



Základy jazyka Python

údajové štruktúry
prednáška 7

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
- 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
 - iba jeden údajový typ hodnôt

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

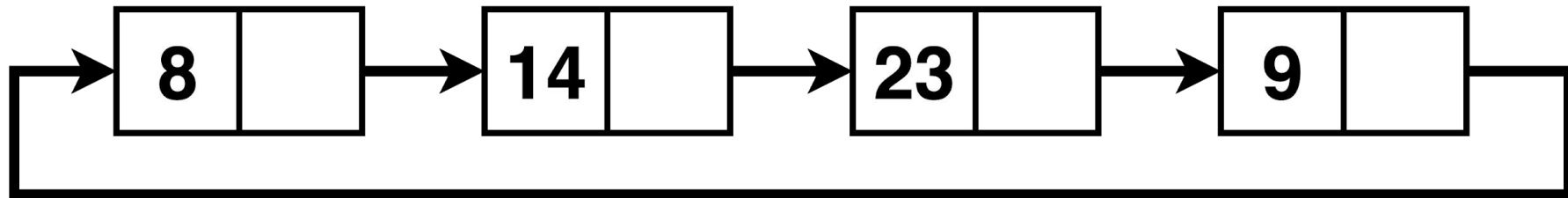
Jednosměrný spojkový zoznam



Obojsmerný spojkový zoznam



Kruhový zoznam

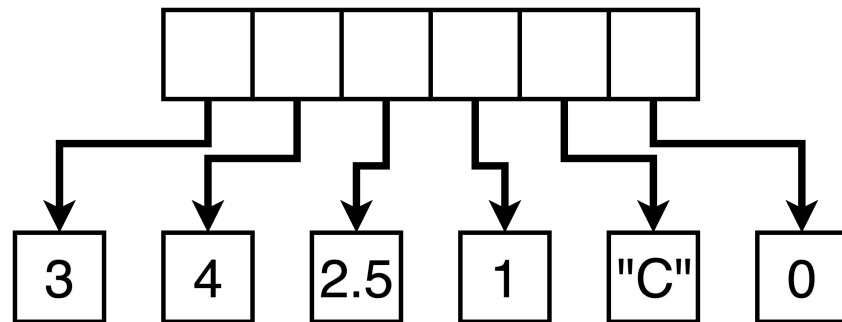
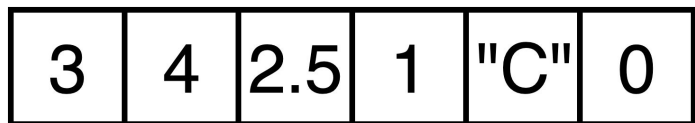


Porovnanie polí a spojkových zoznamov

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

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 na princípe 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

Hašovanie

- rozdeľuje údaje na menšie skupiny (buckety)
- 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
 - a. hašovacia funkcia vypočíta hash hodnotu
 - b. prístupujeme 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:

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

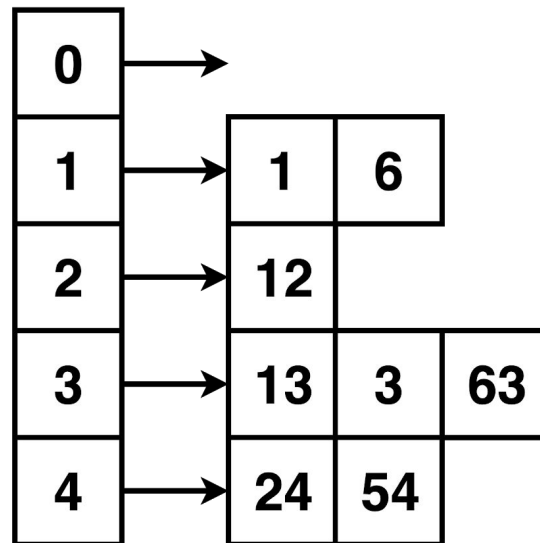
n - počet hodnôt

k - počet bucketov

Hashovanie - ukážka

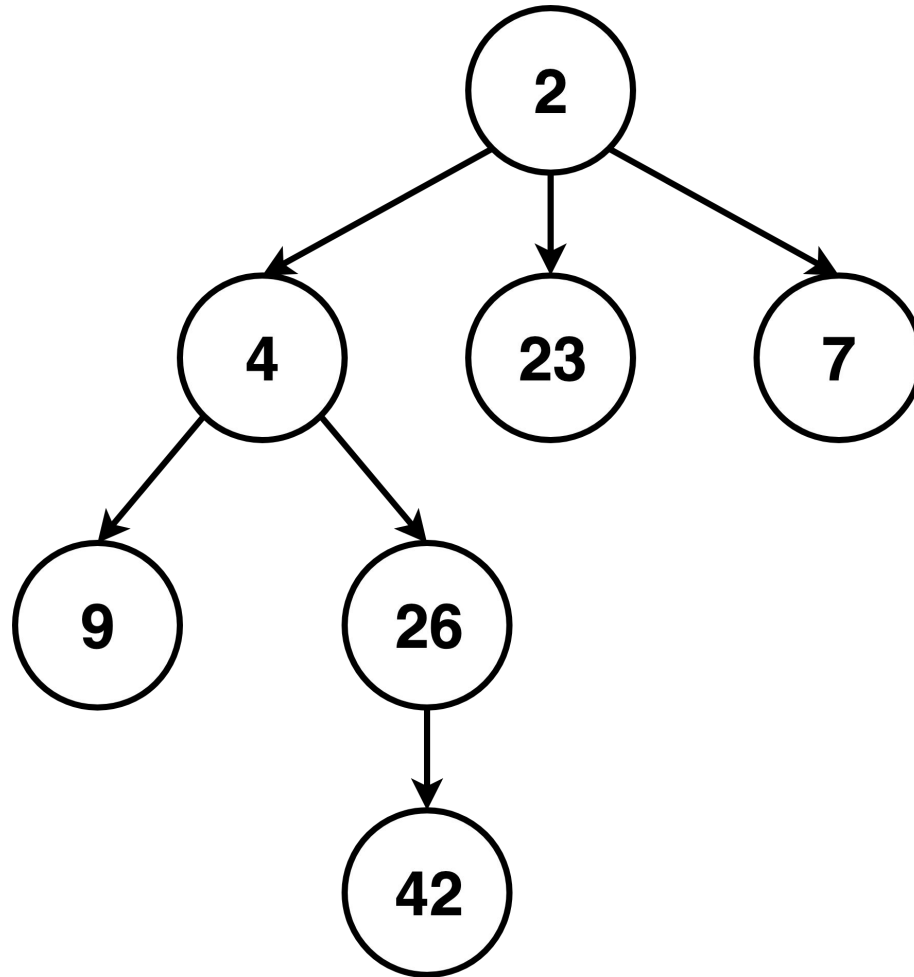
- hašovacia funkcia - zvyšok po delení 5

