Heurisztika: Feladattól származó, annak modelljében nem rögzített, a megoldást segítő speciális ismeret. Közvetlenül építjük be az algoritmusba, hogy annak hatékonysága és eredményessége javuljon, habár erre semmiféle garanciát nem nyújt. (Kombinatorikus robbanás elkerülése) A hatékonyság növelése alatt a memóriaigény és a futásidő csökkentését értjük.

Sok feladat fogalmazható át **útkeresési problémá**vá úgy, hogy a feladat modellje alapján megadunk egy olyan élsúlyozott irányított gráfot, amelyben adott csúcsból adott csúcsba vezető utak jelképezik a feladat egy-egy megoldását. Ezt a feladat **gráfreprezentáció**jának is szokás nevezni. Ebben a reprezentációs gráfban keresünk egy startcsúcsból kiinduló célcsúcsba futó utat, esetenként egy legolcsóbb ilyet. A reprezentációs gráffal megfogalmazott feladatok problématere (a megoldandó probléma lehetséges válaszainak halmaza) többnyire a startcsúcsból kiinduló utak halmaza. Néha a problématér a reprezentációs gráf csúcsainak halmaza. (lásd n-királynő probléma)

Kereső rendszerek fő részei

- **globális munkaterület**: tárolja a keresés során megszerzett és megőrzött ismeretet (egy részgráfot) (kezdeti érték ~ start csúcs, terminálási feltétel ~ célcsúcs)
- **keresés szabályok**: megváltoztatják a globális munkaterület tartalmát (előfeltétel, hatás)
- **vezérlési stratégia**: végrehajtható szabályok közül kiválaszt egy „megfelelőt” (általános elv + heurisztika)

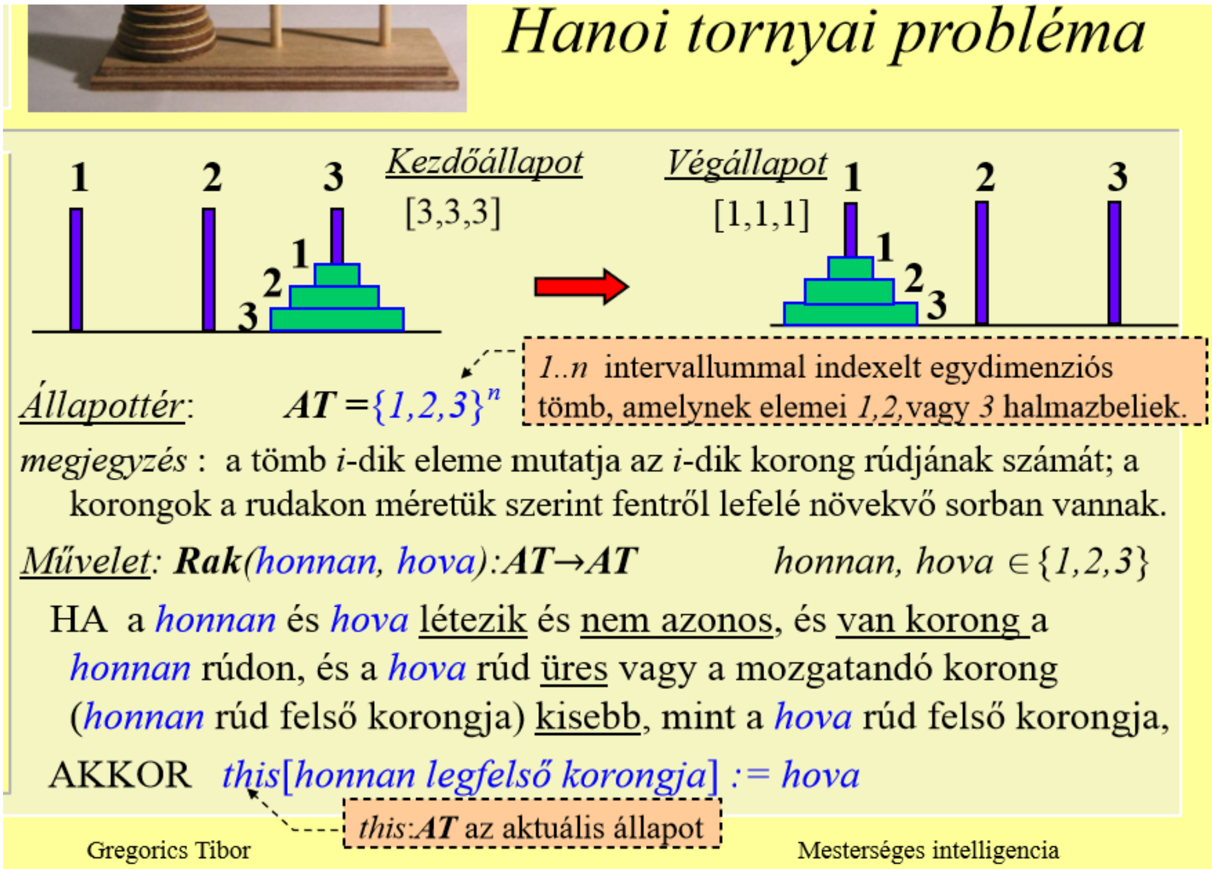
Algoritmusa:

```
Procedure KR
        ADAT := kezdeti érték
        while not terminálási feltétel(ADAT) loop
                select SZ from alkalmazható szabályok
                ADAT := SZ(ADAT)
endloop
end
```

