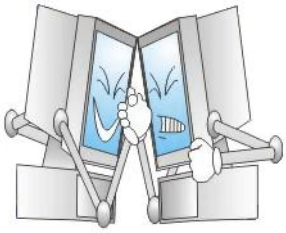

Uobičajne greške na nacionalnim takmičenjima iz programiranja

Autor: Andreja Ilić
e-mail: andrejko.ilic@gmail.com

Svake godine Komisija za organizaciju takmičenja iz programiranja organizuje takmičenja za učenike osnovnih i srednjih škola. Godišnji ciklus takmičenja se sastoji od četiri nivoa takmičenja: okružnog takmičenja, državnog takmičenja, srpske informatičke olimpijade (SIO) i izbornog takmičenja. Izborno takmičenje se organizuje radi odabira nacionalnog tima, koji će učestvovati na međunarodnim takmičenjima. Ovaj nivo takmičenja je vanredan nivo, koji ne mora uvek biti održan. Nedavno je uvedeno i dodatno kvalifikaciono takmičenje za učenike srednjih škola, kao takmičenje najnižeg ranga.

Ovaj članak namenjen je svim takmičarima, kako početnicima tako i onim iskusnijim. Učenici koji počinju svoje takmičarske dane, među ovim redovima mogu naći korisne savete i unapred se upoznati sa nekim uobičajnim greškama bivših takmičara. Sa druge strane, iskusnijim takmičarima bih preporučio da sve jedno prođu kroz ovaj tekst, jer nekada ove greške mogu imati velikog uticaja. Iskustvo je samo lep naziv za kojim nazivamo naše greške.

Sav materijal je sakupljen tokom dugogodišnjeg autorovog učešća na nacionalnim i međunarodnim takmičenjima - bilo kao takmičar ili kao član komisije. Sve što budete pročitali barem jednom se stvarno dogodilo. Dalji tekst je organizovan u manje sekcije gde će svaka od njih analizirati neku od grešaka (ili klasu sličnih). Sekcije nisu sortirane, međutim preporučljivo je čitati ih redom zbog raznih komentara i sugestija. Neke od njih će sadržati delove naznačene kao savet**,** na koje treba obratiti posebnu pažnju.

Ukoliko imate komentara ili dodatnih sugestija povodom teksta, nemojte oklevati da se obratite bilo autoru bilo takmičarskoj komisiji.

Datoteke

Jedna od najstandardnijih grešaka naših takmičara, nažalost ne retko i onih iskusnijih, vezana je za samu komunikaciju sa datotekama. Najfrekventnije su sledeće:

- ***Pogrešan naziv ulazne / izlazne datoteke*****:** Pod ovim podrazumevamo i same štamparske greške ali i one gde se datotekama pristupa relativnom putanjom (primera radi „D:\Okruzno takmicenje\nazivProblema.in“). Ukoliko i pristupate relativnom putanjom u toku rada, kada predajete svoj kod pazite da ovu putanju promenite.

- ***Ispisivanje dodatnih informacija u izlaznoj datoteci:*** Takmičari često štampaju dodatne informacije u datotekama prilikom analize i testiranja njihovih programa. Naravno ovo je u redu koristiti pri debug-iranju, ali treba voditi računa da se na kraju ove stvari uklone iz koda. Pored

toga pažnju posvetiti i formatu izlaza – ne stavljati novi red na kraju, brojeve razdvajati na način specifiran u problemu...

- ***Nezatvaranje izlazne datoteke:*** U *Pascal*-u ukoliko se nakon ispisa traženih informacija u datoteku, naročito onih većih, datoteka ne zatvori (`Close()`) neke od informacija koje su upisane se mogu izgubiti. Zatvaranje datoteka podrazumeva i flush-ovanje podataka, tako da možete biti sigurni da podaci neće „iscureti".

Mnogi će reći da su ovo nebitne greške koje nemaju veze sa samim algoritmom, što je tačno, i kao takve ukoliko do njih dođe treba ih ispraviti nakon takmičenja. Međutim u pravilniku piše: ***„Žalbe koje su zasnovane na izvornom kodu ili ispisu u izlazni fajl se po strogom pravilu odbijaju. Takmičar nema pravo da traži da mu se ispravi greška u izvornom kodu, bez obzira koliko mala ona bila."*** Možda zvuči surovo ali ovakva pravila moraju postojati. U suprotnom vrlo brzo bi došlo do pitanja gde onda povući granicu za menjanje koda, pošto će se dobijati svakakve žalbe šablona: ukoliko se u mom kodu na tom i tom mestu promeni karakter x u y moj algoritam bi bio korektan.