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



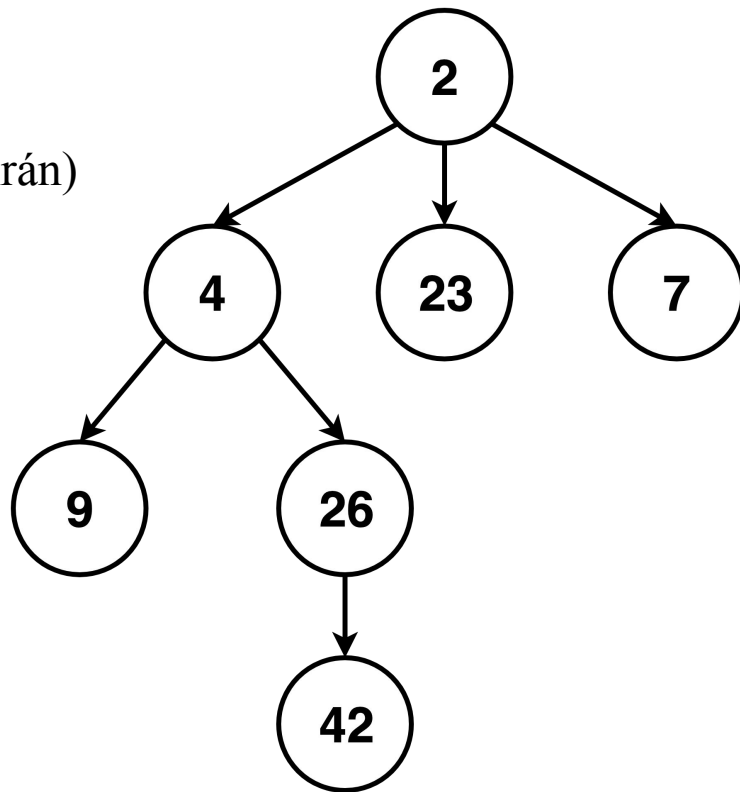
Stromy

- 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



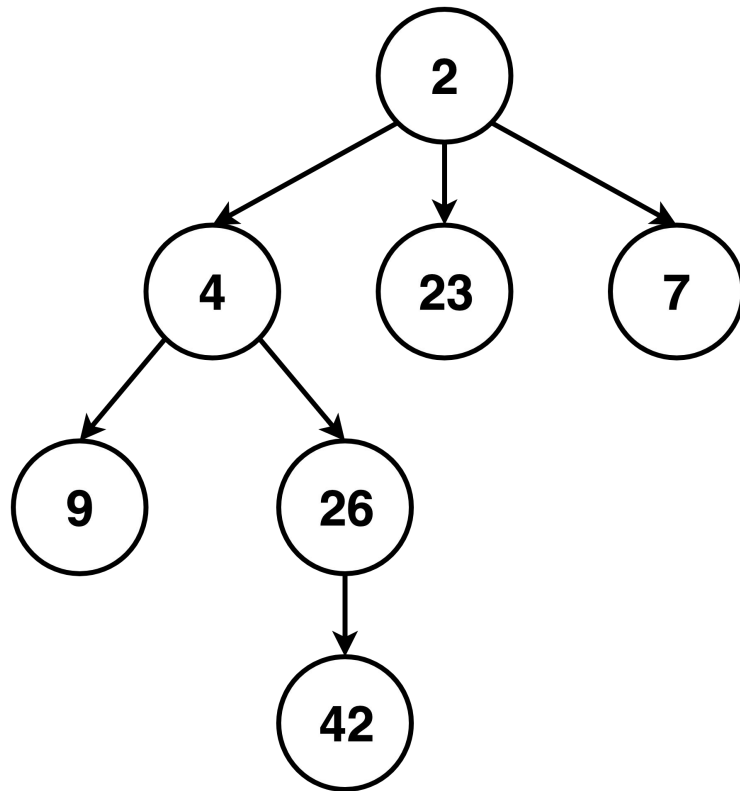
Úrovne stromu

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



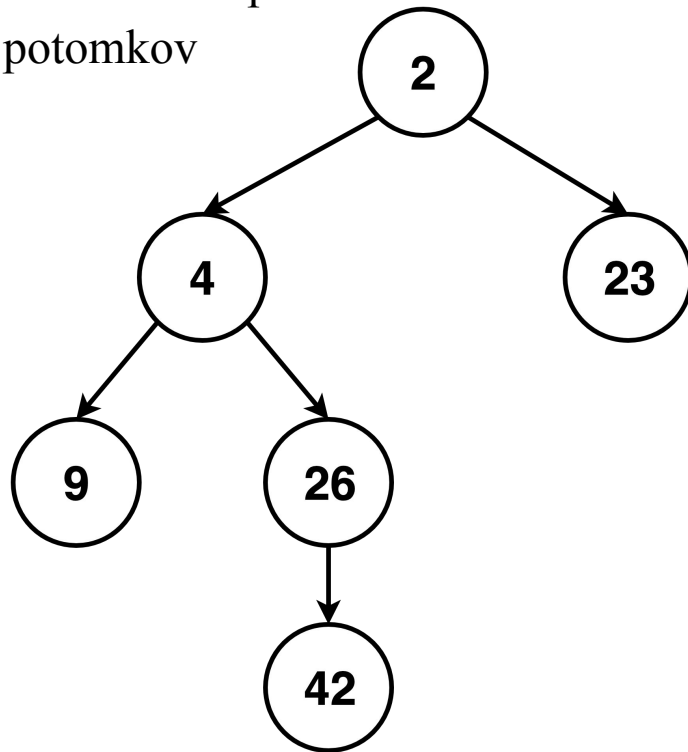
Výška stromu

