2013. 11. 11.

ÓBUDAI EGYETEM NIK

# SAKKPROGRAMOK

*A gépek győzelme az ember felett* | Rusvai Tamás

# Sakkprogramok

*A gépek győzelme az ember felett*

## Bevezetés

A sakk döntéselméleti szempontból egy teljesen megfigyelhető, stratégiai, sorozatszerű, statikus, diszkrét, többágenses környezet. Ennek tükrében mindig is a mesterséges intelligencia középpontjában áll.

A 18.században született a sakkgépek ötlete. Kempelen Farkas 1769-ben alkotta meg a saját „sakkozógépét", a Törököt. Ebben igaz, hogy ember volt a fő cselekvőmotor, de újszerűségéből nem von le semmit, mert különféle tükrökkel oldották meg, hogy ő ne legyen látható, de ő be tudja látni az egész pályát.

1912-ben Leonardo Torres y Quevedo megalkotta a történelem első sakk gépét El Ajedrecista-t. A tábla alatti elektromágnesek segítségével automatikusan lejátszotta a végjátékot három figurával, mozgatva egy királyt és egy bástyát, az ember által vezérelt király ellen.
Az 1950-es évektől kezdve kezdtek gyártani sakkprogramokat. Az első sakkprogram 1947 készült el Türing jóvoltából. A 70-es években a kutatók már komoly veszélynek érezték, hogy egy sakkgép letud győzni egy embert is. 1978-ban David Levy miután leverte a Chess 4.7-ás nevezetű programot, kijelentette, hogy a következő tíz évben sem lesz olyan program, mely leverné őt. 1988-ban, pontosan 10 év múlva a Deep Thought a Deep Blue elődje legyőzte a David Levy-t.

1996-ban Garry Kasparov, mindenidők legnagyobb sakkozójának tartott embere legyőzi a Deep Blue nevezetű sakkgépet 4-2-re. Egy évvel később 1997-ben 3½-2½-re kapott ki.2005-ben Ruslan Ponomariov legyőzi a Fritz nevezetű sakkprogramot, de csapatszinten 8-4-re kikaptak. A Hydra súlyos vereséget mér a világranglista hetedik helyén álló Michael Adams-re 5½-½. 2006-ban Vladimir Kramnik az akkori világbajnok, kikapott a Deep Fritz-től 4 döntetlen, 2 vereséggel.

Jelen írásomban kifejtem a sakkprogramok általános felépítését. Ezután kitérek két speciális sakkgépre a Deep Blue-ra, és a Hydrára, mindkettőnek saját egyedi hardvere volt, így nagyobb számítási kapacitást értek el, mint a mai PC-s sakkprogramok. Végül egy mai sakkprogramot mutatok be, a Fritz programot, ennek is a többmagos processzoros támogatással ellátott Deep Fritz verzióját.

## 1. Sakkgépek általános felépítése és működésük

A sakkgépek általános felépítése – beleértve a bemutatott ábrákat – a [12] forrásanyagra támaszkodik.

A korszerű sakkgépek része három fő egységre bontható. A megnyitási adatbázisra, a végjáték adatbázisra, és egy gráfkeresési algoritmusra , amely felelős a középjáték lebonyolításához.

### *Megnyitás*

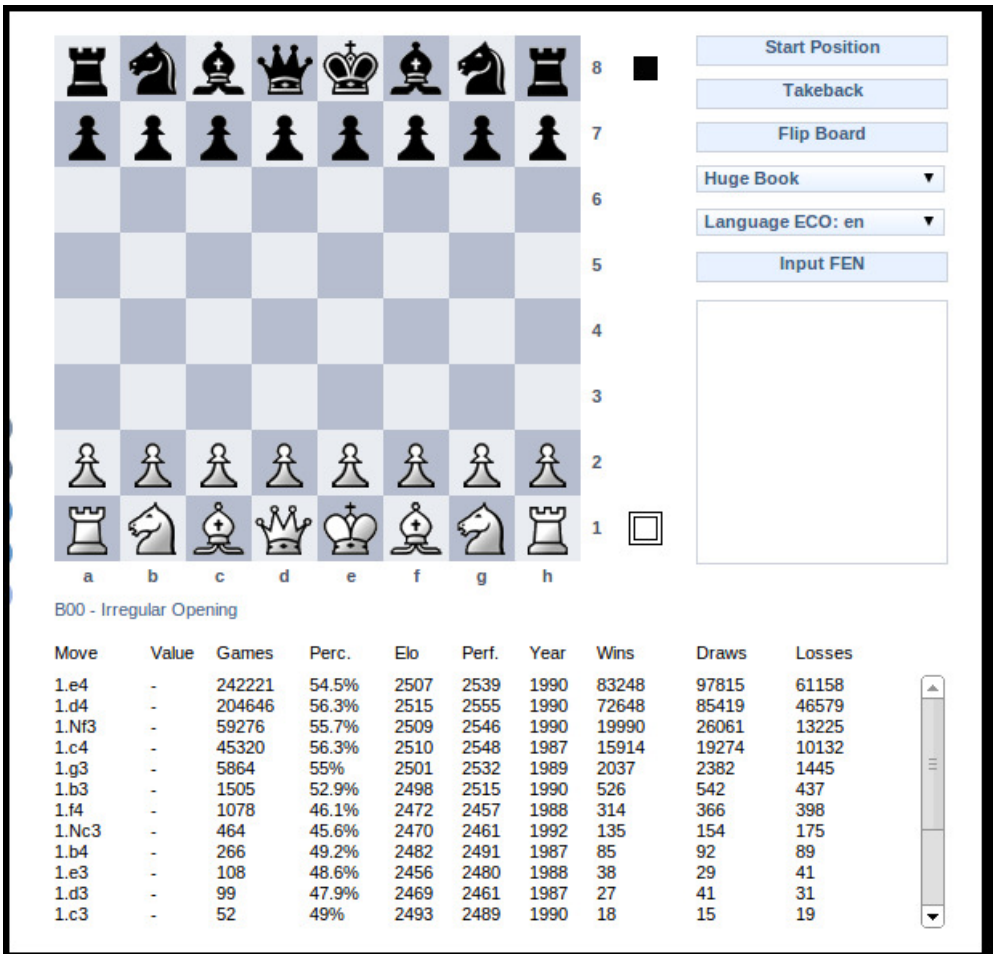
A megnyitás adatbázis egy olyan gyűjtemény, amely a világklasszis játékosok lejátszott játékait tartalmazza. Általában 2500 Élő[1]-pont feletti játékosok játszmáit tartalmazza. Egy megnyitás adatbázis annál jobb, minél több és erősebb játékos játékait tartalmazza. Ezen játékok lejegyzése időigényes feladat, ezért vannak külön erre a célre kifejlesztett programok, sőt az ilyen adatbázisokat külön meg is lehet vásárolni. A Shredder, vagy a Chessbase cégeknél lehet kapni ilyen adatbázisokat. Az áruk a mérettől függően változik, míg a 4,5 milliós megnyitási adatbázis 100$[2]-ba kerül, addig az 5,4 milliós adatbázis már 150$[3]-ba. Jelenlegi

