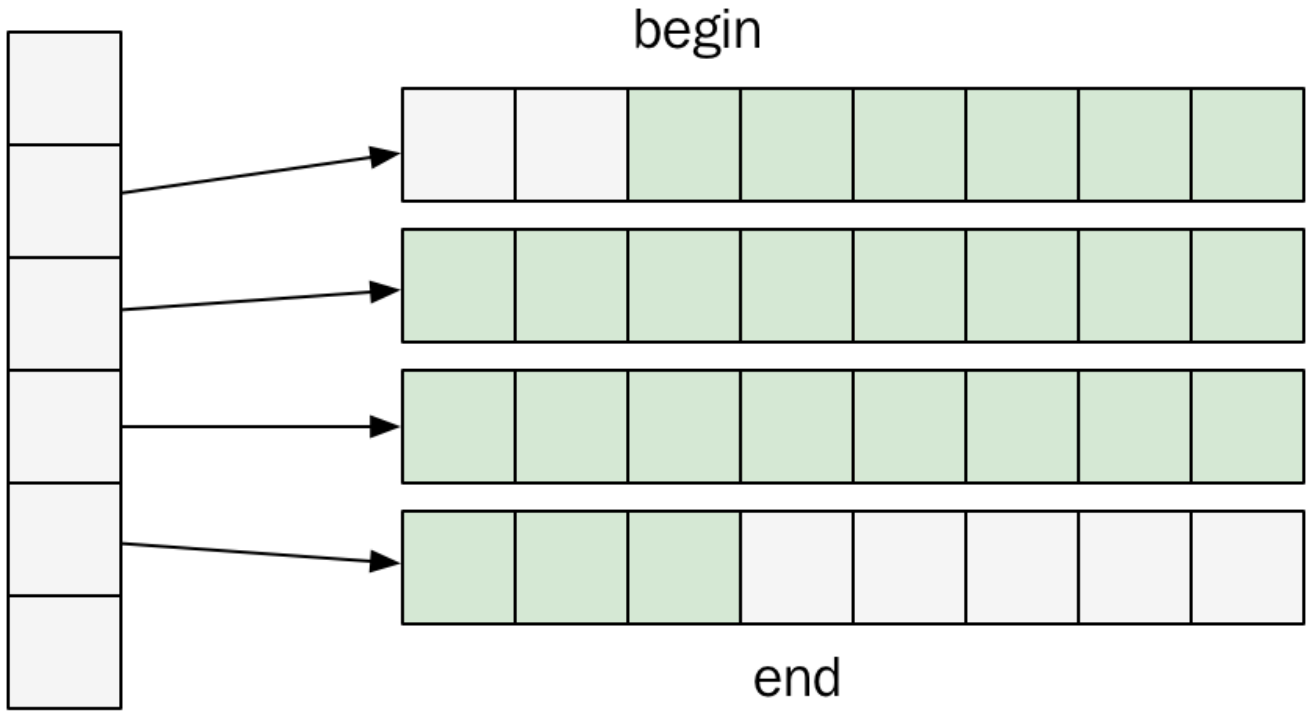
ZALG 2. cvičení

Z minula:

- Spojové seznamy **neumožnují** náhodný přístup (přístup k i-tému prvku) v konstantním čase $O(1)$
- Náhodný přístup v konstantním čase (současně lze k prvkům přistupovat přes indexy) obecně umožňují datové struktury, které ke své reprezentaci využívají dynamické pole – **vektor** a **deque**
- Nevýhodou datových struktur využívající dynamické pole je mazání prvků, které se nenachází po konci polí. Při mazání i-tého prvku vektoru se všechny prvky od indexu i+1 musí posunout o 1 pozici doleva.

