Izbirni projekt razpoznavanje vzorcev: 2b – Hierarhičen postopek iskanja rojev

Mentor: as. dr. Klemen Grm, mag. inž. el.

Avtor: Blaž Ardaljon Mataln Smehov

Datum: 27.12.2021

Osnovna naloga:

Pri osnovni nalogi smo morali program preizkusiti na šolski učni množici iz zgleda v knjigi. Učna množica je sestavljena iz osmih točk, ki so opisane z dvema koordinatama v kartezičnem koordinatnem sistemu.

V prvem koraku sem moral iz tekstovne datoteke prebrati točke, shranjene vrednosti so bile formata string, zato sem jih moral spremeniti v tip float to sem storil s tema funkcijama:

```
    with open('D:/ljubljana/razpoznavanje_vzorcev_real/izbirni_projekt/vaja2b.txt') as f:
        tocke = f.readlines()

    seznam_tock1 = pretvorba(tocke)
    seznam_tock = np.array(seznam_tock1)
```

```
def pretvorba(seznam_string):
    seznam_pretvorb = []
    for i in range(len(seznam_string)):
        tocka = seznam_string[i].split(",")
        tocka[0] = float(tocka[0])
        tocka[1] = float(tocka[1])
        seznam_pretvorb.append(tocka)
    return seznam_pretvorb
```

Med vsako točko sem izračunal evklidove razdalje in to shranil v numpy matriko. Slednjo sem večkrat kopiral, ker sem potreboval več matrik za računanje:

```
for i in range(len(seznam_tock)):
        for j in range(len(seznam_tock)):
            evklidove_razdalje[i][j] = euc(seznam_tock[i],seznam_tock[j])
```

```
def euc(tocka1, tocka2):
    euc_distance = np.linalg.norm(tocka1 - tocka2)
    return euc_distance
```

V matriki so podane torej evklidske razdalje, v prvem koraku med temi program poišče minimalno in shrani indeksa na katerih se nahaja ta minimalna razdalja:

```
seznam_razdalj = []
    minimum = np.amin(evklidove_razdalje)
    index = np.where(evklidove_razdalje == minimum)

    N = N - 1

    seznam_razdalj.append(minimum)
    seznam_razdalj_2 = seznam_razdalj.copy()
    seznam_razdalj_3 = seznam_razdalj.copy()

    lista_tock = list(zip(index[0], index[1]))
    tocke = []

    for cord in lista_tock:
        if(cord in tocke or cord[::-1] in tocke):
            continue
        tocke.append(cord)

    tocke1 = []

    for i in range(len(tocke)):
        tocka1 =  tocke[i]
        tocke1.append(tocka1[0])
        tocke1.append(tocka1[1])
```

ta indeksa ustrezata točkama med katerima je ta minimalna evklidska razdalja. Minimalno razdaljo v matriki sem poiskal s funkcijo iz knjižnice numpy np.amin. Indeksa na katerih se nahaja ta minimalna vrednost pa sem poiskal s funkcijo iz knjižnice numpy np.where. Minimalna razdalja je nato dodana v seznam, ki beleži na katerih razdaljah so se križali roji. Najdenima indeksoma priredimo točke s for zanko na zgornji sliki, pri tem program preveri ali sta dani točki že vpisani v seznam, to je pomembno saj je matrika evklidovih razdalj simetrična, v tem primeru se minimum pojavi 2x in imamo duplirani vrednosti indeksov.

Kot sem prej omenil sem naredil več kopij matrike evklidskih razdalj, ena je namenjena računanju razdalj med novo nastalimi roji, ena je namenjena iskanju minimumov. V matriko za iskanje minimumov sem na indeksa na katerih je ležal minimum vpisal vrednosti 1000, tako program v nadaljevanju ne bo več upošteval teh minimalnih vrednosti:

```
for i in tocke1:
        for j in range(len(seznam_tock)):
            evklidove_razdalje[i][j] = 1000
            evklidove_razdalje[j][i] = 1000
```

Koda do tega koraka je identična za vse tri načine računanja evklidovih razdalj med roji, v nadaljevanju sem naredil funkcije in vsaki funkciji podal svoje matrike. Torej naredil sem tri kopije matrik za vsakega izmed možnih načinov računanja.

