PRESEK
List za mlade matematike, fizike, astronome in računalnikarje

ISSN 0351-6652
Letnik **33** (2005/2006)
Številka 5
Strani 23–26

Branko Kaučič:

KAKO DELUJEJO ŠAHOVSKI PROGRAMI (DRUGI DEL)

Ključne besede: računalništvo, miselne igre, programske rešitve, šah, postopek minimax.

Elektronska verzija: http://www.presek.si/33/1631-Kaucic.pdf

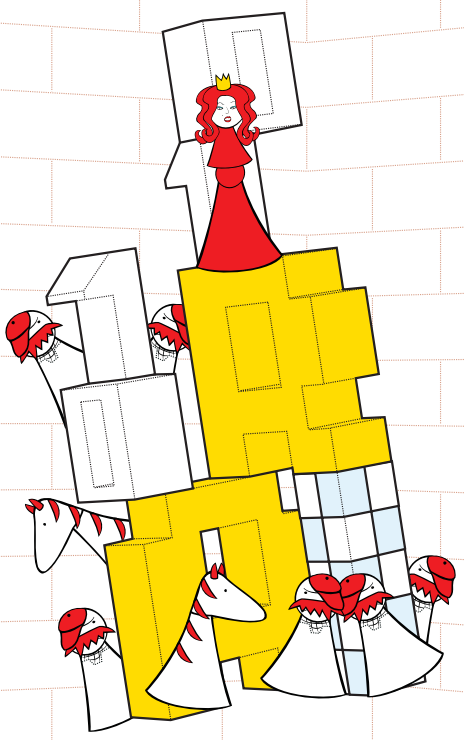
© 2006 Društvo matematikov, fizikov in astronomov Slovenije
© 2010 DMFA – založništvo

Vse pravice pridržane. Razmnoževanje ali reproduciranje celote ali posameznih delov brez predhodnjega dovoljenja založnika ni dovoljeno.

Kako delujejo šahovski programi (drugi del)

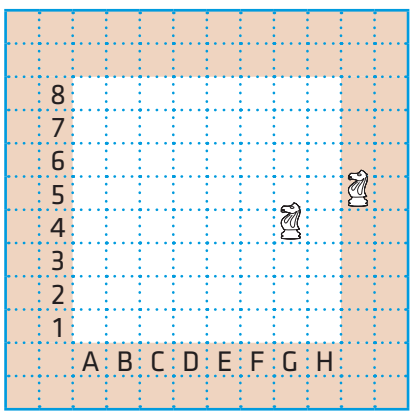
Branko Kaučič

V prvem delu (Presek 2) smo si zadali nalogo napisati računalniški šahovski program in si že ogledali glavne gradnike vsakega šahovskega programa. Povedali smo tudi, da se isti gradniki uporabijo za katerokoli računalniško igro s popolno informacijo. Stopimo v tem prispevku korak dlje in si nekoliko podrobneje poglejmo še osnove predstavitve šahovnice, premikanja figur ter iskanja naslednje poteze.



Na katerem polju stoji črna kraljica

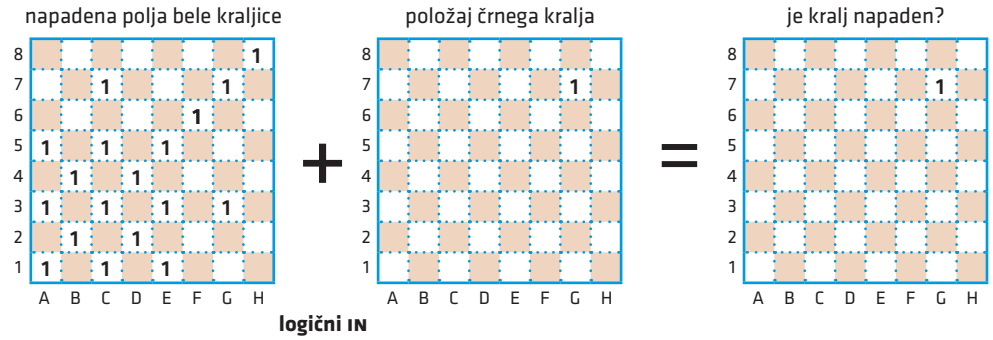
Na začetku naj poudarimo, da že več kot trideset let pametne glave niso odkrile nič revolucionarnega na tem področju (morda uspe vam!). Predstavljenih je bilo kar nekaj različnih zamisli kako bi bilo mogoče predstaviti položaj figur v računalniku. Zadnjo tako zamisel že uporabljajo vsi boljši šahovski programi, ker omogoča izredno hitro generiranje potez obeh igralcev.



