4.

Függvények, görbék, felületek leírása és számítógépes ábrázolása.

Függvények leírása és számítógépes ábrázolása.

Függvény segítségével jeleníthető meg egy ábra: minden x -hez tartozik egy y érték. A grafikon elméleti, ezt csak közelíthetjük, mivel a gyakorlatban behelyettesítünk, és az adott értékeket ábrázoljuk, majd ezeket összekötjük. A gyakorlati függvény tehát szakaszokból áll. A behelyettesített értékek számától, sűrűségétől függ, hogy hány ponton lesz törött a vonal. Számítógép annyi értéket helyettesít, hogy legalább 1 px -t ugorjon (ami függ a képernyő felbontásától is), így görbének látszik, viszont mindig töröttvonal marad.

1. Polinomiális függvények

$a_n \times x^n + a_{n-1} \times x^{n-1} + ... + a_1 \times x^1 + a_0 \times x^0$

Polinom: x polinomját valamilyen hatványon megszorozzuk (n:fokszám, a:együtthatók, x:változó), konstans függvény: 0-adfokú polinom.

f(x)=ax+b 1.fokú polinom, ahol a:meredekség, b:hol metszi f(x)-et (vagy y tengelyt) (mivel a értéke nem tud elég nagy lenni, ezért függőlegest nem tudunk ábrázolni.
Elképzeléshez kitalálni függvényt:
-ha 2 pont van megadva, éppen egyértelmű
- 1 ponttal aluldeterminált
- több, mint 2 pont esetén ha 1 egyenesre esnek, könnyű helyzet, ha nem, akkor egyik ponton sem megy át, hanem minden pontot közelít a legkisebb eltéréssel.

$f(x)=ax^2+bx+c$ 2.fokú polinom
Elképzeléshez függvényt találni:
- próbálkozás különböző a, b és c értékekkel
- kideríteni hol a csúcspont
- megadott minimum 3 ponttal egyenletrendszer megoldása

Hullám ábrázolása esetén is célszerűbb polinomiális függvény használata szögfüggvény helyett, mert az x-hez tartozó f(x) értékek keresése szögfüggvény esetén időigényes, lassú, és apró változtatástól is összeomlik.

$f(x)=ax^3+bx^2+cx+d$ 3.fokú polinom
Megtalálása próbálkozással vagy minimum 4 pont megadásával (4 ismeretlen miatt).
Ábrázolható 2 parabolával is, de úgy törés lesz a csatlakozási ponton.

2. Implicit függvények

Ha nincs egyértelmű hozzárendelés, tehát x-hez több y érték tartozik, pl egy kör, vagy egy függőleges esetén
Eddig: $x \rightarrow f(x)$, ahol $R \rightarrow R$
Implicit esetben: $x,y \rightarrow F(x,y)$, ahol $R \times R \rightarrow R$
$F(x,y) = 0$

Minden eddigi függvény átalakítható ilyenné:

explicit: $y= 3x^2+5x+1$ $y=f(x)$
implicit: $0=3x^2+5x+1-y$ $0=f(x)-y$

tehát pl: $x^2+y^2=9$
$x^2+y^2-9=0$
$F(x,y)=0$

3. Vektor alapján ábrázolok függvényt

Ezek a paraméteres vektorfüggvények, vagy paraméterekkel megadott vektor görbék.
Mozgó helyvektor végpontjai rajzolják a görbét, tehát idő függvényében vektort írunk le.
Értelmezési tartomány: idő.

$r(t): [a,b] \rightarrow V^2$, ahol V^2 a szabad vektorok halmaza, a t paraméter valós érték, [a,b] intervallum az ábrázolás idejének kezdő és végpontja közötti intervallum, a függvény értéke pedig a vektor.

2 függvényként is lehet kezelni:
Van (x,y) koordinátám
x(t) és y(t) is függvénye a t-nek
$x(t): [a,b] \rightarrow R$, $y(t): [a,b] \rightarrow R$
A vektorfüggvény tehát koordináta függvények együttese. Minden t (azaz [a,b] intervallumon értelmezett idő) értékre x és y koordinátákat ad.

Ez a módszer egyesíti az eddigiek előnyeit:
- könnyű kirajzolás (pl y=f(x) esetén)
- a görbe maga alá is görbülhet, bármilyen alakú lehet (F(x,y)=0 esetén)

- ## Görbék leírása és számítógépes ábrázolása.

- **Implicit típusú görbék**
  Behelyettesítés esetén bonyolult egyenletet kapunk (Pl $4x^5-5x^3y^7+\sin(x)+\cos(y^2)-7=0$)
  Ábrázolás módja: (0,0) behelyettesítése, pontonkénti (pixelenkénti) kiértékelés: ha az eredmény rajta van a vonalon, akkor 0-val egyenlő, egyébként nincs rajta.
  Minden pixelt be kell helyettesíteni. Lassú kiértékelés, és nem egyértelmű, akár a szoftver is elbizonytalanodhat, ezért ritkán használjuk computer grafikában.
  Tulajdonságai:
  -az (x_0,y_0) koordinátájú pont akkor és csakis akkor illeszkedik a görbére, ha $F(x_0,y_0)=0$
  - ha $F(x_0,y_0)>0$, akkor az görbe fölött helyezkedik el (fel és jobbra)
  - ha $F(x_0,y_0)<0$, akkor az görbe alatt helyezkedik el (le és balra)
  Értékek behelyettesítése a távolságot is mutatja (minél nagyobb az érték, annál távolabb van a pont)

