3. Kombinatorika

3.1. Įvadas

Ne taip paprasta pateikti išsamų kombinatorikos apibrėžimą. Kažkuria prasme žodį "kombinatorika" galima suprasti kaip "diskrečiosios matematikos" sinonimą, t.y. mokslą, tyrinėjantį diskrečiasias baigtines matematines struktūras. Šia prasme žodis "kombinatorika" atsispindi žymaus lenkų matematiko Vytoldo Lipskio knygos "Witold Lipski. Kombinatorika dla programistow. Wydawnictva Naukowo – Techniczne. Warszava, 1982" (yra vertimas į rusų kalbą žr. [Lip88]) pavadinime.

Paprastai kombinatorika suprantama kaip matematikos sritis, kurioje tiriami klausimai, kiek skirtingų kombinacijų, tenkinančių vienokias ar kitokias sąlygas, galima sudaryti iš turimų objektų.

Kombinatorika atsirado 16 amžiuje ir buvo susijusi su azartiniais lošimais. Buvo tiriama, keliais būdais galima išmesti kokį nors akių skaičių, lošiant dviem ar trim kauliukais, arba keliais būdais galima gauti, pavyzdžiui, du karalius, lošiant kortomis. 17 amžiuje prancūzų matematikai Paskalis ir Ferma nagrinėjo statytos sumos padalijimo uždavinį. Problema buvo tokia: lošimas turėjo vykti iki t išloštų partijų, bet buvo nutrauktas, kai vienam žaidėjui iki išlošimo lieka r partijų, o antrajam – s partijų. Kaip žaidėjai turi pasidalyti statytąją sumą?

Tolesnė kombinatorikos raida susijusi su Jakobo Bernulio, Leibnico ir Oilerio vardais.

Pastaraisiais metai kombinatorikos vystymasis glaudžiai siejasi su optimizavimo uždavinų sprendimu, šifravimo ir dekodavimo uždavinias bei kitomis informacijos teorijos problemomis.

3.2. Bendrieji kombinatorikos dėsniai

Sprendžiant įvairius kombinatorikos uždavinius, dažniausiai naudojami ***sumavimo***, ***sandaugos***, ***priskirties*** ir ***išskirties*** dėsniai.

Sumavimo dėsnis

Jei kokiam nors objektui A pasirinkti yra m būdų, o kitam objektui B pasirinkti yra n būdų, tai pasirinkti "arba A, arba B" yra $m + n$ būdų.

Pastaba. Taikant taip suformuluotą sumavimo dėsnį, reikia žiūrėti, kad nei vienas objekto A pasirinkimo būdas nesutaptų su kuriuo nors objekto B pasirinkimo būdu. Jei tokių sutapimų yra k, tai susidaro $m + n - k$ pasirinkimo būdų.

Pavyzdys. Pintinėje yra 12 obuolių ir 10 apelsinų. Jonukas pasirenka arba obuolį, arba apelsiną, o paskui Onutė ima ir obuolį, ir apelsiną. Keliais būdais Jonukas gali paimti vaisių ir kada Onutei didesnė pasirinkimo laisvė: ar kai Jonukas paima obuolį, ar kai apelsiną?

Aišku, kad Jonukas obuolį gali pasirinkti 12 skirtingų būdų, o apelsiną – 10 skirtingų būdų. Vadinasi, pasirinkti arba obuolį, arba apelsiną galima 12 + 10 = 22 būdais.

Į antrą klausimo dalį atsakysime po sandaugos dėsnio.

Sandaugos dėsnis

Jei objektui A pasirinkti yra m būdų, o po kiekvieno tokio pasirinkimo objektą B galima pasirinkti n būdais, tai porą (*A, B*) *pasirinkti nurodytąja tvarka galima m · n būdais.*

Pavyzdys. Panagrinėkime aukščiau pateiktą pavyzdį. Tarkime, Jonukas pasirinko obuolį. Tada Onutei galimų pasirinkimo būdų yra 11 · 10. Jei Jonukas būtų pasirinkęs apelsiną, tai Onutei paimti ir obuolį, ir apelsiną būtų galima 12 · 9 būdais. Kadangi 11 · 10 > 12 · 9, tai Onutei didesnė pasirinkimo laisvė bus tada, jei Jonukas pasirinks obuolį.

Priskirties ir išskirties dėsnis

Sakykime, kad kai kuriems iš N turimų daiktų būdingos savybės $\alpha_1, \alpha_2, ..., \alpha_n$. Simboliu $N(\alpha_i, \alpha_j, ..., \alpha_k)$ pažymėkime skaičių daiktų, turinčių savybes $\alpha_i, \alpha_j, ..., \alpha_k$ (į kitas tų daiktų savybes nekreipiame dėmesio). Jei reikės pabrėžti, kad imami tik tie daiktai, kurie neturi kurios nors savybės, tai tą savybę rašysime su brūkšneliu. Pavyzdžiui, $N(\alpha_1, \alpha_2, \alpha_4')$ žymi skaičių daiktų, turinčių savybes α_1 ir α_2, bet neturinčių savybės α_4 (į kitas jų savybes nekreipiame dėmesio).

Tada priskirties ir išskirties dėsnis bus užrašomas taip:

$$\begin{aligned} N(\alpha_1', \alpha_2', ..., \alpha_n') &= N - N(\alpha_1) - N(\alpha_2) - ... - N(\alpha_n) + \\ &+ N(\alpha_1, \alpha_2) + N(\alpha_1, \alpha_3) + ... + N(\alpha_1, \alpha_n) + ... + N(\alpha_{n-1}, \alpha_n) - \\ &- N(\alpha_1, \alpha_2, \alpha_3) - ... - N(\alpha_{n-2}, \alpha_{n-1}, \alpha_n) + ... + (-1)^n N(\alpha_1, \alpha_2, ..., \alpha_n). \end{aligned} \quad (3.2.1)$$

Įrodymas. Šį dėsnį įrodysime pilnosios matematinės indukcijos metodu.

Dėsnis teisingas, kai kiekvienas daiktas turi savybę α. Tada $N(\alpha') = N - N(\alpha)$.

Tarkime, kad dėsnis teisingas, kai savybių skaičius lygus $n-1$:

$$N(\alpha_1', \alpha_2', ..., \alpha_{n-1}') = N - N(\alpha_1) - N(\alpha_2) - ... - N(\alpha_{n-1}) +$$
$$+ N(\alpha_1, \alpha_2) + ... + N(\alpha_{n-2}, \alpha_{n-1}) - N(\alpha_1, \alpha_2, \alpha_3) - ... - \quad (3.2.2)$$
$$- N(\alpha_{n-3}, \alpha_{n-2}, \alpha_{n-1}) + ... + (-1)^{n-1} N(\alpha_1, \alpha_2, ..., \alpha_{n-1}).$$

Įrodysime, kad dėsnis teisingas, kai savybių skaičius yra *n*.

(3.2.2) formulę taikykime daiktų $N(\alpha_n)$ aibei. Gausime:

$$N(\alpha_1', \alpha_2', ..., \alpha_{n-1}', \alpha_n) = N(\alpha_n) - N(\alpha_1, \alpha_n) - N(\alpha_{n-1}, \alpha_n) +$$
$$+ N(\alpha_1, \alpha_2, \alpha_n) + ... + N(\alpha_{n-2}, \alpha_{n-1}, \alpha_n) - N(\alpha_1, \alpha_2, \alpha_3, \alpha_n) - ... - \quad (3.2.3)$$
$$- N(\alpha_{n-3}, \alpha_{n-2}, \alpha_{n-1}, \alpha_n) + ... + (-1)^{n-1} N(\alpha_1, \alpha_2, ..., \alpha_n).$$

Iš (3.2.2) lygybės atėmę (3.2.3), gausime

$$N(\alpha_1', \alpha_2', ..., \alpha_{n-1}') - N(\alpha_1', \alpha_2', ..., \alpha_{n-1}', \alpha_n) =$$
$$N - N(\alpha_1) - N(\alpha_2) - ... - N(\alpha_n) + N(\alpha_1, \alpha_2) +$$
$$+ N(\alpha_1, \alpha_3) + ... + N(\alpha_{n-1}, \alpha_n) - N(\alpha_1, \alpha_2, \alpha_3) - \quad (3.2.4)$$
$$- N(\alpha_1, \alpha_2, \alpha_4) - ... - N(\alpha_{n-2}, \alpha_{n-1}, \alpha_n) + ... +$$
$$+ (-1)^{n-1} N(\alpha_1, \alpha_2, ..., \alpha_n).$$

(3.2.4) formulės kairiosios pusės skirtumas $N(\alpha_1', \alpha_2', ..., \alpha_{n-1}') - N(\alpha_1', \alpha_2', ..., \alpha_{n-1}', \alpha_n)$ yra lygus $N(\alpha_1', \alpha_2', ..., \alpha_{n-1}', \alpha_n')$, nes iš skaičiaus daiktų, kurie neturi savybių $\alpha_1, \alpha_2, ..., \alpha_{n-1}$ atimame skaičių daiktų, neturinčių tų pačių savybių ir turinčių savybę α_n. Vadinasi, atėmus liks skaičus daiktų, neturinčių savybių $\alpha_1, \alpha_2, ..., \alpha_n$.

Pavyzdys. Pasinaudodami priskirties ir išskirties formule, apskaičuokime, kiek tarp natūraliųjų skaičių nuo vieneto iki 100 yra skaičių, kurie nesidalo nei iš 2-jų, nei iš 3-jų, nei iš penkių.

Tarkime, α_1 – tai skaičiaus savybė dalintis iš 2-jų, α_2 – tai skaičiaus savybė dalintis iš 3-jų, o α_3 – tai skaičiaus savybė dalintis iš 5-ių. Tada

$$N(\alpha_1', \alpha_2', \alpha_3') = N - N(\alpha_1) - N(\alpha_2) - N(\alpha_3) +$$
$$+ N(\alpha_1, \alpha_2) + N(\alpha_1, \alpha_3) + N(\alpha_2, \alpha_3) - N(\alpha_1, \alpha_2, \alpha_3).$$

Norint sužinoti, kiek skaičių nuo 1 iki *N* dalijasi iš *n*, reikia *N* padalinti iš *n* ir imti gautojo dalmens sveikąją dalį į mažesniąją pusę. Todėl

$$N(\alpha_1) = \left\lfloor \frac{100}{2} \right\rfloor = 50, \qquad N(\alpha_2) = \left\lfloor \frac{100}{3} \right\rfloor = 33, \qquad N(\alpha_3) = \left\lfloor \frac{100}{5} \right\rfloor = 20,$$

$N(\alpha_1,\alpha_2)=\left\lfloor\frac{100}{6}\right\rfloor=16$, $N(\alpha_1,\alpha_3)=\left\lfloor\frac{100}{10}\right\rfloor=10$, $N(\alpha_2,\alpha_3)=\left\lfloor\frac{100}{15}\right\rfloor=6$,

$N(\alpha_1,\alpha_2,\alpha_3)=\left\lfloor\frac{100}{30}\right\rfloor=3$.

Vadinasi, $N(\alpha_1',\alpha_2',\alpha_3')=100-50-33-20+16+10+6-3=26$.

3.3. Junginiai

Šiame paragrafe aptarsime pagrindinius junginius ir jų savybes.

3.3.1. Gretiniai be pasikartojimų

Pirmiausia aptarsime gretinių be pasikartojančių elementų uždavinį.

Turime n skirtingų daiktų. Kiek iš jų galima sudaryti k-elemenčių junginių, sudarytų iš skirtingų elementų, kai junginys nuo junginio skiriasi arba bent vienu elementu, arba jų tvarka?

Tokie junginiai vadinami ***gretiniais be pasikartojimų*** ir žymimi simboliu A_n^k.

Nesunku parodyti, kad $A_n^k=n(n-1)\ldots(n-k+1)$. Šios formulės teisingumas gali būti pagrįstas taip. Sudarant *k*-elementį junginį, pirmąjį junginio elementą galime pasirinkti *n* skirtingais būdais; antrąjį – $(n-1)$ skirtingais būdais, ir t.t. *k*-tąjį junginio elementą galima pasirinkti $(n-k+1)$ skirtingais būdais. Tada, pritaikius sandaugos dėsnį, gausime

$$A_n^k=n(n-1)\ldots(n-k+1).$$

Pavyzdys. Keliais skirtingais būdais galima sudaryti trispalvę vėliavą, turint 5 skirtingų spalvų audinius?

Kadangi vėliava nuo vėliavos skiriasi arba spalvų rinkiniu, arba jų tvarka, tai skirtingų vėliavų skaičius yra $A_5^3=5\cdot4\cdot3=60$.

3.3.2. Gretiniai su pasikartojimais

Turime n skirtingų rūšių daiktų. Kiek iš jų galima sudaryti k-elemenčių junginių, sudarytų iš skirtingų elementų, kai junginys nuo junginio skiriasi arba bent vienu elementu, arba jų tvarka ir elementai junginyje gali kartotis?

Tokie junginiai vadinami ***gretiniais su pasikartojančiais elementais*** ir žymimi $\overline{A_n^k}$.

Sudarant tokį junginį, bet kuri *i*-toji, $1 \le i \le k$, junginio pozicija gali būti užpildyta *n* skirtingais būdais. Vadinasi, $\overline{A_n^k} = n^k$.

Pavyzdys. Kiek skirtingų triženklių skaičių galima parašyti dešimtainėje skaičiavimo sistemoje?

Kadangi šiuo atveju $n = 10$, o $k = 3$, tai $\overline{A_{10}^3} = 10^3 = 1000$. Iš tikro, tai skaičiai 000, 001, …, 998, 999.

3.3.3. Kėliniai be pasikartojimų

Sudarydami gretinius be pasikartojimų iš *n* elementų po *k*, gavome junginius, kurie vienas nuo kito skiriasi ir pačiais elementais, ir jų išsidėstymo tvarka.

Tačiau, jei sudarinėtume junginius iš visų n elementų, tai jie galėtų skirtis vienas nuo kito tik juose esančių elementų tvarka.

Tokie junginiai vadinami ***kėliniais iš n elementų*** arba, trumpiau, ***n-elemenčiais kėliniais***. Juos žymėsime P_n .

Aišku, kad $P_n = A_n^n$. Vadinasi,

$$P_n = n(n-1)...3 \cdot 2 \cdot 1 .$$

Ši sandauga žymima $n!$ (skaitoma: en faktorialas).

Sprendžiant praktinius uždavinius, tenka naudoti 0!. Yra susitarta, kad 0! yra lygus 1. Faktorialą galima nusakyti ir rekurentiškai:

$$n! = n \cdot (n-1)!$$

Remiantis faktorialu, gretinių formulę galima užrašyti ir taip:

$$A_n^k = \frac{n!}{(n-k)!} .$$

Pavyzdys. Susirinkime turi kalbėti 5 žmonės. Keliais būdais galima sudaryti kalbėtojų sąrašą?

Aišku, kad kalbėtojų sąrašų skaičius yra $P_5 = 5! = 120$.

3.3.4. Kėliniai su pasikartojančiais elementais

Turime k skirtingų tipų daiktus. Kiek kėlinų galima sudaryti iš n_1 *pirmojo tipo elementų, iš* n_2 *antrojo tipo elementų, ...,*n_k *k-tojo tipo elementų?*

Kiekvieną kėlinį turi sudaryti $n = n_1 + n_2 + ... + n_k$ elementų. Jei visi elementai būtų skirtingi, tai kėlinių skaičius būtų *n*! Kadangi kai kurie elementai sutampa, gauname mažiau kėlinų. Kad taip yra iš tikrųjų, įsitikinsime, išnagrinėję, pavyzdžiui, kėlinį

$$\underbrace{aa...a}_{n_1}\quad\underbrace{bb...b}_{n_2}\quad ...\quad\underbrace{xx...x}_{n_k},$$

kuriame iš pradžių surašyti visi pirmojo tipo elementai, paskui – visi antrojo tipo elementai, …, pagaliau – visi *k*-tojo tipo elementai.

Pirmojo tipo elementus sukeitinėti vietomis galima $n_1!$ būdų. Kadangi tie elementai yra vienodi, tai tokie perstatinėjimai kėlinio nepakeičia. Taip pat nieko nepakeičia ir $n_2!$ antrojo tipo elementų perstatinėjimų, …, $n_k!$ *k*-tojo tipo elementų perstatinėjimų.

Bet kurio tipo elementus galima sukeitinėti vieną su kitu nepriklausomai nuo visų kitų tipų elementų perstatinėjimo. Todėl kėlinio su pasikartojimais elementus sukeitinėti vieną su kitu vietomis taip, kad kėlinys nepasikeistų, yra $n_1!\cdot n_2!\cdot ...\cdot n_k!$ būdų (taikome sandaugos dėsnį). Vadinasi, skirtingų kėlinių su pasikartojimais skaičius lygus

$$P(n_1, n_2, ..., n_k) = \frac{n!}{n_1! n_2! ... n_k!}.$$

Pavyzdys. Kiek kėlinių galima sudaryti iš žodžio "sakalas" raidžių?

Šiuo atveju turime dvi raides "s", tris raides "a", vieną raidę "k" ir vieną raidę "l"; iš viso septynias raides. Todėl

$$P(2,3,1,1) = \frac{7!}{2!\cdot 3!\cdot 1!\cdot 1!} = 420.$$

3.3.5. Deriniai be pasikartojimų

Visi galimi k-elemenčiai junginiai iš n elementų, kai junginys nuo junginio skiriasi bent vienu elementu, vadinami deriniais ir žymimi C_n^k.

Derinių skaičiaus formulę lengva sudaryti iš aukščiau išvestos gretinių skaičiaus formulės. Iš tikrųjų, sudarę visus *k*-elemenčius gretinius iš *n* elementų, perstatinėkime kiekvieno derinio elementus visais galimais būdais. Taip mes sudarysime visus *k*-elemenčius gretinius iš *n* elementų; be to, kiekvienas gretinys bus gautas tik vieną kartą. Vadinasi,

$$k!\cdot C_n^k = A_n^k.$$

Tada

$$C_n^k = \frac{A_n^k}{k!} = \frac{n!}{k!(n-k)!} = \frac{n(n-1)...(n-k+1)}{k!}.$$

Pavyzdys. Iš 52 kortų komplekto ištraukta 10 kortų. Keliais atvejais ištrauktųjų kortų grupėje bus bent vienas tūzas? Keliais atvejais – tik vienas tūzas? Keliais atvejais – nemažiau kaip du tūzai? Lygiai du tūzai?

Ištraukti 10 kortų yra C_{52}^{10} būdų. Skaičius atvejų, kai nėra nė vieno tūzo, lygus C_{48}^{10}. Todėl yra $C_{52}^{10} - C_{48}^{10}$ atvejų, kai ištraukiamas bent vienas tūzas.

Tik vieną kartą ištraukti tūzą yra $C_4^1 \cdot C_{48}^9$ būdų.

Ištraukti ne mažiau kaip du tūzus yra $C_{52}^{10} - C_{48}^{10} - 4C_{48}^9$ būdų.

Tiksliai du tūzus galima ištraukti $C_4^2 \cdot C_{48}^8$ būdais (du tūzus galima pasirinkti C_4^2 būdais, kitas aštuonias kortas – C_{48}^8 būdais).

3.3.6. Deriniai su pasikartojimais

Turime n skirtingų rūšių daiktų. Kiek skirtingų k-elemenčių junginių iš n skirtingų rūšių daiktų galima sudaryti, jei junginys nuo junginio skiriasi bent vienu elementu ir elementai junginyje gali kartotis?

Pavyzdžiui, konditerijos parduotuvėje yra 4 rūšių pyragaičių: eklerų, smėlinių, napoleonų ir sluoksninių. Keliais būdais galima nusipirkti 7 pyragaičius?

Derinius su pasikartojančiais elementais žymėsime $\overline{C_n^k}$. Apskaičiuoti derinių su pasikartojimais skaičių galima įvairiais būdais. Čia aptarsime du dūdus.

Pirmasis būdas. Kiekvieną *k*-elementį derinį su pasikartojimais (kiekvieną 7-ių pyragaičių rinkinį) užkoduosime taip: pirmiausia rašome tiek vienetukų, kiek pirmos rūšies daiktų yra junginyje, po to rašome 0; toliau rašome tiek vienetukų, kiek antros rūšies daiktų yra junginyje, o po jų rašome 0 ir t.t.; galiausiai rašome tiek vienetukų, kiek *n*-osios rūšies daiktų yra junginyje. Tuo būdu kiekvienas kodas turės *k* vienetukų ir $(n-1)$ nulį. Pavyzdžiui, pyragaičių pirkiniai 5 eklerai ir du napoleonai bei 2 eklerai, 1 smėlinis, 3 napoleonai ir 1 sluoksninis bus atitinkamai užkoduoti taip: 1111100110 ir 1101011101.

Aišku, kad kiekvienas junginys turės vienintelį kodą ir kiekvienas kodas atitiks skirtingą junginį. Vadinasi,

$$\overline{C_n^k} = P(k, n-1) = \frac{(n+k-1)!}{k!(n-1)!} = C_{n+k-1}^k .$$

Antrasis būdas. Tarkime, turime aibę $A = \{1,2,3,...,n\}$ ir *k*-elementį derinį su pasikartojimais $C = (m_1, m_2, ..., m_k)$, sudarytą iš aibės *A* elementų; be to, $m_1 \le m_2 \le ... \le m_k$.