Uvek odvojite 10ak minuta na kraju takmičenja gde ćete proveriti ovu i slične moguće greške (neke od njih ćemo obratiti u daljem tekstu) koje vas mogu skupo koštati. Zadnjih 10ak minuta vam ne mogu mnogo značiti u implementaciji ideja, ali vas svakako mogu spasiti ovih previda.

Veličine nizova i matrica

Pored gore navedenog problema sa datotekama, takmičari često prave slične greške vezane za veličine nizova ili matrica. Starija okruženja imaju jako mali memorijski limit (kod ranijih verzija *Pascal*-a čak 64k). Takmičari su tada prinuđeni da pri kodiranju i testiranju definišu male dimenzije niza i matrica, sa idejom da na kraju promene na odgovarajuće. Takva implementacija se ne bi kompajlirala na lokalnom računaru, međutim na takmičenjima predati kodovi kasnije komplajliraju zvaničnim kompajlerom koji podržava memorijska ograničenja data u problemu[1]. Međutim neretko takmičari zaborave da promene ove vrednosti, što nažalost dovodi to toga da ideja dobija mnogo manje bodova nego što zaslužuje. Na kraju obavezno proverite ograničenja!

Korisno bi bilo veličine nizova i matrica pamtiti kao konstante na početku programa. Ukoliko bi koristili hardkodiranu veličinu (što nikada nije preporučljivo), na kraju bi morali da menjate dosta vrednosti u implementaciji – dok je u ovom slučaju dovoljno promeniti samo par karaktera. Sa druge strane i sam kod je pregledniji.

```
#define MAXN 1001     odnosno     const MAXN = 1001;
```

Preporučljivo je da definisati veličine barem za 1 veće od onih datih u problemu. Ovim možete izbeći neke dodatna ispitivanja za granice indekasa ukoliko imate određene rekurentne veze. Takođe, ne zaboravite da u *C/C++* -u indeksi počinju od 0.

[1] Detaljnije o programskim okruženjima i kompajlerima koji se koriste na zvaničnim takmičenjima možete pogledati u pravilniku za srednjoškolsla takmičenja na adresi: http://www.yuoi.nis.edu.rs/pravilnik.html

Opseg promenjivih

Još jedana jako jednostavna, a kobna, greška je vezan za opseg promenjivih. Takmičari jako brzo prelazne sa samu implementacionu fazu, tako da često zaboravljaju da posvete pažnju ograničenjima promenjivih – kako ulaznih tako i onih potrebnih za čuvanje međurezultata.

Zavisno od programskog jezika koji koristite, tipovi podataka imaju različite opsege. Primera radi u *Pascal*-u tip podataka `Integer` ima opseg $[-32.768, 32.767]$ dok `LongInt` obuhvata $[-2.147.483.648, 2.147.483.647]$. Skoro da ne prođe takmičenja a da neki učenik nije obratio pažnju na ulazna ograničenja i definisao elemente niza tipom koji ne može obuhvatiti vrednosti. Malo komplikovaniji problem je definisanje tipa za neke usputne promenjive. Često je potrebno njih definisati nekim većim tipom: `Int64` za *Pascal* odnosno `long long` za *C/C++*.

Pre početka same implementacije pokušajte da na papiru izdvojite okvirno koje usputne promenjive i strukture će vam biti potrebne. Posebnu pažnju obratite na tipove kao i moguć opseg pomoćnih promenjivih. Ukoliko niste sigurno i dvoumite se između dva tipa, a memorijsko ograničenje vam to dopušta, uvek odaberite veći.

Debug-iranje preko ekrana

Tokom srednjoškolskog školovanja, a nažalost nekada i akademskog, većina učenika nije imala priliku da se upozna sa **Debug** okruženjem. Alternativni način ispravljanja implementacionih grešaka je ispisivanje dodatnih informacija na ekran (trenutne vrednosti promeljivih, uslova, vrednosti određenih izraza...). Dodatno ispisivanje može dosta povećati vreme izvršavanja, tako da takmičari mogu dobiti *Time Limit Exceeded* za pojedine test primere iako je algoritam korektan (pošto se standardni izlaz ignoriše ukoliko se u problemu traži ispis u datoteci). Sa druge strane ukoliko se ispis rešenja vrši na standardni izlaz, a ne preko datoteka, dobijamo netačno rešenje zbog suvišnih infromacija.

