Specialeji rūšiavimo algoritmai

Raimondas Čiegis

Matematinio modeliavimo katedra, e-paštas: rc@vgtu.lt

Lapkričio 5 d., 2023

Šioje paskaitoje susipažinsime su specialiaisiais rūšiavimo algoritmais, kurie yra spartesni už bendruosius algoritmus, kadangi naudojamės papildoma informacija apie aibės elementus arba uždavinys tenkina papildomus apribojimus.

Šioje paskaitoje susipažinsime su specialiaisiais rūšiavimo algoritmais, kurie yra spartesni už bendruosius algoritmus, kadangi naudojamės papildoma informacija apie aibės elementus arba uždavinys tenkina papildomus apribojimus.

Aišku, tokius uždavinius galima spręsti ir universaliais rūšiavimo algoritmais, pavyzdžiui QuickSort, bet specialieji algoritmai tą patį darbą atliks net ir asimptotiškai greičiau. Esame įrodę, kad blogiausiuoju atveju bet kurio universalaus algoritmo apatinis sudėtingumo įvertis yra $\Omega(N \log N)$.

Šioje paskaitoje susipažinsime su specialiaisiais rūšiavimo algoritmais, kurie yra spartesni už bendruosius algoritmus, kadangi naudojamės papildoma informacija apie aibės elementus arba uždavinys tenkina papildomus apribojimus.

Aišku, tokius uždavinius galima spręsti ir universaliais rūšiavimo algoritmais, pavyzdžiui QuickSort, bet specialieji algoritmai tą patį darbą atliks net ir asimptotiškai greičiau. Esame įrodę, kad blogiausiuoju atveju bet kurio universalaus algoritmo apatinis sudėtingumo įvertis yra $\Omega(N \log N)$.

Mūsų tikslas yra susipažinti su rūšiavimo algoritmais, kurie daug greičiau sprendžia specialius, bet svarbius taikymuose uždavinius, t.y. net ir blogiausiuoju atveju užtenka $\Theta(N)$ veiksmų.

Sveikųjų skaičių rūšiavimas Skaičiavimo algoritmu (angl. Counting Sort)

Reikia rūšiuoti duomenis, kurių raktai k yra sveikieji skaičiai ne didesni kaip K:

$$1 \leq a_i.key \leq K, \quad i = 1, \ldots, N.$$

Sveikųjų skaičių rūšiavimas Skaičiavimo algoritmu (angl. Counting Sort)

Reikia rūšiuoti duomenis, kurių raktai k yra sveikieji skaičiai ne didesni kaip K:

$$1 \leq a_i.key \leq K, \quad i = 1, \ldots, N.$$

Naudosime visai kito tipo rūšiavimo algoritmą, kuriame nereikia lyginti skirtingų elementų. Tokia galimybė labai keista ir prieštarauja mūsų intuicijai, nes atrodytų, kad be dviejų elementų palyginimo nerealu tikėtis juos surūšiuoti.

Inicializuojame masyvą L, kuriame saugome K tiesinių sąrašų, pradžioje visi sąrašai yra tušti.

Inicializuojame masyvą L, kuriame saugome K tiesinių sąrašų, pradžioje visi sąrašai yra tušti.

Perkeliame duomenis į tiesinius sąrašus, atitinkančius pasirinkto elemento raktą:

for j= 1, . . . , N
  $L[a_j.key]$.append(a_j)

Inicializuojame masyvą L, kuriame saugome K tiesinių sąrašų, pradžioje visi sąrašai yra tušti.

Perkeliame duomenis į tiesinius sąrašus, atitinkančius pasirinkto elemento raktą:

for j= 1, . . . , N
  $L[a_j.key]$.append(a_j)

Šios algoritmo dalies sudėtingumas $\Theta(N)$.

Antrame etape sujungiame visus gautus tiesinius sąrašus į vieną naują tiesinį sąrašą S, kuriame saugome jau surūšiuotus duomenis:

for k= 1, . . . , K
  S.extend($L[k]$)

Antrame etape sujungiame visus gautus tiesinius sąrašus į vieną naują tiesinį sąrašą S, kuriame saugome jau surūšiuotus duomenis:

for k= 1, . . . , K

S.extend($L[k]$)

Įvertinsime antrojo etapo sudėtingumą.