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



Binárny strom

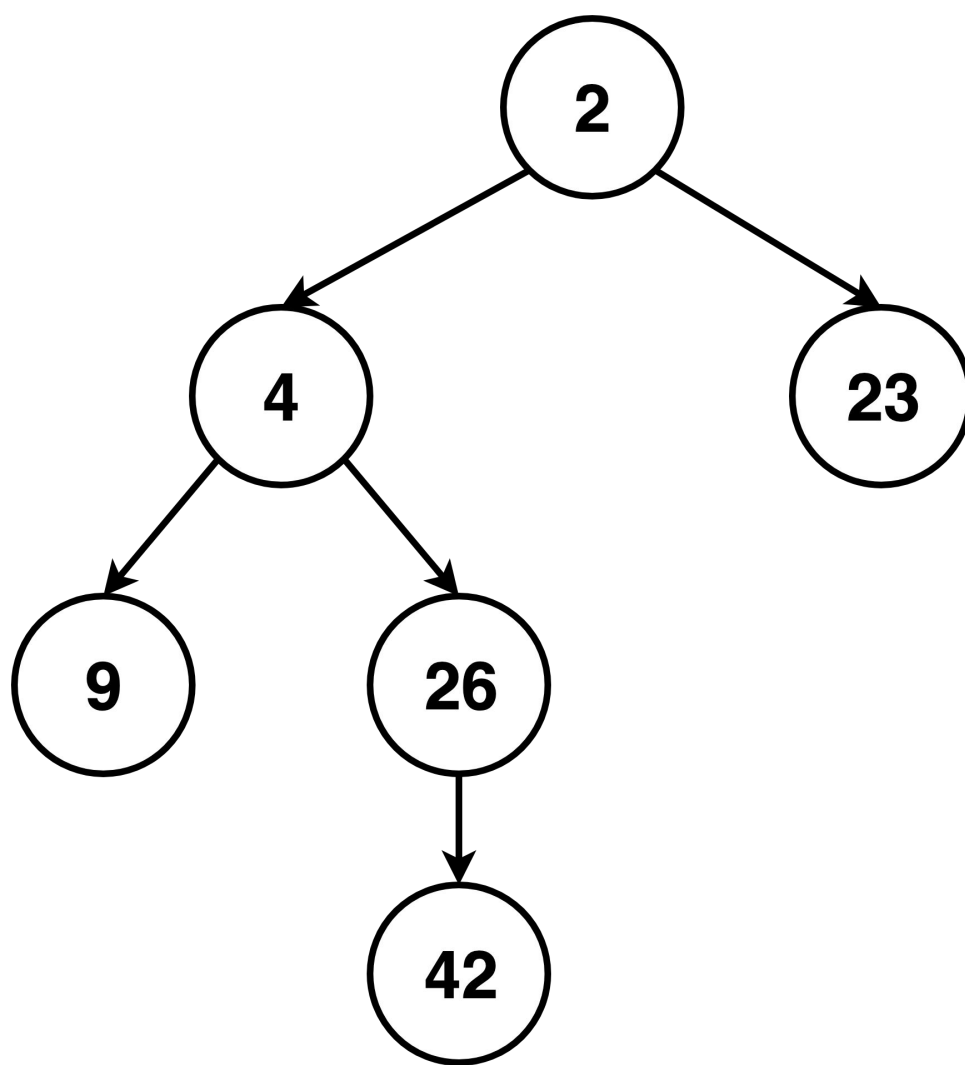
- stromová štruktúra, v ktorej každý uzol má maximálne dvoch potomkov
- striktný binárny strom - každý uzol má 0 alebo 2 potomkov
- maximálny počet uzlov na úrovni i :
 2^i
- maximálny počet uzlov v strome s výškou h :
 $2^{h+1} - 1$

