

Алгоритм и анализ сложности

Лабораторная работа 2

Крюков Никита А. (АТ-01)
РИ-230915

Вариант: 14

Задача 2. Эффективная быстрая сортировка.

Ограничение по времени: 3 с. Ограничение по памяти: 64 Мб.

Тимофей решил организовать соревнование по спортивному программированию, чтобы найти талантливых стажёров. Задачи подобраны, участники зарегистрированы, тесты написаны. Осталось придумать, как конце соревнования будет определяться победитель.

Каждый участник имеет уникальный логин. Когда соревнование закончится, к нему будут привязаны два показателя: количество решённых задач P_i размер штрафа F_i . Штраф начисляется за неудачные попытки и время, затраченное на задачу.

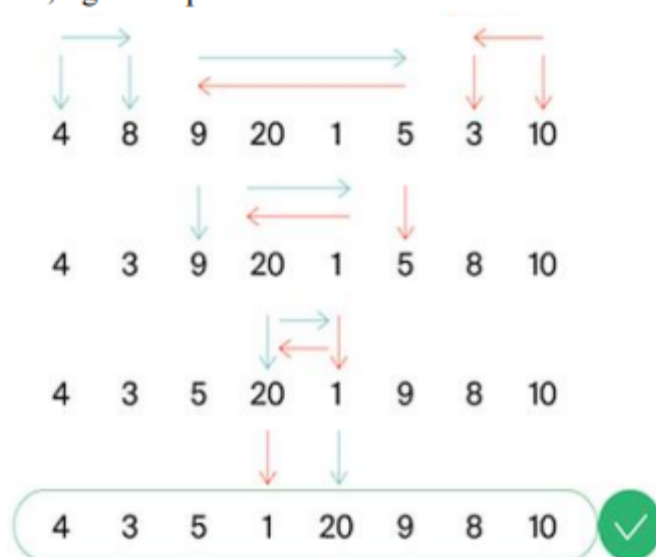
Тимофей решил сортировать таблицу результатов следующим образом: при сравнении двух участников выше будет идти тот, у которого решено больше задач. При равенстве числа решенных задач первым идет участник с меньшим штрафом. Если же и штрафы совпадают, то первым будет тот, у которого логин идёт раньше в алфавитном (лексикографическом) порядке.

Тимофей заказал толстовки для победителей и накануне поехал за ними в магазин. В своё отсутствие он поручил вам реализовать алгоритм быстрой сортировки (англ. quick sort) для таблицы результатов. Так как Тимофей любит спортивное программирование и не любит зря расходовать оперативную память, то ваша реализация сортировки не может потреблять $O(n)$ дополнительной памяти для промежуточных данных (такая модификация быстрой сортировки называется "in-place").

Затем сортировка вызывается рекурсивно для двух полученных частей. Именно на этапе разделения элементов на группы в обычном алгоритме используется дополнительная память. Теперь разберёмся, как реализовать этот in-place.

Пусть мы как-то выбрали опорный элемент. Заведём два указателя `left` и `right`, которые изначально будут указывать на левый и правый концы отрезка соответственно. Затем будем двигать первый указатель вправо до тех пор, пока он указывает на элемент, меньший опорного. Аналогично двигаем правый указатель влево, пока он стоит на элементе, превосходящем опорный. В итоге окажется, что что левее от `left` все элементы точно принадлежат первой группе, а правее от `right` — второй. Элементы, на которых стоят указатели, нарушают порядок. Поменяем их местами (в большинстве языков программирования используется функция `swap()`) и продвинем указатели на следующие элементы. Будем повторять это действие до тех пор, пока `left` и `right` не столкнутся.

На рисунке представлен пример разделения при `pivot=5`. Указатель `left` — голубой, `right` — оранжевый.



Формат входных данных:

В первой строке задано число участников n , $1 \leq n \leq 100000$.

В каждой из следующих n строк задана информация про одного из участников.

i -й участник описывается тремя параметрами:

- уникальным логином (строкой из маленьких латинских букв длиной не более 20)
- числом решённых задач P_i
- штрафом F_i

F_i и P_i — целые числа, лежащие в диапазоне от 0 до 10^9 .

Формат выходных данных:

Для отсортированного списка участников выведите по порядку их логины по одному в строке.

Код

```
72  def main():
73      # Чтение количества участников
74      n = int(input())
75
76      participants = []
77      for _ in range(n):
78          # Чтение данных участника
79          input_login, input_solved, input_penalty = input().split()
80          input_solved = int(input_solved)
81          input_penalty = int(input_penalty)
82          # Добавление участника в список
83          participants.append(Participant(input_login, input_solved, input_penalty))
84
85      # Запуск таймера
86      start_time = time.perf_counter()
87
88      # Сортировка участников
89      sorter = QuickSort(participants)
90      sorted_participants = sorter.sort()
91
92      # Вывод отсортированных логинов
93      for participant in sorted_participants:
94          print(participant.login)
95
96      stop_time = time.perf_counter()
97
98      # Вывод времени выполнения
99      print("=" * 42)
100     print(f"    Время выполнения: {stop_time - start_time:0.5f} секунд.")
101     # Получение и вывод используемой памяти
102     final_memory = get_memory_usage()
103     print(f"    Использовано памяти: {final_memory:.2f} Mb.")
104     print("=" * 42)
105
106
107  if __name__ == "__main__":
108      main()
109
```

```

35  class QuickSort:
36  def __init__(self, participants: List[Participant]) -> None:
37      """Инициализация быстрой сортировки."""
38      self.participants = participants
39
40  3 usages
41  def __quick_sort(self, low: int, high: int) -> None:
42      """Алгоритм быстрой сортировки."""
43      if low < high:
44          # Разделение массива
45          mid = self.__partition(low, high)
46          # Рекурсивная сортировка левой части
47          self.__quick_sort(low, mid - 1)
48          # Рекурсивная сортировка правой части
49          self.__quick_sort(mid + 1, high)
50
51  1 usage
52  def __partition(self, low: int, high: int) -> int:
53      """Функция разделения массива."""
54      # Выбор опорного элемента
55      pivot = self.participants[high]
56      i = low - 1
57      for j in range(low, high):
58          # Сравнение элементов с опорным
59          if self.participants[j] < pivot: # (__lt__)
60              i += 1
61              # Обмен элементов
62              self.participants[i], self.participants[j] = self.participants[j], self.participants[i]
63      # Обмен опорного элемента
64      self.participants[i + 1], self.participants[high] = self.participants[high], self.participants[i + 1]
65      return i + 1
66
67  3 usages
68  def sort(self) -> List[Participant]:
69      """Сортировка массива с помощью быстрой сортировки."""
70      # Запуск быстрой сортировки
71      self.__quick_sort(low=0, len(self.participants) - 1)
72      return self.participants

