



# **Programovanie v jazyku Python**

Údajové štruktúry, numpy a pandas  
prednáška 4

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

# Údajová štruktúra

- všeobecný popis
- implementovaná cez údajové typy
- zvyčajne sa skladá zo skupiny elementárnych hodnôt
- cieľom je zvýšiť efektivitu práce s údajmi
- štandardizované šablóny pre reprezentáciu údajov v počítačoch
- rôzne úrovne abstrakcie
- vieme ich implementovať rôznymi spôsobmi, ale spôsob fungovania sa nemení

# Vybrané údajové štruktúry

- polia
- zoznamy
- zásobníky
- fronty
- hašovacie tabuľky
- stromy

# 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 nemenniteľné (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

- celkový počet prvkov: `my_array.size`
- 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 postupujeme 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
  - -1 na automatické výpočty rozmeru
- `numpy.flatten(order='C')`
  - vracia nové pole - vektorová reprezentácia (jeden riadok, resp. stĺpec)



# zeros(), ones() a full()

- metódy slúžia na inicializáciu matice s hodnotami 0, 1 alebo vlastnou hodnotou
- `numpy.zeros(shape, dtype=float, order='C')`
- `numpy.ones(shape, dtype=float, order='C')`
- `numpy.full(shape, fill_value, dtype=None, order='C')`
- `numpy.empty(shape, dtype=float, order='C', *, like=None)`

# Generovanie rozsahov

- `numpy.arange(start, stop, step)`
  - funguje rovnako ako `range()`
  - možnosť práce s floatmi – skôr použiť `linspace`
- `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`

# Pridávanie prvkov

- implementuje konkaténáciu 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])
```

# Indexovanie pomocou polí

- ako index vieme použiť aj druhé pole – načítajú sa hodnoty pod indexom podľa druhého poľa
- druhé pole môže mať rôzne tvary – určí tvar výsledku
- vieme použiť aj viac polí, kde hodnoty udávajú index po jednotlivé dimenzie

# Podmienené indexovanie

- výsledky podmienok vieme použiť aj na podmienené indexovanie, teda na generovanie masiek

```
a = np.arange(12).reshape(3, 4)
```

```
b = a > 4
```

```
a[b]
```

vyberie hodnoty väčšie ako 4

```
a[b] = 0
```

vynulujeme dané čísla

# Aritmetické operácie nad maticami

- primitívne operácie fungujú element-wise

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

```
test += 1
```

```
test [2, 3, 4]
```

- ak je potrebné pretypovanie, tak sa to robí na všeobecnejší typ

# Štatistické metódy

- `np.min(array)` / `np.argmin(array)`
- `np.max(array)` / `np.argmax(array)`
- `np.mean(array)`
- `np.median(array)`
- `np.std(array)`
- všetky podporujú parameter `axis`, ktorý určí, po ktorej osi sa má vypočítať štatistická metrika
- numpy ponúka aj matematické funkcie



# 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 matic
- násobenie matic je možné aj cez operátor `@` alebo `array1.dot(array2)`

# Kópie a viewy

- pomocou metódy `view()` vieme vytvoriť pohľad na pôvodné pole

```
c = a.view()
c is a           False
c.base is a      True
```

- používajte metódy `copy()` pre hlboké kópie

```
d = a.copy()
```

# Pandas

- knižnica pre dátovú analytiku
- nadstavba nad numpy
- pre tabuľkové údaje a časové rady
- umožňuje všeobecné indexovanie (cez názvy stĺpcov)
- zvyčajne sa používa importom  
`import pandas as pd`

# Základné objekty

- postupnosť prvkov – Series

```
s = pd.Series([1, 3, 5, np.nan, 6, 8])
```

- údajový rámec – DataFrame

```
my_ids = pd.date_range("20220214", periods=4)
```

```
df = pd.DataFrame(  
    np.random.randn(4, 4),  
    index=my_ids,  
    columns=['A', 'B', 'C', ,D'])
```

# Generovanie údajových rámcov

- z dictionary

```
df2 = pd.DataFrame({  
    "A": 1.0,  
    "B": "2022-02-14",  
    "C": ["test", "train", "eval", "train"]  
})
```

- načítanie zo súboru, napr.

```
df3 = pd.read_csv(path, sep=',')
```

# Indexovanie v dataframeoch

- funguje všeobecné Pythonovské indexovanie
- ďalšie pomocné možnosti prístupu: `at`, `iat`, `loc`, `iloc`
- výber stĺpca (vráti postupnosť `Series`)  
`df["A"]`
- výber riadkov  
`df[0:3]`  
podľa indexov  
`df["20220214": "20220216"]`
- výber riadkov a stĺpcov  
`df.loc["20220214": "20220216", ["A", "B"]]`
- `iat` a `iloc` fungujú podobne ale iba s číselnými indexmi

# Podmienený výber

- pomocou podmienok (podmieňovacie operátory)

```
df[df["A"] > 0]
```

- pre filtrovanie môžeme použiť metódu `isin()`

```
df[df["C"].isin(["test", "train"])]
```

# Nastavenie hodnôt

- rozšírenie o stĺpec:  
`df["F"] = "new"`
- nastavenie hodnoty podľa (číselného) indexu  
`df.at[0, "A"] = 0 / df.iat[0, "A"] = 0`
- nastavenie niekoľkých hodnôt pomocou polí  
`df.loc[:, "D"] = np.array([5] * len(df))`
- podmienené nastavenie (napr. nastavenie na absolútne hodnoty)  
`df[df < 0] = -df`



# Základná práca s dataframeami

- ukážkové riadky zo začiatku/konca dataframeu  
`df.head(n=g) / df.tail(n=5)`
- získanie zoznamu indexov a stĺpcov  
`df.index / df.columns`
- konvertovanie na numpy polia (iba pri rovnakých údajových typoch)  
`df.to_numpy()`
- prehľad štatistických metrík  
`df.describe()`
- transpozícia  
`df.T`

# Triedenie údajov v dataframeoch

- triedenie indexov

```
df.sort_index(axis=1, ascending=False)
```

- triedenie podľa hodnôt

```
df.sort_values(by="B")
```

# Spracovanie chýbajúcich hodnôt

- vymazanie riadkov s chýbajúcimi hodnotami  
`df.dropna(how="any")`
- doplnenie chýbajúcich hodnôt  
`df.fillna(value=0)`
- získanie prehľadu o prítomnosti chýbajúcich hodnôt  
`pd.isna(df)`

# Štatistické metódy

- priemer podľa stĺpcov / riadkov

