**P O L I T E C H N I K A R Z E S Z O W S K A**

**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**

Spis treści

[Opis problemu 3](#_Toc57452802)

[Analiza algorytmu 3](#_Toc57452803)

[Wczytywanie ciągów z plików 3](#_Toc57452804)

[Wyeliminowanie najprostszych przypadków 5](#_Toc57452805)

[Główny algorytm wyszukiwania podciągów 6](#_Toc57452806)

[Podstawowy algorytm wyszukiwania podciągów 6](#_Toc57452807)

[Optymalizowanie algorytmu wyszukiwania podciągów 9](#_Toc57452808)

[Porównanie algorytmów 14](#_Toc57452809)

[Dokumentacja z doświadczeń 17](#_Toc57452810)

[Wczytywania danych 17](#_Toc57452811)

[Wypisywania wyników 17](#_Toc57452812)

[Wnioski 18](#_Toc57452813)

[Legenda pseudokodów i schematów blokowych 18](#_Toc57452814)

[Spisy odwołań 18](#_Toc57452815)

[Bibliografia 19](#_Toc57452816)

[Kod programu 20](#_Toc57452817)

[podciagi.py 20](#_Toc57452818)

[assistant\_module.py 30](#_Toc57452819)

# 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

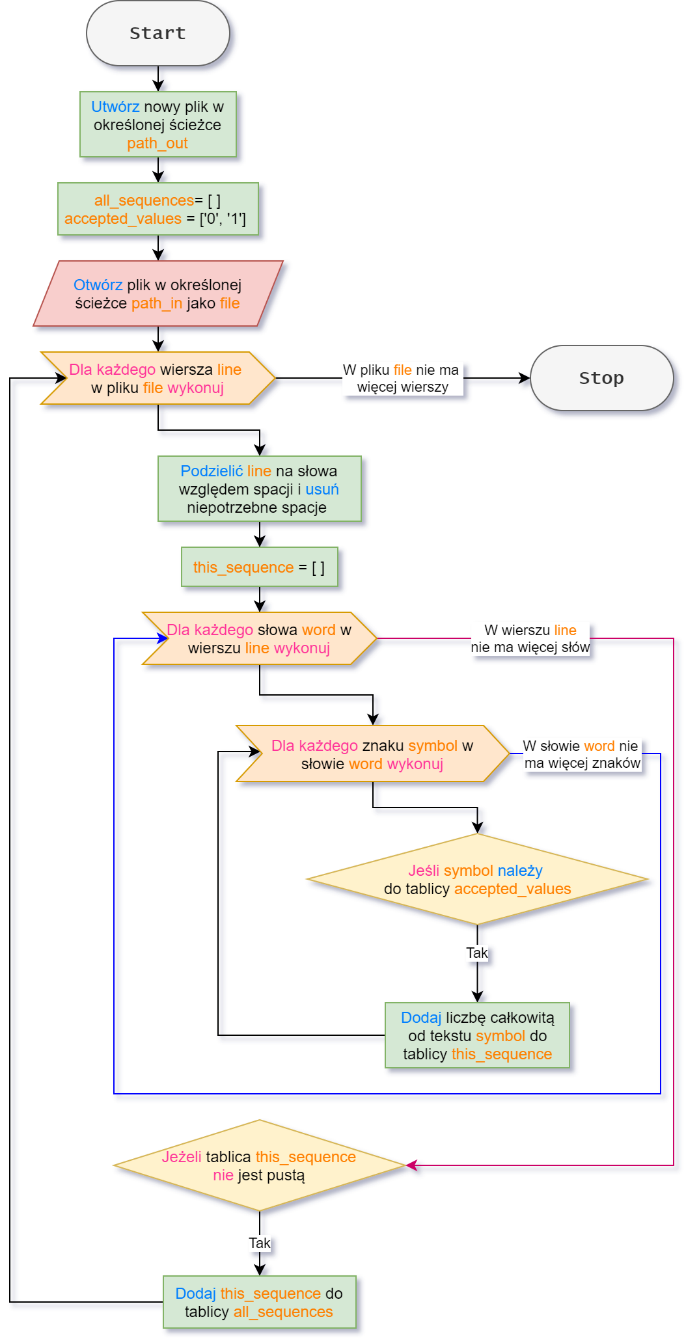
W danym rozdziale chciałbym szczegółowo wyjaśnić każdy ułamek mojego algorytmu.

## 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 read\_file():



Schemat blokowy 1 read\_file()  
Źródło: opracowanie własne.

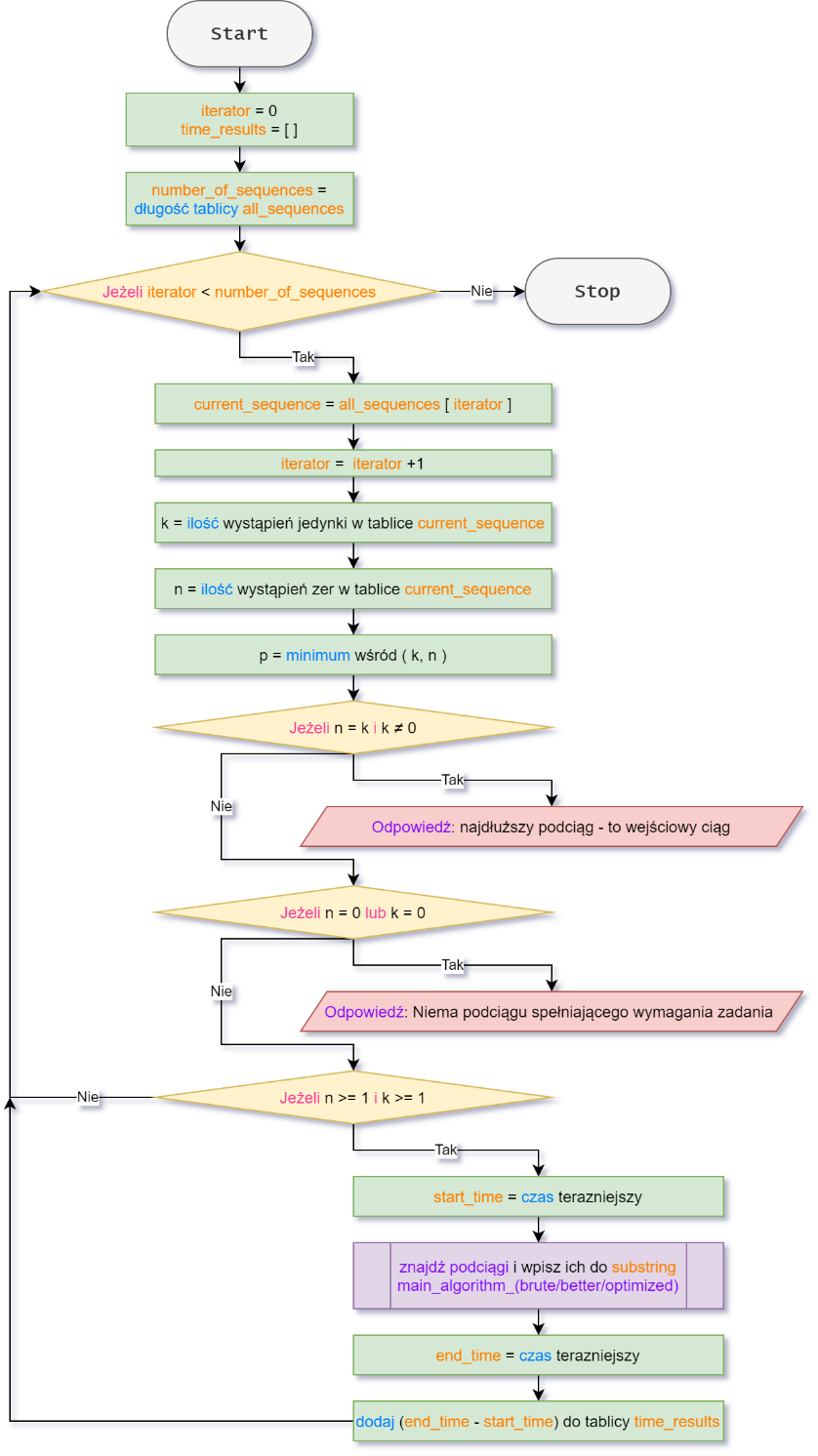
Pseudokod funkcji read\_file():

1. Utwórz nowy plik w określonej ścieżce path\_out // Tworzenie wyjściowego pliku
2. all\_sequences= [ ]
3. accepted\_values = ['0', '1']
4. Otwórz plik w określonej ścieżce path\_in jako file
5. Dla *każdego wiersza line w pliku file* wykonuj:
6. | Podzielić line na słowa względem spacji
7. | Usuń niepotrzebne spacje z wierszu line
8. | this\_sequence = [ ]
9. | Dla *każdego słowa word w wierszu line* wykonuj:
10. | | Dla *każdego symbolu symbol w słowie word* wykonuj:
11. | | Jeśli *symbol należy do accepted\_values*, to
12. | | | Dodaj int(symbol) do tablicy this\_sequence
13. | Jeśli *this\_sequence ≠ [ ],* to
14. | | Dodaj this\_sequence do tablicy all\_sequences
15. 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():

1. iterator = 0
2. time\_results = [ ]
3. number\_of\_sequences = długość (all\_sequences)
4. Dopóki *iterator < number\_of\_sequences* wykonuj:
5. | current\_sequence = all\_sequences [ iterator ]
6. | iterator += 1
7. | k = zlicz ilość wystąpień jedynki w tablice current\_sequence
8. | n = zlicz ilość wystąpień zer w tablice current\_sequence
9. | p = minimum wśród (k, n)
10. | Jeżeli *n = k i k ≠ 0*, to:
11. | | Odpowiedź: najdłuższy podciąg - to wejściowy ciąg // Funkcja odpowiedź w kodzie źródłowym posiada nazwę give\_answer()
12. | W przeciwnym razie, jeśli *n = 0 lub k = 0*, to:
13. | | Odpowiedź: Niema podciągu spełniającego wymagania zadania
14. | W przeciwnym razie, jeśli *n >= 1 i k >= 1*, to:
15. | | start\_time = czas teraźniejszy
16. | | substring = main\_algorithm\_(brute/better/optimized) // Wybieramy algorytm zgodnie z decyzją użytkownika
17. | | end\_time = czas teraźniejszy
18. | | dodaj (end\_time - start\_time) do tablicy time\_results
19. Zakończ

