POLITECHNIKA RZESZOWSKA

im. Ignacego Łukasiewicza

WYDZIAŁ MATEMATYKI I FIZYKI STOSOWANEJ

PROJEKT Z ALGORYTMÓW I STRUKTUR DANYCH
STUDENTA PIERWSZEGO ROKU STUDIÓW
KIERUNKU INŻYNIERIA I ANALIZA DANYCH

ZERO A JEDEN 1.0

VITALII MORSKYI

Rzeszów 2020

SPIS TREŚCI

OPIS PROBLEMU

W danym rozdziale chciałbym opisać zadanie, które otrzymałem do wykonania.

Dla głównego zadania projektu program musi umieć wczytywać i wypisywać dane z plików tekstowych. Dane wejściowe są podane w postaci tablicy i przedstawiają sobą ciąg zawierający wyłącznie wartości 0 lub 1. Główna funkcja musi odnaleźć wszystkie najdłuższe podciągi, zawierające równą liczbę zer i jedynek.

Teraz już możemy określić najważniejsze zadania naszego algorytmu. Zaczniemy od początku - wczytywania danych. Ponieważ praktycznie nic nie wiemy o tym jak będzie wyglądał plik wejściowy, to musimy wykorzystać taki algorytm, żeby mógł wczytywać dane w poprawny sposób niezależnie od tego jak oni są zapisane. Na przykład, są podane trzy ciągi: „0,0,1,0,1,0,0", „0010100" i „0 0 1 0 1 0 0". Poprawny algorytm musi rozumieć, że one są identyczne. Także prawdopodobnym wydaje się fakt, że będziemy chcieli używać tego algorytmu do wielu ciągów jednocześnie. Dla tego program, który mógł by wczytywać dużo tablic z jednego pliku byłby bardziej użyteczny.

Ale oczywiście istnieją przypadki, kiedy chcemy generować dane wejściowe automatycznie. Dla tego byłoby dobrze, gdyby algorytm posiadał opcję automatycznego tworzenia plików wejściowych.

Kolejnym zadaniem programu jest wyeliminowanie możliwości najprostszych przypadków, kiedy w podanym ciągu:

- jest równa ilość zer i jedynek;
- są wyłącznie zera lub wyłącznie jedynki.

Tylko po takim wyeliminowaniu możemy już zaczynać wyszukiwać podciągi.

I na koniec dobrze by było, gdyby nasz program był w stanie poinformować użytkownika o takich danych jak: jaki plik program próbował odczytać, w którym zapisał wyniki, ile ciągów znalazł w pliku wejściowym oraz ile czasu zajęło jego działanie.

Podsumowując, nasz algorytm musi być w stanie:

1) odczytać wejściowy plik tekstowy niezależnie od formy zapisu danych;
2) odczytywać dużą liczbę ciągów z jednego pliku;
3) wyeliminować najprostsze przypadki;
4) wyszukać najdłuższe podciągi w najbardziej szybki sposób;
5) poinformować użytkownika o przydatnych rzeczach.

W następnym rozdziale wyjaśnię bardziej szczegółowo, w jaki sposób mój program wykonuje każdy z wcześniej wymienionych punktów.

ANALIZA ALGORYTMU

W danym rozdziale chciałbym szczegółowo wyjaśnić każdy ułamek mojego algorytmu.

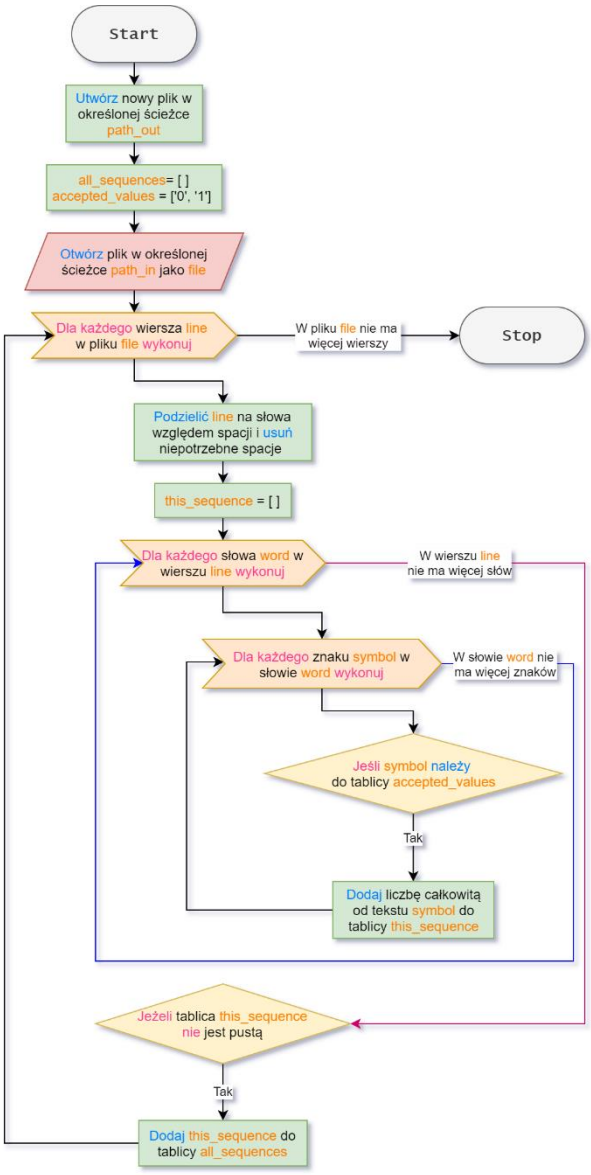
WCZYTYWANIE CIĄGÓW Z PLIKÓW

Już wcześniej zaznaczyłem, że poprawny algorytm musi być w stanie wczytywać dane w poprawny sposób niezależnie od ich formy zapisu. Dla tego myślę, że najprostszym rozwiązaniem takiego problemu byłoby sprawdzanie czy każdy element jest jedynką lub zerem. Oczywiście taki sposób będzie działać troszeczku dłużej, niż identyczny sposób bez sprawdzania każdego elementu, ale dlatego, że działanie całego programu zależy od

poprawności tej funkcji, to już lepiej zostawić tutaj taki algorytm, który będzie dawał poprawny wynik w największej ilości przypadków, zwłaszcza że jego czas wykonania zajmuje mniej niż 2 procent czasu wykonania całego algorytmu.

Pseudokod działania takiego algorytmu oraz schemat blokowy (Schemat blokowy 1) znajdują się poniżej. W kodzie źródłowym funkcja, która wykonuje te działania, nazywa się *reading_file()* i znajduje się w klasie *binary_sequences*.

SCHEMAT BLOKOWY FUNKCJI *reading_file()*:



Schemat blokowy 1 read_file()
Źródło: opracowanie własne.

PSEUDOKOD FUNKCJI *reading_file()*:

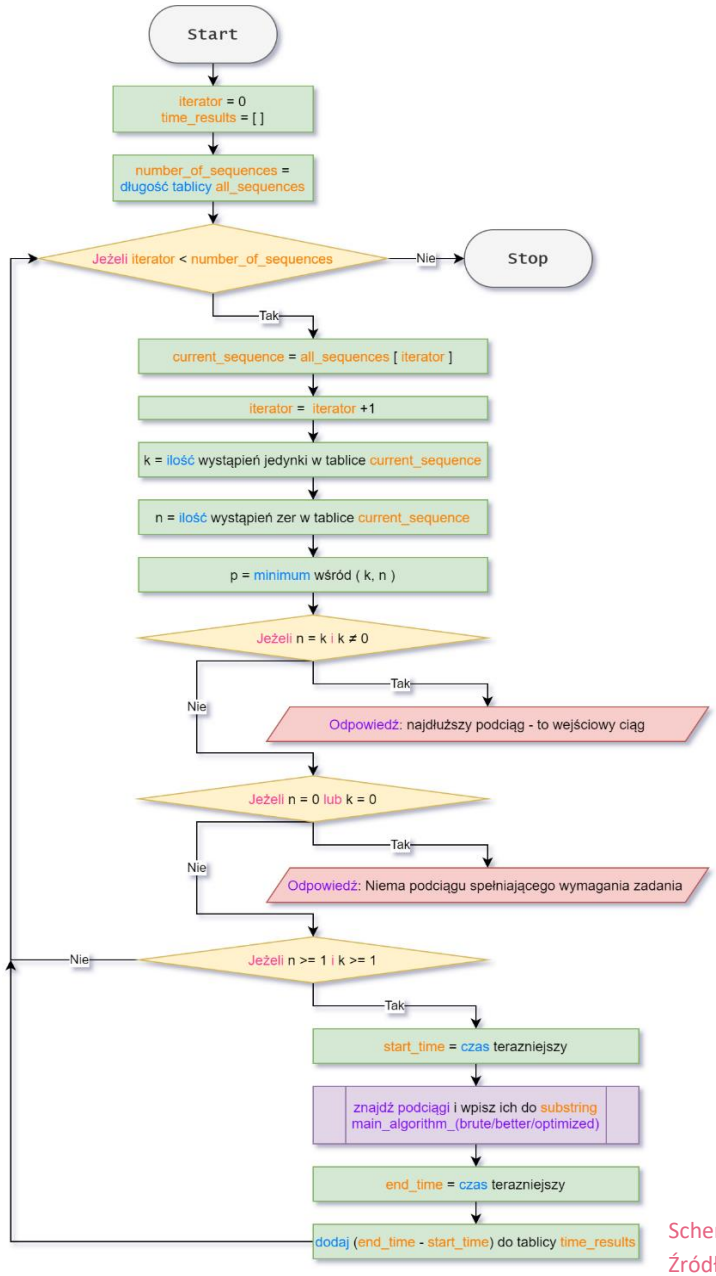
```
K01:  Utwórz nowy plik w określonej ścieżce path_out        // Tworzenie wyjściowego pliku
K02:  all_sequences= [ ]
K03:  accepted_values = ['0', '1']
K04:  Otwórz plik w określonej ścieżce path_in jako file
K05:  Dla każdego wiersza line w pliku file wykonuj:
K06:  |     Podzielić line na słowa względem spacji
K07:  |     Usuń niepotrzebne spacje z wierszu line
K08:  |     this_sequence = [ ]
```

```
K09:  |     Dla każdego słowa word w wierszu line wykonuj:
K10:  |     |     Dla każdego symbolu symbol w słowie word wykonuj:
K11:  |     |     Jeśli symbol należy do accepted_values, to
K12:  |     |     |     Dodaj int(symbol) do tablicy this_sequence
K13:  |     Jeśli this_sequence ≠ [ ], to
K14:  |     |     Dodaj this_sequence do tablicy all_sequences
K15:  Zakończ
```

WYELIMINOWANIE NAJPROSTSZYCH PRZYPADKÓW

Żeby wyeliminować takie ciągi, w których już od początku niema zer lub jedynek, a także ciągi, w których liczba zer i jedynek jest równa, potrzebujemy wiedzieć, ile jest jedynek i zer w ciągu. Poniżej znajdują się pseudokod i schemat blokowy (Schemat blokowy 2) funkcji, która wykonuje powyżej opisane działania. W kodzie źródłowym funkcja, która wykonuje te działania, nazywa się *solve_problem()* i znajduje się w klasie *binary_sequences*.

SCHEMAT BLOKOWY FUNKCJI *solve_problem()*:



Schemat blokowy 2 solve_problem()
Źródło: opracowanie własne.

PSEUDOKOD FUNKCJI *solve_problem()*:

```
K01:  iterator = 0
K02:  time_results = [ ]
K03:  number_of_sequences = długość (all_sequences)
K04:  Dopóki iterator < number_of_sequences wykonuj:
K05:  |     current_sequence = all_sequences [ iterator ]
K06:  |     iterator += 1
K07:  |     k = zlicz ilość wystąpień jedynki w tablice current_sequence
K08:  |     n = zlicz ilość wystąpień zer w tablice current_sequence
K09:  |     p = minimum wśród (k, n)
K10:  |     Jeżeli n = k i k ≠ 0, to:
K11:  |     |     Odpowiedź: najdłuższy podciąg - to wejściowy ciąg      // Funkcja odpowiedź w kodzie
      źródłowym posiada nazwę give_answer()
K12:  |     W przeciwnym razie, jeśli n = 0 lub k = 0, to:
K13:  |     |     Odpowiedź: Niema podciągu spełniającego wymagania zadania
K14:  |     W przeciwnym razie, jeśli n >= 1 i k >= 1, to:
K15:  |     |     start_time = czas teraźniejszy
K16:  |     |     substring = main_algorithm_(brute/better/optimized)   // Wybieramy algorytm zgodnie z
      decyzją użytkownika
K17:  |     |     end_time = czas teraźniejszy
K18:  |     |     dodaj (end_time - start_time) do tablicy time_results
K16:  Zakończ
```

GŁÓWNY ALGORYTM WYSZUKIWANIA PODCIĄGÓW

Starałem się maksymalnie zoptymalizować algorytm, więc teraz mam kilka wersji mojego algorytmu.

