

**EGE UNIVERSITY**

**FACULTY OF ENGINEERING**

**COMPUTER ENGINEERING DEPARTMENT**

**204 DATA STRUCTURES (3+1)**

**2024–2025 FALL SEMESTER**

**PROJECT-1 REPORT**

**(Arrays, Matrices, Methods, Classes, Random Numbers)**

**DELIVERY DATE**

16/11/2024

**PREPARED BY**

05230000334, İrfan Burak Ege

, Yusuf Elkaan

, Aref Faraji

İçindekiler

[1) KARAYOLLARI UZAKLIK HESAPLAMALARI 2](#_Toc180684140)

[1.a Uzaklık Matrisi Oluşturma ve 10 Şehir Çifti İçin Şehir Uzaklıkları 2](#_Toc180684141)

[1.a.1 Kodlar 2](#_Toc180684142)

[1.a.2 Ekran görüntüleri 2](#_Toc180684143)

[1.b Uzaklık Matrisinin Komşu İllere Göre Düzenlenmesi 2](#_Toc180684144)

[1.b.1 Kodlar 2](#_Toc180684145)

[1.b.2 Ekran görüntüleri 2](#_Toc180684146)

[1.c Komşu Olmayan Şehir Çiftleri Arasındaki En Kısa Yolların Dijkstra Algoritması ile Hesaplanması ve Fark Değerlerinin Hesaplanması 2](#_Toc180684147)

[1.c.1 Kodlar 2](#_Toc180684148)

[1.c.2 Ekran görüntüleri 2](#_Toc180684149)

[1.d İzmir’ in İlçelerinin Uzaklık Matrisinin Oluşturulması 2](#_Toc180684150)

[1.d.1 Kodlar 2](#_Toc180684151)

[1.d.2 Ekran görüntüleri 2](#_Toc180684152)

[1.e İzmir’in İlçeleri için Komşu Olmayan İlçe Çiftleri Arasındaki En Kısa Yolların Dijkstra Algoritması ile Hesaplanması ve Fark Değerlerinin Hesaplanması 2](#_Toc180684153)

[1.e.1 Kodlar 2](#_Toc180684154)

[1.e.2 Ekran görüntüleri 3](#_Toc180684155)

[2) IMAGE RECOGNITION USING ARTIFICIAL NEURAL NETWORKS 4](#_Toc180684156)

[2.a Veri Seti Oluşturma 4](#_Toc180684157)

[2.a.1 Kaynak Kod 4](#_Toc180684158)

[2.a.2 Açıklama 4](#_Toc180684159)

[2.a.3 Ekran Görüntüleri 4](#_Toc180684160)

[2.b Neuron (Sinir Hücresi) Sınıfı 4](#_Toc180684161)

[2.b.1 Kaynak Kod 4](#_Toc180684162)

[2.b.2 Açıklama 4](#_Toc180684163)

[2.c Neural Network (Yapay Sinir Ağı) Sınıfı 4](#_Toc180684164)

[2.c.1 Kaynak Kod 4](#_Toc180684165)

[2.c2 Açıklama 4](#_Toc180684166)

[2.d Eğitim Metodunun Yazılması 4](#_Toc180684167)

[2.d.1 Kaynak Kod 4](#_Toc180684168)

[2.d.2 Açıklama 4](#_Toc180684169)

[2.e Eğitim 4](#_Toc180684170)

[2.e.1 Kaynak Kod 4](#_Toc180684171)

[2.e.2 Ekran Görüntüleri / Tablolar 4](#_Toc180684172)

[2.f Modelin Görmediği Bir Matristen Sayıyı Tahminleme 4](#_Toc180684173)

[2.d.1 Kaynak Kod 4](#_Toc180684174)

[2.f.2 Sonuçlar / Ekran Görüntüleri 5](#_Toc180684175)

[Öz değerlendirme Tablosu 5](#_Toc180684176)

# 1) KARAYOLLARI UZAKLIK HESAPLAMALARI

Microsoft Visual Studio Community 2022 (64-bit), Version 17.11.6, using C#

## 1.a Uzaklık Matrisi Oluşturma ve 10 Şehir Çifti İçin Şehir Uzaklıkları

### 1.a.1 Kodlar

using System;

using System.Collections;

using System.ComponentModel;

using System.Security.Cryptography.X509Certificates;

using System.Threading.Channels;

namespace project1\_1

{

internal class Program

{

static double INFINITY = double.MaxValue; // defining infinity

static void Main(string[] args

{

//cities name list index 0 is null for plates

string[] citiesNames = {"", "Adana", "Adıyaman", "Afyon", "Ağrı", "Amasya", "Ankara", "Antalya", "Artvin", "Aydın", "Balıkesir", "Bilecik", "Bingöl", "Bitlis", "Bolu", "Burdur", "Bursa", "Çanakkale", "Çankırı", "Çorum", "Denizli", "Diyarbakır", "Edirne", "Elazığ", "Erzincan", "Erzurum", "Eskişehir", "Gaziantep", "Giresun", "Gümüşhane", "Hakkari", "Hatay", "Isparta", "Mersin", "İstanbul", "İzmir", "Kars", "Kastamonu", "Kayseri", "Kırklareli", "Kırşehir", "Kocaeli", "Konya", "Kütahya", "Malatya", "Manisa", "Kahramanmaraş", "Mardin", "Muğla", "Muş", "Nevşehir", "Niğde", "Ordu", "Rize", "Sakarya", "Samsun", "Siirt", "Sinop", "Sivas", "Tekirdağ", "Tokat", "Trabzon", "Tunceli", "Şanlıurfa", "Uşak", "Van", "Yozgat", "Zonguldak", "Aksaray”, "Bayburt", "Karaman", "Kırıkkale", "Batman", "Şırnak", "Bartın", "Ardahan", "Iğdır", "Yalova", "Karabük", "Kilis", "Osmaniye", "Düzce" };

//jagged array that keeps distances of every province indexes are plate numbers

double[][] jaggedArray = new double[82][];

//1'st index is 0 to match the plate numbers

jaggedArray[1] = new double[] { 0, 0 };

jaggedArray[2] = new double[] { 0, 339, 0 };

jaggedArray[3] = new double[] { 0, 578, 917, 0 };

jaggedArray[4] = new double[] { 0, 980, 647, 1320, 0 };

jaggedArray[5] = new double[] { 0, 602, 625, 595, 740, 0 };

jaggedArray[6] = new double[] { 0, 491, 732, 255, 1055, 330, 0 };

jaggedArray[7] = new double[] { 0, 546, 885, 287, 1424, 822, 542, 0 };

jaggedArray[8] = new double[] { 0, 1003, 715, 1224, 364, 679, 960, 1430, 0 };

jaggedArray[9] = new double[] { 0, 880, 1219, 340, 1634, 934, 595, 345, 1564, 0 };

jaggedArray[10] = new double[] { 0, 902, 1241, 324, 1579, 845, 536, 505, 1454, 291, 0 };

jaggedArray[11] = new double[] { 0, 785, 1032, 207, 1361, 626, 314, 472, 1236, 525, 257, 0 };

jaggedArray[12] = new double[] { 0, 633, 347, 1096, 359, 641, 887, 1188, 371, 1398, 1410, 1187, 0 };

jaggedArray[13] = new double[] { 0, 743, 410, 1287, 237, 832, 1078, 1289, 545, 1589, 1601, 1378, 192, 0 };

jaggedArray[14] = new double[] { 0, 688, 920, 417, 1146, 411, 190, 682, 1021, 735, 434, 215, 1075, 1266, 0 };

jaggedArray[15] = new double[] { 0, 647, 986, 166, 1401, 760, 421, 121, 1390, 286, 393, 351, 1165, 1356, 561, 0 };

jaggedArray[16] = new double[] { 0, 856, 1104, 278, 1418, 683, 385, 543, 1293, 441, 152, 95, 1259, 1450, 272, 422, 0 };

jaggedArray[17] = new double[] { 0, 1093, 1383, 515, 1697, 962, 665, 693, 1572, 448, 192, 374, 1538, 1729, 551, 586, 270, 0 };

jaggedArray[18] = new double[] { 0, 590, 762, 398, 985, 245, 131, 685, 881, 737, 666, 447, 918, 1109, 233, 563, 504, 784, 0 };

jaggedArray[19] = new double[] { 0, 582, 697, 504, 830, 91, 240, 732, 720, 844, 785, 563, 759, 950, 348, 670, 634, 914, 155, 0 };

jaggedArray[20] = new double[] { 0, 761, 1100, 221, 1515, 816, 476, 226, 1445, 124, 285, 406, 1279, 1470, 616, 167, 435, 473, 619, 725, 0 };

jaggedArray[21] = new double[] { 0, 539, 206, 1108, 441, 703, 898, 1085, 509, 1409, 1421, 1199, 141, 204, 1086, 1176, 1270, 1550, 929, 770, 1290, 0 };

jaggedArray[22] = new double[] { 0, 1183, 1414, 682, 1641, 906, 685, 922, 1516, 667, 419, 480, 1570, 1761, 495, 826, 428, 231, 727, 843, 702, 1581, 0 };

jaggedArray[23] = new double[] { 0, 496, 283, 959, 499, 535, 749, 1051, 511, 1260, 1272, 1050, 141, 332, 937, 1027, 1121, 1401, 780, 602, 1141, 152, 1432, 0 };

jaggedArray[24] = new double[] { 0, 676, 546, 955, 370, 375, 690, 1059, 376, 1269, 1214, 996, 272, 463, 781, 1036, 1053, 1332, 620, 465, 1150, 407, 1276, 267, 0 };

jaggedArray[25] = new double[] { 0, 809, 521, 1140, 182, 560, 876, 1244, 194, 1454, 1400, 1181, 177, 351, 966, 1221, 1238, 1517, 806, 651, 1335, 315, 1461, 317, 191, 0 };

jaggedArray[26] = new double[] { 0, 690, 952, 140, 1298, 573, 233, 421, 1202, 474, 303, 80, 1107, 1298, 290, 300, 152, 431, 376, 482, 355, 1119, 555, 970, 933, 1118, 0 };

jaggedArray[27] = new double[] { 0, 218, 151, 796, 762, 576, 647, 764, 846, 1098, 1120, 948, 476, 525, 835, 865, 1020, 1299, 678, 613, 979, 321, 1330, 338, 602, 652, 868, 0 };

jaggedArray[28] = new double[] { 0, 720, 704, 867, 539, 322, 603, 1094, 357, 1207, 1098, 879, 537, 711, 664, 1033, 936, 1215, 524, 363, 1088, 675, 1159, 556, 295, 360, 845, 694, 0 };

jaggedArray[29] = new double[] { 0, 780, 677, 1027, 380, 436, 763, 1163, 312, 1367, 1257, 1039, 378, 552, 824, 1139, 1096, 1375, 684, 523, 1248, 516, 1319, 397, 136, 201, 1005, 754, 160, 0 };

jaggedArray[30] = new double[] { 0, 899, 642, 1477, 429, 1169, 1328, 1445, 755, 1778, 1801, 1628, 492, 324, 1516, 1546, 1700, 1979, 1359, 1249, 1659, 471, 2011, 631, 800, 612, 1548, 681, 969, 810, 0 };