Kereső rendszerek vizsgálata

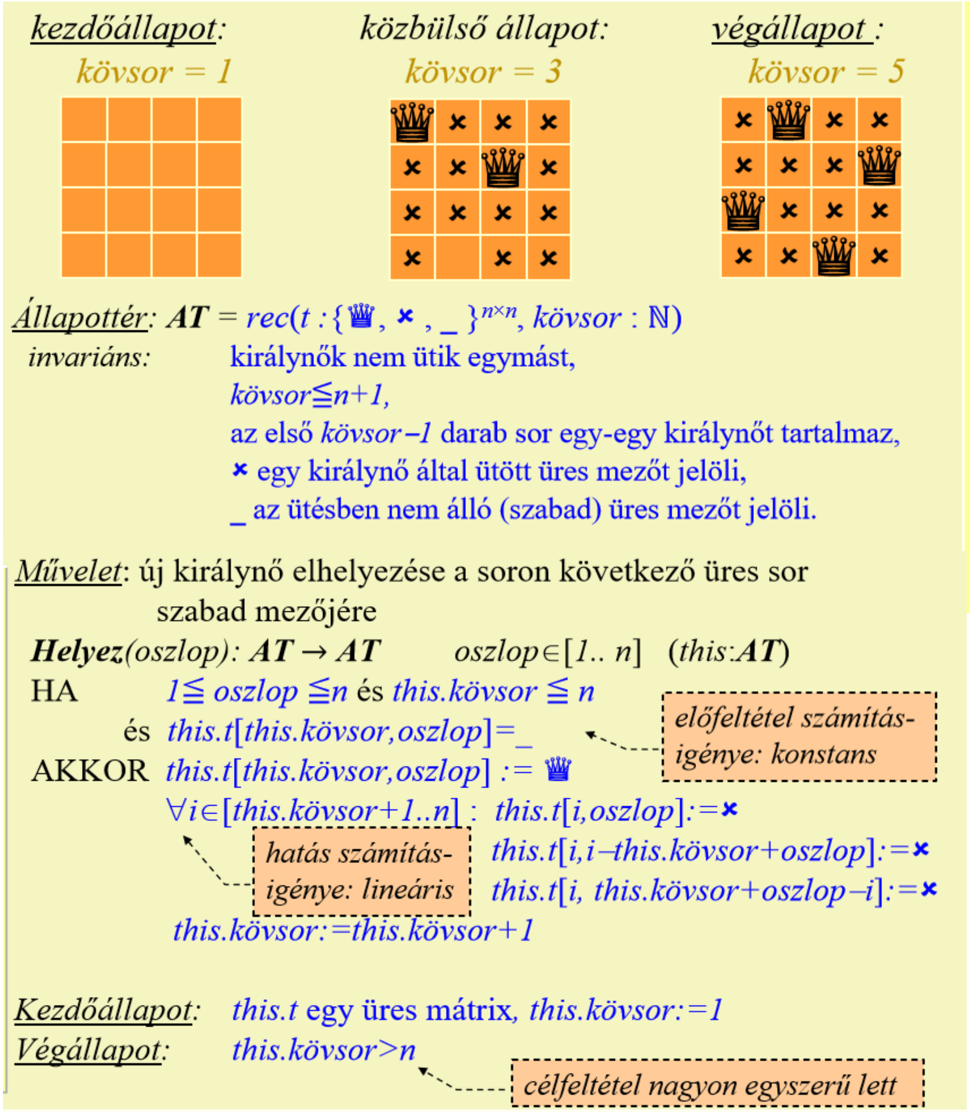
1. helyes-e (azaz korrekt választ ad-e)
2. teljes-e (minden esetben választ ad-e)
3. optimális-e (optimális megoldást ad-e)
4. idő bonyolultság
5. tár bonyolultság