Deriniui *C* priskirkime *k*-elementį derinį be pasikartojančių elementų, sudarytą iš aibės $A' = \{1,2,3,...,n-k+1\}$ tokiu būdu:

$$C' = (m_1, m_2 + 1, ..., m_k + k - 1).$$

Šis priskyrimas yra abipus vienareikšmis:

1) jei $C_1 \neq C_2$, tai ir $C_1' \neq C_2'$, t.y jei $(m_1, m_2, \ldots, m_k) \neq (m_1^*, m_2^*, \ldots, m_k^*)$, tai ir $(m_1, m_2 + 1, \ldots, m_k + k - 1) \neq (m_1^*, m_2^* + 1, \ldots, m_k^* + k - 1)$;
2) jei $(m_1', m_2', ..., m_k')$ yra *k*-elementis derinys be pasikartojimų, sudarytas iš aibės A' elementų, tai jam atitinkantis *k*-elementis derinys su pasikartojimais, sudarytas iš aibės *A* elementų, yra $(m_1', m_2' - 1, ..., m_k' - k + 1)$.

Vadinasi, $\overline{C_n^k} = C_{n+k-1}^k$.

Panagrinėkime aukščiau pateiktą pavyzdį: keliais būdais galima nusipirkti 4-ių rūšių 7-is pyragaičius. Aišku, kad variantų skaičius yra $\overline{C_4^7} = C_{7+4-1}^7 = C_{10}^7 = \frac{10 \cdot 9 \cdot 8}{1 \cdot 2 \cdot 3} = 120$.

3.3.7. Derinių savybės

Skaičiai C_n^k turi daug nuostabių savybių [Kn76]. Žemiau aptarsime pagrindines šių skaičių savybes.

1. ***Simetriškumo savybė***

$$C_n^k = C_n^{n-k}.$$

Ši savybė įrodoma remiantis formule:

$$C_n^k = \frac{n!}{k!(n-k)!}$$

Pritaikius šią formulę skaičiui C_n^{n-k}, gausime:

$$C_n^{n-k} = \frac{n!}{(n-k)!(n-n+k)!} = \frac{n!}{(n-k)!k!}.$$

2. ***Sudėties savybė***

$$C_n^k = C_{n-1}^{k-1} + C_{n-1}^k.$$

Šią savybę galima įrodyti dvejopai.

Pirmasis būdas. Taikykime derinių skaičiaus formulę lygybės kairiajai ir dešiniajai pusėms. Tada $C_n^k = \frac{n!}{k!(n-k)!}$, o

$$C_{n-1}^{k-1}+C_{n-1}^{k}=\frac{(n-1)!}{(k-1)!(n-k)!}+\frac{(n-1)!}{k!(n-1-k)!}=$$
$$=\frac{(n-1)!(k+n-k)}{k!(n-k)!}=\frac{n\cdot(n-1)!}{k!(n-k)!}=\frac{n!}{k!(n-k)!}.$$

Iš čia išplaukia, kad lygybė yra teisinga.

Antrasis būdas. Visus k-elemenčius derinius iš n elementų $a_1, a_2, ..., a_{n-1}, a_n$ suskirstykime į dvi klases. Pirmajai klasei priklausys deriniai, kuriuose yra elementas a_n, o antrajai – deriniai, kuriuose to elemento nėra. Jei iš bet kurio pirmosios klasės derinio išmesime elementą a_n, tai liks $(k-1)$-elementis derinys, sudarytas iš elementų $a_1, a_2, ..., a_{n-1}$. Tokių derinių skaičius lygus C_{n-1}^{k-1}. Antrosios klasės deriniai yra k-elemenčiai deriniai, sudaryti iš $(n-1)$ elementų $a_1, a_2, ..., a_{n-1}$. Todėl jų skaičius lygus C_{n-1}^{k}.

Kadangi kiekvienas k-elementis derinys iš elementų $a_1, a_2, ..., a_n$ priklauso vienai ir tik vienai iš tų dviejų klasių, o visų derinių skaičius yra C_n^k, tai sumavimo savybė yra teisinga.

Sumavimo savybė yra tampriai susijusi su Paskalio trikampiu

1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1
⋮

Jei iš eilės einančias šio trikampio eilutes sunumeruosime skaičiais $0,1,2,..., n$, tai n-oji eilutė bus sudaryta iš skaičių $C_n^0, C_n^1, \ldots, C_n^n$, o šios eilutės k-asis elementas C_n^k, $k \neq 0$, $k \neq n$, bus lygus $C_{n-1}^{k-1}+C_{n-1}^{k}$, t.y. $(n-1)$-osios eilutės narių, esančių nario C_n^k kairėje ir dešinėje, sumai.

3. ***Iškėlimo prieš sklaustus savybė***

$$C_n^k=\frac{n}{k}C_{n-1}^{k-1}.$$

Jos teisingumas išplaukia iš derinių skaičiaus formulės.

$$C_n^k = \frac{n!}{k!(n-k)!} = \frac{n}{k}\frac{(n-1)!}{(k-1)!(n-k)!} = \frac{n}{k}C_{n-1}^{k-1}.$$

1-oji ir 3-ioji savybės įgalina rekurentiškai apskaičiuoti derinių skaičių C_n^k.

Derinių skaičiaus apskaičiavimo algoritmas

Duota: *n*-elementų skaičius (natūralusis skaičius),
k – elementų skaičius derinyje (natūralusis skaičius).

Rasti: $C = C_n^k$.

```
begin
if k > n then begin C := 0; exit; end;
if k > n div 2  then k := n – k;
C := 1;
l := n – k + 1;
t := 1;
for i =1 to k do
    begin
        if  l mod t = 0 then  begin  temp:=l div t;  C:=C*temp; end
                        else  begin  temp:=C div t;  C:=l*temp; end;
        t :=  t+1;
        l := l+1;
    end;
end;
```

4. *Sumavimo savybės*

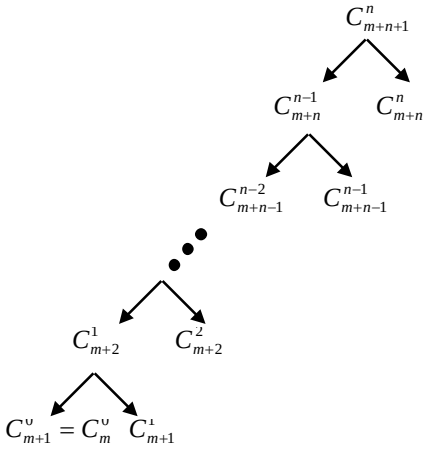
Aptarsime dvi sumavimo savybes, kurių teisingumą galima įrodyti remiantis sudėties savybe:

a) $\sum_{k=0}^{n} C_{m+k}^k = C_m^0 + C_{m+1}^1 + ... + C_{m+n}^n = C_{m+n+1}^n$,

b) $\sum_{k=0}^{n} C_k^m = C_0^m + C_1^m + ... + C_n^m = C_{n+1}^{m+1}$,

čia *n* ir *m* – natūralieji skaičiai.

Pavyzdžiui, savybės a) teisingumą parodo žemiau pateikta schema, gauta nuosekliai taikant sumavimo savybę.



Savybė b) naudojama natūraliųjų skaičių laipsnių sumoms apskaičiuoti. Pavyzdžiui, kai $m = 1$, tai

$$C_0^1 + C_1^1 + C_2^1 + ... + C_n^1 = 0 + 1 + 2 + ... + n = C_{n+1}^2 = \frac{n(n+1)}{2}.$$

Kai $m = 2$, galima apskaičiuoti natūraliųjų skaičių kvadratų sumą: $1^2 + 2^2 + ... + n^2$.

Nesunku pastebėti, kad $k^2 = 2C_k^2 + C_k^1$. Vadinasi,

$$\sum_{k=0}^{n} k^2 = 2\sum_{k=0}^{n} C_k^2 + \sum_{k=0}^{n} C_k^1 = 2C_{n+1}^3 + C_{n+1}^2 = \frac{1}{3} n(n+1/2)(n+1).$$

Kai $n = 3$, galima apskaičiuoti natūraliųjų skaičių kubų sumą. Tam tikslui k^3 išreikšime tiesiniu derinių dariniu.

Kadangi $C_k^3 = \frac{k(k-1)(k-2)}{6}$, tai $k^3 = 6C_k^3 + 3k^2 + 2k$. Įvertindami, kad $k^2 = 3C_k^2 + C_k^1$, o $k = C_k^1$, gausime $k^3 = 6C_k^3 + 6C_k^2 + C_k^1$.

Vadinasi,

$$\sum_{k=0}^{n} k^3 = 6\sum_{k=0}^{n} C_k^3 + 6\sum_{k=0}^{n} C_k^2 + \sum_{k=0}^{n} C_k^1 =$$

$$= 6C_{n+1}^4 + 6C_{n+1}^3 + C_{n+1}^2 = \frac{n^2(n+1)^2}{4}.$$

Gautas rezultatas rodo, kad

$$\sum_{k=0}^{n} k^3 = \left(\sum_{k=0}^{n} k \right)^2 .$$

5. ***Binominių koeficientų savybė***

$$C_n^0 + C_n^1 + C_n^2 + \ldots + C_n^n = 2^n .$$

Šią savybę galima įrodyti remiantis garsiąja Niutono binomo formule:

$$(a+b)^n = C_n^0 a^n + C_n^1 a^{n-1} b + \ldots + C_n^{n-1} a b^{n-1} + C_n^n b^n .$$

Jei $a = b = 1$, gausime

$$2^n = C_n^0 + C_n^1 + C_n^2 + \ldots + C_n^n .$$

Kitas šios formulės įrodymo būdas būtų toks. Visus *n* skilčių dvejetainius skaičius 00…0, 00…1, …, 11…1 suskirstykime į klases: *k*-ąją klasę sudarykime iš skaičių, kurie sudaryti iš *k* vienetukų ir $n-k$ nulių. Aišku, kad kiekvienas skaičius priklauso tik vienai klasei, o *k*-osios klasės elementų skaičius yra $P(k, n-k) = C_n^k$.Kadangi skaičių yra 2^n, tai $2^n = C_n^0 + C_n^1 + C_n^2 + \ldots + C_n^n$.

3.4. Kombinatorinių objektų generavimo algoritmai

Šiame paragrafe aptarsime pagrindinius kombinatorinių objektų (derinių, kėlinių, aibės išskaidymo ir kt.) generavimo algoritmus.

Optimali tokių objektų generavimo algoritmo struktūra yra:

```
begin
    generuoti pradinį junginį (kombinatorinį objektą);
    while "generavimo sąlyga" do
        begin
            nagrinėti (spausdinti) junginį (kombinatorinį objektą);
            generuoti junginį (kombinatorinį objektą), gretimą išnagrinėtam
            (atspausdintam).
        end;
end;
```

Priklausomai nuo pasirinktų objektų bei jų generavimo tvarkos, šiame algoritme turi būti konkretizuoti sakiniai "generuoti pradinį junginį", "generavimo sąlyga", "generuoti gretimą junginį".

3.4.1. Derinių generavimo algoritmai

Derinių generavimas leksikografine tvarka

Pirmiausia aptersime derinių C_n^k generavimo leksikografine didėjimo tvarka algoritmą. Apie leksikografinę vektorių tvarką kalbėjome 1.3 paragrafe. Čia priminsime, kad jei į derinį žiūrėsime kaip į skaičių, tai deriniams atitinkantys skaičiai bus išdėstyti didėjimo tvarka.

Pavyzdys. Panagrinėkime C_5^3. Deriniai, surašyti leksikografine tvarka, bus:

123, 124, 125, 134, 135, 145, 234, 235, 245, 345.

Derinius C_n^k nuosekliai talpinsime masyve $c[0..k]$ ir šio masyvo $c_1, c_2, ..., c_k$ elementai apibrėš derinį.

Pradinio derinio generavimas. Aišku, kad pradinis derinys generuojamas taip:

for $l := 0$ *to* k *do* $c[l] := l$;

Kaip generuoti patį mažiausią leksikografiškai didesnį derinį nei turimas?

Tarkime, $c_1\, c_2 \ldots c_l \ldots c_k$ – turimas derinys. Leksikografine tvarka pats didžiausias derinys yra: $n-k+1,\ldots,n-1,n$. Vadinasi, $c_l \le n-k+l$, $l=\overline{1,k}$. todėl, norint apskaičiuoti patį mažiausią leksikografiškai didesnį derinį nei nagrinėjamas derinys, elgsimės taip:

1) masyve $c[0..k]$ rasime pirmą iš dešinės elementą c_l, $1 \le l \le k$, tenkinantį sąlygą $c_l < n-k+l$;
2) šį elementą padidinsime vienetu, t.y. $c_l := c_l + 1$;
3) likusius elementus užpildysime iš eilės einančiais natūraliaisiais skaičiais, t.y.
   for $i := l + 1$ *to* k *do* $c_i := c_{i-1} + 1$;

Generavimo pabaigos sąlyga. Jei pirmasis iš dešinės elementas, tenkinantis sąlygą $c_l < n-k+l$, yra c_0, tai reiškia, kad nagrinėjame derinį: $0, n-k+1,\ldots,n$. Vadinasi, jau visi deriniai sugeneruoti.

Tuo būdu, derinių C_n^k generavimo leksikografine tvarka algoritmas yra.

Derinių skaičiaus apskaičiavimo algoritmas

Duota: n – objektų skaičius,
k – objektų skaičius derinyje.

Rasti: generuoti derinius leksikografine tvarka.

```
begin
    { c [0..k] – masyvas, kurio elementai c1, c2,..., ck nusako derinį }
    for l := 0 to k do c [l] := l; { Pradinio derinio generavimas }
    p := true; { Generavimo sąlyga }
    while p do
        begin
            spausdinti derinį c1, c2,..., ck ;
            { Generuoti leksikografiškai didesnį derinį }
            l := k;
            while (l >= 1) and c [l] >= n – k + l do l := l – 1;
                if l = 0 then p := false
                        else begin
                            c [l] := c [l] + 1;
                            for i := l + 1 to k do c [i] := c [i – 1] + 1;
                        end;
        end; { while p }
end;
```

Trečiasis Grėjaus kodų generavimo algoritmas

Kaip matyti iš antrojo Grėjaus kodo generavimo algoritmo (žr. 1.3 paragrafą), Grėjaus kodus galima generuoti iš pradinio kodo, žinant seką $T_n = t_1, t_2,..., t_i,..., t_{2^n-1}$. Šios sekos elementas t_i žymi skiltį, kurią reikia invertuoti, kad iš $(i-1)$-ojo kodo gautume *i*-ąjį kodą.

Pavyzdžiui, kai $n = 3$, seka T_3 yra tokia: $T_3 = 1, 2, 1, 3, 1, 2, 1$.

Vadinasi, pakanka mokėti efektyviai generuoti seką T_n, kad galėtume generuoti Grėjaus kodus.

Seką T_n galima efektyviai generuoti naudojant steką.

Pradžioje stekas užpildomas elementais: $n, n-1, n-2,..., 3, 2, 1$ (viršuje yra elementas 1). Algoritmas ima viršutinį steko elementą *i* ir patalpina jį į seką T_n; po to į steką įrašomi elementai $i-1, i-2,..., 1$. Būtent tokiu būdu žemiau pateiktas algoritmas generuoja Grėjaus kodus.

```
begin
    s := 0; { s – tuščias stekas }
    for j := n downto 1 do begin
                                g [j] := 0;
```

```
                    s ⇐ j;
                end;
        while s ≠ ∅ do
            begin
                spausdinti (g_n, g_{n-1},..., g_1);
                i ⇐ s;
                g [i] := 1 – g [i];
                for j := i – 1 downto 1 do s ⇐ j;
            end; { while }
    end;
end;
```

Skaičiuojant pagal šį algoritmą, steko turinys (viršutinis elementas yra steko viršūnėje) kis taip:

1								
2	2	1		1				
3	3	3	3	2	2	1	0	

, o i reikšmės sudarys T_3 seką $1, 2, 1, 3, 1, 2, 1$.

Pateiktą algoritmą galima patobulinti. Kiekvieną kartą, kai į steką patalpinamas elementas $j > 0$, mes apriori žinome, kad virš jo bus patalpinti elementai $j-1,..,1$. Protingas šios informacijos panaudojimas įgalina vietoj steko naudoti masyvą $\tau = (\tau_n, \tau_{n-1},..., \tau_1, \tau_0)$ tokiu būdu. Elementas τ_0 yra viršutinis steko elementas ir kiekvienam $j > 0$ τ_j yra elementas, steke esantis tuoj po elemento *j* (žemiau elemento *j*), jei *j* yra steke. Jei elemento *j* steke nėra, tai elemento τ_j reikšmė negali turėti jokios įtakos skaičiavimams ir todėl τ_j priskiriame reikšmę $j+1$, kadangi mes žinome, kad kai kitą kartą elementas $j+1$ bus patalpintas į steką, elementu, esančiu virš jo, bus elementas *j*. Dar daugiau, kadangi elementai $i-1, i-2,...,1$ bus talpinami į steką pašalinus elementą *i*, mums reikia tiktai priskirti $\tau_{i-1} := \tau_i$, laikant, kad visiems *j*, $1 \le j \le i-2$, reikšmė τ_j buvo priskirta, kai *j* buvo pašalintas iš steko (primename, tada $\tau_j = j+1$). Pagaliau, kai $i \ne 1$, elementai į steką patalpinami operacijos $\tau_0 := 1$ dėka.

Dabar gausime tokį algoritmą.

```
begin
    for j := 0 to n + 1 do begin
                    g [j] := 0;
```

```
                    τ[j] := j + 1;
                end;
    i := 0;
    while i < n + 1 do
        begin
            spausdinti (g_n, g_{n–1},..., g_1) ;
            i := τ[0];
            g [i] := 1 – g [i];
            τ[i – 1] := τ[i];
            τ[i] := i + 1;
            if i ≠ 1 then τ[0] := 1;
        end; { while }
end;
```

Kai $n = 3$, masyvo τ turinys skaičiavimo eigoje kis taip:

τ_4	5	5	5	5	5	5	5	5
τ_3	4	4	4	4	4	4	4	4
τ_2	3	3	3	3	4	4	4	4
τ_1	2	2	3	2	2	2	4	2
τ_0	1	2	1	3	1	2	1	4

.

seka T_3

Pateiktąjį algoritmą galima truputį patobulinti. Pastebėsim, jei operatorių $\tau_0 := 1$ patalpintume po operatoriaus *g* [*i*] := 1 – *g* [*i*], tai operatorių *if* $i \neq 1$ *then* τ[0]:=1 galima išmesti. Iš tikro, jei $i = 1$, tai operatorius τ [*i* – 1] := τ [*i*] tuoj pat ištaisys neteisingą τ [0] reikšmę. Tuo būdu galutinis Grėjaus kodų generavimo algoritmas bus toks.

```
begin
    for j := 0 to n + 1 do begin
                    g [j] := 0;
                    τ[j] := j + 1;
                end;
    i := 0;
    while i < n + 1 do
        begin
            spausdinti (g_n, g_{n–1},..., g_1) ;
            i := τ[0];
            g [i] := 1 – g [i];
```

```
        τ[0] := 1; τ[i – 1] := τ[i]; τ[i] := i + 1;
    end; { while }
end;
```

Derinių generavimas minimalaus pokyčio tvarka

Aptarkome derinių generavimo minimlaus pokyčio tvarka algoritmą. Minimlaus pokyčio tvarka – tai tokia derinių seka, kai bet koks šios sekos derinys yra gaunamas iš prieš jį stovinčio derinio, pakeitus jame vieną elementą kitu.

Pavyzdys. Panagrinėkime C_5^3. Deriniai, surašyti minimalaus pokyčio tvarka, bus tokie:

123, 134, 234, 124, 145, 245, 345, 135, 235, 125.

Nesunku pastebėti, kad bet koks šios sekos derinys, išskyrus pirmąjį, yra gaunamas iš prieš jį stovinčio derinio, pakeitus vieną elementą kitu.

Jei pirmajame derinyje elementą 2 pakeisime elementu 4, tai gausime antrąjį derinį. Jei jame elementą 1 pakeisime elementu 2, tai gausime trečiąjį derinį ir t.t.

Reikia pastebėti, kad ši derinių seka yra užciklinta, t.y. jei paskutiniame derinyje elementą 5 pakeisime elementu 3, tai gausime pirmąjį šios sekos derinį.

Kiekvieną derinį galima užkoduoti *n*-skilčiu dvejetainiu skaičiumi $g_1\, g_2 \ldots g_i \ldots g_n$: jei $g_i = 1$, tai elementas *i* priklauso deriniui; jei $g_i = 0$, tai elementas *i* nepriklauso deriniui. Tuo būdu deriniai C_n^k bus užkoduoti *n*-skilčiais dvejetainiais skaičiais, kiekvienas iš kurių turi *k* vienetukų ir $(n-k)$ nulių.

Aišku, kad minimalaus pokyčio derinių sekos kodai bus dvejetainiai skaičiai, turintys *k* vienetukų, ir bet kokie du gretimi skaičiai skirsis dviem skiltimis: vienoje nulis keičiamas vienetu, o kitoje vienetas keičiamas nuliu.

