САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ
ФАКУЛЬТЕТ ИНФОКОММУНИКАЦИОННЫХ ТЕХНОЛОГИЙ

Отчет по лабораторной работе №1
по курсу «Алгоритмы и структуры данных»
Тема: Жадные алгоритмы. Динамическое программирование

Выполнил:
Малыхин Н.С.
К3141

Санкт-Петербург 2025

Содержание

Задачи

Задание №2. Заправки (0.5 балла)

Вы собираетесь поехать в другой город, расположенный в d км от вашего родного города. Ваш автомобиль может проехать не более m км на полном баке, и вы начинаете с полным баком. По пути есть заправочные станции на расстояниях stop1, stop2, ..., stopn из вашего родного города. Какое минимальное количество заправок необходимо?

- Формат ввода / входного файла (input.txt). В первой строке содержится d - целое число. Во второй строке - целое число m. В третьей находится количество заправок на пути - n. И, наконец, в последней строке - целые числа через пробел - остановки stop1, stop2, ..., stopn.
- Ограничения на входные данные. $1 \le d \le 105$, $1 \le m \le 400$, $1 \le n \le 300$, $1 < stop1 < stop2 < ... < stopn < d$
- Формат вывода / выходного файла (output.txt). Предполагая, что расстояние между городами составляет d км, автомобиль может проехать не более m км на полном баке, а заправки есть на расстояниях stop1, stop2, ..., stopn по пути, выведите минимально необходимое количество заправок. Предположим, что машина начинает ехать с полным баком. Если до места назначения добраться невозможно, выведите -1.

Решение:

```
from utils.file_utlis import read_from_file, write_to_file
from utils.time_memory_utlis import measure_time_and_memory



input_path = "../txtf/input.txt"
output_path = "../txtf/output.txt"



@measure_time_and_memory
def min_refuels(d, m, n, stops):
```

```
    stops += [d] # Добавляем точку назначения в конец списка
    refuels = 0 # Счетчик заправок
    last_refuel = 0 # Позиция последней заправки

    # Проходим по всем заправочным станциям
    for i in range(n):
        # Проверяем, сможем ли мы доехать до следующей станции
        if stops[i+1] - last_refuel > m:
            # Если не можем, проверяем, можем ли доехать до текущей станции
            if stops[i] - last_refuel <= m:
                # Если можем, заправляемся здесь
                refuels += 1
                last_refuel = stops[i]
            else:
                # Если не можем доехать даже до текущей станции, то маршрут невозможен
                return -1

    return refuels


def main():
    # Чтение входных данных из файла
    input_data = read_from_file(input_path)

    # Извлечение параметров из входных данных
    d = input_data[0]
    m = input_data[1]
    n = input_data[2]
    stops = input_data[3]

    # Вычисление результата
    result = min_refuels(d, m, n, stops)

    # Запись результата в выходной файл
    output_data = str(result)
    write_to_file(output_path, output_data)


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

Объяснение:

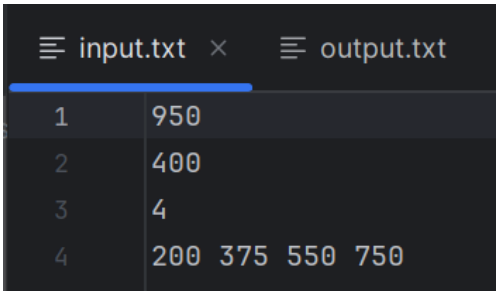
Алгоритм использует жадный подход - каждый раз заправляемся на самой дальней возможной заправке, чтобы минимизировать количество остановок. Если на каком-то этапе оказывается, что до следующей заправки не дотянуть даже с полным баком - значит, маршрут невозможен.

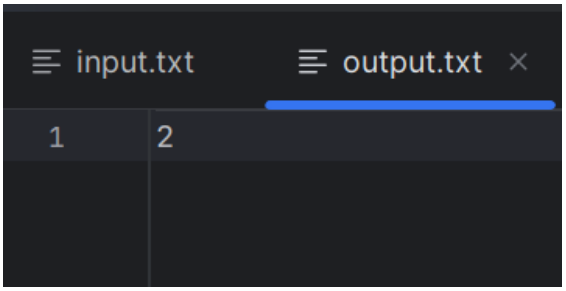
1. Сначала добавляем сам город назначения (d) в список заправок, чтобы он был последней точкой маршрута.
2. Заводим три переменные:
    - count - счётчик заправок (изначально 0)
    - last - где последний раз заправлялись (сначала это 0 - старт)
    - current - где сейчас находимся (тоже сначала 0)
3. Проходим по всем заправкам по порядку:
    - Если от последней заправки до следующей точки можно доехать без дозаправки - просто едем дальше
    - Если не можем - проверяем, можем ли доехать до текущей заправки:
        - Если можем - заправляемся здесь (увеличиваем count, запоминаем эту заправку как last)
        - Если не можем - значит, маршрут невозможен (возвращаем -1)
4. В конце проверяем, сможем ли доехать от последней заправки до города. Если да - возвращаем count, если нет - -1.

Сложность:

Алгоритм проходит по всем заправкам один раз, поэтому работает за O(n), где n - количество заправок. Это оптимальное решение.

Результат работы кода:





Задание №6. Максимальная зарплата (0.5 балла)

В качестве последнего вопроса успешного собеседования ваш начальник дает вам несколько листков бумаги с цифрами и просит составить из этих цифр наибольшее число. Полученное число будет вашей зарплатой, поэтому вы очень заинтересованы в максимизации этого числа. Как вы можете это сделать? На лекциях мы рассмотрели следующий алгоритм составления наибольшего числа из заданных однозначных чисел.