jaggedArray[31] = new double[] { 0, 198, 316, 776, 956, 673, 689, 744, 980, 1077, 1100, 983, 610, 719, 886, 845, 1054, 1291, 776, 711, 959, 515, 1380, 472, 748, 786, 887, 194, 791, 851, 875, 0 };

jaggedArray[32] = new double[] { 0, 618, 957, 167, 1372, 723, 383, 131, 1352, 288, 395, 352, 1136, 1327, 563, 29, 423, 588, 525, 632, 169, 1148, 828, 999, 1007, 1193, 301, 836, 995, 1111, 1517, 816, 0 };

jaggedArray[33] = new double[] { 0, 70, 409, 571, 1050, 635, 485, 466, 1073, 811, 895, 778, 703, 813, 681, 574, 849, 1087, 583, 575, 692, 608, 1176, 565, 746, 879, 683, 288, 790, 849, 968, 268, 563, 0 };

jaggedArray[34] = new double[] { 0, 951, 1182, 450, 1409, 674, 453, 715, 1284, 681, 393, 248, 1338, 1529, 263, 594, 244, 314, 495, 611, 649, 1349, 232, 1200, 1044, 1229, 323, 1098, 927, 1087, 1779, 1148, 595, 944, 0 };

jaggedArray[35] = new double[] { 0, 905, 1244, 327, 1647, 922, 582, 447, 1551, 128, 175, 430, 1423, 1614, 607, 388, 325, 327, 725, 831, 226, 1435, 546, 1286, 1282, 1467, 412, 1123, 1194, 1354, 1804, 1103, 390, 898, 566, 0 };

jaggedArray[36] = new double[] { 0, 1014, 726, 1342, 214, 762, 1078, 1447, 203, 1656, 1602, 1383, 382, 452, 1168, 1424, 1440, 1719, 1008, 853, 1537, 520, 1663, 522, 393, 205, 1320, 857, 562, 403, 554, 991, 1395, 1084, 1431, 1669, 0 };

jaggedArray[37] = new double[] { 0, 695, 868, 503, 990, 255, 237, 790, 846, 843, 680, 462, 892, 1082, 247, 669, 518, 798, 106, 192, 724, 957, 741, 789, 625, 810, 481, 784, 489, 649, 1419, 881, 631, 689, 509, 830, 1012, 0 };

jaggedArray[38] = new double[] { 0, 335, 414, 523, 816, 341, 315, 615, 822, 824, 847, 616, 569, 760, 504, 592, 688, 967, 346, 281, 706, 581, 998, 432, 451, 636, 536, 330, 494, 554, 1010, 428, 563, 328, 766, 850, 838, 452, 0 };

jaggedArray[39] = new double[] { 0, 1159, 1391, 659, 1617, 883, 662, 924, 1492, 684, 436, 457, 1546, 1737, 471, 803, 445, 248, 704, 820, 719, 1558, 70, 1409, 1252, 1438, 532, 1307, 1135, 1295, 1987, 1357, 804, 1152, 209, 563, 1640, 718, 975, 0 };

jaggedArray[40] = new double[] { 0, 376, 549, 425, 940, 312, 183, 568, 941, 765, 706, 483, 704, 895, 371, 545, 555, 834, 214, 221, 646, 715, 866, 566, 575, 760, 403, 464, 584, 678, 1145, 562, 516, 369, 634, 752, 962, 319, 133, 842, 0 };

jaggedArray[41] = new double[] { 0, 840, 1072, 340, 1298, 563, 342, 605, 1173, 569, 280, 138, 1227, 1418, 152, 483, 131, 398, 385, 501, 539, 1239, 345, 1090, 933, 1118, 213, 988, 816, 976, 1668, 1038, 485, 833, 113, 453, 1321, 399, 656, 321, 523, 0 };

jaggedArray[42] = new double[] { 0, 357, 696, 226, 1111, 509, 259, 271, 1117, 527, 550, 433, 875, 1066, 455, 295, 504, 741, 354, 419, 409, 887, 908, 738, 746, 932, 337, 575, 781, 850, 1256, 555, 266, 350, 676, 553, 1134, 460, 302, 885, 255, 566, 0 };

jaggedArray[43] = new double[] { 0, 675, 1014, 97, 1377, 652, 312, 361, 1281, 414, 227, 110, 1193, 1384, 321, 240, 182, 419, 455, 561, 296, 1204, 586, 1055, 1012, 1197, 79, 893, 924, 1084, 1573, 873, 242, 668, 354, 334, 1399, 567, 619, 562, 482, 243, 323, 0 };

jaggedArray[44] = new double[] { 0, 399, 186, 861, 593, 457, 652, 953, 605, 1163, 1175, 953, 235, 426, 840, 930, 1024, 1304, 683, 524, 1044, 246, 1335, 97, 361, 411, 872, 241, 536, 491, 725, 375, 902, 468, 1103, 1189, 615, 711, 335, 1312, 469, 992, 641, 958, 0 };

jaggedArray[45] = new double[] { 0, 885, 1224, 307, 1627, 902, 562, 427, 1531, 152, 138, 393, 1403, 1594, 570, 368, 288, 328, 705, 811, 206, 1415, 556, 1266, 1262, 1447, 392, 1103, 1174, 1334, 1784, 1083, 370, 878, 529, 37, 1649, 810, 830, 572, 732, 417, 533, 314, 1169, 0 };

jaggedArray[46] = new double[] { 0, 199, 162, 781, 815, 500, 572, 746, 827, 1083, 1095, 873, 457, 572, 760, 850, 944, 1224, 603, 538, 964, 368, 1255, 319, 575, 633, 792, 78, 618, 678, 759, 176, 822, 269, 1023, 1109, 837, 708, 254, 1232, 389, 912, 560, 878, 222, 1089, 0 };

jaggedArray[47] = new double[] { 0, 551, 294, 1129, 518, 798, 980, 1097, 604, 1430, 1453, 1281, 236, 281, 1168, 1198, 1352, 1632, 1011, 865, 1312, 94, 1663, 247, 501, 410, 1200, 333, 770, 611, 376, 527, 1169, 620, 1431, 1456, 614, 1051, 663, 1640, 797, 1321, 908, 1225, 341, 1436, 411, 0 };

jaggedArray[48] = new double[] { 0, 855, 1194, 348, 1642, 943, 604, 309, 1572, 98, 389, 533, 1406, 1597, 743, 240, 539, 547, 746, 853, 150, 1418, 766, 1269, 1277, 1463, 482, 1073, 1215, 1375, 1754, 1053, 269, 775, 776, 226, 1665, 852, 833, 782, 773, 666, 536, 423, 1172, 251, 1055, 1406, 0 };

jaggedArray[49] = new double[] { 0, 743, 455, 1206, 246, 751, 996, 1298, 464, 1507, 1520, 1297, 111, 81, 1185, 1275, 1369, 1648, 1027, 869, 1389, 249, 1679, 251, 381, 270, 1217, 586, 630, 471, 381, 719, 1246, 813, 1447, 1533, 347, 1001, 679, 1656, 814, 1337, 985, 1302, 344, 1513, 566, 362, 1516, 0 };

jaggedArray[50] = new double[] { 0, 287, 497, 444, 888, 355, 275, 536, 894, 745, 768, 576, 652, 843, 463, 513, 648, 927, 306, 295, 627, 664, 958, 515, 523, 709, 496, 413, 567, 627, 1093, 485, 484, 280, 726, 771, 911, 412, 79, 935, 92, 616, 223, 540, 418, 751, 338, 746, 754, 762, 0 };

jaggedArray[51] = new double[] { 0, 207, 546, 461, 937, 435, 337, 541, 943, 762, 785, 635, 701, 892, 544, 530, 706, 976, 386, 375, 644, 713, 1039, 564, 572, 757, 555, 425, 616, 676, 1105, 405, 501, 200, 806, 788, 960, 492, 128, 1015, 173, 696, 240, 558, 467, 768, 386, 758, 771, 811, 80, 0 };

jaggedArray[52] = new double[] { 0, 707, 730, 823, 584, 278, 558, 1050, 402, 1162, 1053, 835, 582, 756, 620, 988, 891, 1171, 480, 319, 1044, 720, 1114, 601, 340, 405, 801, 681, 45, 205, 1014, 779, 951, 777, 882, 1150, 607, 445, 482, 1091, 540, 772, 737, 880, 562, 1130, 605, 815, 1171, 675, 614, 603, 0 };

jaggedArray[53] = new double[] { 0, 917, 770, 1076, 429, 531, 811, 1303, 148, 1415, 1306, 1088, 426, 600, 873, 1242, 1145, 1424, 733, 572, 1297, 564, 1368, 560, 299, 249, 1054, 891, 209, 164, 858, 989, 1204, 987, 1135, 1403, 344, 698, 692, 1344, 793, 1025, 990, 1133, 654, 1383, 816, 659, 1424, 519, 764, 813, 254, 0 };

jaggedArray[54] = new double[] { 0, 803, 1034, 302, 1261, 526, 305, 567, 1136, 620, 319, 100, 1190, 1381, 115, 446, 157, 436, 347, 463, 501, 1201, 380, 1052, 896, 1081, 176, 950, 779, 939, 1631, 1000, 448, 796, 148, 492, 1283, 361, 618, 357, 486, 37, 528, 206, 955, 455, 875, 1283, 629, 1299, 578, 659, 734, 988, 0 };

jaggedArray[55] = new double[] { 0, 753, 754, 676, 731, 131, 411, 903, 548, 1016, 906, 688, 718, 909, 473, 842, 745, 1024, 333, 172, 897, 832, 968, 664, 451, 551, 654, 705, 191, 351, 1160, 802, 804, 746, 736, 1003, 753, 298, 453, 944, 393, 625, 590, 733, 586, 983, 629, 927, 1024, 828, 467, 546, 147, 400, 588, 0 };

jaggedArray[56] = new double[] { 0, 717, 384, 1294, 333, 890, 1085, 1264, 641, 1596, 1608, 1386, 288, 96, 1273, 1363, 1457, 1737, 1116, 957, 1477, 187, 1768, 339, 558, 447, 1306, 499, 807, 648, 280, 694, 1335, 787, 1536, 1622, 547, 1144, 768, 1745, 902, 1426, 1074, 1391, 433, 1602, 547, 235, 1605, 177, 851, 900, 852, 696, 1388, 1019, 0 };

jaggedArray[57] = new double[] { 0, 865, 877, 673, 891, 254, 406, 960, 709, 1012, 858, 640, 859, 1050, 425, 838, 696, 976, 275, 260, 894, 956, 920, 788, 593, 712, 651, 828, 352, 512, 1321, 926, 800, 858, 687, 1000, 914, 179, 541, 896, 489, 577, 629, 745, 710, 980, 753, 1050, 1021, 969, 581, 661, 307, 561, 540, 161, 1143, 0 };

jaggedArray[58] = new double[] { 0, 428, 412, 706, 618, 221, 442, 811, 624, 1020, 987, 765, 479, 670, 630, 787, 836, 1116, 442, 287, 901, 490, 1125, 322, 253, 438, 684, 402, 296, 356, 969, 499, 759, 497, 892, 1033, 640, 474, 202, 1101, 326, 782, 498, 763, 244, 1013, 326, 585, 1029, 589, 275, 324, 307, 494, 745, 350, 677, 473, 0 };

