Kereséssel történő problémamegoldás

Ormándi Róbert

Problémamegoldás kereséssel

- Célorientált ágensek egyik típusa
- Általános keret, melyben:
  - Meghatározásra kerül(nek) a cél(ok)
  - Megfogalmazásra kerül a probléma → **állapottér reprezentáció**
  - Probléma megoldása → **keresés** az állapottérben
  - A keresés eredményének (**cselekvéssorozat**) végrehajtása

Problémamegoldás kereséssel

- Alkalmazható, ha a környezet:
  - Statikus:
    - A környezet változatlan
  - Diszkrét
    - Megszámlálható(an véges vagy végtelen számú) állapottal leírható
  - Determinisztikus
    - A következő állapot teljes egészében meghatározott az **aktuális állapot** és egy választott **cselekvés** által
    - Nincs véletlen komponens
  - Teljesen megfigyelhető

Problémamegoldás kereséssel

- Célorientált ágensek egyik típusa
- Általános keret, melyben:
  - Meghatározásra kerül(nek) a cél(ok)
  - Megfogalmazásra kerül a probléma → **állapottér reprezentáció**
  - Probléma megoldása → **keresés** az állapottérben
  - A keresés eredményének (**cselekvéssorozat**) végrehajtása

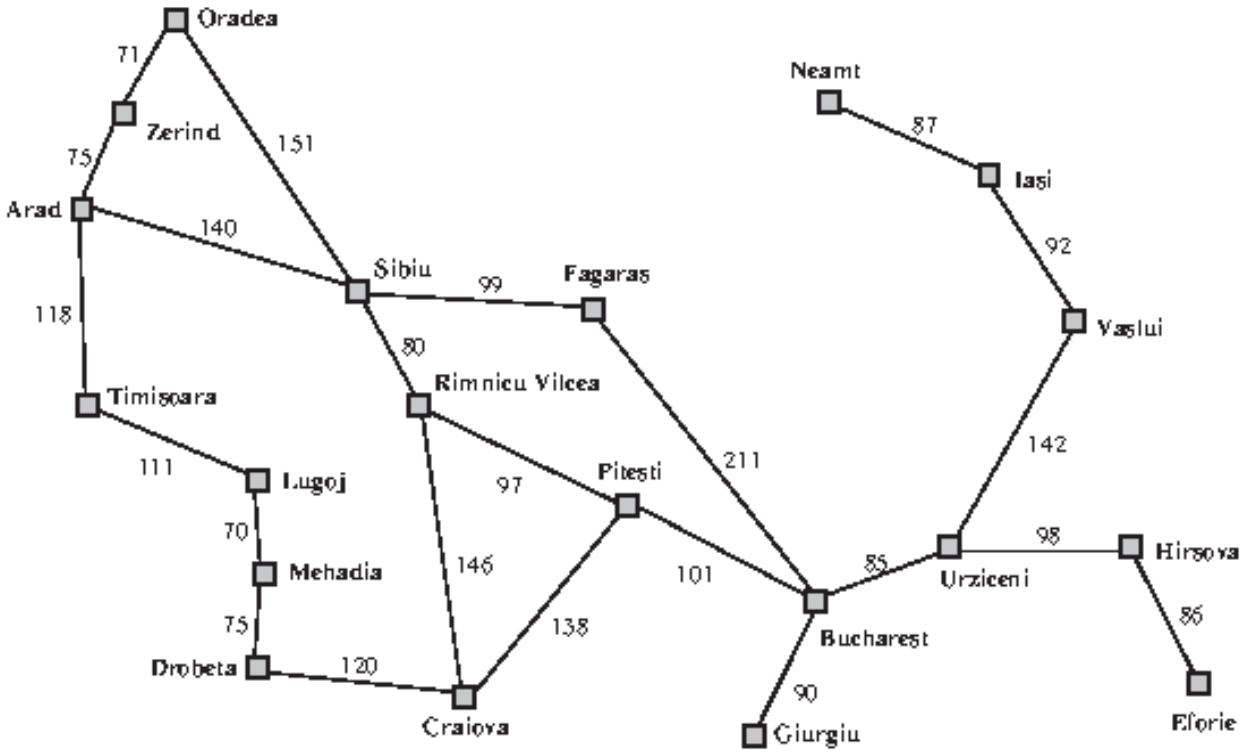
Állapottér

- Probléma megfogalmazása, absztrakció
- Problémaspecifikus részletek megtartása
- Meg kell határozni a következőket:
  - Lehetséges állapotok halmaza (**csomópontok**)
  - Lehetséges cselekvések halmaza (**élek**)
    - Minden cselekvés → cselekvés, állapot párok halmazát
  - Kezdőállapot
  - Végállapotok halmaza
  - Állapotátmenet költség függvénye (**élsúlyok**)

→ súlyozott, irányított gráf → állapottér

Állapottér

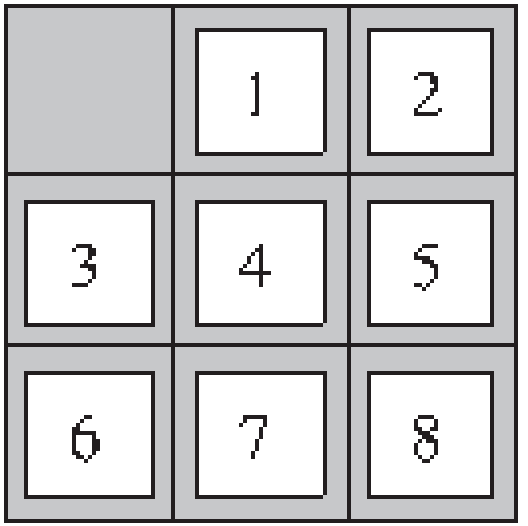