PODSTAWOWY ALGORYTM WYSZUKIWANIA PODCIĄGÓW

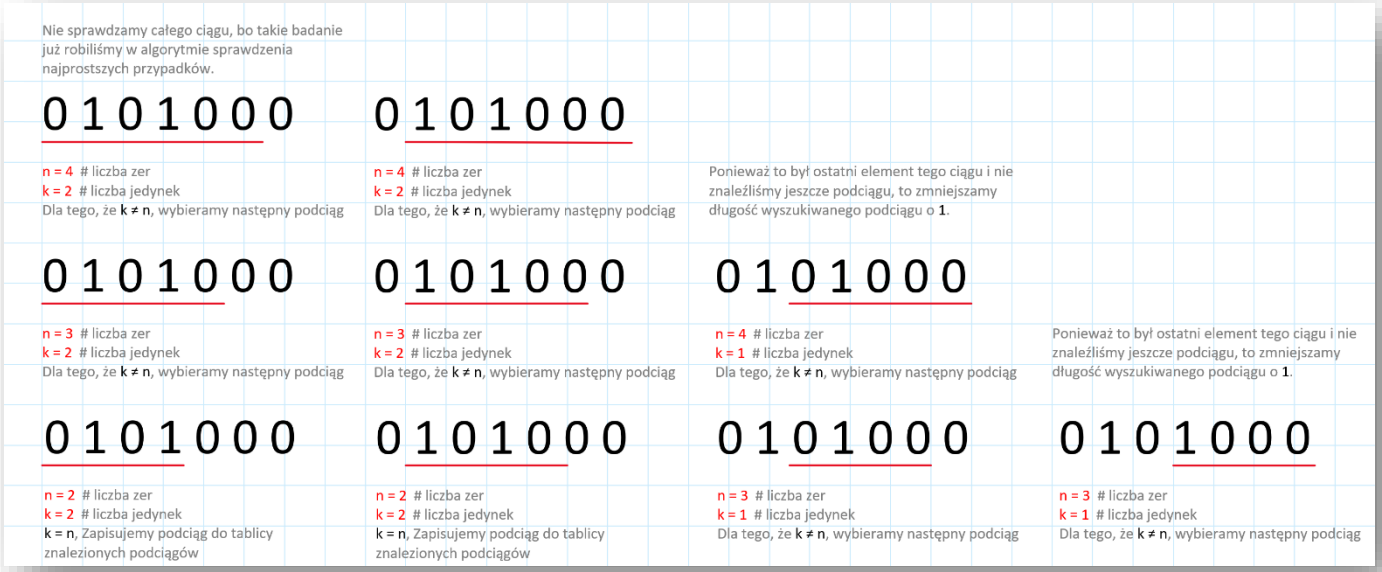
Nasz algorytm ma znaleźć różne najdłuższe podciągi, w których liczba zer i jedynek jest równa. Dlatego możemy sprawdzać wszystkie możliwe podciągi licząc ilości zer i jedynek w każdym. Ponieważ potrzebujemy odszukać najdłuższy podciąg, to będziemy zaczynać od najdłuższego i iść do najkrótszego możliwego podciągu, żeby nie sprawdzać zbędne podciągi.

Na przykład bierzemy ciąg:

0 1 0 1 0 0 0

Oczywistym jest fakt, że ma dwa najdłuższe podciągi:

0 1 0 1 oraz 1 0 1 0



Schemat 1 Kroki działania aktualnego algorytmy dla ciągu 0101000.
Źródło: opracowanie własne.

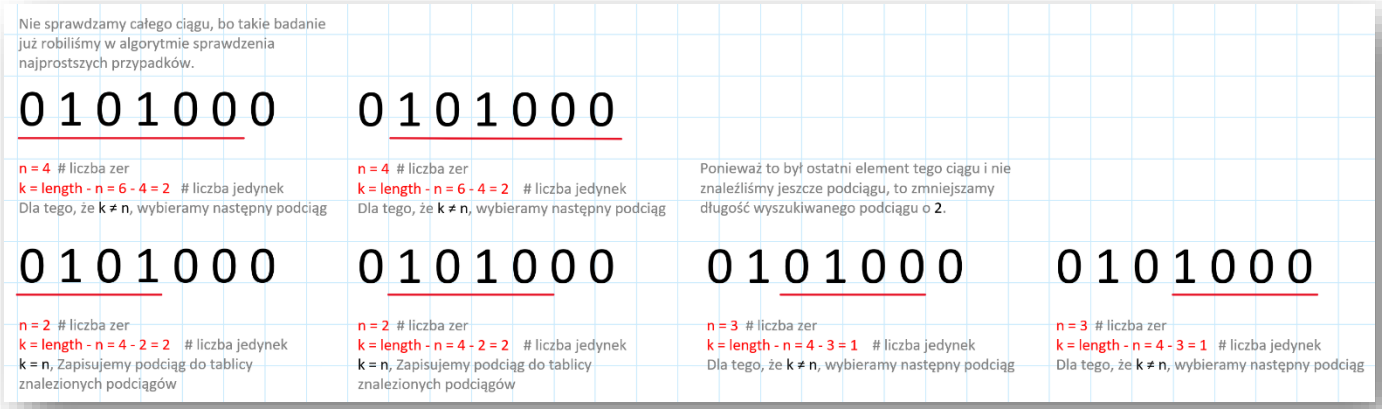
Na schemacie działania mojego algorytmu (Schemat 1) widać, że program bada każdy podciąg. Tylko wtedy, kiedy ilość zer i jedynek jest równa, algorytm dopisuje podciąg do tablicy wyników. Tak jak funkcja #2 eliminuje możliwość nieistnienia podciągu, to taki algorytm zawsze potrafi znaleźć podciąg.

Wiemy, że liczba wystąpień jedynek ma być równą liczbie wystąpień zer. Czyli długość całego podciągu:

$$length = k + n = 2 * n = 2 * k$$

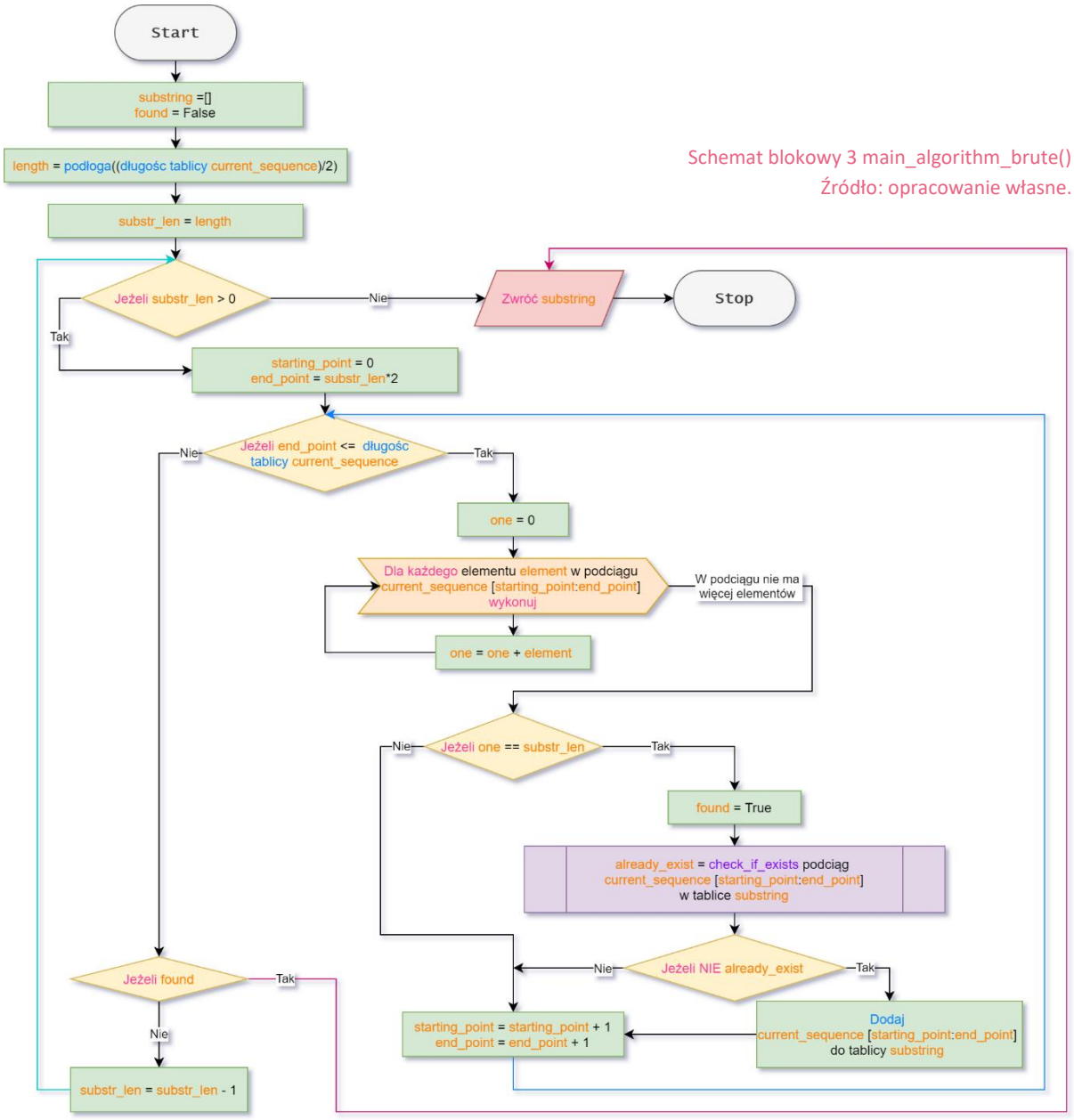
To znaczy, że możemy sprawdzać tylko parzyste długości podciągów, skracając w ten sposób czas potrzebny do wykonania programu. Skoro będziemy mieć tylko parzyste podciągi, to możemy zliczać tylko ilość któregoś jednego z elementów (0, 1) i porównywać ją z połową długości podciągu, co także zmniejsza złożoność algorytmu.

Poszczególne etapy realizacji takiego algorytmu przedstawiono na poniższym schemacie (Schemat 2). Jego pseudokod oraz schemat blokowy (Schemat blokowy 3; Schemat blokowy 4) są podane poniżej; w kodzie źródłowym ten algorytm posiada nazwę *main_algorithm_brute()* i znajduje się w klasie *binary_sequences*.



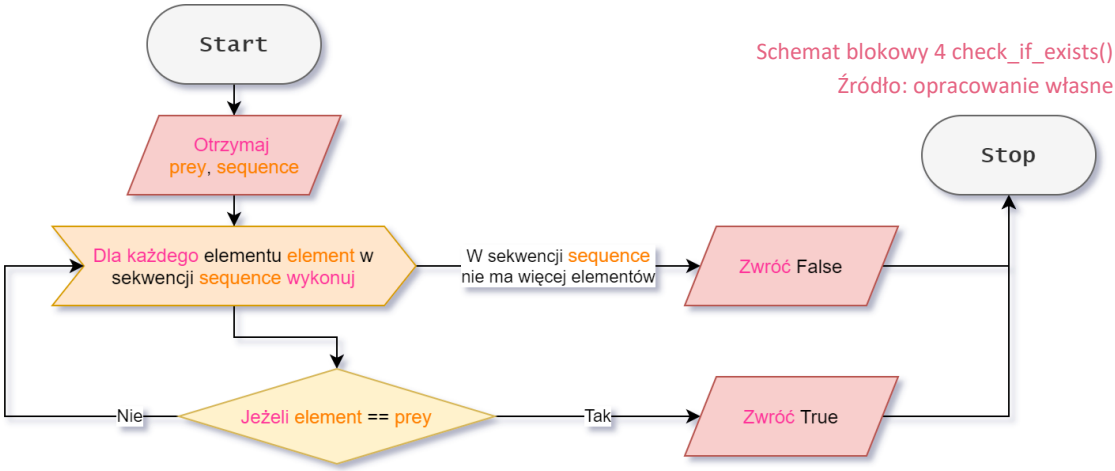
Schemat 2 Kroki działania aktualnego algorytmy dla ciągu 0101000.
Źródło: opracowanie własne.

SCHEMAT BLOKOWY FUNKCJI *main_algorithm_brute()*:



Schemat blokowy 3 main_algorithm_brute()
Źródło: opracowanie własne.

