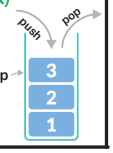
Algoritmi

Algoritem - jasen, nedvoumen, mehaničen postopek za reševanje računskih problemov
Računski problem - sestoji iz vseh možnih nalog (za dani problem) in njihovih rešitev
Vrste računskih problemov - odločitveni (Da/Ne), preštevalni (št. primernih objektov), naštevalni (vrne vse primerne objekte), isakalni (samo en objekt), optimizacijski (najboljša rešitev, kriterijski/ konstrukcijski)
Opis algoritma - naravni jezik, diagram poteka, psevdokoda
Metode razvoja - seštej, deli in vladaj, dinamično, linearno programiranje, brute force, požrešno
Snovanje algoritmov - predpogoj je dobro razumevanje problema
Cilj: opis problema; kriterij: pravilnost, učinkovitost, preprostost, implementabilnost
Pravilnost algoritma: algoritem za vse dovoljene naloge vrne pravilno (veljavno) rešitev
Dokazovanje pravilnosti:
Iniutivno razumevanje: nepopolno, subjektivno, podvženo zmotam
Testni primer: pomembni robni primeri, nemogoče preveriti vse različne primere
Formalni dokaz: zanesljivost, zahtevnost, matematika, dokaz je lahko zelo dolg
Formalni dokaz:
Zanča invarianta: trditev mora veljati pred, med in po zanki
Indukcija: metoda za dokazovanje za vse n-je
Hipoteza: trditev, ki jo hočeš dokazati
ind. predpostavka: da trditev velja za n
ind. korak: da trditev velja za n+1
Zahtevnost algoritma:
Katere in koliko virov potrebuje za svoje izvajanje v nekem modelu računanja (čas, prostor). Zahtevnost je odvisna od podatkov in velikosti naloge.
Računamo: Best, Average in Worst case
Računski problem: za vse naloge je možno najti rešitev
Abstraktni podatkovni tip je model, ki je bil uveljavljen zaradi varnosti. Kot matimatičen model je podoben algeberski strukturam in nudi natančno definicijo operaciji. Deluje po principu vmesnikov in razredov, nima implementacije in nudi uporabniški pogled na podatke.

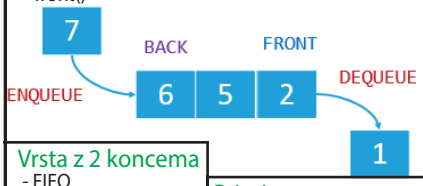
Podatkovni tipi

Sklad (stack)
- LIFO
- push(x)
- pop()
- isEmpty()
- top()



Zaporedje
- get(i)
- set(i, x)
- insert(i, x)
- delete(i)

Vrsta (queue)
- FIFO
- enqueue(x)
- dequeue()
- isEmpty()
- front()



Vrsta z 2 koncema
- FIFO
- enqueue F/B(x)
- dequeue F/B()
- front(f), back(b)

Prioritetna vrsta
- enqueue(p, x)
- dequeue()
- dequeue F/B()
- front()

Drevesa

Poddrevo: Vozlišče in vsi njegovi potomci
Globina: dolžina poti od korena do vozlišča
Višina: dolžina najdaljše poti od vozljišča do lista
Popolno dervo: vsi listi so na istem nivoju
Stopnja vozljišča: število otrok
Stopnja drevesa: max stopnja vozljišča
Polno drevo: vsa vozlišča so polna (binarno drevo - vsako vozljišče 2 otroka)
Celovito drevo: vsi nivoji razen zadnjih so polni

Popolno drevo	Dvojiško	Trojiško	d-tiško
Št. Vozlj na nivoju:	2^i	3^i	d^i
Št. Listov (n)	2^n	3^n	d^n
Št. Notranjih vozljišč	$2^n - 1$	$\frac{3^n-1}{2}$	$\sum_{i=0}^{n-1} d^i = \frac{d^n-1}{d-1}$
Št. Vozljišč, n	$2^{n+1}-1$	$\frac{3^{n+1}-1}{2}$	$\sum_{i=0}^{n} d^n = \frac{d^{n+1}-1}{d-1}$
Višina	$\ln(n+1)-1$	$\log_3(2n+1)-1$	$\log_d((d-1)\cdot n+1)-1$
Celovito drevo			
Št vozljišč	$2^n \le n \le 2^{n+1}-1$	$\frac{3^n+1}{2} \le n \le \frac{3^{n+1}-1}{2}$	
višina	lgn	$\log_3(2n)$	$\log_d((d-1)\cdot n)$

