

Języki Skryptowe

dokumentacja projektu Randka olimpiada informatyczna XIX

Paweł Habrzyk, grupa 3F, Wydział Matematyki Stosowanej - Informatyka semeste 3

11 stycznia 2021

Spis treści

1	Opis problemu przedstawionego w zadaniu	3
1.1	Treść zadania	3
1.2	Założenia	3
1.3	Przykładowy format danych wejściowych i wyjściowych	4
2	Model Matematyczny	4
2.1	Analiza problemu	4
2.2	Rozwiązanie siłowe	5
2.3	Instrukcja obsługi	5
2.4	Wymagania sprzętowe	6
3	Algorytm	6
3.1	Pseudokod	6
3.2	Schemat blokowy	7
4	Implementacja	7
5	Podsumowanie	14
5.1	Co zostało zrobione	14
5.2	Dalsze prace	14

1 Opis problemu przedstawionego w zadaniu

1.1 Treść zadania

Bajtazar jest strażnikiem przyrody i pracuje w Jaskini Strzałkowej — znanym miejscu schadzek zakochanych par. Jaskinia ta składa się z n komór połączonych jednokierunkowymi korytarzami. W każdej komorze dokładnie jeden wychodzący z niej korytarz jest oznaczony strzałką. Każdy korytarz prowadzi bezpośrednio z jednej komory do pewnej (niekoniecznie innej) komory. Notorycznie zdarza się, że zakochane pary, które umawiają się na randki w Jaskini Strzałkowej, zapominają dokładnie ustalić miejsce spotkania i nie mogą się odnaleźć. W przeszłości prowadziło to do wielu nieporozumień i pomyłek. . . . Od czasu, gdy w każdej komorze zainstalowano telefon alarmowy łączący z dyżurnym strażnikiem przyrody, głównym zajęciem strażników stało się pomaganie zakochanym parom w odnajdowaniu się. Strażnicy wypracowali następującą metodę. Wiedząc, w których komorach znajdują się zakochani, mówią każdemu z nich, ile razy, odpowiednio, powinien przejść z komory do komory korytarzem oznaczonym strzałką, aby oboje mogli się spotkać. Przy tym, zakochani bardzo chcą spotkać się jak najszybciej — zależy im przecież, aby razem miło spędzać czas, a nie samotnie przemierzać korytarze. Strażnicy starają się podawać zakochanym parom takie liczby, aby ich maksima były możliwie jak najmniejsze. Bajtazar jest już zmęczony ciągłym pomaganiem zakochanym i poprosił Cię o napisanie programu, który by to usprawnił. Program ten, na podstawie opisu Jaskini Strzałkowej oraz aktualnego położenia k zakochanych par, powinien wyznaczyć k par liczb x_i i y_i , takich że:

- jeżeli i -ta para zakochanych przejdzie odpowiednio: x_i i y_i korytarzami oznaczonymi strzałkami, to spotkają się w jednej komorze jaskini,
- $\max(x_i, y_i)$ jest jak najmniejsze,
- w drugiej kolejności $\min(x_i, y_i)$ jest jak najmniejsze,
- jeżeli rozwiązanie wciąż nie jest jednoznaczne, to kobieta powinna pokonywać mniejszy dystans. Może się tak zdarzyć, że takie liczby x_i i y_i nie istnieją — wówczas przyjmujemy, że $x_i = y_i = -1$.

1.2 Założenia

Wejście - **folder: input**

W pierwszym wierszu standardowego wejścia znajdują się dwie dodatnie liczby całkowite n i k ($1 \leq n \leq 500\,000$, $1 \leq k \leq 500\,000$), oddzielone pojedynczym odstępem i określające liczbę komór w Jaskini Strzałkowej i liczbę zakochanych par, które chcą się odnaleźć. Komory są ponumerowane od 1 do n , natomiast zakochane pary są ponumerowane od 1 do k . W drugim wierszu wejścia znajduje się n liczb całkowitych dodatnich, pooddzielanych pojedynczymi odstępami: i -ta liczba w tym wierszu określa numer komory, do której prowadzi korytarz oznaczony strzałką wychodzący z komory numer i .

W kolejnych k wierszach znajdują się kolejne zapytania zakochanych par. Każde zapytanie składa się z dwóch liczb całkowitych dodatnich oddzielonych pojedynczym odstępem — oznaczają one numery komór, w których znajduje się dana para zakochanych — najpierw x_i , a potem y_i . W testach wartych łącznie $40k \leq 2\,000$.