Klic funkcije za računanje rojev:

```
tocke, seznam_razdalj, euc_za_rac = racunanje_rojev(evklidove_razdalje,
euc_za_rac, tocke1, stevec, N, seznam_razdalj, tocke, 1, stevilo_vzorcev1)
    tocke2, seznam_razdalj_2, euc_za_rac1 =
racunanje_rojev(evklidove_razdalje1, euc_za_rac1, tocke1_1, stevec1, N1,
seznam_razdalj_2, tocke2, 2, stevilo_vzorcev2)
    tocke3, seznam_razdalj_3, euc_za_rac2 =
racunanje_rojev(evklidove_razdalje2, euc_za_rac2, tocke1_2, stevec2, N3,
seznam_razdalj_3, tocke3, 3, stevilo_vzorcev3)
```

V zgornjih klicih funkcije so: evklidove_razdalje – matrika evklidovih razdalj za iskanje minimumov, euc_za_rac – matrika razdalj v katero ne vpisujem vrednosti 1000, v njej program bere vrednosti za nadaljnje računanje rojev, tocke1 – seznam točk po vrsti kako se sekajo, seznam_razdalj – seznam v katerem so razdalje na katerih se te točke sekajo.

Funkcija je zelo podobna prejšnjemu delu kode, kjer program išče minimume in indekse ter na ta mesta vpisuje vrednosti 1000, zato bom v ta dokument prilepil le del kode do tega dela, ko se računajo minimumi in indeksi:

```
def racunanje_rojev(evklidove_razdalje,euc_za_rac,tocke1,stevec,N,seznam_razdalj,tocke,a,stevilo_vzorcev):
    if(a == 1):
        a_i = 0.5
        a_j = 0.5
        b = 0
        c = -0.5
    if(a == 2):
        a_i = 0.5
        a_j = 0.5
        b = 0
        c = 0.5
    while(N > 1):

        seznam_razdalj1 = []

        for i in range(len(evklidove_razdalje)):
            if(a == 3):
                imenovalec = (stevilo_vzorcev[i] + stevilo_vzorcev[tocke1[stevec+1]] + stevilo_vzorcev[tocke1[stevec]])
                a_i = (stevilo_vzorcev[i] + stevilo_vzorcev[tocke1[stevec]])/imenovalec
                a_j = (stevilo_vzorcev[i] + stevilo_vzorcev[tocke1[stevec+1]])/imenovalec
                b = -stevilo_vzorcev[i]/imenovalec
                c = 0

            razdalja = a_i*euc_za_rac[tocke1[stevec]][i] + a_j*euc_za_rac[tocke1[stevec+1]][i] + b*euc_za_rac[tocke1[stevec+1]][tocke1[stevec]] + c*abs(euc_za_rac[tocke1[stevec]][i] - euc_za_rac[tocke1[stevec+1]][i])
            seznam_razdalj1.append(razdalja)

        seznam_razdalj1_np = np.array([seznam_razdalj1])

        evklidove_razdalje = np.append(evklidove_razdalje, seznam_razdalj1_np, 0)
        euc_za_rac = np.append(euc_za_rac, seznam_razdalj1_np, 0)

        seznam_razdalj1.append(1000)

        seznam_razdalj2_np = np.array([seznam_razdalj1])

        seznam_razdalj1_rotiran = np.rot90(seznam_razdalj2_np, axes = (1,0))
```

```
        evklidove_razdalje = np.append(evklidove_razdalje, seznam_razdalj1_rotiran, 1)
        euc_za_rac = np.append(euc_za_rac, seznam_razdalj1_rotiran, 1)
```

V prvem delu je podan if stavek, odvisno od tega za katero računanje koeficientov se odločimo, potem koda poda, oziroma zračuna koeficiente za računanje novih razdalj med roji.

V naslednjem delu funkcija izračuna nove razdalje po enačbi, ki sem jo našel v knjigi z matriko za računanje. Razdalje program doda v seznam razdalj, ki ga nato pripne matriki evklidovih razdalj, pripne jo na x-osi in y-osi. Da sem lahko pripel matriko po y-osi sem moral najprej seznam rotirati za 90 stopinj, to sem storil s funkcijo rot90.