1 Élő Árpád, magyar származású fizikus által kifejlesztett pontrendszer. Jelenleg ezt használja a FIDE.

http://en.chessbase.com/home/TabId/211/PostId/4004326

2 http://www.chessbase-shop.com/en/products/opening_encyclopedia_2013

tudásom alapján nem létezik ingyenes megnyitási adatbázis. Az ilyen adatbázisok azért fontosak a sakkprogram szempontjából, mivel egyszerűbb egy adatbázisban keresni,mint a gépi algoritmusra bízni, hogy megtalálja a lehető legoptimálisabb megoldást, úgy hogy egy rossz nyitás egyből vesztes játszmát eredményezhet, még ha körülbelül 1Gb-nyi tárhely igényű egy ilyen adatbázis.



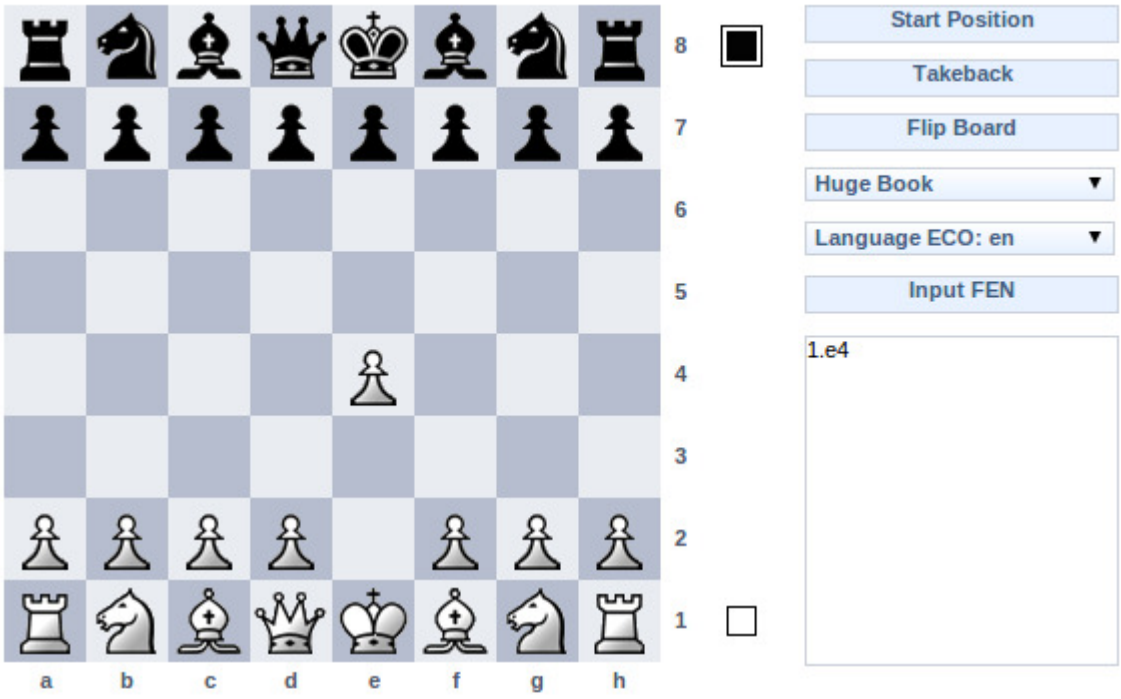

| Move | Value | Games | Perc. | Elo | Perf. | Year | Wins | Draws | Losses |
|---|---|---|---|---|---|---|---|---|---|
| 1.e4 | - | 242221 | 54.5% | 2507 | 2539 | 1990 | 83248 | 97815 | 61158 |
| 1.d4 | - | 204646 | 56.3% | 2515 | 2555 | 1990 | 72648 | 85419 | 46579 |
| 1.Nf3 | - | 59276 | 55.7% | 2509 | 2546 | 1990 | 19990 | 26061 | 13225 |
| 1.c4 | - | 45320 | 56.3% | 2510 | 2548 | 1987 | 15914 | 19274 | 10132 |
| 1.g3 | - | 5864 | 55% | 2501 | 2532 | 1989 | 2037 | 2382 | 1445 |
| 1.b3 | - | 1505 | 52.9% | 2498 | 2515 | 1990 | 526 | 542 | 437 |
| 1.f4 | - | 1078 | 46.1% | 2472 | 2457 | 1988 | 314 | 366 | 398 |
| 1.Nc3 | - | 464 | 45.6% | 2470 | 2461 | 1992 | 135 | 154 | 175 |
| 1.b4 | - | 266 | 49.2% | 2482 | 2491 | 1987 | 85 | 92 | 89 |
| 1.e3 | - | 108 | 48.6% | 2456 | 2480 | 1988 | 38 | 29 | 41 |
| 1.d3 | - | 99 | 47.9% | 2469 | 2461 | 1987 | 27 | 41 | 31 |
| 1.c3 | - | 52 | 49% | 2493 | 2489 | 1990 | 18 | 15 | 19 |

1. ábra A Shredder online megnyitási adatbázisa

Ahogy  látszik az 1. ábrán, lehetőségünk van kipróbálni egyes megnyitásokat. Az adatbázis tárolja a lépést, hogy hány játékban lépték ezt, és hogy milyen arányban vezetett győzelemre a lépés. Az arányt úgy számoljuk, hogy minden nyert játszma 1 pontot ér, döntetlenért fél pont, vereségért 0 pont (ezt nem is kell belevenni a képletbe). A képlet pedig úgy alakul, hogy a (gyözelmek száma+döntetlenek száma) / összes játék. Például az e4 egy jó lépésnek számít, mivel 54,5% esélyben a fehér nyert, amikor ezt lépte, és ehhez a lépéshez tartozik a legtöbb lejátszott játszma is. Lépjük is meg; ekkor a sötét lehetséges lépéseit mutatja a táblázat.

