Projekt ŻÓŁW - dokumentacja końcowa

Artur Krawczyński, Krzysztof Kicun (zespół 14)

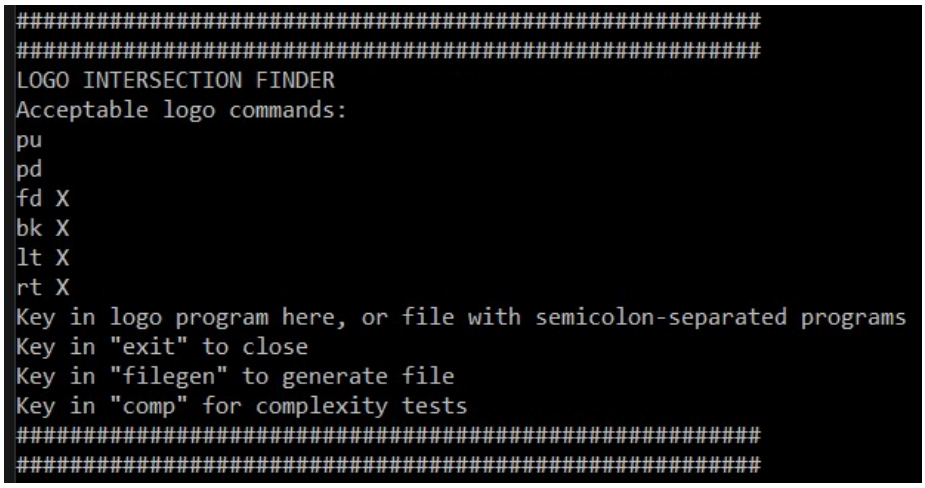
Maj 2021

Spis treści

1 Zmiany w stosunku do dokumentacji wstępnej

Jedyną zmianą w stosunku do dokumentacji wstępnej jest użyta w miotle i kolejce wydarzeń struktura danych. Zakładaliśmy użycie wbudowanej w .NET struktury *SortedDictionary*, jednak okazało się, że nie pozwala ona na wyszukanie elementu następnego lub poprzedniego w czasie krótszym niż $O(n)$. Zatem użycie tej struktury pogorszyło by złożoność czasową naszej implementacji.
W projekcie użyliśmy więc samodzielnie zaimplementowanego zrównoważonego binarnego drzewa poszukiwań AVL. Jako że sami tworzyliśmy tę strukturę, mogliśmy dodać do niej konieczne operacje wyszukania elementu następnego i poprzedniego. Operacje te, w naszej implementacji, korzystają z własności binarnego drzewa poszukiwań i mają wymaganą przez algorytm złożoność czasową $O(log(n))$

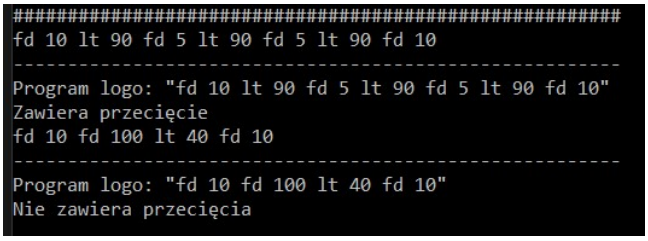
2 Opis programu dla użytkownika



Rysunek 1: Interfejs aplikacji

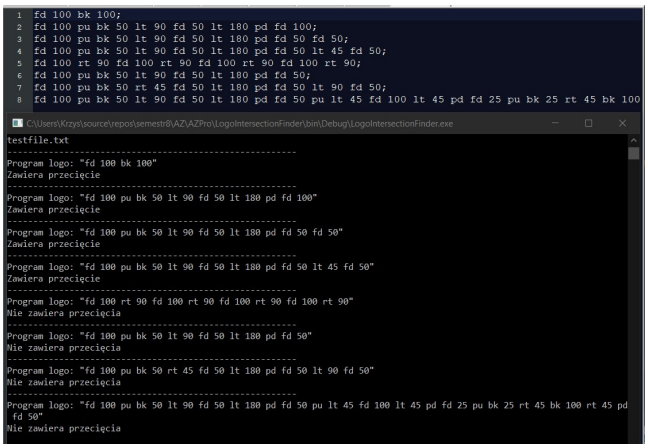
Interfejs programu jest konsolowy i bardzo prosty. Wszystko, co jest konieczne do użycia, jest wypisywane na ekran po uruchomieniu programu. Funkcjonalności to:

- Wykonanie algorytmu dla podanego bezpośrednio programu logo. Po wpisaniu dowolnego programu logo otrzymamy odpowiedź, czy zawiera on przecięcie, czy nie.