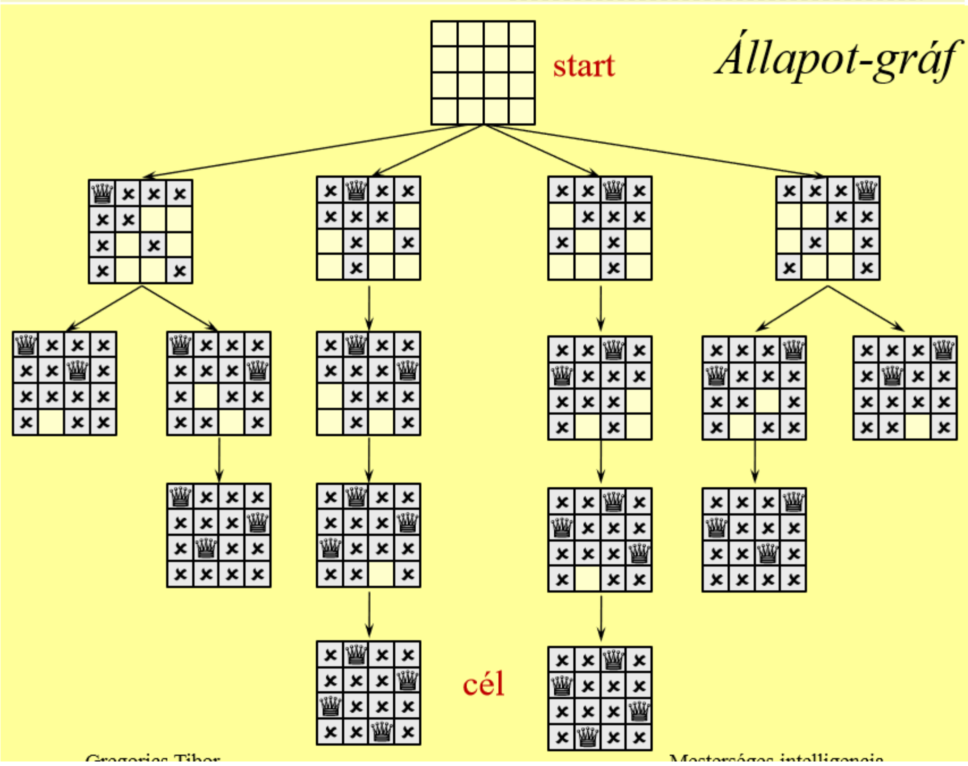
=============2.EA ============



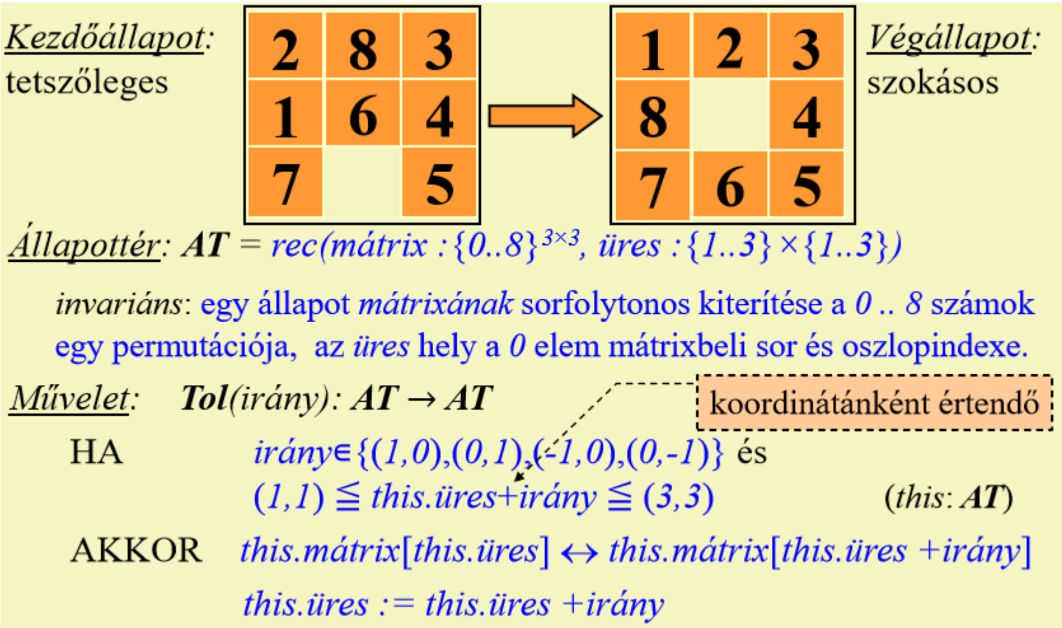
Hanoi problématér mérete: 2^(k+1)-1

N-királynő probléma





Tologató játék



Egy **probléma dekomponálása** során a problémát részproblémákra bontjuk, majd azokat tovább részletezzük, amíg nyilvánvalóan megoldható problémákat nem kapunk. Általában többféle módon is fel lehet bontani a problémákat.