---

3 http://www.chessbase-shop.com/en/products/mega_database_2013




B00 - Irregular Opening

| Move | Value | Games | Perc. | Elo | Perf. | Year | Wins | Draws | Losses |
|---|---|---|---|---|---|---|---|---|---|
| 1...c5 | - | 115998 | 46.4% | 2511 | 2481 | 1991 | 31218 | 45441 | 39339 |
| 1...e5 | - | 49163 | 44.7% | 2511 | 2478 | 1989 | 11069 | 21837 | 16257 |
| 1...e6 | - | 31007 | 44.3% | 2505 | 2468 | 1991 | 7470 | 12548 | 10989 |
| 1...c6 | - | 16320 | 44.7% | 2508 | 2473 | 1991 | 3708 | 7181 | 5431 |
| 1...d6 | - | 9509 | 44.1% | 2503 | 2463 | 1991 | 2425 | 3553 | 3531 |
| 1...g6 | - | 8499 | 44.9% | 2499 | 2460 | 1990 | 2335 | 2977 | 3187 |
| 1...Nf6 | - | 5823 | 43.4% | 2494 | 2450 | 1988 | 1444 | 2169 | 2210 |
| 1...d5 | - | 4280 | 42.6% | 2488 | 2446 | 1994 | 1020 | 1611 | 1649 |
| 1...Nc6 | - | 1268 | 44.6% | 2491 | 2457 | 1990 | 369 | 395 | 504 |
| 1...b6 | - | 273 | 45.9% | 2498 | 2470 | 1993 | 82 | 87 | 104 |
| 1...a6 | - | 60 | 29.1% | 2466 | 2333 | 1992 | 12 | 11 | 37 |
| 1...g5 | - | 10 | 45% | 2434 | 2455 | 1987 | 4 | 1 | 5 |
| 1...h6 | - | 7 | 21.4% | 2440 | 2259 | 1993 | 0 | 3 | 4 |
| 1...Na6 | - | 3 | 83.3% | 2438 | 2714 | 1998 | 2 | 1 | 0 |
| 1...f6 | - | 1 | 0% | 2460 | 2150 | 1995 | 0 | 0 | 1 |

2. ábra Shredder online megnyitási adatbázisa az e4 lépés után

Értelemszerűen a programnak a legnagyobb esélyű lépést kell választania, mivel ezzel segíti elő legjobban a győzelmét; ez esetben ez a Na6-os lépés lenne, aminek a győzelmi esélye 83.3%. Mivel ehhez a lépéshez, csak 3 darab játszma tartozik, ezért a program nem ezt választja, hanem a c5-s lépést, mivel ehhez tartozik a második legnagyobb esély, és bőven elég játékot játszottak vele.

Mivel egy idő után a lépésfánk olyan széles lesz, hogy nem találunk az adatbázisban elég releváns adatot, így nem tudjuk eldönteni, hogy valóban jó-e a lépés, ezért be kell vezetnünk egy minimális értéket a játszott meccsek számára, ami alatt nem vizsgáljuk az adott lépést. Ez az érték tetszőleges lehet. Lehet egy konstans érték, de lehet akár egy bonyolult függvény értéke is.

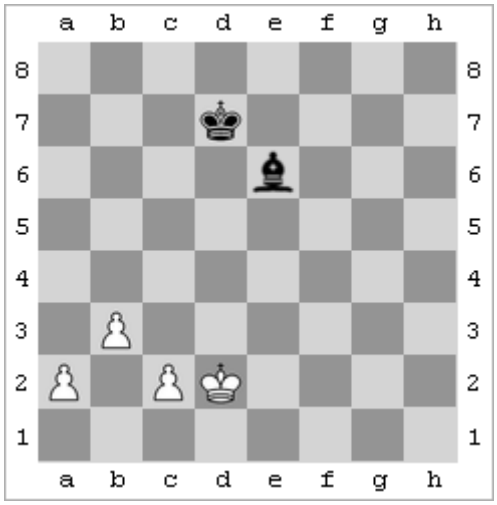
Az adatbázis faszerkezetként írható le, ahol minden egyes lépésből, egy vagy több lépés is ágazik. Ennélfogva egy idő múlva elérkezünk egy olyan ponthoz, ahol már az összes lehetséges lépést a megadott értéknél kevesebbszer játszották. Ekkor beszélünk középjátékról, és innen fogva a mesterséges intelligencia komponens veszi át az irányítást.

Mielőtt még belevágnánk a középjáték leírásába, vegyük szemügyre a végjáték adatbázis felépítését.

### *Végjáték*

Beszélünk 3-, 4-, 5-, 6- és 7-bábus végjáték adatbázisról. A számok azt szimbolizálják, hogy hány darab figura van még a táblán bármilyen felállásban. Csak ezekben az esetekben tudjuk használni a végjáték adatbázist, amikor is a kereső azonnal megadja az összes lehetséges lejátszást. Ez nagyon nagyméretű adatbázist igényel, de még így is gyorsabb, mint ha a mesterséges intelligencia komponensnek kéne gráfkereséssel megoldani a feladatot, ami nem biztosítja nyertes pozícióban a matt, vesztes pozícióban a döntetlen esélyét.

Een adatbázis megléte elengedhetetlen a professzionális szintű sakkprogram elkészítéséhez, mivel a végjátékban általában már nincs elég idő, hogy minden lépést a mesterséges intelligencia komponensre bízzunk, főleg úgy, hogy létezik olyan végjáték is, amely 100 lépésnél hosszabb. Ebben az esetben, ha a mesterséges intelligencia komponens működik, lépésenként 1 percet vesz igénybe (ami elég jó idő), akkor is 100 perc kellene neki, hogy kiszámolja a győzelmet, annak tükrében, hogy nem is biztos a megoldás, mivel a legjobb programok is csak 14 körüli lépéssel tudnak előre tervezni.



