1 skyrius

KOMBINATORINIAI OBJEKTAI IR ALGORITMAI

Paprasčiausi kombinatoriniai objektai yra sveikieji skaičiai, aibės, sekos, medžiai ir grafai. Panagrinėsime galimus jų vaizdavimo būdus. Nuo pasirinktos duomenų struktūros ir nuo jos programinės realizacijos dažnai priklauso realizuojamo algoritmo žingsnių skaičius. Realizuojant konkretų algoritmą efektyviausias nagrinėjamų objektų klasės realizacijos būdas priklauso nuo:

(1) kam mes tuos objektus naudosime, ir

(2) kokias operacijas su jais atlikinėsime.

1.1 Sveikieji skaičiai

Kombinatoriniai algoritmai dažniausiai operuoja sveikaisiais skaičiais. Kadangi visada 1 bitą galima paskirti skaičiaus ženklo kodavimui, tai laikysime, kad sveikieji skaičiai yra neneigiami.

Sveikųjų skaičių vaizdavimas skaičiavimo sistemoje su pagrindu r. Labiausiai paplitęs sveikųjų skaičių vaizdavimo būdas yra skaičiaus vaizdavimas *pozicinėje skaičiavimo sistemoje su pagrindu* r:

$$N = (d_k d_{k-1} \dots d_1 d_0)_r = d_0 + d_1 r + d_2 r^2 + \cdots + d_k r^k,$$

kur $0 \leqslant d_i < r$ ir $d_k \neq 0$, jei $N \neq 0$. Natūralusis skaičius $r > 1$ vadinamas sistemos pagrindu. Kasdieniame gyvenime naudojame pagrindą 10, o kompiuterio atmintis ir programavimo kalbos naudoja pagrindus 2, 8 ir 16. Pateiksime gerai žinomą algoritmą kaip rasti skaičiaus N išraišką r-ėje skaičiavimo sistemoje:

$d_0 := 0$;
$q := N$;

$k := 0$;
while $q \neq 0$ **do**
  $d_k := q \bmod r$;
  $q := \lfloor q/r \rfloor$;
  $k := k + 1$;
if $k \neq 0$ **then** $k := k - 1$

Pavyzdžiui, $13_{10} = 1101_2$, nes $13 = 6 \cdot 2 + 1$, $6 = 3 \cdot 2 + 0$, $3 = 1 \cdot 2 + 1$ ir $1 = 0 \cdot 2 + 1$.

Sveikųjų skaičių vaizdavimas mišrioje skaičiavimo sistemoje. Kartais sveikieji skaičiai yra vaizduojami *mišrioje skaičiavimo sistemoje su pagrindais* $r_0, r_1, \ldots, r_{k-1}$:

$$N = d_0 + d_1 r_0 + d_2 r_0 r_1 + d_3 r_0 r_1 r_2 + \cdots + d_k \prod_{i=0}^{k-1} r_i,$$

kur $0 \leqslant d_i < r_i$ ir $d_k \neq 0$, jei $N \neq 0$. Perėjimo nuo dešimtainės skaičiavimo sistemos prie mišrios skaičiavimo sistemos su pagrindais $r_0, r_1, \ldots, r_{k-1}$ algoritmas yra visiškai panašus į ankstesniame skyrelyje pateiktą algoritmą:

$d_0 := 0$;
$q := N$;
$k := 0$;
while $q \neq 0$ **do**
  $d_k := q \bmod r_k$;
  $q := \lfloor q/r_k \rfloor$;
  $k := k + 1$;
if $k \neq 0$ **then** $k := k - 1$

Pavyzdys 1.1.1. Skaičiuojant laiką, naudojame mišrią skaičiavimo sistemą su pagrindais $60, 60, 24, 7, 52$, pavyzdžiui 1000000 sek. $=$ 1 sav. 4 d. 13 val. 46 min. 40 sek., nes $1000000 = 16666 \cdot 60 + 40$, $16666 = 277 \cdot 60 + 46$, $277 = 11 \cdot 24 + 13$, $11 = 1 \cdot 7 + 4$ ir $1 = 0 \cdot 52 + 1$. Beje, šią skaičiavimo sistemą jau naudojome įvado 2 pavyzdyje, skaičiuodami CPU laiką!

Pavyzdys 1.1.2. Kartais sveikieji skaičiai yra išreiškiami per faktorialus:

$$N = d_0 \cdot 0! + d_1 \cdot 1! + d_2 \cdot 2! + \cdots + d_k \cdot k!,$$

t.y., mišrioje skaičiavimo sistemoje su pagrindais $1, 2, \ldots, k$.

Sveikųjų skaičių vaizdavimas liekanų vektoriais. Kai sveikieji skaičiai yra dideli, dvejetainėje skaičiavimo sistemoje jų išraiškos tampa ilgos, todėl veiksmai su tokiais skaičiais reikalauna daug dvejetainių operacijų. Pasirodo, veiksmams su dideliais sveikaisiais skaičiais galima sukonstruoti efektyvesnius algoritmus, operuojančius ne su pačiais skaičiais, o su liekanomis, gautomis dalinant tuos sveikus skaičius iš pasirinktų mažesnių sveikųjų skaičių. Tai, kad pagal liekanų vektorių galima vienareikšmiškai rasti jas atitinkantį sveikąjį skaičių, kiniečiai jau žinojo daugiau kaip prieš 2000 metų. Todėl veiksmai su liekanomis yra vadinama *moduline aritmetika*, arba *kiniečių aritmetika*.

Teorema 1.1.1 (Kiniečių teorema apie liekanas). *Tegu $p_0, p_1 \ldots, p_{k-1}$ yra poromis tarpusavyje pirminiai natūralieji skaičiai. Lyginių sistema*

$$\begin{aligned} u &\equiv r_0 \pmod{p_0}, \\ u &\equiv r_1 \pmod{p_1}, \\ &\vdots \\ u &\equiv r_{k-1} \pmod{p_{k-1}} \end{aligned}$$

turi vienintelį sprendinį $u \in \mathbb{Z}$: $0 \leqslant u < p_0 p_1 \cdots p_{k-1}$.

Taigi, kiekvieną sveikąjį skaičių u: $0 \leqslant u < p_0 p_1 \cdots p_{k-1}$ vienareikšmiškai atitinka jo liekanų vektorius $(r_0, r_1, \ldots, r_{k-1})$. Tai reiškia, kad vietoje to, kad atlikinėti veiksmus su dideliais sveikais skaičiais, mes galime juos koduoti liekanų vektoriais, po to atlikti tuos veiksmus su liekanomis ir gautą rezultatą (liekanų vektorių) vėl paversti sveikuoju skaičiumi. Skaičiavimus galima dar labiau pagreitinti veiksmus su liekanų vektoriais atliekant lygiagrečiai su k procesorių.

Pavyzdys 1.1.3. Mes galime sudaryti daugybos lentelę visiems natūraliesiems skaičiams, mažesniems už šimtą (t.y., matricą M dydžio 100×100). Tada tokių skaičių daugyba bus vykdoma labai greit: $i \cdot j = M(i, j)$. Pasirinkę, pavyzdžiui, tarpusavyje pirminius skaičius $99, 98, 97$ ir 95, mes galėsime greitai atlikinėti veiksmus su sveikais teigiamais skaičiais mažesniais už $99 \cdot 98 \cdot 97 \cdot 95 = 89\,403\,930$. Tarkime, reikia atlikti veiksmą $9999 \cdot 6666 + 12345678$. Nesunku patikrinti, kad šiuos skaičius atitiks tokie liekanų vektoriai (atitinkamai moduliu $99, 98, 97$ ir 95): $9999 \sim (0, 3, 8, 24)$, $6666 \sim (33, 2, 70, 16)$ ir $12345678 \sim (81, 30, 3, 48)$. Sudauginę pasirinktais moduliais vektorius $(0, 3, 8, 24)$ ir $(33, 2, 70, 16)$, gauname vektorių $(0, 6, 75, 4)$. Taigi, atsakymas bus liekanų vektorius $(0, 6, 75, 4) + (81, 30, 3, 48) = (81, 36, 78, 52)$. Vienintelis sistemos

$$\begin{aligned} u &\equiv 81 \pmod{99}, \\ u &\equiv 36 \pmod{98}, \\ u &\equiv 78 \pmod{97}, \\ u &\equiv 52 \pmod{95} \end{aligned}$$

sprendinys yra $u = 78\,999\,012$. Tai ir yra ieškomas atsakymas.

1.2 Sekos

Priminsime keletą apibrėžimų iš aibių teorijos. *Multiaibe* arba *šeima* vadiname aibę su pasikartojančiais elementais, t.y., rinkinį bet kokių objektų, kur vienodi objektai gali pasikartoti keletą kartų. Pilnai sutvarkytą baigtinę multiaibę vadiname *baigtine seka* arba *vektoriumi*. Pilnai sutvarkytą begalinę multiaibę vadiname *seka*. Terminas "pilnai sutvarkytą" reiškia, kad kiekvienam sekos elementui yra priskirta jo vieta; sukeitę du nelygius sekos elementus vietomis, gausime jau kitą seką. Baigtinės sekos pavyzdys gali būti žodis bet kokioje abėcėlėje A: $u = u_1u_2 \dots u_m$, kur $u_i \in A$. Begalinės sekos pavyzdys yra pirminių skaičių aibė

$$P = \{2, 3, 5, 7, 11, 13, 17, 19, \dots\}$$

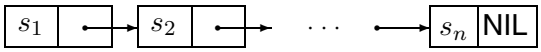
arba visų galimų žodžių abėcėlėje $A = \{a_1, a_2, \dots, a_n\}$ aibė

$$A^* = \{a_1, \dots, a_n, a_1a_1, a_1a_2, \dots, a_na_n, a_1a_1a_1, \dots\},$$

kur trumpesni žodžiai stovi prieš ilgesnius, o vienodo ilgio žodžiai yra išdėstyti leksikografine tvarka, t.y., taip, kaip žodyne. Kombinatoriniai algoritmai operuoja su baigtinėmis sekomis arba baigtiniais begalinių sekų fragmentais. Todėl toliau žodis "seka" reikš baigtinę seką.

Nuoseklus sekų vaizdavimas. Paprasčiausias sekų vaizdavimo būdas yra sekos $S = \{s_1, s_2, \dots, s_n\}$ elementus saugoti nuosekliai išdėstytus masyve ilgio n. Šis būdas leidžia lengvai surasti sekos elementą pagal jo numerį, tačiau yra nepatogus, kai tenka į seką įtraukti naujus elementus arba šalinti elementus iš sekos. Tada tenka perstumti ir kitus sekos elementus.

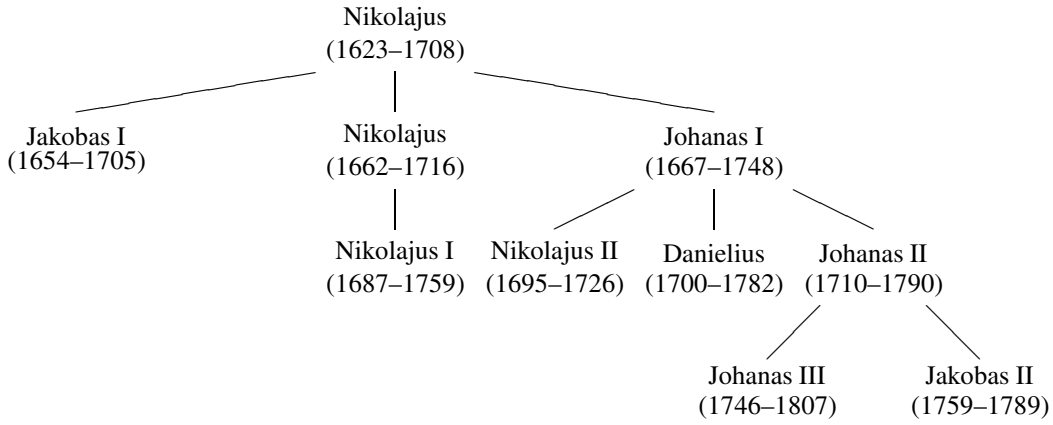
Sekų vaizdavimas sąrašais. Dinaminę seką $S = \{s_1, s_2, \dots, s_n\}$ patogu vaizduoti sąrašu. Kiekvieną sąrašo elementą sudaro informacinė dalis, kur talpiname pačius sekos elementus, ir adresinė dalis, kuri nurodo kokiu adresu rasime kitą sekos elementą. Pradiniu momentu toks sąrašas atrodo taip:



Sekos vaizdavimas sąrašais leidžia greitai vykdyti elementų įterpimą ir šalinimą iš sekos. Iš kitos pusės, šis būdas nėra patogus, kai norime rasti i-ąjį sekos elementą s_i. Be aukščiau pavaizduoto paprasto sąrašo dar naudojami dvigubai susieti sąrašai, kurių elementus sudaro ankstesnio elemento adresas, informacinė dalis ir sekančio elemento adresas.