- Probléma: Aradról Bukarestbe szeretnénk navigálni
  - Térkép alapján elkészítjük az állapottér reprezentációt
  - Keresést végzünk az állapottérben
  - Keresés eredményét végrehajtjuk



- Kereső algoritmus csak egy gráfot lát → lehet általános
- Útvonal költsége, hossza: az alkotó élek összsúlya

Állapottér – további példák

- 8-as kirakó:
  - Állapottér: 8 számérték + üres hely pozíciója
  - Állapotátmenet: üres hely 4 irányba mozgatásával (balra, jobbra, fel és le) előálló legális állapotok
  - Állapotátmenet költsége: 1
  - Kezdő állapot véletlenszerű
  - Végállapot: ábrán látható



Állapottér – további példák

- 8 királynő (v. 1.):
  - Állapottér: 0-8 királynő tetszőleges elhelyezése a táblán
  - Állapotátmenet: helyezz egy királynőt egy üres mezőbe
  - Állapotátmenet költsége: 1
  - Kezdő állapot: üres tábla
  - Végállapotok: 8 királynő a táblán, egyik sincs támadás alatt
- ~ 10^{14} számú állapot

- 8 királynő (v. 2.):
  - Állapottér: *n* (0-8) királynő elhelyezése a táblán az *n* baloldali oszlopban, úgy hogy nincs támadás
  - Állapotátmenet: helyezz egy királynőt a legbaloldalibb még üres mezőbe, hogy ne legyen támadás
- ~ 2057 számú állapot
- és az állapottér egy fa
- → jó állapottér reprezentáció fontos

Problémamegoldás kereséssel

- Célorientált ágensek egyik típusa
- Általános keret, melyben:
  - Meghatározásra kerül(nek) a cél(ok)
  - Megfogalmazásra kerül a probléma → **állapottér reprezentáció**
  - Probléma megoldása → **keresés** az állapottérben
  - A keresés eredményének (**cselekvéssorozat**) végrehajtása