SCHEMAT BLOKOWY FUNKCJI *check_if_exists()*:



Schemat blokowy 4 check_if_exists()
Źródło: opracowanie własne

PSEUDOKOD FUNKCJI *main_algorithm_brute()*:

```
K01:  substring =[]
K02:  found = False
K03:  length = podłoga((długośc tablicy current_sequence)/2)
K04:  substr_len = length
K05:  Dopóki substr_len > 0 wykonuj K06:K24
K06:  |     starting_point = 0
K07:  |     end_point = substr_len*2
K08:  |     Dopóki end_point <=  długośc tablicy current_sequence wykonuj K09:K21
K09:  |     |     one = 0
K10:  |     |     Dla każdego elementu element w podciągu current_sequence [starting_point:end_point]
      wykonuj K11:
K11:  |     |     |     one = one + element
K12:  |     |     Jeżeli one == substr_len wykonuj K13:K19
K13:  |     |     |     found = True
K14:  |     |     |     already_exist = False
K15:  |     |     |     Dla każdego elementu element w tablice substring wykonuj K16:K17      // Ta część
      pseudokodu (K14:K17) w schemacie blokowym, zarówno jak i w kodzie źródłowym, oznaczona przez
      funkcję „check_if_exists()”, bo powtarza się kilka razy w całym programu.
K16:  |     |     |     |     Jeżeli element == current_sequence [starting_point:end_point] wykonuj K17
K17:  |     |     |     |     |     already_exist = True
K18:  |     |     |     Jeżeli NIE already_exist wykonuj K19
K19:  |     |     |     |     Dodaj current_sequence [starting_point:end_point] do tablicy substring
K20:  |     |     starting_point = starting_point + 1
K21:  |     |     end_point = end_point + 1
K22:  |     Jeżeli found wykonuj K23
K23:  |     |     break
K24:  |     substr_len = substr_len – 1
K25:  Zwróć substring
K26:  Zakończ
```

OPTYMALIZOWANIE ALGORYTMU WYSZUKIWANIA PODCIĄGÓW

Możemy zacząć od tego, że po prostu nie musimy sprawdzać wszystkich podciągów, a tylko te, których długość jest mniejszą lub równą dwukrotnej liczbie elementów, których jest mniej w ciągu. Czyli będziemy sprawdzać tylko te podciągi, długość których jest teoretycznie możliwą. Na przykład, w już wcześniej wspominanym ciągu 0 1 0 1 0 0 0 maksymalna teoretyczna długość podciągu będzie zależała od ilości jedynek dlatego, że ich jest mniej. Czyli:

$$theoretical\ subsequence\ length = 2 * \min(n, k) = 2 * 2 = 4$$

Weźmiemy na przykład inny ciąg, długość którego posiada 20 symboli:

$$1\,0\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,0\,1\,0$$

W tym ciągu liczba zer $n = 3$, a liczba jedynek $k = 17$. Nasz algorytm już wie, że maksymalna teoretyczna długość podciągu wynosi:

$$theoretical\ subsequence\ length = 2 * \min(n, k) = 2 * 3 = 6$$

Chociaż zaczynamy szukać podciągów o długości co najmniej 6 znaków, nadal wykonujemy wiele zbędnych obliczeń. Na przykład na tym schemacie (Schemat 3) widać, że co najmniej trzy razy obliczamy ilość jedynek w zielonym prostokącie chociaż od początku wiemy, że cztery jedynki nie mogą znajdować się w podciągu z sześciu elementów.

Schemat 3 Kroki działania aktualnego algorytmy dla ciągu 101111111111111111010.
Źródło: opracowanie własne.

Moim zdaniem w najlepszy sposób rozwiązuje ten problem następne ulepszenie: kiedy wiemy dokładnie ilość zer i jedynek w złym podciągu, to możemy zaprognozować, gdzie może znajdować się najbliższy poprawny podciąg za pomocy następnej formuły:

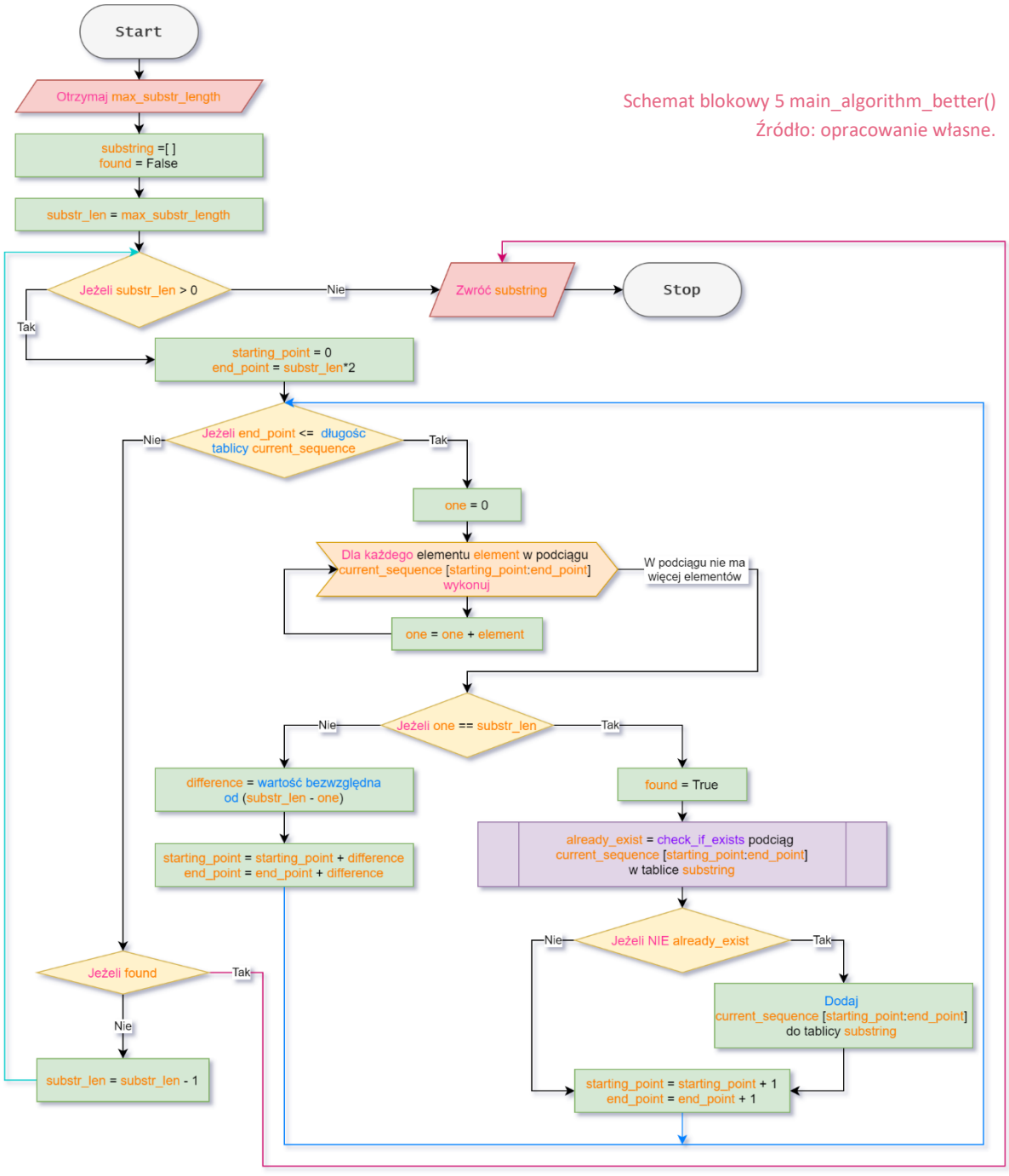
$$difference = \left|\frac{length}{2} - k\right| = \left|\frac{length}{2} - n\right|,$$

Schemat 4 Kroki działania aktualnego algorytmy dla ciągu 101111111111111111010.
Źródło: opracowanie własne.

gdzie difference – to odległość pomiędzy początkami aktualnego i następnego podciągów. Inaczej mówiąc, gdy w aktualnym podciągu jest na $\left|\frac{length}{2} - k\right| = \left|\frac{length}{2} - n\right|$ elementów więcej niż powinno być, to znaczy, że musimy przenieść się przynajmniej na $\left|\frac{length}{2} - k\right| = \left|\frac{length}{2} - n\right|$ elementów po ciągu, żeby uzupełnić warunek równości zer a jedynek. Schemat działania kolejnego ulepszonego algorytmu pokazany na Schemat 4Schemat blokowy 5.

Schemat blokowy (Schemat blokowy 5) i pseudokod tego algorytmu podaję poniżej, w programie ten algorytm nazywa się *main_algorithm_better()* i znajduje się w klasie *binary_sequences*.

SCHEMAT BLOKOWY FUNKCJI *main_algorithm_better()*:



Schemat blokowy 5 main_algorithm_better()
Źródło: opracowanie własne.

PSEUDOKOD FUNKCJI *main_algorithm_better()*:

```
K01:  Otrzymaj max_substr_length            // Funkcja przejmuje taki argument
K02:  substring =[]
K03:  found = False
K04:  substr_len = max_substr_length
K05:  Dopóki substr_len > 0 wykonuj K06:K28
K06:  |     starting_point = 0
K07:  |     end_point = substr_len*2
K08:  |     Dopóki end_point <=  długośc tablicy current_sequence wykonuj K09:K25
K09:  |     |     one = 0
K10:  |     |     Dla każdego elementu element w podciągu current_sequence [starting_point:end_point]
      wykonuj K11:
K11:  |     |     |     one = one + element
K12:  |     |     Jeżeli one == substr_len wykonuj K13:K21
K13:  |     |     |     found = True
K14:  |     |     |     already_exist = False
K15:  |     |     |     Dla każdego elementu element w tablice substring wykonuj K16:K17     // Ta część
      pseudokodu (K14:K17) w schemacie blokowym, zarówno jak i w kodzie źródłowym, oznaczona przez
      funkcję „check_if_exists()”, bo powtarza się kilka razy w całym programu.
K16:  |     |     |     |     Jeżeli element == current_sequence [starting_point:end_point] wykonuj K17
K17:  |     |     |     |     |     already_exist = True
K18:  |     |     |     Jeżeli NIE already_exist wykonuj K19
K19:  |     |     |     |     Dodaj current_sequence [starting_point:end_point] do tablicy substring
K20:  |     |     |     starting_point = starting_point + 1
K21:  |     |     |     end_point = end_point + 1
K22:  |     |     W przeciwnym razie wykonuj K23:K25:
K23:  |     |     |     difference = abs(substr_len-one)
K24:  |     |     |     starting_point = starting_point + difference
K25:  |     |     |     end_point = end_point + difference
K26:  |     Jeżeli found wykonuj K27
K27:  |     |     break
K28:  |     substr_len = substr_len – 1
K29:  Zwróć substring
K30:  Zakończ
```

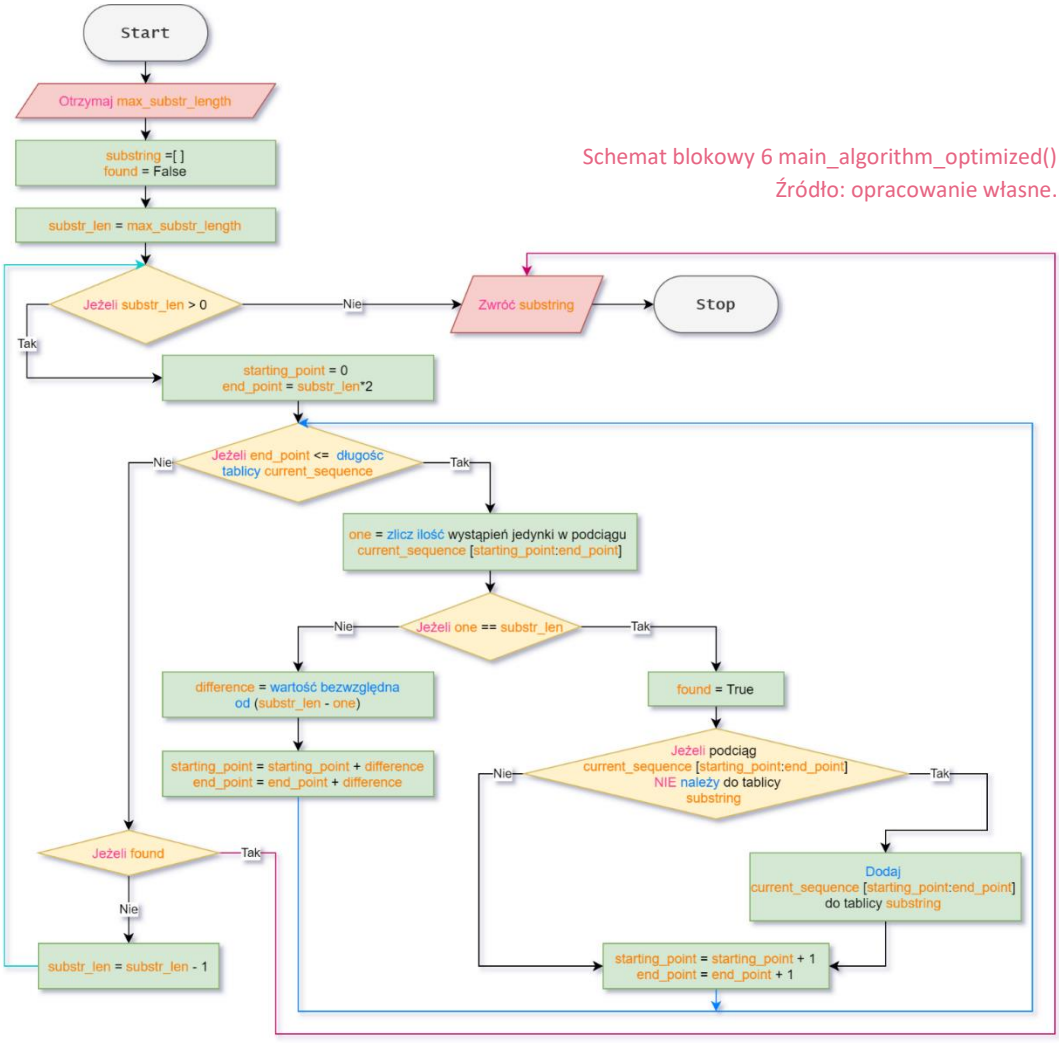
Ostatnia optymizacja, której możemy dokonać – to wykorzystywanie funkcji natywnych języka programowania dla takich działań jako zliczenie ilości wystąpień elementu w tablice i sprawdzenie czy istnieje element w tablice. Taki algorytm jest ustawiony jako domyślny w moim programie. Jego pseudokod oraz schemat blokowy (Schemat blokowy 6) są podane poniżej, w kodzie źródłowym ten algorytm posiada nazwę *main_algorithm_optimized()* i znajduje się w klasie *binary_sequences*.

