FAKULTA INFORMATIKY A INFORMAČNÝCH TECHNOLÓGIÍ SLOVENSKÁ TECHNICKÁ UNIVERZITA
Ilkovičova 2, 842 16 Bratislava 4

2023/ 2024

Umelá inteligencia

Zadanie č.3

Cvičiaci: **Ing. Martin Komák, PhD.**
Čas cvičení: 16:00

Vypracoval: **Adam Vrabeľ**
AIS ID: 116327

Obsah

KLASTROVANIE (ZADANIE 3C) 3

ZNENIE ZADANIA 3
RIEŠENÝ PROBLÉM 4
ZÁKLADNÉ POJMY 4
ZHLUK / CLUSTER *4*
CENTROID *4*
MEDOID *4*
AGLOMERATÍVNE ZHLUKOVANIE *4*
IMPLEMENTÁCIA 5
INFORMÁCIE K IMPLEMENTÁCII *5*
GENEROVANIE BODOV *5*
VIZUALIZÁCIA ZHLUKOV *7*
ZHLUKY *8*
AGLOMERATÍVNE ZHLUKOVANIE *9*
VÝPOČET VZDIALENOSTÍ *10*
TESTOVANIE 11
ZÁVER 13

KLASTROVANIE (zadanie 3C)

Znenie zadania

Máme 2D priestor, ktorý má rozmery X a Y, v intervaloch od -5000 do +5000. Tento 2D priestor vyplňte 20 bodmi, pričom každý bod má náhodne zvolenú polohu pomocou súradníc X a Y. Každý bod má unikátne súradnice (t.j. nemalo by byť viacej bodov na presne tom istom mieste).

Po vygenerovaní 20 náhodných bodov vygenerujte ďalších 20000 bodov, avšak tieto body nebudú generované úplne náhodne, ale nasledovným spôsobom:

1. Náhodne vyberte jeden zo všetkých doteraz vytvorených bodov v 2D priestore. (nie len z prvých 20) Ak je bod príliš blízko okraju, tak zredukujete príslušný interval, uvedený v nasledujúcich dvoch krokoch.

2. Vygenerujte náhodné číslo X_offset v intervale od -100 do +100

3. Vygenerujte náhodné číslo Y_offset v intervale od -100 do +100

4. Pridajte nový bod do 2D priestoru, ktorý bude mať súradnice ako náhodne vybraný bod v kroku 1, pričom tieto súradnice budú posunuté o X_offset a Y_offset

Vašou úlohou je naprogramovať zhlukovač pre 2D priestor, ktorý zanalyzuje 2D priestor so všetkými jeho bodmi a rozdelí tento priestor na k zhlukov (klastrov). Implementujte rôzne verzie zhlukovača, konkrétne týmito algoritmami:

- **aglomeratívne zhlukovanie**, kde stred je centroid
- **aglomeratívne zhlukovanie**, kde stred je medoid (stačí 5000 bodov)

Vyhodnocujte úspešnosť/chybovosť vášho zhlukovača. Za úspešný zhlukovač považujeme taký, v ktorom **žiaden klaster nemá priemernú vzdialenosť bodov od stredu viac ako 500**.

Vizualizácia: pre každý z týchto experimentov vykreslite výslednú 2D plochu tak, že označkujete (napr. vyfarbíte, očíslujete, zakrúžkujete) výsledné klastre.

Dokumentácia musí obsahovať opis konkrétne použitých algoritmov a reprezentácie údajov. Uveďte aj vizualizácie viacerých pokusov. V závere zhodnoťte dosiahnuté výsledky ich porovnaním.

RIEŠENÝ PROBLÉM

V tomto zadaní bolo mojou úlohou vytvoriť program, ktorý najprv vygeneruje body podľa zadaných pravidiel a následne podľa zvolenej možnosti vykoná aglomeratívne zhlukovanie podľa centroidu alebo medoidu.

ZÁKLADNÉ POJMY

ZHLUK / CLUSTER

Zhluk je skupina bodov.

CENTROID

Centroid zhluku je bod, ktorý je priemerom všetkých bodov v zhluku, je to fiktívny bod, umelo vytvorený, centroid nieje bod zo zhluku.

MEDOID

Medoid zhluku je konkrétny bod v zhluku, ktorý má minimálnu celkovú vzdialenosť od všetkých ostatných bodov v zhluku.