V naslednjem koraku kličem funkcijo za izračun kofenetične matrike:

```
    kofeneticna_matrika_1 = kofeneticna_matrika(seznam_tock1,tocke,euc_za_rac)
    kofeneticna_matrika_2 =
kofeneticna_matrika(seznam_tock1,tocke2,euc_za_rac1)
    kofeneticna_matrika_3 =
kofeneticna_matrika(seznam_tock1,tocke3,euc_za_rac2)
```

V funkciji za kofenetično matriko sem ustvaril slovar, v slovarju je dejansko zapisano kateri roj je sestavljen iz katerih točk:

```
def kofeneticna_matrika(seznam_tock1,tocke,euc_za_rac):

    l = len(seznam_tock1)
    dict={}
    for i in range(len(tocke)):
        dict[i+l]=list(tocke[i])

    nova = np.zeros((l,l))

    for k in dict:
        h = dict[k]
        e = h.copy()
        if e[0]>l-1:
            e[0] = dict[e[0]]
        else:
            e[0]=[e[0]]
        if e[1]>l-1:
            e[1] = dict[e[1]]
        else:
            e[1]=[e[1]]
        c = list(itertools.product(e[0], e[1]))
        for x in c:
            x=list(x)
            if(x[1] > x[0]):
```

```
                x_1 = x[0]
                x[0] = x[1]
                x[1] = x_1
            nova[x[0]][x[1]] = euc_za_rac[h[0]][h[1]]

        enovi=[]
        enovi.extend(e[0])
        enovi.extend(e[1])
        dict[k] = enovi

    return nova
```

To dosežem z zanko, ker so točke indeksirane od 0 do 7, se prvi roj v matriki evklidskih razdalj pojavi na indeksu 8. V naslednji for zanki program preveri, ali je točka v tem seznamu večja od 7, če je pomeni da je ta točka dejansko roj in namesto roja vpiše katere točke so. Ko imamo indekse točk v posamičnem roju lahko pogledamo na kateri razdalji se prvič srečajo v seznamu evklidovih razdalj.

Vse kaj ostane je še računanje koeficientov. Koeficienta Q in CPCC sem izračunal po formuli, ki je podana v knjigi. V tem primeru gre za enostavno zanko, s katero program računa vsoto vrste:

```
def izracun_Q(kofeneticna_matrika_1,evklidove_razdalje_za_Q):
    stevec = 0
    imenovalec = 0
    for i in range(len(kofeneticna_matrika_1)-1):
        for j in range(i+1,len(kofeneticna_matrika_1)):
            stevec = stevec + (evklidove_razdalje_za_Q[j][i] -
kofeneticna_matrika_1[j][i])**2
            imenovalec = imenovalec + (evklidove_razdalje_za_Q[j][i])**2
    Q = stevec/imenovalec
    return Q
```

Podobno je zapisana funkcija za izračun koeficienta CPCC:

```
def izracun_CPCC(kofeneticna_matrika_1,evklidove_razdalje_za_Q):
    N = len(kofeneticna_matrika_1)
    R = N*(N - 1)/2
    stevec1 = 0
    D = 0
    D_c = 0
    imenovalec1 = 0
    imenovalec2 = 0

    for i in range(N - 1):
        for j in range(i+1,N):
            stevec1 = stevec1 +
evklidove_razdalje_za_Q[j][i]*kofeneticna_matrika_1[j][i]
            D = D + evklidove_razdalje_za_Q[j][i]
            D_c = D_c + kofeneticna_matrika_1[j][i]
            imenovalec1 = imenovalec1 + evklidove_razdalje_za_Q[j][i]**2
            imenovalec2 = imenovalec2 + kofeneticna_matrika_1[j][i]**2

    crta_D = (1/R)*D
    crta_D_c = (1/R)*D_c
    stevec1 = (1/R)*stevec1
    stevec2 = crta_D*crta_D_c
    stevec = stevec1 - stevec2

    imenovalec1 = math.sqrt((1/R)*imenovalec1-crta_D**2)
    imenovalec2 = math.sqrt((1/R)*imenovalec2-crta_D_c**2)
    imenovalec = imenovalec1*imenovalec2

    CPCC = stevec/imenovalec
    return CPCC
```

