

# **Stromy**

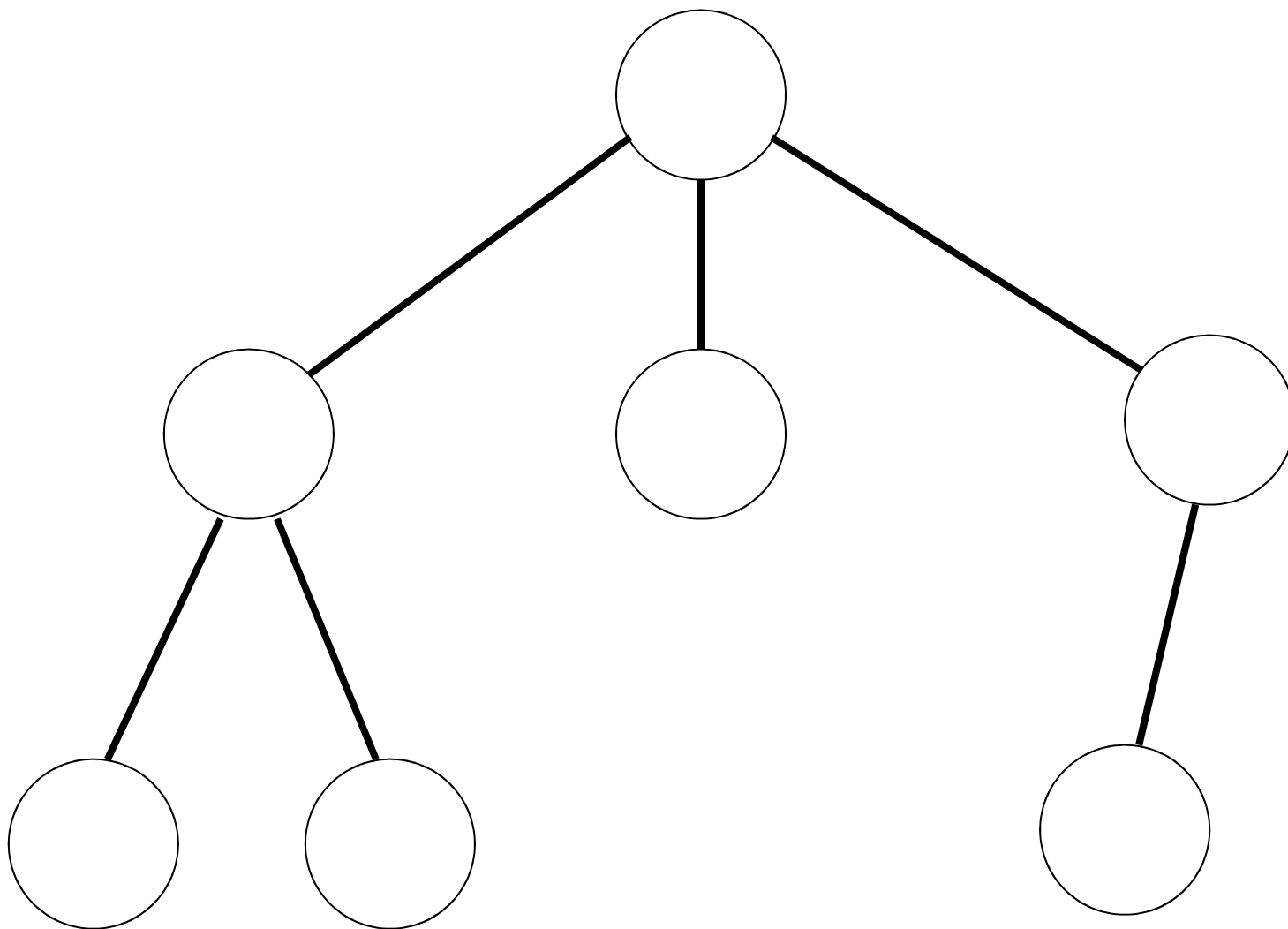
úvod

# Stromy

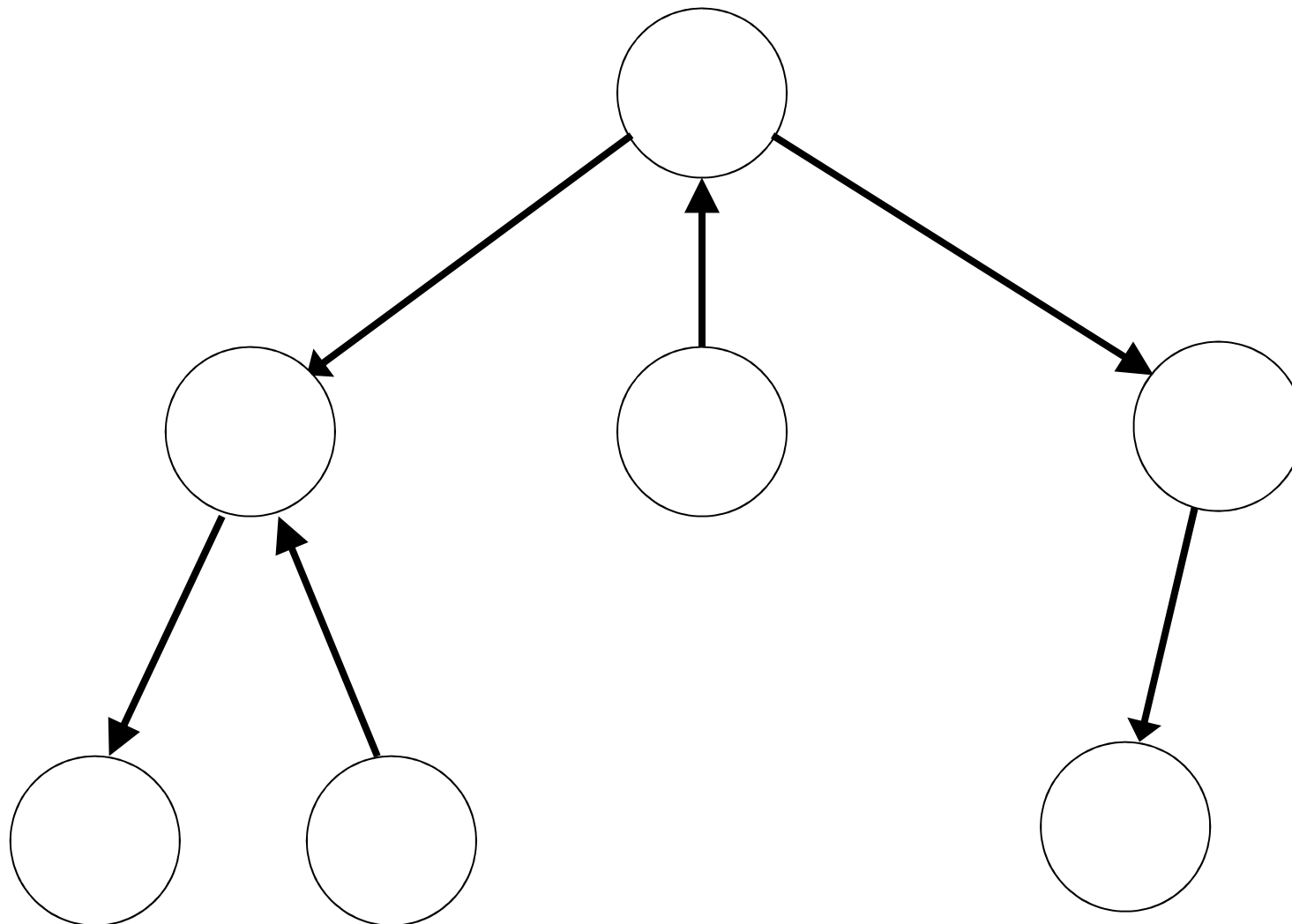
## *Strom:*

- souvislý graf bez kružnic
- využití:
  - počítačová grafika – seznam objektů
  - efektivní vyhledávání
  - výpočetní stromy
  - rozhodovací stromy

