Programovanie v jazyku Python

údajové štruktúry

prednáška 6

Katedra kybernetiky a umelej inteligencie
Technická univerzita v Košiciach
Ing. Ján Magyar

Údajová štruktúra

- “údajový typ”
- zvyčajne sa skladá zo skupiny elementárnych hodnôt
- cieľom je zvýšiť efektivitu práce s údajmi
- štandardizované blueprinty (šablóny) pre reprezentáciu údajov v počítačoch
- rôzne úrovne abstrakcie
- vieme ich implementovať rôznymi spôsobmi, ale základná myšlienka a spôsob fungovania sú stále rovnaké

Pole (Array)

- (viacrozmerné) usporiadanie hodnôt rovnakého typu
- k údajom pristupujeme pomocou indexov - na základe ukladania v pamäti
- v Pythone dve možnosti:
  - zoznam zoznamov (zoznamov zoznamov...)
  - pomocou knižnice `numpy`
    - C-čková implementácia polí
    - vyšší výkon a rýchlejšie výpočty ako pomocou zoznamov

Numpy

- implementácia polí v Pythone (v skutočnosti sú to matice)
- podobné, ako zoznamy, iba nemeniteľné (hodnoty však vieme aktualizovať)
- vytvorenie polí:

```
import numpy as np
my_array = np.array([2, 3, 4])
```

- konvertovanie vstupu na pole:

```
numpy.asarray(a, dtype=None, order=None)
```

Typ polí

- každé pole má typ `ndarray`, jednotlivé prvky musia byť rovnakého typu

  ```
  my_array.dtype
  ```

- typy prvkov nie sú pythonovské primitívne typy, sú to vlastné implementácie knižnice `numpy`

Tvar polí

- každé pole má jednu alebo viac dimenzií
- tvar poľa je definovaný rozmermi

```
my_array.shape
```

- vracia n-ticu s rozmermi, napr.: `(2, 3)`, kde prvá hodnota je vonkajší rozmer, a postupuje smerom dnu
- ak je pole dvojdimenzionálne, prvý rozmer je počet riadkov, druhý je počet stĺpcov*

Úprava rozmerov poľa

- `numpy.reshape(array, newshape, order='C')`
  - upraví tvar poľa na požadované rozmery
  - počet prvkov v poli a počet prvkov v novom tvare musí byť rovnaký
  - order určí poradie pridávania prvkov (`C` alebo `F`)
  - vracia nové pole
- `numpy.flatten(order='C')`
  - vracia nové pole - vektorová reprezentácia (jeden riadok, resp. stĺpec)

zeros() a ones()

- metódy slúžia na inicializáciu matice s hodnotami 0 alebo 1
- `numpy.zeros(shape, dtype=float, order='C')`
- `numpy.ones(shape, dtype=float, order='C')`

Pridávanie prvkov

- implementuje konkatenáciu polí
- `numpy.hstack(tup)`
  `numpy.hstack((array1, array2))`
- `numpy.vstack(tup)`
  `numpy.vstack((array1, array2))`
- pre vytvorenie polí dynamicky je lepšie použiť zoznamy, a následne vygenerovať pole

Indexovanie a slicing v numpy

- zásady sú rovnaké ako v Pythone, numpy ale umožňuje skrátený zápis

```
np.array([(1,2,3), (4,5,6)])
print('First row:', e[0])
print('Second column:', e[:,1])
print('Second row, first two values:', e[1, :2])
```

Aritmetické operácie nad maticami

- primitívne operácie fungujú element-wise

```
test = np.array([1, 2, 3])
test += 1

test [2, 3, 4]
```

Štatistické metódy

- `np.min(array)`
- `np.max(array)`
- `np.mean(array)`
- `np.median(array)`
- `np.std(array)`

Maticové operácie

- `np.transpose(array, axes=None)`
- `np.dot(array1, array2)`
- `np.matmul(array1, array2)`
  - ak polia sú dvojrozmerné - štandardné násobenie
  - ak polia majú jeden rozmer - vygeneruje sa z nich dvojrozmerné pole
  - ak polia majú viac rozmerov, ako dva - považujú sa za zásobník dvojrozmerných matíc

Generovanie rozsahov

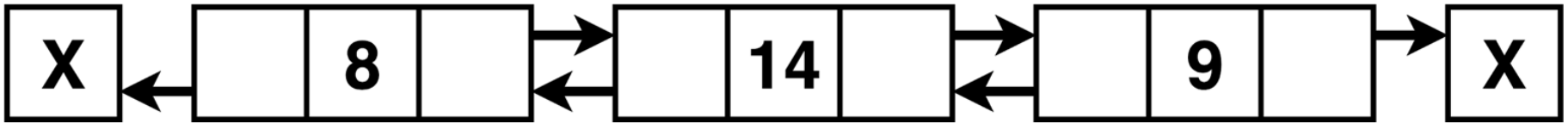
- `numpy.arange(start, stop, step)`
  - funguje rovnako ako `range()`
- `numpy.linspace(start, stop, num=50, endpoint=True)`
  - vygeneruje `num` hodnôt v rovnakej vzdialenosti od seba v intervale `[start, stop]`
  - `endpoint` určí, či interval má obsahovať hodnotu stop

Spojkový zoznam

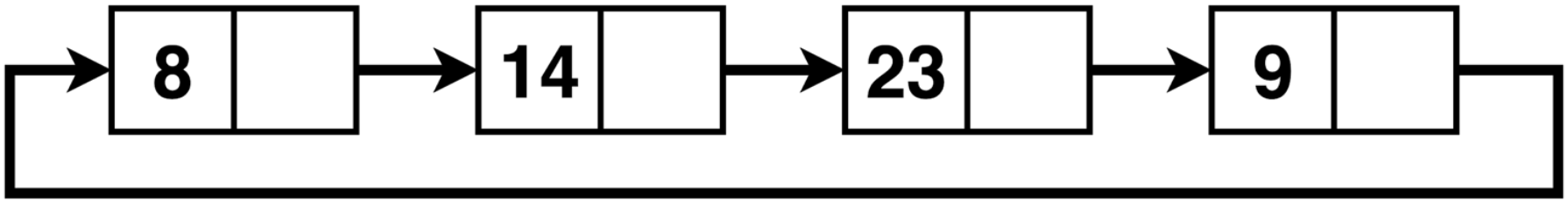
- lineárna skupina údajov, kde prvky nemusia byť uložené v pamäti vedľa seba
- každý prvok obsahuje
  - údaj
  - odkaz na ďalší prvok v zozname (alebo NULL)
- efektívne pridávanie nových prvkov na ľubovoľnú pozíciu
- varianty
  - obojsmerné spojkové zoznamy
  - kruhové spojkové zoznamy

Jednosmerný spojkový zoznam

Obojsmerný spojkový zoznam



Kruhový zoznam

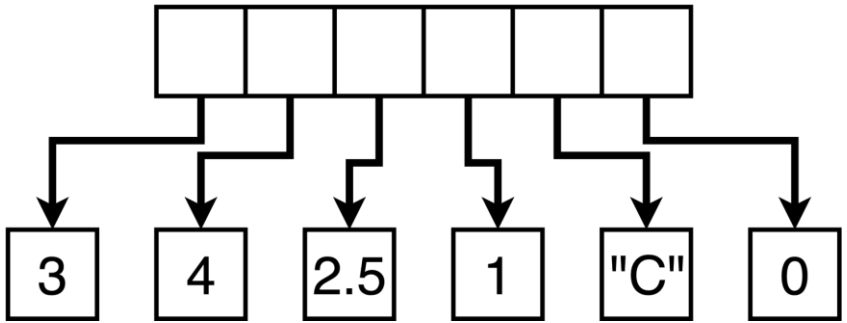


Porovnanie polí a spojkových zoznamov

Operácia	**Pole**	**Spojkový zoznam**
Pristupovanie	*O(1)*	*O(n)*
Pridanie na začiatok	*O(n)*	*O(1)*
Pridanie (priemer)	*O(n)*	*O(1)**
Pridanie na koniec	*O(1)*	*O(n)*
Zmazanie	*O(1)/O(n)*	*O(1)**

