Instrukcja użytkowania na potrzeby testów czarnoskrzynkowych

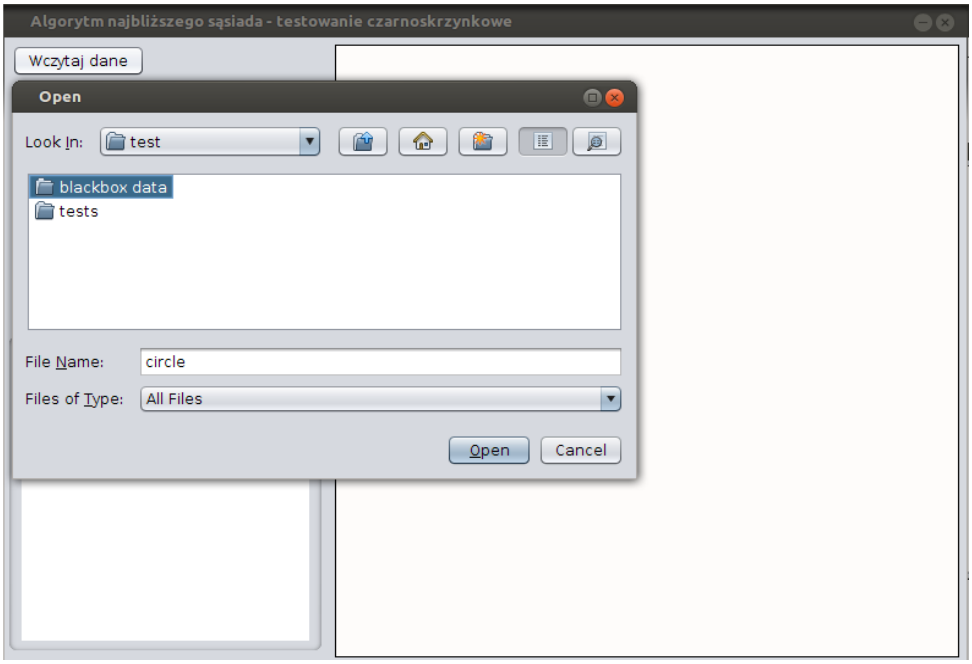
Opis programu

Graficzny program do testowania algorytmów najbliższego sąsiada umożliwia:

- wybór algorytmu (BruteForce/KD-Drzewo)
- wybór zbioru danych
- testowanie wydajności – pomiar czasu działania
- wizualizacja problemu

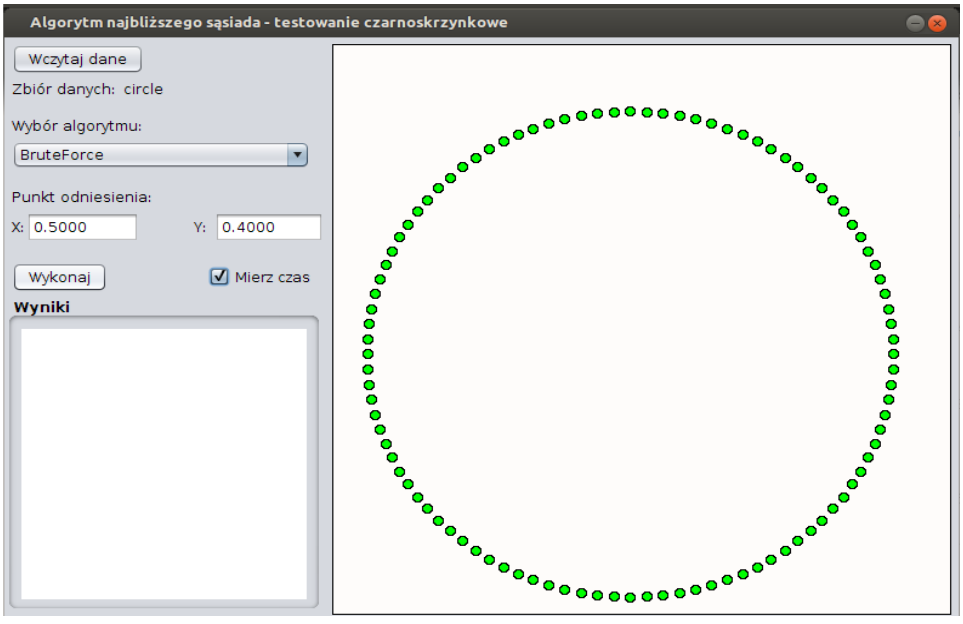
Instrukcja przykładowego użycia:

1. Wybierz przycisk „wczytaj dane” i wybierz plik z przygotowanymi danymi testowymi

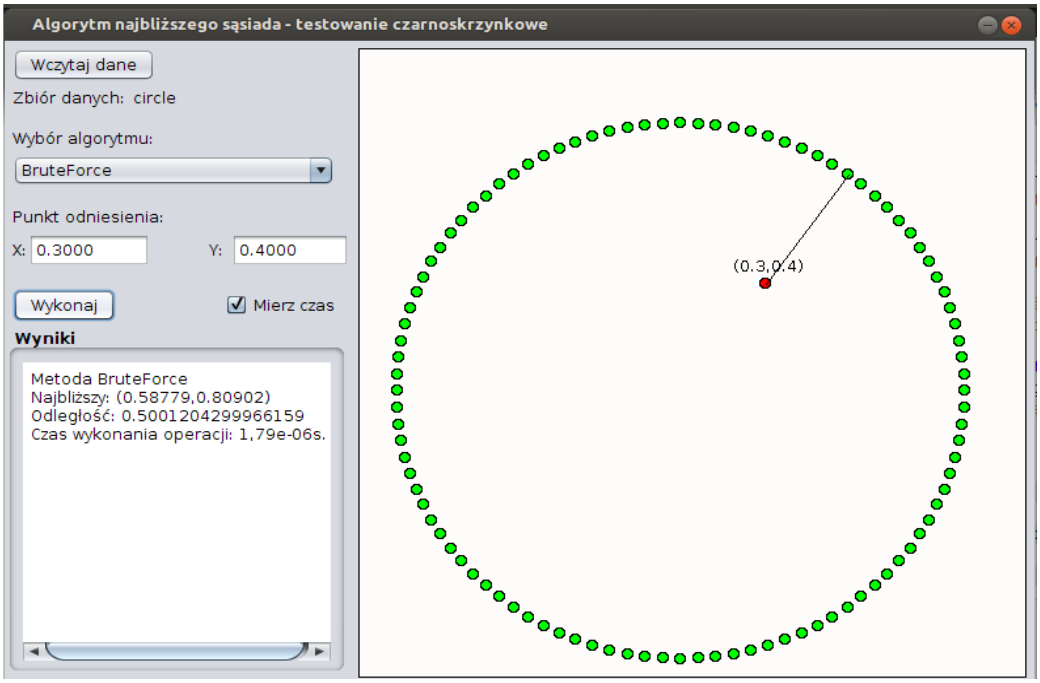


2. Wybierz algorytm (BruteForce/KD-Drzewo)

3. Wprowadź pozycję (X,Y) punktu pomiarowego i kliknij „Wykonaj”



4. Wprowadź pozycję punktu kontrolnego i kliknij „wykonaj”



W panelu graficznym został naniesiony wybrany punkt ze swoimi współrzędnymi oraz linia łącząca go z najbliższym sąsiadem. W polu tekstowym Wyniki została wypisana nazwa użytego algorytmu, współrzędne najbliższego sąsiada, odległość do niego oraz czas wykonania operacji (uśredniony).

