

## Randomizovani algoritmi

Algoritmi koje smo do sada razmatrali bili su *deterministički* — svaki naredni korak bio je unapred određen. Kad se deterministički algoritam izvršava dva puta sa istim ulazom, način izvršavanja je oba puta isti, kao i dobijeni izlazi. Za razliku od njih *randomizovani algoritmi* (eng. randomized algorithms) sadrže korake koji zavise ne samo od ulaza, nego i od nekih slučajnih događaja. Naime, randomizovani algoritam koristi slučajne brojeve koji utiču na izbore koji se prave tokom njegovog izvršavanja. Stoga njegovo ponašanje, koje se uobičajeno meri kroz njegovo vreme izvršavanja ili kvalitet izlaza, za fiksiranu vrednost ulaza varira od jednog do drugog izvršavanja.

Koreni randomizovanih algoritama leže u Monte Karlo metodi<sup>1</sup> koja se koristi u numeričkoj analizi. Danas randomizovani algoritmi nalaze sve veći broj primena u raznim oblastima. Naime, randomizacija u mnogim slučajevima omogućava konstrukciju jednostavnijih algoritama, kao i algoritama koji su efikasniji od najboljih poznatih determinističkih algoritama.

Postoji veliki broj varijacija randomizovanih algoritama. Mi ćemo u ovom materijalu razmotriti dve vrste njih:

- *Las Vegas algoritme* koji uvek daju tačan rezultat, ali im vreme izvršavanja nije garantovano i
- *Monte Karlo algoritme* koji mogu dati netačan rezultat sa određenom (tipično jako malom) verovatnoćom, ali se uvek zaustavljaju u konačnom vremenu i vreme izvršavanja im je bolje nego za najbolji poznati deterministički algoritam.

Primetimo da se algoritmi tipa Las Vegas mogu izvršavati brzo, ali se mogu izvršavati i proizvoljno dugo. Oni su korisni samo ako im je *očekivano* vreme izvršavanja malo.

Randomizovani algoritmi su posebno korisni u situacijama kada se “suočavamo” sa protivnikom koji pokušava da prosledi loš ulaz u algoritam, kao što je recimo slučaj sa primenama u kriptografiji. To je jedan od razloga što je slučajnost sveprisutna u oblasti kriptografije. Na primer, pseudoslučajni brojevi nisu poželjni u primenama u kriptografiji jer ih napadač može predvideti, već se koriste kriptografski sigurni pseudoslučajni brojevi.

---

<sup>1</sup>Monte Karlo metode predstavljaju klasu računarskih algoritama koje koriste slučajno uzorkovanje za aproksimaciju rešenja matematičkih i statističkih problema. One uključuju višestruko generisanje nasumičnih ulaznih podataka, simulaciju sistema i analizu prikupljenih podataka kako bi se procenila verovatnoća, očekivanje ili neka druga statistička mera.

## Određivanje elementa binarnog niza sa vrednošću 1

Razmotrimo jedan jednostavan problem. Dat je neuređen niz  $a$  od  $n \geq 2$  celobrojnih vrednosti u kome polovina elemenata ima vrednost 0, a polovina elemenata vrednost 1. Potrebno je pronaći element niza koji ima vrednost 0. Deterministički algoritam za rešavanje ovog problema bi prošao redom kroz elemente niza i nakon najviše  $n/2 + 1$  koraka našao element niza čija je vrednost jednaka 0. Dakle, njegova vremenska složenost bila bi u najgorem slučaju  $O(n)$ .

Razmotrimo kako bi izgledao randomizovani algoritam za rešavanje ovog problema. Algoritam tipa Las Vegas bi *na slučajan način* birao proizvoljni element niza  $a$  i proveravao da li je njegova vrednost jednaka 0 i ove korake ponavljao sve dok ne izaberemo element sa vrednošću 0. Razmotrimo kakve su performanse ovog algoritma. On sa verovatnoćom 1 vraća ispravan rezultat, ali broj iteracija algoritma varira i može biti proizvoljno veliki.