## Główny algorytm wyszukiwania podciągów

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

### Podstawowy algorytm wyszukiwania podciągów

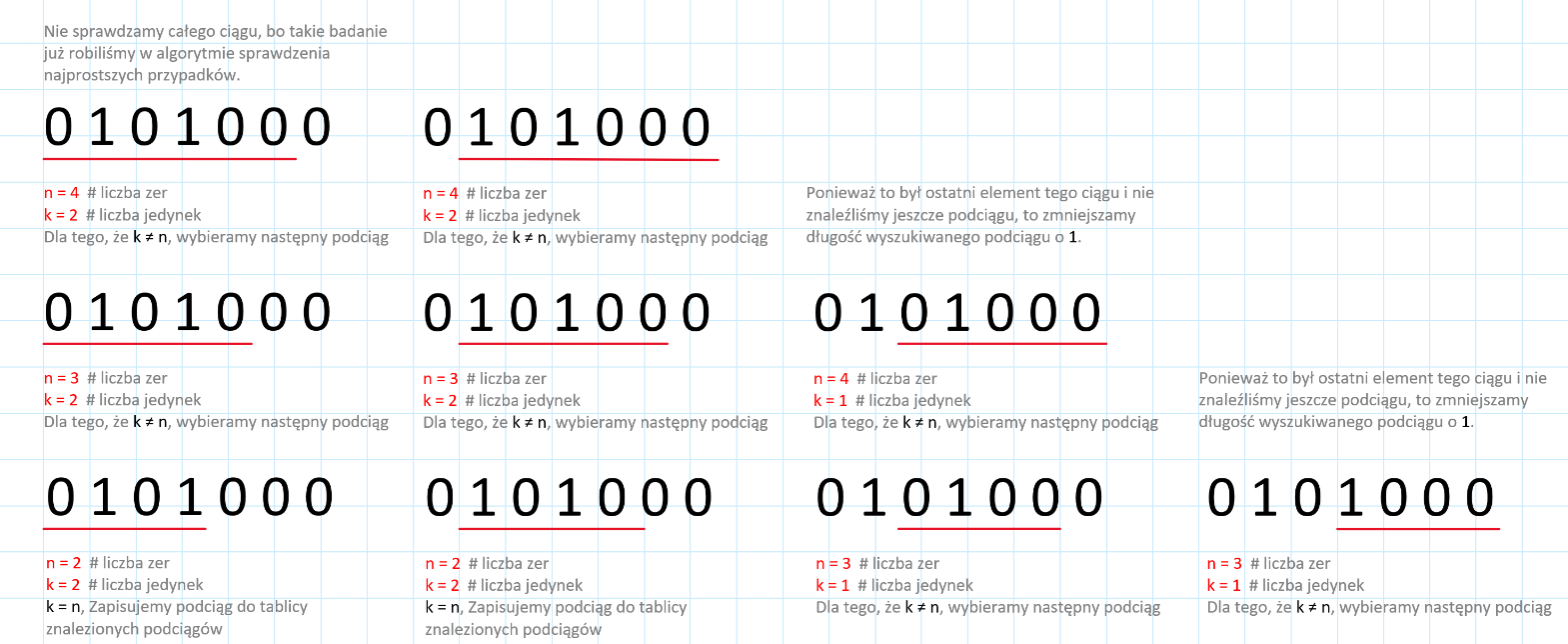
Nasz algorytm ma znaleźć różne najdłuższe podciągi, w których liczba zer i jedynek jest równa. Dlatego możemy sprawdzać wszystkie możliwe podciągi licząc ilości zer i jedynek w każdym. Ponieważ potrzebujemy odszukać najdłuższy podciąg, to będziemy zaczynać od najdłuższego i iść do najkrótszego możliwego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

Na przykład bierzemy ciąg:

Oczywistym jest fakt, że ma dwa najdłuższe podciągi:

oraz

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.

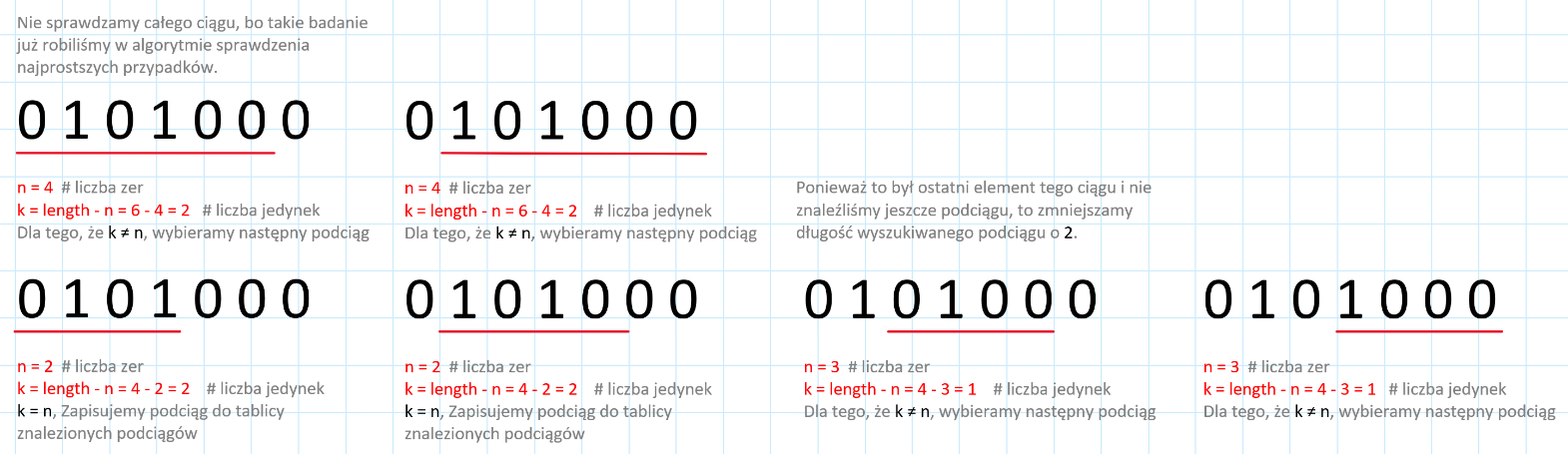


Schemat 1 Kroki działania aktualnego algorytmy dla ciągu 0101000.  
Źródło: opracowanie własne.

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

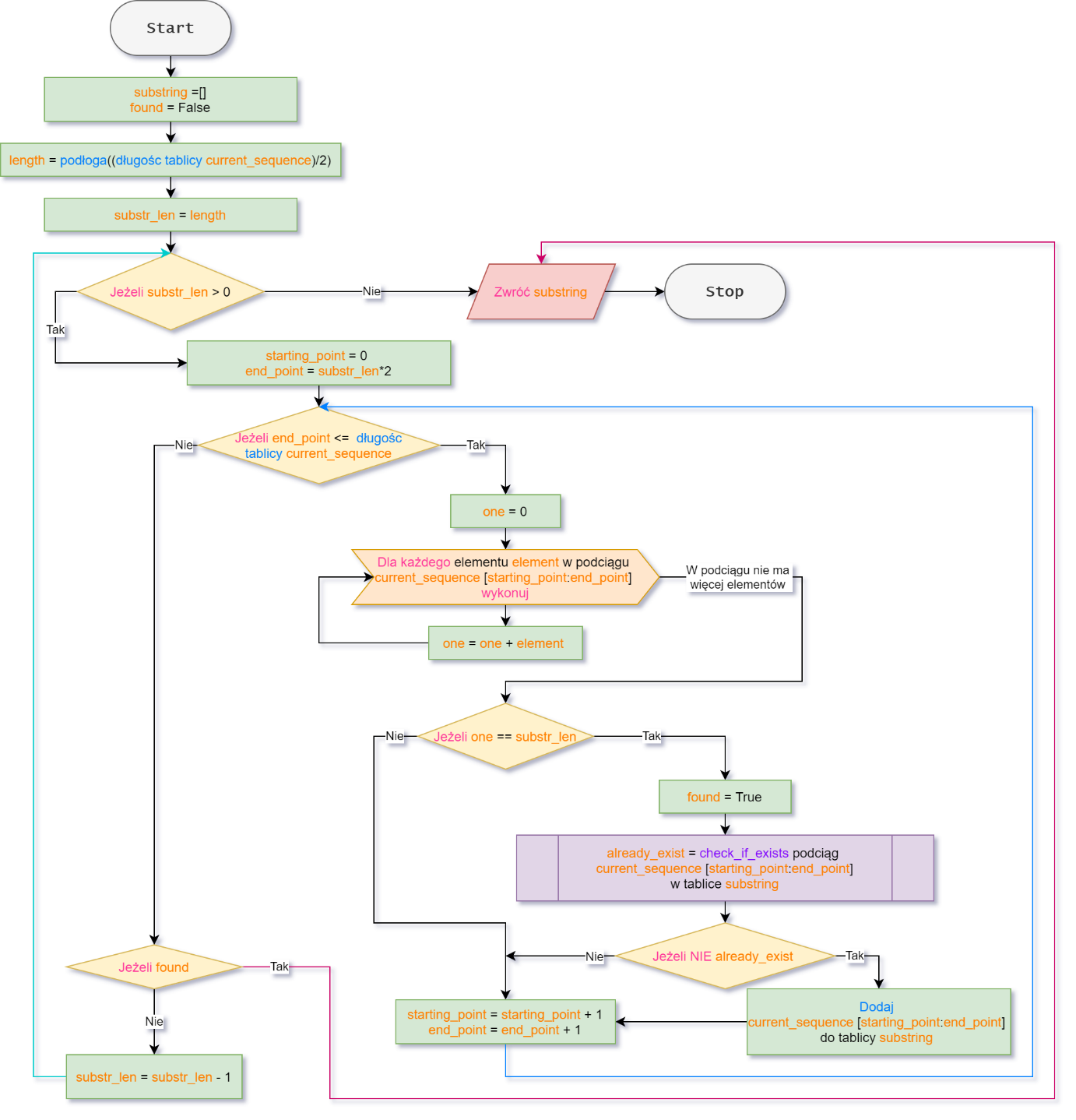
To znaczy, że możemy sprawdzać tylko parzyste długości podciągów, skracając w ten sposób czas potrzebny do wykonania programu. Skoro będziemy mieć tylko parzyste podciągi, to możemy zliczać tylko ilość któregoś jednego z elementów (0, 1) i porównywać ją z połową długości podciągu, co także zmniejsza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

