Projekt na temat algorytmów grafowych

Piotr Gugnowski, Dawid Kozaczuk, Karol Gutkowski, Kamil Kamiński

Teoria Algorytmów i Obliczeń

Spis treści

Opracowanie pojęć

Rozmiar grafu

Przyjmujemy, że rozmiar grafu to ilość krawędzi w grafie.

Cykl w grafie

To ciąg wierzchołków, takich, że dla każdego wierzchołka z tego ciągu istnieje krawędź do wierzchołka występującego w ciągu bezpośrednio po nim. Poza tym istnieje krawędź z ostatniego wierzchołka w ciągu do pierwszego wierzchołka w ciągu.

Maksymalny cykl w grafie

Maksymalny cykl w grafie to cykl, który zawiera najwięcej wierzchołków spośród wszystkich cykli w danym grafie. Może istnieć wiele takich cykli, które mają tę samą, maksymalną liczbę wierzchołków. Liczbę takich cykli nazywamy liczbą maksymalnych cykli w grafie.

Cykl Hamiltona

Cykl Hamiltona to taki cykl w grafie, który zawiera każdy wierzchołek z grafu dokładnie raz.

Minimalne rozszerzenie grafu do grafu zawierającego cykl Hamiltona

Mówiąc o minimalnym rozszerzeniu mamy na myśli rozszerzanie grafu do grafu zawierającego cykl Hamiltona. Przyjmujemy, że minimalne rozszerzenie to taki zbiór krawędzi, który zawiera najmniejszą ilość krawędzi, które należy dodać aby graf zawierał skierowany cykl Hamiltona.

Liczba cykli Hamiltona

Przymujemy, że liczbą cykli Hamiltona będziemy nawyzwać liczbę wszystkich możliwych unikalnych skierowanych cykli Hamiltona w grafie to znaczy, dla których kolejność odwiedzanych wierzchołków jest różna.

Metryka

Na przestrzeni grafów przyjmujemy, że funckja odległości d między grafami G_1 i G_2 zdefinowana jest następująco:
$d(G_1, G_2)$ = minimalna liczba zmian, których trzeba dokonać, aby otrzymać graf G_1 z grafu G_2 (uwzględniając równość izomorficzną). Te zmiany to:

- dodanie wierzchołka do G_2
- usunięcie wierzchołka z G_2
- dodanie krawędzi do G_2
- usunięcie krawędzi z G_2

Metryka - dowód poprawności: nieujemność

Liczba zmian nie może być oczywiście ujemna, więc dla każego G_1 i G_2 mamy: $d(G_1, G_2) \geq 0$, co należało pokazać.

Metryka - dowód poprawności: symetria

Dla G_1 i G_2 wartość $d(G_1, G_2)$ jest sumą:

- libczy mian utożsamianych z dodaniem wierzchołka będzie tyle ile jest wierzchołków w G_1, których nie ma w G_2 ($|V(G_1) \setminus V(G_2)|$).
- liczby zmian utożsamianych z usunięciem wierzchołka będzie tyle ile jest wierzchołków w G_2, których nie ma w G_1 ($|V(G_2) \setminus V(G_1)|$).
- liczby mian utożsamianych z dodaniem krawędzi będzie tyle ile jest krawędzi w G_1, których nie ma w G_2 ($|E(G_1) \setminus E(G_2)|$).
- liczby mian utożsamianych z usunięciem krawędzi będzie tyle ile jest krawędzi w G_2, których nie ma w G_1 ($|E(G_2) \setminus E(G_1)|$).

Metryka - dowód poprawności: symetria cd.

Natomiast gdyby w identycznej kolejności rozważyć składowe wartości $d(G_2, G_1)$ to okazało by się, że pierwsza składowa byłaby taka sama jak punkt drugi, druga taka sama jak punkt pierwszy, trzecia składowa taka sama jak punkt czwarty, czwarta składowa taka sama jak punkt trzeci.
Stąd widać, że wartości $d(G_1, G_2)$ i $d(G_2, G_1)$ są opisane przez te same sumy liczby zmia, co należało pokazać.

Metryka - dowód poprawności: nierówność trójkąta

Należy pokazać, że dla dowolnych grafów G_1, G_2, G_3 mamy:

$$d(G_1, G_3) \leq d(G_1, G_2) + d(G_2, G_3).$$

Załóżmy, że prawdą jest, że:

$$d(G_1, G_3) \geq d(G_1, G_2) + d(G_2, G_3).$$

Metryka - dowód poprawności: nierówność trójkąta cd.

