

Uniwersytet Jagielloński w Krakowie

Wydział Fizyki, Astronomii i Informatyki Stosowanej

Sandra Rudnicka

Nr albumu: 1143863

**Badanie grafów zewnętrznie planarnych
z językiem Python**

Praca licencjacka na kierunku Informatyka

Praca wykonana pod kierunkiem
dra hab. Andrzeja Kapanowskiego
Instytut Informatyki Stosowanej

Kraków 2021

Oświadczenie autora pracy

Świadom odpowiedzialności prawnej oświadczam, że niniejsza praca dyplomowa została napisana przeze mnie samodzielnie i nie zawiera treści uzyskanych w sposób niezgodny z obowiązującymi przepisami.

Oświadczam również, że przedstawiona praca nie była wcześniej przedmiotem procedur związanych z uzyskaniem tytułu zawodowego w wyższej uczelni.

Kraków, dnia

Podpis autora pracy

Oświadczenie kierującego pracą

Potwierdzam, że niniejsza praca została przygotowana pod moim kierunkiem i kwalifikuje się do przedstawienia jej w postępowaniu o nadanie tytułu zawodowego.

Kraków, dnia

Podpis kierującego pracą

*Składam serdeczne podziękowania Promotorowi
Panu dr hab. Andrzejowi Kapanowskiemu za wy-
rozumiałość, poświęcony czas, oraz pomoc w reali-
zacji pracy licencjackiej.*

Streszczenie

W pracy zostały przedstawione niezbędne zagadnienia z teorii grafów pozwalające na zdefiniowanie grafów zewnętrznie planarnych oraz ich właściwości. Graf zewnętrznie planarny jest to graf, który można narysować na płaszczyźnie tak, aby wszystkiego jego wierzchołki leżały na jego ścianie zewnętrznej, a krawędzie nie przecinały się.

W dalszej części zostały zaimplementowane wybrane algorytmy dotyczące tej klasy grafów, to jest algorytm rozpoznawania grafów zewnętrznie planarnych, kolorowania wierzchołków, podziału dowolnego grafu na dwuspójne składowe, sprawdzania czy graf zewnętrznie planarny jest hamiltonowski, wyznaczenie długości najdłuższego cyklu, oraz generowanie maksymalnych grafów zewnętrznie planarnych.

Algorytmy zostały zaimplementowane przy użyciu języka Python oraz biblioteki graphtheory. Stworzony kod działa w Pythonie 2.7 i Pythonie 3.9, co zapewnia jego uniwersalność. Poprawność implementacji algorytmów sprawdzono poprzez testy jednostkowe z wykorzystaniem modułu unittest. Sprawdzono również rzeczywistą wydajność wszystkich algorytmów z użyciem modułu timeit. Testy potwierdziły zgodność z przewidywaniami teoretycznymi.

Słowa kluczowe: grafy zewnętrznie planarne, grafy planarne, kolorowanie grafów, cykl Hamiltona, grafy dwudzielne

English title: Study of outerplanar graphs with Python

Abstract

This work presents the necessary issues from the graph theory, which allow us to define outerplanar graphs with their properties. An outerplanar graph is a graph that has a planar drawing such that all vertices belong to the outer face and no edges cross each other.

In the following part, selected algorithms are implemented for this class of graphs. There is the recognition algorithm, finding biconnected components, recognition of hamiltonian graphs in the class of outerplanar graphs, finding the longest cycle, optimal vertex coloring algorithm, and maximal outerplanar graphs generator.

Python implementation of the algorithms is included, where the graphtheory package is used. Code may be executed in Python 2.7 and Python 3.7, so it is universal. Correctness of algorithms was checked using unit testing with the unittest module. The real computational complexity of all algorithms was also checked by means of the timeit module.

Keywords: outerplanar graphs, planar graphs, graph coloring, Hamiltonian cycle, bipartite graphs

Spis treści

Spis tabel	3
Spis rysunków	4
Listings	5
1. Wstęp	6
1.1. Cele pracy	6
1.2. Organizacja pracy	7
2. Teoria grafów	8
2.1. Grafy - podstawowe zagadnienia	8
2.2. Grafy dwuspójne	12
2.3. Grafy planarne	13
2.4. Grafy zewnętrznie planarne	15
3. Implementacja grafów	19
3.1. Reprezentacje grafów	19
3.2. Reprezentacja grafu w bibliotece graphtheory	20
3.3. Klasa Edge	20
3.4. Klasa Graph	21
3.5. Klasa BipartiteGraphBFS	22
4. Algorytmy	23
4.1. Rozpoznawanie grafów zewnętrznie planarnych	23
4.2. Algorytm podziału grafu na składowe dwuspójne	26
4.3. Cykl Hamiltona i najdłuższy cykl w grafie	28
4.4. Kolorowanie grafów zewnętrznie planarnych	29
4.5. Rozpoznawanie maksymalnych grafów zewnętrznie planarnych	30
4.6. Algorytm generowania maksymalnych grafów zewnętrznie planarnych	32
5. Podsumowanie	34
A. Testy algorytmów	35
A.1. Poprawność	35
A.2. Złożoność obliczeniowa	36
Bibliografia	40

Spis tabel

2.1. Liczba grafów zewnętrznie planarnych.	18
--	----

Spis rysunków

2.1.	Graf nieskierowany i skierowany.	8
2.2.	Graf nieskierowany z zaznaczoną ścieżką (z lewej) i cyklem (z prawej).	9
2.3.	Przykład grafu pełnego K_4	9
2.4.	Przykład grafu dwudzielnego.	10
2.5.	Przykład grafu pełnego dwudzielnego $K_{2,3}$	10
2.6.	Przykład grafów izomorficznych.	10
2.7.	Przykład grafów homeomorficznych.	11
2.8.	Przykład grafów ściągalnych.	11
2.9.	Przykład grafu 3-regularnego (graf K_4).	11
2.10.	Przykład grafu cyklicznego C_4	12
2.11.	Przykład grafu liniowego P_4	12
2.12.	Przykład grafu dwuspójnego.	12
2.13.	Podział przykładowego grafu spójnego na dwuspójne składowe.	13
2.14.	Przykład grafu planarnego (graf K_4).	13
2.15.	Przykład grafu zewnętrznie planarnego.	15
3.1.	Przykładowy graf z wagami.	19
A.1.	Wydażność algorytmu rozpoznawania grafów zewnętrznie planarnych.	37
A.2.	Wydażność algorytmu podziału grafu na dwuspójne składowe.	37
A.3.	Wydażność algorytmu wyznaczania najdłuższego cyklu.	38
A.4.	Wydażność algorytmu kolorowania wierzchołków.	38
A.5.	Wydażność algorytmu rozpoznawania maksymalnych grafów zewnętrznie planarnych.	39
A.6.	Wydażność algorytmu generowania maksymalnych grafów zewnętrznie planarnych.	39

Listings

4.1	Moduł outerplanar.	24
4.2	Moduł bicomponent.	26
4.3	Moduł longest_cycle.	28
4.4	Moduł coloring.	29
4.5	Moduł maximal_outerplanar.	31
4.6	Moduł generate_max_outerplanar.	33

1. Wstęp

Tematem niniejszej pracy są grafy zewnętrznie planarne. Jest to szczególna klasa grafów, dla której można rozwiązać wiele problemów NP-zupełnych w czasie wielomianowym. Przede wszystkim grafy te były jednymi z pierwszych, dla których zauważono możliwość wydajnego rozwiązania tych problemów.

Krótko można powiedzieć, że grafy zewnętrznie planarne to takie grafy, które można narysować na płaszczyźnie bez przecinania się krawędzi, a dodatkowo wszystkie wierzchołki grafu należą do ściany zewnętrznej [1]. Grafy te były po raz pierwszy badane i nazwane w pracy Chartranda i Harary'ego (1967) [2].

W dalszych rozdziałach zostaną przedstawione podstawowe zagadnienia dotyczące teorii grafów, jak również pojęcie zewnętrznej planarności, wraz z przydatnymi własnościami. Na koniec zostaną zaprezentowane wybrane algorytmy dotyczące rozpoznawania, kolorowania wierzchołków i wyznaczania najdłuższej ścieżki dla tych grafów, oraz testowania i generowania maksymalnych grafów zewnętrznie planarnych.

1.1. Cele pracy

Głównym celem pracy jest przedstawienie pojęcia zewnętrznej planarności grafów oraz implementacja efektywnych algorytmów do rozpoznawania i określania ich własności w języku Python. Poprzez wykorzystanie tego języka programowania algorytmy zostały przedstawione za pomocą przejrzystego kodu, co pozwala na łatwe ich zrozumienie. Ponadto Python pozwala na sprawdzenie poprawności algorytmów przy użyciu testów jednostkowych (moduł `unittest`), jak również wyznaczenie empirycznie złożoności czasowej (moduł `timeit`).

Praca może również pomóc w dalszym rozwoju czytelnika, w nauce programowania, oraz myślenia algorytmicznego. Zawiera rozwiązania, które pozwalają na implementację algorytmów w czasie liniowym, co można wykorzystać w innych problemach.

Kolejnym celem pracy było zaprezentowanie w działaniu i rozszerzenie biblioteki `graphtheory`, która jest rozwijana na Wydziale Fizyki, Astronomii i Informatyki Stosowanej w Uniwersytecie Jagiellońskim [3]. Biblioteka została napisana w języku Python i zawiera przydatne klasy do reprezentowania grafów prostych i multigrafów, oraz klasy odpowiadające algorytmom dla ogólnych grafów i dla szczególnych klas grafów, przez co wiele problemów można rozwiązać w bardziej efektywny sposób niż w przypadku ogólnych grafów.

1.2. Organizacja pracy

Rozdział 2 zawiera podstawowe zagadnienia z teorii grafów, definicję grafów dwuspójnych, planarnych i zewnętrznie planarnych. Zostały tam również zaprezentowane podstawowe własności tej klasy grafów, w szczególności analogiczne twierdzenia dotyczące planarności i zewnętrznej planarności. W następnym rozdziale 3 została zaprezentowana biblioteka `graphtheory` wraz z przykładowym kodem wykorzystującym podstawowe funkcjonalności biblioteki niezbędne do implementacji algorytmów związanych z zewnętrzną planarnością. W rozdziale 4 znajdują się algorytmy pozwalające na rozwiązanie problemów dotyczących grafów zewnętrznie planarnych, zaimplementowane w języku Python. Każdy paragraf zawiera kod źródłowy, wyjaśnienie działania algorytmu i informację dotyczącą złożoności obliczeniowej. W podsumowaniu 5 zostały przedstawione główne wyniki pracy oraz końcowe wnioski. Dodatek A zawiera wyniki testów jednostkowych, które wykonano przy użyciu modułu `unittest`. Znajdują się tam również wykresy wygenerowane w programie `Gnuplot`, przedstawiające empirycznie wyznaczoną złożoność obliczeniową zaimplementowanego kodu.

