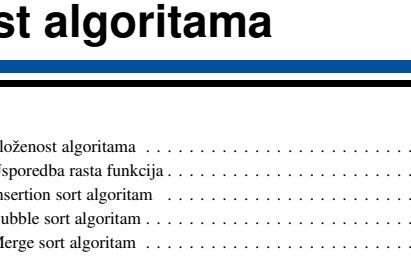
8.
Složenost algoritama

Složenost algoritama 2
Usporedba rasta funkcija 4
Insertion sort algoritam 6
Bubble sort algoritam 8
Merge sort algoritam 9
Quick sort algoritam 11
Podijeli pa vladaj algoritmi 13
Rastav broja u prirodne pribrojnike 18
Podskupovi zadanog skupa 20
Kombinacije iz zadanog skupa 24
Zadatci za vježbu 26



Sl. 8.1.

Složenost algoritma

Pri analizi algoritma realizacija kojeg ovisi o prirodnom broju n korisno je procjeniti njegovu složenost. Ta složenost može se ticati prostora koji podatci mogu zauzimati, njihove veličine te vremena (odnosno broja operacija) potrebnog za realizaciju algoritma.

U ovom ćemo se poglavlju osvrnuti na ovaj posljednji aspekt.

8.1. Primjer. Standardni način za računanje determinante 3×3 jest korištenjem Sarrusovog pravila:

$$\begin{vmatrix} a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \\ c_1 & c_2 & c_3 \end{vmatrix} = a_1b_2c_3 + a_2b_3c_1 + a_3b_1c_2 - a_1b_3c_2 - a_2b_1c_3 - a_3b_2c_1.$$

Ovakva formula vrijedi i za determinantu n-tog reda, i često se navodi kao definicija determinante:

$$\begin{vmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{vmatrix} = \sum (-1)^{\pi(j_1,\dots,j_n)} a_{1j_1}a_{2j_2}\cdots a_{nj_n}.$$

Ovdje se sumira po svim permutacijama $(j_1, j_2, \dots, j_n)$ od $(1, 2, \dots, n)$, a $\pi(j_1, \dots, j_n)$ jednak je broju inverzija u toj permutaciji.

Ova se formula lagano implementira u računalni program. No, taj program je potpuno neuporabljiv za iole veće brojeve n. Razlog tome je što ovaj algoritam zahtijeva $n!\cdot(n-1)$ množenje, broj koji raste fascinantno brzo. Kažemo da je njegov rast *faktorijelnog tipa*.

8.2. Primjer. Premještanje kolutova u problemu Hanojskih tornjeva vodi na algoritam koji ima *eksponencijalni rast*. Naime, vrijedi

$$T_n = 2^n - 1.$$

Za velike brojeve n, i ova funkcija raste vrlo brzo, no sporije od faktorijelne.

8.3. Uobičajeni tipovi složenosti algoritama. Većina interesantnih algoritama imat će složenost proporcionalnu nekoj od sljedećih funkcija:

1	Ovo označava konstantan broj operacija, neovisno o veličini niza ulaznih podataka
$\log n$	Ova se složenost javlja kod algoritama koji problem mogu riješiti tako da ga transformiraju u jednostavniji smanjujući broj elemenata za fiksan omjer. Na primjer, potraga za zadanim elementom u nekoj sortiranoj listi. Algoritmi s ovim rastom su vrlo prihvatljivi, jer se složenost povećava samo za malu konstantu u slučaju udvostručavanja broja ulaznih podataka: $\log(2n) = \log n + \log 2$.
n	Složenost je proporcionalna broju ulaznih podataka kod algoritama u kojima se određena količina posla mora napraviti nad svakim elementom ulaznog niza. To je tipična situacija kad algoritam zahtijeva unos ili ispis n podataka.
$n\log n$	Ova se složenost javlja u algoritmima koji za svaki ulazni podatak definiraju podalgoritam rješiv u logaritamskom vremenu. Mnogi algoritmi sortiranja imat će upravo takvo ponašanje. Za umjerene vrijednosti od n ovakav se algoritam ponaša gotovo kao linearan, $\log n$ sporo raste u odnosu prema veličini broja n. Nazvat ćemo ovakve algoritme *loglinearnim*.
n^2	Kvadratna ovisnost o ulaznim podatcima daje algoritam koji je efikasan samo za umjerene vrijednosti broja n. Dvostruko veći niz ulaznih podataka zahtjeva četverostruko više vremena za rješavanje. Ovi se algoritmi javljaju, na primjer, prolazom kroz dvostruku petlju.
n^α	Razni algoritmi imat će ovu složenost za neku vrijednost od α. Parametar α ne mora biti prirodan broj. Kažemo da algoritmi imaju potencijalni rast. Ako je $\alpha < 1$, algoritmi su brži od linearnog. Za $\alpha > 1$ oni su sporiji od linearnog, ali i od loglinearnog, za iole veći broj n. Ako je n cjelobrojan, govorimo još da su ovi algoritmi s **polinomijalnim** rastom.
a^n	Ovdje se pretpostavlja da je $a > 1$, jer složenost mora rasti s volumenom n. Najčešće je $a = 2$. Ovi algoritmi uporabljivi su samo za početne vrijednosti broja n.

8.4. Primjer. Množenje dviju kvadratnih matrica $A = (a_{ij})$, $B = (b_{ij})$, $i, j \in [1, 2, \dots, n]$ odvija se na način

$$c_{ij} = \sum_{k=1}^{n} a_{ik}b_{kj}.$$

Dakle, za svaki element c_{ij} izlazne matrice treba načiniti n množenja i $(n-1)$ zbrajanje. Operacije zbrajanja zanemarit ćemo u odnosu na množenje. Ukupan broj potrebnih operacija je onda n^3, ali složenost algoritma nije tipa n^3 već $n^{1.5}$. Naime, ulazni niz podataka ima n^2 elemenata, pa je on vezan s izlazom formulom $n^3 = (n^2)^{1.5}$.

8.5. Primjer. Jednostavan način ispitivanja je li neki prirodni broj n prost jest ovaj: podijeli ga sa svim prirodnim brojevima manjim od $\sqrt{n}$. Broj je prost ukoliko nije djeljiv niti s jednim od tih brojeva.

Prema ovom algoritmu, potrebno je učiniti najviše $\sqrt{n}$ operacija, pa bismo mogli zaključiti da je složenost ovog algoritma $n^{0.5}$. Međutim, to nije istina. Naši ulazni podatci nisu niz od n brojeva, već jedan jedini broj. Onda se veličina niza ulaznih podataka treba opisati na drugi način, na primjer, brojem bitova (ili znamenaka) ulaznog broja.

Prikažimo n u binarnom sustavu. Ako je n reda veličine 2^M, onda je upravo M niz ulaznih podataka — to su znamenka u binarnom zapisu broja n. Sad brojeva manjih od $\sqrt{n}$ ima $2^{M/2}$ i to je broj operacija koje moramo učiniti. To znači da M ulaznih podataka zahtjeva $2^{M/2}$ operacija i to nam daje sliku o složenosti ovog algoritma — on je praktički neuporabljiv za iole veće brojeve n. (Brojevi koji se koriste reda su veličine 10^{300}.) Danas još uvijek nije poznat algoritam koji bi ovaj problem rješavao u polinomijalnom vremenu[1].

Usporedba rasta funkcija

U ovoj točki će f i g biti dvije pozitivne funkcije definirane na skupu prirodnih brojeva.

O notacija

Kažemo da je funkcija f ograničena funkcijom g ako postoji konstanta C takva da za sve dovoljno velike n vrijedi

$$f(n) \leqslant C\cdot g(n)$$

Pišemo $f(n) = O(g(n))$.

U ovoj formuli konstanta C može biti i veća od 1, pa je tako $f(n) = n^2$ ograničena sa $g(n) = \frac{1}{2}n^2$. Evo još nekih primjera.

8.6. Primjer. Uvjerite se da vrijedi

$$\binom{n}{3} = O(n^3),$$
$$\log n = O(n),$$
$$\log n = O(n^\alpha) \qquad \text{za svaki } \alpha > 0.$$

Korisno je znati sljedeće usporedbe:

$$P_k(n) = O(n^k)$$

za svaki polinom P_k stupnja k.

Također, jasno je da iz

$$f(n) = O(n^3)$$

slijedi i $f(n) = O(n^4)$ pa i $f(n) = O(n^k)$ za svaki $k > 3$. Naravno, u ocjenama ove vrste interesantno je navesti *najmanji* k za koji vrijedi ovakva ocjena.

Ω i Θ notacija

Ako postoji konstanta $B > 0$ takva da za sve dovoljno velike n vrijedi

$$f(n) \geqslant B\cdot g(n)$$

tada ćemo pisati $f(n) = \Omega(g(n))$.

Ako istovremeno vrijedi $f(n) = O(g(n))$ i $f(n) = \Omega(g(n))$, t.j., ako postoje konstante B i C takve da za sve dovoljno velike n vrijedi

$$B\cdot g(n) \leqslant f(n) \leqslant C\cdot g(n).$$

tada ćemo pisati $f(n) = \Theta(g(n))$.