Wyjście 1 - **folder: output**

Pierwsza linia tego pliku jest obliczony czas w sekundach, a drugą linią tego pliku jest przeliczony czas z sekund na format godzina, minuta, sekunda milisekunda.

Wyjście 2 - **plik: output.html**

Plik zawiera czytelną tabelę stworzoną w html'u z zawartymi wynikami z output.txt

1.3 Przykładowy format danych wejściowych i wyjściowych

input.txt

```
12 5
4 3 5 5 1 1 12 12 9 9 7 1
7 2
8 11
1 2
9 10
10 5
```

output.txt

```
(2,3)
```

output.html

Tabela z danymi wejściowym i wyjściowymi

2 Model Matematyczny

2.1 Analiza problemu

Na początku zastanówmy się, jaką postać ma graf opisany w tym zadaniu. Jest to graf skierowany, w którym z każdego wierzchołka wychodzi dokładnie jedna krawędź. Nasz graf składa się zatem z pewnej liczby cykli, do których mogą być dołączone drzewa¹. Powiemy, że dwa wierzchołki należą do tej samej słabo spójnej składowej grafu, jeśli można przejść z jednego z nich do drugiego po strzałkach, ignorując ich skierowanie (tzn. możemy iść w przód lub w tył). W każdej składowej znajduje się dokładnie 1 Ciekawe własności takich grafów były niejednokrotnie wykorzystywane w rozwiązaniach zadań olimpijskich, np. w zadaniu Mafia z XV Olimpiady Informatycznej [15] i w zadaniu Szpieczy z XI Olimpiady Informatycznej [11]. Randka 79 jeden cykl. W dalszym opisie przyjmiemy, że każdy wierzchołek cyklu stanowi korzeń pewnego drzewa dołączonego do cyklu — w niektórych przypadkach drzewo to składa się tylko z tego wierzchołka. Następnie mamy daną listę par wierzchołków w grafie. Dla każdej pary wierzchołków u, v musimy znaleźć taką optymalną parę liczb x, y , że po przejściu

x kroków z wierzchołka u i y kroków z wierzchołka v dojdziemy do tego samego wierzchołka. Szukana optymalna para x, y powinna przede wszystkim minimalizować $\max(x, y)$, w drugiej kolejności — $\min(x, y)$, a w trzeciej kolejności — liczbę y . Pomyślmy teraz, gdzie może leżeć wierzchołek, w którym nastąpi spotkanie. Możliwe są trzy przypadki, w zależności od wzajemnego położenia u i v :

1. Wierzchołki leżą na jednym drzewie i na nim też nastąpi spotkanie.
2. Wierzchołki leżą na różnych drzewach, których korzenie znajdują się na jednym cyklu. Wówczas spotkanie nastąpi na tym cyklu.
3. Wierzchołki leżą w różnych składowych grafu, zatem spotkanie jest niemożliwe. Zauważmy, że łatwo stwierdzić, czy mamy do czynienia z przypadkiem 3, jeśli dla każdego wierzchołka znamy numer składowej grafu, w której się ten wierzchołek znajduje. Takie numery dla wszystkich wierzchołków można wyznaczyć w czasie $O(n)$, przeszukując graf i ignorując skierowanie krawędzi (opisujemy to także w sekcji „Rozwiązanie wzorcowe”). W dalszej części opisu będziemy zatem zakładać, że mamy do czynienia z przypadkiem 1 lub 2.