- **Polinommal megadott görbe:**
  elsőfokú polinom: 3x-4y-7 = 0, egyenes → elsőrendű görbe
  másodfokú polinom: $x^2+y^2-9=0$, kör → másodrendű görbe
  $x^2-y=0$, parabola → másodrendű görbe
  n-edfokú polinom: $x^n-3y^n+...=0$ → n-edrendű görbe

- Csak polinommal megadott lehet valahanyadrendű görbe vagy algebrai görbe (mivel a függvény is algebrai, nem analitikus. Pl. sin(x) analitikus lenne)

- Egy n-edrendű és egy m-edrendű görbének legfeljebb n×m darab látható metszéspontja lehet (ezen mindkét görbe átmegy)

Vektorral megadott görbe:
Mozgó helyvektor végpontjai rajzolják a görbét, tehát idő függvényében vektort írunk le. (Továbbiak: Függvények, 3.pont) Ezek a legáltalánosabban használt görberajzoló függvények.

Térbeli görbe rajzolására alkalmatlanok az implicit és az explicit függvények.
Csak vektorfüggvénnyel rajzolhatóak.
$r(t)$: $[a,b] \rightarrow V^3$, ahol V^3 a térbeli vektorok halmaza
$x(t)$: $[a,b] \rightarrow R$, $y(t)$: $[a,b] \rightarrow R$, $z(t)$: $[a,b] \rightarrow R$

Görbe alapján írjunk képletet
Egyenes esetén: $r(t)$: $x(t)=a_1t+a_0$
$y(t)=b_1t+b_0$
Keressük a_0, a_1, b_0, b_1 számokat
Legyen t=0 pillanat az (1,6) pontnál, ez $r(0)$.
Legyen t=1 pillanat az (5,4) pontnál, ez $r(1)$.
t=0 => $r(0)$
x=1 $x(0)=a_1\times0+a_0=1$ => $a_0=1$
y=6 $y(0)=b_1\times0+b_0=6$ => $b_0=6$

t=1 => $r(1)$
x=1 $x(1)=a_1\times1+a_0=5=a_1+1=5$ => $a_1=4$
y=6 $y(1)=b_1\times1+b_0=4= b_1+6=4$ => $b_1=-2$
Az egyenes leírása tehát: $r(t)$: $x(t)=4t+1$
$y(t)=-2t+6$

Algoritmikusan: $x(t)=(x_1-x_0)\times t+x_0$
$y(t)=(y_1-y_0)\times t+y_0$
Ahol x0, x1, y0, y1 az egér által meghatározott pontok.

Három pontra illesztett görbe: $r(t)$: $x(t)=a_2t^2+a_1t+a_0$
$y(t)=b_2t^2+b_1t+b_0$
Keressük a_0, a_1,a_2, b_0, b_1, b_2 számokat
Legyen t=0 pillanat az (1,6) pontnál, ez $r(0)$.
Legyen t=0.7 pillanat az (5,2) pontnál, ez $r(0.7)$.
Legyen t=1 pillanat az (7,4) pontnál, ez $r(1)$.
t=0 => $r(0)$
$x(0)=a_20^2+a_10+a_0=1$ => $a_0=1$
$y(0)=b_20^2+b_10+b_0=6$ => $b_0=6$

t=0.7 => $r(0.7)$
$x(0.7)=a_21^2+a_11+a_0=5= a_20.7^2+a_10.7+1=5$
$y(0.7)=b_21^2+b_11+b_0=2= b_20.7^2+b_10.7+6=2$
t=1 => $r(1)$
$x(1)= a_21^2+a_11+a_0=7$
$y(1)= b_21^2+b_11+b_0=4$

t=0.7 és t=1 alapján egyenetrendszer, ami a maradék ismeretleneket megadja

Pontok alapjan torteno abrazolas:

n darab pontra fektetek görbét, tehát megadok n db (x_i, y_i) koordinátájú pontot. Vektoros ábrázoláshoz megadom a t_i értékeit úgy, hogy a t_i görbepont éppen az (x_i,y_i) pontra essen. Ennek a görbének az egyenletét keressük.

Egyenlet megoldásához szükség van az egyenletek fokszámának meghatározására: n db ponthoz n-1-ed fokú polinomra van szükség.

Görbe: $r(t)$

$$x(t)=a_{n-1}t^{n-1}+a_{n-2}t^{n-2}+...+a_1t+a_0$$
$$y(t)=b_{n-1}t^{n-1}+b_{n-2}t^{n-2}+...+b_1t+b_0$$

egyenletek pontonként, ahol ti-t ismerem:

$r(t_1)$

$$x(t_1)=x_1=a_{n-1}t_1^{n-1}+a_{n-2}t_1^{n-2}+...+a_1t_1+a_0$$
$$y(t_1)=y_1=b_{n-1}t_1^{n-1}+b_{n-2}t_1^{n-2}+...+b_1t_1+b_0$$

$r(t_2)$

$$x(t_2)=x_2=a_{n-1}t_2^{n-1}+a_{n-2}t_2^{n-2}+...+a_1t_2+a_0$$
$$y(t_2)=y_2=b_{n-1}t_2^{n-1}+b_{n-2}t_2^{n-2}+...+b_1t_2+b_0$$

