§2 Второй отборочный этап

Второй отборочный этап проводится в командном формате в сети интернет. Продолжительность второго отборочного этапа — 8 недель. Работы оцениваются автоматически средствами stepik. Задачи носят междисциплинарный характер и в упрощенной форме воссоздают инженерную задачу заключительного этапа, участники должны были писать программы на языке C, C++ или python.

Разработана автоматическая система генерации уникального условия конкретной задачи для каждого участника, а также автоматизирован процесс проверки результата решения задачи. Все условия задач доступны участникам с первого дня второго отборочного этапа. Участникам можно было получить суммарно от 0 до 20 баллов.

Задача 2.1 (2 балла)

Условие: Задана строка, содержащая латинские буквы и цифры и пробелы. Напишите программу, которая печатает символы строки в следующем порядке: ПРОБЕЛ, ЦИФРЫ ПО ВОЗРАСТАНИЮ, БУКВЫ ПО АЛФАВИТУ ЗАГЛАВНЫЕ, БУКВЫ ПО АЛФАВИТУ СТРОЧНЫЕ.

Строка вводится с клавиатуры. Ответ заключить в квадратные скобки.

2.1.1. Формат входных данных

На вход программе подаётся строка из латинских букв, цифр и пробелов.

Пример ввода:

ABCBA ZYX F QzxTTTT BB TWT 34 R

2.1.2. Формат выходных данных

Вывести в квадратных скобках без разделителей сначала пробелы этой строки, цифры по возравстанию, буквы по алфавиту, сначала заглавные, потом строчные.

Пример вывода:

[34ABCFQRTWXYZxz]

Способ оценки работы

Условие: Для генерации уникального условия и проверки результата используется следующий код на языке Python. Функция generate возвращает набор в том числе случайным образом сгенерированных под задачу тестов и после прохода через эталонное решение, получаются ответы на сгенерированные тесты (тесты и ответы одинаковые для всех участников, результаты программы участника и ответы сравниваются при помощи функции check):

```
 1 import random
 2 import string
 3
 4 def get_random_letters(l, r):
 5     alphabet = string.ascii_letters + string.digits
+ ' '
 6     return ''.join(random.choice(alphabet) for _ in
range(random.randint(l, r)))
 7
 8 def get_test(l, r):
 9     s = get_random_letters(l, r)
10     return "{}\n".format(s)
11
12
13 def generate():
14     tests = []
15
16     s = 'ABCBA ZYX F QzxTTTT BB TWT 34 R'
17     test_case = "{}\n".format(s)
18     tests.append(test_case)
19
20     for test in range(49):
21         tests.append(get_test(0, 1000))
22     for test in range(50):
23         tests.append(get_test(0, 50))
24     return tests
25
26 def check(reply, clue):
27     return reply == clue
```

Решение:

В этой задаче нужно аккуратно реализовать то, что указано в условии. Пример программы на языке Python:

```
1 def solve(dataset):
2                                                 return
"[{}]".format(''.join(sorted(set(dataset.strip('\n
\r')))))
3
  1.solve(sys.stdin.read())
```

Задача 2.2 (3 балла)

Условие: На вход программе задается целочисленная прямоугольная матрица: в первой строке задается количество строк в матрице N и количество столбцов M. Далее задается матрица соответствующих размеров: N строк по M элементов.

Требуется написать программу, которая:

- Выводит в первой строке номер первой из строк матрицы, содержащих хотя бы один отрицательный элемент (начиная с нуля) или значение "-1", если отрицательные элементы в матрице отсутствуют.
- В следующей строке через пробел выводит минимумы в тех столбцах матрицы, которые содержат максимальный элемент.
- Далее в N строках выводит матрицу с обратным порядком строк (поменять 0-ю строку с последней, 1-ю с предпоследней, и.т.д)

Чтение данных производится со стандартного потока ввода, печать -- в стандартный поток вывода.

Считать исходные данные корректными.

2.2.1. Формат входных данных

На вход программе подаются два целых неотрицательных числа – количество строк и столбцов матрицы.