Sekų vaizdavimas charakteringaisiais vektoriais. Kai nagrinėjamos sekos yra didesnės žinomos sekos $A = \{a_1, a_2, \dots, a_n\}$ posekiai, tai seką $S = \{s_1, s_2, \dots, s_m\}$ galima vaizduoti jos charakteringuoju vektoriumi $\kappa(S) = (\kappa_1, \kappa_2, \dots, \kappa_n)$, kur

$$\kappa_i = \begin{cases} 1, & \text{jei } s_i \in A; \\ 0, & \text{jei } s_i \notin A. \end{cases}$$



Pav. 1.1: Bernoulli matematikų giminės medis.

Kadangi charakteringojo vektoriaus koordinatėms užtenka 1 bito atminties, tai šis sekų vaizdavimo būdas leidžia sutaupyti atmintį, jei nagrinėjami posekiai yra tankūs, t.y., jiems priklauso didelė dalis sekos A elementų.

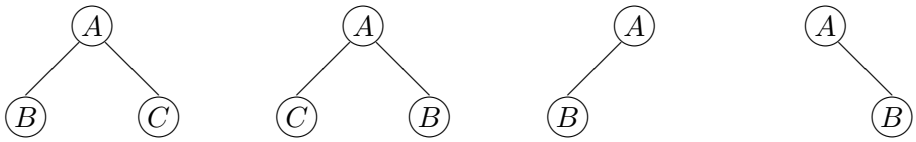
Pavyzdys 1.2.1. Pirminių skaičių, mažesnių už milijoną yra 78498. Kiekvienam skaičiui skiriant po 1 žodį iš 4 baitų, tokių skaičių seka, vaizduojant ją nuosekliai, užims 78498 žodžius. Kadangi, išskyrus skaičių 2, visi kiti pirminiai skaičiai yra nelyginiai, tai pirminių skaičių, mažesnių už milijoną, seką P galima vaizduoti kaip nelyginių natūraliųjų skaičių sekos $\{1, 3, 5, 7, 9, \ldots, 999999\}$ posekį su charakteringuoju vektoriumi $\kappa(P) = (0, 1, 1, 1, 0, 1, 1, 0, \ldots, 0\}$. Tokiam vektoriui reikės $500000/32 = 15625$ žodžių atminties, t.y., apie 5 kartus mažiau, negu pirmuoju būdu.

1.3 Medžiai

Medžiais yra vadinami neorientuoti jungūs grafai be ciklų (žr. 1.5 skyrelį). *Šakniniais medžiais* vadiname medžius, kurių viena viršūnė yra išskirta iš kitų ir vadinama *šaknimi*. Kadangi medžiai yra jungūs ir neturi ciklų, tai šakniniame medyje iš medžio šaknies r į bet kurią jo viršūnę v egzistuoja vienintelis kelias (ta pačia briauna galime eiti tik vieną kartą). Visos šiame kelyje sutinkamos medžio viršūnės u yra vadinamos viršūnės v *protėviais*. Jei viršūnė u yra viršūnės v protėvis, tai viršūnę v vadiname viršūnės u *palikuoniu*. Jei (u, v) yra paskutinė kelio iš šaknies r į viršūnę v briauna, tai viršūnė u yra vadinama viršūnės v *tėvu*, o viršūnė v yra vadinama viršūnės u *vaiku*. Jei kelios viršūnės turi bendrą tėvą, tai tos viršūnės yra vadinamos *broliais*.

Kadangi tikslų grafo apibrėžimą mes pateikiame tik 1.5 skyrelyje, tai čia pateiksime rekursyvų šakninio medžio apibrėžimą:

(i) Viena viršūnė r yra šakninis medis su šaknimi r.



Pav. 1.2: Binarieji medžiai aukščio 1.

(ii) Jei $T_1, T_2, \ldots, T_n$ yra šakniniai medžiai su šaknimis $r_1, r_2, \ldots, r_n$, tai prijungę naują viršūnę r ir sujungę ją briaunomis su kiekviena viršūne $r_1, r_2, \ldots, r_n$, vėl gauname šakninį medį su šaknimi r.

Kadangi algoritmuose paprastai naudojami tik šakniniai medžiai, tai toliau šakninį medį vadinsime tiesiog medžiu. Aukščiau pateikti terminai rodo, kad medžiais yra patogu vaizduoti giminystės ryšius (tokie medžiai yra vadinami *genealoginiais medžiais*). Pavyzdžiui, 1.1 pav. vaizduoja Bernoulli matematikų giminės medį.

Binariuoju medžiu vadiname medį, kurio kiekviena viršūnė turi ne daugiau kaip 2 vaikus, ir kiekvienam vaikui yra žinoma, ar jis kairysis, ar dešinysis vaikas (taigi, viršūnė gali turėti ir vieną vaiką, ir tas vienas vaikas gali būti ir dešinysis!). Pav. 1.2 matome 4 skirtingus binariuosius medžius. Rekursyviai binarieji medžiai apibrėžiami taip:

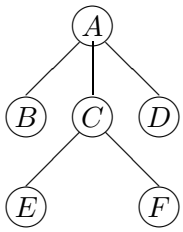
(i) Tuščia aibė yra binarusis medis.

(ii) Jei T_1, T_2 yra binarieji medžiai, tai prijungę naują viršūnę r ir sujungę ją briaunomis su kairiojo pomedžio šaknimi r_1 ir dešiniojo pomedžio šaknimi r_2, vėl gauname binarųjį medį su šaknimi r (jei kuris nors pomedis tuščias, tada su juo nejungiame).

Vaizduojant medžius kompiuterio atmintyje, kiekvienai viršūnei yra skiriamas įrašas, sudarytas iš informacinės dalies ir vienos ar kelių nuorodų. Pagrindiniai medžių vaizdavimo būdai yra šie:

1. *Tėvų nuorodomis*, t.y., kiekvienai viršūnei nurodant jos tėvą. Šis būdas yra naudojamas rekursyviuose algoritmuose, kai norime išsaugoti informaciją apie tai, iš kurios viršūnės mes atėjome. Tačiau jis yra nepatogus, kai reikia surasti viršūnės palikuonius. Be to, jis netinka binariesiems medžiams, nes nurodant tik tėvą, neaišku, ar vaikas yra kairysis, ar dešinysis.

2. *Vaikų nuorodomis*, t.y., kiekvienai viršūnei nurodant visus jos vaikus. Šiuo atveju reikalinga žinoti, kiek daugiausia vaikų gali turėti medžio viršūnės. Jei maksimalus vaikų skaičius lygus m, tai kiekvienas įrašas turės m nuorodų. Kadangi daugelis nuorodų gali būti tuščios (NIL), tai šis būdas reikalauna daug atminties. Tačiau jis labai tinka binariesiems medžiams, kur $m = 2$.



Pav. 1.3: Medžio pavyzdys.

3. Kiekvienai viršūnei nurodant jos *kairyjį vaiką ir dešinyjį brolį* (angl. left-child, right-sibling). Vaizduojant medžius šiuo būdu, kiekvienai viršūnei reikia tik dviejų nuorodų.

Pavyzdys 1.3.1. Visais 3 išvardintais būdais pavaizduosime medį iš 1.3 pav. Vietoje nuorodų naudosime masyvo indeksus.

1. Tėvų nuorodos:

Indeksas	INFO	Tėvas
1	A	NIL
2	B	1
3	C	1
4	D	1
5	E	3
6	F	3

2. Vaikų nuorodos:

Indeksas	INFO	1 vaikas	2 vaikas	3 vaikas
1	A	2	3	4
2	B	NIL	NIL	NIL
3	C	5	6	NIL
4	D	NIL	NIL	NIL
5	E	NIL	NIL	NIL
6	F	NIL	NIL	NIL

3. Nurodant kairyjį vaiką ir dešinyjį brolį:

Indeksas	INFO	Kairysis vaikas	Dešinysis brolis
1	A	2	NIL
2	B	NIL	3
3	C	5	4
4	D	NIL	NIL
5	E	NIL	6
6	F	NIL	NIL

1.4 Aibės

Kadangi algoritmai operuoja tik su baigtinėmis aibėmis, tai sunumeravus kuria nors tvarka duotos aibės elementus, ši aibė virsta seka. Taigi aibėms tinka visi vaizdavimo būdai, kurie yra naudojami sekoms vaizduoti:

1. Nuoseklus vaizdavimas.
2. Vaizdavimas sąrašais.
3. Vaizdavimas charakteringaisiais vektoriais.

Kartais būna patogu aibes vaizduoti dar vienu būdu:

4. Mišku.

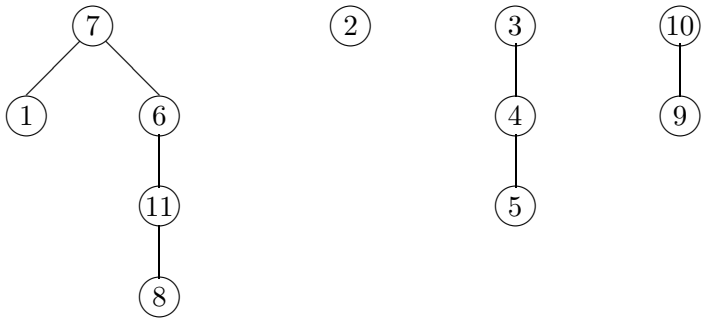
Mišku vadiname vieną ar keletą medžių. Tarkime, kad visos nagrinėjamos aibės yra poromis nesusikertantys didesnės aibės A poaibiai. Kiekvienam poaibiui identifikuoti išskiriame iš kitų bet kurį to poaibio elementą ir jį vadiname poaibio *vardu*. Tarkime, kad mums dažnai reikia rasti, kuriame poaibyje yra duotas aibės A elementas x (operacija $\mathrm{FIND}(x)$), o taip pat dažnai reikia sujungti du poaibius su vardais x ir y į vieną naują poaibį (operacija $\mathrm{UNION}(x, y)$). Tada šiuos poaibius galime vaizduoti medžiais, kurių šaknys yra poaibių vardai. Realizuojant tokią struktūrą kompiuterio atmintyje kiekvienam aibės elementui (medžio viršūnei) pakanka saugoti nuorodą į jos tėvą. Pradiniu momentu laikome, kad kiekvienas aibės elementas sudaro poaibį iš vieno elemento, t.y., jis yra medžio šaknis. Operacija $\mathrm{UNION}(x, y)$ reikš dviejų medžių su šaknimis x ir y sujungimą į vieną naują medį su šaknimi x arba y, o operacija $\mathrm{FIND}(x)$ reikš paiešką miške. Toks aibės vaizdavimo būdas leis atlikinėti minėtas dvi operacijas greičiau, negu tai leidžia kiti aibės vaizdavimo būdai.

Pavyzdys 1.4.1. Duota aibė $A = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11\}$, suskaidyta į 4 poaibius

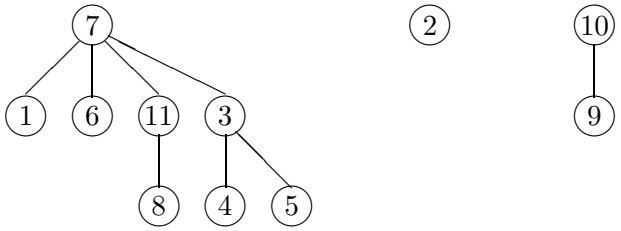
$$\{1, 6, \boxed{7}, 8, 11\}, \quad \{\boxed{2}\}, \quad \{\boxed{3}, 4, 5\}, \quad \{9, \boxed{10}\},$$

kur kvadratėliais pažymėjome poaibių vardus. Tokia struktūra galėjo susidaryti, pavyzdžiui, vykdant štai tokią UNION operacijų seką aibėje A:

$\mathrm{UNION}(8, 11)$;
$\mathrm{UNION}(6, 11)$;
$\mathrm{UNION}(6, 7)$;
$\mathrm{UNION}(1, 7)$;
$\mathrm{UNION}(4, 5)$;
$\mathrm{UNION}(3, 4)$;
$\mathrm{UNION}(9, 10)$



Pav. 1.4: Aibės A vaizdavimas mišku.