```
def LargestNumber(Digits):
    answer = ''
    while Digits:
        maxDigit = float('-inf')
        for digit in Digits:
            if digit >= maxDigit:
                maxDigit = digit
            answer += str(maxDigit)
        Digits.remove(maxDigit)
    return answer
```

К сожалению, этот алгоритм работает только в том случае, если вход состоит из однозначных чисел. Например, для ввода, состоящего из двух целых чисел 23 и 3 (23 не однозначное число!) возвращается 233, в то время как наибольшее число на самом деле равно 323. Другими словами, использование наибольшего числа из входных данных в качестве первого числа не является безопасным ходом. Ваша цель в этой задаче – настроить

описанный выше алгоритм так, чтобы он работал не только с однозначными числами, но и с произвольными положительными целыми числами.

- Постановка задачи. Составить наибольшее число из набора целых чисел.
- Формат ввода / входного файла (input.txt). Первая строка входных данных содержит целое число n. Во второй строке даны целые числа a1, a2, ..., an.
- Ограничения на входные данные. $1 \le n \le 102$, $1 \le ai \le 103$ для всех $1 \le i \le n$.
- Формат вывода / выходного файла (output.txt). Выведите

Решение:

```
from utils.file_utlis import read_from_file, write_to_file
from utils.time_memory_utlis import measure_time_and_memory



input_path = "../txtf/input.txt"
output_path = "../txtf/output.txt"



@measure_time_and_memory
def largest_number(nums):
  # Преобразуем числа в строки
  nums = list(map(str, nums))

  # Сортировка: умножаем строковое представление числа на 4 и сортируем в обратном порядке
  nums.sort(key=lambda x: x * 4, reverse=True)

  # Объединяем отсортированные числа в строку
  max_num = "".join(nums)

  # Если все числа нули
  return max_num if max_num[0] != "0" else "0"



def main():
```

```
    # Чтение входных данных из файла
    input_data = read_from_file(input_path)