Пример ввода:

5 3

1 2 3

4 5 6

-7 -8 -9

10 11 14

13 14 -15

2.2.2. Формат выходных данных:

Вывести в первой строке номер первой из строк матрицы, содержащих хотя бы один отрицательный элемент (начиная с нуля) или значение "-1", если

отрицательные элементы в матрице отсутствуют.

В следующей строке через пробел выводит минимумы в тех столбцах матрицы, которые содержат максимальный элемент.

Далее в N строках выводит матрицу с обратным порядком строк (поменять 0-ю строку с последней, 1-ю с предпоследней, и.т.д)

Пример вывода:

2

-8 -15

13 14 -15

10 11 14

-7 -8 -9

4 5 6

1 2 3

Способ оценки работы:

Для генерации уникального условия и проверки результата используется следующий код на языке Python. Функция generate возвращает набор в том числе случайным образом сгенерированных под задачу тестов и после прохода через эталонное решение, получаются ответы на сгенерированные тесты (тесты и ответы одинаковые для всех участников, результаты программы участника и ответы сравниваются при помощи функции check):

```
 1 import random
 2
 3 def generate():
 4     tests = []
 5     n = 5
 6     m = 3
 7      random_matrix = [[1,  2,  3], [4,  5,  6], [-7,
-8,  -9], [10,  11,  14], [13,  14,  -15]]
 8
 9     test_case = "{} {}\n".format(n, m)
10      test_case += '\n'.join(' '.join(str(e) for e in
line) for line in random_matrix)
```

```
11      test_case += '\n'
12      tests.append(test_case)
13
14      for test in range(2):
15          n = random.randint(1, 5)
16          m = random.randint(1, 5)
17          test_case = "{} {}\n".format(n, m)
18           random_matrix = [[random.randint(-5, 5) for
e in range(m)] for e in range(n)]
19            test_case += '\n'.join(' '.join(str(e) for
e in line) for line in random_matrix)
20          test_case += '\n'
21          tests.append(test_case)
22
23      n = random.randint(1, 5)
24      m = random.randint(1, 5)
25      test_case = "{} {}\n".format(n, m)
26       random_matrix = [[random.randint(0, 0) for e in
range(m)] for e in range(n)]
27       test_case += '\n'.join(' '.join(str(e) for e in
line) for line in random_matrix)
28      test_case += '\n'
29      tests.append(test_case)
30
31      for test in range(3):
32          n = random.randint(1, 5)
33          m = random.randint(1, 5)
34          test_case = "{} {}\n".format(n, m)
35            random_matrix = [[random.randint(0, 5) for
e in range(m)] for e in range(n)]
36            test_case += '\n'.join(' '.join(str(e) for
e in line) for line in random_matrix)
37          test_case += '\n'
38          tests.append(test_case)
39
```

```
40     for test in range(3):
41         n = random.randint(1, 5)
42         m = random.randint(1, 5)
43         test_case = "{} {}\n".format(n, m)
44          random_matrix = [[random.randint(-1, 5) for
e in range(m)] for e in range(n)]
45           test_case += '\n'.join(' '.join(str(e) for
e in line) for line in random_matrix)
46         test_case += '\n'
47         tests.append(test_case)
48
49     for test in range(20):
50         n = random.randint(1, 100)
51         m = random.randint(1, 100)
52         test_case = "{} {}\n".format(n, m)
53                 random_matrix = [[random.randint(-1,
1000000) for e in range(m)] for e in range(n)]
54           test_case += '\n'.join(' '.join(str(e) for
e in line) for line in random_matrix)
55         test_case += '\n'
56         tests.append(test_case)
57
58     for test in range(20):
59         n = random.randint(100, 500)
60         m = random.randint(100, 500)
61         test_case = "{} {}\n".format(n, m)
62               random_matrix = [[random.randint(-100,
1000000) for e in range(m)] for e in range(n)]
63           test_case += '\n'.join(' '.join(str(e) for
e in line) for line in random_matrix)
64         test_case += '\n'
65         tests.append(test_case)
66
67     for test in range(50):
68         n = random.randint(1, 100)
```

```
69          m = random.randint(1, 100)
70          test_case = "{} {}\n".format(n, m)
71                random_matrix = [[random.randint(-100,
1000000) for e in range(m)] for e in range(n)]
72            test_case += '\n'.join(' '.join(str(e) for
e in line) for line in random_matrix)
73          test_case += '\n'
74          tests.append(test_case)
75
76      return tests
77
78 def check(reply, clue):
79      return reply.split() == clue.split()
```

