

AAL Dokumentacja

Podobieństwo cząstkowe ciągów

Zadanie 13

Wojciech Wolny
pod opieką Mgr Inż. Kamila Dei

1. Temat

Dla dwóch ciągów znaków a i b definiujemy podobieństwo jako długość najdłuższego wspólnego prefiksu dla obu ciągów. Np. podobieństwo ciągów "abc" i "abd" wyniesi 2, natomiast podobieństwo ciągów "aaa" oraz "aaab" wynosi 3. Otrzymując na wejściu ciąg znaków S oblicz sumę podobieństw ciągu S do wszystkich jego sufiksów. Przykładowe rozwiązanie:

Wejście:

ababaa

Wyjście: 11 ($6+0+3+0+1+1$)

2. Analiza Zadania:

Głównym problemem w zadaniu jest wyliczenie najdłuższego wspólnego prefiksu dla całego wyrazu oraz jego dowolnego podsłowa zaczynającego się na znaku j -tym. W moim rozwiązaniu stworzyłem tablicę zinicjalizowaną zerami, w której zapisywałem informacje o najdłuższym wspólnym prefiksie dla słowa zaczynającego się na danym indeksie i całego wyrazu. Mając informację o poprzednich wartościach najdłuższego wspólnego prefiksu dla kolejnych słów mogę skrócić czas potrzebny na policzenie tych wartości. W sytuacji, gdy wartość dla pewnego k jest duża mogę przepisać kolejne wartości z początku tablicy, które powtarzają się w pewnym okresie. Gdy dojdę do początku kolejnego okresu zmniejszam o długość okresu poprzednio zapisaną wartość.

3. Tryby Wykonywania Programu

1. Podobieństwo cząstkowe ciągów dla gotowego pliku wejściowego:

- `python aal.py -m1 <nazwa pliku>.txt`
- Tryb: m1
- Parametry:
 - nazwa pliku z rozszerzeniem ".txt"
- Dane wejściowe:
 - Plik .txt powinien składać się z jednego ciągu znaków w każdej linii
- Dane wyjściowe:
 - suma podobieństwa cząstkowego ciągów dla każdego ciągu znaków w poprawnie wprowadzonym pliku

2. Podobieństwo cząstkowe ciągów z generowaniem wartości wejściowej:

- `python aal.py -m2 -n<długość generowanego słowa> -d<stosunek "a" w słowie> -a<size of an alphabet>`
- Tryb: m2
- Parametry:
 - `-n` - docelowa długość generowanego słowa
 - `-d` - stosunek ilości "a" w generowanym słowie do jego całej długości
 - `-a` - wielkość alfabetu w przedziale $\max(1, \min(-a, 52))$
- Dane wyjściowe:
 - Ciąg znaków "a" i innych losowych liter z ASCII o długości `-n` ze stosunkiem ilości "a" do długości ciągu `-d`.

3. Test podobieństwa cząstkowego ciągów

- `python aal.py -m3 -n<długość słowa w pierwszym problemie> -k<liczba problemów> -step<krok między długością słowa w kolejnych problemach> -r<liczba instancji problemu> -a<size of an alphabet>`
- Tryb: m3
- Parametry:
 - `-n` -długość najmniejszego słowa
 - `-k` -liczba problemów
 - `-step` -wielkość o którą zwiększamy długość słowa w kolejnym problemie
 - `-r` -liczba instancji dla każdego problemu
 - `-a` -wielkość alfabetu w przedziale $\max(1, \min(-a, 52))$
- Dane wyjściowe:
 - Tabela z wynikami przeprowadzonego testu. W kolumnach odpowiednio podane informacje o długości słowa w pojedynczym problemie, średnim czasie wykonywania dla `-r` instancji oraz współczynnikiem zgodności oceny teoretycznej z pomiarem czasu.

4. Test podobieństwa cząstkowego ciągów z porównanie dla dwóch algorytmów. O złożonościach kwadratowej i liniowej.

- `python aal.py -m4 -n<długość słowa w pierwszym problemie> -k<liczba problemów> -step<krok między długością słowa w kolejnych problemach> -r<liczba instancji problemu> -a<size of an alphabet>`
- Tryb: m4
- Parametry:
 - `-n` -długość najmniejszego słowa
 - `-k` -liczba problemów
 - `-step` -wielkość o którą zwiększamy długość słowa w kolejnym problemie
 - `-r` -liczba instancji dla każdego problemu
 - `-a` -wielkość alfabetu w przedziale $\max(1, \min(-a, 52))$
- Dane wyjściowe:
 - Dane w formacie "csv" z nagłówkiem. Kolumny są oddzielone przecinkiem, a rzędy znakiem nowej linii.
 - Dane posiadają informację o wielkości słowa w problemie, średnim wynikiem pomiaru czasu, współczynnikiem zgodności oceny teoretycznej i sumie wartości funkcji dla wszystkich jego instancji dla algorytmu o złożoności kwadratowej oraz dla algorytmu o złożoności liniowej.

