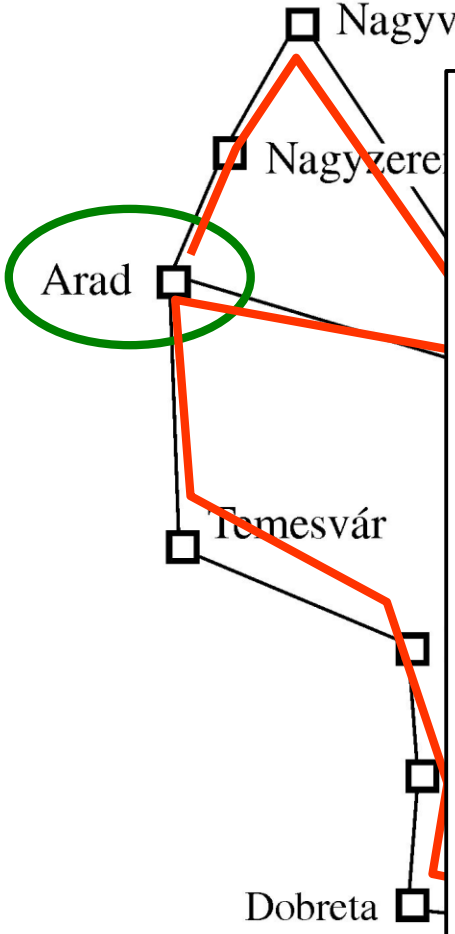
(FONTOS: a keresés, ***nem csak térképen keresést jelent***, csak ez az egyik legszemléletesebben bemutatható probléma! Pl. a nekünk jó lakást/állást/élettársat/éttermet, az országnak legjobb erőművet, a világbékét stb. is kereshetjük)



Milyen utat találunk meg?



A probléma formalizálása



$S_Á$ S_K: pl. hol vannak utak,
vagy van-e útfelbontás valahol stb.

Kezdeti állapot = Arad
Célállapot = Bukarest

Operátorok:

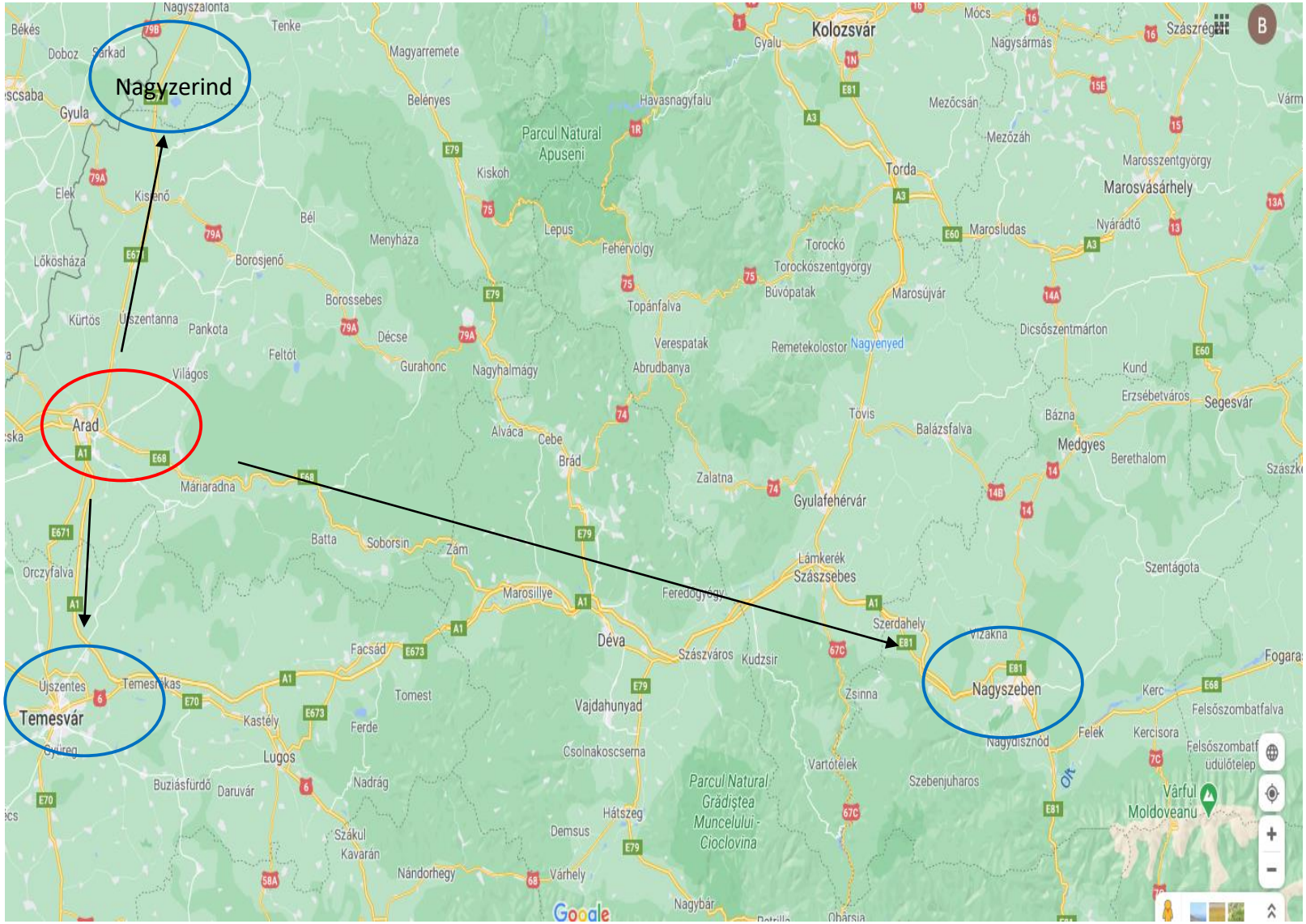
IF (X és (X ⇒ Y)) THEN Y

Útköltség = k + táv(X,Y) + …

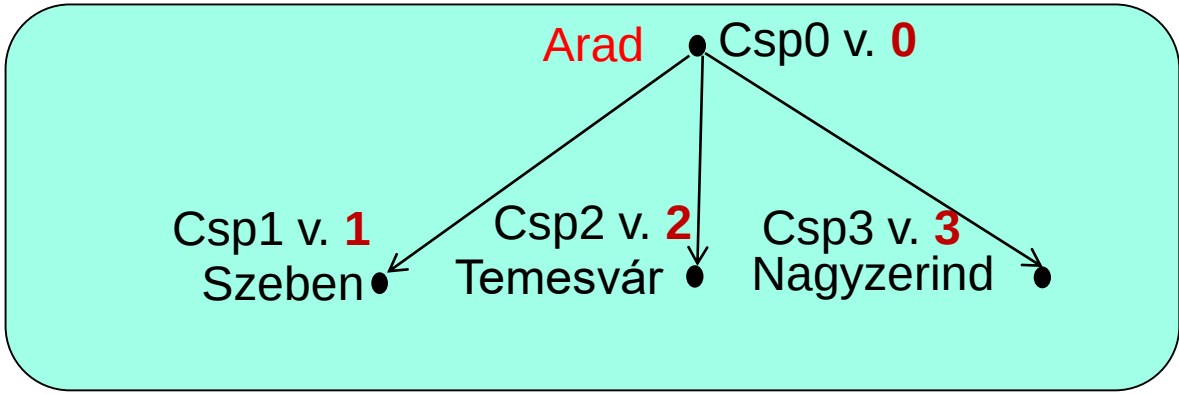
Célállapot teszt: Y = Célállapot?

Adatbázis:

(Arad ⇒ Nagyzerénd)	táv(Arad,Nagyzerénd) = 75
(Arad ⇒ Temesvár)	táv(Arad,Temesvár) = 118
…	…


Nagyzerind
Arad
Temesvár
Nagyszeben

A keresés folyamatát az ún. **keresési fával** ragadjuk meg.



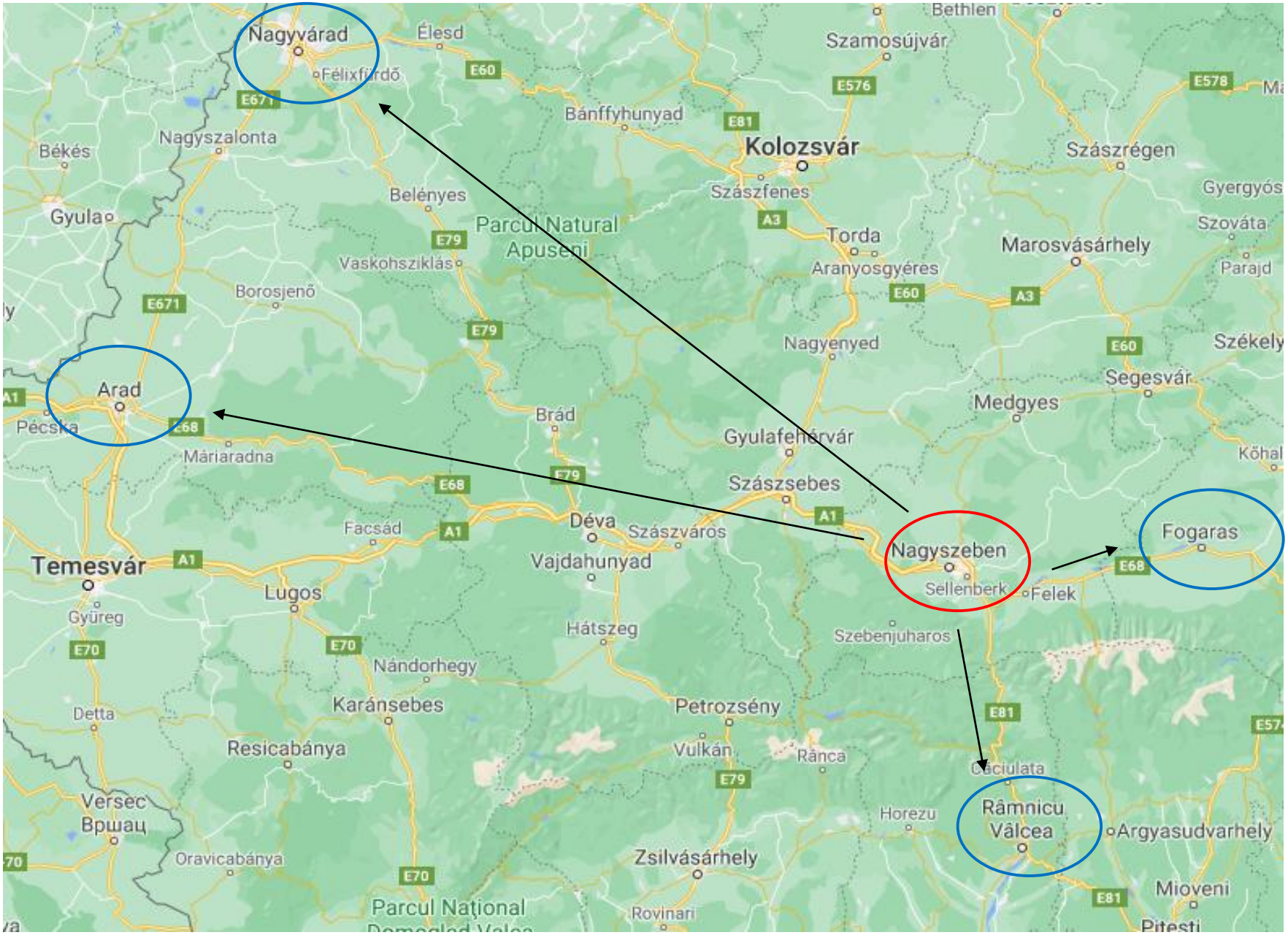
<u>A gráf csomópontjainak adatszerkezete (pl.)</u>:

1. **az állapottérnek a csomóponthoz tartozó állapota**
2. **szülő csomópont**
3. a gyökértől eddig a csomópontig vezető út csomópontjainak száma (mélység – *depth*)
4. (a csomópontig tartó útköltség $g(n)$)
5. (a hátralévő költség becslése)

stb.