Решение:

В этой задаче нужно аккуратно реализовать то, что указано в условии: сначала пройтись по строкам и найти первый минимальные элемент и номер его строки. В случае, если прошли все строки, а отрицательного элемента нет – выводим «-1». Далее осуществляем поиск максимального элемента матрицы, также пробегаясь по всем элементам. Потом просматриваем столбцы матрицы на наличие максимального элемента и ищем минимальный элемент в соответствующем столбце. После этого этапа осуществляем печать матрицы, изменив порядок строк с порядка возрастания на порядок убывания. Пример программы на языке Python:

```
2.
3. 1 def get_firstline_before_zero(matrix):
4. 2     for i, row in enumerate(matrix):
5. 3         if any(x < 0 for x in row):
6. 4             return i
7. 5     return -1
8. 6
9. 7
10. 8 def solve(dataset):
11. 9     input = dataset.split()
12.10     n = int(input[0])
13.11     m = int(input[1])
```

```
14.12     matrix = [[int(input[2 + x * m + y]) for y
  in range(m)] for x in range(n)]
15.13
16.14             firstline_before_zero =
  get_firstline_before_zero(matrix)
17.15
18.16         max_value = max([max(row) for row in
  matrix])
19.17
20.18           min_values = [min(col) for col in
  zip(*matrix) if any(x == max_value for x in col)]
21.19
22.20     ans = "{}\n".format(firstline_before_zero)
23.21           ans += ' '.join(str(e) for e in
  min_values) + '\n'
24.22      ans += '\n'.join(' '.join(str(e) for e in
  line) for line in matrix[::-1]) + '\n'
25.23
26.24     return ans
27.25
28.26 solve(sys.stdin.read())
```

Задача 2.3. (4 балла)

На вход программе парами через пробел подаются 4 точки: сначала координаты по оси абсцисс, потом координаты по оси ординат для каждой из точек.

Требуется написать программу, проверяющую, образуют ли первые три точки треугольник и принадлежит ли четвертая точка ему.

- Если три первые точки не образуют треугольник, выведите «-1".
- Если точка принадлежит треугольнику, выведите «1".
- Если не принадлежит, выведите "0".

Чтение данных производится со стандартного потока ввода, печать -- в стандартный поток вывода.

Считать исходные данные корректными.

2.3.1. Формат входных данных

На вход программе подаются 8 целых чисел, не превосходящих по модулю

$10^6 10^6$

. Эти числа характеризуют значения абсцисс и ординат 4 точек на плоскости.

Пример ввода 1:

-10 0 0 10 10 0 0 5

Пример ввода 2:

-10 0 0 10 10 0 0 -5

Пример ввода 3:

-10 0 -10 0 -10 0 -10 0

2.3.2. Формат выходных данных

Вывести целое число «-1», «0» или «1» -- ответ задачи.

Пример вывода 1:

1

Пример вывода 2:

0

Пример вывода 3:

-1

Примечание:

В первом случае три точки образуют треугольник. Четвертая лежит внутри него. Во втором случае – треугольник тот же, но точка ему не принадлежит. В третьем случае три точки не образуют треугольник. В случае, если точка лежит на стороне треугольника, она ему принадлежит.