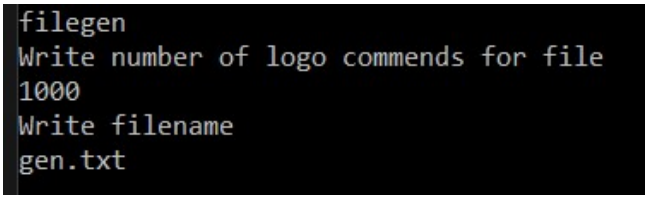
Rysunek 2: Wpisywanie bezpośrednio programów logo

- Wykonanie algorytmu dla programów logo umieszczonych w pliku tekstowym. Po wpisaniu nazwy pliku aplikacja wyszukuje go w obecnym katalogu. Plik może zawierać dowolną ilość programów logo oddzielonych średnikami.



Rysunek 3: Plik *testfile.txt* z programami logo i jego użycie

- Generowanie plików z programem logo o zadanej ilości komend



Rysunek 4: Generowanie pliku *gen.txt* z 1000 komend logo

- Przeprowadzenie testów złożoności. Po wpisaniu komendy *comp* program wykonuje testy omówione w kolejnym rozdziale i zapisuje czasy pracy w pliku csv.
- Aplikacja, zgodnie z wymaganiami, loguje swoją pracę do pliku *logs.txt*, znajdującego się w obecnym katalogu.
- Aplikacja kończy pracę, po wpisaniu komendy *exit*

3 Opis testów oraz wnioski

3.1 Testy

Aplikacja została sprawdzona dla wszystkich przypadków umieszczonych w dokumentacji wstępnej i zwróciła zakładane wyniki. Przeprowadzone zostały również testy, których zadaniem było zweryfikowanie złożoności czasowej zaimplementowanego algorytmu. W tym celu uruchamialiśmy algorytm dla 4 różnych

wzorów, dla różnych rozmiarów (od 1000 do 60000 komend). Ponieważ te wzory nie zawierają przecięć, dodaliśmy dla każdego wzoru odpowiadający mu wzór, ale z dodanymi po prawej stronie dwoma przecinającymi się odcinkami. Miało to na celu sprawdzenie poprawności implementacji, analizę wyników oraz czasu pracy. Dodatkowo wywołaliśmy również algorytm, dla tych samych rozmiarów, dla programu logo generowanego losowo. Zatem, w sumie, testy złożoności zostały przeprowadzone dla 9 różnych wzorów w różnych rozmiarach. Testy były przeprowadzane przy użyciu procesora Intel(R) Core(TM) i7-7700HQ CPU @ 2.8 GHz i 16 GB pamięci RAM

Wzory:

Rysunek 5: Wzór horyzontalny

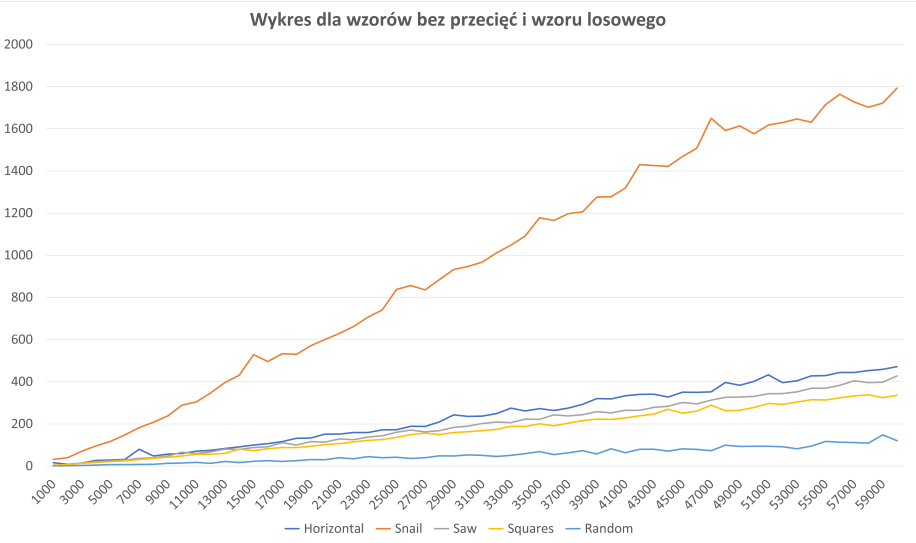
Rysunek 6: Wzór ślimak

Rysunek 7: Wzór piła

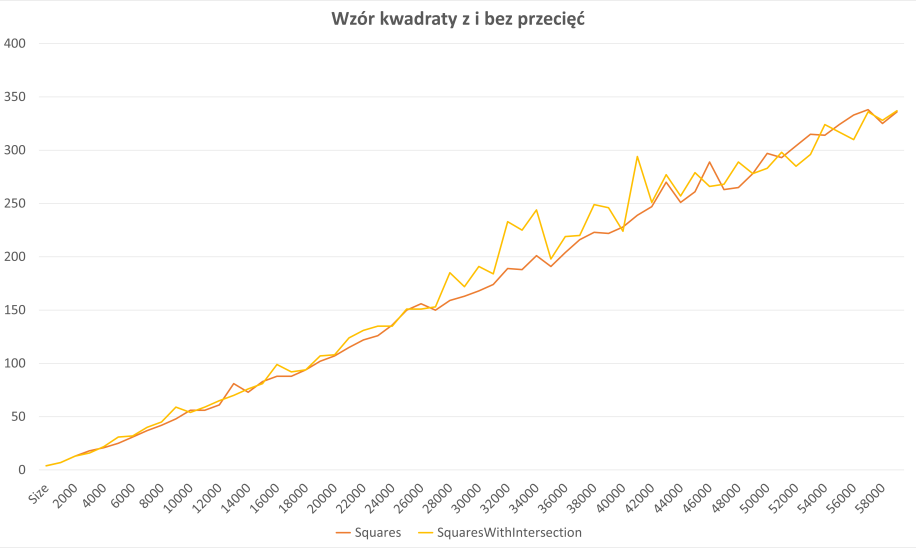
Rysunek 8: Wzór kwadraty

Rysunek 9: Przykładowy wzór z dodanym przecięciem

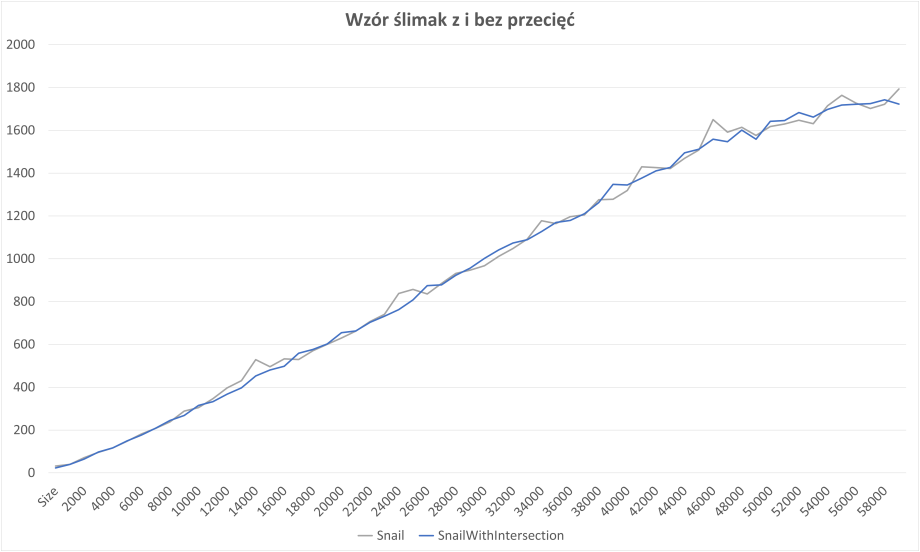
Wyniki:
Uzyskane wyniki (czasy pracy algorytmu w milisekundach) przedstawiamy na wykresach:



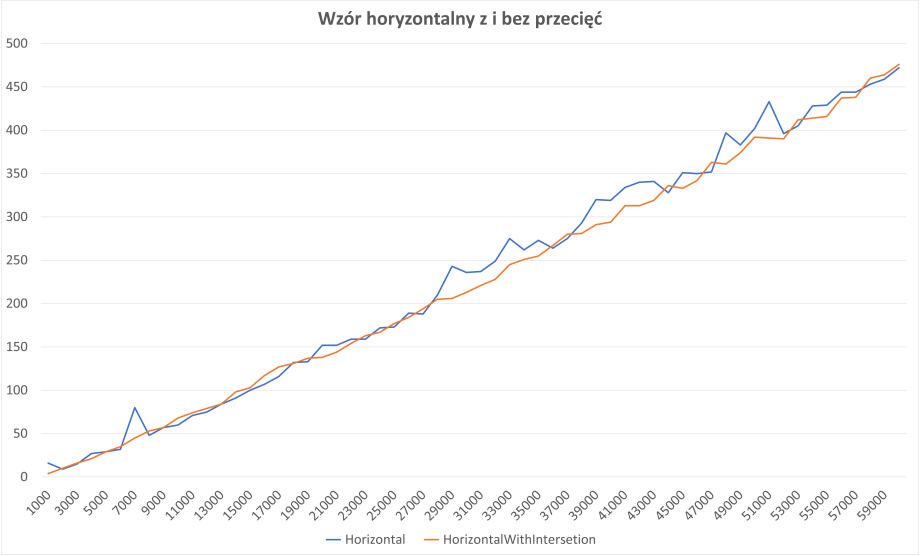
Rysunek 10: Czasy pracy dla wzorów podstawowych (bez przecięć) i wzoru losowego w milisekundach, w zależności od wielkości zadania



