Оглавление

Профиль «Автономные транспортные системы» посвящен решению реальной задачи получения, идентификации и доставки грузов в пункт назначения разными типами транспорта с учетом их взаимодействия с объектами инфраструктуры. Участникам данного профиля предстоит решать инженерные задачи по созданию и управлению беспилотным транспортом на суше, море и в воздухе, а также обеспечивать взаимодействие со спутниками связи.

Профиль включает в себя задачи по двум школьным предметам: **физика** и **информатика**.

§1 Первый отборочный этап

Первый отборочный этап проводится индивидуально в сети интернет, работы оцениваются автоматически средствами системы онлайн-тестирования. Для каждого из уровней участия (9 класс или 10-11 класс) предлагается свой набор задач. На решение задач первого отборочного этапа участникам давалось 3 недели. Решение каждой задачи дает определенное количество баллов. Баллы зачисляются в полном объеме за правильное решение задачи. Участники получают оценку за решение задач в совокупности по всем предметам данного профиля (физика и информатика) — суммарно от 0 до 20 баллов.

1.1. Задачи по физике (9 класс)

Задача 1.1.1 (2 балла)

Навстречу друг другу по параллельным путям с одинаковой начальной скоростью едут два поезда массой M = 1500 т каждый. Каждую секунду из одного поезда в другой скидывается мешок массой 100 кг и наоборот. Считая, что мешки не кончаются, найдите, через какое время скорость поезда уменьшится в два раза. Считайте поезда достаточно длинными для осуществления данного маневра. Ответ выразите в секундах.

РЕШЕНИЕ:

Рассмотрим n-ый бросок и напишем закон сохранения импульса

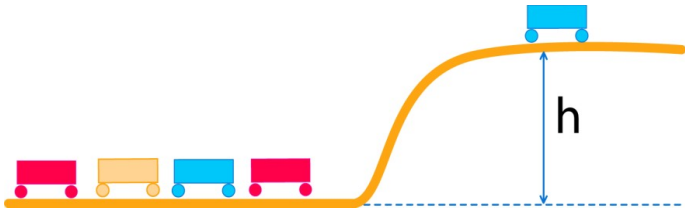
$MV_n - mV_n - mV_n = MV_{n+1}$, откуда получим:

$V_n = V_0\left(\frac{M}{M-2m}\right)^n$, откуда n = 5198.

Задача 1.1.2 (3 балла)

На дороге вплотную стоят не сцепленные вагоны, на них с горки с высоты 2 метра

съезжает такой же вагон. Найдите длину сцепленной цепочки, если минимальная относительная скорость, при которой происходит сцепка вагонов, равна 0,5 м/с. Трением пренебречь, ответ дать в единицах.



РЕШЕНИЕ:

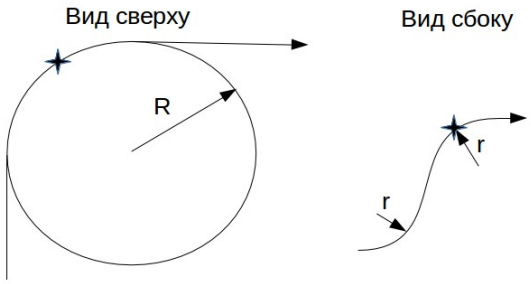
Скорость первого вагона после спуска равна $v = \sqrt{2gh}$. Рассмотрев последовательные соударения, приходим к уравнению:

$mv_1 = nmv_{cr}$,

где n — число вагонов, v_{cr} — минимальная скорость сцепки. Откуда получаем (округлив вниз) $n = 12$.

Задача 1.1.3 (3 балла)

На рисунке приведена траектория маневра уклонения истребителя (виды сверху и сбоку). Найдите ускорение в отмеченной точке. Скорость самолета 400 м/с, радиусы, указанные на рисунках $R = 400$ м, $r = 300$ м. Ответ выразите в м/с^2.



РЕШЕНИЕ:

Рассмотрим движение в двух плоскостях. Центростремительные ускорения выражаются формулами $a_1 = \frac{V^2}{R}$ и $a_2 = \frac{V^2}{r}$.