Például:
(Szülő csp., Akt.áll., Megtett út, Hátralévő út becslése)

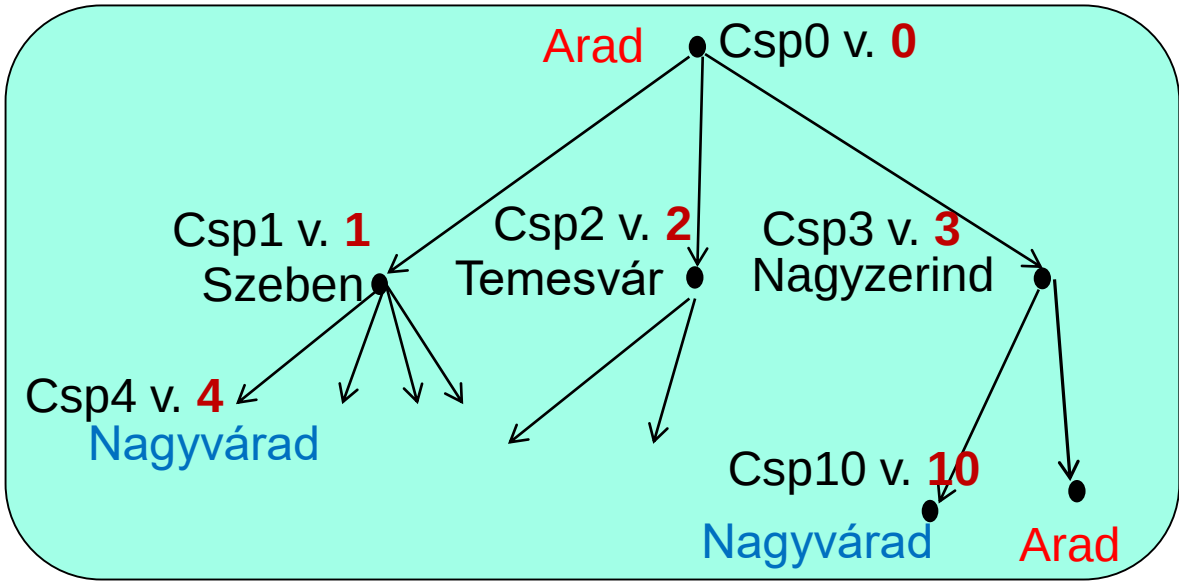
(-,csp0,Arad,0,*hArad*)
(csp0,csp1,Szeben,140,*hSzeben*)
(csp0,csp2,Temesvár,118,*hT*)
(csp0,csp3,Nagyzerind,75,*hNZ*)


Nagyvárad
Élesd
Félixfürdő
E60
E671
Bánffyhunyad
E81
Kolozsvár
Szamosújvár
E576
E578
Bethlen
Békés
Nagyszalonta
Szászrégen
Gyergyós
Gyula
Belényes
Szászfenes
A3
Parcul Natural
Apuseni
Torda
Szováta
Marosvásárhely
Vaskohsziklás
Aranyosgyéres
Parajd
Borosjenő
E671
E60
A3
E79
Nagyenyed
Székely
E60
Segesvár
Arad
Medgyes
Pécska
Brád
Gyulafehérvár
E68
Máriaradna
Kőhal
E79
Szászsebes
E68
A1
Déva
Facsád
Szászváros
Fogaras
A1
Nagyszeben
Temesvár
Vajdahunyad
E68
Sellenberk
Felek
A1
Lugos
Gyüreg
Hátszeg
Szebenjuharos
E70
E70
Nándorhegy
Detta
Karánsebes
Petrozsény
E81
Resicabánya
Vulkán
Ránca
Cáciulata
E79
Râmnicu
Vâlcea
Horezu
Versec
Argyasudvarhely
Oravicabánya
Zsilvásárhely
E70
Mioveni
Parcul Național
Rovinari
E81

A keresés folyamatát az ún. **keresési fával** ragadjuk meg.

Egyetlen „Nagyvárad” , „Arad” vagy „Szeben” stb. állapot van, de több csomóponthoz is tartozhat ez az állapot!

Ha CSAK azt tudjuk, hogy Temesvár állapotban vagyunk, nem tudjuk, és nem is tudhatjuk, hogyan jutottunk ide. A keresési gráf csomópontjaiban több infót tárolunk a puszta állapotnál.



Például: (Szülő csp., Akt.állapot., Megtett út, Hátralévő út *becslése*)

(-,csp0,Arad,0,*hArad*)

(csp0,csp1,Szeben,140,*hSzeben*)

...

(csp1,csp4,Nagyvárad,291,*hNV*)

...

(csp3,csp10,Nagyvárad,146,*hNV*)

Keresési stratégiák

Nem informált keresések (un. **gyenge**, vagy **vak** keresések)

a) ismert:
   - a start- és a célállapot(ok),
   - állapotok közti lehetséges átmeneteket

b) nem ismert:
   - milyen költségű az aktuálisból a célállapotba vezető út,
   - merrefele kell elindulnunk

A vak (nem informált) keresési stratégiákat a **csomópontok kifejtési sorrendje különbözteti meg**. Ez a különbség óriási jelentőséggel bírhat.

