Sprawozdanie końcowe
Projekt „LUPA”
Algorytmy i struktury danych

Mateusz Smoliński Łukasz Knigawka

17 stycznia 2019

Spis treści

1 Wstęp

Sprawozdanie powstało w celu podsumowania projektu „LUPA". Zawiera ono opis zastosowanego rozwiązania i algorytmu rozwiązującego problem podziału mapy na obszary, jak i pomysłów i koncepcji testowanych we wcześniejszych fazach projektu. Opisane zostały także zmiany względem specyfikacji funkcjonalnej i implementacyjnej, które zostały podjęte już w trakcie implementacji projektu. Na końcu dokument zawiera wnioski wyciągnięte w trakcie i po ukończeniu projektu, a także podsumowanie zajęć laboratoryjnych i subiektywna opinia co do tematów zadań.

Projekt „Lupa" ma za zadanie wczytanie z pliku tekstowego mapy, podanej w postaci punktów konturu i punktów kluczowych, a następnie wydzielenie obszarów na mapie na podstawie tych punktów kluczowych. W pliku otrzymujemy także deklarację obiektów, jakie mają być umieszczone na mapie, żeby umożliwić podsumowanie obiektów na danych obszarach. Program ma umożliwiać także dodawanie i usuwanie punktów kluczowych i konturowych w trakcie trwania programu.

Plik tekstowy z danymi przyjmuje następującą postać:

```
# Kontury terenu (wymienione w kolejnosci laczenia): Lp. x y
1. 0 0
2. 0 20
3. 20 30.5
4. 40 20
5. 40 0

# Punkty kluczowe: Lp. x y Nazwa
1. 1 1 KOK Krawczyka
2. 1 19 KOK Kaczmarskiego
3. 30 10 KOK Lazarewicza

# Definicje obiektow: Lp. Typ_obiektu (Nazwa_zmiennej Typ_zmiennej)*
1. SZKOLA Nazwa String X double Y double
2. DOM X double Y double L_MIESZKANCOW int
3. NIEDZWIEDZ X double Y double

# Obiekty: Typ_obiektu (zgodnie z definicja)
1. SZKOLA "Szkola robienia duzych pieniedzy" 4 1
2. DOM 4 3 100
3. DOM 4 17 120
4. DOM 4 18 80
5. NIEDZWIEDZ 20 20
6. NIEDZWIEDZ 40 1
7. NIEDZWIEDZ 39 1
8. NIEDZWIEDZ 39 2
```

Program ma postać aplikacji z graficznym interfejsem użytkownika, co umożliwia wypełnienie wszystkich założeń projektowych. Został napisany w języku C#, na sprzęcie komputerowym zgodnym z tym zadeklarowanym w specyfikacji implementacyjnej.

2 Opis algorytmu

Poszukiwanie algorytmu dzielącego mapę na obszary okazało się największym wyzwaniem w toku realizacji projektu. Można je podzielić na trzy części:

- próby wykorzystania algorytmu Fortune'a,
- próby napisania własnego algorytmu opartego na podziale symetralnych,
- próby rozwiązania problemu na podstawie analizy pikseli.

Pierwsza część, czyli eksperymenty z algorytmem Fortune'a, była pierwszym założeniem, przyjętym jeszcze w fazie planowania projektu. Algorytm ten został wybrany ze względu na najlepszą złożoność czasową $O(nlogn)$. Algorytm okazał się jednak dużo bardziej złożony niż początkowo zakładaliśmy, a jego samodzielna implementacja byłaby trudna i czasochłonna. Stąd też pojawił się pomysł rozwiązania problemu algorytmem alternatywnym, napisanym przez nas na potrzeby tego konkretnego projektu.

