

Algoritmi in podatkovne strukture 1

Visokošolski strokovni študij Računalništvo in informatika

**Podatkovni tipi
in strukture**



Predstavitev podatkov

- Seštevanje celih števil

$$\begin{array}{r} 1357 \\ + 2846 \\ \hline 1110 \\ \hline 4203 \end{array}$$

Seštevanje dveh celih števil

- števíli napišemo eno pod drugo
- začnemo pri enicah in pomikamo proti levi
- seštevamo po dve števki in prenos

Predstavitev podatkov

- Seštevanje celih števil

ROMAN NUMERALS				
I	II	III	IV	V
1	2	3	4	5
VI	VII	VIII	IX	X
6	7	8	9	10
XI	XII	XIII	XIV	XV
11	12	13	14	15
XVI	XVII	XVIII	XIX	XX
16	17	18	19	20
XXV	XXX	XXXV	XL	XLV
25	30	35	40	45
L	LX	LXX	LXXX	XC
50	60	70	80	90
C	CXXV	CCL	D	M
100	125	250	500	1000

Evropa v času,
ko je živel al-Khwārizmī,
uporablja rimske številke!

MCDLI
+ **DLXVII**

MMXVIII

M=1000, D=500, C=100,
L=50, X=10, V=5, I=1

Podatkovni tipi

- Podatkovni tip
 - množica **vrednosti** in način **predstavitve** posameznih vrednosti
 - Java byte: vrednosti od -128 do 127 predstavljene z dvojiškim komplementom
 - množica **operacij** in njihove **implementacije**
 - Java byte: operacije +, -, *, / in njihove implementacije
- Abstraktni podatkovni tip
 - model za podatke
 - brez predstavitve in brez implementacije



Podatkovni tipi

- Objektni opis
 - podatkovna struktura je prejemnik metode
- Klasični opis
 - podatkovna struktura podana kot argument

Tip s
s.operacija(x)

Tip s
operacija(s, x)

Podatkovni tipi

Collection

```
isEmpty()  
isFull()  
clear()  
count()  
contains(x)
```

Set/Bag

```
find(x)  
add(x)  
remove(x)
```

SortedSet

```
min()  
max()
```

Indexed

```
get(i)  
set(i, x)
```

Stack

```
push(x)  
pop()  
top()
```

Queue

```
enqueue(x)  
dequeue()  
front()
```

Deque

```
enqueueFront(x)  
dequeueBack()  
back()
```

PriorityQueue

```
enqueue(p, x)
```

Sequence

```
insert(i, x)  
delete(i)
```

Map

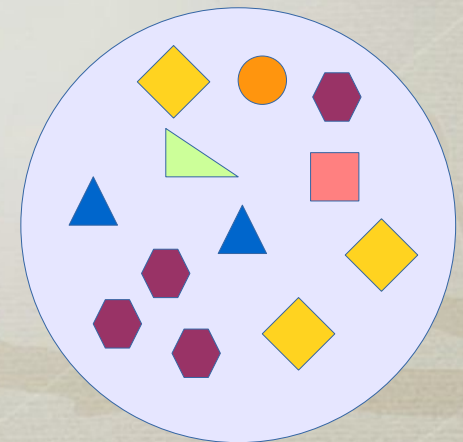
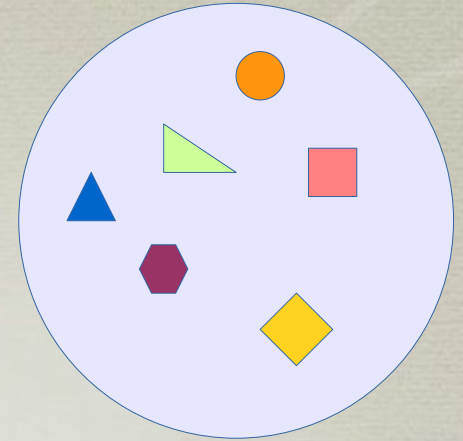
```
get(k)  
put(k, v)  
remove(k)
```

Množica

- Množica (*set*)
 - vsebuje enolične elemente
 - brez vrstnega reda
- Vreča (*bag, multiset*)
 - kot množica
 - dovoljuje ponavljanje elementov
- Urejena množica/vreča
 - vsebuje še operaciji $\min()$ in $\max()$

Set/Bag

```
find(x)  
add(x)  
remove(x)
```



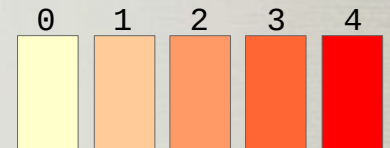
Zaporedje

- Indeksiran seznam (indexed)
 - dostop glede na indeks/pozicijo
`get(i)`, `set(i, x)`