Analizirajmo koliko je očekivano vreme izvršavanja algoritma. Svaki nasumično odabrani element sa verovatnoćom  $1/2$  ima vrednost 0, a sa verovatnoćom  $1/2$  vrednost 1. Verovatnoća da je u  $i$ -toj iteraciji *po prvi put* pronađen element sa vrednošću 0 je  $(1/2)^i$ . Stoga je očekivani broj iteracija algoritma jednak:

$$\lim_{n \rightarrow \infty} \sum_{i=0}^n \frac{i}{2^i}$$

Ako gornju sumu označimo sa  $S$ , važi:

$$S = \frac{1}{2} + \frac{2}{4} + \frac{3}{8} + \frac{4}{16} + \dots$$

Odavde sledi:

$$\frac{S}{2} = \frac{1}{4} + \frac{2}{8} + \frac{3}{16} + \frac{4}{32} + \dots$$

Oduzimanjem ove dve jednakosti dobijamo:

$$\frac{S}{2} = \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \frac{1}{16} + \dots = \sum_{i=0}^{\infty} \left(\frac{1}{2}\right)^i - 1 = \frac{1}{1 - 1/2} - 1 = 2 - 1 = 1$$

odakle sledi da je  $S = 2$ . Dakle, očekivani broj iteracija u kojem bi bio pronađen element sa vrednošću 0 je 2. Stoga je očekivana vremenska složenost ovog Las Vegas algoritma  $O(1)$ , dok nemamo nikakvih garancija za vremensku složenost algoritma u najgorem slučaju.

Razmotrimo kako bi izgledao Monte Karlo algoritam za rešavanje ovog problema. On bi na slučajan način birao element niza  $a$  i proveravao da li je vrednost elementa 0. Ukoliko to nije slučaj iznova bi ponavljao ovaj postupak, ali maksimalno  $m$  puta. Primetimo da u ovom slučaju ne postoji garancija da će algoritam vratiti korektan rezultat. Naime, može se desiti da se u svih  $m$  pokušaja izabere element sa vrednošću 1. Primetimo i da je verovatnoća da se nakon  $k$  iteracija pronađe traženi element jednaka  $1 - (\frac{1}{2})^k$ , što je vrednost bliska 1 za dovoljno veliko  $k$ . Vremenska složenost ovog algoritma je ograničena maksimalnim dopuštenim brojem iteracija  $m$ , te iznosi  $O(1)$  u najgorem slučaju, ali uspešnost algoritma nije garantovana.

Primetimo da se Las Vegas algoritam često jednostavno može pretvoriti u Monte Karlo algoritam tako što ćemo ga pustiti da radi neko fiksirano vreme i vratiti proizvoljni odgovor ako se za to vreme ne zaustavi.

Ako je raspoloživ deterministički algoritam kojim se može ispitati korektnost izlaza koji je randomizovani algoritam vratio, onda je moguće transformisati Monte Karlo algoritam u Las Vegas algoritam. Naime, pozvaćemo odgovarajući Monte Karlo algoritam i nakon toga proveriti da li je vrednost koju je algoritam vratio ispravna i ukoliko nije, ponovo pozvati Monte Karlo algoritam, sve dok ne vrati ispravno rešenje. Ako je vreme izvršavanja Monte Karlo algoritma  $T(n)$  za ulaz veličine  $n$  i ako vraća ispravan rezultat sa verovatnoćom  $p$ , a vreme potrebno za proveru da li je algoritam vratio korektan izlaz  $t(n)$ , onda je očekivano vreme izvršavanja odgovarajućeg Las Vegas algoritma  $(T(n) + t(n))/p$ .

## Određivanje broja iz gornje polovine

Pretpostavimo da je dat skup od  $n$  ne nužno različitih brojeva  $x_0, x_1, \dots, x_{n-1}$  i da među njima treba izabrati neki broj iz gornje polovine, odnosno broj koji je veći ili jednak od bar  $n/2$  ostalih. Na primer, potrebno je izabrati dobrog studenta, pri čemu je kriterijum prosečna ocena.