```

```

1  from typing import List, Tuple
2  import time
3  import os
4  import psutil
5
6
7  usage
8  def get_memory_usage():
9      """Получение используемой памяти в мегабайтах."""
10     process = psutil.Process(os.getpid())
11     mem_info = process.memory_info()
12     return mem_info.rss / (1024 * 1024)
13
14  usages
15  class Participant:
16      def __init__(self, login: str, solved: int, penalty: int):
17          """Инициализация участника."""
18          self.login = login
19          self.solved = solved
20          self.penalty = penalty
21
22      def __lt__(self, other):
23          """Метод сравнения участников для сортировки."""
24          # Сравнение по количеству решенных задач
25          if self.solved != other.solved:
26              return self.solved > other.solved
27
28          # Сравнение по штрафу
29          if self.penalty != other.penalty:
30              return self.penalty < other.penalty
31
32          # Сравнение по логину
33          return self.login < other.login
34

```

```

5
alla 4 100
gena 6 1000
gosha 2 90
rita 2 90
Timofey 4 80
gena
Timofey
alla
gosha
rita
=====
Время выполнения: 0.00006 секунд.
Использовано памяти: 15.72 Mb.
=====

```

Тесты

```
import unittest
from q_2 import Participant, QuickSort

class TestCase(unittest.TestCase):
    def test_sort_1(self):
        """1. Тестирование на сортировку."""
        n = 5

        participants = []
        participants.append(Participant(login="alla", solved=4, penalty=100))
        participants.append(Participant(login="gena", solved=6, penalty=1000))
        participants.append(Participant(login="gosha", solved=2, penalty=90))
        participants.append(Participant(login="rita", solved=2, penalty=90))
        participants.append(Participant(login="timofey", solved=4, penalty=80))

        sorter = QuickSort(participants)
        sorted_participants = sorter.sort()

        participants_result = ["gena", "timofey", "alla", "gosha", "rita"]

        for i in range(len(participants_result)):
            self.assertEqual(sorted_participants[i].login, participants_result[i])

    def test_sort_2(self):
        """2. Тестирование на сортировку."""
        n = 5

        participants = []
        participants.append(Participant(login="alla", solved=0, penalty=0))
        participants.append(Participant(login="gena", solved=0, penalty=0))
        participants.append(Participant(login="gosha", solved=0, penalty=0))
        participants.append(Participant(login="rita", solved=0, penalty=0))
        participants.append(Participant(login="timofey", solved=0, penalty=0))

        sorter = QuickSort(participants)
        sorted_participants = sorter.sort()

        participants_result = ["alla", "gena", "gosha", "rita", "timofey"]

        for i in range(len(participants_result)):
            self.assertEqual(sorted_participants[i].login, participants_result[i])
```

Ran 2 tests in 0.001s

OK

Process finished with exit code 0

Ход решения

В первую очередь, читаем входные данные. Необходимо определить структуру данных, которая будет хранить информацию о каждом участнике. Для этого был создан класс `Participant`, который содержит три основных атрибута: логин участника, количество решенных задач и штрафное время. Сравнение происходит по следующим критериям: сначала по количеству решенных задач (в порядке убывания), затем по штрафному времени (в порядке возрастания) и, наконец, по логину (в алфавитном порядке).

Реализуем алгоритм быстрой сортировки, который будет использоваться для упорядочивания списка участников. Для этого был создан класс `QuickSort`, который принимает список участников и содержит методы для выполнения сортировки. Основной метод `sort` запускает процесс сортировки, вызывая вспомогательный метод `__quick_sort`, который рекурсивно разбивает массив на части и сортирует их. Метод `__partition` выбирает опорный элемент (`pivot`) и разделяет массив на две части: элементы, меньшие опорного, и элементы, большие опорного. Важным моментом здесь является строка `if self.participants[j] < pivot:`. Эта строка использует метод `__lt__`, определенный в классе `Participant`, для сравнения текущего элемента с опорным элементом. Метод `__lt__` позволяет определить, является ли текущий элемент меньше опорного элемента по заданным критериям. Этот процесс повторяется рекурсивно для каждой части массива, пока весь массив не будет отсортирован.

Дополнительно

Если видно плохо или хочется протестировать, то можете посмотреть мой код по заданию тут: https://github.com/ytkinroman/lab_2_complexity/tree/main/quest_2

Задача 4. Сортировка по многим полям.

Ограничение по времени: 2 с. Ограничение по памяти: 64 Мб.

В базе данных хранится N записей, вида (Name, a_1 , a_2 , ..., a_k) – во всех записях одинаковое число параметров. На вход задачи подается приоритет полей – перестановка на числах $1, \dots, k$ – записи нужно вывести по невозрастанию в соответствии с этим приоритетом. В случае, если приоритет полей таков: 3 4 2 1, то это следует воспринимать так: приоритет значений из 3 колонки самый высокий, приоритет значений из колонки 4 ниже, приоритет значений из колонки 2 еще ниже, а приоритет значений из колонки 1 самый низкий.

Формат входных данных:

$N \leq 1000$

$k: 1 \leq k \leq 10$

$p_1 p_2 \dots p_k$ – перестановка на k числах, разделитель – пробел

N строк вида

Name $a_1 a_2 \dots a_k$

Формат выходных данных:

N строк с именами в порядке, согласно приоритету

Примеры:

Стандартный ввод	Стандартный вывод
3	B
3	A
2 1 3	C
A 1 2 3	
B 3 2 1	
C 3 1 2	

Замечание. Так как колонка под номером 2 самая приоритетная, то переставить записи можно только двумя способами: (A, B, C) и (B, A, C). Следующий по приоритетности столбец – первый, и он позволяет выбрать из возможных перестановок только (B, A, C). Так как осталась ровно одна перестановка, третий приоритет не имеет значения.

