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

2022/2023

Peter Bartoš

Umelá inteligencia
Zadanie 3B

Programovacie prostredie: PyCharm (Python)
Čas cvičení: Streda 13:00 – 14:50

Vypracoval: Peter Bartoš
AIS ID: 116143

Obsah

Riešený problém

Zadefinovaný má byť 2D priestor s rozmermi X a Y, v intervaloch od -5000 do +5000. Priestor sa najprv vyplní dvadsiatimi bodmi, pričom každý bod má unikátnu polohu v priestore a súradnice sú náhodne určené. Po vygenerovaní týchto bodov, sa generuje 40 000 ďalších bodov a body budú generované takým spôsobom, že náhodne sa vyberie už vygenerovaný bod a náhodne sa mu určí offset súradníc v intervale od -100 do +100, vytvorí sa nový bod, kde sa sčítajú súradnice vybraného bodu s daným náhodným offsetom.

Úlohou je naprogramovať zhlukovač pre 2D priestor, ktorý zanalyzuje všetky body a rozdelí ich na k zhlukov. Implementované mali byť rôzne algoritmy:

- k-means, kde stred je centroid
- k-means, kde stred je medoid
- divízne zhlukovanie, kde stred je centroid

Zhlukovač úspešne zanalyzoval priestor, ak žiaden klaster nemá priemernú vzdialenosť bodov od stredu viac ako 500.

Zadefinovanie priestoru a bodu

Ako globálne premenné sú zadefinované rozmery priestoru a offsetov pre body:

```
max_rozmer = 5000
min_rozmer = -5000
max_offset = 100
min_offset = -100
```

Ďalej priestor nie je definovaný. V implementácií sú určené len rozmery, ktoré bod nemôže prekročiť.

Bod je zadefinovaný pomocou class objektu, kde pri inicializácií sa mu určujú súradnice x, y a farba. Ďalej pre zhlukový bod (bod, ktorý reprezentuje koreň zhluku) sú vytvorené ďalšie premenné, ktoré obyčajný bod nepoužíva. Ukážka kódu vyzerá nasledovne:

```
class Bod:
    def __init__(self, x, y, farba):
        self.zhluk_bodov_x = []  # zhlukovy bod atribut
        self.zhluk_bodov_y = []  # zhlukovy bod atribut
        self.sucet_vzdialenosti_bodov = 0  # zhlukovy bod atribut
        self.wcss = 0  # zhlukovy bod atribut
        self.x = x
        self.y = y
        self.prechadzajuce_x = None  # zhlukovy bod atribut
        self.prechadzajuce_y = None  # zhlukovy bod atribut
        self.farba = farba

    def suradnice(self, x, y):
```

```
        self.x = x
        self.y = y

    def nastav_farbu(self, farba):
        self.farba = farba

    def zhluk_bodov(self, x, y):  # zhlukovy bod funkcia
        self.zhluk_bodov_x.append(x)
        self.zhluk_bodov_y.append(y)

    def pridaj_vzdialenost_bodu(self, vzdialenost_bodu):  # zhlukovy bod
funkcia
        self.wcss += vzdialenost_bodu**2
```

Generovanie bodov

Prvých dvadsať bodov sa vygeneruje podľa inštrukcií zo zadania. Vytvorí sa bod s náhodnými súradnicami ohraničenými rozmermi priestoru a kontroluje sa v cykle, či sa taký bod už v priestore nenachádza. Ak sa nenachádza, tak program pokračuje ďalej a uloží bod do listu so všetkými bodmi doposiaľ vygenerovanými:

```
def generuj_prvych_20_bodov():
    bod = Bod(random.randint(min_rozmer, max_rozmer),
random.randint(min_rozmer, max_rozmer), '')
    while [bod.x, bod.y] in suradnice_zaciatocnych_bodov:
        bod.suradnice(random.randint(min_rozmer, max_rozmer),
random.randint(min_rozmer, max_rozmer))
    suradnice_zaciatocnych_bodov.append([bod.x, bod.y])
    vsetky_body.append(bod)
```