Polja

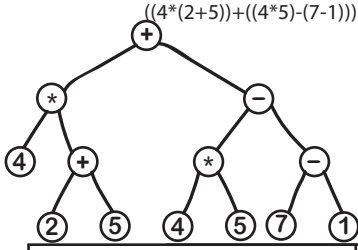
Statično polje: kapaciteta se ne spreminja (lahko je dinamično alocirano). Nevarnost podlivov in prelivov.
Dinamično polje: enostavno spreminjanje velikosti polja. Ob prelivu se prepiše v večje, ali pa manjše pri neizkoriščenosti.
Amortizacija: skupno zahtevnost celovitega zaporedja operacij porazdelimo na (amortiziramo) operacijo.
Enojno povezan seznam: sez. podan s prvim, vsak ima kazalec na naslednjega
Dvojno povezan seznam: enako kot enojni, le da ima kazalec še nazaj
Čuvaj: prazen element, ki služi kot prvi in zadnji d. in je vedno prisoten
Cikel: kazalec iz zadnjega na prvega in/ali obratno
Implicitna pod. str.: potrebuje malo prostora, poleg prostoraza podatke
Eksplicitna pod. str.: potrebuje dodatni prostor za hranjenje kazalca (next, prev, povezave)

Kopica

Kopica: celovito urejeno dvojiško drevo
Min kopica: v korenu je min element
Max kopica: v korenu je max element

	Operacija	Polje	EPS	DPS
shlad, vrsta	enqueue(x)	$O(1)$	$O(1)$	$O(1)$
	dequeue(), pop()	$O(1)$	$O(1)$	$O(1)$
	enqueueF, push()	$O(1)$	$O(1)$	$O(1)$
	dequeueB	$O(1)$	$O(n)$	$O(1)$
zaporedje	get(i)	$O(1)$	$O(n)$	$O(n)$
	set(i, x)	$O(1)$	$O(n)$	$O(n)$
	find(x)	$O(n)$	$O(n)$	$O(n)$
	Insert(i, x)	$O(n)$	$O(n)$	$O(n)$
	delete(i)	$O(n)$	$O(n)$	$O(n)$
množica	remove(x)	$O(n)$	$O(n)$	$O(n)$
	add(x)	$O(1)$	$O(n)$	$O(1)$
	addUnique(x)	$O(n)$	$O(n)$	$O(n)$

Premi obhod
+*4+25-*45-71
Obratni obhod
425+*45*71--+
Vmesni obhod
((4*(2+5))+((4*5)-(7-1)))



Asimptotska notacija

$$\Omega(g(n)) = \{f(n) \mid \exists c, n_0 > 0 \forall n \ge n_0 \ 0 \le cg(n) \le f(n)\}$$
$$O(g(n)) = \{f(n) \mid \exists c, n_0 > 0 \forall n \ge n_0 \ 0 \le f(n) \le cg(n)\}$$
$$\Theta(g(n)) = \{f(n) \mid \exists c_1, c_2 \ n_0 > 0 \forall n \ge n_0 \ 0 \le c_1 g(n) \le f(n) \le c_2 g(n)\}$$

SiftUp

```
fun shiftUp(c)
    while(c>0)
        p=(c-1)/2
        if (items[p] >= items[c]):
            items[c]
        then
            break
        swap(items, c, p)
        c = p
    endwhile
end
```

SiftDown

```
fun shiftDown
    c= 2*p+1
    while c <= last do
        if c+1 >= last and a[c+1] > a[c]
         c+=1
         if a[p] >= a[c]
            break
        swap(a, p, c)
        p = c
        c = p*2 +1
end
```

namig	notacija	limita
<	o	L = 0
≤	O	$0 \le L < \infty$
=	Θ	$0 < L < \infty$
≥	Ω	$0 < L \le \infty$
>	ω	$L = \infty$
~	~	L = 1

dequeue

```
fun dequeue()
    x = items[front]
    items[front] = null # postop.
    front = (front + 1) % length
    return x
end
```

enqueue

```
fun enqueue(x)
    items[back] = x
    back = (back + 1) % length
end
```

Obratni obhod

```
fun postorder(v) is
    if v == null then return
        postorder(v.left)
        postorder(v.right)
        println(v)
end
```

$$L = \lim_{n \to} \frac{f(n)}{g(n)}$$

Premi obhod

```
fun preorder(v)
    if v == null then return
        println(v)
        preorder(v.left)
        preorder(v.right)
end
```

Vmesni obhod

```
fun inorder(v) is
    if v == null then return
        inorder(v.left)
        println(v)
        inorder(v.right)
end
```