Primijetimo da je $f(n) = \Omega(g(n))$ ekvivalentno s s $g(n) = O(f(n))$.

Ako je $f(n) = \Theta(g(n))$, onda očito vrijedi i $g(n) = \Theta(f(n))$.

8.7. Primjer. Vrijedi

$$an^2 + bn + c = \Theta(n^2), \qquad (a > 0),$$
$$\binom{n}{k} = \Theta(n^k),$$
$$P_k(n) = \Theta(n^k).$$

8.8. Primjer. Ako su f i g dvije rastuće funkcije za koje postoji limes

$$\lim_{n\to\infty}\frac{f(n)}{g(n)} = C,$$

za neku konstantu $C \neq 0$, onda je $f(n) = \Theta(g(n))$.

Tvrdnja neposredno slijedi iz definicije limesa: postoji n_0 takav da za $n \geqslant n_0$ vrijedi primjerice

$$(C/2)g(n) < f(n) < 2Cg(n).$$

Posebno je važan slučaj kad je ovaj limes jednak 1.

Asimptotski ekvivalentne funkcije

Ako vrijedi

$$\lim_{n\to\infty}\frac{f(n)}{g(n)} = 1,$$

tada kažemo da su f i g asimptotski jednake kad $n \to \infty$ i pišemo

$$f(n) \sim g(n) \qquad \text{kad } n \to \infty.$$

Na primjer, za broj prostih brojeva manjih od broja n vrijedi

$$\pi(n) \sim \frac{n}{\ln n}.$$

(Taj rezultat nije očevidan, riječ je o složenom i važnom teoremu.)

Ako je $f(n) \sim g(n)$, onda vrijedi i $f(n) = \Theta(g(n))$. Obrat ne mora vrijediti, $\frac{1}{2}n^2 = \Theta(n^2)$, ali naravno da ne vrijedi $\frac{1}{2}n^2 \sim n^2$.

o i ω notacija

Ako za svaku pozitivnu konstantu $c > 0$ postoji $n_0 > 0$ takav da vrijedi

$$f(n) < cg(n)$$

onda kažemo da funkcija g raste brže nego funkcija f i pišemo $f(n) = o(g(n))$. Zapis ekvivalentan ovome je $g(n) = \omega(f(n))$.

Ova se veza može zapisati i korištenjem limesa, $f(n) = o(g(n))$ znači da vrijedi

$$\lim_{n\to\infty}\frac{f(n)}{g(n)} = 0.$$

Još jedan zapis označava identičnu situaciju, umjesto $f(n) = o(g(n))$ možemo pisati

$$f(n) \prec g(n).$$

8.9. Primjer. Uvjerite se da za $a > 1$ i $0 < \alpha < \frac{1}{2}$ vrijede sljedeće relacije:

$$n \prec n^2 \prec a^n \prec n!$$
$$\ln n \prec n^\alpha \prec n \prec n\ln n$$
$$\ln\ln n \prec \frac{(\ln n)^\alpha}{\ln\ln n} \prec \frac{(\ln n)^{1/2}}{\ln\ln n} \prec \ln n.$$

***

Dvije logaritamske funkcije imaju standardne oznake, $\log$ za logaritam po bazi 10, te $\ln$ za logaritam po prirodnoj bazi e. U računarstvu se najčešće koristi logaritam po bazi 2, pa ćemo njega označavati s $\lg$.

Veza među ovim različitim logaritmima je linearna. Naime, iz relacije

$$\log_b x = \log_b a \log_a x$$

dobivamo

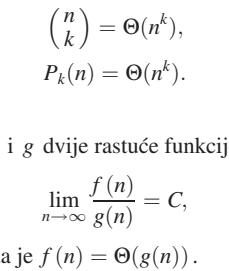
$$\lg x = \log_2 10\cdot\log x = \frac{1}{\log 2}\log x = 3.32\cdot\log x$$
$$\lg x = \log_2 e\cdot\ln x = \frac{1}{\ln 2}\ln x = 1.44\cdot\ln x.$$

***

U nastavku ćemo razmotriti složenost nekih važnih algoritama sortiranja.

Insertion sort algoritam

Na koji način igrač bridža (ili remija) slaže karte u svojoj ruci? Riječ je o trinaest (petnaest) karata koje igrači slažu u nekom poretku. Mnogi igrači to rade tako da pregledavaju jednu po jednu kartu i stavljaju je na njezino ispravno mjesto, s obzirom na karte koje su već složene. Možemo zamisliti da postupak počinje s praznom lijevom rukom, dok se nesortirane karte nalaze ili na stolu ili u desnoj ruci. Uzima se jedna po jedna karta i *umeće* na njezinu ispravnu poziciju u lijevoj ruci. Postupak je gotov nakon 13, odnosno 15 koraka.



Sl. 8.2.

Sortiranje umetanjem pogodno je za podatke umjerene veličine. Promotrimo kako algoritam funkcionira u konkretnom primjeru. Podatci su spremljeni u polje A. Neka je početni raspored $A = \{3, 6, 2, 7, 1, 4, 5\}$. Sortiranje počinje od drugog elementa i završava s posljednjim. U svakom koraku je potcrtan element koji ulazi na sortiranu listu:

$$3\ \underline{6}\ 2\ 7\ 1\ 4\ 5$$
$$3\ 6\ \underline{2}\ 7\ 1\ 4\ 5$$
$$2\ 3\ 6\ \underline{7}\ 1\ 4\ 5$$
$$2\ 3\ 6\ 7\ \underline{1}\ 4\ 5$$
$$1\ 2\ 3\ 6\ 7\ \underline{4}\ 5$$
$$1\ 2\ 3\ 4\ 6\ 7\ \underline{5}$$
$$1\ 2\ 3\ 4\ 5\ 6\ 7$$

Algoritam se može realizirati tako da se pri sortiranju ne koristi nikakvo dodatno polje. Međutim cijena toga jest da će tijekom postupka doći do premještanja velikog broja podataka. Procijenimo potreban broj operacija. Pod operacijom ćemo smatrati *usporedbu dvaju elemenata*.

Neka je n ukupan broj podataka u polju A. Sortiranje počinje s drugim elementom i završava s posljednjim. U koraku $j \in [2, \dots, n]$ moramo usporediti element $A[j]$ s već sortiranim $A[1, 2, \dots, j-1]$. Usporedbu prekidamo u trenutku kad je $A[j] > A[i]$ za neki $i \in [1, j-1]$. U najgorem slučaju trebat će nam za to $j-1$ usporedba.

U najboljem slučaju trebat će nam samo jedna usporedba, ukoliko je $A[j] < A[1]$. Time vidimo da broj operacija nije fiksan i ovisi o početnom rasporedu elemenata u polju. Kod algoritama ovakvog tipa uobičajeno je odrediti njegovu složenost u *najboljem* slučaju i u *najgorem* slučaju. No, najkorisnija informacija je ona o *očekivanom broju* operacija. Pod očekivanim brojem smatramo srednju vrijednost potrebnog broja operacija ako promaramo sve moguće vrste ulaznih podataka, za koje pretpostavljamo da su jednako vjerojatne.

Najgori slučaj za *sortiranje umetanjem* je $(j-1)$ usporedba u svakom od $n-1$ koraka (za svako $j \in [2, n]$). Tada je potreban broj operacija

$$1 + 2 + 3 + \ldots + (n-1) = \frac{(n-1)n}{2}.$$

Interesantno je spomenuti da se taj broj postiže u slučaju kad nikakva promjena u poretku nije potrebna! Naime, propustimo li u algoritam sortiranja polje $A = \{1, 2, 3, \dots, n\}$, svaki korak će zahtijevati maksimalan broj usporedbi, jer se svaki element stavlja na kraj dotad sortirane liste!

S druge strane, najmanji broj usporedbi trebat će za sortiranje liste $A = \{n, n-1, \dots, 2, 1\}$, po jedna usporedba u svakom koraku!

S obzirom da se sortiranje u pravilu primjenjuje na već djelomice sortirane liste, ovoj se nelogičnosti može doskočiti tako da se elementi učitavaju, jedan po jedan, od kraja do početka liste. U tom će slučaju već sortirana lista proći u linearnom vremenu, a ova druga u kvadratnom vremenu.

Efikasnost *insertion sort* algoritma najbolje opisuje očekivani broj operacija za slučajno poredanu listu.

Ako je poredak elemenata u listi potpuno slučajan, tada element $A[j]$ može biti u bilo kojem odnosu s prethodno sortiranim elementima $A[1], \dots, A[j-1]$, pa je srednji broj potrebnih usporedbi jednak

$$\frac{1 + 2 + \ldots + (j-1)}{j} = \frac{j}{2}.$$

Onda je očekivani broj ukupnih operacija jednak

$$\sum_{j=2}^{n}\frac{j}{2} = \frac{n(n+1)}{4} - \frac{1}{2}$$

Dakle složenost algoritma je, za velike vrijednosti od n reda veličine

$$\sim\frac{n^2}{4}.$$

Važno je spomenuti da se algoritam može jednostavno modificirati tako da mu složenost bude reda

$$\sim n\ln n,$$

što je prihvatljiv broj u usporedbi s drugim algoritmima. O tome će biti riječ u kasnijim točkama.

