In [1]:

```
import numpy as np
from copy import deepcopy
```

In [2]:

```
# run pa2 first to create similarity
# get similarity data
similarity = np.genfromtxt("similarity.csv", delimiter=',')
n = similarity.shape[0]
```

use heap to build priority queue of each document

In [3]:

```
def adjust(queue, i, n, stored_index):
    ''' adaptive heap operation

    :param stored_index: list of where the data stored
    :param queue: [0,...,n-1], each store {"sim": , "index": , "stored_index": }
    :type queue: [{}, {}]
    :param i: node id
    :param n: size of tree
    :return: None
    '''
    x = queue[i]
    j = 2*i +1
    while j<=n-1 :
        if j<n-1:
            if queue[j]["sim"]<queue[j + 1]["sim"]:
                j = j+1
        if x["sim"] >= queue[j]["sim"]:
            break
        else:
            queue[(j-1) // 2] = queue[j]
            stored_index[queue[j]["index"]] = (j-1) // 2
            j = 2*j +1
    queue[(j-1) // 2] = x
    stored_index[x["index"]] = (j-1) // 2
    pass

def build(queue, n, stored_index):
    '''

    :param stored_index:
    :param queue: [0,...,n-1]
    :param n: size of tree
    :return: None
    '''
    half_n = np.arange(n//2)
    re_n = n//2 - half_n -1

    for i in re_n:
        adjust(queue, i, n, stored_index)
    pass

def pop_item(queue, i, n, stored_index):
    '''

    :param stored_index:
    :param i: index of queue to pop
    :param queue: [0,...,n-1], each store {"sim": , "index": }
    :type queue: [{}, {}]
    :param n: size of tree
    :return: queue[i] before pop
    '''
    if i >= n:
        raise IndexError("what you wanna pop is out of index")
    x = queue[i]
    stored_index[queue[i]["index"]] = n-1
    queue[i] = queue[n-1]
    stored_index[queue[i]["index"]] = i
    n -=1
    adjust(queue, i, n, stored_index)
    return x

def insert(queue, n, item, stored_index):
    ''' adaptive insert in heap

    :param stored_index:
    :param queue: [0,...,n-1], each store {"sim": , "index": }
    :type: [{}, {}]
    :param n: size of tree
    :type n: int
    :param item:
    :type: [{}, {}]
    :return:
    '''
    queue.append(item)
    n +=1
    child_no = n-1
    parent_no = (child_no-1)//2
    while 1:
        if queue[parent_no]["sim"] >= item["sim"]:
            break
        else:
            queue[child_no] = queue[parent_no]
            stored_index[queue[child_no]["index"]] = child_no
            child_no = (child_no-1)//2
            if child_no == 0:
                break
            parent_no = (child_no-1)//2
    queue[child_no] = item
    stored_index[item["index"]] = child_no
    pass
```