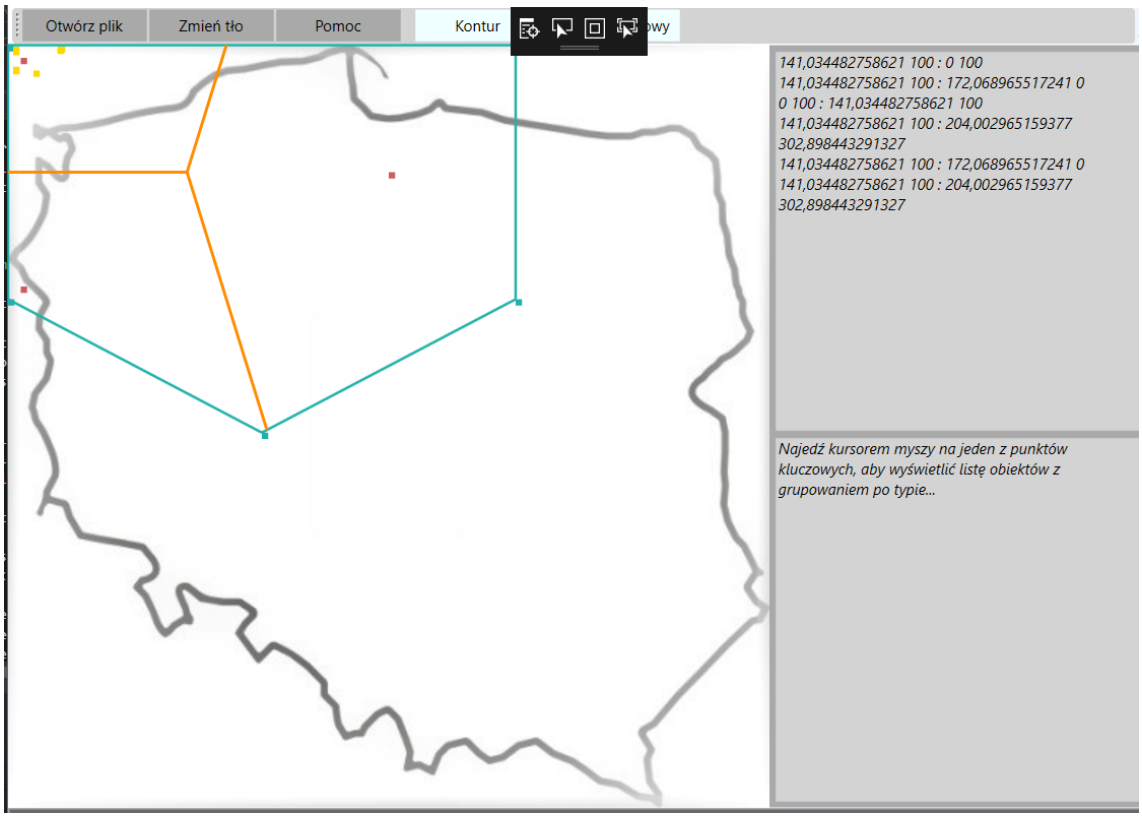
Kolejny algorytm opierał się na własnościach diagramu Woronoja – w szczególności na tym, że dzieli on planszę na figury wypukłe. Logika algorytmu opierała się na analizie planszy z perspektywy pojedynczych punktów kluczowych, żeby wyznaczyć krawędzie ograniczające obszar wyznaczany przez dany punkt kluczowy. Analiza rozpoczynała się od matematycznego wyznaczenia najbliższej symetralnej spośród symetralnych pomiędzy tym punktem kluczowym a wszystkimi pozostałymi. Następnie program miał analizować wszystkie punkty przecięcia pozostałych symetralnych z wyznaczoną i wybierać najbliższe punkty, powtarzając tą czynność dla kolejnych symetralnych przecinających poprzednie aż do utworzenia zamkniętej figury lub napotkania konturu z obu stron. Ten algorytm jednakże również okazał się trudny do zaimplementowania i dopiero po kilku dniach możliwe było zobaczenie pierwszych rezultatów. Algorytm okazał się jednak nieskuteczny, czy to z powodu błędnej jego implementacji czy błędnych założeń matematycznych (algorytm nie przewidywał kilku przypadków) i choć działał poprawnie dla danych przykładowych, to przy większych planszach powodował błędy nie tylko w rysowaniu granic obszarów, ale także prowadząc do zapętlenia programu z przyczyn, których nie udało się ustalić.

Ostatnie podejście zakładało rozstrzygnięcie problemu najprostszą metodą, bardzo kosztowną czasowo dla programu, jednak nie wymagającą długiej implementacji. Tym razem rysowanie miało się odbywać na podstawie odległości dzielące punkty (piksele) na mapie, i na podstawie matematycznie wyznaczanej odległości od punktów kluczowych program miał przypisywać kolory do każdego punktu na mapie tym samym wyznaczając obszary w obrębie konturu. Ta metoda okazała się skuteczna i została zastosowana w ostatecznej wersji programu.