Slika 1. Šahovnica, ki jo je uporabljal SARGON.

V 60-tih letih je bil pomnilnik računalnikov precej bolj cenjen kot dandanes; kompaktnejša, kot je bila predstavitev šahovnice, boljša je bila. Najočitnejša rešitev je bilo polje velikosti 8×8, kjer je vsak element polja predstavljal svojo celico šahovnice. V celici je zapisana celoštevilčna konstanta (velikosti enega zloga), ki na tem mestu predstavlja kodo figure. Hitro se je pojavilo kar nekaj izboljšav te predstavitve, od katerih sta bili najpopularnejši naslednji dve:

- Šahovski program SARGON je polje velikosti 8×8 razširil na 12×12 z dvema vrstama celic na vsaki stranici šahovnice (slika 1). Dodatne celice so predstavljale nedovoljene položaje in so omogočile hitrejše preverjanje nedovoljenih potez (npr. skok skakača izven šahovnice).
- Šahovski program MYCHESS je za šahovnico namesto 64 uporabil 32 zlogov, od katerih je vsak predstavljal po-



ložaj figure na šahovnici. Zajeto figuro je označevala vnaprej definirana konstantna vrednost, preglavice programu pa je povzročalo promoviranje kmeta.

Danes bi bila takšna predstavitev primerna le za začetno razmišljanje o šahovskem programu oz. če bi program pisali za napravo, ki je s pomnilnikom precej omejena (mobilni telefon, dlančnik).

Do revolucionarne zamisli predstavitve šahovnice se je v poznih 60-tih letih dokopal tim Kaissa iz takratne Sovjetske zveze (kaissa je sicer tudi šahu podobna igra). Zamisel, imenovana *bitna plošča,* je 64 bitni podatek, ki si ga lahko predstavljamo kot polje bitov velikosti 8×8. Z več bitnimi ploščami lahko predstavimo stanje igre v celoti. Na primer, ena plošča predstavlja položaj belih kmetov, druga plošča predstavlja položaj belih trdnjav, tretja položaj belih skakačev, položaj napadenih celic ipd. Operacija logičnega seštevanja dveh bitnih plošč omogoča nadaljne izračune v izjemno kratkem času.

Za dvomljivce pokažimo to na naslednjem primeru: v danem položaju figur nas zanima, ali je črni kralj v šahu zaradi bele kraljice. V primeru šahovnice brez bitne plošče bi zagotovo uporabili naslednji postopek:

- Na šahovnici poiščemo položaj bele kraljice, kar v najslabšem primeru pomeni, da preverimo vseh 64 celic šahovnice.
- Za vsako celico, kamor lahko pomaknemo kraljico (v vseh osmih smereh), preverimo, ali se tam nahaja črni kralj. Bližje kot je kraljica robu šahovnice, več celic preverimo. V najslabšem primeru preverimo kar lepo število celic in v nobeni od teh celic ne najdemo črnega kralja.

V primeru šahovnice z bitnimi ploščami je postopek enostavnejši:

- Vzamemo bitno ploščo, ki predstavlja položaj bele kraljice.
- Na podlagi te plošče vzamemo bitno ploščo, ki predstavlja celice, ki jih napada bela kraljica.
- Izračunamo logični IN te plošče in bitne plošče, ki predstavlja položaj črnega kralja.
- Če je rezultat različen od 0 (vsaj ena enica se pojavi v polju), potem bela kraljica s šahom napada črnega kralja (slika 2).

Ker je večina uporabljenih bitnih plošč v pomnilniku računalnika in ker operacijo logičnega IN izvedemo praktično v trenutku, je računanje s takšno zamislijo mnogo hitrejše od prejšnje.

Kam lahko premaknem črno kraljico

Generiranje potez (odločanje, katere poteze so dovoljene v danem položaju figur) in ovrednotenje njihovega položaja je kompleksnejši del vsakega šahovskega programa. Igralec ima namreč večino časa na voljo vsaj 30 ali več potez, od katerih nekatere vodijo k ugodnemu razpletu igre, druge pa v izgubo igre. Izkušeni igralec bo med njimi zlahka prepoznal ugodnejše poteze, obenem pa se zavedal, da je večina potez (od vseh možnih) premalo ugodnih za nadaljevanje igre. A kako takšno razmišljanje vcepiti našemu računalniku? V nadaljevanju podajamo opis dveh strategij generiranja potez in navodilo kdaj ju uporabiti, pred tem pa se posvetimo še pomožni tehniki v šahovskih programih, ki jo prav tako uporabljajo vsi boljši programi.