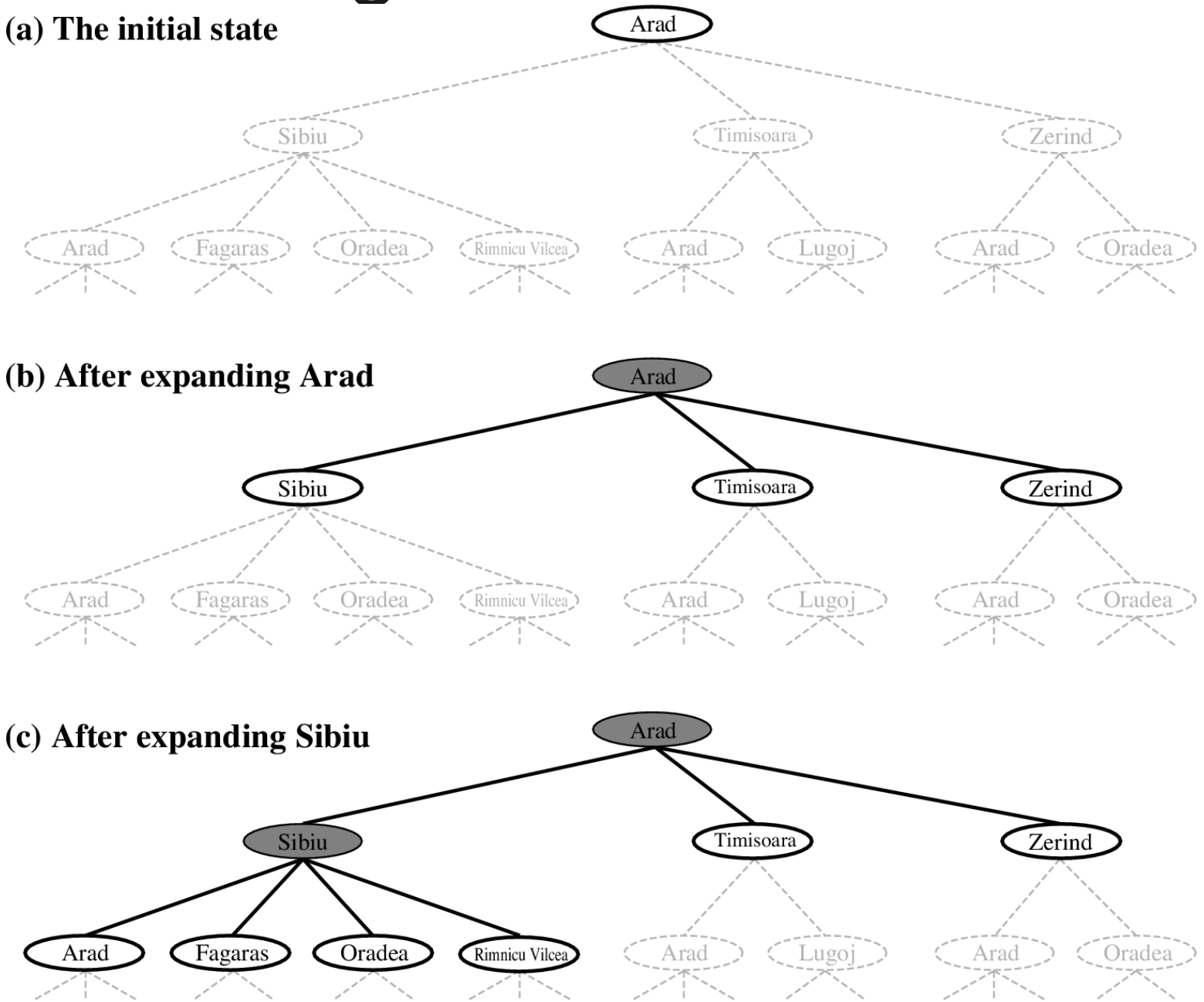
Kereső algoritmusok – fakeresés

- Adott kezdőállapotból, keressünk legrövidebb utat valamelyik célállapotba

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

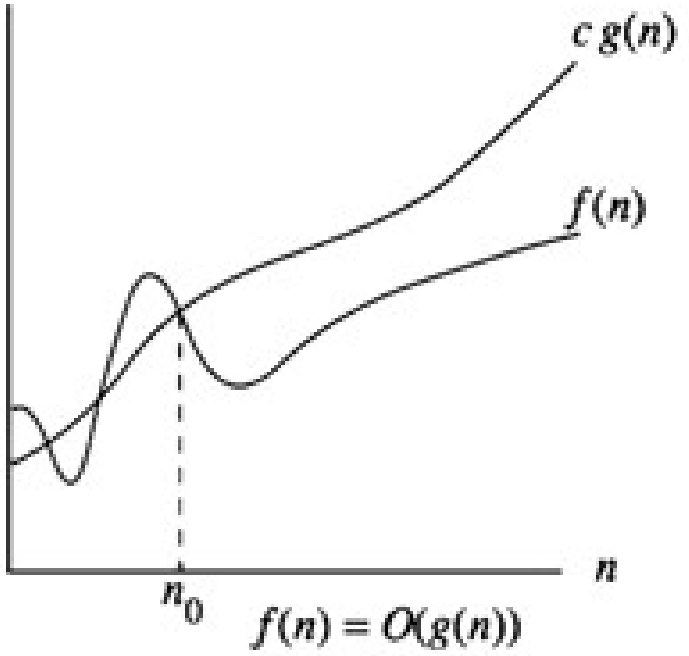
- Ötlet: növesszünk keresőfát a kezdőállapotból, szomszédos állapotok hozzáadásával, amíg célállapotot nem találunk
- Perem/nyílt halmaz karbantartása, eltérő megvalósítás → más stratégia
- Vigyázat! A keresőfa NEM azonos az állapottérrel. Véges méretű állapotterek esetén is kaphatunk végtelen méretű keresőfát (pl. Navigáció probléma)