Zoznamy (list) v Pythone

- zoznamy môžu obsahovať hodnoty rôznych typov - ako zabezpečiť konštantný čas pristupovania k jednotlivým prvkom?
- na najvyššej vrstve sú zoznamy implementované ako spojkové zoznamy



Zásobník (Stack)

- dynamická množina prvkov typu LIFO - last-in, first-out
- základné operácie:
  - inicializácia - CREATE
    - vytvorí prázdny zásobník, môže byť súčasťou PUSH
  - pridanie - PUSH
    - pridá prvok na vrchol zásobníku
  - zmazanie - POP
    - zmaže prvok z vrcholu zásobníku
  - prístup ku vrcholnému prvku - TOP
  - prázdnosť zásobníku - IS_EMPTY

Zoznam ako zásobník

- CREATE

```
stack = list()
```

- PUSH

```
stack.append(value)
```

- POP

```
stack.pop()
```

- TOP

```
stack[-1]
```

- IS_EMPTY

```
len(stack) == 0
```

Použitie zásobníkov

- backtracking pri algoritmoch kde hľadáme riešenie princípom pokus-omyl
  - vyhľadávanie do hĺbky
- zásobník volaní

Front (Queue)

- dynamická množina prvkov typu FIFO - first-in, first-out
- základné operácie:
  - inicializácia - CREATE
    - vytvorí prázdny front, môže byť súčasťou ENQUEUE
  - pridanie - ENQUEUE
    - pridá prvok na koniec frontu
  - zmazanie - DEQUEUE
    - vymaže prvok zo začiatku
  - prístup k prvému prvku - HEAD
  - prístup k ostatným prvkom - TAIL
  - prázdnosť frontu - IS_EMPTY

Fronty v Pythone

- údajový typ deque

  ```
  from collections import deque
  ```

  alebo pomocou obojsmerného spojkového zoznamu
- CREATE

  ```
  queue = deque([maxlen=10])
  ```
- ENQUEUE

  ```
  queue.append(value)
  ```
- DEQUEUE

  ```
  queue.popleft()
  ```

Fronty v Pythone

- HEAD

```
queue[0]
```

- TAIL

```
list(queue)[1:]
```

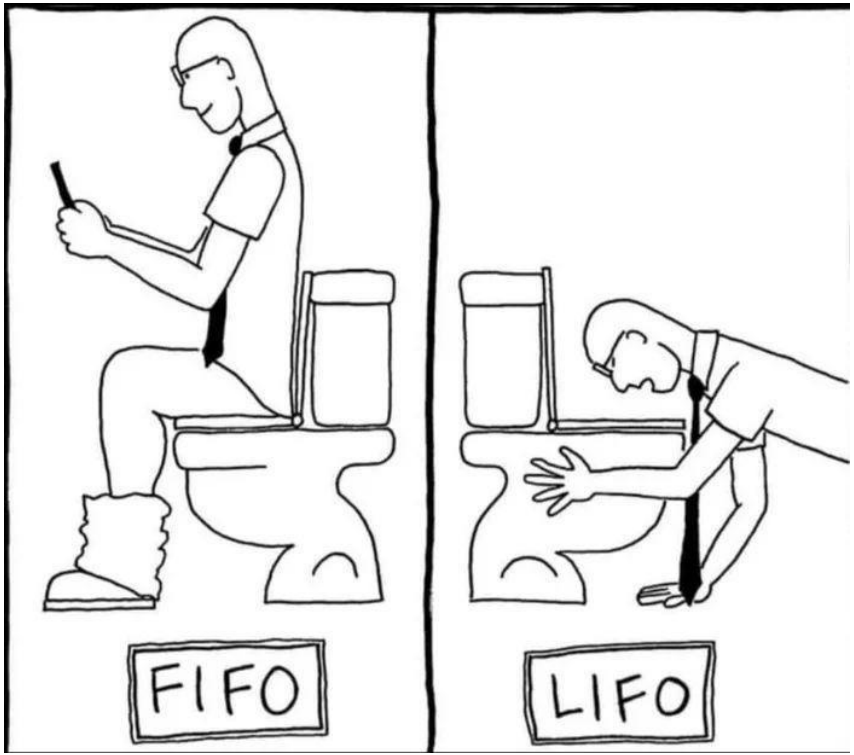
- IS_EMPTY

```
len(queue) == 0
```

Použitie frontov

- spracovanie požiadaviek
- komunikácia medzi dvoma procesmi
- posielanie správ
- dočasné ukladanie údajov pre neskoršie spracovanie

LIFO/FIFO



Hašovanie

- rozdeľuje údaje na menšie skupiny (buckety)
- predpokladá sa, že pristupovanie k jednotlivým skupinám sa uskutoční v konštantnom čase
- zjednodušuje vyhľadávanie z *O(n)* na *O(n/k)* v ideálnom prípade, kde *k* je počet skupín (bucketov)
- spracovanie hodnoty sa uskutoční v niekoľkých krokoch
  a. hašovacia funkcia vypočíta hash hodnotu
  b. pristupujeme ku skupine na základe hash hodnoty (použije sa ako kľúč)
  c. vykoná sa požadovaná operácia (pridávanie, načítanie, zmazanie)

Hašovacia funkcia

- vstupom je hodnota určitého typu, ktorú funkcia mapuje na hash hodnotu
- hash hodnota je vždy jednoduchšia ako vstup
- funkcia dáva vždy rovnaký výstup pre ten istý vstup
- ten istý výstup môže byť vypočítaný pre rôzne vstupy (kolízia)
- obor hodnôt (výstupy) je zvyčajne presne definovaný interval (množina hodnôt)
- ideálna hašovacia funkcia mapuje vstupné hodnoty do skupín uniformne, t.j. každá skupina bude mať približne rovnaký počet prvkov

Hašovacia tabuľka

- údajová štruktúra typu asociatívne pole
- obsahuje hodnoty typu kľúč-hodnota
- kľúč môže byť
  - výsledok hašovacej funkcie

```
idx = hash_function(input_value)
```

  - výsledok hašovacej funkcie namapovaný na určité čísla:

```
hash_value = hash_function(input_value)
idx = hash_value % number_of_buckets
```