std::deque

- Konstantní velikostí chunků

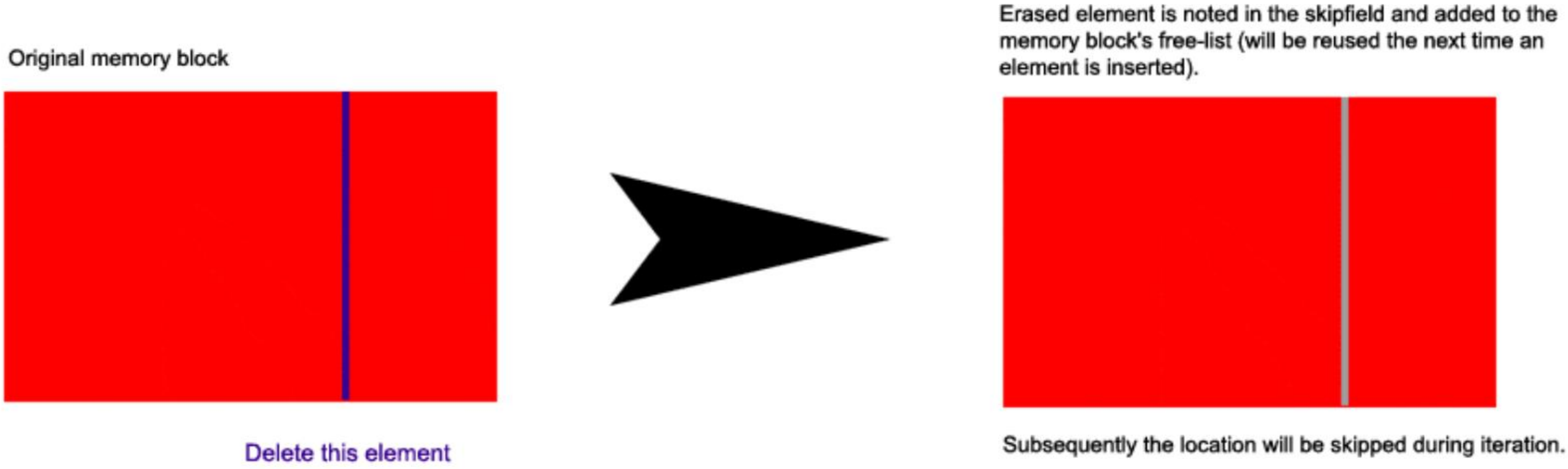


Možné vylepšení: plf::colony

- Místo posouvání prvků v poli, si uchováme v paměti indexy prvků, které byly smazány – použijeme tzv **skipfield**.
- Při iterování danou strukturou, pak dané prvky přeskočíme.
- Naopak při vkládání prvku do struktury nejdříve zaplníme indexy ve skipfieldu a až když bude skipfield prázdný, tak vložíme data na konec struktury.