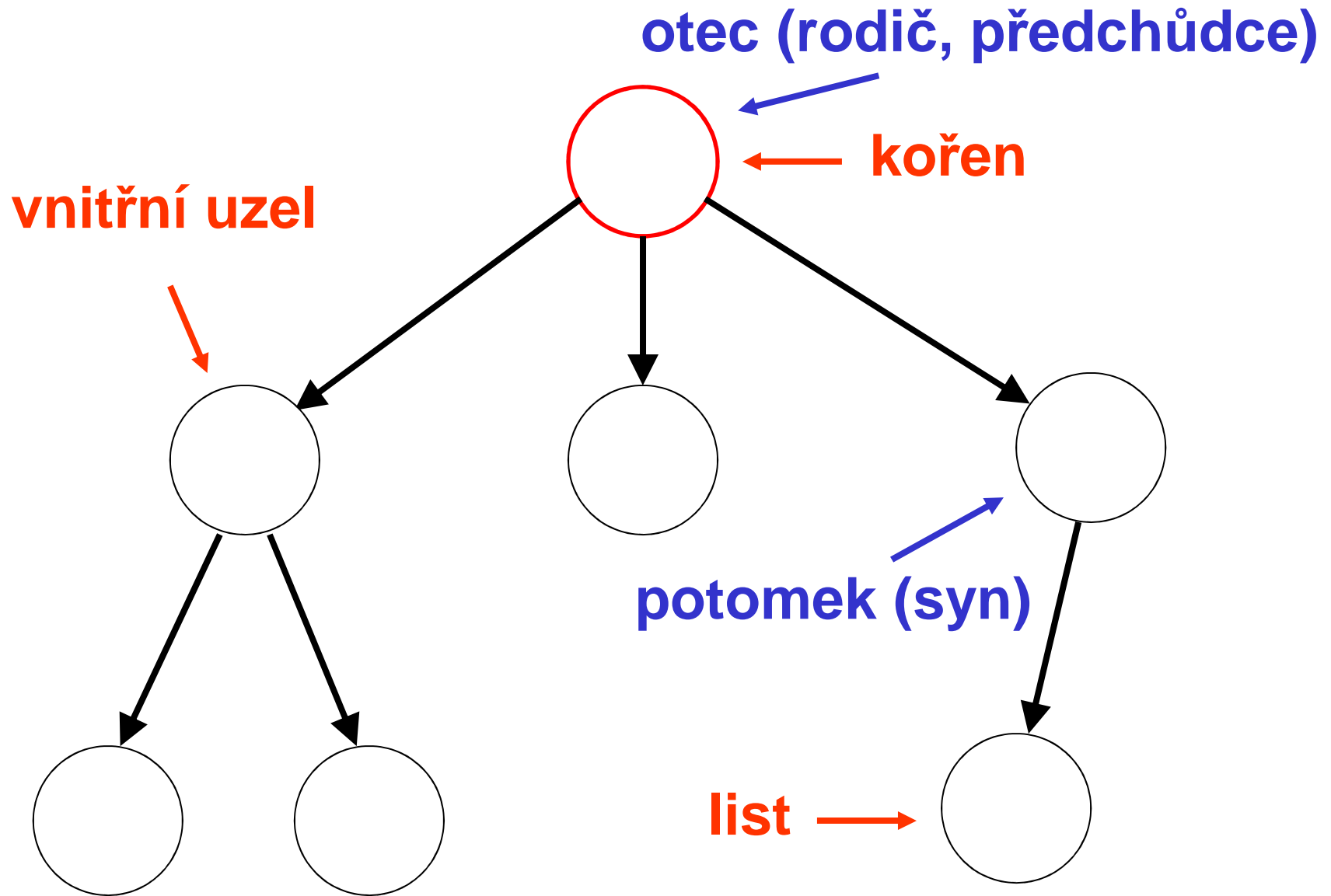
# Neorientovaný strom



# Orientovaný strom



# Kořenový orientovaný strom



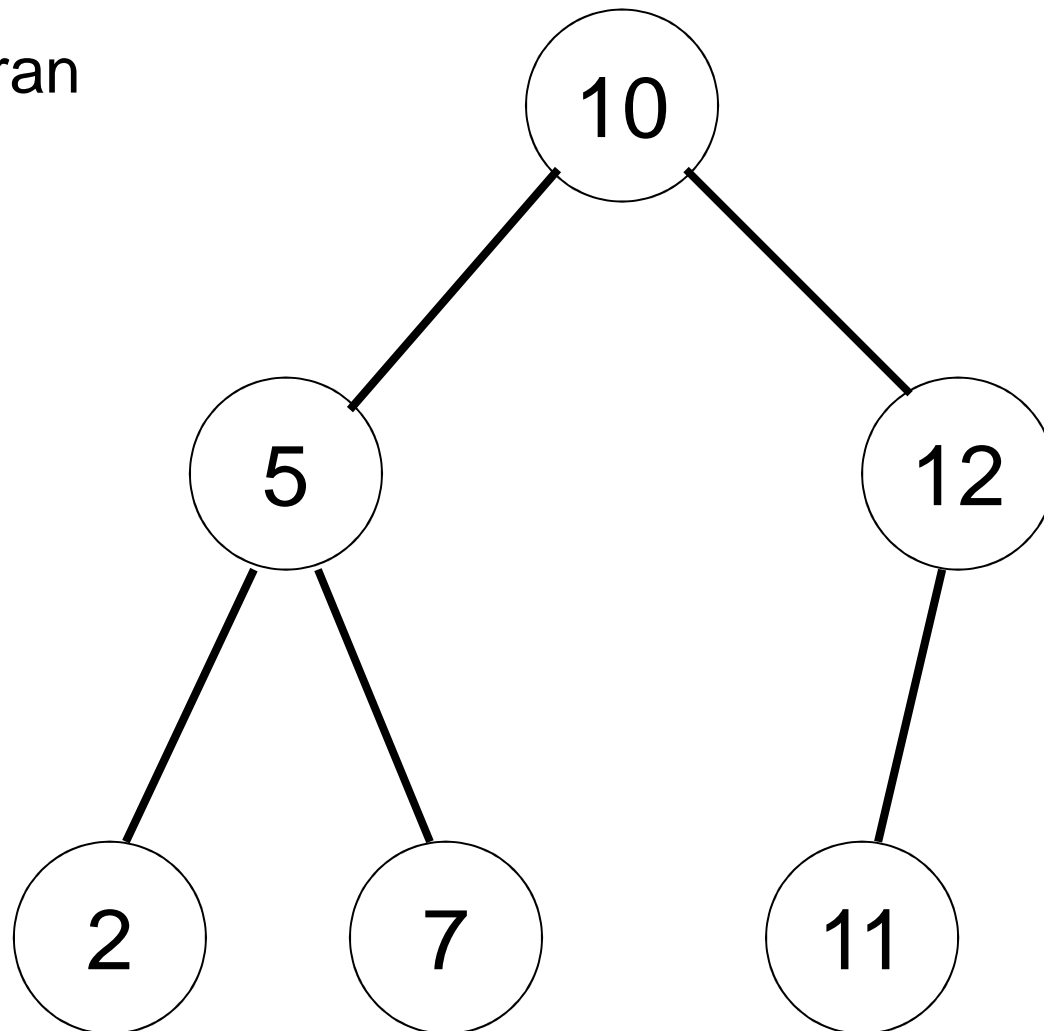
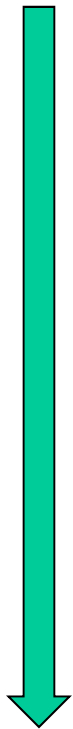
- **kořenový strom** s kořenem **u**
  - orientovaný strom, kde každá cesta  $C(u,v)$  je orientovanou cestou
- **hloubka  $hl(x)$**  uzlu  $x$  v kořenovém stromu
  - vzdálenost od kořene
- **hloubka kořenového stromu**
  - $\max_{x \in V} hl(x)$