Z lewej mamy liczbę minimalną liczbę zmian potrzebnych do przekształcenia G_3 do G_1.
Z prawej strony mamy mamy minimalną liczbę zmian potrzebnych do uzyskania G_1 przekształcając G_2 oraz minimalną liczbę zmian potrzebnych do uzyskania G_2 z G_3.
Stąd możemy zauważyć, że po prawej stronie mamy podaną formułę uzyskanie G_1 przekształcając G_3 (napierw do G_2, a z G_2 do G_1).
Skoro tak, to mamy sposób, który pozwoli uzyskać liczbę zmian mniejszą niż minimalna liczba zmian z lewej strony nierówności, co jest sprzecznością.
Skoro założenie jest fałszywe to oryginalna nierówność jest prawdziwa, co należało pokazać.

Znajdowanie minimalnego rozszerzenia do grafu zawierającego cykl Hamiltona

oraz liczbę cykli Hamiltona w rozszerzonym grafie

Inicjalizacja algorytmu rekurencyjnego

Algorytm polega na zbadaniu każdej możliwej permutacji wszystkich wierzchołków w grafie oraz wyznaczeniu, między którymi wierzchołkami w poszczególnych ciągach brakuje krawędzi

- Algorytm jest wywołany iteracyjnie dla każdego wierzchołka grafu, na początku i-tej iteracji do aktualnego ciągu wierzchołków zostaje dodany wierzchołek i.
- Następnie dla innego wierzchołka następuje sprawdzenie, czy istnieje krawędź do niego z wierzchołka dodanego do ciągu. Jeśli nie, taka krawędź zostaje dodana do aktualnego rozszerzenia.
- Badany wierzchołek zostaje dodany do aktualnej listy wierzchołków, wywoływana jest funkcja rekurencyjna.
- Czynność jest powtarzana dla wszystkich wierzchołków różnych od i-tego.

Funkcja rekurencyjna - kolejne wywołania

Funkcja rekurencyjna na wejściu otrzymuje aktualny ciąg wierzchołków oraz listę brakujących krawędzi między sąsiednimi elementami ciągu. Następnie kolejno dla wszystkich wierzchołków spoza aktualnego ciągu następuje sprawdzenie, czy istnieje krawędź do niego z wierzchołka dodanego do ciągu. Jeśli nie, taka krawędź zostaje dodana do aktualnego rozszerzenia. Badany wierzchołek zostaje dodany do aktualnej listy wierzchołków, wywoływana jest funkcja rekurencyjna.

Funkcja rekurencyjna - warunek stopu

Jeśli aktualny ciąg zawiera wszystkie wierzchołki grafu, nie istnieje możliwość dalszych wywołań funkcji, wówczas podejmowane są następujące działania:

- W przypadku gdy aktualna lista brakujących krawędzi jest pusta, liczba znalezionych cykli Hamiltona jest zwiększana o 1 (pod warunkiem, że nie znaleźliśmy już takiego)
- Jeżeli liczba znalezionych cykli Hamiltona wynosi 0, to porównujemy najmniejsze dotychczas znalezione rozszerzenie z aktualnym, wybieramy mniejsze z nich (lub aktualne jeśli wcześniej nie mieliśmy żadnego rozszerzenia)

Algorytm dokładny

Złożoność algorytmu dokładnego

Złożoność powyższego dokładnego algorytmu rekurencyjnego to O(n!). Związane jest to z faktem, że analizowana jest każda permutacja wierzchołków w grafie.

Testy wydajności - Algorytm dokładny

Czas wykonania [ms]							
Ilość wierzchołków	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Graf pełny	0,0	0,1	0,4	2,2	15,3	118,2	1042,9
Graf losowy |V|/2-regularny	0,0	0,1	0,4	1,9	13,4	95,3	857,1

Rysunek 1: Czas dokładnego wyszukiwania (w milisekundach)

Znajdowanie minimalnego rozszerzenia grafu

Algorytm polega na wielokrotnym wykonaniu (zależnym od średniej ilości krawędzi wychodzących i retryFactor) podążania losową ścieżką po grafie.

- Najpierw wybierany jest losowy wierzchołek.
- Następnie wybierany jest losowy nieodwiedzony sąsiad wybranego wierzchołka po czym następuje przejście do tego sąsiada.
- Przejścia są powtarzane do momentu, aż w trakcie działania procedury znajdzie się wierzchołek, który nie ma krawędzi wychodzących do nieodwiedzonego wierzchołka. Wtedy następuje dodanie krawędzi.
- Krawędź pozwala przejść dalej do nieodwiedzonego wierzchołka z najmniejszą ilością krawędzi wchodzących z nieodwiedzonych do tej pory wierzchołków.

Znajdowanie minimalnego rozszerzenia grafu cd.

- Po takim przejściu następuje kontynuowanie podążanie losową ścieżką.
- Gdy odwiedzone zostaną wszystkie wierzchołki następuje przejście do początkowego wierzchołka. Jeśli brakuje krawędzi to jest dodana.
- Rozszerzenie grafu to zbiór dodanych krawędzi podczas procedury budowania ścieżki.
- Wybierany jest najmniej liczby zbiór spośród każdej próby budowania ścieżki.

