4.4. *P* ir *NP* sudėtingumo uždaviniai

P sudėtingumo uždaviniai

Ankstesnėse paskaitose išnagrinėjome daug uždavinių ir beveik visais atvejais sudarėme algoritmus, kurių sudėtingumas – $\mathcal{O}(n^k)$ veiksmų. Čia n yra dydis, charakterizuojantis duomenų skaičių, k algoritmo sudėtingumo eilė. Tokius uždavinius vadiname *polinominio sudėtingumo* ir jų klasę žymime P (angl. *polynomial time computable*).

Nagrinėkime pirmą paprastą pavyzdį, kai sudedame ir dauginame dvi $n \times n$ dydžio matricas A ir B. Matricų sumą $A + B$ apskaičiuojame atlikę n^2 sudėties veiksmų, o tų pačių matricų sandaugos AB sudėtingumas yra $2n^3$ aritmetinių veiksmų.

Antrajame pavyzdyje rūšiuojame skaičių masyvą $\{v_1, v_2, \ldots, v_n\}$. Burbulo algoritmu šį uždavinį išsprendžiame atlikę $\mathcal{O}(n^2)$ veiksmų, o sparčiaisiais algoritmais šį masyvą surūšiuojame dar greičiau, užtenka tik $\mathcal{O}(n \log n)$ veiksmų.

Polinominio sudėtingumo uždavinius vadiname praktiškai išsprendžiamais uždaviniais, nes jų sprendimo laikas yra gana trumpas ir taikydami galime ieškoti tikslių sprendinių. Daugianario laipsnis k algoritmo sudėtingumo įvertyje turi būti nedidelis, nes net ir $\mathcal{O}(n^{10})$ sudėtingumo algoritmas yra neefektyvus nors kiek didesniam uždaviniui spręsti. Tačiau daugumos šiame vadovėlyje nagrinėjamų algoritmų sudėtingumas neviršija $\mathcal{O}(n^2)$ eilės.

NP sudėtingumo uždaviniai

Egzistuoja daug svarbių taikomųjų uždavinių, kuriems kol kas sukurti tik eksponentinio $\mathcal{O}(a^n)$ arba faktorialinio $\mathcal{O}(n!)$ sudėtingumo algoritmai. Labai svarbu išmanyti tokius uždavinius, nes taikydami juos sprendžiame tik apytiksliai ir naudojame polinominio sudėtingumo euristinius algoritmus.

Eksponentinių algoritmų pavyzdžius nagrinėjome spręsdami Hamiltono ciklo, keliaujančio pirklio ir kuprinės užpildymo uždavinius. Pateiksime dar vieną pavyzdį. Tarkime, kad reikia sudaryti visus skirtingus n daiktų išdėstymo variantus. Jų yra $n!$, todėl net ir tada, kai naują dėstinį gauname atlikę $\mathcal{O}(1)$ veiksmų, visus variantus rasti yra $\mathcal{O}(n!)$ sudėtingumo uždavinys. Ši funkcija didėja labai greitai.

Palyginkime n^2 ir 2^n sudėtingumo algoritmus. Padidinus duomenų skaičių dvigubai, polinominio sudėtingumo algoritmo skaičiavimo trukmė padidėja keturis kartus. Eksponentinio sudėtingumo algoritmo skaičiavimo trukmė padidėja du kartus, pridėjus tik vieną papildomą duomenį, ir keturis

kartus, pridėjus du duomenis. Visuose praktiniuose uždaviniuose toks uždavinio duomenų skaičiaus didėjimas yra visai nereikšmingas, bet skaičiavimo trukmė išauga labai smarkiai.

Šioje grupėje išskiriame NP (angl. *nondeterministic polynomial time*) uždavinių klasę. Pateiksime ne visai griežtą jos apibrėžimą.

Pirma, nagrinėjame tik *sprendimo priėmimo* uždavinius (angl. *decision problems*), kai reikia duoti atsakymą *taip* arba *ne* į pateiktą klausimą. Daugelį vadovėlyje jau išnagrinėtų uždavinių nesunku suformuluoti ir kaip sprendimo priėmimo uždavinius. Pavyzdžiui, galime klausti, ar duotajame grafe egzistuoja Hamiltono ciklas? Dažnai tenka spręsti optimizacijos uždavinius. Nagrinėkime uždavinį apie trumpiausio kelio tarp dviejų G grafo viršūnių $u, v \in V$ radimą. Tada pasirenkame skaičių k ir klausiame, ar egzistuoja kelias, jungiantis šias viršūnes, kurio ilgis nedidesnis už k?