PSEUDOKOD FUNKCJI *main_algorithm_optimized()*:

```
K01:  Otrzymaj max_substr_length            // Funkcja przejmuje taki argument
K02:  substring =[]
K03:  found = False
K04:  substr_len = max_substr_length
K05:  Dopóki substr_len > 0 wykonuj K06:K22
```

K06: | starting_point = 0
K07: | end_point = substr_len*2
K08: | Dopóki end_point <= długosc tablicy current_sequence wykonuj K09:K15
K09: | | one=zlicz ilość wystąpień jedynki w podciągu current_sequence[starting_point:end_point]
K10: | | Jeżeli one == substr_len wykonuj K11:K16
K11: | | | found = True
K12: | | | Jeżeli podciąg current_sequence[starting_point:end_point] NIE należy do tablicy substring wykonuj K13
K13: | | | | Dodaj current_sequence[starting_point:end_point]do tablicy substring
K14: | | | starting_point = starting_point + 1
K15: | | | end_point = end_point + 1
K16: | | W przeciwnym razie wykonuj K17:K19:
K17: | | | difference = abs(substr_len-one)
K18: | | | starting_point = starting_point + difference
K19: | | | end_point = end_point + difference
K20: | Jeżeli found wykonuj K21
K21: | | break
K22: | substr_len = substr_len – 1
K23: Zwróć substring
K24: Zakończ

SCHEMAT BLOKOWY FUNKCJI *main_algorithm_optimized()*:



Schemat blokowy 6 main_algorithm_optimized()
Źródło: opracowanie własne.

PORÓWNANIE ALGORYTMÓW

Skoro już stworzyliśmy aż 3 algorytmy i twierdzimy, że każdy kolejny jest lepszy od poprzedniego, to musimy jakoś to udowodnić. Aby to zrobić, utworzymy plik wejściowy z dużą liczbą znaków. Niech zawiera 20 losowych ciągów zer i jedynek, z których każdy będzie miał 1000 znaków. Aby to zrobić, użyjemy funkcji *random_sequence()*, którą stworzyłem, aby generować losowe sekwencje zer i jedynek. Znajduje się ona w pliku pomocniczym assistant_module.py. Nazwijmy nowo utworzony plik „report_rand_input.txt". Wynik działania naszego algorytmu zostanie zapisany w pliku „report_rand_output.txt".

Zgodnie z przykładem podanym w funkcji *main()* utworzymy obiekt klasy „binary_sequences" i nazwiemy go „test_1". Przypisując odpowiednie wartości do flag, możemy skorzystać z funkcji *solve_problem()*, która znajdzie wszystkie podciągi i zapisze wyniki do plików źródłowych. Ponieważ mamy zamiar porównać nasze algorytmy, potrzebujemy informacji o wynikach czasu. Aby to otrzymać, używamy funkcji *give_time()*.

Po uruchomieniu programu otrzymujemy wyniki działania najmniej optymalizowanego algorytmu, podobne do tych, które są na Rysunek 1.

```
----- Nowe zadanie z pliku: "report_rand_input.txt" ------------------------------------------------------------------


Przetwarzanie pliku "report_rand_input.txt" : [##########] 100% Gotowy...

Rozpoznano 20 wejściowych ciągów w pliku "report_rand_input.txt". Wszystkie odpowiedzi zostały zapisane w pliku "report_
rand_output.txt".
Wybrany 1 poziom optymalizacji.
Czas roboty algorytmu wynosi 13.246 s.
Średni czas przetwarzania jednego ciągu wynosi 0.6623 s.

Czas przetwarzania każdego podciągu:  [0.807, 0.946, 0.329, 0.258, 0.379, 1.573, 0.643, 0.36, 1.401, 0.013, 0.008, 1.021
, 0.12, 0.507, 0.025, 1.232, 1.372, 1.99, 0.007, 0.255]
```

Rysunek 1 Wynik w konsoli po uruchomieniu najmniej optymalizowanego algorytmu.
Źródło: opracowanie własne.

Widać, że mój komputer wykonał to zadanie w 13 sekund. Ale do porównania algorytmów potrzebujemy wyników co najmniej dwóch algorytmów. Aby uzyskać wyniki drugiego algorytmu zmienimy flagę wyboru algorytmu i ponownie rozpoczniemy wyszukiwanie podciągów za pomocą funkcji *solve_problem()*. Aby nie pisać tych samych trzech wierszy po raz trzeci, wyniki trzeciego algorytmu możemy uzyskać za pomocą funkcji *test()*.

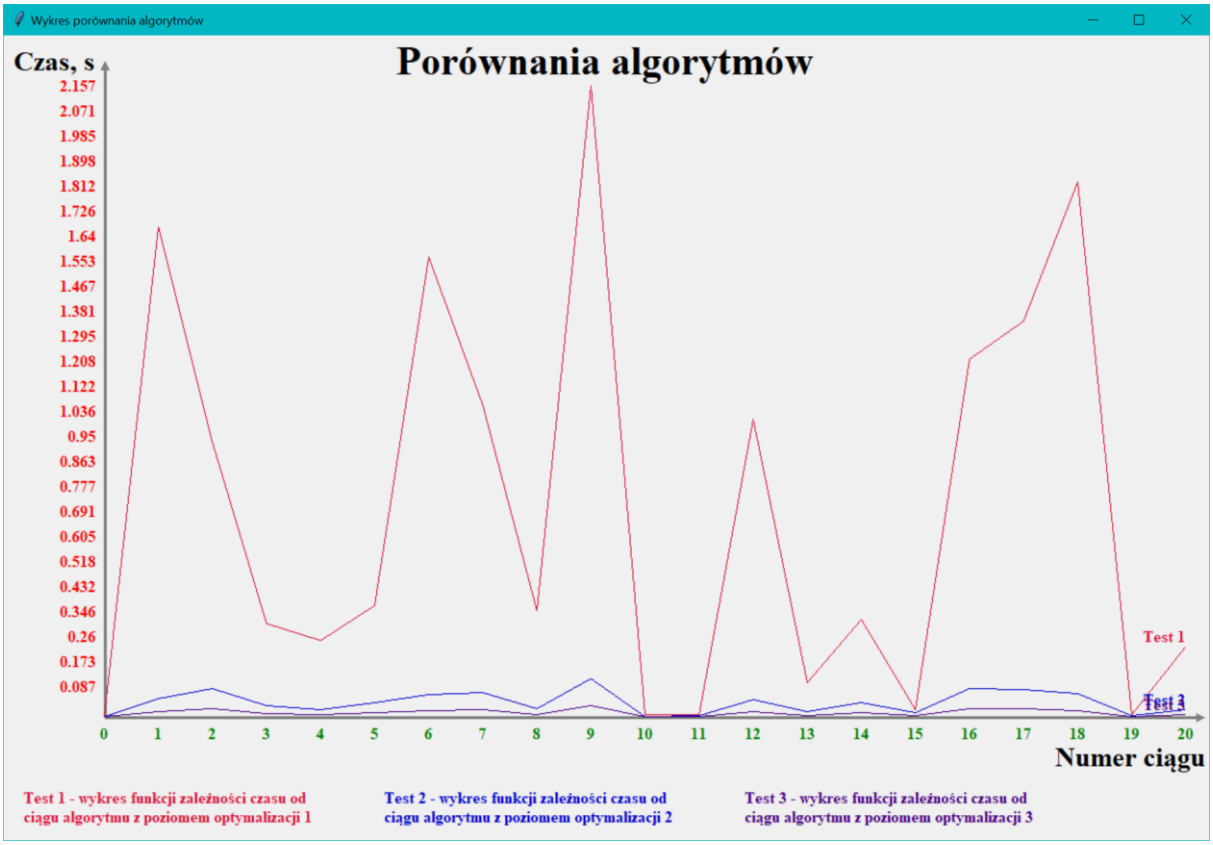
```
Przetwarzanie pliku "report_rand_input.txt" : [##########] 100% Gotowy...

Rozpoznano 20 wejściowych ciągów w pliku "report_rand_input.txt". Wszystkie odpowiedzi zostały zapisane w pliku "report_
rand_output.txt".
Wybrany 2 poziom optymalizacji.
Czas roboty algorytmu wynosi 1.026 s.
Średni czas przetwarzania jednego ciągu wynosi 0.0513 s.

Czas przetwarzania każdego podciągu:  [0.063, 0.096, 0.038, 0.023, 0.048, 0.079, 0.083, 0.027, 0.131, 0.002, 0.003, 0.06
, 0.018, 0.05, 0.008, 0.096, 0.093, 0.082, 0.003, 0.023]




----- Nowe zadanie z pliku: "report_rand_input.txt" ------------------------------------------------------------------


Przetwarzanie pliku "report_rand_input.txt" : [##########] 100% Gotowy...

Rozpoznano 20 wejściowych ciągów w pliku "report_rand_input.txt". Wszystkie odpowiedzi zostały zapisane w pliku "report_
rand_output.txt".
Wybrany 3 poziom optymalizacji.
Czas roboty algorytmu wynosi 0.284 s.
Średni czas przetwarzania jednego ciągu wynosi 0.0142 s.

Czas przetwarzania każdego podciągu:  [0.018, 0.026, 0.01, 0.006, 0.013, 0.021, 0.023, 0.006, 0.037, 0.001, 0.001, 0.017
, 0.005, 0.014, 0.002, 0.028, 0.027, 0.022, 0.001, 0.006]
```

Rysunek 2 Wynik w konsoli po uruchomieniu drugiego i trzeciego algorytmów.
Źródło: opracowanie własne.

Po uruchomieniu programu otrzymujemy wyniki działania trzech algorytmów. Wyniki drugiego i trzeciego algorytmów są podane na Rysunek 2.

Czasy wykonania drugiego i trzeciego algorytmu wynoszą odpowiednio 1 i 0,3 sekundy. W porównaniu do pierwszych 13 sekund widzimy globalną poprawę, więc zoptymalizowane algorytmy działają naprawdę szybciej. Teraz interesuje nas pytanie, o ile szybciej?

W rzeczywistości znacznie przyjemniej byłoby spojrzeć na wyniki jako wykres funkcji osi czasu. Aby to zrobić, możemy skorzystać z funkcji *algorytm_comparison()*, którą napisałem specjalnie do takich celów używając modułu „tkinter".

