

# Vybrané grafové algoritmy v jazyce Python

## 1 Úvod

Tento text vznikl jako doprovodný materiál ke cvičení školy programovacích technik pořádané na katedře Informatiky Vysoké školy báňské - Technická univerzita Ostrava. Programovacím jazykem pro cvičení je Python. Každý úkol ve cvičení obsahuje popis zadání, odkaz na kódy ze kterých při řešení budete vycházet a u některých úkolů je součástí také odkaz na externí video, jenž může danou problematiku pomoci přiblížit.

## Python

Spouštění vzorových kódů a případné vypracování úkolů v jazyce Python vyžaduje jeho instalaci. Python je dostupný ke stažení pro většinu dnešních platforem <sup>1</sup>. Po správném nainstalování Pythonu je možné zkusit vytvořit a spustit první program. K tomuto účelu je možné využít buď příkazový řádek a běžný textový editor, nebo vyvojové prostředí typu PyCharm <sup>2</sup> nebo Visual Studio Code <sup>3</sup>. Vývojové prostředí pro daný jazyk obsahuje nápovědu, doplňování slov a další pomocníky, které programování usnadní. Na webu existuje celá řada tutoriálů <sup>4</sup>, které ukazují jak Python instalovat a spustit, takže podrobnější popis zde vynecháme.

---

<sup>1</sup><https://www.python.org/downloads/>

<sup>2</sup><https://www.jetbrains.com/pycharm/>

<sup>3</sup><https://code.visualstudio.com>

<sup>4</sup><https://www.youtube.com/watch?v=N4mEzFDjqtA>

## Úkol 1: Načtení grafu

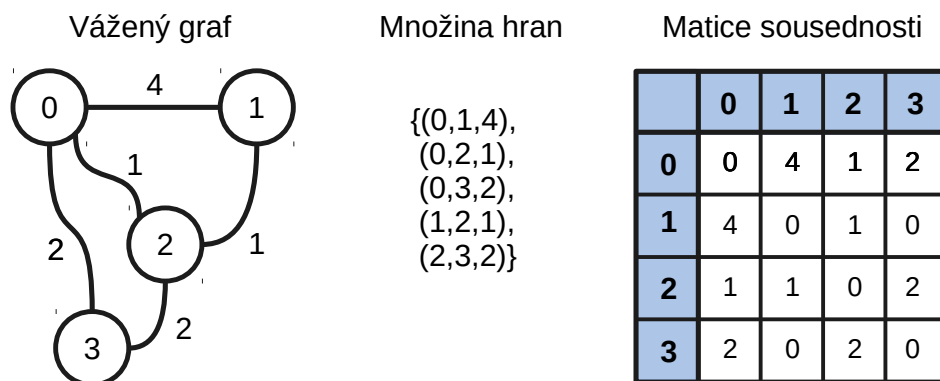
Skript:

<https://github.com/RadimBaca/SOPA/tree/master/ukol1/mustr.py>

Zadání:

Graf je v souboru reprezentován jako množina hran ve formátu `vrchol1;vrchol2;vaha`.

Hlavním úkolem je uložit graf jako matici sousednosti <sup>5</sup>.



Video:

<https://www.youtube.com/watch?v=9C2cpQZVRBA>

Toto video obsahuje podrobnější popis matice sousednosti spolu s popisem jejích výhod a nevýhod.

---

<sup>5</sup>[https://cs.wikipedia.org/wiki/Matice\\_sousednosti](https://cs.wikipedia.org/wiki/Matice_sousednosti)

## Úkol 2: Stupeň grafu

Skript:

<https://github.com/RadimBaca/SOPA/tree/master/ukol2/mustr.py>

Zadání:

Graf je nyní reprezentován jako matice sousednosti. Vážený stupeň vrcholu je dán součtem vah jeho hran. Pro spočítání váženého stupně každého vrcholu v grafu využijte Numpy funkci `sum`.

```
rank = matrix.sum(axis=1)
```

Hlavním úkolem je vypsat prvních deset osob dle jejich váženého stupně. Při vypisování prvních deset osob využijte také mapu `names`, kterou jste načetli ze souboru v rámci předchozího úkolu. Ve skriptu je možné vidět očekávaný výstup.

## Úkol 3: Karate klub

Skript:

<https://github.com/RadimBaca/SOPA/tree/master/ukol3/mustr.py>

Zadání:

Dalším úkolem je analýza Zachary karate klubu. Jde o sociální síť univerzitního karate klubu. Tento karate klub byl studován doktorem Wayne W. Zacharym po dobu tří let (1970-1972). Vrcholy sítě jsou osoby karate klubu a hrany představují skutečnost, že dvě osoby se mezi sebou znají a stýkají i mimo karate klub. V průběhu kdy byl karate klub studován se vystupňoval konflikt mezi administrátorem a instruktorem, který vedl k rozdělení klubu na dva <sup>6</sup>.

Načtěte graf karate klubu (naleznete v adresáři data na github) a použijte předchozí analýzu dle vážených stupňů najít dvě nejdůležitější osoby (administrátora a instruktora) v grafu. Tzn. nalezněte jejich číslo.