Pav. 1.5: Pertvarkyta aibė A.

Tada šią aibę atitiks miškas, pavaizduotas 1.4 pav. Tarkime, kad dabar reikia sujungti poaibius, į kuriuos pateko elementai 11 ir 5. Tai atliks tokia programėlė:

$x :=$ FIND(11);
$y :=$ FIND(5);
if $x \neq y$ **then** UNION(x, y)

Operaciją FIND(x) galima realizuoti, naudojant *kelių suspaudimą*. Tai reiškia, kad kuriame nors medyje eidami viena šaka iš elemento x į medžio šaknį, mes įsimename visas praeitas viršūnes, o radę medžio šaknį y keičiame visų jų nuorodas į y. Tai leidžia nuolat riboti medžių gylį ir tuo pačiu pagreitinti operacijų FIND vykdymą. Taigi, naudojant kelių suspaudimą, trijų aukščiau nurodytų operacijų, pritaikytų miškui iš 1.4 pav. rezultatas bus miškas, vaizduojamas 1.5 pav. Galima įrodyti, kad naudojant kelių suspaudimą ir medžio šakų balansavimą (atliekant operaciją UNION) m FIND ir UNION operacijų galima įvykdyti per $\mathrm{O}(m\alpha(m, m))$ žingsnių, kur $\alpha(m, n)$ yra "atvirkštinė Akermano funkcija". Ši funkcija auga taip lėtai, kad praktikoje galime laikyti, kad $\alpha(m, n) \leq 4$, t.y., gauname praktiškai tiesinį sudėtingumą (žr. [KLR, 22.3]).

1.5 Grafai ir jų vaizdavimas

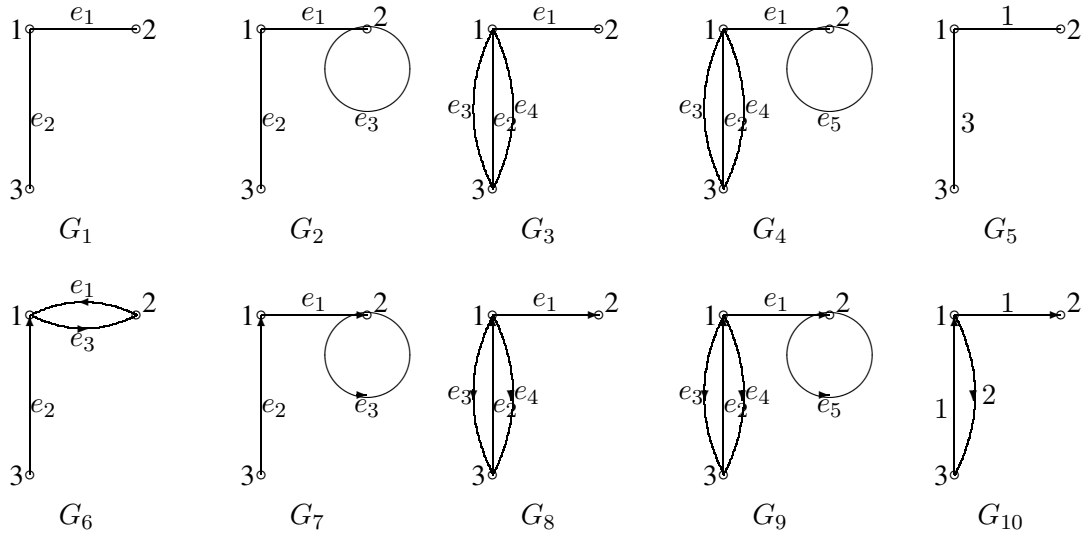
1.5.1 Grafo apibrėžimas ir pagrindinės sąvokos

Grafu vadiname porą $G = (V, E)$, kur V yra bet kokia netuščia aibė, o E yra bet kuris aibės porų iš V elementų multipoaibis. Jei tos poros yra vektoriai, tai grafą vadiname *orientuotu grafu* arba *orgrafu*. Jei poros yra tiesiog aibės V multipoaibiai, tai grafą vadiname *neorientuotu* arba tiesiog grafu ("multi" čia pridėjome todėl, kad gali būti ir poros $\{v, v\} \in E$, kur $v \in V$). Aibė V vadinama grafo G *viršūnių aibe*. Orgrafuose porą $e = (u, v) \in E$ vadiname *lanku* ir sakome, kad lankas $e = (u, v)$ jungia orgrafo G viršūnes u ir v. Dvi poros (u, v) ir (v, u) orgrafe reiškia du skirtingus lankus. Neorientuotuose grafuose viršūnių $u, v \in V$ porą taip pat žymėsime (u, v). Jei $e = (u, v) \in E$, tai porą (u, v) vadiname grafo G *briauna* ir sakome, kad briauna $e = (u, v)$ jungia grafo G viršūnes u ir v. Taip pat sakoma, kad briauna (lankas) $e = (u, v)$ yra *incidentinė* (*-is*) viršūnėms u ir v. Neorientuotuose grafuose dvi poros (u, v) ir (v, u) reiškia tą pačią briauną.

Aukščiau apibrėžti grafai ir orgrafai gali turėti kelias vienodas briaunas (lankus), kurios vadinamos kartotinėmis briaunomis (kartotiniais lankais). Taip pat tokie grafai gali turėti ir *kilpas*, t.y., briaunas (lankus) pavidalo $e = (v, v) \in E$. Kai kuriuose vadovėliuose grafais vadinami tik grafai, neturintys nei kartotinių briaunų, nei kilpų (mes tokius grafus vadinsime *paprastaisiais grafais*), tuo tarpu mūsų apibrėžti grafai ten vadinami *pseudografais*. Grafai su kartotinėmis briaunomis, bet be kilpų, yra vadinami *multigrafais*. Taigi, sprendžiant bet kurį uždavinį, susijusį su grafais, visada reikia pasitikslinti, ar kalbama apie orgrafus ar neorientuotus grafus ir ar grafai gali turėti kartotinių briaunų bei kilpų. Apibendrindami, gauname iš viso 8 galimus variantus: grafai gali būti 2 tipų (orientuoti ir neorientuoti), o kiekvieno tipo grafai dar gali būti 4 rūšių: be kartotinių briaunų ir kilpų, be kartotinių briaunų bet su kilpom, be kilpų bet su kartotinėmis briaunomis ir pagaliau su kartotinėmis briaunomis ir kilpomis.

Be aukščiau išvardintų grafų, grafų teorija taip pat nagrinėja taip vadinamus *svorinius grafus*. Svorinis grafas — tai trejetas $G = (V, E, \omega)$, kur V yra viršūnių aibė, E yra briaunų (lankų) aibė ir $\omega : E \to \mathbb{R}$ yra svorinė funkcija, kiekvienai briaunai (lankui) e priskirianti svorį $\omega(e)$. Vietoje realiųjų skaičių aibės $\mathbb{R}$, briaunų svoriai gali būti iš kitokios aibės, pavyzdžiui $\mathbb{R}^+ = [0, \infty)$ arba $\mathbb{N} = \{0, 1, 2, \dots\}$. Briaunos $e = (u, v)$ svoris $\omega(e)$ dažniausiai reiškia atstumą tarp viršūnių u ir v, tačiau jis gali turėti ir kitą prasmę. Pridėję prie aukščiau išvardintų 8 grafų variantų svorinius grafus bei orgrafus, viso gauname jau 10 galimų variantų. Visi jie yra pavaizduoti 1.6 pav.

Orgrafo viršūnės v *įėjimo laipsniu* $\mathrm{indg}(v)$ vadiname į šią viršūnę įeinančių lankų skaičių, o *išėjimo laipsniu* $\mathrm{outdg}(v)$ — iš šios viršūnės išeinančių lankų skaičių. Neorientuoto grafo viršūnės v *įėjimo laipsniu* $\mathrm{dg}(v)$ vadiname į šią viršūnę įeinančių briaunų skaičių. *Keliu, jungiančiu dvi orgrafo viršūnes u ir v*, vadiname lankų seką $K(u, v) = \{(v_0, v_1), (v_1, v_2), \dots, (v_{k-1}, v_k)\}$, kurioje $v_0 = u$, $v_k = v$ ir du gretimi sekos lankai turi bendrą viršūnę. Kelią $K(u, v)$ sudarančių lankų skaičių k vadiname kelio $K(u, v)$ ilgiu. Vietoje lankų išvardijimo kelią $K(u, v)$ dažnai apibrėžia išvardijant tik viršūnes, per kurias eina šis kelias: $K(u, v) = \{u, v_1, \dots, v_{k-1}, v\}$. Netuščią kelią $K(u, u)$ vadiname *ciklu*. Pav. 1.6 pavaizduotas orgrafas G_6 turi ciklą $C = \{121\}$.



Pav. 1.6: Skirtingų rūšių grafai bei orgrafai. G_1 — paprastas grafas, G_2 — grafas su kilpomis, G_3 — multigrafas, G_4 — grafas, G_5 — svorinis grafas, G_6 — paprastas orgrafas, G_7 — orgrafas su kilpomis, G_8 — multiorgrafas, G_9 — orgrafas, G_{10} — svorinis orgrafas.

Analogiškai keliai ir ciklai yra apibrėžiami ir neorientuotiems grafams. *Oilerio*[3] *ciklu* vadiname ciklą, praeinantį kiekviena grafo briauna (lanku) lygiai po vieną kartą. *Hamiltono*[4] *ciklu* vadiname ciklą, praeinantį per kiekvieną grafo viršūnę lygiai po vieną kartą.

Toliau nagrinėsime baigtinių grafų vaizdavimo būdus. Grafo $G = (V, E)$ viršūnių skaičių žymėsime raide n, o briaunų (lankų) skaičių raide m. Taigi, $|V| = n$ ir $|E| = m$, kur $n = 1, 2, \ldots$ ir $m = 0, 1, 2, \ldots$. Kai $m = 0$, grafo G briaunų aibė yra tuščia. Toks grafas yra sudarytas iš n izoliuotų viršūnių. Jis vadinams *tuščiu grafu* ir žymimas O_n. Jei grafas yra paprastasis, tai

[3]**Leonhard Euler (1707–1783).** Leonardas Oileris gimė kalvinų dvasininko šeimoje netoli Bazelio, Šveicarija. Būdamas 13 metų, jis įstojo į Bazelio universitetą studijuoti teologijos, tačiau vėliau ėmė studijuoti matematiką ir būdamas 16 metų gavo filosofijos magistro laipsnį. 1727 m. Petras Didysis jį pakvietė į Sankt Peterburgą, kur Oileris gyveno iki 1741 m. 1741–1766 m. jis gyveno Berlyne ir dirbo Berlyno Akademijoje, o likusį gyvenimą praleido Sankt Peterburge. Oileris buvo nepaprastai produktyvus mokslininkas, parašęs virš 1100 knygų ir straipsnių. Po savo mirties jis paliko tiek neatspausdintų rankraščių, kad prireikė 47 metų išleisti visiems jo darbams! Oileris įnešė savo įnašą tiek įvairiose matematikos srityse (skaičių teorijoje, kombinatorikoje, matematinėje analizėje), tiek ir jos taikymuose muzikoje bei laivų statyboje. Paskutinius 17 gyvenimo metų Oileris buvo aklas, tačiau jo fenomenalios atminties dėka tai nė kiek nesumažino jo mokslinio produktyvumo.

[4]**William Rowan Hamilton (1805–1865).** Žymiausias airių mokslininkas Wiljamas Hamiltonas gimė Dubline teisininko šeimoje. Būdamas 3 metų, jis jau skaitė ir skaičiavo, o sulaukęs 8 metų mokėjo lotynų, graikų ir hebrajų kalbas. Turėdamas 17 metų jis susidomėjo astronomija ir matematika. Dar būdamas studentu, jis tapo Airijos Karališkuoju Astronomu ir juo buvo iki pat mirties. Hamiltonas pasiekė svarbių rezultatų optikoje, algebroje ir dinamikoje. Algebroje jis pasiūlė taip vadinamus *kvaternionus*. 1857 m. jis sukūrė geometrinį žaidimą, kurio idėją pardavė prekybos agentui. Vienoje iš šio žaidimo versijų reikėjo rasti ciklą, praeinantį per dodekaedro viršūnes lygiai po 1 kartą.

bet kurios dvi jo viršūnės yra sujungtos ne daugiau kaip viena briauna. Be to, briaunos jungia tik skirtingas viršūnes. Taigi paprastasis grafas turi ne daugiau kaip C_n^2 briaunų, t.y., $0 \leqslant m \leqslant C_n^2 = n(n-1)/2$. Grafas, kuriame kiekviena viršūnė yra sujungta su kiekviena kita viršūne, yra vadinamas *pilnu grafu* ir žymimas K_n. Matome, kad paprasti grafai visada turi $m = \mathrm{O}(n^2)$ briaunų. Jei $m = \mathrm{o}(n^2)$, tai grafą vadiname *retu*[5]. Jei $m = \Omega(n^2)$, tai grafą vadiname *tankiu*[6]. Taupant kompiuterio atmintį, priklausomai nuo to ar grafas yra retas ar tankus, dideliems grafams vaizduoti gali būti taikomi skirtingi būdai.

