3 skyrius

Rūšiavimo algoritmai

3.1. Duomenų rūšiavimo uždaviniai

Beveik visose žmogaus veiklos srityse tenka saugoti informaciją ir ją panaudoti reikiamu momentu. Informacijos tvarkymo, paieškos uždavinys darosi vis aktualesniu, kai kompiuterizuojami abu proceso etapai: informacijos gamybos (pvz., skaitmeninis garso, vaizdo, rašytinių šaltinių saugojimas ir atgaminimas) bei jos panaudojimo (duomenų bazės, globalus Internetas). Jau nagrinėdami informacijos paieškos algoritmus minėjome, kad tokia paieška efektyviausia, kai saugome surūšiuotus duomenis. Taip pateikiama informacija ir daugelyje kasdien naudojamų informacijos šaltinių: kataloguose, žinynuose, enciklopedijose, telefonų knygose ir t.t.

Šiame poskyryje susipažinsime su svarbiausiais rūšiavimo algoritmais ir jų sudėtingumo įvertinimo metodais. Yra sukurta daug tokių algoritmų, todėl sprendžiant taikomuosius uždavinius, svarbu mokėti pasirinkti tinkamiausią mums algoritmą.

Uždavinio formulavimas

Tarkime, kad turime duomenų aibę $A = (a_1, a_2, \ldots, a_N)$. Šiuos duomenis galime palyginti, t.y. patikrinti ar teisingas teiginys, kad $a_i < a_j$, kai $i \neq j$. Reikia taip pertvarkyti A, kad gautoje aibėje $A' = (a'_1, a'_2, \ldots, a'_N)$ visi elementai būtų išdėstyti didėjimo tvarka, t.y. $a'_i \leq a'_{i+1}$, kai $i = 0, 1, \ldots, N-1$. Aišku, galime nagrinėti ir aibes, kuriose elementai išdėstyti atvirkščia – mažėjimo tvarka.

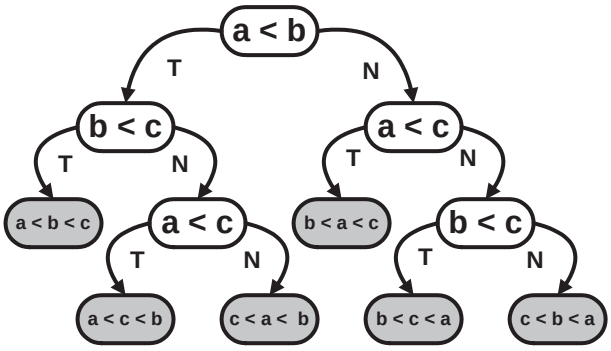
Rūšiavimo uždavinio sudėtingumas

Šiame poskyryje nagrinėsime tokius rūšiavimo algoritmus, kuriuos realizuojame panaudodami tik dvi operacijas

- dviejų aibės A elementų raktų palyginimą,
- dviejų elementų sukeitimą vietomis.

Juos galima pavaizduoti *binarinio medžio* pagalba. Medžio viršūnės žymi dviejų konkrečių elementų lyginimo veiksmą, šio medžio lapai apibrėžia surūšiuotas N elementų aibes.

Imkime paprastą pavyzdį, kai turime tris elementus $A = (a, b, c)$. Šią aibę surūšiuojame lygindami elementų poras. Visos galimos situacijos yra pavaizduotos 3.1 brėžinyje.



3.1 pav. Trijų elementų rūšiavimo algoritmas. T žymi briauną, kai tikrinamoji sąlyga yra teisinga, N – neteisinga

Matome, kad blogiausiu atveju teks atlikti tris elementų lyginimo veiksmus. Todėl toks ir bus bet kurio rūšiavimo algoritmo *blogiausiojo atvejo* sudėtingumas, kai rūšiuojame tris elementus.

Binarinio medžio aukštis (o tuo pačiu ir lyginimų skaičius) priklauso nuo lapų, kuriuos norime gauti, skaičiaus. Priminsime, kad h aukščio binarinis medis turi 2^h viršūnes–lapus. Imdami N elementų galime sudaryti $N!$ skirtingų kombinacijų, todėl bet kurio rūšiavimo algoritmo (taip pat ir geriausiojo, kokį galėsime sukurti) elementų porų lyginimo skaičius K yra nemažesnis už nelygybės

$$2^K \geq N!$$

sprendinį. Jei $N = 3$, tai $3! = 6$ ir vėl gauname, kad $K = 3$, t.y. reikia bent trijų lyginimo operacijų.

