14. Алгоритмы

Вопрос 427. Даны переменные a и b. Как сделать так, чтобы переменная a ссылалась на переменную b, а переменная b на переменную a?

Livecoding: Переменная a ссылается на объект “мама”, переменная b ссылается на объект “мыла”, переменная c ссылается на объект “раму”. Сделать так, чтобы переменная a ссылалась на переменную c, а переменная c на переменную a. Вывести print(f’{a} {b} {c})

Ответ:

В Python не нужно вводить третью переменную для решения этого алгоритма ввиду особенностей языка. Алгоритм реализуется так:

```
a = [76, 81, 9, 83, 35, 8]
b = [-3, -6, -7, 0, -1, 9]
a, b = b, a
print(f"a={a}, b={b}")
```

Результат: a=[-3, -6, -7, 0, -1, 9], b=[76, 81, 9, 83, 35, 8]

———

Вопрос 428. Напишите алгоритм для перевода любого десятичного числа в любую систему счисления, реализованный с помощью схемы Горнера?

Livecoding: перевести число 562715273651 в двоичную систему счисления и сравнить полученный результат с функцией Python – bin()

Ответ:

```
base = 2  # Система счисления
x = int(input())
result = []
while x > 0:
    digit = x % base  # Добываем последнюю цифру
    result.append(digit)
    x //= base  # Удаляем последнюю цифру в числе

result.reverse()
print(result)
```

Задача: ввод 562715273651 вывод [1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 1]

———

Вопрос 429. Напишите алгоритм, который определяет является ли число простым (простое – это число, которое делится только на 1 и на само себя без остатка, на все остальные числа делится с остатком)

Livecoding: даны шесть чисел: число 160, число 161, число 162, число 163, число 164, число 165. Определить какое из этих чисел является простым:

Ответ:

Алгоритм реализуется так:

```
def is_simple(number):
    divisor = 2
    while divisor < number:
        if number % divisor == 0:
            return False
        divisor += 1
    return True

print([f'{n} - {is_simple(n)}' for n in range(160, 166)])
```

Ответ:

['160 - False', '161 - False', '162 - False', '163 - True', '164 - False', '165 - False']

163 является простым числом, все остальные – не являются простыми числами

———

Вопрос 430. Напишите алгоритм разбиения числа на множители?

Livecoding: разбейте число 8721 на множители

Ответ:

```
def factorize_number(x):
    divisor = 2
    while x > 1:
        if x % divisor == 0:
            print(divisor)
            x //= divisor
        else:
            divisor += 1
```

Задача: ввод factorize_number(8721) Вывод 3 3 3 17 19

———

Вопрос 431. Напишите алгоритм линейного поиска в массиве. То есть дан массив чисел и необходимо найти индекс этого числа в массиве. Функцию index() не использовать
Ответ:

```
def array_search(A: list, N: int, x: int):
    """Осуществляет поиск числа x в массиве A  от 0 до N-1
индекса включительно. Возвращает индекс элемента x в массиве A.
    Или -1, если такого нет.
    Если в массиве несколько одинаковых
    элементов, равных x, то вернуть индекс первого по счету.
    """

    for k in range(N):
        if A[k] == x:
            return k
    return -1
```

———

Вопрос 432. Напишите алгоритм инверсии массива. То есть дан массив чисел, надо вывести тот же самый массив, только прочитанный справа налево. Функцию reverse() не использовать
Ответ:

```
def invert_array(A: list, N: int):
    """Обращение массива
    (поворот задом-наперёд)
    в рамках индексов от 0 до N-1.
    """

    for k in range(N // 2):
        A[k], A[N - 1 - k] = A[N - 1 - k], A[k]
    return A
```

———

Вопрос 433. Напишите алгоритм для циклического сдвига в массиве?
Ответ:
Бывает циклический сдвиг влево и вправо.
Очень легко реализуется на классе deque в библиотеки collections, установив maxlen и использую FIFO.
Без применения двусторонней очереди, в Python реализуется так:

```
def cycle_offset(A, N, direction='left'):
    if direction == 'left':
        tmp = A[0]
        for k in range(N-1):
            A[k] = A[k+1]
        A[N-1] = tmp
        return A
    elif direction == 'right':
        tmp = A[N-1]
        for k in range(N-2, -1, -1):
            A[k+1] = A[k]
        A[0] = tmp
        return A
    else:
        return 'ошибка'
```