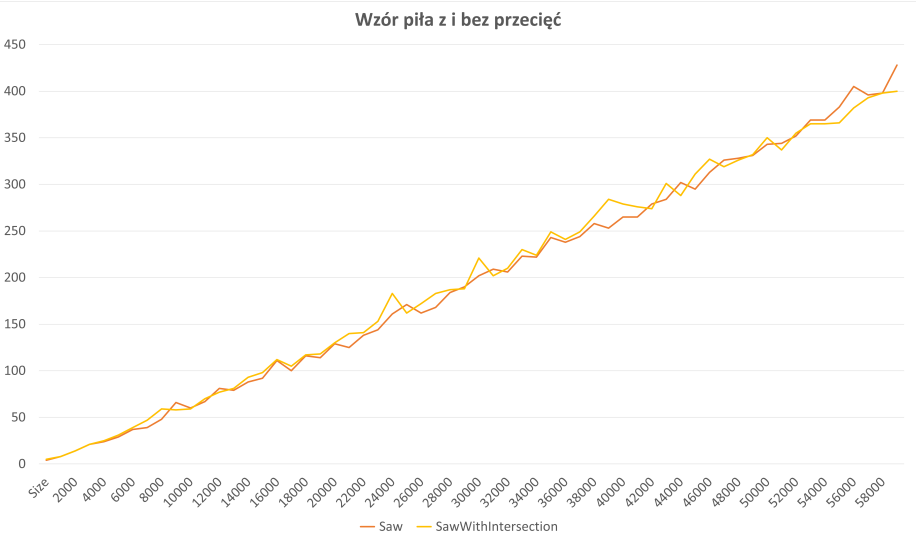
Rysunek 11: Czasy pracy dla wzoru kwadratów i wzoru kwadratów z przecięciem w milisekundach, w zależności od wielkości zadania



Rysunek 12: Czasy pracy dla wzoru ślimak i wzoru ślimak z przecięciem w milisekundach, w zależności od wielkości zadania



Rysunek 13: Czasy pracy dla wzoru horyzontalnego i wzoru horyzontalnego z przecięciem w milisekundach, w zależności od wielkości zadania



Rysunek 14: Czasy pracy dla wzoru piły i wzoru piły z przecięciem w milisekundach, w zależności od wielkości zadania

3.2 Wnioski

- Z kształtu wykresów (z czasów wykonania) widzimy, że nasz algorytm ma zakładaną złożoność czasową, czyli $O(n * log(n))$
- Czasy wykonań dla wzorów bez przecięć i ich odpowiedników z przecięciami dodanymi z prawej strony są zbliżone. Jest to przesłanka o prawidłowym działaniu algorytmu, bowiem dodane przecięcie jest rozpatrywane przy końcu pracy, zatem nie powinno mieć wpływu na czas wykonania
- Zawsze najszybciej wykonuje się wzór losowy. W związku z losowością wzoru, pojawiają się w nim przecięcia, które powodują zakończenie obliczeń przed przeanalizowaniem wszystkich odcinków, co znacząco skraca czas pracy
- Wyraźnie najwięcej czasu wymagał od algorytmu wzór ślimaka. Jest to związane z tym, że w trakcie jego pracy struktura miotły zawiera więcej odcinków niż dla pozostałych wzorów, a w związku z tym operacje na miotle zajmują więcej czasu
- Uzyskanie zakładanych wyników w testach przypadków szczególnych jest przesłanką o prawidłowym działaniu algorytmu

4 Podział prac

- Testy przypadków szczególnych - Artur Krawczyński

- Testy złożoności - Artur Krawczyński, Krzysztof Kicun
- Opisy i wnioski z testów - Krzysztof Kicun
- Implementacja algorytmu - Artur Krawczyński, Krzysztof Kicun
- Implementacja interfejsu, generowania i wczytywania plików - Artur Krawczyński
- Implementacja drzewa AVL - Krzysztof Kicun