---

<sup>6</sup>[https://en.wikipedia.org/wiki/Zachary%27s\\_karate\\_club](https://en.wikipedia.org/wiki/Zachary%27s_karate_club)

## Úkol 4: Průměr grafu a nejkratší cesta

Data:

<https://github.com/RadimBaca/SOPA/tree/master/ukol4/mustr.py>

Zadání:

Dokončete kód jenž bude s pomocí algoritmu Breath-first search (BFS) <sup>7</sup> hledat nejkratší vzdálenost od výchozího vrcholu ke všem ostatním vrcholům v grafu. Vstupem algoritmu je tedy id vychoziho vrcholu a výstupem má být list s těmito vzdálenostmi.

Tyto vzdálenosti pro každý vrchol jsou následně sečteny (řádek 27 ve vzorovém souboru) a spočítáme průměrnou nejkratší vzdálenost. Průměrná nejkratší vzdálenost v grafu se nazývá průměr grafu.

Uveďme ještě základní popis datových struktur v BFS algoritmu, kde  $V$  je počet vrcholů v grafu:

- **distances** - list o velikosti  $V$ , kde každá pozice udává nejkratší vzdálenost od výchozího vrcholu. Na začátku jsou všechny nastaveny na nula.
- **visited** - list o velikosti  $V$ , kde každá pozice uchovává příznak, zda-li byl vrchol navštíven nebo ne. Na začátku jsou všechny nastaveny na **False**.
- **queue** - list dvojic (**vrchol**, **vzdalenost**). Na začátku je v listu pouze jedna dvojice (**vychozi vrchol**, **0**). Pro vkládání nových dvojic použijte funkci **append**.

Video:

[https://www.youtube.com/watch?v=E\\_V71Ejz3f4](https://www.youtube.com/watch?v=E_V71Ejz3f4)

Toto video názorně demonstruje základní ideu algoritmu BFS.

---

<sup>7</sup>[https://en.wikipedia.org/wiki/Breadth-first\\_search](https://en.wikipedia.org/wiki/Breadth-first_search)

## Úkol 5: Floyd-Warshall

Data:

<https://github.com/RadimBaca/SOPA/tree/master/ukol5/mustr.py>

Zadání:

Spočítejte průměr grafu s pomocí Floyd-Warshall algoritmu <sup>8</sup>.

Tento algoritmus je i spolu s pseudokódem a příkladem obstojně popsán na wikipedii <sup>9</sup>. Tento algoritmus funguje i pro vážené grafy.

Využijte knihovnu `time` pro porovnání doby výpočtu průměru grafu s pomocí BFS a s pomocí Floyd-Warshall algoritmu.

Video:

<https://www.youtube.com/watch?v=40QeCuLYj-4>

Toto video názorně demonstruje základní ideu algoritmu.

## Bonusový úkol: Layer of indirection

Pozorný programátor si jistě všiml určitého nedostatku v naší implementaci, která pro určité typy vstupů může vést k nekorektním výsledkům. Počítáme s tím, že vstupní graf indexuje všechny vrcholy s pomocí konečné posloupnosti přirozených čísel  $a_n = n$  kde  $n$  jde od nuly. Problém tedy nastane jakmile tento předpoklad není splněn a posloupnost nezačíná nulou, nebo dokoce obsahuje ‘díry’. Tzn. některá čísla v daném rozsahu se v posloupnosti indexů vrcholů vůbec nevyskytují. První možnost (tzn. indexy nezačínají nulou) se vyskytuje u kolekce karate klubu a toto je lehce řešitelné odečtením jedničky při počítání parametrů. V případě děr je situace komplikovanější a vyžadovala by vytvoření tzv. *layer of indirection* <sup>10</sup>, což je v podstatě datová struktura implementující funkci, která přeloží id vrcholu na přirozené číslo s požadovanými vlastnostmi.

---

<sup>8</sup>[https://en.wikipedia.org/wiki/Floyd%E2%80%93Warshall\\_algorithm](https://en.wikipedia.org/wiki/Floyd%E2%80%93Warshall_algorithm)

<sup>9</sup>[https://en.wikipedia.org/wiki/Floyd%E2%80%93Warshall\\_algorithm](https://en.wikipedia.org/wiki/Floyd%E2%80%93Warshall_algorithm)

<sup>10</sup><https://en.wikipedia.org/wiki/Indirection>

Tato vrstva by se mohla implementovat s pomocí hash tabulky v Pythonu (jednoduše *dictionary* <sup>11</sup>), kde bychom uložili dvojice (`poradi_v_matici`, `index_vrcholu`) pro každý vrchol. Pořadí v matici by už byla posloupnost přirozených čísel, která roste od nuly a neobsahuje mezery.

Zadání:

Cílem tohoto bonusového úkolu je rozšířit naši funkci `read_matrix`, tak aby

- vracela matici, kde nebudou žádné ‘díry’
- vracela dictionary, s pomocí které bude možné pro index řádku matice dohledat původní index vrcholu

---

<sup>11</sup><https://docs.python.org/3/tutorial/datastructures.html#dictionaries>