———

Вопрос 434. Опишите и реализуйте алгоритм сортировки вставками? Какова сложность алгоритма?
Ответ: Сложность алгоритма - O(n^2)
Мы берем элемент 2 и начинаем его сравнивать с элементом 4, потом с элементом 3, а потом с элементом 1. 2 находится между 1 и 3 - туда его вставляем, а 3 и 4 подвигаем на одну клетку вправо.

```
def insert_sort(A):
    """Сортировка вставками."""
    N = len(A)
    for top in range(1, N):
        k = top
        while k > 0 and A[k - 1] > A[k]:
            A[k], A[k - 1] = A[k-1], A[k]
            k -= 1
```

———

Вопрос 435. Опишите и реализуйте алгоритм сортировки выбором? Какова сложность алгоритма?

Ответ:
Сложность алгоритма - O(n^2)
На итерации мы элемент текущей итерации сравниваем со следующим элементом в массиве - если он меньше, то мы их меняем местами, если нет - то идем дальше сравнивать.
Самое важно, чтобы не проходить по уже пройденным итерациям. Последнего сортировать не надо - он последний автоматически сортируется.

```
def selection_sort(A):
    """Сортировка выбором."""
    N = len(A)
    for pos in range(0, N-1):
        for k in range(pos+1, N):
            if A[k] < A[pos]:
                A[k], A[pos] = A[pos], A[k]
```

———

Вопрос 436. Опишите и реализуйте алгоритм сортировки пузырьком? Какова сложность алгоритма?
Ответ:

Сложность алгоритма - O(n^2)

Рассматриваются только 2 соседних элемента, если первый больше второго, то они меняются местами. Отсортированная часть массива начинается с конца.

```
def bubble_sort(A):
    """Сортировка пузырьком."""
    N = len(A)
    for bypass in range(1, N):
        for k in range(0, N-bypass):
            if A[k] > A[k+1]:
                A[k], A[k+1] = A[k+1], A[k]
```

———

Вопрос 437. Опишите и реализуйте алгоритм сортировки подсчётом? Какова сложность алгоритма?

Ответ:

Позволяет сортировать большое количество данных очень быстро

Скорость алгоритма - O(n), по памяти O(m), где m - количество m различных элементов

Считаем сколько штук элементов 0, 1, 2, 3 и т.к.

Проходим один раз циклом for и считаем сколько раз встречается тот или иной элемент.

Алгоритм на следующей странице

```python
def count_sort(A):
    """Сортировка подсчетом."""
    # Частотный анализ
    N = len(a)
    output = [0] * N
    count = [0] * 10

    for i in range(0, N):
        count[A[i]] += 1

    for i in range(1, 10):
        count[i] += count[i - 1]

    i = N - 1
    while i >= 0:
        output[count[A[i]] - 1] = A[i]
        count[A[i]] -= 1
        i -= 1

    for i in range(0, N):
        A[i] = output[i]

    return A
```

———

Вопрос 438. Напишите алгоритм для нахождения наибольшего общего делителя двух целых чисел (Алгоритм Евклида)

Livecoding: даны два числа: число 84 и число 90. Найти НОД этих чисел:

Ответ:

Алгоритм реализуется с помощью рекурсии:

```python
def gcd(a, b):
    """Алгоритм Евклида рекурсивный."""
    return (a if b == 0 else gcd(b, a % b))
```

Задача: ответ 6

———

Вопрос 439. Напишите алгоритм возведения числа в степень с использованием рекурсии?

Ответ:

Алгоритм выглядит так:

Алгоритм на следующей странице

```python
def pow(a: float, n: int):
    """Алгоритм возведения в степень рекурентный.
    a - число.
    n - степень, в которую возводим число a.
    """
    if n ==  0:
        return 1
    elif n % 2 == 1:
        return pow(a, n-1) * a
    else:
        return pow(a * a, n//2)
```

———

Вопрос 440. Что такое перестановки в программировании? Приведите любой пример

Ответ:

Перестановка – это комбинация элементов из N разных элементов взятых в определенном порядке. В перестановке важен порядок следования элементов, и в перестановке должны быть задействованы все N элементов.