Bubble sort algoritam

Ovaj se algoritam koristi u sortiranju liste podataka $\{a_1, a_2, \dots, a_n\}$ s obzirom na relaciju poretka $\leqslant$ definiranu među njezinim elementima.

Algoritam funkcionira ovako. U prvom koraku uspoređujemo prva dva elementa i ako je $a_1 > a_2$ zamijenimo im poredak. Nakon toga uspoređujemo na isti način drugi i treći element liste i tako dalje do posljednjeg elementa.

Time je definiran jedan prolaz kroz listu podataka. Rezultat tog prolaza jest da će se na posljednjem mjestu naći najveći element u listi. Nakon toga postupak nastavljamo s prvih $(n-1)$ elemenata i dalje ok ne poredamo sve elemente u listi.

Recimo da tim algoritmom želimo poredati elemente u listi $\{4, 2, 6, 1, 3, 5\}$. Onda su nam potrebni sljedeći koraci (rezultati su zapisani po stupcima zbog uštede u prostoru)

$$\underline{4,2},6,3,1,5 \qquad 2,3,1,\underline{4,5},6$$
$$2,\underline{4,6},3,1,5 \qquad \underline{2,3},1,4,5,6$$
$$2,4,\underline{6,3},1,5 \qquad 2,\underline{3,1},4,5,6$$
$$2,4,3,\underline{6,1},5 \qquad 2,1,\underline{3,4},5,6$$
$$2,4,3,1,\underline{6,5} \qquad \underline{2,1},3,4,5,6$$
$$\underline{2,4},3,1,5,6 \qquad 1,\underline{2,3},4,5,6$$
$$2,\underline{4,3},1,5,6 \qquad \underline{1,2},3,4,5,6$$
$$2,3,\underline{4,1},5,6 \qquad 1,\underline{2,3},4,5,6$$
$$2,3,1,\underline{4,5},6 \qquad 1,2,3,4,5,6$$

U svakom su koraku potcrtana dva elementa koja se uspoređuju u tom trenutku.

Složenost algoritma mjeri se brojem usporedbi koje treba napraviti. Taj je broj jednak

$$f(n) = (n-1) + (n-2) + \ldots + 2 + 1 = \frac{n(n-1)}{2}.$$

Dakle, složenost *bubble sort* algoritma je $f(n) \sim n^2/2$. Ona ne ovisi o početnom poretku podataka.

Merge sort algoritam.

To je algoritam za sortiranje liste $\{a_1, a_2, \dots, a_n\}$ koji funkcionira na sljedeći način. U prvom koraku lista se podijeli na dva otprilike jednako velika dijela. Zatim se svaka od podlista sortira u uzlaznom poretku i na kraju se te dvije liste spoje.

Da bismo to pojasnili, moramo objasniti dva postupka. Počnimo s drugim. Algoritam spajanja dviju rastućih lista funkcionira ovako. Ako je

$$a_1 < a_2 < \ldots < a_n, \qquad b_1 < b_2 < \ldots < b_m$$

onda vršimo usporedbu elemenata a_1 i b_1. Ako je $a_1 < b_1$, onda a_1 uklonimo iz njegove podliste i stavljamo na početak zajedničke liste jer je taj element najmanji od svih elemenata u cjelokupnoj listi. Postupak se dalje nastavlja na isti način, usporedbom početna dva elementa u listama $a_2 < a_3 < \ldots < a_n$ i $b_1 < b_2 < \ldots < b_m$. Analogno bismo postupili i u slučaju $a_1 > b_1$, element b_1 bismo izdvojili u zajedničku listu.

Očito je za dovršenje postupka usporedbe potrebno načiniti najviše $m + n - 1$ usporedbi. (Koliki je najmanji potrebni broj?)

Za dovršetak opisa *merge sort* algoritma trebamo još objasniti kako se svaka od dviju podlista u prvom koraku algoritma sortira u uzlaznom poretku. Odgovor je očekivan: to se radi uporabom *merge sort* algoritma! Pogledajmo to na jednom primjeru

1. Početna lista glasi $\{6, 3, 2, 5, 1, 7, 4\}$.
2. Dijelimo je u dvije podliste: $\{6, 3, 2, 5\}$ i $\{1, 7, 4\}$
3. Sortiramo podliste koristeći identičan algoritam, kako slijedi:
   - (a) svaku od njih dijelimo na podliste (koje na koncu postupka imaju najviše dva elementa): $\{6, 3\}$, $\{2, 5\}$ i $\{1, 7\}$, $\{4\}$.
   - (b) Sortiramo dobivene podliste: $\{3, 6\}$, $\{2, 5\}$ i $\{1, 7\}$, $\{4\}$.
   - (c) Spajamo podliste: $\{2, 3, 5, 6\}$ i $\{1, 4, 7\}$.
4. Spajamo sortirane podliste: $\{1, 2, 3, 4, 5, 6, 7\}$
5. Kraj postupka.

Procjenimo broj usporedbi koje *merge sort* algoritam mora učiniti. Označimo taj broj s $M(n)$. Tada je $M(1) = 0$, $M(2) = 1$. Postavit ćemo rekurzivnu relaciju za $M(n)$. U tu svrhu moramo razlikovati slučajeve kad je n paran, odnosno kad je on neparan. Ako je $m = 2k$, tada su obje podliste duljine k pa za njihovo spajanje treba najviše $2k - 1$ usporedbi. Zato je

$$M(2k) = 2M(k) + 2k - 1.$$

Slično, za $n = 2k + 1$ vrijedi

$$M(2k+1) = M(k) + M(k+1) + 2k.$$

Obje se formule mogu napisati na identičan način, korištenjem funkcija $\lceil n\rceil$ i $\lfloor n\rfloor$:

$$M(n) = M(\lceil n\rceil) + M(\lfloor n\rfloor) + n - 1.$$

Formula se bitno pojednostavljuje za posebno odabrane brojeve n. Očito je podjela na podliste najjednostavnija kad je n potencija broja 2. Stavimo $n = 2^r$ i označimo $m_r = M(2^r)$. Onda dobivamo rekurziju

$$m_r = 2m_{r-1} + 2^r - 1.$$

Opće rješenje pridružene homogene jednadžbe je $m_r = A\cdot 2^r$. Partikularno rješenje ima oblik $Br\cdot 2^r + C$. Prvi član $Br\cdot 2^r$ je nastao zbog funkcije 2^r, a drugi zbog -1 u rekurzivnoj jednadžbi. Konstante možemo odrediti na temelju početnih vrijednosti niza: $m_0 = M(1) = 0$, $m_1 = M(2) = 1$, $m_2 = M(4) = 5$. Naime, u slučaju liste s četiri člana trebamo učiniti dvije usporedbe u podlistama i najviše tri u spajanju tih podlista. Sad dobivamo

$$m_r = (r-1)2^r + 1.$$

Time smo za $n = 2^r$ dobili

$$M(n) = M(2^r) = (r-1)2^r + 1 = n(\log_2 n - 1) + 1.$$

Ako n nije potencija broja 2, onda možemo (samo za svrhu izvođenja ove ocjene) originalnu listu nadopuniti elementima koji su veći od postojećih tako da njihov ukupan broj bude n_1, potencija broja 2. Pri tom je n_1 najmanji takav i očito vrijedi $n_1 < 2n$. Provodimo postupak sortiranja i na koncu će se sortirana originalna lista biti prvi na početku završne sortirane liste. Potreban broj usporedbi nije veći od

$$n_1(\log_2 n_1 - 1) + 1.$$

Budući je $n_1 < 2n$, dobivamo $\log_2 n_1 < \log_2 n + 1$. Zaključujemo da vrijedi

$$M(n) < M(n_1) < 2n\log_2 n + 1$$

pa je zato uvijek $M(n) \leqslant 2n\log_2 n$.

Quick sort algoritam

Još je jedan algoritam prikladan za sortiranje lista. Definira se također na rekurzivan način.

Ako je $n = 2$, algoritam uspoređuje dvije vrijednosti i sortira ih ako je potrebno.

Za $n > 2$ odabire se po volji neki element liste. Cilj je odabrati neki element iz sredine sortirane liste, pa se obično uzima element s indeksom $i = \lfloor n/2\rfloor$. Međutim, lista u početku nije sortirana, tako da se zapravo radi o izboru na sreću nekog elementa liste.

Nakon što smo odabrali element x_i, on se uspoređuje s preostalih $(n-1)$ elemenata liste i oni se razvrstavaju u dvije skupine. U skupinu L_i stavljamo sve koji su manji od x_i, a u skupinu D_i sve one koji su veći od x_i. Postupak se rekurzivno nastavlja na svakoj od podlista L_i i D_i.

Pogledajmo algoritam na primjeru.