Antrame etape sujungiame visus gautus tiesinius sąrašus į vieną naują tiesinį sąrašą S, kuriame saugome jau surūšiuotus duomenis:

for k= 1, . . . , K
S.extend($L[k]$)

Įvertinsime antrojo etapo sudėtingumą.

Vieno $L[k]$ tiesinio sąrašo prijungimo kaštai yra $\Theta(|L[k]| + 1)$.
Visų sąrašų sujungimo sudėtingumas yra $\Theta(N + K)$.

Antrame etape sujungiame visus gautus tiesinius sąrašus į vieną naują tiesinį sąrašą S, kuriame saugome jau surūšiuotus duomenis:

for k= 1, . . . , K
  S.extend($L[k]$)

Įvertinsime antrojo etapo sudėtingumą.

Vieno $L[k]$ tiesinio sąrašo prijungimo kaštai yra $\Theta(|L[k]| + 1)$.
Visų sąrašų sujungimo sudėtingumas yra $\Theta(N + K)$.

Toks yra ir viso CountingSort algoritmo sudėtingumas.

Antrame etape sujungiame visus gautus tiesinius sąrašus į vieną naują tiesinį sąrašą S, kuriame saugome jau surūšiuotus duomenis:

for k= 1, . . . , K
S.extend($L[k]$)

Įvertinsime antrojo etapo sudėtingumą.

Vieno $L[k]$ tiesinio sąrašo prijungimo kaštai yra $\Theta(|L[k]| + 1)$.
Visų sąrašų sujungimo sudėtingumas yra $\Theta(N + K)$.

Toks yra ir viso CountingSort algoritmo sudėtingumas.

Jeigu duomenų raktų ilgiai yra aprėžti iš viršaus konstanta arba gali didėti, bet galioja įvertis $K \leq cN$ ir c yra mažas skaičius, pvz. $c = 2$, tai CountingSort rūšiavimo algoritmo asimptotinis sudėtingumas yra tiesinis

$$\Theta(N).$$

O jeigu rakto ilgis yra daug didesnis, pavyzdžiui $K = N^3$?

O jeigu rakto ilgis yra daug didesnis, pavyzdžiui $K = N^3$?

Akivaizdu, kad tada jau negalime rekomenduoti CountingSort algoritmo, nes jis ne tik, kad skaičiuos labai ilgai, bet dar reikės išskirti labai didelius atminties resursus.

O jeigu rakto ilgis yra daug didesnis, pavyzdžiui $K = N^3$?

Akivaizdu, kad tada jau negalime rekomenduoti CountingSort algoritmo, nes jis ne tik, kad skaičiuos labai ilgai, bet dar reikės išskirti labai didelius atminties resursus.

Bandysime modifikuoti algoritmą taip, kad visgi galėtume efektyviai spręsti ir tokius uždavinius.

Radix rūšiavimo algoritmas

Pateikdami algoritmą naudosime dešimtainę skaičiavimo sistemą. Tačiau galime imti bet kokį kitą skaičiavimo pagrindą b (dvejetainę, aštuntainę, šešioliktainę ar pagrindu $b = 36$ sistemą).

Radix rūšiavimo algoritmas

Pateikdami algoritmą naudosime dešimtainę skaičiavimo sistemą. Tačiau galime imti bet kokį kitą skaičiavimo pagrindą b (dvejetainę, aštuntainę, šešioliktainę ar pagrindu $b = 36$ sistemą).

Tarkime, kad turime A aibę, kurios elementai yra natūralieji n-ženkliai skaičiai:

$$0 \leqslant a_i < 10^n ,$$

tačiau nebūtinai visi šio intervalo skaičiai yra aibės elementai.

Radix rūšiavimo algoritmas

Pateikdami algoritmą naudosime dešimtainę skaičiavimo sistemą. Tačiau galime imti bet kokį kitą skaičiavimo pagrindą b (dvejetainę, aštuntainę, šešioliktainę ar pagrindu $b = 36$ sistemą).