Полное ускорение $a = \sqrt{a_1^2 + a_2^2} \approx 667$м/с^2.

Задача 1.1.4 (2 балла)

В лодке два человека. Во сколько раз изменится скорость лодки, если не один, а оба

сядут на весла? Гребцов считать одинаковыми.

РЕШЕНИЕ:

Развиваемую мощность можно выразить как $N = FV \propto V^2$, т.к. сила сопротивления воды пропорциональна скорости движения лодки. Отсюда удвоение количества гребцов приведет к удвоению мощности, а скорость возрастет в $\sqrt{2}$ раз.

1.2. Задачи по физике (10-11 класс)

Задача 1.2.1 (2 балла)

В прошлом веке была предложена модель хитрого парохода, умеющего плавать против течения без затрат топлива. Для этого он бросает якорь и запасает энергию, полученную от вращения лопастей течением, затем он использует её для движения против течения. Оцените устоявшуюся на долгом промежутке времени среднюю скорость движения такого парохода (относительно берега), если известно, что коэффициент вязкого трения для лопастей $\alpha_1 = 10^4$ Н·с/м, а для парохода $\alpha_2 = 10^3$ Н·с/м, скорость течения реки $v_{теч} = 3$км/ч, а скорость движения парохода $V_{пар} = 18$ км/ч. Ответ дайте в км/ч с точностью до целых.

РЕШЕНИЕ:

Перейдем в систему отсчета, связанную с водой. В этой системе отсчета работа, совершаемая во время подзарядки и во время движения одинакова.

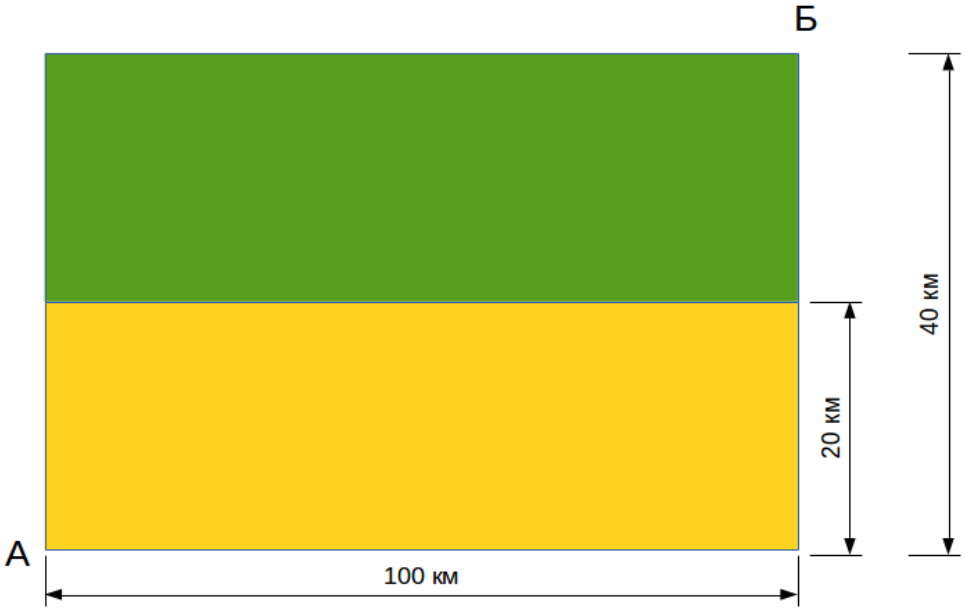
$A_1 = U^2 \alpha_1 \Delta t_1$, а $A_2 = (V+U)^2 \alpha_2 \Delta t_1$, откуда:

$\frac{\Delta t_2}{\Delta t_1} = \frac{\alpha_1 U^2}{\alpha_2 (V+U)^2}$, тогда средняя скорость относительно береза будет определяться выражением:

$V_{отн} = \frac{V\Delta t_2 + U\Delta t_1}{\Delta t_1 + \Delta t_2} - U = 3.051$м/с.