Antra, NP klasės uždaviniams nežinome polinominio sudėtingumo sprendimo algoritmų, tačiau, atlikę $\mathcal{O}(n^k)$ veiksmų, galime patikrinti, ar duotasis objektas yra uždavinio sprendinys. Pavyzdžiui, jei turime skaičių masyvą $\{v_1, v_2, \ldots, v_n\}$, tai, lygindami visas skaičių poras (v_i, v_{i+1}), patikriname, ar masyvas surūšiuotas. Panašiai, jei duotas kelias p, jungiantis grafo viršūnes, tai, atlikę $\mathcal{O}(n)$ veiksmų, patikriname, ar tai Hamiltono ciklas.

Nagrinėsime NP klasės uždavinių specialius sprendimo algoritmus, sudarytus iš dviejų dalių.

1. Atlikę $\mathcal{O}(n^m)$ veiksmų, generuojame naują potencialų sprendinį (tai gali būti paprastas spėjimas ar sprendinys, kurį konstruojame naudodami atsitiktinių skaičių generatorių).

2. Naudodami polinominio sudėtingumo $\mathcal{O}(n^k)$ algoritmą (angl. *verification algorithm*), patikriname ar radome sprendinį. Jei spėjimas buvo nesėkmingas, tai vėl atliekame pirmąjį žingsnį.

Jeigu tikrinant sprendinį nepavyksta esmingai sumažinti bandymų skaičiaus, ir tikriname beveik visus galimus variantus, tai tokio algoritmo sudėtingumo funkcija yra $\mathcal{O}(a^n)$ arba $\mathcal{O}(n!)$ eilės.

Nagrinėkime du pavyzdžius. Pirmajame rūšiuojame n skaičių masyvą, o antrajame ieškome Hamiltono ciklo grafe, kurį sudaro n viršūnių. Tada egzistuoja $n!$ skirtingų šių skaičių ar viršūnių išdėstymo būdų. Tikrindami, ar pasirinktasis kelias p sudaro Hamiltono ciklą, negalime smarkiai sumažinti likusių variantų skaičiaus, visi šiuo metu žinomi šio uždavinio sprendimo algoritmai yra eksponentinio sudėtingumo.

Sprendžiant skaičių rūšiavimo uždavinį situacija yra kitokia: tikrindami, ar masyvas jau surūšiuotas, galime atmesti neperspektyvius variantus. Tarkime, kad tikrindami vykdome burbulo algoritmą ir sukeičiame vietomis tuos elementus, kuriems ši sąlyga negalioja. Tada, atlikę $\mathcal{O}(n)$ veiksmų, gauname naują masyvą, kurio didžiausias elementas jau atsidūrė reikiamoje vietoje. Taigi duomenų aibė sutrumpėja vienu vienetu, todėl skirtingų variantų lieka tik $(n-1)!$. Atlikę dar $\mathcal{O}(n-1)$ veiksmų vėl sumažiname variantų skaičių dar $(n-1)$ kartą (dabar jau du didžiausi elementai yra surūšiuoti). Kartodami burbulo algoritmo pagrindinį žingsnį, surūšiuojame visą masyvą, ir tokio algoritmo sudėtingumas yra $\mathcal{O}(n^2)$.

Taigi P klasės uždavinius galime greitai (t. y. atlikę $\mathcal{O}(n)$ veiksmų) išspręsti, o spręsdami NP klasės uždavinius, mokame tik greitai patikrinti sprendinį. Aišku, kad $P \subset NP$, t. y. kiekvienas polinominio sudėtingumo uždavinys priklauso ir NP klasei, nes, radę sprendinį galime nebetikrinti jo teisingumo.

Iš savo patirties žinome, kad išspręsti uždavinį yra daug sunkiau, nei patikrinti, ar turime sprendinį. Imkime Hamiltono ciklo radimo uždavinį. Net ir sparčiausių jo sprendimo algoritmų sudėtingumas yra $\mathcal{O}(2^n)$, o tikrindami sprendinį atliekame tik $\mathcal{O}(n)$ veiksmų. Tačiau nors šis skirtumas toks didelis ir žinome šimtus panašių pavyzdžių, kol kas neįrodyta, kad $P \neq NP$, t. y. nežinome nė vieno uždavinio, kurio sprendinio patikrinimo algoritmas yra polinominio sudėtingumo, o patį sprendinį galime rasti tik atlikę eksponentinį skaičių veiksmų. Tai įrodyti sudėtinga todėl, kad reikia nagrinėti visus įmanomus uždavinio sprendimo algoritmus bei pateikti apatinį sudėtingumo funkcijos įvertį $\Omega(n)$. Tokių įverčių sudarymo metodų yra labai nedaug ir jie dažniausiai tinka tik konkretiems algoritmams.