2. Teoria grafów

W tym rozdziale zostaną przybliżone podstawowe pojęcia z teorii grafów, które są niezbędne do określania zewnętrznej planarności grafów, oraz innych właściwości omawianych w dalszych rozdziałach.

2.1. Grafy - podstawowe zagadnienia

Definicja 2.1.1. *Grafem skierowanym G nazywamy graf, składający się z niepustego, skończonego zbioru wierzchołków V , oraz zbioru krawędzi E takich, że $E = \{(v, w) : v, w \in V\}$, gdzie (v, w) jest parą uporządkowaną.*

Definicja 2.1.2. *Graf nieskierowany to graf, który składa się z niepustego, skończonego zbioru wierzchołków V , oraz zbioru krawędzi E takiego, że $E = \{(v, w) : v, w \in V\}$, gdzie $(v, w) \equiv (w, v)$ jest parą nieuporządkowaną.*



Rysunek 2.1. Graf nieskierowany i skierowany.

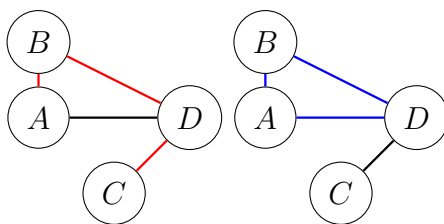
Rysunek po lewej przedstawia graf nieskierowany, natomiast po prawej graf skierowany. Powyższe definicje pozwalają na podział grafów na te dwa rodzaje. W dalszej części pracy mówiąc o grafach będziemy rozważać domyślnie grafy nieskierowane.

Definicja 2.1.3. *Pętlą nazywamy krawędź, która łączy wierzchołek z nim samym.*

Definicja 2.1.4. *Stopień wierzchołka w grafie nieskierowanym to liczba krawędzi łączących się z danym wierzchołkiem. Pętle w grafie liczymy jako dwie krawędzie dochodzące do wierzchołka.*

Definicja 2.1.5. *Graf prosty to graf nie zawierający pętli i krawędzi wielokrotnych.*

W opisie grafów planarnych i zewnętrznie planarnych będziemy zakładać, że grafy są proste. W ogólności można badać również multigrafy.



Rysunek 2.2. Graf nieskierowany z zaznaczoną ścieżką (z lewej) i cyklem (z prawej).

Definicja 2.1.6. Ścieżką od v_1 do v_m nazywamy skończony ciąg wierzchołków v_1, v_2, \dots, v_m , przy czym pomiędzy sąsiednimi wierzchołkami w ciągu istnieją krawędzie w grafie.

Definicja 2.1.7. Drogą nazywamy ścieżkę, w której każdy wierzchołek występuje nie więcej niż jeden raz.

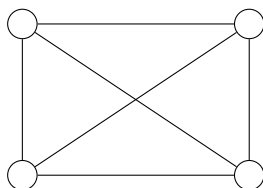
Definicja 2.1.8. Cykl to ścieżka, w której wierzchołek początkowy i końcowy jest taki sam, oraz pozostałe wierzchołki nie występują więcej niż jeden raz.

Grafy na rysunku 2.2 przedstawiają odpowiednio ścieżkę niebędącą cyklem oraz cykl. W pierwszym przypadku ścieżka może być ciągiem wierzchołków A, B, D, C , natomiast w drugim A, B, D, A .

Definicja 2.1.9. Podgrafem grafu $G = (V, E)$ nazywamy graf $G' = (V', E')$ taki, że $V' \subseteq V$ i $E' \subseteq E$.

Podgraf $G' = (V', E')$ grafu $G = (V, E)$ możemy skonstruować, usuwając z G wierzchołki i krawędzie w taki sposób, że jeżeli wierzchołek $x \in V$ oraz $x \notin V'$, to zbiór E' nie zawiera żadnych krawędzi łączących się z x .

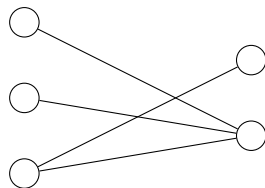
Definicja 2.1.10. Grafem pełnym nazywamy graf nieskierowany, którego każda para wierzchołków jest ze sobą połączona krawędzią. Grafy te oznaczamy przez K_n , gdzie n oznacza liczbę wierzchołków.



Rysunek 2.3. Przykład grafu pełnego K_4 .

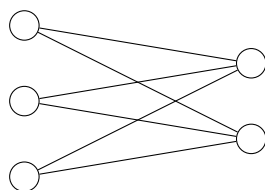
Definicja 2.1.11. Graf $G = (V, E)$ jest grafem dwudzielnym, jeżeli istnieją dwa zbiory rozłączne A i B , zawierające wierzchołki G ($A \cup B = V$) takie, że każda krawędź z E łączy wierzchołek ze zbioru A z wierzchołkiem ze zbioru B .

Definicja 2.1.12. Graf pełny dwudzielny to graf dwudzielny, w którym każdy wierzchołek ze zbioru A jest połączony krawędzią z każdym wierzchołkiem ze



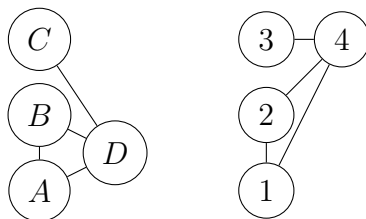
Rysunek 2.4. Przykład grafu dwudzielnego.

zbioru B . Taki graf oznaczamy jako $K_{n,m}$, gdzie n, m oznaczają odpowiednio licznosc zbiorów A i B .



Rysunek 2.5. Przykład grafu pełnego dwudzielnego $K_{2,3}$.

Definicja 2.1.13. Grafy G i H są izomorficzne, jeżeli istnieje bijekcja taka, że dla dowolnej pary wierzchołków z grafu G połączonych krawędzią istnieje odpowiadająca im para wierzchołków w grafie H również połączona krawędzią.



Rysunek 2.6. Przykład grafów izomorficznych.

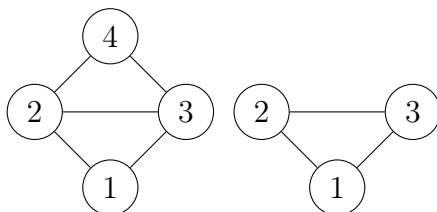
Jak łatwo zauważyć w powyższym przykładzie, istnieje bijekcja, w której wierzchołkami odpowiadającymi są: $(A, 1)$, $(B, 2)$, $(C, 3)$ i $(D, 4)$.

Definicja 2.1.14. Grafy G_1, G_2 są homeomorficzne, jeśli istnieje graf G taki, że oba grafy możemy otrzymać poprzez zastępowanie krawędzi w tym grafie ścieżkami i liczba operacji oraz długość tych ścieżek są skończone.

Definicja 2.1.15. Graf G jest ściągalny do grafu H , jeżeli z grafu H można otrzymać graf G poprzez operację usunięcia krawędzi $e = (u, v)$ i utożsamienie ze sobą wierzchołków u i v , oraz jeśli w przypadku tego zastąpienia w grafie pojawią się krawędzie wielokrotne, to pozostawiamy wyłącznie jedną taką krawędź.



Rysunek 2.7. Przykład grafów homeomorficznych.



Rysunek 2.8. Przykład grafów ściągalnych.

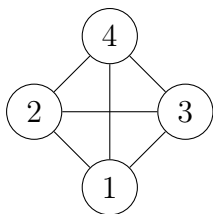
Drugi graf na rysunku 2.8 można otrzymać przez ściągnięcie wierzchołka 4 do wierzchołka 2 lub 3. W wyniku tej operacji powstają krawędzie wielokrotne, które są pomijane.

Definicja 2.1.16. *Graf nieskierowany G nazywamy spójnym, jeżeli dla dowolnych dwóch wierzchołków istnieje ścieżka łącząca te wierzchołki.*

Definicja 2.1.17. *Cykl Hamiltona to cykl, w którym każdy wierzchołek grafu jest odwiedzany dokładnie raz, oprócz wierzchołka pierwszego.*

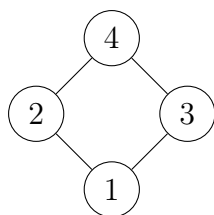
Definicja 2.1.18. *Graf nazywamy hamiltonowskim, jeżeli zawiera cykl Hamiltona.*

Definicja 2.1.19. *Graf regularny to graf, którego każdy wierzchołek ma ten sam stopień $r \in \mathbb{N}$. Mówimy wtedy, że graf jest r -regularny lub że jest regularny stopnia r .*



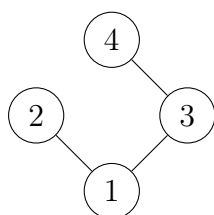
Rysunek 2.9. Przykład grafu 3-regularnego (graf K_4).

Definicja 2.1.20. *Grafem cyklicznym nazywamy graf spójny, 2-regularny. Dla grafu cyklicznego o n wierzchołkach stosujemy oznaczenie C_n .*



Rysunek 2.10. Przykład grafu cyklicznego C_4 .

Definicja 2.1.21. *Grafem liniowym o n wierzchołkach nazywamy graf, który powstał z grafu cyklicznego C_n przez usunięcie jednej krawędzi. Graf oznaczamy przez P_n .*

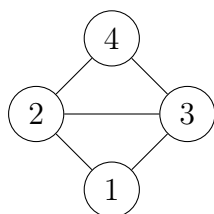


Rysunek 2.11. Przykład grafu liniowego P_4 .

2.2. Grafy dwuspójne

Grafy dwuspójne to szczególna klasa grafów, która stanowi podstawę algorytmów znajdowania najdłuższego cyklu, oraz sprawdzania istnienia cyklu Hamiltona w grafie zewnętrznie planarnym. W tym rozdziale zostanie przedstawiona definicja tej klasy grafów, oraz zagadnień niezbędnych do rozpoznawania i znajdowania podgrafów dwuspójnych.

Definicja 2.2.1. *Graf dwuspójny to graf spójny, w którym nie da się usunąć pojedynczego wierzchołka i jego krawędzi w taki sposób, aby otrzymać dwa rozłączne podgrafy.*

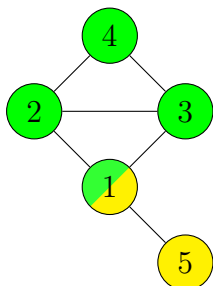


Rysunek 2.12. Przykład grafu dwuspójnego.