Способ оценки работы

Для генерации уникального условия и проверки результата используется следующий код на языке Python. Функция generate возвращает набор в том числе случайным образом сгенерированных под задачу тестов и после прохода через эталонное решение, получаются ответы на сгенерированные тесты (тесты и ответы одинаковые для всех участников, результаты программы участника и ответы сравниваются при помощи функции check):

```
1 import random
2
3 def generate():
```

```
 4     tests = []
 5
 6     coords = [-10, 0, 0, 10, 10, 0, 0, 5]
 7         test_case = ' '.join(str(e) for e in
coords) + '\n'
 8     tests.append(test_case)
 9
10     coords = [-10, 0, 0, 10, 10, 0, 0, -5]
11         test_case = ' '.join(str(e) for e in
coords) + '\n'
12     tests.append(test_case)
13
14     coords = [-10, 0, -10, 0, -10, 0, -10, 0]
15         test_case = ' '.join(str(e) for e in
coords) + '\n'
16     tests.append(test_case)
17
18      coords = [0, -2, 0, 2, 1000000, 0, 499999,
1]
19         test_case = ' '.join(str(e) for e in
coords) + '\n'
20     tests.append(test_case)
21
22      coords = [0, -2, 0, 2, 1000000, 0, 500001,
1]
23         test_case = ' '.join(str(e) for e in
coords) + '\n'
24     tests.append(test_case)
25
26      coords = [0, -2, 0, 2, 1000000, 0, 500000,
1]
27         test_case = ' '.join(str(e) for e in
coords) + '\n'
28     tests.append(test_case)
29
```

```
30      coords = [0, -2, 0, 2, 999999, 0, 500000,
1]
31          test_case = ' '.join(str(e) for e in
coords) + '\n'
32      tests.append(test_case)
33
34      coords = [0, -2, 0, 2, 1000001, 0, 500000,
1]
35          test_case = ' '.join(str(e) for e in
coords) + '\n'
36      tests.append(test_case)
37
38          coords = [-1000001, -1000000, 1000000,
1000001, 1, -1, 0, 0]
39          test_case = ' '.join(str(e) for e in
coords) + '\n'
40      tests.append(test_case)
41
42          coords = [-1000001, -1000000, 1000000,
1000001, 1, -1, -1, 1]
43          test_case = ' '.join(str(e) for e in
coords) + '\n'
44      tests.append(test_case)
45
46          coords = [-1000001, -1000000, 1000000,
1000001, 0, 1, -1, 1]
47          test_case = ' '.join(str(e) for e in
coords) + '\n'
48      tests.append(test_case)
49
50          coords = [-1, 0, 1, 0, 0, 1000000, 0,
1000000]
51          test_case = ' '.join(str(e) for e in
coords) + '\n'
52      tests.append(test_case)
53
```

```
54         coords = [-1, 0, 1, 0, 0, 1000000, 0,
1000001]
55          test_case = ' '.join(str(e) for e in
coords) + '\n'
56     tests.append(test_case)
57
58         coords = [-1, 0, 1, 0, 0, 1000000, 0,
999999]
59          test_case = ' '.join(str(e) for e in
coords) + '\n'
60     tests.append(test_case)
61
62     for test in range(11):
63           coords = [random.randint(-1, 1) for e
in range(8)]
64           test_case = ' '.join(str(e) for e in
coords) + '\n'
65         tests.append(test_case)
66     for test in range(45):
67          coords = [random.randint(-100, 100) for
e in range(8)]
68           test_case = ' '.join(str(e) for e in
coords) + '\n'
69         tests.append(test_case)
70     for test in range(30):
71              coords = [random.randint(-1000000,
1000000) for e in range(8)]
72           test_case = ' '.join(str(e) for e in
coords) + '\n'
73         tests.append(test_case)
74
75     return tests
76
77 def check(reply, clue):
78     return int(reply) == int(clue)
```

Решение:

В этой задаче нужно понять, что три точки образуют треугольник тогда, когда они не лежат на одной прямой. Проверить это можно, сместив все точки таким образом, чтобы хотя бы одна совпала с началом координат. Так мы из двух других точек получаем координаты векторов. Если вектора коллинеарны друг другу, то точки не образуют треугольник. Проверить это можно из соотношений координат $\frac{x_3-x_1}{x_2-x_1}=\frac{y_3-y_1}{y_2-y_1}\frac{x_3-x_1}{x_2-x_1}=\frac{y_3-y_1}{y_2-y_1}$. При решении задач в целых числах, а также при наличии нулевых компонент вектора формула в данном виде работать не будет, однако, ничего не мешает избавиться от деления, домножив левую и правую часть на знаменатели. Следующий этап решения задачи – проверка принадлежности точки решается путем проверки знака знаковой площади треугольников, образуемых вершинами треугольника с проверяемой вершиной. Это делается путем проверки знака псевдоскалярного произведения векторов, образующих данные треугольники. Если все знаки совпадут, то точка принадлежит треугольнику.

Пример программы на языке Python:

```
 1 def not_is_triangle(x1, y1, x2, y2, x3, y3):
 2     return (x3 - x1) * (y2 - y1) == (y3 - y1) * (x2 - x1)
 3
 4 def is_in_triangle(x1, y1, x2, y2, x3, y3, x0, y0):
 5     a = (x1 - x0) * (y2 - y1) - (x2 - x1) * (y1 - y0)
 6     b = (x2 - x0) * (y3 - y2) - (x3 - x2) * (y2 - y0)
 7     c = (x3 - x0) * (y1 - y3) - (x1 - x3) * (y3 - y0)
 8     return all(x <= 0 for x in [a, b, c]) or all(x >= 0 for x in [a, b, c])
 9
10 def solve(dataset):
11     x1, y1, x2, y2, x3, y3, x0, y0 = [int(x) for x in dataset.split()]
12     if not_is_triangle(x1, y1, x2, y2, x3, y3):
13         return "-1\n"
```

```
14        if is_in_triangle(x1, y1, x2, y2, x3, y3, x0,
y0):
15            return "1\n"
16      return "0\n"
17
18 solve(sys.stdin.read())
```

Задача 2.4. (5 баллов)

Условие: Вы решили запрограммировать летающего робота, который максимально быстро сможет пройти трехмерный лабиринт. Вам повезло, и у Вас есть план этого лабиринта. Внешне он выглядит как трёхмерный куб, состоящий из $n^3\,n^3$ маленьких кубиков и ограниченный со всех сторон стенкой. Все внутренние стены перпендикулярны сторонам куба, причем стоят они, разделяя внутренние кубики. У каждого внутреннего кубика есть своя целочисленная координата от 0 до $n^3-1\,n^3-1$, причём нумеруются они подряд идущими слоями, то есть для кубика с гранью 3:

первый слой: 00 01 02 второй слой: 09 10 11 третий слой: 18 19 20

03 04 05 12 13 14 21 22 23

06 07 08 15 16 17 24 25 26

Для лабиринтов больших размеров, аналогично.

На вход Вашей программе подается размер лабиринта n, координата входа в лабиринт (некоторая координата внутреннего кубика, лежащего на одной из граней), координата выхода из кубика (некоторая координата внутреннего кубика, лежащего на одной из граней, не совпадающая с координатами входа) и количество внутренних стен k, за которым следует соответственное количество пар координат кубиков, между которыми стоит стена. Ваша задача через пробел вывести наикратчайшую последовательность координат, по которым пролетит робот от входа к выходу. Если таких последовательностей несколько – выведите наименьшую. Например, в первом тесте различных путей 6:

0 1 3 7 ; 0 1 5 7 ; 0 2 3 7 ; 0 2 6 7 ; 0 4 5 7 ; 0 4 6 7

Они упорядочены по возрастанию, поэтому ответ к задаче -- 0 1 3 7.

Если до выхода добраться нельзя, выведите «-1».

2.4.1. Формат входных данных

В первой строке натуральное число от 0 до 6 – размер лабиринта n, в следующей строке через пробел целочисленная координата входа в лабиринт и координата выхода из кубика, далее в следующей строке неотрицательное количество внутренних стен k, за которым в k строках следуют пары целочисленных координат кубиков, между которыми стоит стена.