Rez dendrograma se izvede na točki, kjer je razlika med dvema zaporednima evklidskima razdaljama na dendrogramu največja. Da sem dobil točko reza, sem samo računal razdalje med zaporednimi indeksi in poiskal največjo, pri tej sem shranil vrednost indeksa:

```
    maksimum = 0
    for i in range(len(seznam_razdalj_3)-1):
        razdalja = seznam_razdalj_3[i+1] - seznam_razdalj_3[i]
        if razdalja > maksimum:
            maksimum = razdalja
            spodnja_meja_reza = seznam_razdalj_3[i]
            index = i
```

Z določeno mejo lahko potem izračunamo koliko vzorcev pripada porezanim rojem, to sem storil z uporabo konfenetične matrike. V njej je zapisano na kateri razdalji se prvič v istem roju pojavita dve točki:

```
    indeksi_roj_za_rac = []
    seznam_razdalj_za_rac = seznam_razdalj_3[index:]

    for i in range(len(seznam_razdalj_za_rac)):
        vrednost = seznam_razdalj_za_rac[i]
        indeks_roj = np.where(kofeneticna_matrika_3 == vrednost)
        lista_tock = list(zip(indeks_roj[0],indeks_roj[1]))
        tocke_roj = []
        for cord in lista_tock:
            if(all(cord[0] not in k for k in indeksi_roj_za_rac)):
                if(cord[0] not in tocke_roj):
                    tocke_roj.append(cord[0])
            if(all(cord[1] not in k for k in indeksi_roj_za_rac)):
                if(cord[1] not in tocke_roj):
                    tocke_roj.append(cord[1])

        indeksi_roj_za_rac.append(tocke_roj)
```

Najprej sem odrezal seznam razdalj od točke reza naprej. Prva točka je spodnja meja reza, število točk, ki je na spodnji meji reza lahko izračunamo s pomočjo for zanke. Ponovno sem s funkcijo np.where poiskal indekse teh točk, ti indeksi ustrezajo točkam, ki so v tem roju. Ker je matrika zrcalna je ponovno potrebno preveriti, ali je indeks že zapisan v seznamu, hkrati je potrebno preveriti ali je indeks že zapisan v prejšnjem seznamu. Rezultat zanke je seznam seznamov, število seznamov je število rojev po rezu, število vzorcev v seznamu je število točk, ki pripada temu roju.

Rezultati:

```
rezultat 1:
[(4, 5), (0, 1), (3, 8), (6, 10), (2, 9), (11, 12), (7, 13)]
[2.0, 2.23606797749979, 2.23606797749979, 2.23606797749979, 2.82842712474619, 5.0, 5.385164807134505]
rezultat 2:
[(4, 5), (0, 1), (2, 9), (3, 8), (6, 7), (11, 12), (10, 13)]
[2.0, 2.23606797749979, 3.0, 4.123105625617661, 6.0, 8.48528137423857, 15.620499351813308]
rezultat 3:
[(4, 5), (0, 1), (2, 9), (3, 8), (6, 11), (7, 12), (10, 13)]
[2.0, 2.23606797749979, 3.140262090664197, 3.5727824020783, 4.786391201039151, 8.437593446898877, 26.426128508762936]
```

Rezultat 1, 2 in 3 se nanašajo na načine računanja razdalj med novo nastalimi roji. V zgornjem seznamu so podane točke, oziroma roji v zaporedju po katerem se združujejo, v drugem seznamu so evklidske razdalje pri katerih se to zgodi.

Vrednosti Q in CPCC za različne načine računanja:

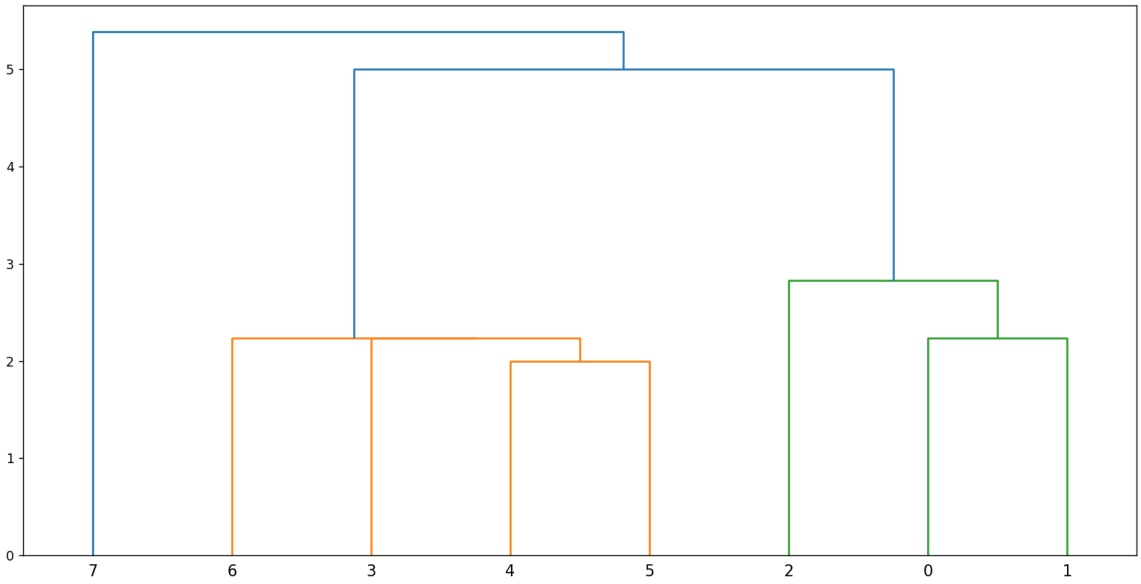
```
Q1:
0.26609435681995947
CPCC1:
0.7327997359043674
Q2:
0.3869005717526838
CPCC2:
0.7705630562309194
Q3:
2.302917893037251
CPCC3:
0.7698088221503798
```

Primer kofenetične matrike za prvi primer računanja:

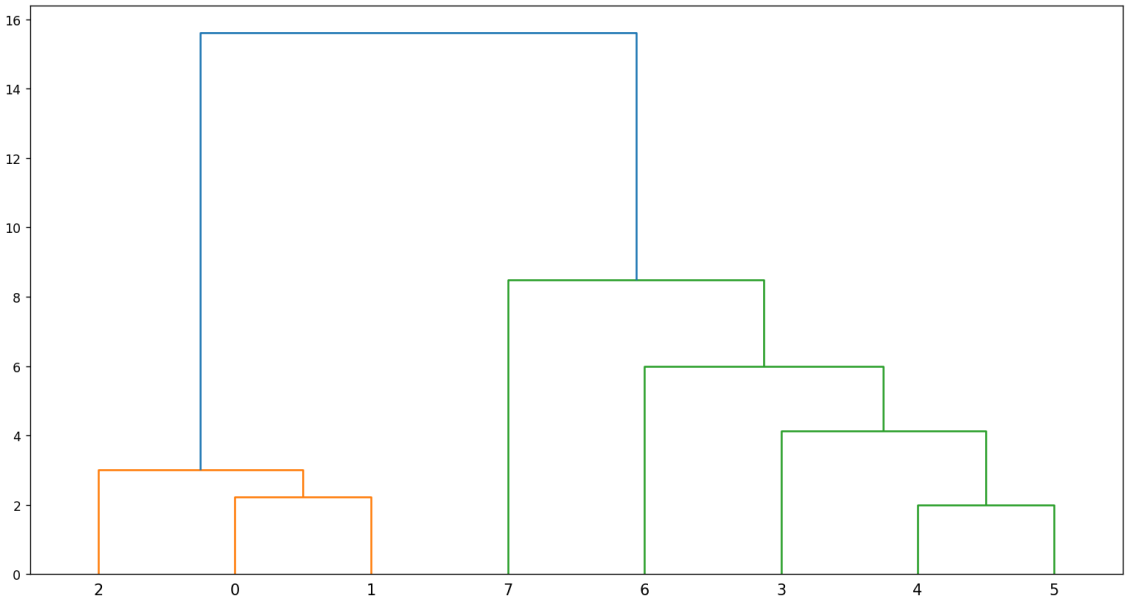
	0	1	2	3	4	5	6	7
0	1000	2,236068	3	7,211103	9,433981	11,18034	12,64911	15,6205
1	2,236068	1000	2,828427	5,385165	7,615773	9,486833	11,18034	13,60147
2	3	2,828427	1000	5	7,071068	8,602325	9,848858	13,45362
3	7,211103	5,385165	5	1000	2,236068	4,123106	6	8,485281
4	9,433981	7,615773	7,071068	2,236068	1000	2	4,123106	6,403124
5	11,18034	9,486833	8,602325	4,123106	2	1000	2,236068	5,385165
6	12,64911	11,18034	9,848858	6	4,123106	2,236068	1000	6
7	15,6205	13,60147	13,45362	8,485281	6,403124	5,385165	6	1000