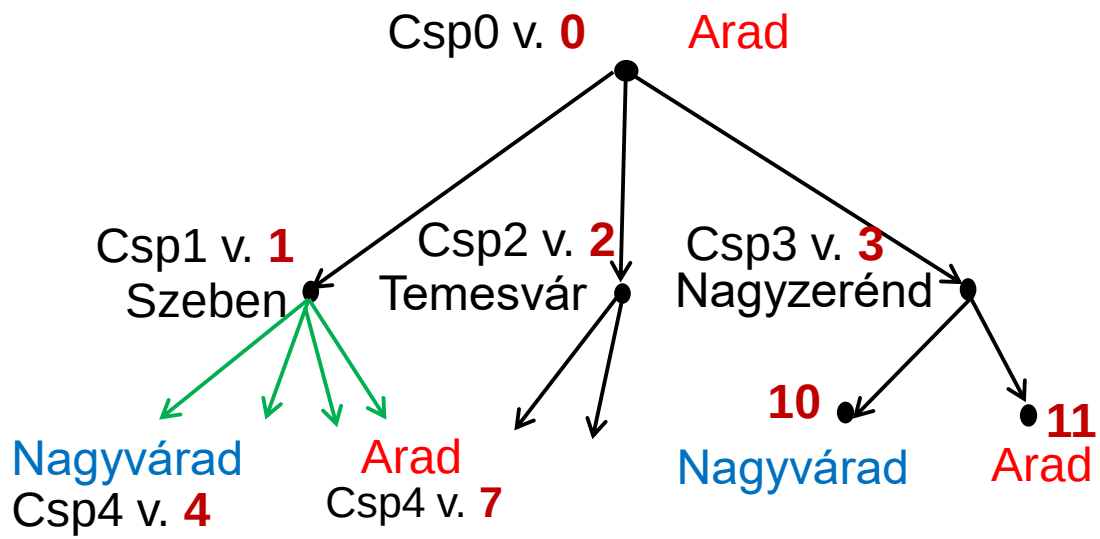
Szélességi keresés

Két listát kezelünk,

- **openList** (oL)
- **closedList** (cL)

0. Iniciálás:

- cL={} üres
- oL={ Csp0 }



1. Vesszük az oL első elemét, megvizsgáljuk, hogy elértük-e a célt. Ha igen, kész vagyunk,ha nem, kifejtjük: megkeressük a követő állapotokat, és létrehozzuk a gyermekcsomópontokat felvesszük oL ***végére***. A vizsgált első elemet áttesszük oL-ből cL-be.

oL={Csp1, Csp2, Csp3}, cL={Csp0}

2. A létrejött gyermekcsomópontokat oL ***végére tesszük***. A vizsgált első elem oL→ cL.

oL={Csp1, Csp2, Csp3}, cL={Csp0}

3. oL első eleme – cél?. Ha nem, kifejtjük, - gyermekcsomópontok oL ***végére***.

A vizsgált első elem oL→ cL.

oL={Csp2, Csp3, Csp4, Csp5, Csp6, Csp7}, cL={Csp0, Csp1}

…. ezt ismételjük, amíg a sor első csomópontjának állapota nem lesz a Cél!

+ *minden lépésnél a célvizsgálat után meg kell vizsgálni, hogy nem jött-e létre hurok!*

Mindegyik keresési módszernél ez lesz a eljárásunk:

0. Létrehozzuk a kezdeti (gyökér) csomópontot

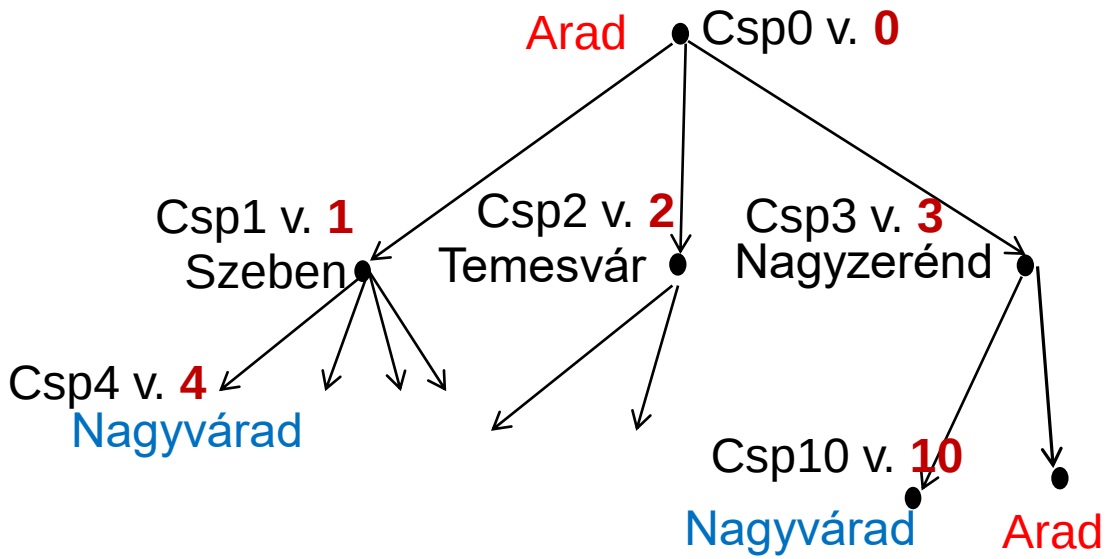
Majd a következő ciklust alkalmazva építjük a keresési gráfot, az open és close listát:

1. **Vesszük az open lista első elemét (mi ebben a tárgyban– ha mást nem mondunk – mindig a legbaloldalibbat vesszük annak)**
2. **Megvizsgáljuk, hogy a célállapot tartozik-e ehhez a csomóponthoz**
3. **Ha igen, akkor kész vagyunk, kialakítjuk a megtalált megoldási lépéssort** ⇢
4. **Ha nem, akkor kifejtjük az éppen vizsgált csomópontot (létrehozzuk a hozzá tartozó gyermekcsomópontokat)**
5. ***A létrejött gyermekcsomópontokat betesszük az open listába, az adott eljárástól függ, hogy az open listába hova tesszük őket***
6. ⇨ **1.**

Az eljárások csak az 5. lépésben különböznek!

