Projektovanje algoritama

L05. Heap. Heapsort

Algoritmi sortiranja

Sortiranje je fundamentalni problem izučavanja algoritama.

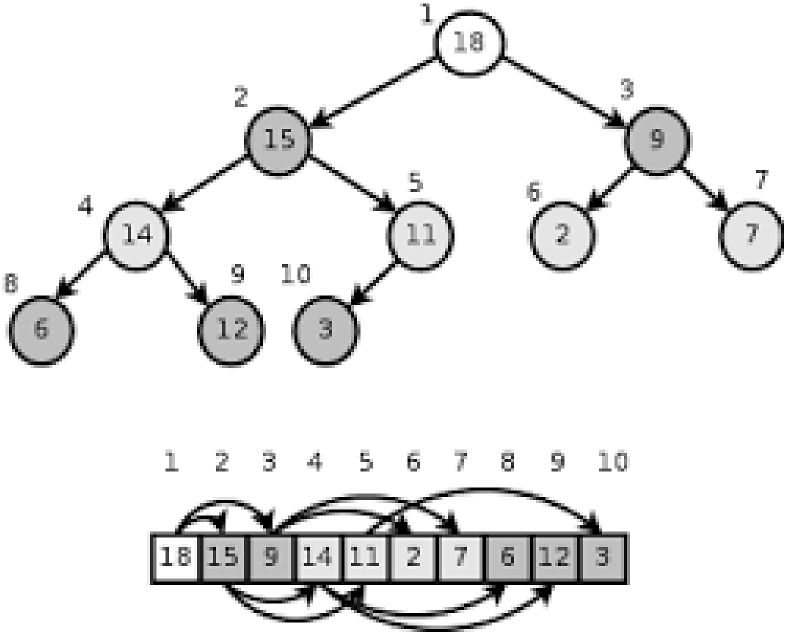
Mi ćemo se fokusirati na sortiranje **brojeva**, a algoritmi se mogu primeniti na sortiranje bilo kojih drugih tipova podataka, ukoliko je definisana relacija poređenja.

Ukoliko su podaci složeni i sastavljeni od **ključeva** i **pratećih podataka**, sortiranjem ključeva možemo da sortiramo čitavu strukturu podataka.

Algoritmi sortiranja

Naziv algoritma	Očekivano vreme trajanja	Prostorni zahtev
Insertion Sort	$O(n^2)$	unutar niza
Merge Sort	$O(n \lg n)$	dodatni prostor
	$O(n \lg n)$	unutar niza
	$O(n)$	

Struktura podataka: heap



Svaki element ima najviše 2 deteta.
Svaki element ima najviše 1 roditelja.

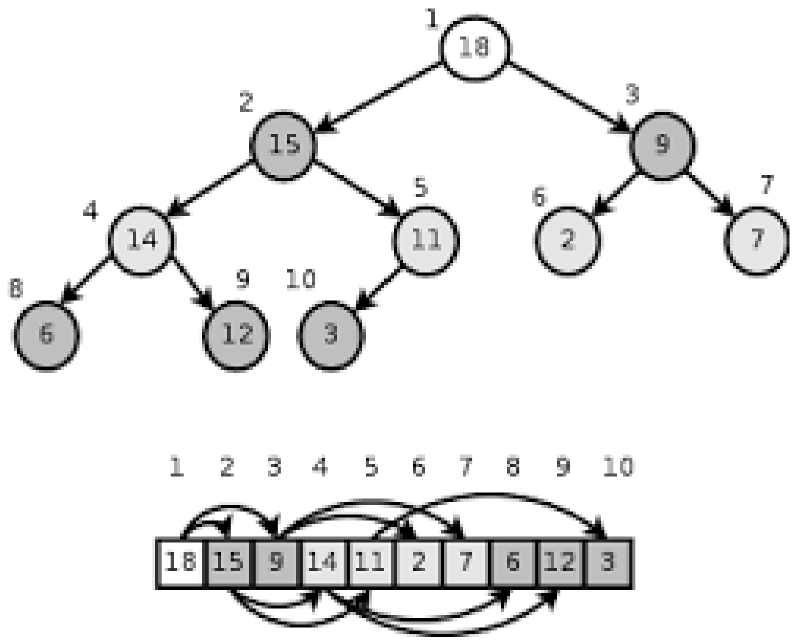
Indeksi elemenata se definišu od gore ka dole, s leva na desno.

