Matematički fakultet

Projekat iz predmeta Konstrukcija i analiza algoritama 2

Školska 2018/2019

Radix sort

Student:
Jelena Ćosić 1081/2018

Mentori:
dr Vesna Marinković
Mirko Spasić

February 1, 2019

Sadržaj

1 Algoritam

Najjednostavniji postupak sortiranja niza prirodnih brojeva bi se sastojao u tome da se obezbedi dovoljan broj lokacija, i da se onda svaki element smesti na svoju lokaciju. Taj postupak zove se **sortiranje razvrstavanjem (eng. bucket sort)**. Ovaj algoritam sortiranja razvrstavanjem radi dobro ako su elementi iz malog, jednostavnog opsega, koji je unapred poznat.

Prirodno uopštenje ove ideje je **sortiranje višestrukim razvrstavanjem (engl. radix sort)**. Ovaj algoritam sortiranja koristi pozicionu reprezentaciju i odvojeno analizira cifre na različitim pozicijama. Ipak, ako su elementi stringovi koje treba sortirati leksikografski, možemo ih upoređivati znak po znak, pa se dobija leksikografsko sortiranje.

Postoji i druga varijanta iste ideje a to je verzija sortiranja višestrukim razvrstavanjem (cifre se prolaze sleva udesno) poznata je kao **sortiranje obratnim višestrukim razvrstavanjem**. Opseg se može podeliti na bilo koji odgovarajući način.

Način realizacije sortiranja višestrukim razvrstavanjem zasniva se na primeni indukcije obrnutim redosledom: sortiranje se radi zdesna ulevo, polazeći od najnižih cifara. Pretpostavljamo da su elementi veliki brojevi, predstavljeni sa k cifara u sistemu sa osnovom d (cifre su iz opsega od 0 do $d-1$).

Induktivna hipoteza: Umemo da sortiramo brojeve sa manje od k cifara.

Razlika između ovog metoda i sortiranja obratnim višestrukim razvrstavanjem je u načinu na koji se proširuje hipoteza (ideja primene induktivne hipoteze obrnutim redosledom slična je kao kod Hornerove šeme). Kod datih brojeva sa k cifara mi najpre ignorišemo najvišu (prvu) cifru i sortiramo brojeve prema ostatku cifara indukcijom. Tako dobijamo listu brojeva sortiranih prema najnižih $k-1$ cifara. Zatim prolazimo još jednom kroz sve elemente, i razvrstavamo ih prema najvišoj cifri u d pregrada. Konačno, objedinjujemo redom sadržaje pregrada. Ovaj algoritam zove se **sortiranje direktnim višestrukim razvrstavanjem**. Pokazaćemo da su elementi na kraju sortirani po svih k cifara.

Tvrdimo da su dva elementa svrstana u različite pregrade u ispravnom poretku. Za to nam nije potrebna induktivna hipoteza, jer je najviša cifra prvog od njih veća od najviše cifre drugog. S druge strane, ako dva elementa imaju iste najviše cifre, onda su oni prema induktivnoj hipotezi dovedeni u ispravan redosled pre poslednjeg koraka. Bitno je da elementi stavljeni u istu pregradu ostaju u istom redosledu kao i pre razvrstavanja. Ovo se može postići upotrebom liste za svaku pregradu i objedinjavanjem d lista na kraju svake etape u jednu globalnu listu od svih elemenata (sortiranih prema najnižih i cifara).

U primerima koje ćemo razmatrati podela će biti izvršena u skladu sa dekadnim prikazom brojeva. Naime, posmatrajući najmanje značajnu cifru u svakom broju, brojevi se razvrstavaju u različite skupove označene ciframa od 0 do $d-1$. Nakon toga, dolazi do ponovnog objedinjavanja nastalih skupova brojeva i algoritam se nastavlja posmatrajući narednu cifru po značajanosti. Postojaće onoliko različitih koraka koliko cifara ima najveći broj u nizu koji se sortira. Algoritam se završava kada su obrađene sve cifre u svim brojevima i tada se dobija sortiran niz brojeva.

Na narednoj slici (Slika 1) nalazi se pseudokod algoritma za sortiranje direktnim višestrukim razvrstavanjem.

