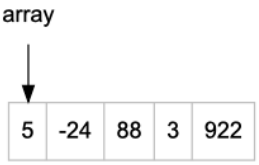
Spojové struktury

Spojové struktury, seznamy, stromy

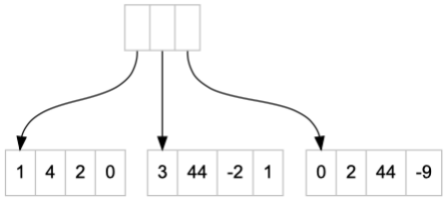
Joseph Soukup – P4

Spojové struktury

- Pole:
  - Nejjednodušší, nejstarší a nejčastěji používaná datová struktura v informatice
  - V paměti je uloženo jako lineárně uspořádaná posloupnost prvků stejného typu, ke kterým lze přistupovat pomocí tzv. indexu
  - Ten lze chápat jako vzdálenost požadovaného prvku od počátku (prvního prvku) pole – tedy souřadnici v jedno rozměrovém prostoru
  - **Jednosměrná pole**
    - Každý prvek je určen právě jedním indexem (souřadnicí)
    - Jednorozměrná pole se nejčastěji používají pro ukládání vektorů (pole čísel) a řetězců (pole znaků)
    - 

    - **Situace v paměti**

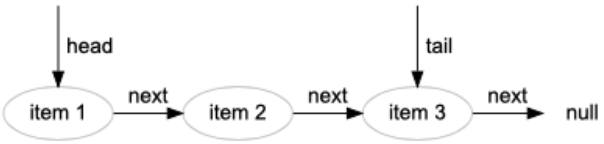
Adresa paměťové buňky	Index hodnoty	Hodnota
proměnná POLE		ukazatel na 0x01A2
...	...	...
0x01A2	POLE+0	5
0x01A3	POLE+1	-24
0x01A4	POLE+2	88
0x01A5	POLE+3	3
0x01A6	POLE+4	922
...	...	...

    - Pole mají zpravidla pevně danou velikost
    - Při nedostatečné kapacitě pole požádá program operační systém o větší (resp. menší) volný prostor (nemusí být k dispozici), do kterého kopíruje stávající data
    - Původní pole se následně uvolní
  - **Vícerozměrná pole**
    - Matice a tabulky jsou příkladem polí dvojrozměrných
    - Ukázka jednotkové matice 3x4, uložené v „poli polí“
    - 

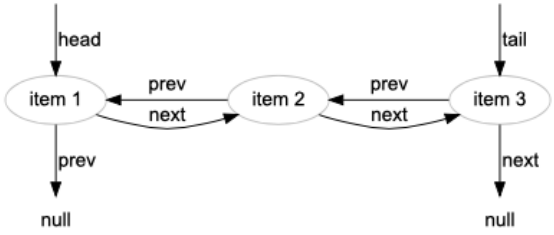
Situace v paměti

Adresa paměťové buňky	Index hodnoty	Hodnota
proměnná POLE		ukazatel na 0x5A01
...	...	...
0x5A01	POLE+0	ukazatel na 0x6122
0x5A02	POLE+1	ukazatel na 0x6132
0x5A03	POLE+2	ukazatel na 0x610A
...	...	...
0x610A	POLE[2]+0	0
0x610B	POLE[2]+1	0
0x610C	POLE[2]+2	1
...	...	...
0x6122	POLE[0]+0	1
0x6123	POLE[0]+1	0
0x6124	POLE[0]+2	0
...	...	...
0x6132	POLE[1]+0	0
0x6133	POLE[1]+1	1
0x6134	POLE[1]+2	0
...	...	...