AGLOMERATÍVNE ZHLUKOVANIE

Aglomeratívne zhlukovanie je jednou z metód v oblasti zhlukovacej analýzy, ktorá sa vyznačuje postupným spájaním (aglomeráciou) jednotlivých bodov alebo malých zhlukov do väčších a väčších zhlukov, až kým nedosiahne požadovaný počet zhlukov alebo jeden celkový zhluk obsahujúci všetky prvky. Tento proces je nazývaný "hierarchický", pretože vytvára hierarchiu zhlukov od jednotlivých prvkov až po jediný zhluk obsahujúci všetky dáta.

Môj aglomeratívny algoritmus funguje následovne:

Na začiatku je každý bod jeden zhluk. Pre každý zhluk sa vypočíta podľa určenia centroid alebo medoid vypočíta sa matica vzdialeností centroidov (resp. medoidov) pre všetky zhluky a dva zhluky s najmenšou vzdialenosťou sa zlúčia, prepočítajú sa vzdialenosti a pokračuje sa v zhlukovaní, až pokiaľ nieje splnená podmienka, že aktuálny počet zhlukov je rovnaký ako má byť výsledný počet zhlukov.

IMPLEMENTÁCIA

INFORMÁCIE K IMPLEMENTÁCII

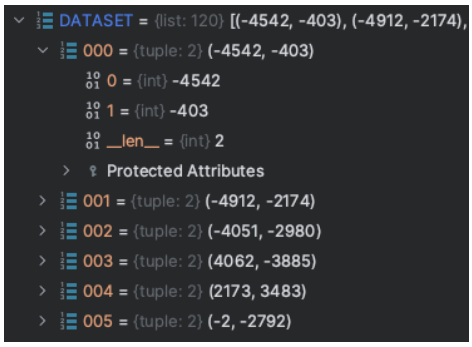
Môj program som implementoval vo vývojovom prostredí PyCharm, v jazyku Python s použitím algoritmu aglomeratívneho zhlukovania. Používal som knižnicu numpy, ktorá efektívnejšie pracuje s matematickými dátami (matice) a obsahuje funkcie na výpočty nad celou maticou. Na vytvorenie grafov som použil knižnicu matplotlyb. Knižnicu random som použil pre náhodné generovanie bodov pri vytvárani DATASETU.

```
import random
import numpy as np
import matplotlib.pyplot as plt
```

GENEROVANIE BODOV

V mojom programe sú body reprezentované ako list tuple dvoch čísel, čize (x, y).

```
##########################################################################################
# X a Y súradnice budú od -5000 do +5000
MIN_VALUE = -5000
MAX_VALUE = 5000

NUM_OF_START_POINTS = 20              # NAJPRV VYGENERUJE x POČIATOČNÝ BODOV
NUM_OF_ANOTHER_POINTS = 20000         # NÁSLEDNE ĎALŠÍCH y BODOV, v okolí doteraz vygenerovaných
##########################################################################################

# DATASET ==>  list bodov (x, y)
DATASET = generate_points()
```



Body generujem podľa zadania, najprv vygenerujem vrámci mojích určených okrajov, MAX_VALUE a MIN_VALUE prvých 20 (NUM_OF_START_POINTS) bodov. Pre tento projekt aj po x-ovej aj po y-ovej osi vieme ísť od -5000 až po +5000.

Následne generujem body až pokiaľ ich nieje celkovo 20020 (NUM_OF_START_POINTS + NUM_OF_ANOTHER_POINTS) a to s následovnou logikou:

Náhodne si zvolím jeden bod z existujúcich, vygenerujem zadaný offset pre x a y os, skontrolujem či som nevyšiel za hranice, ak som mimo, zmenším offsety tak, aby som nevyšiel mimo a vygenerujem offset ešte raz a pridám bod ku všetkým doteraz vegenerovaným. Tým zabezpečím krajší tvar DATASETU, všetky body, ktoré by boli generované mimo plochu nebudu natvrdo pridané na kraj ale dogeneruje sa im nahodé miesto vrámci offsetu a možnosti aby nevyšli mimo hranice.