5. Dodatkowa funkcja pomocnicza do generowania pseudolosowych ciągów S znajduje się w pliku `gen.py`:

- `python gen.py -n<length of generated word> -d<ratio of a's to the word> -a<size of an alphabet>`
- Tryb: Generuj
- Parametry:
 - `-n` -docelowa długość generowanego słowa
 - `-d` -stosunek ilości "a" w generowanym słowie do jego całej długości
 - `-a` -wielkość alfabetu w przedziale $\max(1, \min(-a, 52))$
- Dane wyjściowe:
 - Ciąg znaków o długości `-n` ze stosunkiem ilości "a" do długości ciągu `-d`.

4. Metoda rozwiązania

W celu rozwiązania zadania wykorzystałem tablicę prefikso-prefiksową. Tablica ta podaje wartość najdłuższego wspólnego słowa dla całego wyrazu oraz jego podsłowa zaczynającego się na pozycji j , gdzie $j > 0$ (indeksując od 0). Algorytm budujący tę tablicę wykorzystuje informacje o budowie wyrazu na podstawie wcześniej policzonych wartości najdłuższych wspólnych prefiksów.

Zastosowanie algorytmu:

Algorytm został wykorzystany w celu policzenia sumy najdłuższych prefiksów całego słowa dla kolejnych sufiksów.

Złożoności:

Dla każdego ciągu zostaje wytworzona tablica o długości ciągu. Złożoność pamięciowa algorytmu jest równa $O(N)$, wpływa na to potrzeba stworzenia dodatkowej tablicy o długości N . Na złożoność czasową nie wpływa wielkość alfabetu. Na podstawie przeprowadzonych testów potwierdziłem, że nie ma różnicy przy zmianie wielkości alfabetu. Algorytm w optymalny sposób wyszukuje sumę podobieństw częściowych.

Dla programu testowego, czyli trybu czwartego przygotowałem dodatkowo rozwiązanie problemu o złożoności kwadratowej. Algorytm ten sprawdza dla każdego ciągu po kolei każdy sufiks porównując go z ciągiem wejściowym. Metoda ta ma złożoność $O(N^2)$ równolegle dla tych samych ciągów zostaje sprawdzany wynik dla optymalnego algorytmu $O(N)$.

5. Elementy programu

Program składa się z dwóch plików:

1. aal.py:

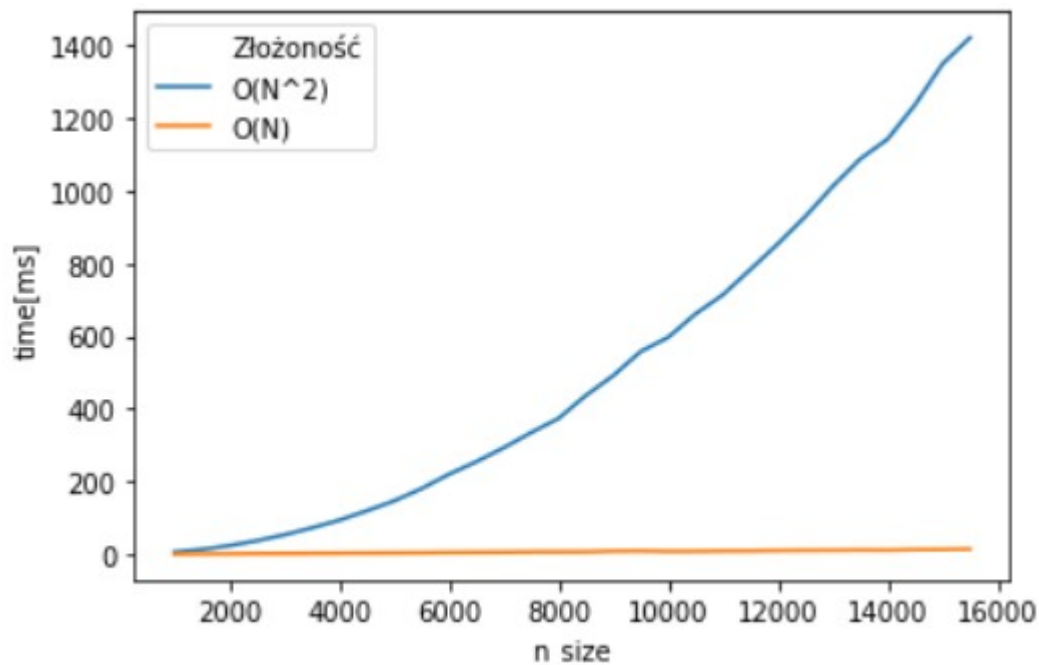
- Odpowiada za przeliczanie 4 trybów pracy. W celu skorzystania z programu na potrzeby interfejsu graficznego można skorzystać z klasy PrefixSum, która posiada wszystkie potrzebne metody do korzystania ze wszystkich trybów pracy. Głównym elementem jest ustawienie odpowiednich atrybutów dla obiektu klasy PrefixSum i wywołanie funkcji run(), która wykonuje żądany tryb. Klasa PrefixSum potrzebuje klasy Generate z pliku gen.py w celu wygenerowania słowa. Następną klasą w pliku aal.py jest klasa ReadTerminal, która posiada metody do odpowiedniego wyodrębnienia informacji z argumentów podanych z terminalu.