Пример ввода 1:

2

0 7

0

Пример ввода 2:

2

0 7

1

0 1

Пример ввода 3:

2

0 7

2

0 1

2 3

Пример ввода 4:

2

0 7

3

0 1

2 3

6 7

Пример ввода 5:

2

0 7

4

0 1

2 3

6 7

4 5

Пример ввода 6:

2

0 4

5

0 4

1 5

3 7

0 2

4 6

2.4.2. Формат выходных данных

Вывести последовательность кубиков, через которые пройдет робот. Или «-1», если пути нет.

Пример вывода 1:

0 1 3 7

Пример вывода 2:

0 2 3 7

Пример вывода 3:

0 2 6 7

Пример вывода 4:

0 4 5 7

Пример вывода 5:

-1

Пример вывода 6:

0 1 3 2 6 7 5 4

Примечание:

В примерах наглядно приведены различные случаи комбинаций стенок внутри кубика с длиной ребра 2.

Способ оценки работы

Для генерации уникального условия и проверки результата используется следующий код на языке Python. Функция generate возвращает набор в том

числе случайным образом сгенерированных под задачу тестов и после прохода через эталонное решение, получаются ответы на сгенерированные тесты (тесты и ответы одинаковые для всех участников, результаты программы участника и ответы сравниваются при помощи функции check):

```
 1 import random
 2 from collections import deque
 3
 4
 5 def get_test_case(n, enter, exit, k, walls):
 6     test_case = "{}\n".format(n)
 7     test_case += "{} {}\n".format(enter, exit)
 8     test_case += "{}\n".format(k)
 9      test_case += '\n'.join(' '.join(str(e) for e in
edge) for edge in walls) + '\n'
10     return test_case
11
12 def add_tests(qty, size, tests):
13     n = size
14
15     graph = {}
16
17     for x in range(n * n * n):
18         graph[x] = get_near_values(x, n)
19
20      borders = [x for x in range(n * n * n) if (x //
(n * n) in [0, n - 1] or (x % (n * n)) // n in [0, n -
1] or x % n in [0, n - 1])]
21
22       near_cubes = [[x, y] for x, z in graph.items()
for y in z if (x < y)]
23
24     for test in range(qty):
25         random.shuffle(borders)
```

```
26          random.shuffle(near_cubes)
27          enter = borders[0]
28          exit = borders[1]
29                  k = int(random.uniform(0.2, 0.9) *
len(near_cubes))
30          walls = near_cubes[:k]
31            tests.append(get_test_case(n, enter, exit,
k, walls))
32
33
34
35 def generate():
36      tests = []
37      n = 2
38      enter = 0
39      exit = 7
40      k = 0
41      walls = []
42
43        tests.append(get_test_case(n, enter, exit, k,
walls))
44
45      n = 2
46      enter = 0
47      exit = 7
48      k = 1
49      walls = [[0, 1]]
50
51        tests.append(get_test_case(n, enter, exit, k,
walls))
52
53      n = 2
54      enter = 0
55      exit = 7
```

```
56      k = 2
57      walls = [[0, 1], [2, 3]]
58
59        tests.append(get_test_case(n, enter, exit, k,
walls))
60
61      n = 2
62      enter = 0
63      exit = 7
64      k = 3
65      walls = [[0, 1], [2, 3], [6, 7]]
66
67        tests.append(get_test_case(n, enter, exit, k,
walls))
68
69      n = 2
70      enter = 0
71      exit = 7
72      k = 4
73      walls = [[0, 1], [2, 3], [6, 7], [4, 5]]
74
75        tests.append(get_test_case(n, enter, exit, k,
walls))
76
77      n = 2
78      enter = 0
79      exit = 4
80      k = 5
81        walls = [[0, 4], [1, 5], [3, 7], [0, 2], [4,
6]]
82
83        tests.append(get_test_case(n, enter, exit, k,
walls))
84
```

```
85      add_tests(60, 2, tests)
86      add_tests(20, 3, tests)
87      add_tests(10, 4, tests)
88      add_tests(2, 5, tests)
89      add_tests(2, 6, tests)
90      return tests
91
92  def check(reply, clue):
93      return reply.split() == clue.split()
```

Решение:

В этой задаче надо представить каждый из внутренних кубиков как вершину графа, а возможность перехода между парами кубиков как ребра графа. Там, где появляются стены ребра отсутствуют. Далее требуется найти наименьший путь без цикла от одной вершины графа до другой. Если таких несколько, вывести наименьшую по номерам вершин последовательность.