1.5.2 Grafų vaizdavimo būdai

Tegu $G = (V, E)$, kur $V = \{v_1, v_2, \ldots, v_n\}$ ir $E = \{e_1, e_2, \ldots, e_m\}$. Vaizduojant grafus kompiuterio atmintyje įvairiomis stuktūromis, laikysime, kad sveikieji skaičiai yra vaizduojami žodžiais ilgio l. Pvz., jei žodį sudaro 4 baitai, tai $l = 32$.

Grafinis vaizdavimo būdas. Nedidelius grafus patogu vaizduoti grafiškai plokščia diagrama, kurioje kiekviena viršūnė $v \in V$ vaizduojama tašku (arba mažu apskritimu) su greta prirašyta žyme v, o kiekviena briauna $e = (u, v)$ vaizduojama tiesės atkarpa ar kitokia linija, jungiančia taškus pažymėtus u ir v (ši linija negali daugiau eiti per jokią kitą grafo viršūnę). Jei grafas orientuotas, tai lanką (u, v) atitinkanti linija savo antrame gale turi rodyklę, nukreiptą į viršūnę v (žr. 1.6 pav.). Kadangi ne kiekvieną grafą galima pavaizduoti plokščia diagrama taip, kad briaunos nesusikirstų, tai reikia atkreipti dėmesį, kad paprasti briaunų susikirtimo taškai nelaikomi grafo viršūnėmis. Sprendžiant su grafais susijusius uždavinius, dažnai būna patogu pradinį grafą grafiškai vaizduoti kompiuterio ekrane ir su pelyte kaitalioti šio grafo viršūnes bei briaunas.

Gretimumo matrica. Grafo (arba orgrafo) $G = (V, E)$ *gretimumo matrica* vadiname matricą $A = (a_{ij})$, kur

$$a_{ij} = \begin{cases} 1, & \text{jei } (v_i, v_j) \in E; \\ 0, & \text{priešingu atveju.} \end{cases}$$

Taigi, gretimumo matricos A elementas a_{ij} yra vienetas, jei viršūnes v_i ir v_j jungia briauna (t.y., jos yra *gretimos*), ir nulis priešingu atveju. Gretimumo matrica kartais dar yra vadinama sujungimų arba jungumo matrica. Multigrafams gretimumo matricoje vietoje 1 ir 0 tiesiog imame dvi viršūnes jungiančių briaunų skaičių. Svoriniuose grafuose gretimumo matricos elementai yra briaunų svoriai (jei dvi skirtingos viršūnės v_i ir v_j nėra sujungtos briauna, tai laikoma $a_{ij} = \infty$). Tokia matrica dar vadinama *svorine* arba *atstumų* matrica.

[5] $f(n) = \mathrm{o}(g(n))$, jei $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$.

[6] $f(n) = \Omega(g(n))$, jei $\exists N \in \mathbb{N}$ ir $\exists c > 0$: $f(n) \geqslant cg(n)\ \forall n \geq N$.

Pateiksime 1.6 pav. vaizduojamų grafų gretimumo matricas:

$$A_1 = \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix}, \qquad A_6 = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix},$$
$$A_2 = \begin{pmatrix} 0 & 1 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix}, \qquad A_7 = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix},$$
$$A_3 = \begin{pmatrix} 0 & 1 & 3 \\ 1 & 0 & 0 \\ 3 & 0 & 0 \end{pmatrix}, \qquad A_8 = \begin{pmatrix} 0 & 1 & 2 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix},$$
$$A_4 = \begin{pmatrix} 0 & 1 & 3 \\ 1 & 1 & 0 \\ 3 & 0 & 0 \end{pmatrix}, \qquad A_9 = \begin{pmatrix} 0 & 1 & 2 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix},$$
$$A_5 = \begin{pmatrix} 0 & 1 & 3 \\ 1 & 0 & \infty \\ 3 & \infty & 0 \end{pmatrix}, \quad A_{10} = \begin{pmatrix} 0 & 1 & 2 \\ \infty & 0 & \infty \\ 1 & \infty & 0 \end{pmatrix}.$$

Išvardinsime akivaizdžias paprasto grafo gretimumo matricos savybes:

1. Matrica A dvejetainė, t.y., $A \in \{0,1\}^{n^2}$.
2. Matrica A simetriška, t.y., $a_{ij} = a_{ji}\ \forall i, j = 1, 2, \ldots, n$.
3. Matricos A įstrižainė yra sudaryta iš nulių, t.y., $a_{ii} = 0\ \forall i = 1, 2, \ldots, n$.
4. Matricos A i-osios eilutės (stulpelio) suma yra lygi viršūnės v_i laipsniui:

$$\sum_{j=1}^{n} a_{ij} = \sum_{j=1}^{n} a_{ji} = \mathrm{dg}(v_i) \quad \forall i = 1, 2, \ldots, n.$$

Nesunku pastebėti, kad tokiai matricai reikės n^2 bitų arba $n]n/l[$ žodžių atminties (naują eilutę talpiname nuo naujo žodžio pradžios).

Gretimumo struktūra. Grafo (arba orgrafo) $G = (V, E)$ *gretimumo struktūra* vadiname n sąrašų pavidalo

$$v_i \to v_{i1} \to v_{i2} \to \cdots \to v_{ik_i}, \quad i = 1, 2, \ldots, n,$$

kur $v_{i1}, v_{i2}, \ldots, v_{ik_i}$ yra tos viršūnės, kurios su viršūne v_i yra sujungtos briaunomis (lankais). Pavyzdžiui, grafo G_1 gretimumo struktūra bus

$$1 \to 2 \to 3 \to \mathsf{nil}$$
$$2 \to 1 \to \mathsf{nil}$$

$$3 \to 1 \to \mathsf{nil}$$

Matome, kad gretimumo struktūrai reikės $2(2m+n)l$ bitų arba $2(2m+n)$ žodžių atminties (pusė atminties bus skirta rodyklėms, nurodančioms kito sąrašo elemento adresą).

Incidencijų matrica. Grafo be kilpų $G = (V, E)$ *incidencijų matrica* vadiname $n \times m$ matricą $B = (b_{ij})$, kur

$$b_{ij} = \begin{cases} 1, & \text{jei viršūnė } v_i \text{ yra incidentiška briaunai } e_j; \\ 0, & \text{priešingu atveju.} \end{cases}$$

Taigi, incidencijų matricos B elementas b_{ij} yra vienetas tada ir tik tada, kai viršūnė v_i yra vienas iš briaunos e_j galų.

Orgrafo be kilpų $G = (V, E)$ *incidencijų matrica* vadiname $n \times m$ matricą $B = (b_{ij})$, kur

$$b_{ij} = \begin{cases} 1, & \text{jei } e_j = (v_i, v_k); \\ -1, & \text{jei } e_j = (v_k, v_i); \\ 0, & \text{jei viršūnė } v_i \text{ nėra incidentiška lankui } e_j; \end{cases}$$

Pavyzdžiui, grafų G_1 ir G_6 incidencijų matricos bus

$$B_1 = \begin{pmatrix} 1 & 1 \\ 0 & 1 \\ 1 & 0 \end{pmatrix} \quad \text{ir} \quad B_6 = \begin{pmatrix} -1 & 1 & -1 \\ 0 & -1 & 1 \\ 1 & 0 & 0 \end{pmatrix}.$$

Išvardinsime paprasto grafo incidencijų matricos savybes:

1. Matrica B dvejetainė, t.y., $A \in \{0, 1\}^{nm}$.
2. Matricos B i-osios eilutės suma yra lygi viršūnės v_i laipsniui:
$$\sum_{j=1}^{n} a_{ij} = \deg(v_i) \quad \forall i = 1, 2, \ldots, n.$$
3. Matricos B bet kurio stulpelio suma yra lygi 2.

Incidencijų matricai reikės nm bitų arba $n]m/l[$ žodžių atminties.

Briaunų masyvas. Vienas iš paprasčiausių (or)grafo vaizdavimo būdų yra išvardinti visas jo briaunas. Kadangi grafas gali turėti ir izoliuotų viršūnių, tai reikia ne tik išvardinti visas briaunas, bet ir nurodyti grafo viršūnių skaičių n. Taigi *briaunų masyvu* vadinsime vektorių $\vec{b} = (n, v_{11}, v_{12}, v_{21}, v_{22}, \ldots, v_{m1}, v_{m2})$, kur $e_i \in E \to e_i = (v_{i1}, v_{i2})$ $(i = 1, \ldots, m)$. Pavyzdžiui, grafus G_1 ir G_6 atitiks vektoriai $\vec{b_1} = (3, 1, 3, 1, 2)$ ir $\vec{b_6} = (3, 3, 1, 1, 2, 2, 1)$. Dar patogiau yra lankų (briaunų) pradžias saugoti viename masyve ($\vec{p}$), o galus kitame ($\vec{g}$).

Briaunų masyvui reikia $(2m+1)l$ bitų arba $2m+1$ žodžių atminties.

Grafų vaizdavimo būdo pasirinkimas priklauso nuo sprendžiamo uždavinio ir nuo to, kiek grafas gali turėti briaunų, t.y., ar jis yra tankus ar retas. Jei mes taupome atmintį, tai retiems grafams ekonomiškiausi būdai yra grafo vaizdavimas briaunų masyvu arba gretimumo struktūra, o tankiems grafams — gretimumo matrica. Kadangi pereiti nuo vieno būdo prie kito pakanka $O(n^2)$ operacijų, tai algoritmų, kurių sudėtingumas yra $\geqslant \mathrm{const} \cdot n^2$, vykdymo laikas nepriklauso nuo grafo vaizdavimo būdo!

1.6 Algoritmai ir jų sudėtingumas

Nors *algoritmo* sąvoką paprastai sieja su programomis bei kompiuteriais, šis žodis jau yra žinomas daugiau kaip tūkstantį metų. Jis kilęs iš žymaus Rytų mokslininko al-Khowârizmî[7] vardo. XII amžiuje Europoje pasirodė šio mokslininko traktato apie aritmetiką vertimas iš arabų į lotynų kalbą, kuris vadinosi "Dixit algorizmi" ("Al Chorezmis pasakė"). Kadangi šiame traktate buvo aprašomi veiksmai su skaičiais pozicinėje skaičiavimo sistemoje (pavydžiui, sudėtis stulpeliu), tai šie veiksmai, o vėliau ir kitos įvairios procedūros buvo pradėta vadinti algoritmais. Vieną iš tokių procedūrų, kuri tebenaudojama iki šiol dviejų sveikų skaičių bendram didžiausiam dalikliui rasti, dar apie 300 m. prieš Kristų aprašė Euklidas[8].

"Algoritmas" yra pirminė matematikos bei informatikos sąvoka, panašiai kaip sąvokos "skaičius", "aibė" ir kitos. Neformaliai (intuityviai) algoritmą galima apibrėžti kaip objektą, turintį šias savybes:

(1) *Programiškumas*. Tai reiškia, kad algoritmas suprantamas kaip baigtinis taisyklių arba komandų rinkinys (dar vadinamas programa), nusakantis kaip vienus objektus (duomenis), atlikus baigtinį skaičių nesudėtingų operacijų perdirbti į kitus objektus (rezultatus).

(2) *Diskretumas*. Tai reiškia, kad algoritmas apibrėžia nuoseklų duomenų apdorojimo (skaičiavimo) procesą, suskirstytą į atskirus etapus (žingsnius).