Алгоритам DVR_sort(X, n, k);
Улаз: X (низ од n целих ненегативних бројева са по k цифара).
Излаз: X (сортирани низ).
begin
Претпостављамо да су на почетку сви елементи у глобалној листи GL
{GL се користи због једноставности; листа се може реализовати у X}
for $i := 0$ **to** $d - 1$ **do**
{d је број могућих цифара; $d = 10$ у декадном случају}
иницијализовати листу $Q[i]$ као празну листу;
for $i := k$ **downto** 1 **do**
while GL није празна **do** {разврставање по i-тој цифри}
скини x из GL; {x је први елемент са листе}
$c := i$–та цифра x; {гледано слева удесно}
убаци x у $Q[c]$;
for $t := 0$ **to** $d - 1$ **do** {обједињавање локалних листа}
укључи $Q[t]$ у GL; {додавање на крај листе}
for $i := 1$ **to** n **do** {преписивање елемената из GL у низ x}
скини $X[i]$ из GL
end

Slika 1: Sortiranje direktnim višestrukim razvrstavanjem

2 Složenost

Potrebno je n koraka za kopiranje elemenata u globalnu listu GL i d koraka za inicijalizaciju lista $Q[i]$. U glavnoj petlji algoritma, koja se izvršava k puta, svaki elemenat se vadi iz globalne i stavlja u neku od lista $Q[i]$. Na kraju se sve liste $Q[i]$ ponovo objedinjavaju u GL. Ukupna vremenska složenost algoritma je $O(k \cdot (n + d))$.

Slično, neka je d broj ponavljanja algoritma sortiranja razvrstavanjem (odnosno, d je broj cifara najvećeg elementa), k broj pregrada koje se koriste i n broj elemenata zadatog niza. Tada, ukupna vremenska složenost algoritma **radix sort** je $O(d \cdot O(n + k)) = O(d \cdot (n + k))$, u opštem slučaju. Ipak, broj cifara i opseg su često konstantni pa je složenost u prosečnom slučaju $O(n)$.

Sa naredne slike (Slika 2) može se zaključiti da **radix sort** je jedan od najefikasnijih algoritama sortiranja što se tiče vremenske složenosti. Ako posmatramo prostornu složenost, tu je situacija malo lošija jer je potrebno prostora za smeštanje $n + k$ elemenata neophodnih u toku sortiranja.

Array Sorting Algorithms

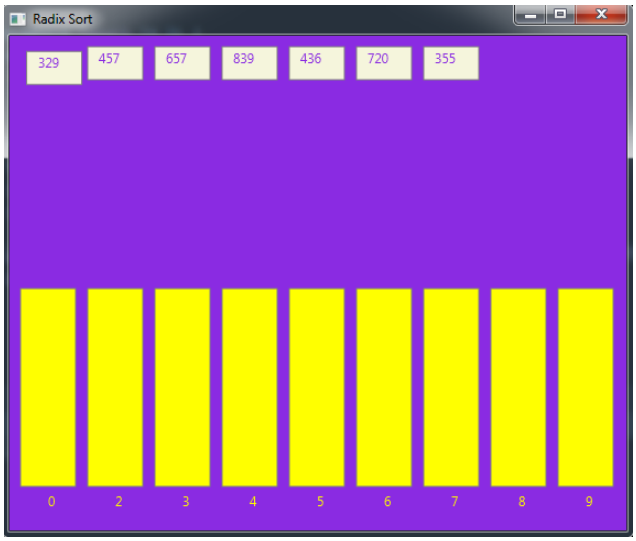
Algorithm	Time Complexity			Space Complexity
	Best	Average	Worst	Worst
Quicksort	Ω(n log(n))	Θ(n log(n))	O(n^2)	O(log(n))
Mergesort	Ω(n log(n))	Θ(n log(n))	O(n log(n))	O(n)
Timsort	Ω(n)	Θ(n log(n))	O(n log(n))	O(n)
Heapsort	Ω(n log(n))	Θ(n log(n))	O(n log(n))	O(1)
Bubble Sort	Ω(n)	Θ(n^2)	O(n^2)	O(1)
Insertion Sort	Ω(n)	Θ(n^2)	O(n^2)	O(1)
Selection Sort	Ω(n^2)	Θ(n^2)	O(n^2)	O(1)
Tree Sort	Ω(n log(n))	Θ(n log(n))	O(n^2)	O(n)
Shell Sort	Ω(n log(n))	Θ(n(log(n))^2)	O(n(log(n))^2)	O(1)
Bucket Sort	Ω(n+k)	Θ(n+k)	O(n^2)	O(n)
Radix Sort	Ω(nk)	Θ(nk)	O(nk)	O(n+k)
Counting Sort	Ω(n+k)	Θ(n+k)	O(n+k)	O(k)
Cubesort	Ω(n)	Θ(n log(n))	O(n log(n))	O(n)