Ostale kofenetične matrike so v priloženi mapi.

Dendrogram za prvi način računanja:



Dendrogram za drugi način računanja:



Dodatna naloga:

Pri dodatni nalogi smo morali enako kodo preizkusiti še na testni zbirki iris. Ker je koda zelo podobna osnovni nalogi bom opisal samo dele, ki so različni. Prvo sem datoteko data spremenil v tekstovno datoteko in jo potem v programu prebral na enak način kot točke v osnovni nalogi. Ker so v tem primeru točke 4d sem malo spremenil funkcijo za pretvorbo:

```
def pretvorba(seznam_string):
    seznam_pretvorb = []
    for i in range(len(seznam_string)-1):
        tocka = seznam_string[i].split(",")
        tocka[0] = float(tocka[0])
        tocka[1] = float(tocka[1])
        tocka[2] = float(tocka[2])
        tocka[3] = float(tocka[3])
        seznam_pretvorb.append(tocka[:4])
        if i == 1:
            print(seznam_pretvorb)

    return seznam_pretvorb
```

Naslednja sprememba je v zanki za izračun tega kateri vzorci so v katerem roju po rezu:

```
    for i in range(len(seznam_razdalj_za_rac)):
        vrednost = seznam_razdalj_za_rac[i]
        if(vrednost == 15):
            vrednost = seznam_razdalj_2[index-1]
        indeks_roj = np.where(kofeneticna_matrika_2 == vrednost)
        lista_tock = list(zip(indeks_roj[0],indeks_roj[1]))
        tocke_roj = []
        for cord in lista_tock:
            if(all(cord[0] not in k for k in indeksi_roj_za_rac)):
                if(cord[0] not in tocke_roj):
                    tocke_roj.append(cord[0])
            if(all(cord[1] not in k for k in indeksi_roj_za_rac)):
                if(cord[1] not in tocke_roj):
                    tocke_roj.append(cord[1])
        print("vrednost: ")
        print(vrednost)
        indeksi_roj_za_rac.append(tocke_roj)
        print(len(indeksi_roj_za_rac[i]))
```

V bistvu vse kaj sem dodal, je to da si izpišeš dolžino seznama točk, tako vem koliko elementov je v posamičnem roju.

Rezultati:

V primerjavi z osnovnim primerom je pri tej nalogi 150 točk in izpis rezultata 1, 2 in 3 ni smiseln. Vseeno je v kodi funkcija print() za izpis, vendar slike ne bom prilepil sem.

Izračun Q in CPCC:

```
Q1:
0.4202333788835364
CPCC1:
0.8509486565778696
Q2:
0.7193953903905048
CPCC2:
0.3450889671513577
Q3:
2.251758587442006
CPCC3:
-0.430634435047725
```

CPCC je za 3 primer očitno napačen, saj bi morala biti vrednost pozitivna med 0 in 1. Za točke iz osnovnega primera je enaka funkcija delovala pravilno.