```
df.mean() / df.mean(1)
```

- aplikácia ľubovoľnej funkcie

```
df.apply(np.cumsum) # suma
```

```
df.apply(lambda x: x.max() - x.min())
```

- početnosť hodnôt

```
df["C"].value_counts()
```

# Ďalšia funkcionálnosť

- konkaténácia – `concat()`
- spojenie ako v SQL – `join()`
- zgrupovanie – `groupby()`  
`df.groupby("A")` / `df.groupby(["A", "B"])`
- zhustenie / rozbalenie dimenzií  
`df.stack()` / `df.unstack()`

# 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

# Jednosměrný spojkový zoznam

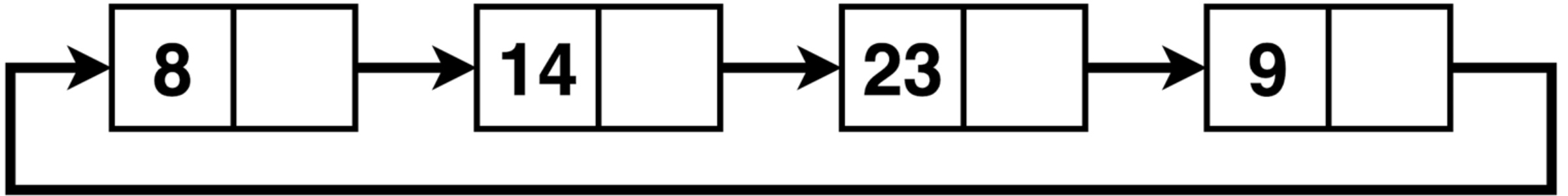