Zvyšok bodov sa generuje v cykle, kde sa kontroluje počet všetkých bodov a program skončí, keď dovŕšia 40 000. Vyberie sa náhodne bod z listu všetkých bodov a náhodne sa určí k jeho súradniciam offset. Ďalej sa vytvorí nový bod, kde sa sčítajú súradnice s vygenerovaným offsetom a náhodne vybraným bodom. Tento nový bod sa následne uloží do listu so všetkými bodmi:

```
def generuj_zvysne_body(pocet):
    while len(vsetky_body) + 1 <= pocet:
        vyber = random.randint(0, len(vsetky_body) - 1)
        nahodny_bod = vsetky_body[vyber]
        bod = Bod(nahodny_bod.x + (random.randint(min_offset, max_offset)),
                  nahodny_bod.y + (random.randint(min_offset, max_offset)),
'')
        vsetky_body.append(bod)
```

K-means algoritmus

Implementácia tohto algoritmu v tejto dokumentácií začína tým, že v cykle sa vyberá predom určený počet náhodných bodov, sa budú považovať za „zhlukové body“. Tieto body dostanú farbu a vždycky sa skontroluje, že či sú vzdialené od ostatných zhlukových bodov aspoň o tisíc. Ak je, tak pokračuje algoritmus vo vytváraní týchto bodov a ak nie, tak sa v cykle hľadá bod, ktorý spĺňa túto podmienku:

```
def kmeans(k_cislo, medoid_centroid):
    for j in range(k_cislo):
        nahodny_index_farby = random.randint(0, len(farby_k_bodov) - 1)
        farba = farby_k_bodov[nahodny_index_farby]
        nahodny_index_boda = random.randint(0, len(vsetky_body) - 1)
        k_bod = vsetky_body[nahodny_index_boda]
        if j != 0:
            while True:
                rozhodovanie = True
                for predosly_k_bod in k_body:
                    if predosly_k_bod == k_bod:
                        continue
                    vzdialenost = math.sqrt(((k_bod.x - predosly_k_bod.x)
** 2) + ((k_bod.y - predosly_k_bod.y) ** 2))
                    if vzdialenost < 1000:
                        rozhodovanie = False
                if rozhodovanie is True:
                    break
                nahodny_index_boda = random.randint(0, len(vsetky_body) -
1)
                k_bod = vsetky_body[nahodny_index_boda]
        k_bod.nastav_farbu(copy.deepcopy(farba))
        k_bod.predchadzajuce_x = k_bod.x
        k_bod.predchadzajuce_y = k_bod.y
        farby_k_bodov.pop(nahodny_index_farby)
        k_body.append(k_bod)
```

Po zadefinovaní týchto bodov za začne v nekonečnom cykle vykonávať farbenie ostatných obyčajných bodov. Farbenie týchto bodov funguje tak, že sa v cykle postupne prechádzajú všetky obyčajné body a uchováva sa vzdialenosť ku zhlukovému bodu, ktorá je najbližšia. Následne sa tento najbližšie lokalizovaný zhlukový bod vyberie a pridá sa do jeho listu s obyčajnými bodmi, ktoré spadajú do jeho zhluku:

```
def farbenie_k_bodov():
    for bod in vsetky_body:
        if bod in k_body:
            continue
        vzdialenost_ku_najblizsiemu_k_bodu = -1
        index_k_bodu = 0
        najblizsi_k_bod = None
        for k_bod in k_body:
            nova_vzdialenost = math.sqrt(((bod.x - k_bod.x)**2) + ((bod.y -
k_bod.y)**2))
            if vzdialenost_ku_najblizsiemu_k_bodu > nova_vzdialenost or
vzdialenost_ku_najblizsiemu_k_bodu == -1:
```

```
                vzdialenost_ku_najblizsiemu_k_bodu = nova_vzdialenost
                bod.farba = copy.deepcopy(k_body[index_k_bodu].farba)
                najblizsi_k_bod = k_bod
            index_k_bodu += 1
        najblizsi_k_bod.zhluk_bodov(bod.x, bod.y)
        najblizsi_k_bod.sucet_vzdialenosti_bodov +=
vzdialenost_ku_najblizsiemu_k_bodu

najblizsi_k_bod.pridaj_vzdialenost_bodu(vzdialenost_ku_najblizsiemu_k_bodu)
```

Po zafarbení všetkých bodov sa začne hýbanie zhlukových bodov buď na centroid alebo medoid.