let's start

In [5]:

```
c_sim = np.copy(similarity)
c_index = np.full((n, n), np.arange(n))
stored_index = np.full((n, n), np.arange(n))
merge_flag = [1] *n
merge_dict = {}
# use dict to store cluster
for i in range(n):
    merge_dict[i] = [i]


queue = []
queue_len = []
c = 0
for i in c_sim:
    queue.append([])
    cc = 0
    for j in i:
        queue[c].append({"sim": j,
                         "index": cc})
        # queue[n]["sim"] -> priorityQ of c_sim[n]
        # queue[n]["index"] -> index of c[n][i]
        cc += 1
    c +=1

# build priorityQ
for i in range(n):
    build(queue[i], n, stored_index[i])
    pop_item(queue[i], stored_index[i][i], n, stored_index[i])
    queue_len.append(n-1)



# argmax
def argmax_index(queue):
    '''

    :param queue:
    :type: [{},{}]
    :return:
    :rtype: dict
    '''
    for i in range(n):
        if merge_flag[i]:
            local_max = queue[i][0]
            local_max_index = i
    for i in range(n):
        if merge_flag[i]:
            if local_max["sim"] < queue[i][0]["sim"]:
                local_max = queue[i][0]
                local_max_index = i
    return local_max_index

# finding place of c[i][index] in queue[i]
def get_place(queue, index):
    '''

    :param queue:
    :param index:
    :return:
    '''
    for i in range(len(queue)):
        if queue[i]["index"] == index:
            return i
    raise IndexError

# use dict to store result
cluster = {8:{},
           13:{},
           20:{}}


for _ in range(n-1):
    max_index = argmax_index(queue)
    max_index_max_index = queue[max_index][0]["index"]
    k1 = max_index
    k2 = max_index_max_index
    # check feasibility
    if queue[k1][0]["index"] != k2:
        raise IndexError("iter ", _, "found", "k1= ", k1, ", k2= ", k2, "k1, k2 should be the same\n")
    if k1 > k2:
        k1 = max_index_max_index
        k2 = max_index
    merge_dict[k1] += merge_dict[k2]
    # check feasibility
    if not merge_flag[k1]:
        print(False)
        break
    if not merge_flag[k2]:
        print(False)
    merge_flag[k2] = 0
    queue[k1] = []
    queue_len[k1] = 0
    stored_index[k1] = stored_index[k1] *0

    for index in range(n):
        if merge_flag[index] and index != k1:
            pop_item(queue[index], i=get_place(queue[index], k1), n=queue_len[index], stored_index=stored_index[index])
            queue_len[index] -=1
            pop_item(queue[index], i=get_place(queue[index], k2), n=queue_len[index], stored_index=stored_index[index])
            queue_len[index] -=1
            # use single link to updata cosine similarity
            if c_sim[index][k1] <= c_sim[index][k2]:
                bigger_sim = c_sim[index][k1]
                bigger_sim_id = k1
            else:
                bigger_sim = c_sim[index][k2]
                bigger_sim_id = k2

            c_sim[index][k1] = c_sim[k1][index] = bigger_sim

            insert(queue=queue[index], n=queue_len[index], item={"sim": c_sim[index][k1], "index": k1}, stored_index=stored_inde
x[index])
            queue_len[index] +=1
            insert(queue=queue[k1], n=queue_len[k1], item={"sim": c_sim[index][k1], "index": index}, stored_index=stored_index[k
1])
            queue_len[k1] +=1
    # save cluster
    if sum(merge_flag) in cluster:
        c=0
        for index in range(n):
            if merge_flag[index]:
                cluster[sum(merge_flag)][c] = deepcopy(merge_dict[index])
                c += 1
    # early stop
    if sum(merge_flag) == 8:
        print("meet 8 cluster")
        break
```

```
meet 8 cluster
```

