Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки

Розрахунково-графічна робота з дисципліни “Компоненти програмної інженерії-3. Архітектура програмного забезпечення” на тему “Архітектурні діаграми та бенчмарки”

Виконали:
Третяк Владислав,
Голік Владислав,
Королюк Ярослав,
Пєнской Володимир

Група: ІП-94
Перевірив: Мазур Р.Ф.

Київ 2021

Зміст

Мета: Закріплення навичок ілюстрації організації програмних систем та оцінки часу виконання алгоритмів

Завдання 1

У другій лабораторній роботі було розроблено функцію перетворення префіксного виразу в постфіксний.

Щоб підтвердити лінійний час виконання даної функції, було підготовлено бенчмарк для неї, що запускає функцію для даних різного розміру. При формуванні тестового вхідного виразу тип оператору та розташування оператору з операндом вибираються випадковим чином за допомогою rand.Intn(). Код нижче запускає функцію PrefixToPostfix 20 разів, передаючи їй вхідний вираз типу string різної довжини(від 5 до 2097153 символів).

```
var convertRes string

func BenchmarkPrefixToPostfix(b *testing.B) {
    const baseLen = 1
    for i := 0; i < 20; i++ {
      l := baseLen * 1 << i
      in := ""
      operators := [5]string{"+", "-", "*", "/", "^"}
      for k := 0; k < l; k++ {
            if rand.Intn(2) == 0 {
                  in = operators[rand.Intn(5)] + " " + in + "0 "
            } else {
                  in += operators[rand.Intn(5)] + " 0 "
            }
      }
      in += "0"
      b.Run(fmt.Sprintf("len=%d", len(in)), func(b *testing.B) {
            convertRes, _ = PrefixToPostfix(in)
      })
    }
}
```

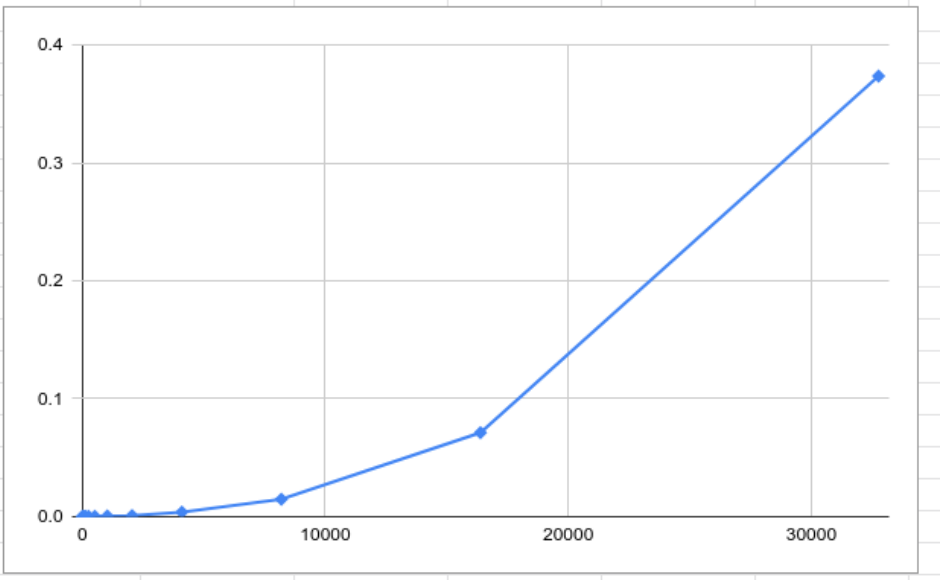
Запуск даного бенчмарка дає нам 20 точок для візуалізації:

```
goos: linux
goarch: amd64
pkg: github.com/invictoprojects/architecture-lab-2
cpu: Intel(R) Core(TM) i5-8265U CPU @ 1.60GHz
BenchmarkPrefixToPostfix/len=5          1000000000          0.0000617 ns/op
```

```
BenchmarkPrefixToPostfix/len=9              1000000000          0.0000341 ns/op
BenchmarkPrefixToPostfix/len=17             1000000000          0.0000911 ns/op
BenchmarkPrefixToPostfix/len=33             1000000000          0.0001060 ns/op
BenchmarkPrefixToPostfix/len=65             1000000000          0.0000570 ns/op
BenchmarkPrefixToPostfix/len=129            1000000000          0.0001753 ns/op
BenchmarkPrefixToPostfix/len=257            1000000000          0.0001046 ns/op
BenchmarkPrefixToPostfix/len=513            1000000000          0.0001676 ns/op
BenchmarkPrefixToPostfix/len=1025           1000000000          0.0005721 ns/op
BenchmarkPrefixToPostfix/len=2049           1000000000          0.0008273 ns/op
BenchmarkPrefixToPostfix/len=4097           1000000000          0.003907 ns/op
BenchmarkPrefixToPostfix/len=8193           1000000000          0.01485 ns/op
BenchmarkPrefixToPostfix/len=16385          1000000000          0.07126 ns/op
BenchmarkPrefixToPostfix/len=32769          1000000000          0.3735 ns/op
BenchmarkPrefixToPostfix/len=65537                 1            1311174771 ns/op
BenchmarkPrefixToPostfix/len=131073                1            5185979236 ns/op
BenchmarkPrefixToPostfix/len8=262145               1            22141843663 ns/op
^Csignal: interrupt
FAIL    github.com/invictoprojects/architecture-lab-2   122.128s
```