$r(t_n)$

$$x(t_n)=x_n=a_{n-1}t_n^{n-1}+a_{n-2}t_n^{n-2}+...+a_1t_n+a_0$$
$$y(t_n)=y_n=b_{n-1}t_n^{n-1}+b_{n-2}t_n^{n-2}+...+b_1t_n+b_0$$

n pont esetén (n-1)×2 egyenlet, melyek megoldása behelyettesítéssel nem megvalósítható.
Megoldása **Lagrange-interpoláció**val (megadott pontokon átmenő görbe egyenlete):

$a_0, a_1, \ldots, a_{n-1}$, és $b_0, b_1, \ldots, b_{n-1}$ skalárok, melyekkel a görbe felírható így:
$r(t)$

$$x(t)=a_{n-1}t^{n-1}+a_{n-2}t^{n-2}+...+a_1t+a_0$$
$$y(t)=b_{n-1}t^{n-1}+b_{n-2}t^{n-2}+...+b_1t+b_0$$

Megoldása Gauss eliminációval (átló alatt kinullázom). N db ismeretlen és n db független egyenlet esetén kapok csak egyértelmű megoldást (jól meghatározott). Kevesebb egyenlet esetén végtelen sok megoldás (alulhatározott), több esetén nem tudom az összes egyenletet igazzá tenni, tehát nincs megoldás(túlhatározott). Mivel ez polinom, ezért gyorsan és pontosan dolgozhatok vele, viszont paraméterezése nehézkes, emellett oszcillál, tehát ha a görbe közepén egy egyenes van, akkor a pontok az egyenes körül oszcillálva hullámot adnak.

<u>Módszerek paraméterek hozzárendelésére a pontokhoz:</u>

1. Uniform módszer: [0,1] intervallumot egyenlő részekre osztjuk. Maga alá görbülő görbékre nem alkalmazható.
2. Húrhossz szerinti paraméterezés: intervallum felosztása a pontok távolságának arányában.

Hermite-ív:

Ha a görbém egy része egyenes, akkor tudnom kell, hogy adott ponton merre tartson a görbe. Ehhez az adott pont érintőjére van szükség, tehát pontonként függvényre és deriváltra is szükség van.

Mivel 2 megadott pont esetén 4 feltétel teljesül (2 pont és 2 vektor), 3-adfokú polinomra van szükség 2 pontra fektetett görbe esetén.

1-1 harmadfokú polinommal megadott koordináta függvényt keresünk az alábbi alakban:

r(t)

x(t)

y(t)

Ehhez adottak:

$P_0(p_0x, p_0y)$ $v_0(v_0x, v_0y)$

$P_1(p_1x, p_1y)$ $v_1(v_1x, v_1y)$

Feltételek:

$r(0)= P_0$	→	$x(0)= p_0x$	$y(0)= p_0y$
$r(1)= P_1$	→	$x(1)= p_1x$	$y(1)= p_1y$
$r'(0)= v_0$	→	$x'(0)= v_0x$	$y'(0)= v_0y$
$r'(1)= v_1$	→	$x'(1)= v_1x$	$y'(1)= v_1y$

Polinomiális függvényeket keresünk, koordináta függvényenként 4 ismeretlenünk van, tehát összesen 4 egyenletet keresünk.

r(t)

$x(t)=a_3t^3+a_2t^2+a_1t+a_0$

$y(t)=b_3t^3+b_2t^2+b_1t+b_0$

r'(t)

$x'(t)= a_3\times3t^2+a_2\times2t+a_1$

$y'(t)=b_3\times3t^2+b_2\times2t+b_1$

x a 0 helyen vegye fel az általunk megadott koordinátát:

$x(0)= p_0x$	→	$\mathbf{a_0= p_0x}$
$x(1)= p_1x$	→	$a_3+a_2+a_1+a_0=p_1x$
$x'(0)= v_0x$	→	$\mathbf{a_1= p_1x}$
$x'(1)= v_1x$	→	$3a_3+2a_2+a_1=v_1x$

Megoldás:

r(t)

$x(t)=(-2p_1x+2p_0x+v_0x+v_1x)\times t^3+(3p_1x-3p_0x-2v_0x-v_1x)\times t^2+v_0x\times t+p_0x$

$y(t)=(-2p_1y+2p_0y+v_0y+v_1y)\times t^3+(3p_1y-3p_0y-2v_0y-v_1y)\times t^2+v_0y\times t+p_0y$

Átalakítva pontokra és vektorokra:

$r(t)=(-2p_1+2p_0+v_0+v_1)\times t^3+(3p_1-3p_0-2v_0-v_1)\times t^2+v_0\times t+p_0$

$r(t)=(2t^3-3t^2+1)p_0+(-2t^3+3t^2)p_1+(t^3-2t^2+t)v_0+(t^3-t^2)v_1$

$r(t)=H_0(t)\times p_0+H_1(t)\times p_1+H_2(t)\times v_0+H_3(t)\times v_1$

Computer grafikában minden görbe polinom, paraméteres előállítású, ahol geometriai adatokat (pontok és érintők) szorzunk meg polinomokkal.
Hermite-ív problémái: hosszú vektor esetén visszahúzódás, és nem lehet megjósolni mekkora vektornál torzul el a görbe.

Bézier görbe

Ennek a továbbfejlesztése a **Bézier görbe**, ahol vektorok helyett újabb pontokkal kontrollálom a görbét, ezek a kontroll pontok. 4 pont határoz meg egy töröttvonalat, mivel a 2 kiegészítő pont is a körülbelüli haladást mutatja.

