



# Politechnika Śląska

## Języki Skryptowe

Dokumentacja projektu PIONEK

Kacper Grabiec, grupa 2C

Wydział Matematyki Stosowanej

Informatyka

# Część I

## Opis programu

### Zadanie: PIO

#### Pionek



XXV OI, etap I. Plik źródłowy `p1o.*` Dostępna pamięć: 128 MB.

16.10–13.11.2017

W punkcie  $(0,0)$  nieskończonej kratki stoi pionek. Pionek ma  $n$  dozwolonych ruchów. Każdy z nich jest opisany za pomocą wektora o współrzędnych całkowitych. Pionek może każdy z ruchów wykonać co najwyżej raz, w dowolnej kolejności. Wektory opisujące ruchy mogą się powtarzać i wtedy pionek może wykorzystać każdy z nich.

Naszym celem jest dostać się pionkiem do punktu położonego możliwie najdalej od punktu początkowego (w odległości euklidesowej). Jak daleko może on dotrzeć?

### Wejście

Pierwszy wiersz standardowego wejścia zawiera jedną dodatnią liczbę całkowitą  $n$  oznaczającą liczbę możliwych ruchów pionka. Każdy z kolejnych  $n$  wierszy zawiera dwie liczby całkowite  $x_i, y_i$  ( $-10^4 \leq x_i, y_i \leq 10^4$ ) oddzielone pojedynczym odstępem i oznaczające wektor  $[x_i, y_i]$  opisujący możliwy ruch pionka.

### Wyjście

Twój program powinien wypisać na standardowe wyjście liczbę całkowitą oznaczającą kwadrat odległości od punktu  $(0,0)$  do najdalszego punktu, do którego może doskoczyć pionek.

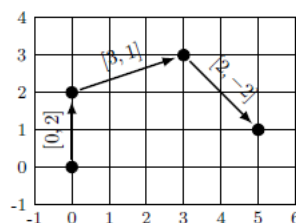
### Przykład

Dla danych wejściowych:

```
5
2 -2
-2 -2
0 2
3 1
-3 1
```

poprawnym wynikiem jest:

26



**Wyjaśnienie do przykładu:** Na rysunku przedstawiono rozwiązanie optymalne wykorzystujące ruchy opisane wektorami  $[0,2]$ ,  $[3,1]$  oraz  $[2,-2]$ . Inne, równie dobre rozwiązanie uzyskujemy za pomocą wektorów  $[0,2]$ ,  $[-3,1]$  oraz  $[-2,-2]$ .

## Instrukcja obsługi

W celu rozwiązania zadania należy uruchomić plik *bat.bat*. Po jego włączeniu ukaże się nam menu z kilkoma opcjami do wyboru.

```
-----MENU-----
1. Stworz raport
2. Wyświetl ostatni raport
3. Usun raporty
4. Uruchom skrypt py1
5. Uruchom skrypt py2
6. Koniec
-----
Podaj opcje:
```

Program czeka, aż wybierzemy jedną z dostępnych opcji:

- **Stwórz raport** – program rozwiązuje problem dla zadanych danych wejściowych oraz tworzy raport na podstawie danych wyjściowych;
- **Wyświetl ostatni raport** – program wyświetla nam najnowszy plik *.html* zawierający gotowy raport;
- **Usuń raporty** – program usuwa wszystkie raporty znajdujące się w folderze */raporty*;
- **Uruchom skrypt py1.py** – program uruchamia tylko plik *py1.py*, odpowiedzialny za rozwiązanie zadania;
- **Uruchom skrypt py2.py** – program uruchamia tylko plik *py2.py*, odpowiedzialny za stworzenie raportu;
- **Koniec** – koniec działania programu;

Po wykonaniu każdej opcji menu wyświetla się ponownie.

Każdy plik zawierający dane wejściowe (np. *in1.txt*) znajduje się w folderze */input\_data*. Jego zawartość wygląda następująco:

```
1 5
2 2 -2
3 -2 -2
4 0 2
5 3 1
6 -3 1
```

Pierwszy wiersz standardowego wejścia zawiera jedną dodatnią liczbę całkowitą  $n$  oznaczającą liczbę możliwych ruchów pionka. Każdy z kolejnych  $n$  wierszy zawiera dwie liczby całkowite  $x, y$  oddzielone pojedynczym odstępem i oznaczające wektor  $[x, y]$  opisujący możliwy ruch pionka.

Każdy plik zawierający dane wyjściowe (n. *out1.txt*) znajduje się w folderze */output\_data*. Jego zawartość wygląda następująco:

```
1 26
```

Program wypisuje na standardowe wyjście liczbę całkowitą oznaczającą kwadrat odległości od punktu  $(0, 0)$  do najdalszego punktu, do którego może doskoczyć pionek.

Wygenerowany raport wygląda następująco:

Kacper Grabiec Języki Skryptowe: PIONEK		
5 2 -2 -2 -2 0 2 3 1 -3 1		26
10 -3367 407 1495 5978 -1201 1172 -3809 -4844 -1862 -6169 -7967 5501 1643 9629 9635 2644 4478 -167 -5833 -1862		892299938
1 4909 -9148		107784185

Po lewej stronie wyświetlone zostają kolejne pliki zawierające dane wejściowe, po prawej dane wyjściowe.