Алгоритм працює дуже повільно і на 15-ому кроці бенчмарк починає рахувати час тільки одного запуску функції, а не середнє значення, отримане за 1 млрд операцій. Тому, виконання бенчмарку було призупинено. З отриманих точок побудуємо графік:

5	0.0000617
9	0.0000341
17	0.0000911
33	0.000106
65	0.000057
129	0.0001753
257	0.0001046
513	0.0001676
1025	0.0005721
2049	0.0008273
4097	0.003907
8193	0.01485
16385	0.07126
32769	0.3735



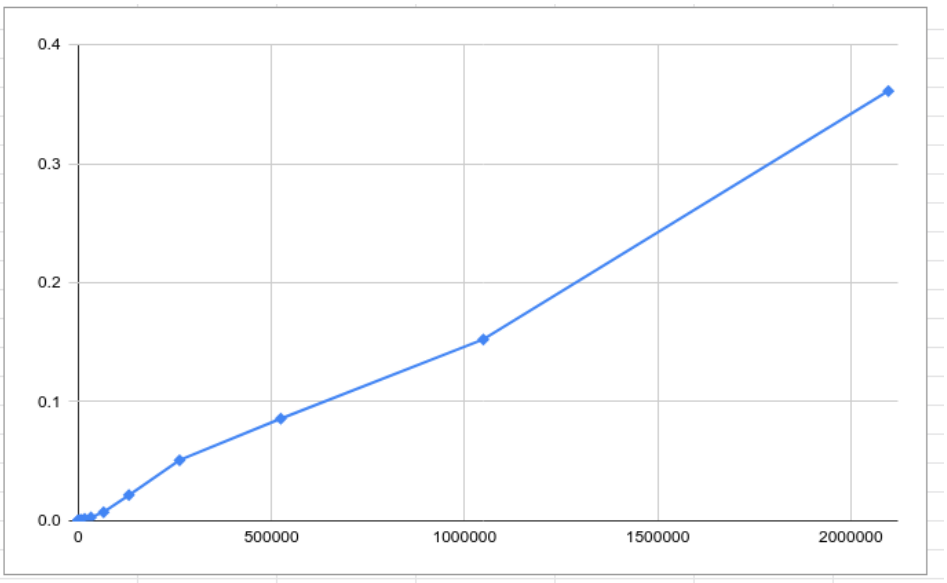
Графік не підтверджує лінійний час виконання функції і більш подібний до квадратичного. Тому було прийнято рішення дослідити функцію на недолік, що призводить до такого результату.