jaggedArray[59] = new double[] { 0, 1081, 1313, 581, 1539, 805, 584, 846, 1414, 617, 369, 379, 1468, 1659, 394, 725, 379, 181, 626, 742, 652, 1480, 148, 1331, 1174, 1360, 454, 1229, 1057, 1217, 1909, 1279, 726, 1074, 131, 496, 1562, 640, 897, 124, 764, 243, 807, 484, 1234, 506, 1154, 1562, 716, 1578, 857, 937, 1013, 1266, 279, 866, 1667, 818, 1023, 0 };

jaggedArray[60] = new double[] { 0, 492, 515, 645, 674, 112, 380, 872, 616, 984, 926, 703, 582, 773, 521, 810, 775, 1054, 333, 178, 866, 594, 1016, 426, 309, 494, 623, 466, 259, 370, 1073, 564, 772, 562, 784, 972, 696, 365, 267, 992, 332, 673, 559, 702, 348, 952, 391, 688, 993, 692, 339, 388, 215, 468, 636, 241, 781, 364, 111, 914, 0 };

jaggedArray[61] = new double[] { 0, 838, 770, 997, 474, 452, 733, 1224, 227, 1337, 1228, 1009, 471, 646, 794, 1163, 1066, 1345, 654, 493, 1218, 610, 1289, 491, 229, 294, 975, 812, 130, 94, 903, 910, 1125, 908, 1057, 1324, 422, 619, 613, 1265, 714, 946, 911, 1054, 584, 1304, 737, 704, 1345, 564, 685, 734, 175, 79, 909, 321, 741, 482, 415, 1187, 389, 0 };

jaggedArray[62] = new double[] { 0, 632, 419, 1078, 418, 498, 813, 1182, 424, 1392, 1337, 1119, 145, 336, 904, 1159, 1176, 1455, 743, 589, 1273, 280, 1399, 140, 128, 239, 1056, 475, 418, 259, 635, 609, 1130, 702, 1167, 1405, 441, 748, 574, 1375, 698, 1056, 869, 1135, 233, 1385, 455, 374, 1400, 254, 646, 695, 463, 422, 1019, 575, 431, 716, 376, 1297, 432, 352, 0 };

jaggedArray[63] = new double[] { 0, 357, 111, 935, 618, 708, 786, 903, 686, 1236, 1259, 1087, 318, 381, 974, 1004, 1158, 1438, 817, 752, 1118, 177, 1469, 318, 573, 492, 1006, 139, 787, 703, 535, 333, 975, 426, 1237, 1262, 697, 922, 468, 1446, 603, 1126, 714, 1031, 269, 1242, 217, 188, 1212, 425, 552, 564, 813, 741, 1089, 837, 355, 960, 495, 1368, 598, 796, 445, 0 };

jaggedArray[64] = new double[] { 0, 690, 1029, 112, 1432, 707, 367, 290, 1336, 272, 224, 249, 1208, 1399, 459, 169, 320, 418, 510, 616, 153, 1219, 642, 1070, 1067, 1252, 217, 908, 979, 1139, 1589, 888, 170, 683, 492, 215, 1454, 615, 635, 658, 537, 381, 338, 138, 973, 195, 893, 1241, 280, 1318, 556, 573, 935, 1188, 344, 788, 1406, 785, 818, 592, 757, 1109, 1190, 1047, 0 };

jaggedArray[65] = new double[] { 0, 902, 569, 1422, 233, 972, 1212, 1448, 558, 1723, 1735, 1513, 327, 159, 1378, 1491, 1585, 1864, 1218, 1063, 1604, 363, 1873, 466, 603, 415, 1433, 684, 772, 613, 197, 878, 1462, 972, 1641, 1749, 357, 1223, 895, 1850, 1029, 1531, 1201, 1518, 560, 1729, 731, 440, 1732, 216, 978, 1027, 817, 661, 1493, 963, 255, 1124, 804, 1772, 908, 707, 470, 540, 1533, 0 };

jaggedArray[66] = new double[] { 0, 489, 613, 479, 841, 199, 214, 681, 828, 818, 760, 537, 699, 890, 402, 644, 609, 888, 245, 108, 700, 710, 897, 542, 476, 661, 457, 529, 471, 580, 1189, 626, 606, 482, 665, 806, 864, 299, 197, 874, 113, 555, 368, 536, 464, 786, 453, 805, 827, 809, 205, 286, 427, 680, 517, 280, 897, 368, 228, 796, 219, 601, 599, 667, 591, 1024, 0 };

jaggedArray[67] = new double[] { 0, 764, 996, 483, 1205, 471, 266, 748, 1058, 801, 500, 281, 1107, 1298, 158, 627, 338, 617, 292, 408, 682, 1162, 561, 1013, 840, 1026, 356, 912, 701, 861, 1635, 962, 629, 757, 329, 673, 1228, 214, 580, 538, 447, 218, 531, 387, 916, 636, 836, 1244, 810, 1217, 539, 620, 657, 910, 181, 510, 1349, 392, 690, 460, 581, 832, 964, 1050, 525, 1438, 478, 0 };

jaggedArray[68] = new double[] { 0, 268, 573, 368, 964, 420, 224, 460, 970, 669, 692, 522, 728, 919, 420, 437, 594, 873, 322, 330, 551, 740, 915, 591, 599, 785, 442, 486, 693, 703, 1166, 466, 408, 261, 683, 695, 987, 428, 155, 891, 108, 572, 147, 464, 494, 675, 414, 819, 678, 838, 76, 113, 648, 840, 535, 501, 927, 597, 351, 813, 415, 761, 722, 624, 480, 1054, 222, 496, 0 };

jaggedArray[69] = new double[] { 0, 801, 674, 1079, 304, 457, 815, 1183, 310, 1393, 1297, 1078, 302, 476, 863, 1160, 1135, 1414, 703, 548, 1274, 440, 1358, 395, 157, 125, 1057, 730, 235, 76, 734, 872, 1132, 870, 1126, 1406, 327, 724, 575, 1335, 699, 1015, 871, 1136, 489, 1386, 699, 535, 1402, 395, 648, 696, 280, 239, 978, 426, 572, 587, 377, 1257, 391, 169, 256, 617, 1191, 537, 600, 923, 724, 0 };

jaggedArray[70] = new double[] { 0, 292, 631, 339, 1112, 621, 370, 376, 1118, 641, 663, 546, 876, 1035, 566, 408, 617, 854, 465, 530, 522, 831, 1021, 739, 747, 933, 451, 510, 791, 851, 1191, 490, 379, 233, 789, 666, 1135, 571, 303, 998, 320, 679, 118, 436, 642, 646, 492, 843, 649, 986, 256, 175, 848, 988, 642, 702, 1010, 740, 499, 920, 563, 909, 870, 649, 451, 1202, 433, 642, 212, 872, 0 };

jaggedArray[71] = new double[] { 0, 485, 658, 338, 985, 260, 74, 562, 889, 678, 619, 397, 813, 1004, 262, 504, 468, 748, 104, 169, 559, 824, 757, 675, 620, 805, 316, 574, 532, 692, 1254, 671, 466, 479, 525, 665, 1007, 210, 242, 733, 109, 414, 249, 395, 578, 645, 498, 906, 687, 923, 201, 282, 488, 741, 377, 341, 1011, 379, 371, 655, 309, 662, 743, 712, 450, 1138, 143, 338, 218, 744, 361, 0 };

jaggedArray[72] = new double[] { 0, 630, 297, 1207, 370, 803, 998, 1176, 563, 1508, 1521, 1298, 195, 133, 1186, 1276, 1370, 1649, 1028, 870, 1390, 99, 1681, 252, 467, 369, 1218, 412, 729, 570, 363, 607, 1247, 700, 1448, 1534, 585, 1056, 680, 1657, 815, 1338, 986, 1304, 346, 1514, 460, 148, 1517, 214, 763, 812, 774, 618, 1301, 932, 87, 1055, 590, 1579, 693, 663, 340, 268, 1319, 292, 810, 1262, 839, 494, 922, 924, 0 };

jaggedArray[73] = new double[] { 0, 720, 464, 1298, 429, 996, 1150, 1267, 737, 1600, 1622, 1450, 384, 192, 1338, 1367, 1522, 1801, 1180, 1063, 1481, 293, 1833, 445, 654, 543, 1370, 502, 903, 744, 180, 697, 1339, 790, 1600, 1626, 643, 1286, 832, 1809, 967, 1490, 1078, 1395, 539, 1606, 580, 198, 1575, 273, 915, 927, 948, 792, 1453, 1125, 100, 1248, 783, 1731, 886, 837, 527, 357, 1410, 351, 1003, 1414, 988, 668, 1013, 1076, 183, 0 };

jaggedArray[74] = new double[] { 0, 783, 1014, 517, 1171, 436, 285, 804, 1026, 856, 588, 369, 1072, 1263, 176, 682, 426, 705, 278, 373, 738, 1138, 649, 970, 806, 991, 444, 930, 669, 829, 1600, 980, 645, 776, 417, 761, 1193, 181, 598, 625, 466, 306, 550, 475, 892, 724, 855, 1232, 865, 1182, 558, 638, 624, 877, 269, 478, 1325, 359, 655, 548, 546, 799, 929, 1069, 629, 1403, 497, 88, 515, 904, 661, 357, 1237, 1432, 0 };

jaggedArray[75] = new double[] { 0, 1039, 751, 1361, 305, 781, 1097, 1465, 117, 1675, 1564, 1345, 407, 484, 1130, 1442, 1402, 1681, 990, 829, 1556, 545, 1625, 546, 412, 230, 1339, 881, 466, 422, 641, 1015, 1414, 1109, 1393, 1688, 89, 955, 857, 1602, 981, 1282, 1153, 1418, 640, 1668, 862, 639, 1684, 403, 930, 978, 511, 257, 1245, 657, 580, 818, 659, 1524, 715, 336, 460, 721, 1473, 444, 882, 1168, 1006, 346, 1154, 1026, 599, 676, 1135, 0 };

jaggedArray[76] = new double[] { 0, 1078, 746, 1464, 144, 884, 1199, 1568, 336, 1777, 1723, 1505, 501, 336, 1290, 1545, 1561, 1841, 1129, 974, 1659, 540, 1785, 640, 514, 326, 1442, 860, 683, 524, 418, 1055, 1516, 1148, 1552, 1791, 135, 1134, 960, 1761, 1084, 1442, 1255, 1521, 734, 1771, 908, 616, 1786, 390, 1032, 1081, 728, 477, 1405, 875, 431, 1035, 762, 1683, 818, 556, 562, 716, 1576, 222, 985, 1349, 1108, 448, 1256, 1128, 469, 527, 1315, 222, 0 };

