Kako je propalo Rimsko carstvo

Da bismo shvatili kako je propalo rimsko carstvo, moramo se prvo upoznati sa rimskim načinom zapisivanja brojeva. Stari Rimljani su koristili dosta čudan sistem, koji je bio pozicioni ali ne i težinski. Korišteno je 7 cifara M, D, C, L, X, V i I čije su vrijednosti redom 1000, 500, 100, 50, 10, 5 i 1. Samo računanje vrijednosti broja (pod uvjetom da je on ispravan je krajnje jednostavan). Vrijednost svake cifre se prosto sabira, osim ako se cifra sa manjom vrijednošću ne nalazi ispred cifre sa većom vrijednošću. U tom slučaju se cifra sa manjom vrijednošću odbija od cifre sa većom vrijednošću umjesto da se sabira. Uz ovo pravilo, posve je jednostavno napraviti algoritamsku funkciju za pretvaranje vrijednosti rimski zapisanog broja u klasičnu arapsku notaciju. Vrednovanje svake cifre izvešćemo pretragom odgovarajućeg niza, da bismo izbjegli višestruka testiranja (gomulu "**if**" naredbi). Sabiraćemo sve cifre slijeva nadesno. Međutim, primijetimo li da je tekuća cifra veće vrijednosti od prethodne cifre, vrijednost prethodne cifre ćemo odbiti dvaput od sume (s obzirom da smo je u prethodnom koraku dodali na sumu, a trebali smo je oduzeti od sume – odatle dvostruko odbijanje). Ovakvom "korekcijom unazad" izbjegava se potreba za "gledanjem unaprijed", što pojednostavljuje kôd. Sve u svemu, funkcija koja vrši pretvorbu rimskih brojeva u arapske bez ikakvog testiranja ispravnosti mogla bi izgledati recimo ovako:

```
int RimskiUArapskiBezTestiranja(const string &rimski) {
  const char cifre[] = "MDCLXVI";
  int vrijednosti[] = {1000, 500, 100, 50, 10, 5, 1};
  int suma(0), prethodna(0), vrijednost;
  for(int i = 0; i < rimski.length(); i++) {
    for(int j = 0; j < 7; j++)
      if(rimski[i] == cifre[j]) vrijednost = vrijednosti[j];
    suma += vrijednost;
    if(vrijednost > prethodna) suma -= 2 * prethodna;
    prethodna = vrijednost;
  }
  return suma;
}
```

Neka trivijalna testiranja ispravnosti moguće je odmah dodati. Recimo, ukoliko se ne prepozna ispravna cifra, broj je sigurno neispravan. Isto tako, on je sigurno neispravan ukoliko se kao rezultat konverzije dobije broj veći od 3999 (najveći broj koji klasična rimska notacija dozvoljava). Ugradimo li ove provjere, dolazimo do sljedeće funkcije:

```
int RimskiUArapskiSaTrivijalnomProvjerom(const string &rimski) {
  const char cifre[] = "MDCLXVI";
  int vrijednosti[] = {1000, 500, 100, 50, 10, 5, 1};
  int suma(0), prethodna(0);
  for(int i = 0; i < rimski.length(); i++) {
    int vrijednost(0);
    for(int j = 0; j < 7; j++)
      if(rimski[i] == cifre[j]) vrijednost = vrijednosti[j];
    if(vrijednost == 0) throw "Greška";   // cifra nije nađena
    suma += vrijednost;
    if(vrijednost > prethodna) suma -= 2 * prethodna;
    prethodna = vrijednost;
  }
  if(suma > 3999) throw "Greška";
  return suma;
}
```

Eh, sada ta provjera ispravnosti. Prije nego što se upustimo u taj problem, razmotrimo kako se uopće neki broj zapisuje u rimskoj notaciji. U principu, trebali bismo poći od veće ka manjoj težini, tj. prvo da vidimo koliko puta se broj 1000 javlja u datom broju, zatim koliko se puta u onome što ostane javlja broj 500, zatim koliko puta se u onome što ostane javlja broj 100, itd. Drugim riječima, razmatrani broj