Код

```

1 usage
48 def main():
49     N = int(input())
50     k = int(input())
51     priorities_str = input().split()
52     priorities = []
53     for p in priorities_str:
54         priorities.append(int(p))
55
56     records = []
57     for _ in range(N):
58         line = input().split()
59         name = line[0]
60         values = []
61         for v in line[1:]:
62             values.append(int(v))
63         records.append((name, values))
64
65     # Запуск таймера
66     start_time = time.perf_counter()
67
68     sorter = QuickSort(records, priorities)
69     sorted_records = sorter.sort()
70
71     for record in sorted_records:
72         print(record[0])
73
74     stop_time = time.perf_counter()
75
76     # Вывод времени выполнения
77     print("=" * 42)
78     print(f"    Время выполнения: {stop_time - start_time:0.5f} секунд.")
79     # Получение и вывод используемой памяти
80     final_memory = get_memory_usage()
81     print(f"    Использовано памяти: {final_memory:.2f} Mb.")
82     print("=" * 42)
83
84
85 if __name__ == "__main__":
86     main()
87

```

```

14 class QuickSort:
15     def __init__(self, array: List[Tuple[str, List[int]]], priorities: List[int]) -> None:
16         self.array = array
17         self.priorities = priorities
18
19     3 usages
20     def __quick_sort(self, low: int, high: int) -> None:
21         if low < high:
22             mid = self.__partition(low, high)
23             self.__quick_sort(low, mid - 1)
24             self.__quick_sort(mid + 1, high)
25
26     1 usage
27     def __partition(self, low: int, high: int) -> int:
28         pivot = self.array[high]
29         i = low - 1
30         for j in range(low, high):
31             if self.__compare(self.array[j], pivot):
32                 i += 1
33                 self.array[i], self.array[j] = self.array[j], self.array[i]
34             self.array[i + 1], self.array[high] = self.array[high], self.array[i + 1]
35         return i + 1
36
37     1 usage
38     def __compare(self, item1: Tuple[str, List[int]], item2: Tuple[str, List[int]]) -> bool:
39         for priority in self.priorities:
40             if item1[1][priority - 1] > item2[1][priority - 1]:
41                 return True
42             elif item1[1][priority - 1] < item2[1][priority - 1]:
43                 return False
44         return False
45
46     1 usage
47     def sort(self) -> List[Tuple[str, List[int]]]:
48         self.__quick_sort(low=0, len(self.array) - 1)
49         return self.array
50
51

```

```

q_4.py x test_q_4.py
1 from typing import List, Tuple
2 import time
3 import os
4 import psutil
5
6
7     1 usage
8     def get_memory_usage():
9         """Получение используемой памяти в мегабайтах."""
10        process = psutil.Process(os.getpid())
11        mem_info = process.memory_info()
12        return mem_info.rss / (1024 * 1024)
13
14

```

```
3
3
2 1 3
A 1 2 3
B 3 2 1
C 3 1 2
B
A
C
=====
        Время выполнения: 0.00004 секунд.
        Использовано памяти: 15.85 Mb.
=====

Process finished with exit code 0
```

Тесты

```

1  import unittest
2  from q_4 import QuickSort
3
4
5  class TestCase(unittest.TestCase):
6      def test_sort_1(self):
7          """1. Тестирование."""
8          priorities = [2, 1, 3]
9          records = [
10             ("A", [1, 2, 3]),
11             ("B", [3, 2, 1]),
12             ("C", [3, 1, 2])
13         ]
14         expected_output = [("B", [3, 2, 1]), ("A", [1, 2, 3]), ("C", [3, 1, 2])]
15
16         sorter = QuickSort(records, priorities)
17         sorted_records = sorter.sort()
18
19         self.assertEqual(sorted_records, expected_output)
20
21     def test_sort_2(self):
22         """2. Тестирование."""
23         priorities = [1, 2]
24         records = [
25             ("A", [1, 2]),
26             ("B", [2, 1]),
27             ("C", [3, 3]),
28             ("D", [4, 4])
29         ]
30         expected_output = [('D', [4, 4]), ('C', [3, 3]), ('B', [2, 1]), ('A', [1, 2])]
31
32         sorter = QuickSort(records, priorities)
33         sorted_records = sorter.sort()
34
35         self.assertEqual(sorted_records, expected_output)

```

```

36
37 def test_sort_3(self):
38     """3. Тестирование."""
39     priorities = [3, 1, 2]
40     records = [
41         ("A", [1, 2, 3]),
42         ("B", [2, 3, 1]),
43         ("C", [3, 1, 2]),
44         ("D", [4, 4, 4]),
45         ("E", [5, 5, 5])
46     ]
47     expected_output = [("E", [5, 5, 5]), ("D", [4, 4, 4]), ("A", [1, 2, 3]), ("C", [3, 1, 2]), ("B", [2, 3, 1])]
48
49     sorter = QuickSort(records, priorities)
50     sorted_records = sorter.sort()
51
52     self.assertEqual(sorted_records, expected_output)
53
54 def test_sort_4(self):
55     """4. Тестирование."""
56     priorities = [2, 1]
57     records = [
58         ("A", [1, 2]),
59         ("B", [2, 1])
60     ]
61     expected_output = [("A", [1, 2]), ("B", [2, 1])]
62
63     sorter = QuickSort(records, priorities)
64     sorted_records = sorter.sort()
65
66     self.assertEqual(sorted_records, expected_output)

```

Ran 4 tests in 0.001s

OK

Process finished with exit code 0

Ход решения

Код представляет собой реализацию алгоритма быстрой сортировки (QuickSort) для сортировки списка записей по нескольким приоритетам. Каждая запись состоит из имени и списка значений, а приоритеты определяют, какие значения в списке имеют больший вес при сравнении.

Класс QuickSort принимает список записей и список приоритетов. Метод `__quick_sort` рекурсивно разделяет массив на две части вокруг опорного элемента и сортирует их. Метод `__partition` выбирает опорный элемент (последний в подмассиве) и перемещает элементы так, чтобы все меньшие оказались слева, а большие — справа. Метод `__compare` сравнивает две записи по приоритетам, начиная с наиболее важного.

