



**POLITECHNIKA LUBELSKA  
WYDZIAŁ ELEKTROTECHNIKI I  
INFORMATYKI**

**KIERUNEK STUDIÓW  
INFORMATYKA**

*MATERIAŁY DO ZAJĘĆ  
LABORATORYJNYCH*

**Wprowadzenie do informatyki**

Autor:  
mgr inż. Marek Kamiński

Lublin 2019

## **LABORATORIUM 3. KLASYCZNE ALGORYTMY, WYSZUKIWANIE I SORTOWANIE.**

### **Cel laboratorium:**

Przedstawienie zasady działania algorytmów operujących na zbiorach danych. Przedstawienie obsługi tablic wielowymiarowych oraz technik trwałego przechowywania danych.

### **Zakres tematyczny zajęć:**

1. Realizacja algorytmów klasycznych
  - a. Znajdowanie liczb pierwszych w określonym przedziale liczbowym
2. Sortowanie i wyszukiwanie danych
  - a. Sortowanie bąbelkowe
  - b. Trwałe przechowywanie danych
  - c. Wyszukiwanie binarne
  - d. Obsługa wyjątków
3. Tablice wielowymiarowe
  - a. Definicja tablicy wielowymiarowej
  - b. Przykłady przedstawiające wykorzystanie tablic dwuwymiarowych

### **Pytania kontrolne:**

1. Czym jest operacja modulo?
2. Czym jest tablica?
3. W jaki sposób można odnaleźć w tablicy wszystkie elementy parzyste?
4. Czym są funkcje? Jak się je definiuje i jak się ich używa?



**Fundusze  
Europejskie**  
Wiedza Edukacja Rozwój



**Rzeczpospolita  
Polska**

**Unia Europejska**  
Europejski Fundusz Społeczny



## **ROZDZIAŁ 9. REALIZACJA ALGORYTMÓW KLASYCZNYCH.**

### **Cel rozdziału:**

Celem rozdziału jest zapoznanie czytelnika z zasadą funkcjonowania podstawowych algorytmów operujących na danych liczbowych.

### **Zakres tematyczny:**

1. Znajdowanie liczb pierwszych w określonym przedziale liczbowym.

### **Pytania kontrolne:**

1. Czym jest operacja modulo?
2. Czym jest tablica?
3. W jaki sposób można odnaleźć w tablicy wszystkie elementy parzyste?

### **1. Znajdowanie liczb pierwszych w określonym przedziale liczbowym.**

Pierwszym omówionym algorytmem będzie algorytm znany pod nazwą Sita Eratostenesa, który pozwala na znalezienie wszystkich liczb pierwszych w określonym przedziale liczbowym.

#### Kącik autora

Metoda przypisywana jest Eratostenesowi urodzonemu w roku 276 p.n.e., co oznacza, że algorytm powstał przeszło 2000 lat temu! – na długo przed powstaniem pierwszego komputera opartego na technice tranzystorowej.

Metoda realizowana jest w oparciu o jednowymiarową tablicę, która przechowuje będzie status liczb począwszy od liczby 2, a skończywszy na wskazanej przez użytkownika wartości. Każda kolejna komórka w tablicy przechowuje status odpowiadającej jej liczbie całkowitej, tj.

- `tab[0]` przechowuje informację o tym, czy liczba 2 jest liczbą pierwszą,
- `tab[1]` przechowuje informację o tym, czy liczba 3 jest liczbą pierwszą,
- stąd `tab[i]` przechowuje informację o tym, czy liczba  $i+2$  jest liczbą pierwszą.

W zależności od implementacji, do oznaczania statusu liczby używane są najczęściej: typ logiczny (prawda-fałsz) lub typ liczbowy (0 dla liczby pierwszej, 1 w przeciwnym razie). Na początku działania wszystkie liczby powinny być oznaczone, jako liczby pierwsze (rys. 9.1).



Fundusze  
Europejskie  
Wiedza Edukacja Rozwój



Rzeczpospolita  
Polska



Unia Europejska  
Europejski Fundusz Społeczny

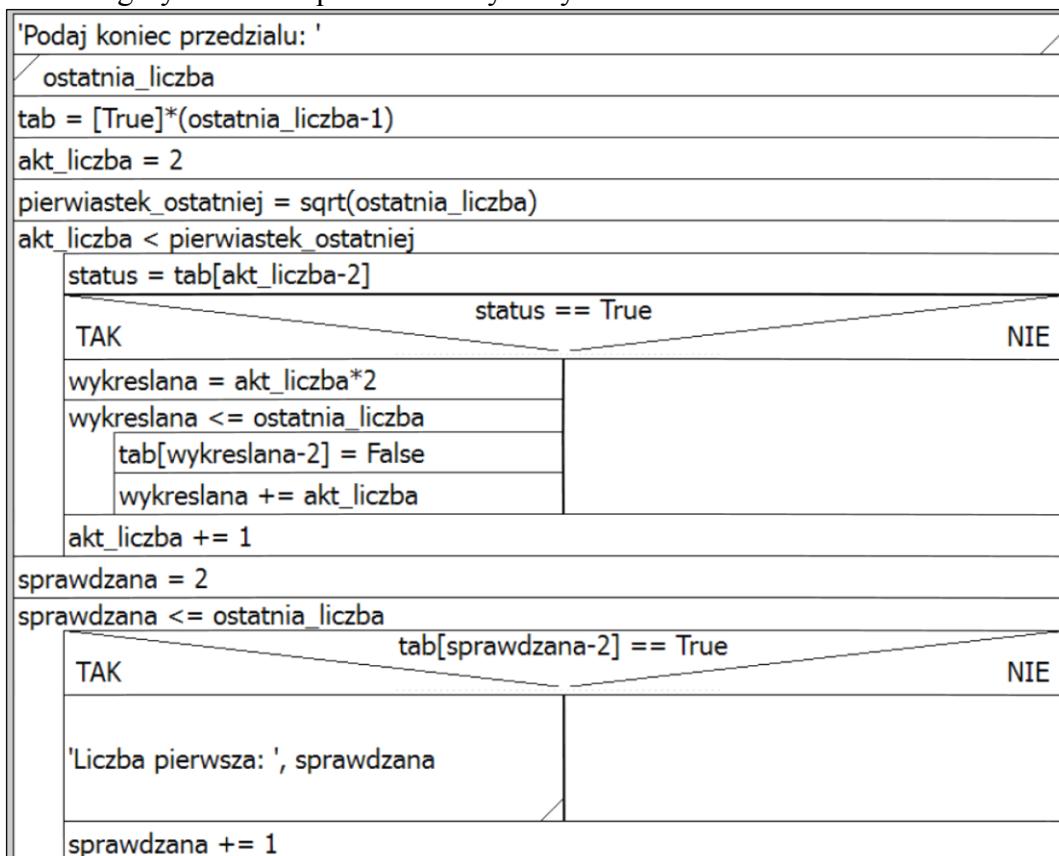
2	3	4	5	6	7	8	9	10	11
True									
tab[0]	tab[1]	tab[2]	tab[3]	tab[4]	tab[5]	tab[6]	tab[7]	tab[8]	tab[9]

Rys. 9.1. Zawartość pierwszych 10 elementów tablicy przed rozpoczęciem pracy algorytmu

Po odpowiedniej inicjalizacji tablicy oraz pobraniu wartości końca przedziału poszukiwań, następuje przejście do części głównej algorytmu, którą można opisać następująco:

1. Przed wejściem do pętli przypisz aktualnej liczbie wartość 2.
2. Wykonuj pętlę do momentu, gdy wartość aktualnej liczby będzie mniejsza od wartości pierwiastka kwadratowego liczby określającej koniec przedziału.
3. Pobierz status aktualnej liczby – w przypadku, gdy jest oznaczona jako True, wykreśl z tablicy (poprzez przypisanie wartości False) wszystkie liczby odpowiadające wielokrotności aktualnej liczby.
4. Zwięksź wartość aktualnej liczby i przejdź do kroku drugiego.

Schemat NS algorytmu został przedstawiony na rys. 9.2.



Rys. 9.2. Schemat NS Sita Eratostenesa

Stan tablicy po dwóch pierwszych iteracjach przedstawiony został na rys. 9.3.



Fundusze Europejskie  
Wiedza Edukacja Rozwój



Rzeczypospolita  
Polska



Unia Europejska  
Europejski Fundusz Społeczny

2	3	4	5	6	7	8	9	10	11
True	True	False	True	False	True	False	True	False	True
tab[0]	tab[1]	tab[2]	tab[3]	tab[4]	tab[5]	tab[6]	tab[7]	tab[8]	tab[9]
2	3	4	5	6	7	8	9	10	11
True	True	False	True	False	True	False	False	False	True
tab[0]	tab[1]	tab[2]	tab[3]	tab[4]	tab[5]	tab[6]	tab[7]	tab[8]	tab[9]

Rys. 9.3. Zawartość pierwszych 10 elementów tablicy po dwóch pierwszych przebiegach pętli (wykreślono wielokrotności liczby dwa oraz liczby trzy)

W przypadku podania jako koniec przedziału liczby mniejszej od 16, rysunek przedstawia również efekt końcowy działania aplikacji (pierwiastek kwadratowy liczby mniejszej od 16 jest mniejszy od czterech, co oznacza, że pętla wykona się jedynie dwa razy). W pierwszym wierszu zaznaczono wykreślone wielokrotności liczby 2, w drugim natomiast wielokrotności liczby 3.

Po wyjściu z pętli pozostaje jedynie ostatni raz przejść przez każdy z elementów tablicy i sprawdzić, czy jego wartość wynosi True, czy False – w przypadku wartości True, wskazywana liczba jest liczbą pierwszą (na rys. 9.3. będą to liczby: 2, 3, 5, 7 oraz 11).

### Zadanie

1. Zaimplementować algorytm Sita Eratostenesa w języku Python w postaci funkcji `znajdz_liczby_pierwsze()`, która jako parametr przyjmie wartość końca przedziału poszukiwań. Funkcja powinna zwrócić nową tablicę zawierającą znalezione liczby pierwsze.
2. Zaproponować algorytm sprawdzenia, czy podana przez użytkownika liczba jest liczbą pierwszą bez wykorzystania tablicy.



## **ROZDZIAŁ 10. SORTOWANIE I WYSZUKIWANIE DANYCH.**

### **Cel rozdziału:**

Zaznajomienie z zasadą funkcjonowania podstawowych algorytmów operujących na danych liczbowych w postaci tablicowej. Przedstawienie sposobu trwałego przechowywania danych oraz obsługi sytuacji wyjątkowych w programie.

### **Zakres tematyczny:**

1. Sortowanie bąbelkowe
2. Trwałe przechowywanie danych
3. Wyszukiwanie binarne
4. Wstęp do obsługi wyjątków

### **Pytania kontrolne:**

1. Jak są reprezentowane różne typy zmiennych w pamięci komputera?
2. Czym jest tablica?
3. Jak wygląda budowa funkcji?

### **1. Sortowanie bąbelkowe**

Nieraz zdarza się sytuacja, w której operując na pewnym zbiorze danych zachodzi chęć uporządkowania zbioru w określony sposób. Operacja ta nosi nazwę sortowania. W informatyce, na przestrzeni lat opracowano wiele różnych algorytmów sortowania, z których każdy charakteryzuje się odmienną implementacją oraz złożonością obliczeniową. W tym rozdziale przedstawiona zostanie technika znana pod nazwą sortowania bąbelkowego.

Na wstępie należy zaznaczyć, iż sortowanie bąbelkowe nie należy do najszybszych technik sortowania (problem złożoności obliczeniowej zostanie przedstawiony w laboratorium szóstym). To, co wyróżnia sortowanie bąbelkowe to jego niezwykle prosta implementacja. W każdym cyklu pętli głównej pośrednio wyszukiwany jest element największy („bąbelek”), który „wypływa” na koniec tablicy. Podczas procesu wyszukiwania porównywane są pary sąsiadujących ze sobą wartości i zamieniane są tak, aby element większy znalazł się po prawej stronie. Z każdym kolejnym cyklem pętli zmniejsza się jej zasięg (na końcu tablicy umieszczane są sukcesywnie największe wartości), by ostatecznie zakończyć działanie w momencie, gdy nie pozostanie już żadna para do sprawdzenia. Diagram NS przedstawiający zasadę działania algorytmu przedstawiony został na rys. 10.1.

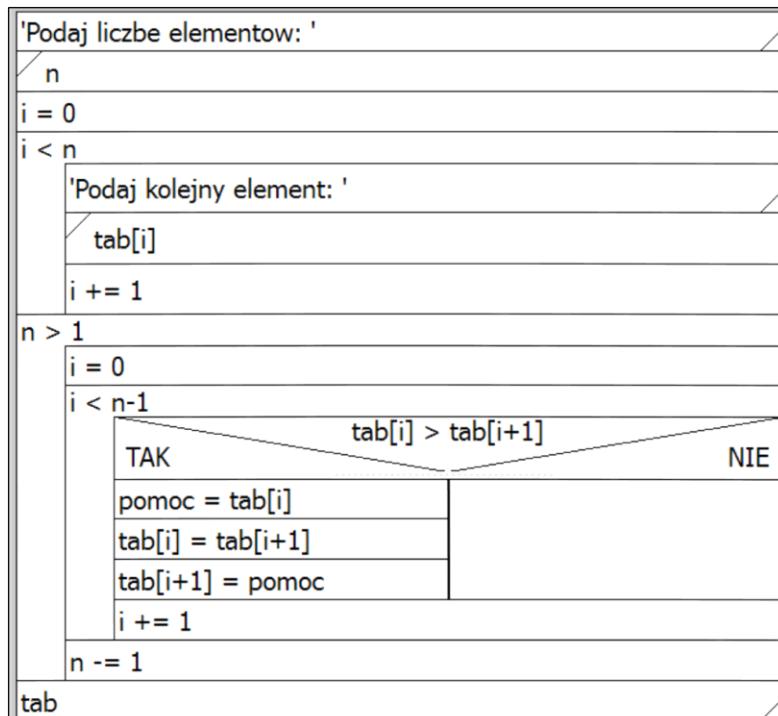


**Fundusze Europejskie**  
Wiedza Edukacja Rozwój



**Rzeczpospolita Polska**





Rys. 10.1. Diagram NS sortowania bąbelkowego

W celu zobrazowania zasady działania algorytmu, na rys. 10.2. przedstawiony został również przykładowy przebieg programu dla tablicy ograniczonej do pięciu elementów.

czy $7 > 3?$ zamiana	7	3	5	8	2
czy $7 > 5?$ zamiana	3	7	5	8	2
czy $7 > 8?$ bez zmian	3	5	7	8	2
czy $8 > 2?$ zamiana	3	5	7	8	2
czy $3 > 5?$ bez zmian	3	5	7	2	8
czy $5 > 7?$ bez zmian	3	5	7	2	8
czy $7 > 2?$ zamiana	3	5	7	2	8
czy $3 > 5?$ bez zmian	3	5	2	7	8
czy $5 > 2?$ zamiana	3	5	2	7	8
czy $3 > 2?$ zamiana	3	2	5	7	8
wynik sortowania	2	3	5	7	8

Rys. 10.2. Zawartość tablicy dla kolejnych przebiegów pętli

Kolorem pomarańczowym oznaczone zostały wszystkie porównania w danym cyklu pętli, podczas gdy kolor niebieski użyty został do przedstawienia elementów znajdujących się już na finalnej pozycji.



Fundusze Europejskie  
Wiedza Edukacja Rozwój



Rzeczypospolita  
Polska

Unia Europejska  
Europejski Fundusz Społeczny



## 2. Trwałe przechowywanie danych

W trakcie procesu sortowania wykorzystywana jest moc obliczeniowa procesora – w przypadku większych zbiorów danych, wielokrotne sortowanie tych samych danych wejściowych oznacza marnowanie zasobów jednostki centralnej. Aby uniknąć tej sytuacji skorzystać można z możliwości zapisu raz posortowanych danych do zewnętrznego pliku na dysku twardym komputera (lub innym nośniku pamięci), by później móc te dane w dowolnym momencie odczytać.

W języku Python na potrzeby operacji dyskowych wykorzystywane są zazwyczaj następujące funkcje:

- `open(nazwa_pliku, tryb_pracy)` – funkcja pozwala na utworzenie lub otwarcie istniejącego już pliku o podanej nazwie (ścieżce). Utworzenie pliku skutkuje utworzeniem obiektu pliku, który należy przypisać do zmiennej.

Wśród najczęściej używanych trybów pracy wyróżnić można:

- "a" – otwarcie pliku w trybie dopisywania. W tym trybie wszystkie zapisywane dane dopisywane są na koniec pliku (istniejące dane nie są nadpisywane) W przypadku, gdy plik nie istnieje, zostanie on utworzony.
  - "w" – otwarcie pliku w trybie zapisu. Po otworzeniu pliku w tym trybie, wszystkie poprzednie dane znajdujące się w pliku zostają usunięte. W przypadku, gdy plik nie istnieje, zostanie on utworzony.
  - "r" – otwarcie pliku w trybie odczytu. Domyslny tryb, w którym można z pliku jedynie odczytać dane (nie jest możliwy zapis). Funkcja zwróci błąd w przypadku nieistniejącego pliku.
  - "x" – utworzenie pliku o podanej nazwie. Tryb pozwala na zabezpieczenie się przed przypadkowym nadaniem istniejącego pliku. W trybie tym, plik zostanie utworzony i otwarty jedynie w momencie, gdy nie istnieje już plik o takiej nazwie – w przeciwnym razie zostanie zwrócony błąd.
- `write(dane_do_zapisu)` – funkcja wykorzystywana do zapisu danych do pliku zwróconego wcześniej przez funkcję `open()`.
  - `read()` – funkcja wykorzystywana do odczytu danych z pliku zwróconego wcześniej przez funkcję `open()`. Funkcja zwraca odczytane dane, które należy przypisać do określonej zmiennej.
  - `close()` – po wykonaniu wszystkich niezbędnych operacji należy zamknąć plik (gwarantuje to między innymi zapisanie wszystkich danych wskazanych do zapisania poprzez użycie funkcji `write()` – zdarza się, że system operacyjny w celu optymalizacji zapisu czeka na zebranie się większej ilości danych).