Riešenie kolízií

- ideálna hašovacia funkcia by mala prideliť práve jeden prvok do každej skupiny
- v skutočnosti sa dochádza ku kolíziám, každá hodnota v hash tabuľke je v podstate zoznam (môže byť implementovaný rôznymi spôsobmi)
- efektivita hashovania sa vyhodnocuje na základe load factoru:

$$load\ factor = n/k$$

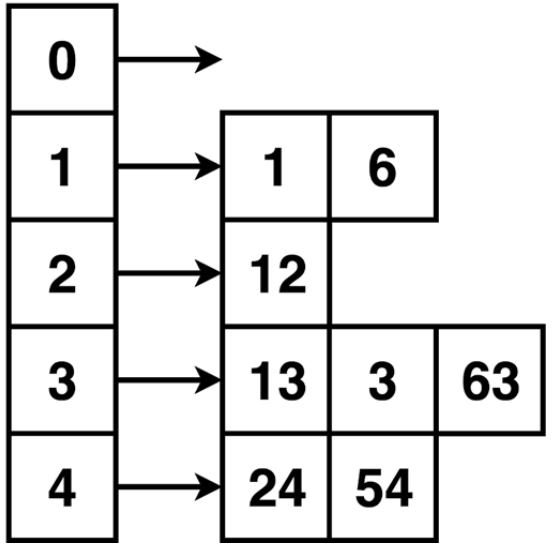
n - počet hodnôt

k - počet bucketov

Hashovanie - ukážka

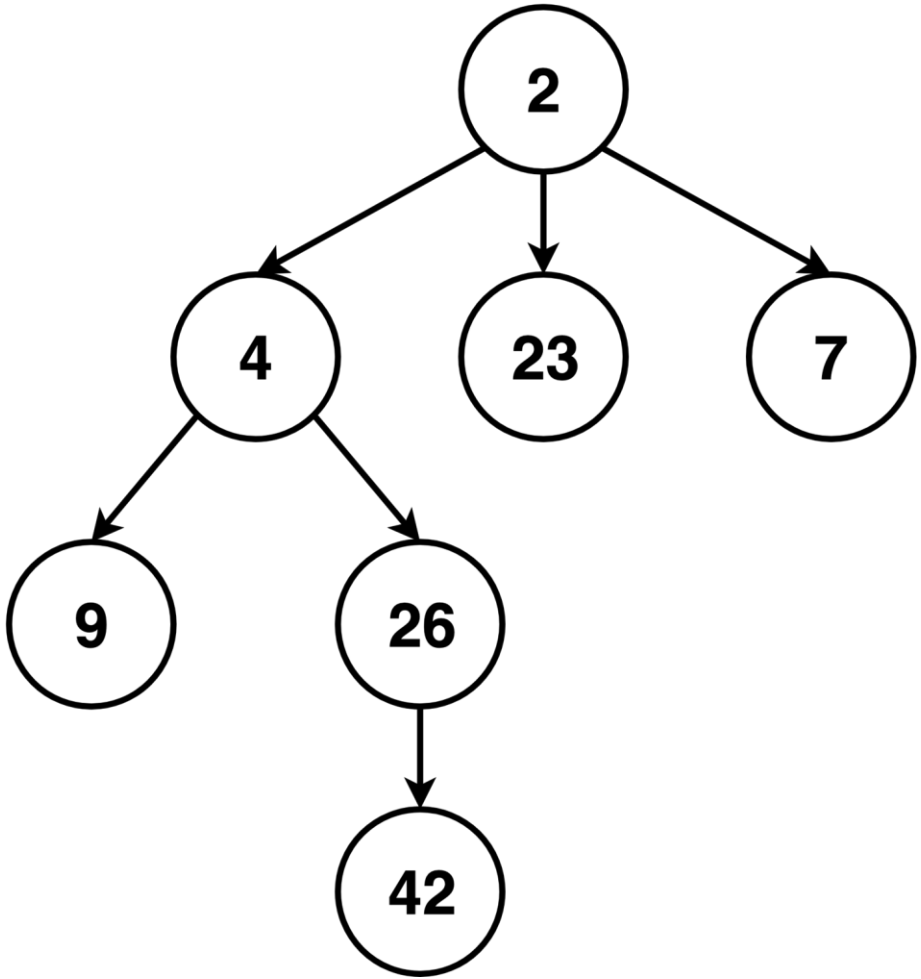
- hašovacia funkcia - zvyšok po delení 5

1	13	24	3	6	12	54	63



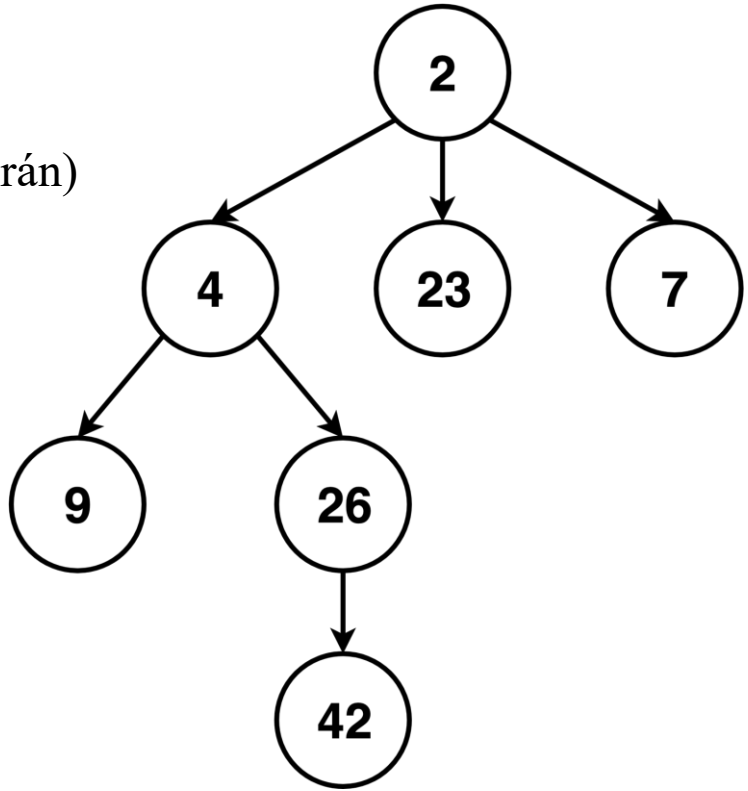
Stromy

- všeobecná údajová štruktúra, ktorá usporiada prvky do hierarchie
- strom sa skladá z uzlov (node), ktoré môžu mať potomkov (children)
- každý uzol obsahuje hodnotu a odkaz na potomkov
- uzol bez predkov (parents) je koreňový uzol (root node)
- uzol bez potomkov je listový uzol alebo list (leaf node)
- každý potomok je ďalší strom - rekurzívna definícia


2
4
23
7
9
26
42

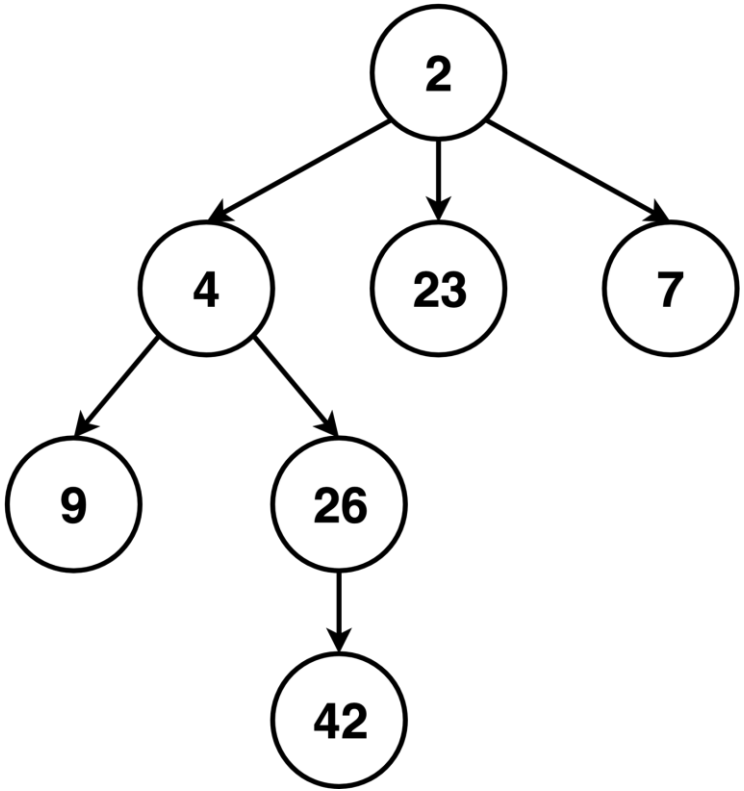
Úrovne stromu

- koreňový uzol je na nultej úrovni
- hĺbka - vzdialenosť od koreňového uzla (počet hrán)
- uzly v rovnakej hĺbke sú na jednej úrovni