Tarkime, kad turime A aibę, kurios elementai yra natūralieji n-ženkliai skaičiai:

$$0 \leqslant a_i < 10^n\,,$$

tačiau nebūtinai visi šio intervalo skaičiai yra aibės elementai.

RadixSort algoritmas gaunamas modifikuojant CountingSort algoritmą.

Pirmiausia visus elementus suskirstome į dešimt (bendruoju atveju b) poaibių pagal paskutinį rakto skaitmenį (mažiausiai reikšmingą arba vienetų poziciją).

Pirmiausia visus elementus suskirstome į dešimt (bendruoju atveju *b*) poaibių pagal paskutinį rakto skaitmenį (mažiausiai reikšmingą arba vienetų poziciją).

Paskui visus poaibius jų eiliškumo tvarka sujungiame į vieną aibę.

Pirmiausia visus elementus suskirstome į dešimt (bendruoju atveju *b*) poaibių pagal paskutinį rakto skaitmenį (mažiausiai reikšmingą arba vienetų poziciją).

Paskui visus poaibius jų eiliškumo tvarka sujungiame į vieną aibę.

Gautąją aibę vėl skirstome į dešimt poaibių pagal priešpaskutinį rakto skaitmenį (dešimčių poziciją).

Pirmiausia visus elementus suskirstome į dešimt (bendruoju atveju b) poaibių pagal paskutinį rakto skaitmenį (mažiausiai reikšmingą arba vienetų poziciją).

Paskui visus poaibius jų eiliškumo tvarka sujungiame į vieną aibę.

Gautąją aibę vėl skirstome į dešimt poaibių pagal priešpaskutinį rakto skaitmenį (dešimčių poziciją).

Šį procesą kartojame n kartų.

Surūšiuosime dviženklių sveikųjų skaičių masyvą

$$A = (73,\ 29,\ 92,\ 14,\ 74,\ 45,\ 54,\ 18,\ 3,\ 97,\ 9,\ 61,\ 11,\ 63,\ 35,\ 37).$$

Surūšiuosime dviženklių sveikųjų skaičių masyvą

$$A = (73, 29, 92, 14, 74, 45, 54, 18, 3, 97, 9, 61, 11, 63, 35, 37).$$

Aibę suskirstome į dešimt poaibių pagal paskutinį rakto skaitmenį

$$\begin{aligned}
0 &: \\
1 &: \quad 61,\ 11\,, \\
2 &: \quad 92 \\
3 &: \quad 73,\ 3,\ 63\,, \\
4 &: \quad 14,\ 74,\ 54\,, \\
5 &: \quad 45,\ 35\,, \\
6 &: \\
7 &: \quad 97,\ 37\,, \\
8 &: \quad 18\,, \\
9 &: \quad 29,\ 9\,.
\end{aligned}$$

Visus poaibius jų eiliškumo tvarka sujungiame į vieną aibę

$$A = (61,\ 11,\ 92,\ 73,\ 3,\ 63,\ 14,\ 74,\ 54,\ 45,\ 35,\ 97,\ 37,\ 18,\ 29,\ 9).$$

Visus poaibius jų eiliškumo tvarka sujungiame į vieną aibę

$$A = (61, 11, 92, 73, 3, 63, 14, 74, 54, 45, 35, 97, 37, 18, 29, 9).$$

Gautąją aibę vėl skirstome į dešimt poaibių dabar pagal pirmąjį rakto skaitmenį

$$\begin{aligned}
0 &: \quad 03,\ 09\,, \\
1 &: \quad 11,\ 14,\ 18\,, \\
2 &: \quad 29\,, \\
3 &: \quad 35,\ 37\,, \\
4 &: \quad 45\,, \\
5 &: \quad 54\,, \\
6 &: \quad 61,\ 63\,, \\
7 &: \quad 73,\ 74\,, \\
8 &: \\
9 &: \quad 92,\ 97\,.
\end{aligned}$$

Sujungę poaibius, gauname surūšiuotą elementų aibę

$$A = (3, 9, 11, 14, 18, 29, 35, 37, 45, 54, 61, 63, 73, 74, 92, 97).$$

Sujungę poaibius, gauname surūšiuotą elementų aibę

$$A = (3,\ 9,\ 11,\ 14,\ 18,\ 29,\ 35,\ 37,\ 45,\ 54,\ 61,\ 63,\ 73,\ 74,\ 92,\ 97).$$

Šiame pavyzdyje algoritmas sėkmingai surūšiavo duomenis. Bet ar taip bus ir pasirinkus kitokius duomenis?