Pavyzdys. Aukščiau pateikto minimalaus pokyčio derinių C_5^3 sekos kodai bus:

5	4	3	2	1		5	4	3	2	1	
0	0	1	1	1	{1,2,3},	1	1	0	1	0	{2,4,5},
0	1	1	0	1	{1,3,4},	1	1	1	0	0	{3,4,5},
0	1	1	1	0	{2,3,4},	1	0	1	0	1	{1,3,5},
0	1	0	1	1	{1,2,4},	1	0	1	1	0	{2,3,5},
1	1	0	0	1	{1,4,5},	1	0	0	1	1	{1,2,5}.

Tokių kodų generavimas yra glaudžiai susijęs su 1.3 paragrafe išnagrinėtu Grėjaus kodų generavimu.

Tarkime, $G(n)$ – Grėjaus kodai, o $G(n,k)$, $0 \le k \le n$ – seka Grėjaus kodų, turinčių lygiai k vienetukų. Tada, kaip parodyta literatūroje [RND80], $G(n,k)$ seka sutampa su minimalaus pokyčio derinių seka.

Pavyzdys. Panagrinėkime Grėjaus kodus, kai $n=5$. Remiantis 1.3 paragrafe išnagrinėtu algoritmu, gausime tokią $G(5)$ seką:

	5	4	3	2	1		5	4	3	2	1		5	4	3	2	1
0	0	0	0	0	0	11	0	1	1	1	0	22	1	1	1	0	1
1	0	0	0	0	1	12	0	1	0	1	0	23	1	1	1	0	0
2	0	0	0	1	1	13	0	1	0	1	1	24	1	0	1	0	0
3	0	0	0	1	0	14	0	1	0	0	1	25	1	0	1	0	1
4	0	0	1	1	0	15	0	1	0	0	0	26	1	0	1	1	1
5	0	0	1	1	1	16	1	1	0	0	0	27	1	0	1	1	0
6	0	0	1	0	1	17	1	1	0	0	1	28	1	0	0	1	0
7	0	0	1	0	0	18	1	1	0	1	1	29	1	0	0	1	1
8	0	1	1	0	0	19	1	1	0	1	0	30	1	0	0	0	1
9	0	1	1	0	1	20	1	1	1	1	0	31	1	0	0	0	0
10	0	1	1	1	1	21	1	1	1	1	1						

Kodai, turintys 3 vienetukus, – pabraukti. Išrašę šiuos kodus, gausime minimalaus pokyčio derinio C_5^3 sekos kodus.

Norėdami efektyviai generuoti $G(n,k)$, $0 \le k \le n$, kodus, pasinaudokime $G(n,k)$ rekursyviniu apibrėžimu:

$$G(n,k) = \begin{pmatrix} 0 & G(n-1,k) \\ 1 & G(n-1,k-1)^R \end{pmatrix}, \tag{3.4.1}$$

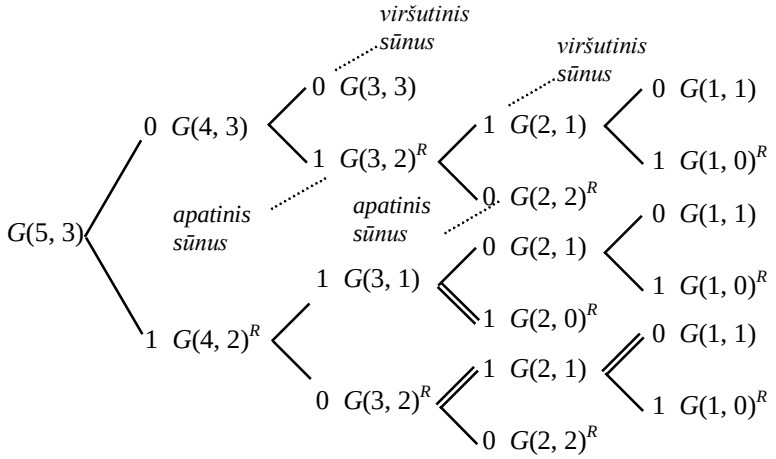
ir

$$G(n,0) = 0^n,$$
$$G(n,n) = 1^n,$$

čia $G(n-1,k-1)^R$ žymi kodus, surašytus atvirkščia tvarka, o 0^n reiškia, kad n skilčių užpildome nuliais, o 1^n – kad n skilčių užpildome vienetais.

Pastaroji rekurentinė formulė generuoja C_n^k pradedant $(1,2,...,k)$ ir baigiant $(1,2,...,k-1,n)$.

Pavyzdys. Remdamiesi (3.4.1) formule, pavaizduokime $C_5^3 = G(5,3)$ generavimo medį (žr. 3.4.1 pav.).



3.4.1 pav. $G(5, 3)$ generavimas, remiantis (3.4.1) rekurentine formule

Remdamiesi šiuo medžiu, gausime seką:

Eilės nr.	Kodas	Eilės nr.	Kodas
1	00111	6	11010
2	01101	7	11100
3	01110	8	10101
4	01011	9	10110
5	11001	10	10011

Nesunku parodyti (žr. [RND80]), kad 3.4.1 pav. pavaizduoto binarinio medžio kabančios viršūnės yra arba $1\,G(m,0)^R$ arba $0\,G(m,m)$.

Kodų $G(n,k)$ generavimo uždavinys transformuojasi į binarinio medžio kabančių viršūnių peržiūrą: pradedant viršutine ir baigiant apatine.

Norint pereiti nuo vienos kabančios viršūnės į gretimą kabančią viršūnę, nagrinėjame kelią iš pirmos kabančios viršūnės medžio šaknies link. Tuo keliu link šaknies keliaujame viršūnėmis, kurios yra apatiniai sūnūs (žr. 3.4.1 pav.). Pasiekę pirmąją viršūnę, kuri yra viršutinis sūnus, pereiname pas jo brolį (apatinį sūnų) ir iš jo, eidami viršutiniais sūnumis, einame į kabančią viršūnę.

Pavyzdžiui (žr. 3.4.1 pav.), išnagrinėję kabančią viršūnę $1\,G(2,0)^R$, pirmiausia pasieksime viršutinį sūnų $1\,G(3,1)$ ir pereisime pas jo brolį (apatinį sūnų) $0\,G(3,2)^R$. Ir iš jo viršutiniais sūnumis $1\,G(2,1)$, $0\,G(1,1)$ pasieksime viršūnei $1\,G(2,0)^R$ gretimą kabančią viršūnę $0\,G(1,1)$. (3.4.1 pav. šis kelias paryškintas).

Nagrinėdami kabančią binarinio medžio viršūnę $(g_n, g_{n-1},..., g_1)$, mes tuo pačiu žinome kelią iš šaknies į kabančią viršūnę. Norėdami pereiti prie gretimos kabančios viršūnės, turime saugoti informaciją apie viršūnę $g_i\,G(i-1,t)$, į kurią turime sugrįžti (viršūnės $g_i\,G(i-1,t)^R$ visada yra apatiniai sūnūs). Šios viršūnės bus saugomos steke. Toliau pateiktame C_n^k generavimo algoritme stekas bus organizuojamas taip pat, kaip jis buvo organizuotas generuojant Grėjaus kodus (žr. trečiąjį Grėjaus kodų generavimo algoritmą). Parametras *t* rodo skaičių vienetukų tarp skilčių $(g_{i-1}, g_{i-2},..., g_1)$ ir gali būti nesunkiai apskaičiuojamas. Todėl steke saugoma tik *i* reikšmė.

Atidžiai ištyrinėjus binarinį medį, galima pastebėti, kad nagrinėjamo kodo, kai iš jo grįžtama į viršūnę $g_i\,G(i-1,t)$, perskaičiavimas į gretimą kodą reiklauja nagrinėti žemiau pateiktus keturis atvejus, priklausomai nuo g_i ir *t* reikšmės:

g_i = 0, t > 1 $g_i - \downarrow$ $\ldots 010^{i-t-1}11^{t-2}$ $\downarrow$ $110^{i-t-1}01^{t-2}$	**g_i = 1, t > 0** $g_i - \downarrow$ $\ldots 110^{i-t-2}11^{t-1}$ $\downarrow$ $\ldots 010^{i-t-2}11^{t-1}$
g_i = 0, t = 1 $g_i - \downarrow$ $\ldots 010^{i-2}$ $\downarrow$ $\ldots 100^{i-2}$	**g_i = 1, t = 0** $g_i - \downarrow$ $\ldots 110^{i-2}$ $\downarrow$ $\ldots 010^{i-2}$

Šie pertvarkymai turi įtakos vienetukų skaičiui tarp skilčių $(g_{i-1}, g_{i-2},..., g_1)$: *t* reikšmę turime sumažinti 1, jei $g_i = 0$, ir padidinti 1, jei $g_i = 1$.

Pereinant prie gretimo kodo yra invertuojamos dvi skiltys:

1) skiltis g_i,

ir

2) g_{t-1}, jei $g_i = 0$ ir $t > 1$,
   g_{i-1}, jei $g_i = 0$ ir $t = 1$,
   g_t, jei $g_i = 1$ ir $t > 0$,
   g_{i-1}, jei $g_i = 1$ ir $t = 0$.

Atlikus šiuos veiksmus, į steką reikia įrašyti papildomą informaciją: tarpines viršūnes, esančias tarp nagrinėjamos viršūnės ir nagrinėjamos kabančios viršūnės. Kadangi tas priklauso nuo viršūnės steko viršuje, tai *t* reikšmė gali vėl pasikeisti.

Jei $t = 0$ arba $t = i - 1$, tai reiškia, kad viršūnė, kurioje mes esame, yra kabanti viršūnė: $g_i\ G(i-1, i-1)$ arba $g_i\ G(i-1,0)^R$ ir jokių tarpinių viršūnių saugoti nereikia. Šiuo atveju *t* visada lygus $t+1$.

Kita vertus, jei $t \neq 0$ ir $t \neq i-1$, tai į steką turime įtraukti viršūnes $i-1, i-2, ..., t+1$, išskyrus tą atvejį, kai $g_{i-2} = ... = g_1 = 0$. Šiuo atveju į steką įtraukiame tik $i-1$.

Parametro t reikšmė turi būti perskaičiuojama. Kadangi tarp $g_{i-1}, g_{i-2}, ..., g_{t+1}$ vienetukų gali būti tik ϑ^{i-t}, tai $t := t - g_{i-1}$. Be to, jei šiuo atveju *t* tampa lygus 0, tai $g_{i-2} = ... = g_1 = 0$.

Žemiau pateiktas derinių C_n^k generavimo minimalaus pokyčio tvarka algoritmas. Šiame algoritme stekas organizuojamas masyvu τ, o jo viršutinis elementas yra τ_1 (žr. trečiąjį Grėjaus kodų generavimo algoritmą). Kadangi $\tau_{i-1} := \tau_1$, tai $\tau_1 := t+1$ ekvivalentu, kad į steką įtraukiame $i-1, i-2, ..., i+1$.

```
const nn = 20;
type mas = array [0..nn] of integer;
var n, i, k : integer;
    g, tau : mas;
procedure der (n, k : integer);
var i, j, t : integer;
g, tau : mas;
begin
    for j := 1 to k do
        begin
            g [j] := 1;
            tau [j] := j + 1;
```

```
        end;
    for j := k + 1 to n + 1 do
        begin
            g [j] := 0;
            tau [j] := j + 1;
        end;
    t := k;
    tau [1] := k + 1;
    i := 0;
    while i <> n + 1 do
        begin
            writeln;
            for j := n downto 1 do write (g [j] : 3);
            writeln;
            i := tau [1];
            tau [1] := tau [i];
            tau [i] := i + 1;
            if g [i] = 1 then
                begin
                    if t <> 0 then g[t]:=1-g[t]
                    else g[i-1]:=1-g[i-1];
                    t:=t+1;
                end
                        else
                begin
                    if t <> 1 then g [t – 1] := 1 – g [t – 1]
                    else g [i – 1] := 1 – g [i – 1];
                    t := t – 1;
                end;
            g [i] :=1 – g [i];
            if (t = i – 1) or (t = 0) then t := t + 1
                                      else
                begin
                    t := t – g [i – 1];
                    tau [i – 1] := tau [1];
                    if t = 0 then tau [1] := i – 1
                             else tau [1]:=t+1;
                end;
        end;
end;
begin
```

```
writeln ('įveskite n');
readln (n);
writeln ('įveskite k');
readln (k);
der (n, k);
end.
```

Atsitiktinio derinio generavimas

Aptarsime atsitiktinio poaibio, turinčio k elementų, iš aibės, turinčios n elementų $\{a_1, a_2, ..., a_n\}$, generavimą. Aišku, kad tokio poaibio generavimo tikimybė yra $1/C_n^k$.

Tam tikslui atsitiktinai parenkame vieną iš n elementų a_i (pasirinkimo a_i tikimybė $1/n$) ir po to atsitiktinai generuojame $(k-1)$-elementį poaibį iš likusių $(n-1)$ elementų.

Tada šios procedūros eigoje k-elemenčio poaibio $(a_{i_1}, a_{i_2}, ..., a_{i_k})$ išsirinkimo tikimybė yra tokių tikimybių sandauga:

P (pirmasis išsirinktas elementas – vienas iš k elementų $a_{i_1}, a_{i_2}, ..., a_{i_k}$) $= \frac{k}{n}$,

P (antrasis išsirinktas elementas – vienas iš $k-1$ likusių elementų) $= \frac{k-1}{n-1}$,

– – – — – – — – – — – – — – – — – – –

P (k-asis išsirinktas elementas – paskutinysis iš likusių elementų) $= \frac{1}{n-k+1}$.

Ši sandauga bus lygi:

$$\frac{k}{n}\frac{k-1}{n-1}\cdot ... \cdot \frac{1}{n-k+1} = \frac{k!(n-k)!}{n!} = \frac{1}{C_n^k}.$$

Aptarsime atsitiktinio derinio generavimo algoritmą.

Algoritmo realizavimui panaudosime papildomą masyvą p, kuriame saugosime dar neišrinktų elementų numerius. Po to, kai pirmieji $j-1 \le k$ elementai išsirinkti, tai $p_j, p_{j+1}, ..., p_n$ bus sąrašas $n-j+1$ elementų, kurie dar neišsirinkti. Kadangi nebūtina išsaugoti elementų tvarką, tai, išsirinkus elementą p_r, $r \in [j, n]$, įjo vietą įrašome neišsirinktą elementą p_j.

Žemiau pateikiamas atsitiktinio derinio generavimo algoritmas.

```
begin
   for j := 1 to n do p [j] := j;
```

```
    R := ∅;
    for j := 1 to k do
        begin
            r := rand (j, n);
            { rand (j, n) – tolygiai pasiskirsčiusių atsitiktinių skaičių iš intervalo
            [j, n] generavimo procedūra }
            R := R ∪ {a [p [r]]};
            p [r] := p [j];
        end;
end;
```

3.4.2. Kėlinių generavimo algoritmai

Kėlinių generavimo leksikografine tvarka algoritmas

Aptarsime kėlinių iš n elementų generavimo leksikografine tvarka uždavinį. Spręsdami šį uždavinį, laikysime aukščiau pateiktos (žr. 3.4 paragrafo pradžią) kombinatorinų objektų generavimo algoritmo struktūros. Norėdami išsiaiškinti šio algoritmo pagrindinius momentus, panagrinėkime pavyzdį, kai $n = 8$.

Pradinio kodo generavimas. Kėlinį talpinsime masyve $p[0..n]$, kurio elementai $p_1, p_2, ..., p_n$ ir nusako kėlinį. Aišku, kad pradinis kėlinys yra $1,2,3,..., n$. Pavyzdžiu, jei $n = 8$, tai 1, 2, 3, 4, 5, 6, 7, 8.

Paties mažiausio, leksikografiškai didesnio kėlinio nei nagrinėjamas, apskaičiavimas. Tarkime, $p_1, p_2, ..., p_n$ – nagrinėjamas kėlinys. Pavyzdžiui, $2, 1, 4, 8, 7, 6, 5, 3$. Jei į kėlinį žiūrėsime kaip į skaičių, tai reikia rasti patį mažiausią skaičių, didesnį nei nagrinėjamas. Tam tikslui reikia:

1) rasti pirmą iš dešinės porą (p_i, p_{i+1}) tokią, kad $p_i < p_{i+1}$,
2) rasti $p_j = \min_{i+1 \le l \le n} (p_l \mid p_l > p_i)$, t.y tarp elementų $p_{i+1}, p_{i+2}, ..., p_n$ rasti patį mažiausią elementą, didesnį už p_i,
3) elementus p_i ir p_j sukeisti vietomis,
4) elementus $p_{i+1}, p_{i+2}, ..., p_n$ ("uodegą") surašyti atvirkščia tvarka ("apversti").

Pavyzdžiui, pirmoji iš dešinės pora, tenkinanti 1) punkto reikalavimą, yra $(4,8)$, o elementas $p_j = 5$. Tada, sukeitę 4 su 5, gausime $2,1,5,8,7,6,4,3$. "Apvertę uodegą", gausime patį mažiausią leksikografiškai didesnį kėlinį nei duotas: $2,1,5,3,4,6,7,8$.

Generavimo pabaigos sąlyga. Aišku, kad pats didžiausias kėlinys yra $0, n, n-1, ..., 3, 2, 1$ (masyvo $p[0..n]$ elementas $p[0]$ yra pagalbinis, ir jo reikšmė lygi nuliui). Tada pirma iš dešinės pora (p_i, p_{i+1}), tenkinanti sąlygą $p_i < p_{i+1}$ yra $(0, n)$. Kitaip tariant, jei $i = 0$, tai generavimo pabaiga.

Vadinasi, kėlinių P_n generavimo leksikografine tvarka algoritmas gali būti užrašytas taip.

```
begin
    fo i:= 0 to n do p [i] :=i;
    t := true;
    while t do
        begin
            spausdinti (p1, p2,.., pn);
            { Leksikografiškai didesnio kėlinio apskaičiavimas }
            i := n – 1;
            while p [i] > p [i+ 1] do i := i – 1;
            if i = 0 then t := false
                    else
                        begin
                            { Rasti  pj = min (pl | pl > pi)  }
                            j := n;
                            while p [j] < p [i] do j := j – 1;
                            { Elementus pi ir pj sukeisti vietomis }
                            pp := p [i]; p [i] := p [j]; p [j] := pp;
                            { Elementų  pi+1, pi+2,..., pn–1, pn  surašymas atvirkščia
                            tvarka }
                            k := i + 1; l := n;
                            while k < l do
                            begin
                            pp := p [k]; p [k] := p [l]; p [l] := pp;
                            k := k + 1; l := l – 1;
                            end;
                        end; { else }
        end; { while }
end;
```

Kėlinių generavimas antileksikografine tvarka

Be kėlinių generavimo leksikografine tvarka naudojamas ir kėlinių generavimo antileksikografine tvarka metodas.

Apibrėžimas. Vektorius $\boldsymbol{x} = (x_1, x_2, ..., x_n)$ yra antileksikografiškai mažesnis už vektorių $\boldsymbol{y} = (y_1, y_2, ..., y_n)$ (žymime $\boldsymbol{x} <' \boldsymbol{y}$), jei egzistuoja toks $k \le n$, kad $x_k > y_k$, o $x_i = y_i$, $i = \overline{k+1, n}$. Pastebėsime, jei vietoje skaičių $1,2,..., n$ paimtume raides $a, b, c, \ldots, z$, išdėstytas natūralia, pavyzdžiui, abėcėlės tvarka: $a < b < c < ... < z$, tai leksikografinė tvarka apibrėš n ilgio žodžių išsidėstymą žodyne įprasta tvarka. Tuo tarpu antileksikografinė tvarka apibrėžia žodžių tvarką, kai tiek pozicijų eiliškumas sekoje, tie ir aibės elementų išsidėstymas žodyje yra atvirkščias.

Pavyzdžiui, užrašykime aibės $X = \{1,2,3\}$ kėlinius leksikografine ir antileksikografine tvarka.

Leksikografine tvarka:

1, 2, 3; 1, 3, 2; 2, 1, 3; 2, 3, 1; 3, 1, 2; 3, 2, 1.

Antileksikografine tvarka:

1, 2, 3; 2, 1, 3; 1, 3, 2; 3, 1, 2; 2, 3, 1; 3, 2, 1.

Iš leksikografinės tvarkos kėlinio 1, 3, 2 gausime antileksikografinės tvarkos kėlinį, jei 1-tą keisime 3-tu, 3-tą keisime 1-tu (imame aibės X elementus atvirkščia tvarka) ir kėlinio pozicijas surašome atvirkščia tvarka. Gausime 3, 1, 2, o po to 2, 1, 3.

Kėlinių generavimas antileksikografine didėjimo tvarka yra labai panašus į kėlinių generavimo leksikografine didėjimo tvarka algoritmą. Kėlinių P_n generavimui įveskime masyvą $p[1..n+1]$, kurio elementai $p_{i+1}, p_{i+2}, ..., p_n$ apibrėžia nagrinėjamąjį kėlinį.

Pradinio kėlinio generavimas. for i := 1 *to n* + 1 *do p* [*i*] := *i*;

Paties mažiausio antileksikografiškai didesnio kėlinio apskaičiavimas. Tam tikslui reikia:

1) rasti pirmą iš kairės porą (p_{i-1}, p_i), tokią, kad $p_{i-1} < p_i$,
2) rasti $p_j = \max\limits_{1 \le l \le i-1} (p_l \mid p_l < p_i)$, t.y tarp elementų $p_1, p_2, ..., p_{i-1}$ rasti patį didžiausią elementą, mažesnį nei p_i,
3) elementus p_i ir p_j sukeisti vietomis,
4) elementus $p_1, p_2, ..., p_{i-1}$ surašyti atvirkščia tvarka.

