Звіт

У програмі реалізований пошук найкоротшого шляху через алгоритм Левіта.

Алгоритм:

1)Створити масив (у нашому випадку список)для збереження шляхів до точок;

2) На початку ініціалізувати його зі значеннями infinity(у програмі таке значення позначає None), а відстань до початкової точки позначити 0.

3)Створити масив(список) ‘предків’ точок у найкоротших шляхах до них, з початковими значеннями відповідними точками для кожної позиції.

4)Створити множини для точок до яких обраховується відстань(нехай тут м1(у оригінальному алгоритмі м1 – черга, у програмі список)), до яких ще не була обрахована відстань(нехай тут м2) і для яких уже виконані обрахунки(нехай тут м0)

(У програмі є ще 2 кроки, конвертації заданого ландшафту зі списку у numpy.array і отримання його розмірностей. Також створюється допоміжна множина для ребер, довжина яких обрахована.)

5)Ітерувати через множину м1, для кожного елементу, переміщувати його в м0, знаходити суміжні точки, ітерувати через них і виконувати наступні дії:

А)Обраховувати відстань ребра з цією суміжною точкою;

Б)Якщо ця суміжна точка у множині м2,

то присвоїти значення знайденої відстані, якщо відстань до точки з м1 None або значення суми відстані до цієї точки і довжини ребра між даними точками. Видалити суміжну точку з множини м2 і перемістити у м1. Повторити для всіх суміжних точок.

В)Якщо точка в м1, оновити її значення мінімальним значенням серед старої обрахованої відстані і нової.

Г)Якщо точка у м0, оновити її значення мінімальним значенням серед старої обрахованої відстані і нової.

(Якщо значення предка для точки з м1, яку ми розглядаємо, співпадає зі значенням суміжної точки, ребро з якою ми обраховуємо, існує варіант розвитку подій при якому пройти з такої суміжної точки туди назад до точки з м1 швидше ніж зі ‘старого предка’. Тоді програма шлях від предка зациклюється і це унеможливлює пошук необхідного шляху. Для цього у програмі передбачена додаткова умова.)

6)Результат конвертується у одинарні індекси і повертається як список.

У програмі додатково реалізовані функції для визначення мінімальної відстані, пошуку суміжних точок і визначення відстані:

```
    def compute_distance(height1, height2, step):
        '''
        Returns a distance between points
using their heights and given step.

        >>> round(compute_distance(50, 60, 2))
        10
        >>> round(compute_distance(100, 200, 5))
        100
        '''

        #Finding distance between points using heights and step.
        distance = sqrt((height1 - height2)**2 + step**2)
        return distance

    def get_minimum(distance1, distance2, ancestors, element, end):
        '''
        Returns a minimum distance.

        >>> ancestors = [[(2,4), (2,5), (3,7), (4,5)]]
        >>> get_minimum(50, 60, ancestors, (2,2), (0,2))
        60
        >>> print(ancestors)
        [[(2, 4), (2, 5), (2, 2), (4, 5)]]
        >>> ancestors = [[(2,4), (2,5), (3,7), (4,5)]]
        >>> get_minimum(100, 60, ancestors, (2,2), (0,2))
        100
        >>> print(ancestors)
        [[(2, 4), (2, 5), (3, 7), (4, 5)]]
        '''

        #Compares data of distances and returns minimal existing.
        if distance1 is None:
            return distance2
        if distance1 > distance2:
            return distance1
        ancestors[end[0]][end[1]] = element
        return distance2
```

```
    def find_ribs(point, size):
        '''
        Returns a list
of the ends of the ribs formed
with point as a start

        >>> list(sorted(list(find_ribs((2,4), (10,10)
        [(1, 4), (2, 3), (2, 5), (3, 4)]
        >>> list(sorted(list(find_ribs((2,4), (2,5)))
        [(1, 4), (2, 3), (3, 4)]
        '''

        #Finding sumizhni rebra of given point.
        ribs = set()
        index_x = point[0]
        index_y = point[1]

        #Finding rebra if the point is on the edge.
        if index_x == size[0] - 1:
            if index_y == 0:
                ribs.add((index_x - 1, index_y))
                ribs.add((index_x, index_y + 1))
            elif index_y == size[1] - 1:
                ribs.add((index_x - 1, index_y))
                ribs.add((index_x, index_y - 1))
            else:
                ribs.add((index_x - 1, index_y))
                ribs.add((index_x, index_y + 1))
                ribs.add((index_x, index_y - 1))
            return ribs
```

```
        if index_x == 0:
            if index_y == 0:
                ribs.add((index_x + 1, index_y))
                ribs.add((index_x, index_y + 1))
            elif index_y == size[1] - 1:
                ribs.add((index_x + 1, index_y))
                ribs.add((index_x, index_y - 1))
            else:
                ribs.add((index_x + 1, index_y))
                ribs.add((index_x, index_y + 1))
                ribs.add((index_x, index_y - 1))
            return ribs

        #Finding rebra if the point is inside the grid.
        if index_y == 0:
            ribs.add((index_x + 1, index_y))
            ribs.add((index_x - 1, index_y))
            ribs.add((index_x, index_y + 1))
        elif index_y == size[1] - 1:
            ribs.add((index_x + 1, index_y))
            ribs.add((index_x - 1, index_y))
            ribs.add((index_x, index_y - 1))
        else:
            ribs.add((index_x + 1, index_y))
            ribs.add((index_x - 1, index_y))
            ribs.add((index_x, index_y - 1))
            ribs.add((index_x, index_y + 1))
        return ribs
```

Виконали:

Броницький Михайло
Роман Кипибіда
Трескот Вадим
Ростик Сидор
Гоєв Олексій