Algorytmy Tekstowe - Łukasz Sochacki

Raport grupa A środa 16:15 – 17:45

Raport

1. Implementacja algorytmów

1.1. Algorytm naiwny

Poniżej znajduje się implementacja algorytmu naiwnego znajdującego wystąpienia danego wzorca w danym tekście w postaci funkcji naive.

```
def naive(word, pattern):
    res = 0
    for i in range(len(word) - len(pattern) + 1):
        if word[i:i + len(pattern)] == pattern:
            res += 1
    return res
```

1.2. Automat skończony

W celu zaimplementowania automatu skończonego najpierw została zaimplementowana funkcja przejścia. Mechanizm tworzenia takiej funkcji został zaimplementowany za pomocą funkcji transition_table.

```
def transition_table(pattern):
    result = []
    n = len(pattern)
    signs = []
    for i in range(n):
        if not pattern[i] in signs:
            signs.append(pattern[i])

    for q in range(n + 1):
        result.append({})
        for el in signs:
            k = min(q + 2, n + 1)
            while True:
                k -= 1
                s = pattern[:q] + el
                if pattern[:k] == s[q - k + 1:]:
                    break
            result[q][el] = k

    return result
```

Po zaimplementowaniu funkcji przejścia została zaimplementowana funkcja finite_auto, która wykorzystując funkcję przejścia odnajduje wszystkie wystąpienia danego wzorca w tekście.

```
def finite_auto(word, pattern):
    res = 0
    q = 0  # Stan
    table = transition_table(pattern)  # Funkcja przejscia
    for el in word:
        if el not in table[q]:
            q = 0
        else:
            q = table[q][el]
            if q == len(table) - 1:
                res += 1
    return res
```

1.3. Algorytm KMP

Najpierw została zaimplementowana funkcja prefiksowa prefix_function w celu późniejszego wykorzystania podczas wykonywania algorytmu KMP.

```
def prefix_function(pattern):
    prefix = [0]
    k = 0
    for q in range(1, len(pattern)):
        while k > 0 and pattern[k] != pattern[q]:
            k = prefix[k - 1]
        if pattern[k] == pattern[q]:
            k = k + 1
        prefix.append(k)
    return prefix
```

Następnie została zaimplementowana funkcja kmp wyszukująca wystąpienia danego wzorca w podanym tekście.

```
def kmp(word, pattern):
    pi = prefix_function(pattern)
    q = 0
    res = 0
    for i in range(0, len(word)):
        while q > 0 and pattern[q] != word[i]:
            q = pi[q - 1]

        if pattern[q] == word[i]:
            q = q + 1

        if q == len(pattern):
            res += 1
            q = pi[q - 1]
    return res
```

2. Zaimplementowane testy

W celu przetestowania zaimplementowanych algorytmów pod względem poprawności znajdywania wzorca w tekście oraz szybkości znajdywania go zostały zaimplementowane następujące testy:

1. Znalezienie wystąpień wzorca „Art” w załączonej ustawie znajdującej się w pliku **1997_714.txt**.
2. Znalezienie wystąpień wzorca „kruszwil” we fragmencie polskiej Wikipedii znajdującej się w pliku **wikipedia-tail-kruszwil.txt**.
3. Ilość wystąpień wzorca „0fjas” w tekście składającym się z 666 kopii napisu:

„0iajsd-0fjas-9fjk-asjfg-0aiesdjnmtg-0iesjmtg0iesjw=0ijesr0thnwa9ejnmoaiwtnaseptghn-a0defjk0aisdjg0a]hntg-anp0j”.

4. Test znajdujący wystąpienia wzorca „aaaaaaa” w tekście składającym się z 542 kopii napisu: **"aaaaaaaaaaaaaaaaaaaaaaabbbbbbbbbbbbbbbbbbbbbbbbbcccccccccccccccccccccccccccdddddddddddddddddddddddddddddddeeeeeeeeeeeeeeeeeeeeeeeeeffffffffffffffgggggggggghhhhhhhhhhhhhiiiiiiiiiiiijjjjjjjjjjjjjjkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkklllllllllllllmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmm".**
5. Test szukający wystąpień wzorca „dobrze” w tekście składającym się z 1111 kopii napisu : **"Jak zrobic cos by to bylo dobrze zrobione, a jednoczeniesnie by nie bylo to za dobrze. dobrze jedziemy z tym xd" .**

3. Porównanie osiągniętych wyników czasowych oraz ilości znalezionych wystąpień wzorca.

Po przygotowaniu testów nastąpiło ich wykonanie oraz zapisanie uzyskanych rezultatów wyświetlonych na wyjściu. Poniżej zostały przedstawione rezultaty osiągnięte przez poszczególne algorytmy dla konkretnych testów.