Ak sa na vstupe vyberie centroid, tak sa v cykle postupne začnú prechádzať všetky zhlukové body a vypočítajú sa súradnice, na ktoré sa tento zhlukový bod premiestni. Súradnice sa vypočítajú tým spôsobom, že sa spriemerujú súradnice všetkých obyčajných bodov, ktoré patria pod tento zhlukový bod. Ku každému zhlukovému bodu sa uchovávajú aj jeho súradnice pred týmto výpočtom a určením nových súradníc. Ak sa tieto hodnoty nezmenia, tak sa funkcia na hýbanie zhlukových bodov už nevykoná. Ak sa hodnoty zmenia, tak sa premaže list s jeho obyčajnými bodmi a začne sa farbenie znova:

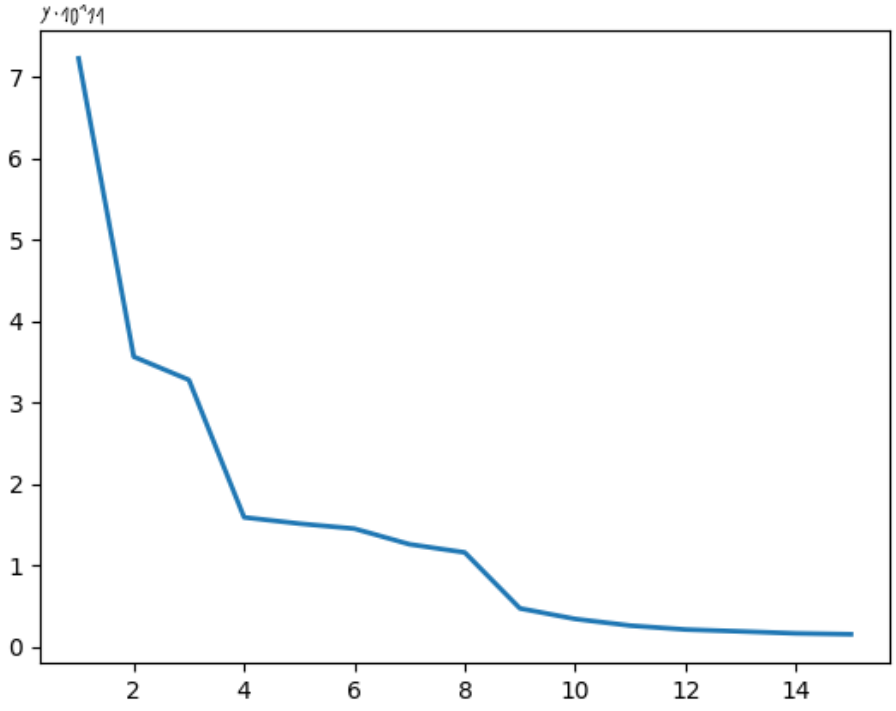
```
def hybanie_k_bodov_centroid(iteracia_hybania):
    bude_pokracovat = True
    for k_bod in k_body:
        sucet = 0
        pocet = 0
        k_bod.predchadzajuce_x = k_bod.x
        k_bod.predchadzajuce_y = k_bod.y
        for x in k_bod.zhluk_bodov_x:
            sucet += x
            pocet += 1
        k_bod.x = math.floor(sucet / pocet)
        sucet = 0
        pocet = 0
        for y in k_bod.zhluk_bodov_y:
            sucet += y
            pocet += 1
        k_bod.y = math.floor(sucet / pocet)
        if k_bod.predchadzajuce_x == k_bod.x and k_bod.predchadzajuce_y ==
k_bod.y:
            bude_pokracovat = False
    if bude_pokracovat is True:
        for k_bod in k_body:
            k_bod.zhluk_bodov_x = []
            k_bod.zhluk_bodov_y = []
            k_bod.sucet_vzdialenosti_bodov = 0
            k_bod.wcss = 0
    return bude_pokracovat
```