jaggedArray[77] = new double[] { 0, 905, 1137, 346, 1363, 628, 407, 610, 1238, 507, 218, 124, 1292, 1483, 217, 489, 69, 335, 450, 566, 501, 1304, 410, 1155, 998, 1184, 219, 1053, 881, 1041, 1733, 1103, 491, 898, 178, 391, 1386, 464, 721, 387, 588, 65, 572, 249, 1058, 354, 978, 1386, 605, 1402, 681, 761, 837, 1090, 103, 690, 1491, 642, 847, 309, 738, 1011, 1121, 1191, 387, 1596, 620, 284, 637, 1080, 685, 479, 1403, 1555, 371, 1347, 1507, 0 };

jaggedArray[78] = new double[] { 0, 714, 945, 448, 1106, 371, 216, 735, 958, 787, 567, 349, 1007, 1198, 134, 613, 405, 685, 192, 308, 669, 1073, 629, 905, 741, 926, 426, 861, 601, 761, 1535, 911, 576, 707, 396, 740, 1128, 113, 529, 605, 397, 286, 481, 454, 826, 704, 786, 1167, 796, 1117, 489, 570, 557, 810, 249, 410, 1260, 291, 590, 527, 481, 731, 864, 1000, 560, 1338, 428, 101, 446, 823, 592, 288, 1172, 1364, 86, 1067, 1250, 351, 0 };

jaggedArray[79] = new double[] { 0, 253, 210, 832, 817, 639, 711, 800, 885, 1133, 1156, 1011, 517, 580, 899, 901, 1083, 1362, 741, 676, 1014, 375, 1394, 402, 665, 691, 931, 63, 757, 817, 735, 146, 872, 323, 1161, 1159, 895, 847, 393, 1370, 528, 1051, 611, 928, 304, 1139, 141, 388, 1109, 624, 476, 460, 744, 954, 1014, 768, 554, 891, 465, 1292, 529, 875, 538, 193, 943, 739, 592, 975, 521, 793, 546, 637, 467, 557, 993, 920, 915, 1116, 925, 0 };

jaggedArray[80] = new double[] { 0, 91, 244, 669, 885, 602, 582, 637, 909, 970, 993, 876, 539, 648, 779, 738, 947, 1184, 681, 640, 852, 444, 1273, 401, 677, 715, 780, 123, 720, 780, 804, 127, 709, 161, 1041, 996, 919, 786, 356, 1250, 467, 931, 448, 766, 304, 976, 105, 456, 946, 648, 378, 298, 707, 917, 893, 731, 623, 855, 428, 1172, 492, 839, 537, 262, 781, 807, 555, 855, 359, 801, 383, 576, 535, 626, 873, 944, 984, 996, 804, 159, 0 };

jaggedArray[81] = new double[] { 0, 734, 966, 371, 1192, 458, 237, 636, 1067, 688, 387, 168, 1122, 1312, 47, 514, 225, 505, 279, 395, 570, 1133, 448, 984, 827, 1013, 244, 882, 710, 870, 1562, 932, 516, 727, 216, 560, 1215, 293, 550, 425, 417, 106, 501, 274, 887, 523, 807, 1215, 697, 1231, 510, 590, 666, 919, 68, 519, 1320, 471, 676, 347, 567, 840, 950, 1021, 412, 1425, 449, 113, 466, 910, 613, 308, 1232, 1384, 201, 1177, 1336, 171, 180, 945, 825, 0 };

Random random = new Random();

// Select and print 10 random distances method

static void TenRandomCityCoupleDistance(string[] cities, Random random, double[][] jaggedArray)

{

HashSet<Tuple<int, int>> used = new HashSet<Tuple<int, int>>(); // HashSet to hold used city pairs

Console.WriteLine("10 Random Distances from the Jagged Array:");

for (int i = 0; i < 10; i++)

{

int provinceIndex1, provinceIndex2;

Tuple<int, int> cityPair;

do

{

// Generate two different random city indices

provinceIndex1 = random.Next(1, jaggedArray.Length);

provinceIndex2 = random.Next(1, jaggedArray.Length);

cityPair = provinceIndex1 < provinceIndex2 ? Tuple.Create(provinceIndex1, provinceIndex2)

: Tuple.Create(provinceIndex2, provinceIndex1);

}

while (provinceIndex1 == provinceIndex2 || used.Contains(cityPair));

used.Add(cityPair); // add city pair to the HashSet

// getting the distance

double distance = GetDistance(jaggedArray, cityPair.Item1, cityPair.Item2);

Console.WriteLine($"Distance of {provinceIndex1} {cities[provinceIndex1]} to {provinceIndex2} {cities[provinceIndex2]} is: {distance} km");

}

Console.WriteLine();

}

//calling tenRandomCityCoupleDistance method for jagged array:

TenRandomCityCoupleDistance(citiesNames, random, jaggedArray);

### 1.a.2 Ekran görüntüleri

## 1.b Uzaklık Matrisinin Komşu İllere Göre Düzenlenmesi

### 1.b.1 Kodlar

// array copy method to compare after dijkstra

static double[][] copyArray(double[][] sourceArray)

{

int size = sourceArray.Length;

double[][] highwayDistancesArray = new double[size][];

for (int i = 1; i < size; i++)

{

if (sourceArray[i] != null)

{

// Copy each inner array

highwayDistancesArray[i] = new double[sourceArray[i].Length];

Array.Copy(sourceArray[i], highwayDistancesArray[i], sourceArray[i].Length);

}

}

return highwayDistancesArray;

}

//use deepcopy method for jaggedArray to compare after dijkstra

double[][] highwayCityArray = copyArray(jaggedArray);

//array that holds neighbours of provinces

int[][] neighboursArray = new int[82][];

neighboursArray[1] = new int[] { 31, 80, 46, 38, 51, 33 }; // Adana

neighboursArray[2] = new int[] { 63, 21, 44, 46, 27 }; // Adiyaman

neighboursArray[3] = new int[] { 32, 42, 26, 43, 64, 20, 15 }; // Afyon

neighboursArray[4] = new int[] { 65, 76, 36, 25, 49, 13 }; // Agri

neighboursArray[5] = new int[] { 66, 60, 55, 19 }; // Amasya

neighboursArray[6] = new int[] { 42, 68, 40, 71, 18, 14, 26 }; // Ankara

neighboursArray[7] = new int[] { 33, 70, 42, 32, 15, 48 }; // Antalya

neighboursArray[8] = new int[] { 53, 25, 75 }; // Artvin

neighboursArray[9] = new int[] { 48, 20, 45, 35 }; // Aydin

neighboursArray[10] = new int[] { 35, 45, 43, 16, 17 }; // Balikesir

neighboursArray[11] = new int[] { 43, 26, 14, 54, 16 }; // Bilecik

neighboursArray[12] = new int[] { 21, 49, 25, 24, 62, 23 }; // Bingol

neighboursArray[13] = new int[] { 56, 65, 4, 49, 72 }; // Bitlis

neighboursArray[14] = new int[] { 26, 6, 18, 67, 78, 81, 54, 11 }; // Bolu

neighboursArray[15] = new int[] { 48, 7, 32, 3, 20 }; // Burdur

neighboursArray[16] = new int[] { 10, 43, 11, 54, 41, 77 }; // Bursa

neighboursArray[17] = new int[] { 10, 59, 22 }; // Canakkale

neighboursArray[18] = new int[] { 6, 71, 19, 37, 14, 78 }; // Cankiri

neighboursArray[19] = new int[] { 66, 5, 55, 57, 37, 18, 71 }; // Corum

neighboursArray[20] = new int[] { 48, 15, 3, 64, 45, 9 }; // Denizli

neighboursArray[21] = new int[] { 63, 47, 72, 49, 12, 23, 44, 2 }; // Diyarbakir

neighboursArray[22] = new int[] { 17, 39, 59 }; // Edirne

neighboursArray[23] = new int[] { 21, 12, 62, 24, 44 }; // Elazig

neighboursArray[24] = new int[] { 23, 62, 12, 25, 69, 29, 28, 58, 44 }; // Erzincan