Kod pojedinih *C/C++* kompajlera nakon izvršavanja programa, komandni prozor se automatski zatvara. Zbog ovoga nije moguće videti rešenje ili dodatne informacija koje su ispisane. Kako bi se ovo zaustavilo, takmičari često na kraju pograma:

- pauziraju proces: `system("pause");`
- učitavaju dodatni karakter: `scanf ("%c", &c)` karakter `ReadLn(s)`

Oba prisupa će nam pomoći u testiranju. Međutim, kako i kod nekih ranijih grešaka o kojima smo pričali, nikako nemojte zaboraviti da uklonite ovo pre predaje kodova.

Moduo

Operacija koja računa ostatak pri deljenju[2] dva cela broja se naziva moduo. Skoro sigurno se svaki takmičar nekada upustio u koštac sa nekim od zamki koje moduo može da nam postavi. Ovde ćemo izneti neke od njih. U daljem tekstu ćemo operaciju označavati sa mod.

- ***Problem se performansama**:* Računanje modula je jako sporo. Neki kompajleri imaju otpimizacije za odredjenje vrednosti modula. U većini slučajeva, moduo je implementiran kao:

$$a \bmod m = a - n \left\lfloor \frac{a}{n} \right\rfloor .$$

  Jako često je potrebno tražiti ostatak pri deljenju sa stepenom dvojke. Ovo se može izračunati u konstantom vremenu koristeći bit operacije:

$$x \ mod \ 2^n = x \ and \ (2^n - 1)$$

- ***Rezultat po modulu**:* Kod nekih takmičarskih problema, rezultat može biti jako veliki broj. Kako se od takmičara ne bi dodatno zahtevalo da implementira rad sa velikim brojevima, obično se traži rezultat po nekom datom modulu. Za vrednost modula se uzima veliki prost broj, kako bi se dodatno izbeglo dobijanje tačnog rezultata za nekorektan algoritam (Zašto?). Međutim, većina takmičara greši tako što rešenje tokom celog algoritma vuče i tek na kraju pre štampanja uzme njegov moduo. Na ovaj način dobija se prekoračenje.

  Sa druge strane, zbog gore navedenog problema, nije pametno ni u svakom koraku uzimati moduo. Ovo možete izbeći nekada ispitivanjem uslova:

```
sol = 0;
for i = 1 to n do
     sol = f (sol);
     sol = sol mod m;
```

Sporija varijanta

```
sol = 0;
for i = 1 to n do
     sol = f (sol);
     if (sol >= m)
         sol = sol mod m;
```

Brža varijanta

  Ukoliko imate neke dodatne informacije o tome kako se može menjati vrednost promenjive u toku iteracije, moduo možete i "ručno" računati. Primera radi ukoliko znate da se u svakoj iteraciji vrednost ne može povećati za više od 2 puta, to možete implementirati ovako:

```
 sol = 0;
for i = 1 to n do
     sol = f (sol);
     sol = sol mod m;
```

Sporija varijanta

```
sol = 0;
for i = 1 to n do
     sol = f (sol);
     if (sol >= m)
         sol = sol - m;
```

Brža varijanta

- ***Izraz po modulu***: Nekada se i pri samom računanju izraza može doći do prekoračenja, dok po modulu to naravno nije slučaj. Razmotrimo primer: $ab \bmod m$. Pri izvršavanju ovog izraza prvo se računa vrednost proizvoda a tek onda ostatak pri deljunju sa m. Ukoliko su promenjive a i b

[2] Za više informacija o ovome i kako se ostatak računa možete pogledati bilo koju knjigu iz teorije brojeva.

jako velike, njihov proizvod može dovesti do prekoračenja tj. kao rezultat bi dobili neki "čudan" broj. Ukoliko niste sigurni da li možete dobiti prekoračenje, izraz možete računati kao:

$$(a \ mod \ m)(b \ mod \ m) \ mod \ m$$

- ***Moduo negativnih brojeva**:* Računanje modula negativnih brojeva može biti neočekivano na prvi pogled. Posmatrajmo to kroz primer ispitivanje da li je ceo broj n neparan. Ukoliko bi samo ispitali da li je $n \ mod \ 2 = 1$, dobili bi smo grešku za negativne vrednosti – tada je rezultat moduo operacije jednak -1.

