Úloha 1

- vytvořte funkci (**zip** L1 L2), která spojí seznamy L1 a L2 „zipem“ – ve výsledku se budou střídat hodnoty z obou seznamů
- (zip '(10 30 5 17) '(2 14 6)) vrací
  (10 2 30 14 5 6 17)

Řešení 1

```
(define (zip L1 L2)
    (if (empty? L1) L2
            (cons (car L1)
                    (zip L2 (cdr L1)))))
```

Úloha 2

- vytvořte funkci (**sectivetsi** L1 L2), která dostane dva seznamy čísel, jejím výsledkem je seznam, který pro každé číslo X z L1 obsahuje součet všech čísel z L2 větších než X
- např. (sectivetsi '(15 8 35) '(10 20 30 5)) vrací (50 60 0)

- tip: vytvořte si pomocnou funkci počítající součet čísel větších než X

Řešení 2

```
(define (sectivetsi L1 L2)
    (if (empty? L1) '()
            (cons (soucetvetsich (car L1) L2)
                    (sectivetsi (cdr L1) L2))))

(define (soucetvetsich X L)
    (cond
        [(empty? L) 0]
        [(> (car L) X) (+ (car L) (soucetvetsich X (cdr L)))]
        [else (soucetvetsich X (cdr L))]))
```

Řešení 2a

```
(define (sectivetsi L1 L2)
    (local (
        (define (soucetvetsich X L)
            ... stejně ...
        (define (sectivetsi L1 L2)
            (if (empty? L1) '()
                (cons (soucetvetsich (car L1) L2)
                      (sectivetsi (cdr L1) L2))))
    )
    (sectivetsi L1 L2)))
```

Úloha 3

- stáhněte si https://www.nti.tul.cz/~satrapa/vyuka/altprog/bvs-data.rkt – obsahuje datovou strukturu pro uzel binárního vyhledávacího stromu (BVS)
  (define-struct **uzelBVS** (klic levy pravy))
  a konstantu testovaci obsahující takový strom
- definujte funkci (**hloubka** strom), která spočítá hloubku daného BVS (délku nejdelší cesty z kořene do listu)

Řešení 3

```
(define (hloubka strom)
    (if (empty? strom) 0
            (+ 1   (max (hloubka (uzelBVS-levy strom))
                        (hloubka (uzelBVS-pravy strom))))))
```

Úloha 4

- vytvořte funkci
  (**linearizuj** strom)
  která vrátí seznam všech klíčů obsažených v BVS
- procházejte strom v infixovém pořadí, výsledný seznam pak bude seřazen

Řešení 4

```
(define (linearizuj strom)
    (if (empty? strom) '()
        (append
            (linearizuj (uzelBVS-levy strom))
            (list (uzelBVS-klic strom))
            (linearizuj (uzelBVS-pravy strom)))))
```

Úloha 5

- vytvořte funkci
  ;jevBVS?: X (X X → boolean) Y → boolean
  (**jevBVS?** hodnota jemensi? strom)
  která hledá danou hodnotu v BVS
- hodnotou nemusí být číslo, funkcionální parametr jemensi? porovná hodnotu s klíčem uzlu a vrací #t, pokud je první parametr menší než druhý
- např. pro BVS, kde klíči jsou řetězce znaků, voláme
  (jevBVS? "cosi" string<? textovyBVS)

Řešení 5

```
(define (jevBVS? hodnota jemensi? strom)
    (cond
        [(empty? strom) #f]
        [(jemensi? hodnota (uzelBVS-klic strom))
            (jevBVS? hodnota jemensi?
                                    (uzelBVS-levy strom))]
        [(jemensi? (uzelBVS-klic strom) hodnota)
            (jevBVS? hodnota jemensi?
                                    (uzelBVS-pravy strom))]
        [else #t]))
```

Úloha 6

- vytvořte funkci (**pocet-uzlu** strom), která vrátí počet uzlů v **obecném** stromě
- uzel obecného stromu je implementován seznamem:
  (klíč podstrom1 podstrom2 ...)
- krok stranou: jak jednoduše spočítat uzly BVS s využitím dříve definovaných funkcí?

Řešení 6

```
(define (pocet-uzlu strom)
    (local (
        (define (spocitej-podstromy podstromy)
            (if (empty? podstromy) 0
                (+  (pocet-uzlu (car podstromy))
                    (spocitej-podstromy (cdr podstromy)))))
        (define (pocet-uzlu strom)
            (if (empty? strom) 0
                (+ 1 (spocitej-podstromy (cdr strom)))))
    )
    (pocet-uzlu strom)))
```

Úloha 7

- vytvořte funkci (**obecny->BVS** strom), která převede obecný strom na binární vyhledávací
- vytvořte funkci (**BVS->obecny** strom), která převede binární vyhledávací strom na obecný ve stejném tvaru

Úloha 8

- vytvořte funkci (**maximum** strom), která vrátí maximální hodnotu obsaženou v obecném stromu