1. substring =[]
2. found = False
3. length = podłoga((długośc tablicy current\_sequence)/2)
4. substr\_len = length
5. Dopóki substr\_len > 0 wykonuj K06:K24
6. | starting\_point = 0
7. | end\_point = substr\_len\*2
8. | Dopóki end\_point <= długośc tablicy current\_sequence wykonuj K09:K21
9. | | one = 0
10. | | Dla każdego elementu element w podciągu current\_sequence [starting\_point:end\_point] wykonuj K11:
11. | | | one = one + element
12. | | Jeżeli one == substr\_len wykonuj K13:K19
13. | | | found = True
14. | | | already\_exist = False
15. | | | 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.
16. | | | | Jeżeli element == current\_sequence [starting\_point:end\_point] wykonuj K17
17. | | | | | already\_exist = True
18. | | | Jeżeli NIE already\_exist wykonuj K19
19. | | | | Dodaj current\_sequence [starting\_point:end\_point] do tablicy substring
20. | | starting\_point = starting\_point + 1
21. | | end\_point = end\_point + 1
22. | Jeżeli found wykonuj K23
23. | | break
24. | substr\_len = substr\_len – 1
25. Zwróć substring
26. 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 maksymalna teoretyczna długość podciągu będzie zależała od ilości jedynek dlatego, że ich jest mniej. Czyli:

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

W tym ciągu liczba zer , a liczba jedynek . Nasz algorytm już wie, że maksymalna teoretyczna długość podciągu wynosi:

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 10111111111111111010.  
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:

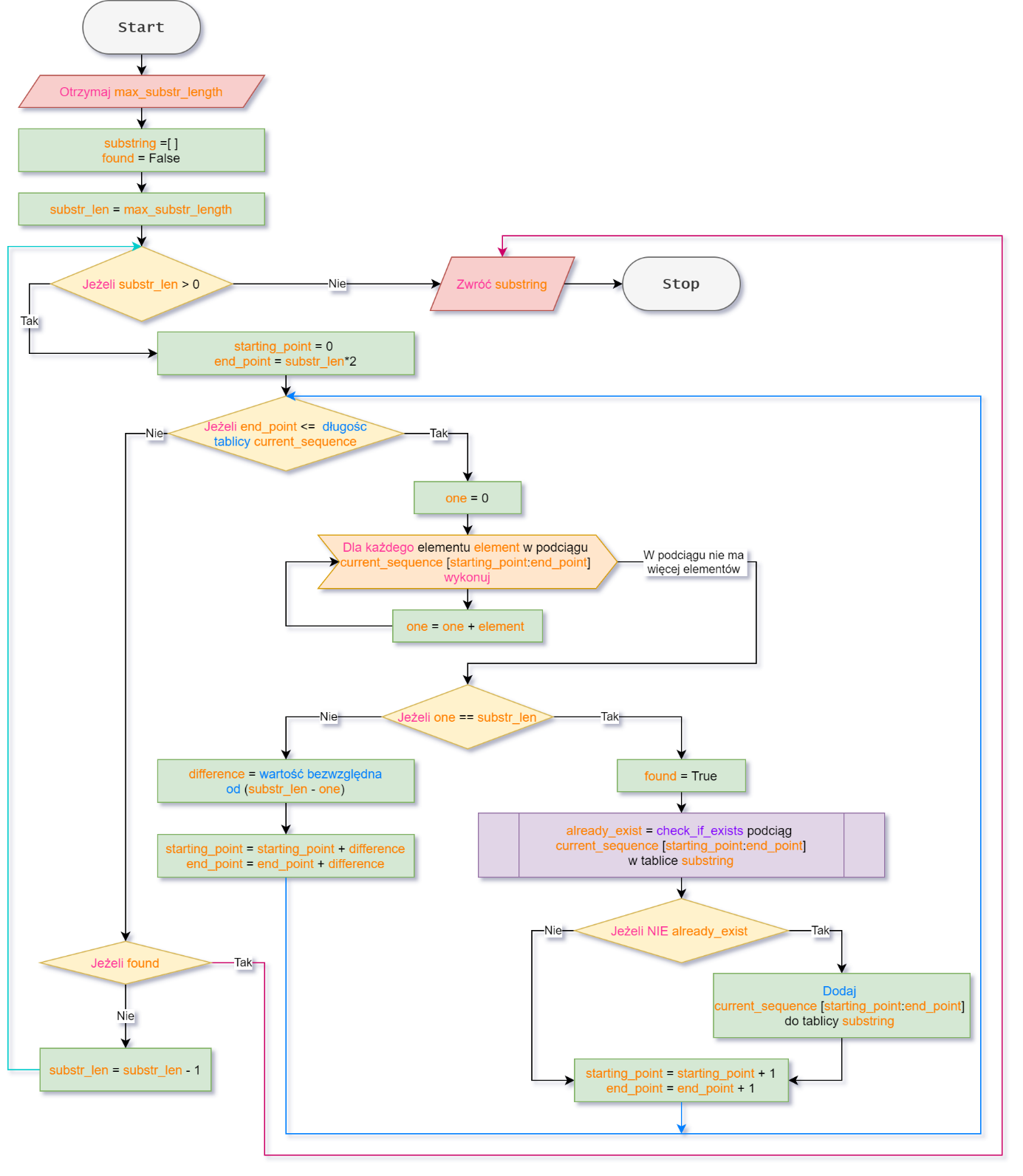


Schemat 4 Kroki działania aktualnego algorytmy dla ciągu 10111111111111111010.  
Źródło: opracowanie własne.

gdzie difference – to odległość pomiędzy początkami aktualnego i następnego podciągów. Inaczej mówiąc, gdy w aktualnym podciągu jest na elementów więcej niż powinno być, to znaczy, że musimy przenieść się przynajmniej na elementów po ciągu, żeby uzupełnić warunek równości zer a jedynek. Schemat działania kolejnego ulepszonego algorytmu pokazany na Schemat 4.

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():

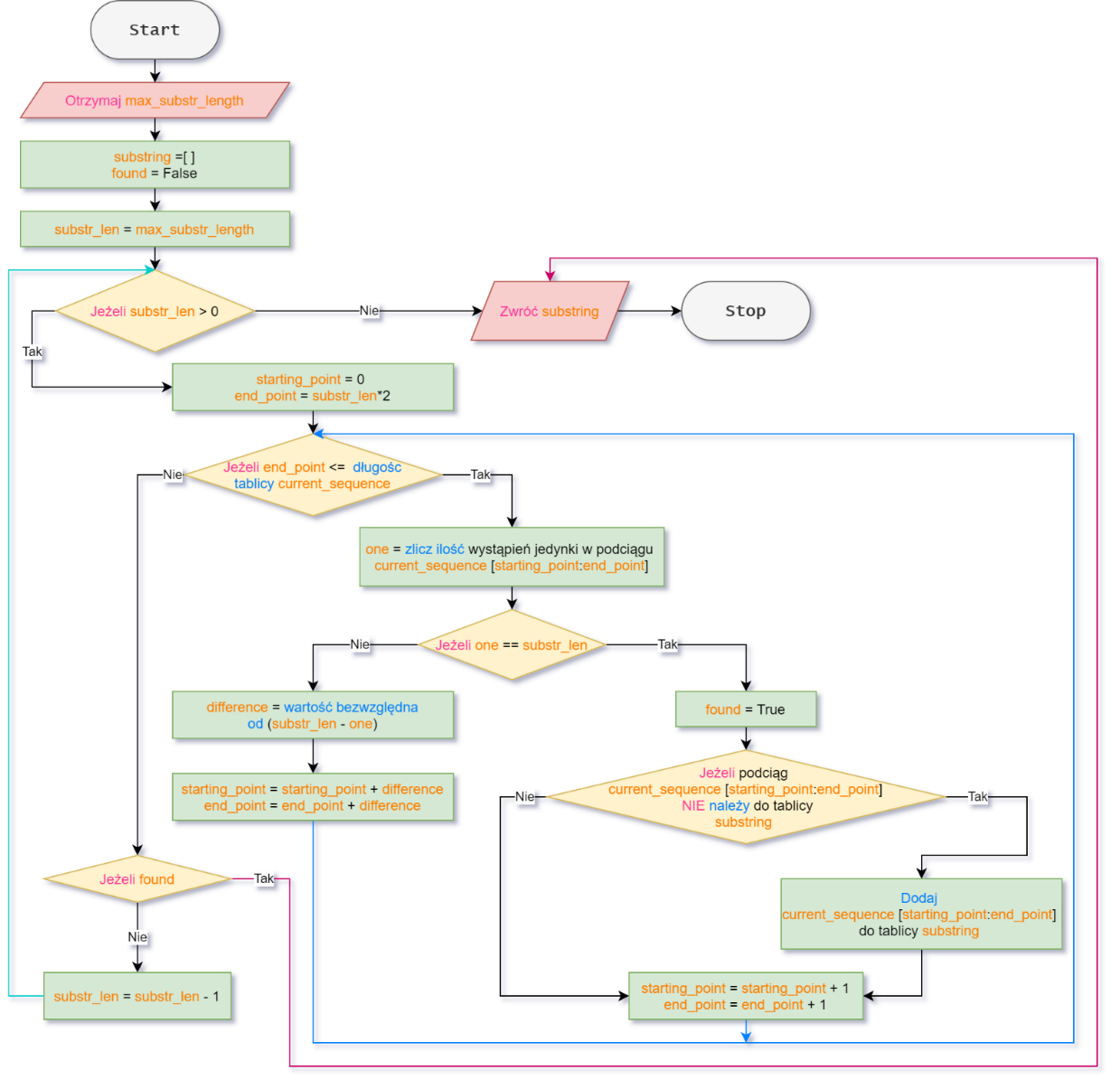
1. Otrzymaj max\_substr\_length // Funkcja przejmuje taki argument
2. substring =[]
3. found = False
4. substr\_len = max\_substr\_length
5. Dopóki substr\_len > 0 wykonuj K06:K28
6. | starting\_point = 0
7. | end\_point = substr\_len\*2
8. | Dopóki end\_point <= długośc tablicy current\_sequence wykonuj K09:K25
9. | | one = 0
10. | | Dla każdego elementu element w podciągu current\_sequence [starting\_point:end\_point] wykonuj K11:
11. | | | one = one + element
12. | | Jeżeli one == substr\_len wykonuj K13:K21
13. | | | found = True
14. | | | already\_exist = False
15. | | | 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.
16. | | | | Jeżeli element == current\_sequence [starting\_point:end\_point] wykonuj K17
17. | | | | | already\_exist = True
18. | | | Jeżeli NIE already\_exist wykonuj K19
19. | | | | Dodaj current\_sequence [starting\_point:end\_point] do tablicy substring
20. | | | starting\_point = starting\_point + 1
21. | | | end\_point = end\_point + 1
22. | | W przeciwnym razie wykonuj K23:K25:
23. | | | difference = abs(substr\_len-one)
24. | | | starting\_point = starting\_point + difference
25. | | | end\_point = end\_point + difference
26. | Jeżeli found wykonuj K27
27. | | break
28. | substr\_len = substr\_len – 1
29. Zwróć substring
30. 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():