# Binární strom

- každý uzel má maximálně dva potomky
- uspořádaný binární strom
  - uzly jsou ohodnoceny prvky (čísla,...)
  - potomek na levé straně má vždy menší hodnotu nebo rovnu než rodič syn
  - potomek na pravé straně má vždy větší hodnotu
  - uspořádané binární stromy se využívají zejména jako **vyhledávací stromy**; složitost hledání je v průměrném případě  $\log_2 n$

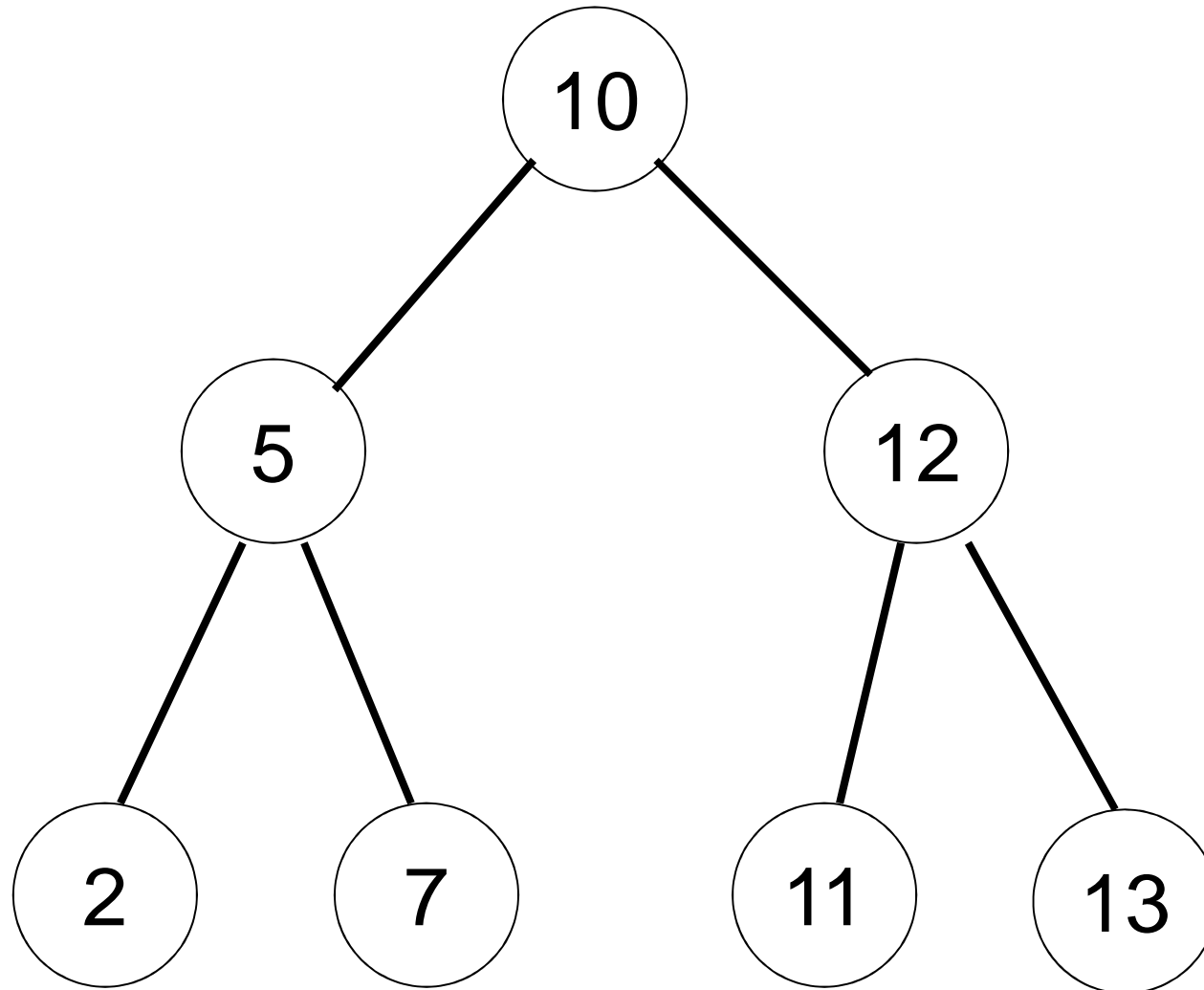
# Uspořádaný binární strom

orientace hran





# Úplný pravidelný binární strom (strom stupně 2) hloubky 2




- úplný pravidelný binární strom hloubky  $k$  má počet uzlů

$$n = 2^{k+1} - 1$$

- binární strom (nemusí být úplný) o  $n$  uzlech má minimální výšku

$$k = \lfloor \log_2(n) \rfloor$$

dolní celá část  
čísla



- příklad: strom o  $n = 5$  uzlech má výšku

$$k = \lfloor \log_2(5) \rfloor = \lfloor 2,32 \rfloor = 2$$

# Reprezentace stromu

- binární strom je nejčastěji reprezentován ukazatelem na kořen (ukazatel na uzel stromu)
- uzel je reprezentován strukturou:
  - prvkem, nesoucí informaci
  - ukazateli na levý a pravý podstrom

```
typedef struct TUzel {  
    int hodnota;  
    TUzel *levy;  
    TUzel *pravy;  
} TUzel;
```

