Greitieji rūšiavimo algoritmai

Raimondas Čiegis

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

Lapkričio 10 d., 2022

Jau nagrinėjome paprastus rūšiavimo algoritmus ir parodėme, kad atliekamų operacijų skaičius yra proporcingas N^2.

Jie nėra efektyvūs, kai turime daug duomenų, nes žinome, kad rūšiavimo algoritmų apatinis operacijų įvertis yra gerokai mažesnis $N \log N$.

Jau nagrinėjome paprastus rūšiavimo algoritmus ir parodėme, kad atliekamų operacijų skaičius yra proporcingas N^2.

Jie nėra efektyvūs, kai turime daug duomenų, nes žinome, kad rūšiavimo algoritmų apatinis operacijų įvertis yra gerokai mažesnis $N \log N$.

Šioje paskaitoje susipažinsime su rūšiavimo algoritmais, kurių sudėtingumas yra artimas optimaliam.

Spartusis rūšiavimo algoritmas

Algoritmas labai plačiai naudojamas daugelyje taikomųjų programų.

Parodysime, kad jo vidutinis sudėtingumas yra $\mathcal{O}(N \log N)$, o realizacija paprasta ir nereikia papildomos atminties rūšiuojamiems elementams saugoti. Todėl jis ir vadinamas sparčiuoju algoritmu (angl. *quick sort*).

Spartusis rūšiavimo algoritmas

Algoritmas labai plačiai naudojamas daugelyje taikomųjų programų.

Parodysime, kad jo vidutinis sudėtingumas yra $\mathcal{O}(N \log N)$, o realizacija paprasta ir nereikia papildomos atminties rūšiuojamiems elementams saugoti. Todėl jis ir vadinamas sparčiuoju algoritmu (angl. *quick sort*).

Spartusis rūšiavimo algoritmas sukurtas remiantis skaldyk ir valdyk metodu. Paaiškinsime, kaip realizuojami trys pagrindiniai jo etapai.

Uždavinio skaidymas. Visus aibės A elementus dalijame į du poaibius. Tuo tikslu parenkame pagrindinį elementą a_j, tada pirmajam poaibiui priskiriame elementus, mažesnius už a_j, o antrajam poaibiui – lygius ir didesnius už a_j.

Uždavinio skaidymas. Visus aibės A elementus dalijame į du poaibius. Tuo tikslu parenkame pagrindinį elementą a_j, tada pirmajam poaibiui priskiriame elementus, mažesnius už a_j, o antrajam poaibiui – lygius ir didesnius už a_j.

Dalinio uždavinio sprendimas. Jei poaibį sudaro vienas elementas, tai jis jau surūšiuotas, priešingu atveju jį vėl rūšiuojame sparčiuoju algoritmu.

Uždavinio skaidymas. Visus aibės A elementus dalijame į du poaibius. Tuo tikslu parenkame pagrindinį elementą a_j, tada pirmajam poaibiui priskiriame elementus, mažesnius už a_j, o antrajam poaibiui – lygius ir didesnius už a_j.

Dalinio uždavinio sprendimas. Jei poaibį sudaro vienas elementas, tai jis jau surūšiuotas, priešingu atveju jį vėl rūšiuojame sparčiuoju algoritmu.

Dažnai rekursiją baigiame anksčiau, pasirenkame nedidelį skaičių M, jei poaibio elementų skaičius yra ne didesnis už M, tai poaibį rūšiuojame kuriuo nors paprastu algoritmu.

Viso uždavinio sprendinio radimas. Kadangi visi pirmojo poaibio elementai yra mažesni už antrojo poaibio elementus, tai, išsprendę dalinius uždavinius, surūšiuojame visą aibę.