Premikalne tabele – zakladnica znanja

V šahu pogosto na več različnih načinov dosežemo isti položaj figur. Na primer, vseeno je, ali igramo najprej P-K4 in nato P-Q4, ali pa najprej P-Q4 in nato P-K4. V obeh primerih je položaj figure na šahovnici enak. Kot že rečeno, vsak položaj figur oz. stanje šahovnice ovrednotimo. Razmišljajmo dalje – če smo s takšnim ali drugačnim naporom izvedli ovrednotenje, zakaj bi ta izračun ponavljali, ko naletimo na že znano stanje figur. Podobno so razmišljali tudi drugi. Odkar je v uporabi Greenblattov program MAC HACK VI, vsi boljši šahovski programi uporabljajo premikalne tabele (ang. transposition tables). Te tabele, ki so pravzaprav zbirka ovrednotenih stanj figur, v največji možni meri preprečujejo ovrednotenje že znanega stanja figur. Ko ovrednotimo novo stanje figur, ga shranimo v takšni tabeli. Vsakič ko naletimo na novo stanje figur, najprej v tej tabeli preverimo, ali se je takšno stanje že pojavilo. Uporabimo že izračunano informacijo in se izognemu nepotrebnemu računanju. V šahovskih programih je namreč vsak privarčevani trenutek zlata vreden. Večja tabela oz. večje število različnih položajev figur je seveda zaželeno; uporabnost tabele se pokaže, kadar le-ta vsebuje več tisoč položajev. Pri milijon položajih bi imeli že pravo zakladnico znanja, a žal takšna velikost za računalniške programe ni sprejemljiva. Vsi boljši programi premikalno tabelo pri otvoritvah (začetne poteze) napolnijo z najbolj pogostimi položaji figur takoj, ko pričnemo z novo igro šaha. Pri igranju končnic

Slika 2. Uporaba bitnih plošč

(zadnje poteze šaha) v tej tabeli najdemo ovrednotena stanja figur za skoraj 90 odstotkov vseh primerov.

Nazaj k črni kraljici

Vrnimo se h generiranju potez. Leta 1949 (presenetljiva letnica, če upoštevamo dejstvo, kdaj je bil predstavljen prvi računalnik) je Claude Shannon opisal dva postopka igranja šaha:

- preverimo vse možne poteze in od vseh možnih potez spet vse možne poteze,
- preverimo le »najboljše« poteze in odgovore na najboljšo potezo nasprotnika na te poteze.

Idealno bi bilo, da bi preverili vse možne poteze in med njimi izbrali najboljšo. Žal že nekaj osnovne matematike pokaže, da bi najboljšo potezo iskali morda tudi nekaj dni (ali več). Zatorej moramo poskrbeti, da poteze generiramo hitro in kolikor se le da učinkovito. Ločimo dve strategiji:

- *selektivno generiranje,* kjer raziščemo položaj figur na šahovnici, predlagamo manjše število najbolj verjetnih potez, medtem ko vse ostale poteze ignoriramo;
- *inkrementalno generiranje,* kjer generiramo nekaj potez in upamo, da se za poteze izkaže, da so, ali tako dobre ali pa tako slabe, da se razmišljanje o njih, in o nadaljnjih potezah izkaže za primerno oz. neprimerno.

Ljudje igramo s pomočjo selektivnega generiranja – razmišljamo le o potezah, ki se nam zdijo najboljše. Na žalost rezultati dokazujejo, da se programi na podlagi selektivne strategije ne obnesejo najbolje. Dosežejo sicer stopnjo zahtevnosti srednje dobrih šahistov, a večkrat sredi najmanj primernega položaja generirajo katastrofalne poteze. Dobro znan primer tovrstne strategije je delo svetovnega šahovskega prvaka Mikhaila Botvinnika, ki je skupaj z ekipo programerjev razvijal šahovski program, ki bi čim bolje oponašal razmišljanje odličnega šahista. Program je generiral le nekaj potez, za katere je bil prepričan, da so najboljše in segajo v veliko globino (število naslednjih potez). Program žal ni dosegel vidnih uspehov. Sredi 70-ih sta Slate in Atkin dokazala, da tovrstni način ni primeren, in selektivno generiranje je v šahovskih programih skoraj izginilo.

Poiskati je torej potrebno generiranje potez, ki bo učinkovitejše od selektivnega generiranja in hitrejše od generiranja vseh možnih potez. Običajno uporabimo naslednji postopek:

- v danem položaju figur poiščemo vse dovoljene poteze,
- poteze po nekem kriteriju uredimo, s čimer upamo, da bomo pospešili naslednji korak,
- preverjamo generirane poteze.

