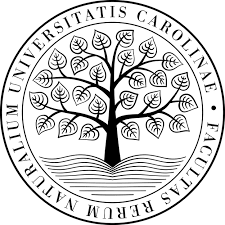
**úvod do programování**

Bubble sort, medián

****

**Jiří Mýl**

**Aplikovaná geografie – Sociální geografie a geoinformatika**

**31.1.2022**

**Bubble sort**

**Třízení/řazení**

Jde proces seřazení neuspořádané množiny prvků. Seřazení probíhá buďto podle číselné posloupnosti nebo abecedně. Prvky mohou být zároveň seřazeny vzestupně nebo sestupně. Řazení jako takové je v programování velmi častou úlohou, a proto existuje velká snaha tento proces co nejvíce zefektivnit. K samotnému třízení se používají tzv. třídící algoritmy (Wikipedie 2022).

**Třídící algoritmy**

Jsou to takové algoritmy, které slouží k setřídění neuspořádané množiny prvků. Třídících algoritmů je celá řada, zásadní rozdíl mezi nimi je především v jejich efektivitě. Efektivita lze určit pomocí asymptotické složitosti O(n). Ta udává vztah mezi dobou běhu algoritmu a velikostí vstupních dat. Většina třídících algoritmů se pohybuje v O(n2) až O(n\*log(n)). Některé algoritmy mohou ve specifickém případu, kdy je vstupní množina prvků již setříděna dosahovat časové složitosti O(n). Algoritmy se dále mohou dělit na vnitřní a vnější, v závislosti na způsobu práce se vstupními daty. Vnitřní algoritmy pracují s daty, které jsou obsaženy uvnitř paměti počítače, je potřeba tedy znát celou množinu předem. Tyto algoritmy mohou přistupovat k prvkům v libovolném pořadí. Vnější algoritmy pracují s externě uloženými daty a počet prvků tak není znám předem. K datům se přistupuje sekvenčně (Programiz).

Další rozdílnost spočívá v práci s prvky se stejnou hodnotou. Pokud algoritmus neprohazuje prvky se stejnou hodnotou říkáme mu stabilní. Takovéto algoritmy jsou zároveň přirozené, což jsou algoritmy, které pracují rychleji s již částečné seřazenou množinou prvků oproti práci s množinou zcela nesetříděnou. Opakem takových algoritmů jsou algoritmy nepřirozené (Algoritmy.net).

Kromě časové náročnosti lze algoritmy porovnávat také pomocí paměťové náročnosti. Ta se liší podle toho, jaké množství paměti je k setřídění množiny prvků potřeba. Pokud je toto množství malé a především konstantní, je algoritmus označován jako tzv. In-place algoritmus. Zpracování dat probíhá v místě, kde jsou uložena. Paměťová náročnost takovýchto algoritmů je O(1). Opakem těchto algoritmů jsou tzv. not-in-place algoritmy (Wikipedie 2022).

**Typy algoritmů**

**Sloužící k představení principů třídění.**

Bubble sort

Selection Sort

Insert Sort

**Používají se v praxi:**

Merge Sort

Quick Sort

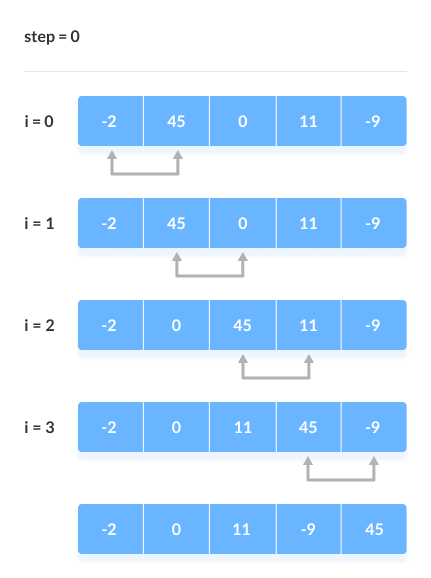
Heap Sort

**Princip bubble sortu**

*Bubble sort* je velmi jednoduchý třídící algoritmus. To se bohužel projevuje na jeho efektivitě, která není dobrá. Používá se především pro představení principu třídění.

Algoritmus funguje na principu porovnávání dvou sousedních prvků, pokud tyto prvky nesplňují třídící pravidlo jsou prohozeny.