Zápočtové téma číslo 1 – **plf:colony/std::hive a std::deque**

- („Vektor“ s bloky a skipfieldem a vektor s vkládáním



Zdroj: https://isocpp.org/files/papers/P0447R21.html

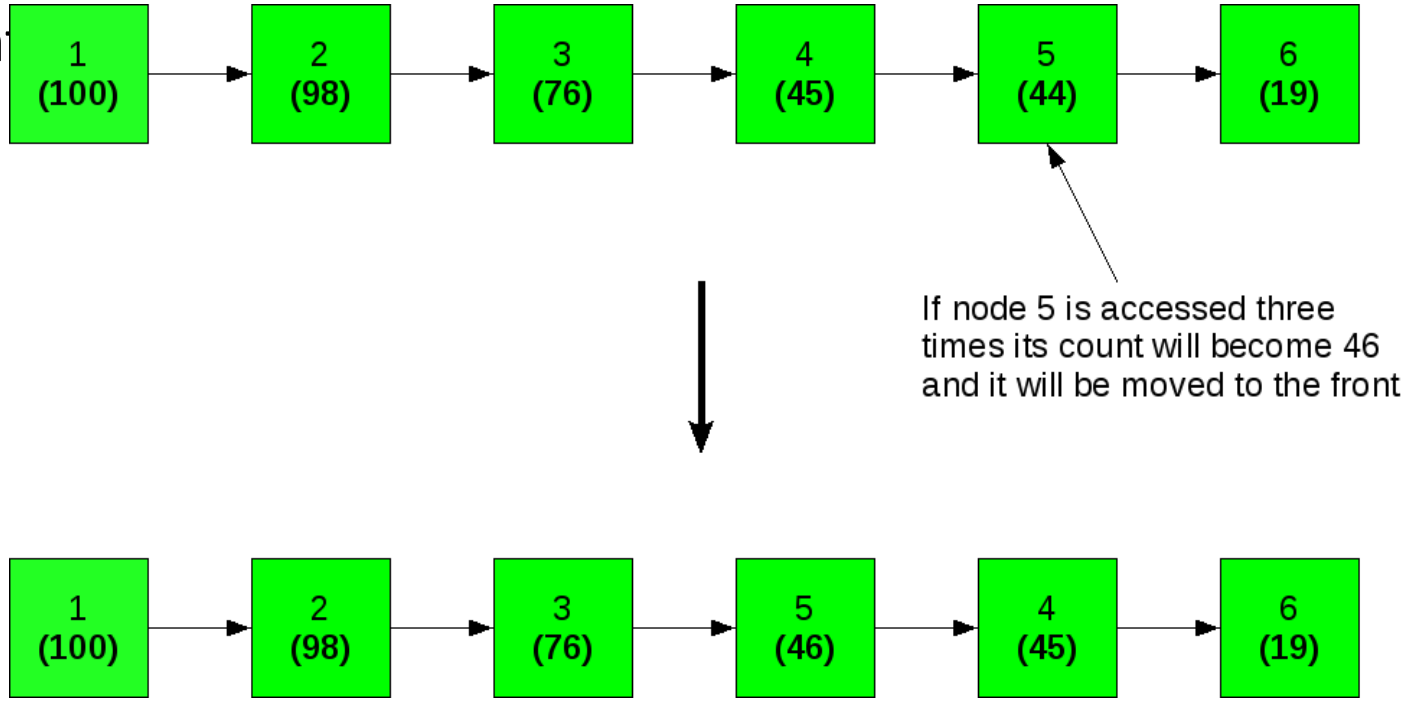
Nevýhody lineárních datových struktur

- Vyhledávání prvku v lineárních datových strukturách $O(\mathrm{n})$
- Mazání prvku v lineární datové struktuře $O(\mathrm{n})$

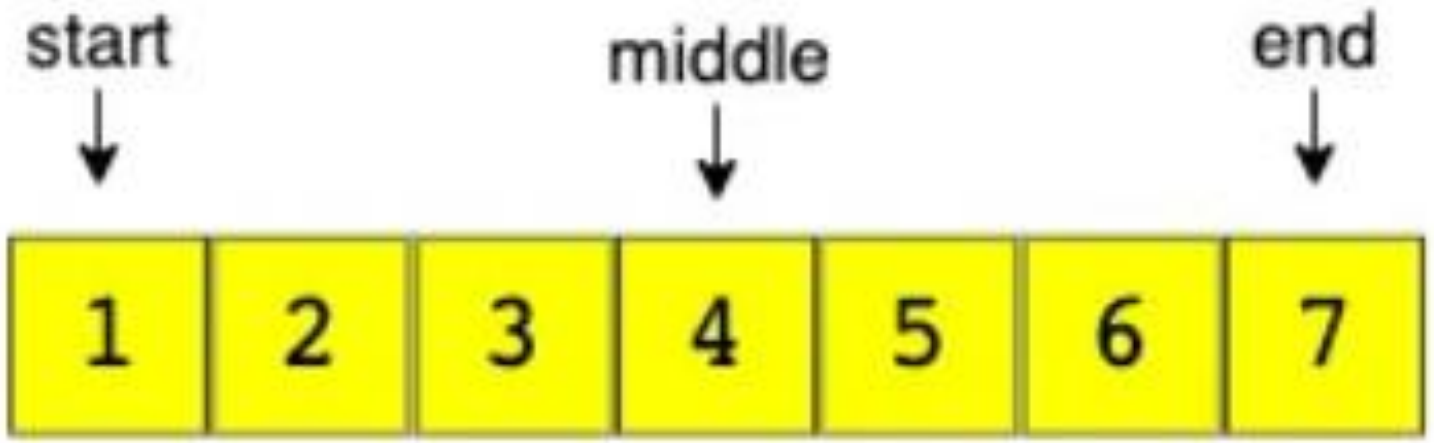
- Pokud se prvek s vyhledávanou hodnotou ve struktuře nenachází, musíme projít celou strukturou, abychom zjistili, že se v ní prvek nenachází