Kereső algoritmusok – fakeresés



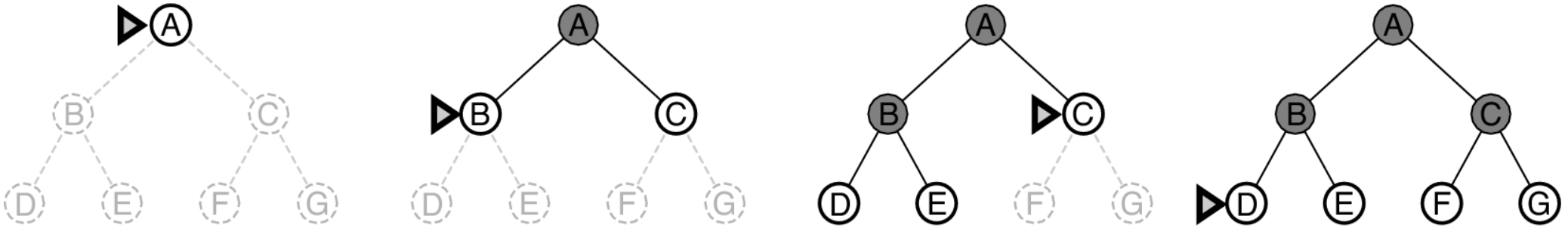
Kereső algoritmusok elemzése

- Teljesség: akkor és csak akkor, ha minden esetben, amikor létezik véges számú állapot érintésével elérhető célállapot, az algoritmus megtalálja
- Optimalitás: akkor és csak akkor, ha teljes és a megtalált célállapot optimális költségű
- Futásidő és tárigény NEM az állapottér méretének függvényében vizsgáljuk, hanem:
  - b: szomszédok maximális száma
  - m: keresőfa maximális mélysége
  - d: legkisebb mélységű célállapot