Výška stromu

- udávaná uzlom s najväčšou hĺbkou
- dĺžka najdlhšej cesty od koreňového uzlu po uzol s najväčšou hĺbkou

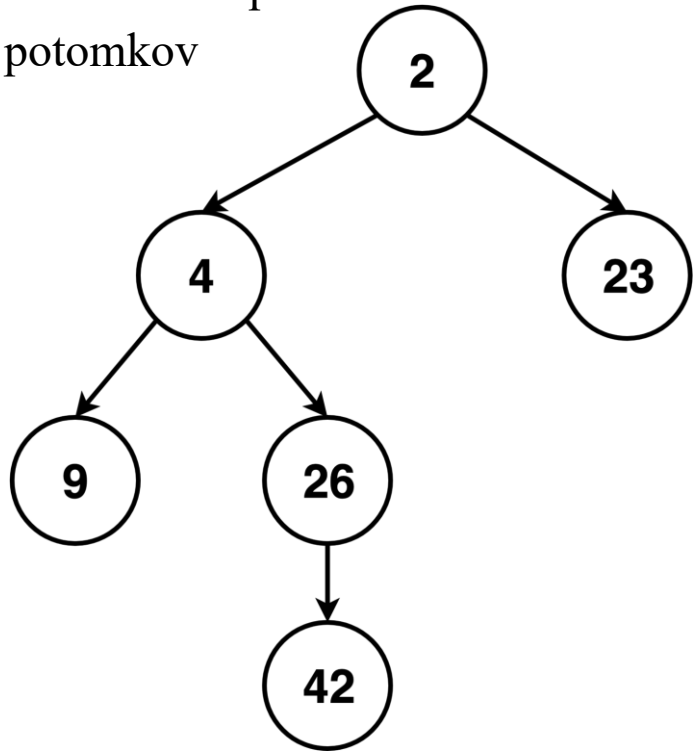


Binárny strom

- stromová štruktúra, v ktorej každý uzol má maximálne dvoch potomkov
- striktný binárny strom - každý uzol má 0 alebo 2 potomkov
- maximálny počet uzlov na úrovni *i*:

  2^i
- maximálny počet uzlov v strome s výskou *h*:

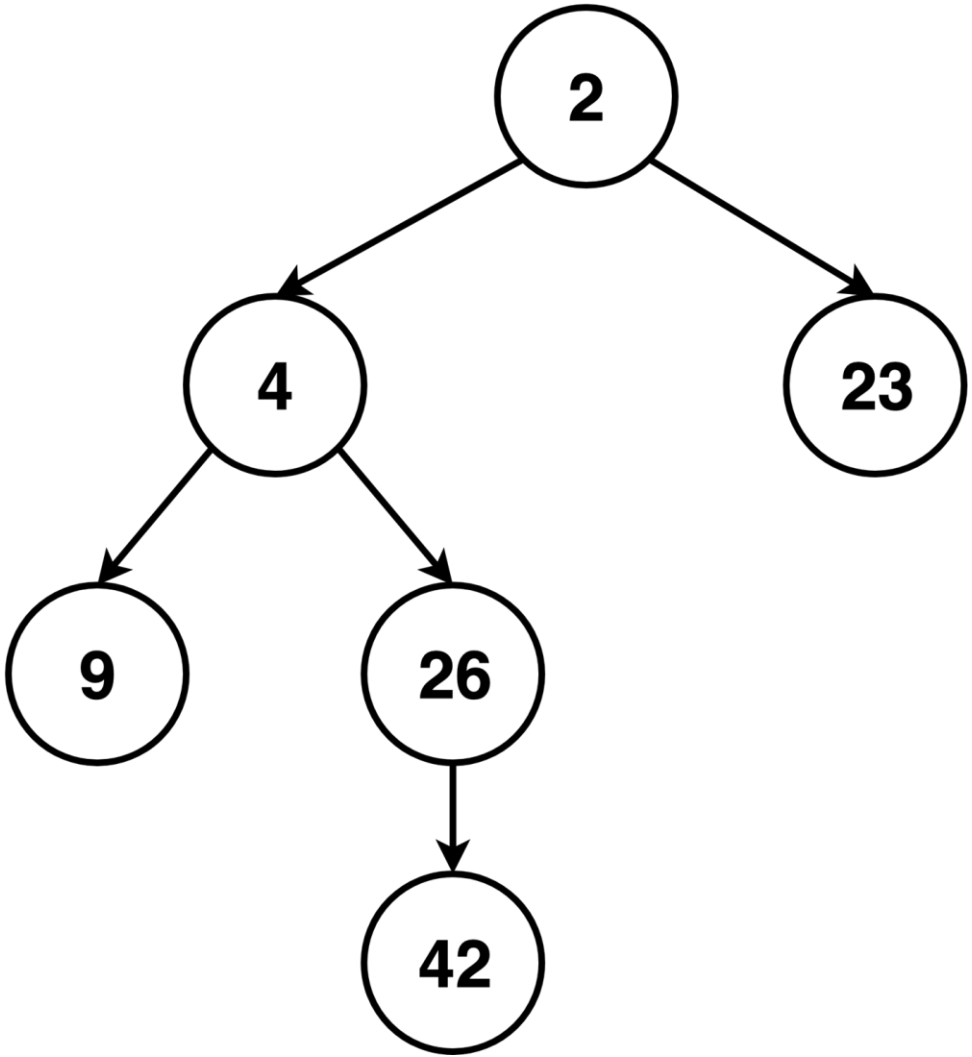
  $2^{h+1} - 1$

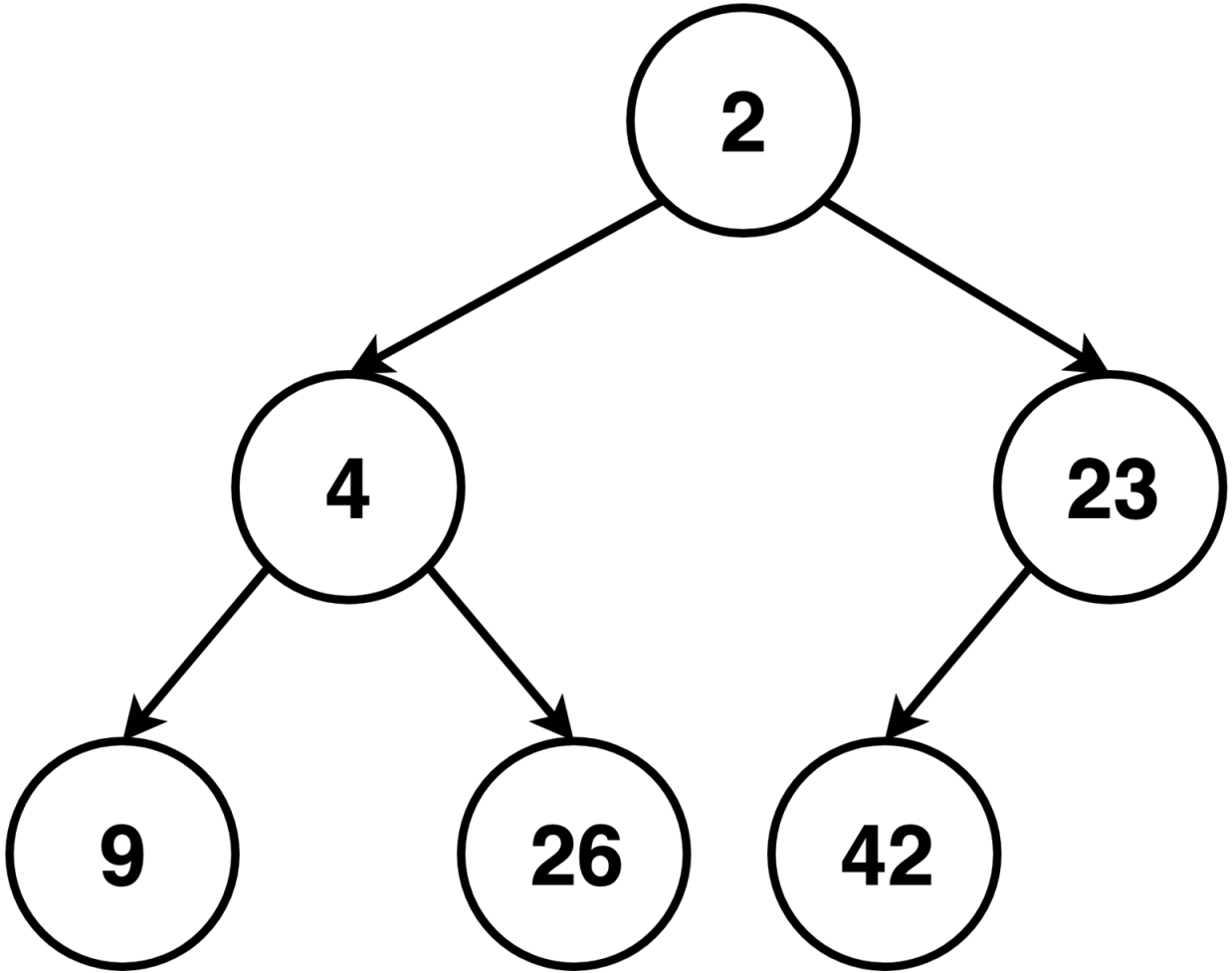


Vyvážený binárny strom

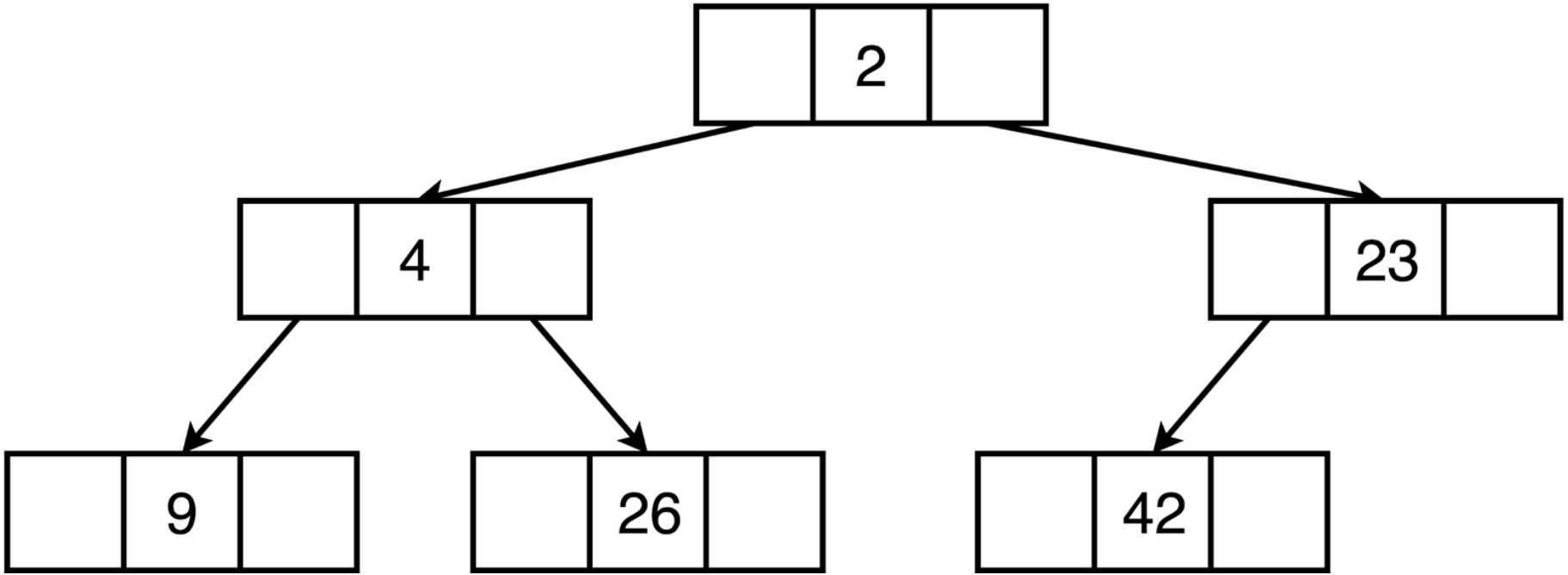
- binárny strom, v ktorom každá úroveň je plne obsadená až na poslednú, v ktorej uzly sú čo najviac vľavo
- rozdiel medzi výškou dvoch potomkov je maximálne 1
- ak aj posledná úroveň je plne obsadená, nazýva sa to aj perfektný binárny strom
- výška stromu s *n* uzlami:

$$h = \lfloor \log_2(n + 1) - 1 \rfloor$$

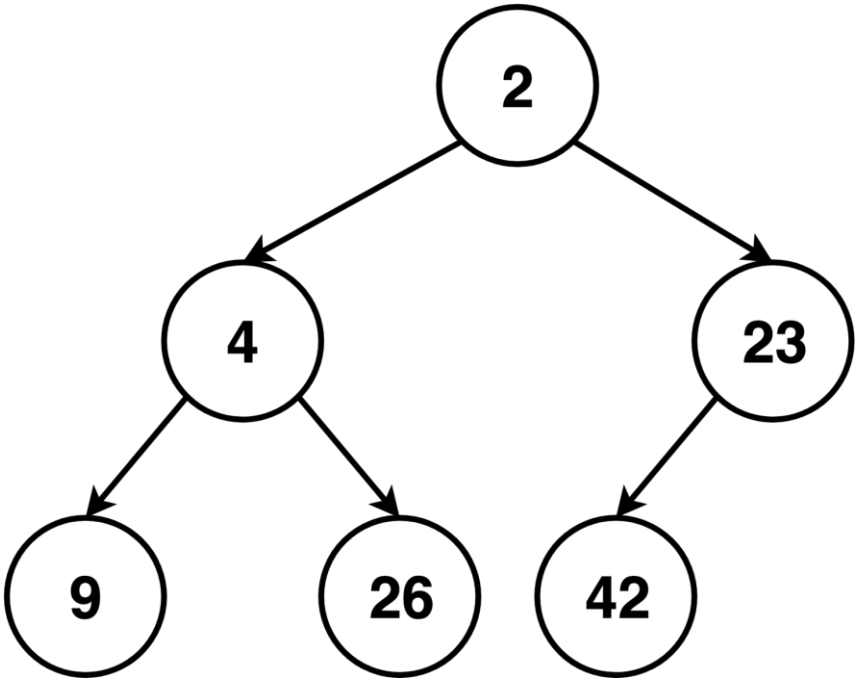

2
4
23
9
26
42


2
4
23
9
26
42

Reprezentácia stromov pomocou dynamických uzlov



Reprezentácia stromov pomocou zoznamu

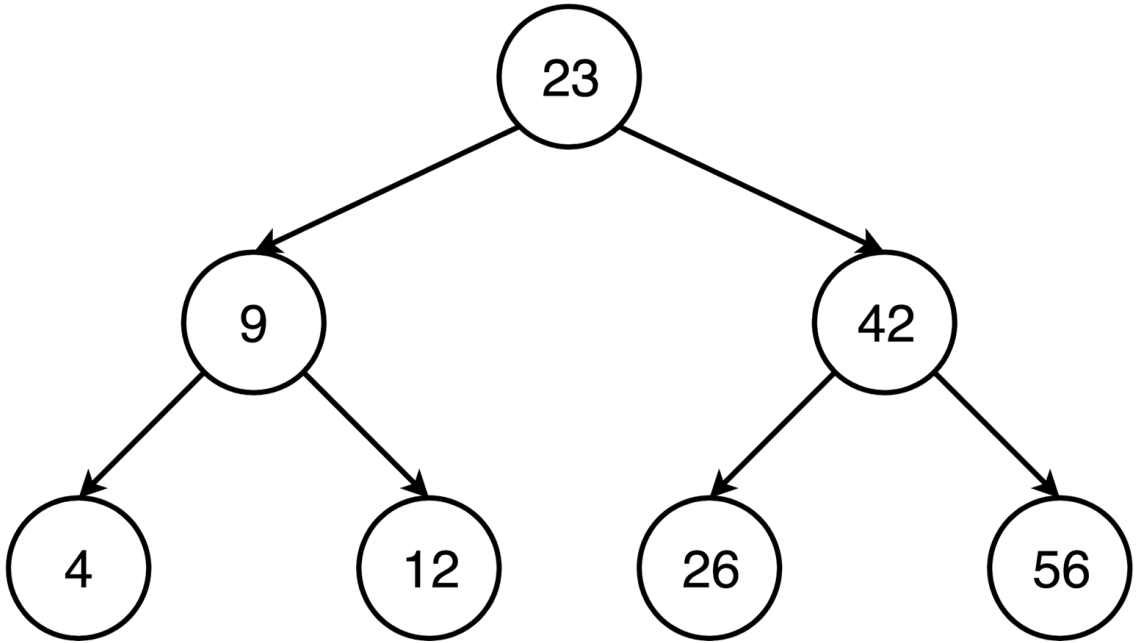


- pre uzol s indexom *i*:
  - ľavý potomok má index *2i+1*
  - pravý potomok má index *2i+2*

2	4	23	9	26	42

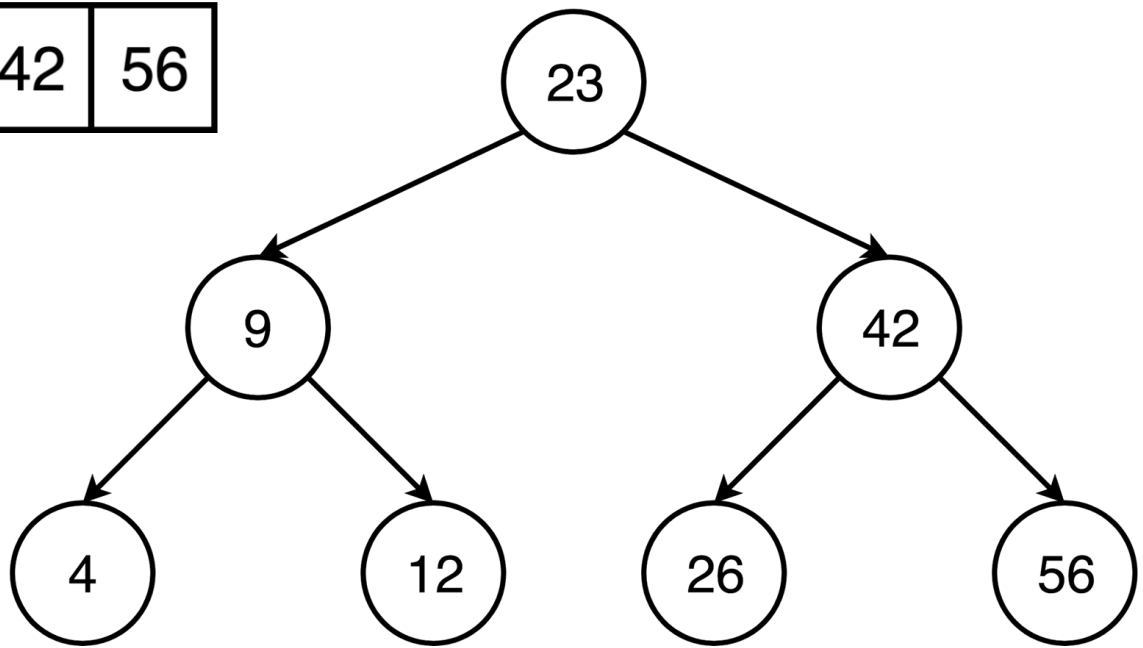
Binárny vyhľadávací strom

- binárny strom, v ktorom pre každý uzol platí, že hodnoty v ľavom potomkovi sú menšie ako hodnota uzla, a hodnoty v pravom potomkovi sú vyššie ako hodnota uzla