(Szélességi keresésnél az 5. lépés: az open lista végére tesszük a gyermekcsomópontokat)

Keresési fa



Fontos fogalmak:

- kifejtés (*expand*) – általában ez az időigényes!
- (átlagos) elágazási tényező – *b* (*branching factor*)
- mélység – *d* (*depth*)
- hullámfront (*frontier*)

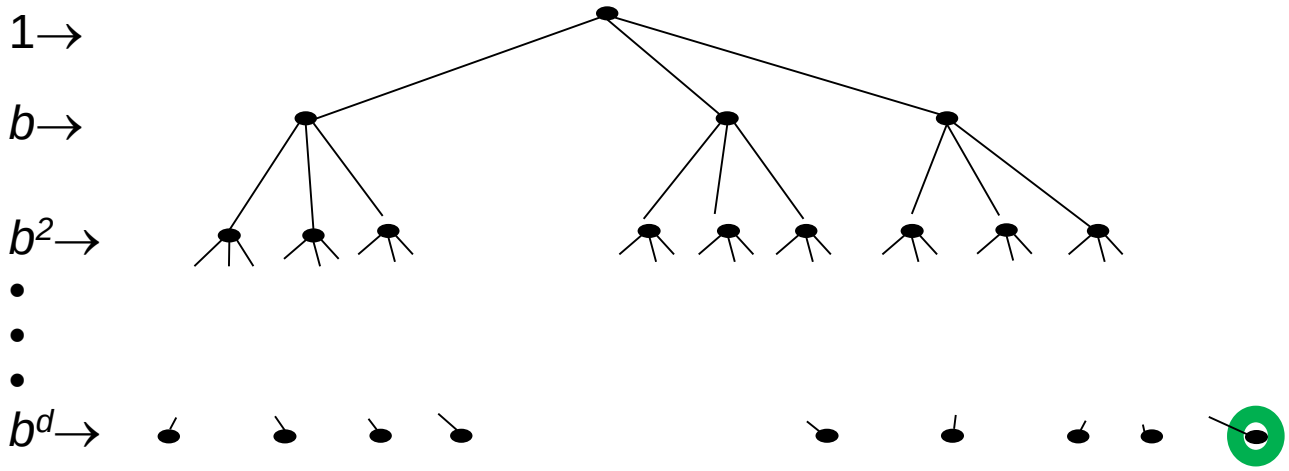
Szélességi keresés:

- Teljes
- Optimális? (csak akkor, ha minden átmenet költsége azonos, ekkor egy adott mélységben az összes csp. ugyanolyan költséggel érhető el, a mélyebben lévők pedig „drágábban”)

A legalsó szinten lévő összes csomópontot ki kell fejtsük, benne kell legyenek a tárban, ha a legrosszabb esetet, worst case-t vesszük!

- Időigény (csomópontkifejtések száma) $\rightarrow O(b^d)$ EXPONENCIÁLIS!!! ☹
- Tárigény $\rightarrow O(b^d)$ EXPONENCIÁLIS!!! ☹

Csomópontok
száma 0,1,2,…*d*
mélységben



Egyenletes költségű keresés (Dijkstra)

A szélességi keresés módosítása:

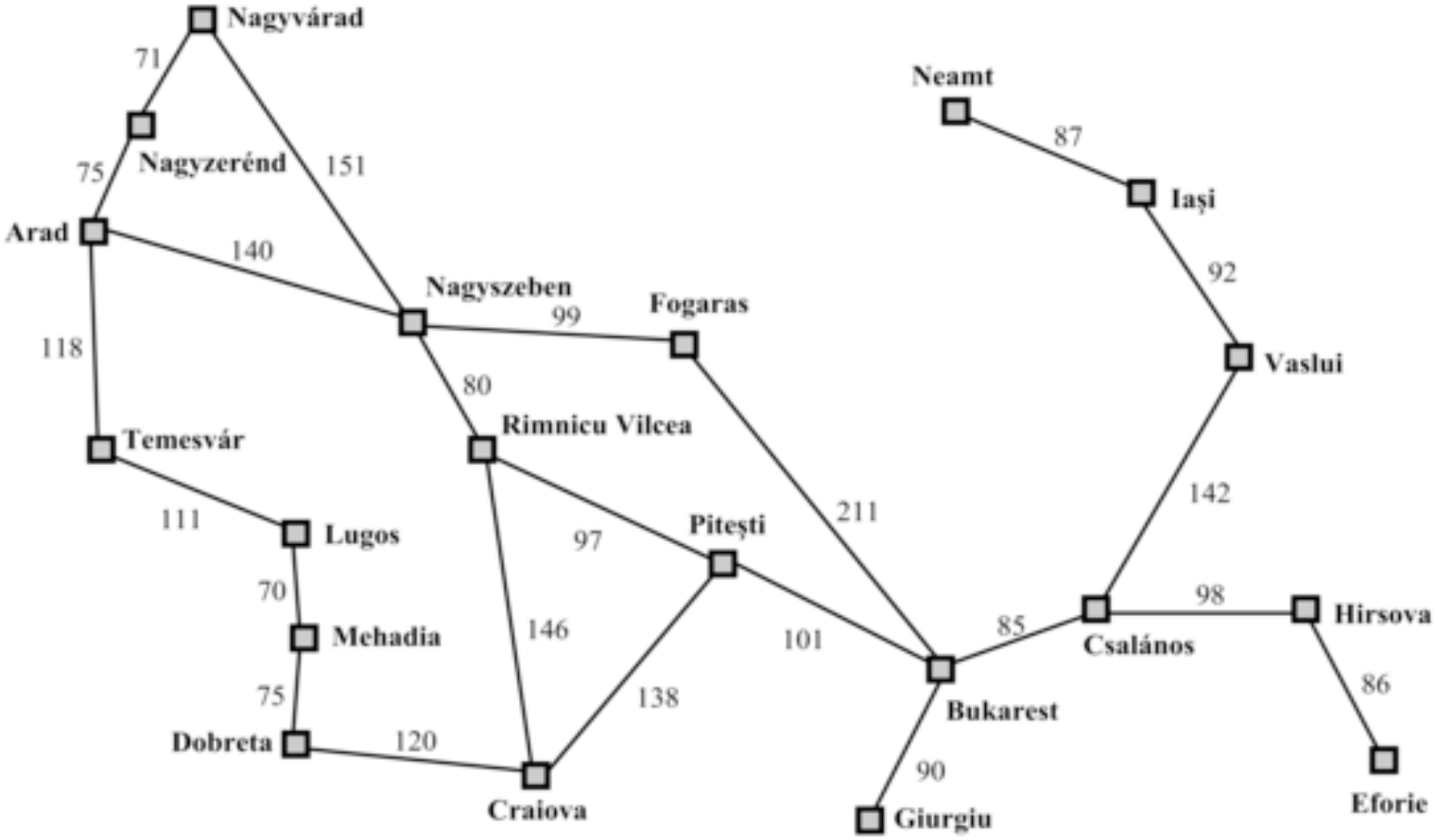
- a csomópontoknál a startállapottól eddig (az *n*-dik csomópontig) tartó útköltséget is tároljuk – g(*n*)
- *a hullámfront g(n)-nel mért legkisebb költségű csomópontját fejti ki először (nem a legkisebb mélységű csomópontot)*