Задача 1.2.2 (3 балла)

Автомобилю требуется проехать из точки А в точку Б за минимальное время (см. рисунок). Известно, что в желтой («нижней») области максимальная скорость движения автомобиля равна 50 км/ч, а в зеленой («верхней») области 70 км/ч. Найти минимальное время, требуемое на проезд. Запишите ответ в часах с точностью до десятых.



РЕШЕНИЕ:

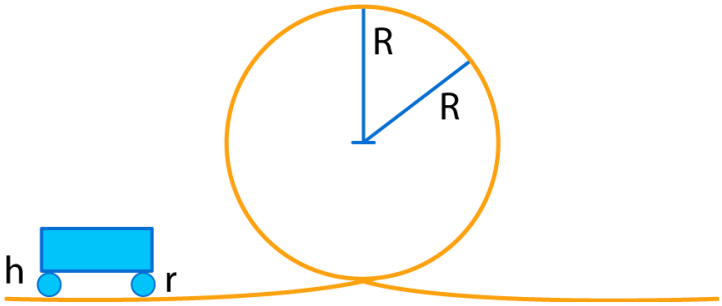
Задача во многом аналогична прохождению света через 2 среды. Если расстояние от левого края границы до точки пересечения траектории с границей x км, то минимальное время определяется:

$$t_{\min} = \min_{x}\left(\frac{\sqrt{20^2 + x^2}}{50} + \frac{\sqrt{20^2 + (100 - x)^2}}{70}\right)$$

Путем несложных вычислений, получаем t_{min} = 1.744 ч.

Задача 1.2.3 (2 балла)

Найдите скорость, с которой должна ехать тележка с высотой центра масс h = 20 см с радиусом колес r = 15 см в верхней точке, чтобы проехать «мертвую петлю» радиусом R = 5 м. Расстояние между центрами колес 0.5 м. Ускорение свободного падения считайте равным 9.8 м/с^2. Запишите ответ в м/с с точностью до сотых.



РЕШЕНИЕ:

Несмотря на конечный размер, будем считать, что масса тележки сосредоточена в центре масс, т.е. пренебрежем энергией вращательного движения тележки вокруг

своей оси. Таким образом задача сводится к классической задаче про мертвую петлю с эффективным радиусом, равным расстоянию от центра петли до центра масс.

$R_{eff} = R - R + \sqrt{(R-r)^2 - \left(\frac{d}{2}\right)^2} - h + r$, тогда скорость в верхней точке:

$$V = \sqrt{gR_{eff}} \approx 6.854\text{м/с}$$

Задача 1.2.4 (3 балла)

С какой частотой стоит толкать из ямы, являющейся частью сферы радиуса r = 20 м, машину с высотой центра масс 0,75 м? Ускорение свободного падения 10 м/с^2.

Ответ дайте с точностью до сотых.

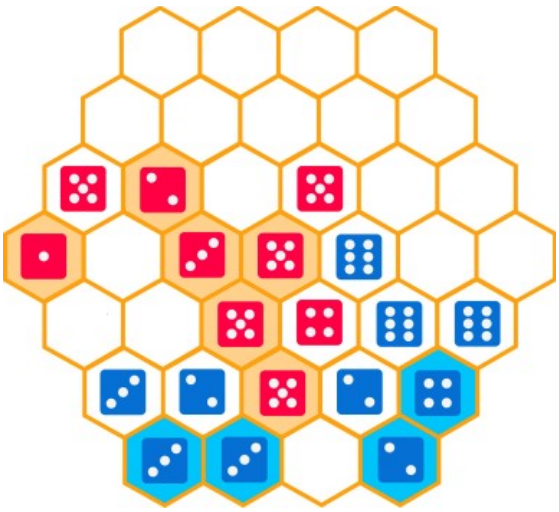
РЕШЕНИЕ:

Для того, чтобы вытащить машину из ямы нужно сделать так, чтобы частота вынуждающей силы оказалась близка к собственной частоте колебаний такой системы. Считая машину достаточно небольшой по сравнению с размерами ямы, т.е рассматривая движения центра масс получим по аналогии с движением математического маятника:

$$\nu = \frac{1}{2\pi}\sqrt{\frac{g}{R-r}} \approx 0.115\text{Гц}$$

1.3. Задачи по информатике (9 класс и 10-11 класс)

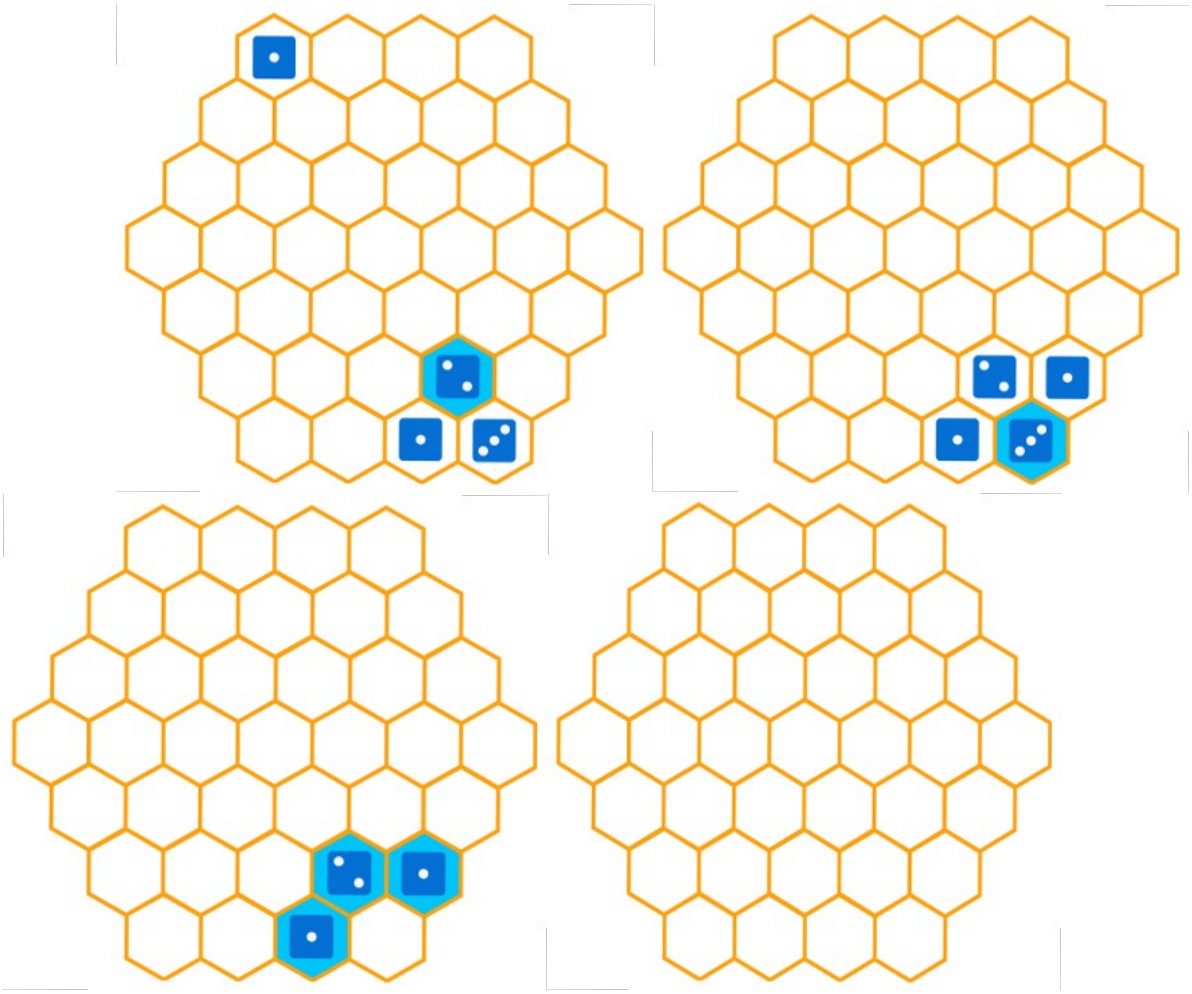
Задача 1.3.1 «Сибирские кости» (1 балл)



Сибирские кости — абстрактная симметричная настольная игра для двух игроков.