1. Početna lista glasi $\{6, 3, 2, 5, 1, 7, 4\}$.
2. Odaberemo stožerni element 5.
3. Uspoređujemo taj element s ostalima i time dobivamo podliste $\{3, 2, 1, 4\}$, $\{5\}$ i $\{6, 7\}$.
   - (a) Sortiramo drugu podlistu (ostavimo je kakva jest).
   - (b) U prvoj podlisti biramo stožer 2.
   - (c) Uspoređujemo taj element i dobivamo nove podliste $\{1\}$, $\{2\}$, $\{3, 4\}$
     - (i) Sortiramo podlistu $\{3, 4\}$
4. Spajamo sve podatke: $\{1\}$, $\{2\}$, $\{3, 4\}$, $\{5\}$, $\{6, 7\}$.
5. Kraj postupka.

Potreban broj koraka ovisi o izboru stožernih elemenata. Učinit ćemo njegovu procjenu računajući *očekivani broj*. Za listu duljine n, neka je C_n očekivani broj koraka u ovom algoritmu. Onda vrijedi $C_0 = 0$ i za $n \geqslant 1$:

$$C_n = n - 1 + \frac{1}{n}\sum_{i=1}^{n}(C_{i-1} + C_{n-i}) = n - 1 + \frac{2}{n}\sum_{i=0}^{n-1}C_i.$$

Član $n-1$ bilježi koliko usporedbi treba načiniti da bi se ostali elementi iz liste razvrstali u podliste L_i i D_i. Budući je x_i na sreću odabrani element liste, on se može s vjerojatnošću $1/n$ nalaziti na bilo kojem mjestu (uređene) liste, to nam daje drugi član u ovom izrazu. Odavde

$$nC_n = n(n-1) + 2\sum_{i=0}^{n-1}C_i, \qquad n \geqslant 1$$

Zamijenimo n sa $(n-1)$:

$$(n-1)C_{n-1} = (n-1)(n-2) + 2\sum_{i=0}^{n-1}C_i, \qquad n \geqslant 2$$

Oduzimanjem ovih dviju relacija dobivamo

$$nC_n - (n-1)C_{n-1} = 2(n-1) + 2C_{n-1}$$
$$nC_n = (n+1)C_{n-1} + 2(n-1).$$

Za $n = 1$ ova relacija daje $C_1 = 0$, pa ona vrijedi za sve $n \geqslant 1$. Trebamo riješiti tu rekurziju. Dijeljenjem s $n(n+1)$ dobivamo

$$\frac{C_n}{n+1} = \frac{C_{n-1}}{n} + \frac{2(n-1)}{n(n+1)}$$

Zbrajanjem dobivamo:

$$\frac{C_n}{n+1} = \sum_{k=1}^{n}\frac{2(k-1)}{k(k+1)} = 2\sum_{k=1}^{n}\left(\frac{2}{k+1} - \frac{1}{k}\right)$$

Označimo sumu harmonijskog reda:

$$H_n = 1 + \frac{1}{2} + \ldots + \frac{1}{n},$$

onda je

$$\frac{C_n}{n+1} = 4H_n + \frac{4}{n+1} - 4 - 2H_n = 2H_n - \frac{4n}{n+1}.$$

Dakle,

$$C_n = 2(n+1)H_n - 4n.$$

Nekoliko prvih vrijednosti ovog niza su: $C_1 = 0$, $C_2 = 1$, $C_3 = \frac{8}{3}$, $C_4 = \frac{29}{6}$, $C_5 = \frac{37}{5}$, …. Razlomljene vrijednosti ne trebaju čuditi, jer je riječ o *matematičkom očekivanju* broja koraka u ovom algoritmu.

Na primjer, ako je $n = 3$, tad se algoritam razlikuje ovisno o tome koju vrijednost odaberemo kao prvi ključni element. Ako odaberemo drugi po redu (vjerojatnost toga je $\frac{1}{3}$), onda ćemo nakon dvije usporedbe završiti postupak, jer će manji biti spremljen u dio L_i, a veći u dio D_i. Ako odaberemo najveći ili najmanji (vjerojatnost je $\frac{2}{3}$) trebat će nam dvije usporedbe da to ustanovimo, i onda još jedna usporedba elemenata unutar skupa koji sadrži preostala dva. To su ukupno tri usporedbe. Ukupan očekivani broj usporedbi je:

$$\frac{1}{3}\cdot 2 + \frac{2}{3}\cdot 3 = \frac{8}{3} = C_3.$$

Broj H_n se za velike vrijednosti od n ponaša kao

$$H_n \sim \ln n + \gamma.$$

Tu je $\gamma \approx 0.577215$ Eulerova konstanta. Zato vrijedi

$$C_n \sim 2n\ln n - n(4 - 2\gamma) + 2\ln n + 2\gamma.$$

Za veliko n ostali su sumandi zanemarivi prema prvom, pa vrijedi

$$C_n \sim 2n\ln n.$$

Podijeli pa vladaj algoritmi

Rekurzije koje su povezane uz algoritme tipa *podijeli pa vladaj* obično imaju oblik

$$f(n) = bf(n/k) + g(n) \tag{8.1}$$

Tu je k prirodan broj koji govori na koliko se dijelova u algoritmu dijeli početni skup. Broj b je konstanta ovisna o algoritmu, kao i funkcija smetnje $g(n)$.

Ovu rekurzivnu relaciju treba uzimati *cum grano salis*. Ona evidentno ima smisla samo ako je n/k cijeli broj. Ako n/k nije cijeli broj, tada će dobivena formula za opći član $f(n)$ uglavnom odgovarati ili će se tek zanemarivo razlikovati od prave vrijednosti tog niza. O tome više u primjerima koji će slijediti.

Izvedimo sad formulu za opći član za neke važnije odabire funkcije smetnje $g(n)$.

Složenost *divide and conquer* algoritama

8.10. Teorem *Rekurzivna relacija* (8.1) *ima sljedeća rješenja, u ovisnosti o vrijednosti konstante b i funkcije smetnje $g(n)$:*

	b	$g(n)$	$f(n)$
1.	1	c	$c\log_k n + a_0$
2.	1	cn	$\frac{ck}{k-1}n + a_0 - \frac{ck}{k-1}$
3.	k	c	$An - \frac{c}{k-1}$
4.	k	cn	$n(c\log_k n + A)$
5.	$\neq k$	c	$An^{\log_k b} - \frac{c}{b-1}$
6.	$\neq k$	cn	$An^{\log_k b} + \frac{kc}{k-b}n$

Tu je A neodređena konstanta (koja se računa iz početnih uvjeta).

Dokaz. **1.** Promatrajmo članove niza s indeksima oblika $n = k^m$. Niz (a_m) definiran s

$$a_m = f(n) = f(k^m)$$

zadovoljava rekurzivnu jednadžbu

$$a_m = a_{m-1} + c.$$

Njezino je rješenje

$$a_m = a_0 + mc.$$

Budući je $m = \log_k n$, dobivamo

$$f(n) = c\log_k n + a_0.$$

Primjetimo da za n koji nije oblika k^m ovaj broj ne mora biti cijeli, i on predstavlja samo aproksimaciju prave vrijednosti za $f(n)$.

2. Na isti način dobivamo rekurziju

$$a_m = a_{m-1} + cn = a_{m-1} + k^m$$

pa je

$$a_m = a_0 + c(k^m + k^{m-1} + \ldots + k) = a_0 + ck\frac{k^m - 1}{k-1} = a_0 + ck\frac{n-1}{k-1}$$

i odavde slijedi tvrdnja.

3. Neka je sada $b \neq 1$. Onda rekurzija ima oblik

$$a_m = ba_{m-1} + g(n)$$

i njezino je opće rješenje oblika

$$a_m = A\cdot b^m + g_p.$$

Tu je g_p partikularno rješenje rekurzije. Sada imamo

$$b^m = b^{\log_k n} = n^{\log_k b}.$$

Ukoliko je $g(n) = c$, onda partikularno rješenje ima oblik $g_p = B$, pri čemu je B konstanta koja se dobiva uvrštavanjem ove funkcije u rekurziju. Tako dobivamo **3.** i **5.**

Neka je sad $g(n) = cn$. Ako je $b \neq k$, partikularno rješenje ima oblik Bn, pa uvrštavanjem dobivamo **6.**

Konačno, za $g(n) = cn$ i $b = k$ rekurzija glasi

$$a_m = k\cdot a_{m-1} + ck^m.$$

Sad je opće rješenje oblika $A\cdot k^m = An$, a partikularno ima oblik Bmk^m. Uvrštavanjem dobivamo $B = c$ i odatle **4.**

8.11. Primjer. Definirajmo rekurzivni postupak tipa *podijeli pa vladaj* koji pronalazi najmanji i najveći broj u skupu od n cijelih brojeva. Odredimo broj potrebnih operacija.

▷ Neka je to skup S. Tražimo njegov najveći broj l i najmanji broj d. U svakom koraku podijelit ćemo ga na dva dijela i rekurzivnim postupkom odrediti najmanji i najveći broj l_1, d_1 i l_2, d_2 u svakom od podskupova. Zatim ćemo usporediti d_1 s d_2 i l_1 sa l_2 da otkrijemo ekstreme u cijelom skupu.