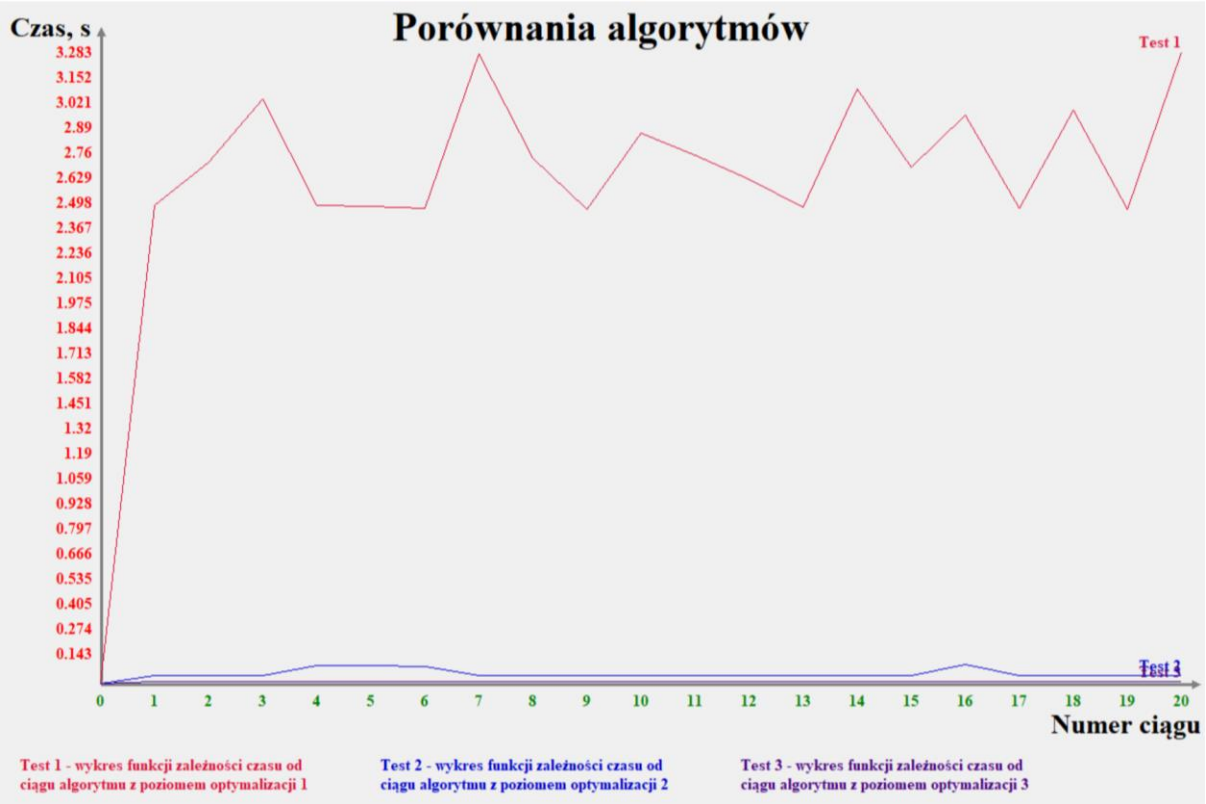


Wykres 1 Porównanie trzech algorytmów w trybie losowych danych wejściowych.
Źródło: opracowanie własne.

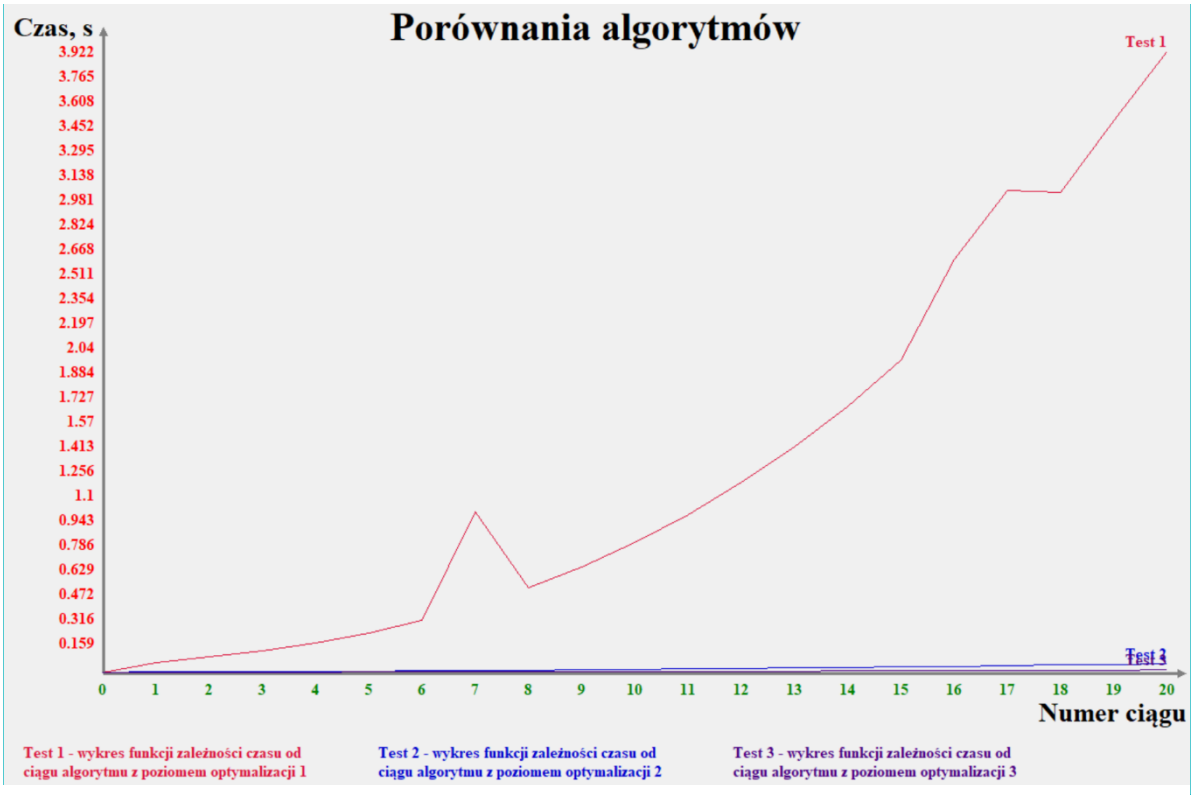
Ta funkcja daje nam wykres pokazany na Wykres 1. Widzimy, że chociaż wszystkie ciągi wejściowe mają identyczną długość, czas wykonania każdej z nich jest inny, ponieważ dane są losowe. Ponadto większość ciągów dążą do ideału, ponieważ dane nie są całkowicie losowe. Wynika to z faktu, że duża liczba liczb pseudolosowych dąży do idealnego podziału 1/1, co oznacza, że nasz algorytm będzie mógł bardzo łatwo znaleźć podciąg z taką samą liczbą jedynek i zer. Aby zobaczyć rzeczywisty czas wyszukiwania podciągów, sugeruję użycie stałego stosunku jedynek do zer, czyli 2/1. Zależność ta będzie równie trudna zarówno dla pierwszego algorytmu, jak i dla trzeciego, ponieważ jeśli będziemy dalej zwiększać stosunek (na przykład 3/1), to trzeci algorytm będzie mógł jeszcze łatwiej znaleźć podciąg w takich sekwencjach, a pierwszemu przeciwnie będzie bardziej trudno.

Aby otrzymać plik wejściowy o stałym stosunku zer do jedynek 2/1, należy skorzystać z funkcji *worst_sequence()*, którą stworzyłem dla tych celów. Znajduje się ona w pliku pomocniczym assistant_module.py. Utworzymy plik wejściowy zawierający 999 zer i jedynek z sekwencją „101101101 ...". Nazwijmy go „report_worst_input.txt", a plik dla wyniku działania naszego algorytmu - „report_worst_output.txt".

Na wykresie (Wykres 3) otrzymujemy bardziej stabilne odczyty, gdzie wahania zależą głównie od tego, jak system operacyjny ładował procesora.



Wykres 3 Porównanie trzech algorytmów w trybie stałych danych wejściowych.
Źródło: opracowanie własne.



Wykres 2 Porównanie trzech algorytmów w trybie zwiększających się danych wejściowych.
Źródło: opracowanie własne.

Do tej pory pracowaliśmy ze stałą liczbą zer i jedynek w ciągu wejściowym. Spróbujemy stworzyć plik, w którym z każdym wierszem liczba jedynek i zer rośnie o 45, zaczynając od 100. Aby to zrobić, należy zmienić parametr „increment” na 15 („101”*15 = 45 symbolów), a „start_repeats” na 100 w poprzednio używanej funkcji *worst_sequence()*. Otrzymujemy wykres gałęzi paraboli (Wykres 2).

DOKUMENTACJA Z DOŚWIADCZEŃ

WCZYTYWANIA DANYCH

Program akceptuje prawie każdy możliwy format danych wejściowych. Na przykład: „1, 0, 1, 0”; „1,0,1,0”; „1 0 1 0”; „1 * 0 * 1 * 0”; „1, qwe0 e1 * 0”; „1010”. Każde z tych wejść deklaruje następującą sekwencję: [1, 0, 1, 0] .W przypadku więcej niż jednego wejścia, należy je oddzielić nową linią. Na przykład to wejście:

1111010111
1101011010
1011101010
1111010101

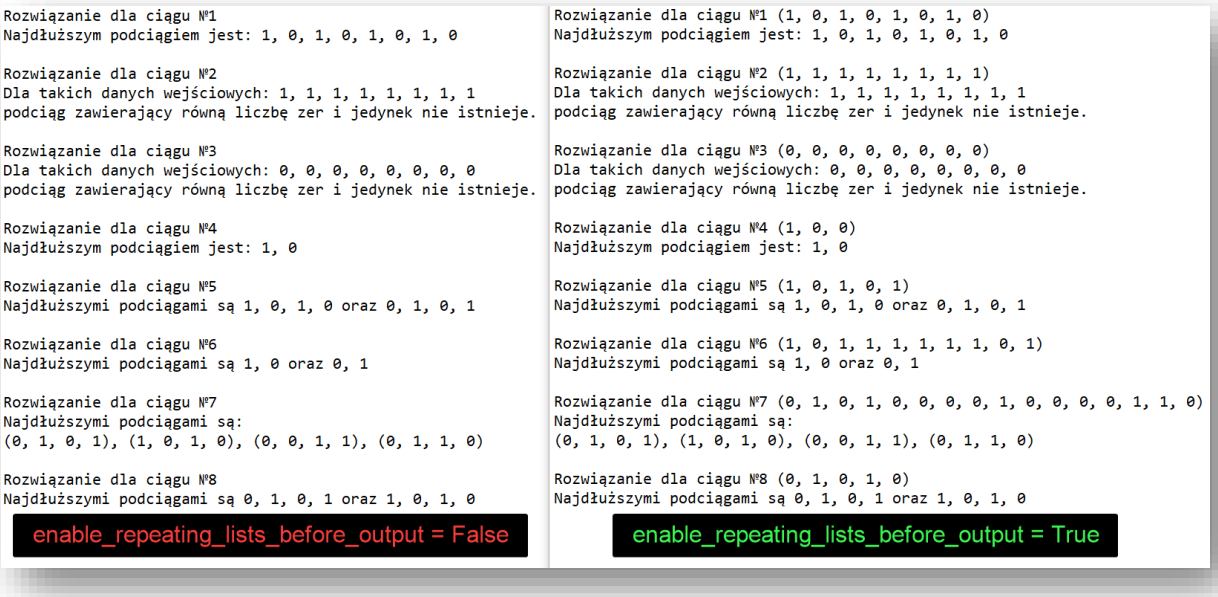
będzie przyjęte jako 4 sekwencji:

[1, 1, 1, 1, 0, 1, 0, 1, 1, 1], [1, 1, 0, 1, 0, 1, 1, 0, 1, 0], [1, 0, 1 , 1, 1, 0, 1, 0, 1, 0], [1, 1, 1, 1, 0, 1, 0, 1, 0, 1]

WYPISYWANIA WYNIKÓW

Przykład standardowych wyników konsoli podano na rysunku numer 1 (Rysunek 1).

Istnieją dwie opcje zapisu podciągów wyjściowych w pliku: z powtarzaniem ciągów wejściowych i bez. Flaga „enable_repeating_lists_before_output” jest odpowiedzialna za wybranie odpowiedniej opcji. Przykład podano na rysunku poniżej (Rysunek 3).

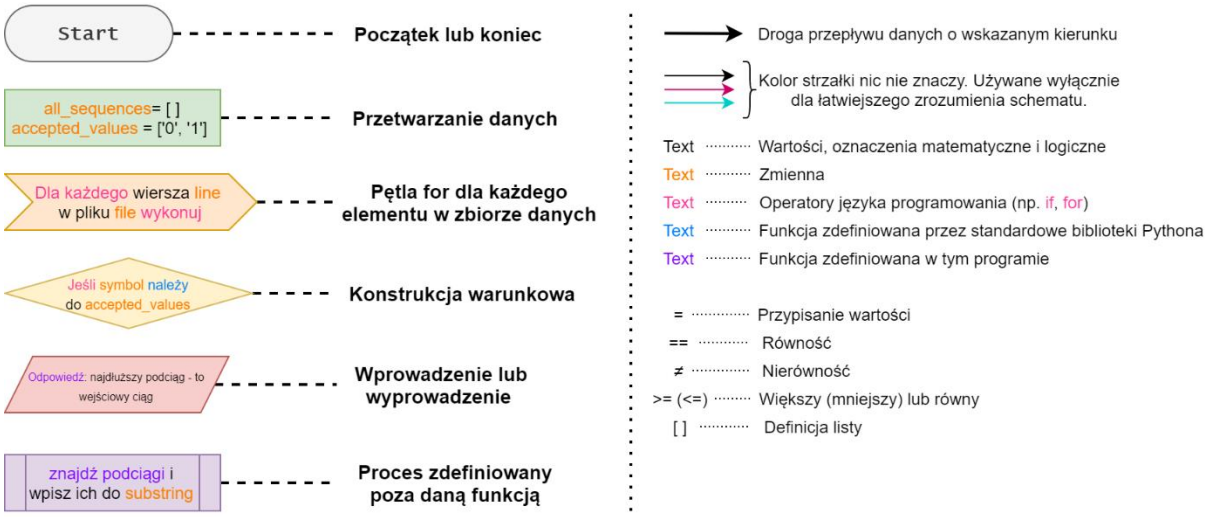


Rysunek 3 Opcje zapisu pliku wyjściowego.
Źródło: opracowanie własne.