Пример программы на языке Python:

```
 1  def shortest_path(graph, start, end):
 2      q = [start]
 3      used = [start]
 4      p = {start:-1}
 5      d = {x:0 for x in range(len(graph))}
 6      while q:
 7          v = q.pop(0)
 8          for el in graph[v]:
 9              if used.count(el) == 0:
10                  used.append(el)
11                  q.append(el)
12                  d[el] = d[v] + 1
13                  p[el] = v
14      if used.count(end) == 0:
15          return None
16
17      v = end
```

```
18      path = []
19      while v != -1:
20          path += [v]
21          v = p[v]
22      return path[::-1]
23
24  def get_near_values(coord, size):
25      z = int(coord // (size * size))
26      coord -= z * size * size
27      y = int(coord // size)
28      x = int(coord - y * size)
29      n = size
30      ans = []
31      if z + 1 < n:
32          ans += [(z + 1) * n * n + y * n + x]
33      if z > 0:
34          ans += [(z - 1) * n * n + y * n + x]
35      if y + 1 < n:
36          ans += [z * n * n + (y + 1) * n + x]
37      if y > 0:
38          ans += [z * n * n + (y - 1) * n + x]
39      if x + 1 < n:
40          ans += [z * n * n + y * n + x + 1]
41      if x > 0:
42          ans += [z * n * n + y * n + x - 1]
43      return sorted(ans)
44
45  def solve(dataset):
46      input = dataset.split()
47
48      n = int(input[0])
49      enter = int(input[1])
50      exit = int(input[2])
```

```
51      k = int(input[3])
52
53       walls = [[int(input[4 + 2 * x]), int(input[5 +
2 * x])] for x in range(k)]
54
55      graph = {}
56
57      for x in range(n * n * n):
58          graph[x] = get_near_values(x, n)
59
60      for x, y in walls:
61          graph[x].remove(y)
62          graph[y].remove(x)
63
64      path = shortest_path(graph, enter, exit)
65
66      if path == None:
67          ans = "-1\n"
68      else:
69          ans = ' '.join(str(e) for e in path) + '\n'
70
71      return ans
72
73 solve(sys.stdin.read())
```

Задача 2.5 (6 баллов)

Условие: У Вас есть робот, который умеет перемещаться между точками на плоскости. Ваша задача составить наикратчайший маршрут из стартовой точки в конечную, проходящий через несколько заданных промежуточных точек.

На вход программе подается количество точек n (3<n<12). Первая точка, промежуточные точки и конечная точка: абсцисса x и ордината y (−100.0<x,y<100.0).

Для решения задачи требуется в первой строке вывести последовательность номеров проходимых вершин, при которой суммарный путь наименьший. А

во второй строке – длину кратчайшего пути в формате числа с плавающей точкой, с точностью ровно 6 знаков после запятой. Если таких последовательностей несколько, вывести наименьшую по индексам (из маршрутов выбирается тот, у которого номер второй вершины в пути меньше. При их равенстве тот, у которого третий номер в пути меньше и так далее).

2.5.1. Формат входных данных

В первой строке натуральное число от 4 до 11 – количество точек. Далее соответствующее количество пар вещественных координат точек.

Пример ввода 1:

5

2 0

3 5

4 5

5 5

0 6

Пример ввода 2:

6

0 0

1 0

-1 0

0 1

0 -1

2 0

Пример ввода 3:

6

0 0

1 0

-1 0

0 1

0 -1

0 0

2.5.2. Формат выходных данных

Вывести n чисел – последовательность обхода точек. И вещественное число – наименьшую длину пути.