# Obojsmerný spojkový zoznam





# Kruhový zoznam

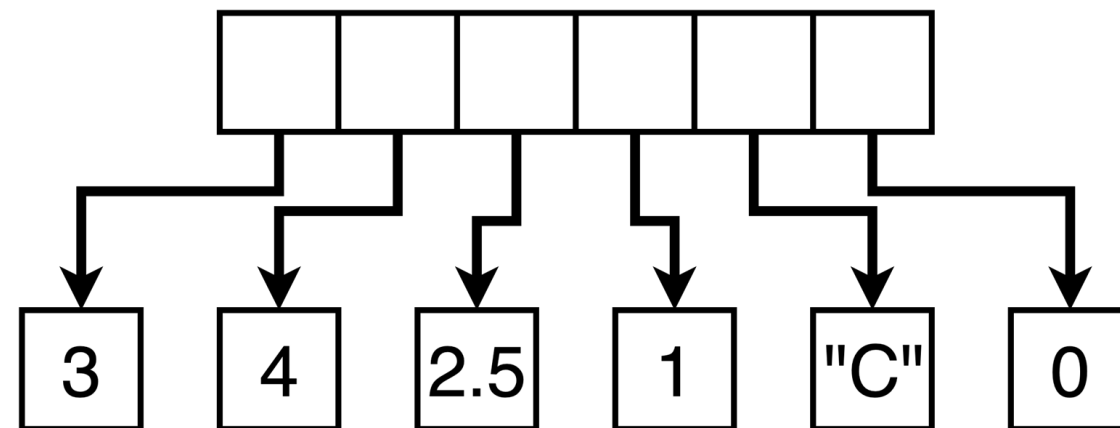
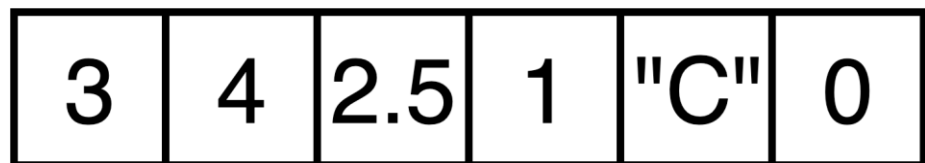


# Porovnanie polí a spojkových zoznamov

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

# Použitie zásobníkov

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



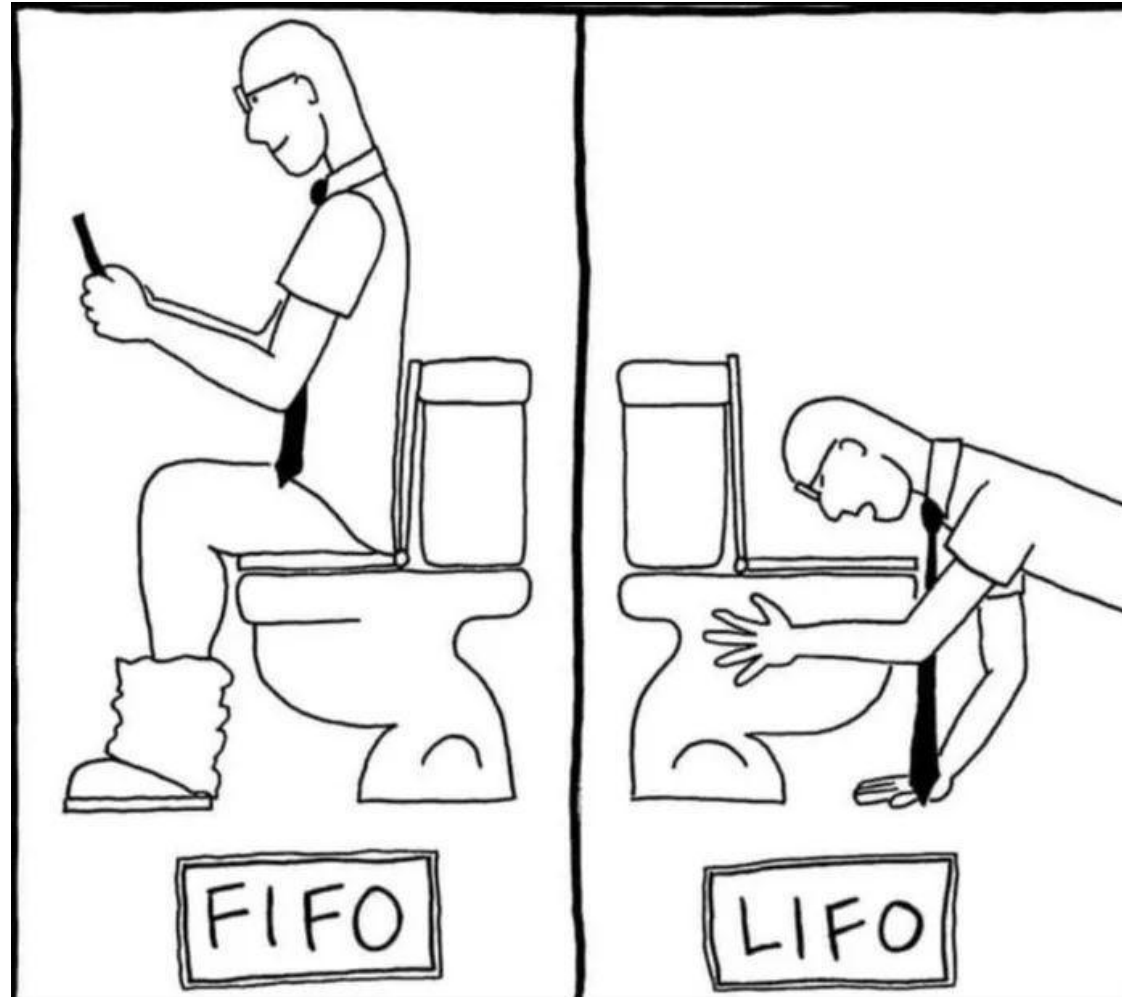
# Front (Queue)

- 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

# Použitie frontov

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

# LIFO/FIFO





# Hašovanie

- rozdeľuje údaje na menšie skupiny (buckets)