Каждая запись представляет собой кортеж, состоящий из имени и списка значений. Приоритеты — это список индексов, определяющих, какие значения в списке имеют больший вес при сравнении. Метод проходит по каждому

приоритету, начиная с наиболее важного. Для каждого приоритета сравниваются соответствующие значения из списков обеих записей. Если значение в первой записи больше, чем во второй, метод возвращает True, что означает, что первая запись должна стоять перед второй. Если значение меньше, возвращается False.

Дополнительно

Если видно плохо или хочется протестировать, то можете посмотреть мой код по заданию тут: https://github.com/ytkinroman/lab_2_complexity/tree/main/quest_4

Задача 5. Оболочка.

Ограничение по времени: 2 с. Ограничение по памяти: 64 Мб.

Имеется массив из N целочисленных точек на плоскости.

Требуется найти периметр наименьшего охватывающего многоугольника, содержащего все точки.

Формат входных данных:

N

$x_1 y_1$

$x_2 y_2$

...

$x_n y_n$

$5 \leq N \leq 500000$

$-10000 \leq x_i, y_i \leq 10000$

Формат выходных данных:

Одно вещественное число – периметр требуемого многоугольника с двумя знаками после запятой.

Код


```

105 if __name__ == "__main__":
106     n = int(input())
107
108     # Проверка на допустимое количество точек.
109     if not (MIN_POINTS <= n <= MAX_POINTS):
110         raise ValueError(f"Количество точек должно быть в диапазоне от {MIN_POINTS} до {MAX_POINTS}!")
111
112     my_points = []
113     for i in range(n):
114         x, y = input().split()
115         x, y = int(x), int(y)
116
117         if not (MIN_COORD <= x <= MAX_COORD) or not (MIN_COORD <= y <= MAX_COORD):
118             raise ValueError(f"Координаты точки должны быть в диапазоне от {MIN_COORD} до {MAX_COORD}!")
119
120         my_points.append((x, y))
121
122     start_time = time.perf_counter()
123
124     start_point = find_start_point(my_points)
125     # print(start_point, "точка старта")
126     my_points.remove(start_point)
127
128     sorter = QuickSort(my_points, start_point)
129     sorted_points = sorter.sort()
130     # sorted_points.append(start_point)
131     # print(sorted_points, "точки после сортировки без стартовой точки")
132
133     hull = graham_scan(sorted_points, start_point)
134     # print("Выпуклая оболочка:", hull)
135
136     p = perimeter(hull)
137     # print("Периметр:", round(p, 2))
138     print(round(p, 2))
139
140     stop_time = time.perf_counter()
141
142     print("=" * 42)
143     print(f"    Время выполнения: {stop_time - start_time:0.5f} секунд.")
144     final_memory = get_memory_usage()
145     print(f"    Использовано памяти: {final_memory:.2f} Mb.")
146     print("=" * 42)
147

```

```

60 def find_start_point(points: List[Tuple[int, int]]) -> Tuple[int, int]:
61     """Находит начальную точку (самую нижнюю и самую левую) из списка точек. """
62     current_min_point = points[0]
63
64     for current_point in points:
65         current_y = current_point[1]
66         current_x = current_point[0]
67
68         min_y = current_min_point[1]
69         min_x = current_min_point[0]
70
71         if current_y < min_y or (current_y == min_y and current_x < min_x):
72             current_min_point = current_point
73
74     return current_min_point
75
76 1 usage
77 def cross_product(o: Tuple[int, int], a: Tuple[int, int], b: Tuple[int, int]) -> float:
78     """Вычисляет векторное произведение для определения поворота."""
79     return (a[0] - o[0]) * (b[1] - o[1]) - (a[1] - o[1]) * (b[0] - o[0])
80
81 1 usage
82 def distance(p1: Tuple[int, int], p2: Tuple[int, int]) -> float:
83     """Вычисляет векторное произведение для определения поворота."""
84     return sqrt((p1[0] - p2[0]) ** 2 + (p1[1] - p2[1]) ** 2)
85
86 5 usages
87 def graham_scan(sorted_points: List[Tuple[int, int]], start_point: Tuple[int, int]) -> List[Tuple[int, int]]:
88     """Алгоритм Грэхема для нахождения выпуклой оболочки."""
89     hull = [start_point]
90     for point in sorted_points:
91         while len(hull) > 1 and cross_product(hull[-2], hull[-1], point) <= 0:
92             hull.pop() # Удаляем точку из списка, если она создаёт правый поворот. Если результат векторного произведения отрицательный.
93         hull.append(point)
94     return hull
95
96 5 usages
97 def perimeter(hull: List[Tuple[int, int]]) -> float:
98     """Вычисляет периметр выпуклой оболочки."""
99     result_perimeter = 0
100     for i in range(len(hull)):
101         result_perimeter += distance(hull[i], hull[(i + 1) % len(hull)])
102     return result_perimeter
103
104

```

```

25  class QuickSort:
26      def __init__(self, array: List[Tuple[int, int]], s_point: Tuple[int, int]) -> None:
27          self.array = array
28          self.start_point = s_point
29
30      2 usages
31      def __polar_angle(self, point: Tuple[int, int]) -> float:
32          """Вычисляет полярный угол точки p относительно начальной точки."""
33          return atan2(point[1] - self.start_point[1], point[0] - self.start_point[0])
34
35      3 usages
36      def __quick_sort(self, low: int, high: int) -> None:
37          """Алгоритм быстрой сортировки."""
38          if low < high:
39              mid = self.__partition(low, high)
40              self.__quick_sort(low, mid - 1)
41              self.__quick_sort(mid + 1, high)
42
43      1 usage
44      def __partition(self, low: int, high: int) -> int:
45          """Функция разделения массива."""
46          pivot = self.array[high]
47          pivot_angle = self.__polar_angle(pivot)
48          i = low - 1
49          for j in range(low, high):
50              if self.__polar_angle(self.array[j]) <= pivot_angle:
51                  i += 1
52                  self.array[i], self.array[j] = self.array[j], self.array[i]
53          self.array[i + 1], self.array[high] = self.array[high], self.array[i + 1]
54          return i + 1
55
56      4 usages
57      def sort(self) -> List[Tuple[int, int]]:
58          """Сортировка массива с помощью быстрой сортировки."""
59          self.__quick_sort(low: 0, len(self.array) - 1)
60          return self.array