Hermite-ívhez képest:

$v_0=3(P_1-P_0)$

$v_1=3(P_3-P_2)$

$r(t)=(2t^3-3t^2+1)p_0+(-2t^3+3t^2)p_3+(t^3-2t^2+t)\times 3(p_1-p_0)+(t^3-t^2)\times 3(p_3-p_2)$

$r(t)=\binom{3}{0}t^0\,(1-t)^3p_0+\binom{3}{1}t^1\,(1-t)^2p_1+\binom{3}{2}t^2\,(1-t)^1p_2+\binom{3}{3}t^3\,(1-t)^0p_3$

Bézier görbe n+1 darab pontra n-edfokú polinomokkal:

$r(t)=\binom{n}{0}t^0\,(1-t)^np_0+\binom{n}{1}t^1\,(1-t)^{n-1}p_1+...+\binom{n}{n-1}t^{n-1}\,(1-t)^1p_{n-1}+\binom{n}{n}t^n\,(1-t)^np_n$

$$r(t)=\sum_{i=0}^{n}\binom{n}{i}t^i(1-t)^{n-i}\times p_i$$

- ## Felületek leírása és számítógépes ábrázolása.

- 1. **explicit** előállítás z=f(x,y)
  minden (x,y) pontra ad egy z értéket. $R\times R\rightarrow R$
  x,y síkon megkeresem a pontot, majd z irányába tolom.
  1. összes x-et behelyettesítem, y=0
  2. összes y-t behelyettesítem, x=0
  3. a többi pontot is behelyettesítem, kiszámolom
  Dróthálós megjelenítés:
  Minden pontot ábrázolni kell, mert a köztes pontokat nem határozzák meg a szélsők.
  Így a felületi görbéket ábrázolom, nem magát a felületet
  Előny: könnyű ábrázolás
  Hátrány: maga alá görbülő felületet nem tud ábrázolni

- 2. **implicit** F(x,y,z) = 0
  1 (x,y) értékhez több z érték társul, tehát egymás alá görbül a felület.
  A pont 3 koordinátája egy nagy egyenletben egyesül. Azok a pontok vannak a felületen, ahol kiértékelés során 0-t kapunk.
  Előnye: maga alá görbülő felületet is tud ábrázolni
  Hátránya: pontonkénti kiértékelés
  Ha adott az F(x,y,z) n-edfokú polinom, akkor az F(x,y,z)=0 egyenletet kielégítő pontok összességét n-edrendű (vagy algebrai) felületnek nevezzük.

- Pl n=1: ax+by+cz+d=0 egy sík
  n=2: $ax^2+by^2+cz^2+dxy+exz+fyz+gx+hy+jz+k = 0$, ahol dxy,exz és fyz is másodfokú tagok, teház ha nincs x^2,y^2,z^2, akkor is másodfokú.
  Metszéspontok: felület+görbe, felület+felület.
  Egy n-edrendű felületnek és egy m-edrendű görbének n×m látható metszéspontja lehet. Tehát egy felület egyenletének fokszámát eldönthetem úgy, hogy egyenessel metszem, és a metszéspontok

száma megadja a felület egyenletének fokszámát.
Egy n-edrendű és egy m-edrendű felület metszésvonala egy m×n-edrendű görbe.

3. **paraméteres** megadás
r: [a,b]×[c,d] (a két időintervallum Descartes szorzata) $\rightarrow V^3$
x: [a,b]×[c,d] $\rightarrow$R, y: [a,b]×[c,d] $\rightarrow$R, z: [a,b]×[c,d] $\rightarrow$R
Míg a görbék ábrázolása során pontokat kötök össze, itt paraméter vonalakkal rácsozom be
Lépései ([a,b] intervallumot az u tengelyen, [c,d] intervallumot a v tengelyen ábrázolva):
1. u értéke fix, a v értéke pedig c-ről d-re egységenként nő. (1. paraméter állandó, 2. változó)
Eredményei pontok, amelyeket összekötve görbét kapok
2. új, de szintén fix u érték, míg a-ból b-be nem érünk (az intervallumokon minél kisebb a lépték, annál simább a felület, annál pontosabb a végeredmény)
3. az első 2 lépést a v értékein is megismételni

Előnyök, hátrányok (implicit vs paraméteres)

	IMPLICIT	PARAMÉTERES
MEGJELENÍTÉS	-	+
KOORDINÁTA ILLESZKEDÉSE A FELÜLETRE	+, behelyettesítés után 0 eredmény rajta van nem 0, nincs	–, 3 egyenletből álló 2 ismeretlenes egyenletrendszer

Előfordul, hogy az egyik implicit, a másik paraméteres formában jó.

Felület érzékeltetése: rácsvonalakkal, színezéssel, szintvonalakkal

Problémák reprezentálása állapottéren. A megoldás keresése visszalépéssel. Szisztematikus és heurisztikus gráfkereső eljárások: a szélességi a mélységi és az A algoritmusok. Kétszemélyes játékok és reprezentálásuk. A nyerő stratégia. Lépésajánló algoritmusok.

Problémák reprezentálása állapottéren

Legyen adott egy probléma, amit jelöljünk p-vel.

Ha a megadott jellemzők épp rendre a h1, … , hm értékekkel rendelkeznek azt mondjuk, hogy p világa (h1, … , hm) érték m-essel leírt állapotban (state) van.

A világunk állapotainak halmaza az állapottér (state space).