(3) *Determinuotumas*. Paprastai reikalaujama, kad po kiekvieno žingsnio būtų vienareikšmiškai apibrėžtas kitas žingsnis arba būtų nurodyta, jog skaičiavimo procesas pasibaigė. Teorinėms

[7] **Abu Ja'far Mohammed ibn Mûsâ al-Khowârizmî (783–850).** Astronomas ir matematikas Al Chorezmis yra kilęs iš Chorezmio miesto (dabartinė Chiva Uzbekistane). Jis buvo Bagdado mokslininkų akademijos, vadintos Išminčių Rūmais, nariu. Vakarų europiečiai algebros pradmenis sužinojo iš jo darbų, išverstų į lotynų kalbą. Žodis *algebra*, kaip ir žodis *algoritmas*, atsirado iš aukščiau tekste minimos knygos pavadinimo, nes arabų kalba jos pavadinimas skambėjo "Kitab al-jabr w'al muquabala".

[8] **Euclid (325–265 P.K.).** Euklidas parašė žymiausią matematinį veikalą pasaulio istorijoje. Jo knyga "Elementai" nuo seniausių iki dabartinių laikų įvairiomis kalbomis buvo išleista daugiau kaip 1000 kartų. Apie jo gyvenimą išliko mažai žinių, tačiau neabejotina, kad Euklidas dėstė žymiojoje Aleksandrijos akademijoje. Jis nesidomėjo matematikos taikymais. Kai vienas jo mokinių paklausė, kur jis galės pritaikyti geometrijos žinias, Euklidas jam atsakė, kad žinių įgyjimas turi būti vertinamas pats savaime, ir liepė savo tarnui duoti šiam mokiniui monetą, jei jis taip nori gauti pajamų iš to, ką jis mokosi.

reikmėms kartais naudojami ir nedeterminuoti algoritmai, leidžiantys keletą galimų kito žingsnio pasirinkimo variantų.

(4) *Žingsnių elementarumas (lokalumas).* Taisyklė, nusakanti kaip pakeisti duomenis per 1 žingsnį turi būti paprasta ir lokali (nežymiai pakeičianti duomenis). Pavyzdžiui, algoritmas negali turėti tokios komandos kaip "Dabar įrodykite Didžiąją Ferma[9] Teoremą laipsnio rodikliui $n = 73$".

(5) *Masiškumas*. Tai reiškia, kad algoritmas turėtų būti pritaikomas įvairiems duomenims iš tam tikros leistinų pradinių duomenų aibės (paprastai begalinės), o algoritmo darbo rezultatai taip pat priklauso apibrėžtai leistinų rezultatų aibei. Pavyzdžiui, taisyklė "norint sudėti du skaičius 2 ir 3 reikia užrašyti, kad atsakymas lygus 5" nėra laikoma algoritmu, nes jos negalima pritaikyti norint sudėti du bet kokius sveikuosius skaičius. Paprastai algoritmuose duomenys ir rezultatai būna *konstruktyvūs objektai*. Konstruktyviu objektu vadiname baigtinį objektą, turintį diskrečią struktūrą ir vidinę koordinačių sistemą. Pavyzdžiui, sveikasis skaičius užrašytas pozicinėje skaičiavimo sistemoje yra konstruktyvus objektas. Tuo tarpu baigtinė aibė, kaipo tokia, dar nėra konstruktyvus objektas. Ji tampa konstruktyviu objektu, tik įvedus kokią nors tvarką tarp jos elementų.

Prie aukščiau išvardintų savybių dažnai dar yra pridedama *rezultatyvumo* savybė, reikalaujanti, kad algoritmas po baigtinio žingsnių skaičiaus sustotų ir išduotų kokį nors rezultatą. Tačiau algoritmų teorijoje paprastai nagrinėjami ir tie algoritmai, kurie kai kuriems pradiniams duomenims gali dirbti be galo arba jiems sustojus rezultatas būna neapibrėžtas. Tokie algoritmai realizuoja ne visur apibrėžtas funkcijas. Dabar pateiksime algoritmų pavyzdžių.

1 pavyzdys: Paieška sąraše (problema SEARCH). Duotas skirtingų objektų (pvz., skaičių) sąrašas (masyvas) $A = \{a_1, \ldots, a_n\}$ ir objektas b. Reikia rasti objekto b vietą sąraše A arba nustatyti, kad tokio objekto sąraše nėra, t.y., apskaičiuoti funkciją

$$\mathrm{location}(A, b) = \begin{cases} i, & \exists a_i = b, \\ 0, & a_i \neq b \ \ \forall i = 1, \ldots, n. \end{cases}$$

Pavyzdyje suformuluotam uždaviniui spręsti naudosime nuosekliosios (tiesinės) paieškos algoritmą LIN_SEARCH ir binariosios paieškos algoritmą BIN_SEARCH. Antrasis algoritmas

[9] **Pierre de Fermat (1601–1665).** Vienas žymiausių XVII a. matematikų Pjeras Ferma buvo teisininkas. Jis yra pats žymiausias pasaulio istorijoje matematikas mėgėjas. Apie jo darbus daugiau yra žinoma tik iš jo susirašinėjimo su kitais matematikais. Ferma buvo vienas iš analizinės geometrijos kūrėjų. Jis taip pat prisidėjo prie integralinio skaičiavimo ir tikimybių teorijos vystymo. Ferma suformulavo uždavinį, kuris ilgą laiką buvo tapęs žymiausia neišspręsta matematikos problema. Tai taip vadinama Didžioji Ferma Teorema, kuri teigia, kad lygtis $x^n + y^n = z^n$ visiems $n \geq 2$ neturi netrivialių sveikaskaitinių sprendinių. Senovės graikų matematiko Diofanto knygos paraštėse Ferma rašė, kad jis žino šios teoremos įrodymą, bet paraštėse neužtenka vietos įrodymui išdėstyti. Tūkstančiai viso pasaulio matematikų daugiau kaip 300 metų nesėkmingai bandė įrodyti šią teoremą, kol 1994 m. Andrew Wiles, remdamasis pačiais naujausiais ir sudėtingais šiuolaikinės matematikos metodais, ją įrodė. Todėl yra manoma, kad arba Ferma žinomas įrodymas buvo klaidingas, arba jis jo nežinojo, o tik bandė kitus paskatinti įrodyti šią teoremą.

tinka tik paieškai pilnai sutvarkytame sąraše, t.y., kai $a_1 < a_2 < \cdots < a_n$ ir objektą b taip pat galime palyginti su visais objektais a_i.

function location $=$ LIN_SEARCH(A, b)
$i := 1$; $a_{n+1} := b$;
while $b \neq a_i$ **do** $i := i + 1$;
if $i \leq n$ **then** location $:= i$ **else** location $:= 0$

Pastebėkite, kad čia mes pateikiame šiek tiek "patobulintą" nuosekliosios paieškos algoritmą. Tam, kad nereikėtų tikrinti ciklo **while** viduje tikrinti jo pabaigos sąlygos $i \leqslant n$, mes prijungiame prie sąrašo patį ieškomą objektą b, ko pasėkoje ciklas visada užsibaigs po $\leqslant n + 1$ žingsnių.

Dabar įvertinsime algoritmo LIN_SEARCH sudėtingumą. Paprastai algoritme galima išskirti vieno ar kelių tipų esmines operacijas, nuo kurių kiekio priklauso bendras algoritmo sudėtingumas (bendras sudėtingumas paprastai būna konstantą kartų didesnis už esminių operacijų skaičių). Sprendžiant paieškos ar rūšiavimo uždavinius esminės operacijos yra objektų palyginimai tarpusavyje. Kai duoti objektai yra ne skaičiai bet sudėtingesnės struktūros objektai (pavyzdžiui, sąrašo elementas gali būti asmens vardas ir pavardė), tai šios esminės operacijos dažnai yra vykdomos ilgiau už paprastas aritmetines ar priskyrimo operacijas, todėl jų kiekis ir lemia bendrą algoritmo sudėtingumą. Taigi, spręsdami paieškos uždavinį skaičiuosime tik objekto b palyginimus su objektais a_i. Kai sąrašo A ilgis n artės į begalybę, bendras žingsnių skaičius skirsis nuo atliktų palyginimų skaičiaus ne daugiau kaip pastoviu daugikliu (tiksliau, jei palyginimų bus P, tai viso bus įvykdyta $2P + \text{const}$ žingsnių).

Pažymėkime $\widetilde{L}(A, b)$ skaičių palyginimų, kuriuos reikės atlikti, kol rasime objekto b vietą sąraše A, naudodami nuoseklios paieškos algoritmą. Pavyzdžiui, $\widetilde{L}(\{1, 3\}, 1) = 1$, $\widetilde{L}(\{1, 3\}, 3) = 2$, $\widetilde{L}(\{1, 3\}, 5) = 3$. Jei sąrašo A ilgis yra n, tai geriausiu atveju reikės atlikti 1 palyginimą (kai $b = a_1$), o blogiausiu atveju reikės $n + 1$ palyginimo (kai objekto b sąraše A nėra). Kadangi algoritmo sudėtingumas uždaviniui dydžio n apibrėžiamas kaip sudėtingumas blogiausiems duomenims dydžio n, tai paieškos sąraše uždavinio sudėtingumas, sprendžiant šį uždavinį nuosekliosios paieškos algoritmu, yra

$$L^{\text{SEARCH}}_{\text{LIN\_SEARCH}}(n) \leqslant 2(n + 1) + \text{const} = \text{O}(n).$$

Dabar panagrinėkime *binariosios paieškos* algoritmą, kurio idėja yra sąrašą nuosekliai dalinti pusiau ir lyginti objektą x su viduriniu sąrašo objektu tam, kad nustatyti, kuriame iš dviejų gautų perpus trumpesnių sąrašų reikia tęsti paiešką.

function location $=$ BIN_SEARCH(A, b)
$i := 1$; $j := n$;
while $i < j$ **do**
  $k := \lfloor (i + j)/2 \rfloor$;
  if $b \leqslant a_k$ **then** $j := k$ **else** $i := k + 1$;
if $b = a_i$ **then** location $:= i$ **else** location $:= 0$;

Kadangi po kiekvienos ciklo **while** iteracijos sąrašo, kuriame yra tęsiama paieška, ilgis $i + j - 1$ sumažėja maždaug perpus, tai nesunku įsitikinti, kad šis algoritmas atlikęs ne daugiau kaip $c \cdot \log_2 n$ žingsnių (kur c yra nedidelė konstanta) sustos ir išduos atsakymą (rezultatą) location. Taigi, jo sudėtingumas bus

$$L^{\mathrm{SEARCH}}_{\mathrm{BIN\_SEARCH}}(n) = \mathrm{O}(\log_2 n).$$

2 pavyzdys: Sąrašo rūšiavimas (problema SORT). Duotas objektų (pvz., skaičių) sąrašas (masyvas) $A = \{a_1, \ldots, a_n\}$, kuriame apibrėžtas pilnos tvarkos sąryšis $\leqslant$. Reikia duotą sąrašą surūšiuoti, t.y. išdėstyti jo elementus nemažėjančia tvarka: $A' = \{a_{i_1} \leqslant a_{i_2} \leqslant \cdots \leqslant a_{i_n}\}$ (kitaip sakant, reikia nurodyti kėlinį arba bijekciją $\pi : \{1, \ldots, n\} \to \{1, \ldots, n\}$ tokią, kad $\pi(j) = i_j$).

Trivialus rūšiavimo algoritmas (dar vadinamas brutalios jėgos algoritmu, BRUTE_FORCE_SORT gali būti toks: išrenkame mažiausią duoto sąrašo elementą, pašaliname jį iš pradinio sąrašo ir patalpiname į 1-ą naujo sąrašo vietą; po to iš likusių pradinio sąrašo elementų vėl išrenakame mažiausią ir patalpiname į 2-ą naujo sąrašo vietą ir t.t. Šio algoritmo sudėtingumas

$$L^{\mathrm{SORT}}_{\mathrm{BRUTE\_FORCE\_SORT}}(n) = \mathrm{O}(n^2).$$

Yra žinomi asimptotiškai greitesni rūšiavimo algoritmai (t.y., greitesni pakankamai didelėms n reikšmėms). Tokie yra, pavyzdžiui, sąlajos rūšiavimo algoritmas MERGE_SORT ir "krūvą" (specialaus pavidalo duomenų struktūrą) naudojantis algoritmas HEAP_SORT. Yra įrodyta (žr. 1.8.1 skyrelį), kad jų sudėtingumas yra

$$L^{\mathrm{SORT}}_{\mathrm{MERGE\_SORT}}(n) = L^{\mathrm{SORT}}_{\mathrm{HEAP\_SORT}}(n) = \mathrm{O}(n \log_2 n).$$

Bet kurio algoritmiškai išsprendžiamo uždavinio sudėtingumas priklauso nuo:

(1) uždavinio dydžio;

(2) pasirinkto algoritmo;

(3) konkrečių duomenų;

(4) vidinio problemos sudėtingumo (paieškos sąraše problema yra paprasta, sąrašo rūšiavimo problema yra sudėtingesnė, o visų galimų skirtingų sąrašų ilgio n generavimo problema dar sudėtingesnė, nes pareikalaus $\mathrm{const} \cdot n!$ elementarių operacijų).