Generavimo pabaigos sąlyga. Paskutinysis antileksikografinės tvarkos kėlinys yra $n, n-1, ..., 3, 2, 1$, o $p_{n+1} = n+1$, t.y. masyvo $p[1..n+1]$ elementai yra $n, n-1, ..., 3, 2, 1, n+1$. Aišku, kad šiuo atveju pirmoji iš dešinės pora

(p_{i-1}, p_i), tenkinanti 1)-ojo punkto reikalavimą: $p_{i-1} < p_i$, yra $(1, n+1)$. Vadinasi, jei $i = n+1$, tai generavimo pabaigos sąlyga.

Pateiksime kėlinių P_n generavimo antileksikografine tvarka algoritmą.

```
begin
    fo i:= 0 to n + 1 do p [i] :=i;
    t := true;
    while t do
        begin
            spausdinti (p1, p2,..., pn);
            { Antileksikografiškai didesnio kėlinio apskaičiavimas }
            i := 2;
            while p [i-1] > p [i] do i := i + 1;
            if i = n + 1 then t := false
                    else
                        begin
                            { Rasti  pj = max(pl | pl < pi)  }
                            j := 1;
                            while p [j] > p [i] do j := j + 1;
                            { Elementus pi ir pj sukeisti vietomis }
                            pp := p [i]; p [i] := p [j]; p [j] := pp;
                            { Elementus  p1, p2,..., pi–1 surašyti atvirkščia tvarka }
                            k := 1; l := i - 1;
                            while k < l do
                            begin
                            pp := p [k]; p [k] := p [l]; p [l] := pp;
                            k := k + 1; l := l – 1;
                            end;
                        end; { else }
        end; { while }
end;
```

Kai $n = 4$, tai pateiktas algoritmas apskaičiuos kėlinius:

1234	1243	1342	2341
2134	2143	3142	3241
1324	1423	1432	2431
3124	4123	4132	4231
2314	2413	3412	3421
3214	4213	4312	4321

Kėlinių generavimas minimalaus pokyčio tvarka

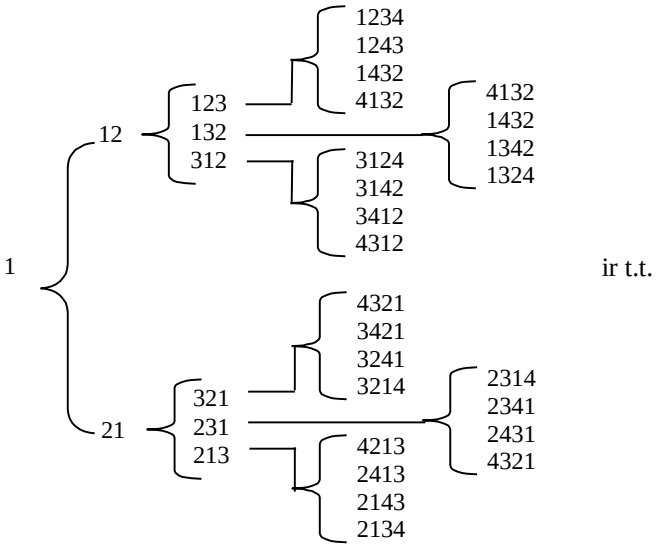
Panagrinėkime kėlinių generavimo minimalaus pokyčio tvarka metodą ir algoritmą. Kėlinių minimalaus pokyčio tvarka – tai tokia kėlinių generavimo seka, kai bet kokie du gretimi kėliniai skiriasi tik dviejų elementų padėtimi. Pavyzdžiui, kai $n=3$, gautume seką: 1, 2, 3; 1, 3, 2; 3, 1, 2; 3, 2, 1; 2, 3, 1; 2, 1, 3.

Aptarsime kėlinių generavimo tokia tvarka metodą.

Kai $n=1$, tai egzistuoja vienintelis kėlinys: 1. Tarkime, kad turime P_{n-1} kėlinius, tenkinančius minimalaus pokyčio tvarką. Tada kėlinį P_n generuosime taip.

Sunumeruokime visus P_{n-1} kėlinius numeriais nuo 1 iki $(n-1)!$. Tada, jei kėlinio iš $n-1$ elemento numeris yra nelyginis skaičius, tai prie šio kėlinio prirašykime elementą n iš dešinės ir nuosekliai atlikime visus galimus to elemento postūmius į kairę. Jei kėlinio iš $n-1$ elemento numeris yra lyginis, tai prie jo prirašykime elementą n iš kairės ir nuosekliai atlikime visus galimus to elemento postūmius į dešinę.

Atlikę šiuos veiksmus, gausime kėlinių P_n minimalaus pokyčio seką (žr. 3.4.2 pav.).



3.4.2 pav. Kėlinų generavimas minimalaus pokyčio tvarka

Aptartas kėlinių minimalaus pokyčio sekos generavimo metodas neatitinka aukščiau pateiktos kombinatorinių objektų generavimo algoritmo schemos: skaičiavimo eigoje reikia saugoti visus P_{n-1} kėlinius. Literatūroje [RND80] pateikta šio metodo realizacija, atitinkanti kombinatorinių objektų generavimo algoritmo scemą.

Aptarkime šią realizaciją.

Įveskime tris masyvus: $p[0..n+1]$, $a[1..n]$ ir $d[1..n]$.

Masyvo *p* elementai $p_1, p_2, ..., p_n$ apibrėš nagrinėjamą kėlinį. Elementai p_0 ir p_{n+1} yra pagalbiniai elementai.

Masyvo *a* elementai nusako atvirkštinį kėlinį: a_k nurodo vietą (adresą), kurioje masyve *p* yra elementas *k*. Kitaip tariant, $p_{a_k} = k$. Masyvo *d* elementai apibrėžiami taip:

$$d_i = \begin{cases} -1, \text{ jei elementas } i \text{ kėlinyje yra "stumiamas" į kairę} \\ \quad \text{(keičiamas su kaimynu iš kairės)}, \\ +1, \text{ jei elementas } i \text{ kėlinyje yra "stumiamas" į dešinę} \\ \quad \text{(keičiamas su kaimynu iš dešinės)}, \\ 0, \quad \text{jei elementas } i \text{ kėlinyje nejuda.} \end{cases}$$

Kėlinyje elementas *i* “stumiamas” iki jis pasiekia elementą, didesnį už jį patį. Tada yra keičiama šio elemento judėjimo kryptis ir bandoma “stumti” (jei galima) elementą $i - 1$. Kadangi turime masyvą *a*, tai šis elementas lengvai randamas masyve *p*.

Tam, kad nutrauktume elemento *n* judėjimą, į masyvą *p* įvesti du papildomi elementai p_0 ir p_{n+1}, kurių reikšmės yra lygios $n+1$.

Elemento d_1 reikšmė yra lygi 0, nes *p* elemento, lygaus vienetui, judinti nereikia. Tai sekos generavimo pabaigos sąlyga.

Kėlinių generavimo minimalaus pokyčio tvarka algoritmas

```
begin
   { Pradinio kėlinio generavimas }
   for i := 1 to n do begin
                          p [i] := i;
                          a [i] := i;
                          d [i] := –1;
                       end;
   d [1] := 0; p [0] := n + 1; p [n + 1] := n + 1; t := true;
```

```
    while t do
        begin
            Spausdinti ( p1, p2,..., pn );
            { "Gretimo" kėlinio apskaičiavimas }
            m := n; { Judinsime (jei galima) patį didžiausią elementą }
            while p [a [m] + d [m]] > m  do begin
                                                d [m] := – d [m];
                                                m := m – 1;
                                            end;
    if m = 1 then t := false
                else
                  begin
                      { Sukeisti p [a [m]] su p [a [m] + d [m]] }
                      pp := p [a [m]];
                      p [a [m]] := p [a [m] + d [m]];
                      p [a [m] + d [m]] : =pp;
                      { Sukeisti elementų m ir p [a [m] + d [m]] adresus. Po
                      sukeitimo elementas p [a [m] + d [m]] yra p [a [m]] vietoje.
                      Vadinasi, keisime a [m] ir a [p [a [m]]] }
                      pp := a [m]; k := p [a [m]];
                      a [m] := a [k];
                      a [k] : =pp;
                  end;
        end; { while }
end;
```

Kai $n = 4$, pateiktas algoritmas apskaičiuoja:

1234	1342	4321	2431
1243	1324	3421	4231
1423	3124	3241	4213
4123	3142	3214	2413
4132	3412	2314	2143
1432	4312	2341	2134

Atsitiktinio kėlinio generavimas

Tarkime, kad $p = (p_1, p_2,..., p_n)$ – bet koks kėlinys. Perskaičiuokime šį kėlinį pagal algoritmą:

```
begin
    for i := n downto 2 do
    sukeisti p [i] su p [rand (1, i)],
```

čia rand [*k, l*] *– tolygiai pasiskirsčiusių atsitiktinių skaičių iš atkarpos* [*k, l*] *generavimo procedūra*;
end;

Taip gautas kėlinys *p* su tikimybe $1/n!$ gali sutapti su bet kuriuo kėliniu iš visų kėlinių aibės.

Tam, kad įrodytume, jog visi $n!$ kėliniai vienodai galimi, pasinaudokime matematinės indukcijos metodu.

Kai $n=1$ – akivaizdu.

Tarkime, kad algoritmas generuoja atsitiktinį kėlinį, kai elementų skaičius yra $n-1$. Tegu $\Sigma=(s_1,s_2,..,s_n)$ yra bet kuris aibės $\{1,2,...,n\}$ kėlinys. Tada tikimybė, kad $p=\Sigma$ yra:

$$P(p=\Sigma)=P(p_n=s_n)P((p_1,p_2,...,p_{n-1})=(s_1,s_2,...,s_{n-1}),$$

$$\text{prie sąlygos,kad } p_n=s_n)=\frac{1}{n}\cdot\frac{1}{(n-1)!}=\frac{1}{n!}.$$

3.4.3. Aibės išskaidymas

n-elementės aibės *X* išskaidymas į *k* poaibių (blokų) – tai aibės *X* poaibių šeima $\pi=\{B_1,B_2,...,B_k\}$, kurios elementai tenkina reikalavimus:

1) $B_1\cup B_2\cup...\cup B_n=X$,
2) $B_i\cap B_j=\varnothing, i\neq j$,
3) $B_i\neq\varnothing, i=\overline{1,k}$.

Poaibius B_i vadinsime blokais.

Aibės *X* visų galimų išskaidymų į *k* blokus aibę žymėsime $\Pi_k(x)$, o aibės *X* visų galimų išskaidymų aibę žymėsime $\Pi(x)$. Aišku, kad

$$\Pi(x)=\Pi_1(x)\cup\Pi_2(x)\cup...\cup\Pi_n(x).$$

Su aibės *X* išskaidymu į blokus yra susiję Stirlingo ir Belo skaičiai.

Antrosios rūšies Stirlingo skaičius $S(n,k)$ – tai skirtingų *n*-elementės aibės išskaidymo į *k* blokus skaičius, t.y.

$$S(n,k)=|\Pi_k(x)|,\ \text{čia}\ |X|=n.$$

Pavyzdžiui, $S(4,2)=7$, kadangi yra 7 skirtingi 4-ių elementų aibės $\{1,2,3,4\}$ išskaidymo į du blokus būdai:

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

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

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

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

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

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

Akivaizdu, kad $S(n,k)=0$, kai $k>n$. Be to, $S(0,0)=1$, kadangi tuščia blokų aibė sutinkamai su apibrėžimu yra tuščiosios aibės išskaidymas.

Su antrosios rūšies Stirlingo skaičiais, kaip ir su binominiais koeficientais, yra susiję daug tapatybių.

Pirmiausia įrodysime tapatybę, primenančią derinių sudėties savybę, kuri yra susijusi su Paskalio trikampiu:

a) $S(n,k)=S(n-1,k-1)+k\cdot S(n-1,k)$, $0\le k\le n$,

b) $S(n,n)=1$, $n\ge 0$, (3.4.2)

c) $S(n,0)=0$, $n>0$.

Formulių $S(n,n)=1$ ir $S(n,0)=0$ teisingumas akivaizdus. 3.4.1 a) formulės teisingumą įrodysime taip. Visus *n*-elementės aibės išskaidymus į *k* blokų suskirstykime į dvi klases. Pirmajai klasei priskirkime visus išskaidymus, kuriuse yra vienelementinis blokas $\{n\}$, o antrajai klasei priskirkime visus išskaidymus, kai elementas *n* yra didesnio bloko (mažiausiai dviejų elementų aibės) elementas.

Aišku, kad pirmosios klasės galia yra $S(n-1,k-1)$, t.y. tokia, kokia yra aibės $\{1,2,...,n-1\}$ išskaidymo į $(k-1)$ blokus galia.

Antrosios klasės galia yra $k\cdot S(n-1,k)$, kadangi prie kiekvieno $S(n-1,k)$ išskaidymo į *k* blokus *k* skirtingais būdais galima prijungti elementą *n*: elementas *n* paeiliui jungiamas prie kiekvieno bloko.

(1.3.4) formulė įgalina lengvai apskaičiuoti $S(n,k)$. Žemiau pateikta $S(n,k)$ reikšmių lentelė, primenanti Paskalio trikampį.

Šią lentelę galima traktuoti kaip “Stirlingo trikampį”. *i*-osios eilutės elementas yra lygus $(i-1)$-osios eilutės elemento iš kairės ir $(i-1)$-osios eilutės elemento iš viršaus, padauginto iš *k,* sumai.

6 lentelė. Antrosios rūšies Stirlingo skaičiai

$n\backslash k$	0	1	2	3	4	5	6	7	8	9	10
0	1	0	0	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	0	0	0
2	0	1	1	0	0	0	0	0	0	0	0
3	0	1	3	1	0	0	0	0	0	0	0
4	0	1	7	6	1	0	0	0	0	0	0
5	0	1	15	25	10	1	0	0	0	0	0
6	0	1	31	90	65	15	1	0	0	0	0
7	0	1	63	301	350	140	21	1	0	0	0
8	0	1	127	966	1701	1050	266	28	1	0	0
9	0	1	255	3025	7770	6951	2646	462	36	1	0
10	0	1	511	9330	34105	42525	22827	5880	750	45	1

Antrosios rūšies Stirlingo skaičiai yra susiję su *n*- osios eilės polinomo $P(x)$, užrašyto bazėje $1, x,.., x^n$, užrašymu bazėje $1,[x]_1,[x]_2,...,[x]_n$, čia

$$[x]_k = x(x-1)...(x-k+1).$$

Teorema [Lip88]. Kiekvienam $n \geq 0$

$$x^n = \sum_{k=0}^{n} S(n,k)[x]_k .$$

Pavyzdžiui, $x^3 = S(3,0)[x]_0 + S(3,1)[x]_1 + S(3,2)[x]_2 + S(3,3)[x]_3$.

Iš tikro,

$$x^3 = S(3,0)[x]_0 + S(3,1)[x]_1 + S(3,2)[x]_2 + S(3,3)[x]_3 =$$
$$= 0 \cdot 1 + 1 \cdot x + 3x(x-1) + 1 \cdot x(x-1)(x-2) =$$
$$= x + 3x^2 - 3x + x^3 - 3x^2 + 2x = x^3.$$

Pirmosios rūšies Stirlingo skaičiai. Šie skaičiai žymimi $s(n,k)$ ir jie yra koeficientai prie *x* laipsnių daugianaryje $[x]_n$, t.y.

$$[x]_n = \sum_{k=0}^{n} s(n,k)x^k .$$

Kitaip tariant, skaičiai $s(n,k)$ vaidina atvirkščią vaidmenį atžvilgiu $S(n,k)$, t.y. leidžia polinomą, užrašytą bazėje $1,[x]_1,[x]_2,...,$ pervesti į polinomo užrašą bazėje $1, x, x^2,...$.

Nesunku parodyti (palyginant koeficientus prie x^k laipsnių lygybėje $[x]_n = [x]_{n-1}(x-n+1)$), kad $s(n,k)$ tenkina sąryšius:

$s(n,k) = s(n-1,k-1) - (n-1)\cdot s(n-1,k)$, $0 \le k \le n$,

$s(n,n) = 1$, $n \ge 0$, (3.4.3)

$s(n,0) = 0$, $n > 0$.

$S(n,k)$.

Žemiau pateikta pirmosios rūšies Stirlingo skaičių lentelė.

7 lentelė. Pirmosios rūšies Stirlingo skaičiai

$n\backslash k$	0	1	2	3	4	5	6	7	8	9	10
0	1	0	0	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	0	0	0
2	0	-1	1	0	0	0	0	0	0	0	0
3	0	2	-3	1	0	0	0	0	0	0	0
4	0	-6	11	-6	1	0	0	0	0	0	0
5	0	24	-50	35	-10	1	0	0	0	0	0
6	0	-120	274	-225	85	-15	1	0	0	0	0
7	0	720	-1764	1624	-735	175	-21	1	0	0	0
8	0	-5040	13068	-13132	6769	-1960	322	-28	1	0	0
9	0	40320	-109584	118124	-67284	22449	-4536	546	-36	1	0
10	0	-362880	1026576	-1172700	723680	-269325	63273	-9450	870	-45	1

Belo skaičius B_n. Belo skaičius – tai visų galimų n-elementės aibės X išskaidymo į blokus skaičius:

$$B_n = |\Pi(x)|, \text{ čia } |X| = n.$$

Kitaip tariant,

$$B_n = \sum_{k=0}^{n} S(n,k).$$

Įrodysime Belo skaičius siejantį rekurentinį sąryšį:

$$B_{n+1} = \sum_{i=0}^{n} C_n^i B_i, \; B_0 = 1. \tag{3.4.4}$$

Aibės $X = \{1,2,...,n+1\}$ visų galimų išskaidymų aibę galima išskaidyti į klases, priklausomai nuo bloko B, turinčio savyje elementą $n+1$, arba, kas ekvivalentu, priklausomai nuo aibės $X \setminus B$. Kiekvienai aibei $X \setminus B \subseteq \{1,2,...,n\}$ egzistuoja tiksliai $|\Pi(X \setminus B)| = B_{|X \setminus B|}$ aibės X išskaidymų, turinčių savyje bloką B. Grupuodami klases priklausomai nuo aibės $X \setminus B$ galios, gauname (3.4.3) formulę.

Žemiau pateikta Belo skaičių lentelė.

8 lentelė. Belo skaičiai

n	B_n
0	1
1	1
2	2
3	5
4	15
5	52
6	203
7	877
8	4 140
9	21 147
10	115 975
11	678 570
12	4 213 597
13	27 644 437
14	190 899 322
15	1 382 958 545

Aibės išskaidymo generavimo algoritmas

Aptarsime n-elementės aibės visų galimų išskaidymų generavimo algoritmą. Šio algoritmo idėją lengviausia paaiškinti suformulavus ją rekurentinėje formoje.

Aišku, kad kiekvienas aibės $\{1,2,\ldots,n\}$ išskaidymas π vienareikšmiškai nusako aibės $\{1,2,...,n-1\}$ išskaidymą π_{n-1}, gautą iš išskaidymo π, kai iš jo atitinkamo bloko pašalinamas elementas n (ir pašalinamas tuščiasis blokas, jei elementas n sudarė vienelementį bloką).

Ir atvirkščiai, jei turime aibės $\{1,2,...,n-1\}$ išskaidymą $\sigma = \{B_1, B_2, ..., B_k\}$, tai lengva rasti visus aibės $\{1,2,\ldots,n\}$ išskaidymus π, kad $\pi_{n-1} = \sigma$, t.y.

$$\begin{aligned} &B_1 \cup \{n\}, B_2, ..., B_k, \\ &B_1, B_2 \cup \{n\}, ..., B_k, \\ &\text{– – – – – – – – –} \\ &B_1, B_2, ..., B_k \cup \{n\}, \\ &B_1, B_2, ..., B_k, \{n\}. \end{aligned} \qquad (3.4.5)$$

Ši savybė įgalina sudaryti paprastą rekurentinį metodą, generuojantį visus galimus aibės $X = \{1,2,..., n\}$ išskaidymus: jei mes žinome aibės $\{1,2,..., n-1\}$ visų galimų išskaidymų sąrašą L_{n-1}, tai visų galimų aibės $\{1,2,\ldots,n\}$ išskaidymų sąrašas L_n gaunamas pakeičiant kiekvieną sąrašo L_{n-1} išskaidymą σ (3.4.5) seka. Be to, nesunku pastebėti, jei kas antram sąrašo sąrašo L_{n-1} išskaidymui σ invertuosime (rašysime atvirkščia tvarka) (3.4.5) seką, tai sąrašo L_n gretimi išskaidymai mažai skirsis vienas nuo kito. Tiksliau kalbant, kiekvienas gretimas sąrašo L_n išskaidymas bus gautas iš prieš jį esančio išskaidymo pašalinant kažkokį elementą iš kažkurio bloko ir patalpinant šį elementą į kitą bloką, arba sukuriant iš jo vienelementinį bloką. Pavyzdžiui, jei $n = 3$, tai gausime tokius sąrašus:

$$L_1: \quad \{1\}$$

$$L_2: \quad \{1,2\}, \{1\}\{2\}$$

$$L_3: \quad \{1,2,3\}, \{1,2\}\{3\}, \{1\}\{2\}\{3\}, \{1\}\{2,3\}, \{1,3\}\{2\}.$$

Toliau aptarsime nerekurentinę šio metodo realizaciją.