```

```

q_5.py x
1  # Задача 5.
2
3  from typing import List, Tuple
4  from math import atan2, sqrt
5  import time
6  import os
7  import psutil
8
9  # Константы.
10
11  MIN_POINTS = 5
12  MAX_POINTS = 500000
13  MIN_COORD = -10000
14  MAX_COORD = 10000
15
16
17  # Считаем затраченную память.
18  usage
19  def get_memory_usage():
20      """Получение используемой памяти в мегабайтах."""
21      process = psutil.Process(os.getpid())
22      mem_info = process.memory_info()
23      return mem_info.rss / (1024 * 1024)

```

```

6
2 1
3 2
1 1
1 2
0 1
1 0
7.66
=====
Время выполнения: 0.00006 секунд.
Использовано памяти: 15.77 Мб.
=====

```

Тесты

```

import unittest
from q_5 import find_start_point, QuickSort, graham_scan, perimeter

class TestCase(unittest.TestCase):
    def test_find_start_point(self):
        """1. Тестирование на поиск начальной точки."""
        self.assertEqual(find_start_point([(2, 1), (2, 2), (2, 3), (3, 2), (1, 2)]), second: (2, 1)) # 5.
        self.assertEqual(find_start_point([(1, 0), (0, 1), (-1, 0), (0, -1), (0, 0)]), second: (0, -1)) # 5.
        self.assertEqual(find_start_point([(0, 1), (1, 1), (1, 0), (2, 1), (3, 2), (1, 2)]), second: (1, 0)) # 6.
        self.assertEqual(find_start_point([(0, 3), (1, 1), (2, 2), (4, 4), (0, 0), (1, 2), (3, 1), (3, 3)]), second: (0, 0)) # 8.

    def test_quick_sort_1(self):
        """2. Тестирование на сортировку."""
        points = [(2, 2), (2, 3), (3, 2), (1, 2)]
        pivot = (2, 1)
        expected_sorted_points = [(3, 2), (2, 3), (2, 2), (1, 2)]
        quick_sort = QuickSort(points, pivot)
        sorted_points = quick_sort.sort()
        self.assertEqual(sorted_points, expected_sorted_points)

    def test_quick_sort_2(self):
        """2. Тестирование на сортировку."""
        points = [(1, 0), (0, 1), (-1, 0), (0, 0)]
        pivot = (0, -1)
        expected_sorted_points = [(1, 0), (0, 1), (0, 0), (-1, 0)]
        quick_sort = QuickSort(points, pivot)
        sorted_points = quick_sort.sort()
        self.assertEqual(sorted_points, expected_sorted_points)

    def test_quick_sort_3(self):
        """2. Тестирование на сортировку."""
        points = [(0, 1), (1, 1), (2, 1), (3, 2), (1, 2)]
        pivot = (1, 0)
        expected_sorted_points = [(2, 1), (3, 2), (1, 1), (1, 2), (0, 1)]
        quick_sort = QuickSort(points, pivot)
        sorted_points = quick_sort.sort()
        self.assertEqual(sorted_points, expected_sorted_points)

```

```

def test_graham_scan_1(self):
    """3. Тестирование на правильность работы алгоритма Ерохима."""
    sorted_points = [(3, 2), (2, 3), (2, 2), (1, 2)]
    start_point = (2, 1)
    test_hull = [(2, 1), (3, 2), (2, 3), (1, 2)]

    hull = graham_scan(sorted_points, start_point)
    self.assertEqual(hull, test_hull)

def test_graham_scan_2(self):
    """3. Тестирование на правильность работы алгоритма Ерохима."""
    sorted_points = [(1, 0), (0, 1), (-1, 0)]
    start_point = (0, -1)
    test_hull = [(0, -1), (1, 0), (0, 1), (-1, 0)]

    hull = graham_scan(sorted_points, start_point)
    self.assertEqual(hull, test_hull)

def test_graham_scan_3(self):
    """3. Тестирование на правильность работы алгоритма Ерохима."""
    sorted_points = [(2, 1), (3, 2), (1, 1), (1, 2), (0, 1)]
    start_point = (1, 0)
    test_hull = [(1, 0), (3, 2), (1, 2), (0, 1)]

    hull = graham_scan(sorted_points, start_point)
    self.assertEqual(hull, test_hull)

def test_result_1(self):
    """4. Тестирование на результат работы алгоритма Ерохима."""
    hull = [(2, 1), (3, 2), (2, 3), (1, 2)]
    res_perimeter = 5.66
    p = round(perimeter(hull), 2)
    self.assertEqual(p, res_perimeter)

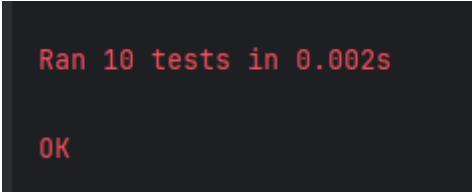
def test_result_2(self):
    """4. Тестирование на результат работы алгоритма Ерохима."""
    hull = [(0, -1), (1, 0), (0, 1), (-1, 0)]
    res_perimeter = 5.66
    p = round(perimeter(hull), 2)
    self.assertEqual(p, res_perimeter)

```

```

def test_result_3(self):
    """4. Тестирование на результат работы алгоритма Ерохима."""
    hull = [(1, 0), (3, 2), (1, 2), (0, 1)]
    res_perimeter = 7.66
    p = round(perimeter(hull), 2)
    self.assertEqual(p, res_perimeter)

```



```
Ran 10 tests in 0.002s
```

```
OK
```

Ход решения

Для решения задачи есть несколько способов, использовал алгоритм Грэхема.

Определил начальную точку, которая будет использоваться в качестве опорной для сортировки остальных точек. Для этого есть функция `find_start_point`, которая находит самую нижнюю и самую левую точку из списка точек. Эта точка будет начальной для всех последующих вычислений.

Далее отсортировал все точки по полярному углу относительно начальной точки. Для этого использовал класс `QuickSort`, который реализует алгоритм быстрой сортировки. Внутри этого класса определены методы для вычисления полярного угла (`__polar_angle`), разделения массива (`__partition`) и самой сортировки (`__quick_sort`). Метод `sort` запускает процесс сортировки и возвращает отсортированный массив точек.