Slika 2: Složenost različitih algoritama sortiranja

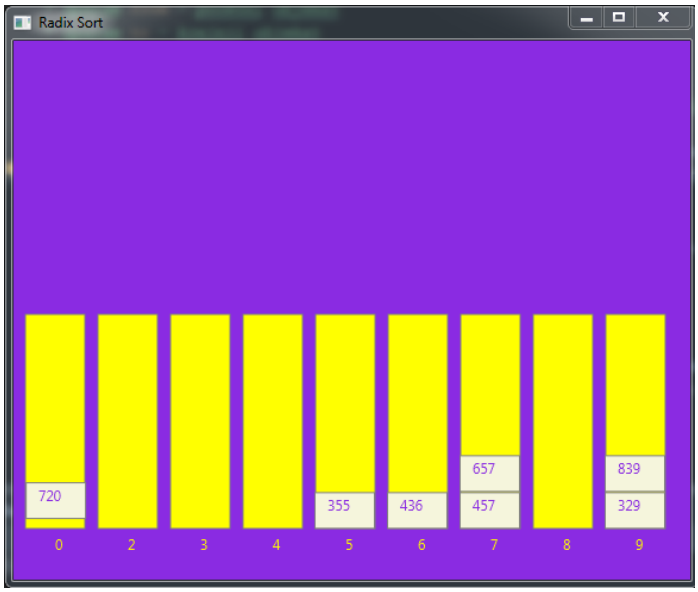
3 Primeri

Primer 3.1 *Postepeni prikaz sortiranja brojeva : 329, 457, 657, 839, 436, 720, 355 algoritmom radix sort.*

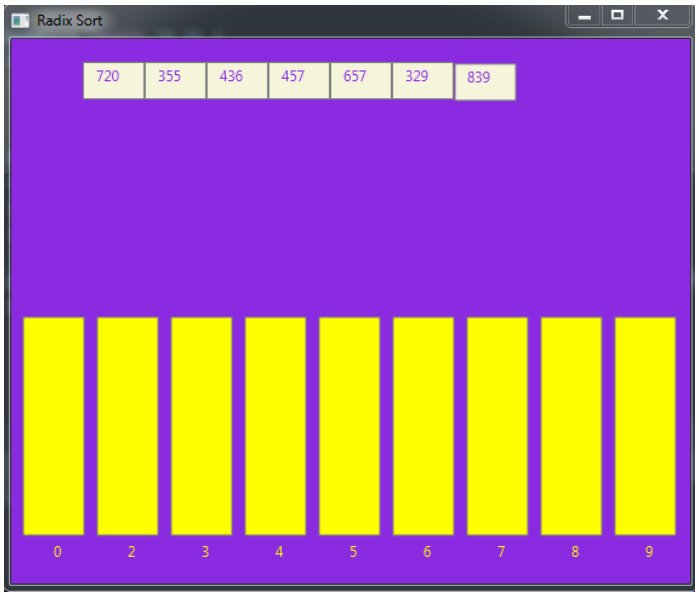
Rešenje: Na sledećim slikama su predstavljeni svi koraci prilikom sortiranja.