Sujungę poaibius, gauname surūšiuotą elementų aibę

$$A = (3,\ 9,\ 11,\ 14,\ 18,\ 29,\ 35,\ 37,\ 45,\ 54,\ 61,\ 63,\ 73,\ 74,\ 92,\ 97).$$

Šiame pavyzdyje algoritmas sėkmingai surūšiavo duomenis. Bet ar taip bus ir pasirinkus kitokius duomenis?

Įrodysime, kad įvykdžius Radix algoritmą visada teisingai surūšiuojame aibę.

Sujungę poaibius, gauname surūšiuotą elementų aibę

$$A = (3, 9, 11, 14, 18, 29, 35, 37, 45, 54, 61, 63, 73, 74, 92, 97).$$

Šiame pavyzdyje algoritmas sėkmingai surūšiavo duomenis. Bet ar taip bus ir pasirinkus kitokius duomenis?

Įrodysime, kad įvykdžius Radix algoritmą visada teisingai surūšiuojame aibę.

Užtenka išnagrinėti dviženklių skaičių atvejį, o bendrojo n-ženklių skaičių atvejo teisingumas įrodomas indukcijos metodu.

Imkime du dviženklius skaičius X ir Y:

$$X = 10\,a + b, \quad Y = 10\,c + d, \quad 0 \leqslant a, b, c, d \leqslant 9.$$

Imkime du dviženklius skaičius X ir Y:

$$X = 10\,a + b, \quad Y = 10\,c + d, \quad 0 \leqslant a, b, c, d \leqslant 9.$$

Nelygybė $X < Y$ yra teisinga, jei

$$(a < c) \quad \text{arba} \quad (a = c)\,\&\,(b < d).$$

Imkime du dviženklius skaičius X ir Y:

$$X = 10\,a + b, \quad Y = 10\,c + d, \quad 0 \leqslant a, b, c, d \leqslant 9.$$

Nelygybė $X < Y$ yra teisinga, jei

$$(a < c) \quad \text{arba} \quad (a = c)\,\&\,(b < d).$$

Jei $a < c$, tai antrajame Radix rūšiavimo algoritmo žingsnyje X patenka ¿ poaibį su mažesniu numeriu nei Y.

Imkime du dviženklius skaičius X ir Y:

$$X = 10\,a + b, \quad Y = 10\,c + d, \quad 0 \leqslant a, b, c, d \leqslant 9.$$

Nelygybė $X < Y$ yra teisinga, jei

$$(a < c) \quad \text{arba} \quad (a = c)\,\&\,(b < d).$$

Jei $a < c$, tai antrajame Radix rūšiavimo algoritmo žingsnyje X patenka ¡ poaibį su mažesniu numeriu nei Y.

Jei $(a = c)\,\&\,(b < d)$, tai pirmuoju Radix rūšiavimo algoritmo žingsniu X pateko ¡ poaibį su mažesniu numeriu nei Y. Po antrojo žingsnio abu elementai pateks ¡ tą patį poaibį, bet X bus ¡trauktas anksčiau.

Imkime du dviženklius skaičius X ir Y:

$$X = 10\,a + b, \quad Y = 10\,c + d, \quad 0 \leqslant a, b, c, d \leqslant 9.$$

Nelygybė $X < Y$ yra teisinga, jei

$$(a < c) \quad \text{arba} \quad (a = c)\,\&\,(b < d).$$

Jei $a < c$, tai antrajame Radix rūšiavimo algoritmo žingsnyje X patenka į poaibį su mažesniu numeriu nei Y.