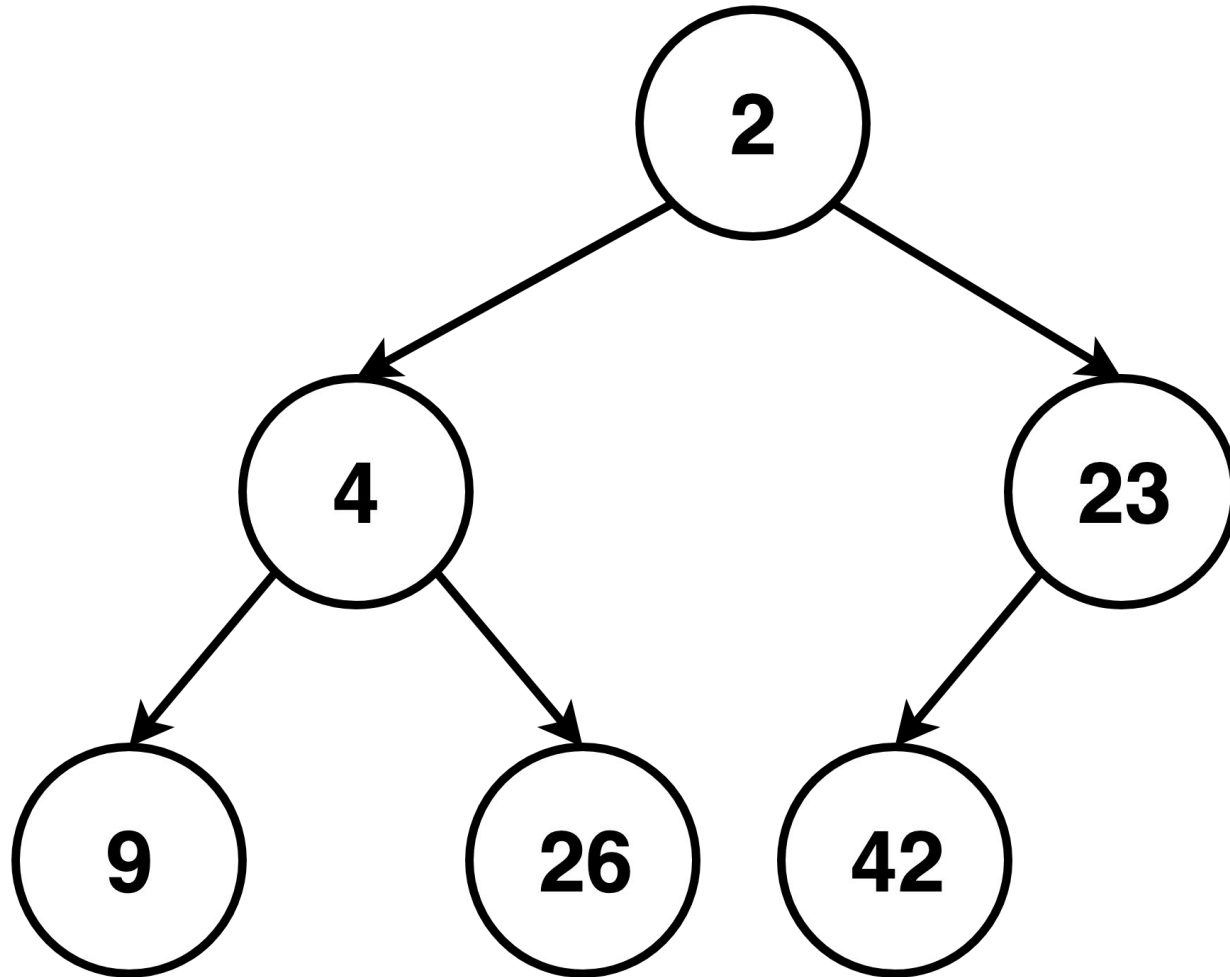


Vyvážený binárny strom

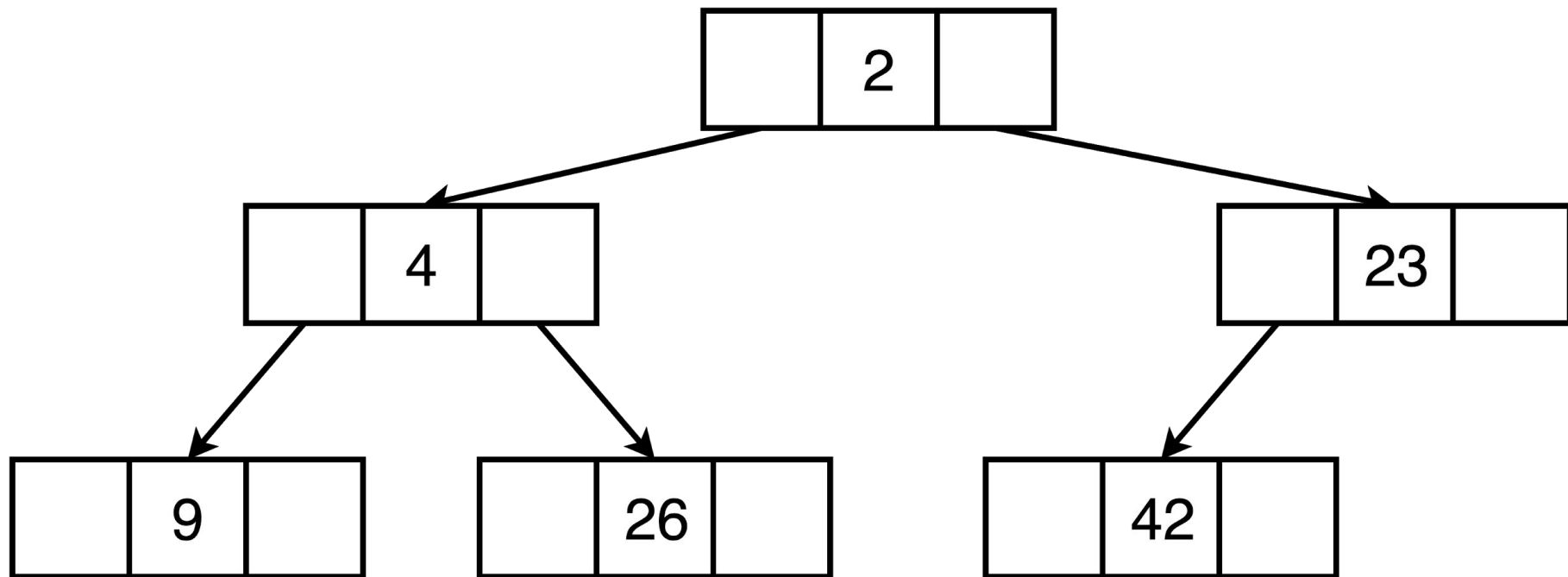
- binárny strom, v ktorom každá úroveň je plne obsadená až na poslednú, v ktorej uzly sú čo najviac vľavo
- rozdiel medzi výškou dvoch potomkov je maximálne 1
- ak aj posledná úroveň je plne obsadená, nazýva sa to aj perfektný binárny strom
- výška stromu s n uzlami:

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

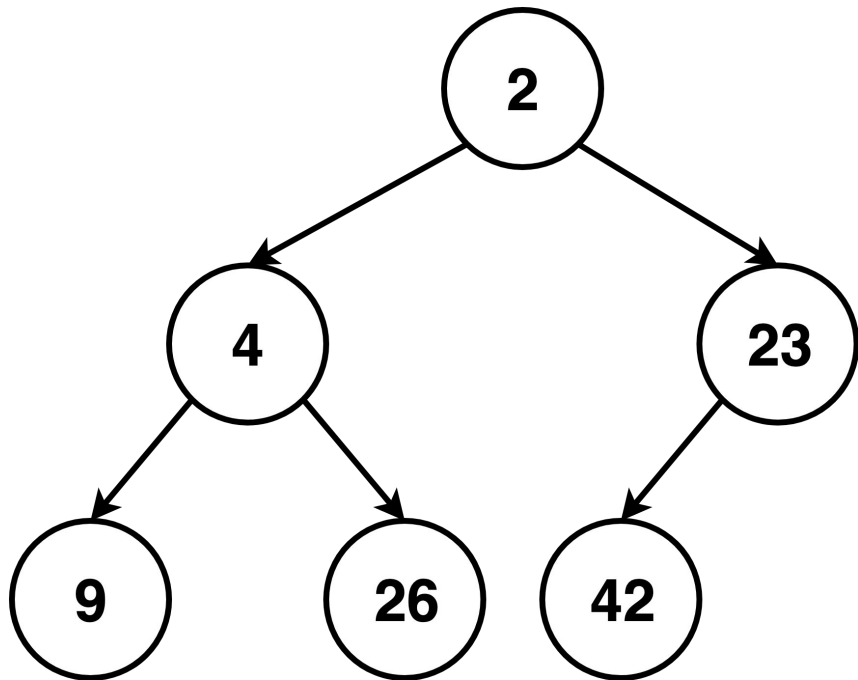




Reprezentácia stromov pomocou dynamických uzlov



Reprezentácia stromov pomocou zoznamu

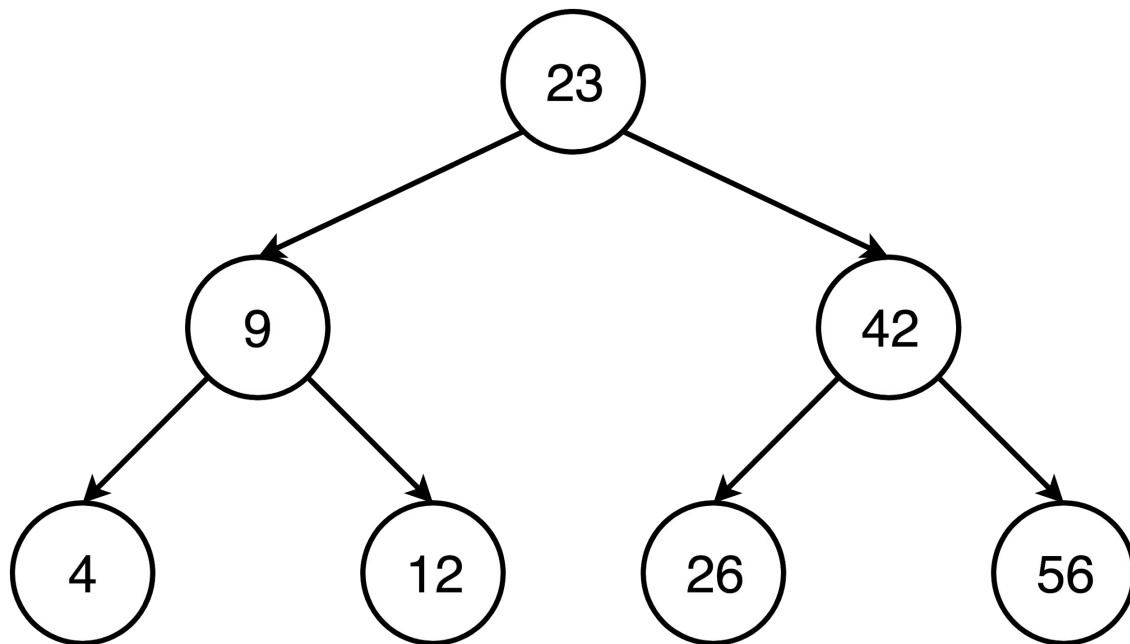


- pre uzel s indexom i :
 - ľavý potomok má index $2i+1$
 - pravý potomok má index $2i+2$

