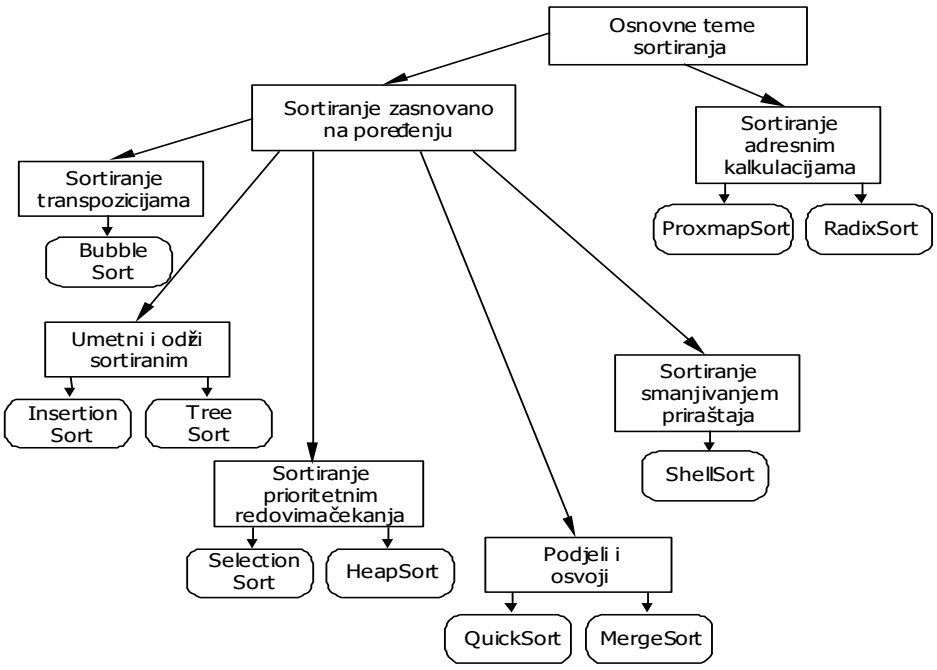
Datum: 2.11.2018.

Algoritmi za sortiranje

Problem sortiranja neuređene kolekcije ključeva u uređenu, jedan je od najčešćih u informatici i mnogo se zna o efikasnosti različitih rješenja, kao i o ograničenjima u najboljim mogućim rješenjima. Na primjer, ako neki algoritam koristi poređenje između ključeva za odlučivanje o njihovom uređenju, onda on ne može, u prosjeku, sortirati niz od *n* ključeva za vrijeme proporcionalno manje od nlogn. Veličina nlogn je donja granica za sortiranja bazirana na poređenju. Postoje i O(n) algoritmi za sortiranje koji se oslanjaju na takozvane tehnike adresnih kalkulacija (*address-calculation techniques*).

Ponekad tehnika sortiranja koja odlično radi u glavnoj memoriji, nije tako efikasna za sortiranje velikih fajlova koji se nalaze na spoljnim memorijskim medijumima. Takođe, QuickSort može sortirati *n* ključeva, u prosjeku, dva puta brže od uporedne O(nlogn) tehnike, kao što je HeapSort, iako se vrijeme sortiranja za QuickSort ponaša kao $O(n^2)$, a za HeapSort kao O(nlogn).

Upoznavanje sa karakteristikama različitih metoda sortiranja može biti od izuzetne važnosti, jer na taj način možete lakše odabrati tehniku sortiranja koja najviše odgovara posebnostima problema koji rješavate. Metode sortiranja se grupišu u različite klase koje sadrže slične teme. Jedan način za njihovo organizovanje (mada ne i jedini mogući) je prikazan na Slici 1.



Slika 1: Tehnike sortiranja

Bubble sort

Bubble sort je jedan od prvih unapređenih algoritama za sortiranje, ali se i dalje smatra veoma sporim jer radi u vremenu $O(n^2)$. Ime algoritma potiče od ideje da tokom postepenog sortiranja elementi niza kao mehurići „isplivaju" na pravo mjesto.