Šiame etape nereikia atlikti jokių papildomų veiksmų

Spartusis rūšiavimo algoritmas

```
QuickSort (l, r)
begin
  (1)  if  ( l < (r - M) )  then
  (2)       Partition ( l, r, m );
  (3)       QuickSort ( l, m-1 );
  (4)       QuickSort ( m+1, r );
       else
  (5)       if ( l < r ) SelectionSort (l, r);
       end if
end QuickSort
```

Partition (l, r, m)

begin

(1) v $= a_l$;

(2) i = l; j = r;

(3) **while** (i $<$ j) **do**

(4) **while** (($a_j \geqslant$ v) && (i $<$ j)) j = j $-$ 1;

(5) **if** (i $\neq$ j) **then**

(6) $a_i = a_j$; i++;

end if

(7) **while** (($a_i \leqslant$ v) && (i $<$ j)) i = i $+$ 1;

(8) **if** (i $\neq$ j) **then**

(9) $a_j = a_i$; j−−;

end if

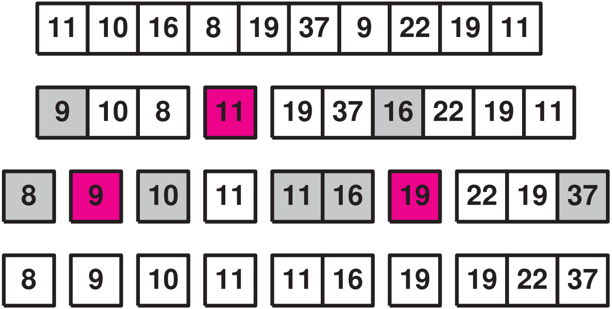
end do

(10) $a_i = v$; m = i;

end Partition

Sparčiuoju algoritmu surūšiuokime skaičių masyvą

$$E = (11,\ 10,\ 16,\ 8,\ 19,\ 37,\ 9,\ 22,\ 19,\ 11).$$



Pagrindiniu elementu visada imtas pirmasis poaibio elementas, pilka spalva pavaizduoti elementai, kurie buvo sukeisti vietomis aibės dalijimo etapu, raudona spalva – pagrindiniai elementai

Algoritmo sudėtingumo įvertinimas

Nagrinėsime aibės A elementų lyginimų skaičių L_N.

Algoritmo sudėtingumo įvertinimas

Nagrinėsime aibės A elementų lyginimų skaičių L_N.

Skaidydami uždavinį, visus elementus lyginame su pagrindiniu elementu. Todėl bendras lyginimų skaičius priklauso tik nuo dalinių uždavinių apimties.

Algoritmo sudėtingumo įvertinimas

Nagrinėsime aibės A elementų lyginimų skaičių L_N.

Skaidydami uždavinį, visus elementus lyginame su pagrindiniu elementu. Todėl bendras lyginimų skaičius priklauso tik nuo dalinių uždavinių apimties.

Analizę pradėkime nuo blogiausio atvejo, kada pagrindiniu elementu parenkame mažiausią aibės elementą. Tada gauname tokį sąryšį

$$L_B(N) = L_B(N-1) + N - 1\,.$$

Kai turime tik vieną elementą, aibė jau surūšiuota:

$$L(1) = 0\,.$$

Pritaikę rekurenčiąją lygybę $(N-1)$ kartą, apskaičiuojame elementų lyginimų skaičių

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

Pritaikę rekurenčiąją lygybę $(N-1)$ kartą, apskaičiuojame elementų lyginimų skaičių

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

Taigi blogiausiu atveju spartusis rūšiavimo algoritmas toks pat lėtas, kaip ir mūsų anksčiau išnagrinėti metodai.

Pritaikę rekurenčiąją lygybę $(N-1)$ kartą, apskaičiuojame elementų lyginimų skaičių

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

Taigi blogiausiu atveju spartusis rūšiavimo algoritmas toks pat lėtas, kaip ir mūsų anksčiau išnagrinėti metodai.

Ypač netikėta, kad tokį blogą rezultatą gauname ir tada, kai rūšiuojame jau sutvarkytą aibę, o pagrindiniu elementu renkamės pirmąjį aibės elementą.

Nagrinėkime geriausią atvejį, kai kiekvienu žingsniu pavyksta parinkti tokį pagrindinį elementą, kuris padalija visą aibę į dvi lygias dalis.