Definicja 2.2.2. *Maksymalnym dwuspójnym podgrafem H grafu G nazywamy taki podgraf, który jest dwuspójny i nie istnieje pograf grafu G , który zawiera H i jest dwuspójny.*

Definicja 2.2.3. Dwuspójna składowa to maksymalny dwuspójny podgraf danego grafu.

Własność 2.2.1. Dowolny graf spójny można rozłożyć na dwuspójne składowe, które posiadają co najwyżej jeden wspólny wierzchołek.



Rysunek 2.13. Podział przykładowego grafu spójnego na dwuspójne składowe.

Definicja 2.2.4. Punkt artykulacji to wierzchołek w grafie spójnym, którego usunięcie powoduje, że graf przestaje być spójny.

Własność 2.2.2. Graf dwuspójny nie posiada punktów artykulacji.

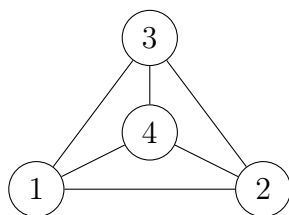
Własność wynika bezpośrednio z definicji i stanowi podstawę algorytmu podziału grafu na dwuspójne składowe, który zostanie przedstawiony w rozdziale 4.

2.3. Grafy planarne

Poniższy podrozdział poświęcony jest grafom planarnym oraz ich podstawowym własnościom.

Definicja 2.3.1. Grafem planarnym nazywamy graf, który możemy narysować na płaszczyźnie w taki sposób, aby krzywe reprezentujące krawędzie nie przecinały się ze sobą.

Definicja 2.3.2. Grafem płaskim nazywamy reprezentację grafu planarnego na płaszczyźnie.



Rysunek 2.14. Przykład grafu planarnego (graf K_4).

Definicja 2.3.3. *Ścianą grafu płaskiego nazywamy część płaszczyzny wyznaczoną przez krawędzie grafu.*

Definicja 2.3.4. *Ścianą zewnętrzną grafu płaskiego nazywamy część płaszczyzny nieograniczoną przez krawędzie grafu.*

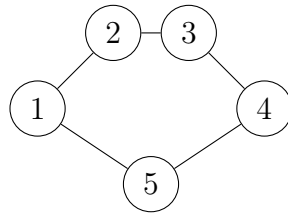
Własność 2.3.1. *Każdy graf płaski posiada jedną ścianę zewnętrzną.*

Własność 2.3.1 wynika bezpośrednio z definicji ściany zewnętrznej. Jeżeli istniałyby dwie ściany zewnętrzne nieograniczone przez graf, to otrzymujemy sprzeczność, ponieważ nie są w żaden sposób oddzielone od siebie, zatem nakładają się na siebie.

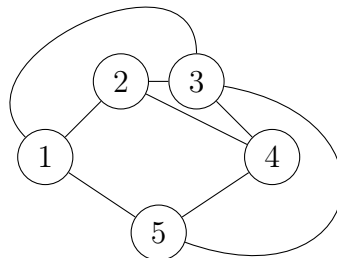
Definicja 2.3.5. *Ścianą wewnętrzną nazywamy część płaszczyzny ograniczoną krawędziami.*

Twierdzenie 2.3.1. *Grafy K_5 i $K_{3,3}$ nie są planarne.*

Dowód. Załóżmy że graf K_5 jest planarny, więc możemy przedstawić go na rysunku w taki sposób, aby żadna z krawędzi nie przecinała się. Zaważmy, że K_5 posiada cykl Hamiltona, zatem wierzchołki są rozmieszczone następująco:



Wierzchołki 1 i 3 można połączyć na zewnątrz cyklu lub wewnątrz (oba przypadki są analogiczne). Załóżmy, że łączymy wierzchołki na zewnątrz. W tym przypadku 2 i 4 można połączyć wyłącznie wewnątrz. Postępując analogicznie 3 i 5 można połączyć jedynie na zewnątrz, zatem graf ma postać:



Ostatecznie wierzchołki 1 i 4 można połączyć wyłącznie wewnątrz, co powoduje, że wierzchołki 2 i 5 nie mogą zostać połączone, nie przecinając żadnej krawędzi, co prowadzi do sprzeczności.

W przypadku grafu $K_{3,3}$, który również posiada cykl Hamiltona, to dowód jest analogiczny.

Uwaga 2.3.1. *Podgraf grafu planarnego jest grafem planarnym. Natomiast graf zawierający graf nieplanarny jest grafem nieplanarnym.*

Twierdzenie 2.3.2. *Jeżeli graf jest planarny, to:*

$$|E| \leq 3|V| - 6,$$

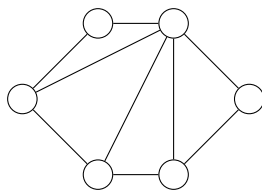
gdzie E oznacza zbiór krawędzi, natomiast V zbiór wierzchołków [9].

Twierdzenie 2.3.3. *(Twierdzenie Kuratowskiego)*

Graf G jest grafem planarnym wtedy i tylko wtedy, gdy nie zawiera pografu homeomorficznego z grafem K_5 lub $K_{3,3}$ [7].

2.4. Grafy zewnętrznie planarne

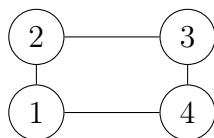
Definicja 2.4.1. *Graf G jest grafem zewnętrznie planarnym, gdy jest grafem planarnym i można go narysować tak, aby wszystkie jego wierzchołki leżały na jego ścianie zewnętrznej.*



Rysunek 2.15. Przykład grafu zewnętrznie planarnego.

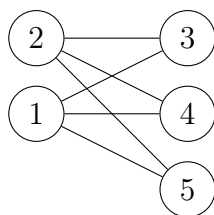
Twierdzenie 2.4.1. *Grafy K_4 i $K_{2,3}$ nie są zewnętrznie planarne.*

Dowód. Załóżmy, że K_4 jest zewnętrznie planarny, zatem możemy go narysować tak, aby wszystkie wierzchołki leżały na ścianie zewnętrznej oraz krawędzie się nie przecinały. Zauważmy, że graf posiada cykl Hamiltona, zatem możemy postąpić podobnie jak w twierdzeniu 2.3.1.



Ze względu na fakt, że wierzchołki muszą leżeć na ścianie zewnętrznej, pozostałe krawędzie możemy prowadzić wyłącznie wewnątrz cyklu, zatem łącząc wierzchołki 1, 3 oraz 2, 4 krawędzie muszą się przeciąć, co prowadzi do sprzeczności.

Założmy teraz, że graf $K_{2,3}$ jest zewnętrznie planarny.



Zauważmy, że graf posiada cykle długości cztery. Załóżmy, że jest to cykl 1, 3, 2, 4. Zatem postępując podobnie, otrzymujemy podział na ścianę wewnętrzną i zewnętrzną. Dołączając na zewnątrz wierzchołek 5 i łącząc z 1 (dla 2 sytuacja jest analogiczna) nie możemy go połączyć z wierzchołkiem 2 w taki sposób, aby 1 nie znalazł się w ścianie wewnętrznej.

Twierdzenie 2.4.2. *Graf G jest zewnętrznie planarny wtedy i tylko wtedy, gdy nie zawiera grafu homeomorficznego z K_4 ani $K_{2,3}$, oraz grafu ściągającego do nich.*

Jak łatwo zauważyć, to twierdzenie ma swój odpowiednik w klasie grafów planarnych (2.3.3). Jego dowód można znaleźć w [12].

Twierdzenie 2.4.3. *Jeżeli graf jest zewnętrznie planarny i $|V| \geq 3$, to*

$$|E| \leq 2|V| - 3,$$

gdzie E oznacza zbiór krawędzi, natomiast V zbiór wierzchołków.

Dowód. Dla grafów o $|V| = 3$ zawsze otrzymamy graf zewnętrznie planarny i jeżeli połączymy wszystkie wierzchołki ze sobą, to $|E| = 3$, zatem twierdzenie jest prawdziwe. Aby uzyskać graf o czterech wierzchołkach i maksymalnej liczbie krawędzi, która nie zaburza zewnętrznej planarności, należy dodać wierzchołek i połączyć go z dwoma sąsiadującymi. Jeżeli połączymy inaczej, to pewien wierzchołek będzie leżał wyłącznie na ścianach wewnętrznych. Analogicznie konstruujemy maksymalny graf zewnętrznie planarny o n wierzchołkach. Co prowadzi do wniosku, że do grafu zewnętrznie planarnego o $n - 1$ wierzchołkach możemy dodać maksymalnie dwie krawędzie i wierzchołek, z którego wychodzą, aby uzyskać graf $G = (V, E)$ o $|V| = n$ nie zaburzając zewnętrznej planarności. Stąd wynika, że

$$|E| \leq 3 + 2(n - 3) = 2n - 3.$$

Własność 2.4.1. *Każdy podgraf grafu zewnętrznie planarnego jest zewnętrznie planarny.*

Własność 2.4.2. *Jeśli graf jest zewnętrznie planarny, to posiada co najmniej dwa wierzchołki stopnia co najwyżej 2 [10].*

Własność 2.4.3. *Jeżeli G jest grafem zewnętrznie planarnym, to dla wszystkich wierzchołków stopnia mniejszego lub równego 2 wychodzące krawędzie leżą na ścianie zewnętrznej.*

Własność jest dość oczywista, ponieważ gdyby leżały wewnątrz, to wierzchołek również leżałby na ścianach wewnętrznych.

Własność 2.4.4. *Każdy prosty graf zewnętrznie planarny może być pokolorowany przy użyciu maksymalnie trzech kolorów [13].*

Poniższe definicje dzielą krawędzie grafu na trzy klasy: wewnętrzne, zewnętrzne, oraz mosty. Taki podział pozwala na zdefiniowanie odwzorowania kolorującego krawędzie grafu.

Definicja 2.4.2. *Krawędź grafu nazywamy wewnętrzną, gdy sąsiaduje ze ścianami wewnętrznymi.*

Krawędź grafu nazywamy zewnętrzną, gdy sąsiaduje ze ścianą zewnętrzną i ze ścianą wewnętrzną.

Krawędź grafu nazywamy mostem, gdy po jej usunięciu graf jest niespójny. Most sąsiaduje tylko ze ścianą zewnętrzną.

Twierdzenie 2.4.4. *Graf zewnętrznie planarny jest hamiltonowski wtedy i tylko wtedy, gdy jest dwuspójny [12].*

Interesującym zagadnieniem dotyczącym grafów zewnętrznie planarnych jest również kolorowanie krawędzi, które polega na przyporządkowaniu kolorów w taki sposób, aby krawędzie łączące ten sam wierzchołek były różnego koloru. W przypadku ogólnym znalezienie najmniejszej liczby kolorów, aby dokonać tej procedury, jest problemem NP-zupełnym. W klasie grafów zewnętrznie planarnych sytuacja nie jest tak pesymistyczna. W 1964 Vadim Georgievich Vizing opublikował następujące twierdzenie, które wraz z dowodem można znaleźć w [16].

