Олимпиада школьников «Гранит науки» Шифр работы Ч01-02
БЛАНК ОЛИМПИАДНОЙ РАБОТЫ (не заполнять)

Вариант № 4

Оценка выполнения олимпиадной работы
(заполняется проверяющим)

№ задания		1	2	3	4	5	6	7	8	9	10	Σ
Полученный балл		—	7	—	—	20	3					30
I проверка	Подпись проверяющего				Σ баллов прописью	девяносто						
	Подпись проверяющего				Σ баллов прописью	девятнадцать						
II проверка	Подпись проверяющего				Σ баллов прописью	тридцать						
	Подпись проверяющего				Σ баллов прописью	тридцать						

Задача №2

Решение:

Учитывая данные в условии исходную и зашифрованные строки, можно составить таблицу соответствия каждого символа текста его зашифрованному значению в зависимости от значения вводимого числа:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35
n	а	б	в	г	д	е	ё	ж	з	и	й	к	л	м	н	о	п	р	с	т	у	ф	х	ц	ч	ш	щ	ъ	ы	ь	э	ю	я	,	.	?
1	е	ё	ж	з	и	й	к	л	м	н	о	п	р	с	т	у	ф	х	ц	ч	ш	щ	ъ	ы	ь	э	ю	я	,	.	?	а	б	в	г	а
-1	,	.	?	а	б	в	г	а	е	ё	ж	з	и	й	к	л	м	н	о	п	р	с	т	у	ф	х	ц	ч	ш	щ	ъ	ы	ь	э	ю	я

В данной таблице n - вводимое число (в первом случае - 1, во втором - -1), алфавит - возможные значения символов текста (кроме пробела,

Вариант № 4

так как его значение не менялось), индексы - номера символов алфавита.
В 3 и 4 строчках расположены принимающие символами алфавита значение в шифре в зависимости от n.

По этой таблице можно заметить, что шифром является шифр Цезаря со сдвигом на 5 для $n=1$ и на -3 для $n=-1$. То есть символ с номером X до шифрования станет символом с номером $X+5$ для $n=1$ и $X-3$ для $n=-1$. Также стоит помнить, что индексы закольцованы, то есть после номера 36 следует 1.

Тогда, учитывая условие, можно составить функцию шифрования (индекса) символа X в зависимости от n:

$F(X, n) = X + 1 + 4 \cdot n$, где n - вводимое число, а X - индекс шифруемого символа.

Например, при $n=1$, $F(X,1) = X+5$, то буква „П", имеющая номер 16, зашифруется в букву „Ф" с номером 21 ($F(16,1) = 16+5 = 21$), а знак „)" с номером 33 в букву „В", имеющую индекс 2 ($f(33,1) = 33+5 = 38$, учитывая закольцованность $f(33,1) = 2$).

Вариант № 4

Более точно функцию можно записать с помощью оператора деления:

$F(x,n) = (x + 1 + 4n) \% 36$, где % - операция взятия остатка при делении на 36.

Ответ: правило шифрование символа с индексом x и введённом числе n: $F(x,n) = (x+1+4 \cdot n) \% 36$;
таблица соответствия в решении, в ней первая строка - индексы символов, вторая - исходные символы, а третья и четвёртая строки - принимые исходными символами значение для введенных значений чисел 1 и -1 соответственно.

Задание 5.

Решение:

Для нахождение минимального количества клеток, которые необходимо преобразовать в сушу, чтобы соединить острова, можно воспользоваться следующей идеей:
Переберем для каждой клетки одного острова все клетки другого и для каждой пары найдем минимальное расстояние между клетками.

Вариант № 4

Таким образом, ответом будет минимальное такое расстояние. Понятно, что оптимальным ответом будет дорожка шириной в одну клетку от одного острова до другого. Тогда, перебрав всевозможные пары клеток двух островов ~~программа~~ можно между которыми найдём две такие, длина дорожки будет наименьшей.

Так как поле состоит из квадратов (клетка - квадрат), то расстояние между двумя клетками - Манхэттенское расстояние, равное ~~сумме~~ сумме разниц по двум координатам этих клеток. То есть если ввести Ox и Oy параллельно сторонам карты, то кол-во клеток между клеткой $A(x_1, y_1)$ и $B(x_2, y_2)$ равно $|x_1 - x_2| + |y_1 - y_2| - 1$.

Этот алгоритм будет работать за $O(A \cdot B + N \cdot M)$, где A и B - количество клеток, принадлежащих двум островам соответственно, а N и M - размеры карты.

Вариант № 4

Код программы на языке Python 3:

```
def get_dist(a, b): # ф-я расстояния между 2 клетками
    return abs(a[0]-b[0]) + abs(a[1]-b[1]) - 1

def main():
    n, m = map(int, input().split()) # размеры карты
    a_isl = list()
    b_isl = list()
    pole = [[0 for _ in range(m+2)] for _ in range(n+2)]
    for i in range(n):
        for j, sym in enumerate(input()):
            if sym == '#':
                if not a_isl and not b_isl:
                    a_isl.append((i,j))
                    # если sym - первый символ острова
                    pole[i+2][j+1] = 1
                    pole[i][j+1] = 1
                    pole[i+1][j+2] = 1
                    pole[i+1][j] = 1
                    pole[i+1][j+1] = 1
                elif pole[i+1][j+1]:
                    a_isl.append((i,j))
                else:
                    b_isl.append((i,j))
# Теперь в a_isl и b_isl лежат пары-координаты
# клеток соответствующих островов
```

Вариант № 4

```
if not a_isl or not b_isl:
    # Если какого-то острова нет, т.е. все
    # земли принадлежат одному, то ответ 0.
    print(0)
    return
ans = 1000000000 # здесь храним минимальный ответ
for a_cell in a_isl:
    for b_cell in b_isl:
        ans = min(ans, get_dist(a_cell, b_cell))
        # перебираем каждую пару клеток
        # двух островов и находим
        # минимальную длину дороги
print(ans)
```

20 б

```
main()
```

```
* pole[i+2][j+1] = 1
  pole[i][j+1] = 1
  pole[i+1][j+2] = 1
  pole[i+1][j] = 1
  pole[i+1][j+1] = 1
```

10 б

Битовая матрица карты pole нужна, чтобы быстро определять, какому из двух островов принадлежит клетка во время ввода.

Вариант № 4

Задание 6.

Решение:

3б

Для нахождения максимальной площади, которую робот может очистить, можно построить граф, основываясь на входных данных и запустить из каждой клетки поиск в глубину (DFS). Таким образом, алгоритм переберёт всевозможные пути следования для данной ~~комнаты~~ квартиры и для каждого найдёт максимальную площадь.

Чтобы построить граф, достаточно рас~~смотреть~~ на каждую клетку (свободную) и на её соседей. Для данной клетки всегда есть ребро в соседнюю свободную клетку (при правильном направлении движения). В случае с соседней клеткой - препятствием необходимо добавить ~~ребро~~ петлю (ребро из данной клетки в неё же) со своей направлением движение робота. Так у каждой свободной клетки будет по 4 ребра: со своей направлением или нет.

Вариант № 4

Теперь достаточно запустить из каждой клетки (вершины) запустить поиск в глубину DFS и пересчитывать каждый запуск ответ.

Этот алгоритм работает за $O(NM+4(A+4A))$ - $O(NM+20A)$, где N и M - размеры квартиры, а A - кол-во свободных клеток, соответственно кол-во вершин в графе равно A, а рёбер - $4A$.