Razmotrimo kako bi izgledao deterministički algoritam za rešavanje ovog problema. Jedna mogućnost je uzeti najveći broj (koji je uvek u gornjoj polovini). Već smo videli da je za određivanje maksimuma potrebno  $n - 1$  upoređivanja. Druga mogućnost je započeti sa izvršavanjem algoritma za nalaženje maksimuma, i zaustaviti se kad se prođe polovina brojeva. Broj koji je veći ili jednak od jedne polovine brojeva je sigurno u gornjoj polovini. Algoritam zahteva oko  $n/2$  upoređivanja. Može li se ovaj posao obaviti efikasnije? Nije teško pokazati da je nemoguće garantovati da broj pripada gornjoj polovini ako je izvršeno manje od  $n/2$  upoređivanja. Prema tome, opisani algoritam je optimalan.

Ovaj algoritam je, međutim, optimalan samo ako insistiramo na garanciji da ćemo sigurno vratiti tačan rezultat. U mnogo slučajeva garancija nije neophodna, već je dovoljna pristojna verovatnoća da je rešenje tačno. Na primer, kod heš tabela nije moguće garantovati da do kolizija neće doći, ali je verovatnoća da do nji dođe razumno mala i postoji način za rešavanje problema izazvanih pojavom kolizija (operacije sa heš tabelama se takođe mogu smatrati randomizovanim

algoritmima). Ako odustanemo od garancije, onda postoji bolji algoritam za nalaženje elementa iz gornje polovine.

Izaberimo na slučajan način dva broja  $x_i$  i  $x_j$  iz skupa, tako da je  $i \neq j$ . Pretpostavimo da je  $x_i \geq x_j$ . Verovatnoća da slučajno izabrani broj iz skupa pripada gornjoj polovini je bar  $1/2$  (ona će biti veća od  $1/2$  ako je više brojeva jednako medijani). Verovatnoća da ni jedan od brojeva  $x_i, x_j$  ne pripada gornjoj polovini je najviše  $1/4$ . Zbog  $x_i \geq x_j$  ova verovatnoća jednaka je verovatnoći da  $x_i$  kao veći od ova dva broja ne pripada gornjoj polovini. Prema tome, verovatnoća da  $x_i$  pripada gornjoj polovini je bar  $3/4$ .

Verovatnoća  $3/4$  da je dobijeni rezultat tačan obično nije dovoljna. Međutim, opisani pristup može se uopštiti. Na slučajan način biramo  $k$  brojeva iz skupa i određujemo najveći od njih. Na isti način kao u specijalnom slučaju, zaključujemo da najveći od  $k$  elemenata pripada gornjoj polovini sa verovatnoćom najmanje  $1 - (\frac{1}{2})^k$  (on ne pripada gornjoj polovini akko gornjoj polovini ne pripada ni jedan od izabranih brojeva, što je događaj sa verovatnoćom najviše  $(\frac{1}{2})^k$ ). Na primer, ako je  $k = 10$ , verovatnoća uspeha je približno 0.999, a za  $k = 20$  ta verovatnoća je oko 0.999999. Ako je pak  $k = 100$ , onda je verovatnoća greške za sve praktične svrhe zanemarljiva. Imamo dakle algoritam koji bira broj iz gornje polovine sa proizvoljno velikom verovatnoćom, a koji izvršava mali broj upoređivanja nezavisno od veličine ulaza. Ovo je primer Monte Karlo algoritma.

## Slučajni brojevi

Bitan element randomizovanih algoritama je generisanje slučajnih brojeva. Tradicionalni način za generisanje slučajnih brojeva se često realizuje bacanjem novčića ili kockice. Međutim, potrebno je imati efikasne računarske metode za rešavanje ovog problema.

Deterministička procedura generiše brojeve na fiksirani način, pa tako dobijeni brojevi ne mogu biti slučajni u pravom smislu te reči. Ti brojevi su međusobno povezani na sasvim određen način. Na sreću, to u praksi nije veliki problem: dovoljno je koristiti tzv. *pseudoslučajne brojeve*. Ti brojevi generišu se determinističkom procedurom (pa dakle nisu pravi slučajni brojevi), ali procedura generisanja je dovoljno složena, da je teško uočiti međuzavisnosti između tih brojeva.