Aibės $\{1,2,\ldots,n\}$ išskaidymo blokus surašysime jų mažiausio elemento didėjimo tvarka. Bloko mažiausią elementą mes vadinsime bloko numeriu. Reikia pažymėti, kad gretimų blokų numeriai bendru atveju nėra iš eilės einą natūralieji skaičiai.

Šiame algoritme naudosime kintamuosius $prec[i]$ ir $sek[i]$, $1 \le i \le n$, atitinkamai reiškiančius prieš ir po bloko, kurio numeris *i*, stovinčių blokų numerius. Jei blokas, kurio numeris *i*, yra paskutinysis išskaidymo blokas, tai $sek[i] = 0$.

Naudosime kintamąjį $blok[i]$, $i = \overline{1,n}$. Jei $blok[i] = k$, tai reiškia, kad *i*-tasis aibės $\{1,2,\ldots,n\}$ elementas priklauso blokui, kurio numeris *k*.

Elemento judėjimo kryptį nusakys kintamasis $pirmyn[i]$, $i = \overline{1,n}$. Jei $pirmyn[i] = true$, tai elementas *i* "juda" pirmyn.

Žemiau pateiktas algoritmas generuoja visus galimus aibės $\{1,2,\ldots,n\}$ išskaidymo būdus, kai duotas aibės elementų skaičius *n*. Algoritmo rezultatas: visų galimų aibės $\{1,2,\ldots,n\}$ išskaidymų seka, kurioje kiekvienas išskaidymas gaunamas iš prieš tai buvusio, perkeliant vienintelį elementą į kitą bloką.

```
begin
   for i := 1 to n do { Elementą i patalpinti į pirmąjį bloką }
      begin
         blok [i] := 1; pirmyn [i] := true;
```

```
    end;
sek [1] := 0;
Atspausdinti išskaidymą:
elementai, masyve blok pažymėti tuo pačiu numeriu k,
priklauso k-ajam blokui.
j := n; { j = aktyvusis elementas }
while j > 1 do
    begin
        k := blok [j];
        if pirmyn [j] then { j "juda" į priekį }
            begin
                if sek [k] := 0 then { k yra paskutinysis blokas }
                    begin
                        sek [k] := j; prec [j] := k; sek [j] := 0;
                    end;
                if sek [k] > j {j sudaro naują bloką }
                    begin
                        prec [j] := k;
                        sek [j] := sek [k];
                        prec [sek [j]] := j; sek [k] := j;
                    end;
                blok [j] := sek [k]
            end
                  else {j "juda" atgal }
                    begin
                        blok [j] := prec [k];
                        if k = j then { j sudaro vienelementį bloką }
                        if sek [k] = 0 then sek [prec [k]] := 0;
                                else
                                    begin
                                        sek [prec [k]] := sek [k];
                                        prec [sek [k]] := prec [k];
                                    end;
                    end;
            Atspausdinti išskaidymą.
            j := n;
            while ((j > 1) and ((pirmyn [j]) and (blok [j] = j))
            or ((not pirmyn [j]) and (blok [j] := 1))) do
                begin
                    pirmyn [j] := not pirmyn [j];
                    j := j – 1;
```

```
            end;
        end;
end;
```

Pateiktas algoritmas pradžioje generuoja išskaidymą $\{1,2,3,..., n\}$, t.y. pirmąjį aibės išskaidymą sąraše L_n, kuris gaunamas aukščiau aprašytu rekurentiniu metodu.

Pagrindinio ciklo "*while* $j > 1$ *do*" paskirtis yra "aktyvaus" elemento j radimas ir jo perkėlimas į kaimyninį bloką, t.y. į bloką iš dešinės, jei *pirmyn* [j] turi reikšmę *true* (šiuo atveju gali būti sukuriamas blokas $\{j\}$), ir į bloką iš kairės, jei *pirmyn* [j] = *false*.

Iš aukščiau pateikto rekurentinio metodo išplaukia, kad elementas j gali būti "judinamas", kai visi didesni už jį elementai pasiekia savo ribinę kairiąją arba dešiniąją padėtį. Kitaip tariant, "aktyvusis elementas" j^* yra toks elementas, kai visi didesni už jį elementai j ($j > j^*$) tenkina vieną iš dviejų sąlygų:

a) *pirmyn* [j] *and* (*blok* [j] = j), t.y. elementas juda pirmyn (į dešinę) ir pasiekia savo dešiniąją ribinę padėtį (aišku, kad j negali būti elementu bloko, kurio mažiausias elementas yra didesnis už j);
b) *not pirmyn* [j] *and* (*blok* [j] = 1), t.y. elementas juda atgal (į kairę) ir pasiekia savo kairiąją ribinę padėtį.

Aišku, kad, jei elementas j tenkina vieną iš minėtų sąlygų, tai keičiama jo judėjimo kryptis.

Paanalizuokime "aktyvaus" elemento perkėlimo į kitą bloką procesą.

Pirmiausia randamas numeris bloko, kuriame yra aktyvusis elementas. Tarkime, kad tokio bloko numeris yra k.

Jei "aktyvusis" elementas "juda" į **dešinę**, tai pakanka perkelti jį į bloką, kurio numeris yra $sek[k]$, jei $sek[k] \neq 0$ ir $sek[k] > j$.

Jei $sek[k] = 0$ arba $sek[k] > j$, tai pirmiausia reikia modifikuoti $sek[k]$ reikšmę.

Tarkime, *sek* [*k*] = 0. Vadinasi, "aktyviojo" elemento blokas yra paskutinysis išskaidymo blokas. Šiuo atveju bus sudaromas naujas blokas $\{j\}$. Tam tikslui pakanka priimti: $sek[k] := j$ ir atitinkamai pakeisti $sek[j]$ ir $prec[j]$ reikšmes (žr. algoritmo tekstą).

Tarkime, *sek* [*k*] > *j*. Tai reiškia, kad visi blokai, esantys dešiniau bloko k, sudaryti iš elementų, didesnių nei j (visi tie elementai užima ribinę dešinę padėtį). Vadinasi, (žr. rekurentinį metodą), šiuo atveju reikia sukurti vienelementį bloką $\{j\}$. Tai ir įvykdoma (žr. algoritmo tekstą).

Situacijoje, kai "aktyvusis" elementas juda **atgal (į kairę)**, pakanka patalpinti šį elementą į prieš bloką *k* stovintį bloką ir atlikti atitinkamus masyvų *sek* ir *prec* elementų pakeitimus.

Žemiau surašyti visi galimi aibės {1,2,3,4} išskaidymo būdai, apskaičiuoti pagal aukščiau pateiktą algoritmą.

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

Aišku, kad išskaidymų skaičius yra $B_4 = 15$.

3.4.4. Sveikųjų skaičių kompozicija ir išskaidymas

Nagrinėsime natūraliojo skaičiaus *n* išskaidymą į neneigiamų sveikųjų skaičių $\{p_1, p_2,..., p_k\}$ seką, kad $p_1 + p_2 + ... + p_k = n$.

Jei skaičių p_i tvarka sekoje $\{p_1, p_2,..., p_k\}$ yra svarbi, tai toks skaičiaus ***n išskaidymas*** vadinamas ***kompozicija***. Šiuo atveju paprastai įvedamas apribojimas: $p_i > 0$, $i = \overline{1,k}$.

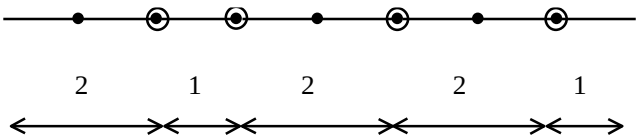
Jei skaičių p_i tvarka nesvarbi ir $p_i > 0$, tai $\{p_1, p_2,..., p_k\}$ yra multiaibė (aibė su pasikartojančiais elementais) ir vadinama skaičiaus ***n išskaidymu***.

Pavyzdžiui, jei $n = 3$, tai (3), (1, 2), (2, 1), (1, 1, 1) yra skaičiaus 3 kompozicija, o (3), (1, 2), (1, 1, 1) – išskaidymas.

Kompozicija

Nagrinėjant kompoziciją, į skaičių n patogu žiūrėti kaip į atkarpą, sudarytą iš vienetinio ilgio atkarpėlių, ir atkarpėlių susijungimo vietos pažymėtos taškais.

Pavyzdžiui, kai $n=8$, turėsime (žr. 3.4.2 pav.).



3.4.2 pav. Grafinis kompozicijos (2, 1, 2, 2, 1) vaizdavimas

Tada kompozicijos elementai yra atstumas tarp gretimų pažymėtų taškų. 3.4.2 pav. pavaizduota skaičiaus $n=8$ kompozicija (2, 1, 2, 2, 1). Aišku, kad kiekvieną kompoziciją galima vaizduoti dvejetainiu skaičiumi, turinčiu $(n-1)$ skiltį: vienetinė skiltis rodo pažymėtą tašką. Pavyzdžiui, kompozicijos (2, 1, 2, 2, 1) kodas yra $\begin{array}{ccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ \hline 0 & 1 & 1 & 0 & 1 & 0 & 1 \end{array}$.

n skaičiaus kompozicija, susidedanti iš k dalių, atitinka $(n-1)$-skiltį dvejetainį skaičių, turintį $(k-1)$ vienetuką, ir todėl egzistuoja C_{n-1}^{k-1} tokių kompozicijų.

Išskaidymas

Išskaidymas skiriasi nuo kompozicijos tuo, kad komponentų tvarka nesvarbi. Pavyzdžiui, išskaidymo atveju nedarome skirtumo tarp tokių skaičiaus 4 išskaidymų: $1+1+2$, $1+2+1$, $2+1+1$. Nagrinėjant skaičiaus n išskaidymą, patogu komponentes $\{p_1, p_2, ..., p_k\}$ išdėstyti didėjimo tvarka, t.y. $p_1 \le p_2 \le ... \le p_k$.

Skaičiaus n išskaidymą į l komponentes patogu generuoti leksikografine tvarka, pradedant išskaidymu $p_1 = p_2 = ... = p_{l-1} = 1$, $p_l = n-l+1$, ir tęsiant procesą tokiu būdu: norėdami gauti leksikografiškai didesnį išskaidymą nei nagrinėjamas, nagrinėjame elementus iš dešinės į kairę iki rasime pirmąjį elementą p_i, tenkinantį sąlygą $p_l - p_i \ge 2$; po to p_j keičiame p_i+1 visiems $j = i, i+1, ..., l-1$, ir elementui p_l suteikiame reikšmę $n - \sum_{j=1}^{l-1} p_j$.

Pavyzdžiui, jei $n=12$, $l=5$ ir nagrinėjame išskaidymą $\{1,1,3,3,4\}$, tai gausime, kad pirmasis iš dešinės vienetukas tenkina sąlygą $p_l - p_i \geq 2$. Todėl kitas išskaidymas bus $\{1,2,2,2,5\}$.

Jei nei vienas išskaidymo elementas netenkina sąlygos $p_l - p_i \geq 2$, tai generavimo procedūra baigiama.

Žemiau pateikiamas aptarto metodo algoritmas.

Skaičiaus n išskaidymų generavimo leksikografine tvarka algoritmas

```
begin
    l := 1;
    p [1] := n;
    p [0] := –1;
    while l ≤ n do
        begin
            spausdinti (p1, p2,..., pl);
            i := l – 1;
            while p [l] – p [i] < 2 do i := i – 1;
            if i ≠ 0 then for j := i to l – 1 do p [j] := p [i] + 1
                    else
                        begin
                            for j := 1 to l do p [j] := 1;
                            l := l + 1:
                        end;
             p[l] := n – Σ_{j=1}^{l–1} p_j ;
        end;
end;
```

Skaičiaus n išskaidymo algoritmą galima iš esmės pagerinti. Skaičiaus n išskaidymą galima užrašyti taip:

$$m_1 \bullet p_1, m_2 \bullet p_2,..., m_l \bullet p_l ,$$

čia m_i rodo, kad komponentė p_i į išskaidymą įeina m_i kartų, t.y.

$$n = \sum_{i=1}^{l} m_i p_i .$$

Pasinaudojant tokiu išskaidymo užrašymu ir generuojant išskaidymus taip, kad $p_1 > p_2 ... > p_l$, galima gauti efektyvesnį išskaidymų generavimo algoritmą nei aukščiau pateiktas.

Pavyzdžiui, kai $n = 7$, generuodami išskaidymus nurodyta tvarka, gausime tokią išskaidymų seką:

$$\{7 \bullet 1\} = \{1,1,1,1,1,1,1\},$$
$$\{1 \bullet 2, 5 \bullet 1\} = \{2,1,1,1,1,1\},$$
$$\{2 \bullet 2, 3 \bullet 1\} = \{2,2,1,1,1\},$$
$$\{3 \bullet 2, 1 \bullet 1\} = \{2,2,2,1\},$$
$$\{1 \bullet 3, 4 \bullet 1\} = \{3,1,1,1,1\},$$
$$\{1 \bullet 3, 1 \bullet 2, 2 \bullet 1\} = \{3,2,1,1\},$$
$$\{1 \bullet 3, 2 \bullet 2\} = \{3,2,2\},$$
$$\{2 \bullet 3, 1 \bullet 1\} = \{3,3,1\},$$
$$\{1 \bullet 4, 3 \bullet 1\} = \{4,1,1,1\},$$
$$\{1 \bullet 4, 1 \bullet 2, 1 \bullet 1\} = \{4,2,1\},$$
$$\{1 \bullet 4, 1 \bullet 3\} = \{4,3\},$$
$$\{1 \bullet 5, 2 \bullet 1\} = \{5,1,1\},$$
$$\{1 \bullet 5, 1 \bullet 2\} = \{5,2\},$$
$$\{1 \bullet 6, 1 \bullet 1\} = \{6,1\},$$
$$\{1 \bullet 7\} = \{7\}.$$

Žemiau pateiktame algoritme skaičiaus n išskaidymą pradėsime nuo išskaidymo $\{n \bullet 1\}$. Norėdami gauti nagrinėjamam išskaidymui $\{m_1 \bullet p_1, m_2 \bullet p_2, ..., m_l \bullet p_l\}$ gretimą išskaidymą, nagrinėsime patį dešinįjį išskaidymo elementą $m_l p_l$.

Jei $m_l > 1$, tai galime pašalinti du elementus p_l ir įvesti dar vieną elementą $p_l + 1$ (arba įvesti vieną elementą $p_l + 1$, jei tokio elemento nagrinėjamu momentu nebuvo).

Jei $m_l = 1$, tai suma $m_{l-1} p_{l-1} + m_l p_l$ pakankamai didelė, kad galėtume įvesti dar vieną $p_{l-1} + 1$. Bet kuriuo atveju tas, kas lieka, paverčiama į atitinkamą vienetukų skaičių.

Skaičiaus n išskaidymų generavimo "žodyno" tvarka algoritmas

begin
    l := 1;
    p [–1] := 0; m [–1] := 0;
    p [0] := n + 1;
    m [0] := 0;

```
p [1] := 1;
m [1] := n;
while l ≠ 0 do
    begin
        spausdinti {m1 • p1, m2 • p2,..., ml • pl};
        sum := m [l] * p [l];
        if m [l] = 1 then
            begin
                l := l – 1
                sum := sum + m [l] * p [l];
            end;
        if p [l – 1] = p [l] + 1 then
            begin
                l := l – 1;
                m [l] := m [l] + 1
            end
                        else
            begin
                p [l] := p [l] + 1;
                m [l] := 1;
            end;
        if sum > p [l] then
            begin
                p [l + 1] := 1;
                m [l + 1] := sum – p [l];
                l := l + 1
            end;
    end; { while }
end;
```

Aišku, kad pateiktas algoritmas yra tiesinis, kadangi skaičius operacijų, reikalingų pereinant nuo vieno išskaidymo prie kito, yra apribotas konstanta, nepriklausančia nei nuo *n*, nei nuo *l*.

3.5. Rekurentiniai sąryšiai

3.5.1. Rekurentinio sąryšio sąvoka ir pavyzdžiai

Sprendžiant daugelį kombinatorikos uždavinių, yra naudojami rekurentiniai sąryšiai[1] (lot. recurrence – grįžti). Naudodamiesi rekurentiniu sąryšiu, uždavinį su n objektų galime pakeisti uždaviniu su $(n-1)$ objektais, o šį – uždaviniu su $(n-2)$ objektais ir t.t. Paeiliui mažindami objektų skaičių, galų gale gausime uždavinį, kurį lengva išspręsti.

Panagrinėkime keletą pavyzdžių.

***Pirmas pavyzdys: kompiuterių mokslo taikymas* [*MKB86*]**

Tarkime, kad duotoje programavimo kalboje norime apskaičiuoti skaičių n ilgio išraiškų, sudarytų iš dešimties simbolių: 0, 1, 2, ..., 9 ir keturių aritmetinių veiksmų: +, –, ×, ÷. Tarkime, kad šios kalbos sintaksė reikalauja, kad kiekviena išraiška baigtųsi skaitmeniu, ir dvi išraiškos gali būti jungiamos, naudojant aritmetinės operacijos simbolį. Kitaip tariant, išraiška yra seka, sudaryta iš vieno arba kelių skaitmenų arba turi formą A • B, čia A ir B – išraiškos, o simbolis • žymi aritmetinę operaciją. Pavyzdžiui, 1 + 2 ir 3 × 45 yra išraiškos; 1 + 2 – 3 × 45 taip pat yra išraiška. Tačiau 1 + + 2 nėra išraiška (du aritmetiniai simboliai negali būti šalia).

Norėdami apskaičiuoti tokių n ilgio išraiškų skaičių, pasinaudosime rekurentiniu sąryšiu.

Simboliu a_n pažymėkime nagrinėjamų n ilgio išraiškų skaičių. Visas tas išraiškas išskaidykime į dvi klases. Pirmąją klasę sudarys išraiškos, kurios $(n-1)$-ojoje pozicijoje turi skaitmenį, o antrąją klasę sudarys išraiškos, kurių $(n-1)$-ojoje pozicijoje yra aritmetinės operacijos simbolis.

Kadangi išraiška turi baigtis skaitmeniu, tai pirmojoje klasėje bus $10 \cdot a_{n-1}$ išraiškų, t.y. kiekviena iš $(n-1)$ ilgio išraiškų gali būti pratęsta vienu iš 10-ties būdų.

Kadangi dvi aritmetinės operacijos negali eiti kartu, tai antrojoje klasėje bus $40 \cdot a_{n-2}$ išraiškų, t.y. $(n-2)$ ilgio išraiška gali būti pratęsta 4 aritmetinių operacijų simboliais $(n-1)$-oje pozicijoje ir 10 skaitinių simbolių n-tojoje pozicijoje. Vadinasi,

$$a_n = 10 \cdot a_{n-1} + 40 \cdot a_{n-2}. \qquad (3.5.1)$$

[1] Kitur literatūroje rekurentiniai sąryšiai vadinami skirtuminėmis lygtimis (difference equations).

Tam, kad galėtume pasinaudoti šia išraiška, reikia žinoti a_0 ir a_1. Aišku, kad $a_0 = 0$, o $a_1 = 10$, nes ilgio 1 išraiška gali būti tik skaitmuo.

Tuo būdu,

$$a_2 = 10a_1 + 40a_0 = 10 \cdot 10 + 40 \cdot 0 = 100 ,$$

$$a_3 = 10a_2 + 40a_1 = 10 \cdot 100 + 40 \cdot 10 = 1400 ,$$

$$a_4 = 10a_3 + 40a_2 = 10 \cdot 1400 + 40 \cdot 100 = 18000 ,$$

ir t.t.

Antras pavyzdys: Fibonačio skaičiai

1202 metais pasirodžiusioje knygoje ”Liber abaci” (Apie skaičiavimą) italų matematikas Fibonačis tarp kitų uždavinių pateikė tokį uždavinį.

Iš triušių poros kas mėnesį gaunamas dviejų triušiukų (*patinėlio ir patelės*) *prieauglis, o iš atvestųjų triušiukų po dviejų mėnesių jau gaunamas naujas prieauglis. Kiek triušių turėsime po metų, jei metų pradžioje turėjime vieną porą?*

Iš uždavinio sąlygos matyti, kad po pirmojo mėnesio turėsime dvi triušių poras. Po antrojo mėnesio turėsime tris poras, nes prieauglį davė tik pirmoji pora. Dar po mėnesio prieauglį duos ir pradinė pora, ir pora, atvesta prieš du mėnesius. Todėl iš viso bus 5 poros. Šio uždavinio sprendimui sudarysime rekurentinį sąryšį. Simboliu $F(n)$ pažymėkime triušių porų skaičių po n mėnesių, skaitant nuo metų pradžios. Aišku, kad n-ojo mėnesio pabaigoje turėsime $F(n-1)$ porą ir dar tiek naujų porų, kiek jų buvo $(n-2)$-ojo mėnesio pabaigoje, t.y. dar $F(n-2)$. Vadinasi,

$$F(n) = F(n-1) + F(n-2) . \qquad (3.5.2)$$

Kadangi iš sąlygos matyti, kad $F(0) = 1$, o $F(1) = 2$, tai rekurentinis sąryšis ir šios pradinės sąlygos nusakys triušių porų skaičių: $F_0 = 1$, $F_1 = 2$, $F_2 = 3$, $F_3 = 5$, $F_4 = 8$,..., $F_{11} = 233$, $F_{12} = 377$, ...