Игра ведется на гексагональном поле со стороной четыре клетки обычными игральными костями двух цветов. Кости, у которых количество соседей равно числу на верхней грани, называются окруженными. Цель обоих игроков — получить шесть окруженных костей своего цвета. Выше изображена позиция, в которой выиграл красный игрок, окружённые кости выделены фоном.

Но мы будем рассматривать упрощенную версию этой игры, а именно вариант для одного игрока. В нем необходимо переместить одну кость так, чтобы убрать с поля наибольшее число костей. После хода одновременно убираются все окруженные кости. Если после этого появились новые окруженные кости, то они так же убираются. Этот процесс длится до тех пор, пока на поле остаются окруженные кости. Ниже изображен итог перемещения единицы из левого верхнего угла к группе остальных кубиков.



Формат входных данных:

В каждой строке входных данных задаётся одна строка поля. В первой строке даётся четыре разделённых пробелом неотрицательных числа, не превышающих 6 — начальное состояние первой строки поля. Ноль означает отсутствие кубика, ненулевое число —

значение на кубике. Во второй строке даётся пять чисел — состояние второй строки в аналогичном формате. В последующих строках описывается оставшаяся часть поля.

Пример ввода:

```
1 0 0 0
0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 2 0
0 0 1 3
```

Формат выходных данных:

Нужно вывести единственное неотрицательное целое число - количество кубиков, которое можно снять с поля.

Пример вывода:

```
4
```

РЕШЕНИЕ:

Для решения задачи достаточно аккуратно реализовать написанный в условии задачи алгоритм. Пример программы, реализующей данный алгоритм на языке Python:

```
1. import copy
2. import sys
3.
4. def neighbours(a, x, y):
5.     res = 0
6.     for dx, dy in [(-1, -1),(-1, 0),(0, 1),(1, 1),(1, 0),(0, -1)]:
7.         xx = x + dx
8.         yy = y + dy
9.         res +=  int(-1<xx<len(a) and -1<yy<len(a[xx]) and a[xx][yy]>0)
10.     return res
11.
12. def clear(a):
13.     b = []
14.     res = 0
15.     for x in range(len(a)):
16.         b.append([-1] * len(a[x]))
17.         for y in range(len(a[x])):
18.             if a[x][y] == 0:
19.                 b[x][y] = 0
```

```
                continue
            if a[x][y] != neighbours(a, x, y):
                b[x][y] = a[x][y]
            else:
                b[x][y] = 0
                res += 1
    return (res, b)


def test(a, x1, y1, x2, y2):
    a = copy.deepcopy(a)
    a[x2][y2] = a[x1][y1]
    a[x1][y1] = 0
    res = 0
    while True:
        q, a = clear(a)
        if q == 0:
            return res
        res += q

def split(s):
    return list(map(int, s.strip().split()))

def solve(dataset):
    lines = dataset.strip().split('\n')

    a = []
    a.append(split(lines[0]))
    a.append(split(lines[1]))
    a.append(split(lines[2]))
    a.append(split(lines[3]))
    a.append([-1] + split(lines[4]))
    a.append([-1, -1] + split(lines[5]))
    a.append([-1, -1, -1] + split(lines[6]))

    ans = 0
    for x in range(len(a)):
        for y in range(len(a[x])):
            if a[x][y] <= 0:
                continue
            for xx in range(len(a)):
```

```
61.                 for yy in range(len(a[xx])):
62.                     if a[xx][yy] != 0:
63.                         continue
64.                     t = test(a, x, y, xx, yy)
65.                     if t > ans:
66.                         ans = t
67.     return str(ans)
68.
69. solve(sys.stdin.read())
```

Задача 1.3.2 «Корни» (3 балла)

У мальчика Пети есть число *N*. Но оно ему не нужно, в отличие от числа *X*. Чтобы его получить Петя может брать целочисленный корень, умножать и складывать числа. Целочисленным корнем степени *k* из натурального числа *n* будем называть наибольшее натуральное число, для которого выполняется соотношение: $x^k \leq n$. Например, целочисленным корнем пятой степени из тысячи будет тройка, так как $3^5 = 243 \leq 1000$, а $4^5 = 1024 > 1000$. Обозначим это как $\sqrt[5]{1000} = 3$. Также будем считать, что степенью корня могут быть только натуральные числа.