Poteze preverjamo po eno naenkrat, dokler ne preverimo vseh potez. Pri vsaki potezi preverjamo tudi nekaj naslednjih potez in pri tem lahko ugotovimo, da poteza vodi v slabo igro. Kakor hitro to ugotovimo, prekinemo preverjanje »slabe« poteze.

Kot že rečeno, poteze uredimo. Z dobro urejenimi potezami hitreje najdemo dobro potezo in hitreje prepoznamo slabo potezo. Vsi boljši šahovski programi poteze najprej razdelijo v tri skupine: figure, ki napadajo, napadene figure in figure v mirnem položaju. Programi običajno najprej preverjajo poteze, ki zajamejo nasprotnikove figure, začenši z zajemanjem nasprotnikovih najmočnejših figur. Temu sledi promoviranje kmetov, kar lahko precej spremeni razmerje figur na šahovnici. Po drugi strani pa napadene figure običajno vodijo do prekinitve preverjanja poteze, čeprav vsak izkušen šahist pozna tudi trik žrtvovanja figure, omenjen v prejšnjem prispevku. Izkaže se, da lahko zmanjšamo število vseh možnih preverjanj iz n na $\sqrt{n}$. Zavedati pa se moramo, da to ne pomeni, da določenih potez med igro ne bomo preverili, temveč da bomo njihovo preverjanje najverjetneje le prestavili na kasneje.

Kdor išče, ta najde

Kaj smo se do sedaj naučili? Vemo, da imamo različne predstavitve šahovnic v računalniku, ki se izkažejo predvsem pri generiranju potez figur. Predpostavimo tudi, da znamo ovrednotiti stanje figur na šahovnici, ki kaže v prid nam ali nasprotniku. Še vedno pa nam manjka znanje, kako se med generiranjem potez odločimo o naslednji potezi, kar imenujemo iskanje poteze. Tehniki, ki ju bomo omenili, nista omejeni zgolj na šahovski program, ampak na igre dveh igralcev s popolno informacijo.

V nadaljevanju se bomo torej posvetili razmišljanju, kateri od igralcev je boljši in kateri slabši oz. ali poteze vodijo v zmago ali poraz. Roko na srce, v šahu je tako velika množica vzorcev, pravil in izjem, da tudi najboljši šahovski programi v nekaterih trenutkih zatajijo. Dobri so »le« v hitrem izračunavanju, kar moramo čim bolje izkoristiti. Programi torej namesto, da z logičnim razmišljanjem generirajo dobre poteze, generirajo vse možne poteze, od teh potez vse možne poteze nasprotnika, od nasprotnikovih potez spet vse svoje poteze itn. Zamislimo si npr. potezo skakača na polje, od koder lahko napade trdnjavo in kraljico. Dober šahovski program bo za nekaj potez vnaprej preveril, kaj se bo zgodilo, če bo nasprotnik zaščitil trdnjavo, in kaj, če bo zaščitil kraljico. Ker v tem postopku iskanja preverimo vse možne poteze (naše poteze in odgovore nasprotnika), se bomo zagotovo lahko odločili za najboljšo potezo. Globlje, kot bomo iskali, bolje se lahko odločimo. Opozorimo pa vendarle, da smo omejeni z globino iskanja, torej s številom naših in nasprotnikovih potez, zato se za najboljšo potezo odločimo na podlagi zmožnosti. Lahko smo namreč omejeni s časom iskanja poteze in/ali z velikostjo razpoložljivega pomnilnika.

Postopek minmax

Zamislimo si, da v igri sodelujeta dva igralca – prvega imenujmo Max, njegovega nasprotnika pa Min. Predpostavimo, da je ovrednotenje figur funkcija, ki vrne pozitivno vrednost, kadar je v prednosti Max, negativno vrednost, kadar je v prednosti Min, in vrednost 0, kadar je enakovredno stanje sil na šahovnici. Naloga Maxa je, da izbere potezo, ki bo povečala vrednost stanja igre, medtem ko je naloga Mina, da izbere potezo, ki bo zmanjšala vrednost stanja igre. Predpostavimo tudi, da igralca poteze izbirata brez napak, torej da se vedno odločita za potezo, ki je v prid igralcu samemu in ne nasprotniku.

Oglejmo si to na primeru. Recimo, da je na potezi Max in ima na voljo potezi *a* in *b*. Ne glede na izbiro poteze je za njim na vrsti Min, ki lahko izbere potezo *c* ali *d*. Zamislimo si vse možne kombinacije potez (imenujemo jih iskalne poti) in jih ovrednotimo (slika 3) – če Max izbere potezo *b* in Min potezo *c*, je stanje igre ovrednoteno z 9.