Gauti skaičiai vadinami Fibonačio skaičiais, o jų seka – Fibonačio skaičių seka.

Pastaba. Tą pačią seką gausime ir paėmę tokias pradines sąlygas $F_0 = 0$, $F_1 = 1$. Todėl tolesniame nagrinėjime imsime šias pradines sąlygas.

Su Fibonačio skaičiais susiduriama įvairiuose kombinatorikos uždaviniuose. Jie turi įdomių savybių. Dargi yra leidžiamas žurnalas “Fibonacci Quarterly” [MKB86].

Pavyzdžiui, Fibonačio skaičių seką generuoja ir tokio uždavinio sprendinys.

Reikia sužinoti, kiek yra n-narių $(n \geq 1)$ *sekų, sudarytų iš nulių ir vienetukų, kuriose nėra dviejų greta parašytų vienetų.*

Imkime bet kurią *n*-narę nulių ir vienetukų seką, tenkinančią sąlygą: du vienetai niekur joje nestovi drauge. Tokio sekos paskutinysis skaitmuo gali būti 0 arba 1. Suskirstykime nagrinėjamas *n*-nares sekas į dvi klases. Į pirmąją klasę talpinkime sekas, kurios baigiasi nuliu, o į antrąją – sekas, kurios baigiasi vienetu. Simboliu $T(n)$ pažymėkime *n*-narių sekų, tenkinančių uždavinio sąlygą, skaičių. Aišku, kad pirmosios klasės narių skaičius bus lygus $T(n-1)$, nes *n*-narė seka, atmetus *n*-ojoje pozicijoje stovintį nulį, duos $(n-1)$ ilgio seką, tenkinančią uždavinio sąlygą.

Antrojoje klasėje yra sekos, kurių *n*-ojoje pozicijoje yra 1. Tada, remiantis uždavinio sąlyga, teigiame, kad kiekvienos sekos $(n-1)$-ojoje pozicijoje yra 0. Vadinasi, jei tokioje sekoje atmestume du paskutiniuosius elementus: 0 ir 1, tai gautume $(n-2)$ ilgio seką, tenkinančią uždavinio sąlygą. Vadinasi, antrosios klasės sekų skaičius yra $T(n-2)$. Tuo būdu,

$$T(n) = T(n-1) + T(n-2). \qquad (3.5.3)$$

Kad ši rekurentinė išraiška generuotų Fibonačio skaičių seką, reikia, kad $T(1)$ $T(2)$ sutaptų su dviem iš eilės einančiais Fibonačio skaičiais.

Aišku, kad $T(1) = 2$ (sekos “0” ir “1”), o $T(2) = 3$ (sekos “00”, “01” ir “10”).

Kadangi $T(1) = F(1)$, o $T(2) = F(2)$, tai (3.5.3) rekurentinė išraiška generuos Fibonačio skaičius.

***Fibonačio skaičių savybės* [*Kn76*]**

Aptarsime paprasčiausias Fibonačio skaičių savybes.

1-oji savybė: $\sum_{i=1}^{n} F_i = F_{n+2} - F_2$.

Pastaba. Čia ir toliai vietoje simbolio $F(n)$ naudosime simbolį F_n. Be to, (3.5.2) išraiškos pradines sąlygas imsime $F_0 = 0$, $F_1 = 1$.

Įrodymas. Iš Fibonačio rekurentinės išraiškos gauname:

$$
\begin{aligned}
&F_1 = F_3 - F_2,\\
&F_2 = F_4 - F_3,\\
+\;&\text{– – – – – – – – –}\\
&\underline{F_n = F_{n+2} - F_{n+1},}\\
&\sum_{i=1}^{n} F_i = F_{n-2} - F_2.
\end{aligned}
$$

2-oji savybė. $F_1 + F_3 + F_5 + ... + F_{2n-1} = F_{2n}$.

Įrodymas.

$$
\begin{aligned}
&F_1 = F_2,\\
&F_3 = F_4 - F_2,\\
+\;&F_5 = F_6 - F_4,\\
&\text{– – – – – – – – –}\\
&\underline{F_{2n-1} = F_{2n} - F_{2n-2}.}\\
&F_1 + F_3 + ... + F_{2n-1} = F_{2n}.
\end{aligned}
$$

3-ioji savybė. $F_2 + F_4 + ... + F_{2n} = F_{2n+1} - 1$.

Įrodymas. Remiantis 1-ąją savybe gausime:

$$F_1 + F_2 + F_3 + ... + F_{2n} = F_{2n+2} - F_2 = F_{2n+2} - 1. \qquad (3.5.4)$$

Remiantis 2-ąją savybe gausime:

$$F_1 + F_3 + F_5 + ... + F_{2n-1} = F_{2n}. \qquad (3.5.5)$$

Iš (3.5.4) atėmę (3.5.5), gausime:

$$F_2 + F_4 + ... + F_{2n} = F_{2n+2} - 1 - F_{2n}.$$

Kadangi $F_{2n+1} = F_{2n+2} - F_{2n}$, tai $F_2 + F_4 + ... + F_{2n} = F_{2n+1} - 1$.

4-oji savybė: $\sum_{i=1}^{n} F_i^2 = F_n \cdot F_{n+1}$.

Įrodymas. Remsimės tapatybe:

$$F_k \cdot F_{k+1} - F_{k-1} \cdot F_k = F_k (F_{k+1} - F_{k-1}) = F_k \cdot F_k = F_k^2. \qquad (3.5.6)$$

Tada

$$
\begin{aligned}
&F_1^2 = F_1 \cdot F_2 - F_0 \cdot F_1,\\
&F_2^2 = F_2 \cdot F_3 - F_1 \cdot F_2,\\
+\;&\text{– – – – – – – – – – –}\\
&\underline{F_n^2 = F_n \cdot F_{n+1} - F_{n-1} \cdot F_n,}\\
&\sum_{i=1}^{n} F_i^2 = F_n \cdot F_{n+1}, \text{ nes } F_0 = 0.
\end{aligned}
$$

5-oji savybė. $F_{n+1} \cdot F_{n-1} - F_n^2 = (-1)^n$.

Įrodymas. Įrodysime matematinės indukcijos metodu.

Kai $n = 1$, tai $F_2 \cdot F_0 - F_1^2 = 1 \cdot 0 - 1^2 = -1$.

Tarkime, kad $F_{n+1} \cdot F_{n-1} - F_n^2 = (-1)^n$. Įrodysime, kad $F_{n+2} \cdot F_n - (F_{n+1})^2 = (-1)^{n+1}$.

$$
\begin{aligned}
&F_{n+2} \cdot F_n - (F_{n+1})^2 = (F_{n+1} + F_n)(F_{n-1} + F_{n-2}) - (F_n + F_{n-1})^2 =\\
&= F_{n+1} \cdot F_{n-1} + F_n \cdot F_{n-1} + F_{n+1} \cdot F_{n-2} + F_n \cdot F_{n-2} - F_n^2 - 2F_nF_{n-1} - F_{n-1}^2 =\\
&= F_{n+1} \cdot F_{n-1} - F_n^2 + F_n \cdot F_{n-2} - F_{n-1}^2 + F_{n+1} \cdot F_{n-2} - F_n \cdot F_{n-1} =\\
&= (-1)^n + (-1)^{n-1} + (F_n + F_{n-1}) \cdot F_{n-2} - F_n \cdot F_{n-1} =\\
&= F_n \cdot F_{n-2} + F_{n-1} \cdot F_{n-2} - F_n \cdot F_{n-1} =\\
&= F_n \cdot F_{n-2} - F_{n-1}^2 + F_{n-1}^2 + F_{n-1} \cdot F_{n-2} - F_n \cdot F_{n-1} =\\
&= (-1)^{n-1} + F_{n-1}(F_{n-1} + F_{n-2}) - F_n \cdot F_{n-1} =\\
&= (-1)^{n-1} + F_{n-1} \cdot F_n - F_n \cdot F_{n-1} = (-1)^{n-1} = (-1)^{n-1} \cdot \frac{(-1)^2}{(-1)^2} = (-1)^{n+1}.
\end{aligned}
$$

6-oji savybė. $F_{n+m} = F_m \cdot F_{n+1} + F_{m-1} \cdot F_n$.

(Įrodymas pagal indukciją.)

7-oji savybė. Sveikasis skaičius c yra F_m ir F_n dalikliu tada ir tiktai tada, kai jis yra skaičiaus F_d, čia $d = dbd(m,n)$ (dbd – didžiausias bendras daliklis) dalikliu.

Išvada. $dbd(F_m, F_n) = F_{dbd(m,n)}$.

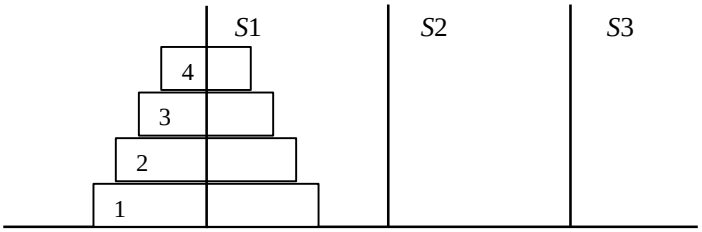
***Trečias pavyzdys. Hanojaus bokštų uždavinys*[2] [*DGP83*]**

Duoti trys strypai: $S1, S2$ *ir* $S3$. *Ant pirmojo strypo užmauta n skirtingo skersmens* $d_1, d_2, ..., d_n$ *diskų, tenkinančių sąlygą:* $d_1 > d_2 > ... > d_n$ (*žr. 3.5.1 pav.*). *Šiuos diskus reikia perkelti nuo strypo S*1 *ant kito strypo, pvz. S*3*, laikantis taisyklių:*

1) *kiekviename žingsnyje galima perkelti tik vieną viršutinį diską;*
2) *niekada negalima uždėti didesnio skersmens disko ant mažesnio skersmens disko;*
3) *bet kuriuo momentu visi diskai turi būti užmauti ant vieno iš strypų.*

Kiek diskų perkėlimų reikės atlikti?

Simboliu $h(n)$ pažymėkime veiksmų (disko perkėlimų) skaičių.



3.5.1 pav. Hanojaus bokštų uždavinys, kai *n* = 4

Tarkime, kad mokame perkelti $(n-1)$ diską. Tuomet *n* diskų perkeliame šiais trimis žingsniais:

1) $(n-1)$ viršutinių diskų perkeliame nuo pirmojo strypo ant antrojo, naudodamiesi laisvu trečiuoju strypu;
2) apatinį *n*-ąjį diską užmauname ant trečiojo strypo;
3) $(n-1)$ diskų nuo antrojo strypo perkeliame ant trečiojo naudodamiesi laisvu pirmuoju strypu.

Dabar galime sudaryti rekurentinį sąryšį, nusakantį $h(n)$. Aišku, kad

$$h(n) = 2h(n-1) + 1, \qquad (2.5.7)$$

[2] Šis uždavinys paimtas iš legendos, kurioje pasakojama, kad kadaise Hanojuje buvo pastatyta šventykla ir prie jos trys bokštai (strypai). Ant pirmojo bokšto buvo užmauti 64 skirtingo skersmens diskai taip, kad didžiausias būtų apačioje o mažiausias – viršuje. Tos šventyklos vienuoliai turėjo perkelti visus diskus nuo pirmojo bokšto ant trečiojo, laikantis uždavinyje nurodytų sąlygų. Legendoje sakoma, kad perkėlus diskus įvyks pasaulio pabaiga.

t.y. norint perkelti *n* diskų, reikės du kartus perkelti $(n-1)$ diskų ir dar vieną *n*-ąjį diską.

Šio rekurentinio sąryšio pradinė sąlyga yra $h(1)=1$.

(3.5.7) rekurentinis sąryšis generuos seką: $h(1)=1$;

$h(2)=2h(1)+1=2\cdot 1+1=3$, $h(3)=2h(2)+1=2\cdot 3+1=7$,

$h(4)=2h(3)+1=2\cdot 7+1=15$ ir t.t.

Iš pateiktų pavyzdžių matyti, kad visi rekurentiniai sąryšiai generuoja skaičių a_n sekas (žymėsime $\{a_n\}_{n=0}^{\infty}$), kurios priklauso tiek nuo rekurentinio sąryšio, tiek ir nuo pradinių sąlygų. Kitaip tariant, rekurentinis sąryšis su fiksuotom pradinėm sąlygom nusako natūraliojo argumento funkciją.

Apibrėžimas. *k*-osios eilės tiesiniu rekurentiniu sąryšiu vadinsime formulę, rišančią a_n su $a_{n-1}, a_{n-2},..., a_{n-k}$:

$$c_0(n)a_n + c_1(n)a_{n-1} + ... + c_k(n)a_{n-k} = \varphi(n),\ n \geq k,$$

čia $c_0(n), c_1(n),..., c_k(n)$ ir $\varphi(n)$ yra natūraliojo argumento funkcijos ir $c_0(n) \neq 0$, $c_k(n) \neq 0$.

Tam, kad sąryšis generuotų skaičių seką, reikia *k* pradinių sąlygų: $a_0, a_1,..., a_{k-1}$. Jei $c_0(n),..., c_k(n)$ yra konstantos, tai sakome, kad turime ***tiesinį rekurentinį sąryšį su pastoviais koeficientais***.

Jei $\varphi(n)=0$, tai rekurentinis sąryšis vadinamas ***homogeniniu***, priešingu atveju – ***nehomogeniniu***.

Pavyzdžiui,

a) $a_n - 3a_{n-1} + 2a_{n-2} = 0$,

b) $a_n - 3a_{n-1} + 2a_{n-2} = n^2 + 1$,

c) $a_n - (n-1)a_{n-1} - (n-1)a_{n-2} = 0$,

d) $a_n - 9a_{n-1} + 26a_{n-2} - 24a_{n-3} = 5n$,

e) $a_n - 3(a_{n-1})^2 + 2a_{n-2} = n$,

f) $a_n = a_0 \cdot a_{n-1} + a_1 \cdot a_{n-2} + ... + a_{n-1} \cdot a_0$,

g) $a_n^2 + (a_{n-1})^2 = -1$.

Visos rekurentinės išraiškos, išskyrus e), f) ir g), yra tiesinės. e) išraiška nėra tiesinė, kadangi turi narį $(a_{n-1})^2$. a), b) ir d) sąryšiai yra tiesiniai su pastoviais koeficientais. Pavyzdžiui, a) ir b) sąryšiai yra 2-osios eilės, o d) sąryšis yra 3-iosios eilės. a) ir c) sąryšiai yra homogeniniai, o b) ir d) – nehomogeniniai.

3.5.2. Rekurentinių sąryšių sprendimas

Turimą *k*-tosios eilės rekurentinį sąryšį

$$f(n) = a_1 f(n-1) + a_2 f(n-2) + ... + a_k f(n-k)$$

tenkina be galo daug skirtingų sekų. Mat *k* pirmųjų sekos narių (pradines sąlygas) galima pasirinkti laisvai, jie nėra susieti sąryšiais. Tačiau, kai *k* pirmųjų narių pasirinkta, visi kiti nariai apskaičiuojami vienareikšmiškai: narys $f(k+1)$ pagal rekurentinį sąryšį išreiškiamas nariais $f(1)$, $f(2)$, ..., $f(k)$, narys $f(k+2)$ – nariais $f(2)$, $f(3)$, ..., $f(k+1)$ ir t.t.

Remiantis rekurentiniu sąryšiu ir pirmaisiais sekos nariais, galima vieną po kito apskaičiuoti sekos narius ir anksčiau ar vėliau surasti bet kokį sekos narį. Tačiau tokiu atveju turime apskaičiuoti ir visus pirmesniuosius narius: juk jų nežinodami, negalime rasti tolesnių narių. Tačiau dažnai reikalingas tik vienas konkretus sekos narys, o kitų narių nereikia. Todėl labai svarbu žinoti rekurentinio sąryšio sprendinį: natūraliojo argumento funkciją, kuri generuoja tą pačią seką, kaip ir rekurentinis sąryšis.

Rekurentinio sąryšio sprendinį galima apibrėžti ir taip: natūraliojo argumento funkcija $f(n)$ yra rekurentinio sąryšio sprendinys, jei ją įstatę į sąryšį, gausime tapatybę.

Pavyzdys. Imkime rekurentinę išraišką

$$f(n) = 3f(n-1) - 2f(n-2).$$

Tada $f(n) = 2^n$ yra šios išraiškos sprendinys, nes

$$3 \cdot 2^{n-1} - 2 \cdot 2^{n-2} = 2^{n-2}(3 \cdot 2 - 2) = 2^n.$$

Reikia pabrėžti, kad $f(n) = 2^n$ yra ***atskirasis rekurentinio sąryšio sprendinys***.

Apibrėžimas. *k*-osios eilės rekurentinio sąryšio sprendinys vadinamas ***bendruoju sprendiniu***, jei jis priklauso nuo *k* laisvųjų konstantų $C_1,..., C_k$, kurias pasirinkus galima gauti bet kurį to sąryšio sprendinį.

Pavyzdžiui, sąryšio

$$f(n) = 3f(n-1) - 2f(n-2) \tag{3.5.8}$$

bendrasis sprendinys yra

$$f(n) = C_1 \ 2^n + C_2 1^n \tag{3.5.9}$$

Iš tikrųjų, lengva patikrinti, kad (3.5.9) funkcija (3.5.8) sąryšį paverčia tapatybe. Todėl belieka patikrinti, ar kiekvieną (3.5.8) sąryšio seką galima išreikšti (3.5.9) pavidalu. Kitaip tariant, reikia parodyti, kad bet kokioms $f(0)$ ir $f(1)$ reikšmėms egzistuoja konstantos C_1 ir C_2.

Įstatę (3.5.9) į (3.5.8), kai $n = 0$ ir $n = 1$, gausime:

$$\begin{cases} C_1 + C_2 = f(0), \\ 2C_1 + 3C_2 = f(1). \end{cases}$$

Kadangi sistemos determinantas nelygus nuliui, tai ši sistema visada turi sprendinį prie bet kokių $f(0)$ ir $f(1)$ reikšmių.

Tiesinių homogeninių rekurentinių sąryšių su pastoviais koeficientais sprendimas

Rekurentiniams sąryšiams spręsti, apskritai kalbant, bendrų metodų nėra. Tačiau labai dažnai pasitaikančių sąryšių klasė, kurią sudaro tiesiniai rekurentiniai sąryšiai su pastoviais koeficientais, sprendžiama bendru metodu.

Nagrinėsime tiesinį homogeninį sąryšį su pastoviais koeficientais (HR)

$$f(n) = a_1 f(n-1) + a_2 f(n-2) + ... + a_k f(n-k),$$

čia $a_1, a_2, ..., a_k$ – kokie nors skaičiai.

Aptarsime tokių sąryšių sprendimą. Yra keli sprendimo metodai [MKB86]:

1) pakeitimo metodas (kitaip vadinamas iteraciniu),
2) generuojančių funkcijų metodas,
3) charakteristinių šaknų metodas,
4) neapibrėžtų koeficientų metodas.

Dažniausiai naudojamas charakteristinių šaknų metodas, kurį čia ir aptarsime.

Homogeninių sąryšių sprendimas

Nesiaurindami klausimo, pirmiausia aptarsime 2-osios eilės su pastoviais koeficientais tiesinių homogeninių rekurentinių sąryšių sprendimo metodą. Šis metodas automatiškai taikomas ir aukštesnės eilės analogiškiems sąryšiams.

Imkime 2-osios eilės su pastoviais koeficientais rekurentinį homogeninį sąryšį

$$f(n) = a_1 f(n-1) + a_2 f(n-2), \qquad (3.5.9)$$

čia a_1 ir a_2 – realieji skaičiai.

Tokių sąryšių sprendimas yra analogiškas homogeninių diferencialinų lygčių su pastoviaisias koeficientais sprendimui: jei turime n tiesiškai nepriklausomų diferencialinės lygties sprendinių, tai jų tiesinis darinys yra bendras diferencialinės lygties sprendinys.

1 Lema. Jei $f_1(n)$ ir $f_2(n)$ yra tiesiškai nepriklausomi (3.5.9) rekurentinio sąryšio sprendiniai, tai bet kokiems skaičiams – konstantoms C_1 ir C_2 funkcija

$$f(n) = C_1 f_1(n) + C_2 f_2(n) \tag{3.5.10}$$

yra (3.5.9) sąryšio sprendinys.

Įrodymas. (3.5.10) įstatykime į (3.5.9) sąryšį. Kairioji (3.5.9) sąryšio pusė bus lygi

$$C_1 f_1(n) + C_2 f_2(n).$$

Apskaičiuokime kairiąją (3.5.9) sąryšio pusę.

$$a_1(C_1 f_1(n-1) + C_2 f_2(n-1)) + a_2(C_1 f_1(n-2) + C_2 f_2(n-2)) =$$
$$= C_1(a_1 f_1(n-1) + a_2 f_1(n-2)) + C_2(a_1 f_2(n-1) + a_2 f_2(n-2)) =$$
$$= C_1 f_1(n) + C_2 f_2(n),$$

nes $f_1(n)$ ir $f_2(n)$ yra atskiri (3.5.9) sąryšio sprendiniai.