Indexed

```
get(i)  
set(i, x)
```

- Zaporedje (*sequence*)
 - vstavljanje na pozicijo: `insert(i, x)`
 - brisanje na dani poziciji: `delete(i)`



Sequence

```
insert(i, x)  
delete(i)
```

Sklad

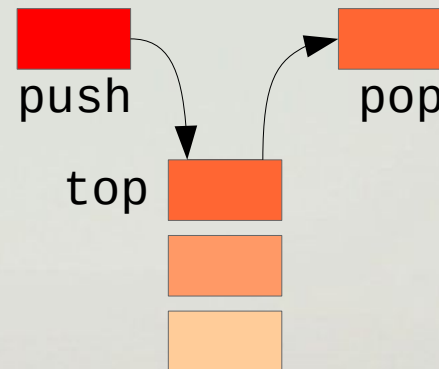
- Sklad (*stack*)
 - LIFO – *last-in, first-out*
 - vrh sklada `top`
 - operaciji `push` in `pop`



Stack

`push(x)`
`pop()`

`isEmpty()`
`top()`



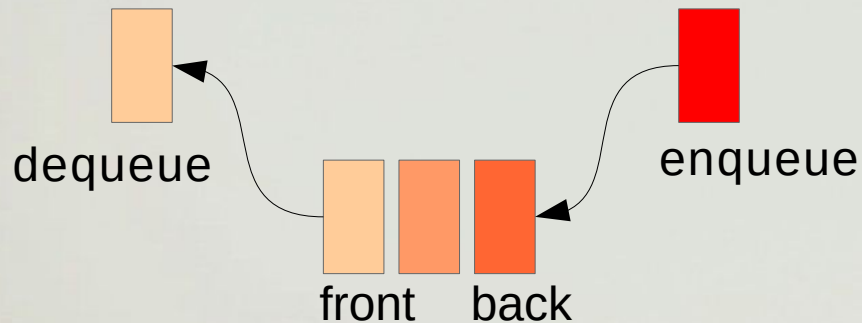
Vrsta

- Vrsta (*queue*)
 - FIFO – *first-in, first-out*
 - sprednji in zadnji konec vrste
 - operaciji enqueue in dequeue

Queue

`enqueue(x)`
`dequeue()`

`isEmpty()`
`front()`



Dvostrana vrsta

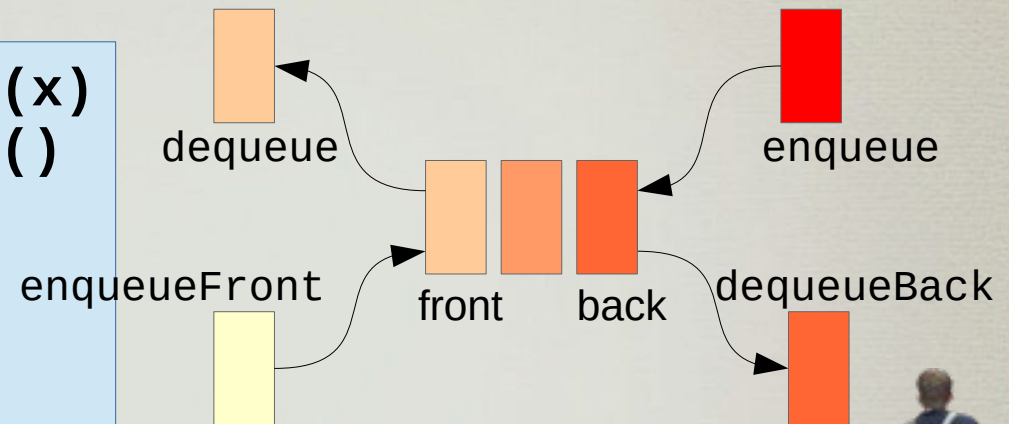
dvrsta?

