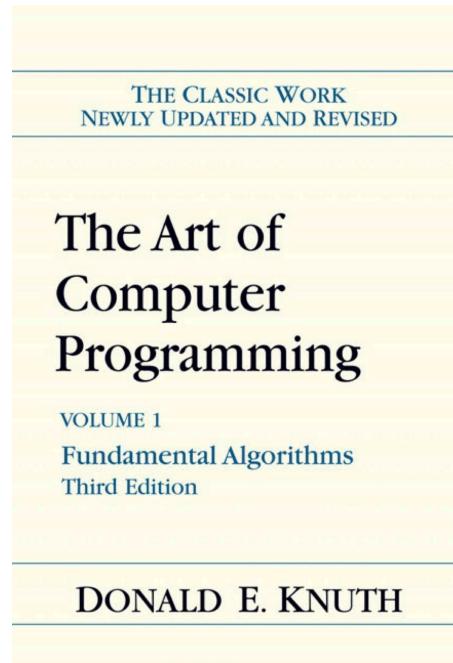


# Algoritmizace

## Základní datové struktury



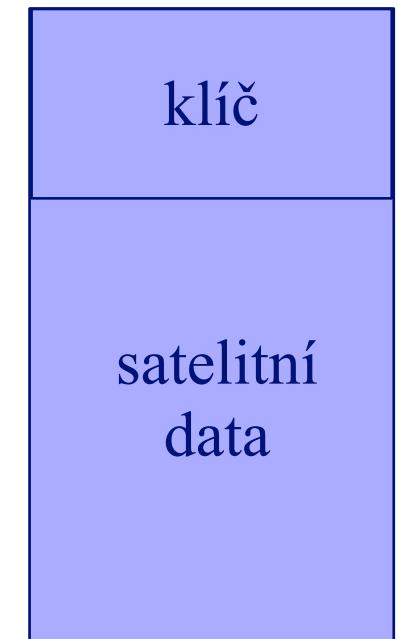
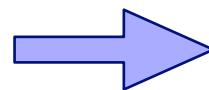
# Dynamické množiny

## Množina

- matematika / informatika
- statická / dynamická

## Abstraktní datový typ

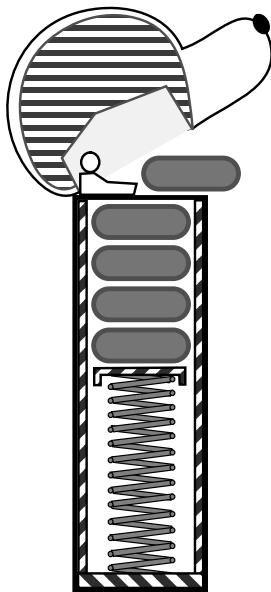
- množina prvků
- operace nad nimi
- různé implementace prostřednictvím
- konkrétních datových struktur



# Zásobník / Stack

Množina prvků, k nimž přistupujeme metodou  
**LIFO**

- last-in, first-out



# Zásobník v reálném světě



# Zásobník – operace

## Operace

- Vlož / Push – vloží zadaný prvek do zásobníku
- Odeber / Pop – odebere a vrátí prvek, který byl vložen jako poslední

## Implementace v poli

- $z[0]$  dno zásobníku
- $z[-1]$  vrchol zásobníku

## Python: seznam (list), metody

- `pop()` odebere a vrátí poslední prvek v čase  $O(1)$
- `append(x)` vloží  $x$  na konec seznamu v čase
  - »  $\Theta(n)$  – nejhorší případ
  - »  $O(1)$  – amortizovaná složitost

# Zásobník v poli

```
class ZasobnikPole:

    def __init__(this):
        this.pole = [ ] # prázdný zásobník

    def push(this,x):
        this.pole.append(x)

    def pop(this):
        # předpokládá neprázdný zásobník
        return this.pole.pop()
```

při nesplnění je vhodné vyvolat  
výjimku !

# Fronta / Queue

Množina prvků, k nimž přistupujeme metodou **FIFO**

- first-in, first-out



# Fronta – operace

## Operace

- Vlož / **Enqueue** – vloží zadaný prvek do fronty
- Odeber / **Dequeue** – odebere a vrátí prvek, který byl vložen jako první

## Implementace v poli

- $z[0]$  začátek fronty, zde se odebírá
- $z[-1]$  konec fronty, za něj se vkládá

## Python: seznam (list), metody

- ✓ `append(x)` vloží  $x$  na konec seznamu, jako u zásobníku
- ✗ `pop(0)` odebere a vrátí první prvek seznamu v čase  $\Theta(n)$ 
  - po odebrání je třeba posunout všechny zbývající prvky “doleva”