Az egyenletes költségű keresés a legolcsóbb megoldást találja meg, feltéve ha az útköltség bármelyik út bejárása során sem csökkenhet:

g(Követő(*n*)) $\geq$ **g(*n*)** -minden *n*-re

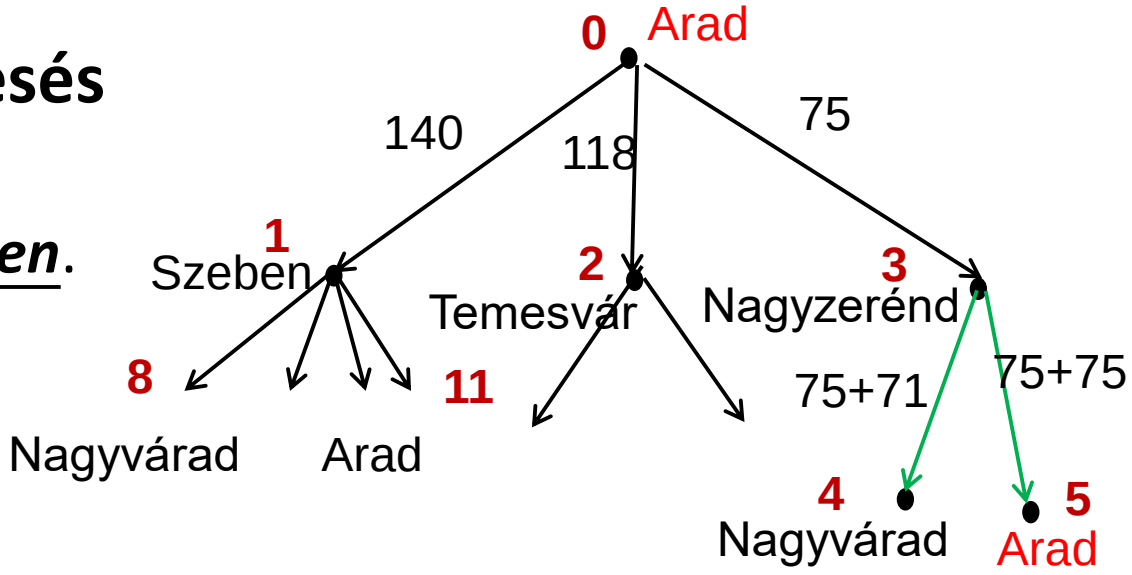
A szélességi keresés olyan egyenletes költségű keresés, amelyben:

g(*n*) = Mélység(*n*)


Nagyvárad
71
Nagyzerénd
75
151
Arad
140
Nagyszeben
99
Fogaras
118
80
Rimnicu Vilcea
Temesvár
111
Lugos
97
Piteşti
211
70
Mehadia
146
101
75
138
Dobreta
120
Craiova
90
Giurgiu
Bukarest
85
Csalános
98
Hirsova
86
Eforie
142
Vaslui
92
Iaşi
87
Neamt

Egyenletes költségű keresés

oL rendezése itt: a ***megtett úthossz növekvő sorrendjében***.



0. Iniciálás:

- cL={} üres
- oL={Csp0}

1. Vesszük az oL első elemét, célvizsgálat. *Ha cél, készen vagyunk*, ha nem, akkor kifejtjük: létrehozzuk a gyermekcsomópontokat.

oL={Csp3, Csp2, Csp1}, cL={Csp0}
75 118 140

2. A gyermek csomópontok ***a megtett úthosszak, g(n) sorrendje szerint teszük*** oL-be. A vizsgált első elem oL→ cL.

oL={Csp2, Csp1, Csp4, Csp5}, cL={Csp0, Csp3}
118 140 146 150

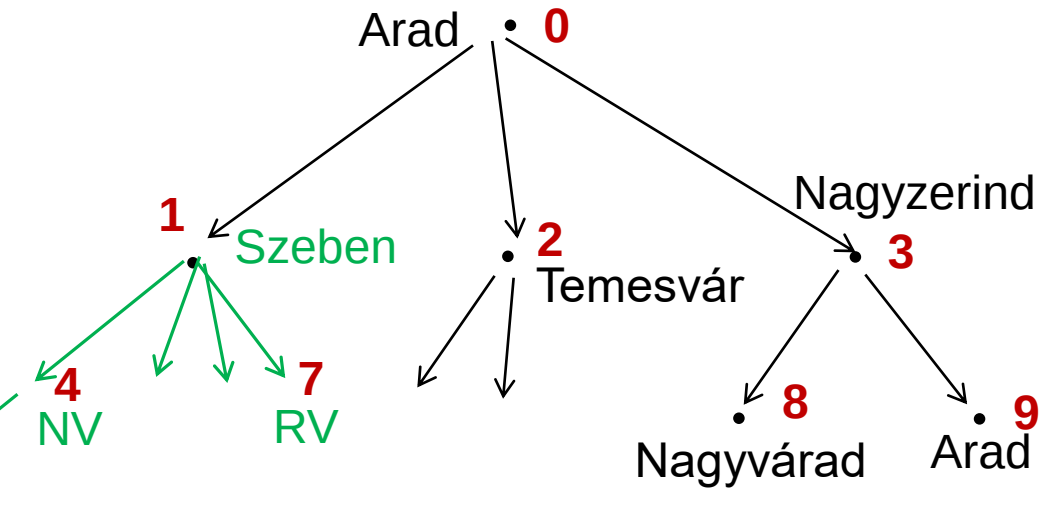
\+ Az oL első elemét vizsgálva a célvizsgálat után érdemes megvizsgálni , hogy nem jutottunk-e olyan állapotba (városba), amiben már voltunk (a startig visszavezető listán szerepel valamelyik csomóponthoz rendelt állapotként*). (Csp5!) Hurkok kizárása!*

