Dejan Tomić

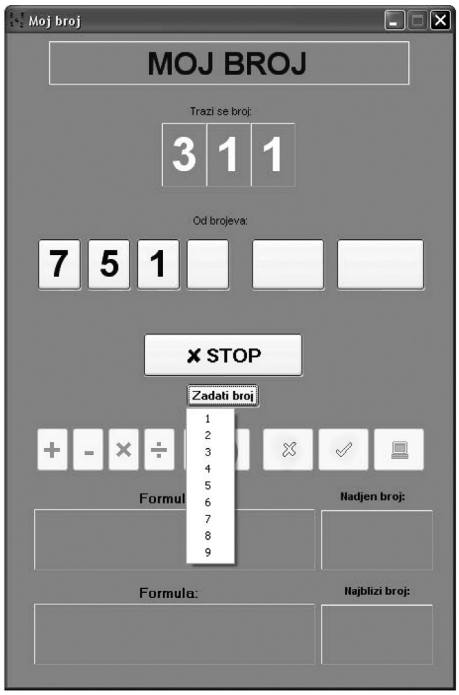
Igra „Moj broj“ sa automatskim traženjem tačnog ili najbližeg rešenja

Projekat „Moj broj“ predstavlja program za rešavanje problema iz istoimene igre, poznate iz kviza „TV slagalica“. Program na osnovu zadatih brojeva traži izraz čija je vrednost jednaka (ili što bliža) zadatom tačnom broju. Glavni cilj projekta je da proces traženja rešenja bude što brži. Zbog toga se pamte svi izračunati međurezultati, kako bi se izbeglo njihovo ponovno računanje u svim daljim izrazima u kojima se pojave. Izbegava se i računanje kombinacija za koje se zna da će dati isti rezultat. Ovaj problem se može rešiti i upotrebom „brute-force“ metode koja je računski zahtevnija od prethodno navedenog postupka, budući da ponovo računa već izračunate međurezultate i ne vodi računa o ekvivalentnim kombinacijama. Testiranjem je utvrđeno da prva metoda radi i do 400 puta brže. Pokrenut na procesoru Intel Core2Duo T7700 na 2.4 GHz program je u najgorem slučaju (kada tačan broj nije mogao da se dobije) radio ne duže od 100 milisekundi.

Uvod

Igra „Moj broj“ je jedna od igara iz popularnog domaćeg kviza „TV slagalica“ koji se emituje svakog radnog dana na RTS-u. Cilj igre je da se od 6 ponuđenih, slučajno izabranih, brojeva (ne obavezno različitih), korišćenjem četiri osnovne matematičke operacije (sabiranja, oduzimanja, množenja i deljenja) i zagrada sastavi izraz čija je vrednost jednaka, ili što približnija, zadatom broju (drugačije zvan i „tačan“ ili „traženi“ broj). Svaki od 6 ponuđenih brojeva se može upotrebiti najviše jednom u izrazu. Prva četiri od šest ponuđenih brojeva su jednocifreni i veći od nule. Na petom mestu je jedan od brojeva 10, 15, 20 ili 25, a na poslednjem 25, 50, 75 ili 100.

Ovaj rad je prvenstveno namenjen rešavanju problema iz igre „Moj broj“. Osim što se brojevi okreću slučajnim redom i mogu da se zaustave, baš kao i u igri mogu da se zadaju i brojevi po želji: umesto dugmeta „stop“ može se kliknuti na „biraj broj“ (slika 1) i izabrati broj koji će biti upisan posle zaustavljanja. Nakon izbora brojeva korisnik može da sastavlja matematički izraz, a program da računa njegovu vrednost, ili može program sam da traži rešenje – izraz koji daje tačan broj, ili ako takav ne postoji, onda onaj koji daje najbliži broj.