Przykładowe testy czarnoskrzynkowe

Format danych wejściowych, na podstawie przykładowego pliku:

3 - liczba punktów

1.0 1.0 - współrzędne (x,y) pierwszego punktu

0.0 1.0

0.0 0.0

Testy poprawności i zgodności algorytmów

Nazwa pliku i opis	Punkt odniesienia	Przewidywany wynik	Wynik testu BF / KD
1_p - 1 punkt o współrzędnych (0.0, 0.0)	(20.0, 20.0)	(20.0, 20.0)	**(20.0, 20.0) / (20.0, 20.0)**
two_p – 2 punkty o współrzędnych (1.0, 1.0) leżące na sobie	(1.0, 1.0)	(1.0, 1.0)	**(1.0, 1.0) / (1.0, 1.0)**
zero_p – brak punktów	(0.0, 0.0)	Wyjątek	**Wyjątek / null** **Wyniki poprawne, ale różniące się między wersjami algorytmu**
quad_p – kwadrat o bokach (0,0), (1,0), (1,1), (0,1)	(0.5, 0.5) – przecięcie przekątnych	Którykolwiek z punktów kwadratu	**(0.0, 0.0) / (0.0, 1.0)** **Wyniki poprawne, ale różniące się między wersjami algorytmu**
quad_p	(1.0, 0.5) – środek boku	(1.0, 0.0) lub (1.0, 1.0)	**(1.0, 1.0) / (1.0, 0.0)** **Wyniki poprawne, ale różniące się między wersjami algorytmu**

quad_p	(-100.0, -100.0) – punkt o ujemnych współrzędnych, poza kwadratem	(0.0, 0.0)	**(0.0, 0.0) / (0.0, 0.0)**
circle – 100 punktów ułożone w okrąg o promieniu 1.0 i środku w punkcie (0,0)	(0.0, -0.9) – wewnątrz okręgu	(0.0, -1.0)	**(0.0, -1.0) / (0.0, -1.0)**
circle	(10000.0, 0.0) – na zewnątrz okręgu	(1.0, 0.0)	**(1.0, 0.0) / (1.0, 0.0)**
Circle_50000 – 50000 punktów ułożone w okrąg jednostkowy o środku w punkcie (0,0)	(0.0, 0.0) – środek okręgu	Dowolny punkt na okręgu	**(0.86195,0.50698) / (0.50698,0.86195) Wyniki poprawne, ale różniące się między wersjami algorytmu**
Circle_100000 – 100000 punktów ułożone w okrąg jednostkowy o środku w punkcie (0,0)	(0.0, 0.0) – środek okręgu	Dowolny punkt na okręgu	**(0.86195,0.50698) / StackOverflow Exception**

Podsumowanie testów poprawności

Nie wykryto błędów logicznych działania algorytmów. Jednak okazało się, że oba algorytmy nie dają takich samych wyników, a przy dużych zbiorach punktów (około 100000), w algorytmie opartym na KD-drzewie występuje przepełnienie stosu (StackOverflow Exception).

Wydajność

Czas wykonania algorytmu metodą BruteForce jest czasem 10000 operacji znajdowania najbliższego sąsiada podzielonym przez 10000.

Czas wykonania algorytmu na KD-drzewie jest sumą czasu pojedynczej konstrukcji KD-drzewa oraz 10000 operacji znajdowania najbliższego sąsiada podzieloną przez 10000.

Przykładowe wyniki Bruteforce vs. KDTree

Zbiór danych	Ilość wierzchołków	Czas wykonania (BF)	Czas wykonania (KD)
circle_10	10	2.01e-07 s	3.95e-07 s
circle	10^2	1.26e-06 s	3.13e-06 s
circle_1000	10^3	6.66e-06 s	1.68e-05 s
circle_10000	10^4	6.10e-05 s	2.61e-04 s
circle_50000	5*10^4	3.06e-04 s	2.20e-03 s
circle_100000	10^5	6.13e-04 s	StackOverflow Exception
circle_1000000	10^6	6.20e-03 s	StackOverflow Exception
liss	10^2	1.22e-06 s	3,46e-07 s
liss_10000	10^4	6.03e-05 s	1.14e-06 s

Za punkt odniesienia wybrano punkt (0.0, 0.0)

Wyniki mają charakter poglądowy. Czasy wykonań odpowiednich testów różnią się od siebie w stopniu znaczącym.

Sprzęt pomiarowy: Laptop Lenovo G-580, system Ubuntu 12.10, procesor Intel Core-i3 4x2.40GHz, 8 GB RAM.