U ovom algoritmu se sukcesivno porede susjedni elementi niza, i vrše se zamjene mjesta ukoliko je to potrebno. Poboljšanje u odnosu na originalni algoritam donosi promjenjiva koja označava je li bilo zamjene (zamjena) pomoću koje se prekida sa izvršavanjem algoritma ukoliko u prethodnoj iteraciji nije bilo nijezne zamjene mjesta. Dodatno poboljšanje je uvođenje promjenjive j, pomoću koje se svaki put opseg koji se sortira smanjuje za 1.

Kod

```
void bubbleSort(int Niz[], int n) {
	bool promjena = true;
	int pom;
	int j = 0;
	while (promjena) {
		promjena = false;
		j++;
		for (int i = 0; i < n - j; i++) {
			if (Niz[i] > Niz[i + 1]) {
				pom = Niz[i];
				Niz[i] = Niz[i + 1];
				Niz[i + 1] = pom;
				promjena = true;
			}
		}
	}
}
```

Analiza kompleksnosti

U svakom prolasku BubbleSort će u najgorem slučaju pomjeriti najmanje jedan ključ na njegovu konačnu poziciju u sortiranom poretku, a neće ga pomjerati u slijedećim prolascima. Zbog toga, nakon najviše n prolazaka, svih n ključeva će biti u konačnom sortiranom poretku. U svakom prolasku algoritam BubbleSort pravi (n–1) poređenja ključeva. Zato je ukupan broj poređenja koji izvodi ovaj algoritam najviše n(n–1), što je $O(n^2)$.

Insertion sort O(n²).

Insertion sort spada u „umetni i održi sortiranim" algoritme i radi u vremenu O(n²). U praksi se često koristi za sortiranje malih nizova.

U ovom algoritmu se niz dijeli na dva zamišljena podniza: sortirani i nesortirani. U svakom koraku, novi ključ K koji treba umetnuti, se odvaja od lijevog kraja nesortiranog podniza i ubacuje u sortirani podniz na odgovarajuću poziciju:

- ključ K se uklanja sa lijevog kraja nesortiranog podniza, na taj način praveći rupu
- svi ključevi u sortiranom podnizu koji su veći od K se pomjeraju za jedno mjesto u desno
- K se umeće u preostalu „rupu" u sortiranom podnizu.

Proces se ponavlja dok se svi ključevi iz U ne umetnu u S, a u tom trenutku S predstavlja cijeli niz A. U početku se smatra da sortirani podniz sadrži prvi element niza, a nesortirani sve ostale.

Kod

```
void insertionSort(int Niz[], int n) {
	int i, j, pom;
	for (i = 1; i < n; i++) {
		j = i;
		while (j > 0 && Niz[j - 1] > Niz[j]) {
			pom = Niz[j];
			Niz[j] = Niz[j - 1];
			Niz[j - 1] = pom;
			j--;
		}
	}
}
```

Analiza kompleksnosti

Prosječna efikasnost algoritma InsertionSort je O(n²). Za nizove koji su skoro sortirani, efikasnost se bliži O(n).

Primjer sortiranja

Nesortirani niz; roze podniz je sortiran, a plavi nesortiran:

Index niza	0	1	2	3	4	5	6	7
Vrijednost niza x	**8**	**2**	**15**	**4**	**7**	**1**	**3**	**6**

Prvi prolaz kroz for petlju, i=1, j=1. Bijeli element se razmatra za sortiranje.

Index niza	0	1	2	3	4	5	6	7
Vrijednost niza x	**8**	**2**	**15**	**4**	**7**	**1**	**3**	**6**

Prolaz kroz while: j>0 (da), Niz[0]>Niz[1] (da) zamjena, umanji j

Index niza	0	1	2	3	4	5	6	7

Vrijednost niza x	**2**	**8**	**15**	**4**	**7**	**1**	**3**	**6**

Drugi prolaz kroz for petlju, i=2, j=2.

Index niza	0	1	2	3	4	5	6	7
Vrijednost niza x	**2**	**8**	**15**	**4**	**7**	**1**	**3**	**6**

Prolaz kroz while: j>0 (da), Niz[1]>Niz[2] (ne) IZLAZ IZ WHILE

Treći prolaz kroz for petlju, i=3, j=3.