Pre medoid to prebieha v tejto implementácií tak isto, iba sa zmení výpočet nových súradníc. Použije sa vzorec pre výpočet mediánu zo štatistiky. Zoberú sa súradnice obyčajných bodov, usporiadajú sa od najmenšieho po najväčšie a vyberie sa stredná hodnota:

```
def hybanie_k_bodov_medoid(iteracia_hybania):
    bude_pokracovat = True
    for k_bod in k_body:
        k_bod.predchadzajuce_x = k_bod.x
        k_bod.predchadzajuce_y = k_bod.y

        len_list_x = len(k_bod.zhluk_bodov_x)
        list_x = copy.deepcopy(k_bod.zhluk_bodov_x)
        list_x.sort(reverse=False)
        k_bod.x = k_bod.zhluk_bodov_x[math.floor(len_list_x / 2)]

        len_list_y = len(k_bod.zhluk_bodov_y)
        list_y = copy.deepcopy(k_bod.zhluk_bodov_y)
        list_y.sort(reverse=False)
        k_bod.y = k_bod.zhluk_bodov_y[math.floor(len_list_y / 2)]
        if k_bod.predchadzajuce_x == k_bod.x and k_bod.predchadzajuce_y ==
k_bod.y:
            bude_pokracovat = False
    if bude_pokracovat is True:
        for k_bod in k_body:
            k_bod.zhluk_bodov_x = []
            k_bod.zhluk_bodov_y = []
            k_bod.sucet_vzdialenosti_bodov = 0
            k_bod.wcss = 0
    return bude_pokracovat
```

Lakeť metóda

Nevýhodou k-means algoritmu je jeho vlastnosť, že nevie koľko zhlukových bodov je ideálne vybrať. Na to sa využíva tzv. „Elbow method“, ktorá to pomôže určiť. Idea je taká, že zvýšenie počtu zhlukov prirodzene zlepší zhodu zhlukovania, pretože existuje viac zhlukov, ktoré je potrebné použiť, ale v určitom bode je to prehnané a na grafe tejto metódy sa to odráža.



Graf poukazuje na prelom, kedy zvyšovanie zhlukov už nepoukazuje na výrazne zlepšenie úspešnosti algoritmu a vyberie si počet zhlukov tam, kde už to nepoukazuje na výrazné zlepšenie úspešnosti.

Implementácia tohto algoritmu v tomto riešení je, že najprv sa k-means vykoná niekoľkokrát a vypočítava sa spriemeruje sa vzdialenosť bodov od jednotlivých zhlukov. Táto hodnota reprezentuje y os v grafe a x os reprezentuje počet zhlukov. Z grafu sa vyčíta hodnota, po ktorej sa už nedeje výrazné zlepšenie a znova sa spraví k-means s počtom zhlukov podľa tejto hodnoty:

```
def laket_metoda(medoid_centroid):
    for k in range(1, 16):
        print("K: " + str(k))
        kmeans(k, medoid_centroid)
        wcss = 0
        for k_bod in k_body:
            wcss += k_bod.wcss
        list_wcss.append(wcss)
        list_poctu_k_bodov.append(len(k_body))
    selected_k = vykresli_laket(list_wcss, list_poctu_k_bodov)
    kmeans_final(selected_k, medoid_centroid)
    vykresli_body()

def vykresli_laket(wcss, pocet_k):
    if len(k_body) == 1:
        return
    fig, ax = plt.subplots()
    predosle_cislo = -1
    pocitadlo = 0
    vysledne_k = None
    for cislo in list_wcss:
        if predosle_cislo == -1:
            predosle_cislo = cislo
            continue
        if (cislo + (predosle_cislo * 0.05) >= predosle_cislo) and (cislo <
predosle_cislo):
            if len(vsetky_body) > 20000 and pocitadlo < 7:
                predosle_cislo = cislo
                pocitadlo += 1
                continue
            vysledne_k = cislo
            break
        predosle_cislo = cislo
        pocitadlo += 1

    ax.plot(pocet_k, wcss, linewidth=2.0)
    fig.canvas.manager.set_window_title('Elbow')
    plt.show()
    if vysledne_k is not None:
        return list_wcss.index(vysledne_k)
    else:
        return list_wcss.index(predosle_cislo)
```