Dysponując powyższą wiedzą można przeprowadzić proces odczytu danych do posortowania, by wyniki zapisać później do określonego pliku. Takie podejście pozwala na opracowanie



Fundusze Europejskie  
Wiedza Edukacja Rozwój



Rzeczpospolita  
Polska



Unia Europejska  
Europejski Fundusz Społeczny

programu wykazującego się większą uniwersalnością – nie ma potrzeby każdorazowego wczytywania danych od użytkownika. Jednocześnie istnieje możliwość pracy z posortowanymi danymi w innych aplikacjach, w których możliwe jest otwarcie i import danych z pliku.

Praktyczny sposób wykorzystania przedstawionych powyżej funkcji został zaprezentowany na rys. 10.3.

```
1  separator = ','
2  tab = []
3
4  plik_we = open("wejście.txt", "r")
5  rozbita_tablica = plik_we.read().split(separator)
6  for wartosc in rozbita_tablica:
7      tab.append(int(wartosc))
8  plik_we.close()
9
10 # miejsce na kod wykonujacy sortowanie
11
12 i = 0
13 plik_wy = open('wykonanie.txt', 'w')
14 for wartosc in tab:
15     if i == len(tab)-1:
16         separator = ''
17
18     plik_wy.write(str(wartosc) + separator)
19     i += 1
20
21 plik_wy.close()
```

Rys. 10.3. Odczyt danych wejściowych z pliku oraz ich ponowny zapis

W pierwszej linii zdefiniowany został separator danych w plikach (zarówno wejściowym, jak i wyjściowym). Obecnie jest nim symbol przecinka, co oznacza, że dane umieszczone są w plikach w formacie: "liczba1, liczba2, liczba3, liczba4" itd. W przypadku chęci umieszczenia każdej wartości w oddzielnej linii należy zmienić wartość zmiennej separator na wartość nowej linii (tj. '\n'). Odczytane dane zostaną umieszczone w tablicy, stąd w drugiej linii pojawia się jej deklaracja.

W dalszej części kodu, pomiędzy linią czwartą, a ósmą pojawia się fragment odpowiedzialny za odczyt danych z pliku oraz zamianę danych na postać liczbową przy jednoczesnym umieszczeniu ich w wcześniej zadeklarowanej tablicy. Zgodnie z przedstawionymi wcześniej funkcjami, do otwarcia pliku w trybie tylko do odczytu odpowiada funkcja `open()` – pierwszy argument określa nazwę pliku do odczytu (może to być również pełna ścieżka). W tym przypadku jest to plik `wejście.txt` znajdujący się w folderze projektu. Drugi argument w postaci jednego znaku `r` mówi o otwarciu w trybie do odczytu.

Dodatkowe pytania może rodzić dalszy kod – do odczytu danych z pliku wykorzystano funkcję `read()`, jednak efekt jej działania został natychmiast przekazany do funkcji



Fundusze Europejskie  
Wiedza Edukacja Rozwój



Rzeczpospolita  
Polska



Unia Europejska  
Europejski Fundusz Społeczny

`split()`. Wykorzystanie funkcji `read()` oznacza odczytanie całej zawartości pliku, stąd dla pięciu przykładowych wartości efektem jej działania mogłoby być zwrocenie ciągu postaci: "3, 5, 2, 1, 7" w formie tekstowej. W związku z potrzebą wykorzystania danych w procesie sortowania, tak otrzymany łańcuch tekstowy należy rozbić na tablicę wartości. Operację tą przeprowadzono z użyciem funkcji `split()`, która rozbija ciąg wartości oddzielonych od siebie wskazanym w drugim parametrze znakiem (w tym przypadku jest to przecinek). Po rozbiciu otrzymujemy tablicę wartości tekstowych: "[3, '5', '2', '1', '7]" umieszczoną w zmiennej `rozbita_tablica`.

Na potrzeby procesu sortowania należy zamienić odczytane wartości z postaci tekstowej na postać liczbową – operacja ta została przeprowadzona w dwóch kolejnych liniach. Wykorzystano fakt, iż w języku Python tablice reprezentowane są za pomocą listy, stąd do początkowo pustej listy dołączane są w pętli zamienione na postać liczbą kolejne wartości. Ostatecznie otrzymywana jest tablica postaci: "[3, 5, 2, 1, 7]". Po odczytaniu wszystkich danych należy zamknąć otworzony wcześniej plik.

W dalszej części kodu pominięto proces sortowania – założono, że po jego zakończeniu posortowane dane będą w dalszym ciągu przechowywane w zmiennej `tab`. W celu zapisania posortowanych danych otworzony zostaje plik o nazwie "wynik.txt" (lub utworzony w przypadku, gdy plik taki nie istnieje) w trybie do zapisu.

W procesie zapisu tworzony jest plik wyjściowy, którego format danych będzie zgodny z plikiem wejściowym – należy jednak zwrócić uwagę na kilka elementów. Do zapisu wykorzystana została pętla `for`, która przechodzi przez kolejne elementy tablicy, by w linii 18 zamienić liczbę na postać tekstową, dołączając niezbędny separator (domyślnie przecinek). Problem tego rozwiązania polega na tym, że zapisując dane w tej postaci, również do ostatniej wartości dodany zostanie nadmiarowy separator, co nie jest operacją pożądaną (nie będzie to rozwiązanie kompatybilne z operacją odczytu). W związku z powyższym, w linii nr 15 następuje sprawdzenie, czy numer zapisywanej wartości jest ostatnią wartością w tablicy i jeżeli tak, to wartość separatora jest ustawiana na pusty znak, co pozwoli na zapis danych zgodnie z formatem wejściowym. Program wieńczy zamknięcie pliku wyjściowego.

## Zadanie

1. Napisać program, który wczyta zbiór liczb całkowitych z podanego przez użytkownika pliku (każda wartość znajduje się w nowej linii), a następnie posortuje i zapisze dane do pliku "wynik.txt". Kod odpowiedzialny za sortowanie zawrzeć w funkcji `posortuj_dane()`.

### 3. Wyszukiwanie binarne

Dotychczas, w celu wyszukania określonej wartości w tablicy należało przejrzeć jej wszystkie elementy od początku do końca. Wynikało to z faktu, iż tablica zawierała dane

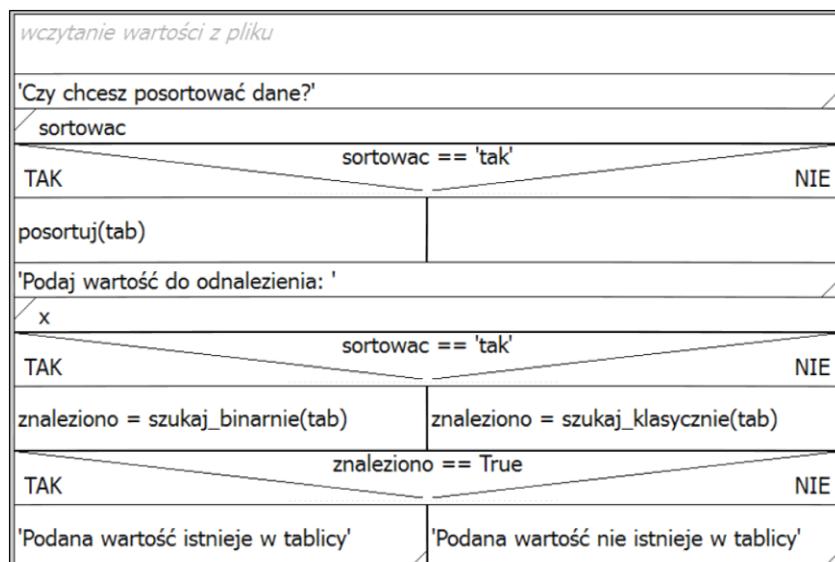
nieposortowane, stąd szukana wartość mogła się znajdować w jej dowolnym miejscu. Sytuacja zmienia się, gdy operacja wyszukiwania wartości przeprowadzana jest na tablicy wcześniej posortowanej – możliwe staje się użycie technik optymalizacyjnych, skracających czas poszukiwań wartości. W podrozdziale tym przedstawiona zostanie jedna z podstawowych technik wyszukiwania wartości w tablicy zawierającej posortowane dane liczbowe. Omówienie zasady działania przedstawione zostanie w oparciu o rozwiązanie przykładowego zadania.

### Przykład

Wczytać z pliku ciąg wartości oddzielonych przecinkami. Zapytać użytkownika, czy chce posortować wartości – zależnie od wyboru posortować wartości rosnąco lub pozostawić kolejność bez zmian. W ostatnim kroku zapytać użytkownika o liczbę do odnalezienia i sprawdzić, czy wartość ta istnieje w tablicy. Wyświetlić stosowny komunikat.

### Rozwiązanie

Po zapoznaniu się z poprzednim rozdziałem, wczytanie wartości z pliku nie powinno stanowić większego problemu. Po odpowiednim wczytaniu danych program pobiera od użytkownika informację o tym, czy dane należy wcześniej posortować – po uzyskaniu odpowiedzi twierdzącej można zastosować wybrany algorytm sortowania (np. sortowanie bąbelkowe). Poglądowy diagram NS aplikacji przedstawiono na rys. 10.4.



Rys. 10.4. Diagram NS aplikacji wyszukującej dane liczbowe

W przypadku, gdy dane są posortowane, wywoływana jest funkcja `szukaj_binarnie()`. W przypadku danych nieposortowanych program przechodzi do alternatywnej ścieżki z wywołaniem funkcji `szukaj_klasycznie()`.



Fundusze Europejskie  
Wiedza Edukacja Rozwój



Rzeczpospolita  
Polska



Unia Europejska  
Europejski Fundusz Społeczny

## Wyszukiwanie liniowe

W przypadku wyszukiwania klasycznego (zwanego wyszukiwaniem liniowym) należy przejść przez całą tablicę w celu wyszukania określonej wartości. Na rys. 10.5. przedstawiono przebieg poszukiwań przy wyszukiwaniu liniowym dla przykładowej, dziesięcioelementowej tablicy.

x = 7	8	3	3	2	9	7	5	2	4	1
8 == 7? szukaj	8	3	3	2	9	7	5	2	4	1
3 == 7? szukaj	8	3	3	2	9	7	5	2	4	1
3 == 7? szukaj	8	3	3	2	9	7	5	2	4	1
2 == 7? szukaj	8	3	3	2	9	7	5	2	4	1
9 == 7? szukaj	8	3	3	2	9	7	5	2	4	1
7 == 7? koniec	8	3	3	2	9	7	5	2	4	1

Rys. 10.5. Przebieg wyszukiwania liniowego dla przykładowej tablicy ( $x=7$ )

W pierwszym wierszu przedstawiono wartość poszukiwaną ( $x = 7$ ) oraz początkową zawartość tablicy, podczas gdy dalsze wiersze przedstawiają zasadę działania wyszukiwania liniowego. Kolorem zielonym oznaczono elementy tablicy, które należy jeszcze sprawdzić, kolorem pomarańczowym aktualnie sprawdzaną wartość – kolor czerwony zarezerwowano dla elementów, które już sprawdzono i nie ma w nich szukanej wartości.

Jak można zauważyć, w przypadku wyszukiwania liniowego należy sprawdzić wszystkie elementy tablicy. W najbardziej optymistycznym przypadku szukany element będzie na samym początku tablicy, podczas gdy w przypadku pesymistycznym będzie na samym końcu lub nie będzie go wcale.

## Wyszukiwanie binarne

W przypadku posiadania posortowanych danych można skorzystać z alternatywnej metody wyszukiwania – wyszukiwania binarnego.

### Kącik autora

W przypadku posiadania danych posortowanych można oczywiście w dalszym ciągu korzystać z metody wyszukiwania liniowego, jednak w większości przypadków (nieznane ułożenie danych wejściowych) nie będzie to metoda najszybsza.

Algorytm opiera się na wielokrotnym podziale zbioru wejściowego na mniejsze podzbiorы. Z aktualnie przetwarzanego zbioru wybierany jest element znajdujący się pośrodku (w przypadku zbioru parzystego jest to najbliższy element o indeksie całkowitym), z którym porównywana jest poszukiwana wartość. Zestawienie akcji w zależności od wyniku porównania przedstawia się następująco:



Fundusze Europejskie  
Wiedza Edukacja Rozwój



Rzeczypospolita  
Polska



- jeżeli porównywany jest ostatni element i nie jest to element szukany – szukany element nie występuje w tablicy,
- jeżeli porównywany element jest elementem szukanym – szukany element występuje w tablicy,
- jeżeli porównywany element jest większy od wartości szukanej – odrzuć wszystkie elementy po prawej stronie tablicy (indeks jest większy lub równy indeksowi elementu porównywanego) przeprowadź wyszukiwanie na nowym zbiorze,
- jeżeli porównywany element jest mniejszy od wartości szukanej – odrzuć wszystkie elementy po lewej stronie tablicy (indeks jest mniejszy lub równy indeksowi elementu porównywanego) i przeprowadź wyszukiwanie na nowym zbiorze.

Tak zaprogramowany algorytm pozwoli na skuteczne wyszukiwanie wartości w tablicy. Kontynuując przykład w oparciu o stan tablicy przedstawiony na rys. 10.5, poniżej przedstawiono zasadę działania algorytmu przy wyszukiwaniu wybranych wartości.

Na rys. 10.6. przedstawiono przebieg wyszukiwania wartości równej siedem.

$x = 7$	1	2	2	3	3	4	5	7	8	9
$3 < 7?$ szukaj po prawej	1	2	2	3	3	4	5	7	8	9
$7 == 7?$ koniec	1	2	2	3	3	4	5	7	8	9

Rys. 10.6. Przebieg wyszukiwania binarnego dla przykładowej tablicy ( $x=7$ )

Warto zauważyć, iż w przypadku wyszukiwania liniowego, do odnalezienia tej samej wartości potrzeba było wykonać sześć porównań, podczas gdy wyszukiwanie binarne ograniczyło ten proces do dwóch porównań. Kolorem czerwonym oznaczono elementy, które zostały odrzucone z poszukiwań.

Proces poszukiwań dla wartości równej jeden można sprawdzić na rys. 10.7.

$x = 1$	1	2	2	3	3	4	5	7	8	9
$3 > 1?$ szukaj po lewej	1	2	2	3	3	4	5	7	8	9
$2 > 1?$ szukaj po lewej	1	2	2	3	3	4	5	7	8	9
$1 == 1?$ koniec	1	2	2	3	3	4	5	7	8	9

Rys. 10.7. Przebieg wyszukiwania binarnego dla przykładowej tablicy ( $x=1$ )

Jako ostatni przedstawiony zostanie przykład dla zakres poszukiwań wartości, która nie znajduje się w tablicy (rys. 10.8). W przypadku wyszukiwania liniowego niezbędne byłoby

wykonania wszystkich dziesięciu porównań, podczas gdy zastosowanie wyszukiwania binarnego pozwala na redukcję do czterech instrukcji porównania.

x = 6	1	2	2	3	3	4	5	7	8	9
3 < 6? szukaj po prawej	1	2	2	3	3	4	5	7	8	9
7 < 6? szukaj po lewej	1	2	2	3	3	4	5	7	8	9
4 < 6? szukaj po prawej	1	2	2	3	3	4	5	7	8	9
5 == 6? koniec	1	2	2	3	3	4	5	7	8	9

Rys. 10.8. Przebieg wyszukiwania binarnego dla przykładowej tablicy ( $x=6$ )

## Zadanie

1. Zaprojektować diagram NS algorytmu wyszukiwania liniowego oraz binarnego, a następnie zaimplementować rozwiązanie powyższego zadania w języku Python.
2. Ile będzie potrzebnych maksymalnie porównań do odnalezienia określonej wartości w dwóch posortowanych zbiorach o rozmiarach: 100 oraz 1000 elementów, wykorzystując wyszukiwanie binarne?



#### 4. Wstęp do obsługi wyjątków

W ostatnim punkcie niniejszego rozdziału przedstawiony zostanie podstawowy mechanizm radzenia sobie z sytuacjami wyjątkowymi, które mogą wystąpić podczas działania aplikacji.

Klasycznym przykładem jest tutaj sytuacja, w której pobierana jest od użytkownika wartość tekstowa, która następnie zostaje zamieniana na wartość liczbową. Zawsze istnieje szansa, że użytkownik wprowadzi błędne dane (separatorem dziesiętnym jest przecinek, czy kropka?), które doprowadzą do nieprawidłowego funkcjonowania programu. Można oczywiście sprawdzić podaną przez użytkownika wartość tekstową i dopiero wtedy zamienić ją na wartość liczbową, jednak prowadzi to dodatkowego zaciemnienia kodu.

Rozwiążaniem sytuacji jest mechanizm obsługi wyjątków, który pozwala zareagować na sytuacje występujące w trakcie działania aplikacji stosunkowo rzadko (stąd nazwa wyjątek). Na rys. 10.9. przedstawiono klasyczne rozwiążanie problemu zamiany tekstu na liczbę.

```
1 import math
2
3 liczba_txt = input('Podaj liczbę całkowitą większą od zera: ')
4 if liczba_txt.isdecimal():
5     liczba = int(liczba_txt)
6
7     if liczba > 0:
8         wynik = math.sqrt(liczba)
9         print('Pierwiastek wynosi: '+str(wynik))
10    else:
11        print('Błąd! Podałeś nieprawidłową wartość!')
```

Rys. 10.9. Sprawdzenie, czy wartość tekstowa jest liczbą przed konwersją typu

W linii trzeciej pobierana jest od użytkownika wartość tekstowa. Do sprawdzenia, czy podana wartość jest w rzeczywistości liczbą wykorzystano funkcję `isdecimal()` wywoływaną na obiekcie typu tekstowego. Dopiero po upewnieniu się, czy podana wartość jest liczbą całkowitą, następuje dalsza konwersja do postaci liczbowej i wypisanie pierwiastka kwadratowego podanej liczby.