Probléma dekompozíciónál egy adott problémát részproblémákra bontunk, és ezt addig ismételjük, amíg atomi problémákhoz nem érünk. *Hasonlóan a redukcióhoz*, operátorokat keresünk, de ezek nem halmazok közti hozzárendeléseket, hanem dekomponálást fognak végezni. A megoldás ebben az esetben sokszor nem egyértelmű. A dekomponálás előre, az operátorok műveleteinek elvégzése hátra felé történik.

Mit tartalmaz egy probléma dekompozíciós reprezentációja?

- A feladat részproblémáinak általános leírása
- A kiinduló probléma
- Az egyszerű problémák, amelyikről könnyen eldönthető, hogy megoldhatóak-e, vagy sem
- A dekomponáló műveleteket:
  D : probléma -> probléma$^+$
  D(p) = <p_1, ..., p_n>

ÉS/VAGY gráf
Olyan *R=(N,A)* élsúlyozott irányított hiper-gráf, ahol:

- *N:* a csúcsok halmaza, kiinduló probléma a startcsúcs, egyzserű problémák a célcsúcsok
- (A ⊆ {(n,M) ∈ NxN+ | 0 != |M|< ∞ } a hiper-élek halmaza,|M| a hiper-él rendje)
- c(n,M) az (n,M) költsége
- *A:* élköteg, ami a dekomponálandó probléma csúcsából dekomponálással kapott részproblémák csúcsaiba vezet. Élköteg élei közötti kapcsolatok fajtái:
  - *"ÉS":* a probléma megoldásához annak minden részproblémáját meg kell oldani
  - *"VAGY":* választhatunk, hogy melyik élköteg mentén oldjuk meg a problémát

Útkeresés és/vagy gráfban

- Egy ÉS/VAGY gráf startból induló hiper-útjainak (a potenciális megoldás gráfoknak) a bejárásai közönséges irányított utakként ábrázolhatók, és ezáltal egy közönséges irányított gráfot írnak le, amelynek csúcsai az eredeti ÉS/VAGY gráf csúcsainak véges sorozatai.
- Ha ebben a közönséges gráfban megoldási (azaz csupa célcsúcsot tartalmazó sorozatba vezető) utat találunk, akkor az egyben az eredeti ÉS/VAGY gráf megoldás-gráfja is lesz.

- Az ÉS/VAGY gráfbeli megoldás-gráf keresést tehát közönséges irányított gráfbeli keresést végző útkereső algoritmusokkal végezhetjük el.

Mi a hiperút?
n^{α}->M hiper-út, ($n \in N$, $M \in N^{+}$) olyan véges részgráf, amiben:

- M csúcsaiból nem indul hiper-él
- M-en kívüli csúcsból csak 1 db hiper-él indul
- Minden csúcs elérhető az *n* csúcsból egy közönséges irányított úton
- Hossza az éleinek a hossza, költsége definiálható

============3.EA============

Osztályozza a vezérlési stratégiákat!
Háromféle vezérlési stratégiát különböztetünk meg:

- Elsődleges vezérlési stratégiák: Független a feladattól, nem merít a feladat ismereteiből, sem a modell sajátosságaiból. Fajtái:
    - Nem módosítható, pl. lokális keresések, evolúciós algoritmus, rezolúció
    - Módosítható, pl. visszalépéses keresések, gráfkeresés
- Másodlagos vezérlési stratégiák: Nem függ a feladattól, de épít a modell sajátosságainak ismeretére
- Heurisztikák: A feladattól származó, annak modelljében nem rögzített, a megoldást segítő speciális ismereta

Lokális keresések

1. Globális munkaterület: A megoldandó útkeresési probléma (reprezentációs) gráfjának egyetlen (az aktuális) csúcsát és annak szűk környezetét tárolja (a globális munkaterületén).
   (Kezdetben az aktuális csúcs a startcsúcs, és a keresés akkor áll le, ha az aktuális csúcs a célcsúcs lesz.)
2. Keresési szabály: Az aktuális csúcsot minden lépésben annak környezetéből vett „jobb” csúccsal cseréli le.
3. Vezérlési stratégia: A „jobbság” eldöntéséhez egy kiértékelő (cél-, rátermettségi-, heurisztikus) függvényt használ, amely reményeink szerint annál jobb értéket ad egy csúcsra, minél közelebb esik az a célhoz.

