Листинг кода

```
from abc import ABC
from dataclasses import dataclass
from typing import List, Any, Callable, Optional
from collections import defaultdict
from functools import wraps


class DatabaseEntity(ABC):
    last_id = 1
    id: int

    def __init__(self):
        self.id = self.__class__.last_id
        self.__class__.last_id += 1


@dataclass
class Language(DatabaseEntity):
    name: str
    version: str

    def __post_init__(self):
        super().__init__()


@dataclass
class IDE(DatabaseEntity):
    name: str
    language_id: Optional[int]

    def __post_init__(self):
        super().__init__()


@dataclass
class LanguageIDE(DatabaseEntity):
    language_id: int
    IDE_id: int

    def __post_init__(self):
        super().__init__()


# алгоритм нечеткого поиска
def find_levenshtein_distance(str1: str, str2: str):
    dp = [[0 for j in range(len(str1) + 1)] for i in range(len(str2) + 1)]
    for j in range(len(dp[0])):
        dp[0][j] = j
    for i in range(len(dp)):
```

```
        dp[i][0] = i

    for i in range(1, len(dp)):
        for j in range(1, len(dp[0])):
            cost = 0
            if str1[j - 1] != str2[i - 1]:
                cost = 1
            dp[i][j] = min(dp[i - 1][j] + 1, dp[i][j - 1] + 1, dp[i - 1][j - 1] +
cost)
    return dp[len(str2)][len(str1)]


# декоратор для печати результата запроса (отчет по запросу)
def print_query(title: str = "Результат запроса", *column_titles):
    def decorator(func: Callable) -> Callable:
        @wraps(func)
        def wrapper(*args, **kwargs):
            print(title)

            result = func(*args, **kwargs)

            column_widths = [0] * len(column_titles)

            for i in range(len(column_titles)):
                max_width = len(column_titles[i])
                for key, values in result:
                    if not isinstance(values, list):
                        values = [values]
                    for value in values:
                        max_width = max(max_width, len(key), len(values))
                column_widths[i] = max_width + 2

            header_parts = [
                f"{column_titles[i]:<{column_widths[i]}}"
                for i in range((len(column_widths)))
            ]
            header_line = " | ".join(header_parts)
            print(header_line)

            print("-" * len(header_line))

            for key, values in result:
                width_key = column_widths[0]
                width_values = column_widths[1]

                formatted_key = f"{key:<{width_key}}"
                if not isinstance(values, list):
                    values = [values]

                for value in values:
                    row_line = []
```

```
                    row_line.append(formatted_key)

                    formatted_value = f"{value:<{width_values}}"
                    row_line.append(formatted_value)

                    print(" | ".join(row_line))
            print("-" * len(header_line))
            print("\n")

            return result

        return wrapper

    return decorator


class Database:
    def __init__(self):
        self.languages: List[Language] = []  # список ЯП
        self.IDEs: List[IDE] = []  # списов IDE
        self.lang_ides: List[LanguageIDE] = (
            []
        )  # список классов Язык-IDE (для связи M:M)

        self.language_map = dict()
        self.ides_map = dict()

    def add(self, object: DatabaseEntity) -> None:
        if isinstance(object, Language):
            self.languages.append(object)
            self.language_map[object.id] = object.name
        elif isinstance(object, IDE):
            self.IDEs.append(object)
            self.ides_map[object.id] = object.name
        elif isinstance(object, LanguageIDE):
            self.lang_ides.append(object)
        else:
            print("Error: incorrect entity type")

    # выводит все IDE и связянные с ней ЯП
    @print_query("Запрос 1", "IDE", "ЯП")
    def first_query(self) -> List[Any]:
        query_list = [
            (ide.name, self.language_map[ide.language_id]) for ide in self.IDEs
        ]
        return sorted(query_list, key=lambda item: item[0])

    # выводит ЯП и кол-во связанных с ним IDE
    @print_query("Запрос 2", "ЯП", "Кол-во IDE")
    def second_query(self) -> List[Any]:
```

```
        query_dict = defaultdict(int)
        for ide in self.IDEs:
            query_dict[self.language_map[ide.language_id]] += 1
        return sorted(query_dict.items(), key=lambda item: item[1], reverse=True)

