§2 Второй отборочный этап

Второй отборочный этап проводится в командном формате в сети интернет, работы оцениваются автоматически средствами системы онлайн-тестирования. Продолжительность второго отборочного этапа составляет 2 недели. Задачи носят междисциплинарный характер и в более простой форме воссоздают инженерную задачу заключительного этапа. Решение задач предполагало написание программ, допускалось использовать язык программирования Python. Решение каждой задачи дает определенное количество баллов. В данном этапе можно получить суммарно от 0 до 45 баллов.

2.1. Задачи по анализу данных

Задача 2.1.1 (10 баллов)

В летний лагерь приехало 1000 школьников. Когда школьник приезжал, его сразу переписывали (ставили порядковый номер начиная с нуля — каким школьник приехал в лагерь). Школьников сразу разбивали по отрядам с разным количеством человек в отряде: первые n1 школьников определили в первый отряд, следующие n2 школьников — во второй отряд, следующие n3 — в третий и так далее.

Однажды все четные отряды увезли на экскурсию. И в лагерь приехала комиссия и переписала всех школьников, которые остались (каждый школьник назвал номер, каким его записали, когда он приехал в лагерь). По ошибке некоторых школьников переписали несколько раз. Как теперь разобраться, какой школьник в каком отряде?

На вход подается массив из чисел, соответствующих порядковым номерам переписанных школьников, которые остались в лагере.

На выход подаются набор пар чисел: порядковый номер первого и конечного школьника в первом отряде, порядковый номер первого и конечного школьника во втором отряде и так далее**.**

За успешное решение задачи участники получают 10 баллов.

Пример ввода:

```
790 443 801 518 63 75 491 91 809 507 367 778 803 809 923 800 428 857 384
49 358 346 499 931 32 842 965 392 883 873 888 824 480 407 854 435 982 464 907
799 829 454 518 867 346 871 830 90 866 893 455 872 867 815 867 87 877 32 479 834
414 463 972 362 503 481 833 885 366 935 828 495 89 944 30 78 930 964 62 75 893
943 495 357 961 462 830 477 967 841 512 882 888 810 855 837 447 418 434 53 891
998 13 82 794 401 782 34 789 358 506 82 880 997 812 72 24 866 22 444 519 857 972
```

