**Datové struktury: Spojový seznam, Strom, Fronta, Zásobník, Halda**

*Předmluva: Nijak složitá témata, spojový seznam jsme brali mnohokrát, takže o tom by měl být schopen něco říct každý, to samé binární strom. Držel bych si povídání o každé části tak 3 minuty, nakreslete k tomu obrázky, popište je a jste v pohodě.*

*Jan Kolman*

## **Pole**

Obsahují fixně alokované místo v pěti. List nejde zvětšit ani zmenšit. Po vytvoření do smazání bude zabírat prvně lakovanou velikost. Jedná se o uspořádanou množinu stejných typů objektů.

Jsou indexované od 0 do n-1 (prvků). Protože všechny hodnoty zabírají stejnou velikost index jenom říká, kolik máme přeskočit bitů (hodnot) abychom se dostali k potřebné hodnotě.

## **ArrayList nebo prostě List**

Jeho velikost je dynamická. Struktura je na začátku alokována s defaultní velikostí nebo s velikostí potřebnou pro prvky při inicializace. Poté pokud je potřeba do struktury přidat další prvek a všechna alokovaná místa v paměti jsou již využita list se vezme překopíruje se do na nové místo kde se alokuje navíc místo pro dalších n (např: 10) prvků. To proběhne pokaždé když potřebujeme přidat nový prvek do takto zaplněné struktury. Pokud se z listu prvky odebírají jeho alokovaná paměť se nezmenšuje zůstává stejná.

## **Spojový seznam**

Spojový seznam je dynamická datová struktura, kde každý prvek obsahuje:

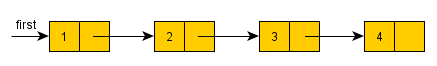
* datovou část
* referenci na následující prvek seznamu
* referenci na předchozí prvek seznamu (u obousměrného seznamu)

**Časová složitost:**

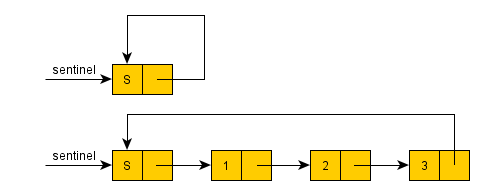
* Najdi prvek - O(n)
* Přidej prvek na začátek - O(1)
* Přidej prvek na konec - O(1) (Pokud si držíme informaci o tom, kde je konec, jinak O(n).
* Přidej prvek někam - O(n)
* Smaž prvek - O(n)
* Smaž začátek - O(1)
* Najdi následníka - O(n)

Rozeznáváme **4 druhy** seznamu:

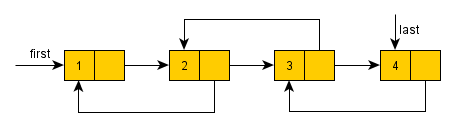
* **Jednosměrně zřetězený seznam** - každý prvek obsahuje právě jednu referenci na následující prvek. Poslední prvek ukazuje na neplatnou adresu. Podle toho při procházení poznáme, že se jedná o poslední prvek.



* **Cyklický jednosměrně zřetězený seznam** - je shodný se seznamen jednosměrně zřetězeným až na to, že poslední prvek má jako následníka prvek první. Tím docílíme toho, že při procházení seznamu nikdy nedojdeme na konec.



* **Obousměrně zřetězený seznam** - každý prvek obsahuje referenci na následující i předchozí prvek. Seznam je tedy možné procházet dvěma směry. Jako následník posledního prvku a předchůdce prvního je nastavena neplatná adresa (NULL), abychom poznali, že jsme na některém konci seznamu.



* **Cyklický obousměrně zřetězený seznam** - následníkem posledního prvku cyklického obousměrně zřetězeného seznamu je prvek první. Předchůdcem prvního prvku prvek poslední.