Для Пети взятие целочисленного корня — тяжёлая задача, и он хочет минимизировать суммарную степень корней, которые встречаются в формуле получения *X*.

А ещё у Пети есть старший брат. Которого зовут Дима. Этот Дима решил добавить интереса задачке Пети и дать такие ограничения:

1. Пете нельзя брать корни от чисел, отличных от *N*.
2. Можно перемножать только те числа, которые Петя получил с помощью операции взятия корня или умножения других чисел.
3. Складывать можно числа, которые получены как результат умножения, взятия корня или суммы других чисел.

Помогите Пете написать выражение, которое будет легко считаться и подходит под ограничения, наложенные Димой. Найдите минимальную сложность искомого выражения.

Формат входных данных:

В единственной строке даны два целых числа N и X ($1 \leq X, N \leq 1000$).

Пример ввода:

```
100 126
```

Формат выходных данных:

Выведите единственное натуральное число — ответ на задачу.

Пример вывода:

```
9
```

Пояснение к примеру: $126 = 100 + 10 + 4 \cdot 4 = \sqrt[1]{100} + \sqrt[2]{100} + \sqrt[3]{100} \cdot \sqrt[3]{100}$

РЕШЕНИЕ:

Решение состоит из трёх этапов. На первом этапе нужно составить набор степеней и корней из N, которые может быть выгодно использовать. Если два корня с разной степенью равны, то нам выгодно использовать тот из них, степень которого меньше. Так как $210 > 1000$, то для любого N будет не более 11 различных корней. На втором этапе методом динамического программирования получаем список чисел, которые можно получить перемножением корней с оптимальной суммарной степенью. На третьем этапе используя тот же метод получается список чисел, которые можно получить сложением чисел с предыдущего этапа с оптимальными суммами.

Пример программы, реализующей данный алгоритм на языке Python:

```
1. import sys
2.
3. INF = int(1e9)
4.
5. def getRoots(n, mx):
6.     ans = [INF, n]
7.     for x in range(n, 1, -1):
8.         while x ** len(ans) <= n:
9.             ans.append(x)
10.     ans.append(1)
11.     roots = dict()
12.     for i in range(len(ans)):
13.         if ans[i] != ans[i - 1] and ans[i] <= mx:
14.             roots[ans[i]] = i
15.     return roots
16.
17. def getProducts(roots, mx):
18.     ans = dict()
19.     ans[1] = 0
20.     for i in range(2, mx + 1):
21.         ans[i] = INF
22.         for k, v in roots.items():
23.             if i % k == 0:
24.                 d = i // k
25.                 if d in ans and ans[d] + v < ans[i]:
```

```
                    ans[i] = ans[d] + v
        if ans[i] == INF:
            ans.pop(i, None)
    ans[1] = roots[1]
    prods = [(k, v) for k, v in sorted(ans.items())]
    return prods

def getSums(prods, mx):
    ans = [INF] * (mx + 1)
    ans[0] = 0
    for i in range(len(ans)):
        for k, v in prods:
            if k > i:
                break
            if ans[i - k] + v < ans[i]:
                ans[i] = ans[i - k] + v
    ans[0] = INF
    return ans

def solve(dataset):
    n, x = list(map(int, dataset.strip().split()))
    roots = getRoots(n, x);
    prods = getProducts(roots, x)
    ans = getSums(prods, x)
    return str(ans[x])

solve(sys.stdin.read())
```

Задача 1.3.3 «Ломаная» (3 балла)

На доске была нарисована замкнутая ломаная, причём каждая пара соседних звеньев образовывала прямой угол, и все звенья были параллельны одной из сторон доски. Любая пара звеньев имела не более одной общей точки, а также никакие две вершины не совпадали. Координаты вершин ломаной записали в случайном порядке, а ломаную стёрли. Ваша задача — восстановить саму ломаную.