2. gen.py:

- Odpowiada za generowanie pseudolosowego ciągu wejściowego.
- W celu skorzystania z programu na potrzeby interfejsu graficznego można skorzystać z klasy Generate, z której następnie można generować słowa o dowolnej długości i o dowolnym stosunku liczby znaków "a" w całym słowie. Klasa ta jest wykorzystywana przez klasę PrefixSum w celu wykonania trybu drugiego oraz trzeciego i czwartego, w których wykorzystywana jest do generowania ciągów testowych.

6. Rezultat

Program umożliwia dwa typy analizy wyników. Pierwszym jest -m4, który przygotowuje dane w formacie wartości oddzielonych przecinkami. Dane tak wygenerowane pozwalają przeprowadzić dodatkowe analizy złożoności czasowej i porównania poprawności wyników dla dwóch algorytmów. Na rycinie pierwszej widać wyniki takiej analizy przeprowadzonej w notebooku [Podobieństwo cząstkowe ciągów.ipynb](#) dla danych wygenerowanych przez wywołanie `pythn aal.py -m4 -n1000 -k30 -step500 -r10 > out.txt`. Jak widać na rycinie algorytm o złożoności liniowej radzi sobie znacznie lepiej.



Ryc. 1 Analiza porównawcza wykonywania programu dla dwóch algorytmów.

Drugim trybem analizy jest tryb aal.py -m3 -n1000 -k30 -step500 -r10, gdzie:

- -m3 -tryb wykonywania
- -n1000 długość słowa w pierwszym problemie
- -k30 liczba problemów
- -step500 różnica długości słowa między dwoma kolejnymi problemami
- -r10 liczba instancji problemu

Program zwraca tabelę z analizą złożoności zaprogramowanego algorytmu dla wybranych przez użytkownika parametrów. Program generuje -r instancji (wyrazów) dla odpowiedniego problemu związanego z długością wyrazu równą długości -n minimalnego słowa powiększoną o numer problemu wymnożony przez -step krok (różnicę długości słów między dwoma kolejnymi problemami). Dla każdej instancji program liczy czas na przeliczenie podobieństwa cząstkowego ciągu w milisekundach. W drugiej kolumnie znajduje się średni czas w milisekundach potrzebny na policzenie podobieństwa cząstkowego ciągu o długości dla danego problemu. Ostatnia kolumna prezentuje współczynnik zgodności oceny teoretycznej z pomiaru czasu dla każdego problemu. Jak widać na załączonej rycinie wartości te oscylują w okolicach pożądanej wartości 1.

###Algorytm z asymptotą $O(T(n))$ ###		
=====		
n	t(n)[ms]	q(n)
=====		
1000	0.85	0.9
1500	1.32	0.93
2000	1.76	0.93
2500	2.22	0.94
3000	2.62	0.93
3500	3.07	0.93
4000	3.44	0.91
4500	3.97	0.93
5000	4.42	0.94
5500	4.79	0.92
6000	5.23	0.92
6500	5.9	0.96
7000	6.71	1.01
7500	7.37	1.04
8000	7.71	1.02
8500	8.02	1.0
9000	8.55	1.01
9500	8.79	0.98
10000	9.39	1.0
10500	9.12	0.92
11000	9.21	0.89
11500	10.74	0.99
12000	11.77	1.04
12500	11.63	0.99
13000	12.75	1.04
13500	12.26	0.96
14000	12.67	0.96
14500	13.74	1.0
15000	13.27	0.94
15500	14.16	0.97
=====		

Ryc. 2 Wynik wykonywania programu dla analizy teoretycznej algorytmu wykonanej przez program.