Наш алгоритм складається з 3-ох етапів: валідація виразу, побудова дерева та проходження побудованого дерева для отримання результату. Якщо

запустити бенчмарк, закоментувавши виклик функції, відповідальної за проходження дерева, то отримаємо наступний результат:

```
goos: linux
goarch: amd64
pkg: github.com/invictoprojects/architecture-lab-2
cpu: Intel(R) Core(TM) i5-8265U CPU @ 1.60GHz
BenchmarkPrefixToPostfix/len=5          1000000000          0.0000569 ns/op
BenchmarkPrefixToPostfix/len=9          1000000000          0.0000228 ns/op
BenchmarkPrefixToPostfix/len=17         1000000000          0.0000350 ns/op
BenchmarkPrefixToPostfix/len=33         1000000000          0.0001251 ns/op
BenchmarkPrefixToPostfix/len=65         1000000000          0.0000341 ns/op
BenchmarkPrefixToPostfix/len=129        1000000000          0.0000553 ns/op
BenchmarkPrefixToPostfix/len=257        1000000000          0.0000873 ns/op
BenchmarkPrefixToPostfix/len=513        1000000000          0.0000846 ns/op
BenchmarkPrefixToPostfix/len=1025       1000000000          0.0001135 ns/op
BenchmarkPrefixToPostfix/len=2049       1000000000          0.0002568 ns/op
BenchmarkPrefixToPostfix/len=4097       1000000000          0.0003786 ns/op
BenchmarkPrefixToPostfix/len=8193       1000000000          0.0007297 ns/op
BenchmarkPrefixToPostfix/len=16385      1000000000          0.001536 ns/op
BenchmarkPrefixToPostfix/len=32769      1000000000          0.002939 ns/op
BenchmarkPrefixToPostfix/len=65537      1000000000          0.007291 ns/op
BenchmarkPrefixToPostfix/len=131073     1000000000          0.02150 ns/op
BenchmarkPrefixToPostfix/len=262145     1000000000          0.05093 ns/op
BenchmarkPrefixToPostfix/len=524289     1000000000          0.08579 ns/op
BenchmarkPrefixToPostfix/len=1048577    1000000000          0.1524 ns/op
BenchmarkPrefixToPostfix/len=2097153    1000000000          0.3611 ns/op
PASS
ok      github.com/invictoprojects/architecture-lab-2   396.384s
```

5	0.0000569
9	0.0000228
17	0.000035
33	0.0001251
65	0.0000341
129	0.0000553
257	0.0000873
513	0.0000846
1025	0.0001135
2049	0.0002568
4097	0.0003786
8193	0.0007297
16385	0.001536
32769	0.002939
65537	0.007291
131073	0.0215
262145	0.05093
524289	0.08579
1048577	0.1524
2097153	0.3611



Маємо графік, що підтверджує лінійний час виконання частини алгоритму. Тобто, недолік знаходиться саме у функції getPostfixFromTree(root *node), що проходить по дереву і повертає постфіксний вираз.

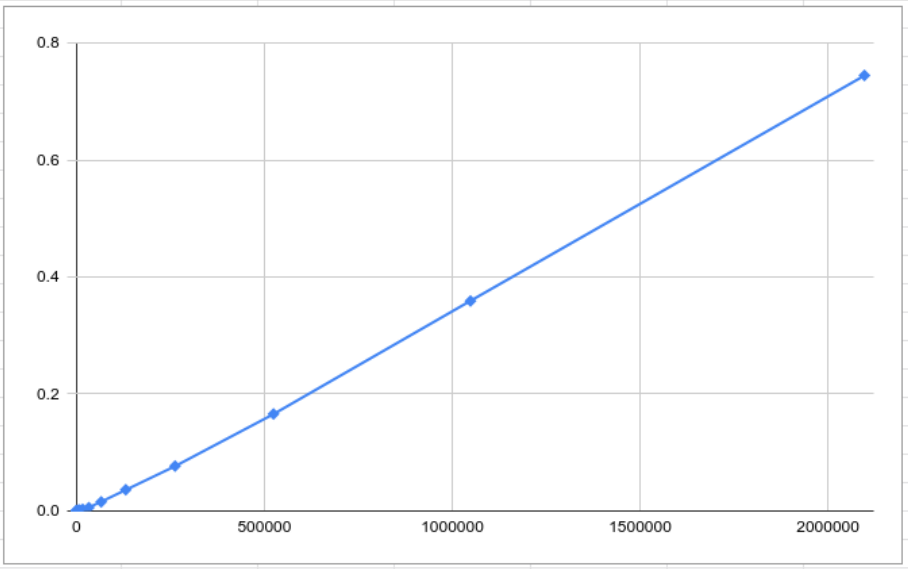
Виконавши аналіз даної функції, було виявлено, що нелінійність алгоритму виникає через конкатенацію за допомогою оператору ‘+=’, яка виконується кожен раз при додаванні елементу з дерева до результуючого рядку. Тип проміжної змінної результату було замінено на strings.Builder, а оператор ‘+=’ на метод WriteString. Результуючий рядок отримуємо за допомогою методу String. Сам алгоритм перетворення не було змінено.