Tarkime, parametras n charakterizuoja uždavinio dydį palyginus jį su kitais tos pačios klasės uždaviniais. Iš pateiktų pavyzdžių matyti, kad konkretaus uždavinio U dydį tarp to paties tipo (pvz., SEARCH arba SORT) uždavinių, susijusių su sąrašais, charakterizuoja pradinio sąrašo ilgis n, nes kuo ilgesnis yra sąrašas, tuo ilgiau užtruks objekto paieška arba sąrašo rūšiavimas. Taigi, uždavinio dydis dažniausiai priklauso nuo jo pradinių duomenų dydžio. Pradiniai duomenys

algoritmuose gali būti sveikieji skaičiai, aibės (masyvai), matricos, grafai ir įvairūs kiti objektai. Pavyzdžiui, kuo didesnis yra natūralusis skaičius, tuo ilgesnis yra jo dvejetainis kodas. Kuo yra didesnė matrica, tuo daugiau ji turi eilučių ir stulpelių. Kuo didesnis grafas, tuo daugiau jis turi viršūnių ir lankų. Taigi, jei A yra natūralusis skaičius, tai jo dydis $n = |A| = \lceil \log_2 A \rceil$; jei $A = \{a_1, \ldots, a_n\}$ yra aibė, tai jos dydis yra elementų skaičius $n = |A|$; jei A yra kvadratinė $n \times n$ matrica, tai jos dydis yra matricos eilė n. Grafo dydžiu, priklausomai nuo nagrinėjamo uždavinio, gali būti laikoma jo viršūnių skaičius n, jo briaunų skaičius m, abu šie parametrai m ir n, arba kokia nors jų kombinacija, pvz. $m+n$ arba $\max(m, n)$. Konkretaus uždavinio U iš klasės $\mathcal{U}$ dydį žymėsime $|U|$.

Tarkime, $\mathcal{U}$ yra uždavinių klasė ir A yra konkretus algoritmas šios klasės uždaviniams spręsti. Algoritmo A, sprendžiančio konkretų uždavinį $U \in \mathcal{U}$, sudėtingumu vadiname algoritmo A žingsnių skaičių iš pradinės konfigūracijos iki sustojimo ir žymime $L_A(U)$ (kartais tą patį dydį vadina konkretaus uždavinio $U \in \mathcal{U}$ sudėtingumu sprendžiant uždavinį U algoritmu A). Algoritmo žingsnių skaičių, sprendžiant uždavinį U, dar vadina laiku arba laiko sudėtingumu ir žymi $T_A(U)$, o panaudotos atminties kiekį vadina erdve arba erdvės sudėtingumu ir žymi $S_A(U)$. Algoritmo A, sprendžiančio klasės $\mathcal{U}$ uždavinius, sudėtingumu vadiname algoritmo A sudėtingumą sprendžiant sudėtingiausią uždavinį iš vienodo dydžio uždavinių iš klasės $\mathcal{U}$ ir žymime

$$L_A^{\mathcal{U}}(n) = \max_{U \in \mathcal{U}:\ |U|=n} L_A(U).$$

Naudojant du skirtingus algoritmus A ir B, gausime skirtingus sudėtingumus $L_A^{\mathcal{U}}(n)$ ir $L_B^{\mathcal{U}}(n)$. Todėl uždavinių klasės $\mathcal{U}$ sudėtingumu vadinsime dydį nepriklausantį nuo konkretaus algoritmo, o būtent pasirinksime paties geriausio algoritmo sudėtingumą:

$$L^{\mathcal{U}}(n) = \min_A L_A^{\mathcal{U}}(n).$$

Kai uždavinių klasė $\mathcal{U}$ yra numanoma, kartais jos sudėtingumą žymi tiesiog $L(n)$. Iš aukščiau pateiktų pavyzdžių matyti, kad paieškos *pilnai sutvarkytame* sąraše uždavinio SEARCH sudėtingumas yra $L^{\mathrm{SEARCH}}(n) = \mathrm{O}(\log_2 n)$, o sąrašo rūšiavimo uždavinio SORT sudėtingumas yra $L^{\mathrm{SORT}}(n) = \mathrm{O}(n \log_2 n)$.

Kai kurie algoritmai blogiausiu atveju dirba ilgai, tačiau vidutiniškai jie dirba trumpiau (juk blogiausias duomenų atvejis gali niekada ir nepasitaikyti!). Todėl kartais naudojamas *vidutinis algoritmo sudėtingumas*. Tarkime, kiekvienam n ir kiekvienam konkrečiam uždaviniui U dydžio n mes žinome tikimybę $p(U)$, su kuria šis uždavinys pasitaikys, sprendžiant uždavinius iš klasės $\mathcal{U}$. Jei mes iš anksto žinome uždavinio dydį n, tai su tikimybe 1 kuris nors konkretus uždavinys mums pasitaikys. Taigi tikimybės turi tenkinti sąlygas

$$p(U) \geqslant 0 \ \ \forall U \quad \text{ir} \quad \sum_{U \in \mathcal{U}:\ |U|=n} p(U) = 1.$$

Algoritmo A, sprendžiančio klasės $\mathcal{U}$ uždavinius, vidutiniu sudėtingumu vadiname

$$\overline{L}_A^{\mathcal{U}}(n) = \sum_{U\in\mathcal{U}:\ |U|=n} p(U)L_A(U).$$

Pavyzdžiui, žymusis greito rūšiavimo algoritmas QUICK_SORT blogiausiu pradinių duomenų išsidėstymo atveju dirbs taip pat ilgai kaip ir paprastesni rūšiavimo algoritmai (pavyzdžiui, "burbulo" algoritmas), t.y., $L^{\mathrm{SORT}}_{\mathrm{QUICK\_SORT}}(n) = \mathrm{O}(n^2)$, tuo tarpu vidutinis šio algoritmo sudėtingumas yra $\overline{L}^{\mathrm{SORT}}_{\mathrm{QUICK\_SORT}}(n) = \mathrm{O}(n\log_2 n)$, todėl šis algoritmas ir yra plačiai naudojamas.

Panagrinėkime tiesinės paieškos algoritmo vidutinį sudėtingumą. Pažymėkime $\widetilde{L}(U)$ palyginimų (t.y., esminių operacijų) skaičių, taikant algoritmą LIN_SEARCH uždaviniui U. Tarkime, nepriklausomai nuo pradinių duomenų, ieškomas objektas b su vienoda tikimybe $1/(n+1)$ gali sutapti su bet kuriuo sąrašo objektu a_i ir su tokia pat tikimybe jo iš viso nebus sąraše A. (Pastebėkite, kad čia mes pateikiame supaprastintas pradines sąlygas, kurios skiriasi nuo sąlygų, pateiktų vidutinio sudėtingumo apibrėžime, nes mes apibrėžiame ne konkretaus uždavinio U tikimybę $p(U)$, o tikimybę p_i, kad konkretus uždavinys pasitaikys iš tam tikro klasės $\mathcal{U}$ poaibio $\mathcal{U}_i$.) Kadangi tuo atveju, kai $b = a_i$, algoritmas LIN_SEARCH atlieka i palyginimų, tai vidutinis jo sudėtingumas bus

$$\overline{\widetilde{L}}^{\mathrm{SEARCH}}_{\mathrm{LIN\_SEARCH}}(n) = \frac{1}{n+1}\sum_{i=1}^{n+1} i = \frac{1}{n+1}\frac{(n+2)(n+1)}{2} = \frac{n}{2}+1.$$

1.7 Viršutiniai įverčiai

Norint gauti uždavinio sudėtingumo viršutinį įvertį, pakanka sukonstruoti algoritmą, kuris sprendžia duotą uždavinį, ir kurio sudėtingumas sutampa su norimu įverčiu. Dažnai net ir standartinėse situacijose, kur, atrodo, nieko naujo nebegalima išgalvoti, iš tiesų pavyksta rasti efektyvesnius algoritmus, o kartu ir pagerinti viršutinius uždavinių sudėtingumo įverčius. Panagrinėsime du tokius pavyzdžius.

1.7.1 Didžiausio ir mažiausio aibės elemento paieška

Duota aibė $A = \{a_1, a_2, \ldots, a_n\}$, kurioje apibrėžtas pilnos tvarkos sąryšis $\leqslant$. Reikia rasti didžiausią tos aibės elementą maxA ir mažiausią elementą minA (uždavinys MAXMIN). Tarkime, kad aibės elementų palyginimo operacijos reikalauna daugiau kompiuterinio laiko už kitokias operacijas, todėl skaičiuosime tik palyginimus. Akivaizdu, kad didžiausią aibės elementą galime rasti, panaudoję $n-1$ palyginimą:

$\mathrm{maxA} := a_1$;
for $i = 2 : n$ **do**
  if $a_i > \mathrm{maxA}$ **then** $\mathrm{maxA} := a_i$;

Analogiškai galime surasti ir mažiausią aibės elementą. Taigi trivialus viršutinis šio uždavinio sudėtingumo įvertis yra $L^{\mathrm{MAXMIN}}(n) \leqslant 2n-2$. Tuo atveju, kai $n = 2^k$, įrodysime, kad šį įvertį galima pagerinti iki $L^{\mathrm{MAXMIN}}(n) \leqslant \frac{3}{2}n - 2$. Naudosime rekursyvią funkciją maxmin:

function $[\mathrm{maxA}, \mathrm{minA}] = \mathrm{maxmin}(A)$
if $n = 2$ **then**
  if $a_1 > a_2$ **then**
    $\mathrm{maxA} := a_1$; $\mathrm{minA} := a_2$
  else
    $\mathrm{maxA} := a_2$; $\mathrm{minA} := a_1$
else
  $k := n/2$;
  $A_1 := \{a_1, a_2, \ldots, a_k\}$;
  $A_2 := \{a_{k+1}, a_{k+2}, \ldots, a_n\}$;
  $[\mathrm{max1}, \mathrm{min1}] := \mathrm{maxmin}(A_1)$;
  $[\mathrm{max2}, \mathrm{min2}] := \mathrm{maxmin}(A_2)$;
  if $\mathrm{max1} > \mathrm{max2}$ **then** $\mathrm{maxA} := \mathrm{max1}$ **else** $\mathrm{maxA} := \mathrm{max2}$;
  if $\mathrm{min1} < \mathrm{min2}$ **then** $\mathrm{minA} := \mathrm{min1}$ **else** $\mathrm{minA} := \mathrm{min2}$;

Matematine indukcija pagal k įrodysime, kad šios funkcijos sudėtingumas $L(n) = \frac{3}{2}n - 2$. Kai $k = 1$, turime $n = 2$. Kadangi funkcija maxmin kai $n = 2$ daro tik 1 palyginimą, tai šiuo atveju teiginys teisingas: $1 = \frac{3}{2} \cdot 2 - 2$. Tarkime, kad teiginys teisingas kai $l = k - 1$, t.y. $L(n/2) = \frac{3}{2}(n/2) - 2$, ir įrodysime jį kai $l = k$. Iš algoritmo matyti, kad kai $n > 2$, uždavinys suskyla į du tokius pat uždavinius dydžio $n/2$, o juos išsprendus dar reikia atlikti du palyginimus, norint gauti maxA ir minA. Todėl

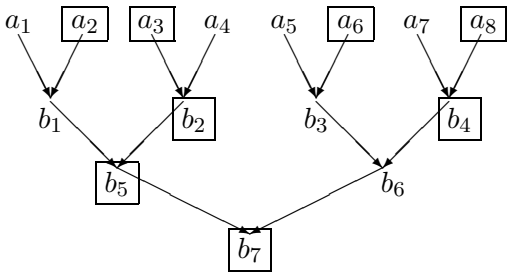
$$L(n) = 2L\left(\frac{n}{2}\right) + 2 = 2\left(\frac{3}{2}\left(\frac{n}{2}\right) - 2\right) + 2 = \frac{3n}{2} - 2.$$

Šiam uždaviniui buvo įrodyta, kad bet kokiam n $L^{\mathrm{MAXMIN}}(n) = \lceil \frac{3}{2}n \rceil - 2$, t.y., apatinis įvertis sutampa su viršutiniu. Tik mažai daliai uždavinių pavyksta rasti tikslius sudėtingumo įverčius.