- Dvostrana vrsta (*deque* ali *dequeue*)
 - dodajanje in odvzemanje spredaj in zadaj

Deque

```
enqueue(x) = enqueueBack(x)  
dequeue() = dequeueFront()  
enqueueFront(x)  
dequeueBack()
```

```
isEmpty()  
front()  
back()
```



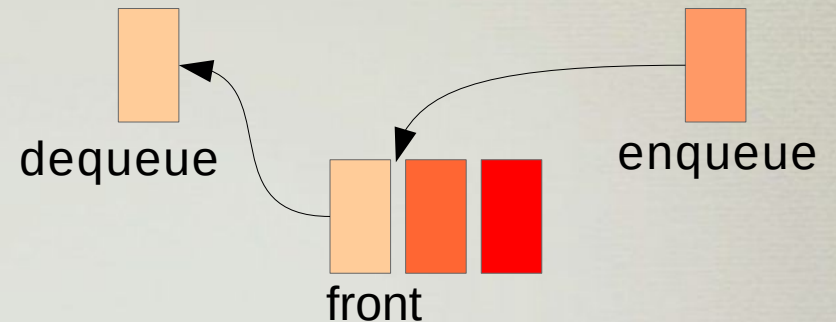
Prednostna vrsta

- Prednostna vrsta (*priority queue*)
 - odvzemanje spredaj
 - dodajanje s prednostjo

PriorityQueue

enqueue(p, x)
dequeue()

front()

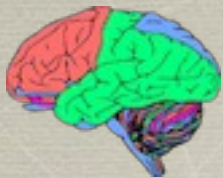


Slovar

- Slovar (*dictionary, map*)
 - podoben množici
 - dostop do elementov preko ključa
 - dostop: `get(k)`, `put(k, v)`
 - odstranjevanje: `remove(k)`

Map

```
get(k)  
put(k, v)  
remove(k)
```



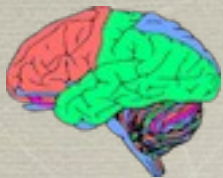
Algebrski pogled

- Algebrske strukture
 - matematična definicija obnašanja tipa
 - strukture
 - brez (dvojiške) operacije
 - npr. množica, množica z unarno operacijo
 - z eno operacijo +
 - grupoid (magma), semigrupa, monoid, grupa, ...
 - z dvema operacijama + in *
 - polkolobar, kolobar, obseg, algebra
 - več operacij?



Algebrski pogled

- $Nat =$
 - types: Nat
 - operations: $0: \rightarrow Nat$
 $succ: Nat \rightarrow Nat$
 $add: Nat, Nat \rightarrow Nat$
 - equations: $n, m \in Nat$
 $add(n, 0) = n$
 $add(n, succ(m)) = succ(add(n, m))$



Algebrski pogled

- *Int* =

- types: Int

- operations: $0: \rightarrow \text{Int}$

- $\text{succ}: \text{Int} \rightarrow \text{Int}$

- $\text{pred}: \text{Int} \rightarrow \text{Int}$

- $\text{add}: \text{Int}, \text{Int} \rightarrow \text{Int}$

- equations: $n, m \in \text{Int}$

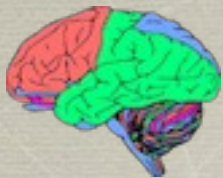
- $\text{succ}(\text{pred}(n)) = n$

- $\text{pred}(\text{succ}(n)) = n$

- $\text{add}(n, 0) = n$

- $\text{add}(n, \text{succ}(m)) = \text{succ}(\text{add}(n, m))$

- $\text{add}(n, \text{pred}(m)) = \text{pred}(\text{add}(n, m))$



Algebrski pogled

- *Stack* =

- types:

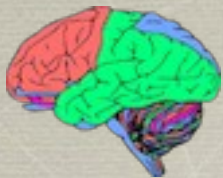
$\text{Item} = \text{Int} \cup \text{Nat}$
 $\text{Stack} = \text{Item}^* \cup \{\text{error}\}$

- operations:

$\text{empty}: \rightarrow \text{Stack}$
 $\text{push}: \text{Item}, \text{Stack} \rightarrow \text{Stack}$
 $\text{pop}: \text{Stack} \rightarrow \text{Stack}$
 $\text{top}: \text{Stack} \rightarrow \text{Item}$

- equations:

$\text{pop}(\text{push}(x, s)) = s$
 $\text{top}(\text{push}(x, s)) = x$
 $\text{pop}(\text{empty}) = \text{empty}$
 $\text{top}(\text{empty}) = \text{error}$



Algebrski pogled

- Zakaj se sploh truditi z *matematiko*?
 - ker omogoča izredno jasen in natančen pogled na podatkovne strukture
 - ker omogoča **dokaz pravilnosti** delovanja
- Kdaj resnično rabimo pravilnost?
 - preprečevanje katastrof
 - življenjsko kritične aplikacije
 - jedrske elektrarne, avtonomna vožnja
 - varnostno kritične aplikacije
 - bančne aplikacije