Допустим у нас 2-ая система счисления и мы хотим перебрать все комбинации числа с тремя разрядами

Вывод будет такой:

```
[0, 0, 0]
[0, 0, 1]
[0, 1, 0]
[0, 1, 1]
[1, 0, 0]
[1, 0, 1]
[1, 1, 0]
[1, 1, 1]
```

———

Вопрос 441. Напишите алгоритм генерации перестановок элементов в массиве? Библиотекой itertools не пользоваться

Ответ:

Алгоритм реализуется так:

Алгоритм на следующей странице

```
def find(number, A):
    """Ищет number в A и возвращает True, если такой есть, иначе
False."""
    flag = False
    for x in A:
        if number == x:
            flag = True
            break
    return flag


def generate_permutations(N: int, M: int = -1, prefix=None):
    """Генерирует все числа с лидирующими нулями
    с N-ричной системе счисления (N <= 10)    длины M.    N -
основание системы счисления.    M - количество чисел.    """
    M = N if M == -1 else M
    prefix = prefix or []
    if M == 0:
        print(*prefix)
        return
    for number in range(1, N + 1):
        if find(number, prefix):
            continue
        prefix.append(number)
        generate_permutations(N, M - 1, prefix)
        prefix.pop()
```

———

Вопрос 442. Реализуйте алгоритм перестановок с помощью библиотеки itertools для Python?

Ответ:

Для этого в библиотеке itertools используется функция product:

```
import itertools

numbers = [1, 2, 3, 4]
combinations = list(itertools.product(numbers,
repeat=len(numbers)))
for i in combinations:
    print(i)
```

———

Вопрос 443. Напишите алгоритм быстрой сортировки (сортировка Хоара)? Какая сложность алгоритма?

Livecoding: дан список [7, 4, 7, 3, 0, 2, 1]. Произвести сортировку чисел списка алгоритмом Хоара

Ответ:

Быстрая сортировка quick sort, по-другому сортировка Тони Хоара

- сложность O(N * logN), иногда O(n^2)

- выполняется на прямом ходу рекурсии

- не требует дополнительной памяти

```
def quick_sort(A):
    if len(A) <= 1:
        return
    barrier = A[0]
    L = []
    M = []
    R = []
    for x in A:
        if x < barrier:
            L.append(x)
        elif x == barrier:
            M.append(x)
        else:
            R.append(x)
    quick_sort(L)
    quick_sort(R)
    k = 0
    for x in L + M + R:
        A[k] = x
        k += 1
    return A
```

Задача: ответ [0, 1, 2, 3, 4, 7, 7]

———

Вопрос 444. Напишите алгоритм сортировки слиянием? Какая сложность алгоритма?

Ответ:

- сложность O(N * logN)

- выполняется на обратном ходу рекурсии

- требует дополнительной памяти

Дополнительная память требуется потому, что требуется дополнительный массив.

Сортировка называется устойчивой, если она не меняет порядок равных элементов

Желательно всегда соблюдать устойчивость.

Алгоритм на следующей странице

```python
def merge(A: list, B:list):
    """Получет списки A и B и возвращает список C."""
    C = [0] * (len(A) + len(B))
    i = k = n = 0
    while i < len(A) and k < len(B):
        if A[i] <= B[k]:
            C[n] = A[i]
            i += 1
            n += 1
        else:
            C[n] = B[k]
            k += 1
            n += 1
    while i < len(A):
        C[n] = A[i]
        i += 1
        n += 1
    while k < len(B):
        C[n] = B[k]
        k += 1
        n += 1
    return C

def merge_sort(A):
    if len(A) <= 1:
        return
    middle = len(A) // 2
    L = [A[i] for i in range(middle)]
    R = [A[i] for i in range(middle, len(A))]
    merge_sort(L)
    merge_sort(R)
    C = merge(L,R)
    for i in range(len(A)):
        A[i] = C[i]
    return A
```

—

Вопрос 445. Напишите алгоритм отсортированности? То есть дать ответ: является ли массив отсортированным или нет?

Ответ:

Сложность алгоритма O(n)

Алгоритм на следующей странице

```python
def check_sorted(A, ascending=True):
    """ascending = True - по возрастанию.
    ascending = False - по убыванию.
    """
    flag = True
    s = 2 * int(ascending) - 1
    for i in range(0, len(A) - 1):
        if s*A[i] > s*A[i + 1]:
            flag = False
            break
    return flag
```

—

Вопрос 446. Напишите алгоритм бинарного поиска? Что самое главное при реализации бинарного поиска?