1. Otrzymaj max\_substr\_length // Funkcja przejmuje taki argument
2. substring =[]
3. found = False
4. substr\_len = max\_substr\_length
5. Dopóki substr\_len > 0 wykonuj K06:K22
6. | starting\_point = 0
7. | end\_point = substr\_len\*2
8. | Dopóki end\_point <= długośc tablicy current\_sequence wykonuj K09:K15
9. | | one=zlicz ilość wystąpień jedynki w podciągu current\_sequence[starting\_point:end\_point]
10. | | Jeżeli one == substr\_len wykonuj K11:K16
11. | | | found = True
12. | | | Jeżeli podciąg current\_sequence[starting\_point:end\_point] NIE należy do tablicy substring wykonuj K13
13. | | | | Dodaj current\_sequence[starting\_point:end\_point]do tablicy substring
14. | | | starting\_point = starting\_point + 1
15. | | | end\_point = end\_point + 1
16. | | W przeciwnym razie wykonuj K17:K19:
17. | | | difference = abs(substr\_len-one)
18. | | | starting\_point = starting\_point + difference
19. | | | end\_point = end\_point + difference
20. | Jeżeli found wykonuj K21
21. | | break
22. | substr\_len = substr\_len – 1
23. Zwróć substring
24. Zakończ

Schemat blokowy funkcji main\_algorithm\_optimized():



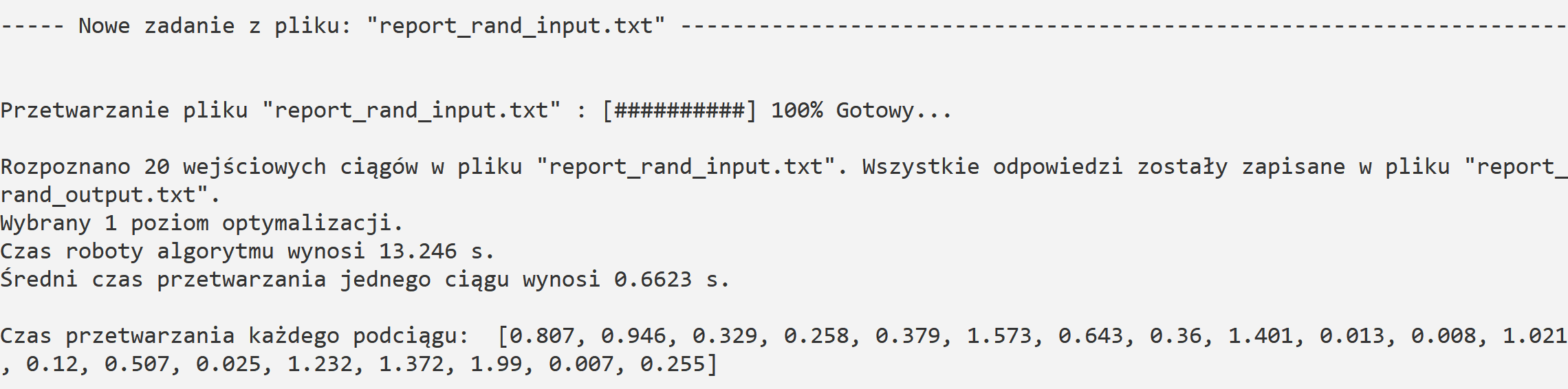
Schemat blokowy 6 main\_algorithm\_optimized()  
Źródło: opracowanie własne.

# 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().

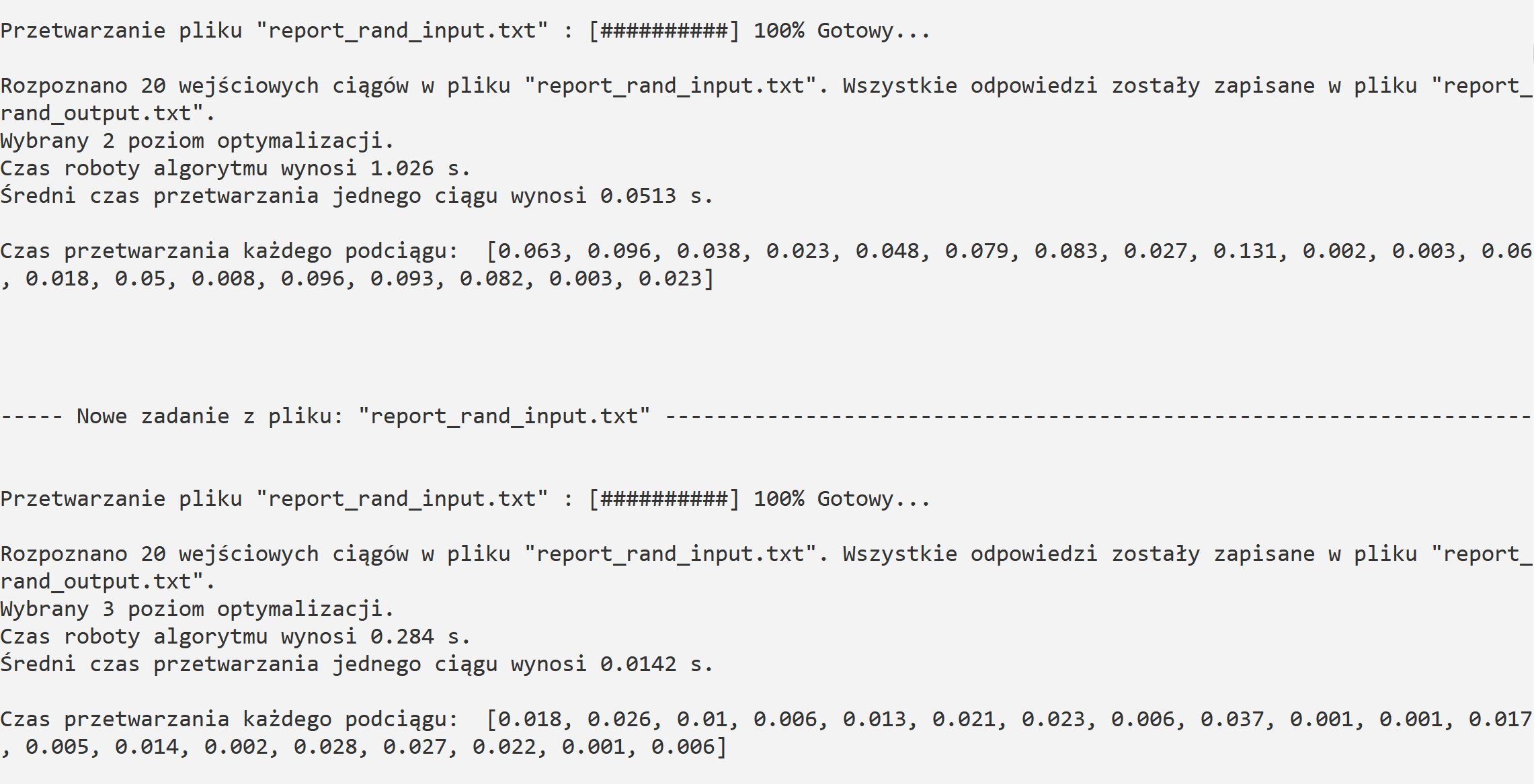
Po uruchomieniu programu otrzymujemy wyniki działania najmniej optymalizowanego algorytmu, podobne do tych, które są na Rysunek 1.



