Języki skryptowe i ich zastosowania. Funkcja "count"

Mikołaj Sumowski

6 marca 2019

Indeks Autora:	160479
Specjalność:	Algorytmy I Technologie Internetowe
Przedmiot:	Języki Skryptowe I Ich Zastosowania

1 Cel

Celem zadania jest dokonanie pomiarów oraz porównanie funkcji count w 3 wariantach:

- wbudowana funkcja języka Python,
- własna implementacja w języku Python,
- własna implementacja w języku C.

Zadanie ma na celu wykazać różnice pomiędzy językami skryptowymi, takimi jak Python, a językami natywnymi.

2 Funkcja Count

Funkcja count jest metodą klasy string. Zwraca liczbę nienakładających się na siebie powtórzeń ciągu znaków. Funkcja umożliwia ograniczenie wielkości przeszukiwanego napisu poprzez opcjonalne parametry.

2.1 Python Funkcja Wbudowana

str.count(sub[,start[,end]])

2.1.1 Dane

Dane wejściowe:

str - ciąg znaków, na którym wykonana zostanie operacja wyszukiwania.

sub - wzorzec jako ciąg znaków wyszukiwanych w str.

start - parametr opcjonalny typu Integer oznaczający początek zakresu poszukiwań interpretowany według notacji kawałkowej (ang.slice notation).

end - parametr opcjonalny typu Integer oznaczający koniec zakresu poszukiwań interpretowany według notacji kawałkowej (ang.slice notation).

Dane wyjściowe:

return - wynik typu Integer określający ilość znalezionych wystąpień wzorca sub w tekście str.

2.1.2 Implementacja

Według źródeł projektu CPython, funkcja implementuje algorytm Boyera-Moore'a ze specjalnym wyszukiwaniem wzorców jedno znakowych(tzw. FASTMODE). Implementacja wykonana w języku C.

2.2 Własna Implementacja

Python: count(str, sub, start=None, end=None)

C: int count(const char* str, const char* sub, int start, int end)

2.2.1 Dane

Dane wejściowe:

str - ciąg znaków na którym wykonana zostanie operacja wyszukiwania.

sub - wzorzec jako ciąg znaków wyszukiwanych w str.

start - parametr opcjonalny typu Integer wyznaczający początek zakresu poszukiwań interpretowany według notacji kawałkowej (ang.slice notation).

end - parametr opcjonalny typu Integer wyznaczający koniec zakresu poszukiwań interpretowany według notacji kawałkowej (ang.slice notation).

Dane wyjściowe:

return - wynik typu Integer określający ilość znalezionych wystąpień wzorca sub w tekście str.

2.2.2 Implementacja

W implementowanym algorytmie możemy wyróżnić 3 etapy.

- Normalizacja oraz przypisanie wartości zakresu poszukiwań.
- Wyjście dla przypadku gdy wyszukiwany wzorzec jest długości 0.
- Porównanie wzorca zaczynając od każdego kolejnego elementu str.

W przypadku, gdy wzorzec zostanie znaleziony poszukiwania zostają wznowione od kolejnego znaku w ciągu wyszukiwanym. Poszukiwania zostają zakończone gdy dojdziemy do indeksu granicznego. Implementacja została tak wykonana aby kod był porównywalny linia w linie, przez co pewne optymalizacje które mogły zostać wykonane w zależności od języka nie zostały zaimplementowane.

2.2.3 Uwagi

W związku z brakiem argumentów domyślnych w języku C podawane jest makro M_INF dla argumentu start oraz INF dla argumentu end. Wartości te związane są z maksymalną długością danych testowych generowanych do testu. Wartości używane są tylko przy testach poprawnościowych zdefiniowanych w celu potwierdzenia spójności zachowań, testy czasowe posiadają zawsze zdefiniowane wszystkie argumenty wejściowe.

3 Opis Pomiarów

Wynik pomiaru stanowić będzie różnica czasu t1 oraz t0 oraz kosztu wykonania pętli. Punkty t1 oraz t0 zostaną wyznaczone przy użyciu metody clock. Metoda ta dostępna jest zarówno w języku C(bezpośrednio) jak i Python(Pośrednio). Dokładność pomiaru funkcji wynosi 1 μs i związana jest z wielkością makra CLOCKS_PER_SEC wynoszącego na środowisku pomiarowym 1'000'000. Funkcja clock języka C wyraża ilość cykli zużycia procesora przez program, czas liczony w sekundach ze wzoru $\frac{clock() \times 1.0}{CLOCKS\_PER\_SEC}$. Funkcja clock języka Python posiada już taką konwersje i zwraca ona czas w sekundach. Do obliczenia czasu wykonywania funkcji, potrzeba 4 użyć funkcji clock, daje to ostateczną dokładność pomiaru równa 4 μs.

