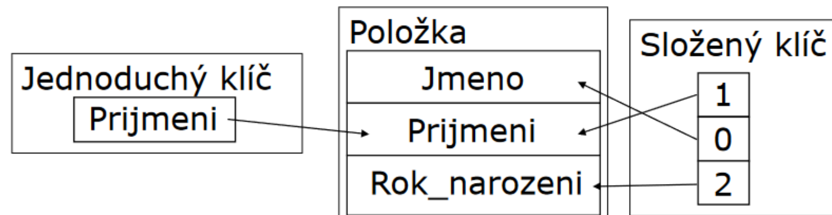
Algoritmy pro vyhledávání a řazení

- **Algoritmy pro vyhledávání**
  - **Vyhledávání (searching)**
    - Algoritmy pro co nejefektivnější nalezení hledané informace.
    - Ne vždy lze v datech vyhledávat efektivně.
    - Rychlé vyhledávání ➔ nutno podniknout jisté kroky už při ukládání dat.
    - Někdy nazýváno Ukládání a vyhledávání dat (Data Storage and Retrieval)
  - **Klasifikace vyhledávacích algoritmů**
    - **Podle místa uložení dat**
      - **Interní –** data jsou uložena v operační paměti
      - **Externí –** data jsou uložena na disku nebo v jiné externí paměti
      - **Kombinované –** nalezení bloku dat na disku a dohledání v rámci nalezeného bloku v operační paměti
    - **Podle principu vyhledávání**
      - **Sekvenční –** sekvenčně se prochází prohledávaná struktura,
      - **Indexsekvenční** – přímým přístupem se nalezne blok a v něm se sekvenčně dohledá (ISAM - Indexed Sequential Access Method) [on line, cit. 2020-11-12]
        - **Vyhledávání v tabulkách s rozptýlenými** položkami (hash table) – index hledaného prvku se „vypočte" speciální funkcí.
      - **Nesekvenční v seřazeném poli** – index prvku se nalezne rychleji než sekvenčním průchodem,
      - **V seřazených stromech** – speciálně organizované struktury pro uložení prohledávaných dat,
        - Stromy jsou struktury následujících datových typů:
        -
```
struct t {
    int value;
    struct t *first;
    struct t *second;
};
```
    - **Podle práce s klíči**
      - **Klíč** = část prvku, podle kterého se vyhledává
        - Index, příjmení, rodné číslo, SPZ, výška, …
      - **Původní** klíče
        - Vyhledává se přímo podle hodnoty klíče
      - **Transformované** klíče
        - Původní klíč → úprava → transformovaný ➔ vyšší efektivita vyhledávání.
        - Vedou k tabulkám s rozptýlenými položkami.
    - **Jednorozměrné vyhledávání**
      - **Adresní**
        - Využívá jednoznačný vztah mezi hodnotou klíče a jeho umístěním ve vyhledávacím prostoru S.
        - Přímý přístup k prvkům pole pomocí indexů.

- **Asociativní**
  - Porovnávání prvků
  - Při hledání prvku $x \in S$ se využívají relace (porovnávání) mezi prvky prostoru S
  - Mezi prvky musí existovat relace uspořádání.
  - Většinou požadujeme, aby byl prostor prvků uspořádán podle této relace

- **Vícerozměrné vyhledávání**
  - Zároveň podle více klíčů = **složený klíč.**
  - Nutno přizpůsobit datové struktury i algoritmy.
  - Dotazy lze rozdělit do tří kategorií
    - Dotazy na **úplnou shodu** – požaduje se shoda na všech klíčích.
    - Dotazy na **částečnou shodu** – požaduje se shoda jen na některých složkách klíče.
    - Dotazy na **intervalovou shodu** – požaduje se, aby hledaný klíč ležel v určeném intervalu.
- **Složený klíč**
  - Více jednoduchých klíčů
  - Každá složka může obsahovat data různých datových typů
  - Porovnávání = zřetězení porovnávacích operací nad jednotlivými složkami ve správném pořadí.
  - 