911 50 842 66 826 92 474 0 395 874 964 909 387 454 352 6 862 458 873 512 394 886
449 431 517 994 512 73 986 515 481 444 867 29 998 884 512 401 805 798 36 993 16
38 373 482 926 69 430 818 862 61 369 469 867 851 865 383 84 780 512 438 935 837
901 918 951 426 39 405 365 55 346 502 958 41 345 414 346 441 375 11 391 59 354
939 372 981 379 972 478 948 52 804 56 803 445 413 799 981 12 947 507 443 973 883
999 936 847 940 808 802 906 997 864 79 462 410 850 364 400 392 363 832 982 795
69 424 468 80 394 968 35 817 907 937 975 447 988 363 992 24 474 358 475 456 9
443 496 76 8 7 453 848 19 494 371 10 407 488 508 466 66 823 18 452 91 843 912
454 512 8 940 455 46 835 57 908 66 436 846 460 921 800 866 924 60 483 794 845 11
941 894 365 454 943 40 512 829 853 353 425 32 507 941 39 26 907 85 470 459 778
492 348 925 437 39 497 807 350 37 377 931 894 847 440 962 870 448 866 966 448
454 446 901 950 47 33 890 792 934 793 844 5 358 928 346 452 995 792 401 346 983
509 51 896 346 972 803 474 877 350 879 32 471 832 848 878 829 501 895 776 400
396 775 357 979 996 862 500 10 791 80 41 457 795 876 907 977 512 8 897 833 938
493 432 506 391 963 813 802 68 987 803 794 923 442 391 962 366 24 944 904 979
821 358 964 780 23 850 832 424 889 867 36 48 518 450 363 910 511 447 445 396 374
942 888 954 65 344 356 423 349 88 468 426 930 386 510 445 408 489 32 907 803 16
16 972 967 848 462 20 962 422 988 812 946 92 869 49 886 60 64 443 914 910 395
933 888 462 460 60 458 442 920 429 819 975 972 45 38 347 33 910 990 834 371 900
781 842 857 949 471 985 839 820 421 505 901 822 77 485 932 978 479 416 23 512 16
786 774 392 519 76 797 907 69 864 60 912 909 384 890 982 788 460 445 925 954 415
520 512 836 498 399 859 48 986 830 380 481 433 894 879 422 978 926 344 853 504
455 952 998 813 512 915 871 481 520 347 406 33 361 971 882 53 512 799 351 867 67
817 19 796 387 437 468 389 412 10 388 912 404 371 400 521 993 965 887 433 393
425 355 940 989 480 779 60 1 430 867 794 790 787 837 470 437 348 455 860 44 370
883 478 847 461 844 81 479 39 478 871 462 956 831 67 465 955 427 856 514 901 459
894 489 902 902 837 871 842 937 69 979 991 411 89 519 854 48 508 66 957 501 935
910 391 903 858 15 807 487 993 889 848 963 796 488 484 852 931 995 777 990 459
363 785 43 841 38 352 368 937 915 972 785 929 875 33 784 868 83 797 887 397 905
821 35 32 937 23 376 366 475 402 390 11 787 454 379 925 467 473 50 40 919 32 372
845 901 899 479 834 859 880 861 927 507 897 516 863 937 863 513 381 508 958 849
50 439 360 69 913 840 783 938 839 966 352 953 385 48 49 806 867 801 877 916 956
857 361 783 2 941 801 860 955 366 476 893 23 788 445 60 54 806 778 835 3 28 511
45 990 442 887 26 507 995 382 959 88 814 490 881 972 390 48 894 519 774 417 984
447 917 903 8 71 410 14 35 366 972 512 359 825 848 507 451 488 984 857 974 834
782 838 513 839 966 464 773 898 792 371 816 0 429 976 410 890 811 45 5 390 794
486 512 426 376 782 358 826 454 4 487 812 369 70 809 484 867 437 976 86 368 349
867 960 889 922 10 512 74 349 359 472 993 830 935 861 908 447 42 461 392 822 446
809 792 476 460 21 910 437 47 411 358 964 27 903 393 908 478 848 443 409 453 969
859 403 819 811 974 871 43 799 890 31 794 809 398 378 74 361 985 970 773 25 429
10 980 462 993 777 398 452 860 366 11 379 39 847 494 945 429 58 832 408 495 917

```
17 912 908 788 419 862 48 512 512 502 827 420 371 89 389 453 488 892 932
```

Пример вывода:

```
[(0, 92), (93, 343), (344, 521), (522, 772), (773, 999)]
```

Ограничение по времени исполнения программы: 15 с

Ограничение по использованию оперативной памяти: 256 Мб

СПОСОБ ОЦЕНКИ РАБОТЫ:

Для генерации уникального условия и проверки результата используется следующий код на языке Python. Функция `generate` возвращает возвращает набор тестов и правильных ответов:

```
def generate():
    num_tests = 30
    task_size=1000
    tests = list()
    for test in range(num_tests):
        a =[0]*task_size
        a= [i for i in range(0,task_size)]
        groupCount=4
        seqLen=task_size/groupCount

        line=int(seqLen)
        line+=1
        oldLen=len(a)
        for ch in range(int(int(groupCount/2))):
            c = random.randrange(oldLen)
            maxI = int(c+line)
            if (maxI>len(a)):
                maxI=len(a)
            for k in reversed(range(c,maxI)):
                if(len(a)>1):
                    del a[k]

        for ch in range(task_size-len(a)):
            c = random.randrange(len(a))
            a.append(a[c])
        random.shuffle(a)
        tc = ''.join(str(e)+' ' for e in a)
        tc+='\n'
        tests.append(tc)
    return tests

def check(reply, clue):
    return reply == clue
```

РЕШЕНИЕ:

Необходимо понять, что раз номера идут подряд, то после сортировки всех предложенных номеров, заполненные подряд идущие интервалы номеров будут нечетные отряды, а отсутствующие интервалы - четные. После этого существует несколько подходов к решению, но все они так или иначе связаны с сортировкой массива. Один из подходов подразумевает использование сортировки слиянием, добавляя новые элементы в уже отсортированный массив номеров школьников, заполняя при этом интервалы. Второй

состоит в заполнении массива и дальнейшей сортировкой стандартными встроенными алгоритмами и втором проходе по массиву выделяя заполненные интервалы.