Bendruoju atveju, panaudodami Stirlingo formulę

$$\sqrt{2\pi n}\left(\frac{n}{e}\right)^n \le n! \le \sqrt{2\pi n}\left(\frac{n}{e}\right)^n e^{1/(12n)},$$

gauname, kad skirtingų elementų porų lyginimo skaičius bus nemažesnis už

$$K \ge N\log N - N\log e + \frac{1}{2}N.$$

Dar kartą priminsime, kad tai blogiausiojo galimo atvejo įvertis. Konkrečiu atveju gali užtekti ir daug mažiau operacijų, pavyzdžiui, kai elementų aibės pradinis pasiskirstymas yra artimas surūšiuotam.

Rūšiavimo uždavinio sudėtingumas mažėja ir tada, kai turime daugiau informacijos apie rūšiuojamus elementus. Tada vieno lyginimo metu galime gauti išsamesnę informaciją apie elementų tarpusavio išsidėstymą.

3.2. Paprasčiausi rūšiavimo algoritmai

Šiame poskyryje išnagrinėsime tris paprastus rūšiavimo algoritmus, kurie naudojami, kai rūšiuojame nedaug elementų. Išmoksime įvertinti algoritmų sudėtingumą ir įsitikinsime, kad jie nėra efektyvūs, kai aibės elementų skaičius yra didelis, nes atliekamų operacijų skaičius yra proporcingas N^2.

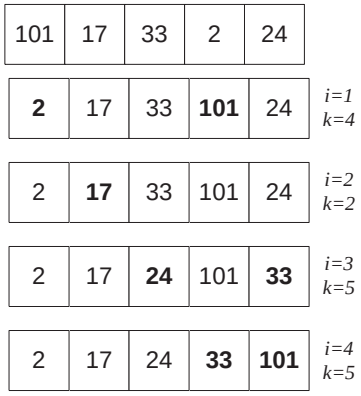
3.2.1. Išrinkimo algoritmas

Išrinkimo algoritmą (angl. *selection sort*) sudaro $N-1$ žingsnis. Jo i-ajame žingsnyje randame mažiausią elementą (tiksliau elementą, kurio raktas yra mažiausias) tarp aibės $i, i+1, \ldots, N$ elementų. Tarkime, kad tai yra p-asis elementas. Jei $i \neq p$, tai šiuos elementus sukeičiame vietomis.

Išrinkimo rūšiavimo algoritmas

```
(1)  for  ( i = 1;  i < N ;  i++ ) do
(2)       k = i;
(3)       for  ( j = i+1;  j <= N;  j++ ) do
(4)            if  ( a_j < a_k )  k = j;
          end do
(5)       if  ( k > i )  swap ( a_i, a_k );
     end do
```

3.1 pavyzdys. Skaičių masyvo rūšiavimas. Išrinkimo algoritmu surūšiuokime skaičių masyvą $E = (101, 17, 33, 2, 24)$. Rūšiavimo eiga pavaizduota 3.2 brėžinyje.



3.2 pav. Skaičių masyvo rūšiavimas išrinkimo algoritmu: i yra ciklo žingsnis, k yra mažiausiojo elemento indeksas, a_i ir a_k elementai sukeičiami vietomis

Algoritmo sudėtingumo įvertinimas. Vykdydami rūšiavimo algoritmą atliekame dvi pagrindines operacijas: dviejų elementų palyginimą ir elementų sukeitimą vietomis. Pažymėkime $L(N)$ elementų lyginimų skaičių, o $S(N)$ – elementų sukeitimų skaičių, kai rūšiuojame aibę E, turinčią N elementų. Priminsime, kad algoritmo sudėtingumą vertiname geriausio, vidutinio ir blogiausio duomenų pasiskirstymo atvejams.

Iš pateiktojo rūšiavimo algoritmo matome, kad (1) ir (3) ciklus įvykdome pilnai nepriklausomai nuo elementų pradinio pasiskirstymo, todėl visais atvejais

$$L(N) = \sum_{i=1}^{N-1} (N - i) = \sum_{i=1}^{N-1} i = \frac{N(N-1)}{2}.$$

Elementų keitimo vietomis skaičius priklauso nuo pradinio duomenų pasiskirstymo. Geriausiu atveju, kai aibės elementai jau yra surūšiuoti, nereikia atlikti nei vieno keitimo, t.y. $S_G(N) = 0$. Blogiausiu atveju tokius keitimus reikės atlikti kiekviename (1) ciklo žingsnyje, todėl $S_B(N) = N-1$.