После сортировки точек по полярному углу, применяю алгоритм Грэхема для нахождения выпуклой оболочки. Функция `graham_scan` последовательно добавляет точки в выпуклую оболочку, удаляя те, которые создают правый поворот (отрицательный вектор). Для определения поворота используется векторное произведение, вычисляемое функцией `cross_product`.

Для вычисления периметра выпуклой оболочки, используется функция `perimeter`, которая суммирует расстояния между соседними точками оболочки. Расстояние между точками вычисляется с помощью функции `distance`.

Дополнительно

Если видно плохо или хочется протестировать, то можете посмотреть мой код по заданию тут: https://github.com/ytkinroman/lab_2_complexity/tree/main/quest_5

Задача 8. Музей.

Ограничение по времени: 1 с. Ограничение по памяти: 16 Мб.

В музее регистрируется в течение суток время прихода и ухода каждого посетителя. Таким образом, за день получены N пар значений, где первое значение в паре показывает время прихода посетителя и второе значение - время его ухода. Требуется найти максимальное число посетителей, которые находились в музее одновременно.

Формат входных данных:

В первой строке входного файла INPUT.TXT записано натуральное число N ($N < 10^5$) – количество зафиксированных посетителей в музее в течении суток. Далее, идут N строк с информацией о времени визитов посетителей: в каждой строке располагается отрезок времени посещения в формате «ЧЧ:ММ ЧЧ:ММ» ($00:00 \leq \text{ЧЧ:ММ} \leq 23:59$).

Формат выходных данных:

В единственную строку выходного файла OUTPUT.TXT нужно вывести одно целое число — максимальное количество посетителей, одновременно находящихся в музее.

Примеры:

input.txt	output.txt
6 09:00 10:07 10:20 11:35 12:00 17:00 11:00 11:30 11:20 12:30 11:30 18:15	4

Код


```

102 usage
103 def main() -> None:
104     # Запуск таймера.
105     start_time = time.perf_counter()
106
107     filename = "input.txt"
108     num_records, intervals = read_file(filename)
109
110     museum = MuseumVisitors(intervals)
111     max_visitors = museum.find_max_visitors()
112     print(max_visitors)
113
114     = with open("output.txt", "w") as file:
115         file.write(str(max_visitors))
116
117     stop_time = time.perf_counter()
118
119     # Вывод времени выполнения.
120     print("=" * 42)
121     print(f"    Время выполнения: {stop_time - start_time:0.5f} секунд.")
122     # Получение и вывод используемой памяти.
123     final_memory = get_memory_usage()
124     print(f"    Использовано памяти: {final_memory:.2f} Mb.")
125     print("=" * 42)
126
127 if __name__ == "__main__":
128     main()

```

2 usages

class MuseumVisitors:

```
def __init__(self, intervals: List[List[Tuple[int, int]]]) -> None:
    self.intervals = intervals
```

3 usages

```
def __quick_sort(self, events: List[Tuple[Tuple[int, int], str]], low: int, high: int) -> None:
    if low < high:
        mid = self.__partition(events, low, high)
        self.__quick_sort(events, low, mid - 1)
        self.__quick_sort(events, mid + 1, high)
```

1 usage

```
def __partition(self, events: List[Tuple[Tuple[int, int], str]], low: int, high: int) -> int:
    pivot = events[high][0]
    i = low - 1
    for j in range(low, high):
        if events[j][0] < pivot or (events[j][0] == pivot and events[j][1] == "enter"):
            i += 1
            events[i], events[j] = events[j], events[i]
    events[i + 1], events[high] = events[high], events[i + 1]
    return i + 1
```

1 usage

```
def find_max_visitors(self) -> int:
    events = []
```

Создаем события входа и выхода.

```
for interval in self.intervals:
    start_time = interval[0]
    end_time = interval[1]
    events.append((start_time, "enter"))
    events.append((end_time, "exit"))
```

Сортируем события по времени, а затем по типу события (enter перед exit)

```
self.__quick_sort(events, low=0, len(events) - 1)
```

```
max_visitors = 0
```

```
current_visitors = 0
```

Проходим по отсортированным событиям

```
for event in events:
    if event[1] == "enter":
        current_visitors += 1
        if current_visitors > max_visitors:
            max_visitors = current_visitors
    elif event[1] == "exit":
        current_visitors -= 1
```

```
return max_visitors
```

```

q_b.py * test_q_b.py  = output.txt  = input.txt
1  from typing import List, Tuple
2  import time
3  import os
4  import psutil
5
6
7  1 usage
8  def get_memory_usage():
9      """Получение используемой памяти в мегабайтах."""
10     process = psutil.Process(os.getpid())
11     mem_info = process.memory_info()
12     return mem_info.rss / (1024 * 1024)
13
14  1 usage
15  def read_file(filename: str) -> Tuple[int, List[List[Tuple[int, int]]]]:
16      """Чтение данных из файла."""
17      with open(filename, "r") as file:
18          lines = file.readlines()
19
20      cleaned_lines = []
21      for line in lines:
22          cleaned_line = line.strip()
23          cleaned_lines.append(cleaned_line)
24
25      lines = cleaned_lines
26
27      num_records = int(lines[0]) # Первая строка содержит количество записей.
28
29      time_intervals = lines[1:] # Остальные строки содержат пары временных интервалов.
30
31      intervals = []
32      for interval in time_intervals:
33          split_interval = interval.split()
34          time_pairs = []
35          for time_l in split_interval:
36              time_parts = time_l.split(':')
37              hours = int(time_parts[0])
38              minutes = int(time_parts[1])
39              time_pairs.append((hours, minutes))
40          intervals.append(time_pairs)
41
42      return num_records, intervals

```

```
5
=====
    Время выполнения: 0.00067 секунд.
    Использовано памяти: 15.84 Mb.
=====

Process finished with exit code 0
```