Ответ: Самое важно требование - **МАССИВ ДОЛЖЕН БЫТЬ ОТСОРТИРОВАН**

Реальный поиск в отсортированном массиве должен сводиться к 2ум поисковым операциям - поиск левой границы и поиск правой границы.

```python
def left_bound(A, key):
    """Поиск левой границы key в списке A."""
    left = -1
    right = len(A)
    while right - left > 1:
        middle = (left + right) // 2
        if A[middle] < key:
            left = middle
        else:
            right = middle
    return left

def right_bound(A, key):
    """Поиск правой границы key в списке A."""
    left = -1
    right = len(A)
    while right - left > 1:
        middle = (left + right) // 2
        if A[middle] <= key:
            left = middle
        else:
            right = middle
    return right
```

—

Вопрос 447. Что такое числа Фибоначчи и для чего они могут быть использованы?

Ответ: Числа Фибоначчи - это последовательность чисел, в которой каждое число (кроме первых двух) является суммой двух предыдущих. Формально, последовательность начинается с 0 и 1, и затем каждый следующий элемент равен сумме двух предыдущих:

Таким образом, последовательность начинается: 0, 1, 1, 2, 3, 5, 8, 13, 21 и так далее.

Использование чисел Фибоначчи:

Математические исследования: Числа Фибоначчи встречаются в различных математических задачах и исследованиях, таких как теория чисел, комбинаторика и золотое сечение.

Финансовая математика: В финансовой математике числа Фибоначчи могут использоваться для моделирования изменения цен на финансовых рынках, особенно в контексте технического анализа

Алгоритмы и программирование: Числа Фибоначчи используются в различных алгоритмах и программировании, например, для оптимизации рекурсивных функций и динамического программирования.

Генерация искусственных структур: Некоторые исследования используют числа Фибоначчи для создания определенных структур в искусстве и дизайне.

Криптография: В криптографии существуют алгоритмы, которые используют числа Фибоначчи для создания последовательностей или ключей.

Моделирование природных явлений: Некоторые природные явления и структуры могут быть приближенно описаны с использованием чисел Фибоначчи.

—

Вопрос 448. Напишите алгоритм для создания массива из чисел Фибоначчи заданной длины?

Ответ: Для реализации чисел Фибоначчи НИКОГДА НЕ ИСПОЛЬЗОВАТЬ РЕКУРСИЮ - скорость работы алгоритма с рекурсией O(2^n)

Поэтому принята следующая реализация O(n) - ЭТО ПРИМЕР ДИНАМИЧЕСКОГО ПРОГРАММИРОВАНИЯ:

```
n = 10
fib = [0, 1] + [0] * (n - 1)

for i in range(2, n + 1):
    fib[i] = fib[i - 1] + fib[i - 2]

print(fib)
```

—

Вопрос 449. Задача «Ханойские башни». Головоломка “Ханойские башни” состоит из трех стержней, пронумерованных числами 1, 2, 3. На стержень 1 надета пирамидка из n дисков различного диаметра в порядке возрастания диаметра. Диски можно перекладывать с одного стержня на другой по одному, при этом диск нельзя класть на диск меньшего диаметра. Необходимо переложить всю пирамидку со стержня 1 на стержень 3 за минимальное число перекладываний.

Ответ:

Алгоритм на следующей странице

```
def tower_of_hanoi(n, source, destination, helper):
    if n==1:
       print ("Move disk 1 from peg", source," to peg",
destination)
       return
    tower_of_hanoi(n-1, source, helper, destination)
    print ("Move disk",n," from peg", source," to peg",
destination)
    tower_of_hanoi(n-1, helper, destination, source)
# n = number of disks
n = 3
tower_of_hanoi(n,' A','B','C')
```

—

Вопрос 450. Задача «Кузнечик». На числовой прямой сидит кузнечик, который может прыгать вправо на одну или на две единицы. Первоначально кузнечик находится в точке с координатой 0. Определите количество различных маршрутов кузнечика, приводящих его в точку с координатой n.