Slika 1. Program „Moj broj”, sa prikazanim menijem za postavljanje proizvoljnog broja

Figure 1. The program window, with a popup menu for choosing a custom number

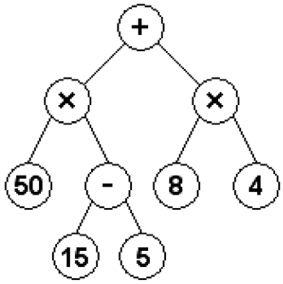
Dejan Tomić (1992), Pančevo, Ljube Kovačevića 12/23, učenik 2. razreda Matematičke gimnazije u Beogradu

MENTORI:
Miloš Savić, PMF Novi Sad
Dragan Toroman, ISP

Metode

Testirano je nekoliko različitih metoda za izračunavanje unetog aritmetičkog izraza, traženje tačnog rešenja i ispis izraza kojim se to rešenje dobija. Tako, na primer, aritmetički izraz koji korisnik unese može se izračunati na sledeće načine:

- Izraz se sačuva kao stek sa brojevima, matematičkim operacijama i zagradama, a zatim prevede u postfiksnu notaciju, tj napravi se drugi stek u kome se zagrade ne koriste. Vrednost takvog izraza se može izračunati veoma lako, budući da se u njemu svaka operacija nalazi iza oba operanda (dakle, izraz (3 + 2) × 5 bi se u postfiksnom obliku zapisao kao 3 2 + 5 *);
- Izraz se može zapisati i u obliku binarnog stabla, tako što se u koren stabla zapiše operacija, a leva i desna grana mogu biti ili listovi (u koje se zapisuje po jedan broj) ili podstabla (tj, drugi izrazi, koji imaju ista svojstva kao i glavno stablo) (slika 2);



Slika 2. Primer zapisa izraza u binarnom stablu

Figure 2. An example of storing the expression as a binary tree

- Izraz se može računati i bez prevođenja u drugi format: potrebno je da se zapamti levi operand i operacija, a kada se pročita ceo desni operand računa se vrednost. Svi elementi izraza (brojevi, operacije i zagrade) se čitaju redom kojim su zapisani u izrazu. Zagrade dodatno komplikuju ovu metodu računanja izraza – i zbog toga se ona ni ne koristi u računarskim programima, već se izraz prevodi u neki drugi format lakši za računanje.

Za projekat je izabrana prva metoda: izraz koji korisnik napiše prevodi se u postfiksni oblik (drugačije zvan i obrnuta poljska notacija) korišćenjem dva steka: jedan za sam postfiksni izraz (glavni stek), a drugi za operacije i zagrade (pomoćni stek). Zagrade se ne dodaju na glavni stek, nego samo privremeno na pomoćni zbog prioriteta operacija.

Računanje formule zapisane na ovaj način je trivijalno – koristi se pomoćni stek, na kome se pamti rezultat, i na koji se dodaju samo brojevi, a operacije se odmah izvršavaju – umesto poslednja dva broja (A i B) dodata na pomoćni stek, upisuje se rezultat operacije (A **op** B, gde je **op** tip operacije). Na kraju na pomoćnom steku ostaje samo jedan broj, i to je vrednost aritmetičkog izraza.

Za traženje tačnog rešenja postoje sledeći načini:

- Standardni, najlakši ali i najsporiji način: *brute-force search* – isprobavanje svih mogućih (i potrebnih i nepotrebnih) kombinacija. Ovaj način je odmah odbačen zbog eksponencijalne vremenske složenosti (Chang 2003);
- Optimizovani *brute-force search* – izbegavanje nepotrebnih kombinacija, na primer B+A nije potrebno računati ako smo već računali A+B (komutativnost). Vremenska složenost *brute-force* algoritma je eksponencijalna, pa je eksponencijalno i ubrzanje koje se postiže optimizacijama;
  Postoje dve varijacije *brute-force* algoritma: može se od ponuđenih brojeva tražiti izraz koji daje tačan broj, ili od ponuđenih brojeva i tačnog broja tražiti izraz čija je vrednost nula. Obe metode se svode na isto i otprilike isto vremena i rade, tako da je svejedno koju od njih koristiti;
- Umesto *brute-force* metode i računanja celih izraza svaki put, mogu se pamtiti izračunati rezultati i koristiti se u daljem izvršavanju. Ovaj metod troši malo više memorije, ali zato ne mora da ponovo računa operande izraza – jer su oba već izračunata ranije.