Пример вывода 1:

1 4 3 2 5

10.993230

Пример вывода 2:

1 4 3 5 2 6

6.242641

Пример вывода 3:

1 2 4 3 5 6

6.242641

Примечание:

Во втором примере есть еще путь 1 5 3 4 2 6, но он по индексам больше пути, которого в ответе. Аналогично, в последнем примере есть еще путь 1 3 4 2 5 6, который так же не подходит.

Способ оценки работы

Для генерации уникального условия и проверки результата используется следующий код на языке Python. Функция generate возвращает набор в том числе случайным образом сгенерированных под задачу тестов и после прохода через эталонное решение, получаются ответы на сгенерированные тесты (тесты и ответы одинаковые для всех участников, результаты программы участника и ответы сравниваются при помощи функции check):

```
 1 import random
 2 import math
 3 import itertools
 4
 5 def get_test_case(n, points):
 6     test_case = "{}\n".format(n)
 7      test_case += '\n'.join(' '.join(str(e) for e in
point) for point in points) + '\n'
 8     return test_case
 9
10 def add_tests(qty, size, tests):
11     n = size
```

```
12
13      for test in range(qty):
14                  points = [[random.uniform(-100, 100),
random.uniform(-100, 100)] for idx in range(n)]
15          tests.append(get_test_case(n, points))
16
17  def generate():
18      tests = []
19      n = 5
20        points = [[2, 0], [3, 5], [4, 5], [5, 5], [0,
6]]
21
22      tests.append(get_test_case(n, points))
23
24      n = 6
25        points = [[0, 0], [1, 0], [-1, 0], [0, 1], [0,
-1], [2, 0]]
26
27      tests.append(get_test_case(n, points))
28
29      n = 6
30        points = [[0, 0], [1, 0], [-1, 0], [0, 1], [0,
-1], [0, 0]]
31
32      tests.append(get_test_case(n, points))
33
34      add_tests(7, 4, tests)
35      add_tests(10, 5, tests)
36      add_tests(10, 6, tests)
37      add_tests(10, 7, tests)
38      add_tests(10, 8, tests)
39      add_tests(10, 9, tests)
40      add_tests(20, 10, tests)
41      add_tests(20, 11, tests)
```

```
42
43      return tests
44
45 def check(reply, clue):
46      return reply.split() == clue.split()
```

Решение:

Задача легко решается перебором. Всего возможных путей в случае наибольшего количества точек равно 9!=362880. Для каждого такого случая легко считаются суммы расстояний между точками и ищется минимальное.

Пример программы на языке Python:

```
 1 def get_distance(pa, pb):
 2       return math.sqrt((pa[1] - pb[1]) ** 2 + (pa[2]
- pb[2]) ** 2)
 3
 4 def get_way_size(points):
 5           return sum([get_distance(points[idx - 1],
points[idx]) for idx in range(1, len(points))])
 6
 7 def solve(dataset):
 8      input = dataset.split()
 9
10      n = int(input[0])
11
12           points = [[x, float(input[1 + 2 * x]),
float(input[2 + 2 * x])] for x in range(n)]
13
14      start = points[0]
15      finish = points[-1]
16      points = points[1:-1]
17
18            ways = [subset for subset in
itertools.permutations(points)]
19
```

```
20       lenways = [get_way_size([start] + list(way) +
[finish]) for way in ways]
21     minlen = min(lenways)
22     idx = lenways.index(minlen)
23     minway = [start] + list(ways[idx]) + [finish]
24
25      ans = ' '.join(str(p[0] + 1) for p in minway) +
'\n'
26     ans += "{:.6f}\n".format(minlen)
27     return ans
28
29 solve(sys.stdin.read())
```

2.6. Критерии определения призеров и победителей

Количество баллов, набранных при решение всех задач суммируется. Участникам второго отборочного этапа было необходимо набрать 13 баллов.