Ответ:

```
def traj_num(N):
    """
    N - координата точки, куда надо пропрыгать.
    Возвращает количество различных маршрутов.
    """
    k = [0, 1] + [0]* N
    for i in range(2, N+1):
        k[i] = k[i-2] + k[i - 1]
    return k[N]
```

—

Вопрос 451. Напишите алгоритм решения задачи нахождения наибольшей общей подпоследовательности.

Ответ:

```
def longest_common_subsequence(A, B):
    F = [[0] * (len(B) + 1) for i in range(len(A) + 1)]
    for i in range(1, len(A) + 1):
        for j in range(1, len(B) + 1):
            if A[i - 1] == B[j - 1]:
                F[i][j] = 1 + F[i - 1][j - 1]
            else:
                F[i][j] = max(F[i - 1][j], F[i][j - 1])
    return F[-1][-1]
```

—

Вопрос 452. Напишите алгоритм наибольшей возрастающей подпоследовательности?

Ответ:

Алгоритм на следующей странице

```
def longest_increasing_subsequence(A):
    F = [0] * (len(A) + 1)
    for i in range(1, len(A) + 1):
        m = 0
        for j in range(1, i):
            if A[i] > A[j] and F[j] > m:
                m = F[j]
        F[i] = m + 1
    return F[len(A)]
```

—

Вопрос 453. Задача о рюкзаке (задача о ранце). Из заданного множества предметов со свойствами «стоимость» и «вес» требуется отобрать подмножество с максимальной полной стоимостью, соблюдая при этом ограничение на суммарный вес.

Начальные условия Максимальная дискретная масса рюкзака M = 3, Количество предметов в рюкзаке N = 3, m[i] – массы предметов, v[i] - стоимости предметов m = [1, 2, 4, 5, 7], v = [100, 20, 30, 20, 10]

Напишите алгоритм реализации.

Ответ:

```
# Макимальная дискретная масса рюкзака
M = 3
# Количество предметов в рюкзаке
N = 3

# m[i] - массы, v[i] - стоимости предметов
m = [1, 2, 4, 5, 7]
v = [100, 20, 30, 20, 10]

F = [[0] * (N + 1) for i in range(M + 1)]
for i in range(1, N + 1):
    for k in range(1, M + 1):
        if m[i] <= k:
            F[k][i] = max(F[k][i - 1], v[i] + F[k - m[i]][i -
1])
        else:
            F[k][i] = F[k][i - 1]

print(F[M][N])
```

—

Вопрос 454. Напишите алгоритм определения редакционного расстояния между строками (алгоритм Левенштейна). Какова сложность алгоритма?

Livecoding: Сколько типографических опечаток (перепутали символ, вставили лишний символ, потеряли нужный символ) в слове «колокол» надо совершить, чтобы получилось слово «молоко».

Ответ:

Алгоритм на следующей странице

```
def levenstein(A, B):
    F = [[(i + j) if i * j == 0 else 0 for j in range(len(B) +
1)] for i in range(len(A) + 1)]
    for i in range(1, len(A) + 1):
        for j in range(1, len(B) + 1):
            if A[i - 1] == B[j - 1]:
                F[i][j] = F[i - 1][j - 1]
            else:
                F[i][j] = 1 + min(F[i-1][j], F[i][j - 1], F[i -
1][j -1])
    return F[len(A)][len(B)]
```

Задача

```
print(levenstein('колокол', 'молоко'))
```

Ответ: 2: букву «к» меняем на букву «м», последнюю букву «л» убираем.

—

Вопрос 455. Написать алгоритм Кнута-Морриса-Пратта (КМП) для поиска подстроки в строке.

Livecoding: сколько раз в строке 'с новым годом' встречается подстрока 'годам'. Не использовать оператор in. Пользоваться алгоритмом КМП

Ответ:

```
def pi_fuction(t: str):
    p = [0] * len(t)
    j = 0
    i = 1
    while i < len(t):
        if t[j] == t[i]:
            p[i] = j + 1
            i += 1
            j += 1
        else:
            if j == 0:
                p[i] = 0
                i += 1
            else:
                j = p[j - 1]
    return p


def knut_morris_pratt(a: str, t: str):
    m = len(t)
    n = len(a)
    p = pi_fuction(t)
```

```
    i = 0
    j = 0
    while i < n:
        if a[i] == t[j]:
            i += 1
            j += 1
            if j == m:
                return 'Образ найден'
        else:
            if j > 0:
                j = p[j - 1]
            else:
                i += 1
    if i == n:
        return 'Образ не найден'
```

Задача:

```
print(knut_morris_pratt('с новыйм годом', 'годам'))
```

Ответ: Образ не найден.