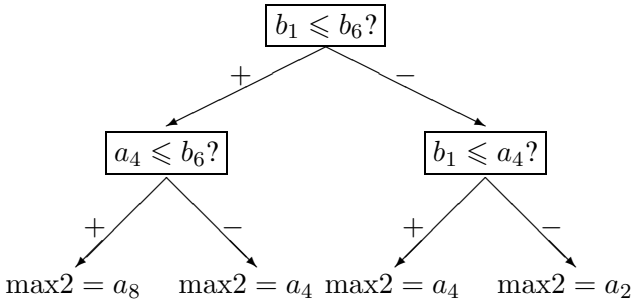
1.7.2 Dviejų didžiausių aibės elementų paieška

Duota aibė $A = \{a_1, a_2, \ldots, a_n\}$, kurioje apibrėžtas pilnos tvarkos sąryšis $\leqslant$. Reikia rasti du didžiausius tos aibės elementus $\mathrm{max1}$ ir $\mathrm{max2}$ (uždavinys $\mathrm{MAX2}$). Nuosekliai ieškodami iš pradžių didžiausio aibės elemento, o po to didžiausio iš likusių elementų, gauname trivialų viršutinį šio uždavinio sudėtingumo įvertį $L^{\mathrm{MAX2}}(n) \leqslant 2n - 3$. Kai $n = 2^k$, įrodysime, kad šį įvertį galima pagerinti iki $L^{\mathrm{MAX2}}(n) \leqslant n + \log_2 n - 2$.

Vietoje to, kad nuosekliai lyginti visus aibės elementus su didžiausiu rastu elementu, konstruojame palyginimų medį, t.y., 1-ame lygyje lyginame a_1 su a_2, a_3 su a_4, …, a_{n-1} su a_n, po to rastus didžiausius elementus 2-ame lygyje vėl lygindami poromis, randame didžiausią iš pirmo ketverto



Pav. 1.7: Palyginimų medis gaunamas ieškant $\max 1$ 8 elementų atveju.



Pav. 1.8: Palyginimų medis gaunamas ieškant $\max 2$ 8 elementų atveju.

a_1, a_2, a_3, a_4, didžiausią iš antro ketverto ir t.t. Galų gale lygyje k palyginę $\max(a_1, \ldots, a_{n/2})$ ir $\max(a_{n/2+1}, \ldots, a_n)$, rasime didžiausią aibės A elementą $\max 1$. Jam rasti mums prireikė

$$\frac{n}{2} + \frac{n}{4} + \cdots + 2 + 1 = \frac{1 - 2^k}{1 - 2} = n - 1$$

palyginimų. Taigi, kol kas mes nieko neišlošėme palyginus su nuoseklia paieška. Tačiau gautas medis turi informacijos, kurią galima panaudoti, norint paprasčiau rasti antrą pagal dydį aibės A elementą.

Panagrinėkime medį, gautą aibės iš 8 elementų atveju (žr. 1.7 pav.). Kiekvienos lygintos poros didesnis elementas medyje pažymėtas rėmeliu. Iš šio medžio matyti, kad $\max 1 = a_3$. Kadangi b_6 yra didžiausias elementas dešiniajame pomedyje, o kairiajame pomedyje paskutinis už b_7 mažesnis elementas buvo b_1, tai ieškant $\max 2$ pirmuoju žingsniu reikia palyginti b_1 ir b_6. Jei b_6 bus didesnis, tai $\max 2$ nebegali būti pomedyje, kurio šaknis yra b_1, tačiau jis dar gal i būti pomedyje su šaknimi b_2 ir antruoju žingsniu lyginame a_4 ir b_6. Jei pirmajame žingsnyje didesnis bus b_1, tai $\max 2$ nebegali būti pomedyje su šaknimi b_6, ir lieka palyginti a_4 ir b_1. Taigi, pradėję $\max 2$ paiešką nuo $k - 2$ lygio, su kiekvienu žingsniu mes pakylame 1 lygiu aukštyn. Vadinasi, po $k - 2$ žingsnių mes pakilsime iki 0 lygio, t.y., elementų a_i, ir atlikę dar 1 palyginimą, rasime $\max 2$. Pav. 1.8 matome palyginimų medį, gaunamą ieškant $\max 2$ atveju $n = 8$. Gauname, kad

bendras algoritmo sudėtingumas yra

$$L(n) = n - 1 + k - 1 = n + \log_2 n - 2.$$

1.8 Apatiniai įverčiai ir rūšiavimo uždavinys

Norint gauti uždavinių klasės $\mathcal{U}$ sudėtingumo apatinį įvertį $L^{\mathcal{U}}(n) \geqslant f(n)$, reikia įrodyti, kad kiekvienas algoritmas, spręsdamas uždavinį $U \in \mathcal{U}$ dydžio n, darys ne mažiau kaip $f(n)$ žingsnių. Nesunku gauti aukštus apatinius įverčius tokiems uždaviniams, kurių pats sprendinys yra didelis. Dažniausiai tai yra įvairių kombinatorinių objektų generavimo ar paieškos uždaviniai, pavyzdžiui: (1) generuoti visus kėlinius ilgio n, (2) rasti visus duoto grafo karkasus, (3) rasti visas duoto grafo klikas. Akivaizdu, kad jei algoritmo išėjimas yra $f(n)$ skirtingų objektų, tai kadangi tie objektai yra skirtingi, tai kiekvienam iš jų reikia bent vieno algoritmo žingsnio, kuris nesutaps su kitais algoritmo žingsniais. Taigi, rezultatų kiekis yra trivialus apatinis įvertis algoritmo sudėtingumui. Pavyzdžiui, visų skirtingų kėlinių ilgio n generavimo uždavinio sudėtingumas yra $L(n) = \Theta(n!)$.[10] Viršutinis įvertis išplaukia iš to, kad nesunku nurodyti algoritmą, kuris generuoja visus kėlinius, pradėdamas nuo kėlinio $12 \ldots n$ ir kiekvieną kartą keisdamas vietomis tik 2 anksčiau gauto kėlinio elementus. Apatinis įvertis gaunamas, naudojantis tuo, kad rezultatų kiekis turi būti $n!$.

Deja, daugumos uždavinių sprendinys yra vienas ar keli skaičiai. Tokiems uždaviniams būna labai sunku gauti gerus apatinius įverčius, t.y., tokius apatinius įverčius, kurie būtų artimi viršutiniams. Netrivialūs apatiniai įverčiai buvo gauti tik nedaugeliui uždavinių. Kadangi algoritmo sąvoka yra neformali, norint gauti apatinį uždavinių klasės sudėtingumo įvertį, reikia pirmiausia formalizuoti algoritmus, t.y., griežtai apibrėžti klasę algoritmų, kuriuos taikysime pasirinktam uždaviniui. Pademonstruosime, kaip galima gauti tikslų apatinį įvertį (t.y., sutampantį su viršutiniu) rūšiavimo uždaviniui SORT.

Taigi, duotas objektų sąrašas $A = \{a_1, \ldots, a_n\}$, kuriame apibrėžtas pilnos tvarkos sąryšis $\leqslant$. Reikia duoto sąrašo elementus išdėstyti nemažėjančia tvarka: $A' = \{a_{i_1} \leqslant a_{i_2} \leqslant \cdots \leqslant a_{i_n}\}$ (žr. 1.6 skyrelį). Nagrinėsime tik tokius rūšiavimo algoritmus, kuriuos galima pavaizduoti *palyginimų medžiu*, t.y., mes kažkuriame algoritmo žingsnyje lyginame tarpusavyje du pasirinktus pradinio sąrašo elementus, po to, priklausomai nuo atsakymo, kuris iš tų elementų buvo didesnis, mes vėl lyginame du sąrašo elementus ir t.t. (žr. 1.9 pav.). Kadangi bendras rūšiavimo algoritmo sudėtingumas būna proporcingas tokių palyginimų skaičiui, tai mes skaičiuosime tik palyginimus. Taigi, $L_A^{\mathrm{SORT}}(n)$ reikš rūšiavimo algoritmo A atliktų palyginimų skaičių blogiausiu atveju, kai rūšiuojamų objektų skaičius yra n. Įrodysime, kad $L^{\mathrm{SORT}}(n) = \Theta(n \log_2 n)$, t.y., rūšiavimo uždavinio sudėtingumo viršutinis ir apatinis įverčiai skiriasi nuo $n \log_2 n$ tik pastoviu daugikliu.

[10] $f(n) = \Theta(g(n))$, jei $f(n) = \mathrm{O}(g(n))$ ir $f(n) = \Omega(g(n))$.

1.8.1 Viršutinis rūšiavimo uždavinio sudėtingumo įvertis

Taikysime rūšiavimą sąlaja (MERGE_SORT). Tai rekursyvus algoritmas, naudojantis metodą "skaldyk ir valdyk". Tarkime, kad $n = 2^k$. Pradinį uždavinį $\mathrm{SORT}(A)$ suskaidome į du perpus mažesnius $\mathrm{SORT}(\{a_1, \ldots, a_{n/2}\})$ ir $\mathrm{SORT}(\{a_{n/2+1}, \ldots, a_n\})$, juos išsprendžiame, o po to du jau surūšiuotus masyvus suliejame į vieną surūšiuotą masyvą A'.

Pirmiausia pateikiame sąlajos algoritmą MERGE, kuris du surūšiuotus masyvus A ir B ilgio m ir n, atitinkamai, sulieja į naują masyvą C ilgio $m + n$.

function $C = \mathrm{MERGE}(A, B)$
$A[m + 1] := \infty$;
$B[n + 1] := \infty$;
$i := 1$;
$j := 1$;
for $k := 1$ **to** $m + n$ **do**
  if $A[i] < B[j]$ **then**
    $C[k] := A[i]$;
    $i := i + 1$;
  else
    $C[k] := B[j]$;
    $j := j + 1$;

Akivaizdu, kad algoritmo MERGE sudėtingumas yra $\mathrm{O}(m + n)$. Pagrindinis algoritmas atrodo taip:

function $A' = \mathrm{MERGE\_SORT}(A)$
if $n = 1$ **then** $A' = A$
else $A' := \mathrm{MERGE}(\mathrm{MERGE\_SORT}(A[1 : n/2]), \mathrm{MERGE\_SORT}(A[n/2 + 1 : n]))$;

Kaip šis algoritmas veikia pademonstruosime pavyzdžiu. Tarkime, $A = \{9, 1, 5, 4, 3, 7, 6, 2\}$. Po 3 rekursijos žingsnių šis masyvas bus suskaidytas į 8 masyvus ilgio 1, kurie grįžtant į aukštesnius rekursijos lygius bus palaipsniui suliejami į didesnius surūšiuotus masyvus:

$$\begin{aligned} A = \{9, 1, 5, 4, 3, 7, 6, 2\} &\to \{9, 1, 5, 4\}\{3, 7, 6, 2\} \\ &\to \{9, 1\}\{5, 4\}\{3, 7\}\{6, 2\} \\ &\to \{9\}\{1\}\{5\}\{4\}\{3\}\{7\}\{6\}\{2\} \\ &\to \{1, 9\}\{4, 5\}\{3, 7\}\{2, 6\} \\ &\to \{1, 4, 5, 9\}\{2, 3, 6, 7\} \\ &\to A' = \{1, 2, 3, 4, 5, 6, 7, 9\}. \end{aligned}$$

Algoritmo MERGE_SORT sudėtingumas

$$\begin{aligned} L(n) &\leqslant 2L\left(\frac{n}{2}\right) + cn \\ &= 2\left(2L\left(\frac{n}{4}\right) + c\frac{n}{2}\right) + cn \\ &= 4L\left(\frac{n}{4}\right) + 2cn = \cdots \\ &= 2^k L\left(\frac{n}{2^k}\right) + kcn = cn\log_2 n, \end{aligned}$$

nes $k = \log_2 n$ ir $L(1) = 0$. Taigi, kai n yra dvejeto laipsnis, viršutinį įvertį įrodėme. Jei $n \neq 2^k$, tai $\exists k$: $2^{k-1} < n < 2^k = n'$. Papildę masyvą A bet kokiais objektais, didesniais už patį didžiausią masyvo A objektą, iki ilgio n', galime pritaikyti algoritmą MERGE_SORT šiam ilgesniam masyvui, o kai algoritmas baigs darbą, paimti tik pirmuosius n surūšiuoto masyvo elementų. Kadangi $n' < 2n$, gauname

$$L(n) \leqslant L(n') \leqslant cn'\log_2 n' < 2cn\log_2(2n) < c'n\log_2 n.$$

Viršutinį įvertį įrodėme, tačiau lieka nelabai aišku, kaip sąlajos algoritmą galima būtų vaizduoti palyginimų medžiu. Pav. 1.9 pateikiame tokio medžio fragmentą tuo atveju, kai $n = 4$. Taigi, algoritmas MERGE_SORT priklauso nagrinėjamų algoritmų klasei.

