Міністерство освіти і науки України

Донецький національний університет імені Василя Стуса Факультет інформаційних і прикладних технологій

Кафедра інформаційних технологій

ЗВІТ

з лабораторної роботи № 11

з дисципліни «Основи програмування»

на тему:

«Обробка тривимірних масивів»

Виконав: студент гр. Б25_д/F3

Кручківський Ю.О.

Перевірив: доц. Бабаков Р. М.

Вінниця – 2025

Завдання	
7	вертикальний стовпець із найбільшими трудовитратами (без урахування прибутку)

Код до завдання:

```
import pprint
from pprint import pformat
from abc import ABC, abstractmethod
from enum import Enum, unique, auto
from dataclasses import dataclass
from random import randint


def gen_3d_dimension_massive(rows=10, cols=10, depth=10, min_n=0,
max_n=7):  # оголошуємо функцію генерації тривимірного масиву чисел
    return [  # повертаємо згенерований тривимірний список
        [
            [
                randint(min_n, max_n)  # генеруємо випадкове число в
заданому діапазоні що представляє тип породи
                for _ in range(cols)]  # генеруємо рядок шириною cols
            for _ in range(rows)]  # генеруємо шар висотою rows
        for _ in range(depth)]  # генеруємо глибину depth


class TypeBreed(Enum):  # визначаємо клас тип даних для базових типів
порід
    rock_breed_soft = auto()  # визначаємо тип м’якої породи
    rock_breed_hard = auto()  # визначаємо тип твердої породи
    rock_breed_super_hard = auto()  # визначаємо тип надтвердої породи
    water = auto()  # визначаємо тип води
    oil = auto()  # визначаємо тип нафти
    stone_coal = auto()  # визначаємо тип кам’яного вугілля
    iron_ore = auto()  # визначаємо тип залізної руди
    gold = auto()  # визначаємо тип золота


@dataclass
class SchemeBreed:  # визначаємо клас контейнер для збереження схеми
породи
    type: TypeBreed  # визначаємо поле що зберігає базовий тип породи
    number: int  # визначаємо поле що зберігає числовий ідентифікатор
породи
    profit: int  # визначаємо поле що зберігає прибутковість породи

    def __repr__(self):  # реалізовуємо метод що визначає строкове
представлення схеми породи
        return f"{self.type}"  # повертаємо строкове представлення типу
породи


@unique
class RegisteredBreed(Enum):  # визначаємо клас що реєструє всі доступні
схеми порід
    rock_breed_soft = SchemeBreed(TypeBreed.rock_breed_soft, 0, 0)  #
реєструємо м’яку породу
    rock_breed_hard = SchemeBreed(TypeBreed.rock_breed_hard, 1, 0)  #
реєструємо тверду породу
    rock_breed_super_hard = SchemeBreed(TypeBreed.rock_breed_super_hard,
2, 0)  # реєструємо надтверду породу
```

```
    water = SchemeBreed(TypeBreed.water, 3, 0)  # реєструємо воду
    oil = SchemeBreed(TypeBreed.oil, 4, 1)  # реєструємо нафту
    stone_coal = SchemeBreed(TypeBreed.stone_coal, 5, 1)  # реєструємо
кам'яне вугілля
    iron_ore = SchemeBreed(TypeBreed.iron_ore, 6, 2)  # реєструємо
залізну руду
    gold = SchemeBreed(TypeBreed.gold, 7, 10)  # реєструємо золото


class IBreed(ABC):  # визначаємо інтерфейс для моделі породи
    @property
    @abstractmethod
    def type(self) -> RegisteredBreed:  # визначаємо абстрактний метод що
повертає зареєстрований тип породи
        pass


class Breed(IBreed):  # ініціалізуємо клас модель породи
    def __init__(self, number: int):  # реалізовуємо метод конструктор
для класу
        self.number = number  # зберігаємо числовий ідентифікатор породи
        self._type = self.__assign_breed_scheme(self.number)  #
визначаємо схему породи на основі її номера

    @property
    def type(self) -> RegisteredBreed:  # реалізовуємо метод що повертає
тип породи
        return self._type  # повертаємо зареєстрований тип породи

    @staticmethod
    def __assign_breed_scheme(number: int):  # оголошуємо статичний метод
для визначення схеми породи за номером
        for breed in RegisteredBreed:  # йдемо по всіх зареєстрованих
породах
            if number == breed.value.number:  # якщо номер співпадає
повертаємо відповідну породу
                return breed
        raise ValueError("Breed number out of range")  # якщо номер не
знайдено викидаємо помилку

    def __repr__(self):  # реалізовуємо метод що визначає строкове
представлення об'єкту породи
        return "breed: " + self.type.name + " - " +