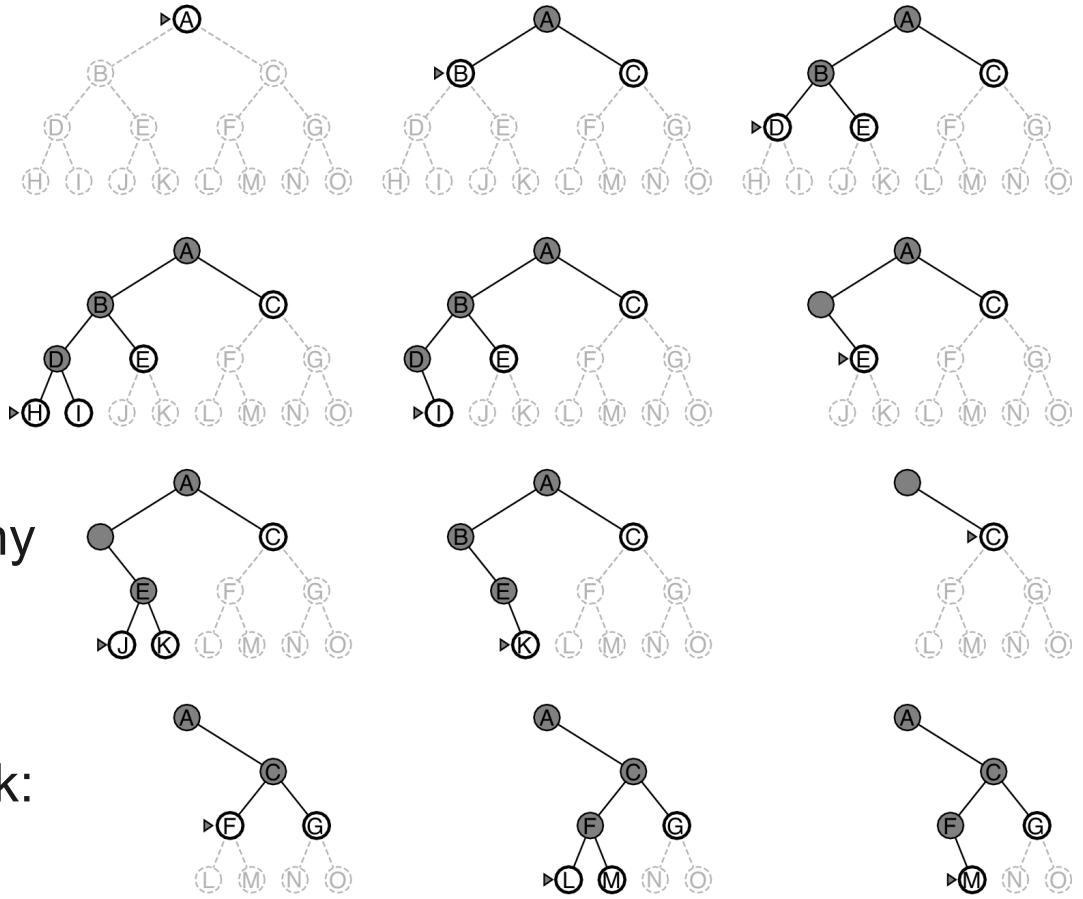
Szélességi keresés

- A fakeresés algoritmusban a perem megvalósítása FIFO (sorral) → szintenkénti bejárás
- Teljes, minden véges számú állapot érintésével elérhető állapotot véges időben elér
- Általában nem optimális, de pl. ha a költség a mélység nemcsökkenő függvénye, akkor igen.
- Időigény = tárigény = $O(b^{d+1})$
  - Bizonyítás:
    - Időigény: generált csomópontok: $1+b+...+b^{d}+b^{d+1}-1= O(b^{d+1})$
    - Tárigény: perem a memóriában + összes ős = generált pontok száma
- Komplexitás exponenciális → nagyon kis mélységek jönnek szóba futásidő és tár tekintetében is



