RADIX SORT
Oblikovanje i analiza algoritama

Martin Vlahović

Matematički odsjek
Prirodoslovno-matematički fakultet
Sveučilište u Zagrebu

10. siječnja 2021.

Definicija (Stabilno sortiranje)

Za algoritam sortiranja početnog polja $A[1,2,\ldots,n]$ u polje $A[p(1),p(2),\ldots p(n)]$ po ključu key, gdje je $p:\{1,2,\ldots,n\}\to\{1,2,\ldots,n\}$ permutacija, kažemo da je ***stabilan*** *ako za svaka dva $i<j=1,\ldots,n$ takva da je $A[i].key=A[j].key$, vrijedi $A[p(i)].key<A[p(j)].key$. Drugim riječima, dva elementa početnog polja koja imaju jednak ključ očuvaju svoj prvobitni poredak u sortiranom polju.*

- **Primjer:** Pretpostavimo da sortiramo polje $A[1,2,3,4]$ riječi:

$$A[1,2,3,4]=\{"oblikovanje","i","analiza","algoritama"\},$$

po ključu:

$$key=\text{prvo slovo riječi.}$$

- Tada je sortirano polje:

$$A[p(1),p(2),p(3),p(4)]=\{"analiza","algoritama","i","oblikovanje"\}.$$

- Osnovni sastojak potreban za implementaciju **Radix sorta** je **neki stabilni sort** na **potpuno uređenom** skupu ključeva $Keys$, koji je predefiniran i konačan s $\mathrm{card}(Keys) = k$ elemenata. Radi jednostavnosti stavljamo $Keys = \{1, 2, \ldots, k\}$.

- Kao primjer takvog algoritma kojeg ćemo koristit navodimo tzv. **Counting sort**.

- Pretpostavimo da imamo polje $A[1, 2, \ldots, n] \subseteq Keys$. Definiramo pomoćno polje $C[1, 2, \ldots, k] \subseteq \mathbb{N}_0$ počeno inicijalizirano na nulu, te polje $B[1, 2, \ldots, n] \subseteq Keys$ u koje ćemo spremiti sotrirano polje.

- Ideja je "prošetati" se koroz polje A, te u polje C unijeti podatke tako da vrijedi: $C[i] \in \mathbb{N}_0$ sadrži broj elemenata polja A koji su manji ili jednaki od $i \in Keys$. Koristeći tu informaciju, u polje B ćemo konstruirati sortirano polje A.

Counting sort - opis algoritma

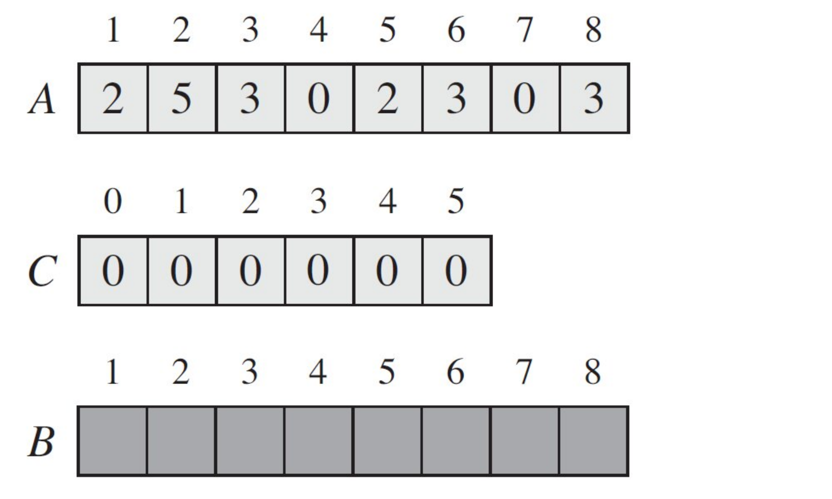
Algoritam radi u tri koraka:

1. Prolaskom $j = 1, \ldots, n$, kroz polje A popuniti polje C tako da za svaku incidenciju ključa $A[j] \in Keys$, inkrementiramo $C[A[j]] = C[A[j]] + 1$, koji je u početku nula.
2. Prolaskom $j = \mathbf{2}, \ldots, k$ kroz polje C izvršavajući: $C[i] = C[i] + C[i-1]$, očito će biti ispunjeno: $C[i] \in \mathbb{N}_0$ sadrži broj elemenata polja A koji su manji ili jednaki od $i \in Keys$.
3. Prolaskom **unazad** $j = n, \ldots, 1$, kroz polje A, koristeći polje C popuniti polje B. Za svaki ključ $A[j]$ mi znamo da se u sortiranom polju B mora nalaziti na mjestu $C[A[j]]$ kada ne bismo u polju A imali duplikate ključeva. Tada stavljamo $B[C[A[j]]] = A[j]$. Ako ipak imamo duplikata u originalnom polju A, tada dekrementiranjem: $C[A[j]] = C[A[j]] - 1$, osiguravamo novo mjesto u polju B gdje će biti spremljen ponovljeni ključ.

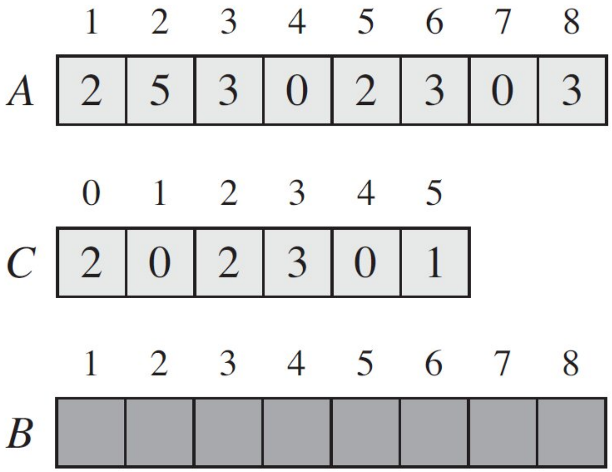
Algoritam će na kraju u polje B spremiti sortirane ključeve iz polja A. Upravo zbog prolaska **unazad** u trećem koraku, te zbog **dekrementiranja** $C[A[j]]$ će dva **jednaka ključa** iz polja A biti spremljena **u originalnom poretku** u polje B. Dakle, Counting sort je stabilan!
Pitanje: Što ako bismo išli **unaprijed**?

Lako vidimo da su vremenska i memorijska složenost algoritma: $\Theta(n+k)$.