FUNKCIA generate_points()

```python
# GENERUJE BODY PODĽA ZADANIA
# BOD = (x, y)
# vráti LIST BODOV
3 usages  xvrabela1
def generate_points():
    ALL_POINTS = []

    # GENEROVANIE JEDINEČNÝCH NÁHODNÝCH BODOV (20bodov)
    generated_points = []  # VYGENEROVANÉ BODY (X, Y, FARBA)

    # GENERUJE JEDINEČNÉ BODY (až pokiaľ ich nieje NUM_OF_START_POINTS dokopy)
    while len(generated_points) < NUM_OF_START_POINTS:
        # pozicia(X,Y)
        # randint(A,B)  od A do B, vrátane
        x = random.randint(MIN_VALUE, MAX_VALUE)  # od -5000 do +5000
        y = random.randint(MIN_VALUE, MAX_VALUE)  # od -5000 do +5000

        if (x, y) not in generated_points:
            generated_points.append((x, y))

    ALL_POINTS.extend(generated_points)  # PRIPOJÍ VYGENEROVANÉ BODY (teraz 20)

    # GENEROVANIE ĎALŠÍCH BODOV (20000bodov) podľa pravidiel

    while len(ALL_POINTS) < (NUM_OF_START_POINTS + NUM_OF_ANOTHER_POINTS):
        # NÁHODNE VYBERIEM BOD Z DOTERAZ VYTVORENÝCH
        choosen_point = random.choice(ALL_POINTS)

        # AK JE PRÍLIŠ BLÍZKO OKRAJU DAJ MENŠÍ OFFSET:    (vlastne ak prejde cez okraj tak ho dám na okraj)
        # VYGENERUJ náhodný X_offset (-100 do +100)
        # VYGENERUJ náhodný Y_offset (-100 do +100)

        # randint(A, B)  od A do B, vrátane
        X_offset = random.randint(-100,  b: 100)
        Y_offset = random.randint(-100,  b: 100)

        new_x = choosen_point[0] + X_offset
        new_y = choosen_point[1] + Y_offset

        # AK NOVÉ ČÍSLA SÚ MIMO PLOCHU:

        # NOVÉ X JE MIMO PLOCHU
        if not (-5000 <= new_x <= 5000):
            change_offset = 0
            new_X_offset = 0
            # print("CISLO X JE MIMO PLOCHU")

            # VYPOCITAM SI O KOLKO MI MÁ ZMENIT OFFSET HODNOTU, ABY SOM NEVYŠIEL MIMO MOJHO PRIESTORU
            if (choosen_point[0] + 100) > 5000:
                # print("VIE VYJST DOPRAVA")
                change_offset = choosen_point[0] + 100 - 5000                # napr. x = 4998   ==> 4998 + 100 = 5198  | 5098 - 5000 = 98
                new_X_offset = random.randint(-100, (100 - change_offset))      # takze novy (posunutý) offset = (-100, (100-98))  => (-100, 2)

            elif (choosen_point[0] - 100) < -5000:
                # print("VIE VYJST DOLAVA")
                change_offset = 5000 + choosen_point[0] - 100  # napr. x = -4998   ==> -4998 - 100 = -5098  | 5000 + -5098 = -98  | abs(-98) = 98
                change_offset = abs(change_offset)
```

VIZUALIZÁCIA ZHLUKOV

Zhluky (body v zhlukoch) vizualizujem pomocou knižnice matplotlib a vlastnej funkcie, kde zhluky oddeľujem farebne. Centroid alebo medoid je čierny nevyplnený krúžok, aby sa dalo jasne ukázať, aký bod to je.

```python
def print_clusters(clusters, filename=None):
    ALL_CLUSTERS = copy.deepcopy(clusters)

    # Vytvorím graf s prázdnymi osami v rozsahu od -5000 do 5000
    # plt.axis((-5000, 5000, -5000, 5000))
    plt.axis((MIN_VALUE, MAX_VALUE, MIN_VALUE, MAX_VALUE))

    # Nastavím popisky osí a titulok
    plt.xlabel('X-ová os')
    plt.ylabel('Y-ová os')
    plt.title('KLASTROVANIE  [zadanie 3C]')

    # Nastavím hodnoty osí po tisícoch
    plt.xticks(range(-5000, 5001, 1000))
    plt.yticks(range(-5000, 5001, 1000))

    # Pridám bod (0,0) s veľkosťou guličky 5
    # plt.scatter(0, 0, color='red', s=5)

    i = 0
    # farba = ["red", "blue", "green", "purple", "black"]
    for cluster in ALL_CLUSTERS:

        hodnoty_x = []
        hodnoty_y = []
        for point in cluster.points:
            hodnoty_x.append(point[0])
            hodnoty_y.append(point[1])

        # ZAPÍŠEM BODY DO GRAFU (rovnakej farby)
        # color = farba[i]
        # generovanie náhodnej farby v HEX formáte
        color = "#{:02x}{:02x}{:02x}".format( *args: np.random.randint( low: 0,  high: 256), np.random.randint( low: 0,  high: 256), np.random.randint( low: 0,  high: 256))

        # plt.scatter(hodnoty_x, hodnoty_y, color="orange", s=5)
        plt.scatter(hodnoty_x, hodnoty_y, color=str(color), s=10, zorder=0)

        global calculate_medoid  # ak TRUE - tak počíta s medoidom, ak FALSE - tak počíta s centroidom
        if calculate_medoid:
            # vykreslenie medoidu
            plt.scatter(cluster.medoid[0], cluster.medoid[1], marker="o", edgecolors="black", facecolors='none', s=10, linewidths=0.8, zorder=1)
        else:
            # vykreslenie centroidu
            plt.scatter(cluster.centroid[0], cluster.centroid[1], marker="o", edgecolors="black", facecolors='none', s=10, linewidths=0.8, zorder=1)

        i += 1

    # Pridáme legendu
    # plt.legend()

    # Uloženie grafu do súboru vo formáte PNG
    if filename is not None:
        filename = str(filename)
        plt.savefig(f"export_graphs/{filename}.png")
```