Twierdzenie 2.4.5. *(Twierdzenie Vizinga) Niech G będzie prostym grafem zewnętrznie planarnym, ρ oznacza maksymalny stopień, oraz $\chi(G)$ jest tak zwanym indeksem chromatycznym, czyli minimalną liczbą kolorów potrzebnych do pokolorowania krawędzi. Wtedy*

$$\rho \leq \chi(G) \leq \rho + 1.$$

W tej sytuacji grafy możemy podzielić na dwie klasy.


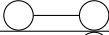
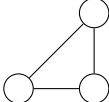
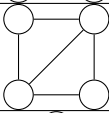
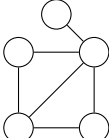
Definicja 2.4.3. *Graf zewnętrznie planarny należy do klasy 1, jeżeli można pokolorować jego krawędzie na ρ kolorów. Natomiast należy do klasy 2, gdy indeks chromatyczny wynosi $\chi(G) = \rho + 1$.*

W [15] zostało zaprezentowane twierdzenie (2.4.6), która wraz z twierdzeniem Vizinga pozwala na jednoznaczne wyznaczenie indeksu chromatycznego dla grafów zewnętrznie planarnych.

Twierdzenie 2.4.6. *Graf zewnętrznie planarny G jest w klasie 1, jeżeli nie posiada cyklu nieparzystego.*

Na koniec teoretycznego wprowadzenia do grafów zewnętrznie planarnych została zaprezentowana tabela z licznością w klasie prostych grafów spójnych oraz prostych grafów spójnych zewnętrznie planarnych w zależności od liczby wierzchołków. W ostatniej kolumnie znajdują się przykładowe grafy zewnętrznie planarne.

Tabela 2.1. Porównanie liczby grafów prostych spójnych o n wierzchołkach z liczbą grafów zewnętrznie planarnych.

n	Grafy proste spójne	Grafy proste, spójne, zewnętrznie planarne	Przykład grafu prostego, spójnego, zewnętrznie planarnego
1	1	1	
2	1	1	
3	2	2	
4	6	5	
5	21	13	

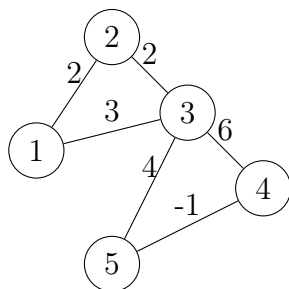
3. Implementacja grafów

Do implementacji algorytmów w rozdziale 4 zostanie wykorzystany język Python wraz z biblioteką `graphtheory` [3]. W tym rozdziale zostaną przedstawione jego podstawowe funkcjonalności, które będą użyte w następnej części.

3.1. Reprezentacje grafów

Istnieje wiele metod reprezentacji grafu jako struktury danych umożliwiającej przetwarzanie za pomocą programów komputerowych. W poniższej sekcji zostaną przedstawione najbardziej popularne sposoby oraz struktura z biblioteki `graphtheory`.

Reprezentacja macierzowa. Graf o n wierzchołkach można przedstawić za pomocą macierzy \mathbf{M} o wymiarach $n \times n$. Komórka macierzy $\mathbf{M}[i, j]$ przechowuje informację o krawędzi (i, j) . Wartość komórki może oznaczać istnienie krawędzi (0 lub 1) albo wagę połączenia między wierzchołkami.



Rysunek 3.1. Przykładowy graf z wagami.

Reprezentacja macierzowa dla grafu z rysunku 3.1 wygląda następująco:

$$\mathbf{M} = \begin{pmatrix} 0 & 2 & 3 & 0 & 0 \\ 2 & 0 & 2 & 0 & 0 \\ 3 & 2 & 0 & 6 & 4 \\ 0 & 0 & 6 & 0 & -1 \\ 0 & 0 & 4 & -1 & 0 \end{pmatrix}$$

Dla grafów nieskierowanych macierz jest symetryczna.

Reprezentacja jako listy sąsiedztwa. Kolejną metodą reprezentacji grafu jest lista sąsiedztwa. Graf o n wierzchołkach można przedstawić za pomocą n elementowej tablicy, w której każdy element jest listą zawierającą sąsiadów wierzchołka utożsamianego z daną pozycją w tablicy.

Lista sąsiedztwa dla grafu z rysunku 3.1 wygląda następująco

```
[[2, 3],  
 [1, 3],  
 [1, 2, 4, 5],  
 [3, 5],  
 [3, 4]],
```

gdzie indeks elementu listy oznacza wierzchołek odpowiednie 1, 2, 3, 4, 5.

3.2. Reprezentacja grafu w bibliotece graphtheory

Implementacja struktury grafowej w bibliotece graphtheory bazuje na klasach Edge oraz Graph. Poniżej zostanie przedstawiona przykładowa sesja interaktywna ilustrująca użycie wybranych metod tych klas.

```
>>> from edges import Edge  
>>> from graphs import Graph  
  
# utworzenie grafu nieskierownego  
>>> G = Graph()  
  
# utworzenie grafu skierownego  
>>> H = Graph(directed=True)  
  
# utworzenie krawedzi skierowanej (1,2) za pomoca klasy Edge  
>>> e = Edge(1,2) # domyslne waga krawedzi wynosi 1  
>>> e = Edge(1,2,10) # krawedz z waga 10  
  
# dodanie wierzcholka 1 do grafu G  
>>> G.add_node(1)  
  
# dodanie krawedzi e do grafu G  
>>> G.add_edge(e)  
  
# usuniecie krawedzi z grafu  
>>> G.del_edge(e)  
  
# metody zwracaja odpowiednio liczbe wierzcholkow i krawedzi w grafie  
>>> G.v()  
>>> G.e()  
  
# iteracja po wszystkich wierzcholkach grafu G  
>>> for node in G.iternodes():  
>>>     print(node)  
  
# iteracja po wszystkich sasiadach wierzcholka 1 w grafie G  
>>> for node in G.iteradjacent(1):  
>>>     print(node)
```

3.3. Klasa Edge

Instancje klasy Edge reprezentują krawędzie skierowane grafu. Tworzymy ją w następujący sposób:

```
>>> e = Edge(source , target , weight)
```

W konstruktorze `source` jest wierzchołkiem początkowym, `target` wierzchołkiem końcowym, a `weight` wagą krawędzi. Krawędzie są hashowalne, można je porównywać i sortować (waga krawędzi ma najwyższy priorytet wśród atrybutów). Krawędź `~e` oznacza krawędź skierowaną przeciwnie do krawędzi `e`. W grafie nieskierowanym każda krawędź nieskierowana jest wewnętrznie przechowywana jako para krawędzi skierowanych o przeciwnych kierunkach.

3.4. Klasa `Graph`

Instancja klasy `Graph` reprezentuje graf. Tworzymy go w następujący sposób:

```
>>> G = Graph() # graf nieskierowany
>>> H = Graph(directed=True) # graf skierowany
```

Moduł `graphs` zawiera wiele metod ułatwiających implementację algorytmów grafowych. Poniżej zostaną przedstawione wybrane z nich, które będą niezbędne w sekcji 3.5.

1. Metody zwracające liczbę wierzchołków i krawędzi grafu G

```
>>> G.v() # liczba wierzchołków
>>> G.e() # liczba krawędzi
```

2. Dodanie i usunięcie wierzchołka grafu G , oraz sprawdzenie, czy graf posiada podany wierzchołek.

```
>>> G.add_node(1)
>>> G.has_node(1)
>>> G.del_node(2)
```

3. Dodanie i usunięcie krawędzi grafu G , oraz sprawdzenie, czy graf posiada krawędź.

```
>>> G.add_egde(Edge(1,2))
>>> G.del_edge(Edge(1,2))
>>> G.has_edge(Edge(1,2))
```

4. Generator krawędzi grafu.

```
>>> list(G.iteredges())
```

5. Utworzenie kopii grafu.

```
>>> H = G.copy()
```

6. Odczyt stopnia wierzchołka (o ile wierzchołek istnieje).

```
>>> G.degree(1)
```

7. Generator wszystkich wierzchołków.

```
>>> list(G.iternodes())
```

8. Generator wszystkich sąsiadów podanego wierzchołka.

```
>>> list(G.iteradjacent(5))
```

3.5. Klasa BipartiteGraphBFS

Klasa BipartiteGraphBFS, znajdująca się w module bipartite, zawiera implementację rozpoznawania grafów dwudzielnych wraz z algorytmem kolorowania.

Własność 3.5.1. *Wierzchołki każdego grafu dwudzielnego można pokolorować za pomocą dwóch kolorów.*

Z uwagi na powyższą własność algorytm zostanie wykorzystany w implementacji kolorowania wierzchołków grafów zewnętrznie planarnych za pomocą maksymalnie trzech kolorów.

```
# Przykład rozpoznawania i kolorowania grafu dwudzielnego.  
# Jeżeli graf nie jest dwudzielny, to zostanie rzucony wyjątek ValueError.  
>>> algorithm = BipartiteGraphBFS(G)  
>>> algorithm.run()  
>>> print(algorithm.color)    # dict z kolorami wierzchołkow
```

4. Algorytmy

W tym rozdziale przedstawimy implementacje algorytmów dotyczących różnych zagadnień związanych z grafami zewnętrze planarnymi, m.in. rozpoznawania, generowania i kolorowania wierzchołków.

4.1. Rozpoznawanie grafów zewnętrze planarnych

Pierwszym zaprezentowanym algorytmem jest rozpoznawanie grafów zewnętrze planarnych.

Dane wejściowe: Dowolny graf G .

Problem: Sprawdzenie, czy graf G jest zewnętrze planarny.

Opis algorytmu: Algorytm rozpoczynamy od sprawdzenia, czy spełnione jest ograniczenie na liczbę krawędzi $|E| \leq 2|V| - 3$, gdzie $|E|, |V|$ to odpowiednio liczba krawędzi i liczba wierzchołków. Następnie jest wykonywana procedura redukcji grafu wraz z kolorowaniem krawędzi incydentnych z zredukowanym wierzchołkiem stopnia co najwyżej 2. Algorytm korzysta z własności 2.4.1 oraz 2.4.2, rozpoznając kolejno krawędzie zewnętrzne, wewnętrzne i mosty. Proces jest powtarzany rekurencyjnie dla pomniejszonych grafów. Technika jest wystarczająca, aby odrzucić nieprawidłowe grafy przez uzyskanie sprzecznego kolorowania krawędzi.