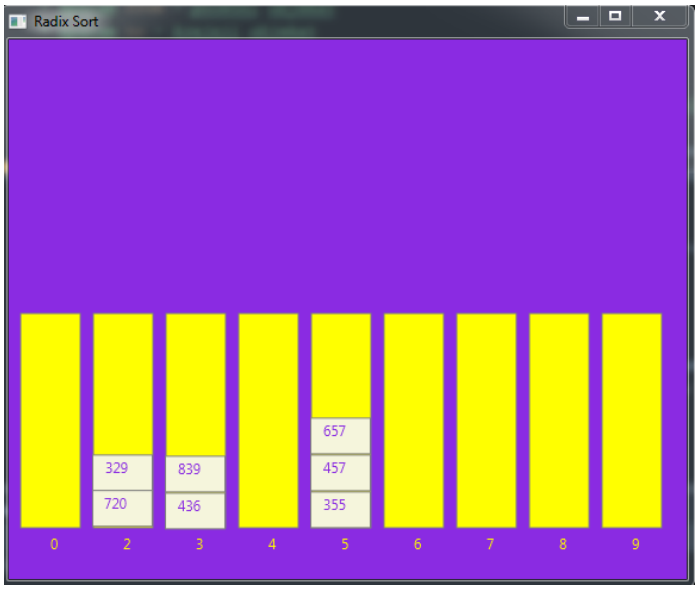
Slika 3: Početak



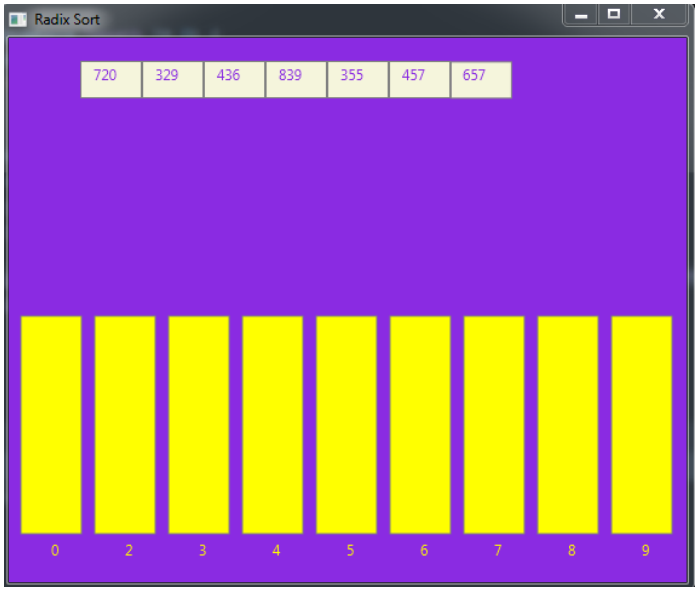
Slika 4: Korak 1



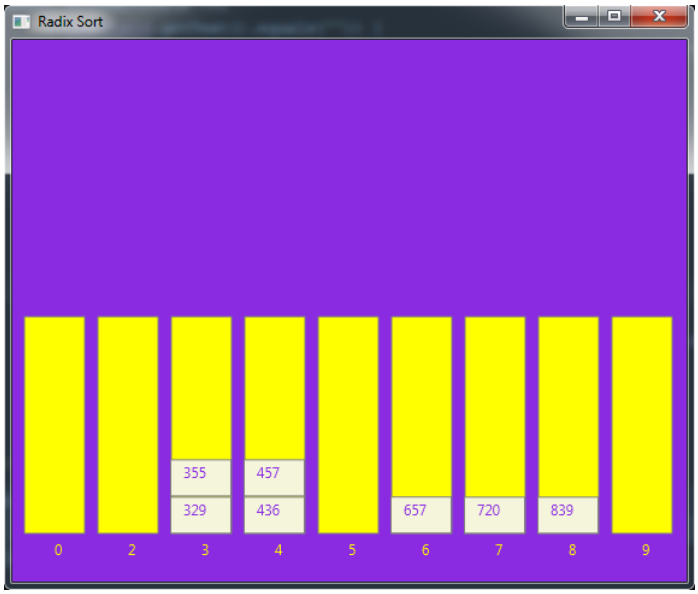
Slika 5: Korak 1



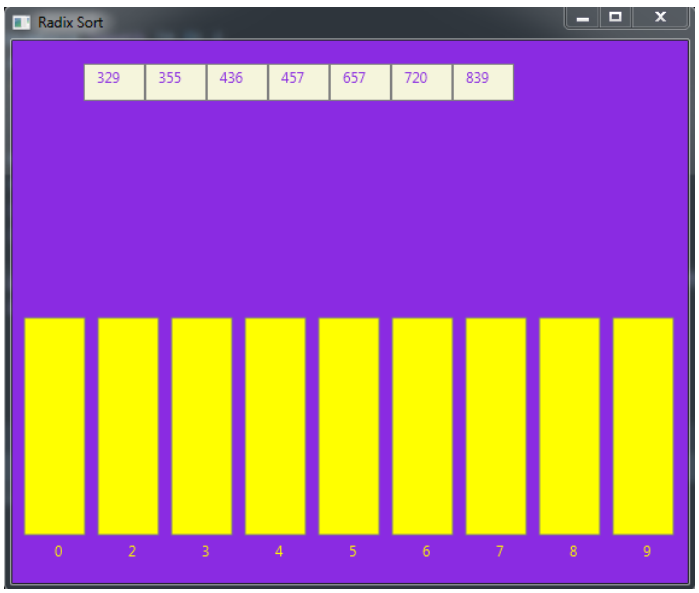
Slika 6: Korak 2



Slika 7: Korak 2



Slika 8: Korak 3



Slika 9: Korak 3

Primer 3.2 *Pokazati kako je moguće sortirati n celih brojeva iz opsega $[0, n^2 - 1]$ u vremenu $O(n)$.*

Rešenje 1: Ako bismo primenili algoritam radix sort, pri čemu su brojevi zapisani u osnovi n, onda je $d = \log_n n^2 = 2$ i $k = n$, pa je vremenska složenost $O(2 \cdot (n+n)) = O(n)$. U implementaciji ovog algoritma nije neophodno pretvarati brojeve iz dekadne osnove u osnovu broja n, već je moguće izdvojiti potrebne cifre korišćenjem operatora mod n i div n.