str(self.type.value.profit) + " - " + str(self.type.value.number)  #
повертаємо строкове представлення породи


class BreedExploration:  # ініціалізуємо клас дослідження карти порід
    def __init__(self, breed_map: list[list[list]]):  # реалізовуємо
метод конструктор для класу
        self.breed_map = self.__calc_breed_map(breed_map)  # конвертуємо
числову карту у карту об'єктів порід

    @staticmethod
    def __calc_breed_map(breed_3d):  # оголошуємо статичний метод для
конвертації чисел у об'єкти Breed
        return [  # повертаємо новий тривимірний список об'єктів Breed
            [
                [Breed(breed_3d[d][x][y]) for y in
range(len(breed_3d[0][0]))]  # створюємо рядок об'єктів Breed
```

```
                for x in range(len(breed_3d[0]))  # створюємо шар
об'єктів Breed
            ]
            for d in range(len(breed_3d))  # створюємо глибину об'єктів
Breed
        ]

    def __repr__(self):  # реалізовуємо метод що визначає строкове
представлення карти
        return pformat(self.breed_map)  # повертаємо форматовану строку
карти


@dataclass
class BreedColumn:  # визначаємо клас що представляє колонку порід по
глибині
    x: int  # визначаємо координату X колонки
    y: int  # визначаємо координату Y колонки
    breed_list: list[Breed]  # визначаємо список порід у колонці

    @staticmethod
    def __calc_breed_list_profit(breed_list):  # оголошуємо статичний
метод для обчислення прибутку списку порід
        return sum(breed.type.value.profit for breed in breed_list)  #
сумуємо прибуток всіх порід у списку

    @property
    def profit(self):  # визначаємо властивість що повертає загальний
прибуток колонки
        return self.__calc_breed_list_profit(self.breed_list)  #
повертаємо обчислений прибуток

    @property
    def pos(self):  # визначаємо властивість що повертає координати
колонки
        return (self.x, self.y)  # повертаємо координати X та Y

    def __iter__(self):  # реалізовуємо метод ітерації по породах колонки
        return iter(self.breed_list)  # повертаємо ітератор списку порід

    def __repr__(self):
        return f"Cords x: {self.x}, x: {self.y}, profit: {self.profit}"


class BreedDepthExploration(BreedExploration):  # визначаємо клас що
розширює дослідження карти для аналізу колонок
    def __init__(self, breed_map: list[list[list]]):  # реалізовуємо
метод конструктор для класу
        super().__init__(breed_map)  # викликаємо конструктор базового
класу

        self.breed_columns = self.__get_columns()  # обчислюємо всі
колонки карти

    def __get_column(self, x, y):  # оголошуємо інкапсульований метод що
повертає колонку за координатами
        breeds = []  # ініціалізуємо список порід
        for i in range(len(self.breed_map)):  # йдемо по всій глибині
карти
```

```
            breed = self.breed_map[i][x][y]  # отримуємо породу на
поточній глибині
            breeds.append(breed)  # додаємо породу до списку
        return BreedColumn(x,y, breeds)  # повертаємо створену колонку

    def __get_columns(self):  # оголошуємо інкапсульований метод що
повертає всі колонки карти
        column = []  # ініціалізуємо список колонок
        for x in range(len(self.breed_map[0])):  # йдемо по всіх
координатах X
            for y in range(len(self.breed_map[0][0])):  # йдемо по всіх
координатах Y
                column.append(self.__get_column(x, y))  # додаємо колонку
до списку

        return column  # повертаємо список колонок


    def get_best_profit_column(self):  # оголошуємо метод що повертає
колонку з максимальним прибутком
        best_column = self.breed_columns[0]  # ініціалізуємо змінну
найкращої колонки
        for column in self.breed_columns:  # йдемо по всіх колонках
            if column.profit > best_column.profit:  # якщо прибуток
більший за поточний максимум
                best_column = column  # оновлюємо найкращу колонку
        return best_column  # повертаємо колонку з максимальним прибутком


b_m = gen_3d_dimension_massive()  # генеруємо тривимірну карту порід
exp = BreedDepthExploration(b_m)  # ініціалізуємо клас дослідження карти
по глибині
b_col = exp.get_best_profit_column()
print(f"Краща колонка за прибутком: {b_col}")  # виводимо прибуток
найвигіднішої колонки
```