NP pilnieji uždaviniai

Klausimas apie uždavinių klasių P ir NP tapatumą yra pagrindinė algoritmų sudėtingumo teorijos problema. Šiame skirsnyje susipažinsime su dar viena uždavinių klase, kurios savybės smarkiai didina tikimybę, kad hipotezė $P \neq NP$ teisinga.

Matematikoje dažnai taikome sprendimo metodą, kai naują uždavinį transformuojame į kitą, jam ekvivalentų, o šį jau mokame spręsti. Nesunku įsitikinti, kad tarp kai kurių jau išnagrinėtų NP sudėtingumo uždavinių taip pat egzistuoja toks ryšys.

Nagrinėkime keliaujančio pirklio maršruto ir Hamiltono ciklo radimo uždavinius. Tarkime, kad žinome polinominio sudėtingumo algoritmą, apskai-

čiuojantį trumpiausią pirklio maršrutą bet kokiame pilnajame grafe. Tada visoms grafo G briaunoms $e_j \in E$ suteikime svorį, lygų vienetui. Jeigu dvi grafo viršūnės $v, w \in V$ nebuvo sujungtos briauna, tai apibrėžiame naują briauną, kurios svoris lygus 2. Tada apskaičiuojame trumpiausią pirklio maršrutą gautajame grafe. Jei jo ilgis lygus grafo G viršūnių skaičiui, tai radome Hamiltono ciklą, jei didesnis – toks ciklas neegzistuoja. Mums pavyko vieną uždavinį transformuoti į kitą. Deja, nemokame greitai spręsti keliaujančio pirklio uždavinio, todėl tokia transformacija negali padėti greitai patikrinti, ar egzistuoja Hamiltono ciklas.

Tarp NP sudėtingumo uždavinių apibrėžiame NP pilnųjų uždavinių poaibį, kurį žymėsime NPC (angl. *NP-complete*). Jam priklauso sudėtingiausi uždaviniai, kurie parenkami taip, kad jei pavyktų sukurti kurio nors šios klasės uždavinio polinominio sudėtingumo sprendimo algoritmą, tai ir bet kurį kitą NP klasės uždavinį galėtume išspręsti atlikę polinominį skaičių veiksmų.

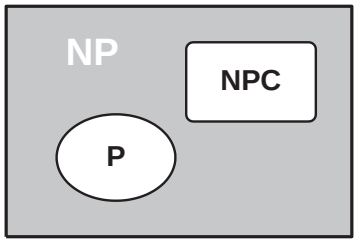
Sakome, kad uždavinys A yra polinomiškai transformuojamas į kitą uždavinį B, jei žinodami B sprendimo polinominio sudėtingumo algoritmą, galime sudaryti ir uždavinio A polinominio sudėtingumo sprendimo algoritmą.

Uždavinys A priklauso NPC klasei, jei $A \in NP$ ir kiekvieną NP klasės uždavinį polinominio sudėtingumo algoritmu galime transformuoti į A. Dažniausiai tai atliekame dviem etapais. Pirmiausia parodome, kad į A galime polinominio sudėtingumo algoritmu transformuoti kitą NP pilnąjį uždavinį B. Tada bet kurį NP klasės uždavinį C transformuojame į A, atlikdami tarpinę C uždavinio transformaciją į $B \in NPC$. Šis įrodymo būdas remiasi tokiu paprastu teiginiu.

4.6 teorema. *Jeigu uždavinys $B \in NPC$ ir jį galime polinomiškai transformuoti į kitą uždavinį $A \in NP$, tai $A \in NPC$.*

Pažymėtina, kad jei uždavinį $A \in NP$ galime polinomiškai transformuoti į kitą uždavinį $B \in NPC$, tai dar nereiškia, kad $A \in NPC$.