- predpokladá sa, že prístupovanie 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
  1. hašovacia funkcia vypočíta hash hodnotu
  2. prístupujeme ku skupine na základe hash hodnoty (použije sa ako kľúč)
  3. 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 číslo:  
`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:

$$\text{load factor} = n/k$$

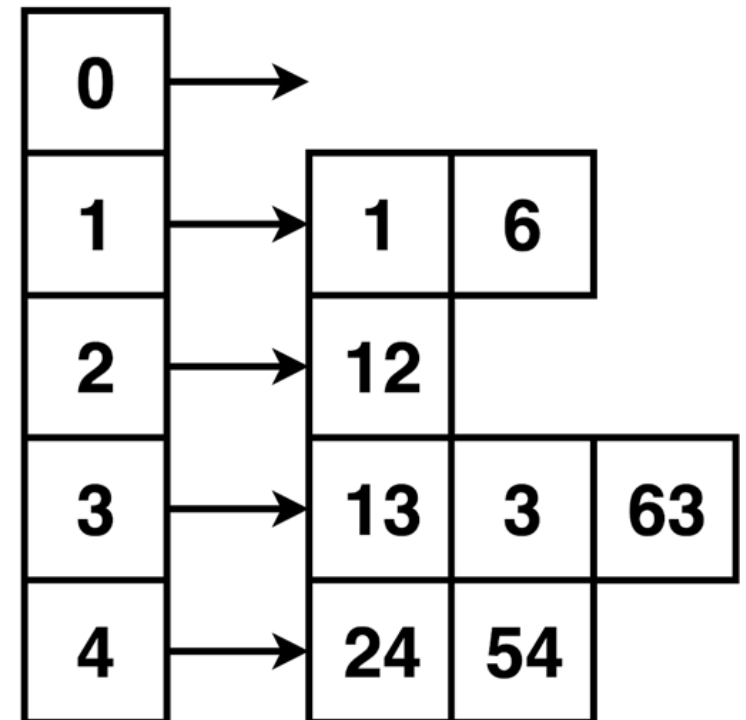
$n$  - počet hodnôt

$k$  - počet bucketov

# Hashovanie - ukážka

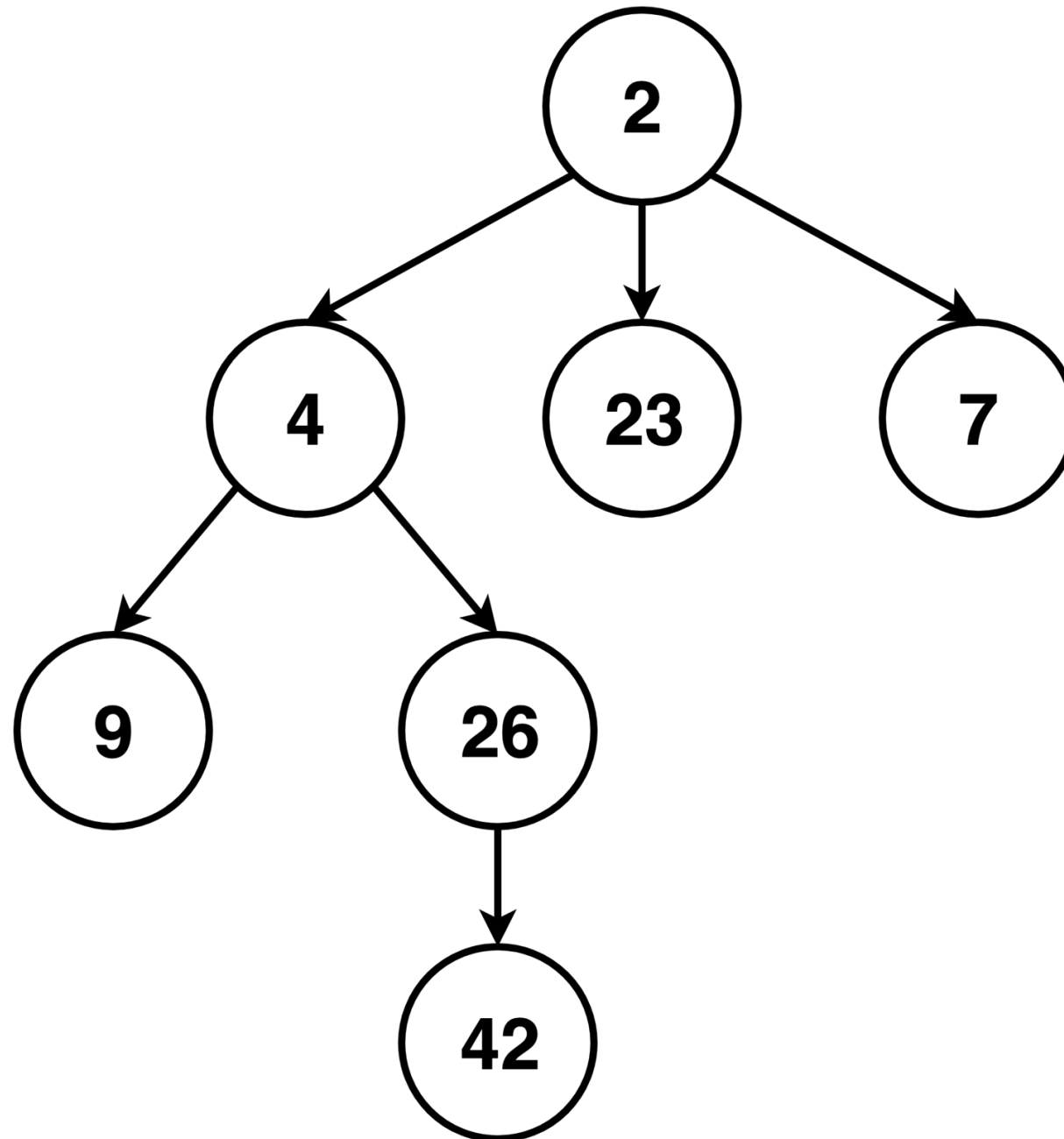
- hašovacia funkcia - zvyšok po delení 5

1	13	24	3	6	12	54	63