Формат входных данных:

В первой строке дано чётное натуральное число N — количество вершин ломаной ($4 \le N \le 100000$). В последующих N строках даны пары целых чисел, разделённых ровно одним пробелом — координаты вершин. Все координаты не превышают 1000000 по абсолютному значению.

Пример ввода:

```
4
1 0
1 1
0 0
0 1
```

Формат выходных данных:

Выведите номера вершин в том порядке, в котором их нужно соединить, чтобы получить первоначальную ломаную. Нумерация вершин с нуля, в том порядке, в котором они даны во входных данных. Если возможно несколько ответов, выведите любой.

Пример вывода:

```
0 2 3 1
```

РЕШЕНИЕ:

Можно показать, что на каждой вертикальной и горизонтальной прямой будет находиться чётное число точек из входных данных. Понятно, что нижняя (левая) точка обязана соединяться со следующей за ней.

Пример программы, реализующей данный алгоритм на языке Python:

```
1. import ast
2. import sys
3.
4. class Point:
5.     def __init__(self, x = 0, y = 0):
6.         self.x = x
7.         self.y = y
8.     def turn(self, p):
9.         return Point(self.smul(p), self.smul(p.norm()))
10.     def smul(self, p):
11.         return self.x * p.x + self.y * p.y
12.     def norm(self):
13.         return Point(self.y, -self.x)
14.     def __lt__(self, p):
15.         if self.x != p.x:
16.             return self.x < p.x
17.         return self.y <= p.y
18.     def area(self, p):
19.         return (self.x - p.x) * (self.y + p.y)
20.     def vec(self, p):
```

```
21.         return Point(self.x - p.x, self.y - p.y)
22.
23. def solveForDirection(points):
24.     n = len(points)
25.     perm = [i for i in range(n)]
26.     perm.sort(key=lambda k: points[k])
27.     neighbours = [None] * n
28.     for i in range(1, n):
29.         if points[perm[i]].x == points[perm[i - 1]].x and neighbours[perm[i
- 1]] is None:
30.             neighbours[perm[i - 1]] = perm[i]
31.             neighbours[perm[i]] = perm[i - 1]
32.     for p in neighbours:
33.         if p is None:
34.             return None
35.     return neighbours
36.
37. def getNeighbours(points):
38.     neighbours = [[], []]
39.     neighbours[0] = solveForDirection(points)
40.     points = [p.norm() for p in points]
41.     neighbours[1] = solveForDirection(points)
42.     if neighbours[0] is None or neighbours[1] is None:
43.         return None
44.     return list(zip(neighbours[0], neighbours[1]))
45.
46. def getPolygon(neighbours, first):
47.     ans = [first, neighbours[first][1]]
48.     while ans[-1] != first:
49.         x = ans[-2]
50.         y = ans[-1]
51.         if neighbours[y][0] == x:
52.             ans.append(neighbours[y][1])
53.         else:
54.             ans.append(neighbours[y][0])
55.     return ans[:-1]
56.
57. def solveForLine(points):
58.     neighbours = getNeighbours(points)
59.     if neighbours is None:
60.         return None
```

```
61.     return getPolygon(neighbours, 0)
62.
63. def solve(dataset):
64.     lines = dataset.strip().split('\n')
65.     n = int(lines[0])
66.     lines = lines[1:]
67.     points = []
68.     for line in lines:
69.         x, y = list(map(int, line.strip().split()))
70.         points.append(Point(x, y))
71.     perm = solveForLine(points)
72.     return ' '.join(map(str, perm))
73.
74. solve(sys.stdin.read())
```

Задача 1.3.4 «Перевозка груза» (3 балла)

Полигон N на M метров разбит на клетки со стороной равной одному метру. Для каждой клетки полигона нам известна средняя высота, выраженная в сантиметрах. В центре одной из клеток стоит робот, который может двигаться только из центра одной клетки в центр другой, и только в том случае, если эти клетки имеют общую сторону. Ещё одно ограничение, которое наложено конструкцией робота — невозможность перемещаться между клетками, если их средние высоты отличаются более, чем на 100 сантиметров.