Код программы на языке Python:

```
import random
import math
import sys
import array
import random
from collections import OrderedDict
import bisect

def solve(dataset):
    task_size=1000

    ranges=OrderedDict()

    a = dataset.split()

    intEnds=list()
    intBegins=list()

    for item in a:
        c=0
        while((int(c)<int(len(intEnds))) and(int(intEnds[c])<int(item)) ):
            c+=1
        lenEnds=len(intEnds)
        if (int(c) >= int(lenEnds)):
            if (lenEnds>0 and int(intEnds[lenEnds-1]) == int(item)-1):
                intEnds[lenEnds-1]=item
            else:
                intEnds.append(item)
                intBegins.append(item)
        else:
            if (int(intBegins[c])<=int(item)):
                continue
            keyE = intEnds[c]
            keyS = intBegins[c]
            if (int(keyS)==int(item)+1):
                intBegins[c]=item
            else:
                intEnds.insert(c,item)
                intBegins.insert(c,item)
            if (c>0):
                keyP = intEnds[c-1]
                if (int(keyP) == int(item)-1):
                   intBegins[c]=intBegins[c-1]
                   del intBegins[c-1]
                   del intEnds[c-1]
    res1 = list()
    if(int(intBegins[0])>0):
        res1.append((0,int(intBegins[0])))

    for k in range(0,len(intBegins)):
        res1.append((int(intBegins[k]),int(intEnds[k])))
        r1=int(intEnds[k])+1
        r2=task_size
        if(k<len(intBegins)-1):
            r2=int(intBegins[k+1])-1
        if(int(r1)<int(r2)):
```

```
57.             res1.append((int(r1),int(r2)))
58.     return str(res1)
```

Задача 2.1.2 (15 баллов)

Картограф составлял для каждого города список городов, с которыми он связан дорогами. Теперь его спрашивают о том, можно ли проехать между двумя далекими городами.

На вход подается два названия города, которые нужно проверить и дальше идет идет перечисление для городов, с какими другими городами он связан дорогой. Ответ необходимо давать в формате True и False.

За успешное решение задачи участники получают 15 баллов.

Пример ввода:

```
{'find': ('d', 'f'), 'd': ['a', 'b', 'c', 'e', 'f', 'g'], 'b': ['a', 'c', 'd', 'e', 'f', 'g'], 'q': ['n', 'o', 'p', 'r', 's', 't', 'u'], 'f': ['a', 'b', 'c', 'd', 'e', 'g'], 'a': ['b', 'c', 'd', 'e', 'f', 'g'], 's': ['n', 'o', 'p', 'q', 'r', 't', 'u', 'c'], 'e': ['a', 'b', 'c', 'd', 'f', 'g'], 'u': ['n', 'o', 'p', 'q', 'r', 's', 't'], 'r': ['n', 'o', 'p', 'q', 's', 't', 'u'], 'p': ['n', 'o', 'q', 'r', 's', 't', 'u'], 'c': ['a', 'b', 'd', 'e', 'f', 'g'], 'g': ['a', 'b', 'c', 'd', 'e', 'f'], 'o': ['n', 'p', 'q', 'r', 's', 't', 'u'], 'n': ['o', 'p', 'q', 'r', 's', 't', 'u'], 't': ['n', 'o', 'p', 'q', 'r', 's', 'u']}
```

Ограничение по времени исполнения программы: 3000 с

Ограничение по использованию оперативной памяти: 256 Мб

СПОСОБ ОЦЕНКИ РАБОТЫ:

Для генерации уникального условия и проверки результата используется следующий код на языке Python. Функция `generate` возвращает набор тестов и правильные ответы. Пример нескольких тестов с правильными ответами представлен ниже:

```
def generate():
    tests = list()
    tests.append("{'find': ('d', 'f'), 'd': ['a', 'b', 'c', 'e', 'f', 'g'], 'b': ['a', 'c', 'd', 'e', 'f', 'g'], 'q': ['n', 'o', 'p', 'r', 's', 't', 'u'], 'f': ['a', 'b', 'c', 'd', 'e', 'g'], 'a': ['b', 'c', 'd', 'e', 'f', 'g'], 's': ['n', 'o', 'p', 'q', 'r', 't', 'u', 'c'], 'e': ['a', 'b', 'c', 'd', 'f', 'g'], 'u': ['n', 'o', 'p', 'q', 'r', 's', 't'], 'r': ['n', 'o', 'p', 'q', 's', 't', 'u'], 'p': ['n', 'o', 'q', 'r', 's', 't', 'u'], 'c': ['a', 'b', 'd', 'e', 'f', 'g'], 'g': ['a', 'b', 'c', 'd', 'e', 'f'], 'o': ['n', 'p', 'q', 'r', 's', 't', 'u'], 'n': ['o', 'p', 'q', 'r', 's', 't', 'u'], 't': ['n', 'o', 'p', 'q', 'r', 's', 'u']}\n")
    tests.append("{'r': ['n', 'o', 'p', 'q', 's', 't', 'u', 'v', 'w', 'z', 'y', 'x'], 'v': ['n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'w', 'z', 'y', 'x'], 'e': ['a', 'b', 'c', 'd', 'f'], 'y': ['n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'z', 'x'], 'z': ['n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'y', 'x'], 'q': ['n', 'o', 'p', 'r', 's', 't', 'u', 'v', 'w', 'z', 'y', 'x'], 'f': ['a', 'b', 'c', 'd', 'e'], 'd': ['a', 'b', 'c', 'e', 'f'], 's': ['n', 'o', 'p',
```

```
'q', 'r', 't', 'u', 'v', 'w', 'z', 'y', 'x'], 't': ['n', 'o', 'p', 'q', 'r',
's', 'u', 'v', 'w', 'z', 'y', 'x'], 'n': ['o', 'p', 'q', 'r', 's', 't', 'u',
'v', 'w', 'z', 'y', 'x'], 'w': ['n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v',
'z', 'y', 'x'], 'c': ['a', 'b', 'd', 'e', 'f'], 'a': ['b', 'c', 'd', 'e', 'f'],
'o': ['n', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'z', 'y', 'x'], 'find': ('r',
'v'), 'x': ['n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'z', 'y'], 'p':
['n', 'o', 'q', 'r', 's', 't', 'u', 'v', 'w', 'z', 'y', 'x'], 'b': ['a', 'c',
'd', 'e', 'f'], 'u': ['n', 'o', 'p', 'q', 'r', 's', 't', 'v', 'w', 'z', 'y',
'x', 'c']}\n")
    tests.append("{'s': ['n', 'o', 'p', 'q', 'r'], 'r': ['n', 'o', 'p', 'q',
's'], 'n': ['o', 'p', 'q', 'r', 's'], 'c': ['a', 'b', 'd', 'e', 'f', 'g', 'h',
'i'], 'e': ['a', 'b', 'c', 'd', 'f', 'g', 'h', 'i'], 'i': ['a', 'b', 'c', 'd',
'e', 'f', 'g', 'h'], 'g': ['a', 'b', 'c', 'd', 'e', 'f', 'h', 'i'], 'a': ['b',
'c', 'd', 'e', 'f', 'g', 'h', 'i'], 'o': ['n', 'p', 'q', 'r', 's'], 'find':
('h', 's'), 'h': ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'i'], 'p': ['n', 'o', 'q',
'r', 's'], 'f': ['a', 'b', 'c', 'd', 'e', 'g', 'h', 'i'], 'b': ['a', 'c', 'd',
'e', 'f', 'g', 'h', 'i'], 'd': ['a', 'b', 'c', 'e', 'f', 'g', 'h', 'i'], 'q':
['n', 'o', 'p', 'r', 's']}\n")
#...
    return tests;

def check(reply, clue):
    return str(reply) == str(clue)
```

РЕШЕНИЕ:

Необходимо распознать в задаче проблему поиска пути между двумя вершинами в графе. Школьникам необходимо выбрать способ представления графа в памяти и осуществить поиск в ширину (Bread-first search) от одной вершины к другой. Если алгоритм закончил работу не встретив второй вершины, то пути нет, иначе — есть.