Kako Max izbere potezo, ki jo bo igral? Max predvideva, da bo Min vedno odigral najboljšo možno potezo. Zatorej, če bo igral potezo *a*, bo Min odgovoril s potezo *d*, kar bi privedlo do vrednosti –10 (zmaga Min). Ve pa tudi, če bo igral potezo *b*, bo ne glede na potezo Mina (*c* ali *d*) stanje igre njemu v prid. Brez nadaljnjega se bo Max odločil za potezo *b*, čeprav bi lahko imel boljše stanje igre, če bi izbral potezo *a* in upal, da bo Min izbral potezo *c*. Ampak, kot že rečeno, Min bo vedno izbral zase najboljšo potezo, kar zagotovo ne bo *c*.

Težava postopka minmax, ki morda ni takoj razvidna v tem preprostem primeru, nastopi, kadar obstaja eksponentno število poti, ki jih moramo preveriti. Ločimo dva faktorja, ki opisujeta naraščanje števila možnih poti:

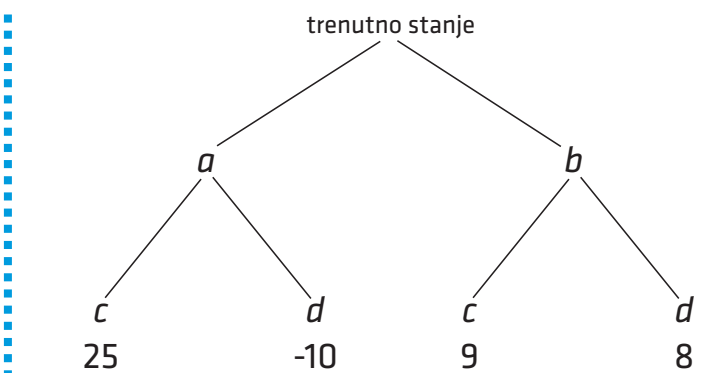
- globina iskanja, ki predstavlja število zaporednih preverjenih potez. Globina 6 pomeni tri poteze igralca Max in tri poteze igralca Min.
- Vejitveni faktor, ki predstavlja število možnih potez vsakega igralca na dani globini.

Zgornji primer je imel vejitveni faktor in globino iskanja enako 2, pri igranju šaha pa imamo v sredini igre vejitveni faktor okoli 35 potez in iskanje v globino 8 potez. 35^8 predstavlja že neverjetnih 2.251.875.390.625 različnih poti! Si drznete računati poteze v globino 9 ali celo 10? K sreči obstajajo načini, da ne preverjamo vseh možnih poti.

Postopek alfabeta

V postopku alfabeta postopek minmax privedemo do obvladljive situacije. Recimo, da smo najprej preiskali stanje igre, ko Max izbere potezo *b*. Vemo, da bo v najslabšem primeru stanje igre vsaj 8. Nadaljujemo s preverjanjem poteze *a* tako, da najprej preverimo potezo *d*. Pot nas vodi do rezultata –25, kar je za Maxa nezaželeno stanje igre. Vemo tudi, da bo ne glede na vrednost pri potezi *c* stanje igre vsaj –25. Če bo vrednost stanja pri potezi *c* namreč večje od –25, bo Min izbral potezo *d*, sicer bo izbral potezo *c*. Zakaj bi torej nadaljevali s pregledovanjem preostalih potez na poti, če že imamo informacijo, da je igralec Min v boljšem položaju?

Osnovna ideja postopka alfabeta je, da ko enkrat generiramo dobro potezo, v iskanju poti takoj odstranimo vsa nadaljnja računanja takoj, ko naletimo na slabše stanje igre. Izkaže se, da je v šahu ogromno takšnih potez. S pomočjo premikalnih tabel, vejitvenega faktorja 23 in globine 8 s postopkom alfabeta v sredini igre običajno preverimo le še približno 2000 različnih poti.



Slika 3. Postopek minmax

Naslednjič

Pustimo za naslednjič nadaljnjo optimizacijo postopka alfabeta in nekatere pomembne malenkosti. Do takrat pa se lahko poigrate s postopkoma minmax in alfabeta ter razmislite o postopku za igro z več kot dvema igralcema.

Viri

- *http://www.geocities.com/SiliconValley/Lab/7378/comphis.htm*
- *http://en.wikipedia.org/wiki/šahovski_računalniški_programi*
- *http://www.cs.biu.ac.il/~davoudo/tutorials.html*