    # Извлечение параметров из входных данных
    n = input_data[0]
    nums = input_data[1]

    # Вычисление результата
    result = largest_number(nums)

    # Запись результата в выходной файл
    output_data = result
    write_to_file(output_path, output_data)


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

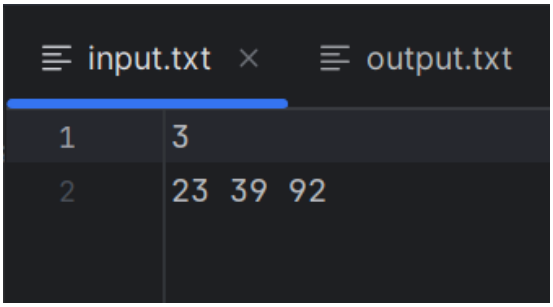
Объяснение:

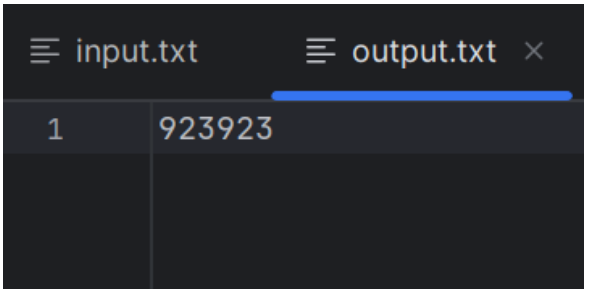
1. Преобразуем все числа в строки для удобства сравнения и конкатенации.
2. Сортируем числа не по их числовому значению, а по их "влиянию" на итоговое число при конкатенации. Для этого:
    - Умножаем строковое представление каждого числа на 4 (чтобы гарантировать достаточную длину для сравнения).
    - Сортируем в обратном порядке (от большего к меньшему).
3. Объединяем отсортированные строки в одну.
4. Проверяем случай, когда все числа нули (чтобы не выводить строку из нулей, а просто "0").

Сложность алгоритма:

- Время: O(n log n) из-за сортировки.
- Память: O(n) для хранения строковых представлений чисел.

Результат работы кода:





Задание №8. Расписание лекций (1 балл)

- Постановка задачи. Вы наверно знаете, что в ИТМО лекции читают одни из лучших преподаватели мира. К сожалению, лекционных аудиторий у нас не так уж и много, особенно на Биржевой, поэтому каждый преподаватель составил список лекций, которые он хочет прочитать студентам. Чтобы студенты, в начале февраля, увидели расписание лекций, необходимо его составить прямо сейчас. И без вас нам здесь не справиться. У нас есть список заявок от преподавателей на лекции для одной из аудиторий. Каждая заявка представлена в виде временного интервала [si, fi) - время начала и конца лекции. Лекция считается открытым интервалом, то есть какая-то лекция может начаться в момент окончания другой, без перерыва. Необходимо выбрать из этих заявок такое подмножество, чтобы суммарно выполнить максимальное количество заявок. Учтите, что одновременно в лекционной аудитории, конечно же, может читаться лишь одна лекция.
- Формат ввода / входного файла (input.txt). В первой строке вводится натуральное число N - общее количество заявок на лекции. Затем вводится N строк с описаниями заявок - по два числа в каждом si и fi для каждой лекции i. Гарантируется, что si < fi
- Время начала и окончания лекции - натуральные числа, не превышают 1440 (в минутах с начала суток).
- Ограничения на входные данные. $1 \le N \le 1000$, $1 \le si, fi \le 1440$

- Формат вывода / выходного файла (output.txt). Выведите одно число максимальное количество заявок на проведение лекций, которые можно выполнить.