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

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



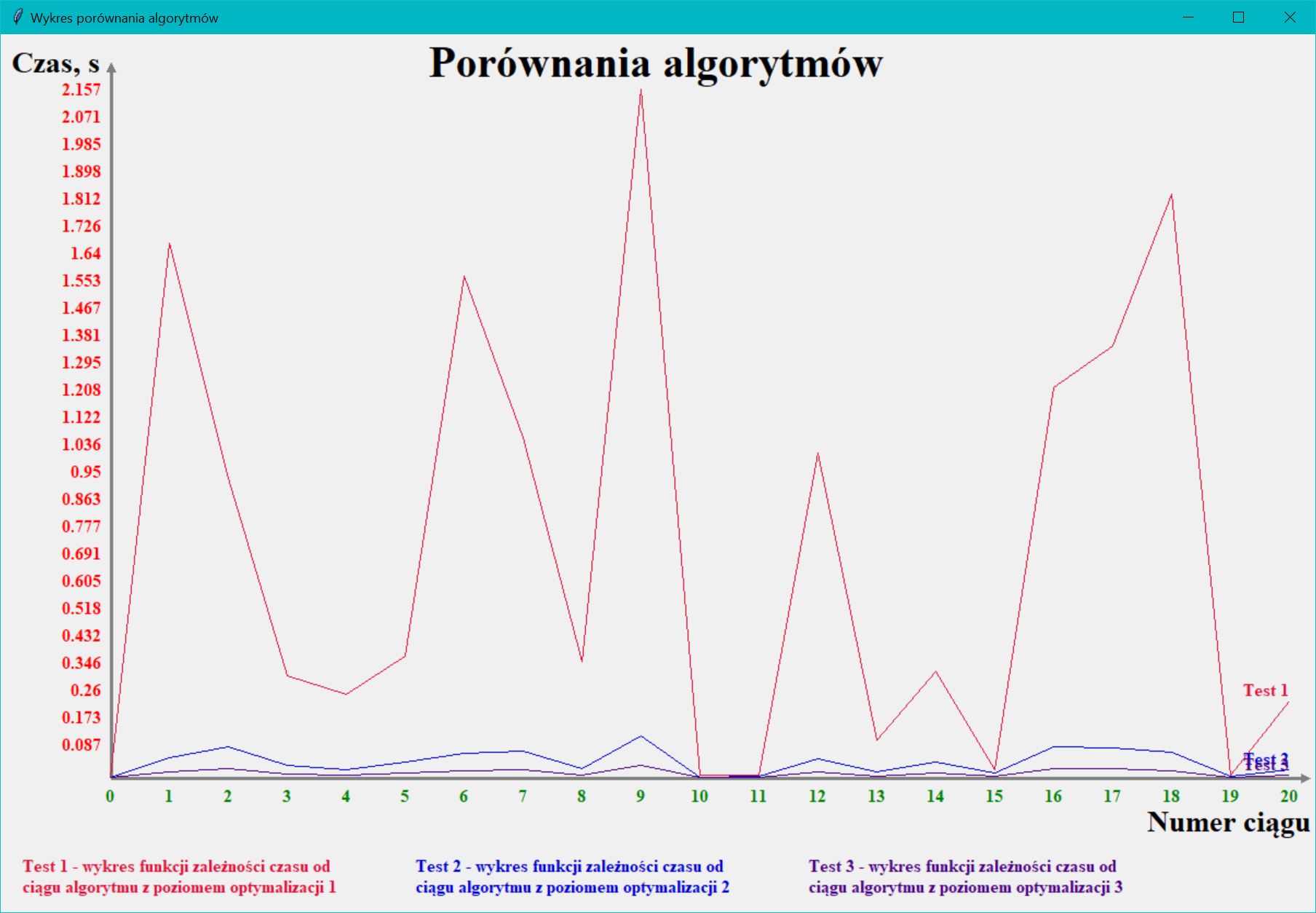
Rysunek 2 Wynik w konsoli po uruchomieniu drugiego i trzeciego algorytmów.  
Źródło: opracowanie własne.

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



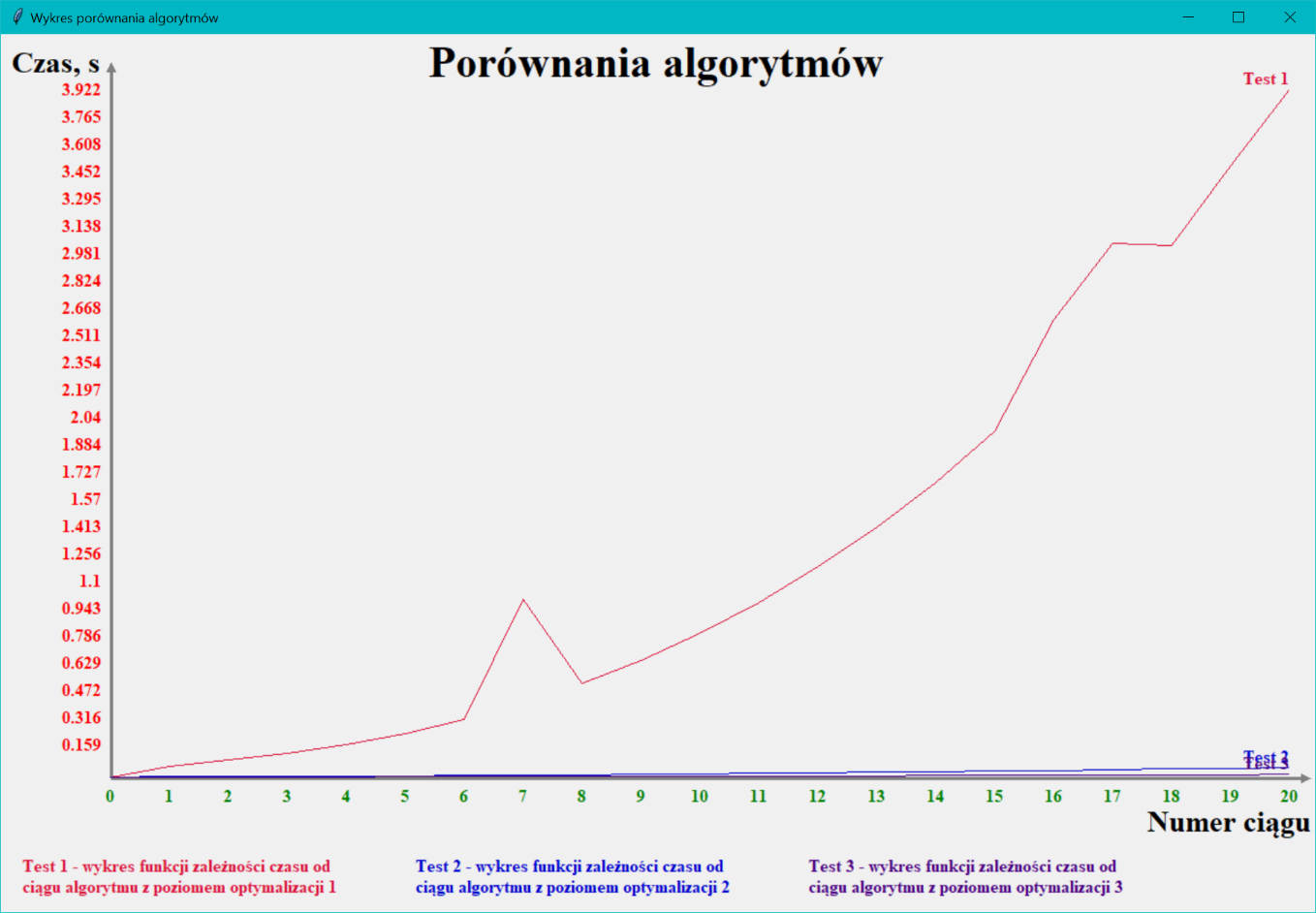
Wykres 1 Porównanie trzech algorytmów w trybie losowych danych wejściowych.  
Źródło: opracowanie własne.

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

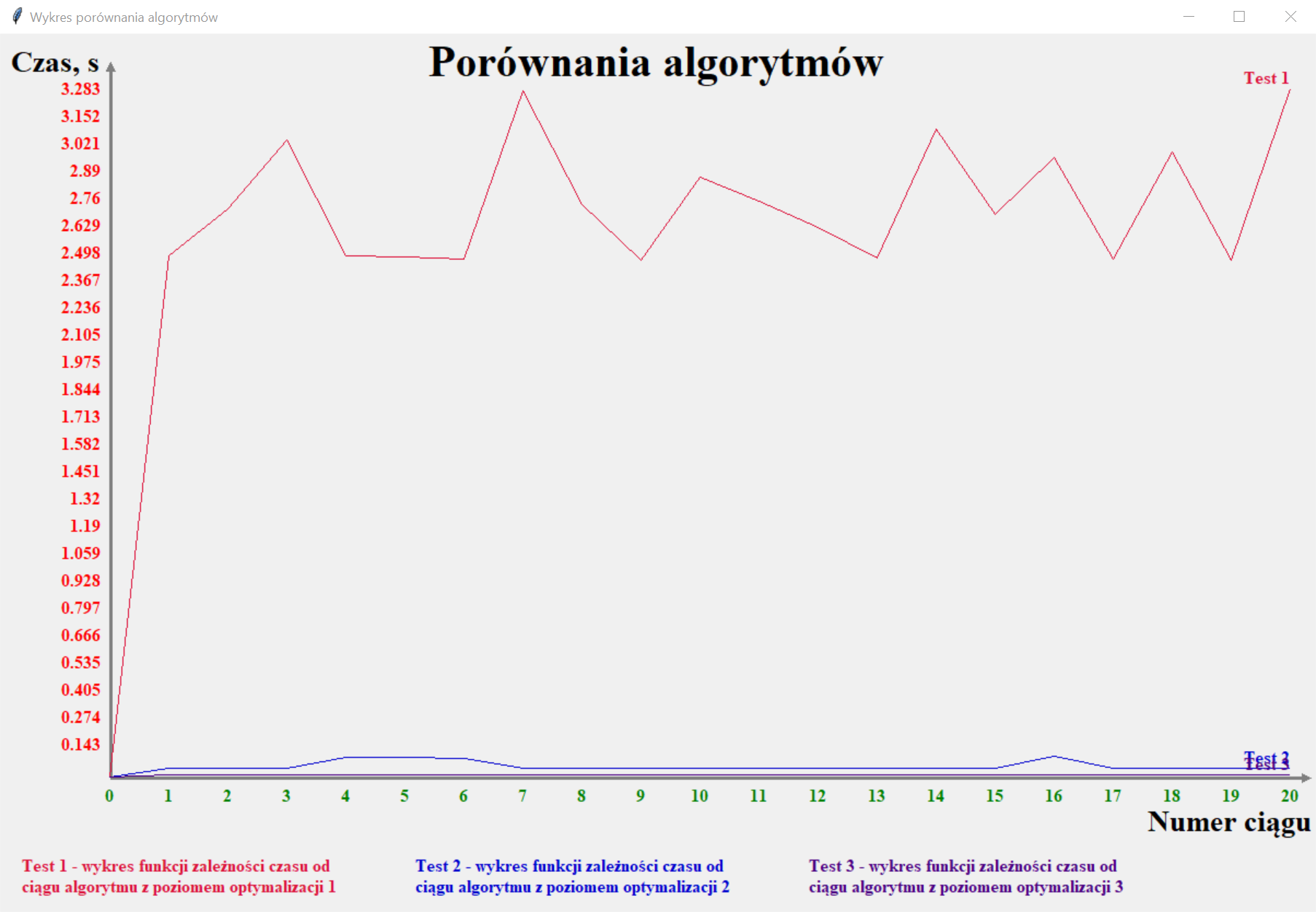
Aby otrzymać plik wejściowy o stałym stosunku zer do jedynek 2/1, należy skorzystać z funkcji worst\_sequence(), którą stworzyłem dla tych celów. Znajduje się ona w pliku pomocniczym assistant\_module.py. Utworzymy plik wejściowy zawierający 999 zer i jedynek z sekwencją „101101101 ...”. Nazwijmy go „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 2 Porównanie trzech algorytmów w trybie zwiększających się danych wejściowych.  
Źródło: opracowanie własne.