Neka je a_n potreban broj operacija. Ako je n paran, dobit ćemo rekurziju

$$a_n = 2a_{n/2} + 2.$$

Njezino rješenje, prema stavci 3. u teoremu ima oblik

$$a_n = An - 2.$$

Konstanta A određuje se iz početnog uvjeta $a_2 = 1$, jer u skupu od 2 elementa trebamo učiniti samo jednu usporedbu. Tako dobivamo $A = \frac{3}{2}$ pa je

$$a_n = \frac{3n}{2} - 2.$$

Kolika se vrijednost treba uzeti ako je n neparan? kako u tom slučaju glasi rekurzivna jednadžba? ◁

8.12. Primjer. Da bismo pomnožili dva n-znamenkasta broja, trebamo učiniti n^2 množenja (tipa znamenka × znamenka). Primjenom *podijeli pa vladaj* algoritma postupak možemo ubrzati.

Broj podijelimo na dva dijela s jednakim brojem znamenaka:

$$a = a_1\cdot 10^{n/2} + a_2, \qquad b = b_1\cdot 10^{n/2} + b_2,$$

Sad je

$$a\cdot b = (a_1\cdot b_1)10^n + (a_1\cdot b_2 + a_2\cdot b_1)10^{n/2} + a_2\cdot b_2.$$

Primjetimo da vrijedi

$$a_1\cdot b_2 + a_2\cdot b_1 = (a_1 + a_2)\cdot(b_1 + b_2) - a_1\cdot b_1 - a_2\cdot b_2$$

pa je u ovom postupku potrebno načiniti samo tri množenja brojeva s $n/2$ znamenaka. Tako dobivamo rekurziju

$$a_n = 3a_{n/2}.$$

Istina, $a_1 + a_2$ ili $b_1 + b_2$ mogu imati znamenku više, ali to može samo zanemarivo promijeniti konačni proračun.

Rješenje rekurzije je $a_n = n^{\log_2 3} \approx n^{1.6}$, što je bitno manje od n^2 za velike vrijednosti od n.

8.13. Primjer. (**Množenje matrica**) Množenje dviju kvadratnih matrica reda n standardnim algoritmom zahtjeva n^3 operacija, pa je složenost algoritma $n^{1.5}$. Može li se ovaj algoritam ubrzati? Konkretno, možemo li uporabom principa *podijeli pa vladaj* smanjiti složenost algoritma množenja dviju (velikih) kvadratnih matrica? Ideja je da se matrica C u jednadžbi

$$C = AB$$

dobije računanjem dvostruko manjih blokova:

$$\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}\begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}$$

Tu je onda

$$C_{ij} = A_{i1}\times B_{1j} + A_{i2}\times B_{2j}, \qquad i, j \in [1, 2] \tag{8.2}$$

Naravno, elemente blokova bismo dobili na isti način, uporabom ovog istog algoritma.

Procjenimo potreban broj operacija. Pretpostavimo da je red matrice n paran. Označimo sa $f(n)$ potreban broj operacija.

U svakom koraku algoritma smo jedno množenje nad matricama reda n zamijenili s računanjem četiriju dvostruko manjih matrica, a svaka od njih zahtjeva množenje dviju matrica reda $n/2$. Prema tome, vrijedi

$$f(n) = 8f(n/2).$$

Dobili smo jednadžbu tipa (8.1), uz $b = 8$ i $k = 2$. Njezino je rješenje

$$f(n) = An^{\log_2 8} = An^3.$$

Promatranjem početnih vrijednosti slijedi $A = 1$. Dakle, ovim postupkom ne dobiva se nikakva ušteda.

8.14. Primjer. (**Množenje matrica, ponovo**) Promotrimo prvo množenje 2×2 matrica:

$$\begin{pmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}\begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix}$$

Ovdje je

$$c_{ij} = a_{i1}\times b_{1j} + a_{i2}\times b_{2j}, \qquad i, j \in [1, 2] \tag{8.3}$$

Da bismo izračunali bilo koji od ovih četiriju elemenata, moramo učiniti dva množenja i jedno zbrajanje. Ukupan broj operacija je 8 množenja i 4 zbrajanja.

Pitanje je, možemo li smanjiti broj množenja na račun povećavanja broja zbrajanja (ili oduzimanja). Pitanje je smisleno, jer se zbrajanje obavlja u neusporedivo kraćem vremenu od množenja.

Odgovor na ovo pitanje dao je V. Strassen 1969. godine: množenje dviju 2×2 matrica može se dobiti pomoću 7 množenja i nešto više od 4 zbrajanja!

Koja je korist od toga? Pokazuje se, popriličnа. Naravno, ona je uočljiva samo za zaista velike matrice. Ukupan broj operacija neće više biti n^3 već je reda veličine $n^{2.81}$. Recimo za matrice reda $n = 10^4$ to znači uštedu od 1.74×10^{11} u odnosu na originalno potrebnih 10^{12} operacija, što je šestina originalnog broja. Za veće matrice, ušteda je još veća.

Kako taj algoritam funkcionira? Umjesto postupka (8.3), koristit ćemo sljedeći. Najprije izračunaj

$$\begin{aligned} d_1 &= (a_{12} - a_{22})\times(b_{21} + b_{22}), \\ d_2 &= (a_{12} + a_{22})\times(b_{11} + b_{22}), \\ d_1 &= (a_{11} - a_{21})\times(b_{11} + b_{12}), \\ e_1 &= (a_{11} + a_{12})\times b_{22}, \\ e_2 &= a_{11}\times(b_{12} - b_{22}), \\ e_3 &= a_{22}\times(b_{21} - b_{11}), \\ e_4 &= (a_{21} + a_{22})\times b_{11}. \end{aligned} \tag{8.4}$$

Nakon toga izračunaj tražene elemente matrice C na način:

$$\begin{aligned} c_{11} &= d_1 + d_2 - e_1 + e_3, \\ c_{12} &= e_1 + e_2, \\ c_{21} &= e_3 + e_4, \\ c_{22} &= d_2 - d_3 + e_2 - e_4. \end{aligned}$$

Da, možemo prebrojati 7 množenja i 18 zbrajanja. Dakle, jedno množenje smo zamijenili s 14 zbrajanja!? Ima li od toga koristi?

Pogledajmo algoritam u cjelosti. Pretpostavimo da je $n = 2^m$ za neki prirodni m. Zamislimo da smo svaku od matrica A i B podijelili u četiri blok-matrice reda 2^{m-1}. Tada se množenje može prikazati u matričnom obliku:

$$\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}\begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}$$

Da bismo odredili C_{11}, C_{12}, C_{21}, C_{22} trebamo primijeniti proceduru identičnu prethodnoj, s tim da u formulama (8.4) brojeve a_{ij} i b_{ij} moramo zamijeniti matricama A_{ij} i B_{ij}, prema (8.2). Time vidimo odmah korist od ovog algoritma, jedno množenje jako velike matrice zamijenili smo s desetak zbrajanja matrica istoga reda.

Naravno, množenje matrica reda 2^{m-1} obavit ćemo primjenom istog rekurzivnog postupka.

Ako početna matrica nije reda 2^m, tada je možemo nadopuniti nulama da bude tog reda i primijenjujemo ovaj algoritam. Koristit ćemo pritom informaciju da se množenje vrši s nul matricama, kad to bude potrebno.

Prebrojimo sada ukupan broj množenja. Neka je $f(n)$ potreban broj operacija za matricu reda n. Promatrat ćemo prvo red koji je potencija broja 2, $n = 2^m$ i označimo taj broj operacija kao funkciju eksponenta m. Neka je $\widehat{f}(m) = f(n)$ taj broj. Ako je $n = 2^m$, problem ćemo svesti na rekurzivni ponavljajući proceduru m puta. U prvom koraku algoritma trebamo načiniti 7 množenja s dvostruko manjim matricama. To se prevodi u formulu

$$\widehat{f}(m) = 7\widehat{f}(m-1).$$

No, ova relacija vrijedi za svaki korak i završava s $\widehat{f}(0) = 1$ (množenje dvaju brojeva). Dakle, vrijedi

$$\widehat{f}(m) = 7^m.$$

Kako je $m = \log_2 n$, onda je broj množenja napisan u funkciji reda matrice n jednak

$$f(n) = 7^{\log_2 n} = n^{\log_2 7} = n^{2.81\cdots}.$$

Na koncu ćemo prebrojati i potreban broj zbrajanja, jer postoji mogućnost da će tu doći do eksponencijalnog rasta.