WNIOSKI

Opracowałem program, który znajduje wszystkie najdłuższe podciągi ciągu wejściowego, które mają tą samą liczbę zer i jedynek. Aby każdy sposób tworzenia pliku działał z moim programem, stworzono zaawansowaną funkcję czytnika plików, która akceptuje prawie każdy plik. Aby nie tworzyć danych za każdym razem ręcznie, napisano kilka funkcji pomocniczych, które mogą generować dane wejściowe w zależności od wprowadzonych ustawień. Algorytm został trzy razy zoptymalizowany i wszystkie trzy wersje zostały porównane ze sobą. Aby wyraźniej zademonstrować optymalizację algorytmów, stworzono kolejny program rysujący wykresy zależności czasu wykonania algorytmu od każdego podciągu. Wszystkie optymalizacje i ulepszenia zostały jasno przedstawione za pomocą graficznych diagramów głównego algorytmu. Schematy blokowe i pseudokody zostały stworzone dla funkcji odczytywania plików, odrzucania najprostszych przypadków i trzech głównych algorytmów.

LEGENDA PSEUDOKODÓW I SCHEMATÓW BLOKOWYCH



Rysunek 4 Legenda pseudokodów i schematów blokowych.
Źródło: opracowanie własne.

SPISY ODWOŁAŃ

Rysunki:

Schematy blokowe:

Schematy:

Wykresy:

BIBLIOGRAFIA

Foundation, P. S. (2001-2020, November 27). *Python 3.8.6 documentation*. Retrieved from Python 3.8.6 documentation: https://docs.python.org/3.8/

Inc, S. E. (2020). *Stackoverflow*. Pobrano z lokalizacji Python Progress Bar: https://stackoverflow.com/questions/3160699/python-progress-bar

planetB. (2018). *Syntax Highlight Code In Word Documents*. Pobrano 11 27, 2020 z lokalizacji http://www.planetb.ca/syntax-highlight-word

KOD PROGRAMU

Utworzono za pomocą (planetB, 2018).