Nagrinėkime geriausią atvejį, kai kiekvienu žingsniu pavyksta parinkti tokį pagrindinį elementą, kuris padalija visą aibę į dvi lygias dalis.

Imkime $N = (2^m - 1)$. Tada gauname sąryšį, susiejantį elementų lyginimų skaičius:

$$L_G(2^m - 1) = \begin{cases} 2L_G(2^{m-1} - 1) + 2^m - 2, & \text{kai} \quad m > 1, \\ 0, & \text{kai} \quad m = 1\,. \end{cases}$$

Pritaikę šią lygybę $(m-2)$ kartus, apskaičiuojame elementų lyginimų skaičių

$$\begin{aligned} L_G(N) &= 2^m - 2 + 2 \cdot (2^{m-1} - 2) + 2^2 \cdot (2^{m-2} - 2) + \dots \\ &\quad + 2^{m-2} \cdot (2^2 - 2) \\ &= (m-1)2^m + 2^m - 2 \\ &= (N+1)\log(N+1) - 2. \end{aligned}$$

Pritaikę šią lygybę $(m-2)$ kartus, apskaičiuojame elementų lyginimų skaičių

$$
\begin{aligned}
L_G(N) &= 2^m - 2 + 2\cdot(2^{m-1}-2) + 2^2\cdot(2^{m-2}-2) + \dots \\
&\quad + 2^{m-2}\cdot(2^2-2) \\
&= (m-1)2^m + 2^m - 2 \\
&= (N+1)\log(N+1) - 2.
\end{aligned}
$$

Visgi tai nėra stebuklingas rezultatas. Priminsime, kad įterpimo rūšiavimo algoritmo geriausiojo atvejo sudėtingumas yra dar geresnis, užtenka atlikti tik N lyginimų.

Spartusis rūšiavimo algoritmas tapo tokiu populiariu todėl, kad ir vidutiniu atveju skaičiavimų apimtis nedaug skiriasi nuo geriausio atvejo

$$L_V(N) = 1,386 N \log N + \mathcal{O}(N)\,.$$

Spartusis rūšiavimo algoritmas tapo tokiu populiariu todėl, kad ir vidutiniu atveju skaičiavimų apimtis nedaug skiriasi nuo geriausio atvejo

$$L_V(N) = 1,386 N \log N + \mathcal{O}(N)\,.$$

Vidutiniškai atliekame tik 40 procentų daugiau lyginimų nei geriausiu atveju, kai aibę visada dalijame pusiau.

Medianos radimas sparčiuoju algoritmu

Kasdieninėje statistinėje analizėje dažnai tenka spręsti tokius uždavinius:

- iš duotosios skaičių sekos išrinkti medianą;

Medianos radimas sparčiuoju algoritmu

Kasdieninėje statistinėje analizėje dažnai tenka spręsti tokius uždavinius:

- iš duotosios skaičių sekos išrinkti medianą;
- rasti k-ąjį mažiausią skaičių.

Medianos radimas sparčiuoju algoritmu

Kasdieninėje statistinėje analizėje dažnai tenka spręsti tokius uždavinius:

- iš duotosios skaičių sekos išrinkti medianą;
- rasti k-ąjį mažiausią skaičių.

Medianos radimas sparčiuoju algoritmu

Kasdieninėje statistinėje analizėje dažnai tenka spręsti tokius uždavinius:

- iš duotosios skaičių sekos išrinkti medianą;
- rasti k-ąjį mažiausią skaičių.

Beje, medianos radimas yra bendresnio antrojo uždavinio atskiras atvejis, kai

$$k = N/2.$$

Šiuos uždavinius lengvai išsprendžiame, kai jau turime surūšiuotą aibę.

Šiuos uždavinius lengvai išsprendžiame, kai jau turime surūšiuotą aibę.

Jau mokame greitai rūšiuoti sparčiuoju algoritmu. Taigi ir naują uždavinį išspręsime atlikę $N \log N$ veiksmų.