trebalo bi razložiti u oblik $c_1 \cdot 1000 + c_2 \cdot 500 + c_3 \cdot 100 + c_4 \cdot 50 + c_5 \cdot 10 + c_6 \cdot 5 + c_7$ gdje se koeficijenti c_i računaju slijeva nadesno i to tako da se uvijek uzme najveća moguća vrijednost a da ono što preostane bude pozitivno. Ovo razlaganje nije teško izvesti u petlji računanjem cjelobrojnih količnika i ostataka pri dijeljenju redom sa vrijednostima 1000, 500 itd. (primijetimo još da koeficijenti c_2, c_4 i c_6 mogu biti samo 0 ili 1). Sada, da nije jedne "zvrčke", bilo bi dovoljno uzeti c_1 puta cifru "M", c_2 puta cifru "D", c_3 puta cifru "C" itd. A zvrčka je u činjenici da nastaju izuzeci ukoliko je od koeficijenata c_i jednak 4, s obzirom da se nikada ista cifra ne javlja 4 puta zaredom. Na primjer, $470 = 4 \cdot 100 + 1 \cdot 50 + 2 \cdot 10$, ali korektan rimski zapis ovog broja nije "CCCCLXX" nego "CDLXX". Dakle, umjesto da c_i puta ponavljamo odgovarajuću cifru, izgleda da kada je $c_i = 4$ treba uzeti jedanput tu cifru i dodati iza nje prvu sljedeću cifru po vrijednosti (recimo "CD" umjesto "CCCC"). Međutim, nije uvijek tako. Na primjer, imamo rastavu $397 = 3 \cdot 100 + 1 \cdot 50 + 4 \cdot 10 + 1 \cdot 5 + 2 \cdot 1$. Prema ovome što smo maločas zaključili, ispada da bi rimski zapis ovog broja trebao biti "CCCLXLVII". Ipak, to nije ispravan zapis (a pogotovo to nije zapis "CCCLXXXXVII"), nego je ispravan zapis ovog broja "CCCXCVII". Dakle, cifra "L" je potpuno nestala, i zamijenjena je grupom "XC". Nije teško vidjeti da ova situacija nastupa samo ako je posljednja razmatrana cifra koju smo dodali bila "V", "L" ili "D", što zapravo nastupa samo ukoliko je $c_{i-1} \neq 0$. Zaista, kod težina 5, 50 i 500 odgovarajući koeficijent može biti samo 0 ili 1, a s obzirom da situacija $c_i = 4$ može nastupiti samo ako je trenutno razmatrana težina 1, 10, 100 ili 1000, koeficijent c_{i-1} tada je ili 1 ili 0. Dakle, u tom slučaju posljednju cifru koju smo dodali ("L" u ovom primjeru) treba zamijeniti trenutno razmatranom cifrom ("X" u ovom primjeru) iza koje slijedi cifra 10 puta veće težine ("C" u ovom primjeru). Ovim smo kompletirali algoritam za pretvorbu arapskih brojeva u rimske brojeve, koji bi se mogao izvesti funkcijom poput sljedeće:

```
string ArapskiURimski(int broj) {
  const char cifre[] = "MDCLXVI";
  int vrijednosti[] = {1000, 500, 100, 50, 10, 5, 1};
  string rimski;
  int prethodni(0);
  for(int i = 0; i < 7; i++) {
    int kolicnik(broj / vrijednosti[i]);
    broj = broj % vrijednosti[i];
    if(kolicnik < 4)
      for(int j = 0; j < kolicnik; j++) rimski += cifre[i];
    else
      if(prethodni == 0) rimski = rimski + cifre[i] + cifre[i - 1];
      else {
        rimski[rimski.length() - 1] = cifre[i]; rimski += cifre[i - 2];
      }
     prethodni = kolicnik;
  }
  return rimski;
}
```