Divízne zhlukovanie

Na implementáciu tohto algoritmu bol využitý k-means z tohto riešenia, ktorý sa cyklicky opakuje do nekonečna, až kým priemerná vzdialenosť zhlukových bodov je menšia ako 500. Najprv prebehne inicializácia divízneho zhlukovania, kde predelí pomocou k-means algoritmu priestor na dva zhluky. Ďalej potom už len pokračuje, kde všetko drobí na menšie zhluky, až kým priemer vzdialeností bude menší ako 500:

```
def divizne_zhlukovanie():
    j = 0
    priemer_vzdialenosti = 9999
    while priemer_vzdialenosti > 500:  # kým nebude priemer vzdialenosti
bodov mensi ako 500
        priemer_vzdialenosti_na_k_bod = 0
        for k_bod in k_body:
            priemer_vzdialenosti_na_k_bod += k_bod.sucet_vzdialenosti_bodov
/ len(k_bod.zhluk_bodov_x)
        priemer_vzdialenosti = priemer_vzdialenosti_na_k_bod / len(k_body)
        print(str(len(k_body)) + "|" + str(priemer_vzdialenosti))
        if priemer_vzdialenosti <= 500:
            return
        if len(farby_k_bodov) != 0:
            nahodny_index_farby = random.randint(0, len(farby_k_bodov) - 1)
            farba = farby_k_bodov[nahodny_index_farby]
        else:
            nahodny_index_farby = random.randint(0, len(farby) - 1)
            farba = farby[nahodny_index_farby]
        nahodny_index_boda = random.randint(0, len(vsetky_body) - 1)
        k_bod = vsetky_body[nahodny_index_boda]
        if j != 0:
            while True:
                rozhodovanie = True
                for predosly_k_bod in k_body:
                    if predosly_k_bod == k_bod:
                        continue
                    vzdialenost = math.sqrt(((k_bod.x - predosly_k_bod.x)
** 2) + ((k_bod.y - predosly_k_bod.y) ** 2))
                    if vzdialenost < 500:
                        rozhodovanie = False
                if rozhodovanie is True:
                    break
                nahodny_index_boda = random.randint(0, len(vsetky_body) -
1)
                k_bod = vsetky_body[nahodny_index_boda]
        k_bod.nastav_farbu(copy.deepcopy(farba))
        k_bod.predchadzajuce_x = k_bod.x
        k_bod.predchadzajuce_y = k_bod.y
        if len(farby_k_bodov) != 0:
            farby_k_bodov.pop(nahodny_index_farby)
        k_body.append(k_bod)
        j += 1

        pocet_iteracii = 0
        while True:
            farbenie_k_bodov()
            if hybanie_k_bodov_centroid(pocet_iteracii) is False:
                break
            pocet_iteracii += 1
```

Testovanie riešenia

Všetky testy boli robené pre 40 000 bodov.

4 testy boli spravené na k-means s centroidom s použitím „elbow method“. Výsledky vyzerajú nasledovne:

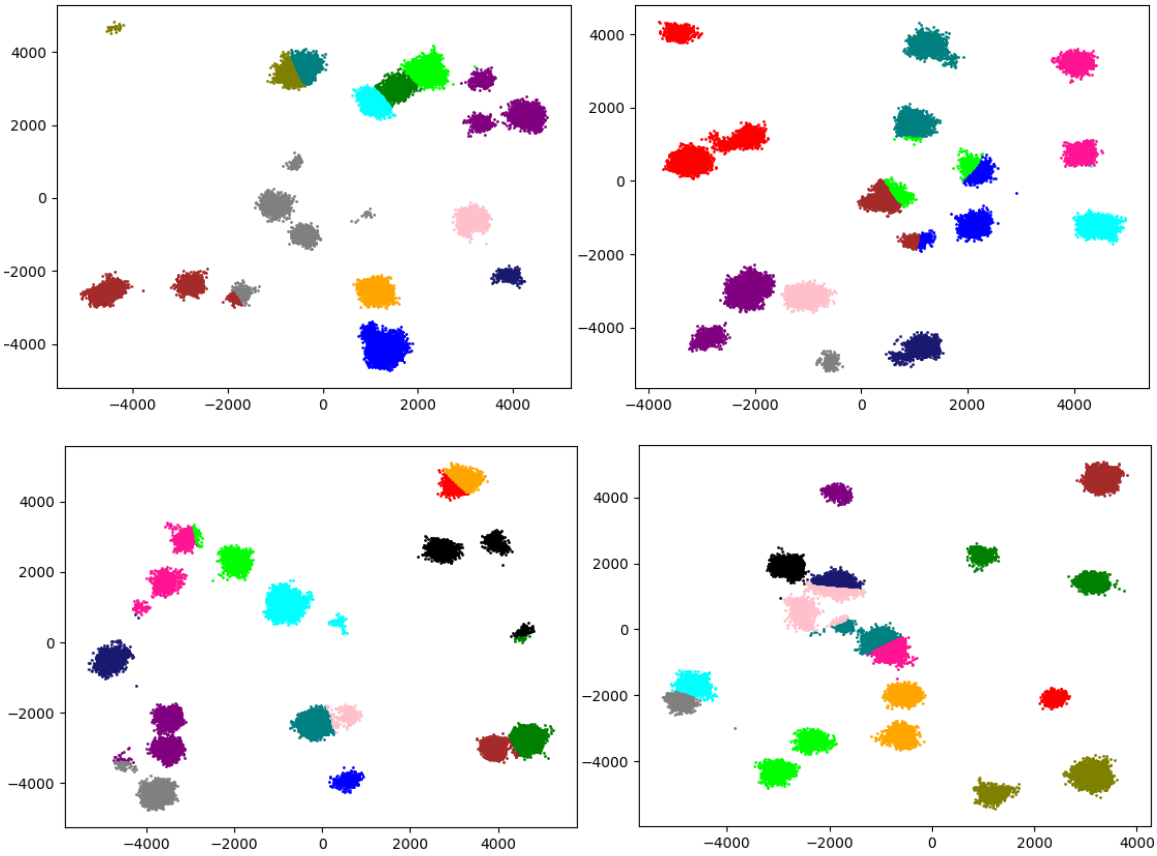
```
Optimalny pocet zhlukov podla lakta: 9

Priemerne vzdialenosti bodov od ich zhlukov: 506.8856709647428
```

```
Optimalny pocet zhlukov podla lakta: 12

Priemerne vzdialenosti bodov od ich zhlukov: 351.9462217755503
```

```
Optimalny pocet zhlukov podla lakta: 11

Priemerne vzdialenosti bodov od ich zhlukov: 482.5148259088958
```

```
Optimalny pocet zhlukov podla lakta: 14

Priemerne vzdialenosti bodov od ich zhlukov: 256.52741345193465
```

```
Optimalny pocet zhlukov podla lakta: 14

Priemerne vzdialenosti bodov od ich zhlukov: 314.20760184395425
```



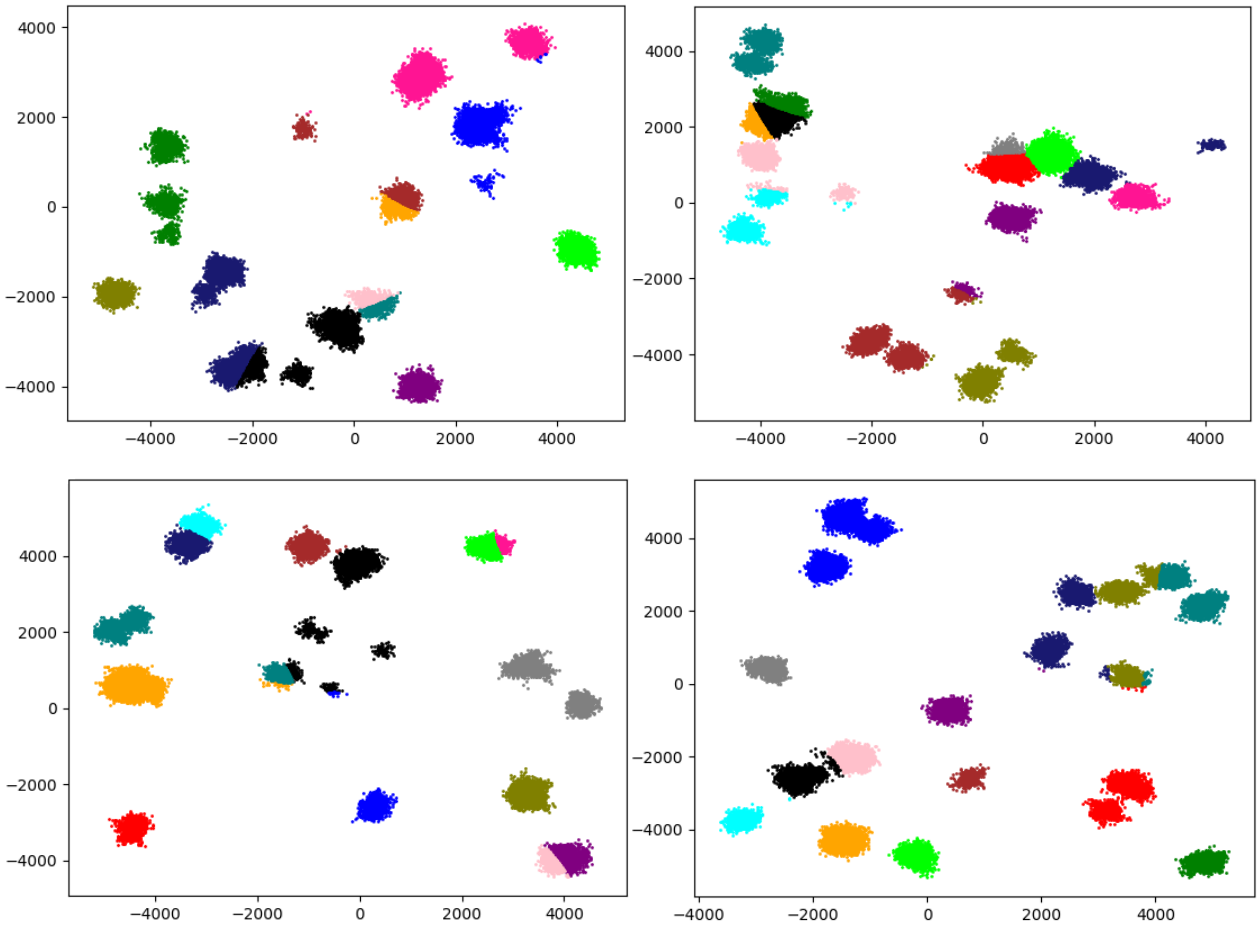
4 testy boli spravené na k-means s medoidom s použitím „elbow method“. Výsledky vyzerajú nasledovne:

```
Optimalny pocet zhlukov podla lakta: 12

Priemerne vzdialenosti bodov od ich zhlukov: 419.59007910415613
```

```
Optimalny pocet zhlukov podla lakta: 14

Priemerne vzdialenosti bodov od ich zhlukov: 339.9327820410798
```

```
Optimalny pocet zhlukov podla lakta: 14

Priemerne vzdialenosti bodov od ich zhlukov: 398.69127102738463
```