—

Вопрос 456. Задача на правильную скобочную последовательность. Задана строка, в которой могут быть встречены 3 типа скобок: фигурные, квадратные и круглые. Помимо скобок в строке встречаются и другие последовательности символов. Вложенность скобок может быть произвольной. Необходимо проверить корректность скобочной записи: каждой открывающей скобке должна соответствовать следующая за ней закрывающая скобка того же типа на том же уровне вложенности, не должно быть открывающей или закрывающей скобки без пары.

Ответ:

Если в выражении всего один вид скобок (например, круглые) - стек не нужен.

Но если, есть и другой или другие виды скобок - тогда нужен стек.

Пример, "[( ])" - некорректная скобочная последовательность.

Для очередной скобки, если она открывается - то кладем её в стек, иначе выполняем проверку - если стек пуст, то сразу некорректная скобочная последованиельность. Но если в стеке что-то есть x = pop(), если x не соответствует y - то некорректная скобочная последовательность.

Реализация алгоритма

Алгоритм на следующей странице

```
from collections import deque


def is_braces_sequence_correct(s: str):
    """
    Проверяет корректность скобочной последовательности
    из круглых () и квадратных [] скобок.
    """
    A_stack = deque()
    for brace in s:
        if brace not in "()[]":
            continue
        if brace in "([":
            A_stack.append(brace)
        else:
            assert brace in ")]", "Ожидалась закрывающая скобка
" + str(brace)
            if not len(A_stack):
                return False
            left = A_stack.pop()
            assert left in "([", "Ожидалась открывающая скобка "
+ str(brace)
            if left == "(":
                right = ")"
            elif left == "[":
                right = "]"
            if right != brace:
                return False
    return len(A_stack) == 0
```

—

Вопрос 457. Задача «Обратная польская запись». В единственной строке записано выражение в постфиксной записи, содержащее однозначные числа и операции +, –, *, /. Строка содержит не более 100 чисел и операций. Числа и операции отделяются друг от друга ровно одним пробелом.

```
Например, строка «8 9 + 1 7 - *»
```

Необходимо вывести значение записанного выражения.

Ответ:

Алгоритм на следующей странице

```python
from dataclasses import dataclass, field
from typing import Any

OPERATIONS = {
    '+': lambda x, y: x + y,
    '-': lambda x, y: y - x,
    '*': lambda x, y: x * y,
    '/': lambda x, y: y // x
}


@dataclass
class StackIsEmptyError(IndexError):
    __error: str

    @property
    def error(self):
        return self.__error

    @error.setter
    def error(self, value):
        self.__error = value


@dataclass
class Stack:
    size: int = 0
    stack: Any = field(default_factory=list)

    def push(self, value):
        self.size += 1
        self.stack.append(value)

    def pop(self):
        if self.size == 0:
            raise StackIsEmptyError('Стек пуст')
        self.size -= 1
        return self.stack.pop()


if __name__ == '__main__':
    stack = Stack()
    for value in '8 9 + 1 7 - *'.split(' '):
        operation = OPERATIONS.get(value)
        stack.push(operation(stack.pop(), stack.pop()) if
operation else int(value))
```

```
print(stack.pop())
```

Решение: -102

___

Вопрос 458. Написать алгоритм пирамидальной сортировки (сортировки кучей)? Какова сложность данного алгоритма?

Ответ:

1. Постройте max-heap из входных данных.

2. На данном этапе самый большой элемент хранится в корне кучи. Замените его на последний элемент кучи, а затем уменьшите ее размер на 1. Наконец, преобразуйте полученное дерево в max-heap с новым корнем.

3. Повторяйте вышеуказанные шаги, пока размер кучи больше 1.

Сложность сортировки по времени O(n * log n)

Шаги к правильному решению

1. Создадим функцию `heapsort`, которая принимает на вход список.

2. Вызовем функцию `build_max_heap` с параметром `alist` для представления листа в виде пирамиды (heap).

3. Поменяем местами первый и последний элемент пирамиды.

4. Вызовем функцию `max_heapify`, учитывая что новая пирамида имеет размер на единицу меньше. Установим `index=0` для удовлетворения параметрам пирамиды.