PODCIAGI.PY

```
1.  from time import *
2.  from math import floor
3.
4.  try:
5.      import assistant_module as assist   # Another part of my program where some cool
     functions are placed.
6.                                          # File assistant_module.py should be located
     in the same folder with following file: (podciagi.py)
7.  except ModuleNotFoundError:
8.      print("Wystąpił błąd!\nUmieść pliki „assistant_module.py” i „podciagi.py” w tym
    samym folderze.")
9.      import sys, os
10.     os.system("pause")
11.     sys.exit()
12. except:
13.     print("Przepraszamy, coś poszło nie tak ... \nSprawdź, czy plik „assistant_modul
    e.py” nie posiada błedów.")
14.     import sys, os
15.     os.system("pause")
16.     sys.exit()
17.
18.
19.
20. class binary_sequences:
21.
22.     def __init__(self, path_in = 'input.txt', path_out = 'output.txt'):
23.         ''''' Initialization of the class and assigning default values to flags'''
24.
25.         self.path_in = path_in
26.         self.path_out = path_out
27.         self.reading_file()
28.         self.enable_repeating_lists_before_output = False
29.         self.optimization_level = 3
30.         self.show_progress_bar = False
31.
32.
33.
34.     def draw_separator(self):
35.         ''''' This function draws a "pretty" separator between tasks'''
36.
37.         heading = " Nowe zadanie z pliku: \"{}\" ".format(self.path_in)
38.         separator = '-'*5 + heading + '-'*(115 - len(heading))
39.         print('\n\n\n\n' + separator + '\n')
40.
41.
42.
43.     def solve_problem(self):
44.         ''''' The main function. Iterates the process of searching substrings for ea
    ch sequence.
45.         After completing the task it writes out small conclusion to the console. '''

46.
47.         global current_sequence
48.         self.iterator = 0                           # If more than one input was giv
    en, then this variable will help to iterate the whole process
49.         self.number_of_sequences = len(self.all_sequences)  # Number of sequences gi
    ven in the input file
```

```
50.         self.time_results = []                      # List of records of the time us
    ed by algorithm function
51.
52.         self.draw_separator()                       # Drawing a "pretty" separator b
    etween tasks
53.
54.         while self.iterator < self.number_of_sequences:
55.             current_sequence = self.all_sequences[self.iterator]
56.
57.             if self.show_progress_bar:              # Updates the progress bar if th
    e appropriate flag is enabled
58.                 assist.update_progress(self.iterator/self.number_of_sequences, self.
    path_in)
59.
60.             self.iterator += 1
61.
62.             k = current_sequence.count(1)           # n, k - the quantity of digits
    "zero" & "one" respectively
63.             n = len(current_sequence)-k
64.             p = min(n,k)                            # p - theoretical maximum of pos
    sible pairs (0,1)
65.
66.             if n==k!=0:                             # Checking for the easiest solut
    ions when k=0, n=0 or n=k
67.                 self.give_answer(1, [current_sequence])
68.                 self.time_results.append(0.0)
69.             elif n==0 or k==0:
70.                 self.give_answer(0)
71.                 self.time_results.append(0.0)
72.             elif n>=1 and k>=1:
73.                 start_time = time()                 # Recording the time of the main
     algorithm start
74.                 if self.optimization_level == 3:
75.                     substring = self.main_algorithm_optimized(p)
76.                 elif self.optimization_level == 2:
77.                     substring = self.main_algorithm_better(p)
78.                 elif self.optimization_level == 1:
79.                     substring = self.main_algorithm_brute()
80.                 else:
81.                     print("\nNieprawidłowo określony poziom optymalizacji. Wybrano u
    stawienia domyślne.")
82.                     self.optimization_level = 3
83.                     substring = self.main_algorithm_optimized(p)
84.                 end_time = time()                   # Recording the time of the main
     algorithm end
85.
86.                 self.time_results.append(round(end_time - start_time,3))
87.                 self.give_answer(len(substring), substring)
88.
89.         else:                                       # Writing of the conclusion with
     respect to conjugation of polish numerals
90.
91.             if self.input_file_exist and not self.reading_file_error and self.number
    _of_sequences>0:
92.
93.                 if self.show_progress_bar:
94.                     assist.update_progress(self.iterator/self.number_of_sequences, s
    elf.path_in)
95.                     print('')
96.
97.                 last_digit=list(str(self.number_of_sequences)).pop()
98.
99.                 teened = False
100.                  if self.number_of_sequences >=10:
101.                      if str(self.number_of_sequences)[-
    2:] in ['11', '12', '13', '14']:      # excluding -teen numbers
```

```
                            teened = True

                    if last_digit=='1' and not teened:
                        insert_text = 'wejściowy ciąg'
                    elif last_digit in ['2', '3', '4'] and not teened:
                        insert_text = 'wejściowy ciągi'
                    else:
                        insert_text = 'wejściowych ciągów'

                    time_used = round(sum(self.time_results),3)
                    avarage_time_used = round(time_used/self.number_of_sequences, 5)

                    print("Rozpoznano {0} {1} w pliku \"{2}\". ".format(self.number_of_sequences, insert_text, self.path_in)+
                        "Wszystkie odpowiedzi zostały zapisane w pliku \"{}\". ".format(self.path_out) +
                        "\nWybrany {} poziom optymalizacji.".format(self.optimization_level)+
                        "\nCzas roboty algorytmu wynosi {} s.".format(time_used) +
                        "\nŚredni czas przetwarzania jednego ciągu wynosi {} s.\n".format(avarage_time_used))

                elif not self.input_file_exist and self.reading_file_error:
                    print("\nNie znaleziono pliku według ścieżki \"{}\".\n".format(self.path_in))

                elif self.input_file_exist and self.reading_file_error:
                    print("\n\nCoś poszło nie tak podczas odczytu pliku.\n")

                elif self.number_of_sequences==0:
                    print("Nie rozpoznano wejściowych ciągów w pliku \"{}\". ".format(self.path_in))


        def reading_file(self):
            ''''' Reading of the file containing input data and creating a clean output file'''

            with open(self.path_out,'w', encoding='utf-8') as file:     # Cleaning the output file
                pass

            self.all_sequences, accepted_values = [], ['0', '1']
            try:
                with open(self.path_in,'r') as file:                    # Opens the input file and reads it symbol by symbol
                    for line in file:
                        this_sequence = []
                        for word in line.strip().split():
                            for symbol in word:
                                if symbol in accepted_values:           # Checks if the symbol is equal to '0' or '1', and if it is
                                    this_sequence.append(int(symbol))   # then adds that symbol to the sequence
                        if this_sequence != []:
                            self.all_sequences.append(this_sequence)
            except FileNotFoundError:
                self.input_file_exist = False
                self.reading_file_error = True
            except:
                self.input_file_exist = True
                self.reading_file_error = True
            else:
                self.input_file_exist = True
                self.reading_file_error = False
```

```
157.
158.
159.
160.     def main_algorithm_optimized(self, max_substr_length):
161.         ''''' Takes a sequence and returns the subsequence so that it would have e
    qual number
162.         of digits "zero" & "one". More detailed explanation can be found in my rep
    ort. '''
163.
164.         global current_sequence
165.         substring =[]                                                          #
    Defining the output array of found substrings
166.         found = False
167.         length_current_sequence = len(current_sequence)
168.
169.         for substr_len in range(max_substr_length, 0, -1):
170.             starting_point, end_point = 0, substr_len*2
171.             while end_point<=length_current_sequence:
172.                 one = current_sequence[starting_point:end_point].count(1)
173.                 if one == substr_len:
174.                     found = True
175.                     if current_sequence[starting_point:end_point] not in substring
    :
176.                         substring.append(current_sequence[starting_point:end_point
    ])
177.                     starting_point+=1
178.                     end_point+=1
179.                 else:
180.                     difference = abs(substr_len-one)
181.                     starting_point+=difference
182.                     end_point+=difference
183.             if found:
184.                 break
185.         return substring
186.
187.
188.
189.     def main_algorithm_better(self, max_substr_length):
190.         ''''' Takes a sequence and returns the subsequence so that it would have e
    qual number
191.         of digits "zero" & "one". More detailed explanation can be found in my rep
    ort. '''
192.
193.         global current_sequence
194.         substring =[]                                                          #
    Defining the output array of found substrings
195.         found = False
196.         length_current_sequence = len(current_sequence)
197.
198.         for substr_len in range(max_substr_length, 0, -1):
199.             starting_point, end_point = 0, substr_len*2
200.             while end_point<=length_current_sequence:
201.                 one = 0
202.                 for element in current_sequence[starting_point:end_point]:      #
    Counts all digits "one" in the substring
203.                     one += element                                             #
    Actually it's equal to the method "array.count(1)"
204.                 if one == substr_len:
205.                     found = True
206.                     already_exist = self.check_if_exists(current_sequence[starting
    _point:end_point], substring)
207.                     if not already_exist:
208.                         substring.append(current_sequence[starting_point:end_point
    ])
209.                     starting_point+=1
210.                     end_point+=1
```

```
211.                 else:
212.                     difference = abs(substr_len-one)
213.                     starting_point+=difference
214.                     end_point+=difference
215.             if found:
216.                 break
217.         return substring
218.
219.
220.
221.     def main_algorithm_brute(self):
222.         ''''' Takes a sequence and returns the subsequence so that it would have e
    qual number
223.         of digits "zero" & "one". More detailed explanation can be found in my rep
    ort. '''
224.
225.         global current_sequence
226.         substring =[]                                                          #
    Defining the output array of found substrings
227.         found = False
228.         length_current_sequence = len(current_sequence)
229.         length = floor(length_current_sequence/2)                              #
    Finding the starting number of searched pairs (0,1)
230.
231.         for substr_len in range(length, 0, -1):
232.             starting_point, end_point = 0, substr_len*2                        #
    Defining the size of the first subsequence (as a partition of an array)
233.             while end_point<=length_current_sequence:                           #
     While we are within sequence edges - do the following
234.                 one = 0
235.                 for element in current_sequence[starting_point:end_point]:     #
    Counts all digits "one" in the substring
236.                     one += element                                             #
    It's equal to the function "array.count(1)"
237.                 if one == substr_len:
238.                     found = True
239.                     already_exist = self.check_if_exists(current_sequence[starting
    _point:end_point], substring)
240.                     if not already_exist:                                      #
    Adding the found sequence to the array, if it isn't there already
241.                         substring.append(current_sequence[starting_point:end_point
    ])
242.                 starting_point+=1                                              #
    Moving forward along the sequence
243.                 end_point+=1
244.             if found:                                                          #
    If the subsequence is found - stop searching
245.                 break
246.         return substring                                                       #
    And return the result
247.
248.
249.
250.     def check_if_exists(self, prey, sequence):
251.         ''''' Checks if the given prey exists in the array "sequence". In other wo
    rds, it is
252.         the equivalent to the native Python method "<object> in <object>".'''
253.
254.         for element in sequence:
255.             if element == prey:
256.                 return True
257.         else:
258.             return False
259.
260.
261.
```

```
262.     def sequence_to_string(self, sequence):
263.         ''''' Creates a beautiful string from the sequence with comas after each e
    lement except for the last one'''
264.
265.         string = ''
266.         last_without_period = str(sequence.pop())
267.         for element in sequence:
268.             string += "{}, ".format(element)
269.         return string + last_without_period
270.
271.
272.
273.     def give_answer(self, n_of_answ, answer = ''):
274.         '''''Creates a file in the given path and adds an answer to it with respec
    t to the amount of answers'''
275.
276.         with open(self.path_out,'a', encoding='utf-8') as file:
277.             if self.iterator != 1:
278.                 file.write('\n\n')
279.
280.             file.write("Rozwiązanie dla ciągu №" + str(self.iterator))
281.             if self.enable_repeating_lists_before_output:
282.                 file.write(' ({})'.format(self.sequence_to_string(current_sequence
    .copy())))
283.             file.write('\n')
284.
285.             if n_of_answ==0:
286.                 file.write("Dla takich danych wejściowych: {}\npodciąg zawierający
     równą".format(self.sequence_to_string(current_sequence)) +
287.                            " liczbę zer i jedynek nie istnieje.")
288.
289.             elif n_of_answ==1:
290.                 file.write('Najdłuższym podciągiem jest: ' + self.sequence_to_stri
    ng(answer[0]))
291.
292.             elif n_of_answ==2:
293.                 file.write('Najdłuższymi podciągami są ' + self.sequence_to_string
    (answer[0]) + ' oraz ' +
294.                            self.sequence_to_string(answer[1]))
295.
296.             elif n_of_answ>=3:
297.                 file.write('Najdłuższymi podciągami są:\n')
298.                 file.write(self.sequence_to_string(['('+self.sequence_to_string(an
    s)+')' for ans in answer]))
299.
300.
301.
302.  def porownanie_algorytmow():
303.      ''''' Here I would like to show instructions I used in
304.      the paragraph "Porównanie algorytmów" of my report'''
305.
306.      # I HIGHLY RECOMMEND performing this function ONLY
307.      # after reading the description in the article,
308.      # because in general it can take a long time to perform
309.      """
310.
311.      # Generating 20 random sequences of digits zero and one and writing them down
    to the file "report_rand_input.txt"
312.      # The following function is commented, because we already created this file be
    fore,
313.      # so we don't need to create it one more time. Moreover, since the input data
    is the same,
314.      # you can test this yourself and get similar results to one, shown in the repo
    rt.
315.      # assist.random_sequence(path = ".\\tests\\report_rand_input.txt", lines = 20,
     start_length = 1000, increment = 0, multiplier = 1)
```

```
316.
317.      # Creating an object of class binary_sequences
318.      test_1 = binary_sequences('.\\tests\\report_rand_input.txt', '.\\tests\\report
    _rand_output.txt')
319.
320.      # Needed values have to be assigned to flags
321.      test_1.optimization_level = 1
322.      test_1.show_progress_bar = True
323.
324.      # Now we can start our test
325.      test_1.solve_problem()
326.
327.      # Since we are going to compare our algorithms, we need the time results infor
    mation
328.      give_time(test_1)
329.
330.      # Second algorithm test
331.      test_1.optimization_level = 2
332.      test_1.solve_problem()
333.      give_time(test_1)
334.
335.      # Third algorithm test
336.      test(optimization_level = 3, path_in = '.\\tests\\report_rand_input.txt', path
    _out = '.\\tests\\report_rand_output.txt',
337.          return_time = True, generate_new_data = False, show_progress_bar = True)
338.
339.      # Creating a comparison of three algorithms using graph
340.      # Maybe we would like to comment all upper rows, since this function creates t
    ests itself.
341.      algorithm_comparison(".\\tests\\report_rand_input.txt", ".\\tests\\report_rand
    _output.txt",generate_new_data = False,
342.          show_progress_bar = True, width=1200, height=800, show_stats = False)
343.
344.      assist.worst_sequence(path = ".\\tests\\report_worst_input.txt", lines = 20, s
    tart_repeats = 333, increment = 0, multiplier = 1)
345.
346.      algorithm_comparison(".\\tests\\report_worst_input.txt", ".\\tests\\report_wor
    st_output.txt",generate_new_data = False,
347.          show_progress_bar = True, width=1200, height=800, show_stats = False)
348.
349.      assist.worst_sequence(path = ".\\tests\\report_worst_increment_input.txt", lin
    es = 20, start_repeats = 100, increment = 15, multiplier = 1)
350.
351.      algorithm_comparison(".\\tests\\report_worst_increment_input.txt", ".\\tests\\
    report_worst_increment_output.txt",generate_new_data = False,
352.          show_progress_bar = True, width=1200, height=800, show_stats = False)
353.      """
354.      pass
355.
356.
357.
358.  def test(optimization_level = 3, path_in = '', path_out = '', return_time = True,
    worst_scenario = False,
359.      lines = 10, start_repeats = 30, start_length = 50, increment = 0, multiplier =
     1, send_to_class = False,
360.      receiver_object = '', generate_new_data = True, show_progress_bar = False):
361.      ''''' Creates the test with random or the worst possible input data.
362.      Be careful with the input and output files you are providing: they will be
363.      replaced with the new automatically generated files, if generate_new_data flag
     is set to True'''
364.
365.      from random import randrange as rand
              # Generation of the file names
366.      rand_koef = rand(1000)
367.      if path_in == '':
```

```
368.          path_in = ".\\tests\\input_opt_{0}_rand{1}.txt".format(optimization_level,
     rand_koef)
369.      if path_out == '':
370.          path_out = ".\\tests\\output_opt_{0}_rand{1}.txt".format(optimization_leve
    l, rand_koef)
371.
372.      try:
373.          if worst_scenario and generate_new_data:
              # Generation of an input file
374.              assist.worst_sequence(path_in, lines, start_repeats, increment, multip
    lier)
375.          elif generate_new_data:
376.              assist.random_sequence(path_in, lines, start_length, increment, multip
    lier)
377.      except FileNotFoundError as error:
378.          print("Wystąpił błąd: ", error)
379.      except:
380.          print("Przepraszamy, coś poszło nie tak ...")
381.      else:
382.          test_object = binary_sequences(path_in, path_out)
              # Solving the problem if a file is created
383.          test_object.optimization_level = optimization_level
384.          test_object.show_progress_bar = show_progress_bar
385.          test_object.solve_problem()
386.          if return_time:
387.              give_time(test_object)
388.              if send_to_class:
389.                  receiver_object.save_new_data(test_object.time_results, optimizati
    on_level)
390.          del test_object
391.
392.
393.
394.  def algorithm_comparison(path_in='', path_out='', worst_scenario = False, generate
    _new_data = True, show_progress_bar = False,
395.      lines = 10, start_repeats = 10, start_length = 100, increment = 0, multiplier
    = 1, width=1200, height=800, show_stats = False):
396.      ''''' Creates the algorithm comparison with random or the worst possible input
     data.
397.      Be careful with the input and output files you are providing: they will be
398.      replaced with the new automatically generated files, if generate_new_data flag
     is set to True'''
399.
400.      if worst_scenario and generate_new_data:
401.          assist.worst_sequence(path_in, lines = lines, start_repeats = start_repeat
    s, increment = increment, multiplier = multiplier)
402.      elif generate_new_data:
403.          assist.random_sequence(path_in,  lines = lines, start_length = start_lengt
    h, increment = increment, multiplier = multiplier)
404.
              # Defining the object of the class  assist.Graph
405.      graph_object = assist.Graph(worst_scenario=worst_scenario, increment=increment
    ,
406.          multiplier=multiplier, width=width, height=height, show_stats = show_stats
    )
407.
              # Creating some tests with different optimization level
408.      test(path_in=path_in, path_out=path_out, show_progress_bar = show_progress_bar
    ,
409.          optimization_level = 1, send_to_class=True, receiver_object=graph_object,
    return_time = True, generate_new_data = False)
410.      test(path_in=path_in, path_out=path_out, show_progress_bar = show_progress_bar
    ,
411.          optimization_level = 2, send_to_class=True, receiver_object=graph_object,
    return_time = True, generate_new_data = False)
```

```
412.      test(path_in=path_in, path_out=path_out, show_progress_bar = show_progress_bar
    ,
413.          optimization_level = 3, send_to_class=True, receiver_object=graph_object,
    return_time = True, generate_new_data = False)
414.      graph_object.paint_graph()
              # Drawing the graph
415.      del graph_object
416.
417.
418.
419.  def give_time(object_):
420.      ''''' Returns time needed for each input sequence to be processed '''
421.      print("Czas przetwarzania każdego podciągu: ", object_.time_results)
422.
423.
424.  def main():
425.      '''''
426.      Main function of the whole program.
427.
428.      Almost any possible format of input is accepted. For example:
429.      "1, 0, 1, 0"; "1,0,1,0"; "1 0 1 0"; " 1*0*1*0 "; "1, qwe0  e1 *0"; "1010".
430.      Every single of those inputs declares the following sequence: [1, 0, 1, 0]
431.
432.      For more than one input, please separate them using a new line. For example th
    is input:
433.      1111010111
434.      1101011010
435.      1011101010
436.      1111010101
437.      will be understood as 4 sequences.
438.      [1, 1, 1, 1, 0, 1, 0, 1, 1, 1], [1, 1, 0, 1, 0, 1, 1, 0, 1, 0],
439.      [1, 0, 1, 1, 1, 0, 1, 0, 1, 0], [1, 1, 1, 1, 0, 1, 0, 1, 0, 1]
440.      '''
441.
442.      example_object = binary_sequences('input.txt', 'output.txt')    # Input the ob
    ject of the class binary_sequences with two
443.                                                                      # parameters:
    the path to the input & output files
444.
445.      # Default values of the flags:
446.      example_object.enable_repeating_lists_before_output = False     # Enable to ad
    d the repetition of the input data in the output file
447.      example_object.optimization_level = 3                           # Level of alg
    orithm optimisation. Accepted values: 1, 2, 3 (bigger is better).
448.      example_object.show_progress_bar = False                        # Enable to se
    e the progress bar. WORKS WELL ONLY IN CONSOLE. In python Shelf it looks ugly.
449.
450.
451.      example_object.enable_repeating_lists_before_output = True      # When you are
     going to read the results it looks prettier, but while working
452.                                                                      # with large a
    mounts of data it is likely to mess up the look of the output file.
453.                                                                      # Since I woul
    d appreciate you to read the file 'output.txt', I am going to enable this flag
454.
455.      # Launching the main function of the class 'binary_sequences'
456.      example_object.solve_problem()
457.
458.
459.      # If you need to know how much time was needed to process each
460.      # sequence you can use one of the next methods:
461.      print(example_object.time_results)                              # Only prints
    the array of time records
462.      give_time(example_object)                                       # Does the sam
    e and also adds some pretty text
463.
```

```
464.      # If you finished working with some object
465.      # it is better to delete it than not, because
466.      # in that case it won't use any RAM
467.      del example_object
468.
469.
470.      test(                                                    # You can also
     create tests using test() function, where:
471.          optimization_level = 1,                              # - level of a
    lgorithm optimisation, accepted values: 1, 2, 3;
472.          path_in = '.\\tests\\input_worst_scenario.txt',       # - path, wher
    e new input file will be generated or existing file opened;
473.          path_out = '.\\tests\\output_worst_scenario.txt',     # - path, wher
    e new output file will be created;
474.          return_time = True,                                  # - flag, whic
    h asks if you would like to see used time results in console;
475.          generate_new_data = True,                            # - flag, whic
    h asks if you would like to create a new input file or use an existing one;
476.          show_progress_bar = True,                            # - flag, whic
    h asks if you would like to see the progress bar (WORKS WELL ONLY IN CONSOLE);
477.
478.          # Next options make changes only if generate_new_data flag is set to True:

479.          worst_scenario = True,                               # - flag, whic
    h asks if the input file has to be filled with
480.                                                               #   the worst
    possible (True) strings or random (False) strings;
481.          lines = 20,                                          # - number of
    strings (lines) in a generated file;
482.          start_repeats = 10,                                  # - number of
    repeats of the sequence '101' in the first line, if the worst scenario is chosen;
483.          start_length = 500,                                  # - length of
    the first string (line), if random scenario is chosen;
484.          increment = 5,                                       # - increment
    of (repeats / length) of the (sequence '101' / line) after each line;
485.          multiplier = 1                                       # - multiplica
    tion of (repeats / length) of the (sequence '101' / line) after each line.
486.          )
487.
488.
489.      algorithm_comparison(                                    # You can also
     create algorithm comparison using algorithm_comparison() function, where:
490.          path_in = '.\\tests\\inp_comparison.txt',             # - path, wher
    e new input file will be generated;
491.          path_out = '.\\tests\\out_comparison.txt',            # - path, wher
    e new output file will be created;
492.          generate_new_data = True,                            # - flag, whic
    h asks if you would like to create a new input file or use an existing one;
493.          show_progress_bar = True,                            # - flag, whic
    h asks if you would like to see the progress bar (WORKS WELL ONLY IN CONSOLE);
494.          height = 800,                                        # - height of
    the created graph (minimum 200, recommended 800);
495.          width = 1200,                                        # - width of t
    he created graph (minimum 650, recommended 1200);
496.          show_stats = True,                                   # - shows the
    information about the increment and multiplier on the plot;
497.
498.          # Next options make changes only if generate_new_data flag is set to True:

499.          worst_scenario = True,                               # - flag, whic
    h asks if the input file has to be filled with
500.                                                               #   the worst
    possible (True) strings or random (False) strings;
501.          lines = 15,                                          # - number of
    strings (lines) in a generated file;
```

```
502.          start_repeats = 50,                                          # - number of
    repeats of the sequence '101' in the first line, if the worst scenario is chosen;
503.          start_length = 500,                                          # - length of
    the first string (line), if random scenario is chosen;
504.          increment = 5,                                               # - increment
    of (repeats / length) of the (sequence '101' / line) after each line;
505.          multiplier = 1                                               # - multiplica
    tion of (repeats / length) of the (sequence '101' / line) after each line.
506.          )
507.
508.
509.      # I HIGHLY RECOMMEND performing this function ONLY
510.      # after reading the description in the article,
511.      # because in general it can take a long time to perform
512.      porownanie_algorytmow()
513.
514.      # If you need the console not to close immediately after
515.      # finishing your task, then please uncomment the following two rows.
516.      # import os
517.      # os.system("pause")
518.
519.
520.
521.  if __name__ == "__main__":
522.      main()
```