3 Działanie programu

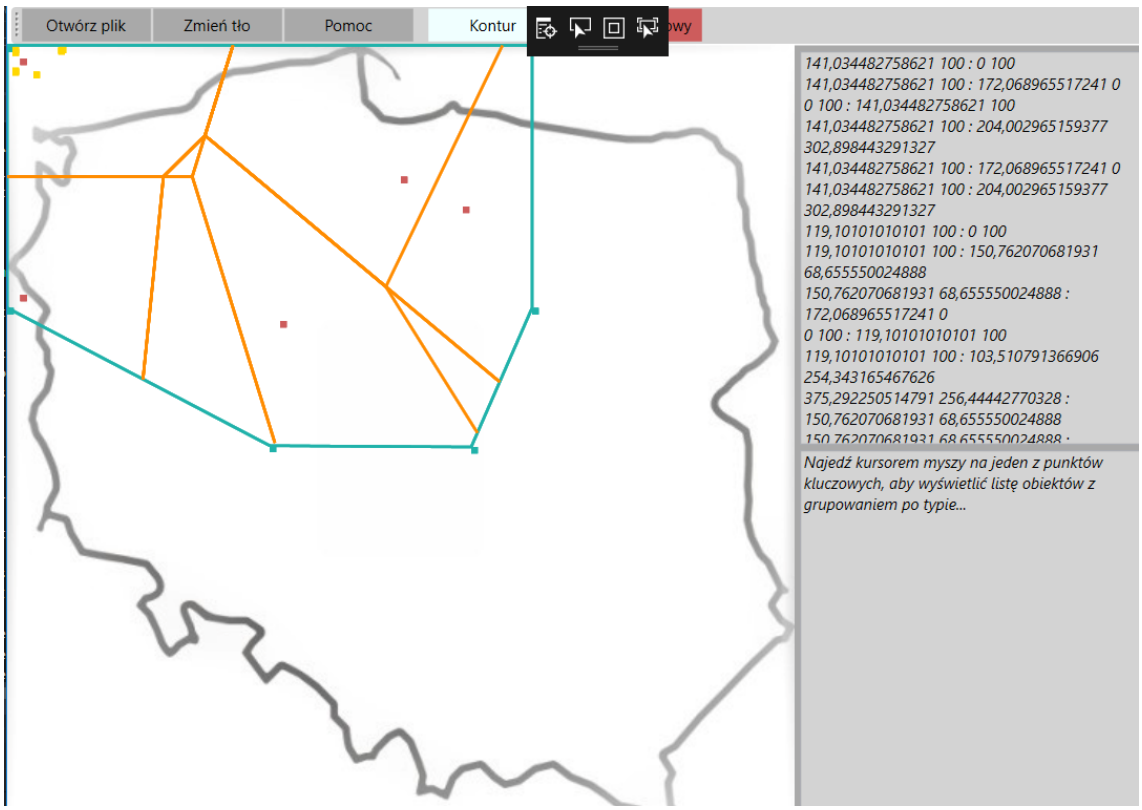
3.1 Działanie pierwszego algorytmu

Na rysunku 1. widać przykład działania programu jeszcze w czasie prób wykorzystywania pierwszego algorytmu. Zdjęcie wykazuje, że algorytm sprawdzał się w prostych przypadkach i rysował poprawny diagram Woronoja ograniczony konturami. Panel po lewej został wykorzystany do wypisania odcinków, po których przechodzi algorytm.



Rysunek 1: Uruchomienie programu z przykładowymi danymi, stary algorytm

Kolejny obrazek ukazuje, że algorytm przy dodawaniu kolejnych punktów zaczął powodować błędy.