ZHLUKY

Jednotlivé zhluky reprezentujem pomocou triedy Cluster, ktorá obsahuje funkcie calculate_centroid(), ktorá vypočíta centroid z daného zhluku, find_medoid(), ktorá nájde medoid v zhluku a funkciu add_points() ktorá pridá body do zhluku a zavolá požadovanú funkciu pre nájdenie centroidu alebo medoidu.

```python
# TRIEDA PRE ZHLUK BODU/BODOV (cluster)
3 usages  xvrabela1
class Cluster:

    # AK VYTVÁRAM CLUSTER ==> LIST = JEDEN BOD a TEN JE ZÁROVEŇ CENTROID
    xvrabela1
    def __init__(self, points):
        self.points = []
        self.points.extend(points)      # LIST BODOV / alebo aj jedného [(x,y), ...]

        global id_counter
        self.id = id_counter            # jedinečné ID pre cluster
        id_counter += 1

        global calculate_medoid     # ak TRUE - tak počíta s medoidom, ak FALSE - tak počíta s centroidom
        if calculate_medoid:
            self.medoid = None
            self.find_medoid()
        else:
            # self.centroid = centroid        # JEDEN BOD (x,y)
            self.centroid = None
            self.calculate_centroid()  # sam si vypocita centroid

    ###########################################################################
```

```python
# NAJDENIE CENTROIDU V ZHLUKU BODOV
2 usages  xvrabela1
def calculate_centroid(self):
    # Centroid (ťažisko) sa vypočíta ako priemerná hodnota súradníc všetkých bodov v zhluku
    # Používame axis=0, aby sme vypočítali priemernú hodnotu pre každý stĺpec (X a Y) zvlášť.
    new_centroid = np.mean(self.points, axis=0)
    new_centroid = tuple(new_centroid)  # NumPy array si konvertujem na tuple (x,y)

    self.centroid = new_centroid        # nastavím nový centroid
#############################################################################################

# NAJDENIE MEDOIDU V ZHLUKU BODOV
2 usages  xvrabela1
def find_medoid(self):
    min_total_distance = float('inf')
    medoid_point = None

    # prechádzam všetky body v zhluku.
    for point1 in self.points:
        total_distance = 0

        # pre každý bod v zhluku vypočítam jeho vzdialenosť od ostatných bodov v zhluku.
        for point2 in self.points:
            total_distance += euclidean_distance(point1, point2)

        # ak nájdem bod s nižšou celkovou vzdialenosťou, aktualizujem medoid
        if total_distance < min_total_distance:
            min_total_distance = total_distance
            medoid_point = point1

    self.medoid = medoid_point
#############################################################################################

# PRIDÁVANIE BODOV DO CLUSTERA (LIST BODOV (x,y) ) aj jeden bod v LISTE bude ok
1 usage  xvrabela1
def add_points(self, points):
    # PRIDÁ BODY K EXISTUJÚCIM
    self.points.extend(points)

    global calculate_medoid  # ak TRUE - tak počíta s medoidom, ak FALSE - tak počíta s centroidom

    if calculate_medoid:
        # VYPOČÍTA MEDOID
        self.find_medoid()
    else:
        # VYPOČÍTA NOVÝ CENTROID
        self.calculate_centroid()
        # PREPOČÍTAM VZDIALENOSTI PRE TENTO ZHLUK ku všetkým ostatným bodom
        # for cyklus listu clusterv, bude niečo robiť, no na sebe samom to vynecha ==> ... if cluster is not self ...
    pass
#####################################################################################################
```