Jelölje az i-edik jellemző által felvehető értékek halmazát Hi. Ekkor p állapotai elemei a H1 X … X Hm 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.

Az 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 C részhalmaza A célállapotok halmazát. Megadása kétféleképpen történhet: felsorolással, célfeltételek megadásával.

Hogy a célállapotba jussunk, 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 O véges halmazából o egy operátort.

Legyen p egy probléma. Azt mondjuk, hogy a p problémát állapottér-reprezentáltuk, ha megadtuk az (A, kezdő, C, O) négyest, azaz

- az A (nem nulla) halmazt, a probléma állapottérét
- a kezdő (eleme A) kezdőállapotot
- a célállapotok C részhalmaza A halmazát
- az operátorok O (nem nulla) véges halmazát

Jelölése: p = (A, kezdő, C, O)

A megoldás keresése visszalépéssel

A mélységi keresés visszalépéses keresésnek (backtracking search) nevezett változata még kevesebb memóriát használ. A visszalépéses keresés az összes követő helyett

egyidejűleg csak egy követőt generál. Minden részben kifejtett csomópont emlékszik, melyik követője jön a legközelebb. Ily módon csak O(m) memóriára van szükség, O(bm) helyett. A visszalépéses keresés még egy memória- (és idő-) spóroló trükkhöz folyamodik. Az ötlet a követő csomópont generálása az aktuális állapot módosításával, anélkül hogy az állapotot átmásolnánk. Ezzel a memóriaszükséglet egy állapotra és O(m) cselekvésre redukálódik. Ahhoz, hogy az ötlet működjön, amikor visszalépünk, hogy a következő követőt generáljuk, mindegyik módosítást vissza kell tudnunk csinálni. Nagy állapottérrel rendelkező problémák esetén, mint például robot-összeszerelés esetén, az ilyen módszerek lényegesek a sikerességhez. Rossz választás esetében azonban elakadhat, nem optimális és nem teljes.

Szisztematikus és heurisztikus gráfkereső eljárások: a szélességi a mélységi és az A algoritmusok

Szélességi kereső

A szélességi keresés (breadth-first search) egy egyszerű keresési stratégia, ahol először a gyökércsomópontot fejtjük ki, majd a következő lépésben az összes a gyökércsomópontból generált csomópontot, majd azok követőit stb. Általánosságban a keresési stratégia minden adott mélységű csomópontot hamarabb fejt ki, mielőtt bármelyik, egy szinttel lejjebbi csomópontot kifejtené.

FIFO (first in first out) sor.

- Teljes, minden véges számú állapot érintésével elérhető állapotot véges időben elér.
- Általában nem optimális, de akkor pl. igen, ha a költség a mélység nem csökkenő függvénye.
- időigény = tárigény = O(bd+1)

A komplexitás exponenciális, tehát nem várható, hogy skálázódik: nagyon kis mélységek jönnek szóba, d = 10 körül. A memória egyébként előbb fogy el.

Mélységi kereső

A mélységi keresés (depth-first search) mindig a keresési fa aktuális peremében a legmélyebben fekvő csomópontot fejti ki. A keresés azonnal a fa legmélyebb szintjére jut el, ahol a csomópontoknak már nincsenek követőik. Kifejtésüket követően kikerülnek a peremből és a keresés „visszalép” ahhoz a következő legmélyebben fekvő csomóponthoz, amelynek vannak még ki nem fejtett követői. Szerény tárigényű, a gyökércsomóponttól egy levélcsomópontig vezető utat kell tárolnia, kiegészítve az út minden egyes csomópontja melletti kifejtetlen csomópontokkal.

LIFO (last in first out) verem.

- Teljes, ha a keresési fa véges mélységű (azaz véges, hiszen b véges). Egyébként nem.

- Nem optimális.
- időigény: a legrosszabb eset O(bm) (nagyon rossz, sőt, lehet végtelen), tárigény: legrosszabb esetben O(bm) (ez viszont biztató, mert legalább nem exponenciális).

Az A algoritmus

A legjobbat-először keresés leginkább ismert változata az A* keresés (a kiejtése 'A csillag'). A csomópontokat úgy értékeli ki, hogy összekombinálja g(n) értékét – az aktuális csomópontig megtett út költsége – és h(n) értékét – vagyis az adott csomóponttól a célhoz vezető út költségének becslőjét:

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

Mivel g(n) megadja a kiinduló csomóponttól az n csomópontig számított útköltséget, és h(n) az n csomóponttól a célcsomópontba vezető legolcsóbb költségű út költségének becslője, így az alábbi összefüggést kapjuk:

f(n) = a legolcsóbb, az n csomóponton keresztül vezető megoldás becsült költsége.

Így amennyiben a legolcsóbb megoldást keressük, ésszerű először a legkisebb g(n) + h(n) értékkel rendelkező csomópontot kifejteni. Ezen stratégia kellemes tulajdonsága, hogy ez a stratégia több mint ésszerű: amennyiben a h függvény eleget tesz bizonyos feltételeknek, az A* keresés teljes és optimális.

A peremben a rendezést f() = h()+g() alapján végezzük: a legkisebb értékű csúcsot vesszük ki. f() a teljes út költségét becsüli a kezdőállapotból a célba az adott állapoton keresztül.

Ha h() 0 (tehát f() g()), és gráfkeresést alkalmazunk, akkor a Dijkstra algoritmust kapjuk.