5. ábra Egy lehetséges 6 bábus végjáték

Vegyünk egy példát. A 3. ábrán látható állásban, már csak 6 bábu maradt; tegyük fel, hogy a világos lép. Döntetlen-gyanús a játék, de tegyük fel, hogy világos be tud vinni egyet a 3 megmaradt gyalogjai közül, és megnyeri a játszmát. Sötét célja már csak döntetlen lehet, mivel nincs elegendő[4] figurája a matt adáshoz. A Shredder végjáték adatbázisa szintén kipróbálható[5]. Látható, hogy a világos összes lehetséges lépéséből a sötét ki tudja harcolni magának a döntetlent. Lépjünk mondjuk az a3-ra; ekkor az alábbi táblázatból látjuk, hogy sötét csak akkor tudja elveszteni ezt a játékot, ha az alábbi lépések közül a futó beáldozását választja.

Ennek fényében pillanatok alatt megadható az optimális lépés.

| Draw | |
|---|---|
| **Move** | **Value** |
| a2-a3 | Draw |
| a2-a4 | Draw |
| c2-c3 | Draw |
| c2-c4 | Draw |
| Kd2-e2 | Draw |
| Kd2-d1 | Draw |
| Kd2-d3 | Draw |
| Kd2-c1 | Draw |
| Kd2-e1 | Draw |
| Kd2-c3 | Draw |
| Kd2-e3 | Draw |
| b3-b4 | Draw |

4. ábra Fehér lehetséges lépései

| Draw | |
|---|---|
| **Move** | **Value** |
| Be6-d5 | Draw |
| Be6-f5 | Draw |
| Be6-g4 | Draw |
| Be6-h3 | Draw |
| Be6-f7 | Draw |
| Be6-g8 | Draw |
| Kd7-c7 | Draw |
| Kd7-e7 | Draw |
| Kd7-d6 | Draw |
| Kd7-d8 | Draw |
| Kd7-c6 | Draw |
| Kd7-c8 | Draw |
| Kd7-e8 | Draw |
| Be6-c4 | Lose in 17 |
| Be6xb3 | Lose in 17 |

3. ábra Sötét lehetséges lépései

4 Matthoz legalább az alábbi párosítás egyike kell:király + vezér, király + bástya, király + 2 futó, király + futó + huszár
http://en.wikipedia.org/wiki/Checkmate

5 http://www.shredderchess.com/online-chess/online-databases/endgame-database.html

Jelenleg a 7 bábus végjáték adatbázisa a legbővebb, ami létezik és kapható[6]. A fentebb említett előnyök miatt, ma a nagymesterek már csak akkor állnak ki programok ellen, ha azok nem használhatnak adatbázist.

A következőkben röviden bemutatom a Deep Blue, a Hydra és a Deep Fritz sakkprogramokat.

## 2. Deep Blue

### *Történet*

A Deep Blue egy sakkszámítógép volt, amit az IBM fejlesztett ki 20 millió dollárért. A mesterséges intelligencia kutatások egyik legnagyobb eredménye volt a 90-es években, hogy sikerült olyan gépet alkotni, mely legyőzi az akkori világbajnokot, Garry Kasparovot. 1996-ben Kasparov 4-2-re győzött a Deep Blue ellen. Rá egy évvel később New Yorkban újra összecsaptak, és a gép 3½-2½ legyőte a világbajnokot.

A projektet 1985-ben egy végzős hallgató, Feng-hsiung Hsu kezdte el a Deep Though nyomdokain a Carnegie Mellon Egyetemen, disszertáció gyanánt. Az akkori projektet úgy nevezte: ChipTest.

Egy csoporttársa, Murray Campbell dolgozott vele a projekten. Az egyetem után mindkettőjüket felvették az IBM-hez 1989-ben. Ott folytatták a munkájukat, más kollégájukkal egyetemben (Joe Hoane,J erry Brody és C. J. Tan). Miután a Deep Thought könnyedén kikapott Garry Kasparovtól, az IBM-nél szavazást tartottak, hogy mi legyen a következő verzió neve és a győztes a „Deep Blue” lett.

1995-ben a Deep Blue prototípusa második helyezést ért el a 8. Gépisakk világbajnokságon. Ekkor még elég gyenge teljesítményt tudott produkálni. A Wchess nevezetű program ellen döntetlent játszott, úgy hogy az egy PC-n futott. Az ötödik körben kikapott a Fritz 3-as programtól 39 lépésben, úgy hogy az egy Intel Pentium 90MHz-es gépen futott, és a Deep Blue fehér színnel játszott.

Rá egy évre sokat fejlődött a Deep Blue. 1996. február 10.-én a Deep Blue lett az első gép, amely sakkmeccset nyert egy világbajnok ellen normál időlimit szerint, habár a játszmát elvesztette 4-2-re.

Sok fejlesztés után 1997 májusában újra összecsapott Kasparovval, de ekkor már győzedelmeskedett 3½-2½. A meccs május 11.-én ért véget. Így a Deep Blue lett az első olyan sakkgép, amely győzelmet aratott egy jelenlegi világbajnok felett.

### *Hardver felépítés*

A Deep Blue sikerét a hatalmas mennyiségű számítási erő adta. Egy 32 processzoros IBM RS/6000 szerverre épült a rendszer, amit 480 speciális, direkt a sakk szabályaihoz tervezett segédprocesszor támogatott.

Másodpercenként 200 millió lépésvariációt tudott kiszámolni és értékelni. 6-12 lépésre előre tudta a meccs összes lehetséges állását elemezni. Négyezer féle megnyitást ismert, és több mint 700 ezer sakkjátszma lépéseit tartotta a memóriájában.

### *Szoftver felépítés*

A Deep Blue alapvetően azt a keresési algoritmust használta, amit Claude Shannon publikált 1950-ben a „Programming a Computer for Playing Chess”[7] című esszéjében.

---

6 http://chessok.com/?page_id=27966

7 http://vision.unipv.it/IA1/ProgrammingaComputerforPlayingChess.pdf

A Shannon módszer lényege az, hogy a gép a minimax stratégia alapján választja ki a következő lépést, amit egy függvény értékel ki több paraméter kiszámítása után. A függvény kiszámolja mind a fehér, mind a fekete értékét, majd kivonja őket egymásból.