Код решения на языке Python:

```
 1. import ast
 2.
 3. class Graph(object):
 4.
 5.     def __init__(self, graph_dict={}):
 6.         """ initializes a graph object """
 7.         self.__graph_dict = graph_dict
 8.
 9.     def vertices(self):
10.         """ returns the vertices of a graph """
11.         return list(self.__graph_dict.keys())
12.
13.     def edges(self):
14.         """ returns the edges of a graph """
15.         return self.__generate_edges()
16.
17.     def add_vertex(self, vertex):
18.         """ If the vertex "vertex" is not in
19.             self.__graph_dict, a key "vertex" with an empty
20.             list as a value is added to the dictionary.
21.             Otherwise nothing has to be done.
22.         """
23.         if vertex not in self.__graph_dict:
24.             self.__graph_dict[vertex] = []
25.
26.     def add_edge(self, edge):
27.         """ assumes that edge is of type set, tuple or list;
```

```
            between two vertices can be multiple edges!
        """
        edge = set(edge)
        vertex1 = edge.pop()
        if edge:
            # not a loop
            vertex2 = edge.pop()
        else:
            # a loop
            vertex2 = vertex1
        if vertex1 in self.__graph_dict:
            self.__graph_dict[vertex1].append(vertex2)
        else:
            self.__graph_dict[vertex1] = [vertex2]

    def __generate_edges(self):
        """ A static method generating the edges of the
            graph "graph". Edges are represented as sets
            with one (a loop back to the vertex) or two
            vertices
        """
        edges = []
        for vertex in self.__graph_dict:
            for neighbour in self.__graph_dict[vertex]:
                if {neighbour, vertex} not in edges:
                    edges.append({vertex, neighbour})
        return edges

    def __str__(self):
        res = "vertices: "
        for k in self.__graph_dict:
            res += str(k) + " "
        res += "\nedges: "
        for edge in self.__generate_edges():
            res += str(edge) + " "
        return res

    def find_isolated_vertices(self):
        """ returns a list of isolated vertices. """
        graph = self.__graph_dict
        isolated = []
        for vertex in graph:
            print(isolated, vertex)
            if not graph[vertex]:
                isolated += [vertex]
        return isolated

    def find_path(self, start_vertex, end_vertex, path=[]):
        """ find a path from start_vertex to end_vertex
            in graph """
        graph = self.__graph_dict


        if (start_vertex==end_vertex):
            return True
        checked=[]
        added=[]
        checked.append(start_vertex)
        added.append(start_vertex)
        res=False
        while (len(added)>0):
```

```
            newAdded=[]
            for vert in added:
               for newVert in graph[vert]:
                  if newVert not in checked:
                      if newVert == end_vertex:
                          res=True
                          break
                      checked.append(newVert)
                      newAdded.append(newVert)
            added=newAdded
        return res


        path = path + [start_vertex]
        if start_vertex == end_vertex:
            return path
        if start_vertex not in graph:
            return None
        for vertex in graph[start_vertex]:
            if vertex not in path:
                extended_path = self.find_path(vertex,
                                               end_vertex,
                                               path)
                if extended_path:
                    return extended_path
        return None


    def find_all_paths(self, start_vertex, end_vertex, path=[]):
        """ find all paths from start_vertex to
            end_vertex in graph """
        graph = self.__graph_dict
        path.append(start_vertex)
        if start_vertex == end_vertex:
            return [path]
        if start_vertex not in graph:
            return []
        paths = []
        nextPath=path
        for vertex in graph[start_vertex]:
            nextPath.append(vertex)

        for vertex in graph[start_vertex]:
            if vertex not in path :
                extended_paths = self.find_all_paths(vertex,
                                                     end_vertex,
                                                     nextPath)