AGLOMERATÍVNE ZHLUKOVANIE

Táto funkcia zabezpečuje samotné aglomeratívne zhlukovanie ako som ho opísal na začiatku.

```python
# ALGORITMUS PRE AGLOMERATÍVNE ZHLUKOVANIE, kde stred je centroid
# ak nastavím medoid=True, tak zhlukuje podľa medoidu
1 usage  xvrabela1 *
def aglomerative_w_centroid(dataset, k):
    ALL_POINTS = copy.deepcopy(dataset)

    # dataset ==>  list dát (x,y)
    # k       ==>  na kolko zhlukov rozdelím dataset

    # NA ZAČIATKU SA KAŽDÝ BOD POČÍTA AKO JEDEN ZHLUK (cluster)
    clusters = []
    for point in ALL_POINTS:
        tmp_cluster = Cluster(points=[point])   # inicializácia zhlukov: každý bod == samostatný zhluk (list s jedným bodom) | jeden bod v liste je sam sebe aj centroid
        clusters.append(tmp_cluster)            # pridá nový cluster do listu

    num_of_iteration = 0

    # postupné zlučovanie zhlukov (clusterov), kým nie je dosiahnuta podmienka
    # is_succsessful = False
    # is_succsessful = control_each_cluster(clusters)     # prvé klastre netreba kontrolovať (určite nebude splnená podmienka)
    # while is_succsessful is False:  # pokial nieje zhlukovač úspešný, tak zhlukujem
    while len(clusters) > k:

        print(f"NUMBER OF ITERATION: {num_of_iteration}")
        num_of_iteration += 1

        global calculate_medoid

        # S CENTROIDOM
        if not calculate_medoid:
            # TERAZ VYPOČÍTAM VŠETKY VZDIALENOSTI PRE JEDNOTLIVÉ CENTROIDY V KLASTROCH
            clusters_distances = make_centroids_distance_matrix(clusters)

        # S MEDOIDOM
        else:
            # TERAZ VYPOČÍTAM VŠETKY VZDIALENOSTI PRE JEDNOTLIVÉ MEDOIDY V KLASTROCH
            clusters_distances = make_medoids_distance_matrix(clusters)

        #
        # VYBERIEM ZHLUKY S NAJMENŠIOU VZDIALENOSŤOU CENTROIDOV, tie neskor spolu zlúčim
        min_distance_index = np.unravel_index(np.argmin(clusters_distances), clusters_distances.shape)

        # Získajte indexy zhlukov s najmenšou vzdialenosťou
        index_cluster1 = min_distance_index[0]
        index_cluster2 = min_distance_index[1]

        # Získajte príslušné zhluky zo zoznamu clusters ktoré treba zlúčiť
        cluster1 = clusters[index_cluster1]
        cluster2 = clusters[index_cluster2]

        # VIEM ŽE cluster1 a cluster2 majú najmenšiu vzdialenosť centroidov

        # ZLÚČENIE DVOCH ZHLUKOV
        # do prvého zhluku pridám body z druhého, ten si sám prepočíta centroid
        cluster1.add_points(cluster2.points)
```

VÝPOČET VZDIALENOSTÍ

Funkcia vypočíta maticu vzdialeností jednotlivých zhlukov (pre centroidy alebo medoidy)

```python
# vypočíta 2D maticu vzdialeností centroidov v ZHLUKOCH
1 usage  xvrabela1
def make_centroids_distance_matrix(clusters):

    # vytvorím 2D maticu, kde element (i, j) obsahuje vzdialenosť medzi bodmi (centroidmi) clustrov i a j
    distances = np.zeros((len(clusters), len(clusters)))    # na začiatku všade inicializované na 0

    # pre celú maticu distances vypočíta všetky vzdialenosti centroidov (z daného zhluku) a zapíše do matice
    for i in range(len(clusters)):
        for j in range(len(clusters)):

            distances[i, j] = centroid_distance(clusters[i], clusters[j])   # POSIELAM CELÉ ZHLUKY, funkcia si spracuje sama a vráti vzdialenosť centroidov
            # print(f"VZDIALENOST CENTROIDOV: {distances[i, j]}")

    # Ignorujte diagonálu matice    (pretože na diagonále su nuly, tie nepotrebujem, bod so semou samým ma nezaujíma)
    np.fill_diagonal(distances, np.inf)

    return distances
```