Counting sort - ilustracija algoritma 1/11



Counting sort - ilustracija algoritma 2/11

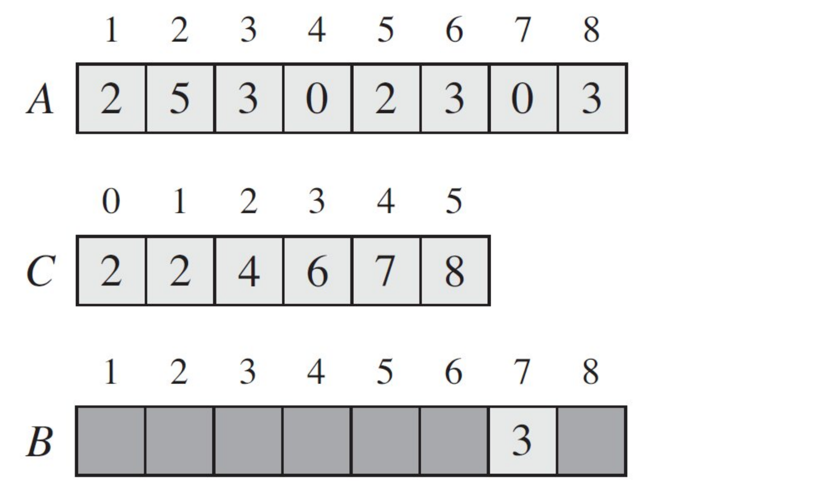


Counting sort - ilustracija algoritma 3/11

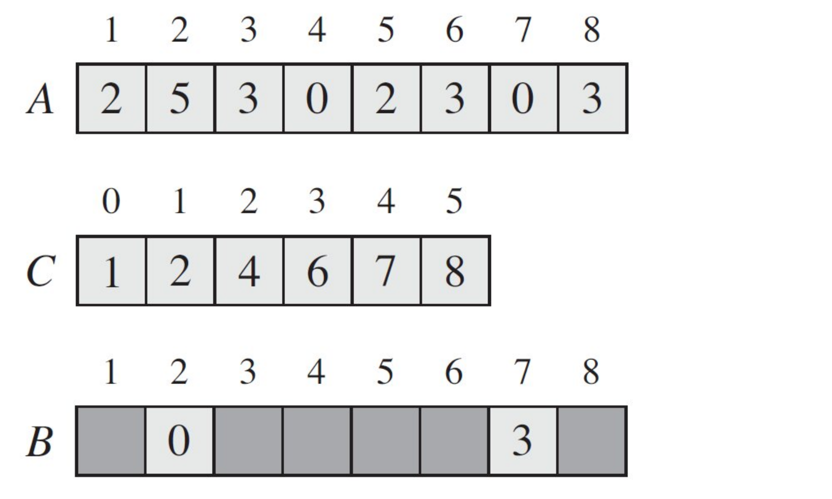
	1	2	3	4	5	6	7	8
A	2	5	3	0	2	3	0	3

	0	1	2	3	4	5
C	2	2	4	7	7	8

	1	2	3	4	5	6	7	8
B								


1 2 3 4 5 6 7 8
A 2 5 3 0 2 3 0 3
0 1 2 3 4 5
C 2 2 4 6 7 8
1 2 3 4 5 6 7 8
B 3

Counting sort - ilustracija algoritma 5/11



Counting sort - ilustracija algoritma 6/11

	1	2	3	4	5	6	7	8
A	2	5	3	0	2	3	0	3

	0	1	2	3	4	5
C	1	2	4	5	7	8

	1	2	3	4	5	6	7	8
B		0				3	3	

Counting sort - ilustracija algoritma 7/11

	1	2	3	4	5	6	7	8
A	2	5	3	0	2	3	0	3

	0	1	2	3	4	5
C	1	2	3	5	7	8

	1	2	3	4	5	6	7	8
B		0		2		3	3	

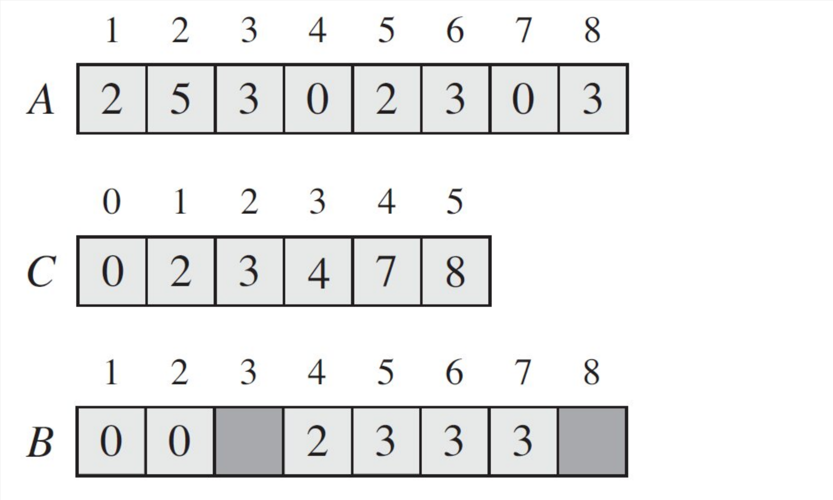
Counting sort - ilustracija algoritma 8/11

	1	2	3	4	5	6	7	8
A	2	5	3	0	2	3	0	3

	0	1	2	3	4	5
C	0	2	3	5	7	8

	1	2	3	4	5	6	7	8
B	0	0		2		3	3	

Counting sort - ilustracija algoritma 9/11



Counting sort - ilustracija algoritma 10/11

	1	2	3	4	5	6	7	8
A	2	5	3	0	2	3	0	3

	0	1	2	3	4	5
C	0	2	3	4	7	7

	1	2	3	4	5	6	7	8
B	0	0		2	3	3	3	5

Counting sort - ilustracija algoritma 11/11

	1	2	3	4	5	6	7	8
A	2	5	3	0	2	3	0	3

	0	1	2	3	4	5
C	0	2	2	4	7	7

	1	2	3	4	5	6	7	8
B	0	0	2	2	3	3	3	5

- Pretostavimo da imamo polje $A[1, \ldots, n]$ i želimo ih sortirati po ključu $key \in K$.
- Pretpostavimo da skup ključeva K možemo zapisati kao:

$$K = \underbrace{Keys \times Keys \times \cdots \times Keys}_{d},$$

  dakle svaki ključ možemo svhatiti kao $d-$znamenkasti broj, gdje je svaka znamenka iz skupa $Keys = \{1, 2, \ldots, k\}$.
- Naivna ideja bi bila rekurzivno sortirati polje A počevši od **prve znamenke**. Nažalost taj pristup stvori velik broj privremenih polja.
- Bolji pristup je kontraintuitivno sortirati stabilnim sortom, počevši od **zadnje znamenke**.
- Obično se koristi kada polje A nosi tzv. **satelitske podatke** koje želimo stabilno sortirati po ključu (npr. trojka (godina, mjesec, dan)).

Radix sort - ilustracija algoritma

- Primjer:

329	720	720	329
457	355	329	355
657	436	436	436
839	457	839	457
436	657	355	657
720	329	457	720
355	839	657	839

- Koristeći Counting sort koji je stabilan, očito je vremenska složenosti: $\Theta(d(n+k))$.
- Treba pokazati korektnost algoritma koristeći činjenicu da je Counting sort stabilan.
- Dokaz provodimo **indukcijom** po broju znamenki $d \in \mathbb{N}$ t.d. $K = (Keys)^d$.

Dokaz:

- **Baza indukcije**: ako je $d = 1$, tada očito Radix sort radi korektno i stabilan je zbog toga što je Counting sort korektan i stabilan.
- **Pretpostavka indukcije**: Pretpostavimo da je Radix sort korektan za $K = (Keys)^{d-1}$.
- **Korak indukcije:** Radix sort d znamenki ekvivalentan je Radix sortiranju najnižih $d-1$ znamenki, nakon čeka sotrira po prvoj znamenci koristeći zadan stabilan sort. Po pretpostavci, nakon Radix sortiranja po najnižih $d-1$ znamenki, elementi polja $A[1,\ldots,n]$ su korektno i stabilno sortirani po zadnjih $d-1$ znamenki. Nakon toga će stabilan sort korektno i stabilno sortirati polje $A[1,\ldots,n]$ po prvoj znamenci. Uzmimo tada dva proizvoljna elementa $a, b \in A[1,\ldots,n] \subseteq (Keys)^d$ i označimo s $a_1, b_1 \in Keys$, njihove prve namenke. Tada ako je:
  - $a_1 \neq b_1$, na kraju Radix sorta će se a i b nalaziti na korektnoj relativnoj poziciji neovisno o poretku nižih znamenki jer će stabilan sort u zanjem koraku obaviti korektno svoj dio posla.
  - $a_1 = b_1$, tada će stabilan sort u zadnjem koraku ostaviti a i b u jednakoj relativnoj poziciji. Ali obzirom da je taj relativan položaj već korektan (jer je korektan relativan položaj u sortiranom polju elemenata a i b zadan zadnjim $d-1$ znamenakama), po pretpostavci indukcije slijedi korektnost i stabilnost Radix sorta na d znamenkastim elementima.

CO**W**	S**E**A	**T**AB	BAR
DO**G**	T**E**A	**B**AR	BIG
SE**A**	M**O**B	**E**AR	BOX
RU**G**	T**A**B	**T**AR	COW
RO**W**	D**O**G	**S**EA	DIG
MO**B**	R**U**G	**T**EA	DOG
BO**X**	D**I**G	**D**IG	EAR
TA**B**	B**I**G	**B**IG	FOX
BA**R**	B**A**R	**M**OB	MOB
EA**R**	E**A**R	**D**OG	NOW
TA**R**	T**A**R	**C**OW	ROW
DI**G**	C**O**W	**R**OW	RUG
BI**G**	R**O**W	**N**OW	SEA
TE**A**	N**O**W	**B**OX	TAB
NO**W**	B**O**X	**F**OX	TAR
FO**X**	F**O**X	**R**UG	TEA

Radix sort - realan primjer 1/3

- Pretpostavimo da imamo polje $A[1, \ldots, n]$ od n elemenata tako da je $A[i] \in \{0, 1, \ldots, n^3 - 1\}$. Želimo pronaći algoritam vremenske složenosti $O(n)$.