2.2 Rozwiązanie siłowe

Na podstawie powyższych obserwacji możemy skonstruować pierwsze rozwiązanie. Zaznaczamy wszystkie wierzchołki, do których można dojść z wierzchołka u . Następnie, startując z wierzchołka v , idziemy po strzałkach, aż dojdziemy do jednego z wierzchołków, które zaznaczyliśmy. Później robimy to samo, odwracając role. Jako wynik podajemy lepsze z dwóch znalezionych miejsc spotkania (używając zadanego kryterium porównywania wyników). Dlaczego otrzymany w ten sposób wynik jest optymalny? Wróćmy do określonych wcześniej przypadków. W pierwszym przypadku, wykonując powyższy algorytm, znajdziemy najbliższego wspólnego przodka wierzchołków u i v w drzewie. Jedynymi innymi potencjalnymi miejscami spotkań są wierzchołki bliższe korzeniom lub wierzchołki leżące na cyklu. Jednak żeby do nich dojść, trzeba zwiększyć zarówno x , jak i y , co z oczywistych względów nie da nam lepszego wyniku. Drugi przypadek jest trochę bardziej skomplikowany. Potencjalne miejsca spotkań leżą na cyklu. Jeśli więc któryś wierzchołek z pary nie leży na cyklu, to musimy przejść z niego w górę drzewa, aż dojdziemy do cyklu. Kiedy już oba wierzchołki znajdują się na cyklu, mamy dwie możliwości: albo przejdziemy po cyklu z pierwszego wierzchołka do drugiego, albo na odwrót. Tym dwóm przypadkom odpowiadają dwa przebiegi naszego algorytmu. Zatem bierzemy pod uwagę dwa potencjalne miejsca spotkań: pierwszy wierzchołek leżący na cyklu należący do ścieżki wychodzącej z u oraz analogiczny wierzchołek należący do ścieżki wychodzącej z v . Każdy inny wierzchołek cyklu może być tylko gorszym kandydatem na miejsce spotkania, ponieważ ścieżki z u oraz z v do dowolnego niewybranego przez nas wierzchołka na cyklu prowadzą, odpowiednio, przez wybrane przez nas miejsca. Złożoność czasowa tego rozwiązania to $O(n \cdot k)$.

2.3 Instrukcja obsługi

W celu uruchomienia programów projektowy należy otworzyć plik `menu.bat` (uprawnienia administracyjne nie są wymagane)

2.4 Wymagania sprzętowe

System operacyjny Windows 10

Interpreter języka Python w wersji 3.X.X

3 Algorytm

3.1 Pseudokod

Data: Dane wejściowe: punkty początkowe, graf skierowany $x \xrightarrow{y}$

Result: (il-kr-ch, il-kr-dz)

Wprowadź dane

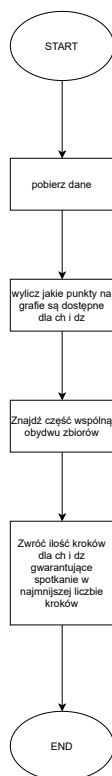
Wylicz wszystkie dostępne punkty dla chłopca