U ovom projektu se koristi poslednja metoda. Algoritam je iterativni i ima optimizacije, iste koje se mogu dodati i u *brute-force* algoritam – jer bez obzira da li se izračunati međurezultati pamte i da li je algoritam rekurzivan ili iterativan, optimizacije se mogu dodati na skoro isti način.

Program traži najbliže rešenje na sledeći način:

1. Prvo doda svih 6 ponuđenih brojeva na jedan stek;

2. Zatim računa sve kombinacije koje se mogu dobiti pomoću neka dva od tih brojeva (to su kombinacije svih parova brojeva i operacija). Naravno, uzimaju se uređeni parovi brojeva kako se ne bi računala oba međusobno komutativna izraza – već se računa A+B, A–B ili B–A, A×B, A/B i B/A. Nikad se ne računaju obe kombinacije A–B i B–A – uvek se pamte samo izrazi sa pozitivnim rezultatom, jer je bitna samo apsolutna vrednost razlike;
3. Kada se izračunaju sve kombinacije dva broja, prelazi se na kombinacije tri, pa četiri, pa pet, i najzad svih šest brojeva. Međutim, potrebno je voditi računa o tome da se izrazi sa četiri broja mogu dobiti kao kombinacija jednog izraza sa tri broja, jednog broja i operacije, ili kao kombinacija dva izraza sa po dva broja i operacije. Slično važi i za izraze od pet ili šest brojeva. Kada se o tome ne bi vodilo računa, ne bi se mogle dobiti kombinacije sa proizvodom ili količnikom dva zbira (ili dve razlike), kao ni sa zbirom ili razlikom dva proizvoda (ili količnika);
4. Nakon svakog novog izračunatog izraza proverava se razlika vrednosti tog izraza i tačnog broja – ukoliko je apsolutna vrednost te razlike manja od trenutne minimalne, dobijen je novi najbliži broj, i izraz koji je do pre toga davao najbliži broj se zamenjuje novim. U slučaju da je razlika jednaka nuli, dobijen je tačan broj i prekida se dalje traženje rešenja;
5. Nakon završetka traženja rešenja, program ispisuje najbliži broj koji je dobijen, kao i izraz kojim je dobijen taj broj.

U algoritam za traženje rešenja ubačene su razne optimizacije – pored navedene provere komutativnosti, nikad se ne množi i ne deli sa jedan – a takođe se izrazi koji daju rezultat nula uopšte ne pamte na steku, tako da se uopšte ni ne koriste u daljem računanju. Osim toga, ne deli se sa proizvodom ili količnikom (zbog pravila suprotnih operacija – A:(B:C) = A:B×C), kao što se ni ne oduzima zbir ili razlika.

Sledeća optimizacija algoritma je provera asocijativnosti – (A×B)×C=(A×C)×B=(B×C)×A (slično i za sabiranje). Takođe, lista ponuđenih brojeva se može zapisati i kao niz sa dva parametra – koji broj i koliko puta se pojavljuje među ponuđenim. Ovim postupkom se izbegava upotreba istog broja više puta u istoj kombinaciji, pa time i računanje istih izraza.

Izrazi koji se računaju tokom traženja najbližeg rešenja se zapisuju u obliku binarnog stabla (slika 2) – pamte se operacija, memorijske adrese levog i desnog operanda, vrednost izraza nakon što je izračunat (potrebno da se ne bi ceo izraz računao ponovo), kao i svi brojevi koji su korišćeni (ovo je potrebno da bi se sprečilo korišćenje istog broja više puta nego što ga ima).

