| МИНОБРНАУКИ РОССИИ | | |
| --- | --- | --- |
| Федеральное государственное автономное  образовательное учреждение высшего образования  «Пермский государственный национальный  исследовательский университет» | | |
|  | Институт компьютерных наук и технологий | |
| **ОТЧЁТ**  по индивидуальной работе №2  по дисциплине «Язык программирования Python»  Вариант 2 | | |
|  | | Работу выполнил  студент группы ИТ-5-2024 1 курса  Шутов С.А  «12» июня 2025 г. |
| Работу проверил  \_\_\_\_\_\_\_\_\_\_\_\_\_\_ Фамилия И.О.  «\_\_\_» \_\_\_\_\_\_\_\_\_ 2025 г. |
| Пермь 2024 | | |

**СОДЕРЖАНИЕ**

[Постановка задачи 3](#_m1b30i7a8e73)

[Алгоритм решения 3](#_afeyqu8cz5ke)

[Тестирование 3](#_j59pgw98e7eq)

[Код программы 3](#_af8hh428f120)

# Постановка задачи

# Игра в слова. Дан набор слов. Известно, что эти слова использовались для игры, в которой очередное слово должно заканчиваться той буквой, на которую закончилось предыдущее слово. С какого слова начинается игра, не имеет значения. Если слово оканчивается на «ь», то используется предыдущая буква. Последнее слово цепочки должно заканчиваться буквой первого слова цепочки. Входной файл содержит все слова в одной строке, слова записаны строчными буквами русского алфавита, между словами может быть один или более пробелов. Требуется выстроить данные слова в цепочку согласно правилам игры. Достаточно найти одно подходящее решение. Возможен вариант, что решений не существует.

# Алгоритм решения

**1. Постановка задачи** Дан набор слов, изначально записанных в одну строку через пробелы. Требуется упорядочить их в «цепочку» так, чтобы каждое последующее слово начиналось на ту букву, на которую окончится предыдущее. Если слово оканчивается на мягкий знак «ь», считается пред­последняя буква. Начать можно с любого слова, но при этом последнее слово должно заканчиваться на первую букву первого. Если такого порядка не существует, сообщить об этом.

**2. Ключевая идея** – Каждое слово представляем как ориентированное ребро: из вершины-буквы, равной первой букве, в вершину-букву, равную последнему значимому символу.  
 – Задача превращается в поиск цикла, проходящего по всем рёбрам ровно один раз (эйлеров цикл).  
 – На практике, поскольку граф небольшой, достаточно выполнить глубинный поиск (DFS) с учётом использованных слов и вернуть первую найденную замкнутую последовательность.

**3. Подробный алгоритм**

1. **Разбор входных данных**
   * Считываем строку, разбиваем по пробелам, фильтруем пустые токены.
   * Проверяем: есть ли хотя бы одно слово.
2. **Определение значимых букв**
   * Для каждого слова word:  
      return word[-2] if word.endswith('ь') and len(word)>1 else word[-1]
3. **Построение графа**
   * Ключ словаря — первая буква слова,
   * значение — список индексов всех слов, начинающихся с этой буквы.
   * Используем setdefault, чтобы создать пустой список, если ключа ещё нет.

graph: dict[str, list[int]] = {}

for i, word in enumerate(words):

start = word[0]

graph.setdefault(start, []).append(i)

1. **Поиск цепочки (DFS + бэктрекинг**
2. Старт цикла по возможным начальным индексам

def find\_chain() -> list[str] | None:

for i in range(len(words)): # пробуем каждое слово первым

path, used = [], set()

if dfs(i, path, used): # если удалось построить цикл

return [words[j] for j in path]

return None

1. Рекурсивная функция dfs

def dfs(index: int, path: list[int], used: set[int]) -> bool:

# добавить текущее слово

path.append(index)

used.add(index)

# базовый случай: все слова использованы

if len(path) == len(words):

return last\_letter(words[path[-1]]) == words[path[0]][0]

# берём букву, на которую заканчивается текущее слово

next\_letter = last\_letter(words[index])

# перебираем все слова, начинающиеся на эту букву

for j in graph.get(next\_letter, []):

if j not in used: # ещё не брали

if dfs(j, path, used): # рекурсивный шаг

return True # найден цикл

# тупик → откат

path.pop()

used.remove(index)

return False # неудача по этой ветке

**4. Сложность**

* **Временная:** в худшем случае при полном переборе путей DFS получается экспоненциальная сложность, примерно O(k·n!), где n — количество слов, k — среднее число вариантов продолжения на каждом шаге. На практике для n≲15 алгоритм работает быстро, поскольку большая часть ветвей отсеивается при проверке использованных слов и замкнутости цикла.
* **Память:** O(n) для хранения графа

**5. Итог** Алгоритм для «Игры в слова» использует только примитивные операции языка (строки, списки, множества, простая арифметика индексов) и собственные структуры данных (словарь для графа и рекурсивный вызов вместо сторонних библиотек). Решение универсально, его легко обосновать через DFS: на каждом шаге текущий путь корректен и не содержит повторений, а проверка замкнутости гарантирует выполнение условий задачи.

# Тестирование

| № | Входная строка | Ожидаемый результат | Что проверяем |
| --- | --- | --- | --- |
| 1 | хмель мороз налог лимон запах дым лошадь холм гол молох налог гол лошадь дым мороз запах холм молох хмель лимон | налог гол лошадь дым мороз запах холм молох хмель лимон | Большой набор, цепочка существует |
| 2 | а б в   |  | | --- |  |  | | --- | | Цепочку составить невозможно.   |  | | --- |  |  | | --- | | Нет пути   |  | | --- |  |  | | --- | |
| 3 | лошадь даль   |  | | --- |  |  | | --- | | лошадь даль   |  | | --- |  |  | | --- | | Учёт «ь»   |  | | --- |  |  | | --- | |
| 4 | дом | Цепочку составить невозможно.   |  | | --- |  |  | | --- | | Одиночное слово, конец ≠ начало   |  | | --- |  |  | | --- | |
| 5 | токот   |  | | --- |  |  | | --- | | токот | Одиночное слово, конец = начало   |  | | --- |  |  | | --- | |
| 6 | абв гва авг   |  | | --- |  |  | | --- | | абв гва авг   |  | | --- |  |  | | --- | | Убираются лишние пробелы |
| 7 | а а а   |  | | --- |  |  | | --- | | а а а   |  | | --- |  |  | | --- | | Повторы слов   |  | | --- |  |  | | --- | |
| 8 |  | Ошибка: не введено ни одного слова.   |  | | --- |  |  | | --- | | Пустая строка   |  | | --- |  |  | | --- | |
| 9 | рак кар   |  | | --- |  |  | | --- | | рак кар   |  | | --- |  |  | | --- | | Минимальная цепочка из двух слов   |  | | --- |  |  | | --- | |
| 10 | мать тьма ам   |  | | --- |  |  | | --- | | мать тьма ам   |  | | --- |  |  | | --- | | «ь» в двух словах подряд   |  | | --- |  |  | | --- | |

# Код программы

[ссылка на гитхаб](https://github.com/SaveliyShutov/IKM_ShutovSaveliy)