- Spojový seznam:
  - Získaní reference či adresu nějakého objektu a tuto adresu v programu používat a předávat jako běžný datový typ, například celé číslo -> **ukazatel**
  - Ukazatel lze rozšířit tak, že umožníme, aby mohl nabývat i speciální hodnoty NULL, která indikuje, že na žádný objekt neukazuje
  - Prvek spojového seznamu tedy „obaluje“ datovou hodnotu informacemi o jejím relativním umístění vůči ostatním prvkům a bez datové hodnoty by asi nemělo vůbec smysl nějaký seznam dělat
  - Při práci se strukturou (přidávání a odebírání) se hodnotami nemusíme vůbec zabývat
  - Vstupním bodem je ukazatel na jeho první prvek, který se nazývá **hlavička (head)**
  - Prázdný seznam = head ukazuje do prázdna
  - Modifikace:
    - Pro snadné přidávání je vhodné přidat ukazatel na konec (tail)
    - Pro obousměrný průchod lze prvky zřetězit i zpětně, takže kromě ukazatele (next) bude prvek ukazovat i na předchozí (prev)
  - **Jednosměrný zřetězené seznamy**



  - **Dvousměrně zřetězené seznamy**



- **Cyklicky zřetězený seznam**
  - Poslední prvek ukazuje na první

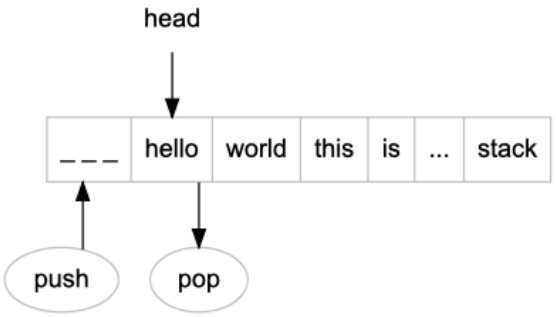
- Složitost:

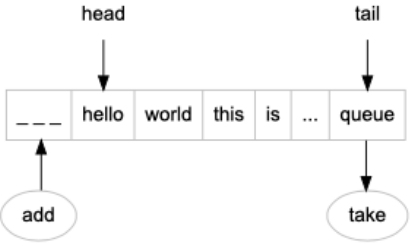
Typ seznamu	Jednosměrně zřetězený	Obousměrně zřetězený
přidání prvku	$O(1)$	$O(1)$
mazání prvku	$O(1)$	$O(1)$
indexace (náhodný přístup k prvku č. `i`)	$O(n)$	$O(n)$
vyhledávání	$O(n)$	$O(n)$

- Výhody:
  - Neomezená kapacita
  - Velikost obsazené paměti je přímo závislá jen na počtu prvků
  - Rychlost přidávání i odebírání prvků je vždy stejně vysoká
- Nevýhoda:
  - Pomalý přístup k prvkům na zadaném indexu
  - Uložené hodnoty nejsou v paměti uložené za sebou
  - Pomalejší procházení
  - Stejné množství dat zabírá více paměti

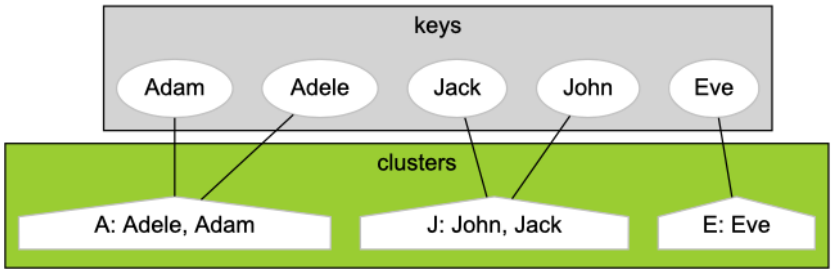
Zásobník

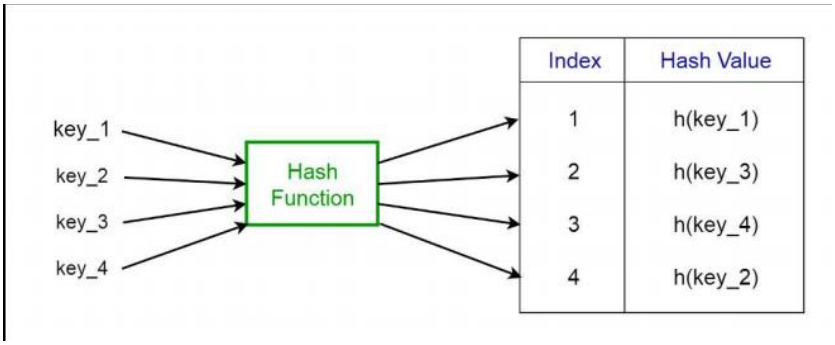
- Datová struktura určená pro ukládání prvků a jejich opětovný výběr v opačném pořadí – poslední přidaný prvek je vybrán jako první (**LIFO –** last in first out)
- Typickým zásobníkem je například:
  - hromada talířů
  - knihy poskládaný na sebe
- Jedna z nejdůležitějších datových struktur v informatice
- Používá se pro:
  - Rekurzi
  - volání podprogramů
  - výpočty matematických výrazů
  - překlad formálních jazyků
  - zpracovávání procesorových instrukcí