Mélységi keresés

Két listát kezelünk, openL és closedL:

0. Iniciálás:

- üres cL={}
- oL={Csp0}



1. Vesszük az oL első elemét, megvizsgáljuk, cél-e?. *Ha igen, kész vagyunk*, ha nem, akkor kifejtjük: létrehozzuk a gyermekcsomópontokat. A vizsgált első elem átkerül cL-be.

oL={Csp1, Csp2, Csp3}, cL={Csp0}

2. A gyermekcsomópontok az oL lista ***elejére***. A vizsgált első elem cL-be.

oL={Csp4, Csp5, Csp6, Csp7, Csp2, Csp3}, cL={Csp0, Csp1}

\+ a célvizsgálat után hurkok kizárása!

…. ezt ismételjük, amíg a sor első csomópontjának állapota nem lesz a cél, vagy nem jutunk zsákutcába! Zsákutca – visszalépés!

Mélységkorlátozott keresés

Az utak maximális mélységére sokszor korlátot ad. Ha ezt elértük – visszalépünk, azt az ágat már nem folytatjuk.

Pl. Románia egyszerűsített térképén 20 város van. Ha létezik megoldás, maximálisan 19 lépés hosszú lehet. Bevezethetünk pl. egy 19-es mélységkorlátot!

A mélységkorlát elérésénél a keresés visszalép. A megoldást biztosan megtaláljuk, ha

- létezik,
- a mélységkorlátnál sekélyebben fekszik.

(Amúgy itt minden város bármelyik városból legfeljebb 9 lépésben elérhető: ez az állapottér **átmérője** = jobb mélységkorlát, de sokkal nehezebb kideríteni.)

De semmi garancia nincs arra, hogy a legkisebb költségű (itt: legrövidebb út) megoldást találjuk meg. (optimalitás)

Túl kis mélységkorlát: létező megoldást esetleg nem talál meg. (teljesség)

Iteratívan mélyülő keresés

Mélységkorlátozott keresésnél: **hogyan válasszunk jó mélységkorlátot**?

A legtöbb esetben mindaddig nem tudunk jó mélységkorlátot adni, amíg meg nem oldottuk a problémát!

Lehetőség: a legjobb mélységkorlát mechanikus kiválasztása:

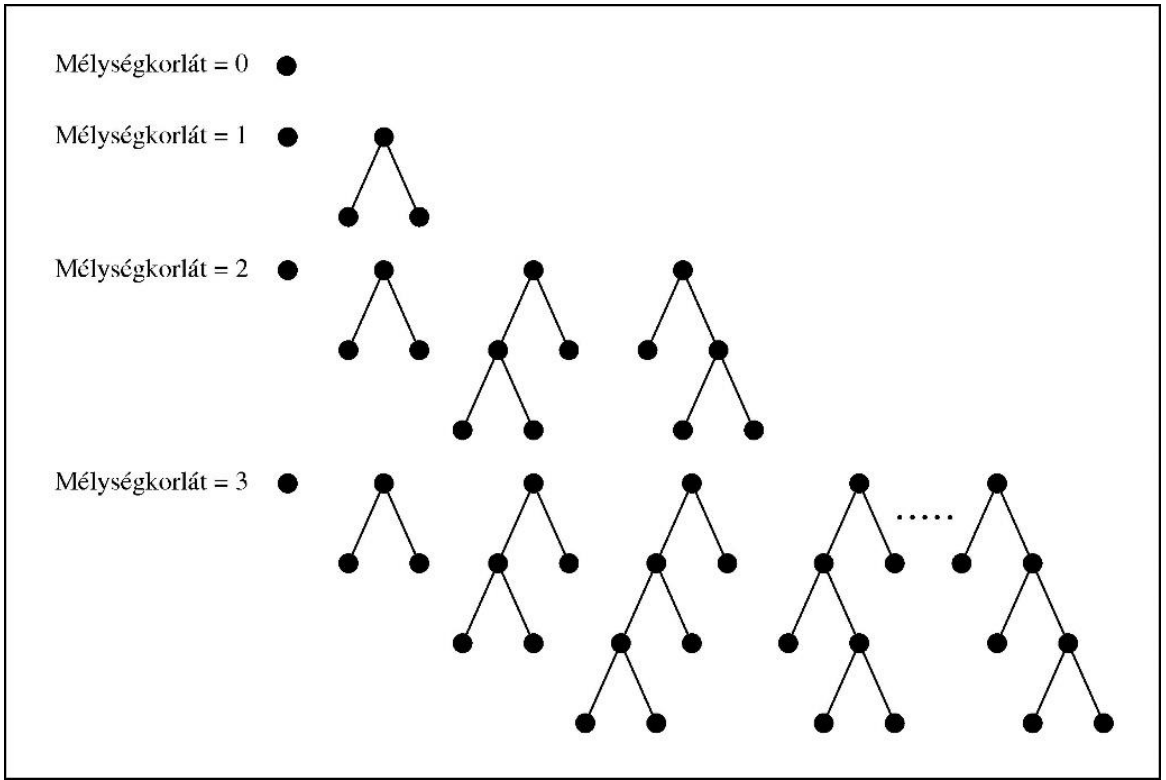
-sorban vesszük az összes lehetséges mélységkorlátot: először 0, majd 1, majd 2, stb.