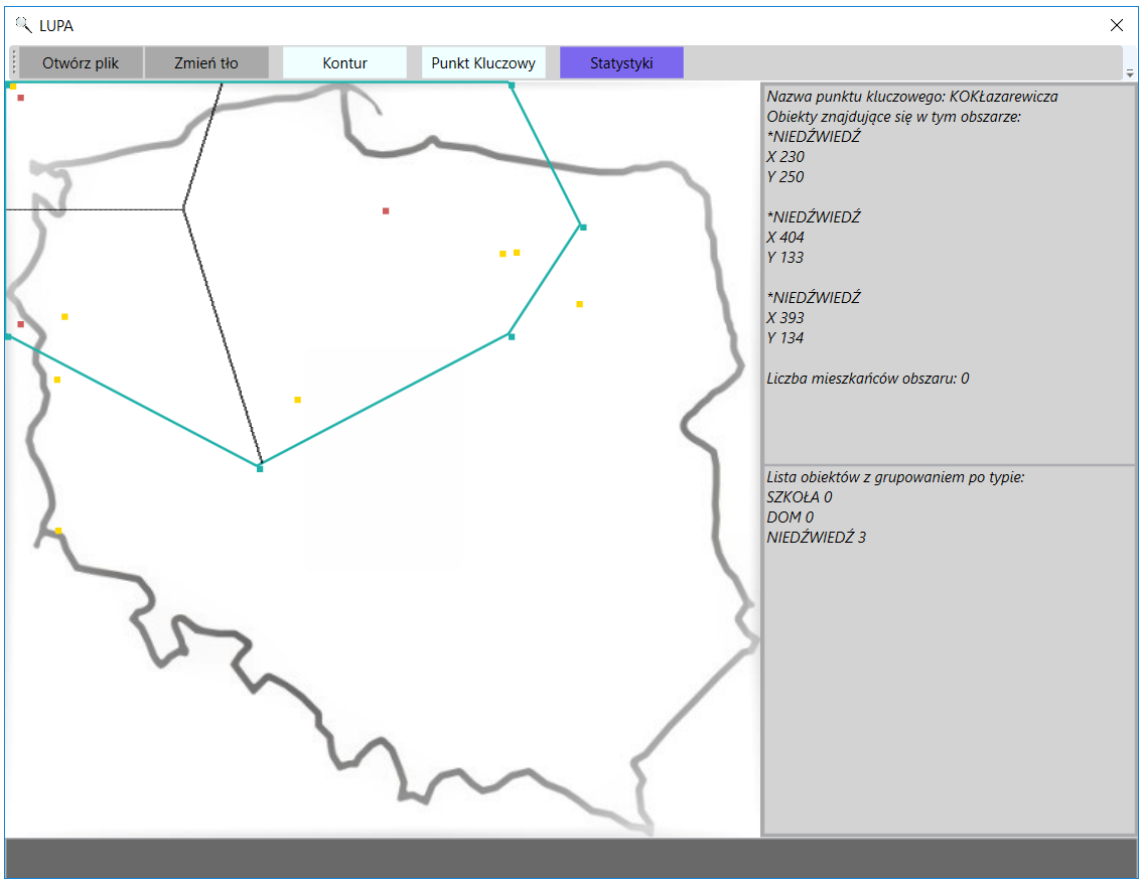


Rysunek 2: Uruchomienie programu z zmienionymi danymi, stary algorytm

Ze względu na dużą ilość zróżnicowanych błędów, jakie pojawiały się przy kolejnych próbach poprawy algorytmu zrezygnowaliśmy z tego podejścia do problemu. Kolejne grafiki przedstawiać będą ostateczną wersję programu.

3.2 Działanie finalnej wersji programu

Udało się zrealizować wszystkie założone funkcje programu. Możliwe jest wczytanie konfiguracji mapy z pliku wejściowego, a także modyfikowanie jej wyglądu podczas działania programu. Wyświetlone mogą zostać także statystyki dotyczące wybranego obszaru.



Rysunek 3: Uruchomienie programu w wersji odpowiadającej wersji aktualnej na dzień obrony projektu. W uchwyconym na zdjęciu ekranu momencie, statystyki wyświetlane w polach tekstowych znajdujących się po prawej stronie okna głównego programu dotyczą obszaru położonego najbardziej na prawo.

4 Zmiany względem specyfikacji funkcjonalnej

4.1 Graficzny interfejs użytkownika

Wygląd programu nie uległ dużej zmianie względem wstępnych projektów. Punkty kluczowe, kontur i obiekty uzyskały odmienne kolory, by ułatwić ich odróżnianie i poprawić ogólny efekt wizualny projektu.