Vidutinis įvertinimas priklauso nuo pradinio duomenų paskirstymo. Jei tarsime, kad visų galimų elementų išsidėstymų tikimybė yra vienoda, tai vidutinis keitimų skaičius yra artimas blogiausiojo atvejo įverčiui $S_B(N)$.

Kadangi vykdant išrinkimo algoritmą elementai perrašomi ne daugiau kaip $N-1$ kartą, tai šis metodas dažnai naudojamas, kai rūšiuojame nedideles aibes, kurių elementų informaciniai laukai yra ilgi.

Algoritmo stabilumas. Realizuodami rūšiavimo algoritmą mes sukeičiame kai kuriuos elementus vietomis. Jeigu elementai yra vienodi, tai jų tarpusavio išsidėstymas yra nesvarbus. Tačiau kai kada reikia garantuoti, kad algoritmas nepakeičia tokių elementų tarpusavio padėties, tokius rūšiavimo algoritmus vadinsime *stabiliais*.

> **3.2 pavyzdys. Prioritetinė eilė.** Pacientai poliklinikoje yra aptarnaujami eilės tvarka ir atsižvelgiama į jų prioritetą. Pirmiausia aptarnaujami sunkiai sergantys ligoniai, po to – pensininkai ir invalidai, ir po jų – visi kiti pacientai. Registratorė surašo visus ligonius ta tvarka, kokia jie atvyko į polikliniką, o vėliau šį sąrašą surūšiuoja atsižvelgdama į ligonių prioritetus. Jeigu rūšiavimo algoritmas yra stabilus, tai sudarytoje eilėje išsaugomas atvykimo eiliškumas tarp pacientų, turinčių tokį patį prioritetą.

Nesunku patikrinti, kad išrinkimo algoritmas yra stabilus.

3.2.2. Įterpimo algoritmas

Įterpimo algoritmą (angl. *insertion sort*) irgi sudaro $(N-1)$ žingsnis. Jo i-ajame žingsnyje įmame i-ąjį elementą ir jį įterpiame tarp jau surūšiuotų pirmųjų $(i-1)$ elementų.

Pastebėsime, kad taip lošėjai dažniausiai rūšiuoja išdalintas kortas.

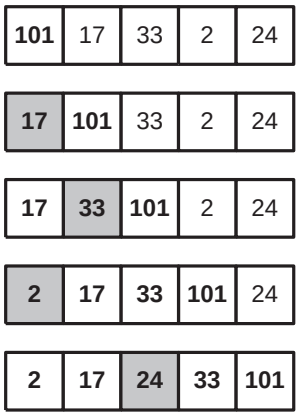
Įterpimo rūšiavimo algoritmas

```
(1)  for  ( i = 2;  i <= N;  i++ ) do
(2)      if  ( a_i < a_{i-1} )  then
(3)          v = a_i;   a_i = a_{i-1};
(4)          a_0 = v;   j = i-1;
(5)          while  ( v < a_{j-1} )  do
(6)                  a_j = a_{j-1};
(7)                  j = j-1;
             end do
(8)          a_j = v;
         end if
     end do
```

Realizuodami algoritmą panaudojome populiarų barjero metodą. Prieš pradėdami (5) ciklą fiktyviam nuliniam masyvo elementui priskiriame a_i reikšmę, tokiu būdu garantuojame, kad ciklas teisingai pasibaigs ir nereikia papildomai tikrinti masyvo indekso reikšmės.

Įterpimo rūšiavimo algoritmas irgi yra stabilus.

3.3 pavyzdys. Skaičių masyvo rūšiavimas. Įterpimo algoritmu surūšiuokime ankstesniojo pavyzdžio skaičių masyvą $E = (101, 17, 33, 2, 24)$. Rūšiavimo eiga pavaizduota 3.3 brėžinyje.