Hegymászó algoritmus

```
akt := start
while akt ∉ T loop
        if Γ (akt) = null then return nem talált megoldást
        if Γ (akt)− π (akt) = null then akt:= π (akt)
        else akt := optf( Γ (akt)− π (akt) )
endloop
return akt
```

A hegymászó algoritmus lokális optimum hely körül, vagy ekvidisztans felületen lévő körön végtelen működésbe eshet. Ezt küszöböli ki a tabu keresés a memórianövelés által.
Nyilvántartjuk az optimális csúcsot, és a tabu halmazt (utolsú néhány érintett csúcs). Minden lépésben:

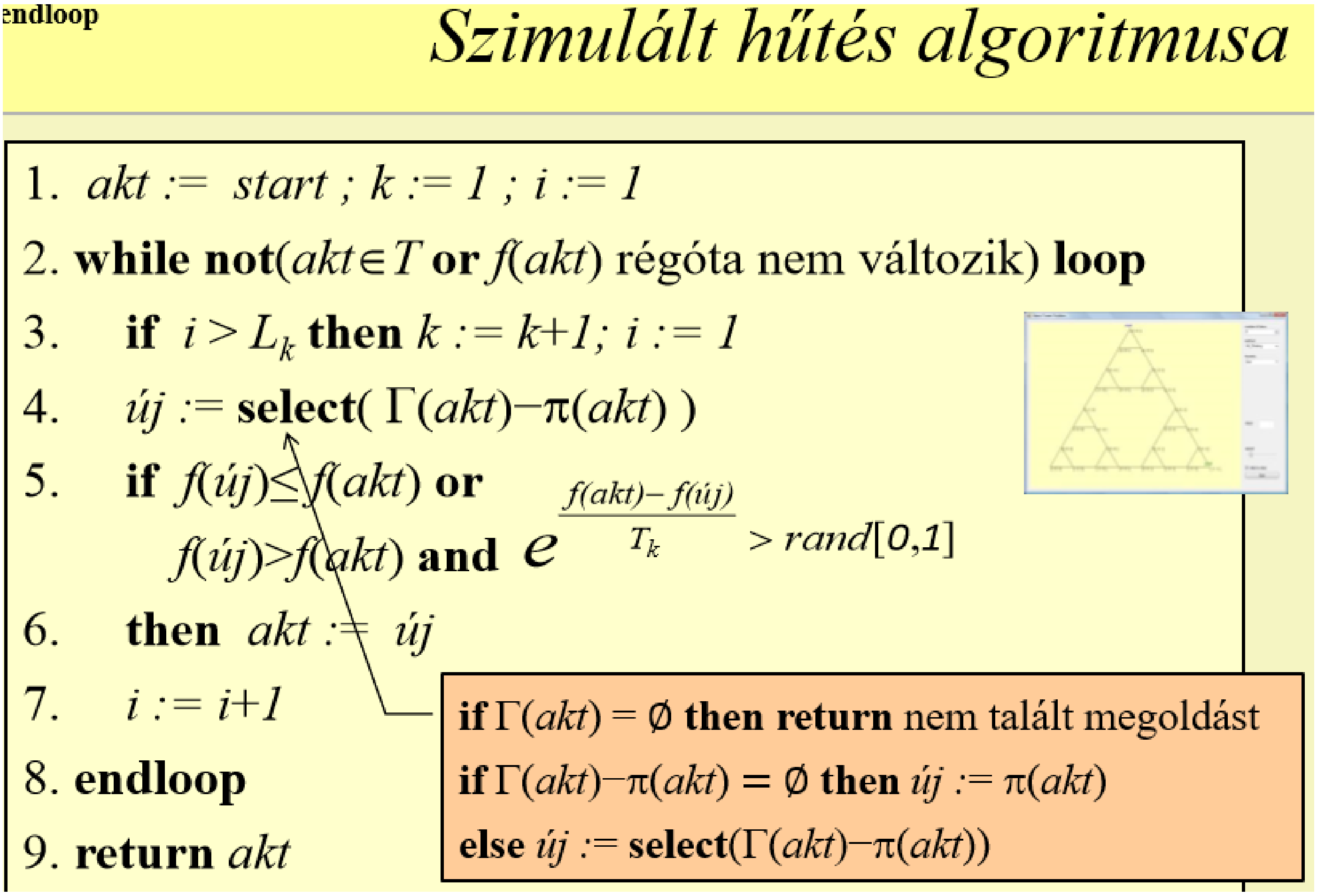
- Az aktuális csúcs legjobb gyerekére lép, ami nincs a tabu halmazban