W algorytmie występują trzy kolory: "cross", "out" i "bridge", które oznaczają odpowiednio krawędzie wewnętrzne, zewnętrzne i mosty. Proces kolorowania rozpoczyna się od przypisania "cross" do każdej krawędzi. Następnie wykonywany jest proces redukcji, w którym następujące działania są powtarzane w każdej iteracji. Wierzchołek u jest zdejmowany ze stosu będącego zbiorem wierzchołków stopnia co najwyżej drugiego. Jeżeli jego stopień wynosi 0, to żadne działanie nie jest wykonane. Graf można traktować jako zredukowany, ponieważ nie istnieją krawędzie z nim incydentne, zatem nie może pojawić się znów na stosie i nie zaburza zewnętrznej planarności. Jeżeli jego stopień wynosi 1, to wierzchołek wraz z jego krawędzią jest usuwany z grafu, ponieważ można go narysować na zewnątrz. Jeśli jest stopnia 2, to kolorowanie wygląda następująco. Gdy istnieje krawędź pomiędzy jego sąsiadami u_1 i u_2 , oraz jego krawędzie incydentne są oznaczone jako "out" lub "cross", to jeśli krawędź pomiędzy u_1 i u_2 jest oznaczona jako:

1. "cross", to zostaje pokolorowana jako "out", ponieważ w kolejnym etapie w grafie zredukowanym będzie musiała być zewnętrzna lub mostem.
2. "out", to zostaje pokolorowana jako "bridge". Analogicznie jak powyżej, może być tylko mostem po redukcji.

3. "bridge", to graf nie jest planarny, ponieważ stopień wierzchołka wynosi co najmniej 2, a to więcej niż stopień mostu.

Natomiast jeżeli nie istnieje krawędź pomiędzy sąsiadami, to zostaje dodana do zredukowanego grafu i jej kolor wynosi "out", gdy (u, u_1) i $(u, u_2) \in ["out", "cross"]$, a w przeciwnym wypadku zostaje jej przypisany "bridge".

W przypadku, gdy nie ma elementów na stosie, a graf nie został całkowicie zredukowany, zewnętrzna planarność zostaje zaburzona. To wynika z własności, że graf zewnętrznie planarny posiada co najmniej dwa wierzchołki stopnia co najwyżej drugiego, oraz że podgraf grafu zewnętrznie planarnego jest zewnętrznie planarny. Dokładny opis wraz z uzasadnieniem poprawności algorytmu można znaleźć w pracy [10].

Złożoność: Algorytm przegląda kolejne wierzchołki stopnia co najwyżej 2 i koloruje krawędzie incydentne. Operacje takie jak znajdowanie sąsiadów i krawędzi danego wierzchołka oraz zapamiętywanie ich kolorów, są wykonywane w czasie $O(1)$, zatem złożoność czasowa całego procesu wynosi $O(V)$, gdzie V to liczba wierzchołków.

Listing 4.1. Moduł outerplanar.

```
#!/usr/bin/python

import sys
from graphtheory.structures.edges import Edge
from graphtheory.connectivity.connected import is_connected

class OuterplanarGraph:
    """Outerplanar graphs detection."""

    def __init__(self, graph):
        """The algorithm initialization."""
        if graph.is_directed():
            raise ValueError("the graph is directed")
        self.graph = graph
        self._graph_copy = self.graph.copy()
        self.color = None # coloring edges (cross, out, bridge)
        self.outerplanar = None

    def run(self):
        """Executable pseudocode."""
        nedges = self.graph.e() # O(V) time
        if nedges > 2 * self.graph.v() - 3:
            self.outerplanar = False
            return False
        # Color all edges.
        self.color = dict((edge, "cross") for edge in self.graph.iteredges())
        # Find vertices of degree < 3.
        # M is a set with vertices where degree <= 2.
        M = set(v for v in self.graph.iternodes()
                if self.graph.degree(v) <= 2)
        self.outerplanar = True

        while len(M) > 0 and self.outerplanar:
```

```

u = M.pop()
Nu = self._graph_copy.degree(u)

if Nu == 0: # isolated vertex
    pass
elif Nu == 1:
    edge = next(self._graph_copy.iteroutedges(u))
    assert edge.source == u
    self._graph_copy.del_node(u) # remove with edges
    if self._graph_copy.degree(edge.target) == 2:
        M.add(edge.target)
    nedges -= 1
elif Nu == 2:
    edge1, edge2 = list(self._graph_copy.iteroutedges(u))
    self._graph_copy.del_node(u) # remove with edges
    color1 = self.find_color(edge1)
    color2 = self.find_color(edge2)
    if edge1.target > edge2.target: # set right orientation
        edge3 = Edge(edge2.target, edge1.target)
    else:
        edge3 = Edge(edge1.target, edge2.target)

    if self._graph_copy.has_edge(edge3):
        # get original edge
        for edge in self._graph_copy.iteroutedges(edge3.source):
            if edge.target == edge3.target:
                edge3 = edge
                break
        nedges -= 2
        if self._graph_copy.degree(edge3.source) == 2:
            M.add(edge3.source)
        if self._graph_copy.degree(edge3.target) == 2:
            M.add(edge3.target)
        if color1 in ("cross", "out") and color2 in ("cross", "out"):
            if self.color[edge3] == "cross":
                self.color[edge3] = "out"
            elif self.color[edge3] == "out":
                self.color[edge3] = "bridge"
            elif self.color[edge3] == "bridge":
                self.outerplanar = False
        else:
            self.outerplanar = False
    else: # graph doesn't contain edge3
        self._graph_copy.add_edge(edge3)
        nedges -= 1
        if color1 in ("cross", "out") and color2 in ("cross", "out"):
            self.color[edge3] = "out"
        else:
            self.color[edge3] = "bridge"
# 'while '.

assert self._graph_copy.e() == nedges
if self.outerplanar:
    self.outerplanar = (nedges == 0)
return self.outerplanar

def find_color(self, edge):

```

```

"""Find color of edge."""
if edge.source > edge.target:
    edge = ~edge
return self.color[edge]

```

4.2. Algorytm podziału grafu na składowe dwuspójne

Algorytm podziału grafu na dwuspójne składowe stanowi postawę kolejnego algorytmu rozpoznawania grafów hamiltonowskich w klasie grafów zewnętrznie planarnych, oraz algorytmu wyznaczania długości najdłuższej ścieżki w tej klasie. Opiera się na algorytmie wyszukiwania w głąb w celu znalezienia punktów artykulacji. Został zaprojektowany przez Johna Hopcrofta i Roberta Tarjana w roku 1973.

Dane wejściowe: Dowolny graf G .

Problem: Podzielenie grafu G na rozłączne składowe dwuspójne.

Opis algorytmu: Algorytm polega na przechowywaniu wierzchołków na stosie, oraz wyszukiwaniu punktów artykulacji za pomocą przeszukiwania w głąb, przechowując informacje o głębokości każdego wierzchołka w drzewie utworzonym podczas przeszukiwania, oraz najniższą głębokość wszystkich dzieci danego wierzchołka. Najniższa głębokość nosi nazwę najniższego punktu i jest obliczana po przejściu wszystkich potomków, stąd algorytm przechowuje również informacje o rodzicach. Algorytm bazuje na własności, że wierzchołek v jest punktem artykulacji wtedy i tylko wtedy, gdy istnieje jego dziecko w takie, że $\text{low}[w] \leq \text{disc}[v]$. Kiedy punkt artykulacji zostanie znaleziony, to wszystkie dotychczas odwiedzone wierzchołki tworzą składową dwuspójną. Następnie algorytm powtarza czynności aż do momentu odwiedzenia wszystkich wierzchołków. Jeżeli nie ma punktów artykulacji, to cały graf tworzy jedną dwuspójną składową [14].

Złożoność: Złożoność czasowa algorytmu wynosi $O(V + E)$, ponieważ podczas jego wykonywania procedura przeszukuje wierzchołek i przechodzi do jego sąsiadów, zatem w każdym etapie przeglądane są również jego krawędzie incydentne.

Listing 4.2. Moduł bicomponent.

```

from graphtheory.structures.edges import Edge
from graphtheory.structures.graphs import Graph

class BicomponentsGraph:

    def __init__(self, graph):
        """Load up a container for biconnected components for graph.
        Parameters
        

---


        graph : undirected graph
        """

```

```

if graph.is_directed():
    raise ValueError("the graph is directed")
self.bicomponents = []
self.edgestack = []
self.disc = {} #depth of nodes
self.low = {} #lowpoint
self.parent = {} #parent of nodes in depth-tree
self.graph = graph
self.time = 0 #use to find depth for nodes

def run(self):

self.disc = dict((node, -1) for node in self.graph.iternodes())
self.low = self.disc.copy()
self.parent = self.disc.copy()

for node in self.graph.iternodes():
    if self.disc[node] == -1:
        self.find_articulation(node)
    if self.edgestack:
        component = Graph()
        while self.edgestack:
            edge = self.edgestack.pop()
            component.add_edge(edge)
        self.bicomponents.append(component)

def show(self):
    """The container presentation."""
    for component in self.bicomponents:
        component.show()

def find_articulation(self, u):
    children = 0
    self.disc[u] = self.time
    self.low[u] = self.time
    self.time += 1

    #iteration over incident edges returns the original edges of the graph
    for incident_edge in self.graph.iteroutedges(u):

        v = incident_edge.target #adjacent vertex
        if self.disc[v] == -1 : #unvisited vertex

            self.parent[v] = u
            children += 1
            self.edgestack.append(incident_edge) #add original edge to stack
            self.find_articulation(v)

        self.low[u] = min(self.low[u], self.low[v])

    #check condition of articulation point
    condition1 = self.parent[u] == -1 and children > 1
    condition2 = self.parent[u] != -1 and self.low[v] >= self.disc[u]
    if condition1 or condition2:
        edge = self.edgestack.pop()
        component = Graph() #creating biconnected component for graph
        #adding edges to component until get articulation point

```

```

        while edge != incident_edge:
            component.add_edge(edge)
            edge = self.edgestack.pop()
        component.add_edge(edge)
        self.bicomponents.append(component)

    elif v != self.parent[u] and self.low[u] > self.disc[v]:
        self.low[u] = min(self.low[u], self.disc[v])
        self.edgestack.append(incident_edge)

```

4.3. Cykl Hamiltona i najdłuższy cykl w grafie

Poniższy algorytm sprawdza, czy graf jest hamiltonowski lub jaka jest największa długość cyklu w grafie.

Dane wejściowe: Graf zewnętrznie planarny.

Problem: Wyznaczenie najdłuższego cyklu w grafie, a w szczególności sprawdzenie istnienia cyklu Hamiltona.