Rešenje 2: Svaka vrednost bi bila razmatrana kao dvocifreni broj, gde je svaka cifra iz opsega $[0, n-1]$. Ovaj algoritam bio bi složenosti $O(2(n+n)) = O(n)$.

Na narednoj slici (Slika 10) prikazan je pseudokod rešenja.

```
Algoritam: Sortiranje_u_linearnom_vremenu(niz)
Ulaz: niz (n celih brojeva iz opsega [0, n^2 − 1])
Izlaz: sortirani niz sort_niz
begin
    niz_1 := niz mod n; // operator mod se primenjuje na svaki element niza niz
    niz_2 := niz div n; // operator div se primenjuje na svaki element niza niz
    // drugi argument algoritma bucket_sort je broj pregrada
    sort_niz_1 := bucket_sort(niz_1, n);
    sort_niz_2 := bucket_sort(niz_2, n);
    objedini nizove sort_niz_1 i sort_niz_2 u niz sort_niz kao u radix_sort algoritmu;
    return sort_niz;
end
```

Slika 10: Sortiranje u linearnom vremenu

Primer 3.3 *Dato je k listi i svaka od njih sadrži n elemenata. Ključevi elemenata su celi brojevi iz opsega $[1, m]$. Pokazati kako se mogu sortirati sve liste tako da vremenska složenost u najgorem slučaju bude $O(kn + m)$.*

Rešenje: Prvo algoritam grube sile: Kada bismo svaku listu sortirali razvrstavanjem, onda bi vremenska složenost bila $O(k \cdot O(n + m)) = O(kn + km)$. Ova složenost je lošija od zahtevane složenosti, pa je treba popraviti.

Umesto da sortiramo k listi pojedinačno, bilo bi poželjno sortirati ih sve odjednom. Svakom elementu j iz liste i možemo pridružiti uređen par (l_i, e_j). Nad ovim parovima primenimo radix sort, koji će prvo sortirati razvrstavanjem po brojevima e_j , a zatim po rednim brojevima listi l_i. Vremenska složenost opisanog algoritma se jednostavno izračunava iz najgore složenosti algoritma radix sort $O(d' \cdot (n' + k'))$ za $d' = 2$, $n' = nk$ i $k' = m + k$, odnosno, $O(2 \cdot (nk + (m + k))) = O(k \cdot (n + 1) + m) = O(kn + m)$.

4 Rezultati

Broj cifara/ velicina ulaza	10	100	1 000	10 000
1	19 248	/	/	/
2	22 669	/	/	/
3	28 230	/	/	/
4	31 651	136 872	979 491	/
5	37 212	173 657	1 085 139	9 119 103
6	41 917	189 910	1 527 407	9 828 273
7	46 194	227 122	1 856 328	14 492 618

Slika 11: Rezultati testiranja (nanosekunde)

Na priloženoj slici (Slika 11) prikazani su rezultati testiranja rada algoritma. Sve unete vrednosti u ćelijama tabele su predstavljene u nanosekundama. Svaki red predstavlja broj cifara ulaznih brojeva, a kolone predstavljaju veličinu ulaza.

Posmatrajući tabelu, možemo zaključiti da se vreme izvršavanja algoritma ne menja drastično kako raste broj cifara u brojevima dok veličina ulaza ostaje ista. Čak i kada se broj cifara poveća duplo, vreme ne bude duplo uvećano. Isto tako, ako posmatramo brojeve sa istim brojem cifara, a da se veličina ulaza povećava svaki put 10 puta više, možemo primetiti da se i u tom slučaju vreme izvršavanja ne uvećava 10 puta, već manje.

Na osnovu svih navedenih test primera, moše se zaključiti da je složenost algoritma radix sort zaista $O(nk)$, gde je n veličina ulaza, a k broj cifara najvećeg elementa.

Literatura

[1] Miodrag Živković, *Algoritmi*, Matematički fakultet, Beograd

[2] Miodrag Živković, Vesna Marinković, *Konstrukcija i analiza algoritama 2, Materijal sa predavanja*, Matematički fakultet, Beograd

[3] Nikola Ajzenhamer, *Konstrukcija i analiza algoritama 2, materijal sa vežbi*, Matematički fakultet, Beograd