Jei $(a = c)\,\&\,(b < d)$, tai pirmuoju Radix rūšiavimo algoritmo žingsniu X pateko į poaibį su mažesniu numeriu nei Y. Po antrojo žingsnio abu elementai pateks į tą patį poaibį, bet X bus įtrauktas anksčiau.

Taigi abiem atvejais elementai surūšiuojami teisingai.

Įvertinsime, kiek bazinių veiksmų reikia atlikti rūšiuojant N elementų, kurių raktai $1 \leq k \leq K$ ir jie užrašyti skaičiavimo pagrindu b.

Įvertinsime, kiek bazinių veiksmų reikia atlikti rūšiuojant N elementų, kurių raktai $1 \le k \le K$ ir jie užrašyti skaičiavimo pagrindu b.

Nagrinėdami CountSort algoritmo sudėtingumą parodėme, kad realizuojant vieną Radix Sort algoritmo žingsnį tokių veiksmų yra $\Theta(N + b)$.

Žingsnių skaičius $n = \log_b K$, todėl viso algoritmo kaštai yra

$$\Theta\big((N + b)\log_b K\big).$$

Įvertinsime, kiek bazinių veiksmų reikia atlikti rūšiuojant N elementų, kurių raktai $1 \leq k \leq K$ ir jie užrašyti skaičiavimo pagrindu b.

Nagrinėdami CountSort algoritmo sudėtingumą parodėme, kad realizuojant vieną Radix Sort algoritmo žingsnį tokių veiksmų yra $\Theta(N + b)$.

Žingsnių skaičius $n = \log_b K$, todėl viso algoritmo kaštai yra

$$\Theta\big((N + b)\log_b K\big).$$

Optimalus skaičiavimo pagrindas yra $b = N$, tada gauname tokį algoritmo sudėtingumo įvertį

$$\Theta\big(N \log_N K\big).$$

Imkime aukščiau suformuluotą uždavinį, kai $K = N^3$.

Tada $\log_N K = 3$ ir Radix rūšiavimo algoritmo sudėtingumo funkcija vėl yra tiesinė $\Theta(N)$.

Išorinio rūšiavimo uždavinys

Nagrinėsime rūšiavimo algoritmus, kai duomenų yra tiek daug, kad jie visi netelpa operatyviojoje kompiuterio atmintyje ir juos saugome talpesnėje, bet daug lėtesnėje išorinėje atmintyje.

Išorinio rūšiavimo uždavinys

Nagrinėsime rūšiavimo algoritmus, kai duomenų yra tiek daug, kad jie visi netelpa operatyviojoje kompiuterio atmintyje ir juos saugome talpesnėje, bet daug lėtesnėje išorinėje atmintyje.

Tada lėčiausia operacija yra duomenų perrašymas iš sparčiosios atminties į išorinę atmintį ir atvirkščiai. Svarbiausia yra minimizuoti tokių veiksmų apimtį.

Pateiksime suliejimo algoritmo modifikaciją, pritaikytą išorinio rūšiavimo uždaviniui spręsti.

Pateiksime suliejimo algoritmo modifikaciją, pritaikytą išorinio rūšiavimo uždaviniui spręsti.

Reikia surūšiuoti N duomenų, saugomų rinkmenoje F. Tarkime, kad operatyviojoje atmintyje telpa tik M elementų.

- Iš rinkmenos F skaitome M dydžio duomenų blokus, juos rūšiuojame kokiu nors sparčiuoju vidinio rūšiavimo algoritmu ir paeiliui užrašome į rinkmenas $F1, F2$. Paskutinio bloko ilgis gali būti ir mažesnis už M.

Pateiksime suliejimo algoritmo modifikaciją, pritaikytą išorinio rūšiavimo uždaviniui spręsti.

Reikia surūšiuoti N duomenų, saugomų rinkmenoje F. Tarkime, kad operatyviojoje atmintyje telpa tik M elementų.