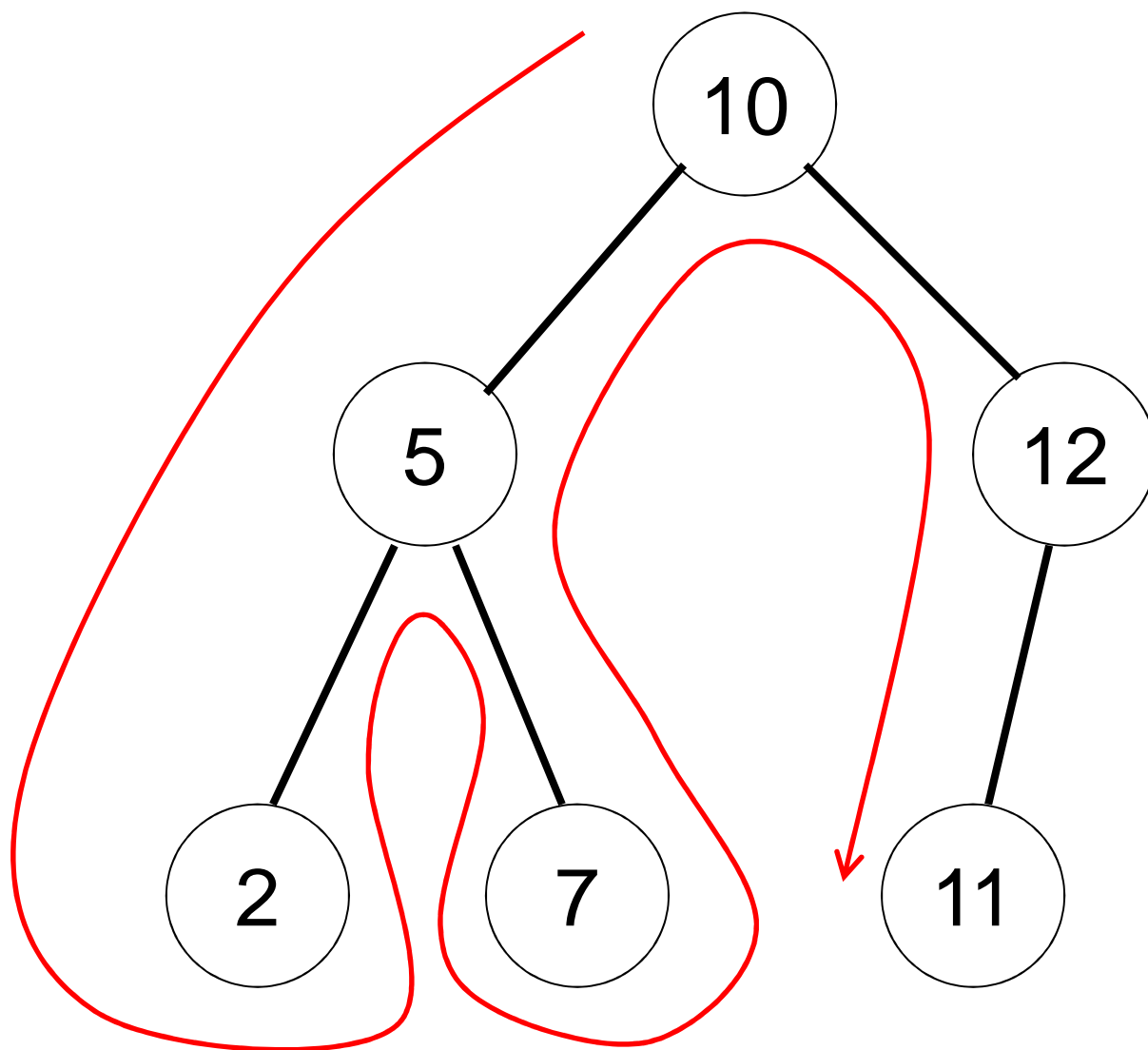
- list má ukazatele na levý a pravý podstrom nastaveny na NULL