Ovim smo razvili algoritam za pretvorbu arapski zapisanih brojeva u rimski zapis. Međutim, kakve sve to veze ima sa testom ispravnosti broja zapisanog u rimskom zapisu? Poenta je u tome da je rimski zapis broja *jedinstven*, odnosno niti jedan broj se ne može zapisati u rimskoj notaciji na više različitih načina. E sada, pretpostavimo da smo pokušali pretvoriti rimski zapisani broj "XMVII" (nekorektno zapisan) u arapsku notaciju. Algoritam koji smo razvili daće kao rezultat pretvorbe broj 997. Međutim, pretvorimo li ovaj broj nazad u rimsku notaciju (pomoću drugog razvijenog algoritma), dobićemo broj "CMXCVII", što je ispravan zapis. Međutim, ovaj zapis se razlikuje od početnog zapisa "XMVII". Pošto je rimski zapis broja jedinstven, to je siguran znak da je početni rimski zapisani broj "XMVII" nekorektno zapisan. Dakle ideja je jasna. Nakon što izvršimo pretvorbu broja iz rimskog u arapski zapis, obavimo i obrnutu pretvorbu. Ukoliko se nakon obrnute pretvorbe dobije isti zapis od kojeg smo krenuli, broj je korektno zapisan, u suprotnom nije. Dakle, jedan način da napravimo korektnu verziju funkcije za pretvorbu iz rimskog u arapski zapis sa detekcijom ispravnosti je da iz nje pozovemo i drugu funkciju za obrnutu pretvorbu. Međutim, kako ove dvije funkcije imaju zajedničkih detalja, moguće ih je sastaviti u jedinstvenu funkciju koja obavlja pretvorbu iz rimskog u arapski zapis sa detekcijom ispravnosti. Takva funkcija mogla bi izgledati recimo ovako:

```
int RimskiUArapski(const string &rimski) {
  const char cifre[] = "MDCLXVI";
  int vrijednosti[] = {1000, 500, 100, 50, 10, 5, 1};
  int suma(0), prethodna(0);
  for(int i = 0; i < rimski.length(); i++) {
    int vrijednost(0);
    for(int j = 0; j < 7; j++)
      if(rimski[i] == cifre[j]) vrijednost = vrijednosti[j];
    if(vrijednost == 0) throw "Greška";
    suma += vrijednost;
    if(vrijednost > prethodna) suma -= 2 * prethodna;
    prethodna = vrijednost;
  }
  if(suma > 3999) throw "Greška";
  string nazad;
  int broj(suma), prethodni(0);
  for(int i = 0; i < 7; i++) {
    int kolicnik(broj / vrijednosti[i]);
    broj = broj % vrijednosti[i];
    if(kolicnik < 4)
      for(int j = 0; j < kolicnik; j++) nazad += cifre[i];
    else
      if(prethodni == 0) nazad = nazad + cifre[i] + cifre[i - 1];
      else {
        nazad[nazad.length() - 1] = cifre[i]; nazad += cifre[i - 2];
      }
     prethodni = kolicnik;
  }
  if(nazad != rimski) throw "Greška";
  return suma;
}
```

U principu, ovim je problem pretvorbe rimskih brojeva u arapske uz testiranje ispravnosti riješen. Zapravo, pravila za ispravnost indirektno su kodirana u pravilima za pretvaranje arapskih brojeva u rimski zapis. Prikazano rješenje je vrlo univerzalno i izuzetno ga je lako proširiti za slučaj da se uvedu recimo nove rimske cifre koje predstavljaju vrijednosti 5000, 10000 itd. Glavnina funkcije ostaje potpuno ista, samo je potrebno proširiti nizove "`cifre`" i "`vrijednosti`". Međutim, definitivno vrijedi postaviti pitanje može li se problem riješiti kraće, pa čak i jednostavnije. Ispostavlja se da može. Na prvom mjestu, primijetimo da je algoritam za pretvaranje arapskih u rimske brojeve relativno kompleksan. Ako se držimo klasičnih rimskih brojeva (tj. ako podržavamo samo cifre koje su klasično podržane), pokazuje se da je pametnije pripremiti tabelu uzoraka kako izgledaju rimski zapisi za arapske cifre od 0 – 9 i to posebno za slučaj kada se cifra nalazi na poziciji jedinica, desetica, stotica i hiljadica. Tada se pretvorba obavlja prosto razlaganjem broja na jedinice, desetice, stotice i hiljadice (što se lako radi pomoću cjelobrojnog dijeljenja i ostataka pri dijeljenju sa 10) i čitanjem odgovarajućih uzoraka iz tabele. To nas vodi do funkcije poput sljedeće:

```
string ArapskiURimskiVerzija2(int broj) {
  string rimski, uzorci[4][10] =
    {{"", "M", "MM", "MMM", "", "", "", "", "", ""},
     {"", "C", "CC", "CCC", "CD", "D", "DC", "DCC", "DCCC", "CM"},
     {"", "X", "XX", "XXX", "XL", "L", "LX", "LXX", "LXXX", "LC"},
     {"", "I", "II", "III", "IV", "V", "VI", "VII", "VIII", "IX"}};
  int broj(suma), red(3);
  while(broj !=0) {
    rimski = uzorci[red][broj % 10] + rimski; broj /= 10; red--;
  }
  return rimski;
}
```

