Київський національний університет імені Тараса Шевченка
Факультет комп’ютерних наук та кібернетики

Звіт
з лабораторної роботи № 1
«Визначення швидкодії обчислювальної системи»

Виконав
студент(ка) групи К-23 Флакей Р.Р.

Київ - 2022

Постановка задачі

Розробити програму, яка вимірює кількість виконуваних базових операцій (команд) за секунду конкретною ОбСист (комп'ютер + ОС + Система програмування). Вимірювання "чистої" команди процесора не потрібне (як і є у реальних програмних комплексах, що типово розробляються на мовах високого рівня, часто навіть на платформенно незалежних) і фактично не має сенсу. Вибір системи програмування за критерієм "яка з них генерує швидший код" зайва, - виберіть ту з них, яка для вас найбільш зручна.

Метод реалізації

Як *Систему Програмування* було обрано Python-3.10.7. Враховуючи відмінність від С-подібних мов програмування, тут є належні при запуску "прямо з коробки" відмінні базові типи даних. Окрім **int, float**, є додаткові базові типи **string, list, dict, set.** Звертаючи на це увагу, і специфічність операцій у кожного типу були підібрані відповідно:

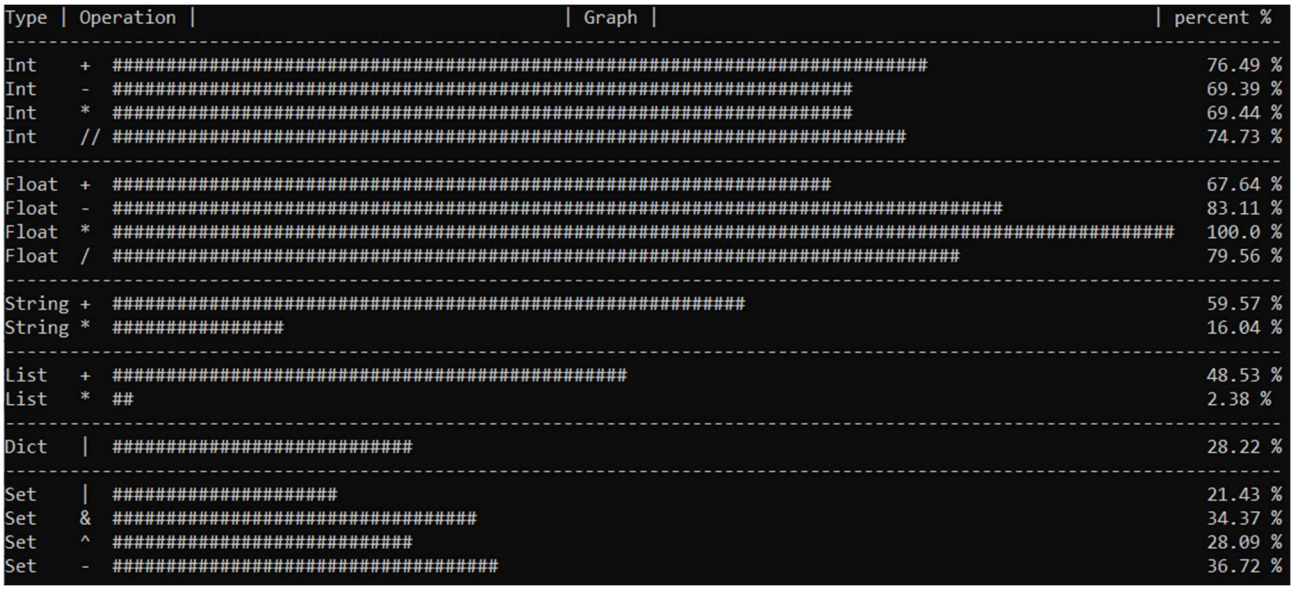
- **int** – "+", "-", "*", "//" (*Пояснення:* операція "/" призводить до перетворення типу на float, на відміну від чого, операція " //" залишається в межах int*)*
- **float** – "+", "-", "*", "/"
- **string** – "+", "*"
- **list** – "+", "*"
- **dict** – "|" (*Пояснення:* операція об'єднання двох словників)
- **set** – "|", "&", "^", "-" (*Пояснення:* множинні операції відповідно: "об'єднання", "переріз", "додавання за модулем два", "різниця")