Př.: Řazení čísel od nejmenšího po největší: prvek napravo musí být větší než prvek nalevo, jinak dojde k prohození (viz. Obr. 1).



Obr. 1 (Zdroj: Programiz)

Takto algoritmus projde celou množinu prvků tolikrát, kolik je v ní prvků mínus poslední prvek. Tedy dokud není setřízena (Programiz). Ve většině případů však upraveno tak, že algoritmus neprochází již setřízené prvky (viz. kapitola ***Problematická místa***). Konkrétně by tedy další postup při řešení zmíněného příkladu vypadal takto:

Obsah obrázku stůl

Popis byl vytvořen automaticky

Obr. 2 (Zdroj: Programiz)

**Struktura programu**

Program je tvořen částí kódu, která získává od uživatele vstupní data. Podmínku je, že tato data musí být čísla a musí být alespoň dvě, tak aby mělo smysl vstupní data řadit. Konkrétně se tato část kódu skládá z *While loopu*, který neustále běží a uživatele vyzívá k zadání vstupních dat. Tato smyčka je přerušena příkazem *Break*. K němu dojde, pokud vstupní data splní již zmíněné požadavky. Seznam vstupních dat je poté uživateli vypsán do konzole.

Samotný *Bubble sort* je součástí funkce, jehož vstupním parametrem je seznam vstupních dat a tento seznam následně setříděný od nejmenšího prvku k největšímu vrací.

*Bubble sort* skládá ze dvou do sebe vnořených cyklů s konečným počtem opakováním (for-cyklus). Vnitřní cyklus porovnává všechny prvky ze seznamu s prvkem na napravo od něj. A popřípadě je prohodí, pokud je pravý prvek větší než ten levý. Tento cyklus je pak opakován vnějším cyklem tolikrát, kolik je počet prvků v seznamu -1.

Nakonec je pro uživatele vypsán setříděný seznam do konzole.

**Vstupní/výstupní data**

Vstupní daty algoritmu je nesetříděný číselný seznam, který je po spuštění programu zadán uživatelem do konzole.

Výstupními daty je seznam setříděných prvků, který se uživateli vypíše do konzole.

**Problematická místa**

Největším problémem je *Bubble sortu* je jeho neefektivita. Ta jde částečně zlepšit určitými úpravami.

Nejjednodušším vylepšením by bylo zkrácení počtu iterací vnitřního cyklu o již setříděné prvky. Jeho podoba by tedy byla:

*for j in range(lenght-1-****i****):*

Vhodným vylepšením je především vytvoření tzv. *Boolean flag*. Tedy proměnné typu *Bool*, která bude mít v základním stavu hodnotu *False*. A vnější cyklus bude na rozdíl od základní verze algoritmu typu *While*. Bude tedy probíhat, dokud nebude změněna jeho podmínka, tedy hodnota proměnné typu *Bool*. Ta bude změněna, pokud nedojde při průchod seznamem k prohození žádných prvků (seznam je tedy seřazený) a *While-cyklus* bude tím pádem ukončen. To zamezuje zbytečnému procházení již setříděného seznamu (Programiz).

Za určitou formu vylepšení *Bubble sortu* lze považovat také třídící algoritmus *Shaker sort*. Ten na rozdíl od Bubble sortu porovnává hodnoty v seznamu v obou směrech. Tím řeší nedostatek *Bubble sortu*, kdy prvky ze začátku seznamu probublávají na jeho konce poměrně rychle, kdežto probublávání prvků z konce seznamu na jeho začátek je pomalé (tzv. problém zajíců a želv) (Algoritmy.net).

**Medián**

Medián je hodnota středního prvku v uspořádané množině čísel. A rozděluje tak množinu na dvě poloviny, kdy 50% prvků má menší hodnotu než medián a 50% má hodnotu vyšší. Pokud je počet prvků v množině sudý je hodnota mediánu definována jako průměr prvků na pozici *n/2* a *(n/2) + 1*, kdy n je počet prvků v množině (IT SLOVNÍK.cz).

**Algoritmus**

Při hledání mediánu z nesetříděného seznamu, spočívá algoritmus v setřídění seznamu. K setřízení seznamu může být teoreticky použit jakýkoliv třídící algoritmus. Měl by ale být brán ohled na časovou náročnost daného algoritmu. Dále je potřeba zjistit, zda počet prvků v seznamu sudý nebo lichý. A na základě této informace je pak rozhodnuto o způsobu získání mediánu.