Šiuos uždavinius lengvai išsprendžiame, kai jau turime surūšiuotą aibę.

Jau mokame greitai rūšiuoti sparčiuoju algoritmu. Taigi ir naują uždavinį išspręsime atlikę $N \log N$ veiksmų.

Ar galima jį išspręsti greičiau?

Šiuos uždavinius lengvai išsprendžiame, kai jau turime surūšiuotą aibę.

Jau mokame greitai rūšiuoti sparčiuoju algoritmu. Taigi ir naują uždavinį išspręsime atlikę $N \log N$ veiksmų.

Ar galima jį išspręsti greičiau?

Sukonstruosime kitą algoritmą, kurio skaičiavimų apimtis daug mažesnė už rūšiavimo algoritmų sąnaudas.

Šiuos uždavinius lengvai išsprendžiame, kai jau turime surūšiuotą aibę.

Jau mokame greitai rūšiuoti sparčiuoju algoritmu. Taigi ir naują uždavinį išspręsime atlikę $N \log N$ veiksmų.

Ar galima jį išspręsti greičiau?

Sukonstruosime kitą algoritmą, kurio skaičiavimų apimtis daug mažesnė už rūšiavimo algoritmų sąnaudas.

Vėl remsimės skaldyk ir valdyk metodu.

Reikiamo elemento išrinkimo algoritmą sudarysime modifikuodami spartųjį rūšiavimo algoritmą.

Sparčiosios paieškos algoritmas

```
int QuickFind (l, r, k)    #  l ≤ k ≤ r
begin
 (1)  if  ( l == r )  then
 (2)      return (l);
      else
 (3)      Partition (l, r, m);
 (4)      if  ( m > k )  then
 (5)          QuickFind ( l, m-1, k );
          else
 (6)          if  ( m == k )  then
 (7)              return (m);
              else
 (8)              QuickFind ( m + 1, r, k );
              end if
          end if
      end if
end QuickFind
```

Apsiribosime tik geriausio atvejo sudėtingumo analize.

Tada po kiekvieno dalijimo žingsnio dalinio uždavinio elementų skaičius sumažėja perpus, todėl gauname tokį elementų lyginimo skaičių:

$$L_G(N) = N + \frac{N}{2} + \frac{N}{4} + \ldots + 2 + 1 = 2N + \mathcal{O}(1)\,.$$

Apsiribosime tik geriausio atvejo sudėtingumo analize.

Tada po kiekvieno dalijimo žingsnio dalinio uždavinio elementų skaičius sumažėja perpus, todėl gauname tokį elementų lyginimo skaičių:

$$L_G(N) = N + \frac{N}{2} + \frac{N}{4} + \ldots + 2 + 1 = 2N + \mathcal{O}(1)\,.$$

Taigi naujasis medianos paieškos algoritmas $\frac{1}{2} \log N$ kartų spartesnis už rūšiavimo algoritmą.

Suliejimo rūšiavimo algoritmas

Susipažinsime su dar vienu sparčiuoju algoritmu, kurio skaičiavimo apimtis net ir blogiausiu atveju yra $\mathcal{O}(N \log N)$.

Suliejimo rūšiavimo algoritmas

Susipažinsime su dar vienu sparčiuoju algoritmu, kurio skaičiavimo apimtis net ir blogiausiu atveju yra $\mathcal{O}(N \log N)$.

Jis grindžiamas pastebėjimu, kad nesunku efektyviai sujungti du jau surūšiuotus poaibius į vieną surūšiuotą aibę.

Todėl metodas ir vadinamas suliejimo algoritmu (angl. *merge sort*).

Suliejimo rūšiavimo algoritmas

Susipažinsime su dar vienu sparčiuoju algoritmu, kurio skaičiavimo apimtis net ir blogiausiu atveju yra $\mathcal{O}(N \log N)$.

Jis grindžiamas pastebėjimu, kad nesunku efektyviai sujungti du jau surūšiuotus poaibius į vieną surūšiuotą aibę.