Решение:

```
from utils.file_utlis import read_from_file, write_to_file
from utils.time_memory_utlis import measure_time_and_memory


input_path = "../txtf/input.txt"
output_path = "../txtf/output.txt"


@measure_time_and_memory
def max_lectures(requests):
  # Сортируем лекции по времени окончания
  requests.sort(key=lambda x: x[1])

  c = 0 # Счетчик выбранных лекций
  last_end = 0 # Время окончания последней выбранной лекции

  for start, end in requests:
    # Если текущая лекция начинается после окончания последней выбранной
    if start >= last_end:
      c += 1 # Увеличиваем счетчик
      last_end = end # Обновляем время окончания последней лекции
  return c


def main():
  # Чтение входных данных из файла
  input_data = read_from_file(input_path)

  # Извлечение параметров из входных данных
  n = input_data[0]
  requests = input_data[1:]

  # Вычисление результата
  result = max_lectures(requests)

  # Запись результата в выходной файл
```

```
    output_data = str(result)
    write_to_file(output_path, output_data)


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

Объяснение:

1. Сортируем все лекции по времени окончания (по f_i)
2. Последовательно выбираем лекции:
    - Первой берем лекцию с самым ранним окончанием
    - Каждую следующую выбираем, если она начинается не раньше окончания предыдущей выбранной

Сложность:

- Сортировка: O(n log n)
- Основной цикл: O(n)
- Общая сложность: O(n log n)

Результат работы кода:

Задание №15. Удаление скобок (2 балла)

- Постановка задачи. Дана строка, составленная из круглых, квадратных и фигурных скобок. Определите, какое наименьшее количество символов необходимо удалить из этой строки, чтобы

оставшиеся символы образовывали правильную скобочную последовательность.

- Формат ввода / входного файла (input.txt). Во входном файле записана строка, состоящая из s символов: круглых, квадратных и фигурных скобок (), [], {}. Длина строки не превосходит 100 символов.
- Ограничения на входные данные. $1 \leq s \leq 100$.
- Формат вывода / выходного файла (output.txt). Выведите строку максимальной длины, являющейся правильной скобочной последовательностью, которую можно получить из исходной строки удалением некоторых символов.

Решение:

```
from utils.file_utlis import read_from_file, write_to_file
from utils.time_memory_utlis import measure_time_and_memory



input_path = "../txtf/input.txt"
output_path = "../txtf/output.txt"



@measure_time_and_memory
def remove_brackets(s):
  # Открывающие скобки соответствующие закрывающим
  pairs = {
    ")": "(",
    "]": "[",
    "}": "{"
  }
  stack = [] # Стек для хранения индексов открывающих скобок
  remove = set() # Множество для хранения индексов скобок, которые нужно удалить

  for i, char in enumerate(s):
    if char in '([{': # Если это открывающая скобка, добавляем ее индекс в стек
      stack.append(i)
    elif char in ')]}':
      if stack and s[stack[-1]] == pairs[char]: # Если пара найдена
        stack.pop() # Удаляем ее из стека
```

```
            else:
                remove.add(i) # Помечаем закрывающую скобку для удаления

    # Открывающие скобки не имеющие пары помечаем для удаления
    remove.update(stack)

    # Исключаем помеченные для удаления скобки
    result = ''.join(s[i] for i in range(len(s)) if i not in remove)
    return result


def main():
    # Чтение входных данных из файла
    input_data = read_from_file(input_path)

    # Извлечение параметров из входных данных
    s = input_data

    # Вычисление результата
    result = remove_brackets(s)

    # Запись результата в выходной файл
    output_data = result
    write_to_file(output_path, output_data)


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

Объяснение:

Создаем словарь pairs, который хранит соответствие закрывающих скобок открывающим

Инициализируем стек для хранения индексов открывающих скобок и множество для индексов скобок, подлежащих удалению.

Проходим по строке:

- При встрече открывающей скобки **(**, **[** , **{** добавляем ее индекс в стек.
- При встрече закрывающей скобки проверяем:

- Если стек не пуст и верхний элемент стека соответствует открывающей скобке текущего типа - удаляем пару из стека.
- Иначе - добавляем индекс закрывающей скобки в множество на удаление.

После прохода все оставшиеся в стеке индексы (непарные открывающие скобки) добавляем в множество на удаление.

Формируем итоговую строку, исключая все помеченные для удаления скобки.

Сложность: O(n)

Результат работы кода:

Вывод

Все задачи решены через оптимальные алгоритмы:

- Жадные методы - для заправок, чисел и расписаний
- Стек - для скобок

Работают быстро (O(n) или O(n log n)) и экономно по памяти