Opis algorytmu: Na początku wywoływany jest algorytm podziału grafu na dwuspójne składowe, który został wcześniej opisany. Korzystając z twierdzenia 2.4.4 oraz własności, że najdłuższy cykl w grafie zewnętrznie planarnym równa się liczbie wierzchołków w największej jego składowej dwuspójnej, sprawdzenie, czy graf jest hamiltonowski, polega na wyznaczeniu liczby składowych. Natomiast wyznaczenie największej długości cyklu na znalezieniu maksymalnej liczby wierzchołków w nich.

Złożoność: Złożoność czasowa algorytmu wynosi $O(V + E)$, ponieważ algorytm podziału na dwuspójne składowe zależy liniowo od liczby krawędzi i wierzchołków w grafie.

Listing 4.3. Moduł longest_cycle.

```

#!/usr/bin/python

from bicomponent import BicomponentsGraph
from outerplanar import OuterplanarGraph

class LongestCycle:

    def __init__(self, graph):
        """The algorithm initialization."""
        if graph.is_directed():
            raise ValueError("the graph is directed")
        if not OuterplanarGraph(graph).run():
            raise ValueError("the graph isn't outerplanar")
        self.algorithm = BicomponentsGraph(graph)
        self.algorithm.run()

    def is_hamiltonian(self):
        return len(self.algorithm.bicomponents) == 1

```



```

def longest_cycle_length(self):
    component = max(self.algorithm.bicomponents, key=len)
    return len(component)

```

4.4. Kolorowanie grafów zewnętrznie planarnych

Kolorowanie (wierzchołków) grafów należy do problemów NP-zupełnych. Polega ono na dobraniu najmniejszej liczby kolorów do wierzchołków, aby wierzchołki połączone krawędzią miały różne kolory. Optymalne kolorowanie grafów zewnętrznie planarnych wymaga użycia najwyżej trzech kolorów, a jeżeli dany graf jest dwudzielny, to wystarczą dwa kolory.

Dane wejściowe: Graf zewnętrznie planarny.

Problem: Kolorowanie wierzchołków grafu zewnętrznie planarnego.

Opis algorytmu: Algorytm rozpoczynamy od sprawdzenia, czy graf jest dwudzielny, ponieważ wtedy można go pokolorować dwoma kolorami. Dwudzielność zostaje sprawdzona za pomocą klasy `BipartiteGraphBFS` z biblioteki `graphtheory`. Jeżeli graf jest dwudzielny, to od razu dostajemy optymalne kolorowanie. W przeciwnym przypadku procedura polega na usuwaniu dowolnego wierzchołka stopnia co najwyżej drugiego. Następnie rekurencyjnie usuwamy następne wierzchołki stopnia co najwyżej drugiego. Przy powrocie z wywołania rekurencyjnego przywracamy usunięty wierzchołek i nadajemy mu kolor inny niż ma jego dwóch sąsiadów [13].

Złożoność: Algorytm przeszukuje wierzchołki stopnia co najwyżej dwa w zredukowanych podgrafach powstałych z oryginalnego grafu, zatem każda procedura rekurencyjna wykonuje się w czasie stałym. Natomiast ich liczba jest równa liczbie wierzchołków w grafie. Stąd wynika, że złożoność czasowa algorytmu wynosi $O(V)$.

Listing 4.4. Moduł coloring.

```

#!/usr/bin/python

from graphtheory.bipartiteness.bipartite import BipartiteGraphBFS
from outerplanar import OuterplanarGraph

class ColorOuterplanar:

    def __init__(self, graph):
        if graph.is_directed():
            raise ValueError("the graph is directed")
        algorithm = OuterplanarGraph(graph) #check outerplanarity
        if not algorithm.run():
            raise ValueError("The graph isn't outerplanar")
        self.graph = graph
        self.vcolor = {}

```

```

def run(self):
    try:
        algorithm = BipartiteGraphBFS(self.graph)
        algorithm.run()
        self.vcolor = algorithm.color
    except ValueError:
        self.coloring()

def coloring(self):
    degree = {}
    d2 = set() # wierzchołki stopnia < 3
    visited = {}
    for node in self.graph.iternodes():
        degree[node] = self.graph.degree(node)
        if degree[node] < 3:
            d2.add(node)
        self.vcolor[node] = -1
        visited[node] = -1

    node = d2.pop()
    self.color_recursive(degree, d2, visited, node)

    colors = [0, 1, 2]
    for v in self.graph.iteradjacent(node):
        if self.vcolor[v] != -1 and self.vcolor[v] in colors:
            colors.remove(self.vcolor[v])
    self.vcolor[node] = colors[0]

def color_recursive(self, degree, d2, visited, node):
    colors = [0, 1, 2]
    visited[node] = 1
    for v in self.graph.iteradjacent(node):
        degree[v] = degree[v] - 1
        if degree[v] < 3 and visited[v] != 1:
            d2.add(v)
    if len(d2) != 0:
        node = d2.pop()
        self.color_recursive(degree, d2, visited, node)
    for v in self.graph.iteradjacent(node):
        if self.vcolor[v] != -1 and self.vcolor[v] in colors:
            colors.remove(self.vcolor[v])
    self.vcolor[node] = colors[0]

```

4.5. Rozpoznawanie maksymalnych grafów zewnętrznie planarnych

Szczególnym przypadkiem omawianych grafów są maksymalne grafy zewnętrznie planarne. Poniżej zostanie przedstawiony algorytm ich rozpoznawania. Aby testować maksymalną zewnętrzną planarność, można również zastosować algorytm sprawdzania zewnętrznej planarności i następnie sprawdzić, czy liczba krawędzi jest maksymalna. Jednak procedura zaprezentowana w pracy [11] zasługuje na uwagę, ponieważ korzysta z podstawowych własności maksymalnych grafów zewnętrznie planarnych i tylko je rozpoznaje.

Dane wejściowe: Graf zewnętrznie planarny.

Problem: Rozpoznawanie maksymalnych grafów zewnętrznie planarnych.

Opis algorytmu: Algorytm rozpoczynamy od sprawdzenia, czy liczba krawędzi wynosi $|E| = 2|V| - 3$, gdzie $|V|$ to liczba wierzchołków. Następnie, poprzez usuwanie wierzchołków stopnia 2, sprawdzana jest maksymalna zewnętrzna planarność powstałych podgrafów, korzystając również z faktu, że jeżeli wierzchołek należy do dwóch trójkątów, to albo nie jest planarny, albo wszystkie wierzchołki nie mogą leżeć na zewnątrz. Szczegółowy opis algorytmu oraz uzasadnienie poprawności można znaleźć w pracy [11].

Złożoność: Algorytm przeszukuje graf podobnie jak w przypadku kolorowania. Stąd wynika, że złożoność czasowa algorytmu wynosi $O(V)$.

Listing 4.5. Moduł maximal_outerplanar.

```
#!/usr/bin/python

from graphtheory.structures.edges import Edge
from graphtheory.structures.graphs import Graph

class MaximalOuterplanar:

    def __init__(self, graph):
        """
        -----
        graph : undirected graph
        """
        if graph.is_directed():
            raise ValueError("the graph is directed")

        self.graph = graph.copy()

    def run(self):
        if self.graph.e() != 2 * self.graph.v() - 3:
            return False
        edges = set()
        vertices = set(self.graph.iternodes())

        for v in vertices:
            for edge in self.graph.iteroutedges(v):
                edges.add(edge)

        list_deg = set()
        for v in self.graph.iternodes():
            if self.graph.degree(v) == 2:
                list_deg.add(v)
        pairs = set()
        if len(list_deg) < 2:
            return False

        L = 1
        w1 = None
        w2 = None
```

```

nvertices = len(vertices)

while L < nvertices - 2:
    v = list_deg.pop()

    adj = list(self.graph.iteroutedges(v))
    edge1 = adj[0]
    edge2 = adj[1]

    w1 = edge1.target
    w2 = edge2.target

    self.graph.del_edge(edge1)
    edges.remove(edge1)
    self.graph.del_edge(edge2)
    edges.remove(edge2)
    vertices.remove(v)
    nvertices -= 1

    self.update_list(list_deg, w1)
    self.update_list(list_deg, w2)

    if w1 < w2:
        pairs.add(Edge(w1,w2))

    if len(list_deg) - L < 2:
        return True
    L = L + 1

for pair in pairs:
    if pair in edges or Edge(pair[1], pair[0]) in edges:
        continue
    else:
        return False
return True

def update_list(self, list_deg, v):
    if v in list_deg and self.graph.degree(v) != 2:
        list_deg.remove(v)
    if self.graph.degree(v) == 2:
        list_deg.add(v)

```

4.6. Algorytm generowania maksymalnych grafów zewnętrznie planarnych

W przedstawionym algorytmie generowanie bazuje na dwóch własnościach. Każdy maksymalny graf zewnętrznie planarny posiada cykl Hamiltona oraz podgraf powstały przez usunięcie wierzchołka i jego krawędzi incydentnych jest również maksymalnym grafem zewnętrznie planarnym.

Dane wejściowe: Liczba wierzchołków V .

Problem: Generowania maksymalnych grafów zewnętrznie planarnych.

Opis algorytmu: Algorytm polega na generowaniu cyklu Hamiltona dla grafu. Następnie losowo wybiera wierzchołek i łączy jego sąsiadów krawędzią. Wierzchołek v jest traktowany, jakby nie istniał i algorytm jest powtarzany aż do momentu uzyskania pografu o trzech wierzchołkach.

Złożoność: Złożoność czasowa algorytmu wynosi $O(V)$, gdzie V jest liczbą wierzchołków. Algorytm wykonuje się w pętli, w której za każdym razem usuwa jeden wierzchołek, stąd złożoność jest liniowa.

Listing 4.6. Moduł `generate_max_outerplanar`.

```
#!/usr/bin/python

import random
from graphtheory.structures.edges import Edge
from graphtheory.structures.graphs import Graph

class MaximalOuterplanarGenerator:

    def __init__(self, n):
        if not isinstance(n, int):
            raise ValueError("Incorrect number of vertices")
        self.graph_set = []
        self.n = n
        self.graph = Graph()

    def run(self):
        # create hamiltonian cycle (cycle graph)
        for i in range(self.n):
            self.graph.add_edge( Edge(i, (i + 1) % self.n) )
        self._generate()
        return self.graph

    def _generate(self):
        G = self.graph.copy()
        vlist = list(G.iternodes()) # O(V) time
        random.shuffle(vlist) # O(V) time
        while G.v() > 3: # O(V) time
            # fill cycle with random edges
            v = vlist.pop() # O(1) time
            u, w = list(G.iteradjacent(v))
            chord = Edge(u, w)
            self.graph.add_edge(chord)
            G.add_edge(chord)
            G.del_node(v) # with edges
```

5. Podsumowanie

W pracy zostały przybliżone podstawowe zagadnienia teoretyczne dotyczące grafów wraz z przykładami wygenerowanymi za pomocą pakietu *tkiz*. Następnie zostały przedstawione możliwe reprezentacje grafów w programach komputerowych, jak również biblioteka *graphtheory*.