4.2 Sytuacje wyjątkowe

Wiele wyjątków związanych z tekstowym plikiem wejściowym zostało zinterpretowane dużo dokładniej, niż początkowo zakładano. System wyjątków zastosowany w implementacji *Parsera* pozwolił nam na precyzyjne wypisywanie komunikatem z informacją o liniach, w których wystąpiły błędy. Wyjątki wewnętrzne, dotyczące pojedynczych linii (nazwane w programie *ParseLineException*) przekazują do głównej metody *Parsera* informację, jakiego rodzaju błąd nastąpił. Jeśli uniemożliwia on kontynuowanie programu, wyjątek zostaje przekazany do funkcji wywołującej wczytywanie pliku z klasy *MainWindow* w postaci wyjątku *ParseFileException* i wypisywany na ekranie, wraz z precyzyjną informacją. Większość wyjątków jednak umożliwia kontynuację działania programu przy ominięciu błędnego punktu lub obiektu – w takim wypadku wyjątek zostaje jedynie dopisany do listy napisów nazwanej *feedback* i wyświetlony w programie w charakterze ostrzeżenia.

W specyfikacji funkcjonalnej pominięte zostało kilka oczywistych błędów, takich jak niedobór (lub nadmiar) argumentów przy deklaracji instancji obiektów czy też brak współrzędnych przy wczytywaniu dowolnych punktów. Wszystkie takie niestandardowe kombinacje argumentów zostały uwzględnione w ostatecznej wersji projektu i w przypadku ich wystąpienia użytkownik zostanie stosownie poinformowany przez program.

5 Zmiany względem specyfikacji implementacyjnej

Struktura programu w ostatecznej wersji nie korzysta z zakładanej wcześniej klasy *AreaDivider*. Pozostała ona w projekcie w celu pokazania wcześniejszej koncepcji rozwiązania problemu autorskim algorytmem, jednak ostateczna logika dzielenia na obszary znalazła się w klasie głównej *MainWindow* w celu uproszczenia interakcji z rzeczywistą planszą w programie.

5.1 Klasy Mathematics i StraightLine

To są zupełnie nowe klasy, które nie wystąpiły w specyfikacji implementacyjnej. Pierwsza z nich pełni funkcję wewnętrznej biblioteki matematycznej programu. Wykonuje obliczenia, takie jak obliczanie dystansu pomiędzy punktami, sprawdza czy linie się przecinają albo czy punkt znajduje się wewnątrz konturu. Wszystkie operacje wykonywane są statycznie, metody przyjmują zarówno obiekty klas systemowych jak i tych stworzonych na potrzeby tego programu.

Druga z tych klas jest abstrakcyjnym przedstawieniem prostej i jest wykorzystywana właśnie do wyżej wspomnianych kalkulacji matematycznych.

5.2 Klasa Parser

Klasa wczytująca dane z pliku została rozbudowana i ostatecznie składa się z większej liczby metod, niż zakładała specyfikacja implementacyjna. Do nowych metod należą:

- `private static void VerifyObjectsDef (List<CustomObjectType> customObjectTypes)`

  `private static void VerifyContourPoints (List<Point> contourPoints)`

- `private static void VerifyKeyPoints (List<Point> contour, List<Point> keyPoints)`

Ich zadaniem jest weryfikacja argumentów po zakończeniu wczytywania danej części pliku – na przykład, gdy cały kontur zostanie wczytany, metoda sprawdzi, czy ma on potrzebną liczbę punktów oraz linie tworzone przez te punkty nie przecinają się. Punkty konturowe są sprawdzane pod kątem umieszczenia ich poza granicami konturu korzystając ze stosownej funkcji matematycznej z klasy opisanej wyżej, a definicje obiektów – czy są w nich zawarte trzy obowiązkowe typy obiektów (szkoła, dom i niedźwiedź).

5.3 Klasy przechowujące dane

W tej grupie klas nastąpiły najmniejsze zmiany. Wynikały one głównie z przyjętych sposobów implementacji algorytmu oraz ułatwiły porównywanie niektórych obiektów.

5.3.1 Klasa Point

W tej klasie nastąpiła zmiana typu podstawowych zmiennych X i Y na typ całkowity. Ma to znaczenie przy graficznej reprezentacji tych punktów na mapie złożonej z pojedynczych pikseli, ułatwiając tym samym wdrożenie algorytmu dzielącego na obszary.

5.3.2 Klasa LineSegment

Ta klasa zmieniła nazwę na *LineSegment* z dwóch powodów:

- żeby uniknąć konfliktów z nazwami używanymi lokalnie przy generowaniu reprezentacji graficznej naszej mapy,
- żeby podkreślić jej skończoną reprezentację w programie, odróżniając między innymi od abstrakcyjnej prostej wykorzystywanej w matematyce.