Jedan od načina za dobijanje pseudoslučajnih brojeva je *linearni kongruentni metod*. Prvi korak je izbor slučajnog celog broja  $X_0$  kao prvog člana niza na neki nezavisan način (na primer, tekuće vreme u mikrosekundama). Ostali brojevi izračunavaju se na osnovu diferencne jednačine

$$X_n = (aX_{n-1} + b) \pmod{m},$$

gde su  $a, b$  i  $m$  konstante, koje je potrebno pažljivo izabrati. I pored pažljivog izbora, ovakvi nizovi često nisu dovoljno „slučajni“. Alternativa je korišćenje diferencnih jednačina višeg reda (sa zavisnošću od većeg broja prethodnih članova).

Na prethodno opisani način dobija se niz brojeva iz opsega od 0 do  $m - 1$ . Ako su potrebni slučajni brojevi iz opsega od 0 do 1, onda se članovi ovog niza mogu podeliti sa  $m$ .

## Provera matičnog množenja

Pretpostavimo da nam je na raspolaganju modul za množenje matrica i da želimo da proverimo da li on radi korektno za neki konkretan ulaz, tj. da za matrice  $A, B$  i  $C$  dimenzije  $n \times n$  proverimo da li važi  $AB = C$ .

Deterministički algoritam bi nad matricama  $A$  i  $B$  pokrenuo neki od standardnih algoritama za množenje matrica i nakon toga poredio svaki element matrice  $AB$  sa odgovarajućim elementom matrice  $C$ . Podsetimo se da je direktni algoritam za množenje matrica složenosti  $O(n^3)$ , dok je Štrasenov algoritam nešto bolje složenosti  $O(n^{\log_2 7})$ . Pokazuje se da je korišćenjem tehnike randomizacije moguće doći do odgovora u vremenskoj složenosti  $O(n^2)$  sa jako malom verovatnoćom pogrešnog odgovora.

Ideja na kojoj se zasniva randomizovani algoritam je da se na slučajan način izabere  $n$ -dimenzioni vektor, pomnože njime obe strane ove matične jednačine i izvrši provera da li je dobijeni rezultat isti. Najjednostavnije je za vektor  $r$  izabrati binarni vektor  $r = (r_0, r_1, \dots, r_{n-1}) \in \{0, 1\}^n$  na slučajan način. Ukoliko je  $(AB)r \neq Cr$ , onda kao odgovor vraćamo da su matrice  $AB$  i  $C$  različite, a inače vraćamo da su jednake.

Ovaj algoritam je vremenske složenosti  $O(n^2)$ , jer je množenje matrica asocijativno, pa se računanje  $(AB)r$  može izraziti kao  $A(Br)$  čime se algoritam svodi na samo tri množenja matrice vektorom ( $Br$ ,  $A(Br)$  i  $Cr$ ), što je ukupne vremenske složenosti  $O(n^2)$ . Jasno je da ako se dobije da važi  $(AB)r \neq Cr$ , onda sa sigurnošću zaključujemo da je  $AB \neq C$ . Međutim, ako su dobijeni vektori jednaki, postoji verovatnoća da su odgovarajuće matrice različite<sup>2</sup>. Ipak, ova verovatnoća nije velika. U nastavku ćemo pokazati da je za  $AB \neq C$  verovatnoća da se za slučajno odabran vektor  $r$  dobije da je  $(AB)r = Cr$  odozgo ograničena sa  $1/2$ :

$$P[A \cdot B \cdot r = C \cdot r] \leq \frac{1}{2}$$

Neka je  $D = C - AB \neq 0$ . Dokažimo da za slučajno odabrani vektor  $r$  važi  $P[Dr \neq 0] \geq \frac{1}{2}$ .

S obzirom na to da je  $D \neq 0$ , postoji  $0 \leq j < n$  tako da je  $j$ -ta kolona matrice  $D$  različita od nula kolone. Fiksirajmo tu vrednost  $j$ . Za proizvoljni vektor  $r$ , definišemo vektor  $r'$  kao vektor dobijen od  $r$  izmenom njegovog  $j$ -tog elementa: ako je on bio 0 postavljamo ga na 1, a ako je bio 1 na 0. Dakle važi  $r' = r + e_j$

<sup>2</sup>Na primer, ako izaberemo vektor  $r = (0, 0, \dots, 0)$ , proizvod proizvoljne matrice ovim vektorom daće isti rezultat.

ili  $r' = r - e_j$ , gde je  $e_j$   $j$ -ti vektor standardne baze koji ima jedinicu na mestu  $j$ -te koordinate, a sve ostale nule. Primetimo da važi  $(r')' = r$ .