Podczas pracy zostało zaimplementowane sześć klas zawierających interesujące algorytmy dla grafów zewnętrznie planarnych. Między innymi procedurę rozpoznawania grafów wewnętrznie planarnych, która stanowi fundament dalszych rozważań. Dalej zaimplementowano algorytm podziału grafu na składowe dwuspójne, który pozwolił na rozpoznawanie grafów hamiltonowskich i wyznaczanie długości najdłuższego cyklu w grafie zewnętrznie planarnym. Następnie został przedstawiony optymalny algorytm kolorowania wierzchołków. Rozwiązanie tych problemów jest szczególnie ważne, ponieważ w ogólnym przypadku są to problemy NP-zupełne. Ostatnimi zaimplementowanymi algorytmami było rozpoznawanie oraz generowanie maksymalnych grafów zewnętrznie planarnych. Na koniec z pomocą modułu *unitest* zostały wykonane testy jednostkowe sprawdzający poprawność algorytmów. Natomiast wykorzystując moduł *timeit* została wyznaczona empirycznie złożoność obliczeniowa programów.

Przedstawione zagadnienia wraz z kodem mogą pozwolić na rozszerzenie wiedzy dotyczącej teorii grafów, jak również rozwinięcie umiejętności programistyczne z wykorzystaniem biblioteki *graphtheory*. Natomiast zagadnienie zewnętrznej planarności może pozwolić na dalsze rozważania ich szczególnych podklas, takich jak na przykład kaktusy.

A. Testy algorytmów

Dodatek przedstawia opis i wyniki testów poprawności i wydajności algorytmów zaimplementowanych w tej pracy.

A.1. Poprawność

Rozpoznawanie grafów zewnętrznie planarnych. Test jednostkowy polega na sprawdzeniu poprawności algorytmu dla kilku przykładowych grafów. W szczególności dla wykluczenia zewnętrznej planarności dla grafu $K_{2,3}$ oraz K_4 .

Algorytm podziału grafu na dwuspójne składowe. Test sprawdza poprawność algorytmu poprzez sprawdzenie liczby składowych, na które został podzielony przykładowy graf.

Algorytm sprawdzania czy graf jest hamiltonowski i wyznaczenia najdłuższej długości cyklu. Test sprawdza algorytm dla grafu hamiltonowskiego i nieposiadającego cyklu Hamiltona oraz długość najdłuższego cyklu w przykładowych grafach.

Kolorowanie wierzchołków grafu. Poprawność algorytmu kolorowania wierzchołków grafu zewnętrznie planarnego polega na sprawdzeniu liczby kolorów, z jakimi zostały pokolorowane grafy, oraz czy algorytm poprawnie zwraca wyjątek w przypadku braku zachowania zewnętrznej planarności.

Algorytm rozpoznawania maksymalnych grafów zewnętrznie planarnych. Klasa testująca poprawność zawiera sześć metod. Pierwsze dwie sprawdzają, czy algorytm poprawnie klasyfikuje grafy jako maksymalnie zewnętrznie planarne, kolejne dwa czy zwraca fałsz w przypadku niezachowania maksymalnej zewnętrznej planarności. Natomiast ostatnie dwa sprawdzają, czy prawidłowo zwraca wyjątek w przypadku grafu skierowanego, oraz grafu niebędącego zewnętrznie planarnym.

Algorytm generowania maksymalnych grafów zewnętrznie planarnych. Ostatnie testy jednostkowe wykorzystują klasę `MaximalOuterplanar`, która sprawdza, czy wygenerowane grafy są zewnętrznie planarne.

A.2. Złożoność obliczeniowa

Testy złożoności obliczeniowej wykorzystane w pracy polegają na empirycznym wyznaczaniu tej wartości. Każdy algorytm, z wyjątkiem generatora maksymalnych grafów zewnętrznie planarnych, jest testowany na przykładowych grafach o liczbie wierzchołków n . Pomiarów są powtarzane trzy razy, a następnie obliczana jest z nich średnia, żeby zmniejszyć błędy pomiarów. Poniżej znajdują się wykresy empirycznie wyznaczonej złożoności obliczeniowej dla każdego algorytmu, przygotowane z wykorzystaniem programu Gnuplot.

Grafy wykorzystane do testów:

1. Graf będący cyklem Hamiltona – wykorzystany do algorytmów testowania zewnętrznej planarności, podziału na dwuspójne składowe, wyznaczania liczby wierzchołków w najdłuższym cyklu grafu zewnętrznie planarnego, oraz kolorowania grafów zewnętrznie planarnych.
2. Maksymalny graf zewnętrznie planarny – wykorzystany do algorytmu testowania maksymalnej zewnętrznej planarności.

Rozpoznawanie grafów zewnętrznie planarnych. Do testów wydajności wykorzystano grafy będące cyklem Hamiltona. Wartość współczynnika a na wykresie A.1 jest bliska 1, zatem wykonany test potwierdza złożoność liniową.

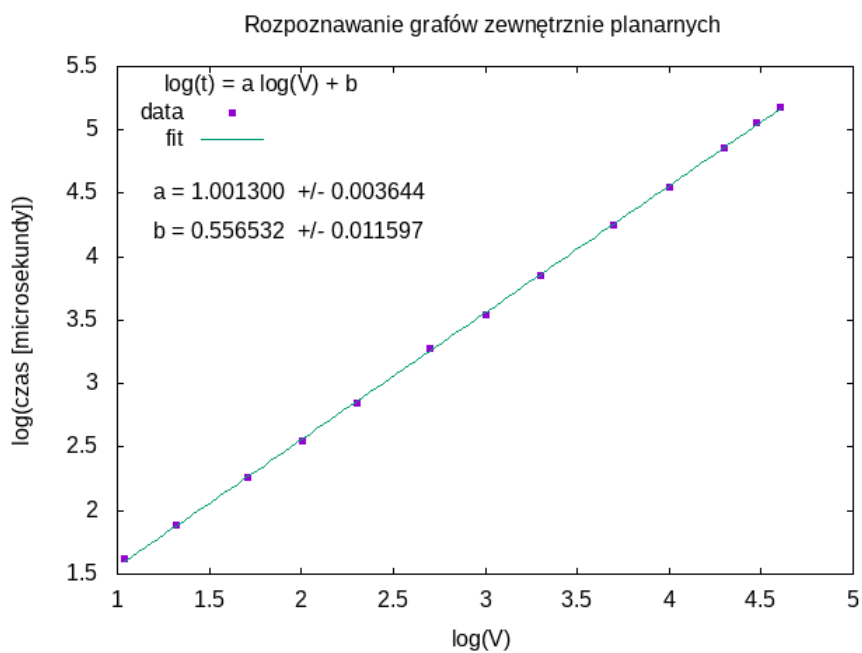
Podział grafu na dwuspójne składowe. Do testów wydajności również zostały użyte grafy będące cyklem Hamiltona. Wartość współczynnika a na wykresie A.2 jest bliska 1, zatem wykonany test potwierdza złożoność liniową.

Wyznaczanie najdłuższego cyklu w grafie zewnętrznie planarnym. W algorytmie wyznaczania liczby wierzchołków najdłuższego cyklu została wykorzystana procedura podziału na dwuspójne składowe, zatem sytuacja jest analogiczna jak w przypadku powyżej - współczynnik a na rysunku A.3 jest bliski 1, co prowadzi do złożoności liniowej.

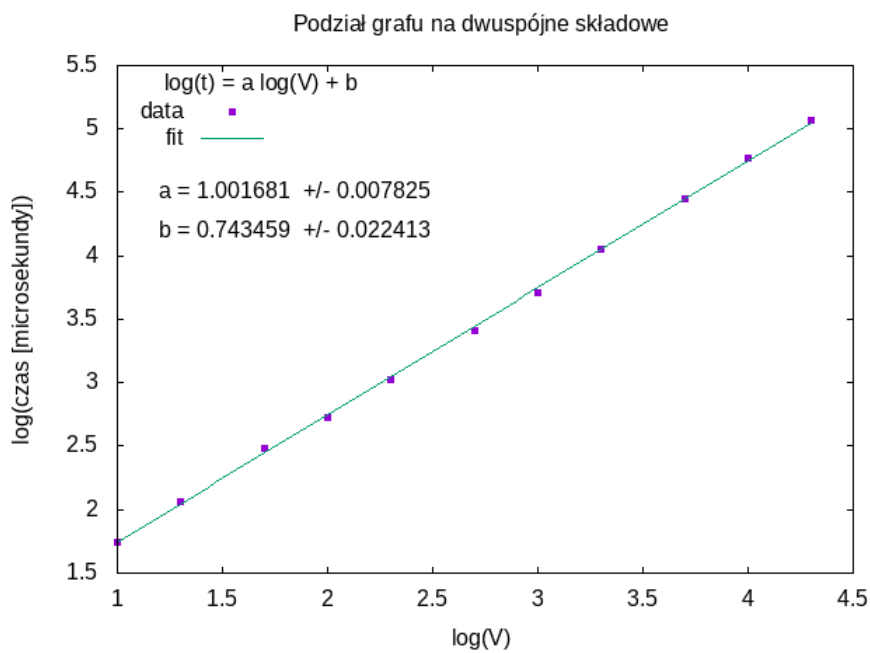
Kolorowanie wierzchołków w grafie zewnętrznie planarnym. Podczas kolorowania grafów zewnętrznie planarnych zostały wykorzystane podstawowe grafy należące do tej klasy. Były to cykle Hamiltona i tu również współczynnik a na rysunku A.4 jest bliski 1. Test skłania ku stwierdzeniu o złożoności liniowej algorytmu.

Rozpoznawanie maksymalnych grafów zewnętrznie planarnych. Do testowania wydajności rozpoznawania maksymalnych grafów zewnętrznie planarnych posłużył zaimplementowany w niniejszej pracy generator takich grafów. Współczynnik a na rysunku A.5 jest bliski 1, zatem test potwierdza złożoność liniową.