Možné vylepšení: Přeuspořádání podle přístupu

- Self-organizing list
- mění pořadí svých prvků na základě určité samoorganizující se heuristiky
  1. MTF – Move to the fron
  2. Count method
  3. Transpose method



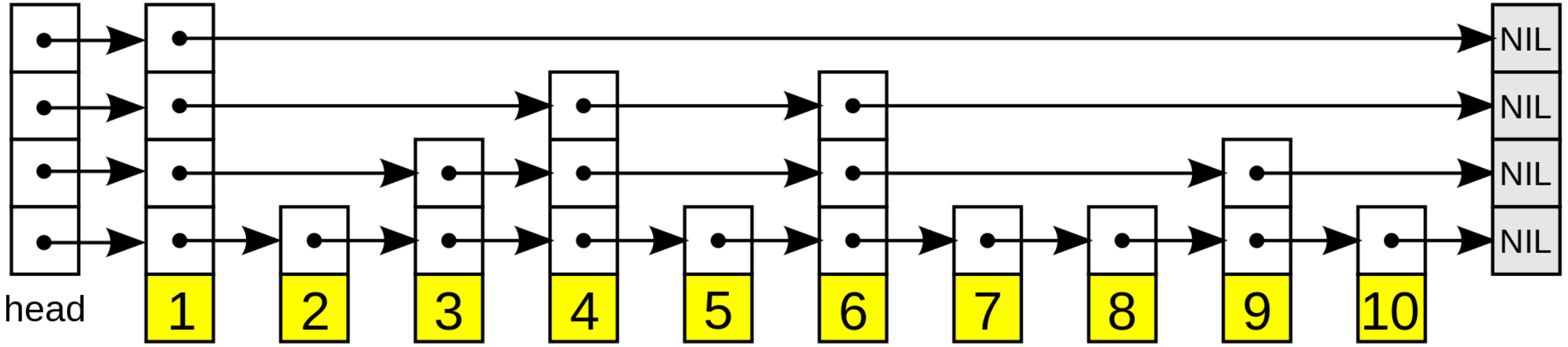
Možné vylepšení: Uspořádání datové struktury



Spojový seznam – složitost operací

	obyčejný			uspořádaný		
	MIN	AVERAGE	MAX	MIN	AVERAGE	MAX
výpis		$O(n)$ vždy			$O(n)$ vždy	
destruktor		$O(n)$ vždy			$O(n)$ vždy	
konstrukor		$O(1)$ vždy			$O(1)$ vždy	
najdi	$O(1)$	$O(n)$ mezi $\frac{n+1}{2}$ a n	$O(n)$	$O(1)$	$O(n)$ o trochu méně	$O(n)$
vlož		$O(1)$ vždy		$O(1)$	$O(n)$ najdi $+$ $O(1)$	$O(n)$
smaž	$O(1)$	$O(n)$ najdi $+$ $O(1)$	$O(n)$	$O(1)$	$O(n)$ najdi $+$ $O(1)$	$O(n)$
postav		$O(n)$ vždy $n*$vlož		O(n)	$O(n^2)$ $n*$vlož	$O(n^2)$

Zápočtové téma číslo 2 – **SkipList a Self-organizing list**



Zdroj: https://www.cs.cmu.edu/~ckingsf/bioinfo-lectures/skiplists.pdf
https://en.wikipedia.org/wiki/Self-organizing_list