Wylicz wszystkie dostępne punkty dla dziewczyny

Wyciągnij część wspólną z obydwu zestawów

Zwróć ten zestaw który gwarantuje najmniejszą ilość kroków

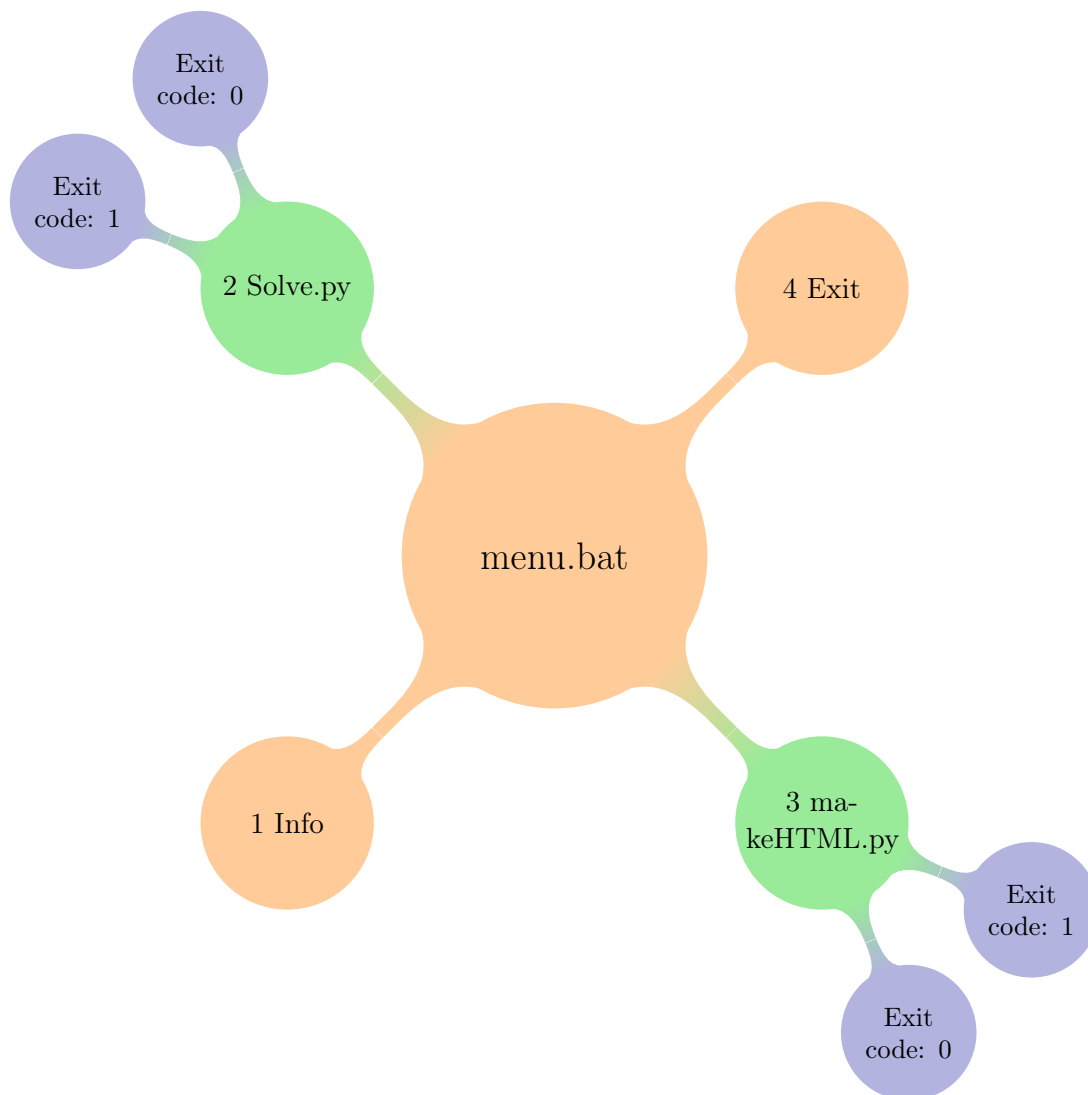
3.2 Schemat blokowy



4 Implementacja

Opis, zasada i działanie programu ze względu na podział na pliki, następnie funkcje programu wraz ze szczegółowym opisem działania (np.: formie pseudokodu, czy odniesienia do równania)

Program menu.bat jest skryptem systemowym, który steruję wszystkimi programami



Kod wyjścia 1 - skrypt zadziałał prawidłowo

Kod wyjścia 0 - błąd zapisu danych

menu.bat

```

1      @echo off
2  :menu
3  cls
4  echo Projekt zaliczeniowy[Wykryty]zyk[Wykryty]skryptowych
5  echo =====
6  echo 1   Informacja na temat polecenia
7  echo 2   Wykonaj obliczenia (python)
8  echo 3   Utworz kopie zapasowa
9  echo 4   Zakończ
10 echo =====
11 set /p select=Wybierz opcje (np. 2):
12 IF %select%==1 GOTO opt1
13 IF %select%==2 GOTO opt2
14 IF %select%==3 goto opt3
  
```



```

15 IF %select%==4 goto exit
16 :opt1
17 echo
=====
18 echo Polecenie jest [U+FFFD]puj[U+FFFD]ce
19 echo Na grafie skierowanym na [U+FFFD]y [U+FFFD] [U+FFFD]najkrut[sz[U+FFFD] dog[U+FFFD]
   [U+FFFD]ra
20 echo doprowadzi do spotkania w jednym punkcie 2 [U+FFFD]agat[U+FFFD]w.
21 echo
22 echo
=====
23 pause
24 goto menu
25 :opt2
26 echo
=====
27 python solve.py
28 If %ERRORLEVEL% == 1 (echo Skrypt [U+FFFD]a[U+FFFD]prawid[U+FFFD]owo)
29 If %ERRORLEVEL% == 0 (echo [U+FFFD] [U+FFFD]zapisu [U+FFFD]wyj[U+FFFD]ciowych)
30 If %ERRORLEVEL% == -1 (echo [U+FFFD] [U+FFFD]otwarcia [U+FFFD]wyj[U+FFFD]ciowych)
31
32 python makeHTML.py
33 If %ERRORLEVEL% == 1 (echo Zapis raport w postaci pliku html
   [U+FFFD]a[U+FFFD]prawid[U+FFFD]owo)
34 If %ERRORLEVEL% == 0 (echo [U+FFFD] [U+FFFD]zapisu [U+FFFD]wyj[U+FFFD]ciowych w
   postaci pliku html)
35 If %ERRORLEVEL% == -1 (echo [U+FFFD] [U+FFFD]otwarcia [U+FFFD]wyj[U+FFFD]ciowych
   podczas tworzenia raportu)
36 echo
=====
37
38 pause
39 goto menu
40 :opt3
41 echo
=====
42 echo Kopiowanie folderu %cd%...
43 echo Usuwanie starej kopii zapasowej...
44 rmdir /S /Q %userprofile%\Backup
45 echo Tworzenie nowej kopii zapasowej...
46 mkdir %userprofile%\Backup
47 xcopy /e /v "%cd%" "%userprofile%\Backup"
48 echo
=====
49 pause
50 goto menu
51 :exit
52 echo Koniec
53 pause