Приклад роботи програми:

```
Краща колонка за прибутком: Cords x: 6, x: 6, profit: 54
```

Блок-схема:

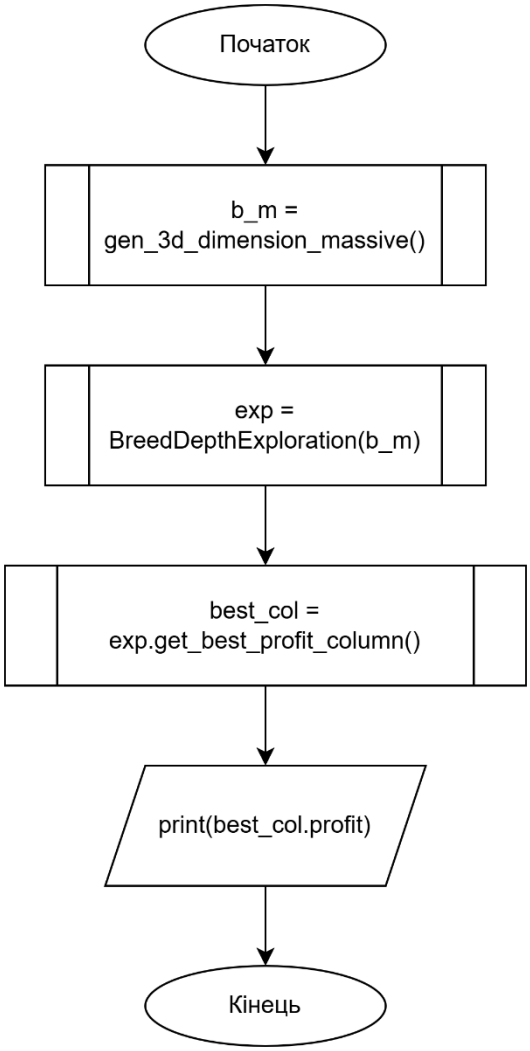

Рисунок 1 - головна блок-схема