Nelineární datové struktury

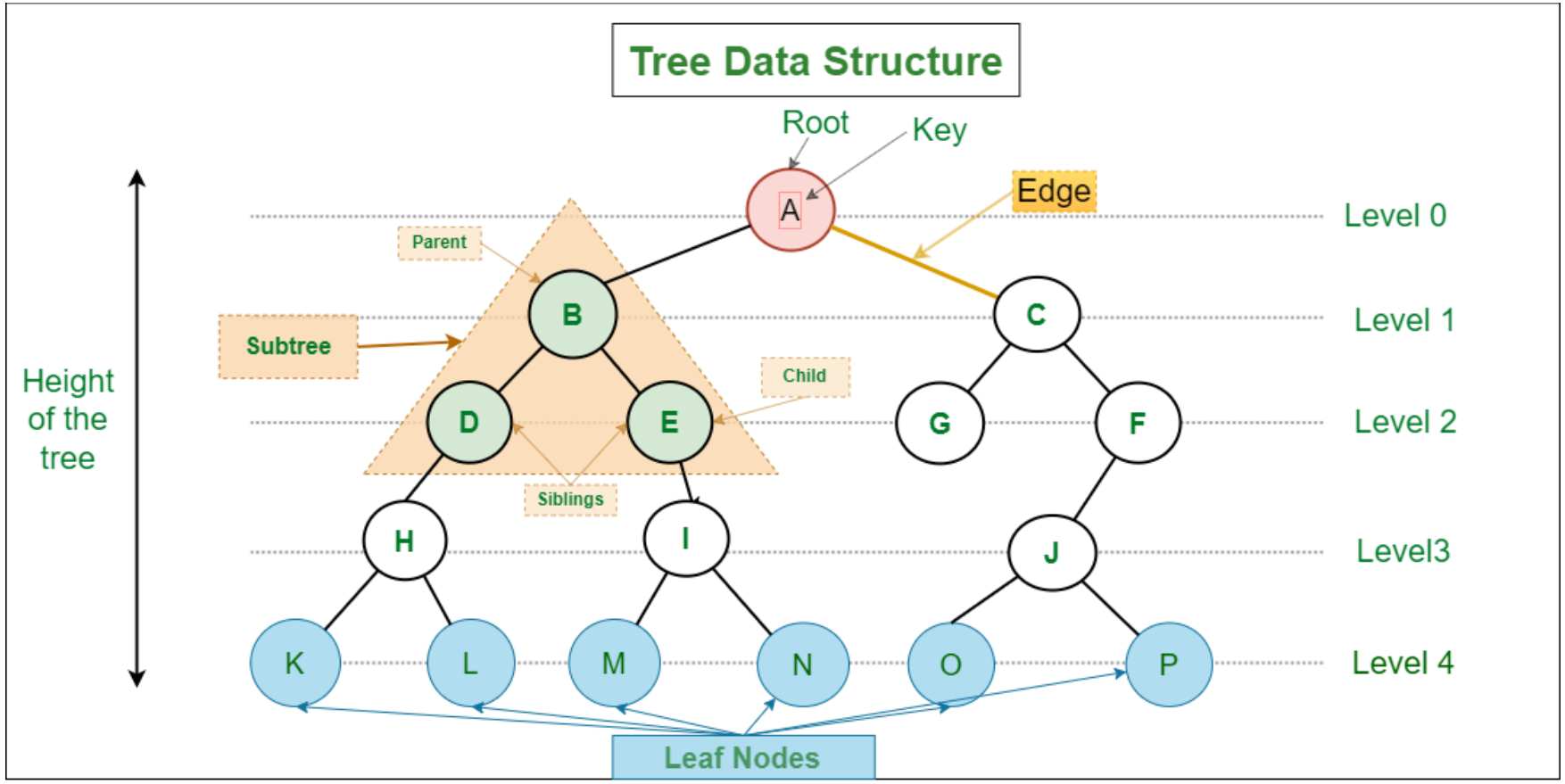
- Stromy
- Hešové tabulky
- Grafy

Stromy

- Abstraktní datový typ představující stromovou strukturu
- d-ární strom je strom, kde k žádnému prvku není připojeno více než d podstromů
- d = 2 binární stromy
- d = 3 ternární stromy