neighboursArray[25] = new int[] { 12, 49, 4, 36, 75, 8, 53, 69, 24 }; // Erzurum

neighboursArray[26] = new int[] { 3, 42, 6, 14, 11, 43 }; // Eskisehir

neighboursArray[27] = new int[] { 79, 63, 2, 46, 80, 31 }; // Gaziantep

neighboursArray[28] = new int[] { 29, 61, 24, 58, 52 }; // Giresun

neighboursArray[29] = new int[] { 24, 69, 61, 28 }; // Gumushane

neighboursArray[30] = new int[] { 65, 73 }; // Hakkari

neighboursArray[31] = new int[] { 79, 27, 80, 1 }; // Hatay

neighboursArray[32] = new int[] { 7, 42, 3, 15 }; // Isparta

neighboursArray[33] = new int[] { 1, 51, 42, 70, 7 }; // Mersin

neighboursArray[34] = new int[] { 41, 59, 39 }; // Istanbul

neighboursArray[35] = new int[] { 9, 45, 10 }; // Izmir

neighboursArray[36] = new int[] { 4, 76, 75, 25 }; // Kars

neighboursArray[37] = new int[] { 19, 57, 18, 74, 78 }; // Kastamonu

neighboursArray[38] = new int[] { 1, 46, 66, 58, 50, 51 }; // Kayseri

neighboursArray[39] = new int[] { 22, 59, 34 }; // Kirklareli

neighboursArray[40] = new int[] { 50, 66, 71, 6, 68 }; // Kirsehir

neighboursArray[41] = new int[] { 77, 34, 16, 54 }; // Kocaeli

neighboursArray[42] = new int[] { 7, 70, 33, 51, 68, 6, 26, 3, 32 }; // Konya

neighboursArray[43] = new int[] { 45, 64, 3, 26, 11, 16, 10 }; // Kutahya

neighboursArray[44] = new int[] { 46, 2, 21, 23, 24, 58 }; // Malatya

neighboursArray[45] = new int[] { 35, 9, 20, 64, 43, 10 }; // Manisa

neighboursArray[46] = new int[] { 27, 2, 44, 58, 38, 1, 80 }; // Kahramanmaras

neighboursArray[47] = new int[] { 63, 21, 72, 56, 73 }; // Mardin

neighboursArray[48] = new int[] { 7, 15, 20, 9 }; // Mugla

neighboursArray[49] = new int[] { 21, 72, 13, 4, 25, 12 }; // Mus

neighboursArray[50] = new int[] { 51, 38, 66, 40, 68 }; // Nevsehir

neighboursArray[51] = new int[] { 50, 38, 1, 33, 42, 68 }; // Nigde

neighboursArray[52] = new int[] { 55, 60, 58, 28 }; // Ordu

neighboursArray[53] = new int[] { 61, 69, 25, 8 }; // Rize

neighboursArray[54] = new int[] { 81, 14, 11, 16, 41 }; // Sakarya

neighboursArray[55] = new int[] { 57, 19, 5, 60, 52 }; // Samsun

neighboursArray[56] = new int[] { 65, 13, 72, 47, 73 }; // Siirt

neighboursArray[57] = new int[] { 55, 19, 37 }; // Sinop

neighboursArray[58] = new int[] { 38, 46, 44, 24, 28, 52, 60, 66 }; // Sivas

neighboursArray[59] = new int[] { 34, 39, 22, 17 }; // Tekirdag

neighboursArray[60] = new int[] { 58, 52, 55, 5, 66 }; // Tokat

neighboursArray[61] = new int[] { 53, 29, 28, 69 }; // Trabzon\*\*\*\*\*\*

neighboursArray[62] = new int[] { 23, 12, 24 }; // Tunceli

neighboursArray[63] = new int[] { 27, 2, 21, 47 }; // Sanliurfa

neighboursArray[64] = new int[] { 45, 20, 3, 43 }; // Usak

neighboursArray[65] = new int[] { 30, 73, 56, 13, 4 }; // Van

neighboursArray[66] = new int[] { 38, 58, 60, 5, 19, 71, 40, 50 }; // Yozgat

neighboursArray[67] = new int[] { 81, 14, 78, 74 }; // Zonguldak

neighboursArray[68] = new int[] { 51, 40, 6, 42, 50 }; // Aksaray

neighboursArray[69] = new int[] { 24, 25, 29, 53, 61 }; // Bayburt

neighboursArray[70] = new int[] { 33, 42, 7 }; // Karaman

neighboursArray[71] = new int[] { 40, 66, 19, 18, 6 }; // Kirikkale

neighboursArray[72] = new int[] { 47, 73, 56, 13, 49, 21 }; // Batman

neighboursArray[73] = new int[] { 56, 72, 47, 30, 65 }; // Sirnak

neighboursArray[74] = new int[] { 37, 67, 78 }; // Bartin

neighboursArray[75] = new int[] { 36, 25, 8 }; // Ardahan

neighboursArray[76] = new int[] { 4, 36 }; // Igdir

neighboursArray[77] = new int[] { 41, 16 }; // Yalova

neighboursArray[78] = new int[] { 14, 18, 37, 74, 67 }; // Karabuk

neighboursArray[79] = new int[] { 31, 27 }; // Kilis

neighboursArray[80] = new int[] { 27, 46, 1, 31 }; // Osmaniye

neighboursArray[81] = new int[] { 67, 54, 14 }; // Duzce

static void notNeighbourInfinite(int[][] neighboursArray, double[][] distanceArray)

{

for (int i = 1; i < neighboursArray.Length; i++)

{

for (int j = 1; j < distanceArray[i].Length; j++)

{

// Check if j is not in neighboursArray[i] and not the same city

if (!neighboursArray[i].Contains(j))

{

distanceArray[i][j] = INFINITY; //defined outside main at the start of the code

}

}

}

}

notNeighbourInfinite(neighboursArray, jaggedArray);//calling method for cities jaggedArray

### 1.b.2 Ekran görüntüleri

//Konsol çıktısına ait ekran görüntülerini buraya ekleyiniz

## 1.c Komşu Olmayan Şehir Çiftleri Arasındaki En Kısa Yolların Dijkstra Algoritması ile Hesaplanması ve Fark Değerlerinin Hesaplanması

### 1.c.1 Kodlar

// distance lookup for triangular matrix. greater one must taken as 1st index

static double GetDistance(double[][] distancesArray, int i, int j)

{

if (i == j)

return 0;

if (i > j)

return distancesArray[i][j];

return distancesArray[j][i];

}

static double[] Dijkstra(double[][] distanceArray, int startNode)

{

int n =distanceArray.Length;

double[] distances = new double[n];

bool[] visited = new bool[n];

// initialize distances infinite

for (int i = 0; i < n; i++)

distances[i] = INFINITY;

distances[startNode] = 0;

var priorityQueue = new SortedSet<(double, int)>();

priorityQueue.Add((0, startNode));

while (priorityQueue.Count > 0)

{

// extract node with minimum distance

var (currentDistance, currentNode) = priorityQueue.Min;

priorityQueue.Remove(priorityQueue.Min);

if (visited[currentNode])

continue;

visited[currentNode] = true;

// Process each neighbour

for (int neighbour = 1; neighbour < n; neighbour++)

{

double edgeDistance = GetDistance(distanceArray, currentNode, neighbour);

if ((edgeDistance < INFINITY) && !visited[neighbour])

{

double newDistance = currentDistance + edgeDistance;

if (newDistance < distances[neighbour])

{

// Update the distance and queue

priorityQueue.Remove((distances[neighbour], neighbour));

distances[neighbour] = newDistance;

priorityQueue.Add((newDistance, neighbour));

}

}

}

}

return distances;

}

static void changeToDijkstra(double[][] distanceArray)

{

int size = distanceArray.Length;

for(int i=1; i < size; i++)

{

distanceArray[i] = Dijkstra(distanceArray, i);

}

}

changeToDijkstra(jaggedArray);//calling the dijkstra method for cities

static void printDijkstraHighwayDistancesDiff(double[][] highwayDistancesArray, double[][] dijkstraDistancesArray, string[] provinceNames, int[][] neighboursArray)

{

//initializing min and max difference between dijkstra and highway

double minDiff = INFINITY;

double maxDiff = -1;

//creating multidimensional lists for holding provinces indexes

List<List<int>> minDiffProvinces = new List<List<int>>();

List<List<int>> maxDiffProvinces = new List<List<int>>();

int size = highwayDistancesArray.Length;

Console.Write("DISTANCES FROM ");

for (int i = 1; i < size; i++)

{

for (int j = 1; j < highwayDistancesArray[i].Length; j++)

{

if (i == j)

continue;

double dijkstraDistance = dijkstraDistancesArray[i][j];

double highwayDistance = highwayDistancesArray[i][j];//variables to hold distances

double distanceDiff = dijkstraDistance - highwayDistance;//distance difference between dijkstra and highway

if (distanceDiff < 0)

distanceDiff \*= -1;

//checking distance difference for max and min Lists

if (distanceDiff < minDiff)

{

minDiff = distanceDiff;

minDiffProvinces.Clear();

minDiffProvinces.Add(new List<int> { i, j });

}

else if (distanceDiff == minDiff)

{

minDiffProvinces.Add(new List<int> { i, j });

}

if (distanceDiff > maxDiff)

{

maxDiff = distanceDiff;

maxDiffProvinces.Clear();

maxDiffProvinces.Add(new List<int> { j, i });

}

else if (distanceDiff == maxDiff)

{

maxDiffProvinces.Add(new List<int> { j, i });

}

Console.WriteLine($"{provinceNames[i]} to {provinceNames[j]} are ; ");

Console.WriteLine($"Highway: {highwayDistance:0.##} km ,Dijksta: { dijkstraDistance:0.##} km and difference between them: { distanceDiff: 0.##} km ");

}

Console.WriteLine();

}

Console.WriteLine("Minimum difference between Dijkstra and Highway: "+ minDiff+" provinces/districts: ");

foreach(List<int> list in minDiffProvinces)

{

Console.WriteLine(provinceNames[list[0]]+", " + provinceNames[list[1]]);

}

Console.WriteLine("Maximum difference between dijkstra and highway: " + maxDiff

+" provinces/districts: ");

foreach (List<int> list in maxDiffProvinces)

{

Console.WriteLine(provinceNames[list[0]] + ", " + provinceNames[list[1]]);

}

Console.WriteLine();

}

printDijkstraHighwayDistancesDiff(highwayCityArray, jaggedArray, citiesNames,neighboursArray);

### 1.c.2 Ekran görüntüleri

A screenshot of a computer screen

Description automatically generatedA screenshot of a computer screen

Description automatically generatedA screenshot of a computer screen

Description automatically generatedA screenshot of a computer screen

Description automatically generatedA screenshot of a computer screen

Description automatically generatedA screenshot of a computer screen

Description automatically generatedA screenshot of a computer screen

Description automatically generatedA screenshot of a computer screen

Description automatically generatedA screenshot of a computer screen

Description automatically generatedA screenshot of a computer screen

Description automatically generatedA screenshot of a computer screen

Description automatically generated

## 1.d İzmir’ in İlçelerinin Uzaklık Matrisinin Oluşturulması

### 1.d.1 Kodlar

// Array that keeps Izmir's Districts Names

string[] districtNames = {"","ALİAĞA", "BALÇOVA", "BAYINDIR", "BAYRAKLI", "BERGAMA","BEYDAĞ","BORNOVA",

"BUCA", "ÇEŞME", "ÇİĞLİ","DİKİLİ", "FOÇA", "GAZİEMİR", "GÜZELBAHÇE",

"KARABAĞLAR","KARABURUN", "KARŞIYAKA", "KEMALPAŞA", "KINIK", "KİRAZ",

"KONAK", "MENDERES", "MENEMEN", "NARLIDERE", "ÖDEMİŞ","SEFERİHİSAR",

"SELÇUK", "TİRE","TORBALI", "URLA"};

//distances jaggedArray of izmir's districts

double[][] districtDistances = new double[31][];

districtDistances[1] = new double[] { 0, 0};

districtDistances[2] = new double[] { 0, 90.62010847, 0 };

districtDistances[3] = new double[] { 0, 124.3799349, 74.53056735, 0 };

districtDistances[4] = new double[] { 0, 55.06395761, 36.88568374, 70.64551021, 0 };

districtDistances[5] = new double[] { 0, 47.33268555, 137.952794, 171.7126205, 102.3966432, 0 };

districtDistances[6] = new double[] { 0, 187.9057912, 138.0564236, 63.52585627, 134.1713665, 235.2384768, 0 };

districtDistances[7] = new double[] { 0, 60.88615999, 34.26253997, 68.02236644, 7.15173526, 108.2188456, 131.5482227, 0 };

districtDistances[8] = new double[] { 0, 76.32388493, 14.29622354, 60.23434381, 22.5894602, 123.6565705, 123.7602001, 19.96631643, 0 };

districtDistances[9] = new double[] { 0, 165.0023576, 74.38224916, 148.9128165, 111.2679329, 212.3350432, 212.4386728, 108.6447891, 88.67847269, 0 };

districtDistances[10] = new double[] { 0, 40.55390684, 46.26772331, 80.02754977, 10.71157245, 87.88659239, 143.553406, 16.53377483, 31.97149977, 120.6499725, 0 };

districtDistances[11] = new double[] { 0, 59.62335529, 150.2434638, 184.0032902, 114.6873129, 29.64652934, 247.5291465, 120.5095153, 135.9472402, 224.6257129, 100.1772621, 0 };

districtDistances[12] = new double[] { 0, 40.66317054, 96.53481716, 130.2946436, 60.9786663, 87.9958561, 193.8204999, 66.80086869, 82.23859362, 170.9170663, 47.62525634, 100.2865258, 0 };

districtDistances[13] = new double[] { 0, 79.89328531, 10.72682316, 63.80374419, 26.15886058, 127.2259709, 127.3296005, 23.53571681, 3.56940038, 85.10907232, 35.54090015, 139.5166406, 85.807994, 0};

districtDistances[14] = new double[] { 0, 107.1366932, 16.51658472, 91.04715206, 53.40226846, 154.4693787, 154.5730083, 50.77912469, 30.81280826, 58.67644035, 62.78430803, 166.7600485, 113.0514019, 27.24340788, 0 };

districtDistances[15] = new double[] { 0, 83.29915707, 14.28978948, 67.20961595, 10.55978912, 130.6318426, 130.7354722, 26.94158857, 6.97527214, 88.67203864, 38.94677191, 142.9225124, 89.21386577, 3.56296632, 30.8063742, 0 };