- Iš rinkmenos F skaitome M dydžio duomenų blokus, juos rūšiuojame kokiu nors sparčiuoju vidinio rūšiavimo algoritmu ir paeiliui užrašome į rinkmenas $F1, F2$. Paskutinio bloko ilgis gali būti ir mažesnis už M.
- Suliejimo algoritmu sujungiame M ilgio blokus iš rinkmenų $F1, F2$, gautuosius $2M$ ilgio blokus įrašome paeiliui į rinkmenas $F3, F4$. Algoritmą kartojame tol, kol gauname surūšiuotą N ilgio bloką, užrašytą vienoje iš rinkmenų.

Skaičių masyvo rūšiavimas suliejimo algoritmu

Rinkmenoje F užrašytas skaičių masyvas, kurio ilgis $N = 29$:

$$(4,\ 5,\ 2,\ 8,\ 4,\ 1,\ 7,\ 9,\ 2,\ 3,\ 0,\ 3,\ 8,\ 6,\ 2, 4,\ 9,\ 3,\ 9,\ 5,\ 0,\\ 4,\ 6,\ 2,\ 5,\ 3,\ 5,\ 1,\ 0).$$

Skaičių masyvo rūšiavimas suliejimo algoritmu

Rinkmenoje F užrašytas skaičių masyvas, kurio ilgis $N = 29$:

$$(4,\ 5,\ 2,\ 8,\ 4,\ 1,\ 7,\ 9,\ 2,\ 3,\ 0,\ 3,\ 8,\ 6,\ 2, 4,\ 9,\ 3,\ 9,\ 5,\ 0,$$
$$4,\ 6,\ 2,\ 5,\ 3,\ 5,\ 1,\ 0).$$

Imkime $M = 3$, tada masyvo rūšiavimo eigos pradžia yra tokia:

$$M = 3:$$
$$\mathrm{F1} = (\,2,\ 4,\ 5 \mid 2,\ 7,\ 9 \mid 2,\ 6,\ 8 \mid 0,\ 5,\ 9 \mid 3,\ 5,\ 5\,)$$
$$\mathrm{F2} = (\,1,\ 4,\ 8 \mid 0,\ 3,\ 3 \mid 3,\ 4,\ 9 \mid 2,\ 4,\ 6 \mid 0,\ 1\,)$$
$$M = 6:$$
$$\mathrm{F3} = (\,1,\ 2,\ 4,\ 4,\ 5,\ 8 \mid 2,\ 3,\ 4,\ 6,\ 8,\ 9 \mid 0,\ 1,\ 3,\ 5,\ 5\,)$$
$$\mathrm{F4} = (\,0,\ 2,\ 3,\ 3,\ 7,\ 9 \mid 0,\ 2,\ 4,\ 5,\ 6,\ 9\,)$$
$$M = 12:$$
$$\mathrm{F1} = (\,0,\ 1,\ 2,\ 2,\ 3,\ 3,\ 4,\ 4,\ 5,\ 7,\ 8,\ 9 \mid 0,\ 1,\ 3,\ 5,\ 5\,)$$
$$\mathrm{F2} = (\,0,\ 2,\ 2,\ 3,\ 4,\ 4,\ 5,\ 6,\ 6,\ 8,\ 9,\ 9\,)$$

Įvertinsime, kokį kiekį duomenų perrašome iš rinkmenos į sparčiąją operatyviąją kompiuterio atmintį.

Įvertinsime, kokį kiekį duomenų perrašome iš rinkmenos į sparčiąją operatyviąją kompiuterio atmintį.

Priminsime, kad kaip tik šios operacijos vykdymas ir sudaro išorinio rūšiavimo algoritmo didžiąją sąnaudų dalį.

Įvertinsime, kokį kiekį duomenų perrašome iš rinkmenos į sparčiąją operatyviąją kompiuterio atmintį.

Priminsime, kad kaip tik šios operacijos vykdymas ir sudaro išorinio rūšiavimo algoritmo didžiąją sąnaudų dalį.

Tegul $N = 2^k M$. Kiekvienu etapu nuskaitomi ir įrašomi visi N elementai, jie persiunčiami porcijomis ir jų skaičius N/M.

Suliejimo etapų skaičius yra $(k+1)$, todėl bendrieji duomenų perrašymo kaštai

$$\frac{N}{M} \log\left(\frac{N}{M}\right).$$