---	----	----	---	---	----	----	----



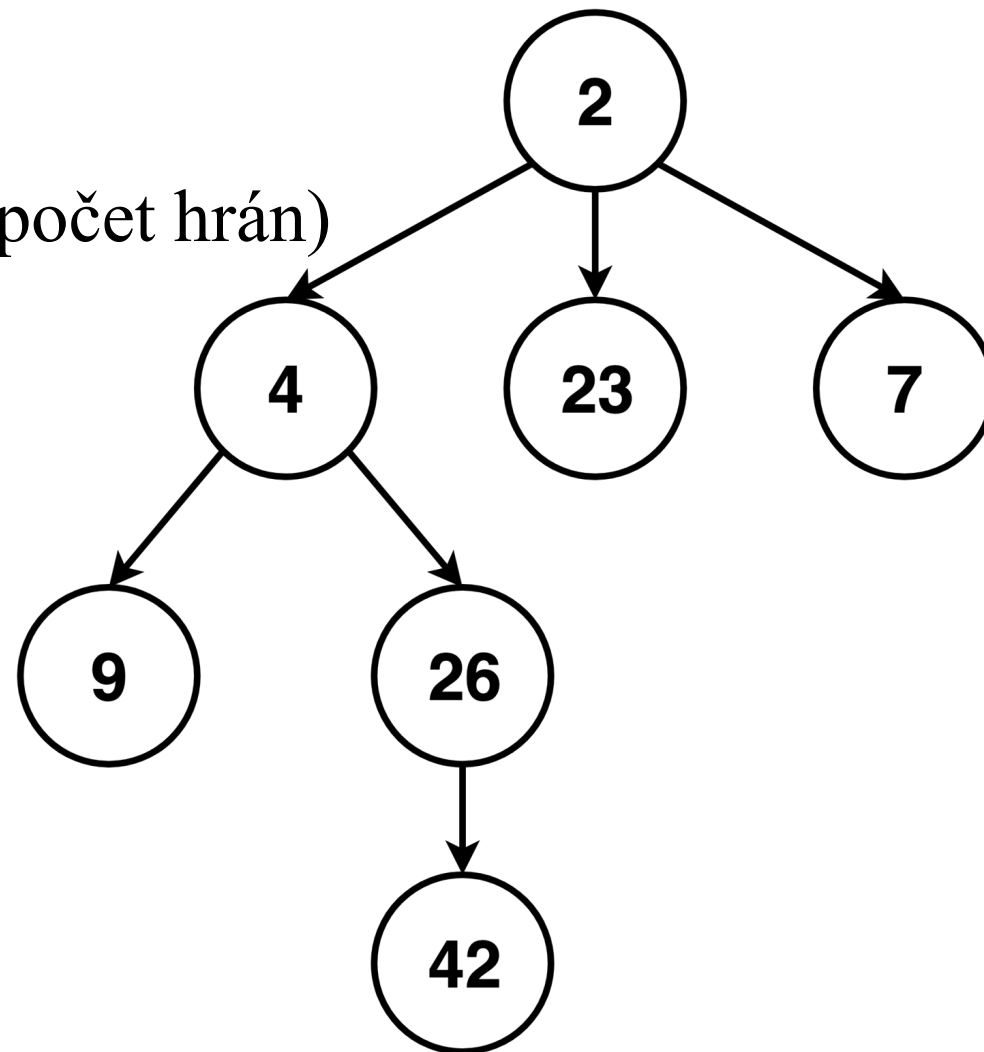
# Stromy

- 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



# Úrovne stromu

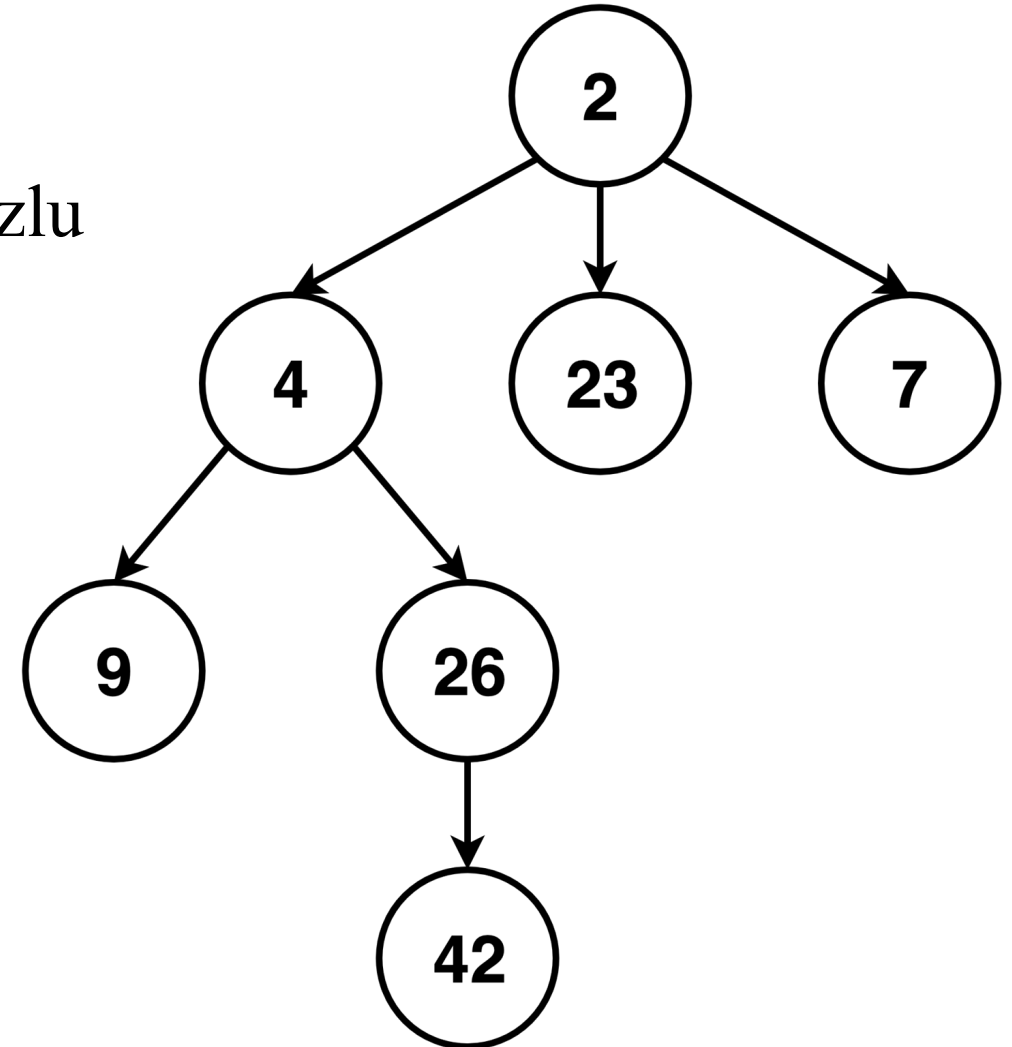
- koreňový uzol je na nulte úrovni
- hĺbka - vzdialenosť od koreňového uzla (počet hrán)
- uzly v rovnakej hĺbke sú na jednej úrovni





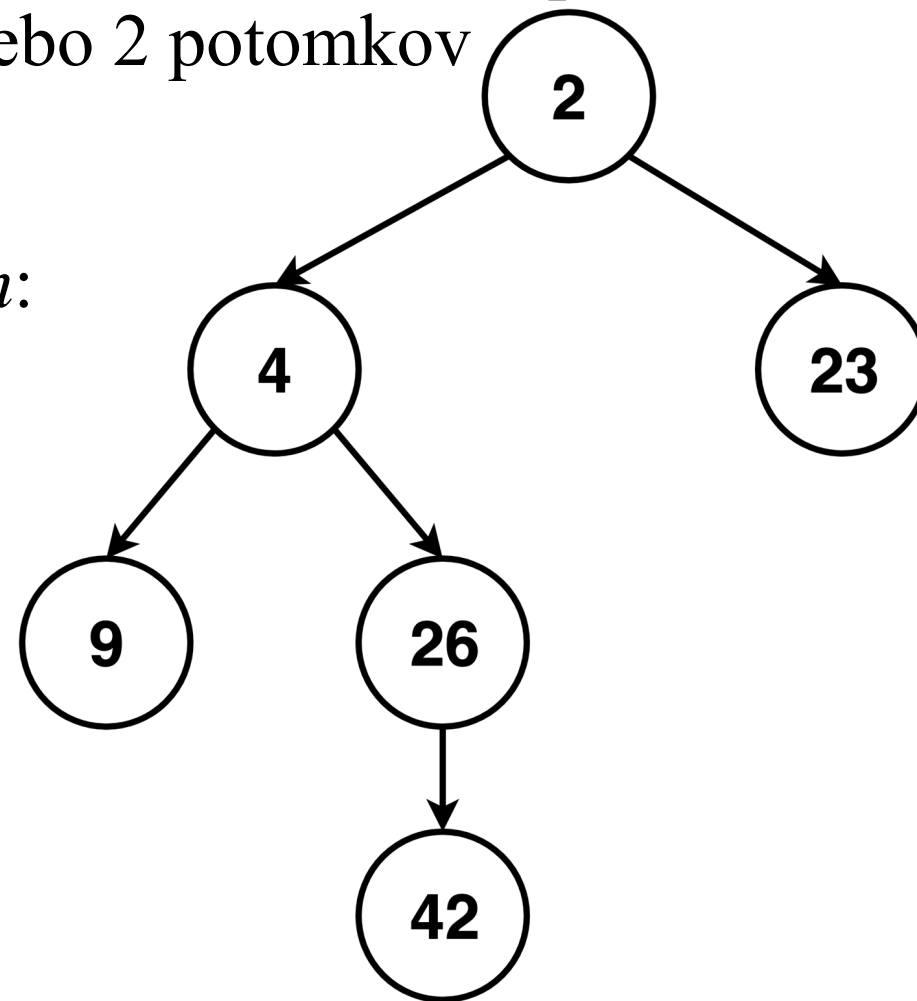
# Výška stromu

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



# Binárny strom

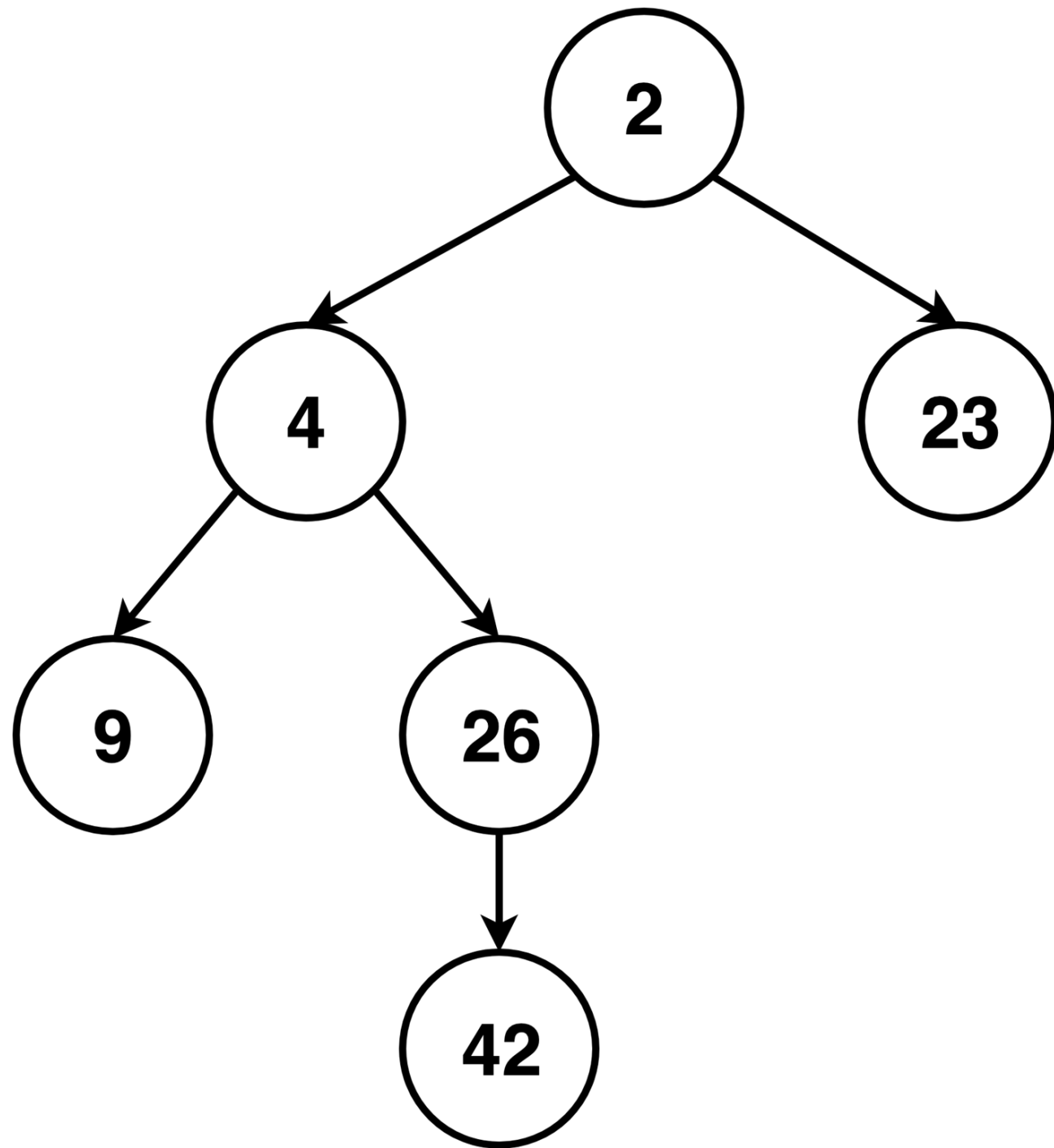
- stromová štruktúra, v ktorej každý uzol má maximálne dvoch potomkov