A paraméterek:

- Figurák :
  - gyalog : 1 pont
  - huszár : 3 pont
  - futó : 3 pont
  - bástya : 5 pont
  - vezér : 9 pont
- Pozíció faktorok
  - dupla gyalog[8] : -½ pont
  - visszamaradt gyalog[9] : -½ pont
  - izolált gyalog[10] : -½ pont
- Mozgás : Minden lehetséges lépésért 0.1 pont
- Sakkmatt : 200 pont

## 3. Hydra

### *Történet*

A Hydrát Dr. Christian „Chrilly" Donninger, Dr. Ulf Lorenz, Christopher Lutz[11] , és Muhammad Nasir Ali fejlesztették; 2006-tól már csak Donninger és Lutz volt a csapat része.

A rendszer elődje a Brutus volt, aminek hasonló felépítése volt, mint a Deep Blue-nak. Sok specifikus chipet használt, hogy ezzel is növelje a teljesítményt. Ez esetben a sakkprocesszorokat FPGA (Field-programmable gate array)-[12]kal valósították meg.

A Hydra 2006 júniusában játszotta az utolsó meccsét. A szponzorok, a PAL Group és a Sheikh Tahnoon Bin Zayed Al Nahyan of Abu Dhabi ez után kiléptek. A Hydra hardveres és szoftveres leírása – beleértve a bemutatott ábrákat is – a [13] forrásanyagra támaszkodik.

---

8 http://en.wikipedia.org/wiki/Doubled_pawn

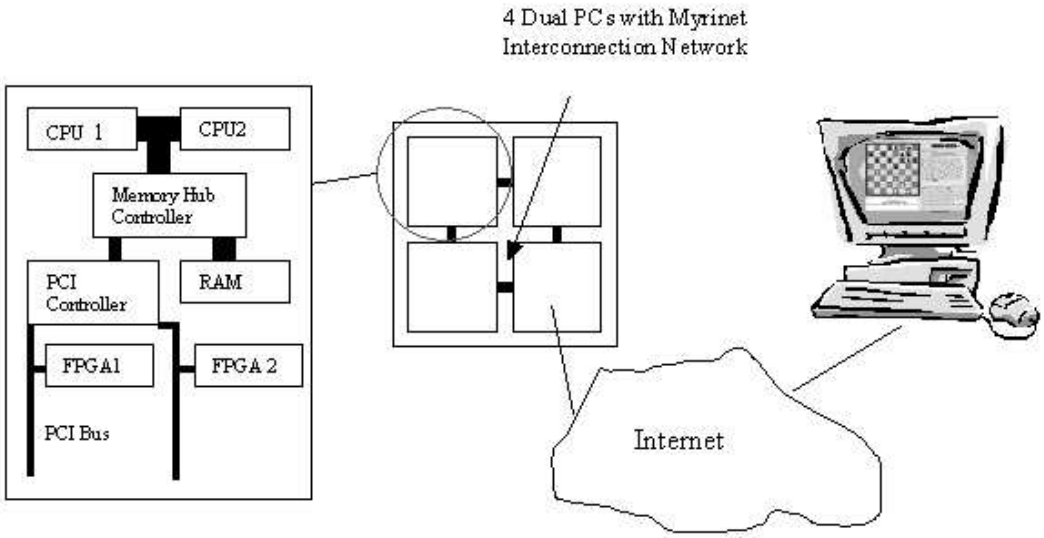
9 http://en.wikipedia.org/wiki/Backward_pawn

10 http://en.wikipedia.org/wiki/Isolated_pawn

11 Sakk Nagymester : http://ratings.fide.com/card.phtml?event=4600193

[12] Programozható logikai áramkör : http://hu.wikipedia.org/wiki/Field-programmable_gate_array

### *Hardver felépítés*




6. ábra A Hydra elvi felépítése

A Hydra a ChessBase/Fritz felhasználói felületét használja. Ez a felület egy Windows XP-t futattó PC-n található, és SSH kapcsolattal csatlakozik a Linux clusterhez, ami tartalmaz 4 Dual PC szerver egységet, amelyek képesek kezelni egyszerre két PCI buszt is. A csomópontok Myrinet hálózatba vannak összekötve. Mindkét buszhoz csatlakozik még egy FPGA kártya is. Minden MPI[13]-folyamatot az egyik processzor, és az egyik FPGA is feldolgozza párhuzamosan.

A használt FPGA egy XilinX alapú VitexV100E kártya volt. A kártyán található 96 BlockRAM-ból 67-et, a 12288 szeletből 9879-et, a 12544 TBUFS-ből 5308-at, a 24576 Flip-Flops-ból 534-et, és a 24576 LUT-ból mindössze 18403-at használt. Az FPGA-k 33MHz-en futottak, a leghosszabb út 51 logikai szintet tartalmazott. A keresési csomópontok maximálisan 9 kört ttartalmaznak. Emellett még egy összetett kombinatorikai logika is bele volt építve, amit egy 54 állapotú véges állapotú gép vezérelt.

### *Szoftver felépítés*

A kereséshez, az Alpha-Béta nyesés Negascout[14] variációját alkalmazza. Ez egy mélységi keresés, amely felhasználja az bal ágak információt, hogy csökkentse a jobb ágakban a keresést.

A lépésgenerátort[15], a kiértékelő függvényt[16], és a keresési algoritmust[17] is hardverrel oldja meg. Ennek több előnye is van. Először is a futási idő gyorsul, mivel az eljárásokat nagyon kevés ciklusban is fel lehet

---

13 http://en.wikipedia.org/wiki/Message_Passing_Interface

14 http://en.wikipedia.org/wiki/Negascout

15 Kiszámolja az összes lehetséges lépést

16 Értéket rendel a lehetséges lépések mellé. Ez az érték egy szakértő által megadott heurisztikus érték.

dolgozni. Másodszor, egy hagyományos PC-n kompromisszumot kell kötni a keresési gyorsaság és az algoritmus implementációja között. Tehát minél magasabb nyelven van megírva a kód, annál kevésbé van optimalizálva. Emellett a legtöbb kiértékelő függvény párhuzamosan futtatható. Így még több időt spórolunk meg, igaz, a helyigény rovására.

A kiértékelő függvényt már részleteztük a Deep Blue esetében. A Hydrában ezt egy összeadóval oldják meg. A részegységek párhuzamosan számolódnak ki.