Definišemo graf  $G$  čiji čvorovi odgovaraju  $n$ -torkama oblika  $\{0, 1\}^n$ , a grane postoje između čvorova  $r$  i  $r'$  za svako  $r$ . Primetimo da je stepen svakog čvora u grafu  $G$  jednak 1 i da skup grana grafa predstavlja savršeno uparivanje u grafu. Tvrđimo da ako su  $r$  i  $r'$  upareni u  $G$ , onda ne može istovremeno da važi  $Dr = 0$  i  $Dr' = 0$ , tj. mora da važi bar jedno od  $Dr \neq 0$  i  $Dr' \neq 0$ . Naime, ako je  $Dr = 0$ , onda je  $Dr' = D(r \pm e_j) = Dr \pm De_j = \pm De_j$ . Međutim, vektor  $De_j$  je baš  $j$ -ta kolona matrice  $D$  koja je po pretpostavci različita od 0. Dakle, za proizvoljni vektor  $r$  važi bar jedno od  $Dr \neq 0$  i  $Dr' \neq 0$ , te za bar polovinu vektora  $r$  važi  $Dz \neq 0$  i odavde sledi prethodno tvrđenje.

Da bi se smanjila verovatnoća da dođe do greške, opisani postupak moguće je ponoviti  $k$  puta za  $k$  slučajno izabranih vektora  $r$ . Na taj način se verovatnoća da dođe do greške smanjuje na  $(\frac{1}{2})^k$ .

## Metod verovatnoće

Ideja randomizovanih algoritama tesno je povezana sa izvođenjem dokaza. Korišćenje verovatnoće za dokazivanje kombinatornih tvrđenja je moćna tehnika. U osnovi, potrebno je dokazati postojanje kombinatornog objekta sa zadatim svojstvima. Nažalost, nekada ne deluje izvodljivo eksplicitno konstruisati takav objekat, a možda nam konkretan primer ni ne treba. Tada možemo razmotriti slučajno generisani objekat iz odgovarajućeg prostora i izračunati verovatnoću da on zadovoljava date uslove. Ako je ova verovatnoća strogo pozitivna, zaključujemo da takav objekat mora postojati (ukoliko nijedan takav objekat ne bi imao data svojstva, verovatnoća bi bila jednaka nuli). Ova ideja može se iskoristiti za konstrukciju randomizovanog algoritma.

Pretpostavimo da je zadatak konstruisati objekat sa nekim osobinama. Pritom znamo da ako generišemo slučajni objekat, on će zadovoljavati željeni uslov sa verovatnoćom većom od nule (što je probabilistički dokaz da traženi objekat postoji). Mi zatim pratimo probabilistički dokaz, generišući slučajne događaje kad je potrebno, i na kraju nalazimo željeni objekat se nekom pozitivnom verovatnoćom. Ovaj postupak može se ponavljati više puta, sve do uspešnog pronalazjenja željenog objekta.

Razmotrimo jedan ovakav problem.

## Bojenje elemenata skupa

Neka je  $S$  skup od  $n$  elemenata, i neka je  $S_1, S_2, \dots, S_k$  kolekcija njegovih različitih podskupova, od kojih svaki sadrži tačno  $r$  elemenata,  $r \geq 2$ , pri čemu je broj podskupova  $k \leq 2^{r-2}$ . Potrebno je obojiti svaki element skupa  $S$  jednom od dve boje, crvenom ili plavom, tako da svaki podskup  $S_i$  sadrži bar jedan plavi i bar jedan crveni element.