- Ha az aktuális csúcs jobb, mint az optimális, akkor opt = akt;
- akt-ot hozzáadja a tabu halmazhoz.

terminálás: ha az opt célcsúcs / sokáig nem változik

Tabu hátrányai:

- a Tabu halmaz méretét kísérletezéssel kell belőni
- zsákutcába futva a nem-módosítható stratégia miatt beragad



============4.EA===========

Visszalépéses keresés munkaterülete, keresési szabályai, vezérlési stratégia

- Munkaterülete: egy út az aktuális csúcsba a startcsúcsból + a leágazó, még ki nem próbált élek
    - kezdetben a startcsúcsot tartalmazó nulla hosszúságú út
    - terminálás célcsúccsal vagy startcsúcsból való visszalépéssel
- Keresés szabályai: a nyilvántartott úthoz egy új él hozzáfűzése, vagy az utoló él törlése (visszalépés)
- Vezérlési stratégia: A visszalépés szabályát csak legvégső esetben alkalmazza
- A visszalépés feltételei:
    - Zsákutca: az aktuális csúcsból (azaz az aktuális út végpontjából) nem vezet tovább él
    - Zsákutca torkolat: az aktuális csúcsból kivezető utak nem vezettek célba
    - Kör: az aktuális csúcs szerepel már korábban is az aktuális úton
    - Mélységi korlát: az aktuális út hossza elér egy előre megadott értéket

Milyen eredményre képes a visszalépéses keresés első, illetve második változata?

- *VL1*: A visszalépés feltételei közül az első kettőt építjük be a kereső rendszerbe. Zsákutca és zsákutca torkolat visszalépési szabályokat implementáljuk
    - Véges körmentes gráfon mindig terminál és talál megoldást, ha van
- *VL2*: a visszalépés feltételei közül mindet beépítjük a kereső rendszerbe. A fenti kettőn kívül implementálja a kör és a mélységi korlát visszalépési feltételeket is
    - Véges δ-gráfon mindig terminál, és talál megoldást, ha van olyan, ami a mélységi korlátnál nem hosszabb. (Amúgy nincs MO)

visszalépéses keresés ELŐNYÖK

- mindig terminál
- talál megoldást (a mélységi korláton belül)
- könnyen implementálható
- kicsi memória igény

visszalépéses keresés HÁTRÁNYOK

- nem ad optimális megoldást. (iterációba szervezhető)
- kezdetben hozott rossz döntést csak sok visszalépés korrigál (visszaugrásos keresés)
- egy zsákutca részt többször is bejárhat a keresés

==================5.EA=====================

Gráfkeresés

globális munkaterülete: a reprezentációs gráf startcsúcsból kiinduló már feltárt útjait tárolja (tehát egy részgráfot), és külön az egyes utak végeit, a nyílt csúcsokat
• kiinduló értéke: a startcsúcs,
• terminálási feltétel: megjelenik egy célcsúcs vagy megakad az algoritmus.
keresés szabálya: egyik útvégi csúcs kiterjesztése
vezérlés stratégiája: a legkedvezőbb csúcs kiterjesztésére törekszik

Általános gráfkereső algoritmus részei

Jelölések:

- keresőgráf (G) : a reprezentációs gráf eddig bejárt és eltárolt része
- nyílt csúcsok halmaza (NYÍLT) : kiterjesztésre várakozó csúcsok, amelyeknek gyerekeit még nem vagy nem eléggé jól ismerjük
- kiterjesztett csúcsok halmaza (ZÁRT) : azok a csúcsok, amelyeknek a gyerekeit már előállítottuk
- kiértékelő függvény (f: NYÍLT → $\mathbb{R}$) : kiválasztja a megfelelő nyílt csúcsot kiterjesztésre

Általános gráf ker. eredményei

1. δ-gráfokban egy csúcsot véges sokszor terjeszt ki
2. *véges* δ-gráfban terminál
3. *véges* δ-gráfban, ha van megoldás, megtalálja és terminál

```
ADAT := kezdeti érték
while ¬ terminálási feltétel(ADAT) loop
    SELECT SZ FROM alkalmazható szabályok
    ADAT := SZ(ADAT)
endloop
```

Általános gráfkereső algoritmus

```
G := ({start}, ∅); NYÍLT := {start}; g(start) := 0; π(start) := nil
loop
    if empty(NYÍLT) then return nincs megoldás
    n := min_f(NYÍLT)
    if cél(n) then return megoldás
    NYÍLT := NYÍLT−{n}
    for ∀m∈Γ(n)−π(n) loop
        if m∉G or g(n)+c(n,m)<g(m) then
            π(m) := n; g(m) := g(n)+c(n,m); NYÍLT := NYÍLT∪{m}
    endloop
```