- **Sekvenční vyhledávání**
  - Vyhledávání prvku se zadaným klíčem znamená postupné prohledání – **sekvenční zpracování**.
  - Nechť je dán datový typ:
  -
```
typedef struct tpolozka
{
  typData data;
  typKlice klic;
} TPolozka;
```
  - Nad typem **typklice** je definována relace ekvivalence.
  - Dále budeme používat funkce **isequal** a **isless** vracející hodnotu typu **bool**.
  - Poznámka 1
    - Porovnávání určitých typů dat není triviální operace, zvláště při použití složených klíčů.
  - Poznámka 2
    - Struktura tpolozka obsahuje složku klic pouze pro názornost.
    - V praktických aplikacích většinou tvoří klíč samotné datové složky struktury.
  - Další deklarace
    -
```
TPolozka pole[N]; // tabulka
typKlice k;       // hledaný klíč
```

Příklad: algoritmus sekvenčního vyhledávání v poli.

```
int i=0;
while (i<N && !isEqual(pole[i].klic, k))
  i++;
bool found = (i < N);
if (found) ... //prvek pole[i] je hledaným prvkem
```

- **Sekvenční vyhledávání v poli se zarážkou**
  - Jednoduchou úpravou lze tento algoritmus zrychlit zjednodušením podmínky pro konec cyklu,
  - Pole vytvoříme o jeden prvek delší – na konec se vloží hledaný klíč – zarážka,
  - Cyklus skončí vždy „úspěšným vyhledáním“, ke kterému dojde buď až na zarážce (neúspěšné vyhledání) nebo dříve (úspěšné vyhledání)

Příklad: sekvenční vyhledávání v poli se zarážkou.

```
TPolozka pole[N+1];
...
pole[N].klic = k; // vložení zarážky
int i=0;
while(!isEqual(pole[i].klic, k))
  i++;
bool found = (i != N);
if (found) ... //prvek pole[i] je hledaným prvkem
```

Poznámka: při rušení položky z neseřazeného pole není nutné posouvat všechny prvky – stačí přesunout poslední prvek na místo rušeného prvku a snížit velikost pole.

Příklad: sekvenční vyhledávání v seřazeném poli.

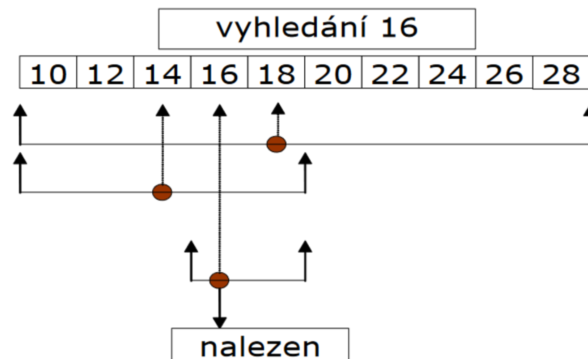
```
int i=0;
while (i<N && isLess(pole[i].klic, k))
  i++;
bool found = (i<N) && isEqual(pole[i].klic, k);
if (found) ... //prvek pole[i] je hledaným prvkem
```

- **Dynamické vlastnosti vyhledávání v seřazeném poli**
  - Při vkládání a rušení prvku nutno zachovat uspořádanost – může jít o „drahou“ operaci.
  - Do určitého počtu nových prvků je lepší přidávat na konec a neuspořádanou část projít sekvenčně.

- **Nesekvenční vyhledávání**
  - **Binární vyhledávání**
    - Nechť pro pole implementující vyhledávací tabulku platí:
      - Pole[0].klic < pole[1].klic < … < pole[N-1].klic,
    - Dále nechť pro vyhledávaný klíč k platí:
      - K >= pole[0].klic a k <= pole[N-1].klic
    - Poznámka
      - Zde uváděné porovnávání – pouze zjednodušující abstrakce

- **Princip binárního vyhledávání:**
    - Vyhledávaný klíč se porovná s klíčem položky, která je umístěna v polovině vyhledávaného pole,
    - Dojde-li ke shodě, končí vyhledávání úspěšně,
    - Je-li vyhledávaný klíč menší, postupuje se porovnáváním prostředního prvku v levé polovině původního pole,
    - Je-li větší v pravé polovině původního pole,
    - Vyhledávání končí neúspěchem v případě, že prohledávaná část pole je prázdná (její levý index je větší než pravý).

Příklad: binární vyhledávání v seřazeném poli.