# Problém: složitost pop(0)

① Zavedeme proměnnou **zacatek** s indexem začátku fronty

- **Dequeue** vrátí pole[ **zacatek** ]
- **zacatek** +=1

✓ **Dequeue** v čase  $O(1)$

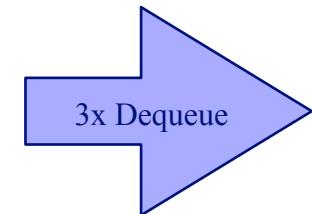
✗ Prostor  $\Theta(m)$

- $m = \#$  operací **Enqueue**

# Dequeue bez pop(0)

0	1	2	3	4	5	6	7	8	9
5	7	6	9	8	15	11	16	14	10

začátek



0	1	2	3	4	5	6	7	8	9
5	7	6	9	8	15	11	16	14	10

začátek



# Problém: složitost pop(0)

## ② Posouvat budeme jen občas

- Dequeue má lepší amortizovanou složitost

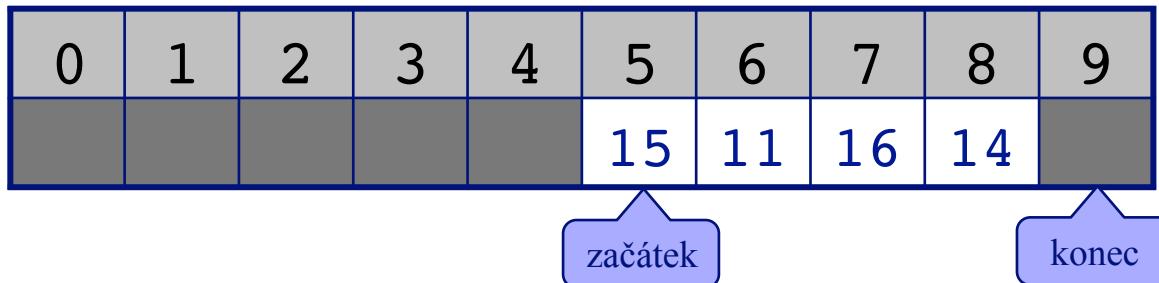
## ③ Stanovíme maximální kapacitu fronty

- posouváme jen když není místo na právě vkládaný prvek
- Dequeue v čase  $O(1)$
- Enqueue v čase  $O(n)$  – nejhorší,  $O(1)$  – nejlepší

## ④ Cyklická fronta

- v “zacykleném” poli stanovené kapacity
- bez posouvání
- po dosažení `pole[-1]` pokračujeme na `pole[0]`
- potřebujeme si pamatovat **zacatek** a **konec** fronty

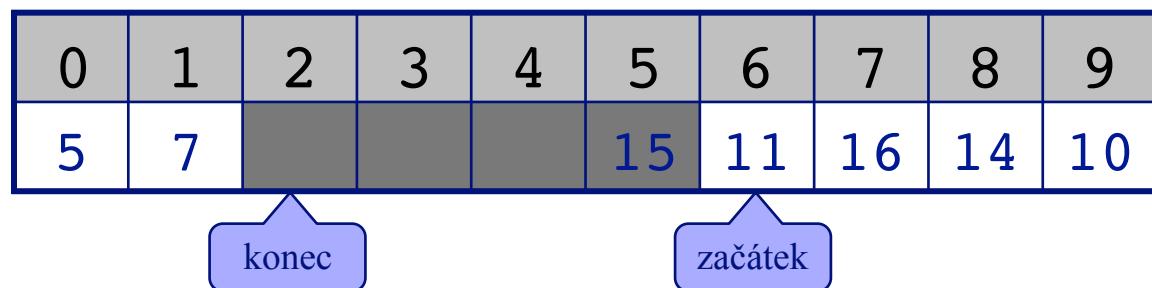
# Cyklická fronta



| Enqueue 10, 5, 7



## Dequeue



# Cyklická fronta v poli

obě operace v čase  $O(1)$  !

```
class FrontaPole:
```

```
    def __init__(this,kapacita):
```

```
        this.zacatek, this.konec = 0, 0
```

```
        this.pole = [None] * kapacita
```

```
        this.kapacita = kapacita
```

```
    def enqueue(this,x):
```

```
        # předpokládá volnou kapacitu
```

```
        this.pole[this.konec] = x
```

```
        this.konec = (this.konec + 1) % this.kapacita
```

nedostatečná kapacita ?  
alokace většího pole  
+ kopírování

```
    def dequeue(this):
```

```
        # předpokládá neprázdnou frontu
```

```
        x = this.pole[this.zacatek]
```

```
        this.zacatek = (this.zacatek + 1) % this.kapacita
```

```
        return x
```

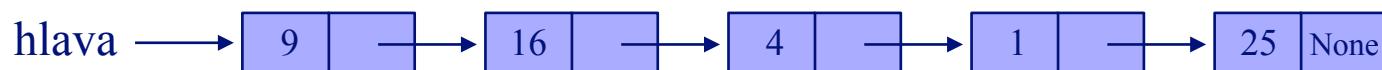
# Spojové seznamy

## Posloupnost prvků

- každý prvek je prezentován **uzlem**
- který kromě dat obsahuje i **odkaz**
- na uzel následující / předchozí (pokud existuje)

## Lineární spojový seznam

- každý uzel vyjma posledního odkazuje na uzel následující



# Uzel spojového seznamu

```
class Uzel:  
    """třída pro reprezentaci uzlu"""  
    def __init__(this,  
                 x = None,  
                 dalsi = None):  
        this.info = x  
        this.dalsi = dalsi
```

# Spojový seznam jako třída

```
class SpojovySeznam:  
    def __init__(this):  
        """vytvoří prázdný seznam"""  
        this.hlava = None # hlava seznamu  
    def najdi(this,klic):  
        """zjistí zda ∃ prvek s klicem"""  
        p = this.hlava  
        while p != None and p.info != klic:  
            p = p.dalsi  
        return p != None
```