5. Повторим шаги 3 и 4, пока размер пирамиды не станет 0 и весь список не отсортируется.

6. Определим функцию `parent`, которая принимает `index` и возвращает индекс родителя.

7. Определим функцию `left`, которая принимает `index` и возвращает индекс левого дочернего элемента.

8. Определим функцию `right`, которая принимает `index` и возвращает индекс правого дочернего элемента.

9. Определим функцию `build_max_heap`, которая принимает список аргументов и переставляет их в соответствии с `max heap`.

10. `build_max_heap` вызывает `max_heapify` на каждом родительском ноде и проходит до вершины.

11. Определим функцию `max_heapify`, которая принимает индекс и изменяет структуру пирамиды на ноде и снизу от индекса так, чтобы удовлетворять правилам пирамиды.

Алгоритм на следующей странице

```
def heapsort(alist):
    build_max_heap(alist)
    for i in range(len(alist) - 1, 0, -1):
        alist[0], alist[i] = alist[i], alist[0]
        max_heapify(alist, index=0, size=i)


def parent(i):
    return (i - 1) // 2


def left(i):
    return 2 * i + 1


def right(i):
    return 2 * i + 2


def build_max_heap(alist):
    length = len(alist)
    start = parent(length - 1)
    while start >= 0:
        max_heapify(alist, index=start, size=length)
        start = start - 1


def max_heapify(alist, index, size):
    l = left(index)
    r = right(index)
    if (l < size and alist[l] > alist[index]):
        largest = l
    else:
        largest = index
    if (r < size and alist[r] > alist[largest]):
        largest = r
    if (largest != index):
        alist[largest], alist[index] = alist[index],
alist[largest]
        max_heapify(alist, largest, size)
```

—

Вопрос 459. Как с помощью клавиатуры создать матрицу смежности?

Ответ:

Алгоритм на следующей странице

```
# M - количество ребер
# N - количество вершин
M, N = [int(x) for x in input().split()]
# Заготавливаем ту струтуру, где будет граф
V = []
index = {}
A = [[0] * N for _ in range(N)]
for i in range(N):
    v1, v2 = input().split()
    for v in v1, v2:
        if v not in index:
            V.append(v)
            index[v] = len(V) - 1
    v1_i = index[v1]
    v2_i = index[v2]
    A[v1_i][v2_i] = 1
    A[v2_i][v1_i] = 1

print(A)
```

—

Вопрос 460. Как с помощью клавиатуры создать список смежности?

Ответ:

```
# M - количество ребер
# N - количество вершин
M, N = [int(x) for x in input().split()]
G = {}
for i in range(N):
    v1, v2 = input().split(")
                           for v, u in (v1, v2), (v2, v1):
    if v not in G:
        G[v] = {u}
    else:
        G[v].add(u)

    print(G)
```

—

Вопрос 461. Напишите алгоритм обхода графа в глубину DFS (deep-first search)

Ответ:

Основано на рекурсии, можно подчитать количество вершин.

Что можно с этим обходом делать:

1. Выделение компоненты связанности

2. Подсчет количества компонент

3. Поиск простого цикла

4. проверка двудольной

```
def dfs(vertex, G):
    """
    vertex - вершины,
    G - сам граф (список смежности),
    used - серые вершины.
    """
    used.add(vertex)
    for neighbor in G[vertex]:
        if neighbor not in used:
            dfs(neighbor, G, used)
```

—

Вопрос 462. Для чего используется алгоритм Косарайю?

Ответ: для выделения сильных компонентов ОРФО графа. Для того чтобы найти компоненты сильной связности, сначала выполняется поиск в глубину, каждый раз выбирается не посещенная вершина с максимальным номером, который был получен при обратном проходе. Полученные деревья являются сильно связными компонентами.

—

Вопрос 463. Для чего используется алгоритм Тарьяна?

Ответ: Этот алгоритм в первую очередь представляет из себя один из вариантов поиска в глубину. Вершины посещаются от корней к листьям, а окончание их обработки происходит на обратном пути.

—

Вопрос 464. Напишите алгоритм обхода графа в ширину BFS? Для чего используется данный алгоритм? Какова его сложность?