- striktný binárny strom - každý uzol má 0 alebo 2 potomkov
- maximálny počet uzlov na úrovni  $i$ :  
 $2^i$
- maximálny počet uzlov v strome s výškou  $h$ :  
 $2^{h+1} - 1$

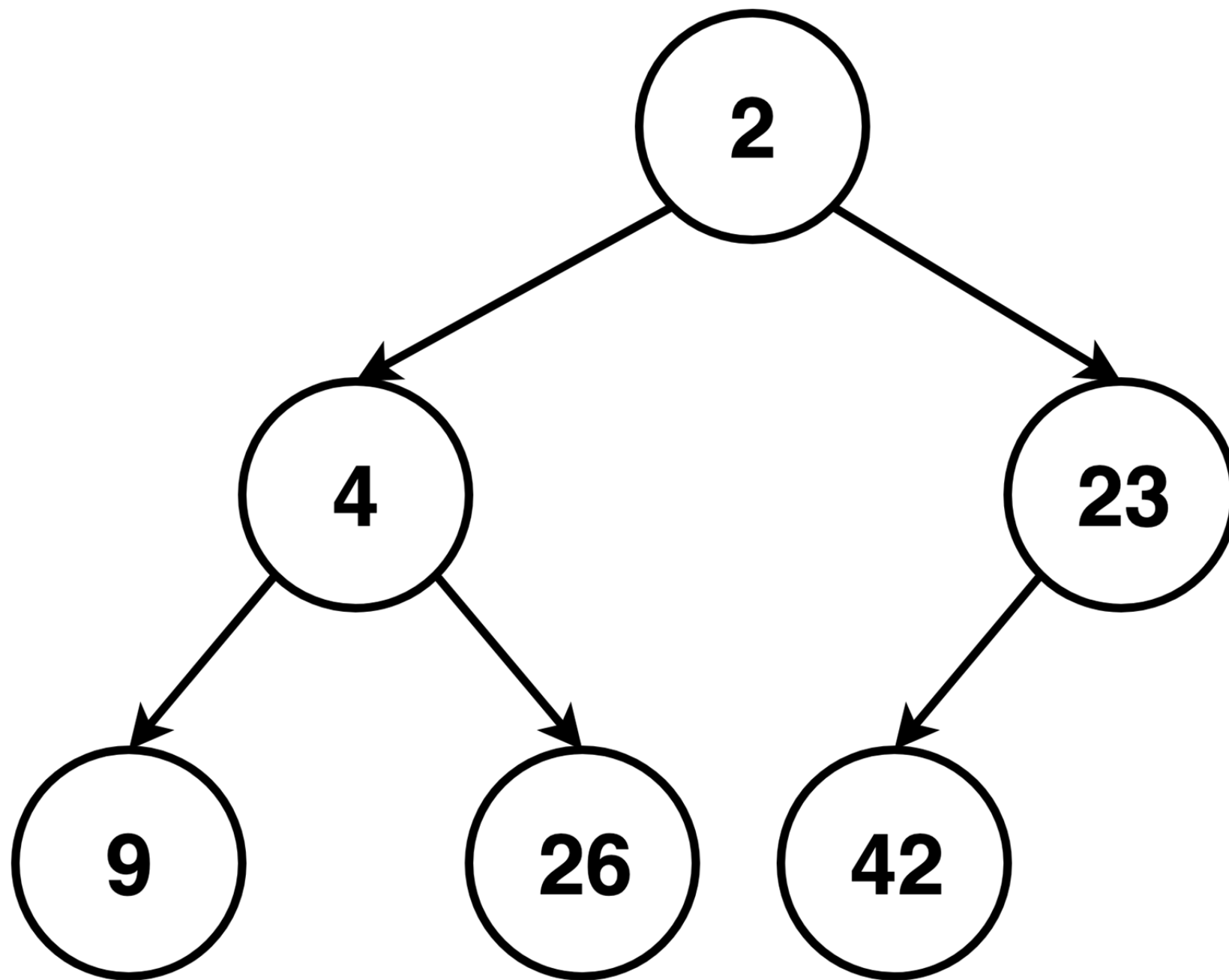


# Vyvážený binárny strom

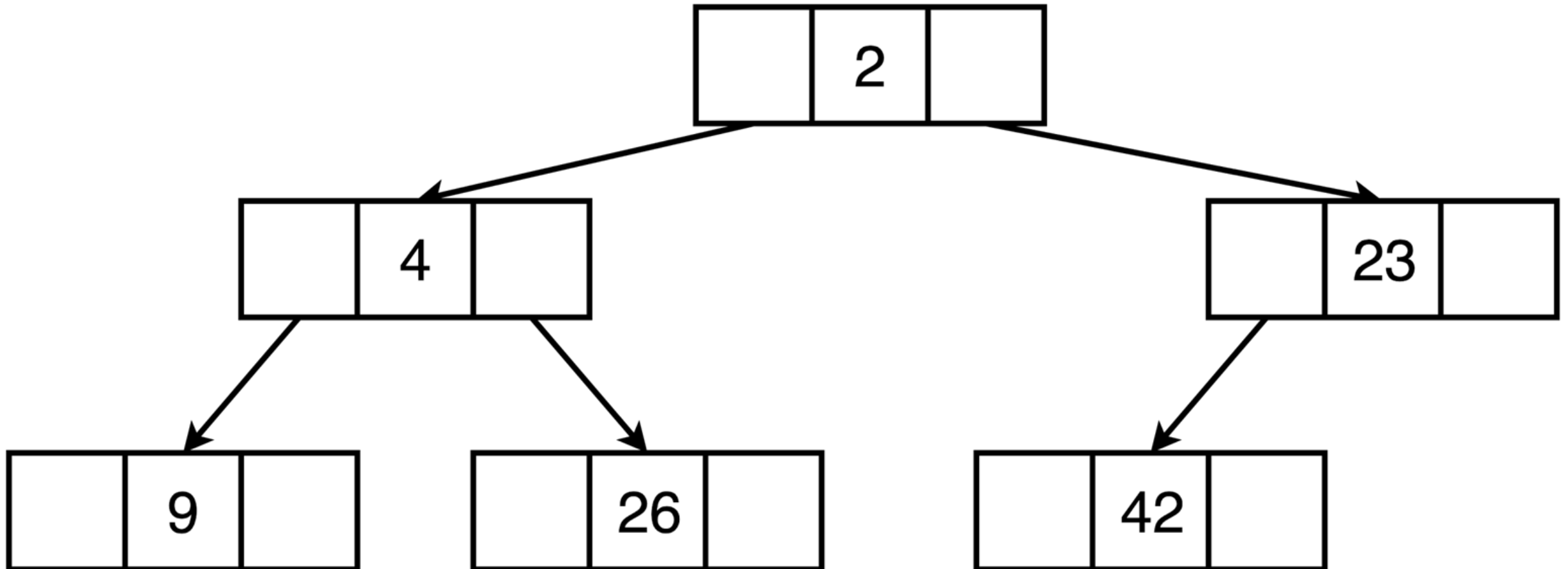
- binárny strom, v ktorom každá úroveň je plne obsadená až na poslednú, v ktorej uzly sú čo najviac vľavo
- rozdiel medzi výškou dvoch potomkov je maximálne 1
- ak aj posledná úroveň je plne obsadená, nazýva sa to aj perfektný binárny strom
- výška stromu s  $n$  uzlami:

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

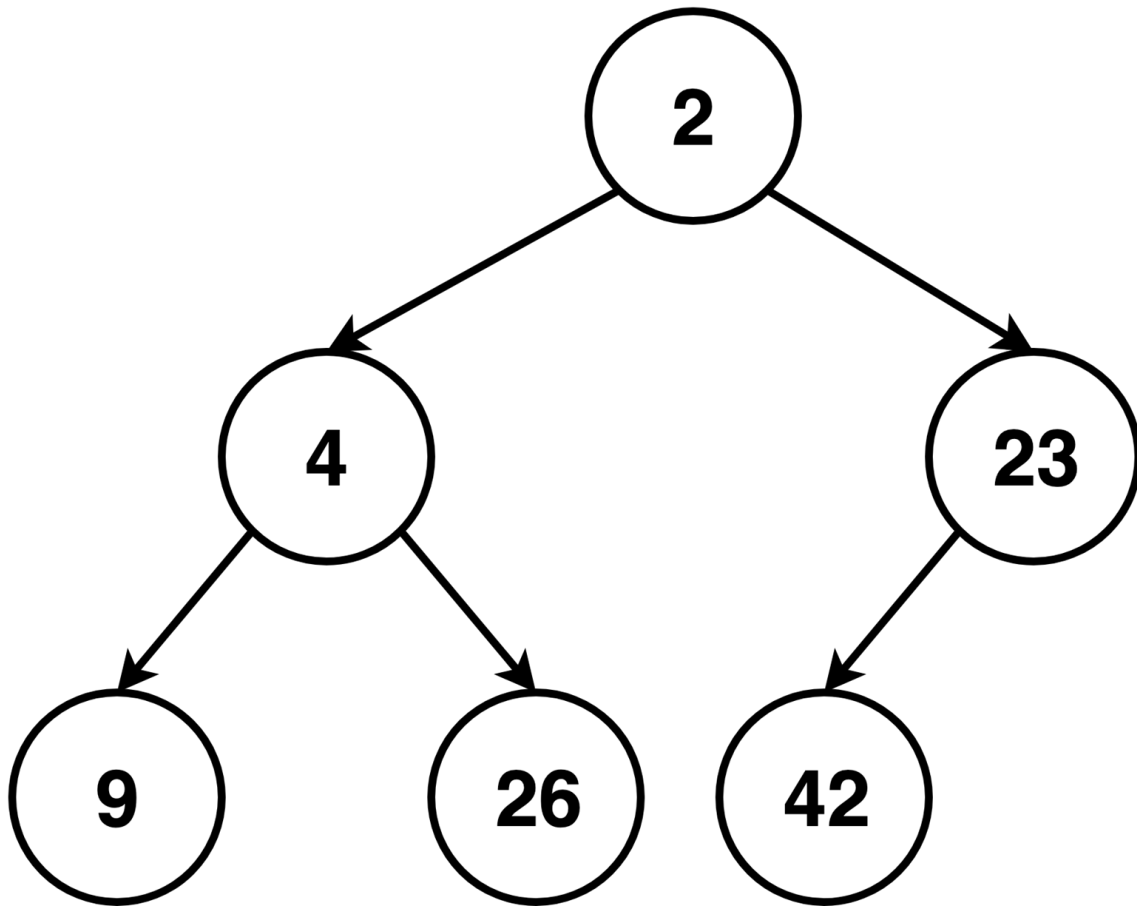




# Reprezentácia stromov pomocou dynamických uzlov



# Reprezentácia stromov pomocou zoznamu

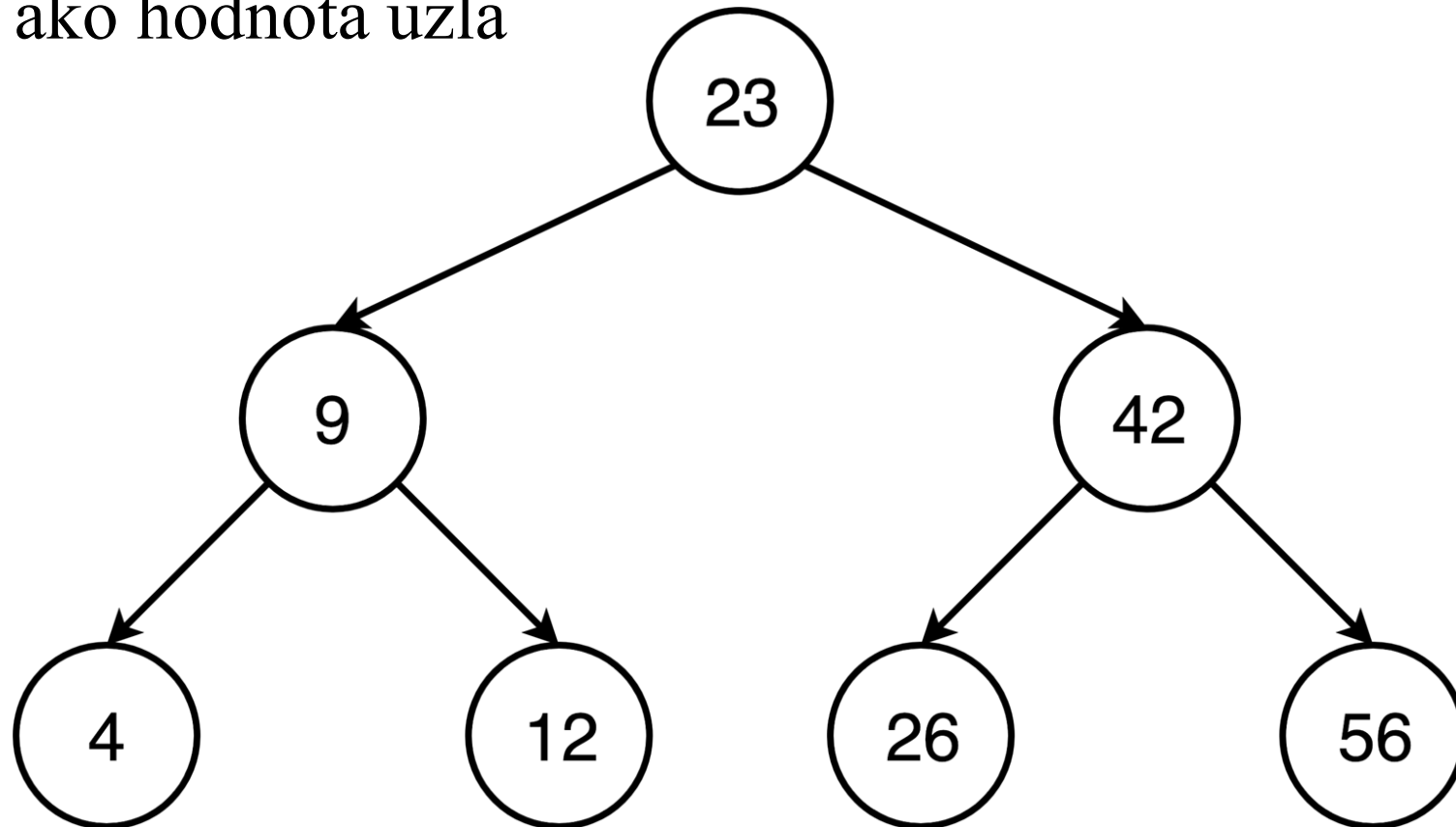


- pre uzol s indexom  $i$ :
  - ľavý potomok má index  $2i+1$
  - pravý potomok má index  $2i+2$

2	4	23	9	26	42
---	---	----	---	----	----

# Binárny vyhľadávací strom

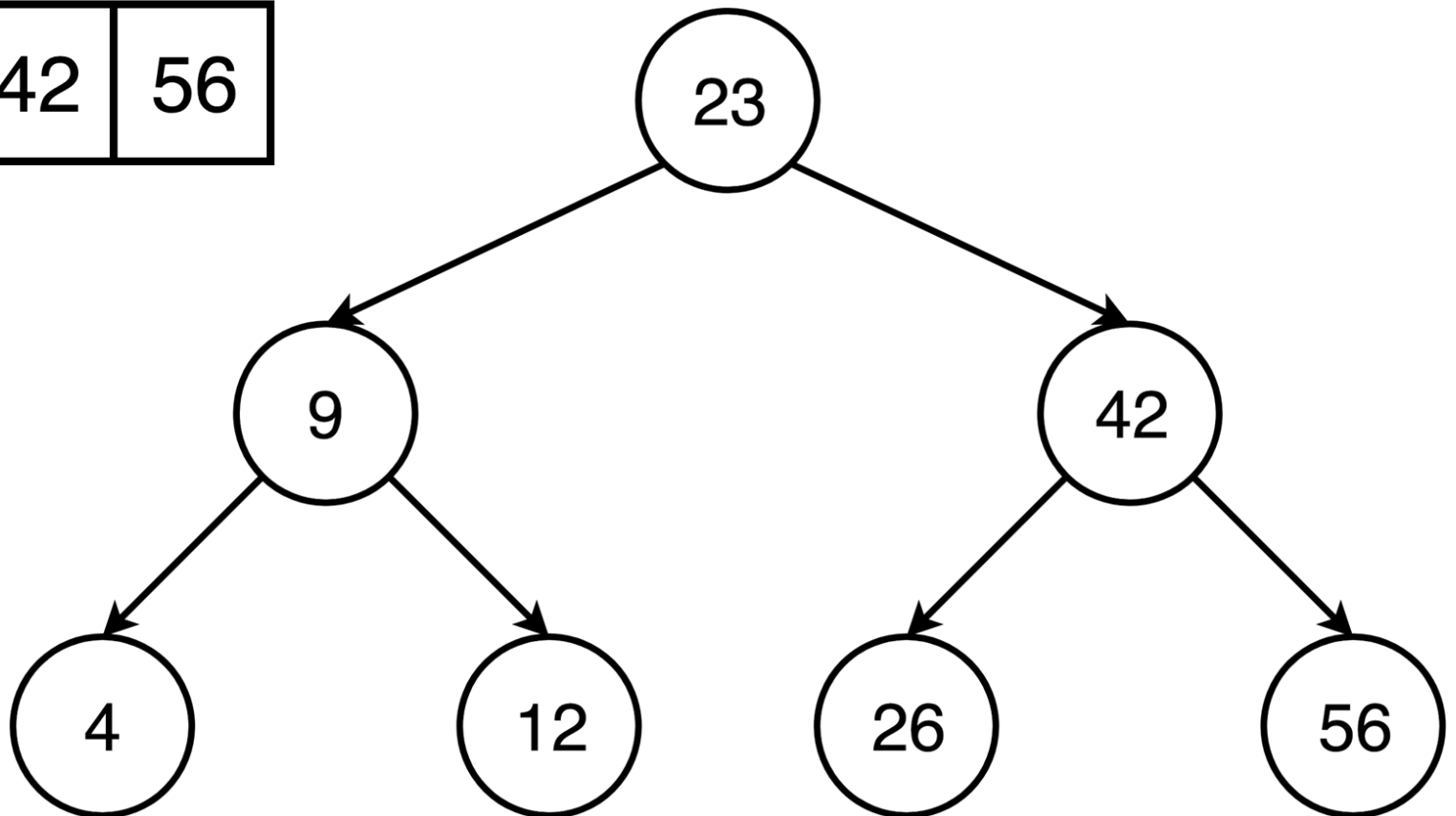
- binárny strom, v ktorom pre každý uzol platí, že hodnoty v ľavom potomkovi sú menšie ako hodnota uzla, a hodnoty v pravom potomkovi sú vyššie ako hodnota uzla