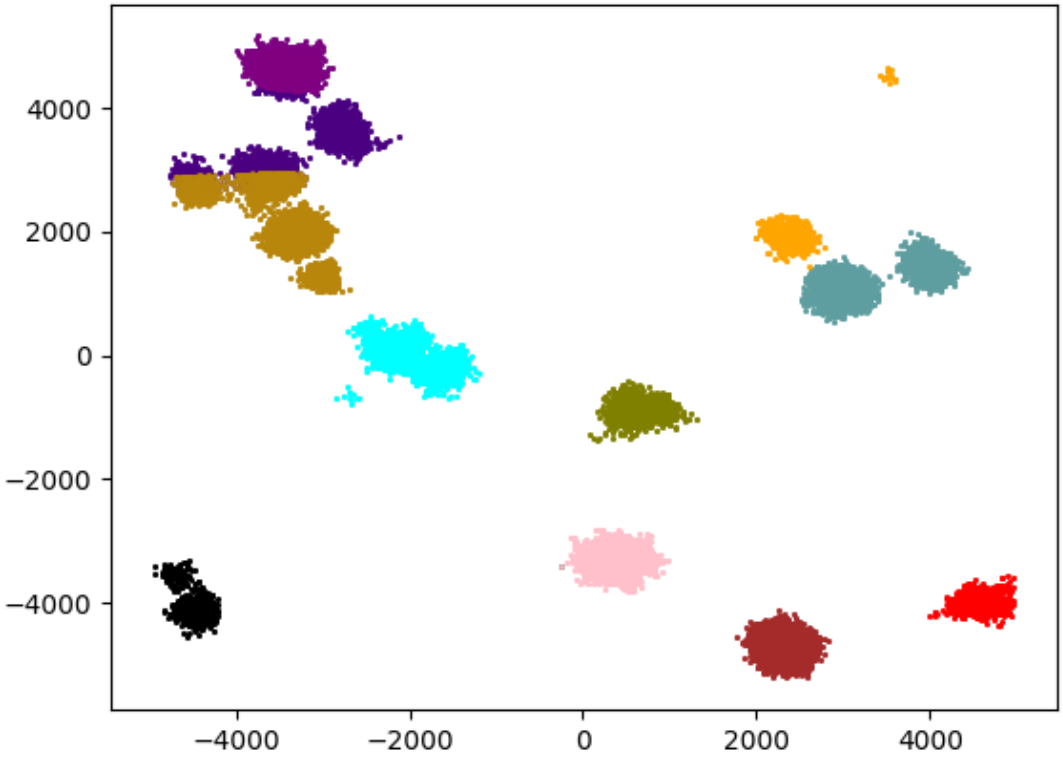
```
Optimalny pocet zhlukov podla lakta: 14

Priemerne vzdialenosti bodov od ich zhlukov: 411.2493397996822
```