Kétszemélyes játékok és reprezentálásuk

A játékok reprezentálása

Teljesen megfigyelhető, véges, determinisztikus, kétszemélyes (a játékot két játékos játssza), zérusösszegű (ugyanannyira jó az egyik játékos számára megnyerni a játékot, mint amennyire rossz a másiknak elveszítenie) játékokkal foglalkozunk.

Ábrázolás

- B: a játékállások halmaza
- b0: a kezdőállás
- J: a játékosok halmaza
- V: a végállások halmaza
- v kalap: meghatározza melyik játékos nyert
- L: a lépések halmaza

Formális megközelítésben egy rendezett hatost kell definiálni, hogy egy játék szükséges jellemzőit összegyűjtsük. Az első eleme ennek a hatosnak a játékállások halmaza (B), azaz a játék menete során a környezetben előforduló különböző állapotoknak a reprezentációi.

Kitüntettek ezek közül a kezdő játékállás, amelyet most b0 jelöl. Össze kell gyűjtenünk a játékosok halmazát, amely garantáltan kételemű. Illetve meg kell határozzuk a játékállások azon kitüntetett részhalmazát, mely esetében a játék befejeződik, ezt hívjuk végállásoknak (J). A végállásokban még van egy fontos feladatunk, nyilatkozni kell tudni arról, hogy hogyan ért véget a játék, melyik játékos nyert, melyik vesztett, esetleg döntetlen alakult e ki. Erre fogjuk használni a v kalap függvényt, amely a végállásokat minősíti. A lépni következő játékos 1 esetén nyert, -1 esetén veszített, 0 esetén döntetlent ért el. Például egy sakkjátszmában matt estén a lépni következő játékos veszít (aki előzőleg lépett, aki a mattot adta, az nyert). Végezetül lépések definíciójára van még szükség, ahol a lépések játékállásokat játékállásokra képeznek le parciális függvények, hiszen nem feltétlenül alkalmazható minden lépés tetszőleges játékállásban.

Játékreprezentációból állapottérreprezentáció készítése

A játék összes állása mellett fel kell tüntetni az adott állásban lépni következő játékost. Ez azt jelenti, hogy az állapotterünk állapothalmaza nem más, mint az állások és játékosok halmazainak vett Descartes szorzata. A kezdőállapot a kezdő állás és a kezdő játékos alkotta pár. Míg végállapotok olyan állás játékos párok, melyek álláskomponense végállás. A lépésekből egyszerűen operátorokat származtathatunk úgy, hogy amennyiben egy állapotban az állás játékos párban az állás részre alkalmazható egy lépés, úgy a belőle származó operátor is és az alkalmazás eredménye pedig egy olyan állás játékos pár, ahol az állás komponens a lépés végrehajtásának eredménye, míg a játékos komponens, a következő játékos az ellenfél.

Ugyanúgy használható a közvetlen elérhetőség és elérhetőség fogalmai is.

Azt mondjuk, hogy az a állapotból a’ állapot közvetlenül elérhető, ha van olyan operátor, amelynek értelmezési tartományába a beleesik és maga, amelynek alkalmazási eredménye pont az a’. Az elérhetőség a közvetlen elérhetőség relációjának lezárásaként definiálható.

A nyerő stratégia

Játszma fogalma: a játszma egy olyan lépéssorozat, amely a játék kezdőállapotából valamely olyan állapotba vezető lépéssorozat, mely esetében végálláshoz érkezünk.

A stratégia az egy olyan leképezés, amely egy tekintett játékos számára minden olyan helyzetben, mely esetben lépni következik, előírja, hogy melyik lépést tegye meg. Azaz, olyan állás és játékos párokhoz rendel lépéseket, ahol a tekintett játékos épp a lépni következő játékos. Az egyik játékos garantáltan rendelkezik nyerő stratégiával, amennyiben döntetlen nem állható elő. Illetve döntetlen előfordulásának esetében az egyik játékosnak garantáltan van nem vesztő stratégiája. Ennek bizonyítása a következő, készíthetünk egy függvényt (w), amely tetszőleges állás és játékospárt fölcímkéz aszerint, hogy abban a helyzetben melyik játékos rendelkezik nyerő startégiával, ki az aki garantáltan nyerhet egy olyan játékban, ahol döntetlen nem állhat elő. A helyzet nagyon egyszerű, hogyha az állás játékos pár állás tagja végállás. Ebben az esetben v kalap

függvény meghatározza, hogy a lépni kívánt játékos nyert vagy vesztett. Ha b állásban j következik lépni és ez végállás, valamint v kalap b 1, tehát a lépni következő játékos a nyertes, akkor ennek az állás játékos párnak a címkéje értelemszerűen j játékos. Míg hogyha ebben az állásban v kalap b értéke -1, azaz éppen a nem lépni következő játékos a nyertes, akkor j'-vel címkézzük meg ezt az állás játékos párt, ahol j' az ellenfél. Azon az esetekben, amin nem végállásban vagyunk j játékos akkor érezheti magát nyerő helyzetben, ha tud olyan lépést választani, azaz közvetlenül elérhető számára olyan állapot a játékban, melyet az ő jelével címkéztünk, tehát amelyben ő van nyerő helyzetben. Tehát ha j a lépni következő játékos tud olyan lépést választani, melyben ő nyer, akkor nyerő helyzetben van. Minden egyéb esetben címkézzük az állást j vesszővel. Ezáltal egy rekurzív függvényt definiáltunk, amely meghatározza a játék kezdetén a0, j0 állapotban ki van nyerő helyzetben, ő az akinek nyerő stratégiája létezik.