- Na všech možných úrovních abstrakce – od fyzické implementace na úrovní hardwarem až po vysoce abstraktní implementace v aplikačních programovacích jazycích.
- Poskytuje tyto funkce:
  - Push(a)
    - Vloží prvek a na vrchol zásobníku
  - Pop()
    - Vrátí vrchol zásobníku a vrchol odebere
  - Peek()
    - Vrátí vrchol zásobníku (beze změny)
  - 

- Fronta
  - Lineární datová struktura určená pro ukládání prvků a jejich opětovný výběr ve stejným pořadí, v jakém byly do fronty přidány – nejstarší přidaný prvek je vybrán jako první (**FIFO** = first in first out)
  - Vyskytují se všude tam, kde je třeba férově vyřešit rozdílné rychlosti vstupů a výstupů, nebo jak odložit zpracování úloh na později, a přitom zachovat jejich pořadí.
  - Funkce:
    - Enqueue(a)
      - Vloží prvek **a** na konec fronty
    - Dequeue()
      - Vrátí první (nejstarší) prvek ze začátku fronty a odebere jej
  - 

- Hashovací tabulka
  - Datová struktura pro ukládání dvojic (klíč, hodnota) nabízející dobrý kompromis mezi rychlostí vyhledávání a paměťovou náročností
  - Vyhledávání podobný jako vyhledávání dokumentů v uklizené kanceláři
  - Pokud chci fakturu, klíčem bude její číslo 20150715
  - Z klíče odvodím, že je to faktura z roku 2015, určitě jí tedy najdu v zásuvce nadepsané „faktury 2015“

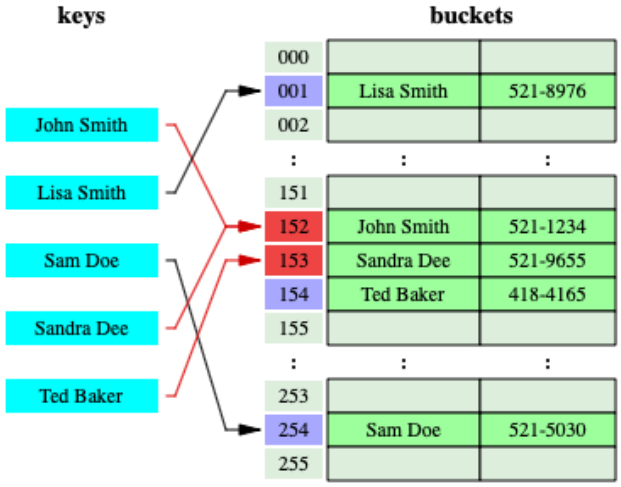
- Hashovací tabulka dělá něco podobného – automaticky pro každý klíč určí jeho kategorii a hledá v přehrádkách, ve kterých by se daný klíč mohl nacházet
- Hashovací tabulka je datová struktura se schopností efektivně vkládat, mazat a hledat datové záznamy **podle klíče**
- **Princip fungování:**
  - Obsahuje pole tzv. **slotů**, do kterých lze ukládat záznamy
  - Dělá jen to, že na základě klíče vybere vhodný slot a operaci provede v něm
  - Abychom docílili efektivity, snažíme se, aby všechny sloty byly využity rovnoměrně, tedy aby různé klíče ideálně padaly do různých slotů
  - Není to možné, protože množina klíčů je mnohem větší než počet slotů
  - Takové situaci říkáme **kolize,** metody, jak kolize řešit:
    - **Zřetězení záznamů:**
      - Každý slot obsahuje spojový seznam, do kterého se postupně řetězí prvky patřící do stejného slotu
    - **Otevřená adresace:**
      - Obsah všech slotů je umístěn v jednom poli a tak mohou data z jednoho slotu „přetékat“ i do jiných slotů a tím zabírat volné místo pro jejich budoucí prvky, což se minimalizuje různými dalšími technikami (leniar probing, double hashing)
  - 

