## Fronta a zásobník

Připomeňme si: Fronta a zásobník jsou struktury připouštějící dvě operace.

- Přidání prvku, které dostane za vstup prvek (proměnnou, hodnotu, jiný objekt) a
- odebrání z fronty, které má nějaký dříve přidaný prvek za výstup.

Fronta potom dodržuje to, že nejdříve přidaný prvek také jako první frontu opustí (FIFO – first in first out). Zásobník dodržuje to, že prvek opouštějící jej, je prvek přidaný jako poslední (LIFO – last in first out).

Přidání prvku do fronty se obvykle označuje jako `enqueue`, odebrání z fronty jako `dequeue`. Přidání prvku na zásobník se obvykle označuje jako `push`, odebrání ze zásobníku jako `pop`.

Mějme nějaký zásobník. Příkaz `push(a)` přidá prvek `a` na zásobník, příkaz `pop()` odebere prvek ze zásobníku a vypíše ho na výstup. jak bude vypadat výstup zadáme-li příkazy:

`push(1), push(2), pop(), push(3), pop(), pop().`

Nepřítel nám teď nabídne takový zásobník. Dovolí nám napsat libovolnou posloupnost příkazů `push` a `pop`. Vymíní si ale, že vstupy k příkazům `pop` doplní sám a sice tak, že to budou přirozená čísla postupně. (Takže například předchozí příklad mohl klidně vzniknout z takového dopnění). Existuje taková posloupnost příkazů `push` a `pop`, aby při daném doplnění vstupů nepřítelem výstupem byla posloupnost

(a) `1 2 3 4 5 6 8 7`

(b) `2 5 4 3 7 6 8 1`

Je možné získat na výstupu libovlnou permutaci? Jak se úloha změní, dostaneme-li frontu a příkazy `enqueue` a `dequeue`?

## Stromy

Reprezentujme stromy jako v materiálu k přednášce č.6. Tedy vrchol stromu je objekt třídy

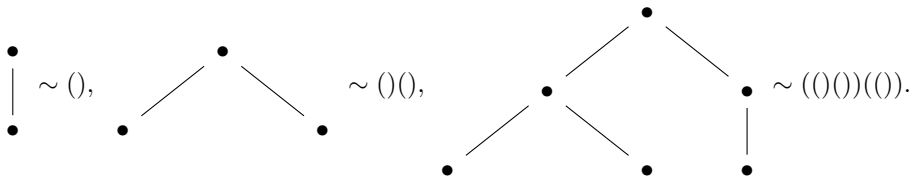
```
class vrchol():
   def __init__(self,x = None):
      self.info = x     \# záznam v uzlu
      self.levy = None  \# levý syn
      self.pravy = None \# pravý syn
```

### Závorkové kódování

Zakořeněný (binární) strom lze zakódovat jako uzávorkování následovně.

- Strom s jedním vrcholem je zakódován jako prázdný řetězec.
- Mějme strom, jehož kořen má jedinného syna, buď zakódování stromu zakořeněného v tomto synu $s$. Celý strom je pak zakódován jako $(s)$.
- Mějme strom, jehož kořen má dva syny. Zakódování levého syna je $l$, zakódování pravého syna je $r$. Celý strom je zakódován jako $(l)(r)$

například:




**Úloha.** Navrhněte algortimus (= popište v rozumném pseoudokódu nebo Pythonu), který dostane na vstupu řetězec závorek a vrátí strom odpovídající tomuto řetězci – postaví takový strom z vrcholů a vrátí kořen. Má-li nějaký vrchol jen jednoho syna, můžeme předpokládat, že je to syn levý.

**Nápověda:** Použijte zásobník, rozmyslete si, co v tomto zápisu znamená levá respektive pravá závorka pro stavění.

### Prohledávání

Několik jednoduchých úloh na prohledávání.

**Úloha 1.** Navrhněte algoritmus, jehož vstupem bude nějaký strom (tedy dostane za vstup vrchol, který je tohoto stromu kořenem), který vrátí list minimální hloubky – tedy takový vrchol, který nemá syny a ze všech vrcholů, které nemají syny, je nejblíže kořeni.

**Úloha 2.** Navrhněte algoritmus, který dostane za vstup strom a nějakou kýženou hodnotu, vrátí odkaz na vrchol ve stromu s touto hodnotou, v minimální možné hloubce.

**Úloha 3.** Navrhněte algoritmus, který dostane nějaký strom a vrátí jeho hloubku – tedy maximální vzdálenost listu od kořene.

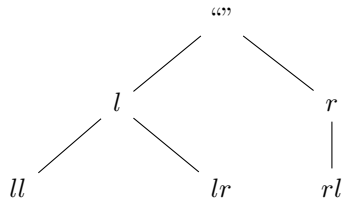
**Nápověda:** Prohledávání do šířky.

### Adresování

Adresa vrcholu v zakořeněném binárním stromě je řetězec daný následovně:

- Adresa kořene je prázdný řetězec.
- Je-li adresa nějakého vrcholu $a$, potom adresa jeho levého syna je $al$ a adresa jeho pravého syna je $ar$.

Například:




**Úloha 1.** Navrhněte algoritmus, který dostane na vstupu nějaký strom a do jeho vrcholů zapíše adresy (můžete předpokládat, že je v každém vrcholu kromě položky `info` i položka `address`).

**Úloha 2.** Navrhněte algoritmus, který dostane na vstupu nějaký strom a nějakou kýženou hodnotu. Vrátí adresu uzlu, ve kterém se tato hodnota nachází.

**Drobnost k zamyšlení.** Jaké budou adresy uzlů které jsou oba v nějakém podstromě? Jaká bude adresa předka vzhledem k adrese potomka. Mám-li adresy dvou uzlů, co o nich mohu zjistit.