Тесты

```

1  import unittest
2  from q_8 import MuseumVisitors
3
4
5  class TestCase(unittest.TestCase):
6      def test_max_visitors_1(self):
7          """1. Тестирование на максимальное количество посетителей."""
8          intervals = [
9              [(10, 0), (12, 0)],
10             [(9, 0), (11, 0)],
11             [(11, 0), (13, 0)]
12         ]
13         museum = MuseumVisitors(intervals)
14         max_visitors = museum.find_max_visitors()
15         self.assertEqual(max_visitors, second: 3)
16
17     def test_max_visitors_2(self):
18         """2. Тестирование на максимальное количество посетителей."""
19         intervals = [
20             [(9, 0), (10, 7)],
21             [(10, 20), (11, 35)],
22             [(12, 0), (17, 0)],
23             [(11, 0), (11, 30)],
24             [(11, 20), (12, 30)],
25             [(11, 30), (18, 15)],
26             [(11, 50), (12, 15)],
27             [(11, 30), (18, 0)]
28         ]
29         museum = MuseumVisitors(intervals)
30         max_visitors = museum.find_max_visitors()
31         self.assertEqual(max_visitors, second: 5)
32
33     def test_max_visitors_3(self):
34         """3. Тестирование на максимальное количество посетителей."""
35         intervals = [
36             [(9, 0), (10, 7)],
37             [(10, 20), (11, 35)],
38             [(12, 0), (17, 0)],
39             [(11, 0), (11, 30)],
40             [(11, 20), (12, 30)],
41             [(11, 30), (18, 15)],
42         ]
43         museum = MuseumVisitors(intervals)
44         max_visitors = museum.find_max_visitors()
45         self.assertEqual(max_visitors, second: 4)
46
47     def test_max_visitors_4(self):
48         """4. Тестирование на максимальное количество посетителей."""
49         intervals = [
50             [(9, 0), (9, 30)],
51             [(10, 0), (12, 0)],
52             [(9, 0), (11, 0)]
53         ]
54         museum = MuseumVisitors(intervals)
55         max_visitors = museum.find_max_visitors()
56         self.assertEqual(max_visitors, second: 2)
57

```

```
Ran 4 tests in 0.001s

OK

Process finished with exit code 0
```

Ход решения

Читаем входные данные. Класс `MuseumVisitors` инициализируется с помощью конструктора, который принимает список интервалов времени посещения музея. Эти интервалы представлены в виде списка списков кортежей, где каждый кортеж содержит пару значений (часы, минуты), обозначающих начало и конец посещения. В основу решения задачи заложен модифицированный алгоритм быстрой сортировки. События сортируются по времени и по состоянию. После сортировки событий метод проходит по отсортированному списку и подсчитывает текущее количество посетителей в музее. Для этого используются две переменные: `max_visitors` и `current_visitors`. Переменная `max_visitors` хранит максимальное количество посетителей, которое было в музее одновременно, а `current_visitors` хранит текущее количество посетителей. Когда метод встречает событие входа (`enter`), он увеличивает `current_visitors` на 1 и проверяет, не превысило ли текущее количество посетителей максимальное значение. Если да, то `max_visitors` обновляется. Когда метод встречает событие выхода (`exit`), он уменьшает `current_visitors` на 1.

Т.е метод создает список событий, где каждое событие представлено кортежем, содержащим время и тип события (`enter` или `exit`). Например, если интервал времени посещения — с 10:00 до 12:00, то будут созданы два события: (10:00, "enter") и (12:00, "exit"). Метод `__partition` выбирает опорный элемент (`pivot`) и разделяет список на две части: элементы, меньшие или равные опорному элементу, и элементы, большие опорного элемента. В нашем случае, опорный элемент — это последний элемент в текущем подсписке.

Метод проходит по списку и перемещает элементы, меньшие или равные опорному элементу, в начало списка, а элементы, большие опорного элемента, - в конец списка. При этом события входа (`enter`) всегда предшествуют событиям выхода (`exit`) при одинаковом времени.

Счетчик пар (посетителей) в данном алгоритме работает путем подсчета текущего количества посетителей на основе отсортированных событий входа и выхода.

Дополнительно

Если видно плохо или хочется протестировать, то можете посмотреть мой код по заданию тут: https://github.com/ytkinroman/lab_2_complexity/tree/main/quest_8

Задача 9. Охрана.

Ограничение по времени: 1 с. Ограничение по памяти: 16 Mb.

На секретной военной базе работает N охранников. Сутки поделены на 10000 равных промежутков времени, и известно когда каждый из охранников приходит на дежурство и уходит с него. Например, если охранник приходит в 5, а уходит в 8, то значит, что он был в 6, 7 и 8-ой промежутке. В связи с уменьшением финансирования часть охранников решено было сократить. Укажите: верно ли то, что для данного набора охранников, объект охраняется в любой момент времени хотя бы одним охранником и удаление любого из них приводит к появлению промежутка времени, когда объект не охраняется.

Формат входных данных:

В первой строке входного файла INPUT.TXT записано натуральное число K ($1 \leq K \leq 30$) – количество тестов в файле. Каждый тест начинается с числа N ($1 \leq N \leq 10000$), за которым следует N пар неотрицательных целых чисел A и B - время прихода на дежурство и ухода ($0 \leq A < B \leq 10000$) соответствующего охранника. Все числа во входном файле разделены пробелами и/или переводами строки.

Формат выходных данных:

В выходной файл OUTPUT.TXT выведите K строк, где в M -ой строке находится слово Accepted, если M -ый набор охранников удовлетворяет описанным выше условиям. В противном случае выведите Wrong Answer.

Код

```
1 from typing import List
2 import time
3 import os
4 import psutil
5
6
7 1 usage
8 def get_memory_usage():
9     """Получение используемой памяти в мегабайтах."""
10    process = psutil.Process(os.getpid())
11    mem_info = process.memory_info()
12    return mem_info.rss / (1024 * 1024)
13
14 1 usage
```

```

14  class QuickSort:
15      def __init__(self, array: List[int]) -> None:
16          self.array = array
17
18      3 usages
19      def __quick_sort(self, low: int, high: int) -> None:
20          if low < high:
21              mid = self.__partition(low, high)
22              self.__quick_sort(low, mid - 1)
23              self.__quick_sort(mid + 1, high)
24
25      1 usage
26      def __partition(self, low: int, high: int) -> int:
27          pivot = self.array[high]
28          i = low - 1
29          for j in range(low, high):
30              if self.array[j] <= pivot:
31                  i += 1
32                  self.array[i], self.array[j] = self.array[j], self.array[i]
33          self.array[i + 1], self.array[high] = self.array[high], self.array[i + 1]
34          return i + 1
35
36      1 usage
37      def sort(self) -> List[int]:
38          self.__quick_sort(low: 0, len(self.array) - 1)
39          return self.array