sort list in each cluster

In [6]:

```
for cluster_cat, clustered_data in cluster.items():
    for index, key in clustered_data.items():
        key.sort()
for index, key in cluster[13].items():
    print(index, key)
```

```
0 [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 11, 12, 13, 14, 15, 16, 17, 19, 20, 21, 22, 23, 24, 25, 26, 27, 29, 31, 32, 33, 34, 35, 36, 3
7, 38, 39, 40, 42, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 68, 70, 71, 72,
73, 74, 75, 76, 77, 78, 79, 80, 81, 83, 84, 86, 87, 88, 89, 92, 93, 94, 95, 96, 97, 98, 100, 102, 104, 105, 106, 107, 108, 109,
110, 111, 113, 115, 116, 117, 118, 119, 120, 121, 123, 124, 125, 126, 127, 225, 231, 258, 630, 631, 648, 663, 687, 688, 711, 72
8, 812, 816, 817, 818, 819, 820, 821, 823, 824, 825, 827, 828, 829, 831, 832, 833, 834, 836, 837, 842, 843, 846, 849, 850, 856,
857, 858, 861, 863, 865, 866, 867, 869, 871, 872, 873, 875, 878, 880, 881, 890, 892, 904, 942, 943, 987, 1014, 1015, 1022, 102
4, 1026, 1027, 1029, 1032, 1034, 1036, 1037, 1038, 1040, 1041, 1042, 1044, 1045, 1046, 1047, 1051, 1052, 1053, 1054, 1055, 105
7, 1061, 1062, 1065, 1067, 1068, 1070, 1071, 1077, 1079, 1080, 1081, 1082, 1083, 1084, 1085, 1086, 1087, 1088, 1089, 1090, 109
2, 1093]
1 [10, 18, 28, 112, 114, 168, 277, 300, 315, 316, 320, 323, 324, 337, 340, 356, 368, 371, 376, 380, 382, 383, 385, 387, 388, 39
5, 399, 401, 404, 418, 421, 422, 424, 428, 430, 434, 443, 450, 459, 463, 466, 467, 475, 476, 478, 481, 488, 493, 523, 542, 624,
809, 810]
2 [30, 43, 69, 82, 85, 91, 99, 101, 304, 308, 314, 319, 325, 326, 327, 330, 333, 339, 343, 344, 350, 354, 357, 359, 361, 364, 3
69, 370, 374, 375, 379, 381, 390, 391, 392, 393, 394, 397, 402, 403, 407, 420, 423, 432, 435, 438, 439, 440, 518, 527, 590]
3 [41, 67, 90, 103, 122, 128, 148, 152, 155, 175, 182, 188, 242, 243, 255, 270, 655, 851, 864, 908]
4 [129, 130, 131, 132, 133, 136, 140, 141, 144, 146, 153, 159, 161, 163, 165, 167, 169, 170, 171, 173, 174, 176, 177, 184, 186,
187, 190, 191, 192, 193, 195, 197, 199, 200, 202, 203, 204, 205, 206, 208, 210, 211, 212, 213, 215, 216, 217, 218, 219, 220, 22
1, 224, 226, 227, 230, 233, 234, 252, 263, 265, 267, 268, 271, 273, 275, 282, 284, 293, 297, 298, 301, 302, 306, 310, 311, 312,
313, 318, 321, 331, 360, 410, 413, 436, 445, 446, 451, 452, 456, 464, 468, 469, 470, 473, 474, 678, 691, 749, 757, 777, 814, 88
4, 885, 886, 887, 889, 891, 906, 909, 910, 911, 914, 940, 941, 947, 967, 968, 969, 970, 972, 975, 977, 980, 995, 996, 1000, 100
1, 1003, 1009, 1031, 1043, 1049, 1056, 1058, 1059]
5 [134, 135, 137, 145, 154, 156, 157, 160, 164, 166, 172, 178, 181, 185, 196, 198, 201, 241, 485, 549, 550, 551, 559, 561, 619,
760, 762]
6 [138, 139, 142, 143, 147, 149, 150, 158, 162, 179, 180, 183, 194, 207, 209, 214, 223, 228, 229, 232, 235, 237, 245, 246, 248,
250, 251, 256, 260, 262, 264, 266, 269, 272, 283, 285, 286, 287, 288, 292, 334, 335, 341, 342, 347, 348, 351, 355, 358, 363, 36
5, 367, 372, 373, 378, 384, 386, 389, 398, 406, 408, 409, 411, 414, 416, 417, 431, 447, 448, 454, 455, 457, 458, 460, 461, 462,