2 lema. Jei r_1 yra kvadratinės lygties

$$r^2 = a_1 r + a_2$$

(ši lygtis vadinama charakteristine lygtimi) šaknis, tai $f(n) = r_1^n$ yra (3.5.9) išraiškos sprendinys.

Įrodymas. Kadangi r_1 yra charakteristinės lygties šaknis, tai

$$r_1^2 = a_1 r_1 + a_2.$$

Šią lygybę padauginę iš r_1^{n-2}, gausime:

$$r_1^n = a_1 r_1^{n-1} + a_2 r_1^{n-2},$$

t.y. $f(n) = r_1^n$ yra (3.5.9) sprendinys.

Teorema. (3.5.9) rekurentinio sąryšio bendrasis sprendinys apskaičiuojamas taip:

1) randame charakteristinės lygties $r_1^2 = a_1 r_1 + a_2$ šaknis r_1 ir r_2,
2) bendrąjį sprendinį užrašome:

$$f(n) = C_1 r_1^n + C_2 r_2^n, \text{ jei } r_1 \neq r_2,$$

$$f(n) = C_1 r^n + C_2 n r^n, \text{ jei } r_1 = r_2 = r.$$

Šios teoremos teisingumas tiesiogiai išplaukia iš 1-osios ir 2-osios lemos ir to fakto, kad nr^n yra (3.5.9) sprendinys, jei $r_1 = r_2 = r$.

Be to, kadangi $f_1(n)$ ir $f_2(n)$ yra tiesiškai nepriklausomi, tai konstantos C_1 ir C_2 egzistuoja prie bet kokių $f(0)$ ir $f(1)$ reikšmių.

Kaip minėjome aukščiau, *k*-osios eilės rekurentinė išraiška sprendžiama analogiškai. Čia reikia įvertinti tik tą faktą, kad, jei charakteristinės lygties šaknys $r_1 = r_2 = ... = r_s = r$, tai atskirieji tiesiškai nepriklausomi rekurentinio sąryšio sprendiniai yra $f_1(n) = r^n$, $f_2(n) = nr^n$, ..., $f_s(n) = n^{s-1} r^n$.

Pirmas pavyzdys. Išspręskime Fibonačio seką nusakantį rekurentinį sąryšį:

$$F_n = F_{n-1} + F_{n-2},$$
$$F_0 = 0,\ F_1 = 1.$$

Sudarome charakteristinę lygtį

$$r^2 = r + 1.$$

Šios lygties šaknys yra

$$r_1 = \frac{1+\sqrt{5}}{2},$$
$$r_2 = \frac{1-\sqrt{5}}{2}.$$

Vadinasi, bendrasis sprendinys yra

$$F_n = C_1\left(\frac{1+\sqrt{5}}{2}\right)^n + C_2\left(\frac{1-\sqrt{5}}{2}\right)^n.$$

Konstantas C_1 ir C_2 parinksime taip, kad jos atitiktų pradines sąlygas: $F_0 = 0,\ F_1 = 1.$

Gausime

$$C_1 + C_2 = 0,$$
$$C_1\frac{1+\sqrt{5}}{2} + C_2\frac{1-\sqrt{5}}{2} = 1.$$

Iš šios lygčių sistemos gausime:

$$C_1 = -C_2 = \frac{1}{\sqrt{5}}.$$

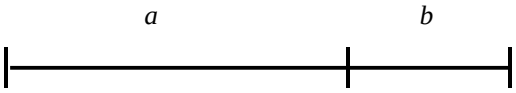
Vadinasi, Fibonačio seką generuoja tokia natūraliojo argumento funkcija

$$F_n = \frac{1}{\sqrt{5}}\left(\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n\right).$$

Seka F_n yra glaudžiai susijusi su auksinio piūvio santykiu ϕ.

Primename, kad atkarpa (žr. 3.5.2 pav.) auksiniu piūvio santykiu daloma į dvi dalis *a* ir *b* taip, kad

$$\frac{a}{b} = \frac{a+b}{a}.$$



3.5.2 pav. Auksinio piūvio santykis

Paėmę $b=1$, gausime

$$\frac{a}{1}=\frac{a+1}{a}=\phi,$$

$$\phi^2=\phi+1,$$

$$\phi=\frac{1\pm\sqrt{5}}{2}.$$

Kadangi ϕ dalo atkarpą vidiniu dalijimu, tai $\phi=\frac{1+\sqrt{5}}{2}$.

Simboliu $\hat{\phi}$ žymėsime $1-\phi$, t.y.

$$\hat{\phi}=1-\phi=1-\frac{1}{2}-\frac{\sqrt{5}}{2}=\frac{1-\sqrt{5}}{2}.$$

Vadinasi, Fibonačio seką galima užrašyti taip:

$$F_n=\frac{1}{\sqrt{5}}\left(\phi^n-\hat{\phi}^n\right).$$

Kadangi $|\hat{\phi}|<1$, tai $F_n\approx\frac{1}{\sqrt{5}}\phi^n$. Pasirodo (žr [Kn76]), kad, kai $n\geq 3$, $F_n=\frac{1}{\sqrt{5}}\phi^n$, apvalinant iki artimiausiojo sveikojo skaičiaus.

Pavyzdžiui, Fibonačio skaičių seka yra:

$F_0=0, F_1=1, F_2=1, F_3=2, F_4=3, F_5=5, F_6=8$ ir t.t.

Paėmę $n=3$, gausime: $\frac{1}{\sqrt{5}}\phi^3\cong 1{,}89$. Suapvalinę gausime 2. Paėmę $n=4$, gausime $\frac{1}{\sqrt{5}}\phi^4\cong 3{,}07$. suapvalinę gausime 3. . Paėmę $n=5$, gausime $\frac{1}{\sqrt{5}}\phi^5\cong 4{,}96$. suapvalinę gausime 5.

Antras pavyzdys. Išspręskime trečiosios eilės rekurentinį homogeninį sąryšį: $f(n)-9f(n-1)+26f(n-2)-24f(n-3)=0$, kai $f(0)=0$, $f(1)=1$, $f(2)=3$.

Sąryšio rekurentinė lygtis yra

$$r^3-9r^2+26r-24=0.$$

Šios lygties šaknys yra: $r_1=2$, $r_2=3$ ir $r_3=4$. Tada sąryšio bendrasis sprendinys yra

$$f(n)=C_1 2^n+C_2 3^n+C_3 4^n.$$

Apskaičiuosime konstantas:

$$\begin{cases} C_1+C_2+C_3=0, \\ 2C_1+3C_2+4C_3=1, \\ 4C_1+9C_2+16C_3=3. \end{cases}$$

Išsprendę lygčių sistemą, gausime

$$C_1=-2,\ C_2=3 \text{ ir } C_3=-1.$$

Vadinasi, nagrinėjamo rekurentinio sąryšio su nurodytomis pradinėmis sąlygomis sprendinys yra

$$f(n)=-2\cdot 2^n+3\cdot 3^n-4^n=-2^{n+1}+3^{n+1}-4^n.$$

Nehomogeninių sąryšių sprendimas

Panagrinėkime tiesinių nehomogeninių rekurentinių sąryšių su pastoviais koeficientais (NHR) sprendimą, kuris yra visiškai analogiškas tiesinių nehomogeninių diferencialinių lygčių su pastoviais koeficientais sprendimui.

Teorema. Tarkime, kad NHR turi pavidalą

$$f(n)+a_1 f(n-1)+...+a_k f(n-k)=\varphi(n).$$

Tarkime, kad HR $f(n)+a_1 f(n-1)+...+a_k f(n-k)=0$ yra nagrinėjamo NHR homogeninis sąryšis. Tada NHR bendrasis sprendinys randamas taip:

1) apskaičiuojamas HR bendrasis sprendinys $f_H(n)$,
2) apskaičiuojamas NHR atskirasis sprendinys $f_N(n)$,
3) šių sprendinių suma ir yra bendrasis NHR sprendinys:

$$f(n)=f_H(n)+f_N(n).$$

Pirmas pavyzdys. Raskime bendrąjį NHR $f(n)-7f(n-1)+10f(n-2)=4^n$, $n\geq 2$, sprendinį.

Pirmiausia rasime HR $f(n)-7f(n-1)+10f(n-2)=0$ sprendinį. Šios HR charakteristinė lygtis yra

$$r^2-7r+10=0,$$

o šaknys $r_1=2$, $r_2=5$. Tada

$$f_H(n)=C_1 2^n+C_2 5^n.$$

Apskaičiuosime atskirąjį NHR sprendinį. Jo ieškosime laisvojo nario $\varphi(n)$ pavidale:

$$f_N(n)=C4^n.$$

Įstatę $f_N(n)$ į NHR, gausime

$$C4^n-7C4^{n-1}+10C4^{n-2}=4^n,$$
$$C\cdot 4^{n-2}(4^2-7\cdot 4+10)=4^n,$$
$$C(16-28+10)=4^2,$$
$$-2C=16,$$
$$C=-8.$$

Vadinasi, $f_N(n)=-8\cdot 4^n=-2\cdot 4^{n+1}$.

Tada NHR bendrasis sprendinys yra

$$f(n)=C_1 2^n+C_2 5^n-2\cdot 4^{n+1}. \qquad (3.5.10)$$

Tarkime, kad turime tokias pradines sąlygas: $f(0)=8$, $f(1)=36$. Apskaičiuosime C_1 ir C_2. Į (3.5.10) formulę įstatę $n=0$ ir $n=1$, gausime:

$$\begin{cases} 8=C_1+C_2-8, \\ 36=2C_1+5C_2-32. \end{cases}$$

Išsprendę lygčių sistemą, gausime: $C_1=4$, o $C_2=12$.

Vadinasi, $f(n)=4\cdot 2^n+12\cdot 5^n-2\cdot 4^{n+1}$.

Antras pavyzdys. Išspręskime Hanojaus bokštų rekurentinį sąryšį:

$$h(n)-2h(n-1)=1,\ h(1)=1.$$

Hanojaus bokštų HR yra

$$h(n)-2h(n-1)=0.$$

Tada charakteristinė lygtis yra

$$r-2=0,$$

ir $r_1=2$.

HR bendrasis sprendinys bus

$$h_H(n)=C_1\cdot 2^n.$$

Kadangi 1-tas nėra charakteristinio polinomo šaknis, tai atskirojo NHR sprendinio ieškosime laisvojo nario pavidale

$$h_N(n) = C.$$

Apskaičiuosime C statydami $h_N(n)$ į NHR.

$$C - 2C = 1.$$

Iš čia $C = -1$.

Vadinasi,

$$h(n) = C_1 \cdot 2^n - 1. \qquad (3.5.11)$$

Apskaičiuosime C_1 įstatydami $n = 1$ į (3.5.11). Gausime:

$$1 = C_1 \cdot 2 - 1,$$

$$C_1 = 1.$$

Tokiu būdu $h(n) = 2^n - 1$.

Atskiro NHR sprendinio parinkimas

Kaip minėjome aukščiau, atskiro NHR sprendinio ieškosime pavidale, kuris priklauso nuo nehomogeninio rekurentinio sąryšio laisvojo nario formos $\varphi(n)$ ir nuo HR charakteristinės lygties šaknų.

Nagrinėjame nehomogeninį rekurentinį sąryšį

$$f(n) + a_1 f(n-1) + a_2 f(n-2) + ... + a_k f(n-k) = \varphi(n),$$

esant duotoms pradinėms sąlygoms, t.y. $f(0), f(1),..., f(k-1)$ reikšmėms.

NHR atskirojo sprendinio pavidalas, kai $\varphi(n) = Da^n$, čia D ir a – konstantos. Šiuo atveju atskirojo sprendinio $f_N(n)$ ieškosime pavidale

$$f_N(n) = \begin{cases} Ca^n, & \text{jei } CH(a) \neq 0, \\ Cn^m a^n, & \text{jei } a \text{ yra kartotinė } m \text{ kartų charakteristinio} \\ & \text{polinomo } CH(n) \text{ šaknis}, \end{cases}$$

čia ir toliau $CH(n)$ yra HR, atitinkančio nagrinėjamąjį NHR, charakteristinis polinomas, t.y. $r^k + a_1 r^{k-1} + ... + a_{k-1} r + a_k = 0$.

Pirmas pavyzdys. Spręskime NHR:

$$f(n) - 6f(n-1) + 8f(n-2) = 9, \quad f(0) = 10, \quad f(1) = 25.$$

Charakteristinis polinomas yra

$$r^2 - 6r + 8 = 0$$

ir jo šaknys yra: $r_1 = 4$, $r_2 = 2$.

Tada $f_H(n) = C_1 4^n + C_2 2^n$.

Kadangi šiame pavyzdyje $a=1$ ir a nėra charakteringojo polinomo šaknis, tai $f_N(n)$ ieškosime pavidale $f_N(n)=C$.

Įstatę $f_N(n)$ į NHR, gausime, kad $C=3$. Tuo būdu bendras NHR sprendinys

$$f(n)=C_1 4^5+C_2 2^n+3.$$

Įvertindami pradines sąlygas: $f(0)=10$, $f(1)=25$, gausime sistemą:

$$\begin{cases} C_1+C_2+3=10, \\ 4C_1+2C_2+3=25, \end{cases}$$

kurios sprendinys yra $C_1=4$, $C_2=3$.

Vadinasi, $f(n)=4\cdot 4^n+3\cdot 2^n+3.$

Antras pavyzdys. Išspręskime NHR:

$$f(n)-4f(n-1)+4f(n-2)=2^n.$$

HR charakteristinė lygtis yra

$$k^2-4k+4=(k-2)^2=0$$

ir jo šaknys yra: $k_1=2$, $k_2=2$. Todėl HR bendrasis sprendinys yra

$$f_H(n)=C_1 2^n+C_2 n2^n.$$

Kadangi 2 yra charakteristinės lygties du kartus kartotinė šaknis, tai $f_N(n)$ ieškosime pavidale:

$$f_N(n)=Cn^2 2^n.$$

Įstatę šią išraišką į NHR, gausime:

$$Cn^2 2^n-4C(n-1)^2 2^{n-1}+4C(n-2)^2 2^{n-2}=2^n.$$

$$C2^{n-2}(4n^2-8(n-1)^2+4(n-2)^2)=2^n,$$

$$8\cdot C\cdot 2^{n-2}=2^n,$$

$$8\cdot C\cdot 2^{-2}=1,$$

$$C=1/2.$$

Vadinasi, $f(n)=\frac{n^2}{2}2^n+C_1 2^n+C_2 n2^n$ yra NHR bendrasis sprendinys.

Teorema. Tarkime, kad $f_{1N}(n)$ yra atskirasis NHR

$$f(n)+a_1 f(n-1)+...+a_k f(n-k)=\varphi_1(n)$$

sprendinys, o $f_{2N}(n)$ yra atskirasis NHR

$$f(n)+a_1 f(n-1)+...+a_k f(n-k)=\varphi_2(n)$$

sprendinys, tai $f_N(n) = f_{1N}(n) + f_{2N}(n)$ yra atskirasis NHR

$$f(n) + a_1 f(n-1) + ... + a_k f(n-k) = \varphi_1(n) + \varphi_2(n)$$

sprendinys.

Pavyzdys. Raskime NHR

$$f(n) - 7f(n-1) + 10f(n-2) = 7 \cdot 3^n + 4^n$$

atskirąjį sprendinį.

Pirmiausia rasime $f(n) - 7f(n-1) + 10f(n-2) = 7 \cdot 3^n$ atskirąjį sprendinį $f_{1N}(n)$, po to – $f(n) - 7f(n-1) + 10f(n-2) = 4^n$ atskirąjį sprendinį $f_{2N}(n)$ ir juos sudėsime.

Nesunku įsitikinti, kad

$$f_{1N}(n) = -\frac{63}{2} \cdot 3^n,$$

o $f_{2N}(n) = -8 \cdot 4^n$.

Vadinasi, nagrinėjamosios NHR atskirasis sprendinys yra

$$f_N(n) = -\frac{63}{2} \cdot 3^n - 8 \cdot 4^n.$$

NHR atskirojo sprendinio pavidalas, kai $\varphi(n)$ ***yra polinomo ir eksponentės sandauga***:

$$\varphi(n) = (p_0 + p_1 n + ... + p_s n^s) \cdot a^n,$$

čia $p_i, i = \overline{0,s}$ ir a – konstantos.

Šiuo atveju atskirojo sprendinio pavidalas

$$f_N(n) = \begin{cases} (q_0 + q_1 n + ... + q_s n^s) a^n, \text{ jei } CH(a) \neq 0, \\ n^m (q_0 + q_1 n + ... + q_s n^s) a^n, \text{ jei } a \text{ yra } m \text{ kartų kartotinė} \\ \qquad\qquad \text{charakteristinio polinomo } CH(n) \text{ šaknis}, \end{cases}$$

čia $CH(n) = r^k + a_1 r^{k-1} + ... + a_{k-1} r + a_k = 0$.

Pavyzdys. Išspręskime NHR $f(n) - 5f(n-1) + 6f(n-2) = n^2 4^n$, $n \geq 2$.

Charakteristinio polinomo $r^2 - 5r + 6 = 0$ šaknys yra $r_1 = 2$, $r_2 = 3$.

Kadangi 4 nėra $CH(n)$ šaknis, tai

$$f_N(n) = (q_2 n^2 + q_1 n + q_0) \cdot 4^n.$$

Įstatę šį sprendinį į NHR, gausime:

$$(q_2 n^2 + q_1 n + q_0)\cdot 4^n - 5(q_2 (n-1)^2 + q_1 (n-1) + q_0)\cdot 4^{n-1} +$$
$$+ 6(q_2 (n-2)^2 + q_1 (n-2) + q_0)\cdot 4^{n-2} = n^2 4^n,$$
$$4^{n-2}((q_2 n^2 + q_1 n + q_0)\cdot 16 - 5(q_2 (n^2 - 2n + 1) + q_1 (n-1) + q_0)\cdot 4 +$$
$$+ 6(q_2 (n^2 - 4n + 4)^2 + q_1 (n-2) + q_0) = n^2 4^n.$$

Sudauginę ir sutraukę panašius narius, gausime:

$$2q_2 n^2 + (16q_2 + 2q_1)n + 4q_2 + 8q_1 + 2q_0 = 16n^2 .$$

Palyginę koeficientus prie tų pačių *n* laipsnių, gausime sistemą

$$\begin{cases} 2q_2 = 16, \\ 16q_2 + 2q_1 = 0, \\ 4q_2 + 8q_1 + 2q_0 = 0, \end{cases}$$

kurios sprendinys yra: $q_2 = 8$, $q_1 = -64$, $q_0 = 240$.

Vadinasi, $f_N(n) = (8n^2 - 64n + 240)\cdot 4^n$.

NHR atskirojo sprendinio pavidalas, kai $\varphi(n)$ ***yra polinomas,*** t.y.

$$\varphi(n) = p_0 + p_1 n + \ldots + p_s n^s .$$

Nesunku pastebėti, kad tai polinomo ir eksponentės sandaugos atskirasis atvejis, kai $a = 1$. Vadinasi,

$$f_N(n) = \begin{cases} q_0 + q_1 n + \ldots + q_s n^s, \text{ jei } CH(1) \neq 0, \\ n^m (q_0 + q_1 n + \ldots + q_s n^s), \text{ jei } 1\text{-tas yra } m \text{ kartų kartotinė} \\ \qquad \text{charakteristinio polinomo šaknis.} \end{cases}$$

Pavyzdys. Raskime $f_N(n)$ išraiškai

$$f(n) - 2f(n-1) + f(n-2) = 5 + 3n .$$

Kadangi 1-tas yra du kartus kartotinė charakteristinio polinomo $r^2 - 2r + 1 = 0$ šaknis, tai $f_N(n) = n^2 (q_0 + q_1 n)$. Įstatę $f_N(n)$ į NHR, gausime: $q_0 = 4$, $q_1 = 1/2$. Tuo būdu bendrasis NHR sprendinys yra

$$f(n) = C_1 + nC_2 + n^2 (4 + n/2) .$$

Išnagrinėtus atvejus galima sutraukti į 9 lentelę.

Iš pateikto nagrinėjimo galime padaryti išvadą, kad NHR atskirojo sprendinio $f_N(n)$ išraiška priklauso nuo NHR laisvojo nario – funkcijos $\varphi(n)$, tačiau $f_N(n)$ turi būti tiesiškai nepriklausomas nuo HR sprendinio $f_H(n)$.

9 lentelė. NHR atskirieji sprendiniai

$\varphi(n)$	$CH(n)$ – charakteristinis polinomas	NHR atskirojo sprendinio pavidalas
Da^n	$CH(a) \neq 0$	Ca^n
Da^n	a yra m kartų kartotinė $CH(n)$ šaknis	$Cn^m a^n$
$Dn^s a^n$	$CH(a) \neq 0$	$(q_0 + q_1 n + \ldots + q_s n^s)a^n$
$Dn^s a^n$	a yra m kartų kartotinė $CH(n)$ šaknis	$n^m(q_0 + q_1 n + \ldots + q_s n^s)a^n$
Dn^s	$CH(1) \neq 0$	$q_0 + q_1 n + \ldots + q_s n^s$
Dn^s	1 yra m kartų kartotinė $CH(n)$ šaknis	$n^m(q_0 + q_1 n + \ldots + q_s n^s)$

3.6. Generuojančios funkcijos

Kombinatorinių uždavinių, kuriuose reikia apskaičiuoti skaičių objektų, tenkinančių nurodytas sąlygas, sprendinys dažnai būna seka $a_0, a_1, a_2, ..., a_k, ...$, čia a_k – ieškomų k "matavimų" objektų skaičius. Pavyzdžiui, jei ieškome skaičiaus n išskaidymo, tai a_k yra skaičiaus n išskaidymo į k dėmenis variantų skaičius; jei nagrinėjame n-elementės aibės išskaidymą į poaibius, tai $a_k = C_n^k$ ir t.t. Šiais atvejais sekai $\{a_k\}_{k=0}^{\infty} = a_0, a_1, ..., a_k, ...$ patogu priskirti formalią eilutę

$$A(x) = \sum_{k=0}^{\infty} a_k x^k \tag{3.6.1}$$

kuri vadinama šią seką ***generuojančia funkcija***.