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

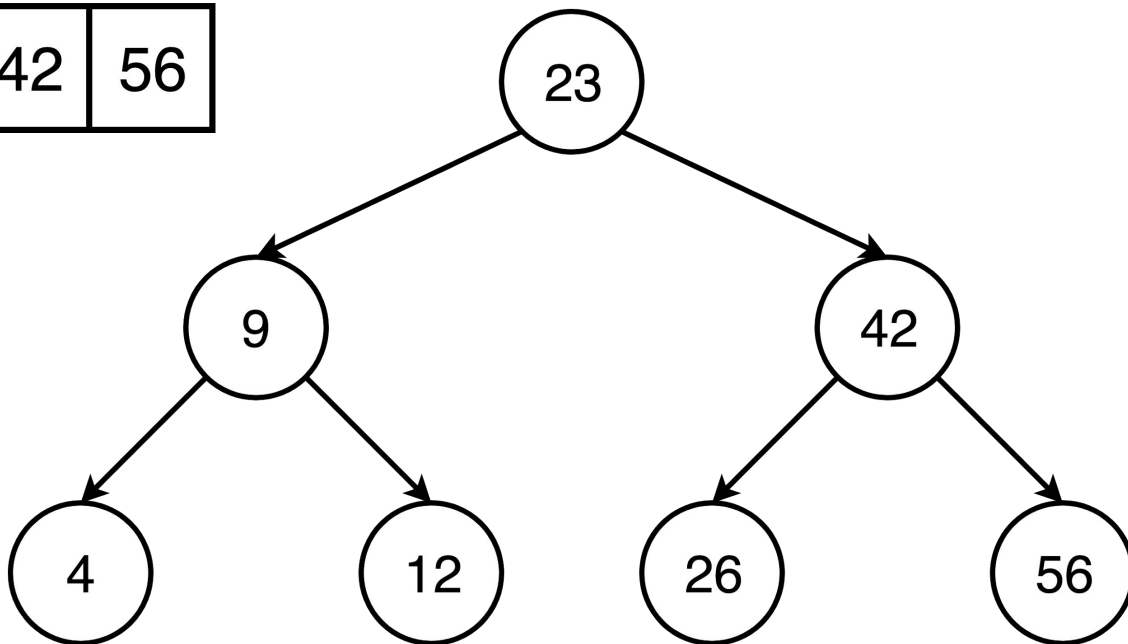
Binárny vyhľadávací strom

- 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



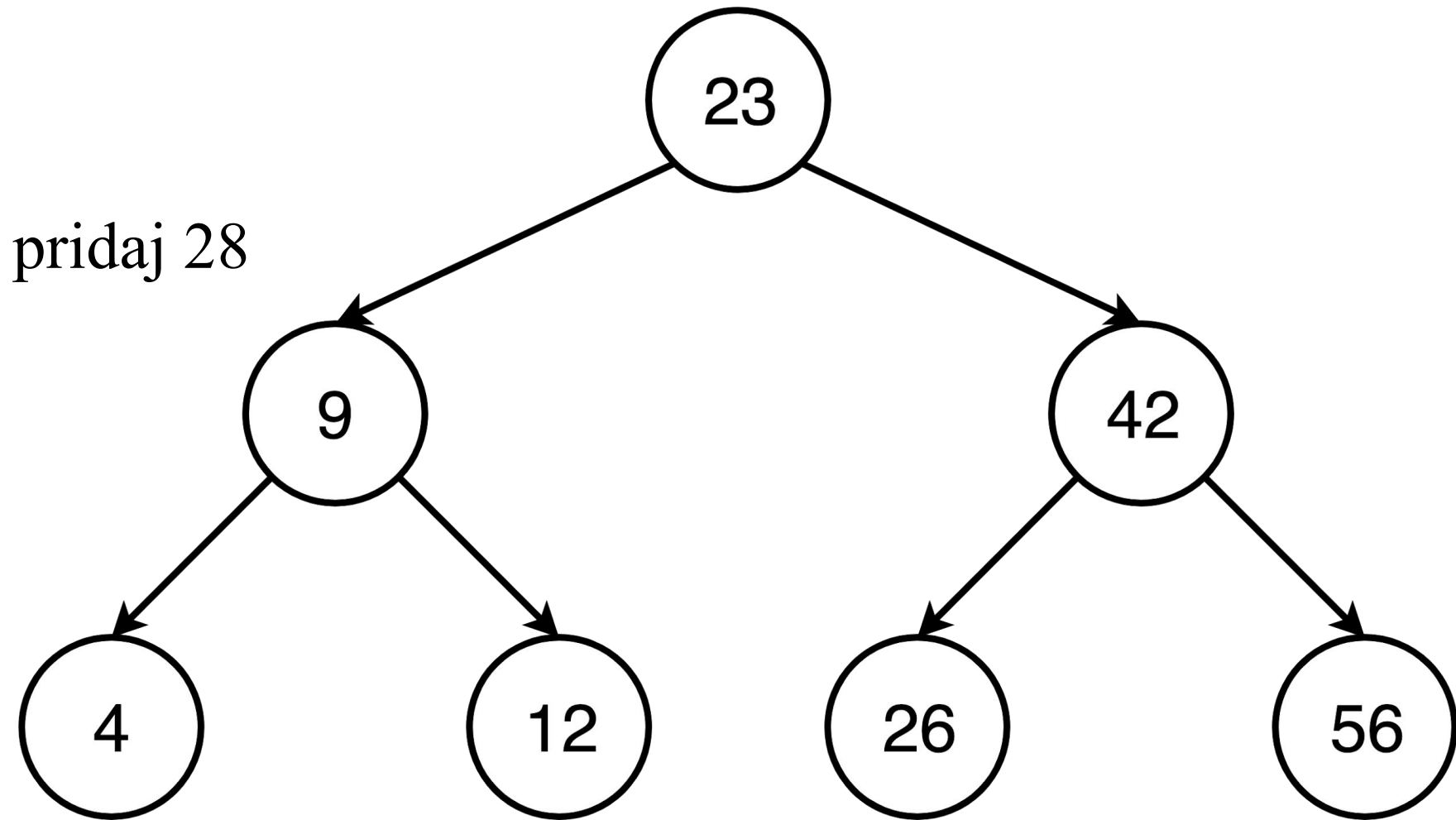