Ova skraćena verzija funkcije za pretvorbu iz arapskog u rimski zapis može se iskoristiti i da se skrati funkcija za obrnutu pretvorbu (uz testiranje ispravnosti), koja sada može izgledati recimo ovako:

```
int RimskiUArapskiVerzija2(const string &rimski) {
  const char cifre[] = "MDCLXVI";
  int vrijednosti[] = {1000, 500, 100, 50, 10, 5, 1};
  int suma(0), prethodna(0);
  for(int i = 0; i < rimski.length(); i++) {
    int vrijednost(0);
    for(int j = 0; j < 7; j++)
      if(rimski[i] == cifre[j]) vrijednost = vrijednosti[j];
    if(vrijednost == 0) throw "Greška";
    suma += vrijednost;
    if(vrijednost > prethodna) suma -= 2 * prethodna;
    prethodna = vrijednost;
  }
  if(suma > 3999) throw "Greška";
  string nazad, uzorci[4][10] =
    {{"", "M", "MM", "MMM", "", "", "", "", "", ""},
     {"", "C", "CC", "CCC", "CD", "D", "DC", "DCC", "DCCC", "CM"},
     {"", "X", "XX", "XXX", "XL", "L", "LX", "LXX", "LXXX", "LC"},
     {"", "I", "II", "III", "IV", "V", "VI", "VII", "VIII", "IX"}};
  int broj(suma), red(3);
  while(broj !=0) {
    nazad = uzorci[red][broj % 10] + nazad; broj /= 10; red--;
  }
  if(nazad != rimski) throw ne_valja;
  return suma;
}
```

Može li bolje od ovoga? Ključna stvar je uočiti da se sistem prepoznavanja "karakterističnih uzoraka" mogao primijeniti i direktno na pretvorbu rimski zapisanih brojeva u arapski zapis. Zaista, posmatrajmo recimo broj "CMLXXIX". Ako prepoznamo da skupina "CM" predstavlja broj 900, skupina "LXX" predstavlja broj 70 i skupina "IX" predstavlja broj 9, lako zaključujemo da se radi o broju 979. Jedino je pitanje kojim redom analizirati skupine. Prva stvar koju možemo uočiti da treba prvo razmatrati uzorke koji odgovaraju hiljadicama, zatim stoticama, te deseticama i na kraju jedinicama, s obzirom da recimo broj "VIICM" nije legalan (uzorak koji "VII" odgovara jedinicama ispred je uzorka "CM" koji odgovara hiljadicama), a "CMVII" jeste. Druga stvar, nakon što prepoznamo jedan uzorak iz iste skupine, sa razmatranjem te skupine treba završiti, jer nije legalno imati dva uzorka iz iste skupine (npr. broj "LXXLXVII" nije legalan, jer sadrži dva uzorka "LXX" i "LX" iz grupe desetica). Treća stvar, ne smije se desiti da uzorak koji je prefiks nekog drugog uzorka testiramo prije tog drugog uzorka, jer u suprotnom taj drugi uzorak nećemo nikad pronaći (npr. ukoliko u broju "DCCVI" odmah prepoznamo uzorak "D", uzorak "DC" ili "DCC" nikada nećemo naći, ako se držimo pravila da odmah moramo preći na razmatranje sljedeće skupine). Skupimo li sve ove ideje zajedno, dolazimo do zaista jednostavne i kratke funkcije koja izgleda recimo ovako:

```
int RimskiUArapskiVerzija3(const string &rimski) {
  int suma(0), i(0), tezina(1000);
  string uzorci[4][9] =
    {{"\n", "\n", "\n", "\n", "\n", "\n", "MMM", "MM", "M"},
     {"CM", "DCCC", "DCC", "DC", "D", "CD", "CCC", "CC", "C"},
     {"LC", "LXXX", "LXX", "LX", "L", "XL", "XXX", "XX", "X"},
     {"IX", "VIII", "VII", "VI", "V", "IV", "III", "II", "I"}};
  for(int k = 0; k < 4; k++) {
    for(int j = 0; j < 9; j++)
      if(rimski.substr(i, uzorci[k][j].length()) == uzorci[k][j]) {
        suma += (9 - j) * tezina;
        i += uzorci[k][j].length(); break;
      }
    tezina /= 10;
  }
  if(suma == 0 || i != rimski.length()) throw "Greška";
  return suma;
}
```