Pavadinimas "formalioji eilutė" reiškia, kad (3.6.1) eilutę traktuosime kaip sekos patogų užrašą. Šiuo atveju nesvarbu, prie kokių x reikšmių ši eilutė konverguoja. Todėl mes niekada neskaičiuosime tos eilutės sumos prie konkrečios x reikšmės. Su šiomis eilutėmis mes vykdysime konkrečias operacijas ir palyginsime koeficientus prie tų pačių x laipsnių. Tarkime, turime dvi eilutes:

$$A(x) = \sum_{k=0}^{\infty} a_k x^k \text{ ir } B(x) = \sum_{k=0}^{\infty} b_k x^k .$$

Apibrėšime šių eilučių ***sumą***.

$$A(x) + B(x) = \sum_{k=0}^{\infty} (a_k + b_k) x^k .$$

Apibrėšime ***sandaugos iš pastovaus skaičiaus*** operaciją.

$$pA(x) = \sum_{k=0}^{\infty} pa_k x^k .$$

Apibrėšime šių eilučių ***sandaugą***.

$$A(x) \cdot B(x) = \sum_{k=0}^{\infty} c_k x^k ,$$

čia $c_k = a_0 b_k + a_1 b_{k-1} + a_2 b_{k-2} + ... + a_k b_0 = \sum_{i=0}^{k} a_i b_{k-i}$.

Jei $a_k = 0$, kai $k > n$, tai (3.6.1) eilutę sutapatinsime su daugianariu:

$$a_n x^n + a_{n-1} x^{n-1} + \ldots + a_0 .$$

Aptarsime, kaip laipsninė eilutė ***dalijama*** iš kitos laipsninės eilutės. Padalinę eilutę $A(x)$ iš $B(x)$, gausime eilutę $C(x)$. Tada pagal dalybos veiksmo apibrėžimą galima parašyti

$$\begin{aligned} &a_0 + a_1 x + a_2 x^2 + ... + a_n x^n + ... = \\ &= (b_0 + b_1 x + ... + b_n x^n + ...) \cdot (c_0 + c_1 x + ... + c_n x^n + ...). \end{aligned} \tag{3.6.2}$$

(3.6.2) formulėje palyginę koeficientus prie tų pačių *x* laipsnių, gausime:

$$\begin{aligned} &a_0 = b_0 c_0, \\ &a_1 = b_0 c_1 + b_1 c_0, \\ &\text{– – – – – – – –} \\ &a_n = b_0 c_n + b_1 c_{n-1} + ... + b_n c_0, \\ &\text{– – – – – – – – – – – – – – –} \end{aligned} \tag{3.6.3}$$

Tos lygybės sudaro begalinę lygčių sistemą, iš kurios reikia rasti koeficientus c_0, c_1, c_2 ir t.t. Iš (3.6.3) nesunku apskaičiuoti koeficientus *c*:

$$\begin{aligned} &c_0 = \frac{a_0}{b_0}, \\ &c_1 = \frac{a_1 - b_1 c_0}{b_0}, \\ &\text{– – – – – – – –} \\ &c_n = \frac{a_n - \sum_{i=1}^{n} b_i c_{n-i}}{b_0}, \\ &\text{– – – – – – – – – – – – – – –} \end{aligned} \tag{3.6.4}$$

Iš matematinės analizės žinome, kad, jei eilutė kažkurioje nulio aplinkoje konverguoja, tai (3.6.1) eilutė yra funkcijos $A(x)$ Makloreno eilutė.

Šiuo atveju generuojanti funkcija yra matematinės funkcijos $A(x)$ Makloreno eilutė. Todėl, šiuo atveju, atliekant operacijas, vietoje eilutės galima imti analitinę funkciją $A(x)$.

Pavyzdžiui, galime rašyti:

$$\sum_{k=0}^{\infty} x^k = (1-x)^{-1}, \tag{3.6.5}$$

$$\sum_{k=0}^{\infty} \frac{1}{k!} x^k = e^x \text{ ir t.t.}$$

Apskaičiuokime kai kurių sekų generuojančias funkcijas.

Imkime seką $C_n^0, C_n^1, \ldots, C_n^n, \ldots$, čia C_n^k, kai $k > n$, yra lygus 0. Šios sekos generuojanti funkcija yra Niutono binomas, t.y.

$$\sum_{k=0}^{\infty} C_n^k x^k = (1+x)^n, \tag{3.6.6}$$

Panagrinėkime (3.6.6) formulės fizinę prasmę. Kiekvieną daugiklį $(1+x)$ galima traktuoti kaip aibės $X = \{a_1, a_2, \ldots, a_n\}$ atitikmenį, kuris nusako elemento a_i pasirodymo skaičių *X* poaibyje: 0 kartų (dėmuo $1 = x^0$) ir vieną kartą (dėmuo $x = x^1$). Aišku, kad kiekvienas aibės *X* poaibis vienareikšmiškai apibrėžiamas, nurodant, kiek kartų kiekvienas elementas a_i pasirodo poaibyje, t.y. iš kiekvieno daugiklio $(1+x)\cdot(1+x)\cdot\ldots\cdot(1+x)$ išrenkant po vieną dėmenį. Vadinasi, šios sandaugos koeficientas prie *x k*-tojo laipsnio ir parodo aibės *X k*-elemenčių poaibių skaičių.

Aišku, kad šį nagrinėjimą galima apibendrinti nagrinėjant aibes su pasikartojančiais elementais. Pavyzdžiui, imsime aibę $X = \{2\cdot a_1, 3\cdot a_2, 1\cdot a_1, 4\cdot a_1\}$, ir simboliu c_k pažymėsime šios aibės *k*-elemenčių poaibių skaičių. Apibendrinus aukščiau pateiktus samprotavimus, sekos $c_0, c_1, c_2, \ldots$ generuojanti funkcija yra

$$\sum_{k=0}^{\infty} c_k x^k = (1+x+x^2)\cdot(1+x+x^2+x^3)\cdot(1+x)\cdot$$
$$\cdot(1+x+x^2+x^3+x^4) = 1+4x+9x^2+15x^3+$$
$$+20x^4+22x^5+20x^6+15x^7+9x^8+4x^9+x^{10}.$$

Iš šios funkcijos matosi, kad skaičius poaibių, turinčių 5-is elementus, yra 22.

Nesunku suvokti, kad, atitinkamai parinkus *i*-ąjį daugiklį, galima įvesti įvairius apribojimus elementui a_i. Pavyzdžiui, jei šis daugiklis yra $1+x^3+x^7$,

tai reiškia, kad poaibyje elementas a_i gali pasirodyti 0, 3, arba 7 kartus; jei daugiklis yra $1+x^2+x^4+\ldots=(1-x^2)^{-1}$, tai reiškia, kad elementas a_i poaibyje gali pasirodyti lyginį skaičių kartų.

Jei mes neįvedame jokio apribojimo elemento a_i, $i=\overline{1,n}$, pasirodymo skaičiui, tai generuojanti funkcija yra

$$(1+x+x^2+...)\cdot(1+x+x^2+...)\cdot...\cdot(1+x+x^2+...)=$$
$$(1-x)^{-1}\cdot(1-x)^{-1}\cdot...\cdot(1-x)^{-1}=(1-x)^{-n}.$$

Aišku, kad aukščiau pateiktiems pavyzdžiams galima suteikti ir kitokią interpretaciją. Pavyzdžiui, nagrinėkime lygtį

$$e_1+e_2=r,$$

čia e_1, e_2, ir r – natūralieji skaičiai. Skaičiams e_1 ir e_2 gali būti įvesti apribojimai. Mus domina šios lygties sprendinių skaičius. Tarkime, e_1 gali būti arba 0, arba 1, arba 9, o e_2 tenkina nelygybę $0\le e_2\le 8$. Simboliu c_r, $r=0,1,2,\ldots$ pažymėkime šios lygties sprendinių skaičių. Tada šios sekos generuojanti funkcija yra dėmenų e_1 ir e_2 apribojimus nusakančių funkcijų $A_1(x)=(1+x+x^9)$ ir $A_2(x)=(1+x+\ldots+x^8)$ sandauga:

$$\sum_{k=0}^{\infty}c_k x^k=(1+x+x^9)\cdot(1+x+x^2+x^3+...+x^8).$$

Šį pavyzdį galima apibendrinti imant lygtį: $e_1+e_2+...+e_n=r$, e_i, r – natūralieji skaičiai. Norint sužinoti šios lygties sprendinių skaičių, kai e_i, $i=\overline{1,n}$, tenkina apribojimus $A_1(x),...,A_n(x)$, turime sudaryti generuojančią funkciją: $A_1(x)\cdot A_2(x)\cdot...\cdot A_n(x)$ ir koeficientas prie x^r duos atsakymą.

Imkime seką $1,2,4,8,...,2^k,...$. Šios sekos generuojanti funkcija yra

$$\sum_{k=0}^{\infty}2^k x^k=\sum_{k=0}^{\infty}(2x)^k=(1-2x)^{-1}$$ (žr. (3.6.5) formulę).

Remiantis tuo, kad analitinę funkciją galima panariui diferencijuoti, išveskime sekos $0,1,2,3,\ldots,n,...$ generuojančią funkciją.

$$\sum_{k=0}^{\infty}kx^k=x\sum_{k=0}^{\infty}kx^{k-1}=x\frac{d}{dx}\sum_{k=0}^{n}x^k=$$
$$=x\frac{d}{dx}(1-x)^{-1}=x(1-x)^{-2}.$$

Žemiau pateikiame lentelę, kurioje surašytos kai kurių sekų generuojančios funkcijos.

10 lentelė. Generuojančios funkcijos

Eil. nr.	Sekos	Generuojančios funkcijos
1	C_n^k	$(1+x)^n$
2	1	$(1-x)^{-1}$
3	a^n	$(1-ax)^{-1}$
4	$(-1)^n$	$(1+x)^{-1}$
5	$(-1)^n a^n = (-a)^n$	$(1+ax)^{-1}$
6	C_{n+k-1}^n	$(1-x)^{-n}$
7	$C_{n+k-1}^n a^n$	$(1+ax)^{-n}$
8	$C_{n+k-1}^n (-a)^n$	$(1+ax)^{-n}$
9	$n+1$	$(1-x)^{-2}$
10	n	$x(1-x)^{-2}$
11	$(n+2)(n+1)$	$2(1-x)^{-3}$
12	$(n+1)n$	$2x(1-x)^{-3}$
13	n^2	$x(1+x)(1-x)^{-3}$
14	$(n+3)(n+2)(n+1)$	$6(1-x)^{-4}$
15	$(n+2)(n+1)n$	$6x(1-x)^{-4}$
16	n^3	$x(1+4x+x^2)(1-x)^{-4}$
17	$(n+1)a^n$	$(1-ax)^{-2}$
18	na^n	$ax(1-ax)^{-2}$
19	$n^2 a^n$	$ax(1+ax)(1-ax)^{-3}$
20	$n^3 a^n$	$ax(1+4ax+a^2x^2)(1-ax)^{-4}$

3.7. Rekurentiniai sąryšiai ir generuojančios funkcijos

Kaip minėjome, rekurentiniai sąryšiai generuoja skaitines sekas, o šioms galima užrašyti generuojančias funkcijas. Šiame paragrafe panagrinėkime, kaip, turint tiesinį rekurentinį sąryšį, sudaryti jam atitinkančią generuojančią funkciją.

Tam tikslui panagrinėkime taisyklingąją algebrinę trupmeną $\frac{f(x)}{\varphi(x)}$, čia $f(x)$ ir $\varphi(x)$ – atitinkamai *n*-osios ir *m*-osios ($n < m$) eilės polinomai:

$$f(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_n x^n,$$
$$\varphi(x) = b_0 + b_1 x + b_2 x^2 + \ldots + b_m x^m,$$

ir $b_0 \neq 0$.

Aišku, kad

$$\frac{f(x)}{\varphi(x)} = c_0 + c_1 x + c_2 x^2 + \ldots + c_k x^k + \ldots \qquad (3.7.1)$$

Vadinasi,

$$a_0 + a_1 x + a_2 x^2 + \ldots + a_n x^n = (b_0 + b_1 x + b_2 x^2 + \ldots + b_m x^m) \cdot$$
$$\cdot (c_0 + c_1 x + c_2 x^2 + \ldots + c_k x^k + \ldots).$$

Sudauginę (3.7.1) dešiniąją pusę ir palyginę koeficientus prie tų pačių *x* laipsnių, gausime

$$\begin{aligned} & b_0 c_0 = a_0 \\ & b_0 c_1 + b_1 c_0 = a_1, \\ & b_0 c_2 + b_1 c_1 + b_2 c_0 = a_2, \\ & - - - - - - - - - - - - - - - - \\ & b_0 c_{m-1} + b_1 c_{m-2} + \ldots + b_{m-1} c_0 = a_{m-1}, \end{aligned} \qquad (3.7.2)$$

čia didžiausia galima *n* reikšmė yra $m-1$.

Visi tolesni sąryšiai bus vienodo pavidalo:

$$b_0 c_{m+k} + b_1 c_{m+k-1} + \ldots + b_m c_k = 0, \quad k = 0,1,2,\ldots . \qquad (3.7.3)$$

Vadinasi, (3.7.1) eilutės koeficientai $c_0, c_1, \ldots, c_n, \ldots$ tenkina (3.7.3) tiesinį rekurentinį sąryšį, o (3.7.2) apibrėžia šio rekurentinio sąryšio pradines sąlygas.

Atvirkščiai, tarkime, kad duotas *m*-osios eilės (3.7.3) rekurentinis sąryšis ir pradinės sąlygos $c_0, c_1, \ldots, c_{m-1}$. Tada šiam sąryšiui atitinkanti generuojanti funkcija yra trupmena

$$\frac{f(x)}{\varphi(x)} = \frac{a_0 + a_1 x + \ldots + a_{m-1} x^{m-1}}{b_0 + b_1 x + \ldots + b_m x^m} \qquad (3.7.4)$$

čia $b_0, b_1, \ldots, b_m$ – rekurentinio sąryšio koeficientai, o $a_0, a_1, \ldots, a_{m-1}$ – koeficientai, apskaičiuoti iš (3.7.2) lygčių.

Pastaba. Tarkime, $r = \min\limits_{1 \le i \le m} |x_i|$, čia x_i – polinomo

$b_0 + b_1 x + b_2 x^2 + \ldots + b_m x^m$ šaknys. Tada (3.7.1) eilutė konverguoja srityje $|x| < r$.

Pavyzdys. Sudarysime Fibonačio skaičių sekos generuojančią funkciją. Fibonačio skaičius generuojantis rekurentinis sąryšis yra

$$F_n - F_{n-1} - F_n = 0, \quad F_0 = 0, \quad F_1 = 1.$$

Vadinasi, Fibonačio skaičius generuojanti funkcija yra

$$\frac{f(x)}{\varphi(x)} = \frac{a_0 + a_1 x}{1 - x - x^2}.$$

Koeficientus a_0 ir a_1 apskaičiuosime pagal (3.7.2) formules:

$$a_0 = b_0 c_0 = 1 \cdot 0 = 0,$$

$$a_1 = b_0 c_1 + b_1 c_0 = 1 \cdot 1 + (-1) \cdot 0 = 1.$$

Tuo būdu, Fibonačio skaičių sekos generuojanti funkcija yra

$$A(x) = \frac{x}{1 - x - x^2}.$$

Iš pirmo žvilgsnio atrodo, kad mažai ką laimėjome, rekurentinį sąryšį pakeisdami generuojančia funkcija: juk vis tiek reikės skaitiklį dalyti iš vardiklio, o iš to gausime tą patį rekurentinį sąryšį. Tačiau (3.7.4) trupmeną galima tapačiai pertvarkyti, o tai palengvina skaičių c_k radimą.

Pavyzdys. Tarkime, generuojanti funkcija yra

$$\frac{x^3 - 2x^2 + 6x + 1}{x^4 - 5x^2 + 4}.$$

Išdėstykime šią trupmeną paprastesnėmis trupmenomis.

Šis uždavinys buvo spręstas matematinės analizės kurse. Tam tikslui raskime vardiklio polinomo šaknis:

$$x^4 - 5x^2 + 4 = (x^2 - 1)(x^2 - 4) = (x - 1)(x + 1)(x - 2)(x + 2).$$

Vadinasi,

$$\frac{x^3 - 2x^2 + 6x + 1}{x^4 - 5x^2 + 4} = \frac{A}{x - 1} + \frac{B}{x + 1} + \frac{C}{x - 2} + \frac{D}{x + 2}.$$

Abi lygybės puses padauginę iš bendro vardiklio ir palyginę skaitiklių koeficientus prie tų pačių x laipsnių, gausime:

$$\begin{cases} A + B + C + D = 1, \\ A - B + 2C - 2D = -2, \\ -4A - 4B - C - D = 6, \\ -4A + 4B - 2C + 2D = 1. \end{cases}$$

Šios sistemos sprendinys yra:

$$A=-1,\ B=-4/3,\ C=13/12 \text{ ir } D=9/4.$$

Trupmenos $\dfrac{A}{x-\alpha}$ eilutė žinoma. Pavyzdžiui,

$$\frac{13}{12}\frac{1}{x-2}=-\frac{13}{24}\left(1-\frac{x}{2}\right)^{-1}=-\frac{13}{24}\left(1+\frac{x}{2}+\frac{x^2}{2^2}+...+\frac{x^n}{2^n}+...\right).$$

Tokiu būdu, gausime:

$$\frac{x^3-2x^2+6x+1}{x^4-5x^2+4}=(1+x+x^2+...+x^n+...)-$$
$$-\frac{4}{3}\left(1-x+x^2-...+(-1)^n x^n+...\right)-$$
$$-\frac{13}{24}\left(1+\frac{x}{2}+\frac{x^2}{2^2}+...+\frac{x^n}{2^n}+...\right)+$$
$$+\frac{9}{8}\left(1-\frac{x}{2}+\frac{x^2}{2^2}-...+(-1)^n\frac{x^n}{2^n}+...\right).$$

Apskaičiavę koeficientą prie x^n, gausime:

$$c_n=1-\frac{4}{3}(-1)^n-\frac{13}{24\cdot 2^n}+\frac{9\cdot(-1)^n}{8\cdot 2^n}.$$

Tuo pačiu gavome generuojančiai funkcijai atitinkančio rekurentinio sąryšio sprendinį.

Šis nagrinėjimas duoda mintį, kaip galima spręsti rekurentinį sąryšį, panaudojant generuojančias funkcijas.

Tam tikslui reikia:

1) sudaryti rekurentiniam sąryšiui generuojančią funkciją $\dfrac{f(x)}{\varphi(x)}$,
2) $\dfrac{f(x)}{\varphi(x)}$ išreikšti elementariųjų trupmenų suma, o jas – laipsninėmis eilutėmis (pagal Niutono formulę),
3) gautos eilutės koeficientas c_n prie x^n ir bus ieškomasis sprendinys.

Pavyzdys. Išspręskime rekurentinį sąryšį:

$$f(n)-5f(n-1)+6f(n-2)=0,\ f(0)=1,\ f(2)=2.$$

Šiuo atveju $b_0=1$, $b_1=-5$ ir $b_2=6$.

Pagal (3.7.2) formules apskaičiuosime a_0 ir a_1:

$$a_0 = b_0 c_0 = 1,$$
$$a_1 = b_0 c_1 + b_1 c_0 = -7.$$

Vadinasi, rekurentinį sąryšį atitinkanti generuojanti funkcija yra

$$\frac{f(x)}{\varphi(x)} = \frac{-7x+1}{x^2-5x+6}.$$

Ši trupmena elementariosiomis trupmenomis išreiškiama taip:

$$\frac{-7x+1}{x^2-5x+6} = \frac{13}{x-2} - \frac{20}{x-3},$$

ir, išskaidę eilute, gausime:

$$-\frac{13}{2}\left(1+\frac{x}{2}+\frac{x^2}{2^2}+...+\frac{x^n}{2^n}+...\right)+\frac{20}{3}\left(1+\frac{x}{3}+...+\frac{x^4}{3^n}+...\right).$$

Todėl

$$f(n) = -\frac{13}{2}\frac{1}{2^n} + \frac{20}{3}\frac{1}{3^n} = -\frac{13}{2^{n+1}} + \frac{20}{3^{n+1}}.$$

Reikia pabrėžti, kad rekurentinių išraiškų sprendimas charakteristinių šaknų metodu yra žymiai efektyvesnis nei generuojančių funkcijų metodas.