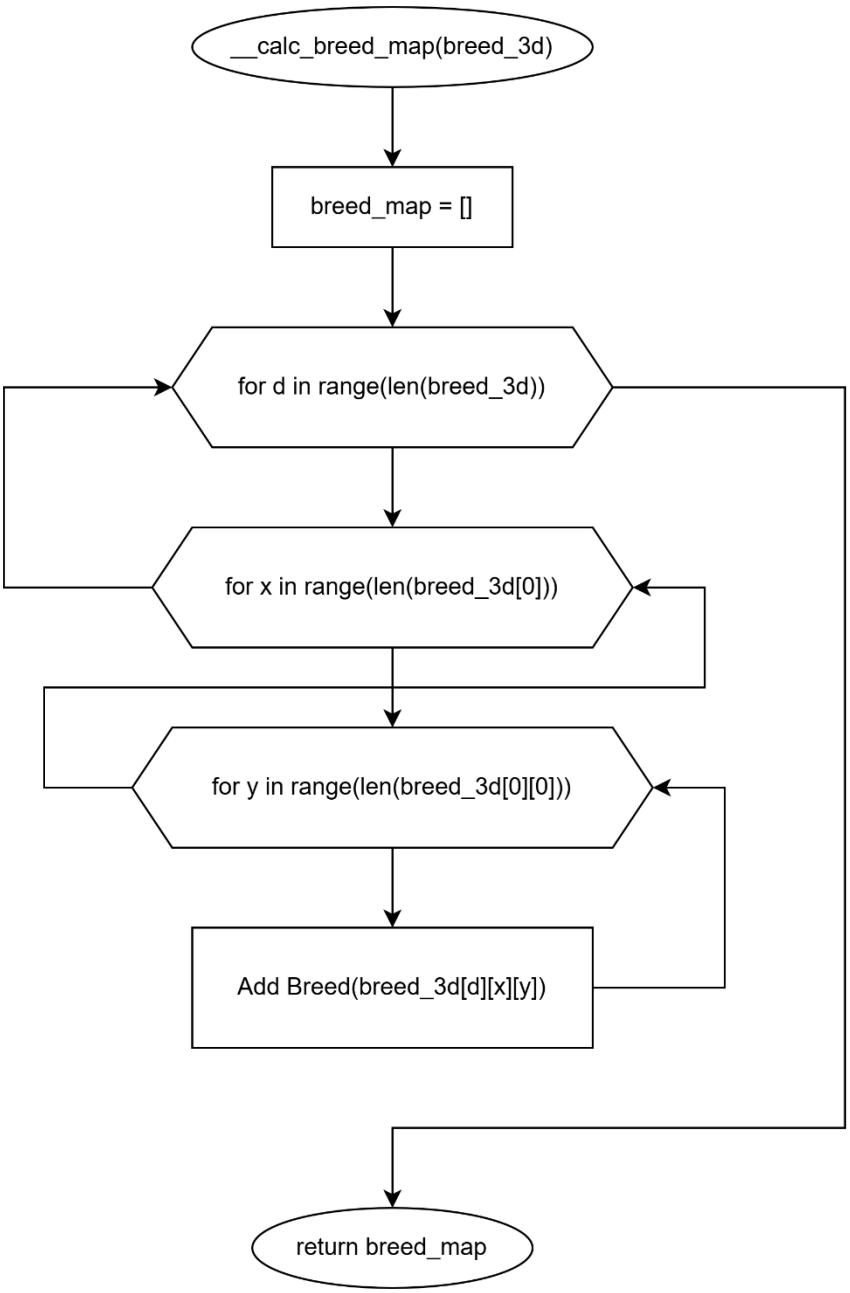

Рисунок 2 - блок-схема розрахунку карти порід

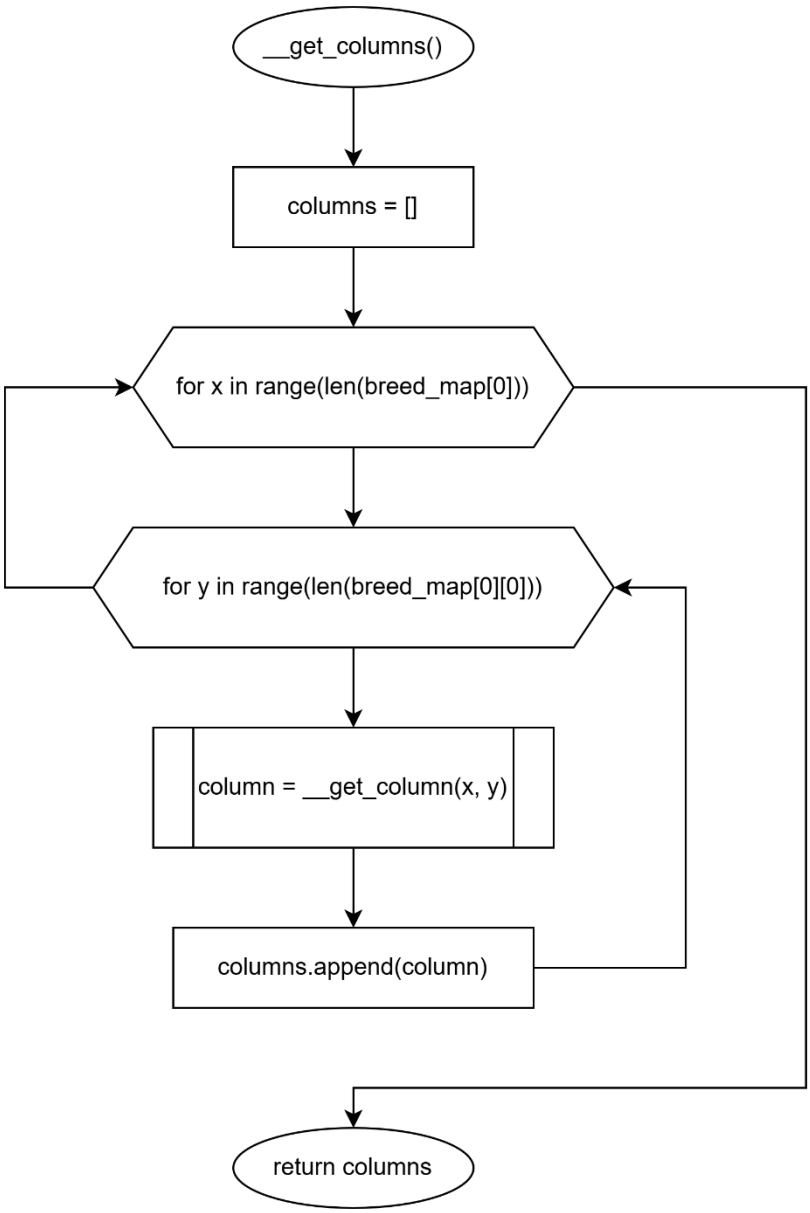

Рисунок 3 - блок-схема розрахунку колонок порід

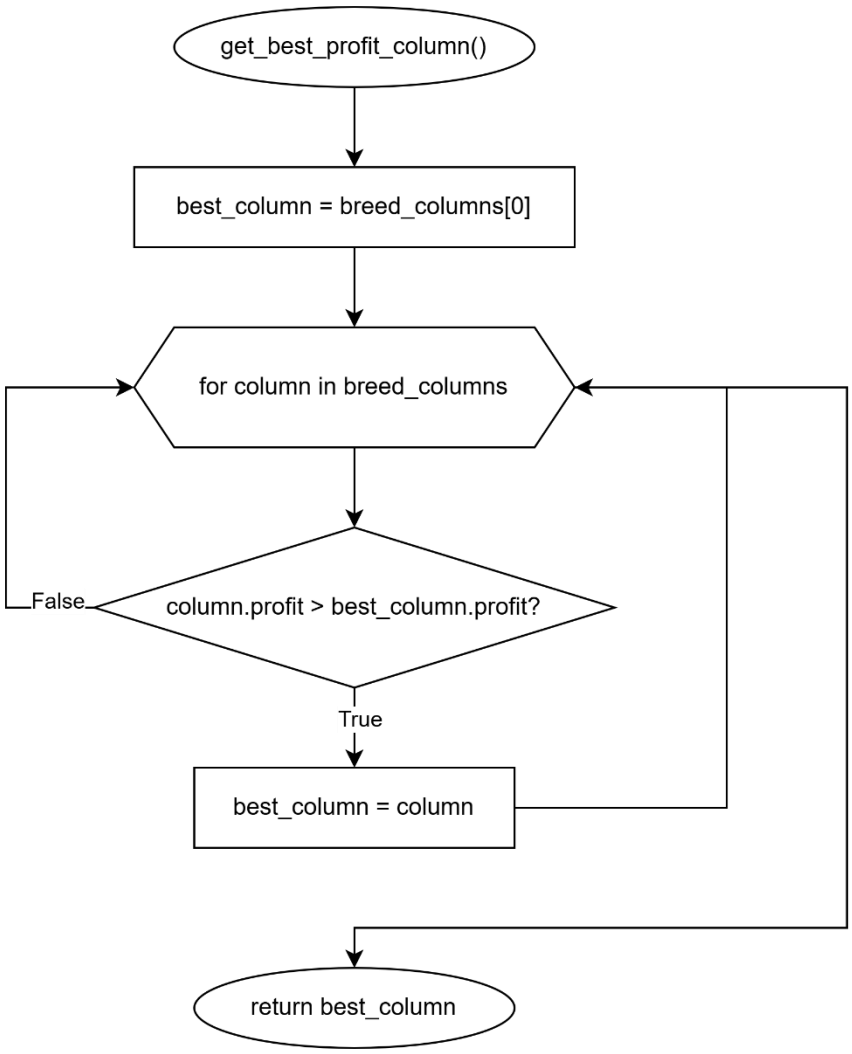

Рисунок 4 - блок-схема розрахунку кращої колонки

Висновки:

Під час виконання лабораторної роботи було створено комплексне програмне рішення що покриває всі варіанти завдань.