Vyhľadávanie - zoznam vs. strom

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

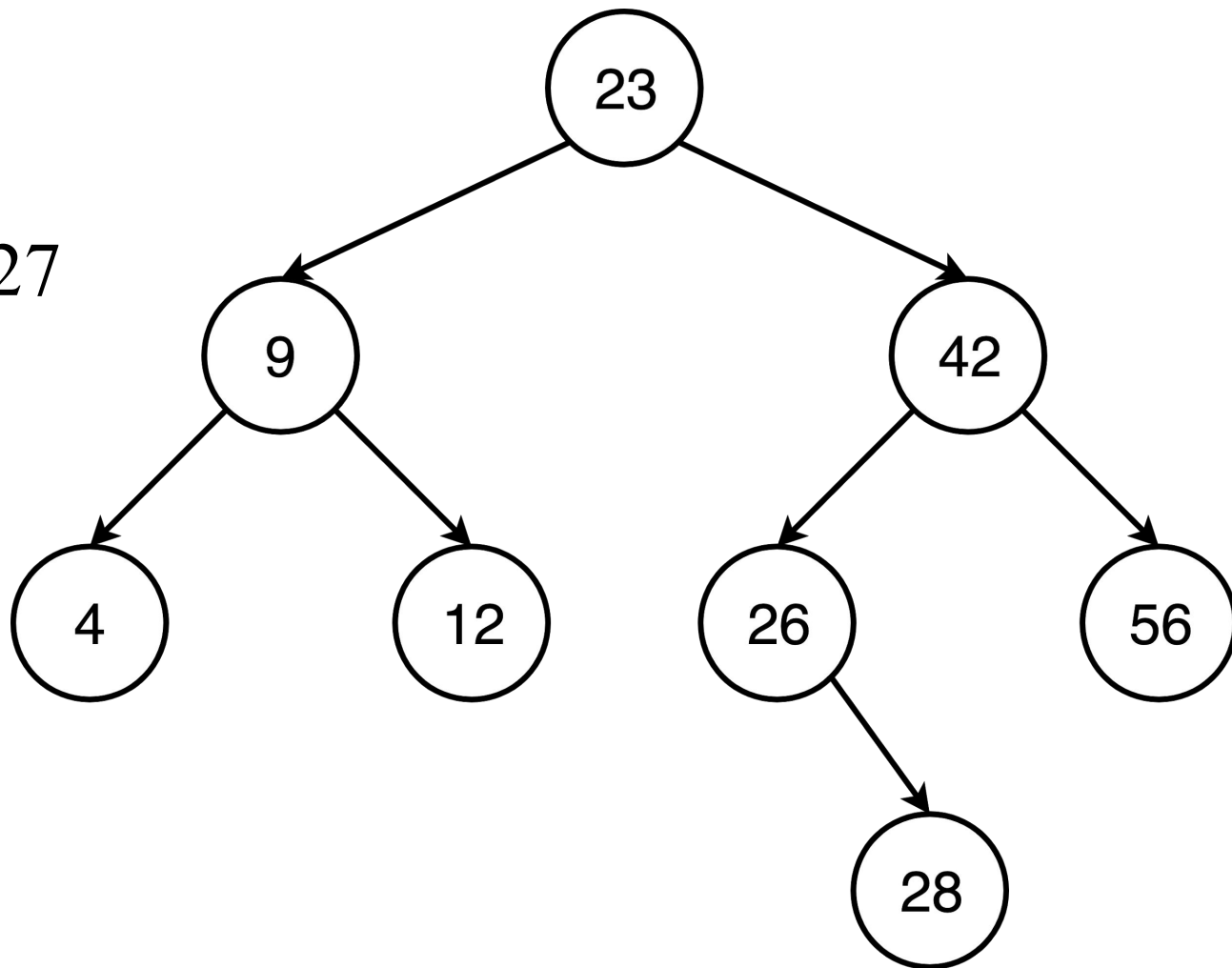


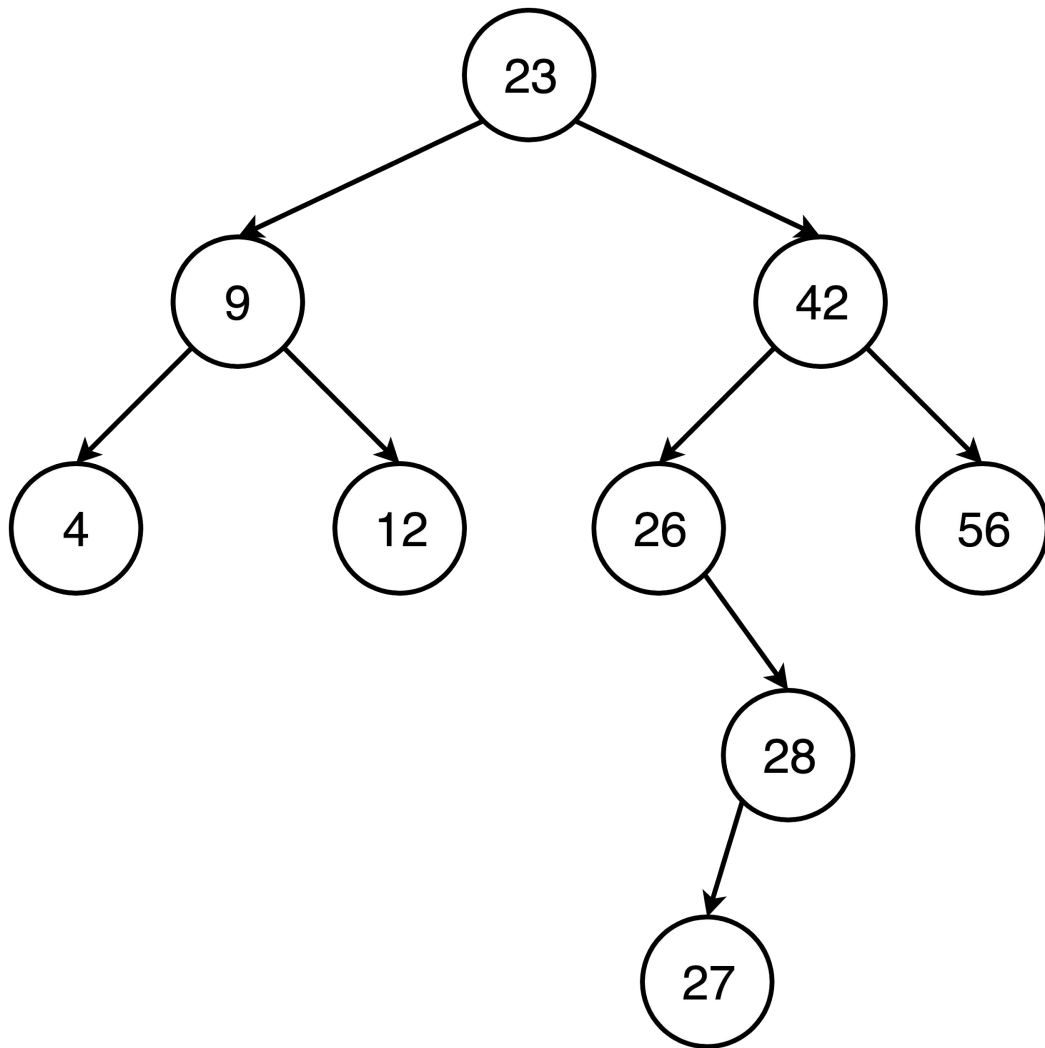
Pridávanie prvkov do stromu