Alternatywną metodę przedstawiono na rys. 10.10.

```
1 import math
2
3 try:
4     liczba = int(input('Podaj liczbę całkowitą większą od zera: '))
5     if liczba > 0:
6         wynik = math.sqrt(liczba)
7         print('Pierwiastek wynosi: ' + str(wynik))
8     except:
9         print('Błąd! Podałeś nieprawidłową wartość!')
```



Rys. 10.10. Sprawdzenie, czy wartość tekstowa jest liczbą przed konwersją typu

Jak można zauważyć, wersja przedstawiona na rys. 10.10 nie dość, że jest krótsza od wersji niekorzystającej z obsługi wyjątków, to jest zdecydowanie bardziej czytelna. Podczas analizy kodu nie występuje dodatkowe rozgałęzienie kodu – niepotrzebna jest również znajomość dodatkowych funkcji operujących na zmiennych tekstowych. Ponadto, pierwsza wersja programu podwójnie sprawdza, czy podana przez użytkownika wartość jest liczbą – raz w pierwszej instrukcji warunkowej wewnętrz funkcji `isdecimal()` oraz przy samej konwersji typu. W drugim programie wartość jest sprawdzana tylko raz i gdy nie jest prawidłowa zgłaszan i obsługiwany jest wyjątek.

Na podobnej zasadzie obsługiwane są wyjątki związane z dostępem do plików na dysku komputera (brak uprawnień do utworzenia nowego pliku, próba zapisu do pliku otworzonego w trybie odczytu) oraz wyjątkowe sytuacje zgłoszone podczas operacji sieciowych (przekroczenie czasu połączenia, nieprawidłowy adres IP, nieoczekiwane zakończenie połączenia przez hosta). Bez systemu obsługi wyjątków, niezbędne byłoby przed wykonaniem zapisu do pliku każdorazowe sprawdzenie, czy plik został otwarty, czy ścieżka jest w dalszym ciągu dostępna (być może ktoś w międzyczasie usunął nośnik pamięci USB?), czy dostępne jest miejsce na dysku – sytuacje, których prawdopodobieństwo zdarzenia jest niezwykle małe, a których sprawdzenie doprowadziłoby do znaczącej utraty czytelności kodu.

**Kącik autora**

Wyobraź sobie obecność w kodzie 10 instrukcji warunkowych przed każdym wywołaniem funkcji `write()`, podczas chęci zapisu danych...



Fundusze Europejskie  
Wiedza Edukacja Rozwój



Rzeczpospolita  
Polska



Unia Europejska  
Europejski Fundusz Społeczny

## ROZDZIAŁ 11. TABLICE WIELOWYMIAROWE.

### Cel rozdziału:

Zaznajomienie z zasadą funkcjonowania tablic wielowymiarowych

### Zakres tematyczny:

1. Definicja tablicy wielowymiarowej
2. Przykłady przedstawiające wykorzystanie tablicy dwuwymiarowej

### Pytania kontrolne:

1. Jak są reprezentowane różne typy zmiennych w pamięci komputera?
2. Czym jest tablica jednowymiarowa i jak się z niej korzysta?
3. Jak wygląda budowa funkcji?

### 1. Definicja tablicy wielowymiarowej

W dotychczasowych zadaniach wykorzystywane były tablice jednowymiarowe, które pozwalały na przechowywanie sekwencji danych określonego typu. W większości przypadków tablice służyły do przechowywania sekwencji danych liczbowych – czasami spotkać można było użycie tablic do przechowywania tekstu (np. tablica wartości zwracana przez funkcję `split()`, która pojawiła się w rozdziale dotyczącym odczytu danych z pliku). Nic nie stoi jednak na przeszkodzie, aby elementom tablicy jednowymiarowej przypisać kolejne wartości typu tablicowego – tak skonstruowana struktura nosić będzie nazwę tablicy wielowymiarowej.

W klasycznym programowaniu stosunkowo najczęściej spotyka się wykorzystanie tablic: jednowymiarowych, dwuwymiarowych i w niektórych przypadkach tablic trójwymiarowych. Tablice o większej liczbie wymiarów są znacznie rzadziej wykorzystywane. Tablice dwuwymiarowe oraz trójwymiarowe są natomiast znacznie częściej wykorzystywane przy programowaniu gier.

#### Kącik autora

Czy jesteś w stanie powiedzieć, które elementy obecne w grach komputerowych opierają się na użyciu tablic dwuwymiarowych? Do czego mogą zostać wykorzystane tablice trójwymiarowe?

### 2. Przykłady przedstawiające wykorzystanie tablic dwuwymiarowych

W celu zobrazowania sposobu wykorzystania tablicy dwuwymiarowej, zaprezentowane zostanie rozwiążanie dwóch przykładowych zadań.

#### Przykład 1

Napisać program, który pozwoli na przechowywanie ocen z wybranych przedmiotów na studiach (liczbę ocen na przedmiot ograniczyć do 20). Po wybraniu odpowiedniej opcji program powinien umożliwić dodanie nowej oceny do przedmiotu oraz wyświetlenie średniej dotychczas wprowadzonych ocen. Przy wyjściu z programu zapisać zawartość tablicy do zewnętrznego pliku o wybranej nazwie.

#### Kącik autora

Na wstępnie należy zaznaczyć, iż praktycznie każdy przykład wykorzystujący tablice wielowymiarowe może być rozwiązany z użyciem odpowiednio dużej tablicy jednowymiarowej (po odpowiednich modyfikacjach).

Prawidłowe użycie tablicy wielowymiarowej pozwala jednak na uproszczenie zapisu, co jest niezwykle istotne podczas analizy kodu programu, wyszukiwaniu błędów oraz jego dalszej rozbudowy.

### Rozwiązanie

Zgodnie z założeniem, zadanie oparte zostanie o zastosowanie tablicy wielowymiarowej – w tym przypadku będzie to tablica dwuwymiarowa. Pomysł na rozwiązania zadania polega na początkowym utworzeniu tablicy jednowymiarowej, której kolejne elementy będą reprezentować zdefiniowane wcześniej przedmiotu. W ten sposób dostęp do każdego przedmiotu będzie mógł być zrealizowany poprzez podanie przez użytkownika jednej wartości liczbowej. Jako, że każdy przedmiot opisany jest szeregiem potencjalnych ocen, elementem przypisanym do przedmiotu o konkretnym indeksie będzie ponownie tablica jednowymiarowa przechowująca wartości liczbowe.

Rysunek poglądowy został przedstawiony na rys. 11.1.

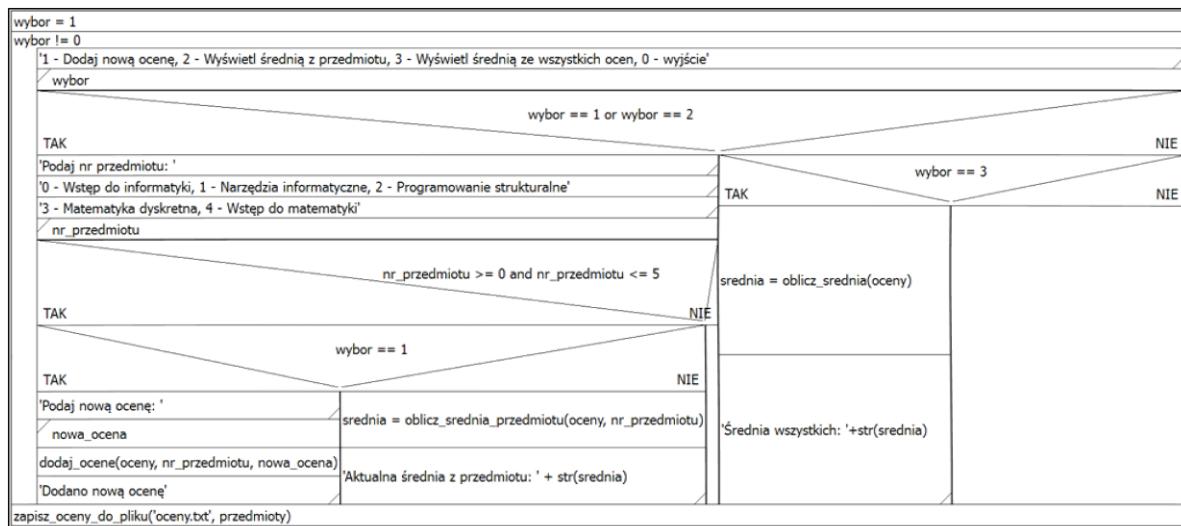
	j = 0	j = 1	j = 2	j = 3	j = 4
i = 0 Wstęp do informatyki	[0][0] 5	[0][1] 5	[0][2] 5		
i = 1 Narzędzia informatyczne	[1][0] 5	[1][1] 4	[1][2] 3		
i = 2 Programowanie strukturalne	[2][0] 4	[2][1] 4	[2][2] 5	[2][3] 5	
i = 3 Matematyka dyskretna	[3][0] 3	[3][1] 2	[3][2] 3		
i = 4 Wstęp do matematyki	[4][0] 3	[4][1] 3			

Rys. 11.1. Struktura tablicy dwuwymiarowej służącej do przechowywania ocen dla poszczególnych przedmiotów na studiach

Na rysunku przedstawiono zawartość tablicy po wprowadzeniu 15 ocen z przypisaniem ich do poszczególnych przedmiotów. Przy dostępie do tablicy dwuwymiarowej wykorzystywane są

dwa indeksy – pierwszy z nich określa pozycję w pierwszej tablicy (w przedstawionym przykładzie odpowiada on za numer wiersza, tj. numer przedmiotu), drugi określa pozycję w tablicy przypisanej do konkretnego wiersza (w przykładzie jest to indeks j odpowiadający konkretnej ocenie w obrębie przypisanej tablicy). Indeksy identyfikujące poszczególne elementy tablicy zostały umieszczone nad liczbą reprezentującą zawartość komórki.

Diagram NS przykładowego rozwiązania przedstawiono na rys. 11.2.



Rys. 11.2. Diagram NS programu przechowującego bazę ocen na studiach

Struktura diagramu nie powinna budzić większych wątpliwości. Po wejściu do głównej pętli programu użytkownik jest pytanego o wybór jednej z czterech akcji – dodania nowej oceny do bazy, wyświetlenia średniej z wybranego przedmiotu, wyświetlenia średniej ze wszystkich przedmiotów oraz wyjścia z programu.

Niezależnie od tego, czy użytkownik wybrał pierwszą, czy drugą opcję, należy pobrać od niego numer przedmiotu, którego będzie dotyczyła akcja. Przed wczytaniem wyboru wyświetlono, które indeksy odpowiadają poszczególnym przedmiotom. W przypadku chęci dodania nowej oceny należy dodatkowo wczytać wartość oceny.

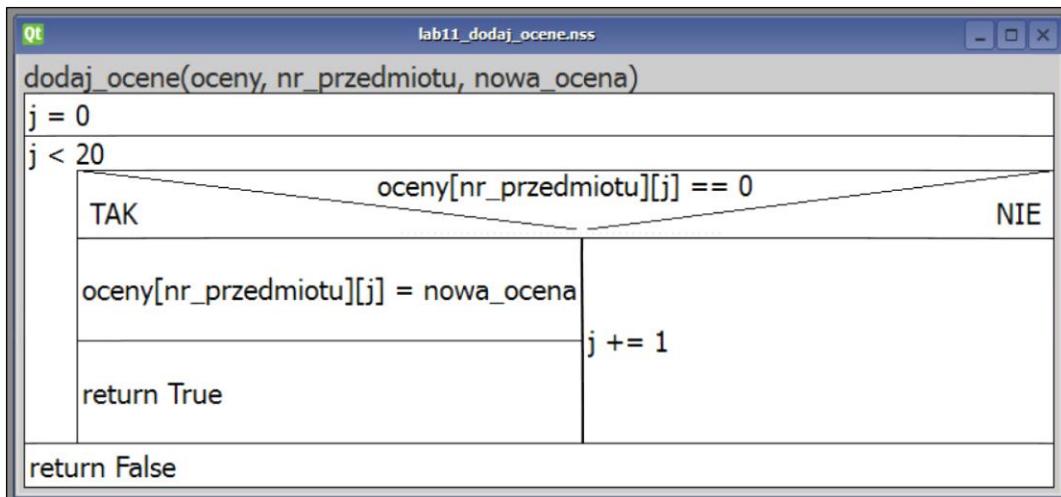
Wszystkie operacje na tablicy dwuwymiarowej przeprowadzane są w obrębie trzech oddzielnych funkcji o następujących nazwach:

- dodaj\_ocene(oceny, nr\_przedmiotu, nowa\_ocena),
  - oblicz\_srednia\_przedmiotu(oceny, nr\_przedmiotu),
  - oraz oblicz\_srednia(oceny).

Do każdej z tych funkcji przekazywana jest jako parametr tablica dwuwymiarowa, przechowująca aktualny stan ocen. W przypadku funkcji obliczającej średnią ze wszystkich ocen jest to jedyny parametr – pozostałe dwie funkcje do prawidłowej pracy potrzebują dodatkowego wskazania na przedmiot, którego dotyczyć będzie działanie (parametr

nr\_przedmiotu). Trzecim parametrem funkcji dodającej nową ocenę jest wartość oceny wprowadzonej przez użytkownika.

W tym momencie warto przyjrzeć się schematom odpowiadającym poszczególnym funkcjom. Funkcje zostaną przedstawione w kolejności występowania, stąd jako pierwsza zostanie zaprezentowana funkcja dodaj\_ocene() (rys. 11.3)



Rys. 11.3. Diagram NS funkcji dodaj\_ocene()

W funkcji dodaj\_ocene() po raz pierwszy pojawia się kod wykonujący operacje na tablicy wielowymiarowej. Idea działania funkcji opiera się na przejściu w pętli przez jeden cały wiersz tablicy dwuwymiarowej oceny i odnalezieniu miejsca, które nie jest zajęte przez inną ocenę – wolne miejsce oznacza element o wartości równej zero.

Pierwszym indeksem tablicy jest wartość przekazana jako nr\_przedmiotu – wartość ta nie jest zmieniana wewnętrz funkcji, co oznacza niezmieniony dostęp do jednego, wybranego wcześniej wiersza. Zmianie ulega natomiast drugi indeks tablicy, który wykorzystywany jest przejścia na kolejne elementy wiersza z ocenami. Funkcja kończy działanie (zwrócenie wartości True) w przypadku znalezienia pustego miejsca, pod które zostaje wpisana nowa wartość lub w momencie, gdy mimo przejścia przez wszystkie 20 elementów nie uda się odnaleźć wolnego miejsca w tablicy (zwrócenie wartości False).

Na konkretnym wierszu tablicy operuje również funkcja, obliczająca średnią z wybranego przedmiotu – oblicz\_srednia\_przedmiotu() (rys. 11.4)

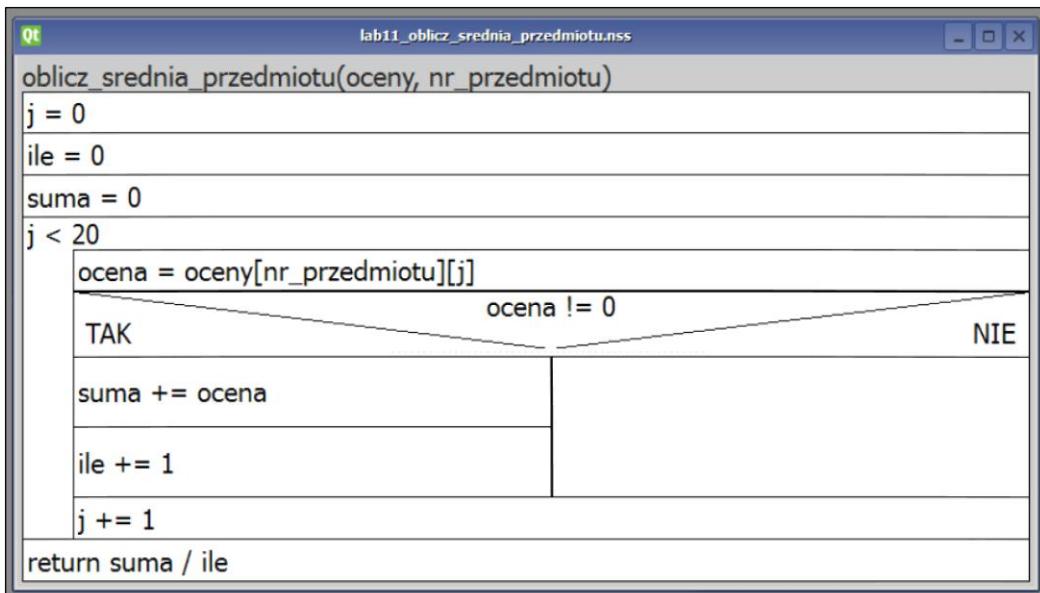


Fundusze Europejskie  
Wiedza Edukacja Rozwój



Rzeczpospolita  
Polska

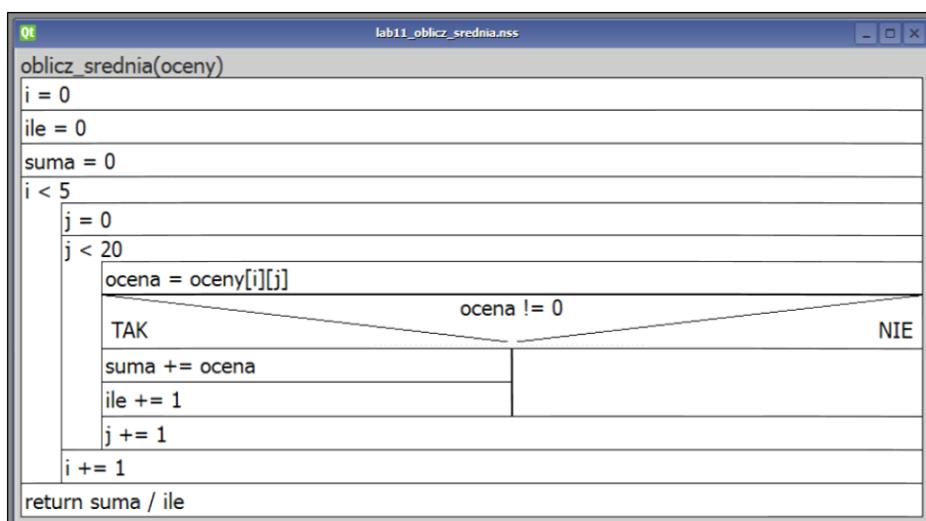




Rys. 11.4. Diagram NS funkcji *oblicz\_srednia\_przedmiotu()*

Struktura funkcji jest zbliżona do tej dodającej nową ocenę – główną część funkcji zajmuje pętla sterowana zmienną *j*, która jest wykorzystywana do zaindeksowania tablicy z ocenami. W każdym cyklu pętli pobierana jest kolejna wartość z konkretnego wiersza (określonego przez *nr\_przedmiotu*) tablicy ocen – w przypadku, gdy znajduje się tam jakaś ocena, zwiększana jest wartość zmiennej *suma* o wartość oceny oraz licznik znalezionych ocen (zmienna *ile*). Po przejściu przez wszystkie elementy obliczana i zwracana jest ich średnia.