districtDistances[16] = new double[] { 0, 181.9900326, 91.36992409, 165.9004914, 128.2556078, 229.3227181, 229.4263477, 125.6324641, 105.6661476, 87.13232506, 137.6376474, 241.6133879, 187.9047413, 102.0967473, 75.66411529, 105.6597136, 0 };

districtDistances[17] = new double[] { 0, 50.29520021, 41.65444114, 75.4142676, 4.7687574, 97.62788576, 138.9401239, 11.92049266, 27.3582176, 116.0366903, 5.94281505, 109.9185555, 56.20990891, 30.92761798, 58.17102586, 15.32854652, 133.0243652, 0 };

districtDistances[18] = new double[] { 0, 79.18384399, 49.88070254, 60.25206959, 25.44941926, 126.5165296, 123.7779259, 19.21266745, 35.584479, 124.2629517, 34.83145883, 138.8071993, 85.09855269, 39.15387938, 66.39728726, 42.55975115, 141.2506266, 30.21817666, 0 };

districtDistances[19] = new double[] { 0, 63.7969012, 154.4170097, 188.1768361, 118.8608588, 19.16663329, 210.4801181, 124.6830612, 140.1207861, 228.7992588, 104.350808, 46.11074499, 104.4600718, 143.6901865, 170.9335944, 147.0960583, 245.7869338, 114.0921014, 142.9807452, 0 };

districtDistances[20] = new double[] { 0, 188.6005432, 138.7511756, 64.22060824, 134.8661185, 235.9332287, 22.86245531, 132.2429747, 124.4549521, 213.1334247, 144.248158, 248.2238985, 194.5152519, 128.0243524, 155.2677603, 131.4302242, 230.1210997, 139.6348759, 124.4726778, 211.1748701, 0 };

districtDistances[21] = new double[] { 0, 63.49443205, 16.41910417, 69.33893064, 8.43047444, 110.8271176, 132.8647869, 12.75735533, 9.10458683, 90.80135332, 19.14204689, 123.1177873, 69.40914075, 5.69228101, 32.93568889, 2.12931468, 107.7890283, 13.19923184, 31.05503933, 127.2913333, 133.5595389, 0 };

districtDistances[22] = new double[] { 0, 95.76637661, 45.91700902, 58.38326577, 42.03195188, 143.0990622, 121.909122, 39.40880811, 31.62078548, 120.2992582, 51.41399145, 155.3897319, 101.6810853, 35.19018586, 62.43359374, 38.59605763, 137.2869331, 46.80070928, 55.02697068, 159.5632778, 122.603874, 40.72537231, 0 };

districtDistances[23] = new double[] { 0, 25.81469679, 64.80541167, 98.56523814, 29.24926082, 73.14738235, 162.0910944, 35.0714632, 50.50918814, 139.1876608, 14.73921005, 85.43805208, 47.12944771, 54.07858851, 81.32199639, 57.48446028, 156.1753358, 24.48050342, 53.3691472, 89.61159799, 162.7858464, 37.67973526, 69.95167982, 0 };

districtDistances[24] = new double[] { 0, 95.79714647, 5.177038, 79.70760535, 42.06272174, 143.129832, 143.2334616, 39.43957797, 19.47326154, 71.36852904, 51.44476131, 155.4205018, 101.7118552, 15.90386116, 13.5028646, 19.46682749, 88.35620398, 46.83147914, 55.05774054, 159.5940477, 143.9282136, 21.59614217, 51.09404703, 69.98244968, 0 };

districtDistances[25] = new double[] { 0, 160.1699393, 110.3205717, 35.79000432, 106.4355145, 207.5026248, 27.73585194, 103.8123708, 96.02434813, 184.7028208, 115.8175541, 219.7932946, 166.084648, 99.59374851, 126.8371564, 102.9996203, 201.6904958, 111.2042719, 96.04207391, 186.8085892, 28.43060392, 105.128935, 94.17327009, 134.3552425, 115.4976097, 0 };

districtDistances[26] = new double[] { 0, 126.7536553, 36.13354678, 110.6641141, 73.01923052, 174.0863408, 174.1899704, 70.39608675, 50.42977032, 78.29340241, 82.40127009, 186.3770105, 132.668364, 46.86036994, 19.61696206, 50.42333627, 95.28107735, 77.78798792, 86.01424932, 190.5505565, 174.8847224, 52.55265095, 58.58117255, 100.9389585, 33.11982667, 146.4541185, 0 };

districtDistances[27] = new double[] { 0, 128.8368824, 78.98751476, 58.49261573, 75.10245762, 176.1695679, 101.8024561, 72.47931385, 64.69129122, 153.3697639, 84.48449719, 188.4602376, 134.7515911, 68.2606916, 95.50409948, 71.66656337, 170.3574389, 79.87121502, 64.709017, 192.6337836, 102.497208, 73.79587805, 53.06014384, 103.0221856, 84.16455277, 74.06660412, 72.06632844, 0 };

districtDistances[28] = new double[] { 0, 137.5588997, 87.70953206, 18.10939632, 83.82447492, 184.8915852, 62.14262049, 81.20133115, 73.41330852, 162.0917812, 93.20651448, 197.1822549, 143.4736083, 76.9827089, 104.2261168, 80.38858066, 179.0794562, 88.59323232, 73.4310343, 201.3558009, 62.83737246, 82.51789535, 71.56223048, 111.7442029, 92.88657006, 34.40676855, 123.8430788, 39.65983558, 0 };

districtDistances[29] = new double[] { 0, 99.6847345, 49.83536692, 29.34046788, 45.95030978, 147.0174201, 92.86632415, 43.32716601, 35.53914338, 124.2176161, 55.33234934, 159.3080898, 105.5994432, 39.10854375, 66.35195163, 42.51441552, 141.205291, 50.71906717, 35.55686915, 163.4816357, 93.56107612, 44.6437302, 33.68806534, 73.87003771, 55.01240492, 65.1304722, 85.9689137, 29.15214785, 37.87416514, 0 };

districtDistances[30] = new double[] { 0, 119.0208306, 28.40072209, 102.9312894, 65.28640583, 166.3535161, 166.4571457, 62.66326206, 42.69694563, 45.98152707, 74.6684454, 178.6441859, 124.9355393, 39.12754525, 12.69491328, 42.69051157, 62.969202, 70.05516323, 78.28142463, 182.8177318, 167.1518977, 44.81982626, 74.31773111, 93.20613376, 25.38700197, 138.7212938, 32.31187535, 107.3882369, 116.1102542, 78.236089, 0 };

### 1.d.2 Ekran görüntüleri

//Konsol çıktısına ait ekran görüntülerini buraya ekleyiniz

## 1.e İzmir’in İlçeleri için Komşu Olmayan İlçe Çiftleri Arasındaki En Kısa Yolların Dijkstra Algoritması ile Hesaplanması ve Fark Değerlerinin Hesaplanması

### 1.e.1 Kodlar

//making list of neighbours of districts

int[][] districtNeighbours = new int[31][];

districtNeighbours[1] = new int[] { 23, 12, 5 };//aliağa

districtNeighbours[2] = new int[] { 15, 21, 24 };

districtNeighbours[3] = new int[] { 18, 29, 28, 25 };

districtNeighbours[4] = new int[] { 17, 21, 7 };

districtNeighbours[5] = new int[] { 1, 19, 11 };

districtNeighbours[6] = new int[] { 20, 25 };

districtNeighbours[7] = new int[] { 23, 17, 4, 21, 8, 18 };

districtNeighbours[8] = new int[] { 7, 21, 15, 13, 22, 29, 18 };

districtNeighbours[9] = new int[] { 30 };

districtNeighbours[10] = new int[] { 17, 23 };

districtNeighbours[11] = new int[] { 5 };

districtNeighbours[12] = new int[] { 23, 1 };

districtNeighbours[13] = new int[] { 8, 15, 22 };

districtNeighbours[14] = new int[] { 30, 26, 15, 24 };

districtNeighbours[15] = new int[] { 26, 22, 13, 8, 21, 2, 24, 14 };

districtNeighbours[16] = new int[] { 30 };

districtNeighbours[17] = new int[] { 4, 7, 23, 10 };

districtNeighbours[18] = new int[] { 7, 8, 29, 3 };

districtNeighbours[19] = new int[] { 5 };

districtNeighbours[20] = new int[] { 25, 6 };

districtNeighbours[21] = new int[] { 15, 2, 8, 7, 4 };

districtNeighbours[22] = new int[] { 27, 29, 8, 13, 15, 26 };

districtNeighbours[23] = new int[] { 10, 17, 7, 1, 12 };

districtNeighbours[24] = new int[] { 14, 15, 2 };

districtNeighbours[25] = new int[] { 6, 20, 3, 28 };

districtNeighbours[26] = new int[] { 22, 15, 14, 30 };

districtNeighbours[27] = new int[] { 22, 28, 29 };

districtNeighbours[28] = new int[] { 25, 3, 29, 27 };

districtNeighbours[29] = new int[] { 27, 28, 3, 18, 8, 22 };

districtNeighbours[30] = new int[] { 26, 14, 16, 9 };//urla

//copying districtDistances to compare after dijkstra

double[][] highwayDistrictArray = copyArray(districtDistances);

//making districts not neighbours distances infinite

notNeighbourInfinite(districtNeighbours, districtDistances);

//calling dijkstra method for izmir districts

changeToDijkstra(districtDistances);

Console.WriteLine("Distances for IZMIR's districts:");

printDijkstraHighwayDistancesDiff(highwayDistrictArray, districtDistances, districtNames,districtNeighbours);

}

}

}

### 1.e.2 Ekran görüntüleri

A black screen with blue text

Description automatically generatedA screen shot of a computer

Description automatically generatedA screenshot of a computer screen

Description automatically generatedA screenshot of a computer screen

Description automatically generatedA screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generatedA screenshot of a computer screen