ASSISTANT_MODULE.PY

```
1.  from random import choice
2.  import time, sys
3.  import tkinter as tk
4.  import tkinter.font
5.  from math import floor
6.  sequence = ["0", "1"]                                                    # Po
    ssible values in the test sequence
7.  worst_situaion = '101'                                                   # In
     my opinion, it is the worst possible input sequence, if repeated
8.
9.  class Graph:
10.     ''''' Creates a graph of the times used to process all
11.     substrings using "tkinter" and other built-in python methods'''
12.
13.     def __init__(self, show_stats = False, worst_scenario=False, increment=0, multip
    lier=1, width=1200, height=400, text_density=25):
14.         self.time_records = []
15.         self.optimalization_level = []
16.
17.         master = tk.Tk()                                                 # Cr
    eating a window for the graph
18.         master.title("Wykres porównania algorytmów")
19.
20.         if height < 200:
21.             height = 200
22.         if width < 650:
23.             width = 650
24.
25.         self.width, self.height = width, height                          # De
    fining some basic distances
26.         self.legend_x, self.legend_y, self.title, self.post_graph_area = 100, 120, 5
    0, 20
27.         self.density = text_density
28.         self.graph_width = self.width - self.legend_x - self.post_graph_area
```

```
29.         self.graph_height = self.height - self.legend_y - self.title
30.
31.
32.         self.canv = tk.Canvas(master, width=self.width, height=self.height)     # Cr
    eating a canvas to write on
33.         self.canv.pack()
34.
35.         self.times12 = tkinter.font.Font(family='Times',                        # De
    fining some fonts and colours, which will be used later
36.             size=12, weight='bold')
37.         self.times30 = tkinter.font.Font(family='Times',
38.             size=30, weight='bold')
39.         self.times20 = tkinter.font.Font(family='Times',
40.             size=20, weight='bold')
41.         self.colors = ['darkgreen', 'crimson', 'mediumblue', 'indigo', 'maroon', 'sa
    ddlebrown', 'darkgoldenrod', 'black']
42.
43.                                                                                 # Cr
    eating a plot
44.
45.         self.canv.create_line( self.legend_x+1, self.title+self.graph_height+1, self
    .width,
46.             self.title+self.graph_height+1, fill = "grey", width = 3, arrow = tk.LAS
    T )
47.         self.canv.create_line( self.legend_x+1, self.title+self.graph_height, self.l
    egend_x+1,
48.             self.title*0.5, fill = "grey", width = 3, arrow = tk.LAST)
49.         self.canv.create_text(self.width/2, self.title/2,  text = "Porównania algory
    tmów", font = self.times30, anchor = tk.CENTER, fill="black")
50.         self.canv.create_text(self.legend_x/2, self.title/2,  text = "Czas, s", font
     = self.times20, anchor = tk.CENTER, fill="black")
51.         self.canv.create_text(self.width, self.title+self.graph_height+self.legend_y
    /3,
52.             text = "Numer ciągu", font = self.times20, anchor = tk.E, fill="black")
53.
54.         if worst_scenario and show_stats:
55.             self.canv.create_text(self.width/2, self.title/2 + 18,  text = "najgorsz
    e dane wejściowe", font = self.times20, anchor = tk.N, fill="black")
56.             self.canv.create_text(self.width/2, self.title+self.graph_height+self.le
    gend_y/4,
57.             text = "length increment: {}  multiplier: {}".format(increment*3, multip
    lier), font = self.times12, anchor = tk.N, fill="black")
58.         elif not worst_scenario and show_stats:
59.             self.canv.create_text(self.width/2, self.title/2 + 18,  text = "przypadk
    owe dane wejściowe", font = self.times20, anchor = tk.N, fill="black")
60.             self.canv.create_text(self.width/2, self.title+self.graph_height+self.le
    gend_y/4,
61.             text = "length increment: {}  multiplier: {}".format(increment, multipli
    er), font = self.times12, anchor = tk.N, fill="black")
62.
63.
64.
65.     def save_new_data(self, times_info, optimalization_level):
66.         ''''' This function takes data and writes it down'''
67.
68.         if sum(times_info)<=0.001:
69.             print("\nPODANO DANE DĄŻĄCE DO ZERA!!!\nWYKRES TEJ FUNKCJI NIE ZOSTANIE
    NARYSOWANY!")
70.         else:
71.             self.time_records.append(times_info)
72.             self.optimalization_level.append(optimalization_level)
73.
74.
75.
76.     def paint_graph(self):
```

```
77.         ''''' This function creates a graph after all data is collected.'''
78.
79.         if self.time_records == []:                                         # Ch
    ecking if data exist at all
80.             print("Za mało danych, aby narysować wykres!")
81.             return -1
82.
83.         all_len, all_max, all_min, all_sum = [], [], [], []                 # De
    fining the common scale for all data
84.         for data_set in self.time_records:
85.             all_len.append(len(data_set))
86.             all_max.append(max(data_set))
87.             all_min.append(min(data_set))
88.         self.distance = self.graph_width/max(all_len)
89.         self.multiplicator = self.graph_height/max(all_max)
90.
91.
92.         quantity_labels = floor(self.graph_height/self.density)             # De
    fining values on Y axis
93.         Min, Max =min(all_min), max(all_max)
94.         step = (Max - Min) / quantity_labels
95.         for i in range(quantity_labels):
96.             self.canv.create_text(self.legend_x-
    8, i*self.density+self.title, text = str(round(Max-i*step, 3)),
97.                 font = self.times12, anchor = tk.E, fill="red")
98.
99.         quantity_labels = floor(self.graph_width/self.density)              # De
    fining values on X axis
100.          step =   floor(max(all_len) / quantity_labels)
101.          if step == 0:
102.              step = 1
103.          for i in range(0, max(all_len)+1, step):
104.              self.canv.create_text(i*self.distance+self.legend_x, self.graph_height
    +self.title+8,  text = str(i),
105.                  font = self.times12, anchor = tk.N, fill="green")
106.
107.
108.          numb_of_records = len(self.time_records)                            #
    Drawing all graphs and writing down the legend under the plot
109.          for i in range(numb_of_records):
110.              self.graph_painter(self.time_records[i], i+1)
111.              self.canv.create_text(self.graph_width/numb_of_records*(i)+self.post_g
    raph_area, self.height - 10,
112.                  text = "Test {0} - wykres funkcji zależności czasu od\nciągu algor
    ytmu z poziomem optymalizacji {1}".format(i+1, self.optimalization_level[i]),
113.                  font = self.times12, anchor = tk.SW, fill=self.colors[(i+1)%len(se
    lf.colors)])
114.          tk.mainloop()
115.
116.
117.      def graph_painter(self, times_info, number):
118.          ''''' This function draws a graph of values given in the array times_info'
    ''
119.
120.          color = self.colors[number%len(self.colors)]
121.          x1, y1= self.legend_x, self.graph_height+self.title
122.          previous = 0
123.          for record in times_info:                                          #
    Drawing a line from the previous record to the current record.
124.              shift = record - previous                                      #
    Null record defined as (0,0)
125.              previous = record
126.              shift =self.multiplicator*shift
127.              x2 = x1+self.distance
128.              self.canv.create_line( x1, y1, x2, y1-
    shift, fill = color, width = 1 )
```

```
129.             x1 = x2
130.             y1 -= shift
131.         self.canv.create_text(x2, y1,  text = "Test "+str(number),              #
    Creating a caption of the graph
132.                 font = self.times12, anchor = tk.SE, fill=color)
133.
134.
135.
136.  def random_sequence(path = "input_rand.txt", lines = 10, start_length = 50, increm
    ent = 0, multiplier = 1):
137.      ''''' This function creates a file filled with random sequence of digits "zero
    " and "one".
138.      More detailed explanation is given below the function'''
139.
140.      with open(path,'w', encoding='utf-8') as file:
141.          for i in range(lines):
142.              line = ''
143.              for j in range(start_length):
144.                  line+=choice(sequence)
145.              file.write(line+'\n')
146.              start_length+= increment
147.              start_length*= multiplier
148.  """ Example:
149.  path_in = '',        - path, where a new input file will be generated;
150.  lines = 20,          - number of strings (lines) in the generated file;
151.  start_length = 50,   - length of the first string (line);
152.  increment = 5,       - increment of the length of the line after each line;
153.  multiplier = 1       - multiplication of the length of the line after each line.
154.  """
155.
156.
157.  def worst_sequence(path = "input_worst.txt", lines = 10, start_repeats = 30, incre
    ment = 0, multiplier = 1):
158.      ''''' This function creates file filled with worst sequence of digit "zero" an
    d "one".
159.      More detailed explanation is given below the function'''
160.
161.      with open(path,'w', encoding='utf-8') as file:
162.          for i in range(lines):
163.              line = worst_situaion * start_repeats
164.              file.write(line+'\n')
165.              start_repeats+= increment
166.              start_repeats*= multiplier
167.  """ Example:
168.  path_in = '',        - path, where a new input file will be generated;
169.  lines = 20,          - number of strings (lines) in the generated file;
170.  start_repeats = 10,  - number of repeats of the sequence '101' in the first line;
171.  increment = 5,       - increment of repeats of the sequence '101' after each line;
172.  multiplier = 1       - multiplication of repeats of the sequence '101' after each l
    ine.
173.  """
174.
175.
176.  # The next function is not written by Vitalii Morskyi (just modified)
177.  # Source: https://stackoverflow.com/questions/3160699/python-progress-bar
178.  def update_progress(progress, path_in):
179.      ''''' Displays or updates a console progress bar. WORKS ONLY WITH CONSOLE
180.      Accepts a float between 0 and 1. Any int will be converted to a float.
181.      A value under 0 represents a 'halt'.
182.      A value at 1 or bigger represents 100%.'''
183.
184.      barLength = 10 # Modify this to change the length of the progress bar
185.      status = ""
186.      if isinstance(progress, int):
187.          progress = float(progress)
188.      if not isinstance(progress, float):
```

```
189.          progress = 0
190.          status = "error: progress var must be float\r\n"
191.      if progress < 0:
192.          progress = 0
193.          status = "Halt...\r\n"
194.      if progress >= 1:
195.          progress = 1
196.          status = "Gotowy...\r\n"
197.      block = int(round(barLength*progress))
198.      text = "\rPrzetwarzanie pliku \"{3}\" : [{0}] {1}% {2}".format( "#"*block + "-
    "*(barLength-block), round(progress*100,1), status, path_in)
199.      sys.stdout.write(text)
200.      sys.stdout.flush()
```