Formula koja daje najbliže rešenje se onda ispisuje tako što se binarno stablo (u kome je ona zapisana) obilazi u redosledu *in-order* (prvo se ispiše levi izraz-operand, pa operacija, pa desni izraz-operand). U program je ubačena i provera prioriteta operacija, tako da se zagrade dodaju samo tamo gde je to potrebno.

Rezultati i diskusija

Nakon testiranja svih navedenih metoda, na procesoru Intel Core2Duo T7700 2.4 GHz (pri čemu program koristi samo jedno procesorsko jezgro), utvrđena su vremena njihovog izvršavanja (slika 3). Za testiranje brzine traženja najbližeg broja, korišćen je slučaj u kojem tačan broj ne može da se dobije.

- Klasični *brute-force* algoritam bez optimizacija radi preko 45 sekundi i računa ceo izraz oko 30 miliona puta; vremenska kompleksnost je eksponencijalna – $O(2^n)$, tačnije oko $(2n-1)!$ izračunavanja (gde je $n = 6$ broj ponuđenih brojeva);
- Nerekurzivni algoritam sa pamćenjem izračunatih rezultata (i proverom komutativnosti) radi nešto manje od 2 sekunde i računa prost izraz (izraz koji se sastoji od jedne matematičke operacije) oko 2 miliona puta; ubrzanje se postiže zahvaljujući pamćenju izračunatih rezultata, a dodata provera komutativnosti daje još veće ubrzanje; vremenska kompleksnost je i ovde eksponencijalna – ali je manja osnova, te ovaj algoritam radi mnogo brže nego *brute-force*;
- Nakon izbačenog množenja i deljenja sa 1 i korišćenja nule, program radi oko 1.2 s;
- Posle treće optimizacije (ne oduzimati zbir/razliku; ne deliti sa proizvodom/ količnikom) ukupno vreme rada se smanjuje na svega oko 0.4 sekunde;

Test primer	Rešenje	Brute-force	Sa pamćenjem rezultata	Bez upotrebe razlomaka	Drugi program
(+) 200 od 9, 4, 3, 1, 15, 75	(9-1)×75:3=200	~ 16ms 4286 izraza	< 1ms 2494 op.	< 1ms 1534 op.	~ 1,3s ~216000 op.
(++) 551 od 2, 2, 1, 3, 15, 25	(2+2+3+15)×25 +1=551	~ 62ms 24302 izraza	~ 16ms 111630 oper.	~ 15ms 41721 op.	~ 1,2s ~234000 op.
(+++) 999 od 2, 2, 2, 2, 10, 25	2×2×10×25- 2:2=999	~ 210ms 52910 izr.	~ 141ms 685771 op.	~ 47ms 285501 op.	~ 4,5s ~781000 op.
(+++) 963 od 1, 2, 4, 8, 10, 25	(25-1)×4×10 +2=962	~ 50s 33665406 izr.	~ 125ms 646693 op.	~ 32ms 176342 op.	~ 3,2s ~527000 op.
(++) 396 od 4, 7, 7, 4, 10, 75	(7+7+10+75)×4 =396	~ 453ms 202242 izr.	~ 16ms 44783 op.	~ 16ms 18849 op.	~ 2,2s ~365000 op.

Slika 3. Rezultati testa

Figure 3. Testing results:
(from left)
Test example
Solution
Brute-force
With result memorizing
Without use of fractions
Second program

– I najzad, nakon dodavanja pravila asocijativnosti maksimalno vreme izvršavanja algoritma postaje manje od 100 milisekundi.

Napomena: Neki rezultati se ne mogu dobiti ako se deli samo ono što je deljivo bez ostatka; ukoliko bi se koristili i razlomci (3/4; 1/5 itd) moglo bi se dobiti mnogo više rezultata. U igri „Moj broj“ je zvanično dozvoljeno samo korišćenje celih brojeva, dok program nije samo na njih ograničen. Ukoliko bi to bio slučaj, program ne bi radio duže od 20 milisekundi.