    # выводит IDE, введенную пользователем и все связанные с ним ЯП
    @print_query("Запрос 3", "IDE", "ЯП")
    def third_query(self, ide_name_prefix: str):
        query_dict = defaultdict(list)

        for el in self.lang_ides:
            ide_name = self.ides_map[el.IDE_id]
            levenshtein_distance = find_levenshtein_distance(
                ide_name_prefix.lower(), ide_name.lower()
            )

            similary_ratio = (
                (len(ide_name) + len(ide_name_prefix)) - 2 * levenshtein_distance
            ) / (len(ide_name) + len(ide_name_prefix))

            if similary_ratio >= 0.5:
                query_dict[(ide_name, levenshtein_distance)].append(
                    self.language_map[el.language_id]
                )

        for el in query_dict.values():
            el.sort()

        sorted_list = sorted(
            query_dict.items(), key=lambda item: item[0][1], reverse=True
        )
        return [(el[0][0], el[1]) for el in sorted_list]


test_langs = [
    # ID: 1, 2, 3, 4, 5
    Language("Python", "3.12"),
    Language("Java", "21"),
    Language("Rust", "1.74"),
    Language("Go", "1.21"),
    Language("C#", "12"),
    # ID: 6, 7, 8, 9, 10
    Language("JavaScript", "ES2023"),
    Language("TypeScript", "5.2"),
    Language("Swift", "5.9"),
    Language("Kotlin", "1.9"),
    Language("PHP", "8.3"),
]


def test_one_to_many_queries():
```

```
    database = Database()

    test_ides_for_one_to_many = [
        IDE("PyCharm", 1),  # Python
        IDE("Eclipse", 2),  # Java
        IDE("IntelliJ", 2),  # Java
        IDE("Visual Studio", 5),  # C#
        # Python
        IDE("VS Code", 1),
        IDE("Sublime Text", 1),
        # Java
        IDE("Android Studio", 2),
        IDE("NetBeans", 2),
        # C#
        IDE("Rider", 5),
        IDE("MonoDevelop", 5),
    ]

    for el in test_langs:
        database.add(el)
    for el in test_ides_for_one_to_many:
        database.add(el)

    first = database.first_query()
    second = database.second_query()


def test_many_to_namy_queries():
    IDE.last_id = 1  # сбрасываем ID у средств разработки
    database = Database()

    test_ides_for_many_to_many = [
        # ID: 1, 2, 3, 4, 5
        IDE("PyCharm", None),  # 1
        IDE("IntelliJ IDEA", None),  # 2
        IDE("VS Code", None),  # 3
        IDE("Eclipse", None),  # 4
        IDE("Sublime Text", None),  # 5
        # ID: 6, 7, 8, 9, 10
        IDE("Rider", None),  # 6
        IDE("Xcode", None),  # 7
        IDE("WebStorm", None),  # 8
        IDE("GoLand", None),  # 9
        IDE("Android Studio", None),  # 10
    ]

    test_lang_ides = [
        LanguageIDE(1, 1),  # Python + PyCharm
        LanguageIDE(2, 2),  # Java + IntelliJ
        LanguageIDE(9, 2),  # Kotlin + IntelliJ
        LanguageIDE(1, 3),  # Python + VS Code
```

```
            LanguageIDE(6, 3),  # JavaScript + VS Code
            LanguageIDE(7, 3),  # TypeScript + VS Code
            LanguageIDE(2, 4),  # Java + Eclipse
            LanguageIDE(1, 5),  # Python + Sublime
            LanguageIDE(5, 6),  # C# + Rider
            LanguageIDE(4, 9),  # Go + GoLand
            LanguageIDE(2, 10),  # Java + Android Studio
            LanguageIDE(6, 8),  # JavaScript + WebStorm
        ]

        for el in test_langs:
            database.add(el)
        for el in test_ides_for_many_to_many:
            database.add(el)
        for el in test_lang_ides:
            database.add(el)

        third = database.third_query(input("Введите название IDE: "))


    def main():
        test_one_to_many_queries()
        test_many_to_namy_queries()


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

Результат выполнения программы

```
juve@DESKTOP-HDCAHDS:~/git/BMSTU-CS-2025-Labs-Python/RK$ python3 main.py
Запрос 1
IDE              | ЯП
-----------------------------------
Android Studio   | Java
Eclipse          | Java
IntelliJ         | Java
MonoDevelop      | C#
NetBeans         | Java
PyCharm          | Python
Rider            | C#
Sublime Text     | Python
VS Code          | Python
Visual Studio    | C#
-----------------------------------

Запрос 2
ЯП       | Кол-во IDE
------------------------
Java     | 4
Python   | 3
C#       | 3
------------------------

 Введите название IDE: VS Cpde
 Запрос 3
 IDE       | ЯП
 --------------------
 VS Code   | JavaScript
 VS Code   | Python
 VS Code   | TypeScript
 --------------------

Введите название IDE: webstorm
Запрос 3
IDE         | ЯП
------------------------
WebStorm    | JavaScript
------------------------
```