Ostatnią funkcją jest funkcja obliczająca średnią wszystkich elementów w dwuwymiarowej tablicy ocen – w przeciwieństwie do dwóch pierwszych funkcji, w tym przypadku wykorzystana została konstrukcja pętli zagnieżdzonej, co pozwoliło na przeprowadzenie podwójnego indeksowania tablicy ocen w oparciu o dwie zmienne sterujące: *i* oraz *j*. (rys. 11.5)



Rys. 11.5. Diagram NS funkcji *oblicz\_srednia()*



**Fundusze Europejskie**  
Wiedza Edukacja Rozwój



**Rzeczypospolita  
Polska**

**Unia Europejska**  
Europejski Fundusz Społeczny



Jako, że w aplikacji dostępnych jest jedynie pięć przedmiotów, pierwsza pętla (sterowana zmienną `i`) wykonuje się zakresie od 0 do 4 (w języku Python do realizacji tego zadania można wykorzystać również funkcję `len()`, która zwraca długość tablicy przekazanej jako parametr – dzięki takiemu podejściu program byłby bardziej elastyczny). Wewnątrz pierwszej pętli osadzona jest druga pętla sterowana zmienną `j`.

Przy takim połączeniu, program przechodzi przez wszystkie elementy tablicy poczynając od elementu o indeksie `[0][0]`, a kończąc na indeksie `[4][19]`. W rezultacie program dokonuje sprawdzenia w pętli 100 wartości (pięć wierszy po dwadzieścia wartości), na podstawie których wyliczana i zwracana jest ostatecznie średnia – analogicznie do wyznaczania średniej w funkcji `oblicz_srednia_przedmiotu()`.

### Zadania do samodzielnej realizacji (do przykładu 1)

1. Dodać sprawdzenie, czy wprowadzona przez użytkownika ocena jest prawidłowa.
2. Dodać sprawdzenie, czy udało się dodać liczbę do tablicy. Wyświetlić stosowny komunikat.
3. Dodać nową akcję w postaci podglądu zawartości całej tablicy.
4. Dodać możliwość usunięcia oceny na wybranej pozycji.
5. Dodać nową akcję sprawdzającą, czy student zaliczył semestr (średnia z każdego przedmiotu musi być większa lub równa 3).
6. Zmodyfikować program tak, aby funkcja `oblicz_srednia()` wykorzystywała funkcję `oblicz_srednia_przedmiotu()`
7. Zaimplementować sekcję wczytywania danych z pliku przed wejściem do pętli głównej programu.

#### Kącik autora

Deklaracja tablicy wielowymiarowej w języku Python przebiega analogicznie do procesu tworzenia tablicy jednowymiarowej przechowującej dane liczbowe. Jedyna różnica polega na tym, że zamiast przypisywać do utworzonej tablicy wartość liczbową, przypisywana jest nowa zmienna typu tablicowego.

## Przykład 2

Napisać program, nawiązujący do popularnej gry „Wężyk”, w której gracz będzie się poruszać swoją postacią po dwuwymiarowej mapie. Zadaniem gracza jest zdobycie 10 jabłek. Na mapie znajduje się zawsze jedno jabłko (współrzędne pierwszego jabłka losowane są przed rozpoczęciem gry – współrzędne każdego kolejnego jabłka losowane są dopiero po zdobyciu poprzedniego jabłka).

Mapę ograniczyć do wymiarów 20x20 – przykładowy stan rozgrywki dla mapy o wymiarach 8x8 przedstawiono poniżej:

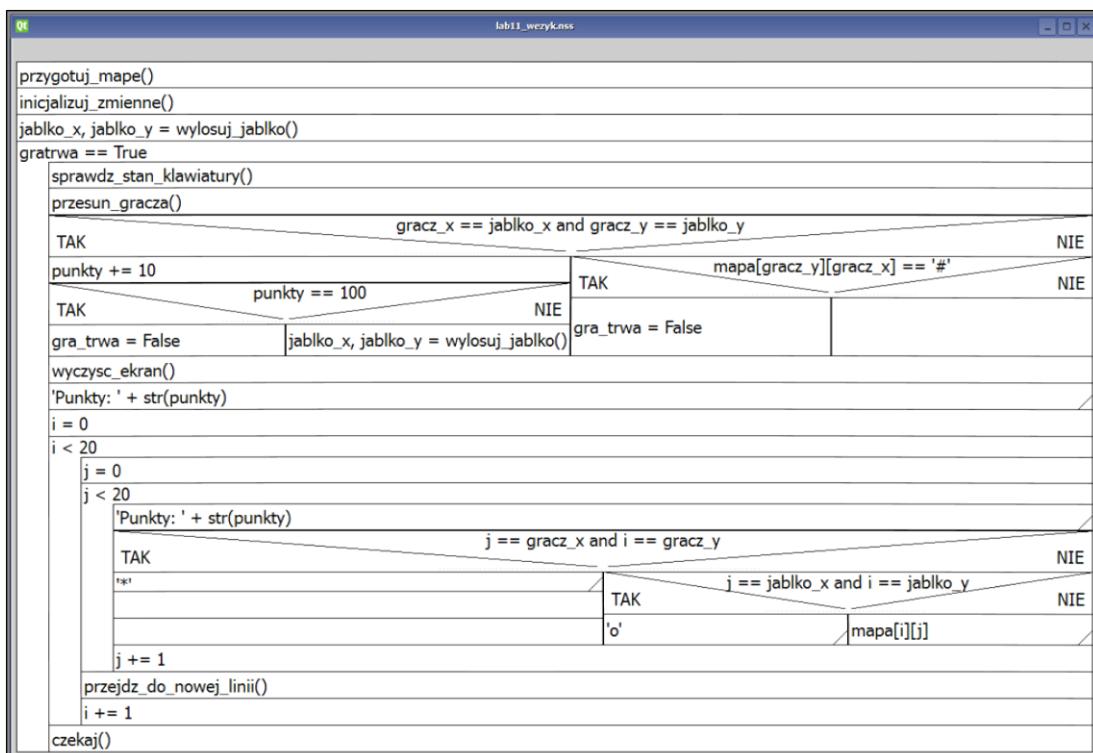
```
# # # # # # # #  
# * . . . . . #  
# . . . . . . . #  
# . . # # . . #  
# . . # # . . #  
# . . . o . . #  
# . . . . . . . #  
# # # # # # # #
```

gdzie:

- '#' – przeszkoda w postaci ściany – wejście gracza na przeszkodę kończy grę,
- '.' – teren, po którym można się poruszać,
- '\*' – postać gracza (jest wyświetlana, choć nie jest elementem mapy)
- 'o' – wylosowane jabłko (jest wyświetlane, choć nie jest elementem mapy)

## Rozwiązanie

W celu uproszczenia procesu omówienia zagadnienia, przy rozwiązyaniu zostanie przedstawiony zarówno schemat poglądowy, jak również pełny kod programu. Najprościej będzie rozpocząć proces tworzenia programu od przygotowania uproszczonego diagramu NS zawierającego jego najważniejsze elementy (rys. 11.6)



Rys. 11.6. Uproszczony diagram NS dla przykładu drugiego

Pierwszym problemem, z którym przyjdzie się zmierzyć podczas implementacji jest wybór reprezentacji mapy w pamięci komputera. Ze względu na swoją charakterystykę, mapę najwygodniej będzie przedstawić w postaci tablicy dwuwymiarowej. Aby uprościć proces wyświetlania, elementami przechowywanymi w tablicy będą symbole, które zostaną wyświetcone później na ekranie (jednoelementowe zmienne typu tekstowego). Pierwszą część programu uwzględniającą dołączenie bibliotek oraz inicjalizację tablicy przedstawiono na rys. 11.7.

Znaczenie poszczególnych bibliotek zostanie przedstawione w dalszej części programu, podczas wykorzystania niezbędnych funkcji.

Istotę pierwszego fragmentu kodu stanowi proces inicjalizacji tablicy dwuwymiarowej – w tym celu skorzystano z funkcji `split()` (przedstawionej w rozdziale poświęconym odczytce z pliku), która rozbija łańcuch tekstowy na mniejsze fragmenty. W tym konkretnym przypadku dokonywane jest rozbicie 20 zmiennych tekstowych reprezentujących kolejne wiersze tablicy. Poszczególne symbole reprezentowane są przez dwa znaki: "#" (przeszkoda) oraz '!' (wolne pole), które są od siebie oddzielone znakami spacji, na podstawie której dokonywane jest rozbicie. Alternatywnym podejściem jest inicjalizacja tablicy w postaci tablicy symboli bez wykorzystania funkcji `split()`, lecz prowadziłoby do zaciemnienia kodu, (przykładowo: `[ '#', '.', '.', '.', '.', '.', '.', '.', '.', '#' ]`).



Fundusze Europejskie  
Wiedza Edukacja Rozwój



Rzeczpospolita  
Polska



Unia Europejska  
Europejski Fundusz Społeczny

```
1 import os
2 import msvcrt
3 import random
4 import time
5
6 mapa = [
7     '# # # # # # # # # # # # # # # #'.split(' '),
8     '# . . . . . . . . . . . . #'.split(' '),
9     '# . . . . . . . . . . . . #'.split(' '),
10    '# . . . . . . . . . . . . #'.split(' '),
11    '# . . . . . . . . . . . . #'.split(' '),
12    '# . . . . . . . . . . . . #'.split(' '),
13    '# . . . . . . . . . . . . #'.split(' '),
14    '# . . . . . . . . . . . . #'.split(' '),
15    '# . . . . . . . . . . . . #'.split(' '),
16    '# . . . . . . . . . . . . #'.split(' '),
17    '# . . . . . . . . . . . . #'.split(' '),
18    '# . . . . . . . . . . . . #'.split(' '),
19    '# . . . . . . . . . . . . #'.split(' '),
20    '# . . . . . . . . . . . . #'.split(' '),
21    '# . . . . . . . . . . . . #'.split(' '),
22    '# . . . . . . . . . . . . #'.split(' '),
23    '# . . . . . . . . . . . . #'.split(' '),
24    '# . . . . . . . . . . . . #'.split(' '),
25    '# . . . . . . . . . . . . #'.split(' '),
26    '# # # # # # # # # # # # # # # #'.split(' ')
27 ]
```

Rys. 11.7. Import niezbędnych bibliotek oraz inicjalizacja dwuwymiarowej tablicy z użyciem funkcji `split()`

Tak zadeklarowaną tablicę będzie można z łatwością wyświetlić na ekranie i sprawdzić ewentualną kolizję gracza ze ścianą. Dalszą część kodu stanowi implementacja niewielkiej funkcji odpowiedzialnej za losowanie pozycji jabłka na mapie (rys. 11.8)

```
29 def wylosuj_jablko():
30     losuj = True
31
32     while losuj:
33         nx = random.randint(0, 19)
34         ny = random.randint(0, 19)
35
36         if mapa[ny][nx] == '.':
37             losuj = False
38
39     return nx, ny
40
```

Rys. 11.8. Implementacja funkcji losującej współrzędne nowego jabłka

Wewnątrz funkcji `wylosuj_jablko()` znajduje się pętla, w obrębie której losowane są współrzędne nowego jabłka – obie współrzędne losowane są w zakresie od 0 do 19. Po wylosowaniu współrzędnych program sprawdza, czy miejsce na mapie, na których



Fundusze  
Europejskie  
Wiedza Edukacja Rozwój



Rzeczypospolita  
Polska



postawione zostanie jabłko jest puste i dopiero wtedy opuszcza pętle. W przypadku wylosowania współrzędnych, na których znajduje się ściana, proces losowania zostałby powtórzony. Efektem działania funkcji jest zwrócenie dwóch nowych współrzędnych.

Znając sposób losowania współrzędnych można przejść do inicjalizacji wszystkich zmiennych niezbędnych do prawidłowego funkcjonowania aplikacji. Etap inicjalizacyjny przedstawiony został na rys. 11.9.

```
41     punkty = 0
42     gracz_x = 1
43     gracz_y = 1
44     jablko_x, jablko_y = wylosuj_jablko()
45     kierunek_x = 0
46     kierunek_y = 0
47     czas = 0.1
```

Rys. 11.9. Inicjalizacja zmiennych przed wejściem do pętli głównej

Początkowo punkty gracza są wyzerowane. Każdorazowe zjedzenie jabłka będzie zwiększać stan zmiennej o 10, co zgodnie z treścią zadania oznacza, że gra zakończy się w momencie osiągnięcia przez zmienną punkty wartości 100.

W związku z definicją mapy w postaci zamkniętego pomieszczenia (wszystkie brzegi mapy zostały zdefiniowane jako przeszkoda), pierwszym wolnym miejscem w którym można postawić gracza jest brzeg planszy o współrzędnych [1, 1], stąd wartości te są przypisane do zmiennych gracz\_x oraz gracz\_y.

W linii 44 wykorzystano zdefiniowaną wcześniej funkcję wylosuj\_jablko() do wylosowania dwóch współrzędnych i przypisania ich do zmiennych: jablko\_x oraz jablko\_y.

#### Kącik autora

W obecnej implementacji jabłko może zostać wylosowane bezpośrednio pod graczem! W sekcji zadań do samodzielnej realizacji będzie okazja do naprawienia tego problemu.

Nieco więcej uwagi będzie wymagało omówienie znaczenia dwóch kolejnych zmiennych, do których będzie okazja wrócić w momencie ich użycia. Na ten moment należy wiedzieć, że zmienne te określają w jakim kierunku będzie przesuwana postać gracza w kolejnych cyklach pętli głównej, w odpowiedzi na znak ostatnio naciśniętego klawisza.

Część inicjalizacyjną wieńczy zdefiniowanie czasu postoju pętli głównej przed wykonaniem następnego cyklu – linia czas = 0.1 oznacza określenie czasu oczekiwania równego 0.1 sekundy (100 milisekund). Funkcja opóźniająca wykonanie dalszej części programu zostanie przedstawiona w dalszej części implementacji.

Opierając się na diagramie przedstawionym na rys. 11.6, dopiero w tym momencie możliwe jest wejście do głównej pętli programu, w której już na początku konieczne będzie sprawdzenie stanu klawiatury oraz przesunięcie postaci gracza na mapie (rys. 11.10).

```
49 gratrwa = True
50 while gratrwa == True:
51
52     if msvcrt.kbhit() == True:
53         klawisz = msvcrt.getwch()
54
55         if klawisz == 'w':
56             kierunek_x = 0
57             kierunek_y = -1
58         if klawisz == 's':
59             kierunek_x = 0
60             kierunek_y = 1
61         if klawisz == 'a':
62             kierunek_x = -1
63             kierunek_y = 0
64         if klawisz == 'd':
65             kierunek_x = 1
66             kierunek_y = 0
67
68         gracz_x += kierunek_x
69         gracz_y += kierunek_y
```

Rys. 11.10. Wejście do pętli głównej, sprawdzenie wciśniętego klawisza oraz zmiana pozycji gracza

Do sterowania pętlą główną wykorzystano zmienną gratrwa. Pierwszą instrukcją w pętli jest wywołanie funkcji kbhit() (skrót od słów: „keyboard” oraz „hit”) z biblioteki msvcrt. W związku z chęcią przemieszczania gracza na mapie w każdym przebiegu pętli, nie jest możliwe wykorzystanie do tego celu funkcji input(), która blokuje dalsze wykonanie programu do momentu wprowadzenia przez użytkownika tekstu i zatwierdzenia go klawiszem enter. Do rozwiązania problemu wykorzystany został alternatywny mechanizm, który polega na wykorzystaniu kombinacji dwóch funkcji niższego poziomu: wspomnianej wcześniej kbhit() oraz getwch().

Funkcja kbhit() pozwala sprawdzić, czy w trakcie działania programu został wciśnięty przycisk. Wciśnięcie nie musi następować idealnie w momencie wywołania funkcji – kod każdego wciśniętego klawisza jest zapamiętywany do momentu, aż nie zostanie przez programistę obsłużony (stąd wystarczy sprawdzić stan klawiatury tylko na początku pętli).