                for p in extended_paths:
                    paths.append(p)
        return paths

    def is_connected(self,
                     vertices_encountered = None,
                     start_vertex=None):
        """ determines if the graph is connected """
        if vertices_encountered is None:
            vertices_encountered = set()
        gdict = self.__graph_dict
        vertices = list(gdict.keys()) # "list" necessary in Python 3
        if not start_vertex:
            # chosse a vertex from graph as a starting point
```

```
            start_vertex = vertices[0]
        vertices_encountered.add(start_vertex)
        if len(vertices_encountered) != len(vertices):
            for vertex in gdict[start_vertex]:
                if vertex not in vertices_encountered:
                    if self.is_connected(vertices_encountered, vertex):
                        return True
        else:
            return True
        return False

    def vertex_degree(self, vertex):
        """ The degree of a vertex is the number of edges connecting
            it, i.e. the number of adjacent vertices. Loops are counted
            double, i.e. every occurence of vertex in the list
            of adjacent vertices. """
        adj_vertices =  self.__graph_dict[vertex]
        degree = len(adj_vertices) + adj_vertices.count(vertex)
        return degree

    def degree_sequence(self):
        """ calculates the degree sequence """
        seq = []
        for vertex in self.__graph_dict:
            seq.append(self.vertex_degree(vertex))
        seq.sort(reverse=True)
        return tuple(seq)

    @staticmethod
    def is_degree_sequence(sequence):
        """ Method returns True, if the sequence "sequence" is a
            degree sequence, i.e. a non-increasing sequence.
            Otherwise False is returned.
        """
        # check if the sequence sequence is non-increasing:
        return all( x>=y for x, y in zip(sequence, sequence[1:]))


    def delta(self):
        """ the minimum degree of the vertices """
        min = 100000000
        for vertex in self.__graph_dict:
            vertex_degree = self.vertex_degree(vertex)
            if vertex_degree < min:
                min = vertex_degree
        return min

    def Delta(self):
        """ the maximum degree of the vertices """
        max = 0
        for vertex in self.__graph_dict:
            vertex_degree = self.vertex_degree(vertex)
            if vertex_degree > max:
                max = vertex_degree
        return max

    def density(self):
        """ method to calculate the density of a graph """
        g = self.__graph_dict
        V = len(g.keys())
        E = len(self.edges())
```

```
211.         return 2.0 * E / (V *(V - 1))
212.
213.     def diameter(self):
214.         """ calculates the diameter of the graph """
215.
216.         v = self.vertices()
217.         pairs = [ (v[i],v[j]) for i in range(len(v)) for j in range(i+1,
len(v)-1)]
218.         smallest_paths = []
219.         for (s,e) in pairs:
220.             paths = self.find_all_paths(s,e)
221.             smallest = sorted(paths, key=len)[0]
222.             smallest_paths.append(smallest)
223.
224.         smallest_paths.sort(key=len)
225.
226.         # longest path is at the end of list,
227.         # i.e. diameter corresponds to the length of this path
228.         print(smallest_paths[-1])
229.         diameter = len(smallest_paths[-1])
230.         return diameter
231.
232.     @staticmethod
233.     def erdoes_gallai(dsequence):
234.         """ Checks if the condition of the Erdoes-Gallai inequality
235.             is fullfilled
236.         """
237.         if sum(dsequence) % 2:
238.             # sum of sequence is odd
239.             return False
240.         if Graph.is_degree_sequence(dsequence):
241.             for k in range(1,len(dsequence) + 1):
242.                 left = sum(dsequence[:k])
243.                 right =  k * (k-1) + sum([min(x,k) for x in dsequence[k:]])
244.                 if left > right:
245.                     return False
246.         else:
247.             # sequence is increasing
248.             return False
249.         return True
250.
251. def solve(dataset):
252.     g = ast.literal_eval(dataset);
253.     a, b = g['find']
254.     g.pop("find", None)
255.     graph = Graph(g)
256.     return str(graph.find_path(a, b));
```