Отримано наступні 20 точок для візуалізації:

```
goos: linux
goarch: amd64
pkg: github.com/invictoprojects/architecture-lab-2
cpu: Intel(R) Core(TM) i5-8265U CPU @ 1.60GHz
BenchmarkPrefixToPostfix/len=5          1000000000          0.0000809 ns/op
BenchmarkPrefixToPostfix/len=9          1000000000          0.0000341 ns/op
BenchmarkPrefixToPostfix/len=17         1000000000          0.0000330 ns/op
BenchmarkPrefixToPostfix/len=33         1000000000          0.0000433 ns/op
BenchmarkPrefixToPostfix/len=65         1000000000          0.0001321 ns/op
BenchmarkPrefixToPostfix/len=129        1000000000          0.0000530 ns/op
BenchmarkPrefixToPostfix/len=257        1000000000          0.0000952 ns/op
BenchmarkPrefixToPostfix/len=513        1000000000          0.0001186 ns/op
BenchmarkPrefixToPostfix/len=1025       1000000000          0.0003266 ns/op
BenchmarkPrefixToPostfix/len=2049       1000000000          0.0004325 ns/op
BenchmarkPrefixToPostfix/len=4097       1000000000          0.0008440 ns/op
BenchmarkPrefixToPostfix/len=8193       1000000000          0.001418 ns/op
BenchmarkPrefixToPostfix/len=16385      1000000000          0.002756 ns/op
BenchmarkPrefixToPostfix/len=32769      1000000000          0.005938 ns/op
BenchmarkPrefixToPostfix/len=65537      1000000000          0.01564 ns/op
BenchmarkPrefixToPostfix/len=131073     1000000000          0.03613 ns/op
BenchmarkPrefixToPostfix/len=262145     1000000000          0.07671 ns/op
BenchmarkPrefixToPostfix/len=524289     1000000000          0.1657 ns/op
BenchmarkPrefixToPostfix/len=1048577    1000000000          0.3592 ns/op
BenchmarkPrefixToPostfix/len=2097153    1000000000          0.7444 ns/op
PASS
ok      github.com/invictoprojects/architecture-lab-2   150.446s
```

З ними ми можемо побудувати графік, який підтверджує лінійний час виконання:

5	0.0000809
9	0.0000341
17	0.000033
33	0.0000433
65	0.0001321
129	0.000053
257	0.0000952
513	0.0001186
1025	0.0003266
2049	0.0004325
4097	0.000844
8193	0.001418
16385	0.002756
32769	0.005938
65537	0.01564
131073	0.03613
262145	0.07671
524289	0.1657
1048577	0.3592
2097153	0.7444