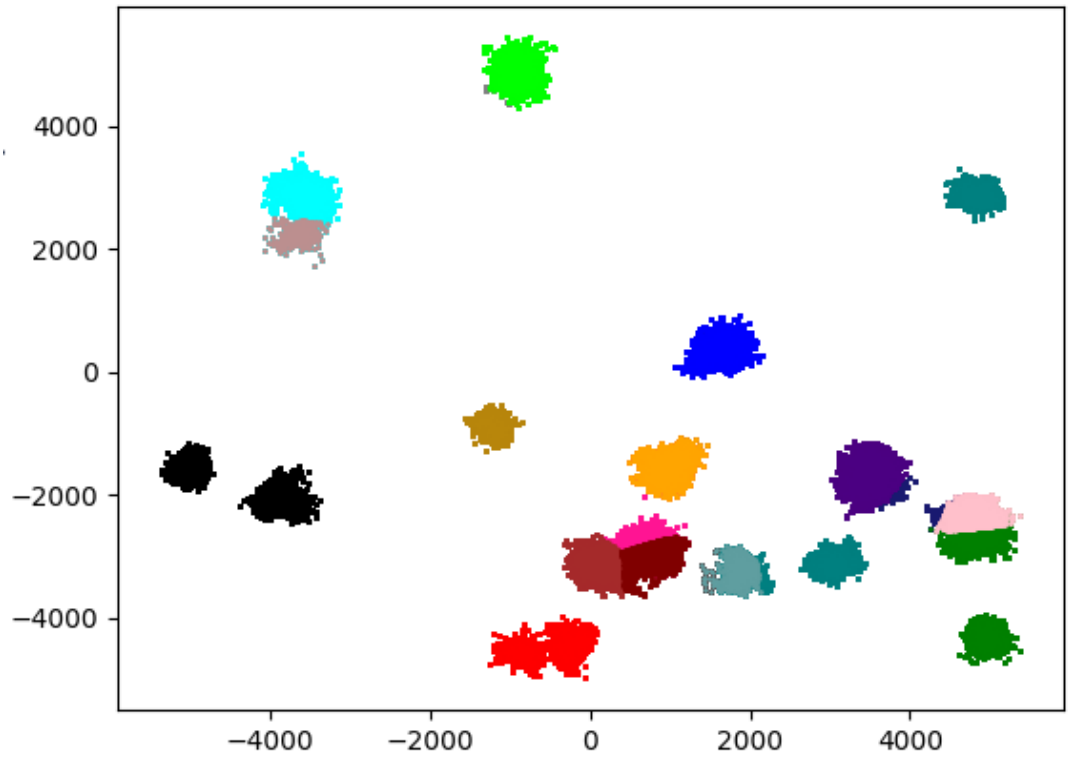


2 testy boli spravené na divízne zhlukovanie. Výsledky vyzerajú nasledovne:

```
Priemerne vzdialenosti bodov od ich zhlukov: 486.2528341428692
```

```
Priemerne vzdialenosti bodov od ich zhlukov: 499.58106297165534
```





Zhrnutie

Z testovania a implementácie tohto zadania vyplýva, že toto riešenie obsahuje zvládnutie problému, ktoré zadanie opisuje. Nie vždy sa podarí algoritmu dostať priemer vzdialeností bodov od zhlukov pod 500, ale pri testovaní sú aj tak na výstupe prijateľné výsledky. Čo sa týka efektivity, tak program v tejto dokumentácii má celkovo dobrú časovú náročnosť, ale je pomerne náročný na pamäť, ale to vychádza z charakteristiky týchto algoritmov.