Todėl metodas ir vadinamas suliejimo algoritmu (angl. *merge sort*).

Įdomu tai, kad ir šį efektyvų rūšiavimo algoritmą sukonstruojame naudodami skaldyk ir valdyk metodą.

Paaiškinsime, kaip jame realizuojami trys pagrindiniai skaldyk ir valdyk metodo etapai.

Paaiškinsime, kaip jame realizuojami trys pagrindiniai skaldyk ir valdyk metodo etapai.

Uždavinio skaidymas. Visą rūšiuojamą aibę dalijame į du poaibius. Pirmajam poaibiui priskiriame elementus nuo pirmojo iki vidurinio, o antrajam poaibiui – visus likusius elementus.

Paaiškinsime, kaip jame realizuojami trys pagrindiniai skaldyk ir valdyk metodo etapai.

Uždavinio skaidymas. Visą rūšiuojamą aibę dalijame į du poaibius. Pirmajam poaibiui priskiriame elementus nuo pirmojo iki vidurinio, o antrajam poaibiui – visus likusius elementus.

Nereikia atlikti jokių skaičiavimo veiksmų!

Paaiškinsime, kaip jame realizuojami trys pagrindiniai skaldyk ir valdyk metodo etapai.

Uždavinio skaidymas. Visą rūšiuojamą aibę dalijame į du poaibius. Pirmajam poaibiui priskiriame elementus nuo pirmojo iki vidurinio, o antrajam poaibiui – visus likusius elementus.

Nereikia atlikti jokių skaičiavimo veiksmų!

Dalinio uždavinio sprendimas. Jei poaibį sudaro vienas elementas, tai jis jau surūšiuotas, priešingu atveju ir poaibį rūšiuojame suliejimo algoritmu.

Paaiškinsime, kaip jame realizuojami trys pagrindiniai skaldyk ir valdyk metodo etapai.

Uždavinio skaidymas. Visą rūšiuojamą aibę dalijame į du poaibius. Pirmajam poaibiui priskiriame elementus nuo pirmojo iki vidurinio, o antrajam poaibiui – visus likusius elementus.

Nereikia atlikti jokių skaičiavimo veiksmų!

Dalinio uždavinio sprendimas. Jei poaibį sudaro vienas elementas, tai jis jau surūšiuotas, priešingu atveju ir poaibį rūšiuojame suliejimo algoritmu.

Taigi panaudojame rekursijos metodą.

Dviejų poaibių sujungimo algoritmas.

Dviejų poaibių sujungimo algoritmas.

Naudojame pagalbinį masyvą, kurio ilgis lygus bendram abiejų poaibių elementų skaičiui.

Dviejų poaibių sujungimo algoritmas.

Naudojame pagalbinį masyvą, kurio ilgis lygus bendram abiejų poaibių elementų skaičiui.

Priminsime, kad abiejų poaibių elementai jau surūšiuoti.

Dviejų poaibių sujungimo algoritmas.

Naudojame pagalbinį masyvą, kurio ilgis lygus bendram abiejų poaibių elementų skaičiui.

Priminsime, kad abiejų poaibių elementai jau surūšiuoti.

Imame pirmuosius poaibių elementus ir juos palyginame, mažesnį talpiname į surūšiuotą aibę. Tarkime, šis elementas priklausė pirmajam poaibiui.

Dviejų poaibių sujungimo algoritmas.

Naudojame pagalbinį masyvą, kurio ilgis lygus bendram abiejų poaibių elementų skaičiui.

Priminsime, kad abiejų poaibių elementai jau surūšiuoti.

Imame pirmuosius poaibių elementus ir juos palyginame, mažesnį talpiname į surūšiuotą aibę. Tarkime, šis elementas priklausė pirmajam poaibiui.

Toliau procesą kartojame, lygindami mažiausią antrojo poaibio elementą su nauju mažiausiu pirmojo poaibio elementu.

Dviejų poaibių sujungimo algoritmas.

Naudojame pagalbinį masyvą, kurio ilgis lygus bendram abiejų poaibių elementų skaičiui.