477, 487, 509, 510, 517, 520, 616, 617, 633, 644, 646, 649, 665, 671, 685, 689, 690, 692, 693, 710, 713, 715, 716, 727, 741, 74
5, 746, 761, 764, 767, 768, 771, 772, 774, 775, 776, 778, 781, 784, 786, 787, 788, 789, 862, 898, 900, 901, 903, 905, 1078, 109
4]
7 [151, 189, 222, 236, 238, 276, 279, 280, 281, 289, 290, 291, 309, 317, 322, 328, 329, 332, 353, 366, 377, 426, 433, 483, 521,
524, 548, 560, 584, 622, 627, 629, 634, 647, 654, 699, 701, 703, 704, 705, 707, 708, 729, 730, 731, 732, 734, 739, 743, 782, 80
7, 877, 915, 924, 925, 926, 927, 937, 950, 983, 1066]
8 [239, 240, 244, 247, 249, 253, 254, 257, 259, 274, 278, 294, 296, 299, 305, 338, 345, 346, 349, 352, 362, 405, 412, 415, 437,
441, 453, 471, 472, 480, 482, 486, 489, 490, 491, 492, 494, 495, 496, 497, 498, 499, 500, 501, 502, 503, 504, 505, 506, 507, 50
8, 511, 513, 514, 515, 516, 651, 714, 720, 724, 748, 752, 808, 815, 882]
9 [261, 295, 303, 307, 336, 396, 400, 425, 442, 444, 449, 465, 479, 512, 532, 533, 596, 623, 637, 641, 650, 652, 653, 656, 657,
658, 659, 660, 661, 662, 664, 666, 667, 668, 669, 670, 672, 673, 674, 675, 676, 677, 680, 681, 683, 684, 694, 695, 696, 697, 69
8, 700, 702, 706, 709, 712, 733, 744, 792, 800, 959, 986, 990]
10 [419, 427, 429, 484, 519, 522, 525, 526, 528, 529, 530, 531, 534, 535, 536, 537, 538, 539, 540, 541, 543, 544, 545, 546, 54
7, 552, 553, 554, 555, 556, 557, 558, 562, 563, 564, 565, 566, 567, 568, 569, 570, 571, 572, 573, 574, 575, 576, 577, 578, 579,
580, 581, 582, 583, 585, 586, 587, 588, 589, 591, 592, 593, 594, 595, 597, 598, 599, 600, 601, 602, 603, 604, 605, 606, 607, 60
8, 609, 610, 611, 612, 613, 614, 615, 618, 620, 621, 625, 626, 628, 632, 635, 636, 638, 639, 640, 642, 643, 679, 682, 686, 717,
718, 719, 721, 722, 723, 725, 726, 735, 736, 737, 738, 740, 742, 747, 791, 795, 813, 826, 835, 853, 859, 899, 907, 912, 913, 91
6, 917, 918, 919, 920, 921, 923, 928, 929, 932, 935, 939, 945, 949, 965, 985, 988, 991, 993, 999, 1007, 1016, 1021, 1033, 1035,
1048, 1060, 1063, 1064, 1069, 1072, 1073, 1075]
11 [645, 750, 780, 793, 797, 798, 811, 822, 830, 838, 839, 840, 841, 844, 845, 847, 848, 852, 860, 876, 879, 888, 893, 894, 89
5, 896, 897, 902, 922, 930, 931, 933, 934, 936, 938, 953, 954, 955, 956, 958, 984, 994, 997, 998, 1002, 1004, 1005, 1006, 1010,
1011, 1012, 1013, 1017, 1018, 1019, 1020, 1023, 1025, 1028, 1030, 1039, 1074, 1076, 1091]
12 [751, 753, 754, 755, 756, 758, 759, 763, 765, 766, 769, 770, 773, 779, 783, 785, 790, 794, 796, 799, 801, 802, 803, 804, 80
5, 806, 854, 855, 868, 870, 874, 883, 944, 946, 948, 951, 952, 957, 960, 961, 962, 963, 964, 966, 971, 973, 974, 976, 978, 979,
981, 982, 989, 992, 1008, 1050]
```

writing down all kinda cluster

In [8]:

```
for cluster_cat, clustered_data in cluster.items():
    data = open("%d.txt" % cluster_cat, "w+")
    for index, value in clustered_data.items():
        for file_id in value:
            print(file_id+1, file=data)
        print("", file=data)
    data.close()
```