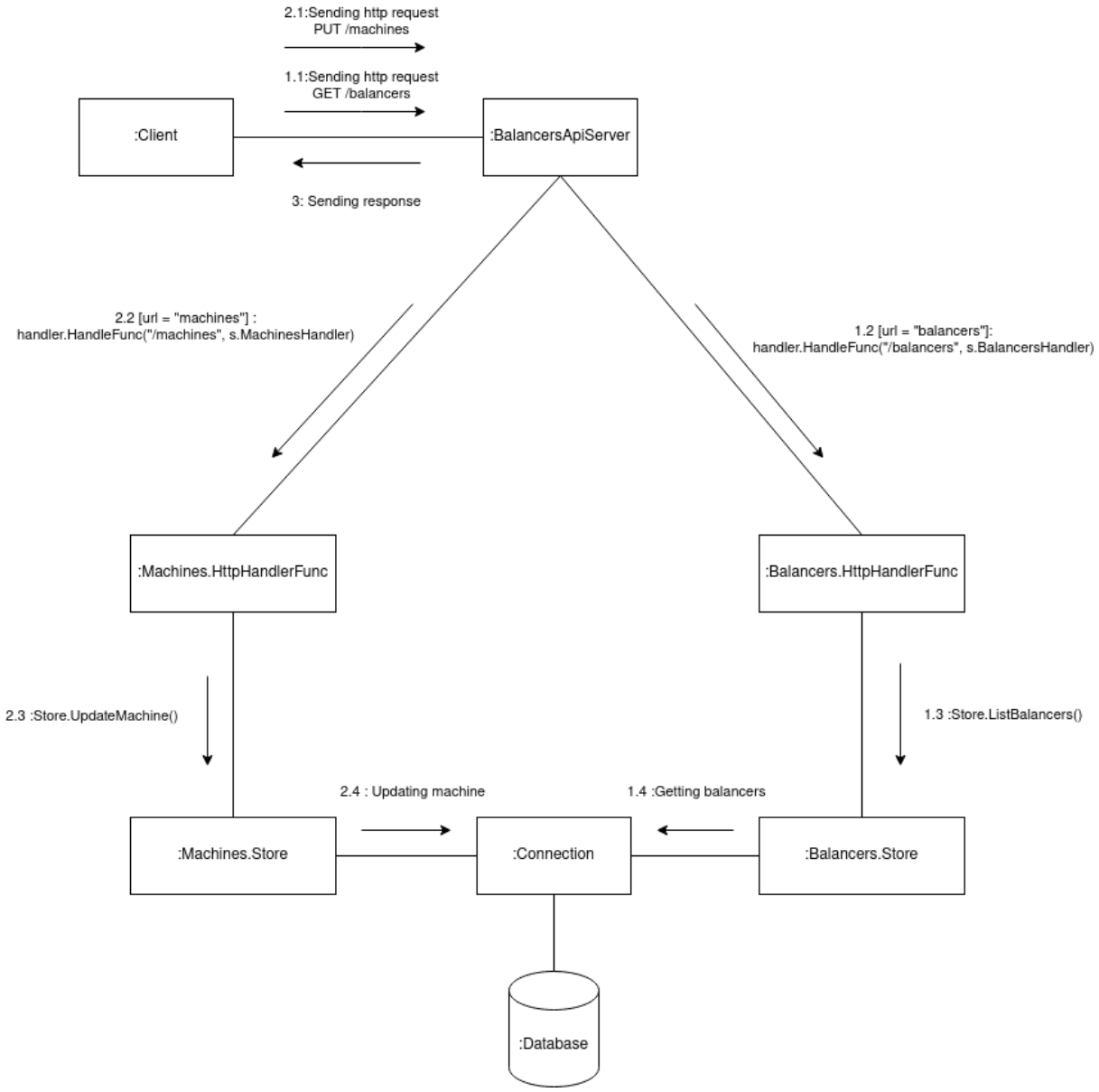


Дана проблема виникає через те, що для кожного оператора ‘+’ тип string виділяє новий блок у пам'яті та копіює в нього все, що у нього є плюс суфікс, що з’єднується. Через це ми отримуємо графік наближений до квадратичного. Builder з пакету strings використовується для ефективного створення рядка за допомогою методів Write, що мінімізує копіювання пам'яті.

Отже, лінійний час виконання функції перетворення вхідного виразу було підтверджено.