3.3 pav. Skaičių masyvo rūšiavimas įterpimo algoritmu: pilka spalva pavaizduotas elementas, kuris įterptas į surūšiuotą seką

Algoritmo sudėtingumo įvertinimas. Kiekviename įterpimo algoritmo (1) ciklo žingsnyje atliekame vienu elementų lyginimo veiksmu daugiau, nei jų keitimu vietomis, todėl

$$L(N) = S(N) + N - 1.$$

Geriausiuoju atveju, kai pradinė elementų aibė jau surūšiuota, gauname, kad lyginimų skaičius yra $L_G(N) = N - 1$. Jeigu elementai yra išdėstyti atvirkštine tvarka, tai kiekviename (1) ciklo žingsnyje atliekame visus tikrinimus, todėl *blogiausiuoju* atveju (čia įvertintas ir lyginimas su barjero elementu):

$$L_B(N) = \sum_{i=2}^{N} i = \frac{(N+2)(N-1)}{2} = \frac{N^2}{2} + \mathcal{O}(N)\,.$$

Vidutinis įvertinimas priklauso nuo pradinio duomenų paskirstymo. Kai elementų pradinio išsidėstymo visų variantų tikimybė yra vienoda, vidutiniškai atliekame pusę algoritmo išorinio (1) ciklo kiekvieno žingsnio tikrinimų, todėl

$$L_V(N) = \frac{N^2}{4} + \mathcal{O}(N)\,.$$

3.2.3. Burbulo algoritmas

Rūšiavimo uždavinį sprendžiame iš eilės lygindami du gretimus aibės A elementus (a_1, a_2), (a_2, a_3), $\ldots$, (a_{N-1}, a_N). Jeigu $a_i > a_{i+1}$, tai tokius elementus sukeičiame vietomis. Nesunku pastebėti, kad po pirmojo visų elementų patikrinimo didžiausias sekos elementas užims paskutiniojo elemento vietą. Todėl antrajame cikle užtenka tikrinti trumpesnę duomenų seką tik iki $(N-1)$-ojo elemento. Tokį procesą kartojame tol, kol eilinio ciklo metu nė karto nekeičiame elementų vietomis. Kadangi po kiekvieno ciklo seka sutrumpėja vienu elementu, tai blogiausiu atveju algoritmą teks kartoti $(N-1)$ kartą.

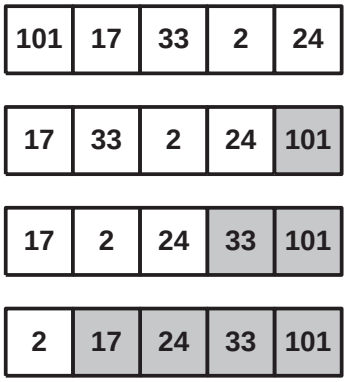
Pastebėję, kad po kiekvieno tikrinimų ciklo jau surūšiuoti visi sekos elementai, esantys po paskutiniojo perstatytojo elemento, dar labiau sumažiname tikrinamos sekos ilgį.

Kadangi didžiausiojo elemento judėjimas panašus į vandens burbulo kilimą į paviršių, tai toks rūšiavimo algoritmas vadinamas *burbulo* metodu (angl. *buble sort*). Ir burbulo rūšiavimo algoritmas yra stabilus.

Burbulo rūšiavimo algoritmas

```
(1)  k = n - 1;
(2)  while  ( k > 0 )  do
(3)         j = 0;
(4)         for  ( i = 1;  i <= k;  i++ )  do
(5)             if  ( a_i > a_{i+1} )  do
(6)                 j = i;
(7)                 swap ( a_i, a_{i+1} );
                end if
            end do
(8)         k = j-1;   // Visi elementai nuo a_{j+1} jau savo vietose
    end do
```

3.4 pavyzdys. Skaičių masyvo rūšiavimas. Burbulo algoritmu surūšiuokime ankstesniojo pavyzdžio skaičių masyvą $E = (101, 17, 33, 2, 24)$. Rūšiavimo eiga pavaizduota 3.4 brėžinyje.



3.4 pav. Skaičių masyvo rūšiavimas burbulo algoritmu, pilka spalva pažymėti jau surūšiuoti elementai

Algoritmo sudėtingumo įvertinimas. Įvertinsime burbulo algoritmo vykdymo kaštus.

Geriausiu atveju, kai pradinis elementų masyvas jau yra surūšiuotas, užteks tik vieną kartą įvykdyti burbulo algoritmo (4) ciklą, todėl

$$L_G(N) = N - 1, \quad S_G(N) = 0\,.$$

Tačiau *blogiausiu* atveju, kai pradiniame masyve elementai yra išdėstyti mažėjančia tvarka, reikės atlikti $N - 1$ algoritmo žingsnį, todėl tokio algoritmo vykdymo kaštai yra

$$L_B(N) = \frac{N(N-1)}{2}, \quad S_B(N) = \frac{N(N-1)}{2}\,.$$

Iš gautų įverčių matome, kad geriausiojo ir blogiausiojo atvejų kaštų skirtumas yra labai didelis. Dažniausiai reikia rūšiuoti daug skirtingų masyvų, kurių elementai yra pasiskirstę atsitiktinai. Todėl svarbu mokėti įvertinti *vidutinius* algoritmo vykdymo kaštus, kai nagrinėjame visus galimus elementų pasiskirstymus (jų yra net $N!$) ir laikome, kad jie visi yra vienodai tikėtini.