# Część II

## Opis działania

Zadanie polega na wyznaczeniu liczby całkowitej będącą kwadratem odległości od punktu  $(0, 0)$  do najdalszego punktu, do którego może doskoczyć pionek. Wartość tą uzyskamy ze wzoru:

$$|\vec{u}| = \sqrt{u_x^2 + u_y^2}$$

Jako, że potrzebujemy kwadrat odległości, wynik otrzymamy w następujący sposób:

$$|\vec{u}|^2 = u_x^2 + u_y^2$$

Ostateczny wektor zmienia się w zależności od tego ile ruchów wykorzystamy. W tym celu musimy także dodawać następny wektory ruchu. Można to zrobić według następującego wzoru:

$$\overrightarrow{AB} = [x_B - x_A, y_B - y_A]$$

## Algorytm

**Data:** Liczba ruchów  $n$ , ruchy

**Result:** Kwadrat odległości

$i := 0$ ;

**while**  $i < n$  **do**

    Sprawdź czy  $i$ -ty ruch nie został już wykorzystany;

**if** *NIE* **then**

        Dodaj  $i$ -ty ruch do wykorzystanych;

        Dla każdego wykorzystanego ruchu oblicz wektor od punktu  $(0, 0)$ ;

        Dla każdego wykorzystanego ruchu oblicz odległość od punktu  $(0, 0)$ ;

**if** *odległość*  $>$  *obecnaOdległość* **then**

            | *obecnaOdległość* = *odległość*;

**end**

        solve();

**end**

**end**

**Algorithm 1:** Funkcja solve().

## Implementacja

Plik *py1.py*

```
4 def load_data(filename: str):
5     file = open('./input_data/' + filename, 'r')
6     data = []
7     for line in file:
8         line = line.strip()
9         data.append([int(value) for value in line.split(' ')])
10    file.close()
11    return data
12
13
14 def print_data(filename: str, data: str):
15     file = open('./output_data/' + filename, 'w')
16     file.write(data)
17     file.close()
```

Powyższe funkcje służą do odczytywania danych z pliku oraz do ich zapisywania. Funkcja *load\_data()* otwiera plik i zwraca zawartość w postaci tablicy.

Funkcja *print\_data()* zapisuje otrzymane wyniki do pliku.

```
20 def calc_sqr_vec_len(vector):
21     return pow(vector[0], 2) + pow(vector[1], 2)
22
23
24 def sum_vectors(v1, v2):
25     return [v1[0] + v2[0], v1[1] + v2[1]]
```

Funkcje matematyczne służące do obliczenia długości wektora oraz do dodawania wektorów.

```

28 def solve():
29     for i in range(0, n):
30         skip = False
31         vector = [0, 0]
32         global max_distance
33         for j in range(0, len(indexes)):
34             if i == indexes[j]:
35                 skip = True
36                 break
37         if skip:
38             continue
39         else:
40             indexes.append(i)
41             for j in range(0, len(indexes)):
42                 vector = sum_vectors(vector, moves[indexes[j]])
43             results.append(vector)
44             distance = calc_sqr_vec_len(results[-1])
45             if distance > max_distance:
46                 max_distance = distance
47             while len(results) > 0:
48                 results.pop()
49             solve()
50             indexes.pop()

```

Funkcja *solve()* jest najważniejszą funkcją w programie. Jest ona wywoływana rekurencyjnie. Sprawdza, czy dany ruch nie został już wykorzystany a następnie oblicza dystans. Rekurencja przerywa się gdy wszystkie ruchy zostaną wykorzystane.

Plik *py2.py*

```
6  def load_data(directory: str):
7      data = []
8      path = os.listdir(directory)
9      for file in path:
10         f = open(str(directory + file))
11         data.append(f.readlines())
12         f.close()
13     return data
```

Funkcja *load\_data()* wczytuje zawartości wszystkich plików w danym katalogu i zwraca je w postaci tablicy.

```
16 def generate_template():
17     template = ''
18     for i in range(0, len(input_files)):
19         template += ''
20         <div class="line">
21             <div class="data">
22                 ''
23                 for j in input_files[i]:
24                     template += f''
25                     <div class="input">{j}</div>
26                     ''
27                 template += f''
28                 </div>
29                 <div class="data">{output_files[i][0]}</div>
30             </div>''
31     return template
```

Funkcja *generate\_template()* dla każdego danych wejściowych oraz wyjściowych tworzy osobny element *HTML*.



```

34 def print_data():
35     time = datetime.now()
36     time = time.strftime("%H-%M-%S")
37     today = date.today()
38     today = today.strftime("%d-%m-%Y")
39     file = open(f'./raporty/{today}_{time}_raport.html', 'w')
40     file.write(f'''
41     <!DOCTYPE HTML>
42     <html lang="pl-PL">
43         <head>
44             <meta charset="utf-8">
45             <title>{today}_{time}_Raport</title>
46             <meta name="viewport" content="width=device-width, initial-scale=1">
47
48             <link rel="stylesheet" href="../style.css">
49         </head>
50
51         <body>
52             <h1>Kacper Grabiec</h1>
53             <h3>Języki Skryptowe PIONEK</h3>
54             {generate_template()}
55         </body>
56     </html>
57     ''')
58     file.close()

```

Funkcja *print\_data()* wczytuje obecną datę oraz czas. Następnie tworzy podstawowy szablon *HTML*, który zestawia dane wejściowe z danymi wyjściowymi.

## Pełen kod aplikacji

<https://github.com/KacperGrabiec/JęzykiSkryptoweProjekt>