Для обчислення швидкодії використовується вбудована бібліотека **timeit**. Це пояснюється тим, що завдяки їй стає можливим відключення одного з оптимізаційних інструментів Python, а саме "Збиральника сміття" (*який займається оптимізацією пам'яті),* для приведення до більш рівних результатів. У своїй структурі обрахунку, ця бібліотека також використовує звичайний цикл. Враховуючи це, для додаткової корекції точності, від результуючого заміру часу віднімається час який займає цикл при виконанні власної ітерації.

З додаткових можливостей цієї бібліотеки, для корекції точності також використовується ще одне оптимізаційне рішення. Кусок коду з певною відповідною операцією, перезапускається, при стандартних параметрах, 10 разів, виконуючи в кожному такому разі 10000 повторів (*Параметри кількості легко змінюються*).

Результат роботи програми

На власному комп’ютері:

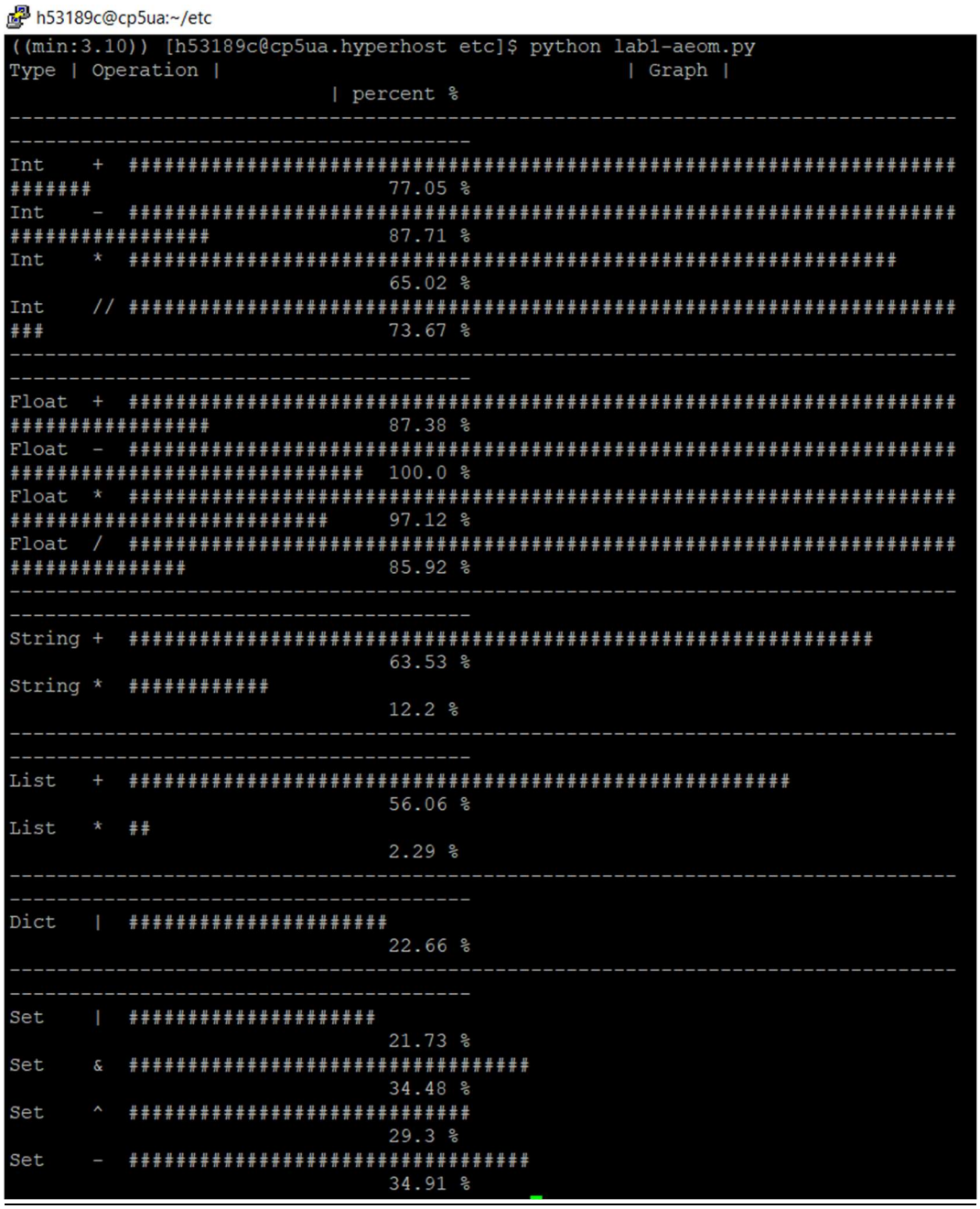


Type	Operation	Graph	percent %
Int	+	##	76.49 %
Int	-	##	69.39 %
Int	*	##	69.44 %
Int	//	##	74.73 %
Float	+	##	67.64 %
Float	-	##	83.11 %
Float	*	##	100.0 %
Float	/	##	79.56 %
String	+	##	59.57 %
String	*	###############	16.04 %
List	+	##	48.53 %
List	*	##	2.38 %
Dict	|	##########################	28.22 %
Set	|	###################	21.43 %
Set	&	################################	34.37 %
Set	^	##########################	28.09 %
Set	-	##################################	36.72 %

Характеристики:

- Процесор: AMD Ryzen 5 4600H with Radeon Graphics 3.00 GHz
- ОС: Windows 10 Home x64 21H2
- Система програмування: Python-3.10.4

На віртуальній машині



Характеристики:

- Процесор: Intel(R) Xeon(R) CPU E5-2623 v3 @ 3.00GHz
- ОС: linux x86_64
- Система програмування: Python-3.10.4