Lépésajánló algoritmusok

MinMax módszer

A lépésajánló algoritmusok abban az állapotban, ahol a lépésajánlást kértük, rögzített k mélységig fogják bejárni a játéknak a fáját. Egészen pontosan azt a részfát, amely a tekintett játékállásból indul ki és legfeljebb k mélységű részfa. Nagyon fontos megérteni, hogy a lépésajánló algoritmusok hogy értékelik a közbenső csúcsokat. A levélelemek értékelése mindig egyszerűbb, a levélelemek értékelésére fogjuk felhasználni vagy a hasznosságfüggvényt vagy a tekintett játékos számára készített hasznosságfüggvényt, ahol a tekintett játékos mindig abban az állapotban ahol a lépésajánlást kértük lépni következő és őt támogatott játékosnak nevezzük. Hogyan lehet értékelni az egyes közbenső csúcsokat MinMax algoritmus esetén, illetve a lépésajánló algoritmus által feltárt legfeljebb k mélységű részfának a csúcsait. Levélelemek kétféleképpen állhatnak elő. Egyrészt levélelem az, amely végállást reprezentál, illetve levélelem az, ahol elértük a mélységi korlátot. Ebben az esetben a MinMax algoritmus a hasznosság függvény, mégpedig a támogatott játékos számára készült hasznosság függvény szerint fogja értékelni a levélelemeket. Innentől kezdve minden olyan közbenső csúcsban, ahol nem a támogatott játékos a lépni következő játékos, ott egy minimum keresést fogunk végezni, a közvetlenül elérhető állapotok tekintetében vizsgáljuk azok MinMax f függvény szerinti hasznosságát és ezek közül a minimálisat választjuk. Míg ha a játékosunk, aki lépni következik, pont a támogatott játékos, akkor ugyanezt maximum kereséssel fogjuk majd implementálni. Az ötlet lényege, hogy minden olyan ponton, amikor a támogatott játékos van döntési helyzetben, akkor ő maga számára a legjobbat akarja, ezért az elérhető állapotok közvetlenül elérhető játékállapotok közül a lehető legjobbat, a maximális jóságértékkel rendelkezőt fogja választani. Míg feltételezzük, hogy az ellenfele önmagának akarja a legjobbat. Ami az ellenfél számára a legjobb, az a támogatott játékos számára a lehető legrosszabb lépés, ezért van ebben az esetben minimum keresés az implementációban, illetve az f(b, j, t, k) rekurzív függvényben, mely szerint a MinMax algoritmus a közbenső csúcsok jóságértékét, hasznosságát meghatározza.

Az általunk vizsgált játékok tekintetében a mesterséges intelligencia feladata lépésajánlás. Ez azt jelenti, hogy a játék menete során tetszőleges állapotban a soron következő lépni következő játékos számára ajánlatot kell tudni tenni a legkedvezőbb lépés vonatkozásában. A hasznosságfüggvényre azért van szükségünk, mert a játék teljes játékfáját feltárni az esetek túlnyomó többségében teljesen reménytelen dolog. Nagyon gyakran előfordulnak végtelen ágak, illetve eleve nagyon terebélyes lehet egy-egy ilyen játékfa. A hasznosság függvény feladata, hogy abban az esetben, amikor már nincs lehetőségünk tovább feltárni a játékfát, nincs lehetőségünk mélyebbre menni, akkor legalább egy becslést adjon arra vonatkozóan, hogy adott játékállás, az ott éppen soron következő lépni következő játékos szemszögéből mennyire jó vagy rossz. Természetesen azt elvárjuk, hogy ez a becslés végállásban pontos legyen. Így formálisan azt mondhatjuk, hogy a hasznosságfüggvény a játék állapotainak leképezése egy olyan -m, m zárt intervallumra, ahol m valamilyen pozitív valós szám. Továbbá ha a hasznosságfüggvényt olyan állás-játékospár esetén vizsgáljuk, ahol az állás tag végállás, akkor a hasznosság függvény értékének m * v kalap b-nek kell lennie, ahol v kalap b a végállás jóságát a lépni következő játékos szempontjából minősítette -1, 0, illetve 1 értékekkel, aszerint hogy a lépni következő játékos vesztett, döntetlent ért el vagy nyert. A hasznosság függvény tehát a játék állapotait, állás és játékos párokat minősít, ahol a játékos az adott játékállásban mindig a lépni következő játékos. Ha már adva van egy ilyen hasznosság függvény, akkor készíthetünk adott játékos számára is külön hasznosság függvényt. Jelöljük ezt Hi-vel, hogyha ez az i-vel jelölt játékos hasznosságfüggvénye. Az i-vel jelölt játékos hasznosságfüggvénye megegyezik a hasznosság függvény értékével, amennyiben i az pont a lépni következő játékos, amennyiben nem, úgy természetesen zérusösszegű játékokbab ennek éppen -1 szeresével lesz egyenlő.

Tegyük fel, hogy mindkét játékos a teljes játékgráfot ismeri, tetszőlegesen komplex számításokat tud elvégezni, és nem hibázik (nagyon erős feltevések). Ezt szokás a tökéletes racionalitás hipotézisének nevezni.

Egy stratégia minden állapotra meghatározza, hogy melyik lépést kell választani. Belátható, hogy mindkét játékos számára a lehető legjobb stratégiát a minimax algoritmus alapján lehet megvalósítani tökéletes racionalitás esetén.