Gore opisane probleme, kao i sve ostale u ovom tekstu, bi trebalo isprobati u toku priprema. Na takmičenjima ovakvi sitni problemi mogu koštati dosta vremena. Zato je bolje biti pripremljen barem za ove standardnije zamke.

Ispitivanje jednakosti dva realna broja

Pri manipulisanju sa realnim vrednostima, jako često se mogu dobiti neočekivani rezultati. Rad sa njima je po prirodi netačan i kao takav može proizvesti rezltat sastavljen samo od "šuma". Jedan od glavnih problema numeričke matematike je upravo određivanje koliko je rezultat nekog metoda korektan – koliko odstupa od prave teoretske vrednosti. Treba imati na umu da se brojevi pamte u binarnom sistemu sa određenom preciznošću.

Jako često se u problemu traži ispitivanje jednakosti dva realna broja. Kako se preciznost menja sa porastom vrednosti realnih brojeva (`double` i `float`), najbolji način je upoređivati ih sa pragom greške koja će zavisiti od samih vrednosti koje upoređujemo.

```
if (Abs(A - B) < delta * Abs(A))
   return true;
else
   return false;
```

Korektna varijanta

```
if (A == B)
   return true;
else
   return false;
```

Pogrešna varijanta

U gornjem kodu eps je vrednost izabrana tako da označava stepen "sličnosti". Naravno treba voditi računa da vrednost promenjive A nije 0. Takođe i samu vrednost $delta$ treba pažljivo birati, shodno traženoj preciznosti u samom problemu.

Varijacija gornjih primera može biti i uslov:

```
if (Abs(A - B) < delta)
```

Međutim ova varijanta takođe može dovesti do određenih problema. Ako se upoređivanje vrednosti menjaju, granicu greške je možda potrebno smanjiti ako se mali brojevi koji bi trebalo biti različiti približavaju zbog fiksirane preciznosti. Sličan slučaj možemo imati i sa velikim brojevima.

Možda najbolji pristup ovom problemu bi bio preko relativne razlike:

```
double RelDif(double a, double b)
{
   double a1 = Abs(a);
   double b1 = Abs(b);
   double max = Max(a1, b1);

   if (d == 0.0)
      return 0.0;
   return Abs(a - b) / max;
}

if (RelDif(A, B) <= delta))            // USLOV
```

Pri upoređivanju dve realne vrednosti nikako nemojte koristiti jednakost. Ovo je jako frekventna greška takmičara. U većini slučajeva

```
if (Abs(A - B) < delta)
```

će zadovoljiti uslove problema. Ukoliko je potrebno, gore navedene komplikovanije varijante će vam pomoći.

"Time Limit Exceeded" ne mora da znači "Time Limit Exceeded"

Pri testiranju programa, bilo od strane sistema za ocenjivanje ili takmičara, nekada se može doneti zaključak da je algoritam ili implementacija prespora za ograničenja problema. Međutim, ukoliko se samo pokuša optimizacija pristupa, a ne i provera memorijskih zahteva i ograničenja, trud neće uroditi plodom. Naredni primer jako lepo ilustruje ovaj problem. Posmatrajmo naradni C kod:

```
#include <stdio.h>
#define MAXN 100

int array[MAXN], i;

void main( void )
{
        for (i = 0; i <= 100; i++)
        {
                array[i] = 0;
        }
}
```

U gore navedenom kodu program će pokušati da pristupi elementu niza sa indeksom 100. Kako je to izvan granica niza (u *C-u* su indeksi elemenata numerisani od 0), pristupiće se elementu na adresi nakon niza a, što predstavlja adresu promenjive i (pošto je ona definisana nakon niza). Tada postavljamo promenjivu i odnosno element $array[i]$ na 0. Na ovaj način smo dobili jako čudnu beskonačnu petnju.

Literatura

[1] Shahriar Monzoor, *Common Mistakes in Online and RealTime Contests*, XRDS Crossroads – The ACM Magazine for Students, Summer 2008/Vol. 14, No. 4

[2] Donald E. Knuth, *The Art of Computer Programming*, Addison Wesley, Volume 2, 1980