### **Vlastnosti seznamu**

* Přestože seznam představuje uspořádanou datovou strukturu, nemusí jednotlivé složky ležet v paměti za sebou.
* Seznam může mít tzv. **head** - první prvek seznamu, který obsahuje v datové části nějakou neplatnou hodnotu. Podle této hodnoty poznáme, že se jedná o první prvek.
* Poslední prvek seznamu se někdy označuje jako **tail** a může obsahovat v datové části nějakou neplatnou hodnotu, podle které poznáme, že se jedná o poslední prvek.
* Seznamy obvykle používáme jako dynamické datové struktury. To znamená, že v programu deklarujeme pouze ukazatel na první prvek seznamu a jednotlivé prvky seznamu podle potřeby dynamicky alokujeme nebo rušíme.
* Seznamy se (spolu se stromy) označují jako **rekurzivní datové struktur**y, neboť každý prvek seznamu obsahuje odkaz na položku stejného typu

## **Strom**

Z hlediska teorie [grafů](https://www.algoritmy.net/article/104/graf) je stromem každý souvislý graf bez kružnic o **|U|-1** hranách(**U** je počet prvků ve stromu). Jedná se o hierarchickou strukturu, kde má každý otec 0 až mnoho dětí a každé dítě právě jednoho otce takovým způsobem, že v této struktuře **nejsou cykly**. Uzel, který je **praotcem** všech ostatních uzlů nazýváme **kořenem**. Uzel, který nemá žádné potomky nazýváme **listem**. Být stromem je rekurzivní vlastnost – každý podstrom stromu S je také stromem. Strom je velmi populární pro svou jednoduchost a použitelnost. Příkladem mohou být **vyhledávací stromy** nebo **haldy**.

### **Vlastnosti stromu**

* **N-arita** – Kolik smí mít každý uzel maximálně potomků, z tohoto hlediska patří mezi nejoblíbenější **binární stromy** (každý uzel má 0, 1 nebo 2 potomky).
* **Hloubka** – Hloubkou rozumíme maximální hloubku libovolného uzlu (kořen je v hloubce 0, potomci v hloubce 1, vnuci v hloubce 2...).
* **Pravidelnost** – N-ární strom je pravidelný, pokud má každý uzel **0** nebo **N** potomků.
* **Vyváženost** – N-ární strom je vyvážený, pokud pro všechny listy platí, že nejsou o nic více hlouběji, než kterýkoliv jiný list. Definice „o nic více hlouběji“ se liší v závislosti na konkrétní implementaci.
* **Úplná pravidelnost** – Úplným N-árním pravidelným stromem hloubky k je strom, jehož každý uzel má 0 nebo N potomků a všechny uzly jsou ve hloubce k.

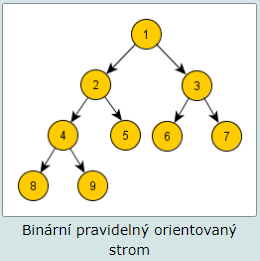
### **Průchody binárním stromem**

* **Preorder** – zpracuje se napřed kořen, poté levý podstrom, a nakonec pravý podstrom (1 -> 2 -> 4 -> 8 -> 9 -> 5 -> 3 -> 6 -> 7)
* **Inorder** – zpracuje se napřed levý podstrom, poté kořen, a nakonec pravý podstrom

(8 -> 4 -> 9 -> 2 -> 5 -> 1 -> 6 -> 3 -> 7)

* **Postorder** – zpracuje se napřed levý podstrom, poté pravý podstrom a nakonec kořen

(8 -> 9 -> 4 -> 5 -> 2 -> 6 -> 7 -> 3 -> 1)



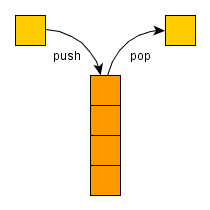
## **Fronta**

Fronta je jedním ze základních datových struktur a slouží k ukládání a výběru dat takovým způsobem, aby prvek, který byl uložen jako první, byl také jako první vybrán. Tomuto principu se říká **FIFO** - *first in, first out*. Speciálním případem fronty je tzv. **prioritní fronta**, ve které mohou prvky s vyšší prioritou předbíhat na výstupu ty s nižší prioritou.