Jeigu tarsime, kad elementai masyve yra pasiskirstę atsitiktinai, tai kiekvieno tikrinimo metu vienodai tikėtina, kad tikrinimo sąlyga gali būti

tenkinama arba netenkinama. Todėl elementus sukeisti vietomis reikia du kartus rečiau, nei juos palyginti. Taigi užtenka tirti tik vieną iš šių operacijų, pavyzdžiui, rasti gretimų elementų lyginimų skaičių $L_V(N)$. Atlikę nesudėtingus skaičiavimus gausime, kad

$$L_V(N) = \frac{N(N - \log N)}{2} + \mathcal{O}(N)\,,$$

taigi vidutiniai burbulo algoritmo vykdymo kaštai yra artimi blogiausiam atvejui.

Elementų perstatymų skaičiaus minimizavimas

Nagrinėdami trijų rūšiavimo algoritmų sudėtingumą parodėme, kad vidutiniu ir blogiausiu atveju atliekami $\frac{1}{2}N^2 + \mathcal{O}(N)$ elementų lyginimo veiksmai. Tačiau elementų perstatymo skaičiai S_N esminiai skiriasi: burbulo ir įterpimo algoritmuose šis skaičius yra proporcingas $\mathcal{O}(N^2)$, o išrinkimo algoritme $S_{N,BV} = N - 1$. Todėl jei kiekvieno elemento informacinė dalis yra labai didelė, tai išrinkimo algoritmas bus efektyvesnis už kitus du rūšiavimo algoritmus.

Tokiais atvejais patartina rūšiuoti tik elementų raktus, o po to jau efektyviai sukeisti vietomis pačius elementus. Sukeitimo algoritmo paskutiniojo etapo sudėtingumas net ir blogiausiu atveju turi būti tik $\mathcal{O}(N)$ eilės dydis.

Apibrėžiame du naujus masyvus: masyve R saugome elementų raktus. Taigi rūšiuojame tik R elementus, o pagrindinio masyvo A elementų vietomis nekeičiame (tai didelės skaičiavimo apimties veiksmas, jei elemente saugome daug duomenų). Masyve B saugome elementų tvarką po rūšiavimo, t.y. galioja lygybė:

$$R(i) = a_{B(i)}.key, \quad i = 1, 2, \ldots, N\,.$$

Vykdydami raktų R rūšiavimo algoritmą, atitinkamai keičiame ir masyvo B elementus.

3.5 pavyzdys. Raktų masyvo rūšiavimas burbulo algoritmu. Burbulo algoritmu surūšiuokime raktus $R = (104, 103, 105, 101, 102)$. Rūšiavimo eiga ir masyvo B elementų kaita yra tokia:

$$\begin{pmatrix} 104 & 103 & 105 & 101 & 102 \\ 1 & 2 & 3 & 4 & 5 \end{pmatrix} \Rightarrow \begin{pmatrix} 103 & 104 & 101 & 102 & \mathbf{105} \\ 2 & 1 & 4 & 5 & 3 \end{pmatrix} \Rightarrow$$

$$\begin{pmatrix} 103 & 101 & 102 & \mathbf{104} & \mathbf{105} \\ 2 & 4 & 5 & 1 & 3 \end{pmatrix} \Rightarrow \begin{pmatrix} 101 & 102 & \mathbf{103} & \mathbf{104} & \mathbf{105} \\ 4 & 5 & 2 & 1 & 3 \end{pmatrix} \Rightarrow$$

$$\begin{pmatrix} \mathbf{101} & \mathbf{102} & \mathbf{103} & \mathbf{104} & \mathbf{105} \\ 4 & 5 & 2 & 1 & 3 \end{pmatrix}$$

Jeigu aibės A elementus nebūtina fiziškai surūšiuoti, tai surūšiuotą aibę A' gauname panaudodami atvaizdavimą

$$a'_i = a_{B(i)}, \quad i = 1, 2, \ldots, N\,.$$

Tokiu atveju papildomai tenka saugoti B masyvą.