Nevezetes nem-informált algoritmusok

Elnevezés	Definíció	Eredmények
Mélységi	f=-g, c(n,m)=1	Végtelen gráfokban mélységi korláttal megoldást garantál
Szélességi	f=g, c(n,m)=1	Végtelen gráfokban a legrövidebb megoldást adja, egy csúcsot csak egyszer terjeszt ki.
Egyenletes	f=g	Végtelen gráfokban a legolcsóbb megoldást adja, egy csúcsot legfeljebb egyszer terjeszt ki.

Nevezetes heurisztikus algoritmusok

Elnevezés	Definíció	Eredmények
Előre tekintő gráfkeresés	f=h	Nincs említésre méltó tulajdonsága
A algoritmus	f=g+h, h≤h*	Megoldást ad, ha van, még végtelen gráfban is
A* algoritmus	f=g+h, h≤h*, h≥0	Optimális megoldást ad, ha van, még végtelen gráfokban is

Elnevezés	Definíció	Eredmények
A^c algoritmus	f=g+h, h≤h*, h≥0, h(n)-h(m)≤c(n,m)	Optimális megoldást ad, ha van és ugyanazt a csúcsot nem terjeszti ki kétszer

A* hatékonysága:

- Memória igény: Zárt csúcsok száma termináláskor jól jellemzi a kereső gráf méretét
- futási idő: Kiterjesztések száma a zárt csúcsok számához viszonyítva

A* memóriaigényre

Az A_1 (h_1 heurisztikával) és A_2 (h_2 heurisztikával) A^* *algoritmusok* közül az A_2 jobban informált, mint az A_1, ha minden $n \in N \setminus T$ csúcsra teljesül, hogy $h_1(n) \leq h_2(n)$.

Minél jobban (közelebbről) becsli (ha lehet, alulról) a heurisztika a h*-ot, várhatóan annál kisebb lesz a memória igénye.

A* futási ideje

- Zárt csúcsok száma: k = |ZÁRT|
- Alsókorlát: k - Egy monoton megszorításos heurisztika mellett egy csúcs legfeljebb csak egyszer terjesztődik ki, habár ettől még a kiterjesztett csúcsok száma igen sok is lehet (lásd egyenletes keresés)
- Felsőkorlát: 2k-1 – lásd. Martelli példáját

B algoritmus

A B algoritmus ugyanúgy működik, mint az A*, azzal a kivétellel, hogy egy árokhoz tartozó csúcsot csak egyszer terjeszt ki.

Futási idő elemzése:
• Legrosszabb esetben
 - minden zárt csúcs először küszöbcsúcsként terjesztődik ki. (Csökkenő kiértékelő függvény mellett egy csúcs csak egyszer, a legelső kiterjesztésekor lehet küszöb.)
 - Az i-dik árok legfeljebb az összes addigi i – 1 darab küszöbcsúcsot tartalmazhatja (a start csúcs nélkül).
• Így az összes kiterjesztések száma legfeljebb ½·k2

==================6.EA==================

Egy játékos nyerő stratégiája egy olyan elv, amelyet betartva az ellenfél minden lépésére tud olyan választ adni, hogy megnyerje a játékot. Ez nem egy konkrét győztes játszma, hanem olyan győztes játszmák összessége, amelyek közül egyet biztosan végig lehet játszani annak, aki rendelkezik a nyerő stratégiával. A két esélyes (győzelem vagy vereség) teljes információjú véges determinisztikus kétszemélyes játékokban az egyik játékos számára biztosan létezik nyerő stratégia. A három esélyes játékokban (van döntetlen is) a nem vesztő stratégiát lehet biztosan garantálni.

Minimax algoritmus

1. A játékfának az adott állás csúcsából leágazó részfáját felépítjük néhány szintig.
2. A részfa leveleit kiértékeljük a kiértékelő függvény segítségével.
3. Az értékeket felfuttatjuk a fában:

• A saját (MAX) szintek csúcsaihoz azok gyermekeinek maximumát: szülő := max (gyerek1,..., gyerekk)
• Az ellenfél (MIN) csúcsaihoz azok gyermekeinek minimumát: szülő := min (gyerek1,..., gyerekk)
4. Soron következő lépésünk ahhoz az álláshoz vezet, ahonnan a gyökérhez felkerült a legnagyobb érték.