A lépés generátort szoftveresen egy többszörösen beágyazott ciklusként szokták leírni. A legkülső ciklus végigmegy az összes figura típuson, és egy eggyel beljebb levő ciklus az összes olyan típusú figurán végigmegy, még beljebb az összes irányt veszi, még beljebb az összes mezőt abban az irányban. Intuitíve belátható, hogy ez egy lassú folyamat, és csak részlegesen lehet párhuzamosítani. Ezzel szemben a Hydra egy hardveres megoldást használ.

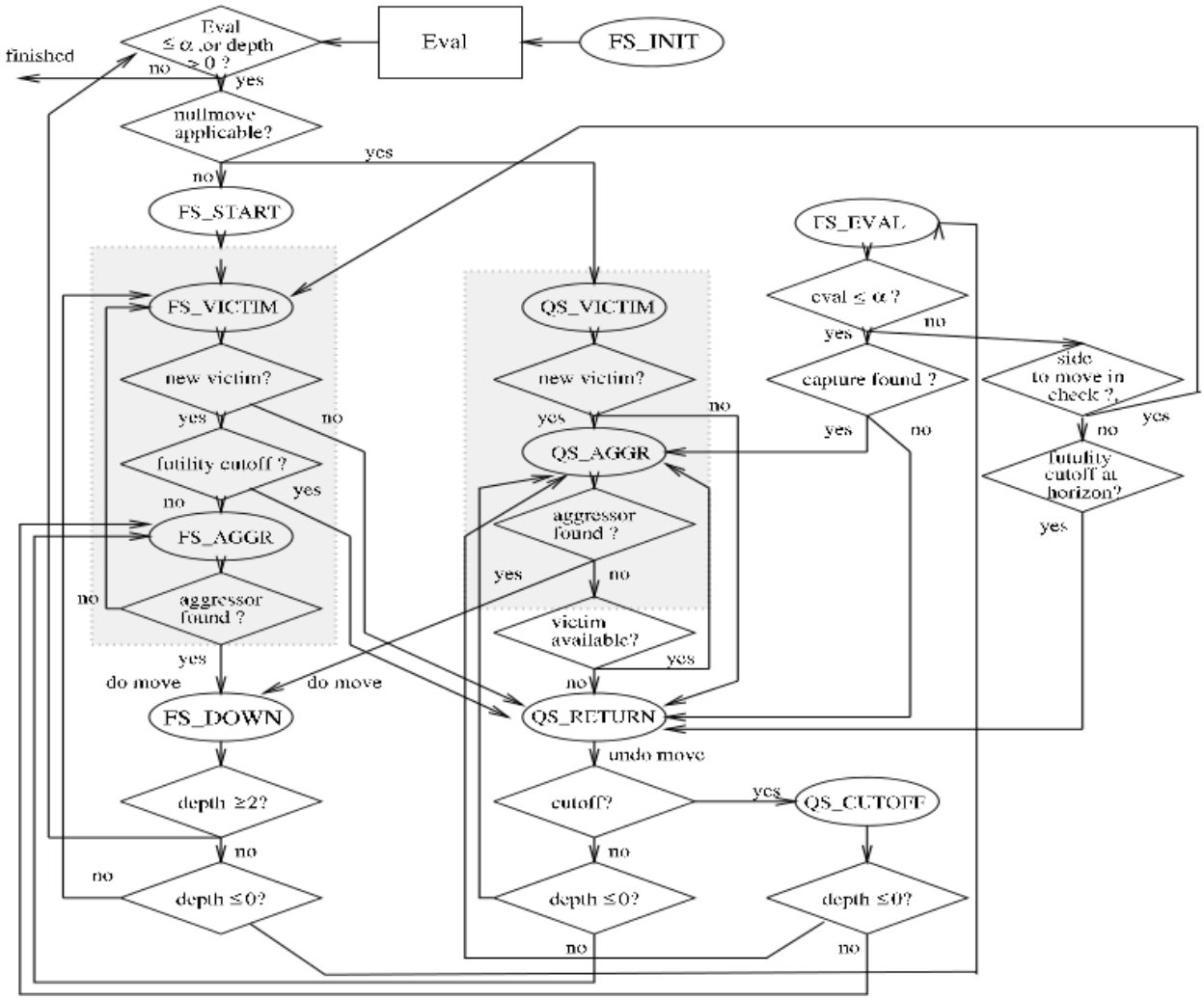
A lépésgenerátor elméletileg egy 8x8-as sakktábla. Két modulja a GenAggressor és a GenVictim, mindkettő megvalósítja ezt a 8x8-as táblát. Mindkettőben meg vannak határozva, mely szomszédos jeleket kell továbbítania. A mezők jeleket küldenek, ha található rajta figura, és továbbítják a távolabbi figurákból beérkező jeleket. Mellesleg minden mező képes úgynevezett 'victim found' (áldozati jel) jelet kibocsátani, ami azt jelenti, hogy ez a mező áldozata egy szabályos lépésnek.Tehát az összes figuránk egy jelet bocsát ki, az aktuális pozíciójáról. Ezeket a jeleket egy komparátor fába gyűjtjük és kiválasztjuk a legjobb, még nem vizsgált áldozatot. A GenAggressor modul a komparátor kimeneteit kapja bemenetként, és küld egy jelet, amely tartalmazza a lehetséges összes figurát. Tehát ha van egy bástyánk, az jelet küld az aktuális pozíciójáról. Ezt a jelet a GenAggressor modul megkapja bemenetként, ezután megvizsgálja az összes lehetséges ellenséges bábut,hogy képes-e leütni, és ha képes, akkor elmenti egy listába. Ezeket a műveleteket párhuzamosan végezzük. A komparátor fa ezeket sorba rendezi aszerint, hogy a támadó figura értéke mennyi, és hogy ez gyilkos lépés[18] volt-e vagy sem.