Příklad: binární vyhledávání v seřazeném poli.

```
int search(Tpolozka a[], typKlice k, int l, int r)
{
  while (r >= l)
  {
    int m = (l + r) / 2;
    if (isEqual(k, a[m].klic)) return m;
    if (isLess(k, a[m].klic))
      r = m - 1;
    else
      l = m + 1;
  }
  return -1;  // nenalezen
}
```

- Binární vyhledávání
    - Délka prohledávané posloupnosti se po každém porovnání zmenší na polovinu.
    - Složitost metody: O(log2(N))

- **Algoritmy pro řazení**
    - Vlastnosti metod řazení.
    - Problematika porovnání český řetězců.
    - Řazení podle více klíčů.
    - Řazení bez přesunu položek.
    - Metoda přímého výběru.
    - Metoda bublinového řazení.
    - Metoda přímého vkládání.

- **Obecná formulace problému řazení**
  - Je dána neprázdná množina
    - $A = \{a_1, a_2, \ldots, a_n\}$,
  - Je potřeba najít permutaci π těchto n prvků, která zobrazuje danou posloupnost do neklesající (nerostoucí) posloupnosti:

$$a_{\pi(1)}, a_{\pi(2)}, \ldots, a_{\pi(n)}$$

  - tak, že: $a_{\pi(1)} \leq a_{\pi(2)} \leq \ldots \leq a_{\pi(n)}$,
  - Množinu U, ze které vybíráme prvky tříděné množiny, nazýváme **univerzum**
  - Prvky množiny **U** se nazývají **klíče** a informace vázaná na klíč spolu s klíčem se nazývá **záznam**
  - Jestliže je velikost vázané informace – záznamů příliš velká je výhodnější seřadit jen klíče s patřičnými odkazy na záznamy, které se v tomto případě nepřesouvají.
  - Bez újmy na obecnosti budeme dále předpokládat, že řadíme pouze klíče
- **Vlastnosti metod řazení**
  - **Sekvenční řazení**
    - **Sekvenčnost** je vlastnost, která vyjadřuje, že řadící algoritmus pracuje se vstupními údaji i s datovými meziprodukty v tom pořadí v jakém jsou lineárně uspořádány v datové struktuře
    - Příklad: lineárně vázaný seznam
  - **Nesekvenční řazení**
    - Opakem sekvenčnosti je **přímý přístup** k jednotlivým položkám vstupní nebo pomocné struktury.
    - Příklad: pole
  - **Přirozenost**
    - T(seřazená) < T(náhodně uspořádaná) < T(opačně uspořádaná)
    - T – doba potřebná pro seřazení posloupnosti
  - **Stabilita**
    - Algoritmus zachovává relativní uspořádání duplicitních klíčů souboru se shodnými klíči.
    - Nutná tam, kde řazení údajů podle klíče s vyšší prioritou nesmí porušit pořadí údajů se shodnými klíči vyšší priority, získané předcházejícím řazením množiny podle klíčů s nižší prioritou.
  - **In situ** – metoda pracuje přímo v místě uložení řazených dat
  - **Složitost**
    - Vyšetřování složitosti se neodlišuje od obvyklých postupů analýzy složitosti.
    - Prostorová a časová složitost označuje míru prostoru, resp. Času potřebnou k realizaci daného algoritmu,
    - U řadicích algoritmů složitost označujeme jako funkci počtu n řazených prvků,
    - Zajímá nás limitní chování složitosti – tzv. **Asymptotická složitost**,
    - Z hlediska prostorové složitosti jsou zajímavé zejména případy, kdy metoda pracuje přímo v místě uložení řazených dat (**in situ**) a nepotřebuje žádný (nebo jen minimální) přídavný prostor.