Nie jest to jedyna zmiana wykonana w tej klasie. Zyskała ona jeszcze jedną metodę:

- `public bool IsIntersecting(LineSegment secondLine)`

Zadaniem tej metody jest sprawdzenie, czy dany odcinek przecina się z innym odcinkiem, do którego referencja jest podana jako parametr. Metoda zwraca prawdę w przypadku, gdy linie się przecinają.

5.4 Klasa MainWindow

Zgodnie z oczekiwaniami, klasa *MainWindow* zyskała wiele metod bezpośrednio powiązanych z działaniem biblioteki graficznej WPF oraz reakcją na działanie użytkownika. W jej skład wchodzą:

- liczne metody rozpoczynające się od *Draw*, odpowiadające za rysowanie elementów na ekranie,
- metody reagujące na zdarzenia myszy oraz aktywowane przez użytkownika przyciski,
- metody pomocnicze, przeprowadzające drobne obliczenia oraz porządkujące dane w obiektach je przechowujących.

6 Wnioski

1. Zagadnienie diagramu Woronoja pomimo prostych założeń nie było oczywiste do rozwiązania w programie.
2. Popularny w rozwiązaniach zagadnień związanych z tym diagramem algorytm Fortune'a mógł być najlepszym rozwiązaniem pomimo swojej matematycznej złożoności. Zastosowanie go zdecydowanie poprawiłoby wydajność programu.
3. W przeciwieństwie do projektu indywidualnego, opracowanie algorytmu okazało się kluczowym problemem w trakcie projektu i pochłonęło ono zdecydowanie najwięcej czasu.
4. Tworzenie algorytmu od podstaw było ciekawym i kształcącym doświadczeniem, zarówno z perspektywy programistycznej, jak i algorytmicznej. Okazało się ono jednak ryzykowne i w związku z niepowodzeniem wymusiło na nas szybkie przygotowanie alternatywnego rozwiązania problemu.

5. Projekt zespołowy okazał się doskonałą okazją do rozwijania umiejętności korzystania z systemu kontroli wersji. Git posiada wiele narzędzi, które pomogły rozwiązać nam konflikty, przydatne okazało się także cofanie do wcześniejszych wersji kodu.

7 Komentarz do zajęć laboratoryjnych

Laboratorium składało się z trzech części. Pierwsza z nich to ćwiczenia bezpośrednio związane z implementacją pojedynczych algorytmów w języku Java. Uważamy, że były one rozwijające i całkiem interesujące, adekwatne do naszego kierunku studiów. Jako uwagę do tej części można zauważyć sporą rozbieżność tematów z wykładu i laboratorium – zagadnienia z ćwiczeń były omawiane na wykładzie dopiero 2-3 tygodnie po zajęciach laboratoryjnych, co wymagało od nas więcej pracy samodzielnej i uniemożliwiało zweryfikowanie informacji, z których przygotowywaliśmy się na zajęcia. Na przykład, wykład o kopcu jako strukturze danych znacznie zmienił nasze podejście względem tego, czego nauczyliśmy się samodzielnie i być może gdyby ten wykład odbył się wcześniej mielibyśmy szansę na uzyskanie lepszej noty z tego ćwiczenia.

Kolejną częścią był projekt indywidualny, który postawił nas przed zagadnieniem wyszukiwania optymalnej wymiany walut. Uważamy, że był to najciekawszy etap zajęć i był on najlepiej zrównoważony pod względem czasu spędzonego na wykonywaniu zadań oraz wymaganych umiejętności do wykonania ćwiczenia. Kolejnym dużym plusem tej części była możliwość wyboru języka programowania, w którym projekt był wykonywany.

Ostatni etap to projekt grupowy, który postawił nas przed trudniejszym zagadnieniem, jakim był podział planszy ograniczonej konturami na obszary i przypisanie obiektów do obszarów. Pełna ocena tego etapu jest dla nas trudna z uwagi na to, że nasza praca nie została jeszcze zweryfikowana i oceniona. Trudnością, na jaką napotkaliśmy była implementacja algorytmu dzielącego – próby z algorytmem Fortune'a oraz algorytmem autorskim zakończyły się niepowodzeniem, co pochłonęło dużo naszego czasu i ostatecznie spowodowało wybór algorytmu dużo bardziej czasochłonnego niż dwa wcześniejsze, ale prostszego do napisania.