```

solve.py

```

1     import os
2
3

```

```

4  ## global
5
6  tree_size, flie_count, save_failures, boy_dat, girl_dat = 0, 0, 0, 0, 0
7
8  def brute(data):
9      # method appends to list every possible path for both boy and girl
10     # on a directional
11     # graph and than compares them to find commony approachable nodes. It
12     # returns numbers
13     # of steeps needed to reach said node for (boy, girl) with shortest
14     # path. In case of
15     # multiple fitting solutions it returns the one in which girl has to
16     # scale the shortest path.
17
18     tree = data[1]
19     for para in range(2, data[0][1]+2):
20
21         #boy
22         current_node = data[para][0]-1
23         approachable = dict()
24         for i in range(data[0][0]):
25             if current_node+1 in approachable:
26                 current_node = tree[current_node]-1
27                 continue
28             approachable[current_node+1]=i
29             current_node = tree[current_node]-1
30
31         #girl
32         current_node = data[para][1]-1
33         approachable2 = dict()
34         for i in range(data[0][0]):
35             if current_node+1 in approachable2:
36                 current_node = tree[current_node]-1
37                 continue
38             approachable2[current_node+1]=i
39             current_node = tree[current_node]-1
40
41         #common nodes
42         combine = {}
43         for i in approachable.keys():
44             try:
45                 combine[(approachable[i], approachable2[i])] = (
46                     approachable2[i]**5+approachable[i]**5)-(approachable2[i]
47                     ]<approachable[i])
48             except KeyError:
49                 pass # in case keys do not match
50
51         try:
52             return min(combine, key=combine.get)
53         except:
54             return '(-1,-1)'
55
56     ### helpres
57
58     def data_miner(f):

```

```

53     parsed_data=[]
54     for line in f.readlines():
55         parsed_data.append(line.rstrip().split(" "))
56     for i in range(len(parsed_data)):
57         for j in range(len(parsed_data[i])):
58             parsed_data[i][j] = int(parsed_data[i][j])
59     return parsed_data
60
61
62
63 try:
64     for filename in os.listdir(os.path.join(os.path.dirname(__file__), "
        input")):
65         with open(os.path.join(os.path.dirname(__file__),"input",
            filename), 'r') as f:
66             try:
67
68                 file_out = open(os.path.join(os.path.dirname(
                    __file__), "output","output_{}.txt".format(
                        filename)), "w")
69                 data = data_miner(f)
70                 cst = brute(data)
71                 file_out.writelines(str(cst))
72                 file_out.close()
73                 flie_count+=1
74                 girl_dat += cst[1]
75                 boy_dat += cst[0]
76                 tree_size+=data[0][0]
77             except IOError:
78                 save_failures+=1
79                 file_out = open("failed_files.txt", "a")
80                 file_out.writelines("W+FFDFFD zapisu danych dla danych
                    z przetwarzania pliku {}".format(filename))
81                 file_out.close()
82 except:
83     exit(0)
84 file_out = open("raport.txt", "w")
85 file_out.writelines("Processed files: \n{}\nFailed files\n{}\nBoy avg:\n
        {}\nGirl data\n{}\nTree size\n{}".format(flief_count,save_failures,
            boy_dat, girl_dat,tree_size))
86 file_out.close()
87 exit(1)