Primer ulaza za ovaj problem prikazan je u narednoj tabeli: skup  $S$  sadrži  $n = 8$  elemenata, razmatramo njegovih  $k = 3$  podskupa, a svaki od tih podskupova sadrži po  $r = 4$  elementa. Primitimo da važi uslov zadatka jer je  $k = 3 \leq 2^2 = 2^{4-2} = 2^{r-2}$ .

podskup/element	1	2	3	4	5	6	7	8
1		x	x		x	x		
2			x	x		x	x	
3	x			x		x	x	

Bojenje koje zadovoljava zadati uslov zvaćemo *ispravnim bojenjem*. Ispostavlja se da pod navedenim uslovima ispravno bojenje uvek postoji. U prethodnom primeru bilo bi dovoljno obojiti element 6 u plavo, elemente 5 i 7 u crveno, a ostale elemente kako god želimo.

Deterministički algoritam bi u najgorem slučaju morao da razmotri sva moguća bojenja datog skupa, kojih ima  $2^n$ . Za svako od njih bilo bi potrebno izvršiti proveru da li je ispravno, što bi zahtevalo prolazak kroz sve elemente svih podskupova.

Jednostavan randomizovani algoritam dobija se prepravkom probablističkog dokaza postojanja takvog bojenja:

*Obojiti svaki element skupa  $S$  slučajno izabranom bojom, plavom ili crvenom, nezavisno od bojenja ostalih elemenata.*

Jasno je da ovaj algoritam ne daje uvek ispravno bojenje.

Izračunajmo verovatnoću neuspeha. Verovatnoća da su svi elementi skupa  $S_i$  obojeni crveno je  $2^{-r}$ , a verovatnoća da su svi obojeni istom bojom (crvenom ili plavom) je  $2 \cdot 2^{-r} = 2^{1-r}$ . Označimo sa  $A$  slučajni događaj da je neki od skupova  $S_i$  neispravno obojen: on je unija svih slučajnih događaja  $A_i$  da je skup  $S_i$  neispravno obojen, za  $i = 1, 2, \dots, k$ . Važi nejednakost:

$$P(A) \leq \sum_{i=1}^k P(A_i) = \sum_{i=1}^k 2^{1-r} = k2^{1-r} \leq 2^{r-2}2^{1-r} = \frac{1}{2}.$$

S obzirom na to da je verovatnoća da je bojenje elemenata bilo uspešno jednaka  $p = 1 - P(A) \geq \frac{1}{2}$ , odavde sledi da ispravno bojenje postoji (u protivnom bi verovatnoća neispravnog bojenja bila tačno 1). Pored toga, vidi se da je ovaj randomizovani algoritam dobar. Ispravnost zadatog bojenja lako se proverava: proveravaju se elementi svakog podskupa dok se ne pronađu dva različito obojena elementa. Ako se u nekom pokušaju dobije neispravno bojenje, postupak bojenja se ponavlja. Očekivani broj pokušaja bojenja je manji ili jednak od dva. Zaista, verovatnoća da se ispravno bojenje pronađe u  $j$ -tom pokušaju je  $(1 - p)^{j-1}p$ , pa je matematičko očekivanje broja pokušaja jednako:

$$\begin{aligned}
\sum_{j=1}^{\infty} j \cdot (1-p)^{j-1} p &= \sum_{j=0}^{\infty} j \cdot (1-p)^{j-1} p = p \sum_{j=0}^{\infty} j(1-p)^{j-1} \\
&= -p \sum_{j=0}^{\infty} \left( (1-p)^j \right)'_p = -p \left( \sum_{j=0}^{\infty} (1-p)^j \right)'_p \\
&= -p \left( \frac{1}{p} \right)'_p = -p \left( -\frac{1}{p^2} \right) = \frac{1}{p} \leq 2
\end{aligned}$$

U ovom nizu jednakosti i nejednakosti koristimo činjenicu da je  $1-p < 1$ , kao i da je  $p \geq \frac{1}{2}$ .

Opisani algoritam bojenja je očigledno Las Vegas algoritam, jer se bojenja proveravaju jedno za drugim, a sa traženjem se završava kad se nađe na ispravno bojenje. Ne postoji garancija uspešnog bojenja u bilo kom fiksiranom broju pokušaja, ali je ovaj algoritam u praksi ipak vrlo efikasan.