Najbrži (i jedini domaći) program pronađen na Internetu, koji rešava problem iz igre „Moj broj“, na istom procesoru radi 5 do 6 sekundi (a pošto nema isprogramirane slučajeve zbira proizvoda i proizvoda zbirova ne nađe uvek tačno rešenje). Autor tog programa navodi da program koristi brzu rekurzivnu pretragu rešenja.

Rezultati pokazuju da je izbegavanje rekurzije i pamćenje rezultata mnogo brže nego *brute-force*. To je zato što se operandi novih izraza ne moraju računati ponovo, već se koriste njihove već izračunate vrednosti. Pri tome je dodatno ubrzanje postignuto ubacivanjem provere asocijativnosti i komutativnosti – budući da za komutativnost postoje dva ekvivalentna izraza, a za asocijativnost tri, ova dva pravila ubrzala su algoritam oko 30 puta. Sve optimizacije zajedno su smanjile vreme izvršavanja algoritma sa oko 10 sekundi na ispod 100 milisekundi (ukoliko bi se računale i kombinacije sa razlomcima), što je veoma značajno ubrzanje.

Napomene

1. Preciznost vremena kod jednostavnijih test primera je manja zbog ograničenja Windowsa (rezultat GetTickCount funkcije korišćene za merenje vremena osvežava se svakih 15-16 ms);

2. Broj znakova + označava kompleksnost test primera (koliko teško / sa koliko brojeva se može naći tačan broj, koliko dugo programu treba da nađe rešenje);

3. Može se primetiti da teži test primeri ne povećavaju vreme izvršavanja kod svih algoritama – *brute-force* je u jednom primeru sporiji nego u drugom, dok je kod dinamičkog obrnuto;

4. Kod *brute-force* algoritma meri se broj računanja celog postfiksnog izraza, a kod dinamičkog broj prostih računica (matematičkih operacija);

5. Konkurentski program ima preciznost na 1000 operacija. Vreme izvršavanja ne prikazuje, tako da je ono moralo ručno da se računa pa nije precizno.

Zaključak

Program trenutno ima sledeće mogućnosti: Ručno zadavanje brojeva ili zaustavljanje okretanja slučajnih cifara; mogućnost pisanja i računanja izraza, pri čemu vodi računa da izraz nikada ne bude neispravan; mogućnost (veoma brzog) traženja formule koja daje tačan broj, ili njemu najbliži ukoliko tačan ne može da se dobije; ispisivanje dobijenog broja i formule kojom se dobija.

Poređenjem rezultata dobijenih testiranjem ovog programa i programa drugog autora, sa različitim test primerima i metodama, potvrđuje se prednost pamćenja međurezultata i provere osnovnih matematičkih pravila u odnosu na *brute-force*.

Literatura

Brown B. 2001. Postfix Notation Mini-Lecture. Dostupno na: http://www.developer.rs.sr/download/download.php?file=544 [6. 10. 2009]

Chang S. 2003. *Data structures and algorithms*. Singapore: World Scientific

Dejan Tomić

The “My Number” Game with Computation of the Exact or Nearest Solution

The “My Number” project is a software for solving the problem of one of the games from the TV Show/Quiz “TV Slagalica” (TV Puzzle) which is shown on Serbia's National TV station (RTS). The program computes a mathematical expression consisting of given numbers, which evaluates to the (also given) result. If no expression which evaluates to the exact result exists, then the closest result is computed.

The main goal of the project (apart from giving correct results) was to make the computing process/algorithm as fast as possible. All the computed intermediate calculations are maintained thus avoiding reduntant calculations of same sub-expressions. Also, all the combinations known to give the same results are detected and avoided. It is also possible to solve this problem using the “Brute-force” approach, but this is computationaly more demanding, because it will go trough all possible combinations, without avoiding redundant cases and equivalent expressions.

Tests have shown that the proposed method works up to 400 times faster. Running on an Intel Core2Duo T7700 CPU (2.4 Ghz), using a worst-case scenario (in which the exact answer/solution does not exist), execution time is not longer than 100 ms.