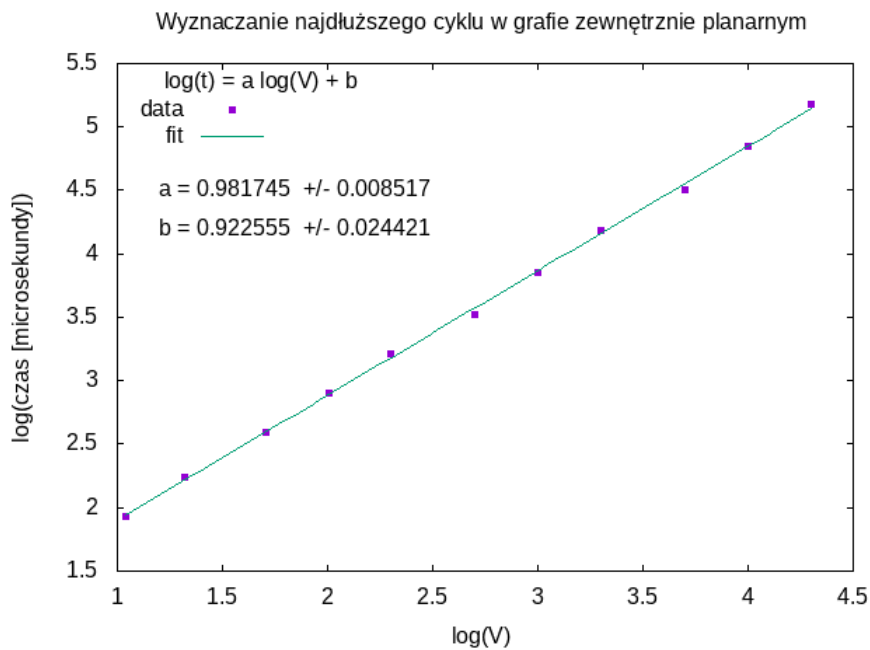
Generowanie maksymalnych grafów zewnętrznie planarnych. Do testu zostały wygenerowane grafy o coraz większej liczbie wierzchołków. Tu również złożoność została potwierdzona, ponieważ współczynnik a na rysunku A.6 jest bliski 1.



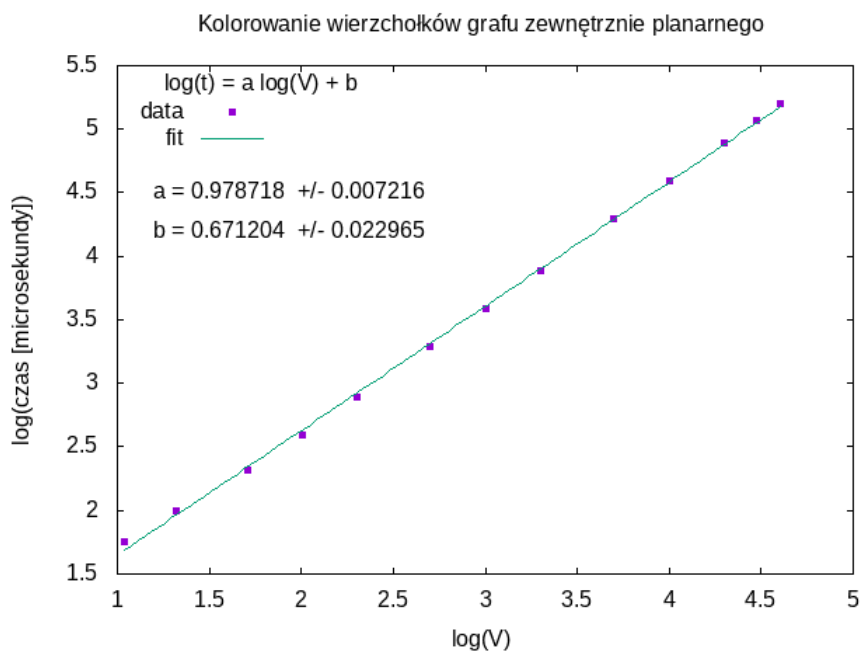
Rysunek A.1. Wykres wydajności algorytmu rozpoznawania grafów zewnętrznie planarnych. Współczynnik $a = 1.001(30)$ potwierdza liniową złożoność obliczeniową.



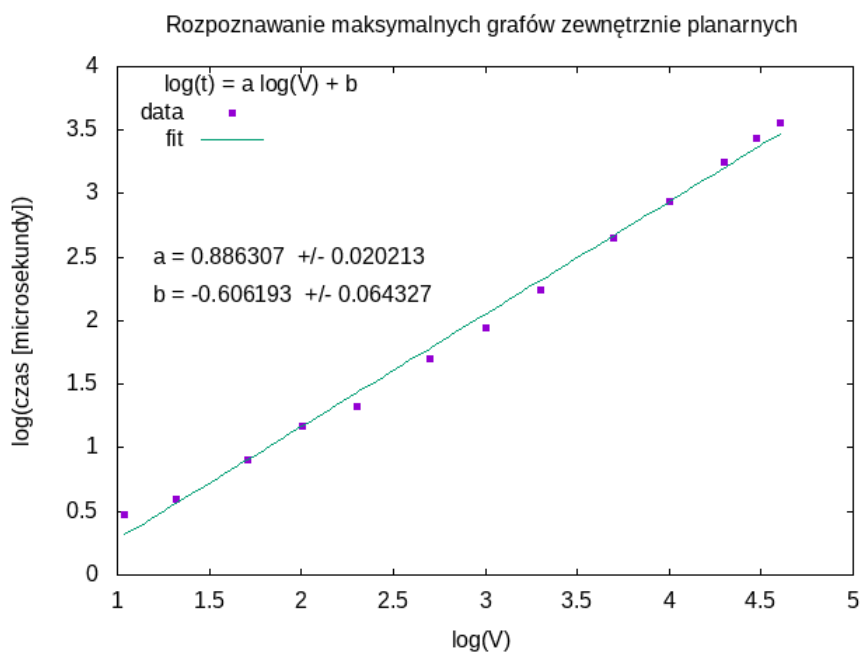
Rysunek A.2. Wykres wydajności algorytmu podziału grafu na dwuspójne składowe. Współczynnik $a = 1.001(7)$ potwierdza liniową złożoność obliczeniową.



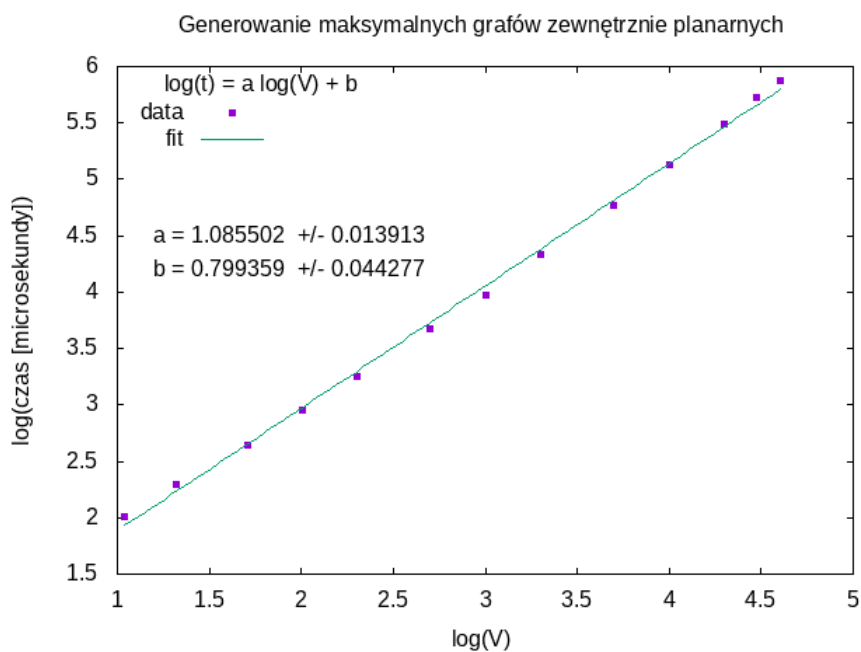
Rysunek A.3. Wykres wydajności algorytmu wyznaczania najdłuższego cyklu. Współczynnik $a = 0.981(8)$ potwierdza liniową złożoność obliczeniową.



Rysunek A.4. Wykres wydajności algorytmu kolorowania wierzchołków. Współczynnik $a = 0.979(7)$ potwierdza liniową złożoność obliczeniową.



Rysunek A.5. Wykres wydajności algorytmu rozpoznawania maksymalnych grafów zewnętrznie planarnych. Współczynnik $a = 0.886(4)$ potwierdza liniową złożoność obliczeniową.



Rysunek A.6. Wykres wydajności algorytmu generowania maksymalnych grafów zewnętrznie planarnych. Współczynnik $a = 1.086(14)$ potwierdza liniową złożoność obliczeniową.

Bibliografia

- [1] Wikipedia, Outerplanar graph, 2021,
https://en.wikipedia.org/wiki/Outerplanar_graph.
- [2] Gary Chartrand, Frank Harary, *Planar permutation graphs*, Annales de l'Institut Henri Poincaré B 3(4), 433-438 (1967).
- [3] Andrzej Kapanowski, graphs-dict, GitHub repository, 2021,
<https://github.com/ufkapano/graphs-dict/>.
- [4] Python Programming Language - Official Website,
<https://www.python.org/>.
- [5] Wikipedia, Bipartite graph, 2021,
https://en.wikipedia.org/wiki/Bipartite_graph.
- [6] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein, *Wprowadzenie do algorytmów*, Wydawnictwo Naukowe PWN, Warszawa 2012.
- [7] Robin J. Wilson, *Wprowadzenie do teorii grafów*, Wydawnictwo Naukowe PWN, Warszawa 1998.
- [8] Jacek Wojciechowski, Krzysztof Pieńkosz, *Grafy i sieci*, Wydawnictwo Naukowe PWN, Warszawa 2013.
- [9] Victor Bryant *Aspekty kombinatoryki*, Wydawnictwa Naukowo-Techniczne, Warszawa 1997.
- [10] Manfred Wieggers, *Recognizing Outerplanar Graphs in Linear Time*, WG 1986: str. 165-176.
- [11] Sandra L. Mitchell, *Linear Algorithms to recognize outerplanar and maximal outerplanar graphs*, Information Processing Letters 9(5), 229-232 (1979).
- [12] G. Chartrand, F. Harary, *Planar permutation graphs*, Ann. Inst. Henri Poincaré 3, 433-438 (1967).
- [13] Andrzej Proskurowski, Maciej M. Sysło, *Efficient vertex-and edge-coloring of outerplanar graphs*, SIAM Journal on Algebraic and Discrete Methods, 7: 131-136 (1986).
- [14] J. Hopcroft, R. Tarjan, *Algorithm 447: efficient algorithms for graph manipulation*, Communications of the ACM, 372-378 (1973).
- [15] Stanley Fiorini, *On the chromatic index of outerplanar graphs*, Journal of Combinatorial Theory, Series B, 18(1), 35-38 (1975).
- [16] V.G.Vizing, *On an estimate of the chromatic class of ap-graph* (Russian), Diskret.Analiz 3 (1964), 25-30.