Tai, kad NP pilnųjų uždavinių klasė yra netuščia, 1971 m. įrodė S. Kukas. Šiuo metu yra žinomi keli šimtai tokių uždavinių, bet dar niekam nepavyko sukurti polinominio sudėtingumo algoritmo, išsprendžiančio kurį nors NPC klasės uždavinį. Todėl labai tikėtina, kad hipotezė $P \neq NP$ yra teisinga. 4.11 pav. pavaizduota schema, išreiškianti daugelio šios srities specialistų požiūrį apie uždavinių klasių P, NP ir NPC tarpusavio sąryšį.



4.11 pav. Hipotezė apie P, NP ir NPC sudėtingumo uždavinių sąryšį

NPC sudėtingumo uždavinių pavyzdžiai

Jau nagrinėjome kelis tokius uždavinius:

1. *Keliaujančio pirklio uždavinys.* Duotas svertinis grafas $G = (V, E)$, reikia rasti trumpiausią pirklio maršrutą, kai jis po vieną kartą aplanko visas grafo viršūnes ir grįžta į pradinę viršūnę.

2. *Hamiltono ciklas.* Reikia patikrinti ar duotajame grafe $G = (V, E)$ egzistuoja ciklas, jungiantis visas jo viršūnes.

3. *Diskretusis kuprinės užpildymo uždavinys.* Turime n daiktų, kurių tūriai yra $v_1, v_2, \ldots, v_n$, o kaina $p_1, p_2, \ldots, p_n$. Reikia rasti tokį daiktų rinkinį, kuris tilptų į V tūrio kuprinę ir krovinio vertė būtų didžiausia.

4. *Dėžių užpildymo uždavinys.* Turime keletą vienetinio tūrio dėžių ir n daiktų, kurių dydžiai v_1, v_2,..., v_n, čia $0 < v_j < 1$. Šiuos daiktus reikia sudėti į kuo mažesnį skaičių dėžių.

Apibrėždami NP uždavinių klasę, nagrinėjome tik sprendimo priėmimo uždavinius. Čia pateikėme ir optimizavimo uždavinius, todėl aptarsime sąryšį tarp abiejų tipų uždavinių.

Spręskime dėžių užpildymo uždavinį, tada formuluojame klausimą: ar galima visus daiktus sukrauti į k dėžių. Jei $k \geqslant n$, tai atsakymas trivialus – *taip*. Taip pat akivaizdu, kad atsakymas bus *ne*, jei $k < v_1 + v_2 + \ldots + v_n$. Tačiau pasirinkę k, artimą minimaliam dėžių skaičiui, gauname labai sudėtingą uždavinį, kurio nemokame spręsti polinominio sudėtingumo algoritmu.

Pateiksime dar tris NPC sudėtingumo uždavinius.

1. *Konjuktyvi normalioji forma.* Nagrinėjame Bulio algebros funkciją, kurios nariai yra jungiami loginiu operatoriumi AND (jį žymime $\wedge$),

o kiekvienas narys yra sudarytas iš Bulio kintamųjų arba jų neiginių, sujungtų loginiu operatoriumi OR (jį žymime $\vee$). Pateiksime tokios funkcijos pavyzdį:

$$(a \vee \bar{b}) \wedge (a \vee c \vee \bar{d}) \wedge (\bar{a} \vee c).$$

Reikia nustatyti, ar egzistuoja kintamųjų reikšmės, kada duotoji Bulio funkcija lygi vienetui. Priminsime, kad Bulio algebroje kiekvienas kintamasis gali įgyti tik dvi reikšmes – 0 arba 1. Kaip tik šį pavyzdį nagrinėjo S. Kukas, apibrėždamas pirmąjį NPC klasės uždavinį.

2. *Grafo viršūnių dažymo uždavinys.* Turime neorientuotąjį grafą $G = (V, E)$. Kiekvieną jo viršūnę nudažome kokia nors spalva. Kiek mažiausiai reikės parinkti spalvų, kad visos grafo briaunos jungtų skirtingų spalvų viršūnes?

3. *Darbų tvarkaraščio sudarymas.* Reikia atlikti $\{t_1, t_2, \ldots, t_n\}$ trukmės darbus, jų baigimo terminai – $\{d_1, d_2, \ldots, d_n\}$. Jei darbai nebus atlikti laiku, tai teks mokėti baudas $\{b_1, b_2, \ldots, b_n\}$. Kokia tvarka reikia vykdyti darbus, kad bauda būtų mažiausia?

Visus šiame paragrafe suformuluotus uždavinius realiai taikydami sprendžiame tik euristiniais algoritmais. Jų pavyzdžius pateikėme nagrinėdami godžiuosius algoritmus.