---

17 Rendszerezi az előrejelzést. Levágja a felesleges ágakat.

18 http://en.wikipedia.org/wiki/Killer_heuristic

## Keresési algoritmus




A keresés az alábbiak szerint zajlik. Beadjuk az FS_INIT-nek a keresést. Ha nem csak gyökér van, és a semmittevés nem engedélyezett, akkor elkezdjük a teljes keresést. Ha lehetséges a mélység növelése, akkor beállítjuk az FS_VICTIM értékenek az aktuális GenVictimet. Ha találuk egy lehetséges lépést, ahova léphetünk, és a nyesés nem lehetséges, akkor megvizsgáljuk az FS_AGGR-t, amiben a GenAggressort tároljuk, és ha tálalunk egy olyan lépést ahonnan léphetünk az előbbi pozícióra, meglépjük a következő lépést és elérkezünk az FS_DOWN-hoz. Ez az egység felel az alpha-béta nyesésért, aminek a keresési ablaka $[\sigma , \sigma + 1]$. Ha a keresés maradék mélysége nagyobb mint 0, akkor megnézzük, mit tárolunk az FS_START-ban. Máskülönben elkezdjük azokat a lépéseket vizsgálni, amelyekben nem történik ütés. Hogy ha ezen állapotok kiértékelése nem nagyobb mint alpha, folytatjuk egy ütő mozgással, ha lehetséges. Hogyha van olyan figura, amit leüthetünk, akkor elérkezünk a QS_AGGR állapothoz, és ha van olyan figuránk, ami ütheti, akkor lépünk egyet és így tovább. A lépéseket töröljük, és kilépünk a rekurzióból, ha elértük a QS_RETURN állapotot. Egy rekurzív algoritmusnak, mint az alpha-béta nyesés, szüksége van egy veremre a működéshez. A vermet a Hydrában 6 darab 2 portos RAM oldja meg (a RAM-ok 16 bitesek).

### *Eredmények*

A Hydra elődje a Brutus harmadik helyet ért el a Paderborni Számítógépes Sakk Bajnokság-on 2003-ban, és megnyerte a Lippstadt-i FIDE Nagymester Tornát; utóbbinál a kalkulát ELO[19]-ja 2768 volt, ami a világranglista első 10 helyezettjébe sorolná. (A mostani, 2013.10.11-es első helyezettnek 2870-es ELO-ja volt.)

Belső tesztek alapján kimutták, hogy az akkori sakkprogram világbajnokoknál a Fritz8 (2667) és a Shredder8-nál 110 ELO ponttal többet ért el egy Pentium 2.4Ghz-es PC-n.

A Hydra kétszer lett világbajnok a Paderborni Bajnokságon 2004-ben és 2005-ben.

2004-ben legyőzte az egykori világbajnokot, Ruslan Ponomariov-ot egy kétjátszmás partiban, mindkét meccset megnyerve.

2005-ben győzött Michael Adams ellen 6 játszmás partiban, 5 győzelemmel és 1 döntetlennel.

## 4. Deep Fritz

### *Történet*

A Deep Fritz a Fritz sakkprogram többprocesszoros változata. Frans Morsch és Mathias Feist fejlesztette a ChessBase megbízásából. Az első verzió 1992-ben lett kész. A Fritz-ekből a 13-as verzió kapható jelenleg. Frans Morsh Ed Schröderrel együtt a 80-as évek elejétől már dolgoztak egy sakkprogramon. A 90-es években egy német cég – ChessBase – felkérte Morschot, hogy írja meg a Fritz programot. (Mathias Feist programozóként dolgozott a ChessBasenek.) A 90-es évek elejétől napjainkig az egyik legsikeresebb sakkprogram a Fritz.

### *Hardver felépítés*

A Deep Fritz esetében nem igazán lehet külön hardveres felépítésről beszélni, mivel a hardveres részét egy közönséges PC adja. A program maga Windows operációs rendszeren fut. Mielőtt léteztek volna a többmagos processzorok, úgy oldották meg a Deep Fritz többmagos felépítését, hogy több gépet összekötöttek, és párhuzamosították a feladatokat.

### *Szoftver felépítés*

A következő leírás – beleértve a bemutatott ábrákat és táblázatokat is – az [5] forrásanyagra támaszkodik.

A három alap implementációs probléma, amit a sakkprogramok esetében meg kell valósani a tábla reprezentáció, a kiértékelés és a kereső algoritmusok. A következőkben ezekről lesz szó.

#### Tábla reprezentáció

A Fritz a Sargon nevezetű eljárást használja, amiszerint a táblát egy tömbként kezeli, amely 64 byte-os. Ezt a tömböt kiegészíti még 2 réteggel, így gyorsítva a lépésgenerálást. A 2 réteg mezőit illegálisnak jelöli, így például a futó lépését tudja úgy generálni, hogy  az illegális mezőig vizsgálja csak a lépéseket, így nem kell

---

[19] Az Élő-pontrendszer a kétszereplős játékokban, mint a sakkban vagy a góban versenyzők egymáshoz viszonyított aktuális játékerejének mérésére létrehozott rendszer. Nemzetközileg ismert neve **Elo** (gyakran nagybetűkkel **ELO**, bár nem betűszó). Nevét Élő Árpád (angol, külföldön ismertebb nevén Arpad Elo) magyar születésű amerikai fizikaprofesszorról kapta.

bonyolult számításokat végeznie, hogy a lehetséges lépéseket meg tudja határozni. A második réteg a huszárok miatt fontos, mert egy huszár a pálya szélén két mezőig tud kiugrani.

### Kiértékelés

A Shannon-féle kiértékelést használja, amelyet a Deep Blue-nál már részleteztünk.

### Kereső algoritmusok

Kereső algoritmusként a Nulla-lépés metszést alkalmazza. A Nulla-lépéses metszés az előremetsző metódusok közül a legelterjedtebb, a lényegesen redukált keresési fájának és a komoly taktikai erejének köszönhetően.

A Nulla-lépés keresés azon a feltevésen alapul, hogy minden sakkállásban a legrosszabb lépés az, ha nem lépünk.  Ez megengedi, hogy az előre láthatóan gyenge lépéseket még a keresés és kiértékelés előtt kizárja a program az optimális lehetőségek halmazából.