Description automatically generatedA screenshot of a computer screen

Description automatically generatedA screen shot of a computer

Description automatically generatedA screen shot of a computer screen

Description automatically generatedA screenshot of a computer screen

Description automatically generatedA screenshot of a computer screen

Description automatically generatedA screenshot of a computer screen

Description automatically generatedA screen shot of a computer

Description automatically generatedA screen shot of a computer screen

Description automatically generatedA screen shot of a computer

Description automatically generatedA screen shot of a computer screen

Description automatically generatedA screen shot of a computer

Description automatically generatedA black screen with white text

Description automatically generated

# 2) IMAGE RECOGNITION USING ARTIFICIAL NEURAL NETWORKS

Intellij IDEA 2023.2.4, Java

## 2.a Veri Seti Oluşturma

### 2.a.1 Kaynak Kod

import java.util.ArrayList;

import java.util.List;

import java.util.Random;

public class NumberDataSet {

int [][] baseOneExample1 = new int[][]{

{ 0, 1, 0, 0, 0 },

{ 0, 1, 0, 0, 0 },

{ 0, 1, 0, 0, 0 },

{ 0, 1, 0, 0, 0 },

{ 1, 1, 1, 0, 0 },

};

int [][] baseOneExample2 = new int[][]{

{ 0, 0, 1, 0, 0 },

{ 0, 1, 1, 0, 0 },

{ 0, 0, 1, 0, 0 },

{ 0, 0, 1, 0, 0 },

{ 0, 0, 1, 0, 0 },

};

int [][] baseOneExample3 = new int[][]{

{0,0,1,1,0},

{ 0,1,1,0,0},

{ 0,0,1,0,0},

{ 0,0,1,0,0},

{ 0,0,1,0,0},

};

int [][] baseOneExample4 = new int[][]{

{0,0,1,0,0},

{0,1,1,0,0},

{0,1,1,0,0},

{0,0,1,0,0},

{0,0,1,0,0},

};

int [][] baseOneExample5 = new int[][]{

{0,1,1,0,0},

{0,1,1,0,0},

{0,0,1,0,0},

{0,0,1,0,0},

{0,0,1,0,0},

};

int [][] baseOneExample6 = new int[][]{

{0,0,1,0,0},

{0,1,1,0,0},

{1,0,1,0,0},

{0,0,1,0,0},

{0,0,1,0,0},

};

int [][] baseOneExample7 = new int[][]{

{1,1,1,0,0},

{0,1,1,0,0},

{0,0,1,0,0},

{0,0,1,0,0},

{0,0,1,0,0}

};

int [][] baseOneExample8 = new int[][]{

{0,0,1,1,0},

{0,1,0,1,0},

{0,0,0,1,0},

{0,0,0,1,0},

{0,0,0,1,0}

};

int [][] baseOneExample9 = new int[][]{

{0,0,0,1,0},

{0,0,1,1,0},

{0,1,0,1,0},

{0,0,0,1,0},

{1,1,1,1,1}

};

int [][] baseOneExample10 = new int[][]{

{0,0,1,0,0},

{0,1,1,0,0},

{0,0,1,0,0},

{0,0,1,0,0},

{0,1,1,1,0},

};

// Base matrix for "2"

int [][] baseTwoExample1 = new int[][]{

{0, 1, 1, 0, 0},

{0, 1, 0, 1, 0},

{0, 0, 0, 1, 0},

{0, 0, 1, 0, 0},

{0, 1, 1, 1, 0}

};

int [][] baseTwoExample2 = new int[][]{

{0, 1, 1, 0, 0},

{0, 1, 0, 1, 1},

{1, 0, 1, 0, 0},

{0, 0, 1, 0, 0},

{1, 1, 1, 1, 1}

};

int [][] baseTwoExample3 = new int[][]{

{0, 0, 1, 1, 0},

{0, 0, 0, 0, 1},

{0, 0, 0, 1, 0},

{0, 0, 1, 0, 0},

{0, 1, 1, 1, 1}

};

int [][] baseTwoExample4 = new int[][]{

{0, 1, 1, 1, 0},

{0, 0, 0, 0, 1},

{0, 0, 0, 1, 0},

{0, 0, 1, 0, 0},

{1, 1, 1, 0, 0}

};

int [][] baseTwoExample5 = new int[][]{

{0, 1, 1, 1, 0},

{0, 0, 0, 0, 1},

{0, 0, 0, 1, 0},

{0, 0, 1, 0, 0},

{1, 1, 1, 0, 0}

};

int [][] baseTwoExample6 = new int[][]{

{0, 1, 1, 1, 0},

{1, 0, 0, 0, 1},

{0, 0, 0, 1, 0},

{0, 1, 1, 0, 0},

{1, 1, 1, 1, 1}

};

int [][] baseTwoExample7 = new int[][]{

{ 1, 1, 1, 1, 0 },

{ 1, 0, 0, 1, 0 },

{ 0, 1, 1, 1, 0 },

{ 1, 0, 0, 0, 0 },

{ 1, 1, 1, 1, 0 }

};

int [][] baseTwoExample8 = new int[][]{

{ 1, 1, 1, 1, 1 },

{ 0, 0, 0, 0, 1 },

{ 0, 1, 1, 1, 0 },

{ 1, 0, 0, 0, 0 },

{ 1, 1, 1, 1, 1 }

};

int [][] baseTwoExample9 = new int[][]{

{ 1, 1, 1, 1, 1 },

{ 0, 0, 0, 0, 1 },

{ 1, 1, 1, 1, 1 },

{ 1, 0, 0, 0, 0 },

{ 1, 1, 1, 1, 1 }

};

int [][] baseTwoExample10 = new int[][]{

{ 0, 1, 1, 1, 0 },

{ 1, 0, 0, 0, 1 },

{ 0, 1, 0, 1, 0 },

{ 0, 0, 1, 0, 0 },

{ 1, 1, 1, 1, 1 }

};

public int[][] getBaseOneExample(int index) {

switch (index) {

case 1: return baseOneExample1;

case 2: return baseOneExample2;

case 3: return baseOneExample3;

case 4: return baseOneExample4;

case 5: return baseOneExample5;

case 6: return baseOneExample6;

case 7: return baseOneExample7;

case 8: return baseOneExample8;

case 9: return baseOneExample9;

case 10: return baseOneExample10;

default: throw new IllegalArgumentException("Index out of bounds");

}

}

public int[][] getBaseTwoExample(int index) {

switch (index) {

case 1: return baseTwoExample1;

case 2: return baseTwoExample2;

case 3: return baseTwoExample3;

case 4: return baseTwoExample4;

case 5: return baseTwoExample5;

case 6: return baseTwoExample6;

case 7: return baseTwoExample7;

case 8: return baseTwoExample8;

case 9: return baseTwoExample9;

case 10: return baseTwoExample10;

default: throw new IllegalArgumentException("Index out of bounds");

}

}

}

### a.2 Açıklama

· İki boyutlu diziler (matrisler), rakamların görsel temsili için kullanılmıştır.

· switch yapısı, varyasyonlar arasında seçim yapmak için kullanılmıştır.

### 2.a.3 Ekran Görüntüleri

//Konsol çıktısına ait ekran görüntülerini buraya ekleyiniz

## 2.b Neuron (Sinir Hücresi) Sınıfı

### 2.b.1 Kaynak Kod

import java.util.Random;

public class Neuron {

private double[] weights;

public Neuron(int numInputs){

weights = new double[numInputs];

initializeWeights();

}

private void initializeWeights(){

Random rand = new Random();

for(int i = 0; i<weights.length; i++) {

weights[i] = rand.nextDouble();

}

}

// method to calculate the output

public double calculateOutput(int[][] inputMatrix){

//turn the matrix into an array

int[] flatInputs = new int[25];

for(int i = 0; i < 5;i++){

System.arraycopy(inputMatrix[i], 0, flatInputs, i \* 5 , 5);

}

if (flatInputs.length != weights.length){

throw new IllegalArgumentException("Number of inputs must match number of weights.");

}

double sum = 0;

for(int i = 0; i < flatInputs.length; i++){

sum += flatInputs[i] \* weights[i];

}

return sum;

}

// Getters for weights

public double[] getWeights() {

return weights;

}

}

### b.2 Açıklama

initializeWeights() fonksiyonu weights listesinin her elemanına random bir şekilde 0 ile 1 arasında değer atar.

calculateOutputs() fonksiyonu

-inut olarak matris alıp bu matrisi a ”arrayCopy” fonksiyonunu kullanarak liste haline çevirir.

-sonra bu matrisin her bir elemanını weights listesinin elemanlarıylaya çarparak sum’a ekler.

## 2.c Neural Network (Yapay Sinir Ağı) Sınıfı

### 2.c.1 Kaynak Kod

import java.util.List;

public class NeuralNetwork {

private Neuron[] neurons; // Array of neurons in the network

// Constructor to initialize the network with 2 neurons

public NeuralNetwork() {

neurons = new Neuron[2];

for (int i = 0; i < 2; i++) {

neurons[i] = new Neuron(25);

}

}

public void neuralNetworkTraining(List<int[][]> onesMatrix, List<int[][]> twosMatrix,

int epoch, double learningFactor) {

for (int i = 0; i < epoch; i++) {

// Training on onesMatrix

for (int[][] dataMatrix : onesMatrix) {

var output = outputCalculator(dataMatrix);

if (output[0] < output[1]) {

// Increase weights for Neuron 1 and decrease for Neuron 2

updateNeuronWeights(learningFactor, dataMatrix, 0, true); // Neuron 1 increments

updateNeuronWeights(learningFactor, dataMatrix, 1, false); // Neuron 2 decrements

}

}

// Training on twosMatrix

for (int[][] dataMatrix : twosMatrix) {

var output = outputCalculator(dataMatrix);

if (output[0] > output[1]) {

// Decrease weights for Neuron 1 and increase for Neuron 2

updateNeuronWeights(learningFactor, dataMatrix, 0, false); // Neuron 1 decrements

updateNeuronWeights(learningFactor, dataMatrix, 1, true); // Neuron 2 increments

}

}

}

}

// Method to calculate the outputs for both neurons

public double[] outputCalculator(int[][] dataMatrix) {

double[] outputs = new double[2];

outputs[0] = neurons[0].calculateOutput(dataMatrix); // Output for Neuron 1

outputs[1] = neurons[1].calculateOutput(dataMatrix); // Output for Neuron 2

return outputs;

}

// Method to update weights for a specific neuron based on the learning factor and input matrix

private void updateNeuronWeights(double learningFactor, int[][] inputMatrix, int neuronIndex, boolean increment) {

for (int i = 0; i < inputMatrix.length; i++) {

for (int j = 0; j < inputMatrix[i].length; j++) {

// Get the current weight of the neuron

double pixelWeight = neurons[neuronIndex].getWeights()[i \* inputMatrix[i].length + j];

// Update the weight based on the increment flag

if (increment) {

// Increase weight by learning factor

pixelWeight += learningFactor \* inputMatrix[i][j];

} else {

// Decrease weight by learning factor

pixelWeight -= learningFactor \* inputMatrix[i][j];

}

// Set the updated weight

neurons[neuronIndex].getWeights()[i \* inputMatrix[i].length + j] = pixelWeight;

}

}

}

}

### 2.c2 Açıklama

**neuralNetworkTraining**  
Bu metot, sinir ağını "1" ve "2" sayılarını tanıyacak şekilde eğitir. Parametre olarak "1" ve "2" sayısını temsil eden giriş matrislerinin listeleri (onesMatrix ve twosMatrix), epoch (kaç iterasyon yapılacağı) ve öğrenme faktörü (learningFactor) alır. Verilen epoch sayısı kadar eğitim döngüsü yapılır. "1" verileri için çıktı hesaplanır; eğer yanlış sınıflandırma varsa Nöron 1'in ağırlıkları artırılır ve Nöron 2'ninkiler azaltılır. Aynı şekilde "2" verileri için de çıktı hesaplanır ve gerektiğinde ağırlıklar güncellenir.

**outputCalculator**  
Bu metot, verilen giriş matrisi için iki nöronun çıktısını hesaplar. Parametre olarak bir 5x5 giriş matrisi (dataMatrix) alır ve iki nöronun çıktısını içeren bir double[] dizisi döndürür. Çıktı, Neuron sınıfının calculateOutput metodu çağrılarak hesaplanır.