Ha a jatékgráfban van kör, akkor nem terminál (fakeresés), de a gyakorlatban ez nem probléma: csak fix mélységig futtatjuk (l. később), ill. a játékszabályok gyakran kizárják a köröket a végtelenségig folytatódó játszmák megelőzésére.

Világos, hogy a minimax érték az optimális hasznosság, amit az adott állapotból egy játékos elérhet, ha az ellenfél tökéletesen racionális.

Futtatás: a játékos tehát először kiszámítja a minimax értékeket a teljes játékfára, majd mindig a számára legjobb minimax értékű szomszédot lépi (ez határozza meg a stratégiát).

NegaMax módszer

A NegaMax módszer esetében is ugyanezek elvek mentén gondolkodunk, tehát a támogatott játékos önmaga számára legjobb, míg az ellenfele a támogatott játékos számára a legrosszabb lépést próbálja megtenni. A NegaMax algoritmusban azonban egy egyszerűbb rekurzív függvényt készítünk, ennek implementációja is könnyebb lesz. Itt a NegaMax algoritmus nem a támogatott játékos szempontjából értékeli az egyes lépéseket, hanem mindig a soron következő, az adott játékállapotban lépni következő játékos szerint így f kalap (b, j, k) a NegaMax algoritmus szerinti értéke egy közbenső csúcsnak, ahol a b állásban j játékos következik lépni és a részfánk maximális mélysége k. A levélelemek értékelése ugyanígy a hasznosságfüggvény segítségével történik. A hasznosság függvény eredendően éppen mindig a lépni következő játékos szempontjából értékeli a tekintett játékállást. Így ez esetben a NegaMax algoritmus rekurzív függvényének értéke és a hasznosság függvény értéke végállásban, illetve a mélységi korlátot elérve megegyezik. A közbenső csúcsok tekintetében maximum keresést végzünk. Mégpedig a közvetlen elérhető állapotok jóságértékének -1 szeresén. Így szintről szintre mindig negálva a közvetlenül elérhető állapotoknak a jóságát alakul ki ugyanaz a végeredmény, ami a MinMax esetében is kialakult volna. A csúcsok értékelése hasonló elvek szerint megy és a végeredmény mindig a gyökércsúcs vonatkozásában tökéletesen ugyanaz az érték. Igazából egyes szinteken a MinMax és NegaMax algoritmus által végzett értékelés eredménye csak előjelben tér el. A fa gyökerében és innentől kezdve lefelé minden második szinten a NegaMax és a MinMax algoritmus értékelései egybeesnek. Míg az ettől különböző szinteken egymásnak éppen -1 szeresei.

Mi legyen az ajánlott lépés, ha már egyszer a különböző lépésajánló algoritmusok elő tudták állítani a legfeljebb k mélységű tekintett állapotból kiinduló részfa minden egyes csúcsának értékelését. Egy lépést kell ajánlani, mégpedig egy olyan lépést, amelyet a gyökér csomópontból elvégezve, olyan közvetlenül elérhető állapotot kapunk, amely a támogatott játékos számára a lehető legjobb, így most a MinMax algoritmust nézve, egy olyan lépést kell választani, amely segítségével a közvetlenül elérhető állapothoz készített értékelés pont megegyezik a gyökér csomópont értékelésével.

Alfa-béta nyesés

Ötlet: ha tudjuk, hogy pl. MAX egy adott csúcs rekurzív kiértékelése közben már talált olyan stratégiát, amellyel ki tud kényszeríteni pl. 10 értékű hasznosságot az adott csúcsban, akkor a csúcs további kiértékelése közben nem kell vizsgálni olyan állapotokat, amelyekben MIN ki tud kényszeríteni < 10 hasznosságot, hiszen tudjuk, hogy MAX sosem fogja ide engedni a játékot (hiszen már van jobb stratégiája).

Algoritmus: az n paraméter mellé az alfa és béta új paramétereket is átadjuk minÉrték-nek és maxÉrték-nek. Az alfa és béta paraméterek jelentése függvényhíváskor:

Alfa: MAX-nak már felfedeztünk egy olyan stratégiát, amely alfa hasznosságot biztosít egy olyan állapotból indulva, ami a keresőfában az n állapotból a gyökér felé vezető úton van.

Béta: MIN-nek már felfedeztünk egy olyan stratégiát, amely béta hasznosságot biztosít egy olyan állapotból indulva, ami a keresőfában az n állapotból a gyökér felé vezető úton van.

Majdnem azonos a minimax-szal, csak propagáljuk alfát és bétát, és vágunk.

A szomszédok bejárási sorrendje nagyon fontos. Ha mindig a legjobb lépést vesszük (optimális eset), akkor O(bm) helyett O(bm=2) lesz az időigény: tehát kétszer olyan mély fát járunk be ugyanazon idő alatt.

Véletlen bejárással O(b3m=4) lesz a komplexitás.

A gyakorlatban használhatunk rendezési heurisztikákat, amik sokszor közel kerülnek az optimális esethez.

Gráf keresés: játékgráfban is sok lehet az ismétlődő állapot. Eddig a fakeresés analógiájára nem tároltuk a már látott állapotokat. Ha tároljuk (mint a zárt halmazban) akkor az alfa-béta algoritmus is jelentősen felgyorsul, ill. nem ragad végtelen ciklusba. Itt a hagyományos elnevezése a zárt halmaznak transzpozíciós tábla.