Uzorci "\n" zapravo predstavljaju bilo šta što se sigurno *ne može* naći u ulaznom stringu (a oznaka kraja reda sigurno se ne može pojaviti kao sastavni dio normalno unesenog stringa. Ukoliko dopustimo upotrebu funkcije "pow", funkcija se može i neznatno skratiti, jer otpada potreba za upotrebom promjenljive "tezina" i njenim ažuriranjem. Jedino moramo voditi računa da je ova funkcija namijenjena za rad pretežno sa realnim brojevima, tako da očekuje barem jedan parametar realnog tipa i vraća rezultat realnog tipa. Sve u svemu, ova neznatno skraćena verzija mogla bi izgledati ovako:

```
int RimskiUArapskiVerzija4(const string &rimski) {
  int suma(0), i(0);
  string uzorci[4][9] =
    {{"\n", "\n", "\n", "\n", "\n", "\n", "MMM", "MM", "M"},
     {"CM", "DCCC", "DCC", "DC", "D", "CD", "CCC", "CC", "C"},
     {"LC", "LXXX", "LXX", "LX", "L", "XL", "XXX", "XX", "X"},
     {"IX", "VIII", "VII", "VI", "V", "IV", "III", "II", "I"}};
  for(int k = 0; k < 4; k++)
    for(int j = 0; j < 9; j++)
      if(rimski.substr(i, uzorci[k][j].length()) == uzorci[k][j]) {
        suma += (9 - j) * int(pow(10., 3 - k));
        i += uzorci[k][j].length(); break;
      }
  if(suma == 0 || i != rimski.length()) throw "Greška";
  return suma;
}
```

Gornja verzija funkcije je možda posljednja "razumljiva" verzija funkcije za pretvorbu rimski zapisanih brojeva u arapske brojeve. Postavlja se još tu nekoliko mogućnosti za skraćivanje. Prvo je pitanje moraju li se zaista uzorci čuvati u matrici, ili može poslužiti i običan niz, odnosno da li je kroz uzorke neophodno prolaziti sa dvije petlje ili je dovoljna samo jedna. Dalje, da li je zaista potrebno čuvati i "lažne uzorke" poput "\n" ili se može proći i bez njih. Razmišljanja na tu temu dovela su do daljih optimizacija, koje su reducirale dvodimenzionalni niz uzoraka na jednodimenzionalni, dvije petlje na jednu i izbacivanje lažnih uzoraka, po cijenu malo komplikovanijih "igrarija" sa indeksom petlje koje spadaju u domen elementarne matematike (uglavnom modularne aritmetike, tj. igranja sa ostacima pri dijeljenju). Slijedi jedna takva verzija koju je zaista teško dalje optimizirati (pitanje je da li je to uopće moguće).

```
int RimskiUArapskiVerzija5(const string &rimski) {
  int suma(0), i(0);
  string uzorci[30] = {"MMM", "MM", "M", "CM", "DCCC", "DCC", "DC",
    "D", "CD", "CCC", "CC", "C", "LC", "LXXX", "LXX", "LX", "L", "XL",
    "XXX", "XX", "X", "IX", "VIII", "VII", "VI", "V", "IV", "III",
    "II", "I"};
  for(int j = 0; j < 30; j++)
    if(rimski.substr(i, uzorci[j].length()) == uzorci[j]) {
      suma += (9 – (j + 6) % 9) * int(pow(10., 3 – (j + 6) / 9));
      i += uzorci[j].length(); j = 2 + 9 * ((j + 6) / 9);
    }
  if(suma == 0 || i != rimski.length()) throw "Greska";
  return suma;
}
```

Ovim je priča o pretvorbi rimskih u arapske brojeve završena. Međutim, kakve ovo veze ima sa propasti Rimskog carstva? Eeee, nekada davno, administrativne službe u Rimskom carstvu zapale su u ozbiljne tehničke probleme, za koje su zaključili da neće biti rješive bez pomoći računara. Stoga su angažirani najbolji programeri Rimskog carstva da razviju prikladan softver koji bi olakšao vođenje administracije. Nevolja je u tome što su tadašnji programeri pisali programe čiji je stil upadljivo podsjećao na stil programa koji su studenti I godine generacije 2010/11 pisali kada im je problem pretvorme rimskih u arapske brojeve postavljen za zadaću. A poznato je šta se nakon toga desilo...