**updateNeuronWeights**  
Belirli bir nöronun ağırlıklarını öğrenme faktörü ve giriş matrisi temelinde güncelleyen metottur. Parametre olarak öğrenme faktörü (learningFactor), giriş matrisi (inputMatrix), ağırlıkları güncellenecek nöronun indeksi (neuronIndex) ve ağırlıkların artırılıp artırılmayacağını belirten bir bayrak (increment) alır. Ağırlıklar giriş matrisine göre ya artırılır ya da azaltılır ve ilgili Neuron nesnesindeki ağırlıklar güncellenir.

**Neuron Sınıfı İle Etkileşim**  
Sinir ağı içerisindeki Neuron nesneleri ile calculateOutput metodu aracılığıyla giriş matrisi ile ağırlıkların çarpımı toplanarak bir çıktı üretilir. getWeights metodu ise her bir nöronun ağırlıklarına doğrudan erişim sağlar.

## 2.d Eğitim Metodunun Yazılması

### 2.d.1 Kaynak Kod

public void neuralNetworkTraining(List<int[][]> onesMatrix, List<int[][]> twosMatrix,

int epoch, double learningFactor) {

for (int i = 0; i < epoch; i++) {

// Training on onesMatrix

for (int[][] dataMatrix : onesMatrix) {

var output = outputCalculator(dataMatrix);

if (output[0] < output[1]) {

// Increase weights for Neuron 1 and decrease for Neuron 2

updateNeuronWeights(learningFactor, dataMatrix, 0, true); // Neuron 1 increments

updateNeuronWeights(learningFactor, dataMatrix, 1, false); // Neuron 2 decrements

}

}

// Training on twosMatrix

for (int[][] dataMatrix : twosMatrix) {

var output = outputCalculator(dataMatrix);

if (output[0] > output[1]) {

// Decrease weights for Neuron 1 and increase for Neuron 2

updateNeuronWeights(learningFactor, dataMatrix, 0, false); // Neuron 1 decrements

updateNeuronWeights(learningFactor, dataMatrix, 1, true); // Neuron 2 increments

}

}

}

}

### 2.d.2 Açıklama

**neuralNetworkTraining**  
Bu metot, sinir ağını "1" ve "2" sayılarını tanıyacak şekilde eğitir. Parametre olarak "1" ve "2" sayısını temsil eden giriş matrislerinin listeleri (onesMatrix ve twosMatrix), epoch (kaç iterasyon yapılacağı) ve öğrenme faktörü (learningFactor) alır. Verilen epoch sayısı kadar eğitim döngüsü yapılır. "1" verileri için çıktı hesaplanır; eğer yanlış sınıflandırma varsa Nöron 1'in ağırlıkları artırılır ve Nöron 2'ninkiler azaltılır. Aynı şekilde "2" verileri için de çıktı hesaplanır ve gerektiğinde ağırlıklar güncellenir.

## 2.e Eğitim

### 2.e.1 Kaynak Kod

// Create a neural network instance with 2 neurons

NeuralNetwork neuralNetwork = new NeuralNetwork();

double learningRate = 0.03; // Set learning rate

int epochs = 40; // Number of epochs for training

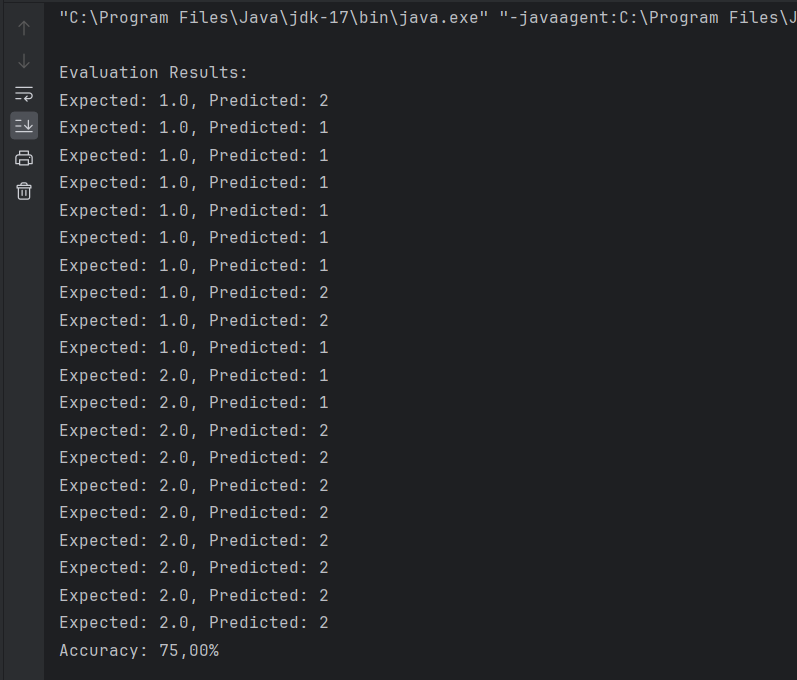
// Training loop

for (int epoch = 0; epoch < epochs; epoch++) {

neuralNetwork.neuralNetworkTraining(onesMatrix, twosMatrix, 1, learningRate); // Epochs are handled inside the method

}

### 2.e.2 Ekran Görüntüleri / Tablolar



## 2.f Modelin Görmediği Bir Matristen Sayıyı Tahminleme

### 2.d.1 Kaynak Kod

int[][][] newOnesMatrices = {

{

{0, 0, 1, 0, 0},

{0, 1, 1, 0, 0},

{1, 0, 1, 0, 0},

{0, 0, 1, 0, 0},

{1, 1, 1, 1, 1}

},

{

{0, 0, 1, 0, 0},

{0, 0, 1, 0, 0},

{0, 0, 1, 1, 0},

{0, 0, 1, 0, 0},

{1, 1, 1, 1, 1}

},

{

{0, 0, 1, 0, 0},

{0, 0, 1, 0, 0},

{0, 0, 0, 0, 0},

{0, 0, 1, 0, 0},

{0, 0, 0, 0, 0}

}

};

int[][][] newTwosMatrices = {

{

{1, 1, 1, 1, 0},

{0, 0, 0, 1, 0},

{0, 0, 1, 0, 0},

{0, 1, 0, 0, 0},

{1, 1, 1, 1, 1}

},

{

{0, 1, 1, 1, 0},

{1, 0, 0, 1, 0},

{0, 0, 1, 0, 0},

{0, 1, 0, 0, 0},

{1, 0, 0, 1, 1}

},

{

{0, 1, 1, 1, 0},

{0, 0, 0, 1, 0},

{0, 0, 0, 0, 0},

{0, 1, 0, 0, 0},

{1, 1, 1, 1, 0}

}

};

// Evaluation phase for the new matrices

System.out.println("\nEvaluation Results for New Matrices:");

correctCount = 0;

int totalNewMatrices = + newTwosMatrices.length;

for (int i = 0; i < totalNewMatrices; i++) {

int[][] dataMatrix;

double expectedOutput;

if (i < newOnesMatrices.length) {

dataMatrix = newOnesMatrices[i];

expectedOutput = 1.0;

} else {

dataMatrix = newTwosMatrices[i - newOnesMatrices.length];

expectedOutput = 2.0;

}

// Calculate outputs for the current input

double[] outputs = neuralNetwork.outputCalculator(dataMatrix);

int predictedLabel = (outputs[0] > outputs[1]) ? 1 : 2;

// Print expected and predicted outputs

System.out.println("Expected: " + expectedOutput + ", Predicted: " + predictedLabel);

// Count correct predictions

if (predictedLabel == expectedOutput) {

correctCount++;

}

}

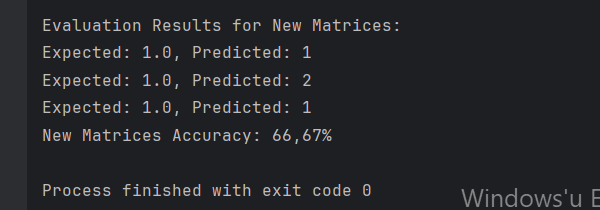
// Calculate and print accuracy for new matrices

double newAccuracy = (double) correctCount / totalNewMatrices \* 100;

System.out.printf("New Matrices Accuracy: %.2f%%\n", newAccuracy);

}

### 2.f.2 Sonuçlar / Ekran Görüntüleri



# Öz değerlendirme Tablosu

|  |  |  |  |
| --- | --- | --- | --- |
| **Proje 1 Maddeleri** | **Not** | **Tahmini Not** | **Açıklama** |
| 1.a) | 5 | 5 | Uzaklık değerleri text dosyasına aktarıp oradan Python üzerinden kısa bir kodla istediğimiz formata düzenleyerek yazdırılıp projeye kopyalandı. |
| 1.b | 7 | 7 | Şehirlerin komşuları internetten bulundu daha sonra kontrol edildi, her şehrin komşularının listesi farklı bir jagged arrayde tutuldu, bu array kullanılarak komşu olmayan değerler INFINITY(double.MaxValue) değerine atandı. |
| 1.c | 10 | 10 | Dijkstra methodunda ayrı bir distances[] arrayi oluşturulup belirtilen şehir için en kısa mesafelerden başlayarak komşular dolaşılarak en kısa mesafeler bulundu daha sonra changeToDijkstra methodunda Dijkstra methoduna for döngüsüyle tüm şehirler için bu methodu kullanarak en kısa mesafeler güncellendi. |
| 1.d | 8 | 8 | Şehirler için olduğu gibi yine değerler text dosyasına atılarak Python kodu ile istenilen formata dönüştürülerek kopyalandı ve districtDistances arrayi oluşturuldu. |
| 1.e | 5 | 5 | Komşu listesi tek tek haritadan bakarak yine districtNeighbours[][] arrayine aktarıldı ve tekrar aynı dijkstra methodları kullanılarak en kısa mesafeler yazdırıldı. |
| 2.a | 5 | 5 | Her bir matris için 10ar tane yeni matris yazılmıştır. |
| 2.b | 5 | 5 | Projede istenildiği gibi random olarak ağırlılar üretilip input matrislerle birlikte kullanılarark ger döndürülmüştür. |
| 2.c | 10 | 10 | Projede istenildiği gibi yazılmıştır. |
| 2.d | 5 | 5 | Eğitim methodu istenildiği gibi yazılıp projede verilen detaylara göre yapılandırılmıştır |
| 2.e | 10 | 10 | Verilen özelliklere göre eğitim yapılmıştır. |
| 2.f | 5 | 5 | Yeni matrisler okutulup bunların dğruluk oranı çıktı olarak konsola yazılmıştır |
| Rapor | 15 | 12 | Biraz daha Detaylı bir şekilde yazılabilirdi |
| Öz değerlendirme Tablosu | 10 | 10 | Detaylı şekilde doldurulmuştur. |
| **Toplam** | **100** | **95** | **Rapor biraz daha detaylandırılabilirdi.** |

**Açıklama kısmında yapıldı, yapılmadı bilgisi ve hangi maddelerin nasıl yapıldığı veya neden yapılamadığı kısaca yazılmalıdır.**

**Not: Raporu teslim edilmeyen projeler değerlendirmeye alınmayacaktır.**