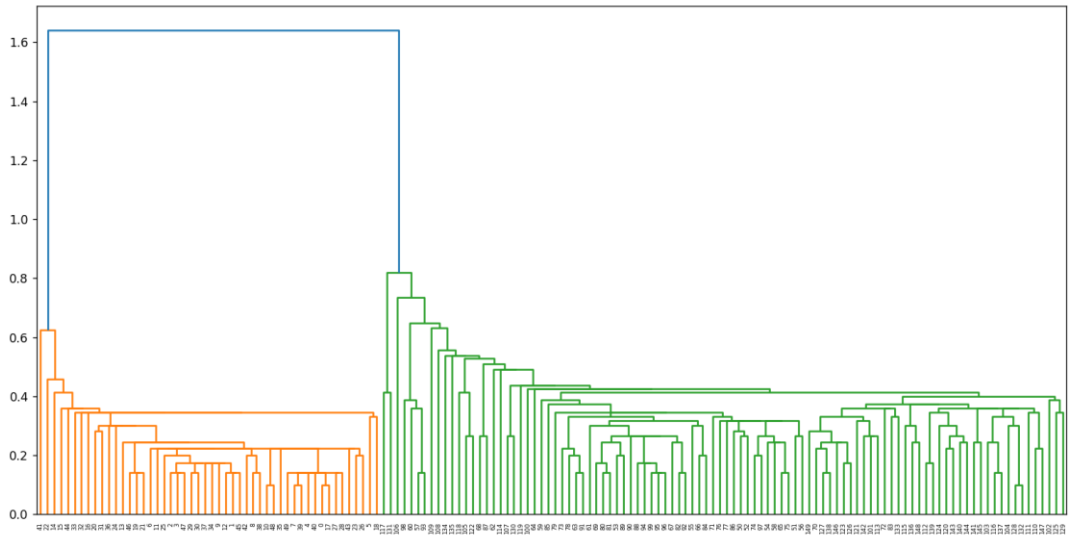
Kofenetične matrike za dodatno nalogo so prav tako priložene, ker je 150 točk tega primera ne bom prilepil v ta dokument.

Dendrogrami:

Za prvi način računanja:

Število vzorcev v posamičnem roju: (100 in 50)

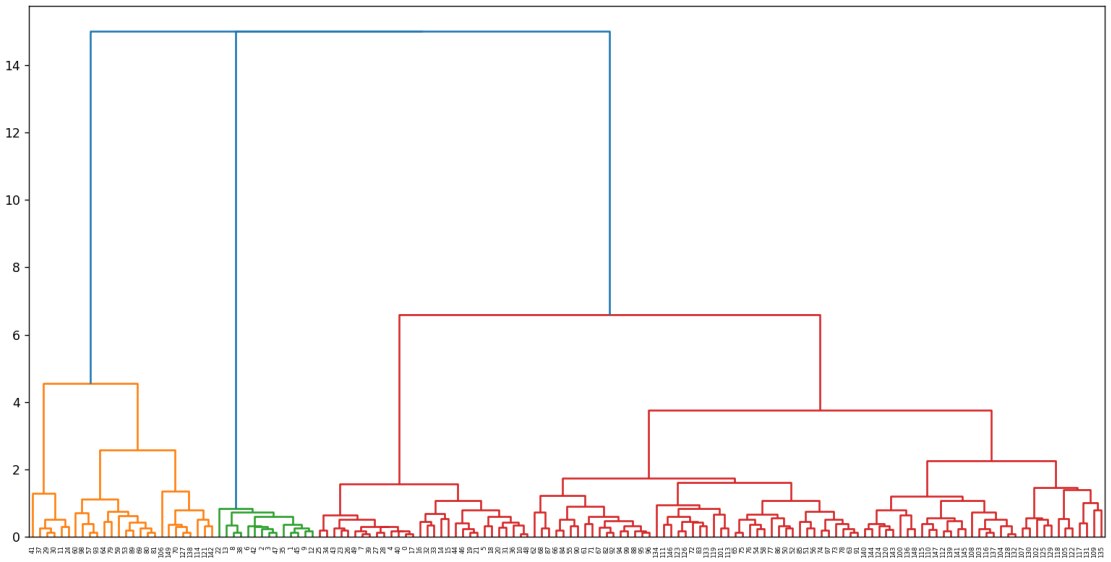
```
vrednost:
0.818535277187245
100
vrednost:
1.6401219466856727
50
```



Za drugi način:

Število vzorcev po rojih: (110, 26, 14)

```
6.592419889539803
110
vrednost:
4.559605246071198
26
vrednost:
0.9949874371066199
14
```



Rezultati za 3 način: dendrograma ne znam izrisat.

```
vrednost:
63.87923467543701
85
vrednost:
127.64237612064494
65
```