Pokud je počet prvků v seznamu lichý, medián je roven hodnotě prostředního prvku *n//2*. Tedy takového prvku, který je na pozici s hodnotou největšího celého čísla po dělení délky seznamu dvěma. A vzhledem k tomu že pozice prvků v seznamu jsou značeny od 0, není přičítána jednička (*3//2 = 1 => druhý prvek v pořadí*). Možný je také zápis *int(n/2)*, který také vrátí největší celé číslo po dělění. Pokud by byl počet prvků v seznamu sudý je potřeba zjistit hodnoty prvků *(n/2) - 1* a *(n/2)* a vypočítat jejich průměr. Zde se je pozice prvků opět ovlivněna tím, že pozice v seznamu jsou značeny od 0 a při výpočtu jsou odečteny jedničky (*n/2 => (n/2) - 1, (n/2) + 1 => n/2*).

**Z toho vyplývá pseudokód:**

zjisti délku seznamu

pokud bude zbytek po vydělení délky seznamu dvěma jiný než nula

medián je rovný hodnotě prostředního prvku

jinak

medián je rovný průměru z hodnot na pozici *(n/2) - 1* a *(n/2)*

**Struktura programu**

Program se skládá, stejně jako program *Bubble sort*, z části, která získává vstupní data od uživatele. Ty jsou pak po jejich verifikaci vypsány do konzole.

Seznam vstupních dat je poté setříděn pomocí funkce *sort()*. A to především vzhledem k tomu, že jde o poměrně efektivní třídící funkci a její použití je zároveň velmi intuitivní a rychlé. Dále je zjištěn počet prvků v seznamu pomocí funkce *len()* a je zjištěno, zda je počet prvků lichý či sudý. To je provedeno pomocí dělení délky seznamu hodnotou 2 s využitím speciálního operátoru %. Ten vrátí zbytek po dělení. Pokud je zbytek nulový, počet prvků v seznamu je sudý. Pokud ne, počet prvků v seznamu je lichý.

V závislosti na tom, zda je počet prvků v seznamu lichý či sudý se pak program větví pomocí funkce *If* na dvě části. Pokud je počet prvků sudý je medián spočítán jako průměr prvků na pozici *(n/2)-1* a *(n/2)*, kdy n je počet prvků. Pokud je počet prvků lichý je medián spočítán jako hodnota prvku na pozici *(n//2)*.

Nakonec je výsledný medián vypsán uživateli do konzole.

**Vstupní/výstupní data**

Vstupními daty je nesetříděný seznam čísel vytvořený uživatelem, vepsáním jednotlivých čísel do konzole.

Výstupními daty je číselná hodnota reprezentující medián vstupních dat. Tato hodnota je pro uživatele vypsána do konzole.

**Problematická místa**

Algoritmus založený na setřídění seznamu pomocí funkce *sort()*, je velmi pochopitelný a přehledně zapsatelný. Nedosahuje však nevyšší možné efektivity, jeho efektivita je definována jako O(n\*log(n)). Vylepšením tohoto algoritmu může být použití algoritmu *Quickselect*. Ten slouží k vyhledání jakéhokoliv prvku v seznamu, a muže být použit i pro vyhledání mediánu. Výhodou je že tento algoritmus má ve většině případů časovou efektivitu O(n). Vztah mezi dobou běhu algoritmu a množstvím vstupních dat je tedy lineární (Cohen 2018).

**Zdroje**

COHEN, R. (2018): My Favorite Algorithm: Linear Time Median Finding, <https://rcoh.me/posts/linear-time-median-finding/> (cit. 30.1.2022).

IT SLOVNÍK.cz: Co je to Medián?, <https://it-slovnik.cz/pojem/median> (cit. 30.1.2022).

Algoritmy.net: Shaker Sort, <https://www.algoritmy.net/article/93/Shaker-sort> (cit. 30.1.2022).

Algoritmy.net: Porovnání řadících algoritmů, <https://www.algoritmy.net/article/75/Porovnani-algoritmu> (cit. 30.1.2022).

Programiz: Bubble Sort., <https://www.programiz.com/dsa/bubble-sort> (cit. 30.1.2022).

Wikipedie (2022): Sorting algorithm, <https://en.wikipedia.org/wiki/Sorting_algorithm> (cit. 30.1.2022).