Mélységi keresés

- A fakeresés algoritmusban a perem megvalósítása LIFO (veremmel) → legmélyebben fekvő csomópontok kifejtése
- Teljes, teljes ha a keresési fa véges mélységű (*m*)
- Nem optimális (ellenpéldával bizonyítható)
- Időigény = $O(b^m)$ (legrosszabb), tárigény = O(bm)
  - Bizonyítás:
    - Időigény: generált csomópontok: $1+b+...+b^m-1= O(b^m)$
    - Tárigény: *m* méység, minden mélységben a szomszédság tárolt
- Futásidő nagyon rossz m>>d, nem teljes és nem optimális, tárigény javítható speciális implementációval



Iteratívan mélységű keresés

- Mélységi keresések sorozata 1, 2, 3, stb., mélységkorlátokkal, amíg célállapotot találunk.
- Teljes és optimális (feltéve, hogy a költség a mélység nemcsökkenő függvénye)
- Időigény = $O(b^d)$ (legjobb, ~ szélességi), tárigény = O(bd) (jobb mint a mélységi)
- Meglepőnek tűnhet, hogy többször generálunk csomópontokat mégis a legjobb futásidőt érjük el, ok: a szélességi a d+1-dik szintről is generál csomópontokat és a levelek számossága a fában hatalmas
  - Bizonyítás:
    - Időigény: generált csomópontok: $d1+(d-1)b+...+1b^d= O(b^d)$ (ahol b^d -t kiemelve a maradék összeg, egy konvergens sorral korlátozható)
    - Tárigény: d mélységig a szomszédság a peremben ~ szélességi

Egyenletes költségű keresés

- A peremben a rendezés költség alapú: először a legkisebb költségű csúcsot terjesztjük ki
- Teljes és optimális, ha minden él költsége nagyobb mint 0
- 0 él és visszamutató él esetén végtelen ciklus!
- Idő és tárigény függ a költségfüggvénytől, nem tárgyaljuk
- A szélességi keresés egyenletes költségfüggvénnyel ennek egy speciális esete

Kétirányú keresés

- Start → Cél irányába és Cél → Start irányban is keresünk
- Csak akkor alkalmazható, ha a végállapotok halmaza explicit adott (sakk, esetén sakk-matt tesztnél nem)
- Mindkét irányban szélességi keresést használva, egyenletes lépésköltség esetén teljes és optimális
- Időigény = tárigény = $O(b^{d/2})$

Gráfkeresés

- Ha az állapottér nem fa, akkor a fakeresés véletlen ciklusban eshet, vagy csak a hatékonysága drasztikusan csökkenhet
- Ötlet: zárt halmaz bevezetése, amiben tároljuk a már kifejtett csomópontokat → minden állapothoz csak a legelső megtalált út lesz tárolva

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

- Probléma: Mi van, ha egy adott állapothoz a később megtalált út jobb?
  - Egyenletes költségű útnál az első megtalált útnál nincs jobb (Dijkstra, >0 költség feltétel kell!)
    - Bizonyítás: indukcióval, optimális útnak valamelyik állapota mindig a nyílt halmazban van
  - Mélységi keresésnél átlinkelés, ha jobbat találtunk → tárigény változik

Informált keresések

- Informálatlan esetben, csak arról volt információnk, hogy milyen úton jöttünk az adott állapotban, de arról nem, hogy az adott állapot mennyire jó, azaz milyen messze van a céltól (becsülve).
- Heurisztika: becslés arra, hogy az adott állapot milyen messze van egy célállapottól
- Pl. légvonalbeli távolság a navigációnál