Neka je $\widehat{g}(m)$ potreban broj zbrajanja za matrice reda 2^m. U prvom koraku algoritma pozivamo se na 7 množenja dvostruko manje matrice, što košta $7\widehat{g}(m-1)$ zbrajanja. Međutim, uz ova zbrajanja, moramo učiniti i 18 zbrajanja matrica reda 2^{m-1} pa je

$$\widehat{g}(m) = 7\widehat{g}(m-1) + 18\cdot(2^{m-1})^2 = 7\widehat{g}(m-1) + \frac{9}{2}\cdot 4^m.$$

Stavimo $\widehat{g}(m) = 7^m\widehat{h}(m)$, pa dobivamo

$$\widehat{h}(m) = \widehat{h}(m-1) + \frac{9}{2}\cdot\left(\frac{4}{7}\right)^m.$$

Vrijedi $\widehat{h}(0) = 0$ jer pri množenju dvaju brojeva nam ne treba operacija zbrajanja. Sada dobivamo

$$\widehat{h}(m) = \widehat{h}(0) + \frac{9}{2}\sum_{k=1}^{m}\left(\frac{4}{7}\right)^k \leqslant \frac{9}{2}\sum_{k=1}^{\infty}\left(\frac{4}{7}\right)^k = \frac{21}{2}$$

pa je

$$\widehat{g}(m) \leqslant \frac{21}{2}\cdot 7^m.$$

Napišemo li taj broj zbrajanja kao funkciju broja n, onda iz $m = \log_2 n$ ponovo slijedi

$$g(n) \leqslant \frac{21}{2}n^{\log_2 7} = \frac{21}{2}\cdot n^{2.81\cdots}.$$

Vidimo da je broj zbrajanja ograničen istim rastom.

***

Neke jednostavne kombinatorne strukture predstavljaju dobar medij za postavljanje različitih algoritama.

Rastav broja u prirodne pribrojnike

Ako je n prirodan broj, tada se prikaz

$$n = p_1 + p_2 + \ldots + p_k, \qquad (p_1 \geqslant p_2 \geqslant \ldots \geqslant p_k)$$

naziva *rastav broja na prirodne pribrojnike*.

Broj takvih rastava označimo s $\operatorname{Rast}(n)$. On ovisi naravno o broju n, no opća formula nije poznata. Tako na primjer, broj 8 ima sljedeće rastave

$8 = 8,$
$8 = 7 + 1,$
$8 = 6 + 2,$
$8 = 6 + 1 + 1,$
$8 = 5 + 3,$
$8 = 5 + 2 + 1,$
$8 = 5 + 1 + 1 + 1,$
$8 = 4 + 4,$
$8 = 4 + 3 + 1,$
$8 = 4 + 2 + 2,$
$8 = 4 + 1 + 1 + 1 + 1,$
$8 = 3 + 3 + 2,$
$8 = 3 + 3 + 1 + 1,$
$8 = 3 + 2 + 2 + 1,$
$8 = 3 + 2 + 1 + 1 + 1,$
$8 = 3 + 1 + 1 + 1 + 1 + 1,$
$8 = 2 + 2 + 2 + 2,$
$8 = 2 + 2 + 2 + 1 + 1,$
$8 = 2 + 2 + 1 + 1 + 1 + 1,$
$8 = 2 + 1 + 1 + 1 + 1 + 1 + 1,$
$8 = 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1,$

Dakle, $\operatorname{Rast}(8) = 21$.

Svi su rastavi, čitano po stupcima, poredani u antiabecednom poretku.

Lako je utvrditi poredak dvaju rastava. Rastav

$$n = p_1 + p_2 + \ldots + p_k,$$

nalazi se na listi prije rastava

$$n = q_1 + q_2 + \ldots + q_r,$$

onda i samo onda ako vrijedi

$$p_1 = q_1,\ p_2 = q_2, \dots,\ p_s = q_s,\ p_{s+1} > q_{s+1}$$

za neki $s \geqslant 0$. Uz pomoć ovog kriterija odredit ćemo *kako izgleda sljedeći rastav po redu u zadanoj listi*. Tako na primjer, na listi rastava broja 36 nalazi se

$$36 = 20 + 8 + 4 + 4.$$

Prvi sljedeći element na toj listi je

$$36 = 20 + 8 + 4 + 3 + 1.$$

Dakle, ako je $p_k > 1$, njega zamjenjujemo s $(p_k - 1) + 1$.

S obzirom da se kompletan algoritam temelji na ispisivanju sljedećeg elementa na zadanoj listi, trebamo još obraditi slučaj kad je $p_k = 1$. Tu je situacija nešto kompliciranija. Evo nekoliko uzastopnih sljednika u promatranoj situaciji:

$36 = 20 + 8 + 4 + 3 + 1$
$36 = 20 + 8 + 4 + 2 + 2$
$36 = 20 + 8 + 4 + 2 + 1 + 1$
$36 = 20 + 8 + 4 + 1 + 1 + 1 + 1$
$36 = 20 + 8 + 3 + 3 + 2$

Rastav prirodnog broja

Napiši algoritam koji će napisati sve rastave prirodnog broja n u antiabecednom poretku.

Svi podskupovi zadanog skupa

Neka je $S = \{a_1, a_2, a_3, \dots, a_n\}$ zadani skup. Naš će zadatak biti *ispisati sve podskupove ovog skupa u nekom poretku*.

Rezultat algoritma treba biti dvojak. Globalno, želimo dobiti poredanu listu svih podskupova. Lokalno, za svaki zadani podskup mora biti jasno kazano kako izgleda njegov sljednik, sljedeći element u listi.

Jednostavnosti radi, označit ćemo elemente skupa S ovako:

$$S = \{1, 2, 3, \dots, n\}.$$

8.15. Prirodni poredak. Podskupova skupa S ima 2^n. Odabir podskupa ekvivalentan je odabiru jednog niza od n elemenata, koji se sastoji od nula i jedinica. Tako na primjer, ako je $n = 6$ onda nizu

$$1\,1\,0\,1\,0\,1$$

odgovara podskup

$$\{1, 2, 4, 6\}.$$

S druge strane, niz nula i jedinica definira prirodni broj u binarnom zapisu:

$$M = \overline{b_nb_{n-1}\dots b_1} = b_1 + b_2\cdot 2 + b_3\cdot 2^2 + \ldots + b_n\cdot 2^{n-1}, \qquad (b_i = 0 \text{ ili } 1),$$

Pri tom vrijedi $0 \leqslant M \leqslant 2^n - 1$. Dakle, svaki prirodni broj manji od 2^n određuje jedan podskup ovog skupa. U tom poretku, sljednik podskupa koji odgovara broju M je podskup koji odgovara broju $M + 1$. Pri tom po dogovoru uzimamo da znamenka b_k u binarnom zapisu broja M određuje hoće li element k pripadati podskupu ili ne.

Na primjer, ako je $n = 6$, onda broju $M = 23$ odgovara binarni prikaz

$$23 = 10111_2$$

pa vidimo da taj niz određuje podskup

$$\{1, 2, 3, 5\}.$$

Sljedeći po redu je

$$24 = 11000_2$$

a njemu odgovara podskup

$$\{4, 5\}.$$

Uvjerite se da su sljedeći podskupovi skupa $\{1, 2, 3, 4\}$, čitani po stupcima, dobiveni ovim postupkom:

$\emptyset$
$\{1\}$
$\{2\}$
$\{1, 2\}$
$\{3\}$
$\{1, 3\}$
$\{2, 3\}$
$\{1, 2, 3\}$
$\{4\}$
$\{1, 4\}$
$\{2, 4\}$
$\{1, 2, 4\}$
$\{3, 4\}$
$\{1, 3, 4\}$
$\{2, 3, 4\}$
$\{1, 2, 3, 4\}$

Podskup 1

Napiši algoritam za određivanje svih podskupova skupa $\{1, 2, \dots, n\}$ u ovom poretku. Posebno, za zadani podskup $\{a_1, a_2, \dots, \alpha_r\}$ odredi sljedeći element na ovoj listi.

8.16. Grayov kod. U prethodnom postupku razlika između dva susjedna elementa u listi može biti drastična. Pokažimo da postoji poredak u kojem će se dva susjedna člana razlikovati najviše u jednom elementu. Algoritam koji daje takav poredak naziva se *Grayov kod*.

Na primjer, takav poredak za skup $\{1, 2, 3\}$ glasi

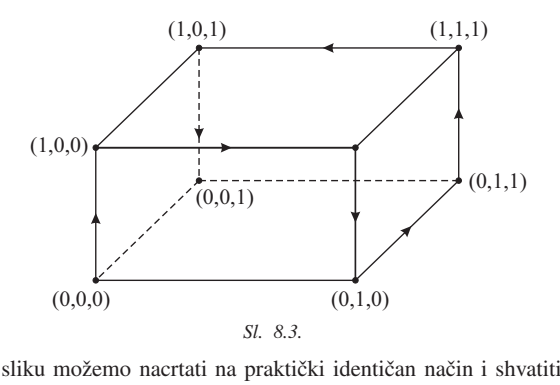
$$\emptyset,\ \{1\},\ \{1, 2\},\ \{2\},\ \{2, 3\},\ \{1, 2, 3\},\ \{1, 3\},\ \{3\}.$$

Po kojem pravilu je ovakav niz sačinjen?