Код lab1-aeom.py:

```
from timeit import Timer;


class TestTimer:

    @staticmethod
    def _Int(operation):
        code_frag = None;

        match operation:
            case "+":
                code_frag = ("a3 = a1+a2;a4 = a3+a1;a5 = a4+a2;a6 = a4+a5");

            case "-":
                code_frag = ("a3 = a1-a2;a4 = a3-a1;a5 = a4-a2;a6 = a4-a5");

            case "*":
                code_frag = ("a3 = a1*a2;a4 = a3*a1;a5 = a4*a2;a6 = a4*a5");

            case "//":
                code_frag = ("a3 = a1//a2;a4 = a3//a2;a5 = a1//a2;a6 = a3//a2;");

            case _:
                raise NotImplementedError(f"TestTimer | Int | {operation = }")

        return Timer(code_frag, globals={"a1": 45262, "a2": 100});

    @staticmethod
    def _Float(operation):
        code_frag = None;

        match operation:
            case "+":
                code_frag = ("a3 = a1+a2;a4 = a3+a1;a5 = a4+a2;a6 = a4+a5");

            case "-":
                code_frag = ("a3 = a1-a2;a4 = a3-a1;a5 = a4-a2;a6 = a4-a5");

            case "*":
                code_frag = ("a3 = a1*a2;a4 = a3*a1;a5 = a4*a2;a6 = a4*a5");

            case "/":
                code_frag = ("a3 = a1/a2;a4 = a3/a1;a5 = a1/a2;a6 = a3/a1");

            case _:
                raise NotImplementedError(f"TestTimer | Float | {operation = }")

        return Timer(code_frag, globals={"a1": 45262.0, "a2": 100.0});
```

```
@staticmethod
def _String(operation):
    code_frag = None;

    match operation:
        case "+":
            code_frag = ("a3 = a1+a2;a4 = a3+a1;a5 = a4+a2;a6 = a4+a5");

        case "*":
            code_frag = ("a3 = a1*b;a4 = a2*b;a5 = a3*b;a6 = a4*b");

        case _:
            raise NotImplementedError(f"TestTimer | String | {operation = }")

    return Timer(code_frag, globals={"a1": "A", "a2": "B","b":100});


@staticmethod
def _List(operation):
    code_frag = None;

    match operation:
        case "+":
            code_frag = ("a3 = a1+a2;a4 = a3+a1;a5 = a4+a2;a6 = a4+a5");

        case "*":
            code_frag = ("a3 = a1*b;a4 = a3*b;a5 = a3*b;a6 = a4*b");

        case _:
            raise NotImplementedError(f"TestTimer | List | {operation = }")

    return Timer(code_frag, globals={"a1": [0], "a2": [1], "b": 10});


@staticmethod
def _Dict(operation):
    code_frag = None;

    match operation:
        case "|":
            code_frag = ("a3 = a1|a2;a4 = a3|a1;a5 = a4|a2;a6 = a4|a5");

        case _:
            raise NotImplementedError(f"TestTimer | Dict | {operation = }")

    return Timer(code_frag, globals={"a1": {1:1,2:2,3:3}, "a2": {2:2}});
```

```
    @staticmethod