Index niza	0	1	2	3	4	5	6	7
Vrijednost niza x	**2**	**8**	**15**	**4**	**7**	**1**	**3**	**6**

Prolaz kroz while: j>0 (da), Niz[2]>Niz[3] (da) zamjena, umanji j

Index niza	0	1	2	3	4	5	6	7
Vrijednost niza x	**2**	**8**	**4**	**15**	**7**	**1**	**3**	**6**

Prolaz kroz while: j>0 (da), Niz[1]>Niz[2] (da) zamjena, umanji j

Index niza	0	1	2	3	4	5	6	7
Vrijednost niza x	**2**	**4**	**8**	**15**	**7**	**1**	**3**	**6**

Prolaz kroz while: j>0 (da), Niz[1]>Niz[2] IZLAZ IZ WHILE

Četvrti prolaz kroz for petlju, i=4, j=4.

Index niza	0	1	2	3	4	5	6	7
Vrijednost niza x	**2**	**4**	**8**	**15**	**7**	**1**	**3**	**6**

Prolaz kroz while: j>0 (da), Niz[3]>Niz[4] (da) zamjena, umanji j

Index niza	0	1	2	3	4	5	6	7
Vrijednost niza x	**2**	**4**	**8**	**7**	**15**	**1**	**3**	**6**

Prolaz kroz while: j>0 (da), Niz[2]>Niz[3] (da) zamjena, umanji j

Index niza	0	1	2	3	4	5	6	7
Vrijednost niza x	**2**	**4**	**7**	**8**	**15**	**1**	**3**	**6**

Prolaz kroz while: j>0 (da), Niz[1]>Niz[2] IZLAZ IZ WHILE

Peti prolaz kroz for petlju, i=5, j=5.

Index niza	0	1	2	3	4	5	6	7
Vrijednost niza x	**2**	**4**	**7**	**8**	**15**	**1**	**3**	**6**

Prolaz kroz while: j>0 (da), Niz[4]>Niz[5] (da) zamjena, umanji j

Index niza	0	1	2	3	4	5	6	7
Vrijednost niza x	**2**	**4**	**7**	**8**	**1**	**15**	**3**	**6**

Prolaz kroz while: j>0 (da), Niz[4]>Niz[5] (da) zamjena, umanji j

Index niza	0	1	2	3	4	5	6	7
Vrijednost niza x	**2**	**4**	**7**	**1**	**8**	**15**	**3**	**6**

Prolaz kroz while: j>0 (da), Niz[3]>Niz[4] (da) zamjena, umanji j

Index niza	0	1	2	3	4	5	6	7
Vrijednost niza x	**2**	**4**	**1**	**7**	**8**	**15**	**3**	**6**

Prolaz kroz while: j>0 (da), Niz[2]>Niz[3] (da) zamjena, umanji j

Index niza	0	1	2	3	4	5	6	7

Vrijednost niza x	**2**	**1**	**4**	**7**	**8**	**15**	**3**	**6**

Prolaz kroz while: j>0 (da), Niz[1]>Niz[2] (da) zamjena, umanji j

Index niza	0	1	2	3	4	5	6	7
Vrijednost niza x	**1**	**2**	**4**	**7**	**8**	**15**	**3**	**6**

Prolaz kroz while: j>0 (da), Niz[0]>Niz[1] (ne) IZLAZ IZ WHILE

Šesti prolaz kroz for petlju, i=6, j=6.

Index niza	0	1	2	3	4	5	6	7
Vrijednost niza x	**1**	**2**	**4**	**7**	**8**	**15**	**3**	**6**

Prolaz kroz while: j>0 (da), Niz[5]>Niz[6] (da) zamjena, umanji j

Index niza	0	1	2	3	4	5	6	7
Vrijednost niza x	**1**	**2**	**4**	**7**	**8**	**3**	**15**	**6**

Prolaz kroz while: j>0 (da), Niz[4]>Niz[5] (da) zamjena, umanji j

Index niza	0	1	2	3	4	5	6	7
Vrijednost niza x	**1**	**2**	**4**	**7**	**3**	**8**	**15**	**6**

Prolaz kroz while: j>0 (da), Niz[3]>Niz[4] (da) zamjena, umanji j

Index niza	0	1	2	3	4	5	6	7
Vrijednost niza x	**1**	**2**	**4**	**3**	**7**	**8**	**15**	**6**

Prolaz kroz while: j>0 (da), Niz[2]>Niz[3] (da) zamjena, umanji j

Index niza	0	1	2	3	4	5	6	7
Vrijednost niza x	**1**	**2**	**3**	**4**	**7**	**8**	**15**	**6**