Zamislimo li ponovo da svakom podskupu odgovara niz nula i jedinica, ovu listu možemo zamijeniti sa:

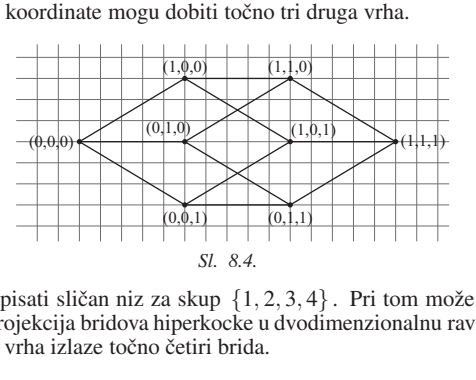
$$(0,0,0),\ (1,0,0),\ (1,1,0),\ (0,1,0),\ (0,1,1),\ (1,1,1),\ (1,0,1),\ (0,0,1).$$

Ovih osam elemenata možemo shvatiti kao koordinate točaka u prostoru. Onda su one vrhovi kocke, a ovim nizom opisan je put koji kreće iz ishodišta i prolazi kroz sve vrhove kocke, tako da se kreće po bridovima kocke i svaki vrh posjeti najviše jednom:



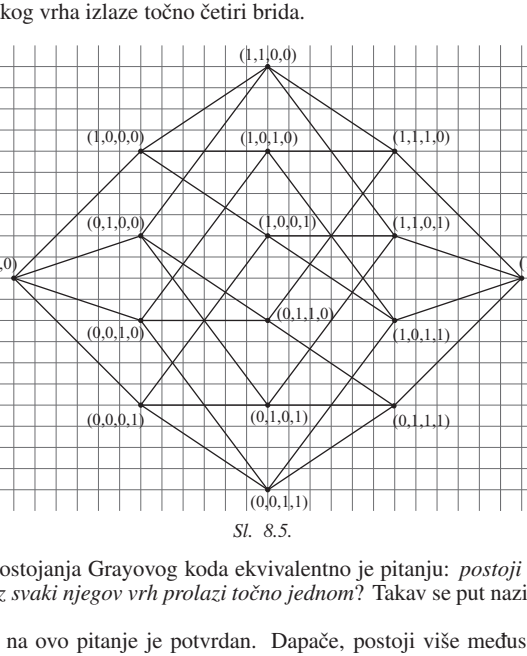
Sl. 8.3.

Tu istu sliku možemo nacrtati na praktički identičan način i shvatiti je kao graf vrhova i bridova kocke. Vidimo da su susjedni oni vrhovi koji se razlikuju samo u jednoj koordinati. Također vidimo da iz svakog vrha grafa izlaze točno tri brida, jer se promjenom jedne koordinate mogu dobiti točno tri druga vrha.



Sl. 8.4.

Pokušajte napisati sličan niz za skup $\{1, 2, 3, 4\}$. Pri tom može pomoći sljedeći graf. Taj je graf projekcija bridova hiperkocke u dvodimenzionalnu ravninu. Primijetite da sada iz svakog vrha izlaze točno četiri brida.



Sl. 8.5.

Pitanje postojanja Grayovog koda ekvivalentno je pitanju: *postoji li put po ovom grafu koji kroz svaki njegov vrh prolazi točno jednom*? Takav se put naziva *Hamiltonov put*.

Odgovor na ovo pitanje je potvrdan. Dapače, postoji više međusobno različitih puteva. To je lako shvatiti ima li se u vidu geometrijska interpetacija. Na primjer, umjesto odabranog puta po vrhovima trodimenzionalne kocke možemo odabrati i bilo koji drugi dobiven rotacijom te kocke oko glavne dijagonale

$$(0,0,0),\ (0,1,0),\ (0,1,1),\ (0,0,1),\ (1,0,1),\ (1,1,1),\ (1,1,0),\ (1,0,0).$$
$$(0,0,0),\ (0,0,1),\ (1,0,1),\ (1,0,0),\ (1,1,0),\ (1,1,1),\ (0,1,1),\ (0,1,0).$$

Primijetite da ove rotacije odgovaraju cikličkoj zamjeni varijabli: $(x, y, z) \mapsto (y, z, x) \mapsto (z, x, y)$. Uvjerite se da time nisu dobivena sva rješenja.

Algoritam za ispisivanje liste Grayovog koda najlakše je dobiti u ovom poretku. Neka $\mathscr{G}_n$ označava traženu listu Grayovog koda. Neka je $\overline{\mathscr{G}}_n$ ta ista lista napisana u suprotnom poretku. Dakle, ako je $\mathscr{G}_n$ put po grafu, onda je $\overline{\mathscr{G}}_n$ put u suprotnom smjeru, od posljednjeg vrha do prvog.

Listu Grayovog koda definiramo rekurzivnim algoritmom. Neka je

$$\mathscr{G}_1 = 0,\ 1.$$

Definiramo

$$\mathscr{G}_n = (\mathscr{G}_{n-1}, 0), (\overline{\mathscr{G}}_{n-1}, 1).$$

Riječima: listu u koraku n dobijemo tako da spojimo dvije liste iz koraka $(n-1)$. Prvu, u kojoj je nadopisana nula na posljednjem mjestu, i drugu, u kojoj je na listu u suprotnom poretku nadopisana jedinica.

Prva četiri koraka glase:

$\mathscr{G}_1$	$\mathscr{G}_2$	$\mathscr{G}_3$	$\mathscr{G}_4$	
0	00	000	0000	0011
1	10	100	1000	1011
	11	110	1100	1111
	01	010	0100	0111
		011	0110	0101
		111	1110	1101
		101	1010	1001
		001	0010	0001

Uvjerite se da je ovaj algoritam dobar za bilo koji n.

U nastavku ćemo promatrati ovaj algoritam i pokušati odgovoriti na pitanje: ako je $(a_1, a_2, \dots, a_n)$ neki element s liste Grayovog koda, kako izgleda njegov sljedbenik?

Primijetimo da se u svakom trenutku mijenja točno jedna koordinata svakog elementa. Stoga je dovoljno znati odgovor na pitanje: *koja koordinata se mijenja u k-tom koraku Grayovog algoritma?*

U primjeru niza $\mathscr{G}_4$ koordinate koje se mijenjaju su:

$$1, 2, 1, 3, 1, 2, 1, 4, 1, 2, 1, 3, 1, 2, 1$$

Ovdje se može naslutiti zakon. Primijetite da će se u općem slučaju, za elemente duljine n promjena događati u prvoj koordinati svaki drugi put, dakle u 2^{n-1} slučajeva, na drugoj koordinati svaki četvrti put, dakle u 2^{n-2} slučajeva, itd, na posljednjoj koordinate samo jednom. Ukupan broj promjena je

$$2^{n-1} + 2^{n-2} + \ldots + 2 + 1 = 2^n - 1.$$

Podskup 2 – Grayov kod

Napiši algoritam za ispisivanje svih podskupova koristeći Grayov kod. Posebno, za zadani podskup $\{a_1, a_2, \dots, \alpha_r\}$ odredi sljedeći element na ovoj listi.

8.17. Leksikografski poredak. Još je jedan način ispisivanja podskupova, abecednim poretkom. U slučaju niza $\{1, 2, 3, 4\}$ rezultat izgleda ovako:

$\emptyset$
$\{1\}$
$\{1, 2\}$
$\{1, 2, 3\}$
$\{1, 2, 3, 4\}$
$\{1, 2, 4\}$
$\{1, 3\}$
$\{1, 3, 4\}$
$\{1, 4\}$
$\{2\}$
$\{2, 3\}$
$\{2, 3, 4\}$
$\{2, 4\}$
$\{3\}$
$\{3, 4\}$
$\{4\}$

Podskup 3 – Leksikografski poredak

Napiši algoritam za ispisivanje svih podskupova u leksikografskom poretku. Posebno, za zadani podskup $\{a_1, a_2, \dots, \alpha_r\}$ odredi sljedeći element na ovoj listi.

Kombinacije iz zadanog skupa

Sad ćemo promatrati izbor k elemenata iz skupa $S = \{1, 2, \dots, n\}$. Nazovimo jedan tako dobiveni podskup k-kombinacijom skupa S. Znamo da takvih k-kombinacija ima $\binom{n}{k}$. Naš je zadatak odrediti algoritam koji će listati sve k-kombinacije u određenom poretku, te otkriti kako se iz nekog člana na dobivenoj listi može odrediti njegov sljedbenik.

Dva su prirodna odabira kad se radi ovakva lista. Prvi je način da se ona sortira abecedno. Drugi način je da se lista niže uz minimalne izmjene, svaki sljedeći element iz prethodnog treba dobiti uz minimalnu izmjenu. S obzirom da su članovi liste skupovi duljine k, minimalna izmjena znači da ćemo jedan element skupa zamijeniti nekim drugim i tako dobiti sljedeći član liste. Nazovimo takvu listu UV-listom, jer u svakom koraku jedan element ulazi na nju, a drugi izlazi van.

Dvije tako dobivene liste 3-kombinacija iz skupa $\{1, 2, 3, 4, 5\}$ izgledaju ovako:

$\{1, 2, 3\}$
$\{1, 2, 4\}$
$\{1, 2, 5\}$
$\{1, 3, 4\}$
$\{1, 3, 5\}$
$\{1, 4, 5\}$
$\{2, 3, 4\}$
$\{2, 3, 5\}$
$\{2, 4, 5\}$
$\{3, 4, 5\}$

$\{1, 2, 3\}$
$\{1, 2, 4\}$
$\{2, 4, 5\}$
$\{2, 3, 5\}$
$\{2, 3, 4\}$
$\{1, 3, 4\}$
$\{1, 3, 5\}$
$\{3, 4, 5\}$
$\{1, 4, 5\}$
$\{1, 2, 5\}$

Primijetimo da je dobivena UV-lista zdesna cirkularna, prvi član se može dobiti iz posljednjeg identičnom izmjenom.