Vyhľadávanie - zoznam vs. strom

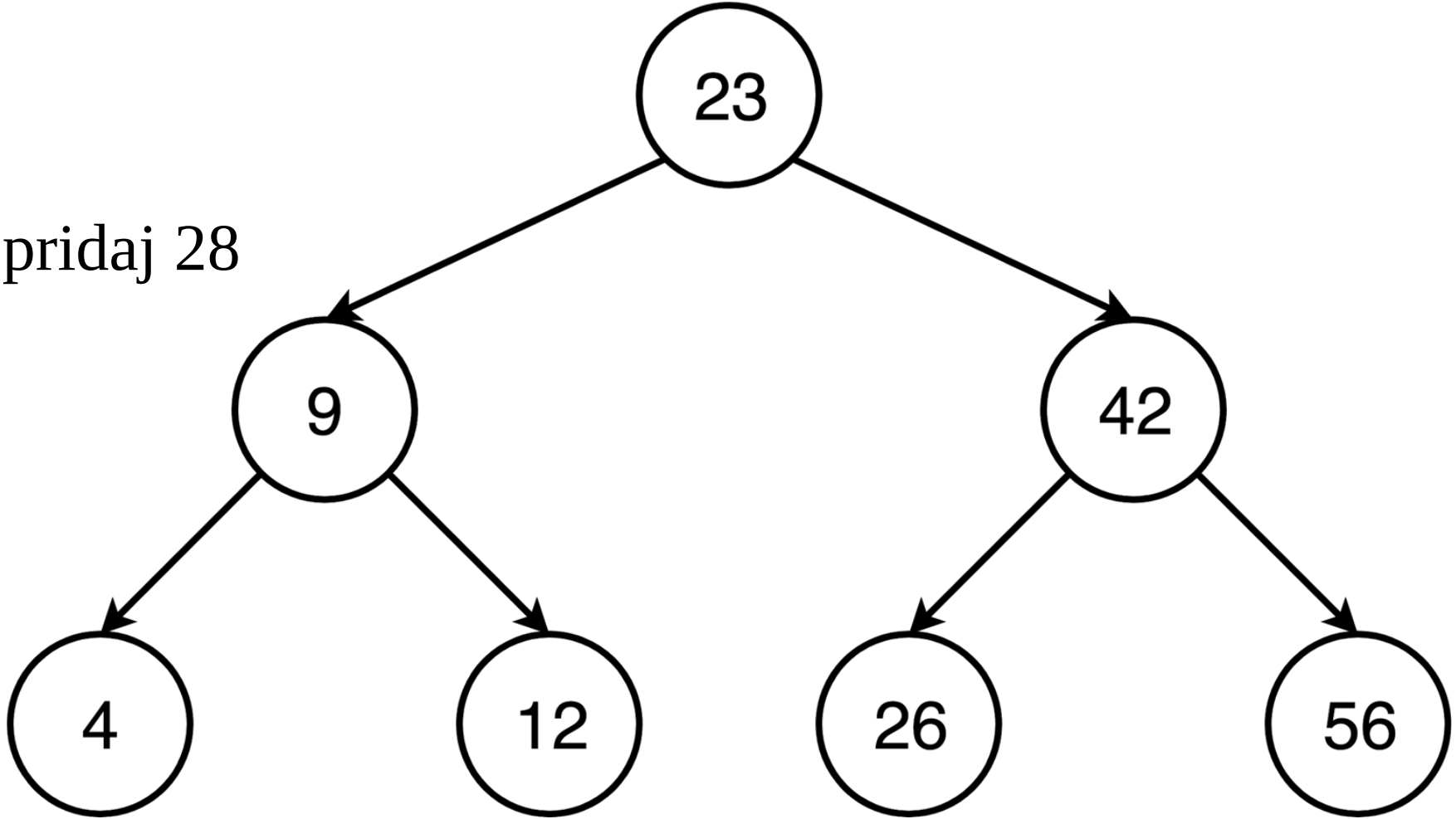
2	4	9	23	26	42	56



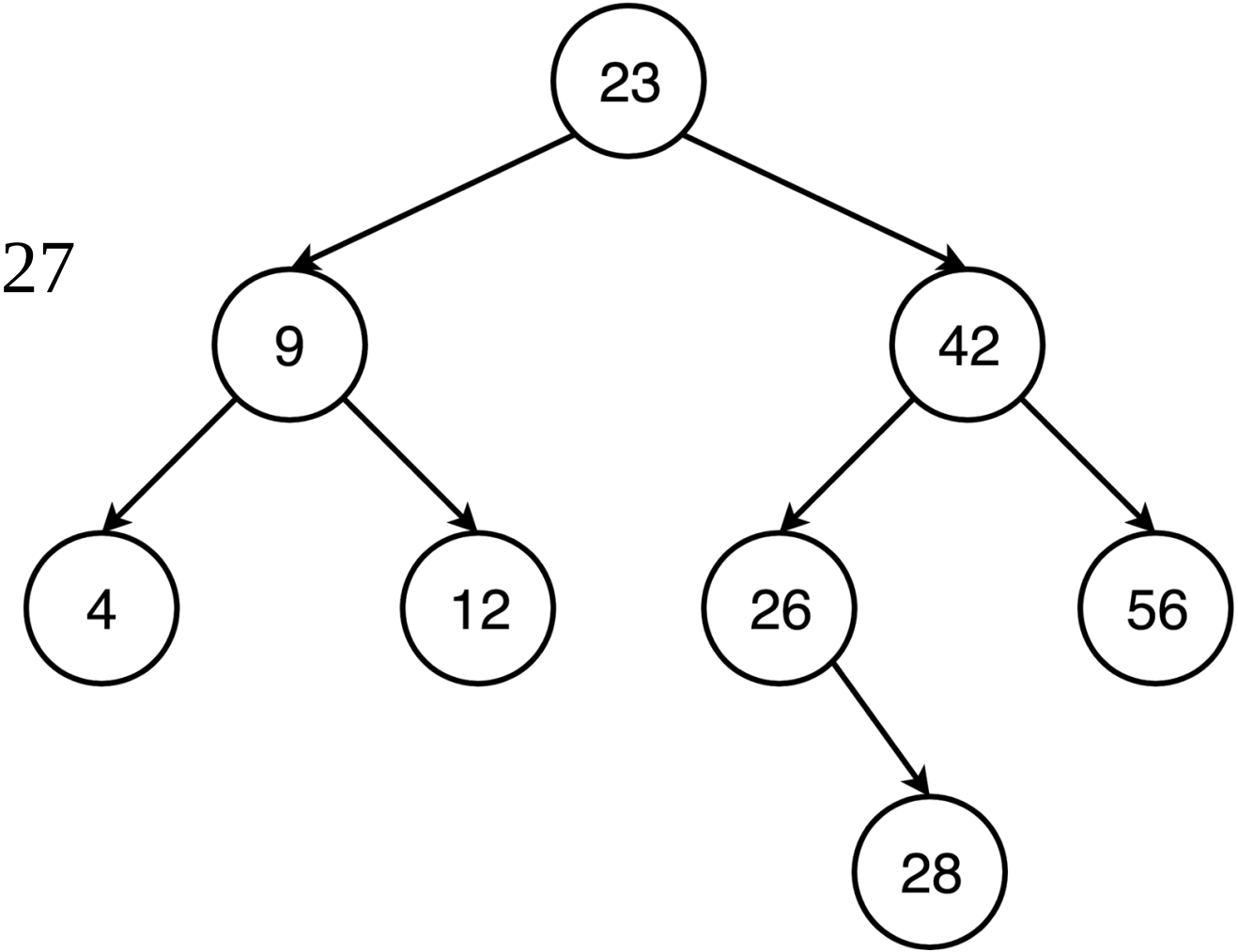
Pridávanie prvkov do stromu

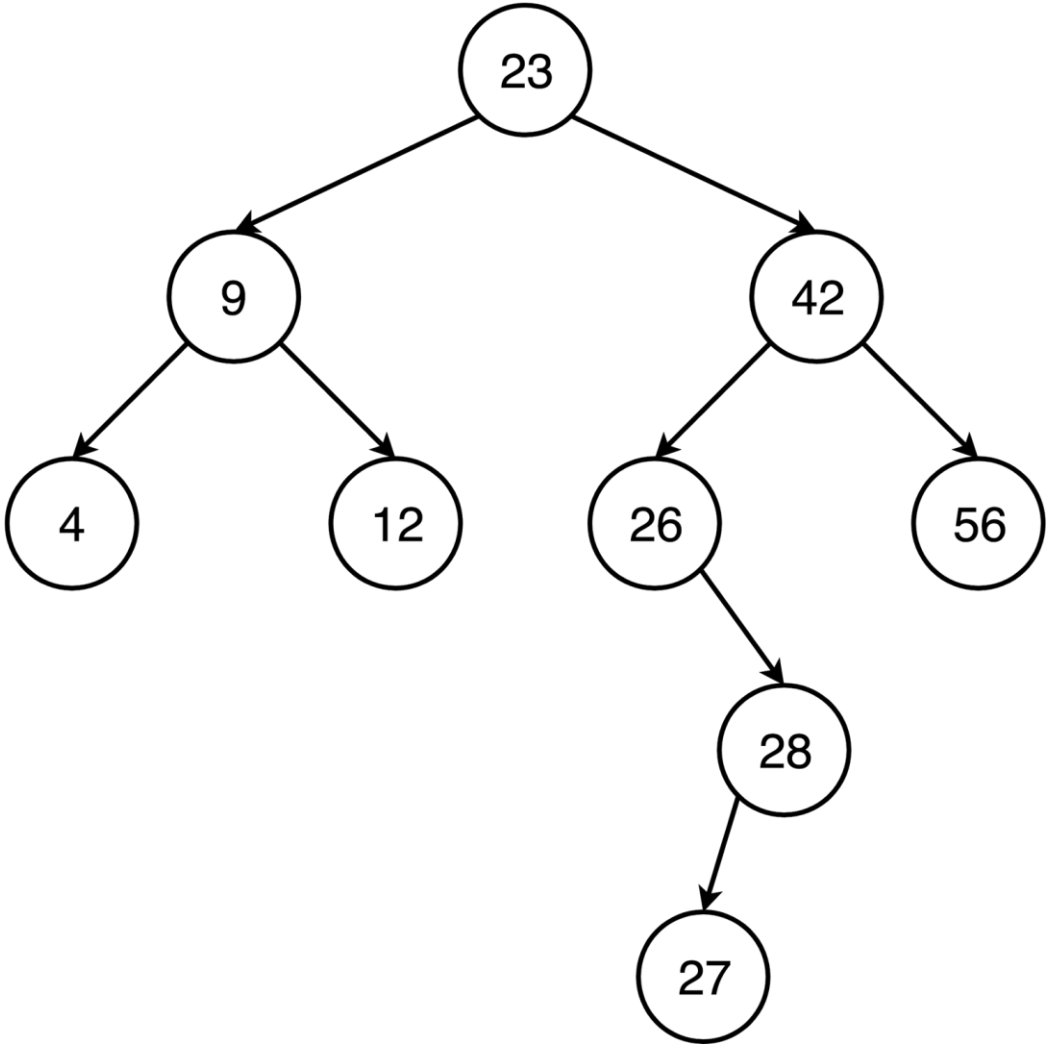
1. nájsť vhodné miesto
2. pridať uzol
3. upraviť strom ak sa stane nevyváženým

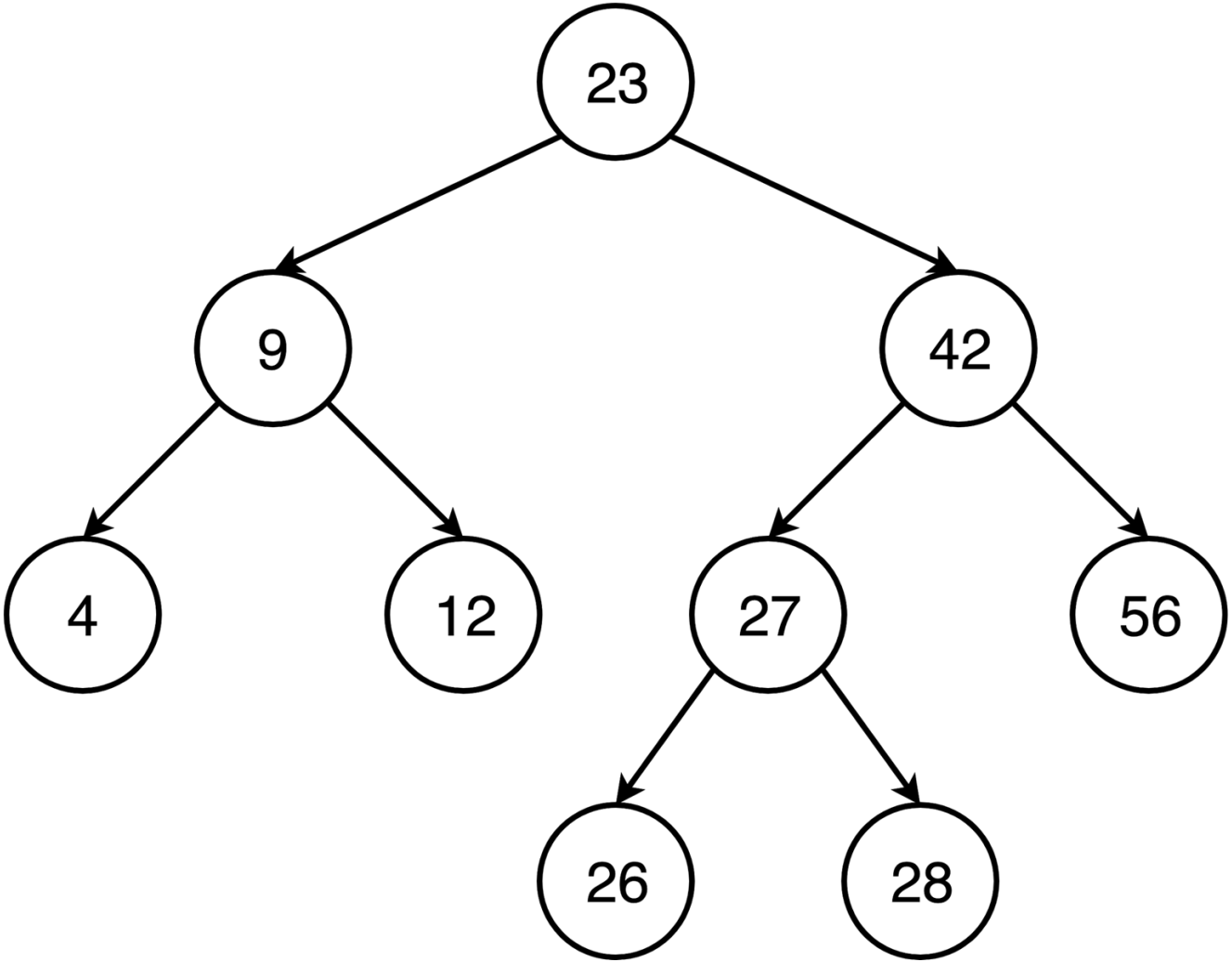
pridaj 28



pridaj 27




23
9
42
4
12
26
56
28
27


23
9
42
4
12
27
56
26
28

Porovnanie náročnosti operácií - zoradené hodnoty

Operácia	**Pole**	**Spojkový zoznam**	**BVS**
Vyhľadávanie	*O(log n)*	*O(n)*	*O(log n)*
Pridávanie	*O(n)*	*O(1)**	*O(log n)**
Zmazanie	*O(n)*	*O(n)*	*O(log n)*

Zhrnutie

- definícia údajových štruktúr
- polia v Pythone – základné funkcie knižnice numpy
- spojkový zoznam
- zoznamy
- zásobník
- front
- stromy
- zložitosť základných operácií v jednotlivých abstraktných údajových štruktúrach