Ответ:

Применение:

- поиск кратчайшего расстояния от центральной вершины до других

- восстанавливать кратчайшие пути

Для правильного порядка обхода вершин используется очередь FIFO.

Сложность алгоритма O(m + n)

Реализация алгоритма:

```
from collections import deque

N, M = map(int, input().split())
graph = {i: set() for i in range(N)}
```

```
for i in range(N):
    v1, v2 = map(int, input().split())
    graph[v1].add(v2)
    graph[v2].add(v1)
distances = [None] * N
start_vertex = 0
distances[start_vertex] = 0
deque = deque([start_vertex])
while deque:
    cur_v = deque.popleft()
    for neigh_v in graph[cur_v]:
        if distances[neigh_v] is None:
            distances[neigh_v] = distances[cur_v] + 1
            deque.append(neigh_v)
print(distances)
```

—

Вопрос 465. Напишите алгоритм поиска кратчайшего пути (алгоритм Дейкстры).

Ответ:

Алгоритм Дейкстры - обход графа в ширину с перезажиганием.

Требование:

- веса ребер - неотрицательные числа

Асимптотика:

зависит от реализации, но как правило O(n ^ 2)

Цель

поиск кратчайших путей от исходной вершины ко всем остальным.

Виды:

- с очередью

- без очереди

- с приватизированной очередью

Алгоритм на следующей странице

```
from collections import deque


def main():
    G = read_graph()
    start = input('С какой вершины начать? ')
    while start not in G:
        start = input('Такой вершины в графе нет. С какой
вершины начать?')
    shortest_distances = dijkstra(G, start)
    finish = input('К какой вершине построить путь')
    while start not in G:
        start = input('Такой вершины в графе нет. С какой
вершины построить?')
    shortest_path = reveal_shortest_path(start, finish,
shortest_distances)


def read_graph():
    M = int(input())  # Количество ребер, далее A, B и вес
    G = {}
    for i in range(M):
        a, b, weight = input().split()
        weight = float(weight)
        add_edge(G, a, b, weight)
        add_edge(G, b, a, weight)
    return G


def add_edge(G, a, b, weight):
    if a not in G:
        G[a] = {b: weight}
    else:
        G[a][b] = weight


def dijkstra(G, start):
    Q = deque()
    S = {}
    S[start] = 0
    Q.append(start)
    while Q:
        v = Q.pop()
        for u in G[v]:
            if u not in S or S[v] + G[v][u] < S[u]:
                S[u] = S[v] + G[v][u]
```

```
                Q.append(u)


if __name__ == '__main__':
    main()
```

—

Вопрос 466. Задача на создание треугольников. Для того чтобы составить треугольники из данных точек и посчитать их периметры, нужно перебрать все возможные комбинации из трех точек и для каждой комбинации проверить, являются ли эти точки вершинами треугольника. Если да, то можно вычислить периметр этого треугольника. Реализовать с помощью Python

Ответ:

```
import itertools


def distance(p1, p2):
    """Вычисляет расстояние между двумя точками p1 и p2."""
    return ((p1[0] - p2[0]) ** 2 + (p1[1] - p2[1]) ** 2) ** 0.5


def is_triangle(p1, p2, p3):
    """Проверяет, являются ли точки p1, p2 и p3 вершинами
треугольника."""
    d1 = distance(p1, p2)
    d2 = distance(p1, p3)
    d3 = distance(p2, p3)
    return d1 + d2 > d3 and d1 + d3 > d2 and d2 + d3 > d1


def perimeter(p1, p2, p3):
    """Вычисляет периметр треугольника, образованного точками
p1, p2 и p3."""
    return distance(p1, p2) + distance(p1, p3) + distance(p2,
p3)


A = {'A': (1, 2), 'B': (2, 4), 'C': (2, 1), 'D': (4, 3), 'E':
(6, 2)}
triangles = {}

# Перебираем все возможные комбинации из трех точек
for comb in itertools.combinations(A.keys(), 3):
    p1, p2, p3 = A[comb[0]], A[comb[1]], A[comb[2]]

    # Проверяем, являются ли точки вершинами треугольника
    if is_triangle(p1, p2, p3):
        triangle_name = ''.join(comb)  # Создаем имя
треугольника из названий точек
        triangles[triangle_name] = perimeter(p1, p2, p3)  #
Вычисляем периметр треугольника

print(triangles)
```