8.18. Abecedno sortiranje k-kombinacija. Algoritam za abecedno sortiranje je jednostavan:

- Ako je $a_k < n$, uvećaj ga za 1,
- inače, odredi najmanji h za koji je $a_{k-h} < n - h$. Ako takvog nema, onda je $a_1 = n - (k-1)$ i postupak je gotov.
- Uvećaj a_{k-h} za jedan,
- Definiraj $a_j = a_{j-1} + 1$ za sve $j = k + 1 - h, k$. Vrati se na početak.

Algoritam za abecedno sortiranje k-kombinacija

Napiši algoritam za ispisivanje svih k-podskupova u leksikografskom poretku. Posebno, za zadani podskup $\{a_1, a_2, \dots, \alpha_k\}$ odredi sljedeći element na ovoj listi.

8.19. Stvaranje UV liste. UV lista se može kreirati na više različitih načina. Mi ćemo koristiti sljedeći rekurzivni postupak.

Rekurzija će ići po obje varijable, k i n. Zato ćemo označiti s $\mathscr{U}(k, n)$ kompletnu traženu listu. Zadat ćemo početak i kraj tražene liste. Prirodno je za prvi element uzeti

$$\{1, 2, \dots, k\}$$

a za posljednji

$$\{1, 2, \dots, k-1, n\}.$$

(Primijetite da se prvi element dobiva iz posljednjeg UV transformacijom, pa će lista biti cirkularna.)

Prije no što nastavimo, dogovorit ćemo jednu oznaku. Ako je $\mathscr{P}$ neka familija skupova, tada sa $\mathscr{P}\oplus\{a\}$ označavamo familiju u kojoj je svakom skupu iz $\mathscr{P}$ dodan element $\{a\}$. Na primjer

$$(\{1\},\ \{1, 3\},\ \{2, 3\})\oplus\{4\} = (\{1, 4\}, \{1, 3, 4\},\ \{2, 3, 4\}).$$

U početku imamo $\mathscr{U}(0, n) = \emptyset$, za sve n i $\mathscr{U}(n, n) = \{1, 2, \dots, n\}$. Također je

$$\mathscr{U}(1, n) = (\{1\}, \{2\}, \dots, \{n\}), \tag{8.5}$$

no to će slijediti i iz algoritma koji upravo opisujemo: $\mathscr{U}(k, n)$ definiramo rekurzivno, na način

$$\mathscr{U}(k, n) = \mathscr{U}(k, n-1), \overline{\mathscr{U}}(k-1, n-1)\oplus\{n\}.$$

Riječima, UV lista k-kombinacija iz skupa od n elemenata dobije se tako da se napiše takva lista k-kombinacija iz skupa od prvih $(n-1)$ elemenata i njoj doda, u obrnutom poretku, lista $(k-1)$-kombinacija tog skupa nadopunjena elementom n.

Konstruirajmo na ovaj način UV listu $\mathscr{U}(3, 5)$. U tu svrhu moramo poznavati dvije liste, $\mathscr{U}(3, 4)$ i $\mathscr{U}(2, 4)$. Rekurzije se dalje nižu, pa je možda bolje odmah krenuti s malim vrijednostima za k. Imamo

$$\mathscr{U}(1, n) = \mathscr{U}(1, n-1), \overline{\mathscr{U}}(0, n-1)\oplus\{n\} = \mathscr{U}(1, n-1), \{n\}$$

i odavde slijedi (8.5). Sada je

$$\begin{aligned} \mathscr{U}(2, n) &= \mathscr{U}(2, n-1), \overline{\mathscr{U}}(1, n-1)\oplus\{n\} \\ &= \mathscr{U}(2, n-1), \{n-1, n\}, \{n-2, n\}, \dots, \{1, n\}. \end{aligned}$$

Odavde dobivamo

$$\mathscr{U}(2, n) = \{1, 2\}, \{2, 3\}, \{1, 3\}, \{3, 4\}, \{2, 4\}, \{1, 4\}, \ldots \{n-1, n\}, \{n-2, n\}, \dots, \{1, n\}$$

Vratimo sa na naš primjer. Imamo

$$\mathscr{U}(2, 3) = \{1, 2\}, \{2, 3\}, \{1, 3\}$$
$$\mathscr{U}(2, 4) = \{1, 2\}, \{2, 3\}, \{1, 3\}, \{3, 4\}, \{2, 4\}, \{1, 4\}$$

pa je

$$\begin{aligned} \mathscr{U}(3, 4) &= \mathscr{U}(3, 3), \overline{\mathscr{U}}(2, 3)\oplus\{4\} \\ &= \{1, 2, 3\}, \{1, 3, 4\}, \{2, 3, 4\}, \{1, 2, 4\} \end{aligned}$$

Konačno, dobivamo

$$\begin{aligned} \mathscr{U}(3, 5) &= \mathscr{U}(3, 4), \overline{\mathscr{U}}(2, 4)\oplus\{5\} \\ &= \{1, 2, 3\}, \{1, 3, 4\}, \{2, 3, 4\}, \{1, 2, 4\}, \\ &\quad \{1, 4, 5\}, \{2, 4, 5\}, \{3, 4, 5\}, \{1, 3, 5\}, \{2, 3, 5\}, \{1, 2, 5\}. \end{aligned}$$

Algoritam za kreiranje UV liste k-kombinacija

Napiši algoritam za ispisivanje svih k-podskupova u UV poretku. Posebno, za zadani podskup $\{a_1, a_2, \dots, \alpha_k\}$ odredi sljedeći element na ovoj listi.

Posljednji zadatak je poprilično težak. Preporučam da se savjeti pronađu u literaturi.

Zadatci za vježbu

8.1. Prvi algoritam ima ponašanje $200n\log_2 n$, drugi $\frac{1}{4}n(n-1)$. Za koje vrijednosti broja n je prvi algoritam efikasniji?

8.2. Za obavljanje jedne operacije kompjutoru treba desetinka mikrosekunde. Kolika je maksimalna veličina n tako da ukupno vrijeme izvođenja programa bude kao u ovoj tablici, za algoritme čija je složenost dana u prvom stupcu tablice?

	sekunda	minuta	sat	dan	mjesec	godina	stoljeće
$\lg n$							
$\sqrt{n}$							
n							
$n\lg n$							
n^2							
n^3							
2^n							
$n!$							

8.3. Neka je $A[1, \dots, n]$ niz različitih brojeva. Ako za $i < j$ vrijedi $A[i] > A[j]$, tada kažemo da par (i, j) čini **inverziju** u A.
- a) Koliko inverzija postoji u nizu $3, 5, 1, 9, 2, 6$?
- b) Koji raspored elemenata iz skupa $\{1, 2, 3, \dots, n\}$ ima najviše inverzija?
- c) Kako algoritam *insertion sort* ovisi o broju inverzija?
- d) Napiši algoritam koji računa broj inverzija u nekoj permutaciji skupa $\{1, 2, \dots, n\}$, a koji ima efikasnost $\Theta(n\lg n)$.

8.4. Da li je neka od funkcija $\lceil\lg n\rceil!$, $\lceil\lg\lg n\rceil!$ polinomijalnog rasta?

8.5. Definirajmo funkcije $\lg^{(k)}(n) := \lg(\lg^{(k-1)}(n))$ (k i n su takvi da je ovaj izraz realan) i

$$\lg^* n := \min\{k \geqslant 0 : \lg^{(k)} n \leqslant 1\}.$$

Ova funkcija je rastuća, no ona raste izrazito sporo. Vrijedi, primjerice,

$$\lg^* 2 = 1,$$
$$\lg^* 4 = 2,$$
$$\lg^* 16 = 3,$$
$$\lg^* 65536 = 4,$$
$$\lg^* 2^{65536} = 5.$$

Istaknuti su brojevi u kojima $\lg^*$ ima skok.

Koja funkcija je asimptotski veća, $\lg(\lg^* n)$ ili $\lg^*(\lg n)$?

8.6. Ako je $k\ln k = \Theta(n)$, dokaži da je onda $k = \Theta(n/\ln n)$.

8.7. U tablici su dane dvije funkcije f i g. Odredi asimptotsko ponašanje funkcije f u odnosu na g. Odgovor treba biti *da* ili *ne* u odgovarajućem polju tablice.

f	g	O	o	Ω	ω	Θ	$\sim$
n^k	a^n						
$\sqrt{n}$	$n^{\sin n}$						
$n^{1+\varepsilon}$	$n^{\ln n}$						
$\ln^k n$	n^ε						
2^n	$2^{n/2}$						
2^n	$2^{n\ln n}$						
$n^{\lg a}$	$a^{\lg n}$						
$\ln(n!)$	$n\ln n$						

Ovdje su $k \geqslant 1$, $\varepsilon > 0$, $a > 1$ konstante.

---

[1] Takav algoritam postoji, ali se realizira na kvantnim računalima koja još nisu u funkciji.