Также нам интересны на этом полигоне две другие клетки — в одной из них лежит груз, в другую должен прибыть робот после того, как заберёт груз. Для того чтобы забрать груз, роботу нужно просто заехать в клетку, в которой он находится.

Какой минимальный путь должен пройти робот, чтобы попасть из начальной клетки в конечную, забрав перед этим груз? Груз не влияет на движение робота. Гарантируется, что искомый путь существует.

Формат входных данных:

В первой строке даны два натуральных числа N и M, разделённых пробелом — размеры полигона ($1 \le N, M \le 300$). В следующих N строках даны по M целых неотрицательных чисел, разделённых пробелами, — средние высоты ячеек, выраженные в сантиметрах. Высоты ячеек не превышают 10 000 000.

В следующей строке дано начальное положение робота — два целых числа, разделённые пробелом. Первое число задаёт строку, второе — номер клетки в строке. Строки нумеруются начиная с 00 и заканчивая $N-1$, а клетки в строках — от 00 до $M-1$. В двух

следующих строках в аналогичном формате описываются положение груза и конечная клетка.

Пример ввода:

```
2 3
0 1000 0
0 0 0
0 0
0 2
1 1
```

Формат выходных данных:

Выведите единственное целое число — расстояние в метрах, которое необходимо проехать роботу, чтобы доставить груз в конечную клетку.

Пример вывода:

```
6
```

РЕШЕНИЕ:

В задаче нужно реализовать любой алгоритм нахождения кратчайшего пути в невзвешенном графе. Например, поиском в ширину.

Пример программы, реализующей данный алгоритм на языке Python:

```
1. import sys
2. from collections import deque
3.
4. MAXD = 100;
5.
6. def getLine(s):
7.     return list(map(int, s.strip().split()))
8.
9. def inMap(x, y):
10.     return -1 < x < n and -1 < y < m
11.
12. def correctMove(x, y, xx, yy):
13.     return inMap(xx, yy) and landscape[xx]yy] >=0 and
abs(landscape[xx][yy] - landscape[x][y]) <= MAXD
14.
15. def bfs(x, y, landscape):
16.    dist = [[None] * len(landscape[0]) for i in range(len(landscape))]
17.     dist[x][y] = 0
18.     q = deque()
```

```
19.     q.append((x, y))
20.     while q:
21.         x, y = q.popleft()
22.         for dx, dy in [(1, 0), (0, 1), (-1, 0), (0, -1)]:
23.             xx = x + dx
24.             yy = y + dy
25.             if correctMove(x, y, xx, yy) and dist[xx][yy] is None:
26.                 dist[xx][yy] = dist[x][y] + 1
27.                 q.append((xx, yy))
28.     return dist
29.
30. def solve(dataset):
31.     lines = dataset.strip().split('\n')
32.     global n, m, landscape
33.
34.     n, m = getLine(lines[0])
35.     lines = lines[1:]
36.     landscape = []
37.     for i in range(n):
38.         landscape.append(getLine(lines[i]))
39.     lines = lines[n:]
40.
41.     x0, y0 = getLine(lines[0]) # car
42.     x1, y1 = getLine(lines[1]) # load
43.     x2, y2 = getLine(lines[2]) # target
44.
45.     dist = bfs(x1, y1, landscape)
46.     return str(dist[x0][y0] + dist[x2][y2])
47.
48. solve(sys.stdin.read())
```

1.4. Критерии определения победителей и призеров

Количество баллов, набранных при решение всех задач, суммируется. Призерам первого отборочного этапа было необходимо набрать 2 балла (для 9 класса) и 5 баллов (для 10-11 класса). Победители первого отборочного этапа должны были набрать 14 баллов.