- **Problematika porovnání českých řetězců**
  - **Normy:**
    - ČSN ISO/IEC 14651- Informační technologie – Mezinárodní řazení a porovnání řetězců – Metoda pro porovnávání znakových řetězců a popis obecné šablony pro přizpůsobení řazení
    - ČSN 97 6030 – Abecední řazení, 1994 – stanoví pravidla uspořádání abecedních sestav všeho druhu, obsahujících česká a cizojazyčná hesla. Dále stanoví zásady pro řazení číslic, různých znamének, značek a obrazců do abecedního řazení.
  - Lexikografické porovnání řetězců
    - Probíhá podle ordinálních čísel znaků
    - Pouze pro angličtinu (bez diakritiky)
    -
      ```
      char str1[20], str2[20];
      ...
      if (strcmp(str1, str2) > 0)
      { ... }
      ...
      ```
    - <0 – str1 je lexikograficky menší než str2
    - =0 – shodné
    - >0 – str1 je lexikograficky větší než str2
  - Znaky s diakritikou mají v běžných variantách kódu ASCII [on line, cit. 2020-11-12] (ASCII - wikipedie) ordinální čísla větší než 127.
    - Příklad českého řazení:
    - Pozor! Písmena s diakritikou:
      základní pořadí: č, ř, š, ž
      řadí se jako bez diakritiky: á,ď,é,ě,í,ň,ó,ť,ú,ů,ý
  - **Základní způsoby řešení:**
    - **konverze** řetězců do váhového kódu a porovnání výsledku normálním operátorem,
    - speciální **porovnávací funkce**.
  - **Knihovní nástroje pro práci s češtinou**
    - **int strcoll(const char *s1, const char *s2);**
      - <string.h>
      - Pro porovnávání lokalizovaných řetězců. Nejdříve je ale nutné nastavit lokalizované prostředí.
    - **char *setlocale(int category, const char *locale);**
      - <locale.h>
      - konstanta: LC_COLLATE
      - Můžete použít pro kontrolu zda jste seřadili vaše data správně česky.

- **Řazení podle více klíčů**
  - **Řešení č. 1**
    - Složená relace uspořádání.
    - Funkce, která bude postupně srovnávat klíče se snižující se prioritou.
    - Nerovnost klíčů vyšší priority ➔ relace je rozhodnuta. Jinak se postupuje k porovnání klíčů s nižší prioritou.
    - **Výhody**
      - **Jednoprůchodové řazení.**
      - **Funguje i pro nestabilní řadící metody**
  - **Řešení č. 2**
    - Postupné řazení podle jednotlivých klíčů.
    - Nutno postupovat od nejmenší priority klíče k vyšší prioritě.
    - Např. seznam podle stáří: nejprve podle DEN, pak podle MESIC, nakonec podle ROK
    - **Výhoda**
      - **Jednodušší algoritmus**
    - **Nevýhody**
      - **Řadící metoda musí být stabilní!**
      - **Víceprůchodové řešení ➔ neefektivní**

- **Řazení bez přesunu položek**
  - Nejvýznamnější operace při řazení
    - porovnání dvou položek a přesun položky (např. výměna dvou položek).
  - Přesun velkých položek ➔ náročné, neefektivní.
  - Řadíme pomoci pole ukazatelů

- **Rozdělení algoritmů řazení**
  - Podle typu paměti, v níž je řazená struktura uložena
    - **Vnitřní** (interní) metody řazení (metody řazení polí) předpokládají uložení seřazované struktury v operační paměti a přímý (nesekvenční) přístup k položkám struktury,
    - **Vnější (**externí) řazení (metody řazení souborů) předpokládají sekvenční přístup k položkám seřazované struktury.
  - **Zvláštní skupina – indexsekvenční struktury** (implementované na discích)
    - Metody řazení často kombinují principy vnitřního i vnějšího řazení
  - **Podle řádu složitosti:**
    - **Od $O(n \log_2 n)$ do $O(n^2)$**
  - **Podle základního principu řazení:**
    - Metody založené na principu **výběru** (selection) – postupný přesun největšího (nejmenšího) prvku do seřazené výstupní struktury,
    - Metody založené na principu **vkládání** (insertion) – postupné zařazování prvku na správné místo ve výstupní struktuře,

- **Metoda přímého výběru**
  - **Algoritmus:**
    - 1. Pole je rozděleno na seřazenou část a neseřazenou část,
    - 2. V neseřazené části se nalezne minimum a vymění se s prvkem bezprostředně následujícím za seřazenou částí a tento prvek se do ní zahrne,
    - 3. Na začátku má seřazená část délku nula, na konci má délku pole

- **Metoda bublinového řazení**
  - **Algoritmus:**
    - 1. Posloupnost rozdělíme na dvě části, setříděnou a nesetříděnou. Setříděná část je prázdná,
    - 2. Postupně porovnáme všechny sousední prvky v nesetříděné části a pokud nejsou v požadovaném pořadí, prohodíme je,
    - 3. Krok 2 opakujeme tak dlouho, dokud nesetříděná část obsahuje více než jeden prvek. Jinak algoritmus končí.