Vzdialenosť počítam ako euklidovskú vzdialenosť bodov v euklidovskej rovine.

```python
# euklidová vzdialenosť medzi dvoma bodmi v euklidovskej rovine
3 usages  xvrabela1
def euclidean_distance(point1, point2):
    # Konvertujte body na NumPy array
    point_1 = np.array(point1)
    point_2 = np.array(point2)

    # Výpočet euklidovskej vzdialenosti
    return np.sqrt(np.sum((point_1 - point_2) ** 2))
```

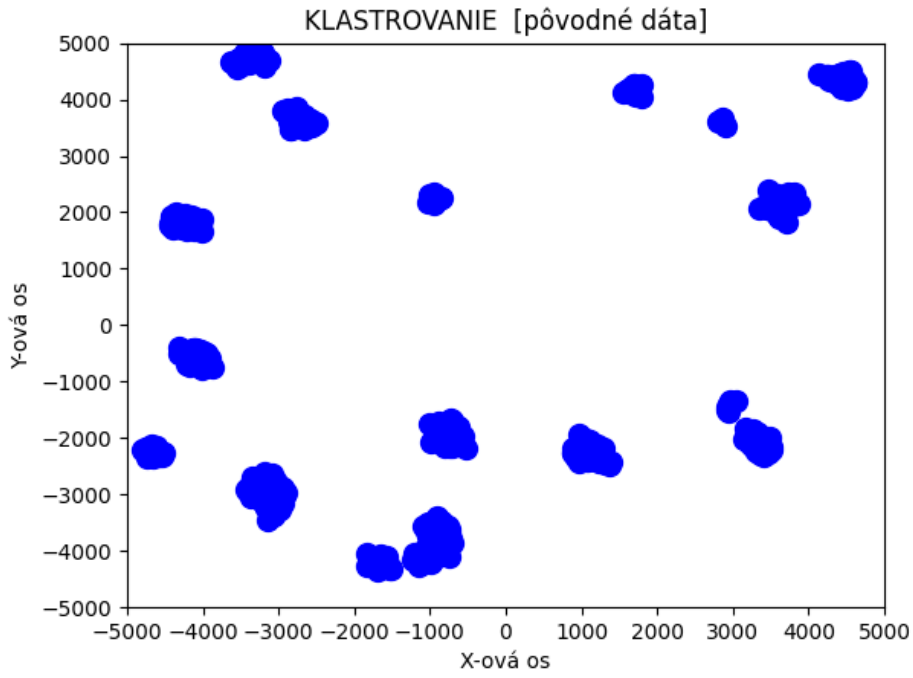
Pokúšal som sa o efektívnejšiu reprezentáciu údajov, kde každý cluster mal jedinečné ID a vzdialenosti som reprezentoval formou KEY VALUE slovníka kde KEY bol (id1, id2) a VALUE bola ich vzdialenosť, táto implementácia bola efektívnejšia ale potrebovala by ešte dolatiť, preto som sa pre odovzdanie rozhodol odovzdať make_centroids_distance_matrix v ktorej sa dá lepšie orientovať.

```python
def calculate_distances_NEW(clusters):
    distances_dict = {}     # key (id_clustera1, id_clustera1)   # value (ich vzdialenost)

    key_1_values = []
    key_2_values = []
    # minimalna_vzdialenost = 20000          # je urcite vacsia ako diagonala v 10 000 x 10 000 (ziadna vzdialenost v poli nebude vacsia ako toto)
    # minimalna_vzdialenost_key = tuple()     # bude to touple(id_clustera1, id_clustera1) kde je minimalna_vzdialenost medzi centroidmi

    # fruits = ['apple', 'banana', 'cherry']
    # for index, value in enumerate(fruits):
    #     print(f'Index: {index}, Value: {value}')

    # prechádzam každý cluster s každým
    for i, cluster1 in enumerate(clusters):
        for j, cluster2 in enumerate(clusters):
            if i < j:  # Zabráni duplicitným párom (i, j) a (j, i)
                # key = (i, j)

                if cluster1.id not in key_1_values:
                    key_1_values.append(cluster1.id)

                if cluster2.id not in key_2_values:
                    key_2_values.append(cluster2.id)

                # moj kľúč = touple(id clustra1, id clustra2)
                key = (cluster1.id, cluster2.id)

                # moja value = vzdialenosť centroidov týchto dvoch clustrov
                distance = centroid_distance(cluster1, cluster2)
                distances_dict[key] = distance

                # # udržiavanie minimalnej vzdialenosti a odkazu k nej cez key
                # if distance < minimalna_vzdialenost:
                #     minimalna_vzdialenost = distance
                #     minimalna_vzdialenost_key = key

    del i, j, key, cluster1, cluster2, distance     # vymazem nepotrebne premenné (pre prehladnosť v debuggeri)

    # ZORADÍM VZDIALENOSTI OD NAJMENŠEJ
    sorted_distance = dict(sorted(distances_dict.items(), key=lambda item: item[1]))

    return_dictionary = {
        # "distances_dict": distances_dict,  # NEZORADENÉ
        "distances_dict": sorted_distance,   # ZORADENÉ OD NAJMENŠEJ VZDIALENOST
        "key_1_values": key_1_values,
        "key_2_values": key_2_values,
        # "min_vzdialenost": minimalna_vzdialenost,
        # "min_vzdialenost_key": minimalna_vzdialenost_key
    }
    # return distances_dict
    return return_dictionary
```