```

makeHTML.py

```

1  # Zapis do pliku html
2  # Zapis do pliku html
3  try:
4      file_in = open("raport.txt", "r")
5      lines = file_in.readlines()
6      file_in.close()
7  except IOError:
8      print("Nie znaleziono pliku")
9      exit(-1)
10

```

```

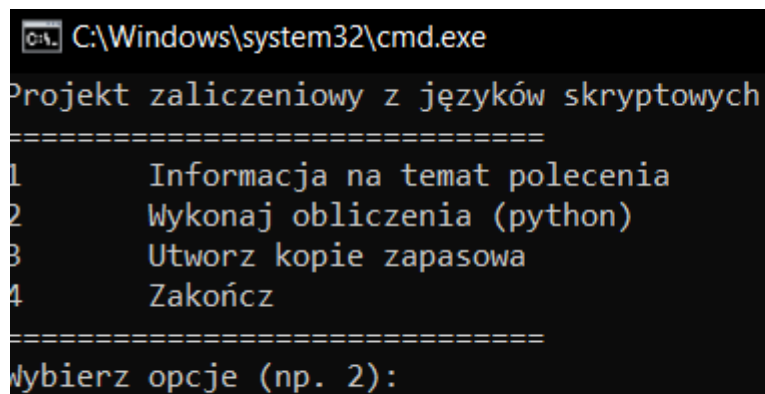
11 try:
12     x = int(lines[1])
13     y = int(lines[3])
14     z = int(lines[5])
15     t = int(lines[7])
16     q = int(lines[9])
17 except ValueError:
18     print("Nieprawidlowe dane wejsciowe, program zostanie wykonany na
        poniższych danych:")
19
20
21 try:
22     file_html = open("output.html", "w")
23
24     message = """
25     <!DOCTYPE html>
26     <html>
27     <head>
28     <link rel="stylesheet" href="styles.css">
29     </head>
30     <body>
31
32
33     <table class="steelBlueCols">
34     <thead>
35     </thead>
36     <tbody>
37     <tr>
38     <td>Przetworzone pliki</td><td>"""
39
40     message_res_1 = x
41     message_res_4 = y
42     message_res_4_1 = q/x
43     message_res_3 = "{}|{}".format(z/x,t/x)
44     message2 = """
45     </td>
46     <td>Przetworzone pliki</td><td>"""
47     message3 = """
48     </td></tr>
49     <tr>
50     <td>redni punkty poczlowe(b|g)</td><td>"""
51
52
53     message4 = """
54     </td></tr>
55     <tr>
56     <td>Pliki nieprzetworzone</td><td>"""
57     message4_1 = """
58     </td></tr>
59     <tr>
60     <td>redni rozmiar drzewa</td><td>"""
61
62
63     message5 = """
64     </td></tr>

```

```

65     <tr>
66     </tbody>
67     </tr>
68     </table>
69
70     </body>
71     </html>"""
72     file_html.writelines(str(message))
73     #file_html.writelines(str(message2))
74     file_html.writelines(str(message_res_1))
75     file_html.writelines(str(message3))
76     file_html.writelines(str(message_res_3))
77     file_html.writelines(str(message4))
78     file_html.writelines(str(message_res_4))
79     file_html.writelines(str(message4_1))
80     file_html.writelines(str(message_res_4_1))
81     file_html.writelines(str(message5))
82     file_html.close()
83 except IOError:
84     print("Błąd przy zapisu danych")
85     exit(0)
86
87 exit(1)

```



C:\Windows\system32\cmd.exe

Projekt zaliczeniowy z języków skryptowych

```

=====
1      Informacja na temat polecenia
2      Wykonaj obliczenia (python)
3      Utworz kopie zapasowa
4      Zakończ
=====
Wybierz opcje (np. 2):

```

Przetworzone pliki	54
Średnie punkty początkowe(b g)	2.0 3.0
Pliki nieprzetworzone	0
Średni rozmiar drzewa	12.0

5 Podsumowanie

5.1 Co zostało zrobione

Program pozwala wczytać wyprowadzone dane, potrzebne do obliczenia czasu, po którym wskazówki zegara ustawią się jednej linii. Problem matematyczny nie był, aż tak trudny w tym zadaniu co pozwoliło skupić się na integracji skryptów i utrwalenia wiedzy zdobytej w trakcie nauki języków skryptowych.

5.2 Dalsze prace

Algorytm jest niewydajny, dobrym pomysłem wydaje się tego poprawa.