# Operace nad stromem

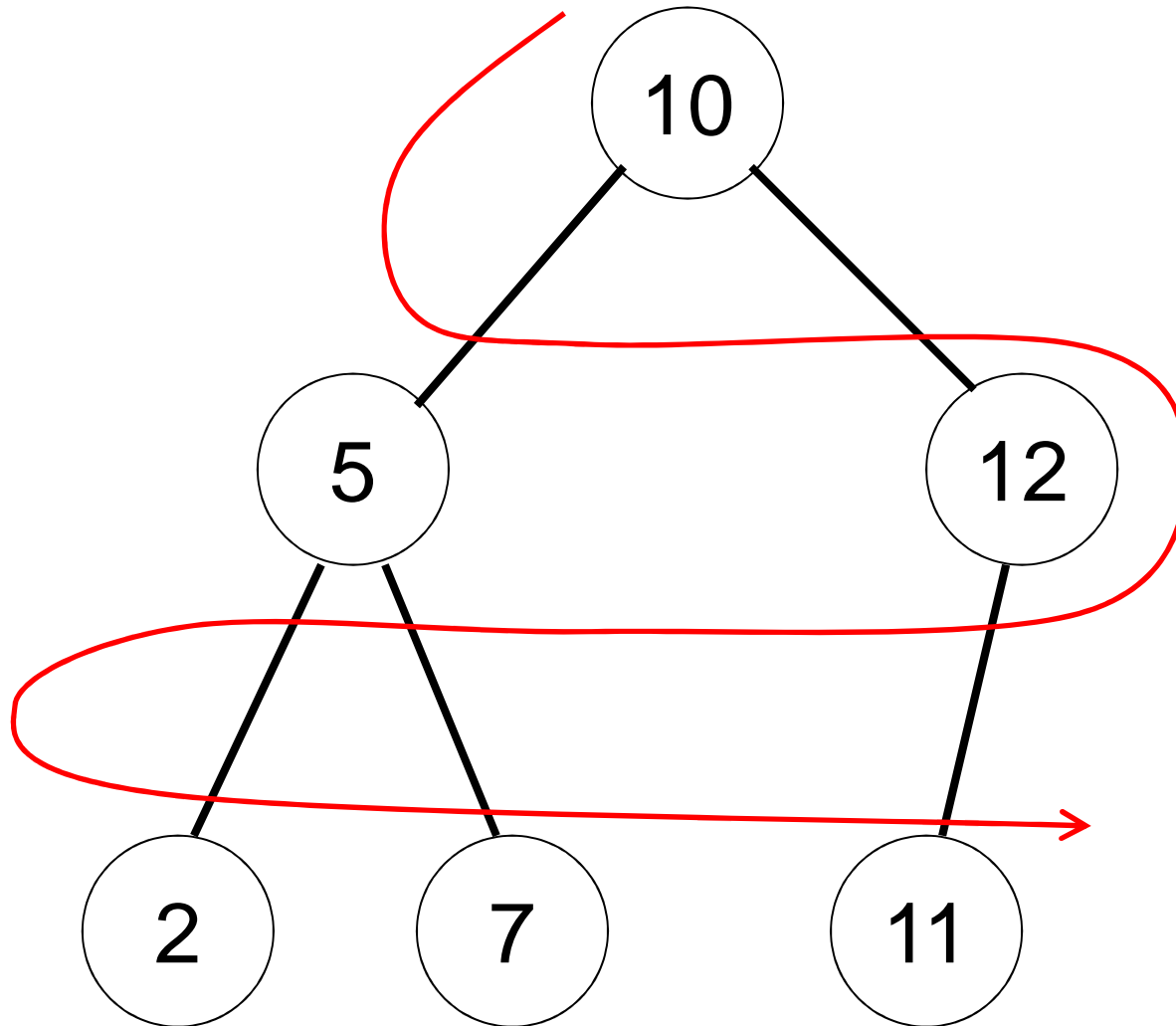
- operace nad stromem jsou (většinou) rekurzivní
  - dle typu úlohy
    - kompletní procházení stromu do hloubky musí být naprogramováno rekurzí
    - „obyčejné“ hledání prvku ve stromě lze naprogramovat i nerekurzivně
    - kompletní procházení stromu do hloubky musí být naprogramováno rekurzí
    - „obyčejné“ hledání prvku ve stromě lze naprogramovat i nerekurzivně

- operace nad stromem
  - procházení stromu (může být spojeno s nějakou akcí)
    - do šířky
    - do hloubky
  - hledání prvku ve stromě
  - vložení nového prvku jako nový list
  - vyjmutí prvku
  - rušení stromu

# Procházení stromu do hloubky



# Procházení stromu do šířky



# Obecný algoritmus procházení binárního stromu do hloubky

```
void projdi(TUzel *u)
{
    if (u==NULL) return;
    akce(u->hodnota);
    projdi(u->levy);
    projdi(u->pravy);
}
```