3.1 Opis Testu Czasowego

Test czasowy ma na celu zbadać czas wykonywania funkcji. W związku z małym czasem wykonywania funkcji czas wykonania obejmuje przetworzenie 3000 danych wejściowych po 1000 razy(Rysunek 1)

```
before = clock();
for (i = 0; i < BATCH_SIZE; ++i) {
    start = atoi(Data[i][1]);
    end = atoi(Data[i][2]);
    subString = Data[i][3];
    src = Data[i][4];
    for (j=0;j<1000;j++)
    {
        t+=1;
        //count(src,subString,start,end);
    }
}
after = clock();
```

Rysunek 1: Test czasowy

3.2 Środowisko

Specyfikacja środowiska pomiarowego:

System operacyjny	Linux Mint 18.2
Architektura procesora	X64
Taktowanie procesora	$2.6GHz(3.6GHz\ Turbo\ Boost)$
Pamięć cache procesora	$6MB(L3)$
Rozmiar pamięci ram	$2 \times 8GB$
Częstotliwość pamięci ram	$1600MHz$
Typ pamięci ram	DDR3
Python	2.7.12
GCC	5.4.0

3.3 Dane Testowe

Zbiór danych zawiera 3000 wpisów. Dane testowane generowane w języku Python. Dane generowane są 3 różnymi zbiorami:

- Zbiór A, 1000 rekordów:
  - tekst o długości losowej w zakresie od 20 do 50000 składający się z liter,
  - szukany wzorzec o długości losowej w zakresie od 20 do 50010 składający się z liter,
  - parametr Start z zakresu od -50000 do 2000,
  - parametr End z zakresu od 0 do 50000.
- Zbiór B, 1000 rekordów:
  - tekst o długości losowej w zakresie od 2 do 50000 składający się z liter,

  - szukany wzorzec o długości losowej w zakresie od 2 do 12 składający się z liter,
  - parametr Start z zakresu od -50000 do 50000,
  - parametr End z zakresu od -50000 do 50000.

- Zbiór C, 1000 rekordów:
  - tekst o długości losowej w zakresie od 0 do 50000 składający się z liter,
  - szukany wzorzec o długości 0 składający się z liter,
  - parametr Start z zakresu od -50000 do 50000,
  - parametr End z zakresu od -50000 do 50000.

Dane w pliku zawierają wynik metody wbudowanej, parametr start, parametr stop, wzorzec poszukiwany, wartość przeszukiwana. Format rekordu przedstawiony w pliku

{wynik};{start};{end};{wzorzec};{wartość przeszukiwań}\n

4 Wyniki

Język Python:

Próba	Czas funkcji wbudowanej	Czas funkcji własnej	Czas pustego przebiegu
1	$11.825799s$	$1494.763681s$	$0.092466s$
2	$10.763044s$	$1505.979943s$	$0.097287s$
3	$11.013391s$	$1414.718501s$	$0.086575s$
4	$10.834281s$	$1408.856009s$	$0.091247s$
Średnia	$11.109128s$	$1456.079534s$	$0.091893s$

Język C:

Próba	Czas funkcji własnej	Czas pustego przebiegu
1	$209.968002s$	$0.007151s$
2	$212.664436s$	$0.007091s$
3	$212.064704s$	$0.009208s$
4	$202.610139s$	$0.007097s$
Średnia	$209.326820s$	$0.007636s$

Wyniki po uwzględnieniu średniego czasu wykonania pętli. Wartości reprezentują 3'000'000 wykonań funkcji count:

Wbudowana funkcja Python	Własna implementacja C	Własna implementacja Python
$11.017235s$	$209.3191835s$	$1455.98764s$
$\sim 11s$	$\sim 3min\ 29s$	$\sim 24min\ 16s$

5 Analiza

Najszybszą metoda jest metoda wbudowana języka Python. Metoda ta implementuje wydajny algorytm Boyera-Moore'a co czyni ją znacznie szybszą niż własna prosta implementacja wyszukiwania wzorca. Zastosowanie tego samego algorytmu daje wynik gorszy w języku skryptowym. Spowodowane jest to nakładem pracy wynikającym z takich cech jak zabezpieczenia oraz elastyczność struktur danych, które nie występują w języku C. Python nie pozwala iterować poza tablicą oraz posiada zmienną wielkość struktury zależną od przechowywanej wartości, natomiast język C posiada struktury stałych rozmiarów ograniczonych ze względu na architekturę procesora.