Завдання 2

Діаграма взаємодії:

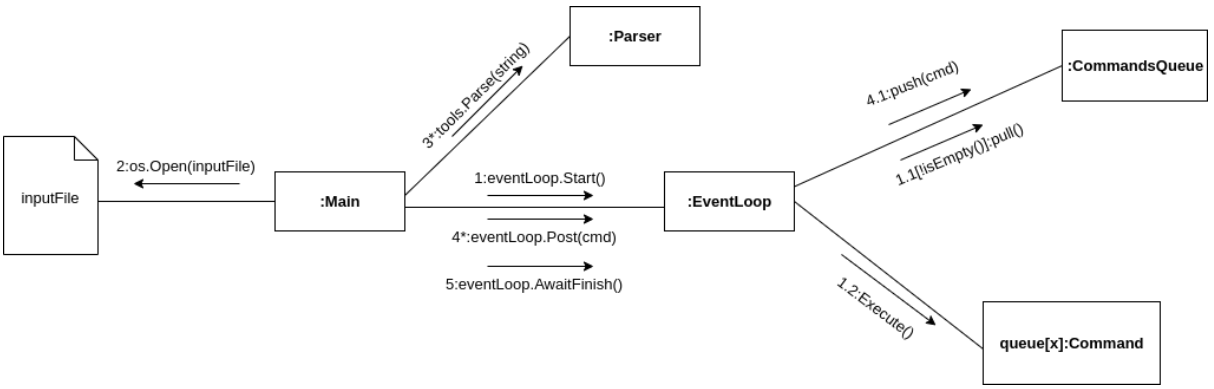


На діаграмі можна побачити як відбувається обробка запитів. Спочатку клієнт надсилає HTTP запит на сервер (для прикладу візьмемо запит GET /balancers). BalancersApiServer обробляє запит за допомогою Balancers.HttpHandlerFunc, яка в свою чергу викликає метод ListBalancers у Balancers.Store. За допомогою Connection виконується підключення до бази даних і отримується дескриптор sql.DB. Він використовується у функції ListBalancers для виконання SQL запиту до бази даних, що повертає

список балансерів. Надалі формується HTTP відповідь, яка повертається клієнту.

Завдання 3

Діаграма взаємодії:



Як видно з діаграми**,** спочатку запускається цикл подій та читається файл з командами, який задається як аргумент командного рядку. Далі, з кожного його рядка парситься команда: print чи cat (за варіантом), і надсилається в чергу. Одночасно з цим, цикл виконує команди з черги, доки не виконає всі команди з черги та не отримає сигналу про завершення, який надсилається після закінчення роботи парсера команд.

Щоб підтвердити лінійний час виконання даної функції, ми створили бенчмарк для неї, запускаючи функцію для даних різного розміру та різної кількості команд.

Код нижче запускає функцію Parse 25 разів, передаючи їй команду для парсингу різного розміру. Кожен запуск оформлено у вигляді окремого дочірнього бенчмарка, що дає можливість отримати середній час виконання функції для кожного з розмірів.

```
package benchmark

import (
    "fmt"
    "strings"
    "testing"

    "github.com/InvictoProjects/Architecture-lab-4/commands"
    "github.com/InvictoProjects/Architecture-lab-4/tools"
)

var cntRes commands.Command
```

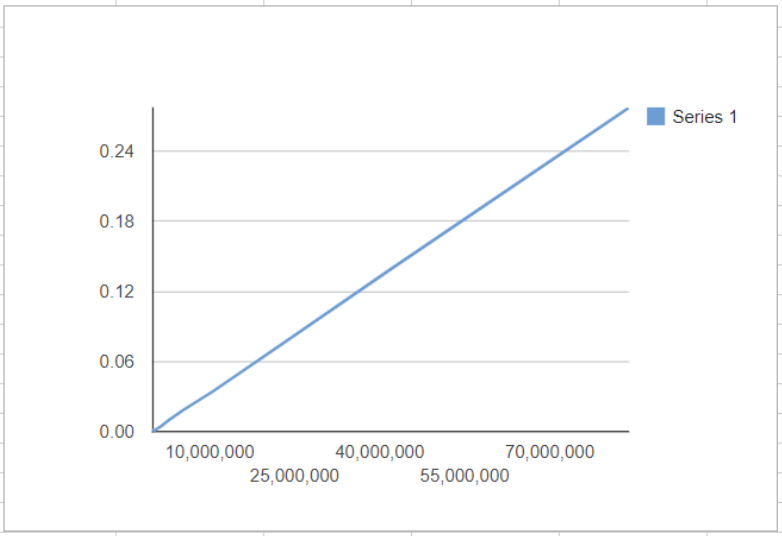
```
func BenchmarkParse(b *testing.B) {
    const baseLen = 5

    for i := 0; i < 25; i++ {
        l := baseLen * 1 << i

                    commandLine   :=   strings.Join([]string{"cat",
strings.Repeat("5", l), strings.Repeat("5", l)}, " ")

        b.Run(fmt.Sprintf("len=%d", l), func(b *testing.B) {
            cntRes = tools.Parse(commandLine)
        })
    }
}
```