Sorolja fel, milyen módosításait ismerte meg a minimax algoritmusnak, és írja melléjük, hogy ezek milyen szempontból javítanak annak működésén?

- Átlagoló kiértékelés
  - A kiértékelő függvény esetleges tévedéseit simítja ki
- Váltakozó mélységű kiértékelés
  - A kiértékelő függvény minden ágon reális értéket mutat
- Szelektív kiértékelés
  - A memóriaigényt csökkenti (csak a lényeges lépéseket értékeli)

Szelektív kiértékelés

Célja a memória-igény csökkentése.
Elkülönítjük a lényeges és lényegtelen lépéseket, és csak a lényeges lépéseknek megfelelő részfát építjük fel.
Ez a szétválasztás heurisztikus ismeretekre épül.

Negamax algo

könnyebb implementálni.

– Kezdetben (–1)-gyel szorozzuk azon levélcsúcsok értékeit, amelyek az ellenfél (MIN) szintjein vannak, majd – Az értékek felfuttatásánál minden szinten az alábbi módon számoljuk a belső csúcsok értékeit: szülő := max(– gyerek1,..., – gyerekk)

Alfa-béta algoritmus

- Visszalépéses algoritmus segítségével járjuk be a részfát (olyan mélységi bejárás, amely mindig csak egy utat tárol). Az aktuális úton fekvő csúcsok ideiglenes értékei:
  – a MAX szintjein α érték: ennél rosszabb értékű állásba innen már nem juthatunk
  – A MIN szintjein β érték: ennél jobb értékű állásba onnan már nem juthatunk
- Lefelé haladva a fában $\alpha := -\infty$, és $\beta := +\infty$.
- Visszalépéskor az éppen elhagyott (gyermek) csúcs értéke (felhozott érték) módosíthatja a szülő csúcs értékét:
  – a MAX szintjein: α := *max(felhozott érték,* α*)*
  – a MIN szintjein: β := *min(felhozott érték,* β*)*
- Vágás: ha az úton van olyan α és β, hogy $\alpha \geq \beta$.

Memória igény: csak egy utat tárol. Futási idő: a vágások miatt sokkal jobb, mint a minimax módszeré

Evolúciós algoritmus lépései és jellemzése, inicializálás, terminálás
Az algoritmus lépései:

- *Szelekció:* a populációból kiválasztunk néhány -- lehetőleg rátermett -- egyedet szülőknek
- *Rekombináció:* a szülőkből gyerekek készülnek úgy, hogy mindkét szülő tulajdonságait megöröklik
- *Mutáció:* az utódok tulajdonságait kis mértékben megváltoztatjuk
- *Visszahelyezés:* új populációt alakítunk ki az utódokból és a régi populációból

Inicializálás: Kialakítjuk a kezdeti populációt
Terminálás: Addig futtatjuk az algoritmust, amíg el nem érjük a kívánt állapotot
• ha a célegyed megjelenik a populációban
• ha a populáció egyesített rátermettségi függvény értéke egy ideje nem változik.

Algoritmusa:

Procedure EA
populáció := kezdeti populáció
while terminálási feltétel nem igaz loop
  szülők := szelekció(populáció)
  utódok := rekombináció(szülők)
  utódok := mutáció(utódok)
  populáció := visszahelyezés(populáció, utódok)
endloop

Szelekció

Célja: a rátermett egyedek kiválasztása úgy, hogy a rosszabbak kiválasztása is kapjon esélyt. Pl: Rátermettség arányos, Rangsorolásos, Versengő, Csonkolásos v. selejtezős

Rekombináció

A feladata az, hogy adott szülő-egyedekből olyan utódokat hozzon létre, amelyek a szüleik tulajdonságait "öröklik".
− Keresztezés: véletlen kiválasztott pozíción jelcsoportok (gének) vagy jelek cseréje
− Rekombináció: a szülő egyedek megfelelő jeleinek kombinálásával kapjuk az utód megfelelő jelét

Pl rekombinációs operátorra:

- Egypontos keresztezés
- Többpontos keresztezés
- Egyenletes keresztezés

Mutáció

A mutáció egy egyed (utód) kis mértékű véletlen változtatását végzi.

Visszahelyezés

A populáció az utódokkal történő frissítése: kiválasztja a populáció lecserélendő egyedeit, és azok helyére az utódokat teszi.
utódképzési ráta(u) = utódok száma/populáció száma
visszahelyezési ráta(v) = lecserélendő egyedek száma/populáció száma

- Ha *u=v:* feltétlenül csere

- Ha $u<v$: egy utód több példányban is bekerülhet
- Ha $u>v$: az utódok közül szelektál