- Primjerice, tip: `unsigned long long int`, na uobičajenim modernim računalima zauzima 8 byte-a, što odgovara rasponu brojeva:

$$0, 1, 2, \ldots, 2^{8 \times 8} - 1 = 18446744073709551615 \approx 1.84 \times 10^{19}.$$

- U stvarnim primjenama nerijetko imamo posla s poljima primjerice veličine $n = 2 \times 10^6$.

- Ako su elementi polja $A[1, \ldots, n]$ tipa `unsigned long long int`, tada doista vrijedi da je: $A[i] \in \{0, 1, 2, \ldots, n^3 - 1\}$, jer je:

$$n^3 - 1 = \left(2 \times 10^6\right)^3 - 1 \approx 8 \times 10^{18} < 1.84 \times 10^{19}.$$

- Iskoristimo ideju Radix sorta. Zapišimo svaki $a \in A[1, \dots, n] \subseteq \{0, 1, \dots, n^3 - 1\}$ element polja u na jedinstven način bazi n:

$$a = a_0 n^0 + a_1 n^1 + a_2 n^2,$$

  gdje su $a_0, a_1, a_2 \in \{0, 1, \dots, n - 1\}$. Za to nam je potrebno očito $\Theta(n)$ vremena. Nakon toga, uzastopnim Counting sortiranjem prvo po znamenci a_0, zatim a_1 te onda a_2, ćemo Radix sortirati cijelo polje $A[1, \dots, n]$.
- Za Counting sortiranje svake od tri znamenke potrebno je $\Theta(n)$ vremena i $\Theta(n)$ dodatne memorije.
- To je u praksi prihvatljivo jer je, kao što smo rekli, pretpostavljamo: $n \approx 2 \times 10^6$.
- *Zaključak*: Ako imamo polje $A[1, 2, \dots, n]$ nenegativnih cijelih brojeva 8 byte-nog tipa:

```
unsigned long long int,
```

  t.d. je $n \leq 2 \times 10^6$, moguće ga je sortirati u vremenskoj i memorijskoj složenosti $\Theta(n)$.

- Implementacija opisanog algoritma se nalazi na *GitHub* repozitoriju [2].

- Rezultati usporedbe vremena izvršavanja opisanog algoritma i STL `std::sort` funkcije:

n	time(`std::sort`) / time(Radix sort)
100	1.154
1000	1.692
10000	1.173
100000	1.133
100000	0.323

- Dakle, vidimo da je Radix sort na realnim primjerima može biti brži od izuzetno optimizirane `std::sort` funkcije, iako za dovoljno velik n, Radix sort algoritam postane *cache-unfriendly*.

- Znamo da sortiranje uspoređivanjem **nije moguće** napraviti bolje od $\Omega(n \lg n)$.
- Ipak, Counting i Radix sort su **asimptotski brži** jer oni ne koriste uspoređivanje nego koriste činjenicu o jako **pravilnoj strukturi** totalno uređenog skupa $Keys$ koji je izomorfan skupu $\{1, 2, \ldots, k\}$.
- U praktičnim implementacijama na tipičnim arhitekturama računala, Quicksort ponekd može biti ipak bolji iako mu je (prosječna) složenosti $O(n \lg n)$ jer efikasnije koristi *cache*-iranje od primjerice Radix sorta [1].
- Također, Counting i Radix sort traže **dodatnu memoriju** za privremena polja, dok je Quicksort *in-place* algoritam.

Reference

T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein,
Introduction to Algorithms (3rd ed.),
MIT Press and McGraw-Hill (2009)

Github repozitorij s implementacijom i testovima Radix sort algoritma,
https://github.com/mavlaho/OAA