# Varianty procházení stromu do hloubky

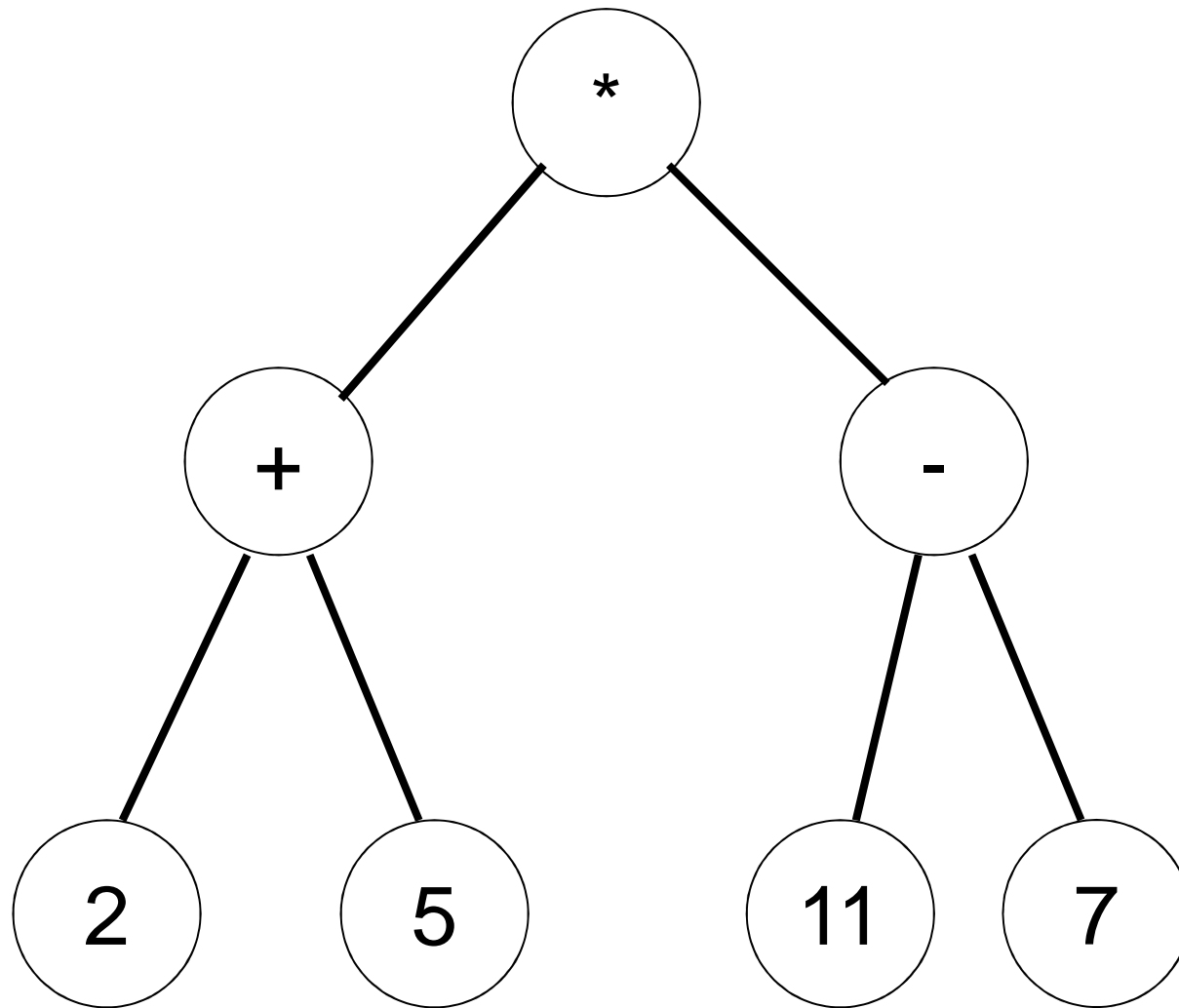
- *left order*
  - levý podstrom, zpracování uzlu, pravý podstrom
- *right order*
  - pravý podstrom, zpracování uzlu, levý podstrom
- *preorder*
  - zpracování uzlu, levý podstrom, pravý podstrom
- *další permutace, mají-li smysl*

## Příklad – left order

- výpis prvků uspořádaného stromu

```
void vypis(TUzel *u)
{
    if (u==NULL) return;
    vypis(u->levy);
    printf("%d ",u->hodnota);
    vypis(u->pravy);
}
```

# Příklad – pre order (reprezentace výrazu stromem)



- strom reprezentuje výraz
$$(2+5)*(11-7)$$
- výpis v pre-order formě (polská notace)
$$* + 2 5 - 11 7$$

```
void vypis_pre(TUzel *u)
{
    if (u==NULL) return;
    printf("%c ",u->hodnota);
    vypis_pre(u->levy);
    vypis_pre(u->pravy);
}
```

## *Poznámka:*

- stromová reprezentace se běžně používá v překladačích programovacích jazyků jako vnitřní reprezentace výrazů, konstruktů

# Hledání prvku

- vrací 1, je-li hodnota nalezena

```
int najdi(TUzel *u, int x)
{
    if (u==NULL) return 0;
    if (u->hodnota==x) return 1;
    if (x < u->hodnota)
        return najdi(u->levy,x);
    else
        return najdi(u->pravy,x);
}
```

# Hledání prvku nerekurzivně

- vrací 1, je-li hodnota nalezena

```
int hledej_nerek(Uzel *koren, int cislo)
{
    while(koren != NULL && koren->hodnota != cislo)
    {
        if (cislo < koren->hodnota)
            koren = koren -> levy;
        else
            koren = koren -> pravy;
    }
    if (koren == NULL) return 0;
    else return 1;
}
```

# Vložení nového prvku

```
void pridej(TUzel **u, int x)
{
    if (*u == NULL)
    {
        *u = (TUzel*)malloc(sizeof(TUzel));
        (*u) -> hodnota = x;
        (*u) -> levy = NULL;
        (*u) -> pravy = NULL;
    }
    else
        if (x <= (*u)->hodnota)
            pridej(&((*u)->levy), x);
        else pridej(&((*u)->pravy), x);
}
```