Mohó legjobb először keresés

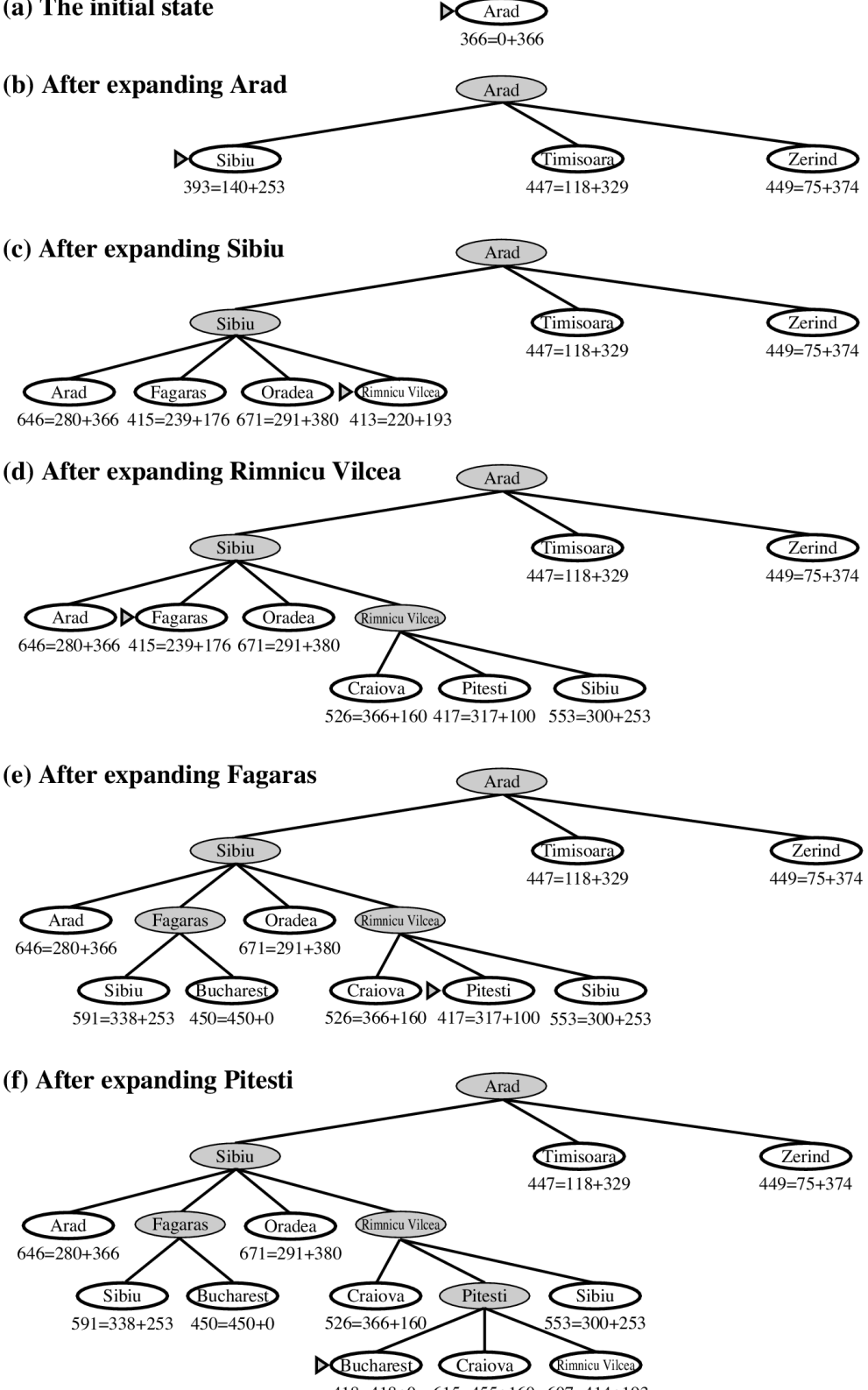
- A peremben a rendezést h(n) alapján végezzük: a legkisebb értékű csúcsot vesszük ki
- Hasonló a mélységi kereséshez, csak h(n) minimalizálása alapján halad a mélységben előre
- Feltétel: h(n) = 0, minden n célállapotra
- Teljes, teljes ha a keresési fa véges mélységű (*m*)
- Nem optimális (h(n) becslés,)
- Időigény = $O(b^m)$ (legrosszabb), tárigény = O(bm)
- Gráfkeresés esetén a zárt halmaz elemeinek átlinkelése szükséges
- A legrosszabb eset nagyon rossz, de jó heurisztika esetén javítható a teljesítménye