Mając informację o fakcie wciśnięcia klawisza, można dokonać odczytu jego wartości – w tym celu wykorzystywana jest funkcja getwch() (z ang. „get wide character”), która zwraca kod pierwszego klawisza oczekującego w kolejce do obsługi. Odczytana wartość umieszczana jest w zmiennej klawisz, której wartość porównywana jest z zestawem klawiszy odpowiedzialnych za poruszanie się postaci, tj. WSAD. W zależności od tego, jaki



Fundusze Europejskie  
Wiedza Edukacja Rozwój



Rzeczpospolita  
Polska



Unia Europejska  
Europejski Fundusz Społeczny

jest to klawisz, modyfikowane są dwie zmienne: `kierunek_x` oraz `kierunek_y`. W przypadku klawiszy: 'a' oraz 'd' postać gracza powinna być przemieszczana w poziomie, stąd wyzerowanie wartości zmiennej `kierunek_y` oraz odpowiednia zmiana wartości zmiennej `kierunek_x`. Sytuacja wygląda odwrotnie w przypadku klawiszy: 'w' oraz 's', gdzie modyfikowane są wartości zmiennych tak, aby postać gracza przemieszczana była jedynie w pionie.

Niezależnie od tego, czy został naciśnięty jakiś klawisz, zaraz za instrukcją warunkową następuje zmiana współrzędnych gracza w oparciu o zmienne kierunkowy. Należy zwrócić uwagę na fakt, iż obie zmienne kierunkowe będą wyzerowane tylko raz, bezpośrednio po uruchomieniu aplikacji – naciśnięcie dowolnego klawisza z zestawu WSAD sprawi, że postać gracza nie będzie się już mogła więcej zatrzymać (poza przypadkiem zakończenia rozgrywki w momencie wygranej lub uderzenia w ścianę).

Po przemieszczeniu gracza należy sprawdzić, w jakiej znalazł się pozycji. W przypadku wejścia na puste pole niepotrzebna jest żadna akcja. Dodatkowej obsługi wymaga jednak sprawdzenie, czy postać gracza zjadła jabłko lub uderzyła w ścianę. Kod sprawdzający te warunki został przedstawiony na rys. 11.11.

```
71     if gracz_x == jablko_x and gracz_y == jablko_y:
72         punkty += 10
73         czas = czas / 1.2
74
75         if punkty >= 100:
76             gratrwa = False
77         else:
78             jablko_x, jablko_y = wylosuj_jablko()
79         elif mapa[gracz_y][gracz_x] == '#':
80             gratrwa = False
```

Rys. 11.11. Kod odpowiedzialny za sprawdzenie pozycji gracza

Sprawdzenie, czy gracz znalazł się na pozycji jabłka dokonywane jest poprzez złożoną instrukcję warunkową, w której zestawiona została ze sobą para współrzędnych przypisanych graczu oraz jabłku – jeżeli obie współrzędne się zgadzają, graczowi udało się zjeść owoc. Zjedzenie jabłka zwiększa liczbę punktów o 10 oraz minimalnie zmniejsza czas przestoiu pomiędzy kolejnymi cyklami pętli (przyspieszenie rozgrywki).

Jeżeli po zjedzeniu jabłka gracz posiada 100 lub więcej punktów następuje zakończenie gry. Mniejsza liczba punktów oznacza potrzebę wylosowania kolejnego jabłka (gracz musi zjeść dziesięć jabłek, aby zakończyć grę).

W przypadku, gdy gracz nie znajduje się na współrzędnej jabłka, sprawdzana jest ewentualna kolizja ze ścianą, która jest elementem dwuwymiarowej tablicy. Do indeksacji mapy wykorzystane zostały współrzędne gracza – w ten sposób można uzyskać informację na temat elementu znajdującego się bezpośrednio pod graczem. W przypadku uderzenia w ścianę, gra jestkończona.



Fundusze Europejskie  
Wiedza Edukacja Rozwój



Rzeczpospolita  
Polska



Unia Europejska  
Europejski Fundusz Społeczny

Ostatnim elementem kodu wymagającym omówienia jest proces wyświetlenia gry na ekranie. W tym celu wykorzystane zostały pętle zagnieżdżone, które w każdym przebiegu pętli głównej rysują w konsoli aktualny stan rozgrywki (rys. 11.12).

```
82     os.system('cls')
83     print('Punkty:' + str(punkty))
84
85     i = 0
86     while i < 20:
87         j = 0
88         while j < 20:
89             if j == gracz_x and i == gracz_y:
90                 print('*', end='')
91             elif j == jablko_x and i == jablko_y:
92                 print('o', end='')
93             else:
94                 print(mapa[i][j], end='')
95             j += 1
96
97         print()
98         i += 1
99
100    time.sleep(czas)
```

Rys. 11.12. Wyczyszczenie ekranu, wyświetlenie liczby punktów oraz mapy

W systemach z rodziny Windows, do wyczyszczenia zawartości konsoli może zostać użyta zewnętrzna instrukcja „cls” (w innych systemach może to być np. instrukcja „clear”). Aplikacja będzie działała prawidłowo również bez tej instrukcji, jednak korzystanie z niej będzie znacznie trudniejsze (co można bardzo łatwo sprawdzić usuwając instrukcję odpowiedzialną za czyszczenie ekranu). Instrukcja „cls” została wykonana poprzez użycie funkcji system zawartej w bibliotece os (angielski skrót oznaczający system operacyjny), stąd obecny w pierwszych liniach import.

Po wyświetleniu aktualnej liczby punktów, następuje przejście do dwóch pętli odpowiedzialnych za wyświetlenie postaci gracza, jabłka oraz wyglądu mapy. Przechodząc przez wszystkie elementy mapy następuje sprawdzenie, czy aktualnie badany element nie jest przypadkiem postacią gracza – jeżeli tak, następuje wyświetlenie symbolu gwiazdki. W przypadku dojścia do współrzędnych odpowiadających jabłku – wyświetlana jest litera 'o'. Jeżeli punkt na mapie nie jest ani graczem, ani jabłkiem, to musi być to element mapy – zadane to realizowane jest w oparciu o stan elementu znajdującego się pod współrzędnymi opisanyimi przez dwie zmienne sterujące: i oraz j (pierwsza z nich odpowiada za wiersz, druga za kolumnę).

Na uwagę zasługuje również lekko zmodyfikowana forma funkcji print(), która przy wyświetlanie posiada dodatkowy argument – określa on znak, który zostanie wstawiony po wyświetlonym tekście. Domyślnie jest to znak nowego wiersza, co oznacza, że standardowe użycie funkcji prowadzi do przejścia do nowej linii – takie zachowanie uniemożliwiłoby



Fundusze  
Europejskie  
Wiedza Edukacja Rozwój



Rzeczpospolita  
Polska



Unia Europejska  
Europejski Fundusz Społeczny

prawidłowe wyświetlenie stanu mapy, stąd znak nowego wiersza został zastąpiony znakiem pustym. Po wyświetleniu całej wiersza w pętli sterowanej zmienną `j`, wywoływana jest pusta funkcja `print()` w celu przejścia do kolejnego wiersza.

Po wyświetleniu całej mapy wywoływana jest funkcja `sleep()` z biblioteki `time`, która opóźnia przejście do następnego cyklu pętli, co pozwala dać graczowi czas na odpowiednią reakcję. Przedstawienie funkcji `sleep()` kończy omówienie implementacji aplikacji w języku Python. Aplikacja w praktyczny sposób wykorzystuje tablicę dwuwymiarową do realizacji postawionego zadania.

**Kącik autora**

Implementację gry (w lekko uproszczonym wydaniu) udało się zamknąć nie przekraczając 100 linii kodu!

**Uwagi końcowe**

Uruchomienie aplikacji bezpośrednio w środowisku PyCharm może nie przynieść oczekiwanych efektów – aby móc skorzystać z następujących instrukcji:

`kbhit(), getwch() oraz os.system('cls')`

aplikację po napisaniu należy uruchomić bezpośrednio z konsoli systemowej!

W systemie Windows należy przejść do „menu start” i uruchomić wiersz poleceń komendą „cmd”. W konsoli należy przejść do folderu z projektem używając polecenia „cd”, np.

„cd c:\projekty\python\wezyk”

Po przejściu do folderu z projektem można uruchomić skrypt wpisując:

„python main.py”



**Fundusze Europejskie**  
Wiedza Edukacja Rozwój



**Rzeczpospolita Polska**



### Zadania do samodzielnej realizacji (do przykładu 2)

1. Zmodyfikować początkową tablicę – dodać więcej przeszkód, utrudniających przejście gry przez gracza.
2. Dodać możliwość przejścia z jednego końca mapy na drugi w momencie, gdy w miejscach tych nie będzie ściany.
3. Dodać do gry obsługę liczby życia.
4. Uniemożliwić wylosowania jabłka bezpośrednio pod graczem
5. Uniemożliwić wylosowanie jabłka w pobliżu gracza (np. powinno być oddalone przynajmniej o pięć pól)
6. Zastanowić się nad możliwością zmiany mapy po zjedzeniu określonej liczby jabłek.
7. Dodać możliwość ładowania mapy z pliku tekstowego.
8. Zaimplementować ogon węża – po każdym zjedzeniu jabłka wąż będzie się stawał coraz dłuższy. W jakiej postaci najlepiej przechowywać ciało węża?
9. Dodać ruchomych przeciwników
10. Zaimplementować dodatkową funkcjonalność wedle własnego uznania

### Zadania klasyczne do samodzielnej realizacji (tablice wielowymiarowe)

1. Pobrać od użytkownika dwa wymiary tablicy ( $n$  oraz  $m$ ), a następnie utworzyć w ich oparciu tablicę dwuwymiarową. Przy użyciu dwóch pętli wczytać od użytkownika  $n*m$  wartości, umieszczając je w utworzonej wcześniej tablicy. Znaleźć i wyświetlić numer wiersza, dla którego suma elementów jest największa.
2. Utworzyć tablicę o wymiarach  $5x5$  elementów. Zapełnić tablicę dodatnimi liczbami całkowitymi pobranymi od użytkownika, a następnie:
  - a. Sprawdzić, czy każdy wiersz w tablicy posiada unikalne wartości.
  - b. Sprawdzić, czy tablica jest symetryczna względem głównej przekątnej.
  - c. Sprawdzić, czy suma elementów brzegowych jest większa od elementów niebrzegowych.
  - d. Sprawdzić, czy w tablicy występują dwie takie same wartości.



Fundusze Europejskie  
Wiedza Edukacja Rozwój



Rzeczpospolita  
Polska



Unia Europejska  
Europejski Fundusz Społeczny

## **Zadania tekstowe do samodzielnej realizacji (tablice wielowymiarowe)**

### **1. Kółko i krzyżyk**

W grze bierze udział dwóch uczestników. Gracze są naprzemiennie odpytywani o podanie dwóch współrzędnych, określających umiejscowienie symbolu (kółka, bądź krzyżka). Program każdorazowo sprawdza, czy postawienie danego symbolu jest możliwe. Gra toczy się do momentu spełnienia kryterium wygranej przez jednego z graczy (ustawienia trzech symboli w linii), bądź uzyskania remisu.

#### Implementacja

- w programie do reprezentacji planszy należy użyć tablicy dwuwymiarowej 3x3,
- utworzyć następujące funkcje:
  - `postaw_symbol(plansza, x, y, symbol)`  
Funkcja ustawiająca na mapie zadany symbol.
  - `czy_obszar_zajety(plansza, wsp_x, wsp_y)`  
Sprawdzenie, czy na danym obszarze planszy można postawić element.
  - `sprawdz_czy_gracz_wygral(plansza, nr_gracza)`  
Funkcja sprawdzająca, czy wskazany gracz wygrał rozgrywkę (True w przypadku wygranej, False w pozostałych przypadkach).
  - `wyswietl_plansze()`  
Funkcja, która wyświetla aktualny stan planszy.



**LABORATORIUM 4. KONSTRUKCJA AUTOMATÓW SKOŃCZONYCH DLA  
WYBRANYCH ZADAŃ TEKSTOWYCH (WYRAŻENIA  
REGULARNE).**

**Cel laboratorium:**

Przedstawienie zasady działania wyrażeń regularnych.

**Zakres tematyczny zajęć:**

1. Mechanizm funkcjonowania wyrażeń regularnych
  - a. Wstęp do wyrażeń regularnych
  - b. Praktyczne wykorzystanie wyrażeń regularnych w rozwiązywaniu wybranych problemów

**Pytania kontrolne:**

1. Jak wygląda składania instrukcji warunkowej? Jakie możemy wyróżnić operatory logiczne i jakie jest ich działanie?
2. Czym jest tablica? Jak wygląda jej deklaracja oraz użycie?
3. Jak jest reprezentowana zmienna typu tekstowego w pamięci komputera?

## ROZDZIAŁ 12. MECHANIZM FUNKCJONOWANIA WYRAŻEŃ REGULARNYCH.

### Cel rozdziału:

Celem rozdziału jest zapoznanie czytelnika z zasadą działania wyrażeń regularnych oraz użyciem ich do rozwiązywania wybranych problemów programistycznych.

### Zakres tematyczny:

1. Wykorzystanie wyrażeń regularnych w rozwiązywaniu wybranych problemów

### Pytania kontrolne:

1. Jak wygląda składania instrukcji warunkowej? Jakie możemy wyróżnić operatory logiczne i jakie jest ich działanie?
2. Czym jest tablica? Jak wygląda jej deklaracja oraz użycie?
3. Jak jest reprezentowana zmienna typu tekstowego w pamięci komputera?

### 1. Wykorzystanie wyrażeń regularnych w rozwiązywaniu wybranych problemów

Podczas pracy ze zbiorem danych może zajść potrzeba wyszukania w zbiorze określonego wzorca. W przypadku danych liczbowych, jako wzorzec rozumiany jest najczęściej podciąg spełniający określone kryteria matematyczne (np. sprawdzenie, czy elementy tablicy ułożone są w kolejności niemalejącej).

W przypadku pracy z danymi tekstowymi, znacznie częściej zachodzi potrzeba sprawdzenia, czy przetwarzana wartość tekstowa pasuje do odpowiedniego wzorca, tj. czy kolejne znaki występujące w tekście spełniają określone we wzorcu kryteria. W celu zilustrowania problemu przytoczony zostanie prosty przykład.

### Przykład 1

Napisać program, który poprosi użytkownika o podanie kodu pocztowego i który następnie sprawdzi jego poprawność. O wyniku sprawdzenia poinformować stosownym komunikatem.

### Rozwiążanie bez wykorzystania wyrażeń regularnych

Ze względu na stosunkowo prostą budowę programu, omówienie programu przedstawione będzie bezpośrednio w postaci implementacyjnej. Czytając treść zadania wydawać się może, że jego rozwiązanie jest stosunkowo proste – wystarczy rozbić podany przez użytkownika kod pocztowy na dwie części i sprawdzić, czy tak otrzymane wartości znajdują się w określonych przedziałach liczbowych. Przychodząca na myśl pierwsza implementacja mogłaby wyglądać tak, jak kod programu na rys. 12.1.

```
1  kod = input('Podaj kod pocztowy do weryfikacji:')

2

3  try:
4      rozbity = kod.split('-')
5      lewa_strona = int(rozbity[0])
6      prawa_strona = int(rozbity[1])
7
8      if lewa_strona >= 0 and lewa_strona <= 99 and prawa_strona >= 0 and prawa_strona <= 999:
9          print('Kod prawidłowy.')
10     else:
11         print('Kod nieprawidłowy.')
12
13 except:
14     print('Kod nieprawidłowy.)
```

Rys. 12.1. Próba rozwiązywania problemu weryfikacji kodu pocztowego bez użycia wyrażeń regularnych (pierwsze podejście)

Na początku pobierany jest od użytkownika kod pocztowy w postaci tekstowej. Pobrany kod jest następnie rozbijany na tablicę dwóch elementów, ale tylko wtedy, gdy w ciągu występuje symbol myślnika. W momencie, gdy będzie to inny znak specjalny zgłoszony zostanie wyjątek przy pierwszej próbie zamiany wartości tekstowej na liczbę (przykładowo, ciąg "20\*430" nie jest liczbą, stąd nie będzie można pomyślnie rzutować tej wartości). W przypadku, gdy myślnik zostanie pominięty całkowicie, funkcja `split()` zwróci jedynie jedną wartość, co doprowadzi do zgłoszenia wyjątku w drugim rzutowaniu (w związku z odwołaniem się do nieistniejącego, drugiego elementu tablicy). Do tego momentu kod wydaje się realizować swoją funkcję prawidłowo.

Po zamianie na dwie wartości liczbowe sprawdzane jest, czy lewa strona kodu zawiera się w przedziale od 0 do 99 oraz czy prawa strona jest w przedziale od 0 do 999.

Po uruchomieniu, przedstawiony program rzeczywiście wyświetli poprawny komunikat dla istniejących kodów pocztowych. Gdzie jest w takim razie błąd? Istnieją przypadki, dla których powyższy kod nie będzie działał prawidłowo – wystarczy po uruchomieniu podać wartość "5-10", która również zostanie zinterpretowana jako prawidłowy kod pocztowy.

Może się wydawać, że to również nie jest problemem, gdyż wystarczy sprawdzić długość lewej i prawej strony (lewa powinna mieć długość dwóch znaków, podczas gdy prawa równą trzem znakom). Rozwiązanie to przedstawiono na rys. 12.2.



```
1  kod = input('Podaj kod pocztowy do weryfikacji: ')
2
3  try:
4      rozbito = kod.split('-')
5      lewa_strona = int(rozbito[0])
6      prawa_strona = int(rozbito[1])
7
8      if len(rozbito[0]) == 2 and len(rozbito[1]) == 3 and \
9          lewa_strona >= 0 and lewa_strona <= 99 and prawa_strona >= 0 and prawa_strona <= 999:
10         print('Kod prawidłowy.')
11     else:
12         print('Kod nieprawidłowy.')
13
14 except:
15     print('Kod nieprawidłowy.'
```

Rys. 12.2. Próba rozwiązywania problemu weryfikacji kodu pocztowego bez użycia wyrażeń regularnych (drugie podejście)

Po modyfikacji kodu, podanie krótszych (np. "5-10") lub dłuższych ciągów liczbowych (np. "0000-300") będzie skutkowało wyświetleniem informacji i błędnym kodzie pocztowym. Dopiero w tej wersji program spełnia swoje zadanie.