Indeks roditelja: $\mathrm{Parent}(i) = \left\lfloor \frac{i}{2} \right\rfloor$

Indeks levog deteta: $\mathrm{Left}(i) = 2i$

Indeks desnog deteta: $\mathrm{Right}(i) = 2i + 1$

Struktura podataka: heap



MAX-HEAP ima sledeću karakteristiku:

$$A[Parent(i)] \geq A[i]$$

MIN-HEAP ima sledeću karakteristiku:

$$A[Parent(i)] \leq A[i]$$

Heap – održavanje karakteristike MAX-HEAP

Pretpostavljamo da su deca elementa sa validnom MAX-HEAP karakteristikom.
Ispravljamo moguću situaciju da je element A[i] manji od nekog svog deteta.

```
MAX-HEAPIFY(A,i)
  l = Left(i)
  r = Right(i)
  if l <= A.heap_size and A[l] > A[i]
    largest = l
  else
    largest = i
  if r <= A.heap_size and A[r] > A[largest]
    largest = r
  if largest != i
    swap(A[i], A[largest])
    MAX-HEAPIFY(A, largest)
```

Indeks niza u pseudokodu počinje od 1

Heap - održavanje karakteristike

Analiza MAX-HEAPIFY procedure

$$T(n) \leq T\left(\frac{2n}{3}\right) + \theta(1) \qquad\qquad T(n) = O(\lg(n))$$

Formiranje heap-a:

Dokazati tačnost algoritma!

```
BUILD-MAX-HEAP(A)
  A.heap_size = A.length
  for i = ⌊A.length/2⌋ downto 1
    MAX-HEAPIFY(A,i)
```

Gruba analiza: $T(n) = O(n\lg(n))$

Precizna analiza: $\boldsymbol{T(n) = O(n)}$

Heapsort

```
HEAPSORT(A)
  BUILD-MAX-HEAP(A)
  for i = A.length downto 2
    exchange A[1] with A[i]
    A.heap_size = A.heap_size - 1
    MAX-HEAPIFY(A,1)
```

$$\boldsymbol{T(n) = O(n \lg(n))}$$

Algoritmi sortiranja

Naziv algoritma	Očekivano vreme trajanja	Prostorni zahtev
Insertion Sort	$O(n^2)$	unutar niza
Merge Sort	$O(n \lg n)$	dodatni prostor
Heapsort	$O(n \lg n)$	unutar niza
	$O(n)$	

Primena heap-a: Prioritetni redovi

Prioritetni red (*Priority queue*) – struktura podataka za smeštanje skupa od S elemenata. Svaki element ima dodeljen **ključ** (*key*).

Operacije:

- INSERT (S, x) – dodavanje novog elementa x u prioritetni red S
- MAXIMUM (S) – vraća element sa najvećim ključem (najvišeg prioriteta)
- EXTRACT-MAX (S) – uklanja i vraća element sa najvećim ključem
- INCREASE-KEY (S, x, k) – povećava vrednost ključa elementa x na vrednost k

Primena? Raspoređivanje zadataka u operativnom sistemu (scheduler) - MAX
Simulacija događaja koji se dešavaju u određenim trenucima - MIN

Implementacija prioritetnih redova

```
HEAP-MAXIMUM(A)
   return A[1]
```

$T(n) = O(1)$

```
HEAP-EXTRACT-MAX(A)
   if A.heap_size < 1
      error "heap underflow"
   max = A[1]
   A[1] = A[A.heap_size]
   A.heap_size = A.heap_size - 1
   MAX-HEAPIFY(A,1)
   return max
```

$T(n) = O(\lg(n))$

Implementacija prioritetnih redova

```
HEAP-INCREASE-KEY(A,i,key)
  if key < A[i]
    error "bad key"
  A[i] = key
  while i > 1 and A[Parent(i)] < A[i]
    exchange A[i] with A[Parent(i)]
    i = Parent(i)
```

$T(n) = O(\lg(n))$

```
HEAP-INSERT(A,key)
  A.heap_size = A.heap_size + 1
  A[A.heap_size] = -Inf
  HEAP-INCREASE-KEY(A,A.heap_size,key)
```

$T(n) = O(\lg(n))$


thank you!