  - 

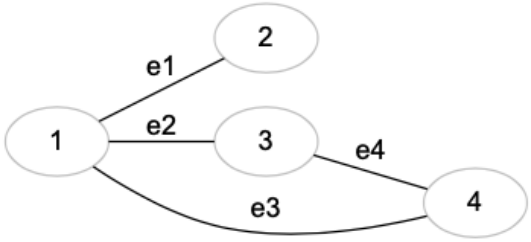
- Převod klíče na index slotu realizuje tzv. **Hashovací funkce**
- Toto zobrazení nemusí být injektivní, ale mělo by mít následující vlastnosti:
  - Vracet různé sloty s rovnoměrnou pravděpodobností
  - Vylmi rychle vypočitatelné
- Asymptotická složitost:

Operace	Typický případ	Nejhorší případ
vyhledávání podle klíče	$O(1)$	$O(n)$
vkládání záznamu	$O(1)$	$O(n)$
mazání záznamu	$O(1)$	$O(n)$

- Implementace:
  - Pomocí pole:

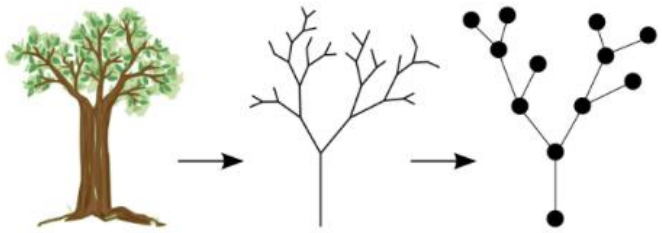


- Graf
  - Způsobu pro reprezentaci grafu v paměti je několik a každá má své výhody a nevýhody
  - Reprezentaci je nutné vždy pečlivě vybrat pro zadanou situaci na základě kritérií:
    - **Druh grafu –** orientovaný a neorientovaný graf, multigraf, atd.
    - **Hustota grafu –** jiné reprezentace jsou vhodnější pro husté grafy, jiné pro řídké grafy
    - **Dynamika grafu –** některé reprezentace jsou vhodnější pro reprezentaci statických grafů (v čase se nemění), jiné pro reprezentaci dynamických grafů (v čase se mění)
    - **Velikost grafu –** některé reprezentace jsou paměťově úspornější než jiné
    - **Režie –** ke každé reprezentaci se vztahuje určitá režie
  - **Reprezentace v paměti**
    - Všechny reprezentace budeme demonstrovat na následujícím grafu:



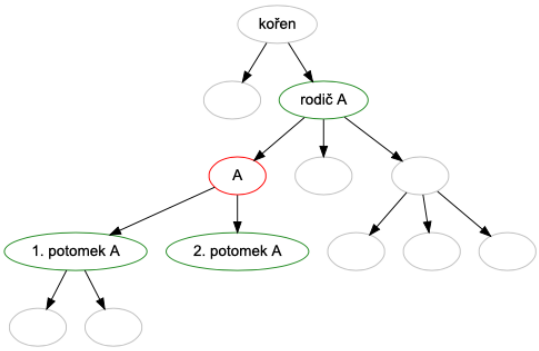
- **Matice incidence (incidence matrix)**
  - V uzly a E hranami -> Matice o velikosti V x E
  - $\begin{pmatrix} 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{pmatrix}$
  - Vlastnosti:
    - Efektivní reprezentace hustých grafů
    - Rychlé získaní sousedících uzlů
    - Složitější přidávání / odebírání v případě, že měníme velikost matice
    - V každým sloupci dvě pravdivé hodnoty
- **Matice souvislosti**
  - **V** uzly a N hranami o velikosti V x V
  - $\begin{pmatrix} \square & 1 & 1 & 1 \\ 1 & \square & 0 & 0 \\ 1 & 0 & \square & 1 \\ 1 & 0 & 1 & \square \end{pmatrix}$
  - Vlastnosti:
    - Efektivní reprezentace hustých grafů
    - Rychlé získaní sousedících uzlů
    - Složitější přidávání / odebírání v případě, že měníme velikost matice
    - Diagonála se ignoruje u grafů, které neumožňují smyčky hrany
    - V případě neorientovaných grafů je matice symetrická podél diagonály
  - Rozšíření:
    - Orientované grafy: matice nebude symetrická podél diagonály a bude zavedeno pořadí indexace
    - Multigrafy: místo pravdivostní hodnoty budou v matici přirozená čísla určující počet hran
    - Vážené hrany: místo pravdivostní hodnoty budou v matici přirozená čísla určující váhu hrany
    - Smyčky: žádnou se číst a používat čísla na diagonále