1.8.2 Apatinis rūšiavimo uždavinio sudėtingumo įvertis

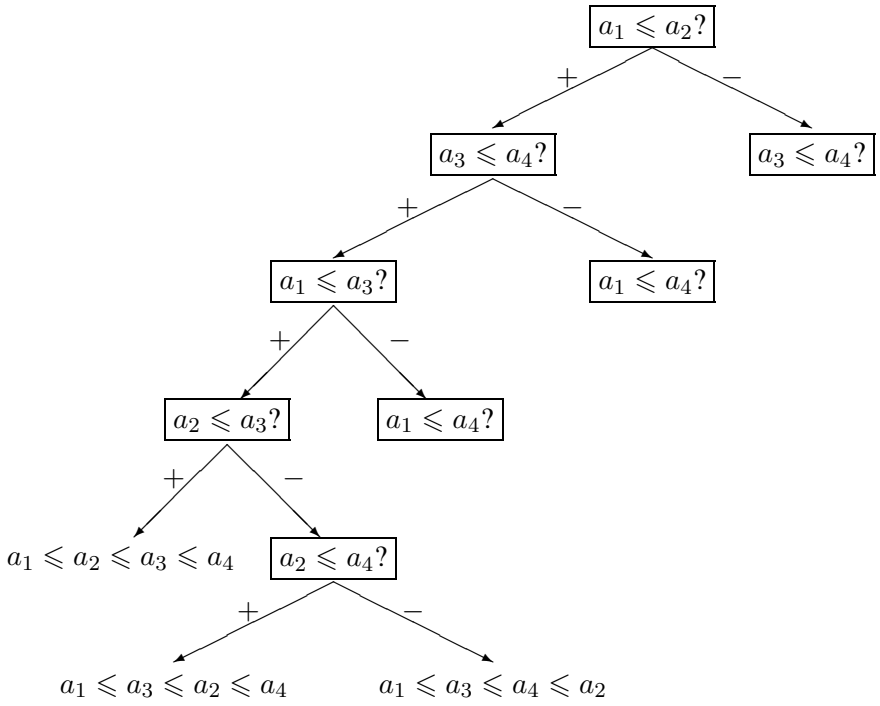
Sąrašą ilgio n galima sutvarkyti $n!$ skirtingų būdų. Tai reiškia, kad bet kuris palyginimų medis privalo turėti ne mažiau kaip $n!$ lapų (lapais vadiname medžio viršūnes, iš kurių neišeina nė vienas lankas). Jei lapų būtų mažiau, tada galima būtų parinkti du skirtingus skaičių $\{1, 2, \ldots, n\}$ kėlinius, kurie atvestų į tą patį lapą, t.y., jiems algoritmo atsakymas sutaptų. Tokiu atveju vienas iš tų kėlinių būtų surūšiuotas klaidingai. Primename, kad medžio aukščiu vadiname vidinių viršūnių (t.y., ne lapų) skaičių ilgiausioje jo šakoje. Kadangi konkretiems pradiniams duomenims rūšiavimo algoritmas praeina lygiai vieną medžio šaką, tai jo sudėtingumas (atliktų palyginimų skaičius blogiausiu atveju) sutampa su medžio aukščiu. Palyginimų medžiai yra binarieji medžiai, todėl palyginimų medis aukščio h gali turėti ne daugiau lapų, negu jų turės pilnas binarusis medis aukščio h, o toks medis turi 2^h lapų.

Tarkime, A yra bet kuris rūšiavimo algoritmas, kurį galima pavaizduoti palyginimų medžiu. Iš aukščiau pateiktų samprotavimų išplaukia, kad algoritmo A sudėtingumas $L(n)$ turi tenkinti nelygybę

$$2^{L(n)} \geqslant n!.$$

Pasinaudoję iš Stirlingo formulės gaunamu įverčiu

$$n! \geqslant \sqrt{2\pi n}\left(\frac{n}{\mathrm{e}}\right)^n,$$



Pav. 1.9: Palyginimų medžio, vaizduojančio MERGE_SORT algoritmo veikimą, fragmentas.

gauname

$$L(n) \geqslant \log_2 \left(\sqrt{2\pi n} \left(\frac{n}{\mathrm{e}} \right)^n \right) = n \log_2 n + \frac{1}{2} \log_2 (2\pi n) - n \log_2 \mathrm{e} \sim n \log_2 n.$$

Matome, kad rūšiavimo uždavinio sudėtingumo viršutinis ir apatinis įverčiai skiriasi tik pastoviu daugikliu.

1.9 Funkcijų augimo greičiai ir kombinatorinis sprogimas

Sudėtingumas $L(n)$ yra funkcija $L : \mathbb{N} \to \mathbb{N}$. Kai uždavinys yra nedidelis, pavyzdžiui, $n \leqslant 10$, net ir eksponentinio sudėtingumo algoritmai baigia darbą labai greitai. Tačiau situacija visiškai pasikeičia, kai uždavinio dydis n auga. Kai $n \geqslant 50$, daug uždavinių, kuriems nežinomi polinominio sudėtingumo algoritmai praktiškai jau tampa sunkiai įveikiami. Todėl labai svarbu žinoti kaip algoritmų ir uždavinių sudėtingumas $L(n)$ elgiasi asimptotiškai, t.y., kai $n \to \infty$. Šiame skyrelyje pateiksime keletą apibrėžimų iš funkcijų teorijos, kurie dažnai naudojami algoritmų analizėje. Kai kuriuos iš čia apibrėžtų žymėjimų mes jau naudojome ankstesniuose skyreliuose.

Tegu $f, g : \mathbb{N} \to \mathbb{R}^+$. Žymėsime:

- $f(n) = \mathrm{O}(g(n))$ (arba $f(n) \preccurlyeq g(n)$) ir sakysime, kad "f asimptotiškai yra ne aukštesnės eilės dydis kaip g", jei $\exists N \in \mathbb{N}$ ir $\exists c > 0$: $f(n) \leqslant cg(n)\ \forall n \geq N$;
- $f(n) = \Omega(g(n))$ (arba $f(n) \succcurlyeq g(n)$) ir sakysime, kad "f asimptotiškai yra ne žemesnės eilės dydis kaip g", jei $\exists N \in \mathbb{N}$ ir $\exists c > 0$: $f(n) \geqslant cg(n)\ \forall n \geq N$; akivaizdu, kad jei $f(n) = \mathrm{O}(g(n))$, tai $g(n) = \Omega(f(n))$;
- $f(n) = \Theta(g(n))$ (arba $f(n) \asymp g(n)$) ir sakysime, kad "f ir g asimptotiškai yra tokios pat eilės dydžiai", jei $f(n) = \mathrm{O}(g(n))$ ir $f(n) = \Omega(g(n))$;
- $f(n) = \mathrm{o}(g(n))$ (arba $f(n) \prec g(n)$) ir sakysime, kad "f asimptotiškai yra žemesnės eilės dydis už g", jei
$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0;$$
- $f(n) \lesssim g(n)$ ir sakysime, kad "f yra asimptotiškai mažesnė arba lygi g", jei
$$\lim_{n\to\infty} \frac{f(n)}{g(n)} \leq 1;$$
- $f(n) \sim g(n)$ ir sakysime, kad "f yra asimptotiškai lygi g", jei $f(n) \lesssim g(n)$ ir $g(n) \lesssim f(n)$.

Pavyzdys.

(i) $1000n^2 + 1000000n \log_2 n = \Theta(n^2)$ ir $1000n^2 + 1000000n \log_2 n = \mathrm{o}(n^3)$,

(ii) $n^{100} = \mathrm{o}(1.1^n)$,

(iii) $10^n = \mathrm{o}(n!)$,

(iv) $1^2 + 2^2 + \cdots + n^2 = \Theta(n^3)$, nes
$$1^2 + 2^2 + \cdots + n^2 > \left(\frac{n}{2} + 1\right)^2 + \cdots + n^2 > \left(\frac{n}{2}\right)^2 \cdot \frac{n}{2} = \frac{n^3}{8}$$
ir $1^2 + 2^2 + \cdots + n^2 < n \cdot n^2 = n^3$.

Kai sudėtingumas $L(n)$ yra ne aukštesnės eilės nei kai kurios dažniau pasitaikančios algoritmų analizėje funkcijos, tokiam sudėtingumui apibūdinti yra naudojami specialūs terminai. Keletą tokių terminų čia ir išvardinsime (sudėtingumo didėjimo tvarka):

- jei $L(n) = \mathrm{O}(\log_2 n)$, tai sudėtingumas vadinamas *logaritminiu*;

Uždavinio dydis n	Algoritmo sudėtingumas					
	$\log_2 n$	n	n^2	n^3	2^n	$n!$
10	3×10^{-9} s	10^{-8} s	10^{-7} s	10^{-6} s	10^{-6} s	3×10^{-3} s
20	4.5×10^{-9} s	2×10^{-8} s	4×10^{-7} s	8×10^{-6} s	10^{-3} s	77 metai
100	7×10^{-9} s	10^{-7} s	10^{-5} s	10^{-3} s	4×10^{13} metų	$*$
1000	10^{-8} s	10^{-6} s	10^{-3} s	1 s	$*$	$*$
1000000	2×10^{-8} s	10^{-3} s	17 min.	32 metai	$*$	$*$

Lentelė 1.1: Kompiuterinio laiko lentelė įvairaus dydžio ir sudėtingumo uždaviniams.

- jei $L(n) = \mathrm{O}(n)$, tai sudėtingumas vadinamas *tiesiniu*;
- jei $L(n) = \mathrm{O}(n^2)$, tai sudėtingumas vadinamas *kvadratiniu*;
- jei $L(n) = \mathrm{O}(n^3)$, tai sudėtingumas vadinamas *kubiniu*;
- jei egzistuoja $k \geqslant 1$: $L(n) = \mathrm{O}(n^k)$, tai sudėtingumas vadinamas *polinominiu*;
- jei egzistuoja $a > 1$: $L(n) = \mathrm{O}(a^n)$, tai sudėtingumas vadinamas *eksponentiniu*.

Kai kuriems uždaviniams spręsti nėra žinoma jokių geresnių algoritmų už visų galimų variantų perrinkimą (brutalios jėgos algoritmą). Jei didėjant uždaviniui variantų skaičius auga greičiau už bet kokį polinomą (pavyzdžiui, eksponentiškai), tai tokį reiškinį vadina *kombinatoriniu sprogimu*. Taip auga, pavyzdžiui, Fibonacci skaičiai (žr. 2.2 skyrelį), kėlinių ilgio n skaičius, Hamiltono ciklų skaičius pilname grafe, pilno grafo klikų skaičius, ir daugelio kitokių kombinatorinių objektų kiekis. Lentelė 1.1 parodo, kad kombinatorinio sprogimo negalės įveikti patys greičiausi kompiuteriai, kiek bedidėtų jų greitis ateityje. Šioje lentelėje pateikiame CPU laiką įvairaus sudėtingumo algoritmams ir įvairaus dydžio uždaviniams, darant prielaidą, kad kompiuteris vykdo 10^9 (t.y., 1 milijardą) operacijų per sekundę. Žvaigždutė žymi laiką ilgesnį nei 10^{100} metų. Iš šios lentelės matyti, kad tobulesnių kompiuterių sukūrimas gali padėti įveikti tik tuos atvejus, kai šiuo metu laikas yra lygus 32 ir 77 metams. Visais atvejais, kurie lentelėje pažymėti žvaigždute, gali padėti tik greitesnių algoritmų sukūrimas arba apytikslių algoritmų naudojimas vietoje tikslių.