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.



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

Rysunek 3 Opcje zapisu pliku wyjściowego.  
Źródło: opracowanie własne.Wykres 3 Porównanie trzech algorytmów w trybie stałych 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

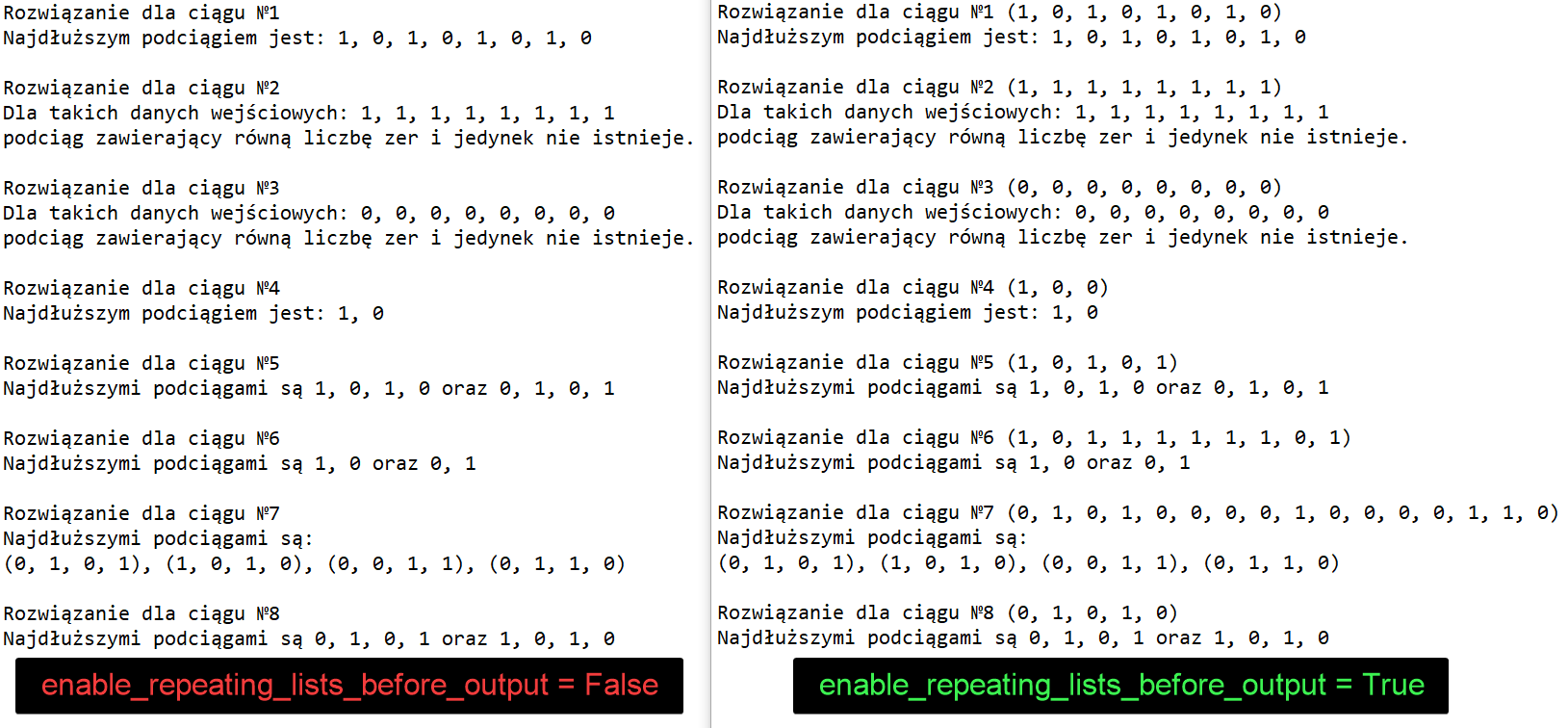
będzie przyjęte jako 4 sekwencji:

[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).



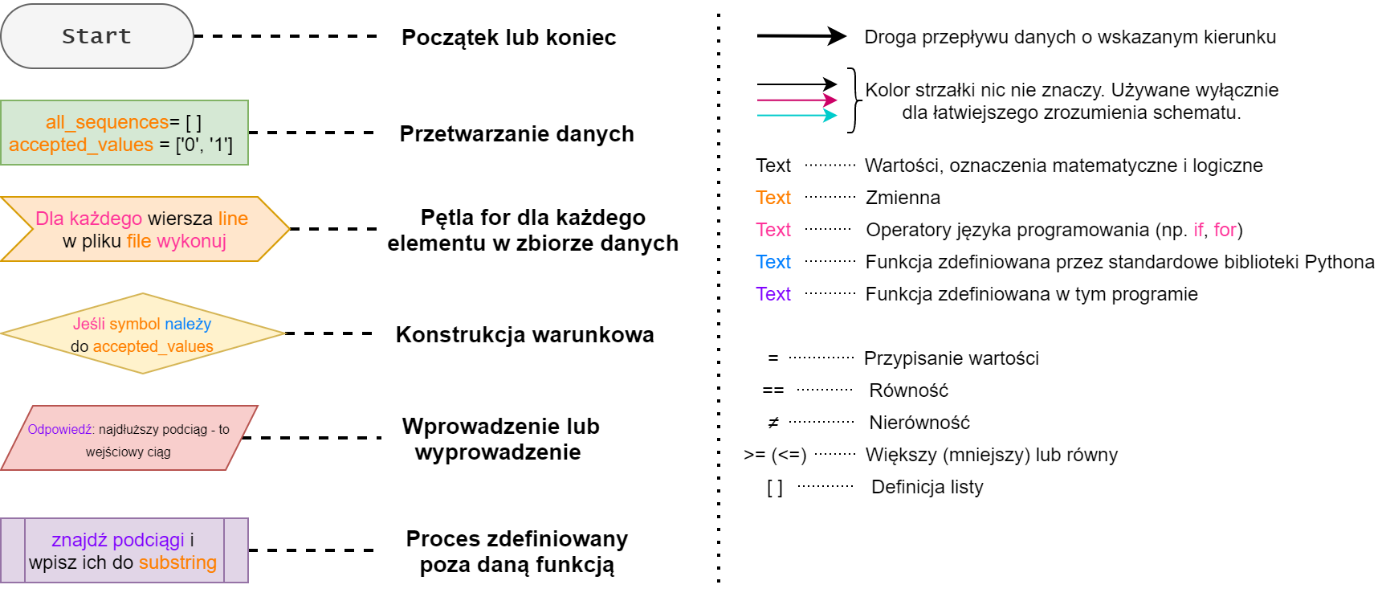
Rysunek 3 Opcje zapisu pliku wyjściowego.  
Źródło: opracowanie własne.