Задача 2.1.3 (0-20 баллов)

В задаче фигурирует придуманный нами граф социальной сети (для каждого пользователя указано, какие пользователя добавили его в друзья). Для каждого пользователя вычисляется его популярность, она основана на том, сколько людей дружит с теми пользователями, с которым он дружит.

Популярность вычисляется как X - суммарное количество людей, которые дружат с его друзьями, считая его самого.

Необходимо рассчитать два перцентиль 50 и 90, т.е. два наименьших значения популярности такие, что с вероятностью 50% для первого и 90% для второго популярность случайного пользователя будет меньше данного значения.

Задачу необходимо решить у себя на компьютере и в систему загрузить решение в виде двух чисел для каждой задачи.

Если для всех тестов вы посчитали хотя бы перцентиль 50 (с вероятностью 50 процентов популярность окажется меньше), то вы получаете половину баллов от задачи.

Пример ввода:

```
[{0: [5, 8], 1: [7, 2], 2: [8], 3: [10, 0], 4: [0, 10, 2, 1], 5: [1, 5, 3, 7], 6: [7, 3, 0], 7: [12, 13, 0, 8], 8: [8, 11], 9: [6, 2, 13], 10: [13], 11: [2, 0, 8], 12: [0, 13], 13: [4, 11, 8], 14: [0, 4, 12, 2]}, {0: [14, 11], 1: [7, 14, 1], 2: [0], 3: [10], 4: [13], 5: [8, 0], 6: [5, 3], 7: [2, 11, 8, 10], 8: [3, 10, 14, 7], 9: [0, 11, 7, 4], 10: [1, 7], 11: [10, 5, 12, 4], 12: [14, 5], 13: [7, 6, 3, 1], 14: [10, 6, 0, 8]}]
```

Пример вывода:

```
[(4, 6), (7, 9)]
```

Ограничение по использованию оперативной памяти: 256 Мб

Время одной попытки: 5 мин

СПОСОБ И КРИТЕРИЙ ОЦЕНКИ РАБОТЫ:

Для генерации уникального условия и проверки результата используется следующий код на языке Python. Функция `generate` возвращает набор тестов и правильные ответы:

```
def generate():
    numOfTests=3
    maxFrCount=200
    numOfV = 10000
    avgFr = 7
    dispersion = 4

    testsList=list()
    for test in range(numOfTests):
        graph={}
        for v in range(numOfV):
            dispVal = int(random.randrange(dispersion) - (int(dispersion)//2))
            sign = random.randrange(2)
            frCount = 0
            if int(sign) == 0:
                dispVal = int(random.randrange(avgFr))
                frCount = int(avgFr )- int(dispVal)
            else:
                dispVal = int(random.randrange(int(maxFrCount) - int(avgFr )))
                frCount=  int(avgFr) +int(dispVal)
            frList=list()
            for frN in range(frCount):

                frN=random.randrange(numOfV)
                while frN in frList:
```

```
                    frN=random.randrange(numOfV)
                frList.append(frN)
            graph.update({v:frList})
        testsList.append(graph)
        return str(testsList)

def check(reply, clue):
    repA= ast.literal_eval(reply)
    clA = ast.literal_eval(clue)
    if len(repA)!=len(clA):
        return False

    resHalf=True
    resFull=True
    for ind in range(len(repA)):
        rep=repA[ind]
        cl=clA[ind]
```

Функция проверки `check` возвращает показатель точности решения (от 0 до 1). Это значение умножается на 20, что дает число баллов, полученных командой.

РЕШЕНИЕ:

Для данное задачи необходимо сначала принять решение о внутреннем представлении данных графа социальной сети. После этого для всех пользователей рассчитать значение популярности. Затем необходимо отсортировать пользователей по популярности (основываясь на знаниях из первой задачи) и взять соотвествующий элемент массива (средний для перцентиля 50 и находящийся на позиции 0.9*N где N-число элементов массива, для перцентиля 90).

Код программы на языке Python:

```
 1. import random
 2. import ast
 3. import math
 4.
 5. def solve(dataset):
 6.     checkTests = ast.literal_eval(dataset)
 7.     answ=list()
 8.     for test in checkTests:
 9.         testLen = len(test)
10.         valList=list()
11.         for v in range(testLen):
12.
13.
14.             frVList=test[v]
15.             frfrList=list()
16.             for frV in frVList:
17.                 frfrList.extend(test[frV])
18.             frfrList=set(frfrList)
19.             val = int(len(frfrList))
20.
21.             uppId=int(len(valList))
22.             lowId=0
23.             found=False
24.             while int(math.fabs(int(uppId)-int(lowId)))>1:
```

```
                med=int((int(uppId) + int(lowId))//2)
                if int(valList[med])<int(val):
                    lowId=med
                else:
                    uppId=med
                    if int(valList[med])== int(val):
                        break
            if (len(valList)>0 and int(val) > int(valList[lowId])):
                valList.insert(uppId,val)
            else:
                valList.insert(lowId,val)

        res1,mod=divmod(testLen,2)
        if mod>0:
            res1+=0
        res1-=1
        res2,mod=divmod(testLen*9,10)

        if (mod>0):
            res2+=0
        res2-=1

        answ.append((valList[res1],valList[res2]))
    return(str(answ))
```