    def _Set(operation):
        code_frag = None;

        match operation:
            case "|":
                code_frag = ("a3 = a1|a2;a4 = a3|a1;a5 = a4|a2;a6 = a4|a5");

            case "&":
                code_frag = ("a3 = a1&a2;a4 = a3&a1;a5 = a4&a2;a6 = a4&a5");

            case "^":
                code_frag = ("a3 = a1^a2;a4 = a3^a1;a5 = a4^a2;a6 = a4^a5");

            case "-":
                code_frag = ("a3 = a1-a2;a4 = a3-a1;a5 = a4-a2;a6 = a4-a5");

            case _:
                raise NotImplementedError(f"TestTimer | Set | {operation = }")

        return Timer(code_frag, globals={"a1": {1,2,3}, "a2": {2}});


    @staticmethod
    def _Free():
        return Timer("pass", globals={});


    def _check_arrange(self, Time_obj):
        time_res = Time_obj.repeat(self.repeat, self.number)
        arrange = (sum(time_res)/self.repeat)/self.number

        return arrange


    def speed_test(self):
        check_list = {"Int": (self._Int,("+", "-", "*", "//")),
        "Float": (self._Float, ("+", "-", "*", "/")),
        "String": (self._String,("+", "*")),
        "List": (self._List,("+", "*")),
        "Dict": (self._Dict, ("|")),
        "Set": (self._Set,("|", "&", "^", "-"))}

        free_cycle_time = self._check_arrange(self._Free())

        result_dict = {}

        for Type, (timer_gen,oper_group) in check_list.items():
            for operation in oper_group:
                result_dict[(Type, operation)] =
self._check_arrange(timer_gen(operation)) - free_cycle_time;
```

```
        return result_dict


    def print_result(self, result_dict):

        min_value = min(result_dict.values())
        prev = None

        print("Type | ".ljust(1) + "Operation | ".ljust(45) + ("| Graph |
").ljust(55) + f"| percent %")

        for (Type, operation), time_res in result_dict.items():

            if prev != Type:
                prev = Type
                print("-"*119)

            res = Type.ljust(7) + operation.ljust(3) + ("#" *
int(min_value*100//time_res)).ljust(102) + f"{round(min_value*100/time_res, 2)} %"
            print(res)

    def __init__(self, repeat=10, number=10000):
        self.repeat = repeat;
        self.number = number;



if __name__=="__main__":
    time_tester = TestTimer();
    res_dict = time_tester.speed_test()
    time_tester.print_result(res_dict)
```

Код Dockerfile.dockerfile:

```
# syntax=docker/dockerfile:1

FROM python:3.10-slim-buster

COPY . ./app
WORKDIR /app

CMD ["python3", "lab1-aeom.py"]
```

Посилання на репозиторій докера:
https://hub.docker.com/r/mintnt/lab1-aoem

Посилання на Github репозиторій:
https://github.com/MinTins/lab1-aoem