Prolaz kroz while: j>0 (da), Niz[1]>Niz[2] (ne) IZLAZ IZ WHILE

Sedmi prolaz kroz for petlju, i=7, j=7

Index niza	0	1	2	3	4	5	6	7
Vrijednost niza x	**1**	**2**	**3**	**4**	**7**	**8**	**15**	**6**

Prolaz kroz while: j>0 (da), Niz[6]>Niz[7] (da) zamjena, umanji j

Index niza	0	1	2	3	4	5	6	7
Vrijednost niza x	**1**	**2**	**3**	**4**	**7**	**8**	**6**	**15**

Prolaz kroz while: j>0 (da), Niz[5]>Niz[6] (da) zamjena, umanji j

Index niza	0	1	2	3	4	5	6	7
Vrijednost niza x	**1**	**2**	**3**	**4**	**7**	**6**	**8**	**15**

Prolaz kroz while: j>0 (da), Niz[4]>Niz[5] (da) zamjena, umanji j

Index niza	0	1	2	3	4	5	6	7
Vrijednost niza x	**1**	**2**	**3**	**4**	**6**	**7**	**8**	**15**

Prolaz kroz while: j>0 (da), Niz[3]>Niz[4] (ne) IZLAZ IZ WHILE

Selection sort O(n²)

Selection sort je jedan od algoritama koji za sortiranje koriste prioritetni red. U ovom slučaju, reprezentacija prioritetnog reda je nesortirani niz. Algoritam je veoma jednostavan, a u pojedinim situacijama može biti i veoma efikasan.

Ideja je sljedeća: niz koji se treba sortirati se podjeli na dva zamišljena podniza: sortirani i nesortirani; inicijalno, sortirani podniz je prazan, a nesortirani sadrži čitav niz. U svakom koraku algoritam određuje najmanji (ili najveći) element u nesortiranom podnizu i dodaje ga na kraj sortiranog podniza, sve dok nesortirani podniz ne postane prazan, odnosno, dok se ne sortira čitav niz.

Kod

```
void selectionSort(int Niz[], int n)
{
	int i, j, minIndex, pom;
	for (i = 0; i < n - 1; i++) {
		minIndex = i;
		for (j = i + 1; j < n; j++)
			if (Niz[j] < Niz[minIndex])
				minIndex = j;
		if (minIndex != i) {
			pom = Niz[i];
			Niz[i] = Niz[minIndex];
			Niz[minIndex] = pom;
		}
	}
}
```

Analiza kompleksnosti

Kako algoritam mjenja Niz[j] sa Niz[i] za svako *i* počev od n −1 do 1, SelectionSort izvodi maksimalno (n−1)-nu zamjenu. Broj izvršenih poređenja (sa ciljem određivanja minimalnog elementa u nesortiranom podnizu) je suma oblika: (n − 1) + (n − 2) + . . . + 2 + 1 , koja ima vrijednost (n*(n−1))/2 (suma prvih (n−1) prirodnih brojeva). Kako SelectionSort vrši O(n²) poređenja i O(n) zamjena, to je SelectionSort O(n²) algoritam. Ukoliko je poređenje znatno „jeftinije" od zamjene, algoritam se može smatrati efikasnim.

Primjer sortiranja

Neka je nesortirani niz:

Index niza	0	1	2	3	4	5	6	7
Vrijednost niza x	**8**	**2**	**15**	**4**	**7**	**1**	**3**	**6**

Sortiranje:

Bijeli podniz je sortiran, a sivo zasenčen nesortiran. U svakom prolazu se traži najmanji element u nesortiranom nizu (crveni) i on zamjeni mjesto sa elementom koji ima minimalan indeks (minIndex) u nesortiranom podnizu (zeleni). Ukoliko je element sa najmanjim indeksom u nesortiranom nizu ujedno i najmanji (kao što je to slučaj sa crvenom dvojkom i četvorkom), onda nema zamjene.

8	2	15	4	7	1	3	6
1	2	15	4	7	8	3	6
1	2	15	4	7	8	3	6
1	2	3	4	7	8	15	6
1	2	3	4	7	8	15	6
1	2	3	4	6	8	15	7
1	2	3	4	6	7	15	8
1	2	3	4	6	7	8	15