Запуск даного бенчмарка дає нам 25 точок для візуалізації:

```
goos: linux
goarch: amd64
pkg: github.com/InvictoProjects/Architecture-lab-4/benchmark
cpu: Intel(R) Core(TM) i5-8265U CPU @ 1.60GHz
BenchmarkParse/len=5                1000000000          0.0000005 ns/op
BenchmarkParse/len=10               1000000000          0.0000006 ns/op
BenchmarkParse/len=20               1000000000          0.0000005 ns/op
BenchmarkParse/len=40               1000000000          0.0000006 ns/op
BenchmarkParse/len=80               1000000000          0.0000007 ns/op
BenchmarkParse/len=160              1000000000          0.0000010 ns/op
BenchmarkParse/len=320              1000000000          0.0000015 ns/op
BenchmarkParse/len=640              1000000000          0.0000024 ns/op
BenchmarkParse/len=1280             1000000000          0.0000045 ns/op
BenchmarkParse/len=2560             1000000000          0.0000084 ns/op
BenchmarkParse/len=5120             1000000000          0.0000162 ns/op
BenchmarkParse/len=10240            1000000000          0.0000319 ns/op
BenchmarkParse/len=20480            1000000000          0.0000630 ns/op
BenchmarkParse/len=40960            1000000000          0.0001252 ns/op
BenchmarkParse/len=81920            1000000000          0.0002495 ns/op
BenchmarkParse/len=163840           1000000000          0.0005221 ns/op
BenchmarkParse/len=327680           1000000000          0.001047 ns/op
BenchmarkParse/len=655360           1000000000          0.002260 ns/op
BenchmarkParse/len=1310720          1000000000          0.004054 ns/op
BenchmarkParse/len=2621440          1000000000          0.009070 ns/op
BenchmarkParse/len=5242880          1000000000          0.01813 ns/op
BenchmarkParse/len=10485760         1000000000          0.03414 ns/op
BenchmarkParse/len=20971520         1000000000          0.06897 ns/op
BenchmarkParse/len=41943040         1000000000          0.1391 ns/op
BenchmarkParse/len=83886080         1000000000          0.2772 ns/op
PASS
ok      github.com/InvictoProjects/Architecture-lab-4/benchmark     7.344s
```

З ними ми можемо побудувати графік, який підтверджує лінійний час виконання:

5	0.0000005
10	0.0000006
20	0.0000005
40	0.0000006
80	0.0000007
160	0.0000010
320	0.0000015
640	0.0000024
1,280	0.0000045
2,560	0.0000084
5,120	0.0000162
10,240	0.0000319
20,480	0.0000630
40,960	0.0001252
81,920	0.0002495
163,840	0.0005221
327,680	0.0010470
655,360	0.0022600
1,310,720	0.0040540
2,621,440	0.0090700
5,242,880	0.0181300
10,485,760	0.0341400
20,971,520	0.0689700
41,943,040	0.1391000
83,886,080	0.2772000