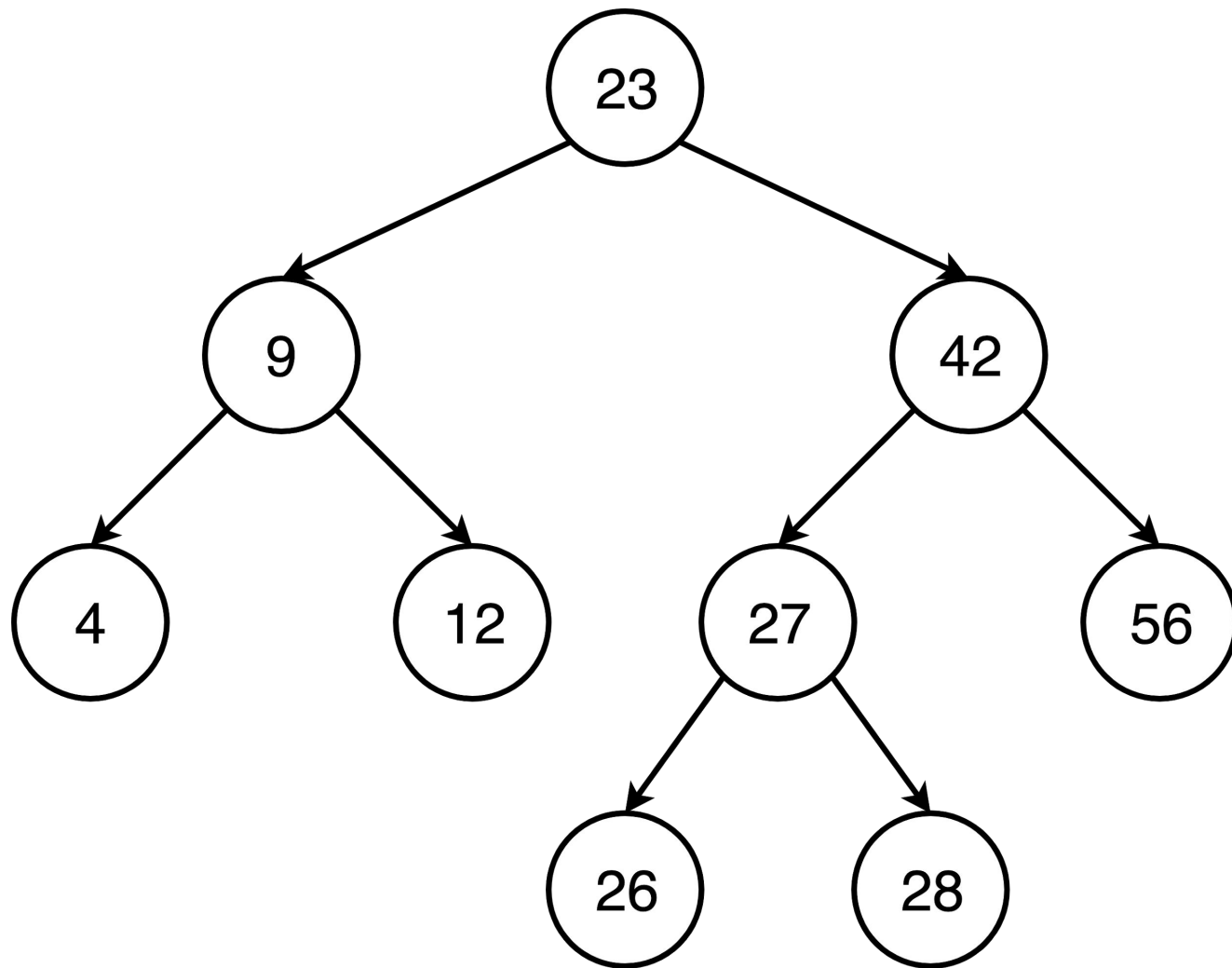
1. nájsť vhodné miesto
2. pridať uzol
3. upraviť strom ak sa stane nevyváženým



pridaj 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	$O(n)$	$O(n)$	$O(\log n)$

Otázky?