A Nulla-lépés keresést elvégezve kapunk a helyzetre egy alsó – $\alpha$ – határértéket. A lépésünkkel nem élünk, és átadjuk a lehetőséget a másik félnek. Ezután indítunk egy mélységi keresést, és elmentjük a kapott értéket. Ezt az eredményt tekinthetjük a helyzet alsó határának, mivel a legjobb lépésünkkel biztosan jobb helyzetet kapunk, mint azzal, hogy nem lépünk. Haa kapott értékünk nagyobb vagy egyenlő, mint az aktuális felső határ, metszést alkalmazunk. Amennyiben értékünk csupán az $\alpha$-nál nagyobb, egy szűkített keresést hajtunk végre. Az így nyert érték lesz az új alsó határunk. Abban az esetben, ha ez az értékünk kisebb mint az $\alpha$, amúgy sem segítene a keresésen.

A Nulla-lépés metszés legnagyobb előnye abban rejlik, hogy lépésünk elhagyása utáni helyzetünk nem lehet jobb, mintha léptünk volna. Az ellenfél által triviálisan hibásnak ítélt lépések kizárásával lényegesen szűkíthetjük a keresési terünket.

```
1.
2.  #define R 2 // depth reduction value
3.  int Search (alpha, beta, depth) {
4.  if (depth <= 0)
5.  return Evaluate(); // in practice, Quiescence() is called here
6.  // conduct a null-move search if it is legal and desired
7.  if (!InCheck() && NullOk()){
8.  MakeNullMove();
9.  // null-move search with minimal window around beta
10. value = -Search(-beta, -beta + 1, depth - R - 1);
11. UndoNullMove();
12. if (value >= beta) // cutoff in case of fail-high
13. return value;
14. }
15. // continue regular alphabeta/PVS search
16. . . .
17. }
```

7. ábra Általános nulla-lépés metszés C nyelven

Az algoritmus alapja tehát, ha az ellenfelet egy lépéses hátrányra kényszerítve sem javítunk a lépésünkkel, nem foglalkozunk vele többet.

Néhány helyzetben nem használhatjuk a nulla-lépés keresést, mivel logikailag hibás eredményt adna; ilyen például a zugzwang (lépéskényszer) pozíció. Azt a pozíciót nevezzük így, amelyben az a játékos veszít, aki jön; általában végjátékokban szokott előfordulni, hogy a királynak el kell lépnie a mellette álló gyalogtól, mert nem tud máshova lépni, így az ellenfél leütheti azt.

Ezen felül akkor sem szabad használni az algoritmust, amikor kevés bábunk van a táblán, vagy fennáll a vereség esélye, vagy királyunk sakkban van. Számos egyéb megszorítást lehet tárolni, de ezek csak opcionálisak.

### *A Deep Fritz eredményei*

Az 1995-ös Gépi Sakk Világbajnoki cím megszerzése Hong-Kongban, ahol nem kis meglepetéssel megverte a konkurens Deep Blue programot.

2002-ben a Deep Fritz döntetlent játszik Vladimir Kramnik, az akkori világbajnok ellen .

2003-ban,az X3D Fritz, egy 3D-s felülettel rendelkező Deep Fritz, döntetlent játszik Garry Kasparovval.

2005. június 23-án egy Fritz 9-es prototípus döntetlen játszik a későbbi világbajnokkal Rustam Kasimdzhanovval.

2006-ban a Deep Fritz győzelmet arat Kramnik ellen, 4-2-re.

A Deep Fritz 13-as SSDF[20] besorolása szerint 3095 ELO pontja van, és a listán a 11. helyet foglalja el a sakkprogramok között. A CCRL[21] szerinti besorolása 3050 ELO pont, és a 14-dik helyezett.

## 5. Összegzés

Ahogy az előbbi példáknál is láttuk, a sakkban a gép mára már jelentős fölénnyel rendelkezik az emberhez képest. A mai FIDE első helyezettnek és világbajnoknak Magnus Carlsen gépet tartja, melynek 2870 pontja van, míg a legjobb sakkprogramok meghaladják a 3000 pontot is. Igaz még sok időnek kell eltelnie, míg a teljes játékfát kiszámolják, hogy tökéletes játékot tudjon produkálni egy sakkprogram, de mára már látszik, hogy ebben a játékban a gépeké a győzelem. Kivételt képeznek ez alól a gyors játékok, ahol még van esélye az embernek a gép ellen, ha az nem használ adatbázist. Mivel ezek a számolások időigényesek, egy 10+1-es játékban nem igazán tudja a gép kihasználni teljes kapacitásfölényét, habár az ember ellen szólva, a gépnek nincsenek érzelmei, amik befolyásolhatnák a gyors, és logikus döntéshozatalba. Végezetül, csak ismételni tudom az elmondottakat. A mennyiség legyőzte a minőséget.

---

20 http://ssdf.bosjo.net/list.htm

21 http://www.computerchess.org.uk/ccrl/4040/

## Irodalomjegyzék

(Minden link tartalmazza a témát.)


[1] http://en.wikipedia.org/wiki/Checkmate (2013.november 9.)

[2] http://en.wikipedia.org/wiki/Deep_Blue_(chess_computer) (2013.november 9.)

[3] http://en.wikipedia.org/wiki/Deep_Blue_versus_Garry_Kasparov (2013.november 9.)

[4] http://en.wikipedia.org/wiki/Deep_Thought_(chess_computer) (2013.november 9.)

[5] http://mialmanach.mit.bme.hu/erdekessegek/fritz_sakkprogramrol (2013.november ..)

[6] http://www.nyu.edu/gsas/dept/philo/courses/mindsandmachines/Papers/mcdermott.html (2013.november.9)
McDermott, D., May 14, 1997 How Intelligent is Deep Blue?. Kiadó: New York Times.

[7] http://en.wikipedia.org/wiki/Hydra_(chess) (2013.november 9.)

[8] http://www-03.ibm.com/ibm/history/ibm100/us/en/icons/deepblue/ (2013.november 9.)

[9] http://chessok.com/?page_id=27966 (2013.november.9)

[10] http://en.wikipedia.org/wiki/Ruslan_Ponomariov (2013.november 9.)

[11] http://hu.wikipedia.org/wiki/Sakk (2013.november 9.)

[12] https://dea.lib.unideb.hu/dea/bitstream/2437/96341/1/Szakdolgozat.pdf (2013.november 9.)
Szabó István. 2010. Szakdolgozat. Debrecen.

[13] http://pdf.aminer.org/000/215/164/the_chess_monster_hydra.pdf (2013.november .).
Donninger, C. , Lorenz, U.. The Chess Monster Hydra. Paderborn


## Tartalom