```



```

1 usage
39 class SecurityTestChecker:
40     def __init__(self, test_count, test_lines):
41         self.test_count = test_count
42         self.test_lines = test_lines
43         self.results = [""] * test_count
44
45     1 usage
46     def __parse_line(self, line):
47         parts = line.split()
48
49         numbers = []
50         for part in parts:
51             number = int(part)
52             numbers.append(number)
53
54         return numbers
55
56     1 usage
57     def __process_events(self, numbers):
58         segment_count = numbers[0]
59         events = [0] * (2 * segment_count)
60         for i in range(1, len(numbers), 2):
61             events[i-1] = (numbers[i], -1, i)
62             events[i] = (numbers[i + 1], 1, i)
63         events = QuickSort(events).sort()

```

```

1 usage
64 def __check_segments(self, events, segment_count):
65     good_segments = []
66     current_segments = []
67     is_good = True
68     previous_time = -1
69
70     for event in events:
71         if event[0] != 0 and len(current_segments) == 0:
72             is_good = False
73             break
74         if len(current_segments) == 1 and event[0] != previous_time:
75             if current_segments[0] not in good_segments:
76                 good_segments.append(current_segments[0])
77         if event[1] == -1:
78             current_segments.append(event[2])
79         else:
80             current_segments.remove(event[2])
81         previous_time = event[0]
82
83     if events[-1][0] != 10000:
84         is_good = False
85
86     return is_good and len(good_segments) == segment_count
87
1 usage
88 def run_tests(self):
89     for test_idx, line in enumerate(self.test_lines):
90         numbers = self.__parse_line(line)
91         events = self.__process_events(numbers)
92         if self.__check_segments(events, numbers[0]):
93             self.results[test_idx] = "Accepted"
94         else:
95             self.results[test_idx] = "Wrong Answer"
96     return "\n".join(self.results)
97
98

```

```

99  ▢ if __name__ == "__main__":
100
101      # Запуск таймера.
102      start_time = time.perf_counter()
103
104      with open("input.txt", "r") as file:
105          lines = file.readlines()
106          test_count = int(lines[0])
107
108      checker = SecurityTestChecker(test_count, lines[1:])
109      result = checker.run_tests()
110
111      with open("output.txt", "w") as file:
112          file.write(result)
113
114      stop_time = time.perf_counter()
115
116      # Вывод времени выполнения.
117      print("=" * 42)
118      print(f"    Время выполнения: {stop_time - start_time:0.5f} секунд.")
119      # Получение и вывод используемой памяти.
120      final_memory = get_memory_usage()
121      print(f"    Использовано памяти: {final_memory:.2f} Mb.")
122      print("=" * 42)

```

```

D:\PycharmProjects\algorithms_2\.venv\Scripts\
=====
    Время выполнения: 0.00074 секунд.
    Использовано памяти: 15.61 Mb.
=====

```

Тесты

```

1 import unittest
2 from q_9 import QuickSort, SecurityTestChecker
3
4
5 class TestQuickSort(unittest.TestCase):
6     def test_sort_empty_array(self):
7         qs = QuickSort([])
8         self.assertEqual(qs.sort(), second: [])
9
10    def test_sort_single_element(self):
11        qs = QuickSort([5])
12        self.assertEqual(qs.sort(), second: [5])
13
14    def test_sort_sorted_array(self):
15        qs = QuickSort([1, 2, 3, 4, 5])
16        self.assertEqual(qs.sort(), second: [1, 2, 3, 4, 5])
17
18    def test_sort_reverse_sorted_array(self):
19        qs = QuickSort([5, 4, 3, 2, 1])
20        self.assertEqual(qs.sort(), second: [1, 2, 3, 4, 5])
21
22    def test_sort_random_array(self):
23        qs = QuickSort([3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5])
24        self.assertEqual(qs.sort(), second: [1, 1, 2, 3, 3, 4, 5, 5, 5, 6, 9])
25
26    def test_run_tests_1(self):
27        lines = ["2", "3 0 3000 2500 7000 2700 10000", "2 0 3000 2700 10000"]
28        checker = SecurityTestChecker(test_count: 2, lines[1:])
29        result = checker.run_tests()
30        self.assertEqual(result, second: "Wrong Answer\nAccepted")
31
32    def test_run_tests_2(self):
33        lines = ["1", "2 0 3000 2700 9000"]
34        checker = SecurityTestChecker(test_count: 1, lines[1:])
35        result = checker.run_tests()
36        self.assertEqual(result, second: "Wrong Answer")
37
38    def test_run_tests_3(self):
39        lines = ["1", "3 0 3000 2500 7000 2700 10000"]
40        checker = SecurityTestChecker(test_count: 1, lines[1:])
41        result = checker.run_tests()
42        self.assertEqual(result, second: "Wrong Answer")
43
44    def test_run_tests_4(self):
45        lines = ["1", "3 0 3000 2500 7000 2700 10000"]
46        checker = SecurityTestChecker(test_count: 1, lines[1:])
47        result = checker.run_tests()
48        self.assertEqual(result, second: "Wrong Answer")

```

Ran 9 tests in 0.002s

OK

Ход решения

Проверка охранников осуществляется с помощью класса SecurityTestChecker. Этот класс обрабатывает входные данные, представляющие собой сегменты времени, в течение которых охранники находятся на своих постах. Каждый сегмент времени представлен парой чисел: начальным и конечным временем.

Метод __process_events обрабатывает события, связанные с началом и концом сегментов времени. Он создает список событий, где каждое событие представлено кортежем, содержащим время события, тип события (-1 для начала сегмента и 1 для конца сегмента) и индекс сегмента. Эти события затем сортируются с помощью алгоритма быстрой сортировки, реализованного в классе QuickSort.

Метод __check_segments проверяет, соответствуют ли сегменты времени определенным условиям. Он проходит по отсортированному списку событий и проверяет, что в каждый момент времени не более одного охранника находится на посту. Если в какой-то момент времени на посту нет ни одного охранника или на посту находятся два охранника одновременно, тест считается не пройденным.

Дополнительно

Если видно плохо или хочется протестировать, то можете посмотреть мой код по заданию тут: https://github.com/ytkinroman/lab_2_complexity/tree/main/quest_9