#### Kącik autora

Dodatkowym problemem przedstawionego rozwiązania jest fakt opierania działania programu na przechwytywaniu i reagowaniu na wyjątki – proces ten nie należy do dobrych technik programistycznych i nie powinien być wykorzystywany w praktyce!

### Rozwiązywanie z wykorzystaniem wyrażeń regularnych

Dla porównania, na rys. 12.3 umieszczono kod programu wykonujący to samo zadanie przy użyciu wyrażeń regularnych.

```
1  import re
2
3  kod = input('Podaj kod pocztowy do weryfikacji: ')
4
5  wynik = re.fullmatch('[0-9][0-9]-[0-9][0-9][0-9]', kod)
6  if wynik != None:
7      print('Kod prawidłowy.')
8  else:
9      print('Kod nieprawidłowy.'
```

Rys. 12.3. Rozwiązywanie problemu weryfikacji kodu pocztowego przy pomocy wyrażeń regularnych (dłuższa wersja)



Fundusze  
Europejskie  
Wiedza Edukacja Rozwój



Rzeczpospolita  
Polska

Unia Europejska  
Europejski Fundusz Społeczny



Ponadto, zawartość ciągu definiującego wyrażanie regularne można dodatkowo uprościć, skracając do postaci widocznej na rys. 12.4.

```
1 import re
2
3 kod = input('Podaj kod pocztowy do weryfikacji: ')
4
5 wynik = re.fullmatch('[0-9]{2}-[0-9]{3}', kod)
6 if wynik != None:
7     print('Kod prawidłowy.')
8 else:
9     print('Kod nieprawidłowy.'
```

Rys. 12.4. Rozwiązywanie problemu weryfikacji kodu pocztowego przy pomocy wyrażeń regularnych (krótsza wersja)

Jak można zauważyć, kodu jest zdecydowanie mniej niż w rozwiązaniu zaproponowanym w implementacji niewykorzystującej wyrażeń regularnych – mniej kodu oznacza zazwyczaj mniejszą szansę na popełnienie błędu. Z kodu wyeliminowana została dodatkowo sekcja obsługi wyjątków, a sam warunek sprawdzający poprawność kodu pocztowego przyjął postać prostej instrukcji warunkowej sprawdzającej, czy wzorzec o zadanej strukturze znajduje się w tekście podanym przez użytkownika.

#### Kącik autora

Wyobraź sobie, ile instrukcji warunkowych byłoby potrzebnych do sprawdzenia bardziej skomplikowanych wzorców! Na szczęście, praktycznie w każdym rozwijanym obecnie języku programowania istnieje możliwość skorzystania z wbudowanych mechanizmów obsługujących wyrażenia regularne.

Pobranie kodu pocztowego następuje we wszystkich programach w taki sam sposób. Mając do dyspozycji kod, program przechodzi linii nr 5, która stanowi główną część aplikacji.

Do zmiennej wynik przypisywany jest rezultat działania funkcji `fullmatch()` dostępnej z poziomu biblioteki `re`. Zadaniem funkcji jest sprawdzenie, czy podany jako pierwszy parametr wzorzec występuje w tekście przekazanym jako drugi parametr. Jeżeli tekst nie pasuje do wzorca, funkcja zwraca wartość `None`, stąd w następnej linii znajduje się instrukcja warunkowa sprawdzająca, czy została zwrocona wartość różna od `None` (w przypadku odnalezienia wzorca zwracany jest obiekt `Match`, jednak korzystanie z niego nie będzie szerzej omawiane). Zasady budowania wzorców zostaną przedstawione poniżej.

Jak można zauważyć, ten sam wzorzec może być zapisany na różne sposoby (choć w tym przypadku, to ten drugi sposób (rys. 12.4) jest zazwyczaj preferowany). Tak zapisany wzorzec pozwala na znaczne uproszczenie procesu weryfikacji, wymagając od programisty znacznie mniejszego wysiłku, minimalizując jednocześnie ryzyko popełnienia błędu.

## Składowe wyrażenia regularnego

Wyrażenia regularne dopuszczają użycie zarówno standardowych symboli (takich jak litery i cyfry), jak również całego zestawu znaków specjalnych. Poniżej przedstawione zostanie znaczeniu podstawowego zestawu znaków (pełne omówienie dostępne jest w ogólnodostępnej dokumentacji języka Python).

Podstawowe zasady konstruowania wyrażeń regularnych:

1. Każdy znak niebędący znakiem specjalnym opisuje we wzorcu dokładnie siebie samego, na przykład wzorzec: "informatyka" będzie odpowiadał za wyszukanie w tekście dokładnie słowa "informatyka" (istotna jest również wielkość liter – słowo "Informatyka" jest tu zupełnie innym wzorcem). Punkt ten dotyczy również cyfr.
2. Znak kropki "." oznacza dowolny znak.
3. W celu wyszukania jednego ze znaków specjalnych (np. wspomnianej wyżej kropki) należy skorzystać z symbolu odwrotnego ukośnika (stąd do wyszukania trzech kropek w tekście potrzebne będzie wyrażanie: "\.\.\.", do wyszukania dwóch odwrotnych ukośników: "\\\\").
4. Ciąg znaków między nawiasami kwadratowymi oznacza jeden dowolny znak, np. aby wyszukać w tekście słowa: "rak", "mak" lub "hak" należy użyć wzorca postaci: "[ rmh ] ak" (z uwzględnieniem dużej litery na początku: "[ rRmMhH ] ak").
  - a. W nawiasach kwadratowych można opisać zbiór znak przy pomocy symbolu myślnika, np. do budowy wzorca na potrzeby weryfikacji kodu pocztowego wykorzystano wzorzec postaci: "[0-9][0-9]-[0-9][0-9][0-9]", w którym każdy nawias oznacza możliwość wystąpienia dowolnej cyfry od 0 do 9. W ten sam sposób można ustalać zakresy dla liter alfabetu np. "[A-Z][a-z][a-z][a-z]" oznacza czteroznakowy wzorzec zbudowany z liter alfabetu (cyfry nie zostały uwzględnione) rozpoczynający się dużą literą.
  - b. W nawiasach większość znaków specjalnych traci swoje znaczenie, stąd szukając na przykład pięcioliterowego słowa zakończonego kropką można użyć wzorca w postaci: "[a-z]{5} [.]". Przy wyszukiwaniu znaków specjalnych tak jak nawiasy, zawsze warto sprawdzić oficjalną dokumentację w celu uniknięcia pomyłek.
  - c. Użycie symbolu "^" na początku wzorca oznacza wszystkie znaki poza tymi zdefiniowanymi w nawiasach (odpowiednik negacji) – wzorzec: "[^rh] ak" będzie oznaczać wszystkie ciągi kończące się na "ak", które nie będą się rozpoczynały od liter "r" oraz "h" (np. słowo "tak" lub "mak", ale nie "rak" i "ptak").



Fundusze Europejskie  
Wiedza Edukacja Rozwój



Rzeczpospolita  
Polska



- d. Wykluczenie może być również połączone z definicją znaku z określonego przedziału, na przykład: "[^0-4]" oznacza wszystkie cyfry spoza przedziału 0-4.
5. Symbol gwiazdki '\*' oznacza wystąpienie symbolu wielokrotne, bądź niewystąpienie go wcale. Symbol ten dołączany jest za znakiem (może być dołączony również po nawiasie zamykającym). Przykładowo: "[a-z]{8}[0-9]\*" określa ciąg ośmiu liter zakończony opcjonalną sekwencją cyfr (np. "komputer" oraz "komputer2000")
6. Symbol plusa "+" działa analogicznie do symbolu gwiazdki z tą różnicą, że określony znak musi wystąpić chociaż raz.
7. Symbol znaku zapytania "?" oznacza wystąpienie znaku nie więcej niż jeden raz (tj. raz albo wcale) – analogicznie do symbolu gwiazdki.
8. Symbol "|" oznacza wystąpienie jednego z dwóch ciągów znaków, np. "hak|mak" będzie wzorcem wyszukującym te dwa słowa.
9. Nawiasy klamrowe określają wystąpienie znaku określona liczbę razy – rozwiążanie to wykorzystano do weryfikacji kodu pocztowego: "[0-9]{2}-[0-9]{3}". W nawiasach można zdefiniować również przedział powtórzeń od pierwszej wartości do drugiej np. "[0-9]{2}-[0-9]{3,5}" oznaczałoby dopuszczenie kodów pocztowych, w których druga część mogłaby mieć również długość równą 4 i 5.
10. Nawiasy okrągłe wykorzystywane są do grupowania określonego podzbioru wyrażenia – pogrupowanie pozwala na zbudowanie podwyrażenia wyszukującego określonego ciągu znaków, np. wyrażanie "(www.)?[a-z]+(.pl|.com)" będzie oznaczało wyszukanie adresu strony internetowej, która może, ale nie musi się zaczynać od sekwencji "www.", później następuje ciąg liter (przynajmniej jedna litera), by ostatecznie sprawdzić, czy ciąg zakończony jest rozszerzeniem ".pl" lub ".com".

W zależności od implementacji dostępne są również inne znaki specjalne, stąd najlepiej potwierdzić znaczenie poszczególnych elementów w oficjalnej dokumentacji.



W standardowej implementacji wyrażeń regularnych języka Python dostępne są również symbole specjalne oznaczające predefiniowane wzorce. Wśród nich można wyróżnić:

- "\w" – symbol alfanumeryczny [A-Za-z0-9\_],
- "\W" – symbol, który nie jest alfanumeryczny [^A-Za-z0-9\_],
- "\s" – symbol spacji,
- "\S" – symbol wszystkiego, tylko nie spacji,
- "\d" – symbol cyfry [0-9],
- "\D" – symbol znaku, który nie jest cyfrą [^0-9].

Posiadając wiedzę na temat budowy wyrażeń regularnych, analiza programu z rys. 12.3 oraz rys. 12.4 nie powinna przysporzyć większych problemów.

#### Kącik autora

Czy zastanawiałeś się, w jaki sposób sprawdzana jest składnia programu napisanego w języku Python lub dowolnie innym języku programowania? Na jakiej zasadzie kompilator/interpreter (pośrednio środowisko programistyczne) jest w stanie wykryć błędy składniowe?

### Zadania

1. Zaprojektować wyrażenie regularne sprawdzające, czy dane osobowe (imię, opcjonalne drugie imię oraz nazwisko) podane przez użytkownika są poprawne. Dane pobierane są do jednej zmiennej tekstowej.
2. Zaprojektować wyrażanie regularne sprawdzające poprawność wprowadzonego adresu e-mail.
3. Zaprojektować wyrażenie regularne sprawdzające, czy podana w postaci tekstu ciąg jest prawidłowo zapisaną:
  - a. liczbą całkowitą,
  - b. liczbą rzeczywistą.
4. Zaprojektować wyrażanie regularne sprawdzające poprawność wprowadzonego przez użytkownika adresu zamieszkania w postaci: nazwy ulicy (miejscowości) oraz numeru domu (i/lub mieszkania)
5. Zaprojektować wyrażanie regularne sprawdzające, czy podany przez użytkownika numer telefonu jest poprawnym numerem telefonu nadawanym w Polsce.



Fundusze Europejskie  
Wiedza Edukacja Rozwój



Rzeczpospolita  
Polska



Unia Europejska  
Europejski Fundusz Społeczny

## **LABORATORIUM 5. MASZYNA TURINGA.**

### **Cel laboratorium:**

Przedstawienie zasady funkcjonowania maszyny Turinga.

### **Zakres tematyczny zajęć:**

1. Realizacja maszyny Turinga
  - a. Charakterystyka maszyny Turinga.
  - b. Realizacja maszyny Turinga na przykładach

### **Pytania kontrolne:**

1. Jak wygląda reprezentacja tablicy w pamięci komputera?



**Fundusze  
Europejskie**  
Wiedza Edukacja Rozwój



**Rzeczpospolita  
Polska**

**Unia Europejska**  
Europejski Fundusz Społeczny



## ROZDZIAŁ 13. REALIZACJA MASZYNY TURINGA.

### Cel rozdziału:

Celem rozdziału jest zapoznanie czytelnika z zasadą funkcjonowania maszyny Turinga.

### Zakres tematyczny:

1. Charakterystyka maszyny Turinga
2. Realizacja maszyny Turinga na przykładach

### Pytania kontrolne:

1. Jak wygląda reprezentacja tablicy w pamięci komputera?

### 1. Charakterystyka maszyny Turinga

Jako maszynę Turinga rozumiemy abstrakcyjną maszynę służącą do wykonywania określonych wcześniej algorytmów. U podstaw, definicja maszyny opiera się model matematyczny określający, jakie operacje może maszyna wykonać oraz w jakich stanach się może znaleźć. Idea maszyny zakłada przetwarzanie danych w oparciu o nieskończenie długą taśmę (stąd implementacja maszyny jest zawsze modelem uproszczonym). W obrębie taśmy maszyna może odczytać wartość komórki, zmienić jej wartość, przesunąć głowicę maszyny o jedną komórkę w lewo lub o jedną komórkę w prawo. Maszyna wykonuje swoje zadanie na podstawie stanu, w którym aktualnie się znajduje oraz wartości odczytanej komórki (formalnie, zależności te zapisywane są w postaci tzw. krotki).

### 2. Realizacja maszyny Turinga na przykładach

Po przejściu przez skrócony proces wprowadzenia teoretycznego, możliwe jest przejście do przykładów wykorzystujących właściwości maszyny Turinga. W celu zwiększenia czytelności, długość taśmy wykorzystywanej na potrzeby przykładów będzie wyraźnie ograniczona.

### Przykład 1

Zaprojektować maszynę Turinga, która będzie wykonywała operację zwiększenia odczytanej na taśmie liczby w systemie dwójkowym o jeden.

### Rozwiążanie

W celu zrealizowania powyższego zadania należy rozpisać algorytm zwiększenia liczby w systemie dwójkowym w postaci serii instrukcji maszyny Turinga. Każda z instrukcji określona jest przez pięć uporządkowanych wartości.

Pierwszym zadaniem jest określenie, ile stanów maszyny jest potrzebnych do rozwiązania postawionego zadania. W operacji zwiększenia wartości liczby o jeden niezbędne jest zdefiniowanie przynajmniej dwóch stanów:

- pierwszego w momencie, gdy wartość została dodana do komórki i nie nastąpiło przeniesienie wartości do kolejnej komórki (zamiana stanu komórki z 0 na 1),
- drugiego w momencie, gdy wartość została dodana do komórki i nastąpiło przeniesienie wartości dalej (zmiana stanu komórki z 1 na 0 i zapamiętanie informacji o przeniesieniu).

W związku z chęcią zwiększenia wartości liczby o 1, maszyny Turinga realizująca powyższe zadanie będzie rozpoczęła swoją pracę w tym drugim stanie, określonym jako Q2 (maszyna rozpoczyna pracę w stanie przeniesienia wartości). Pierwszy stan przyjmie wartość Q1. Podsumowaniem opracowania algorytmu działania jest utworzenie piątek uporządkowanych opisujących pracę maszyny według poniższego schematu:

1.  $(0, Q1, 0, Q1, L)$  – jeżeli maszyna jest w stanie Q1 (bez przeniesienia), a odczytana wartość jest równa zero, to pozostań w tym stanie bez zmieniany wartości komórki i przejdź do komórki po lewej stronie.
2.  $(1, Q1, 1, Q1, L)$  – jeżeli maszyna jest w stanie Q1 (bez przeniesienia), a odczytana wartość jest równa jeden, to pozostań w tym stanie bez zmieniany wartości komórki i przejdź do komórki po lewej stronie.
3.  $(0, Q2, 1, Q1, L)$  – jeżeli maszyna jest w stanie Q2 (przeniesienie), a odczytana wartość jest równa zero, to zmień stan na Q1 (bez przeniesienia), zmień wartość komórki na jeden i przejdź do komórki po lewej stronie.
4.  $(1, Q2, 0, Q2, L)$  – jeżeli maszyna jest w stanie Q2 (przeniesienie), a odczytana wartość jest równa jeden, to pozostań w stanie Q2, zmień wartość komórki na zero i przejdź do komórki po lewej stronie.

Po zdefiniowaniu odpowiedzi maszyny wobec napotkanych wartości, można przetestować działanie maszyny na konkretnym przykładzie.

W pierwszym teście, maszynie podano taśmę postaci: "#01011". Jako, że po skrajnie lewej stronie znajduje się najmniej znaczący bit, maszyna otrzymała liczbę 11 w systemie dziesiętnym – dodanie wartości jeden powinno skutkować otrzymaniem liczby 12. Wizualizacja trzech pierwszych odczytów maszyny przedstawiono na rys. 13.1.

#### Kącik autora

Zazwyczaj do opisu poszczególnych stanów używane są małe litery alfabetu (np. q, p), jednak ze względu na czytelność autor zdecydował się na skorzystanie z wielkich liter (Q oraz P).

W literaturze duża litera Q określa zazwyczaj zbiór wszystkich stanów q.

<b>pozycja głowicy</b>						<b>V</b>
<b>taśma</b>	#	0	1	0	1	1
<b>interpretacja</b>	Maszyna rozpoczyna pracę w stanie Q2. W stanie Q2 odczytanie 1 skutkuje wpisaniem 0, pozostaniem w stanie Q2 i przesunięciem głowicy w lewo					
<b>aktualny stan maszyny</b>	Q2					
<b>pozycja głowicy</b>						<b>V</b>
<b>taśma</b>	#	0	1	0	1	0
<b>interpretacja</b>	W stanie Q2 odczytanie 1 skutkuje wpisaniem 0, pozostaniem w stanie Q2 i przesunięciem głowicy w lewo					
<b>aktualny stan maszyny</b>	Q2					
<b>pozycja głowicy</b>				<b>V</b>		
<b>taśma</b>	#	0	1	0	0	0
<b>interpretacja</b>	W stanie Q2 odczytanie 0 skutkuje wpisaniem 1, przejściem do stanu Q1 i przesunięciem głowicy w lewo					
<b>aktualny stan maszyny</b>	Q2					

Rys. 13.1. Trzy pierwsze ruchy głowicy maszyny Turinga (zwiększenie liczby o jeden)

Większość opisu wyjaśniającego zasadę działania maszyny podczas pracy znalazła się w sekcji „interpretacja”. Zgodnie z założeniami maszyna rozpoczyna pracę w stanie Q2, a pozycja głowicy wskazuje na skrajnie prawy element. W każdym cyklu dokonywane jest sprawdzenie stanu maszyny oraz wartości, którą odczytano i to właśnie na ich podstawie dokonywane są dalsze zmiany na taśmie.