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

[Rysunek 1 Wynik w konsoli po uruchomieniu najmniej optymalizowanego algorytmu. Źródło: opracowanie własne. 14](https://d.docs.live.net/a9a28404614d818a/University/Lectures/Algorytmy%20i%20struktury%20danych/projekt/AiSD%20podciagi.docx#_Toc57418645)

[Rysunek 2 Wynik w konsoli po uruchomieniu drugiego i trzeciego algorytmów. Źródło: opracowanie własne. 14](https://d.docs.live.net/a9a28404614d818a/University/Lectures/Algorytmy%20i%20struktury%20danych/projekt/AiSD%20podciagi.docx#_Toc57418646)

[Rysunek 3 Opcje zapisu pliku wyjściowego. Źródło: opracowanie własne. 17](https://d.docs.live.net/a9a28404614d818a/University/Lectures/Algorytmy%20i%20struktury%20danych/projekt/AiSD%20podciagi.docx#_Toc57418647)

[Rysunek 4 Legenda pseudokodów i schematów blokowych. Źródło: opracowanie własne. 18](https://d.docs.live.net/a9a28404614d818a/University/Lectures/Algorytmy%20i%20struktury%20danych/projekt/AiSD%20podciagi.docx#_Toc57418648)

**Schematy blokowe:**

[Schemat blokowy 1 read\_file() Źródło: opracowanie własne. 4](https://d.docs.live.net/a9a28404614d818a/University/Lectures/Algorytmy%20i%20struktury%20danych/projekt/AiSD%20podciagi.docx#_Toc57452845)

[Schemat blokowy 2 solve\_problem() Źródło: opracowanie własne. 5](https://d.docs.live.net/a9a28404614d818a/University/Lectures/Algorytmy%20i%20struktury%20danych/projekt/AiSD%20podciagi.docx#_Toc57452846)

[Schemat blokowy 3 main\_algorithm\_brute() Źródło: opracowanie własne. 8](https://d.docs.live.net/a9a28404614d818a/University/Lectures/Algorytmy%20i%20struktury%20danych/projekt/AiSD%20podciagi.docx#_Toc57452847)

[Schemat blokowy 4 check\_if\_exists() Źródło: opracowanie własne 8](https://d.docs.live.net/a9a28404614d818a/University/Lectures/Algorytmy%20i%20struktury%20danych/projekt/AiSD%20podciagi.docx#_Toc57452848)

[Schemat blokowy 5 main\_algorithm\_better() Źródło: opracowanie własne. 11](https://d.docs.live.net/a9a28404614d818a/University/Lectures/Algorytmy%20i%20struktury%20danych/projekt/AiSD%20podciagi.docx#_Toc57452849)

[Schemat blokowy 6 main\_algorithm\_optimized() Źródło: opracowanie własne. 13](https://d.docs.live.net/a9a28404614d818a/University/Lectures/Algorytmy%20i%20struktury%20danych/projekt/AiSD%20podciagi.docx#_Toc57452850)

**Schematy:**

[Schemat 1 Kroki działania aktualnego algorytmy dla ciągu 0101000. Źródło: opracowanie własne. 7](https://d.docs.live.net/a9a28404614d818a/University/Lectures/Algorytmy%20i%20struktury%20danych/projekt/AiSD%20podciagi.docx#_Toc57452835)

[Schemat 2 Kroki działania aktualnego algorytmy dla ciągu 0101000. Źródło: opracowanie własne. 7](https://d.docs.live.net/a9a28404614d818a/University/Lectures/Algorytmy%20i%20struktury%20danych/projekt/AiSD%20podciagi.docx#_Toc57452836)

[Schemat 3 Kroki działania aktualnego algorytmy dla ciągu 10111111111111111010. Źródło: opracowanie własne. 10](https://d.docs.live.net/a9a28404614d818a/University/Lectures/Algorytmy%20i%20struktury%20danych/projekt/AiSD%20podciagi.docx#_Toc57452837)

[Schemat 4 Kroki działania aktualnego algorytmy dla ciągu 10111111111111111010. Źródło: opracowanie własne. 10](https://d.docs.live.net/a9a28404614d818a/University/Lectures/Algorytmy%20i%20struktury%20danych/projekt/AiSD%20podciagi.docx#_Toc57452838)

**Wykresy:**

[Wykres 1 Porównanie trzech algorytmów w trybie losowych danych wejściowych. Źródło: opracowanie własne. 15](https://d.docs.live.net/a9a28404614d818a/University/Lectures/Algorytmy%20i%20struktury%20danych/projekt/AiSD%20podciagi.docx#_Toc57452839)

[Wykres 2 Porównanie trzech algorytmów w trybie zwiększających się danych wejściowych. Źródło: opracowanie własne. 16](https://d.docs.live.net/a9a28404614d818a/University/Lectures/Algorytmy%20i%20struktury%20danych/projekt/AiSD%20podciagi.docx#_Toc57452840)

[Wykres 3 Porównanie trzech algorytmów w trybie stałych danych wejściowych. Źródło: opracowanie własne. 16](https://d.docs.live.net/a9a28404614d818a/University/Lectures/Algorytmy%20i%20struktury%20danych/projekt/AiSD%20podciagi.docx#_Toc57452841)

# Bibliografia

Foundation, Python Software. *Python 3.8.6 documentation.* November 27, 2001-2020. https://docs.python.org/3.8/.

planetB. *Syntax Highlight Code In Word Documents.* brak daty. http://www.planetb.ca/syntax-highlight-word.

Stack Exchange Inc. *Stackoverflow.* 2020 . https://stackoverflow.com/questions/3160699/python-progress-bar.

# Kod programu

Utworzono za pomocą (planetB 2018).

## podciagi.py

1. **from** time **import** \*
2. **from** math **import** floor
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\_module.py” nie posiada błedów.")
14. **import** sys, os
15. os.system("pause")
16. sys.exit()


20. **class** binary\_sequences:
22. **def** \_\_init\_\_(self, path\_in = 'input.txt', path\_out = 'output.txt'):
23. ''''' Initialization of the class and assigning default values to flags'''
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


34. **def** draw\_separator(self):
35. ''''' This function draws a "pretty" separator between tasks'''
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')


43. **def** solve\_problem(self):
44. ''''' The main function. Iterates the process of searching substrings for each sequence.
45. After completing the task it writes out small conclusion to the console. '''
47. **global** current\_sequence
48. self.iterator = 0                           # If more than one input was given, then this variable will help to iterate the whole process
49. self.number\_of\_sequences = len(self.all\_sequences)  # Number of sequences given in the input file
50. self.time\_results = []                      # List of records of the time used by algorithm function
52. self.draw\_separator()                       # Drawing a "pretty" separator between tasks
54. **while** self.iterator < self.number\_of\_sequences:
55. current\_sequence = self.all\_sequences[self.iterator]
57. **if** self.show\_progress\_bar:              # Updates the progress bar if the appropriate flag is enabled
58. assist.update\_progress(self.iterator/self.number\_of\_sequences, self.path\_in)
60. self.iterator += 1
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 possible pairs (0,1)
66. **if** n==k!=0:                             # Checking for the easiest solutions 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 ustawienia 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
86. self.time\_results.append(round(end\_time - start\_time,3))
87. self.give\_answer(len(substring), substring)
89. **else**:                                       # Writing of the conclusion with respect to conjugation of polish numerals
91. **if** self.input\_file\_exist **and** **not** self.reading\_file\_error **and** self.number\_of\_sequences>0:
93. **if** self.show\_progress\_bar:
94. assist.update\_progress(self.iterator/self.number\_of\_sequences, self.path\_in)
95. **print**('')
97. last\_digit=list(str(self.number\_of\_sequences)).pop()
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
102. teened = True
104. **if** last\_digit=='1' **and** **not** teened:
105. insert\_text = 'wejściowy ciąg'
106. **elif** last\_digit **in** ['2', '3', '4'] **and** **not** teened:
107. insert\_text = 'wejściowy ciągi'
108. **else**:
109. insert\_text = 'wejściowych ciągów'
111. time\_used = round(sum(self.time\_results),3)
112. avarage\_time\_used = round(time\_used/self.number\_of\_sequences, 5)
114. **print**("Rozpoznano {0} {1} w pliku \"{2}\". ".format(self.number\_of\_sequences, insert\_text, self.path\_in)+
115. "Wszystkie odpowiedzi zostały zapisane w pliku \"{}\". ".format(self.path\_out) +
116. "\nWybrany {} poziom optymalizacji.".format(self.optimization\_level)+
117. "\nCzas roboty algorytmu wynosi {} s.".format(time\_used) +
118. "\nŚredni czas przetwarzania jednego ciągu wynosi {} s.\n".format(avarage\_time\_used))
120. **elif** **not** self.input\_file\_exist **and** self.reading\_file\_error:
121. **print**("\nNie znaleziono pliku według ścieżki \"{}\".\n".format(self.path\_in))
123. **elif** self.input\_file\_exist **and** self.reading\_file\_error:
124. **print**("\n\nCoś poszło nie tak podczas odczytu pliku.\n")
126. **elif** self.number\_of\_sequences==0:
127. **print**("Nie rozpoznano wejściowych ciągów w pliku \"{}\". ".format(self.path\_in))


131. **def** reading\_file(self):
132. ''''' Reading of the file containing input data and creating a clean output file'''
134. with open(self.path\_out,'w', encoding='utf-8') as file:     # Cleaning the output file
135. **pass**
137. self.all\_sequences, accepted\_values = [], ['0', '1']
138. **try**:
139. with open(self.path\_in,'r') as file:                    # Opens the input file and reads it symbol by symbol
140. **for** line **in** file:
141. this\_sequence = []
142. **for** word **in** line.strip().split():
143. **for** symbol **in** word:
144. **if** symbol **in** accepted\_values:           # Checks if the symbol is equal to '0' or '1', and if it is
145. this\_sequence.append(int(symbol))   # then adds that symbol to the sequence
146. **if** this\_sequence != []:
147. self.all\_sequences.append(this\_sequence)
148. **except** FileNotFoundError:
149. self.input\_file\_exist = False
150. self.reading\_file\_error = True
151. **except**:
152. self.input\_file\_exist = True
153. self.reading\_file\_error = True
154. **else**:
155. self.input\_file\_exist = True
156. self.reading\_file\_error = False


160. **def** main\_algorithm\_optimized(self, max\_substr\_length):
161. ''''' Takes a sequence and returns the subsequence so that it would have equal number
162. of digits "zero" & "one". More detailed explanation can be found in my report. '''
164. **global** current\_sequence
165. substring =[]                                                           # Defining the output array of found substrings
166. found = False
167. length\_current\_sequence = len(current\_sequence)
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


189. **def** main\_algorithm\_better(self, max\_substr\_length):
190. ''''' Takes a sequence and returns the subsequence so that it would have equal number
191. of digits "zero" & "one". More detailed explanation can be found in my report. '''
193. **global** current\_sequence
194. substring =[]                                                           # Defining the output array of found substrings
195. found = False
196. length\_current\_sequence = len(current\_sequence)
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


221. **def** main\_algorithm\_brute(self):
222. ''''' Takes a sequence and returns the subsequence so that it would have equal number
223. of digits "zero" & "one". More detailed explanation can be found in my report. '''
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)
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