### **Využití**Fronta

* **Operátor pipe** - komunikace mezi procesy v operačních systémech
* **Dialogy ve hrách**
* **Požadavky zasílány tiskárně**

## **Zásobník**

Zásobník (**stack**) je jednou ze základních datových struktur, která se využívá především pro dočasné ukládání dat v průběhu výpočtu. Zásobník data ukládá způsobem, kterému se říká **LIFO** - *last in, first out* - čili poslední vložený prvek jde na výstup jako první, předposlední jako druhý a tak dále.

### **Využití**

Zásobník se v informatice používá zejména pro ukládání stavu algoritmů a programů. Je použit v prohledávání do hloubky a implicitně ve všech rekurzivních algoritmech. Na zásobníkové architektuře jsou postaveny virtuální stroje pro jazyky *Java* a *Lisp*.

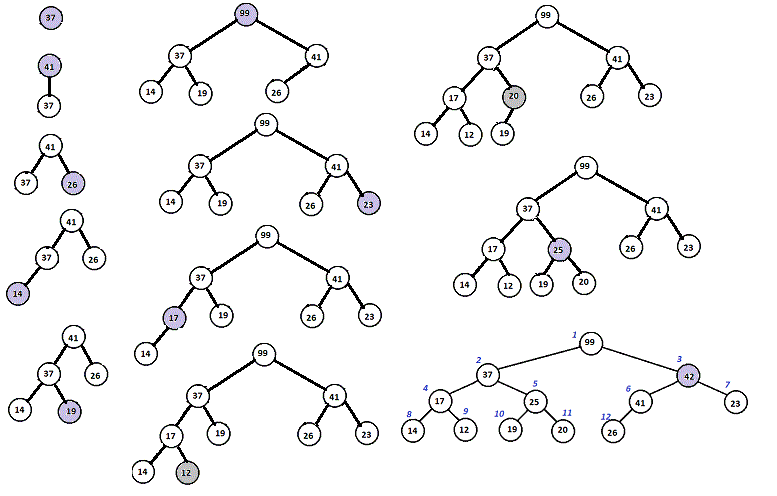
## **Halda**

Binární halda (**binary heap**) je úplný binární strom, ve kterém platí, že každý potomek vrcholku má nižší nebo stejnou hodnotu nežli vrcholek sám, takže kořen je vždy maximum.

Této haldě se říká **Max-Heap**. **Min-Heap** je opačný případ, tedy kořen je vždy minimum a každý potomek vrcholku má stejnou nebo větší hodnotu.V případě, že není poslední hladina haldy zaplněna, jsou uzly ukládány do haldy zleva. Na všech úrovních (s možnou výjimkou té poslední) má tato struktura plný počet uzlů, tedy každý vnitřní vrchol má právě 2 potomky, strom je velmi vyvážený.Poslední patro haldy je zaplněno zleva (může být i zaplněno celé).

Nový prvek se vkládá na **konec haldy**. Pokud to je nutné, bude halda přeuspořádána.

**Ukázka vytvoření haldy v jednotlivých krocích:**

****

**Kruhový buffer:**

Struktura, která je postavena na poli. Vezmeme konec a začátek a spojíme je. Neřeší se zde indexy. Pokud je prázdný head a tail jsou na stejném místě. Jak se vloží jeden prvek do bufferu head se posune o jednu pozici a prvek bude následně na místě tailu. Funguje na principu FIFO.

Výhoda rychlejší všechny jeho operace mají složitost O (1). Hodí se na místa, kde chceme co nejméně realokovat.

Nevýhoda, pokud je pole plné musíme ho stejně jako list vzít a překopírovat do nově vytvořeného většího ring bufferu. Pokud se nejedná o jiný typ, ale v tom případě se budou hodnoty přepisovat.