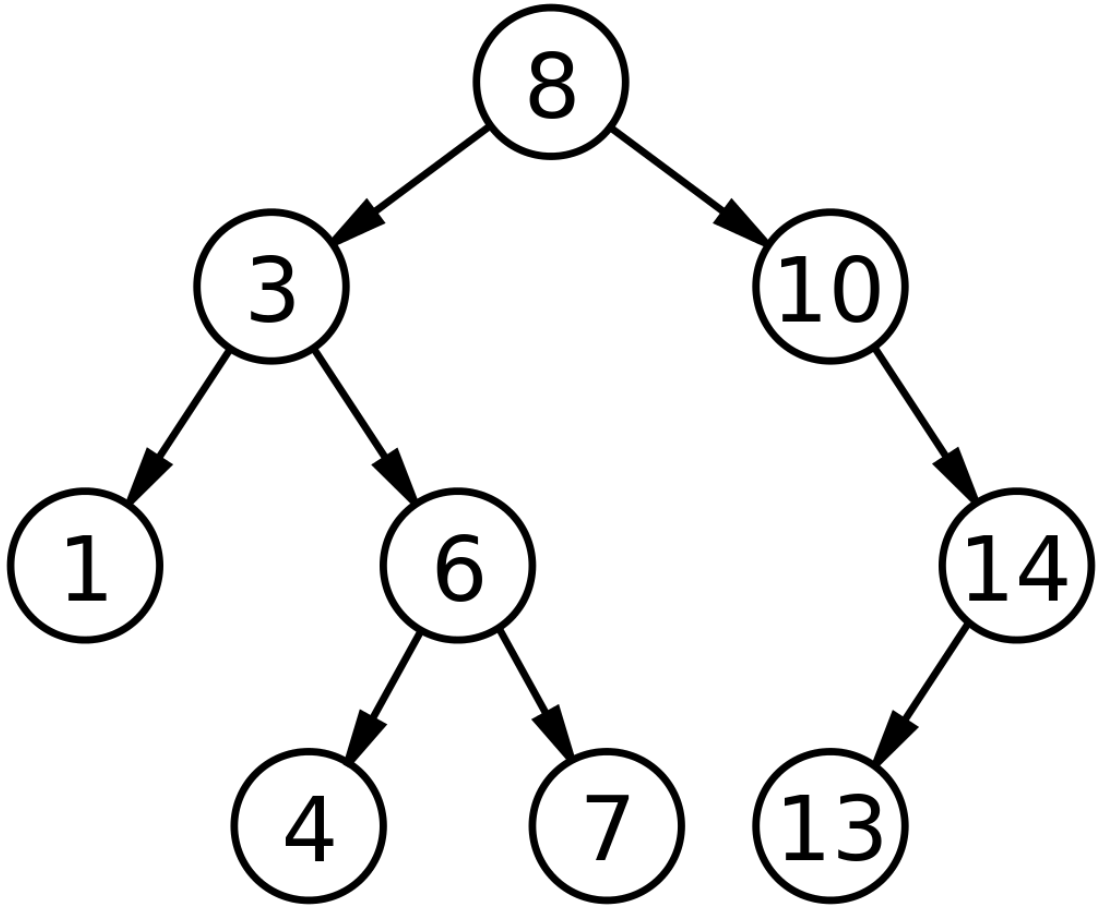
Terminologie

- Vrchol (vertex)
- Kořen (root)
- Rodič (parent)
- Potomek (child)
- List (leaf)
- Úroveň (level)
- Výška (height)



Binární vyhledávací strom

- Binární strom v němž jsou jednotlivé vrcholy uspořádány podle klíče.



Operace nad binárním stromem

	MIN	AVERAGE	MAX
výpis		$O(n)$	
destruktor		$O(n)$ vždy	
konstrukor		$O(1)$ vždy	
najdi	$O(1)$	$O(log_2 n)$ $\sim$ hloubka	$O(n)$
vlož	O(1)	$O(log_2 n)$ najdi $+$ $O(1)$	$O(n)$
smaž	$O(1)$	$O(log_2 n)$ najdi $+$ $O(hloubka)$	$O(n)$
postav	$O(nlog_2 n)$	$O(nlog_2 n)$ $n*$vlož	$O(n^2)$

Implementace binárního stromu

- Třída Strom
  - Atributy:
    - Ukazatel na kořen (root)
  - Metody:
    - Vlož (insert)
    - Najdi (find)
    - Odstraň (remove)
    - Konstruktor
    - Destruktor
    - (Výpis)