- sorban ezekkel a mélységkorlátokkal végzünk mélységkorlátozott keresést.

Probléma: a nem túl mélyen fekvő állapotokat nagyon sokszor kifejtjük!

Időigényes!?)

Pl. ha 3 mélységben lesz a megoldás, akkor a gyökércsp-t 4-szer. (Mélységkorlátok 0,1,2,3)



Az iteratívan mélyülő keresés gyakorlatilag ötvözi a szélességi és mélységi keresés előnyös tulajdonságait

A szélességi kereséshez hasonlóan **optimális** és **teljes**, de csak a mélységi keresés **szerény memóriaigény**ével rendelkezik.

De bizonyos állapotokat az algoritmus többször is kifejt!

Tékozló? - egy exponenciálisan növekvő keresési fában majdnem az összes csomópont a legmélyebb szinten található!

d mélységben a megoldás, *b* elágazási tényező: szélességi/mélységi keresésnél a kifejtések száma (worst case): $1 + b + b^2 + ... + b^{d-2} + b^{d-1} + b^d$

pl. *b* =10 és *d* = 5 esetén ez a szám: 1 + 10 + 100 + ... + 100.000 = **111.111**

az iteratívan mélyülő keresésnél a kifejtések teljes száma:

$(d+1)\cdot 1 + d \cdot b + (d-1) \cdot b^2 + ... + 3 \cdot b^{d-2} + 2 \cdot b^{d-1} + 1 \cdot b^d$

b = 10 és *d* = 5 esetén: $6 \cdot 1 + 5 \cdot 10 + 4 \cdot 100 + ... +$ **$1 \cdot 100.000$** = **123.456** **(csak 123456/111111=1,111 tehát +11,1% a „felesleges” növekedés)**

Minél nagyobb az elágazási tényező, annál kisebb a többletmunka!

(Pl. b = *2* elég rossz, de csak kb. **200%**, azaz kb. *kétszerese* a szélességi/mélységinek! Hányszor több csomópontot fejtünk ki a szélességi/mélységihez képest, ha a megoldás *d* mélységben van, és utólag kiderül, hogy *b* = 1 volt?)

Kétirányú keresés

- egyszerre előrefelé a kiinduló állapotból, illetve hátrafelé a cél állapotból
- a keresés akkor fejeződik be, ha a két keresés valahol találkozik

$$O(2 \times b^{d/2}) = O(b^{d/2})$$

Például: $b = 10$, $d = 6$:
a szélességi/mélységi keresés = **1.111.111** csomópontot fejt ki,

a kétirányú keresés mindkét irányban 3 mélységnél ér célba = **2.222** csomópontot generál.

Elméletben nagyon jó, a megvalósítás nem triviális.

Célállapotból hátrafelé keresni? Az *n* csomópont **előd csomópontjai** azon csomópontok, amelyek követő csomópontja lehet *n*.

- A hátrafelé keresés a célcsomópontból indulva az előd csomópontok egymást követő generálását jelenti. Megfordítható-e a folyamat? (Összerakható-e Csernobil a romokból?)
- Pl. sakk, nagyon sok célállapot, melyikből induljunk visszafele?

A neminformált (vak) keresési stratégiák összehasonlítása

b - elágazási tényező, *d* - a megoldás mélysége,
m - a keresési fa maximális mélysége, *l* - a mélységkorlát.

Jellemző	SzK	EgyKK	MK	MKK	IMK	KK
Idő-igény	b^d	b^d	b^m *(m lehet >>d)*	b^l	b^d	$b^{d/2}$
Tár-igény	b^d	b^d	bm	bl	bd	$b^{d/2}$
Opt.?	Igen (ha…)	Igen	Nem	Nem	Igen (ha…)	Igen (ha…)
Teljes?	Igen	Igen	Nem	Igen, ha $l \geq d$	Igen	Igen

A neminformált (vak) keresési stratégiák összehasonlítása

b - elágazási tényező, *d* - a megoldás mélysége, *m* - a keresési fa maximális mélysége, *l* - a mélységkorlát.

Krit.	SzK	EgyKK	MK	MKK	IMK	KK
Idő igény	b^d	b^d	b^m	b^l	b^d	$b^{d/2}$
Tár igény	b^d	b^d	bm	bl	bd	$b^{d/2}$
Opt.?	Igen (ha)	Igen	Nem	Nem	Igen (ha…)	Igen (ha…)
Teljes?	Igen	Igen	Nem	Igen, ha $l \geq d$	Igen	Igen

Az exponenciális tár-, illetve időigény komoly problémát jelent!
Jobb módszerek kellenek!

A híd (logikai feladvány)
Egy ködös este 4 hallgató megrekedt a Borzalmas Titkok Szigetén, ahonnan a szárazföldre csak egy rozoga híd vezetett át. Az éjszakát a szigeten életveszélyes volna eltölteni, a rothadó híd egyszerre csak 2 személyt bír el, és sötétben nem járható. Viszont a növekvő sötétséghez csak egy zseblámpájuk volt, melynek eleme még pontosan 17 perc világításra volt elég.

Dani, a maratonfutó, 1 perc alatt menne át a túlsó oldalra. Cili, a biciklista, 2 perc alatt végig tudná futni a hidat. A jó tornász Bercel 5 perc alatt kocogná végig. Viszont Andrásnak, az erős dohányosnak, 10 percre lenne szüksége.

Sikerül-e mindannyiuknak megmenekülni a szigetről? Ha nem, miért nem, ha igen, hogyan?

Oldjuk meg mélységkorlátozott kereséssel! Mi lesz a mélységkorlát? (Természetesen a hurkokat zárjuk ki.)