Pojawienie się na początku taśmy dwóch jedynek oznacza pozostanie maszyny w trybie przenoszenia wartości – dopiero napotkanie wartości równej zero pozwala na zmianę stanu z Q2 na Q1. W stanie Q1 maszyna nie wykonuje dalszych modyfikacji stanu taśmy, stąd głowica podąża sukcesywnie do ostatniego elementu. Sytuację tą pokazano na rys. 13.2, który ukazuje pozostałe trzy kroki maszyny.



pozycja głowicy			V			
taśma	#	0	1	1	0	0
interpretacja	W stanie Q1 odczytanie 1 skutkuje wpisaniem 1, pozostaniem w stanie Q1 i przesunięciem głowicy w lewo					
aktualny stan maszyny	Q1					
pozycja głowicy		V				
taśma	#	0	1	1	0	0
interpretacja	W stanie Q1 odczytanie 0 skutkuje wpisaniem 1, pozostaniem w stanie Q1 i przesunięciem głowicy w lewo					
aktualny stan maszyny	Q1					
pozycja głowicy	V					
taśma	#	0	1	1	0	0
interpretacja	Maszyna napotyka koniec taśmy i kończy swoje działanie					
aktualny stan maszyny	Q1					

Rys. 13.2. Pozostałe trzy ruchy głowicy maszyny Turinga (zwiększenie liczby o jeden)

Zgodnie z założeniem, po wejściu do stanu Q1 głowica maszyny dociera do końca taśmy. Po wykonaniu operacji należy dokonać weryfikacji otrzymanego rezultatu. Po zakończeniu pracy maszyny, taśma przyjęła wartość: "#01100", co odpowiada wartości 12 w systemie dziesiętnym, co oznacza, że maszyna prawidłowo zrealizowała swoje zadanie.

## Przykład 2

Zaprojektować maszynę Turinga, która będzie wykonywała operację dodawania dwóch liczb dwójkowych zapisanych na taśmie. Na każdą liczbę przewidziano 3 bity, jednak obie liczby ograniczone są do wartości z przedziału od 0 do 3 (binarnie od "000" do "011"). Wynik dodawania umieścić w miejscu drugiej liczby.

## Rozwiązanie

Poziom skomplikowania zadania drugiego znaczaco przewyższa zadanie pierwsze ze względu na mnogość stanów maszyny Turinga. Wczytując się w treść zadania można zauważyc, że z uwzględnieniem znaku końca, taśma przekazana na wejście maszyny będzie



Fundusze Europejskie  
Wiedza Edukacja Rozwój



Rzeczypospolita  
Polska



miała długość nie mniejszą niż siedem. W zaprezentowaniu rozmieszczenia danych na taśmie pomocny będzie rys. 13.3.

pozycja głowicy								v
taśma	#	0	1	0	0	1	1	
interpretacja								
aktualny stan maszyny	Q1							

Rys. 13.3. Rozmieszczenie danych na taśmie dla przykładu drugiego

Kolorem zielonym zaznaczono pierwszą liczbę ("011" oznacza liczbę 3), kolor pomarańczowy reprezentuje pozycję drugiej liczby ("010" – liczba 2). Koniec taśmy oznaczono kolorem czerwonym. Treść zadania mówi o wykonaniu operacji dodawania tak, aby suma znalazła się na miejscu drugiej liczby. Oznacza to, że napotkaniu wartości równej jeden przy pierwszej liczbie, głowica maszyny Turinga musi przejść trzy pozycje dalej tak, aby wskazywała odpowiadający bit liczby drugiej. Po dotarciu do właściwego miejsca następuje dodanie wartości – jeżeli było tam wcześniej zero następuje przypisanie wartości jeden i głowica musi znowu powrócić do kolejnego bitu pierwszej liczby. W przypadku odczytu wartości jeden przy dodawaniu wartości do drugiej liczby należy przeprowadzić operację dodawania analogiczną do tej z przykładu pierwszego – w najgorszym przypadku głowica maszyny dojdzie do najstarszego bitu drugiej liczby, gdzie po aktualizacji wartości będzie musiała powrócić do liczby pierwszej, aby kontynuować operację.

W rezultacie definiowany jest szereg stanów maszyny – stany rozpoczynające się od litery Q oznaczają stany, w których głowica przesuwa się w lewo. Stany, w których głowica powraca do pozycji wyjściowej przyjmują prefiks P. Kompletna lista stanów przedstawia się następująco:

1.  $(0, Q1, 0, Q1, L)$  – odczytanie 0 nie ma wpływu na dodawanie,
2.  $(1, Q1, 1, Q2, L)$  – odczytanie 1 rozpoczętu proces dodawania,
3.  $(0, Q2, 0, Q3, L)$  – przejście w lewo,
4.  $(1, Q2, 1, Q3, L)$  – przejście w lewo,
5.  $(0, Q3, 0, Q4, L)$  – przejście w lewo,
6.  $(1, Q3, 1, Q4, L)$  – przejście w lewo,
7.  $(0, Q4, 1, P1, P)$  – dodanie wartości bez przeniesienia i przejście w prawo,
8.  $(1, Q4, 0, Q5, L)$  – dodanie wartości z przeniesieniem i przejście w lewo,
9.  $(0, Q5, 1, P2, P)$  – dodanie wartości bez przeniesienia i przejście w prawo,
10.  $(1, Q5, 0, Q6, L)$  – dodanie wartości z przeniesieniem i przejście w lewo,
11.  $(0, Q6, 1, P3, P)$  – dodanie wartości bez przeniesienia i przejście w prawo,
12.  $(0, P3, 0, P2, P)$  – powrót,
13.  $(1, P3, 1, P2, P)$  – powrót,
14.  $(0, P2, 0, P1, P)$  – powrót,
15.  $(1, P2, 1, P1, P)$  – powrót,
16.  $(0, P1, 0, Q1, P)$  – powrót,
17.  $(1, P1, 1, Q1, P)$  – powrót.



## Zintegrowany Program Rozwoju Politechniki Lubelskiej – część druga

Poniżej zaprezentowano efekty działania zaprojektowanej maszyny – ze względu na obszerność, przebieg został rozbity na mniejsze części (rys. 13.4 – rys.13.9)

pozycja głowicy							v
taśma	#	0	1	0	0	1	1
interpretacja	Maszyna rozpoczyna pracę w stanie Q1. W stanie Q1 wartość komórki się nie zmienia, odczytanie 1 prowadzi do zmiany stanu na Q2 i przesunięcia głowicy w lewo						
aktualny stan maszyny	Q1						
pozycja głowicy							v
taśma	#	0	1	0	0	1	1
interpretacja	W stanie Q2 wartość komórki się nie zmienia, następuje zmiana stanu na Q3 i przesunięcie głowicy w lewo						
aktualny stan maszyny	Q2						
pozycja głowicy							v
taśma	#	0	1	0	0	1	1
interpretacja	W stanie Q3 wartość komórki się nie zmienia, następuje zmiana stanu na Q4 i przesunięcie głowicy w lewo						
aktualny stan maszyny	Q3						

Rys. 13.4. Maszyna Turinga realizująca operację dodawania (cz. I)

pozycja głowicy							v
taśma	#	0	1	0	0	1	1
interpretacja	Maszyna rozpoczyna pracę w stanie Q1. W stanie Q1 wartość komórki się nie zmienia, odczytanie 1 prowadzi do zmiany stanu na Q2 i przesunięcia głowicy w lewo						
aktualny stan maszyny	Q1						
pozycja głowicy							v
taśma	#	0	1	0	0	1	1
interpretacja	W stanie Q2 wartość komórki się nie zmienia, następuje zmiana stanu na Q3 i przesunięcie głowicy w lewo						
aktualny stan maszyny	Q2						
pozycja głowicy							v
taśma	#	0	1	0	0	1	1
interpretacja	W stanie Q3 wartość komórki się nie zmienia, następuje zmiana stanu na Q4 i przesunięcie głowicy w lewo						
aktualny stan maszyny	Q3						

Rys. 13.5. Maszyna Turinga realizująca operację dodawania (cz. II)



Fundusze Europejskie  
Wiedza Edukacja Rozwój



Rzeczypospolita  
Polska

Unia Europejska  
Europejski Fundusz Społeczny



## Zintegrowany Program Rozwoju Politechniki Lubelskiej – część druga

pozycja głowicy					V		
taśma	#	0	1	1	0	1	1
interpretacja	W stanie Q2 wartość komórki się nie zmienia, następuje zmiana stanu na Q3 i przesunięcie głowicy w lewo						
aktualny stan maszyny	Q2						
pozycja głowicy				V			
taśma	#	0	1	1	0	1	1
interpretacja	W stanie Q3 wartość komórki się nie zmienia, następuje zmiana stanu na Q4 i przesunięcie głowicy w lewo						
aktualny stan maszyny	Q3						
pozycja głowicy			V				
taśma	#	0	1	1	0	1	1
interpretacja	W stanie Q4 odczytanie 1 skutkuje wpisaniem 0, następuje zmiana stanu na Q5 i przesunięcie głowicy w lewo						
aktualny stan maszyny	Q4						

Rys. 13.6. Maszyna Turinga realizująca operację dodawania (cz. III)

Pierwszy powrót głowicy można zaobserwować na rys. 13.7 – po dojściu głowicy do ostatniego bitu następuje zmiana wartości z 0 na 1 i wyjście ze stanów realizujących funkcję dodawania (stany Q). Głowica powoli cofa się do punktu, w którym rozpoczął się proces dodawania.

pozycja głowicy		V					
taśma	#	0	0	1	0	1	1
interpretacja	W stanie Q5 odczytanie 0 skutkuje wpisaniem 1, następuje zmiana stanu na P2 i przesunięcie głowicy w prawo						
aktualny stan maszyny	Q5						
pozycja głowicy			V				
taśma	#	1	0	1	0	1	1
interpretacja	W stanie P2 wartość komórki się nie zmienia, następuje zmiana stanu na P1 i przesunięcie głowicy w prawo						
aktualny stan maszyny	P2						
pozycja głowicy				V			
taśma	#	1	0	1	0	1	1
interpretacja	W stanie P1 wartość komórki się nie zmienia, następuje zmiana stanu na Q1 i przesunięcie głowicy w prawo						
aktualny stan maszyny	P1						

Rys. 13.7. Maszyna Turinga realizująca operację dodawania (cz. IV)



Fundusze Europejskie  
Wiedza Edukacja Rozwój



Rzeczypospolita  
Polska

Unia Europejska  
Europejski Fundusz Społeczny



## Zintegrowany Program Rozwoju Politechniki Lubelskiej – część druga

Ostatecznie głowica dochodzi do trzeciego bitu pierwszej liczby, który zgodnie z założeniem zadania powinien być zawsze wyzerowany. W kolejnym kroku następuje przejście do kolejnej, której zawartość nie wpływa na zmianę wartości elementów na taśmie (nawet w przypadku odczytania wartości równej jeden, głowica zostanie przesunięta o trzy pozycje w lewo, co zakończy program (rys. 13.8 oraz rys. 13.9)).

pozycja głowicy					V		
taśma	#	1	0	1	0	1	1
interpretacja	W stanie Q1 wartość komórki się nie zmienia, odczytanie 0 nie zmienia stanu, przesunięcie głowicy w lewo						
aktualny stan maszyny	Q1						
pozycja głowicy				V			
taśma	#	1	0	1	0	1	1
interpretacja	W stanie Q1 wartość komórki się nie zmienia, odczytanie 1 skutkuje zmianą zmianą stanu na Q2 i przesunięciem głowicy w lewo						
aktualny stan maszyny	Q1						
pozycja głowicy			V				
taśma	#	1	0	1	0	1	1
interpretacja	W stanie Q2 wartość komórki się nie zmienia, następuje zmiana stanu na Q3 i przesunięcie głowicy w lewo						
aktualny stan maszyny	Q2						

Rys. 13.8. Maszyna Turinga realizująca operację dodawania (cz. V)

Moment dotarcia do pierwszego bitu drugiej liczby powoduje przejście do stanu zmierzającego ku końcowi taśmy. Rezultatem działania programu jest wartość oznaczona kolorem pomarańczowym, tj. "101", co oznacza liczbę 5 ( $3("011") + 2("010") = 5("101")$ ).

pozycja głowicy		V					
taśma	#	1	0	1	0	1	1
interpretacja	W stanie Q3 wartość komórki się nie zmienia, następuje zmiana stanu na Q4 i przesunięcie głowicy w lewo						
aktualny stan maszyny	Q3						
pozycja głowicy	V						
taśma	#	1	0	1	0	1	1
interpretacja	Maszyna napotkała znak końca						
aktualny stan maszyny	Q4						

Rys. 13.9. Maszyna Turinga realizująca operację dodawania (cz. VI)

Otrzymanie poprawnego wyniku kończy omówienie drugiego przykładu.

### Zadania



Fundusze Europejskie  
Wiedza Edukacja Rozwój



Rzeczypospolita  
Polska

Unia Europejska  
Europejski Fundusz Społeczny



1. Zaprojektować oraz zaimplementować maszynę Turinga wykonującą negację wartości wprowadzonych na taśmie.
2. Zaprojektować oraz zaimplementować maszynę Turinga mnożącą liczbę przez dwa.
3. Zaimplementować maszynę Turinga z zadania drugiego, a następnie sprawdzić poprawność jej działania dla wszystkich kombinacji liczb wejściowych.
4. Zaprojektować oraz zaimplementować maszynę Turinga mnożącą liczbę przez trzy.
5. Zaprojektować oraz zaimplementować maszynę Turinga pozwalającą na odjęcie jednej liczby od drugiej (analogicznie do treści drugiego przykładu).



**Fundusze Europejskie**  
Wiedza Edukacja Rozwój



**Rzeczpospolita Polska**

**Unia Europejska**  
Europejski Fundusz Społeczny



## **LABORATORIUM 6. TEORIA ZŁOŻONOŚCI OBliczeniowej.**

### **Cel laboratorium:**

Przedstawienie różnic w czasie działania wybranych algorytmów w zależności od ustawienia danych wejściowych

### **Zakres tematyczny zajęć:**

1. Złożoność obliczeniowa w praktyce
  - a. Wstęp do złożoności obliczeniowej
  - b. Przykład przedstawiający różnicę w czasie wykonania algorytmu z zależnością od ustawienia danych wejściowych

### **Pytania kontrolne:**

1. Jak się definiuje funkcje? Jak wygląda ich nagłówek?
2. Na jakiej zasadzie działa sortowanie bąbelkowe?
3. W jaki sposób indeksuje się tablice jednowymiarowe?



**Fundusze Europejskie**  
Wiedza Edukacja Rozwój



**Rzeczpospolita Polska**



**Unia Europejska**  
Europejski Fundusz Społeczny

## **ROZDZIAŁ 14. ZŁOŻONOŚĆ OBLICZENIOWA W PRAKTYCE.**

### **Cel rozdziału:**

Celem rozdziału jest przedstawienie różnic w czasie procesu sortowania w zależności od sposobu ustawienia danych wejściowych.

### **Zakres tematyczny:**

1. Wstęp do złożoności obliczeniowej
2. Przykład przedstawiający różnicę w czasie wykonania algorytmu z zależności od ustawienia danych wejściowych

### **Pytania kontrolne:**

1. Jak się definiuje funkcje? Jak wygląda ich nagłówek?
2. Na jakiej zasadzie działa sortowanie bąbelkowe?
3. W jaki sposób indeksuje się tablice jednowymiarowe?

### **1. Wstęp do złożoności obliczeniowej**

Zagadnienie złożoności obliczeniowej odnosi się do technik określających ilości zasobów potrzebnych do rozwiązania zadanych problemów obliczeniowych. W świecie algorytmów najbardziej powszechna jest analiza algorytmów w dwóch kategoriach: złożoności czasowej oraz złożoności pamięciowej. Ta pierwsza odnosi się do analizy czasu potrzebnego na rozwiązanie danego problemu, druga natomiast odnosi się do rozmiaru pamięci, która jest wykorzystywana w trakcie obliczeń.

W tym rozdziale przedstawiona zostanie technika pomiaru oraz analizy czasu wykonania sortowania bąbelkowego.

### **2. Przykład przedstawiający różnicę w czasie wykonania algorytmu z zależności od ustawienia danych wejściowych**

Na potrzeby rozdziału zmierzony zostanie rzeczywisty czas potrzebny na wykonanie sortowania bąbelkowego. W procesie tym wykorzystano komputer o stałych parametrach sprzętowych i systemowych. Według ogólnie przyjętego modelu obliczeniowego, pesymistyczna złożoność czasowa sortowania bąbelkowego wynosi  $\mathcal{O}(n^2)$ , podczas gdy złożoność optymistyczna jest klasy  $\mathcal{O}(n)$  – zapis ten oznacza, że wraz ze wzrostem elementów w tablicy, tylko w przypadku danych już wcześniej posortowanych (optymistyczny wariant) widoczna będzie zależność liniowa. W każdym innym przypadku, czas wykonania będzie coraz bardziej zależny od kwadratu liczby elementów do posortowania.