Priminsime, kad abiejų poaibių elementai jau surūšiuoti.

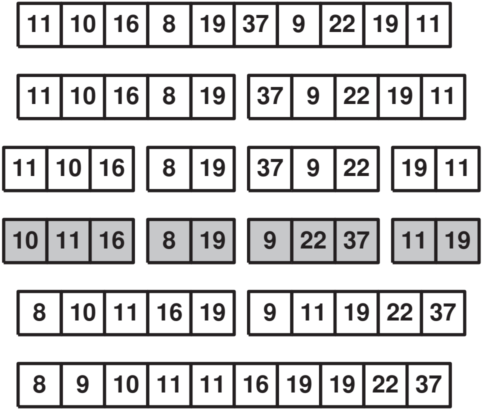
Įmame pirmuosius poaibių elementus ir juos palyginame, mažesnį talpiname į surūšiuotą aibę. Tarkime, šis elementas priklausė pirmajam poaibiui.

Toliau procesą kartojame, lygindami mažiausią antrojo poaibio elementą su nauju mažiausiu pirmojo poaibio elementu.

Lyginimo procesą tęsiame tol, kol ir viename, ir kitame poaibyje yra elementų.

Suliejimo algoritmu surūšiuokime skaičių masyvą

$$E = (11, 10, 16, 8, 19, 37, 9, 22, 19, 11).$$



Pilka spalva pavaizduotas elementų rūšiavimas išrinkimo algoritmu

Suliejimo algoritmo sudėtingumo analizė

Kadangi visi veiksmai atliekami suliejimo žingsnyje, tai užtenka nagrinėti tik šią algoritmo dalį.

Suliejimo algoritmo sudėtingumo analizė

Kadangi visi veiksmai atliekami suliejimo žingsnyje, tai užtenka nagrinėti tik šią algoritmo dalį.

Tarsime, kad dviejų surūšiuotų N_1 ir N_2 ilgio poaibių sujungimo metu lyginame $c(N_1 + N_2)$ elementų. Taigi, laikome, kad lyginame tam tikrą fiksuotą dalį visų abiejų poaibių elementų.

Suliejimo algoritmo sudėtingumo analizė

Kadangi visi veiksmai atliekami suliejimo žingsnyje, tai užtenka nagrinėti tik šią algoritmo dalį.

Tarsime, kad dviejų surūšiuotų N_1 ir N_2 ilgio poaibių sujungimo metu lyginame $c(N_1 + N_2)$ elementų. Taigi, laikome, kad lyginame tam tikrą fiksuotą dalį visų abiejų poaibių elementų.

Imkime $N = 2^m$. Tada gauname tokį sąryšį, susiejantį elementų lyginimų skaičius:

$$L(N) = \begin{cases} 2\,L\left(\frac{1}{2}N\right) + cN, & \text{kai} \quad N > 2, \\ 1, & \text{kai} \quad N = 2\,. \end{cases}$$

Pritaikę šią lygybę $(m-1)$ kartą, apskaičiuojame elementų lyginimų skaičių:

$$\begin{aligned} L(N) &= 2L\left(\frac{1}{2}N\right) + cN \\ &= 4L\left(\frac{1}{4}N\right) + 2cN \\ &= \cdots = \frac{N}{2}L(2) + (m-1)cN \\ &= cN\log N + (0.5-c)N\,. \end{aligned}$$

Pritaikę šią lygybę $(m-1)$ kartą, apskaičiuojame elementų lyginimų skaičių:

$$\begin{aligned} L(N) &= 2L\Big(\frac{1}{2}N\Big) + cN \\ &= 4L\Big(\frac{1}{4}N\Big) + 2cN \\ &= \dots = \frac{N}{2}L(2) + (m-1)cN \\ &= cN\log N + (0.5-c)N \,. \end{aligned}$$

Taigi suliejimo rūšiavimo algoritmo skaičiavimų apimtis yra asimptotiškai optimali net ir blogiausiuoju duomenų pasiskirstymo atveju.