Test nr 1. (Wzorzec „Art”)

Algorytm	Naiwny	Automat skończony	KMP
Ilość wystąpień danego wzorca	58	58	58
Czas wykonywania algorytmu w sek.	0.047998666763305664	0.015005350112915039	0.039005756378173833

Test nr 2. (Wzorzec „kruszwil”)

Algorytm	Naiwny	Automat skończony	KMP
Ilość wystąpień danego wzorca	13	13	13
Czas wykonywania algorytmu w sek.	57.117969274520874	28.861031770706177	49.03200316429138

Test nr 3. (Wzorzec „0fjas”)

Algorytm	Naiwny	Automat skończony	KMP
Ilość wystąpień danego wzorca	666	666	666
Czas wykonywania algorytmu w sek.	0.0149993896484375	0.008036613464355469	0.013998270034790039

Test nr 4. (Wzorzec „aaaaaaa”)

Algorytm	Naiwny	Automat skończony	KMP
Ilość wystąpień danego wzorca	9214	9214	9214
Czas wykonywania algorytmu w sek.	0.034998178482055664	0.010001420974731445	0.032000064849853516

Test nr 5. (Wzorzec „dobrze”)

Algorytm	Naiwny	Automat skończony	KMP
Ilość wystąpień danego wzorca	3333	3333	3333
Czas wykonywania algorytmu w sek.	0.02199697494506836	0.013007879257202148	0.02096271514892578

4. Czas działania algorytmów w szczególnych przypadkach

4.1. Funkcja przejścia w algorytmie KMP oraz automacie skończonym

Funkcja przejścia obecna w algorytmie KMP oraz automacie skończonym jest wykorzystywana do znalezienia danego wzorca w tekście, ale metoda tworzenia samej funkcji jest różna w zależności od algorytmu. Z tego powodu zależnie od podanego wzorca funkcja przejścia może być szybsza/wolniejsza w porównaniu z drugą wersją tej funkcji dla innego algorytmu.

Przykładowo dla pewnych wzorców funkcja przejścia w algorytmie KMP może się okazać pięciokrotnie szybsza od utworzenia tabeli przejścia w automacie skończonym. Żeby osiągnąć taki rezultat należy utworzyć taki wzorzec, by posiadał jak najwięcej unikatowych znaków w sobie (przykładowo

takim wzorem jest napis składający się z 25 kopii napisu „Ale dobrzy ruch trczx sfj”). Duża ilość unikatowych znaków powoduje spowolnienie działania tworzenia funkcji przejścia w automacie skończonym, ponieważ zależy również od ilości unikatowych znaków, gdzie funkcja w algorytmie KMP nie traci przez ten fakt na szybkości działania.

Szybkość tworzenia funkcji przejścia dla danego algorytmu.		
Algorytm	Automat skończony	Algorytm KMP
Czas tworzenia funkcji w sek.	1.6400001049041748	0.0010004043579101562

4.2. Szybkość działania dopasowania wzorca w szczególnych przypadkach

Poszczególne algorytmy wyszukiwania wzorca w tekście w zależności od podanego wzorca oraz tekstu potrafią znacznie zyskiwać na płynności działania. Możliwe jest nawet osiągnięcie pięciokrotnie szybszego czasu działania między poszczególnymi algorytmami. Przykładowo chcąc pokazać, że algorytm KMP oraz automat skończony dopasowują pewien wzorzec minimalnie pięciokrotnie szybciej od algorytmu naiwnego możemy utworzyć pewien tekst oraz wzorzec składający się z bardzo wielu kopii pojedynczego znaku. Przykładowymi wzorcem może być wyraz złożony z 66666 kopii znaku ”X” oraz tekst złożony z 1234567 kopii znaku ”X”. Po osiągniętych rezultatach można wysnuć wniosek, że dla specyficznych wzorców oraz tekstów algorytm naiwny okazuje się wybór innego algorytmu niż naiwny znacznie wpływa na szybkość działania programu.

Szybkość znajdywania wzorca w tekście dla poszczególnych algorytmów (nie licząc czasu preprocessingu)			
Algorytm	Automat skończony	Algorytm KMP	Algorytm naiwny
Czas tworzenia funkcji w sek.	0.27500152587890625	0.47300243377685547	7.3030009269714355