## Przygotowanie platformy pomiarowej

W celu praktycznego sprawdzenia czasu wykonania funkcji sortowania, przygotowano zestaw funkcji odpowiadających za wylosowanie odpowiednich danych wejściowych. W skład zestawu wchodzą funkcje: `utworz_tablice_losowa()` (utworzenie i wypełnienie tablicy o określonym rozmiarze wartościami wylosowanymi z przedziału od 1 do 1000000), `utworz_tablice_rosnaca()` (utworzenie i wypełnienie tablicy ciągiem wartości rosnących) oraz `utworz_tablice_malejaca()` (utworzenie i wypełnienie tablicy ciągiem wartości malejących). Implementacje funkcji przedstawiono na rys. 14.1.

```
5  def utworz_tablice_losowa(rozmiar):
6      tab = [0] * rozmiar
7      i = 0
8      while i < len(tab):
9          tab[i] = randint(1, 1000000)
10         i += 1
11
12     return tab
13
14 def utworz_tablice_rosnaca(rozmiar):
15     tab = [0] * rozmiar
16     i = 0
17     wart = 0
18     while i < len(tab):
19         wart += randint(1, 20)
20         tab[i] = wart
21         i += 1
22
23     return tab
24
25 def utworz_tablice_malejaca(rozmiar):
26     tab = utworz_tablice_rosnaca(rozmiar)
27     tab.reverse()
28     return tab
```

Rys. 14.1. Zestawienie funkcji tworzących oraz wypełniających wartościami liczbowymi tablice o wskazanym rozmiarze

W trakcie pracy z sortowaniem bąbelkowym, złożoność optymistyczna będzie możliwa do sprawdzenia przekazując ciąg danych rosnących, dla złożoności pesymistycznej będzie to ciąg malejący – ciąg losowych wartości posłuży do obliczenia przeciętnej złożoności czasowej.

W przykładzie zaprezentowanym w niniejszym podpunkcie wykorzystane zostaną dwie implementacje funkcji sortowania bąbelkowego – pierwsza z nich będzie wersją zaprezentowaną w rozdziale trzecim (rys. 14.2).



Fundusze  
Europejskie  
Wiedza Edukacja Rozwój



Rzeczpospolita  
Polska



Unia Europejska  
Europejski Fundusz Społeczny

```
31     def sortuj_dane_babelkowo_klasycznie(tab):
32         n = len(tab)
33
34         while n > 1:
35             i = 0
36             while i < n - 1:
37                 if tab[i] > tab[i + 1]:
38                     pomoc = tab[i]
39                     tab[i] = tab[i + 1]
40                     tab[i + 1] = pomoc
41                 i += 1
42         n -= 1
```

Rys. 14.2. Funkcja realizująca sortowanie bąbelkowe bez dodatkowego warunku zakończenia

Analiza kodu funkcji każe przypuszczać, że funkcja powinna posiadać takie same złożoności czasowe: optymistyczną, pesymistyczną oraz przeciętną. Wynika to ze sposobu jej działania – nawet dla tablicy już wcześniej posortowanej funkcja będzie sukcesywnie wykonywać pętle, przeglądając kolejne pary elementów. Algorytm jej działania nie jest zależny od ustawienia danych wejściowych.

Lekko usprawnioną wersją sortowania bąbelkowego jest wersja, w której dodatkowo sprawdzany jest warunek końca pętli. W momencie, gdy wewnętrzna pętla przejdzie przez wszystkie pary elementów i nie zostanie zamieniona żadna z nich, można będzie zakończyć dalsze wykonywanie funkcji. Implementację przedstawiono na rys. 14.3.

```
44     def sortuj_dane_babelkowo_ze_sprawdzaniem_konca(tab):
45         n = len(tab)
46
47         zamiana = True
48         while n > 1 and zamiana == True:
49             i = 0
50             zamiana = False
51             while i < n - 1:
52                 if tab[i] > tab[i + 1]:
53                     pomoc = tab[i]
54                     tab[i] = tab[i + 1]
55                     tab[i + 1] = pomoc
56                     zamiana = True
57                 i += 1
58         n -= 1
```

Rys. 14.3. Funkcja realizująca sortowanie bąbelkowe z dodatkowym warunkiem zakończenia



Fundusze  
Europejskie  
Wiedza Edukacja Rozwój



Rzeczpospolita  
Polska



Unia Europejska  
Europejski Fundusz Społeczny

Funkcja wprowadza nową zmienną zamianą, która służy do sygnalizacji momentu, w którym po pełnym cyklu wewnętrznej pętli nie została zamieniona żadna para wartości.

Mając do dyspozycji funkcje pozwalające na utworzenie tablicy wypełnionej odpowiednimi danymi oraz funkcje wykonujące proces sortowania, pozostaje ostatni element w postaci kodu odpowiedzialnego za pomiar czasu. Zadanie polega na utworzeniu platformy testowej, pozwalającej na sprawdzenie, w jakim stopniu czas wykonania funkcji będzie zależny od wielkości tablicy oraz od sposobu ustawienia w niej elementów. Najprostszym sposobem na sprawdzenie czasu wykonania się określonego fragmentu kodu jest pobranie czasu przed jego wykonaniem oraz po jego wykonaniu – różnica w czasie będzie oznaczała czas wykonania się instrukcji. Pobranie aktualnego czasu wykonuje się z użyciem funkcji `time()` z biblioteki `time`. Sposób pomiaru czasu przedstawiono na rys. 14.4.

```
64     liczba_losowan = 100
65     rozmiary = [200, 400, 600, 800, 1000]
66     for rozmiar_tablicy in rozmiary:
67
68         suma = 0
69         licznik_losowan = 0
70         sumy_czasu = [0] * 3
71         while licznik_losowan < liczba_losowan:
72             wejście = [utwórz_tablice_rosnaca(rozmiar_tablicy),
73                        utwórz_tablice_losowa(rozmiar_tablicy),
74                        utwórz_tablice_malejaca(rozmiar_tablicy)]
75
76             licznik_tab = 0
77             for tab in wejście:
78                 czas1 = time()
79                 sortuj_dane_babelkowo_klasycznie(tab)
80                 czas2 = time()
81                 różnica = czas2 - czas1
82
83                 sumy_czasu[licznik_tab] += różnica
84                 licznik_tab += 1
85
86             licznik_losowan += 1
```

Rys. 14.4. Techniki pomiaru czasu dla różnych rozmiarów tablicy oraz różnego ułożenia danych wejściowych

Pomiar przeprowadzane są z zachowaniem poniższych reguł:

1. Pomiar czasu wykonywany jest dla pięciu różnych rozmiarów tablicy (200, 400, 600, 800 oraz 1000 elementów).
2. Dla każdego z powyższych rozmiarów przeprowadzany jest pomiar czasu. Aby uniezależnić pomiary czasu od czynników losowych, dla każdego rozmiaru tablicy pomiar czasu przeprowadzany jest 100 razy, by następnie obliczyć średni czas wykonania. Wraz ze wzrostem liczba powtórzeń (zmienna `liczba_losowan`), zwiększa się precyzja wyniku, jednak wydłuża się czas testu.



Fundusze Europejskie  
Wiedza Edukacja Rozwój



Rzeczpospolita  
Polska



Unia Europejska  
Europejski Fundusz Społeczny

3. Pomiar czasu realizowany jest dla trzech ułożen danych wejściowych: ciągu rosnącego, ciągu losowego oraz ciągu malejącego. Zarejestrowany czas przechowywany jest w trójelementowej zmiennej tablicowej sumy\_czasu.

Po przeprowadzeniu pomiaru dla każdego z rozmiarów, wynik wyświetlany jest w oknie konsoli zgodnie z rys. 14.5 (kod ten znajduje się zaraz po zakończeniu pętli while, jednak w dalszym ciągu w ciele pętli for).

```
88     print('Rozmiar tablicy: ' + str(rozmiar_tablicy))
89     czas_dla_rosnacego = str(sumy_czasu[0] * 1000 / liczba_losowan)
90     print('Średni czas dla ciągu rosnącego: ' + czas_dla_rosnacego + ' ms')
91     czas_dla_losowego = str(sumy_czasu[1] * 1000 / liczba_losowan)
92     print('Średni czas dla ciągu losowego: ' + czas_dla_losowego + ' ms')
93     czas_dla_malejacego = str(sumy_czasu[2] * 1000 / liczba_losowan)
94     print('Średni czas dla ciągu malejącego: ' + czas_dla_malejacego + ' ms')
```

Rys. 14.5. Fragment kodu odpowiedzialny za wyświetlenie obliczonego czasu wykonania funkcji sortowania

Tak przygotowany kod pozwoli na sprawdzenie czasu wykonania dowolnego zestawu instrukcji – wystarczy w miejsce instrukcji:

```
sortuj_dane_babelkowo_klasycznie()
```

wstawić dowolnie wybraną funkcję lub zestaw instrukcji do wykonania, a zmierzony zostanie czas dla trzech ustawień ciągu wejściowego.

### Pomiar czasu dla klasycznej implementacji sortowania bąbelkowego

Efektem uruchomienia programu dla sortowania bąbelkowego w postaci klasycznej jest przekazanie na wyjście konsoli wartości widocznych poniżej:

```
Rozmiar tablicy: 200
Średni czas dla ciągu rosnącego: 6.850407123565674 ms
Średni czas dla ciągu losowego: 10.400598049163818 ms
Średni czas dla ciągu malejącego: 13.800806999206543 ms
Rozmiar tablicy: 400
Średni czas dla ciągu rosnącego: 27.271575927734375 ms
Średni czas dla ciągu losowego: 41.622371673583984 ms
Średni czas dla ciągu malejącego: 55.01316547393799 ms
Rozmiar tablicy: 600
Średni czas dla ciągu rosnącego: 63.283610343933105 ms
Średni czas dla ciągu losowego: 96.75553321838379 ms
Średni czas dla ciągu malejącego: 126.82725429534912 ms
Rozmiar tablicy: 800
Średni czas dla ciągu rosnącego: 114.80656862258911 ms
Średni czas dla ciągu losowego: 175.40000677108765 ms
Średni czas dla ciągu malejącego: 230.60319423675537 ms
Rozmiar tablicy: 1000
Średni czas dla ciągu rosnącego: 179.14024591445923 ms
```



Fundusze  
Europejskie  
Wiedza Edukacja Rozwój



Rzeczypospolita  
Polska



Średni czas dla ciągu losowego: 276.3957977294922 ms
Średni czas dla ciągu malejącego: 361.8807029724121 ms

Już przy porównaniu czasów dla dwóch pierwszych rozmiarów tablicy widać, że nie jest to zależność liniowa – wraz z podwojeniem rozmiaru danych na wejściu, odpowiadające sobie czasy wzrosły ponad czterokrotnie. Dalsze wyniki potwierdzają powyższe obserwacje. W przypadku liniowej zależności, średni czas dla ciągu malejącego przy największym rozmiarze tablicy powinien wynosić w przybliżeniu 69 ms ( $13,8 \text{ ms} * 5$ ), jednak w praktyce otrzymano wartość równą w przybliżeniu 362 ms. Dalsze obliczenia pozwalają zgodnie z przypuszczeniami stwierdzić kwadratową zależność czasu wykonania od rozmiaru danych na wejściu.

Dodatkowego wyjaśnienia może wymagać różnica w czasach w obrębie tego samego rozmiaru tablicy. Przyglądając się implementacji funkcji z rys. 14.2. można zauważyć, iż funkcja sortowania wykonuje łącznie tyle samo instrukcji warunkowych, niezależnie od ułożenia danych wejściowych. To, co wpływa ostatecznie na różnice w czasie wykonania pomiędzy trzema ustawieniami danych wejściowych jest operacja zamiany dwóch wartości – w przypadku danych posortowanych malejąco, zamiana danych następuje w każdym przebiegu pętli, co wydłuża czas wykonania. Przy danych posortowanych rosnąco, kod znajdujący się w instrukcji warunkowej nie jest nigdy wykonywany, dzięki czemu funkcja kończy swoją pracę około dwukrotnie szybciej (dla 1000 elementów odpowiednio: 362 ms oraz 179 ms). Tego typu różnice zawierają się jednak w obrębie tej samej klasy złożoności (w tym przypadku kwadratowej) i są najczęściej pomijane.

Ostatecznie, w powyższym przykładzie można wyznaczyć złożoności czasowe w następującej postaci:

- optymistyczna złożoność czasowa –  $\mathcal{O}(n^2)$ ,
- przeciętna złożoność czasowa –  $\mathcal{O}(1,5n^2)$ ,
- pesymistyczna złożoność czasowa –  $\mathcal{O}(2n^2)$ .

co potwierdza złożoność czasową klasy  $\mathcal{O}(n^2)$ , niezależnie od ustawienia danych na wejściu.

### Pomiar czasu dla implementacji sortowania bąbelkowego sprawdzającej dodatkowy warunek zakończenia

Drugi test będzie dotyczyć lekko zmodyfikowanej wersji sortowania bąbelkowego, która dodatkowo sprawdza, czy po przejściu przez wszystkie elementy nastąpiła zamiana jakiejkolwiek pary liczb – jeżeli nie, oznacza to zakończenie sortowania. Efektem działania programu testowego jest zwrócenie wartości zaprezentowanych poniżej:

Średni czas dla ciągu rosnącego: 0.07000207901000977 ms
Średni czas dla ciągu losowego: 9.820544719696045 ms
Średni czas dla ciągu malejącego: 13.150763511657715 ms

```
Rozmiar tablicy: 400
Średni czas dla ciągu rosnącego: 0.14200901985168457 ms
Średni czas dla ciągu losowego: 40.805333852767944 ms
Średni czas dla ciągu malejącego: 53.94208312034607 ms
Rozmiar tablicy: 600
Średni czas dla ciągu rosnącego: 0.20701217651367188 ms
Średni czas dla ciągu losowego: 95.04043698310852 ms
Średni czas dla ciągu malejącego: 124.88713884353638 ms
Rozmiar tablicy: 800
Średni czas dla ciągu rosnącego: 0.2700185775756836 ms
Średni czas dla ciągu losowego: 172.92986869812012 ms
Średni czas dla ciągu malejącego: 226.81299448013306 ms
Rozmiar tablicy: 1000
Średni czas dla ciągu rosnącego: 0.3370187282562256 ms
Średni czas dla ciągu losowego: 272.52358531951904 ms
Średni czas dla ciągu malejącego: 357.6744601726532 ms
```

Zestawiając ze sobą wyniki zwrócone dla dwóch implementacji sortowania zauważać można jedynie minimalne różnice w czasie wykonania dla ciągu losowego oraz malejącego. Elementem, który odróżnia drugą implementację jest znaczna redukcja średniego czasu wykonania dla ciągu rosnącego – funkcja sortowania ze sprawdzeniem warunku końca wykonała się niespełna 100 razy szybciej, aniżeli jej klasyczny odpowiednik.

Różnica ta wynika z faktu, iż w przypadku drugiej wersji, przekazanie danych już posortowanych prowadzi do wykonania się tylko jednej iteracji pętli. Przy takim ustawieniu danych wejściowych, czas wykonania funkcji zależy jedynie od rozmiaru danych (im więcej jest danych, tych więcej czasu potrzeba na wykonanie jednego przebiegu pętli). Sytuację tą bardzo dobrze pokazują średnie czasy wykonania dla kolejnych rozmiarów, które w przybliżeniu wynoszą odpowiednio: 0,07 ms, 0,14 ms, 0,21 ms, 0,27 ms oraz 0,34 ms. Każdorazowe zwiększenie rozmiaru tablicy wejściowej skutkuje wydłużeniem czasu wykonania funkcji sortowania o 0,06-0,07 ms, co świadczy o zależności liniowej.

W przypadku ciągu losowego i malejącego funkcja sortowania wykonała się w czasie zbliżonym do czasu z pierwszej wersji, wykazując zależność kwadratową. W związku z powyższym, złożoność czasowa dla drugiej wersji algorytmu przedstawia się następująco:

- optymistyczna złożoność czasowa –  $O(n)$ ,
- przeciętna złożoność czasowa –  $O(1,5n^2)$ ,
- pesymistyczna złożoność czasowa –  $O(2n^2)$ .

Powyższe zestawienie kończy temat pomiaru czasu w ramach sortowania bąbelkowego.



Fundusze  
Europejskie  
Wiedza Edukacja Rozwój



Rzeczpospolita  
Polska



Unia Europejska  
Europejski Fundusz Społeczny

## **Zadania**

1. Obliczyć złożoność czasową programu obliczającego sumę wszystkich elementów tablicy mniejszych od największej liczby podzielnej przez pięć. Zweryfikować obliczenia doświadczalnie.
2. Obliczyć złożoności czasowe dwóch algorytmów wyszukiwania przedstawionych w laboratorium trzecim, tj. wyszukiwania liniowego oraz binarnego. Napisać program weryfikujący doświadczalnie wykonane obliczenia.
3. Porównać złożoności: obliczeniową oraz pamięciową algorytmu obliczania silni w postaci iteracyjnej oraz w postaci rekurencyjnej. Napisać program, który przedstawi te różnice w praktyce.
4. Obliczyć złożoności: czasową oraz pamięciową dwóch algorytmów Sita Eratostenesa. Napisać program weryfikujący doświadczalnie wykonane obliczenia.
5. Zaprojektować algorytm odnajdujący dwie największe wartości w tablicy liczb całkowitych. Obliczyć złożoności czasowe: optymistyczną oraz pesymistyczną.



**Fundusze  
Europejskie**  
Wiedza Edukacja Rozwój



**Rzeczpospolita  
Polska**





**Zintegrowany  
Program  
Rozwoju  
Politechniki  
Lubelskiej -  
część druga**

Materiały zostały opracowane w ramach projektu  
„Zintegrowany Program Rozwoju Politechniki Lubelskiej – część druga”,  
umowa nr **POWR.03.05.00-00-Z060/18-00**  
w ramach Programu Operacyjnego Wiedza Edukacja Rozwój 2014-  
2020  
współfinansowanego ze środków Europejskiego Funduszu  
Społecznego



**Fundusze  
Europejskie**  
Wiedza Edukacja Rozwój



**Rzeczpospolita  
Polska**

**Unia Europejska**  
Europejski Fundusz Społeczny