Перевіримо на лінійність функцію парсери при різній кількості команд. В цьому випадку, кожен запуск оформлено у вигляді окремого дочірнього бенчмарка, що дає можливість отримати середній час виконання функції для кожної кількості команд. Для забезпечення чистоти визначення впливу саме кількості команд, а не довжини аргументів (лінійність чого вже було доведено), встановимо певну сталу довжину аргументів. Таким чином маємо наступне:

```
package benchmark

import (
	"fmt"
	"github.com/InvictoProjects/Architecture-lab-4/tools"
	"math/rand"
	"strings"
	"testing"

	"github.com/InvictoProjects/Architecture-lab-4/commands"
)
```

```
var cntResult commands.Command
var argumentLen = 20
var maxRuneByte = 300

func BenchmarkParse_CommandAmount(b *testing.B) {
	const baseAmount = 5

	for i := 0; i < 20; i++ {
		l := baseAmount * 1 << i

		var commandsSlice []string
		for j := 0; j < l; j++ {
			commandsSlice = append(commandsSlice,
generateCommand())
		}

		b.Run(fmt.Sprintf("len=%d", l), func(b *testing.B) {
			for k := 0; k < len(commandsSlice); k++ {
				cntResult = tools.Parse(commandsSlice[k])
			}
		})
	}
}

func generateCommand() string {
	if param := rand.Intn(2); param%2 == 0 {
		return "print " + generateArgument()
	} else {
		return "cat " + generateArgument() + " " +
generateArgument()
	}
}

func generateArgument() string {
	res := strings.Builder{}
	for i := 0; i < argumentLen+1; i++ {
		res.WriteRune(rune(rand.Intn(maxRuneByte)))
	}
	return res.String()
}
```

Для кожного дочірнього бенчмарка ми формуємо певну кількість команд: print та cat команди та їх аргументи заданої довжини. Функція аргументів може генерувати аргументи, що міститимуть пробіли чи інші подібні символи, що забезпечує перевірку роботи функції для некоректно заданих команд також.

Запуск даного бенчмарка дає нам 20 точок для візуалізації:

```
goos: linux
goarch: amd64
pkg: Architecture-lab-4/benchmark
cpu: Intel(R) Core(TM) i7-8565U CPU @ 1.80GHz
BenchmarkParse_CommandAmount
BenchmarkParse_CommandAmount/len=5              1000000000           0.0000066 ns/op
BenchmarkParse_CommandAmount/len=10             1000000000           0.0000116 ns/op
BenchmarkParse_CommandAmount/len=20             1000000000           0.0000159 ns/op
BenchmarkParse_CommandAmount/len=40             1000000000           0.0000183 ns/op
BenchmarkParse_CommandAmount/len=80             1000000000           0.0000343 ns/op
BenchmarkParse_CommandAmount/len=160            1000000000           0.0000659 ns/op
BenchmarkParse_CommandAmount/len=320            1000000000           0.0001307 ns/op
BenchmarkParse_CommandAmount/len=640            1000000000           0.0002595 ns/op
BenchmarkParse_CommandAmount/len=1280           1000000000           0.0004806 ns/op
BenchmarkParse_CommandAmount/len=2560           1000000000           0.0009219 ns/op
BenchmarkParse_CommandAmount/len=5120           1000000000           0.001753 ns/op
BenchmarkParse_CommandAmount/len=10240          1000000000           0.004029 ns/op
BenchmarkParse_CommandAmount/len=20480          1000000000           0.006002 ns/op
BenchmarkParse_CommandAmount/len=40960          1000000000           0.01234 ns/op
BenchmarkParse_CommandAmount/len=81920          1000000000           0.02333 ns/op
BenchmarkParse_CommandAmount/len=163840         1000000000           0.04808 ns/op
BenchmarkParse_CommandAmount/len=327680         1000000000           0.09335 ns/op
BenchmarkParse_CommandAmount/len=655360         1000000000           0.1870 ns/op
BenchmarkParse_CommandAmount/len=1310720        1000000000           0.3926 ns/op
BenchmarkParse_CommandAmount/len=2621440        1000000000           0.7907 ns/op
PASS

Process finished with the exit code 0
```

Та відповідно графік, що підтверджує лінійність:

5	0,0000066
10	0,0000116
20	0,0000159
40	0,0000183
80	0,0000343
160	0,0000659
320	0,0001307
640	0,0002595
1280	0,0004806
2560	0,0009219
5120	0,001753
10240	0,004029
20480	0,006002
40960	0,01234
81920	0,02333
163840	0,04808
327680	0,09335
655360	0,187
1310720	0,3926
2621440	0,7907