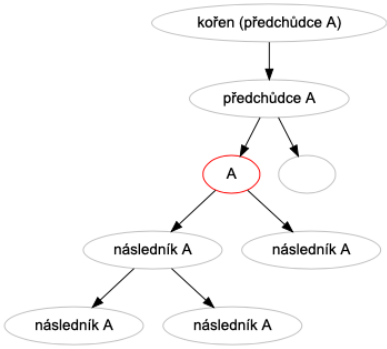
- Strom
  - Souvislé grafy, které neobsahují kružnice
  - Strom je tedy minimální souvislý graf na daných vrcholech
  - Základná koncept **hierarchie**
  - Setkáme se s nimi tam, kde je potřeba rychle vyhledávat, reprezentovat strukturovaná data nebo rozhodovat
  - Strom je přirozený model **rekurze**



- vznik označení "strom"

- Názvosloví:
  - Druhy uzlů:
    - **Kořen** (root)
    - **Vnitřní uzel** (inner node) - uzel, který není kořenem, ani listem
    - **List** (leaf node, external node) - uzel který nemá žádne potomky
    - **Rodič** (parent node) - uzel, který přímo předchází daný uzel na cestě od list uke kořeni
    - **Potomek** (child node) - uzel, který přímo následuje za daným uzlem na cestě od kořenu k listu
    - **Sourozenec** (sibling) - jako sourozenci se označují uzly se stejným rodičem
    - **Předek** (ancestor node, predecessor node) - uzel, který leží před daným uzlem na cestě ke kořeni
    - **Následník** (successor node) - uzel, který leží za daným uzlem na cestě od kořeni k libovolnému listu
    - **Hloubka** (depth) - hloubka stromu je délka nejdelší cesty od kořene k listu, přičemž prázdný strom má definovánu hloubku jako -1
    - **Úroveň** (level) - většinou se používá ve významu manožiny uzlů, které se nachází ve stejné vzdálenosti od kořene, počítano dle počtu uzlů
  - Struktury
    - **Podstrom** (subtree) - podgraf stromu, který je take stromem
    - **Větev** (branch) - každá cesta od kořene k listu





- **Průchody stromem:**
  - Obecně lez stromem, stejně jako kterýmkoliv jiným grafem procházet:
    - Procházení do hloubky
    - Procházení do šířky
  - Specifické pro binární stromy
    - Průchod **pre-order**: zpracuj uzel, zpracuj levý podstrom, zpracuj pravý podstrom
    - Průchod **in-order:** zpracuj levý podstrom, zpracuj uzel, zpracuj pravý podstrom
    - Průchod **post-order:** zpracuj levý podstrom, zpracuj pravý podstrom, zpracuj uzel
    - **Eulerovský průchod:** zpracuj levy podstrom, eulerovsky projdi levy podstrom, zpracuj uzel, eulerovsky projdi pravý podstrom, zparcuj pravý podstrom (každý uzel je navštíven 3x)
- **Druhy stromů**
  - **Binární strom –** strom, ve kterém má každý uzel nejvýše dva potomky—
—
  - **N-ární strom –** strom, ve kterém ná každý uzel nejvýše n potomků
  - **Úplný binární strom –** binární strom, ve kterém jsou zaplněné všechny úrovně kromě poslední, která vyplněná být nemusí
  - **Plný binární strom –** binární strom, ve kterém mají všechny uzly kromě listů mají právě dva potomky