250. **def** check\_if\_exists(self, prey, sequence):
251. ''''' Checks if the given prey exists in the array "sequence". In other words, it is
252. the equivalent to the native Python method "<object> in <object>".'''
254. **for** element **in** sequence:
255. **if** element == prey:
256. **return** True
257. **else**:
258. **return** False


262. **def** sequence\_to\_string(self, sequence):
263. ''''' Creates a beautiful string from the sequence with comas after each element except for the last one'''
265. string = ''
266. last\_without\_period = str(sequence.pop())
267. **for** element **in** sequence:
268. string += "{}, ".format(element)
269. **return** string + last\_without\_period


273. **def** give\_answer(self, n\_of\_answ, answer = ''):
274. '''''Creates a file in the given path and adds an answer to it with respect to the amount of answers'''
276. with open(self.path\_out,'a', encoding='utf-8') as file:
277. **if** self.iterator != 1:
278. file.write('\n\n')
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')
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.")
289. **elif** n\_of\_answ==1:
290. file.write('Najdłuższym podciągiem jest: ' + self.sequence\_to\_string(answer[0]))
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]))
296. **elif** n\_of\_answ>=3:
297. file.write('Najdłuższymi podciągami są:\n')
298. file.write(self.sequence\_to\_string(['('+self.sequence\_to\_string(ans)+')' **for** ans **in** answer]))


302. **def** porownanie\_algorytmow():
303. ''''' Here I would like to show instructions I used in
304. the paragraph "Porównanie algorytmów" of my report'''
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. """
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 before,
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 report.
315. # assist.random\_sequence(path = ".\\tests\\report\_rand\_input.txt", lines = 20, start\_length = 1000, increment = 0, multiplier = 1)
317. # Creating an object of class binary\_sequences
318. test\_1 = binary\_sequences('.\\tests\\report\_rand\_input.txt', '.\\tests\\report\_rand\_output.txt')
320. # Needed values have to be assigned to flags
321. test\_1.optimization\_level = 1
322. test\_1.show\_progress\_bar = True
324. # Now we can start our test
325. test\_1.solve\_problem()
327. # Since we are going to compare our algorithms, we need the time results information
328. give\_time(test\_1)
330. # Second algorithm test
331. test\_1.optimization\_level = 2
332. test\_1.solve\_problem()
333. give\_time(test\_1)
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)
339. # Creating a comparison of three algorithms using graph
340. # Maybe we would like to comment all upper rows, since this function creates tests 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)
344. assist.worst\_sequence(path = ".\\tests\\report\_worst\_input.txt", lines = 20, start\_repeats = 333, increment = 0, multiplier = 1)
346. algorithm\_comparison(".\\tests\\report\_worst\_input.txt", ".\\tests\\report\_worst\_output.txt",generate\_new\_data = False,
347. show\_progress\_bar = True, width=1200, height=800, show\_stats = False)
349. assist.worst\_sequence(path = ".\\tests\\report\_worst\_increment\_input.txt", lines = 20, start\_repeats = 100, increment = 15, multiplier = 1)
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**


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'''
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\_level, rand\_koef)
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, multiplier)
375. **elif** generate\_new\_data:
376. assist.random\_sequence(path\_in, lines, start\_length, increment, multiplier)
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, optimization\_level)
390. **del** test\_object


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'''
400. **if** worst\_scenario **and** generate\_new\_data:
401. assist.worst\_sequence(path\_in, lines = lines, start\_repeats = start\_repeats, increment = increment, multiplier = multiplier)
402. **elif** generate\_new\_data:
403. assist.random\_sequence(path\_in,  lines = lines, start\_length = start\_length, 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


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)

424. **def** main():
425. '''''
426. Main function of the whole program.
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]
432. For more than one input, please separate them using a new line. For example this 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. '''
442. example\_object = binary\_sequences('input.txt', 'output.txt')    # Input the object of the class binary\_sequences with two
443. # parameters: the path to the input & output files
445. # Default values of the flags:
446. example\_object.enable\_repeating\_lists\_before\_output = False     # Enable to add the repetition of the input data in the output file
447. example\_object.optimization\_level = 3                           # Level of algorithm optimisation. Accepted values: 1, 2, 3 (bigger is better).
448. example\_object.show\_progress\_bar = False                        # Enable to see the progress bar. WORKS WELL ONLY IN CONSOLE. In python Shelf it looks ugly.

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 amounts of data it is likely to mess up the look of the output file.
453. # Since I would appreciate you to read the file 'output.txt', I am going to enable this flag
455. # Launching the main function of the class 'binary\_sequences'
456. example\_object.solve\_problem()

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 same and also adds some pretty text
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

470. test(                                                           # You can also create tests using test() function, where:
471. optimization\_level = 1,                                     # - level of algorithm optimisation, accepted values: 1, 2, 3;
472. path\_in = '.\\tests\\input\_worst\_scenario.txt',             # - path, where new input file will be generated or existing file opened;
473. path\_out = '.\\tests\\output\_worst\_scenario.txt',           # - path, where new output file will be created;
474. return\_time = True,                                         # - flag, which asks if you would like to see used time results in console;
475. generate\_new\_data = True,                                   # - flag, which asks if you would like to create a new input file or use an existing one;
476. show\_progress\_bar = True,                                   # - flag, which asks if you would like to see the progress bar (WORKS WELL ONLY IN CONSOLE);
478. # Next options make changes only if generate\_new\_data flag is set to True:
479. worst\_scenario = True,                                      # - flag, which 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                                              # - multiplication of (repeats / length) of the (sequence '101' / line) after each line.
486. )

489. algorithm\_comparison(                                           # You can also create algorithm comparison using algorithm\_comparison() function, where:
490. path\_in = '.\\tests\\inp\_comparison.txt',                   # - path, where new input file will be generated;
491. path\_out = '.\\tests\\out\_comparison.txt',                  # - path, where new output file will be created;
492. generate\_new\_data = True,                                   # - flag, which asks if you would like to create a new input file or use an existing one;
493. show\_progress\_bar = True,                                   # - flag, which 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 the created graph (minimum 650, recommended 1200);
496. show\_stats = True,                                          # - shows the information about the increment and multiplier on the plot;
498. # Next options make changes only if generate\_new\_data flag is set to True:
499. worst\_scenario = True,                                      # - flag, which 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                                              # - multiplication of (repeats / length) of the (sequence '101' / line) after each line.
506. )

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()
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")


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"]                                                           # Possible values in the test sequence
7. worst\_situaion = '101'                                                          # In my opinion, it is the worst possible input sequence, if repeated
9. **class** Graph:
10. ''''' Creates a graph of the times used to process all
11. substrings using "tkinter" and other built-in python methods'''
13. **def** \_\_init\_\_(self, show\_stats = False, worst\_scenario=False, increment=0, multiplier=1, width=1200, height=400, text\_density=25):
14. self.time\_records = []
15. self.optimalization\_level = []
17. master = tk.Tk()                                                        # Creating a window for the graph
18. master.title("Wykres porównania algorytmów")
20. **if** height < 200:
21. height = 200
22. **if** width < 650:
23. width = 650
25. self.width, self.height = width, height                                 # Defining some basic distances
26. self.legend\_x, self.legend\_y, self.title, self.post\_graph\_area = 100, 120, 50, 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

32. self.canv = tk.Canvas(master, width=self.width, height=self.height)     # Creating a canvas to write on
33. self.canv.pack()
35. self.times12 = tkinter.font.Font(family='Times',                        # Defining 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', 'saddlebrown', 'darkgoldenrod', 'black']
43. # Creating a plot
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.LAST )
47. self.canv.create\_line( self.legend\_x+1, self.title+self.graph\_height, self.legend\_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 algorytmów", 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")
54. **if** worst\_scenario **and** show\_stats:
55. self.canv.create\_text(self.width/2, self.title/2 + 18,  text = "najgorsze dane wejściowe", font = self.times20, anchor = tk.N, fill="black")
56. self.canv.create\_text(self.width/2, self.title+self.graph\_height+self.legend\_y/4,
57. text = "length increment: {}  multiplier: {}".format(increment\*3, multiplier), 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 = "przypadkowe dane wejściowe", font = self.times20, anchor = tk.N, fill="black")
60. self.canv.create\_text(self.width/2, self.title+self.graph\_height+self.legend\_y/4,
61. text = "length increment: {}  multiplier: {}".format(increment, multiplier), font = self.times12, anchor = tk.N, fill="black")


65. **def** save\_new\_data(self, times\_info, optimalization\_level):
66. ''''' This function takes data and writes it down'''
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)


76. **def** paint\_graph(self):
77. ''''' This function creates a graph after all data is collected.'''
79. **if** self.time\_records == []:                                             # Checking if data exist at all
80. **print**("Za mało danych, aby narysować wykres!")
81. **return** -1
83. all\_len, all\_max, all\_min, all\_sum = [], [], [], []                     # Defining 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)

92. quantity\_labels = floor(self.graph\_height/self.density)                 # Defining 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")
99. quantity\_labels = floor(self.graph\_width/self.density)                  # Defining 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")

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\_graph\_area, self.height - 10,
112. text = "Test {0} - wykres funkcji zależności czasu od\nciągu algorytmu z poziomem optymalizacji {1}".format(i+1, self.optimalization\_level[i]),
113. font = self.times12, anchor = tk.SW, fill=self.colors[(i+1)%len(self.colors)])
114. tk.mainloop()

117. **def** graph\_painter(self, times\_info, number):
118. ''''' This function draws a graph of values given in the array times\_info'''
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)


136. **def** random\_sequence(path = "input\_rand.txt", lines = 10, start\_length = 50, increment = 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'''
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. """

157. **def** worst\_sequence(path = "input\_worst.txt", lines = 10, start\_repeats = 30, increment = 0, multiplier = 1):
158. ''''' This function creates file filled with worst sequence of digit "zero" and "one".
159. More detailed explanation is given below the function'''
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 line.
173. """

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%.'''
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()