# Zrušení stromu

```
void zrus(TUzel *u)
{
    if (u==NULL) return;
    zrus(u->levy); zrus(u->pravy);
    free(u);
}
```

```
void main(void)
{
    TUzel *strom = NULL;

    pridej(&strom, 10);
    pridej(&strom, 5);
    pridej(&strom, 7);
    pridej(&strom, 2);
    pridej(&strom, 12);
    pridej(&strom, 11);
    najdi(strom, 5);
    zrus(strom);
}
```

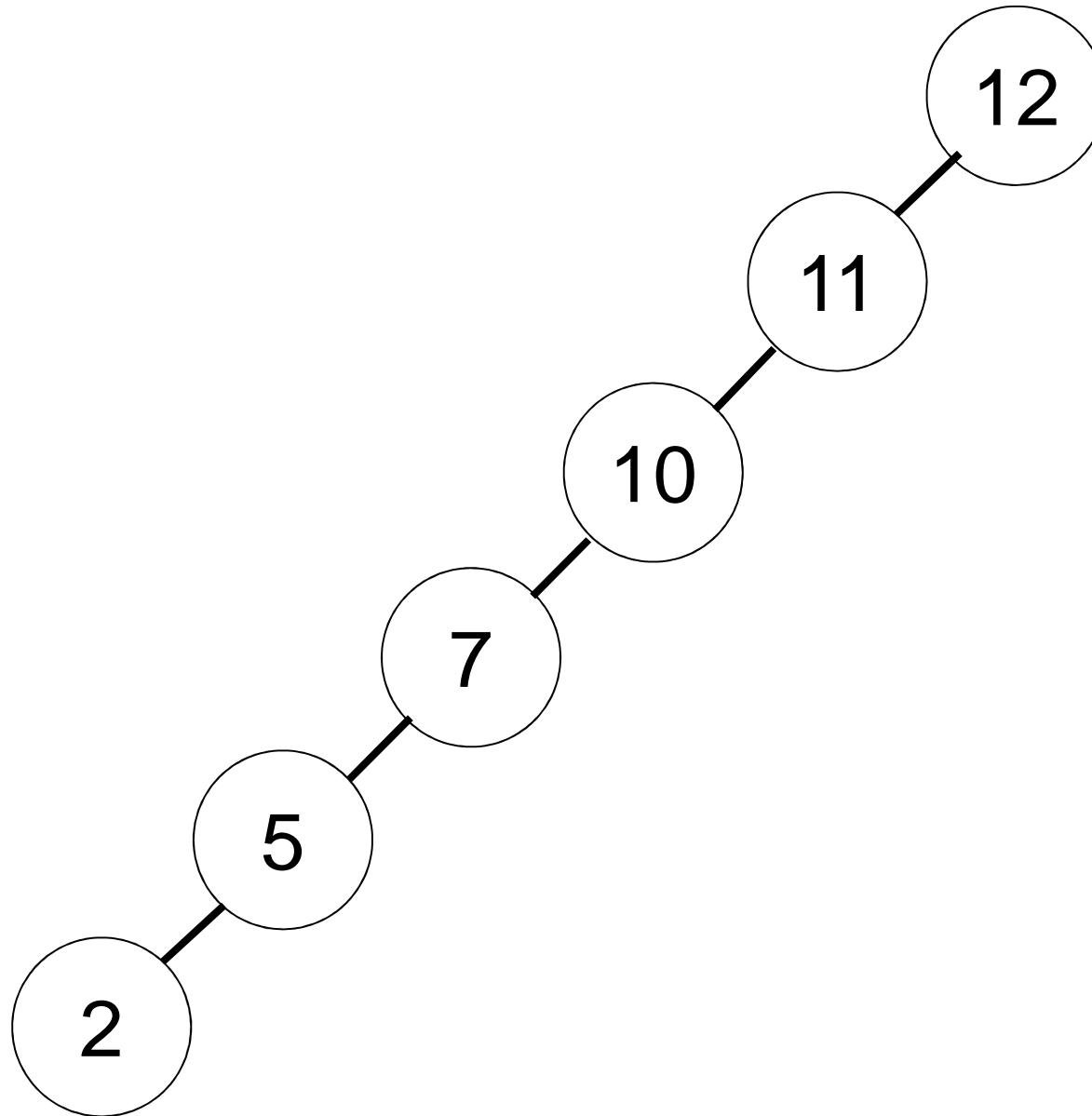
# Jaký vznikne strom nyní?

```
void main(void)  
{  
    TUzel *strom = NULL;  
    pridej(&strom, 10);  
    pridej(&strom, 5);  
    pridej(&strom, 7);  
    pridej(&strom, 2);  
    pridej(&strom, 11);  
    pridej(&strom, 12);  
}
```

# A nyní?

```
void main(void)  
{  
    TUzel *strom = NULL;  
    pridej(&strom, 12);  
    pridej(&strom, 11);  
    pridej(&strom, 10);  
    pridej(&strom, 7);  
    pridej(&strom, 5);  
    pridej(&strom, 2);  
}
```

- strom degraduje na lineární seznam



Snaha je vytvořit **vyvážený** strom, kde výška levé a pravé větve se liší maximálně o 1. To musí platit pro libovolný podstrom. Pokud tomu tak není, strom se vyvažuje pomocí tzv. **rotace uzlů**.