A* algoritmus

- A peremben a rendezést f()=g()+h() alapján végezzük: a legkisebb értéket vesszük ki.
- f() az adott állapoton átmenő legkisebb költségű utat becsli
- h()=0 esetén, gráfkeresés alkalmazásával Dijkstra algoritmust kapjuk
- h() elfogadható: ha nem becsli túl a a tényleges költség (pl. légvonalbeli távolság)
- Fakeresés, és elfogadható h() esetén, véges fa mellett a A* optimális
- Bizonyítás: G szuboptimális állapotre f(G)=g(G)+h(G)=g(G) > C >= (bármely n optimális útrészletre) g(n) + h(n) = f(n) → n hamarabb kerül kifejtésre

A* algoritmus

- Heurisztika konzisztens: bármely (n,n') élre: h(n) <= c(n, a, n') + h(n')
- Háromszög egyenlőtlenség → f(n) értékek bármely út mentén növekszenek
- Bizonyítás: f(n')=g(n') + h(n') = g(n) + c(n,a,n')+h(n') >= g(n) + h(n) = f(n)
- Következmény: gráfkeresést alkalmazva, véges fában, konzisztens heurisztikával A* optimális
- A* optimálisan hatékony, hiszen azokat a csúcsokat terjeszti ki, amire f() < C és ennyit biztosan ki kell, különben optimum kikerülhető
- Hátrány, nagy tárigény! Jó h() sokat segít.


(a) The initial state
Arad
366=0+366
(b) After expanding Arad
Arad
Sibiu
Timisoara
Zerind
393=140+253
447=118+329
449=75+374
(c) After expanding Sibiu
Arad
Sibiu
Timisoara
Zerind
447=118+329
449=75+374
Arad
Fagaras
Oradea
Rimnicu Vilcea
646=280+366
415=239+176
671=291+380
413=220+193
(d) After expanding Rimnicu Vilcea
Arad
Sibiu
Timisoara
Zerind
447=118+329
449=75+374
Arad
Fagaras
Oradea
Rimnicu Vilcea
646=280+366
415=239+176
671=291+380
Craiova
Pitesti
Sibiu
526=366+160
417=317+100
553=300+253
(e) After expanding Fagaras
Arad
Sibiu
Timisoara
Zerind
447=118+329
449=75+374
Arad
Fagaras
Oradea
Rimnicu Vilcea
646=280+366
671=291+380
Sibiu
Bucharest
Craiova
Pitesti
Sibiu
591=338+253
450=450+0
526=366+160
417=317+100
553=300+253
(f) After expanding Pitesti
Arad
Sibiu
Timisoara
Zerind
447=118+329
449=75+374
Arad
Fagaras
Oradea
Rimnicu Vilcea
646=280+366
671=291+380
Sibiu
Bucharest
Craiova
Pitesti
Sibiu
591=338+253
450=450+0
526=366+160
553=300+253
Bucharest
Craiova
Rimnicu Vilcea

Egyszerűsített memóriakorlátozott A*

- Memória korlátos
- Ha elfogyott:
  - Töröljük a legrosszabb utat, ha mind egyforma, akkor a legrosszabbat
  - Tegyünk az ősbe egy bejegyzést, hogy vissza tudjunk térni
- Nagy terek esetén ugrálhat az utak között.