# Vyhľadávanie - zoznam vs. strom

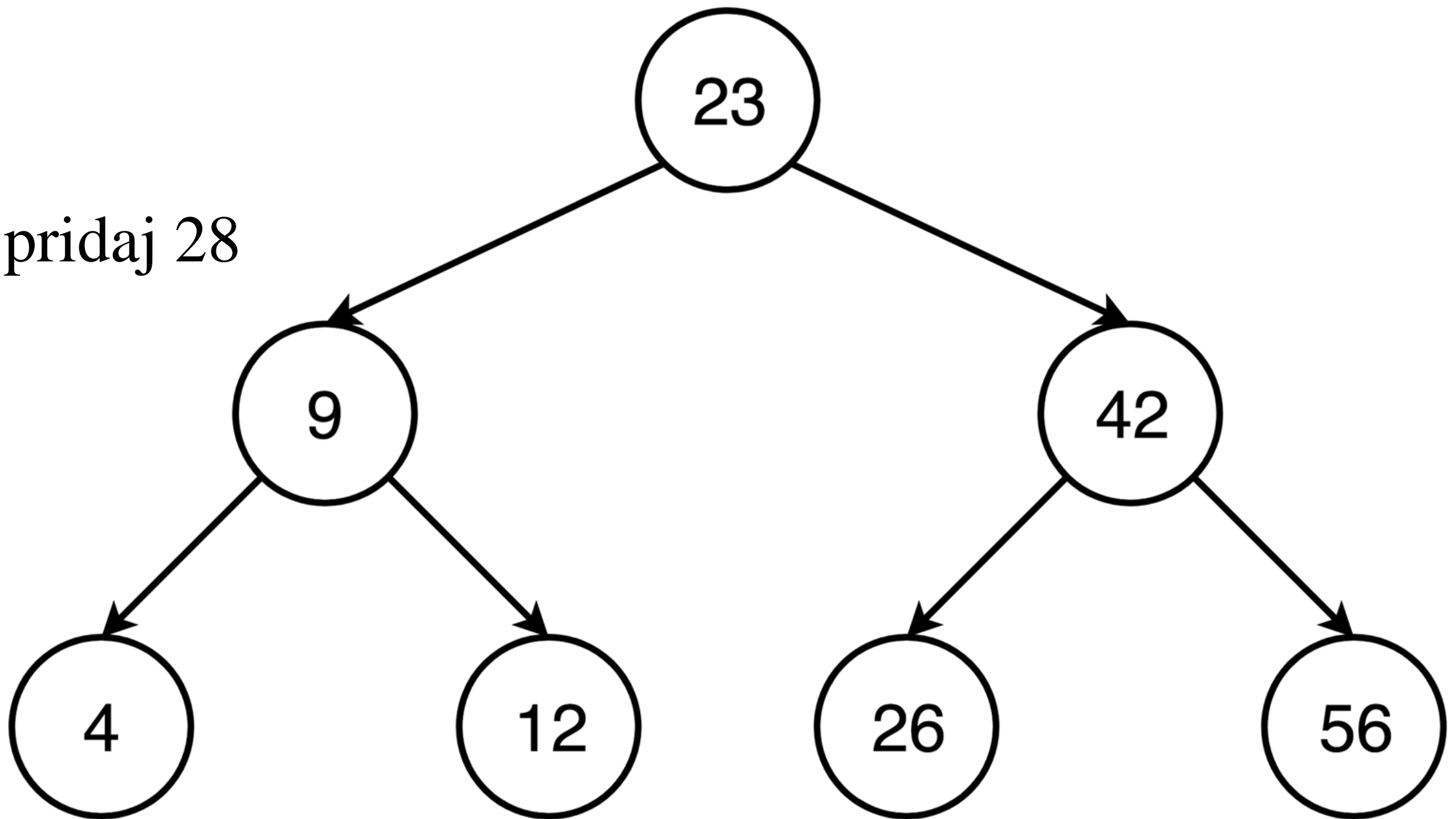
2	4	9	23	26	42	56
---	---	---	----	----	----	----



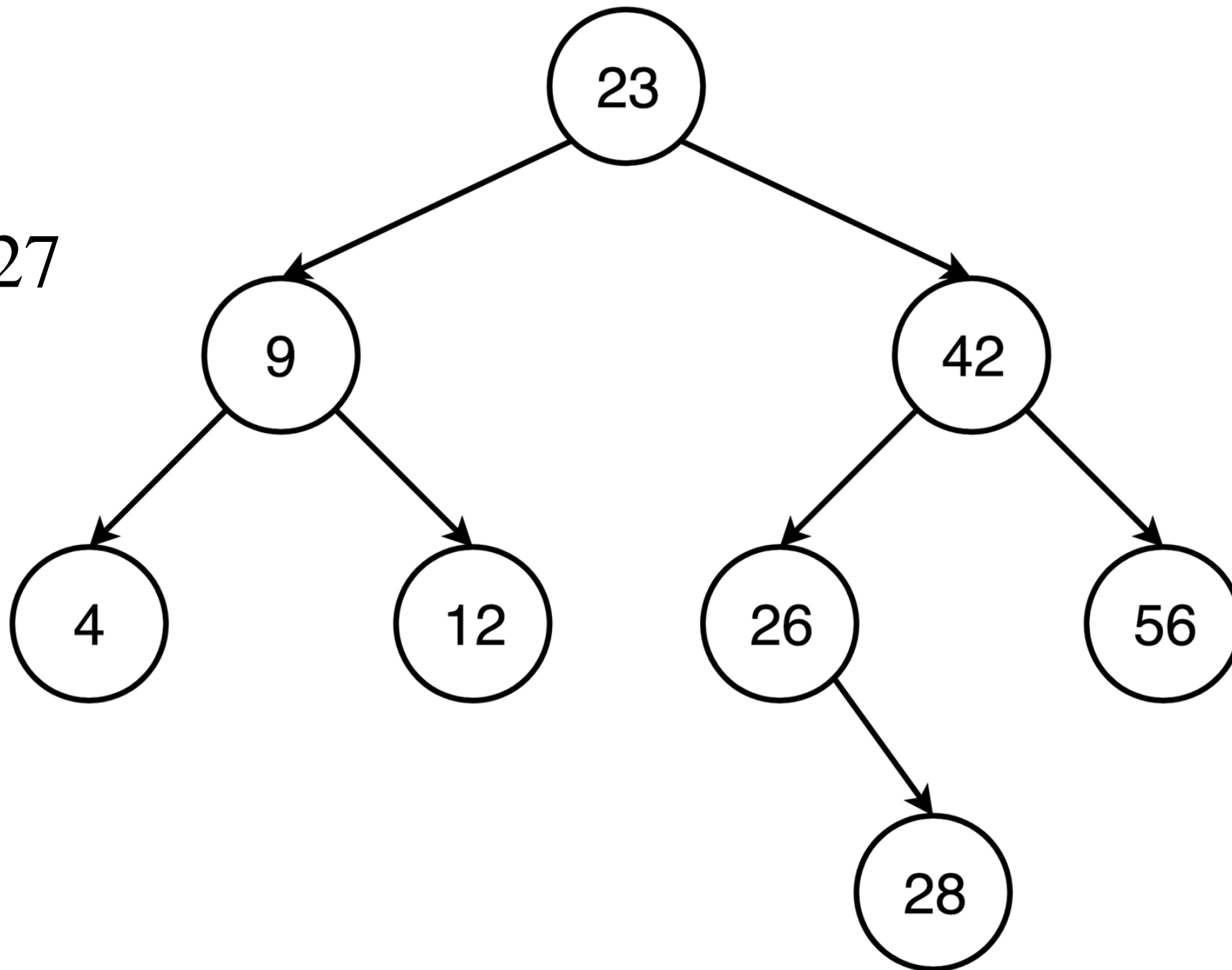
# Pridávanie prvkov do stromu

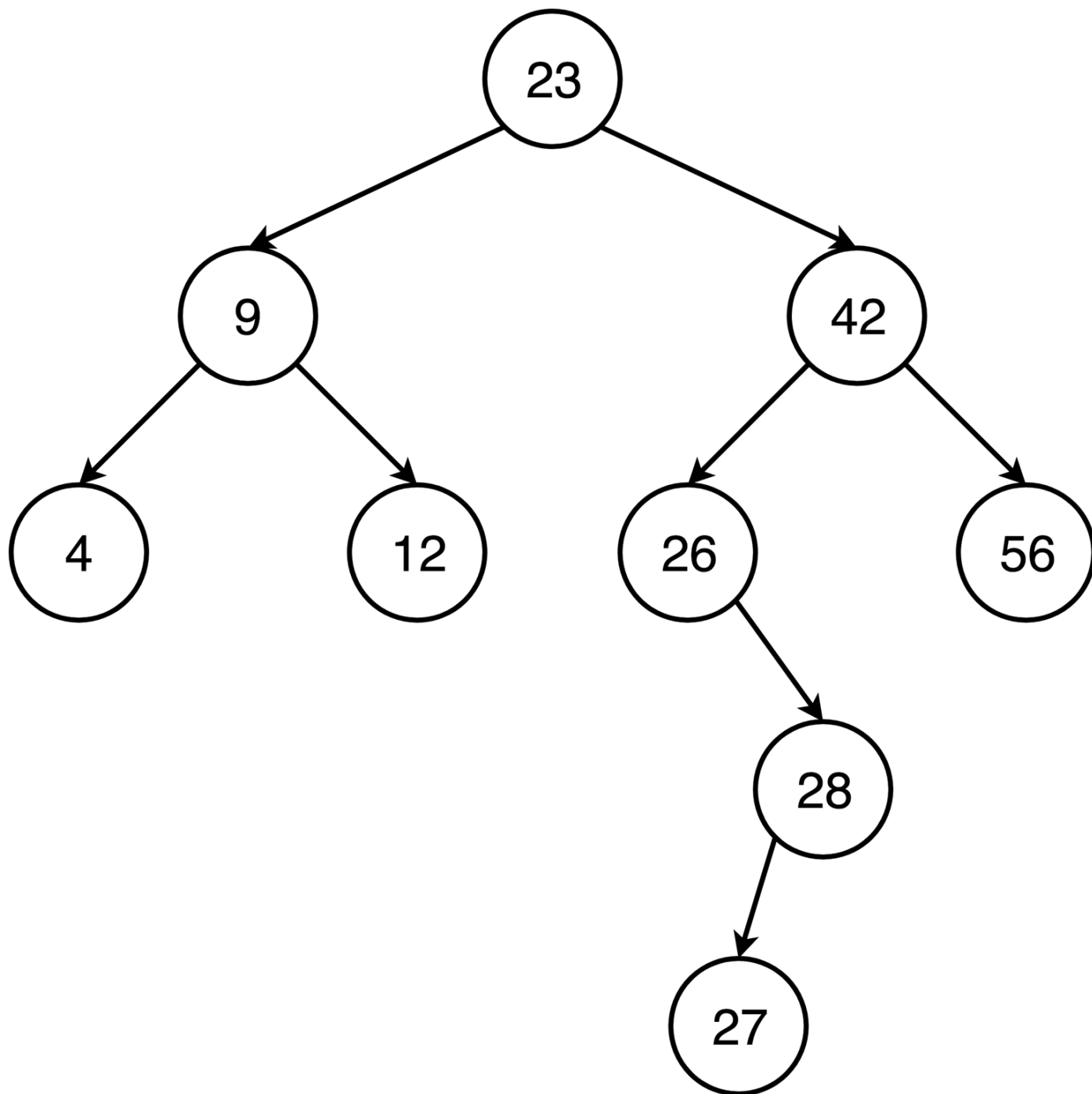
1. nájsť vhodné miesto
2. pridať uzol
3. upraviť strom ak sa stane nevyváženým

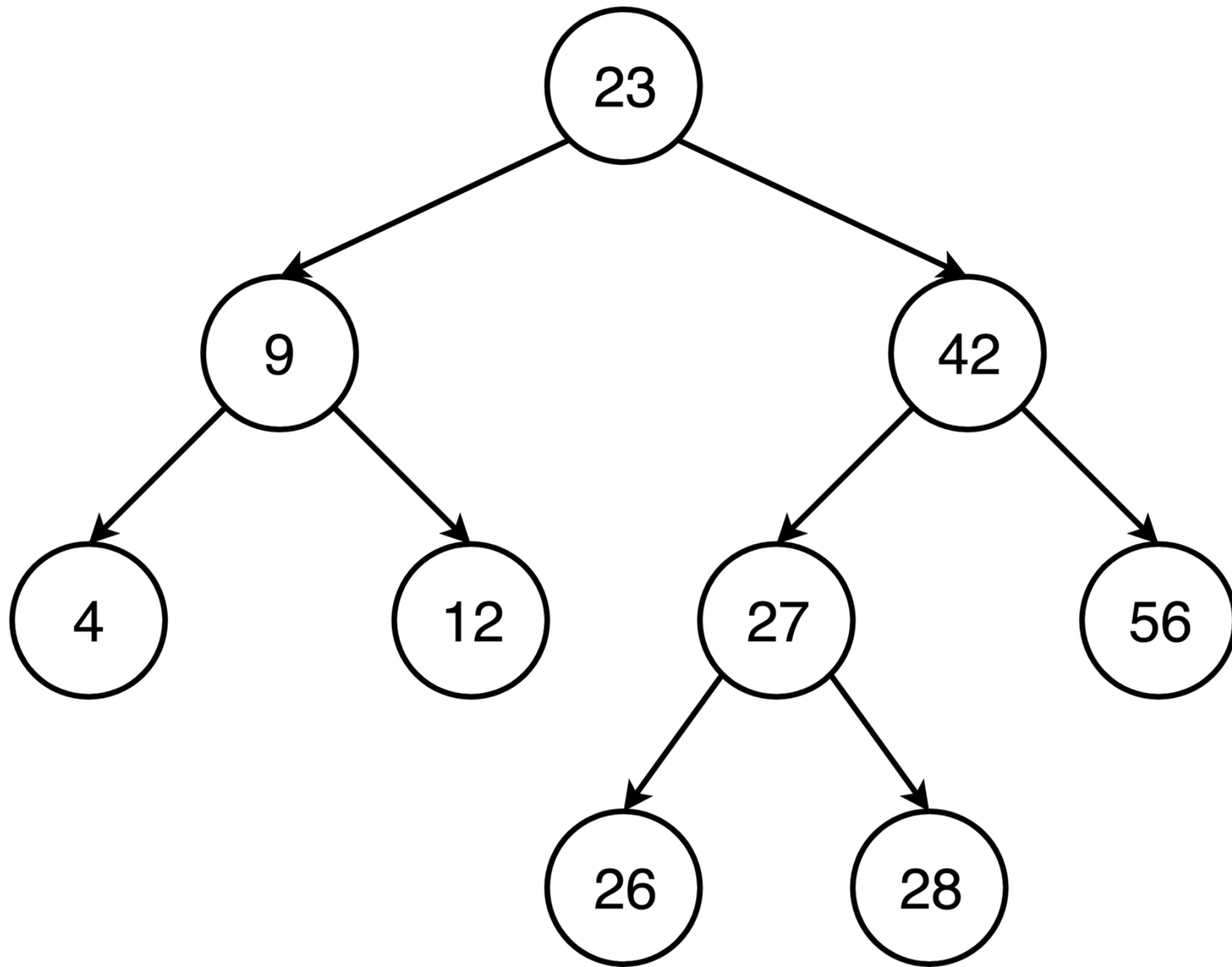
pridaj 28



pridaj 27







# 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

- údajové štruktúry
- polia
- numpy a pandas
- spojkové zoznamy
- zásobník a front
- hašovacia tabuľka
- stromy