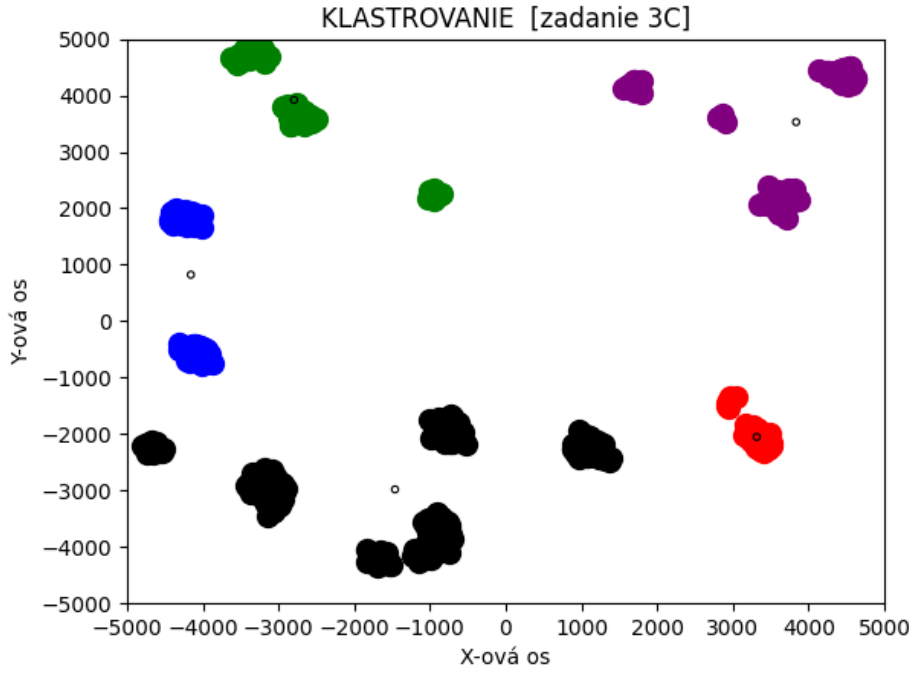
TESTOVANIE

Generovanie 15 020 bodov
Doba generovania 19 hodín 53 minút

Pôvodné zhluky na začiatku:

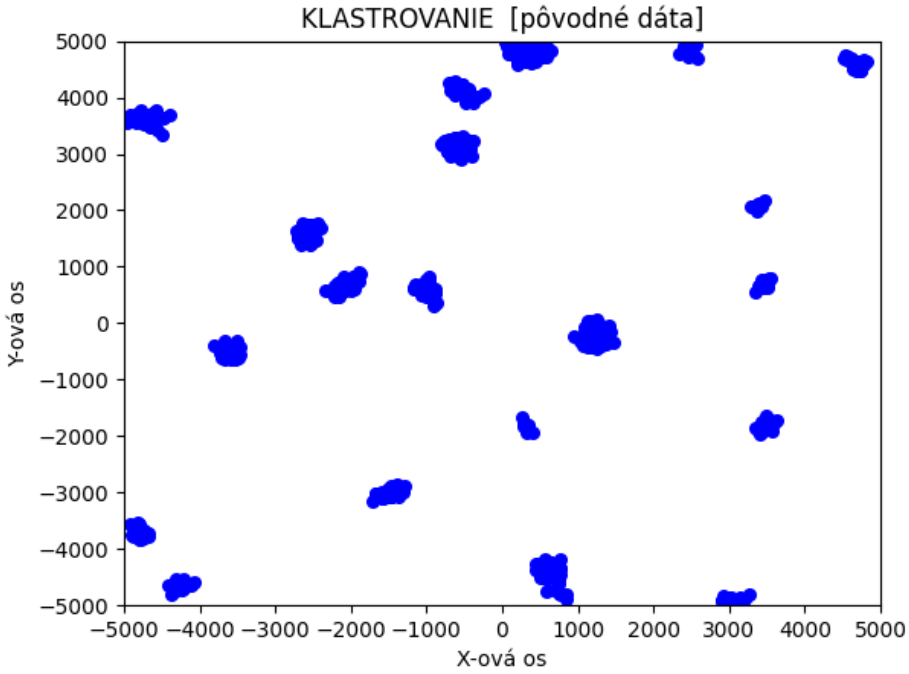


Aglomeratívne zhlukovanie s centroidom:

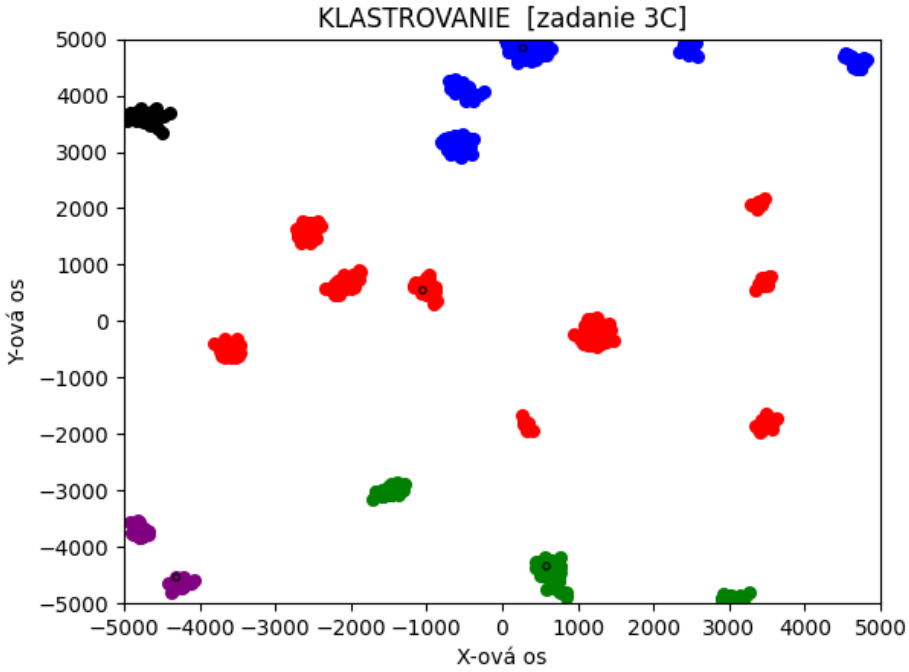


Generovanie 5 020 bodov
Doba generovania 2 hodiny

Pôvodné zhluky na začiatku:



Aglomeratívne zhlukovanie s medoidom:



ZÁVER

Testovanie znázorňuje výsledky zhlukovania s centroidmy a medoidmy. Program dokáže vyriešiť zadaný problém. Program je flexibilný pre rôzne veľkosti plochy a počet bodov. Najväčším obmedzením bola časová náročnosť, keďže už samotný algoritmus aglomeratívneho zhlukovania je časovo neefektívny. Tak isto je problém aj pamäťová náročnosť keďže je potrebné uložiť v najhoršom prípade 20020 x 20020 vzdialenosti (každý bod s každým).

Ako výhodu tohto algoritmu považujem to, že je ľahko pochopiteľný, funkčný pre akúkoľvek plochu s akýmkoľvek počtom bodov. Program som sa snažil komentovať, aby bola každá funkcia pochopiteľná a zrozumiteľna.

Je tu rovnako priestor pre zefektívnenie výpočtu a ukladania vzdialeností, ktorú som začal robiť no nepodarilo sa mi ju vyšperkovať do takej podoby aby bola dostatočne efektívna.