Znajdowanie liczby cykli Hamiltona w rozszerzonym grafie

Algorytm polega na zliczaniu ścieżek, którymi udało się przejść podczas próby przejścia przez wszystkie wierzchołki dokładnie raz oraz takich, dla których istnieje krawędź w grafie od ostatniego do pierwszego wierzchołka. Szukanie ścieżki jest wykonywane wielokrotnie (ilość wywołań zależy od rozmiaru grafu i ln(retryFactor)), każda unikatowa ścieżka jest zapisywana. Na początku wiadomo o jednym cyklu Hamiltona, tym którym podążano przy tworzeniu najmniejszego rozszerzenia do grafu zawierającego cykl Hamiltona.

Znajdowanie liczby cykli Hamiltona w rozszerzonym grafie: bez rozszerzenia

- Jeśli grafu nie trzeba było rozszerzać, aby zawierał cykl Hamiltona każdy punkt rozpoczęcia budowania ścieżki jest równie dobry, dlatego też każda ścieżka zaczyna się od wierzchołka o indeksie 0.
- Wykonywana jest procedura podążania losową ścieżką.
- W przypadku przejścia do wierzchołka, z którego nie można już przejść do nieodwiedzonego wierzchołka sprawdzane jest czy wszystkie wierzchołki zostały odwiedzone i czy istneje krawędź w grafie z ostatniego do pierwszego wierzchołka. Jeśli warunki są spełnione to cykl jest dodawany do zbioru unikalnych cykli.

Znajdowanie liczby cykli Hamiltona w rozszerzonym grafie: z rozszerzeniem

Jeśli graf trzeba było rozszerzyć to jedna z dodanych krawędzi była niezbędna, aby w ogóle jakikolwiek cykl Hamiltona istniał w grafie. Dlatego też opłaca się rozpoczynać budowę ścieżek od krawędzi należących do rozszerzenia grafu. Dalej algorytm działa tak jak dla grafu, którego nie trzeba było rozszerzać.

Uwagi

Uzasadnienie będzie polegać na podaniu rzeczywistych kroków podczas wykonywania algorytmu i dopisywaniu po kroku jego złożoności. Przyjęto, że $G = (V, E)$, $|V| = n$, $|E| = m$. Finalna złożoność jest łatwa do zauważenia przez wybranie największego iloczynu złożoności kroku i złożoności pętli, w których dany krok się znajduje.

Uzasadnienie złożoności

Znajdowanie minimalnego rozszerzenia grafu $O(n^3)$

- Wyznaczana jest liczba powtórzeń procedury budowania ścieżki. Jest to średnia ilość krawędzi wychodzących w grafie zaokrąglona w górę (dla 0 krawędzi 1) pomnożona przez współczynnik powtórzeń (retryFactor). O(n)
- ...

Znajdowanie minimalnego rozszerzenia grafu cd.

- Wykonanie poniższej logiki tyle razy ile wynosi wyznaczona liczba powtórzeń O(n)
  - Wyznaczany jest wierzchołek startowy losowo. O(1)
  - Wykonanie procedury **podążania losową ścieżką** O(n^2) (Wyjaśnienie poniżej)
  - Sprawdzenie czy podążanie nową ścieżką wymagała najmniejszej ilości dodanych krawędzi do grafu. Jeśli nie to przejście do kolejnej iteracji. O(1)
  - Przypisanie dodanych krawędzi jako najmniejszego rozszerzenia. O(1)
  - Ustawienie cyklu tak, by zaczynał się od najmniejszego indeksu O(n)
  - Dodanie przebytej ścieżki do zbioru cykli Hamiltona odpowiadających najmniejszemu rozszerzeniu O(1)
  - Przerwanie pętli jeśli najmniejsze rozszerzenie ma 0 krawędzi. O(1)

Podążanie losową ścieżką $O(n^2)$

Jest to funkcja rekurencyjna, która przyjmuje referencję do zbioru odwiedzonych wierzchołków, wierzchołek startowy, listę relacji i ilość wierzchołków grafu, zbiór dodanych krawędzi, wektor kolejno odwiedzonych wierzchołków, obecnie "odwiedzany"wierzchołek. Obecnie odwiedzany wierzchołek musi być wcześniej nieodwiedzony więc górnym ograniczeniem ilości wywołań funkcji jest ilość wierzchołków w grafie. O(n) (W wywołaniu wykonywana jest poniższa logika.)

- ...

Podążanie losową ścieżką cd.

- Dodaj obecnie odwiedzany wierzchołek do zbioru odwiedzonych wierzchołków. O(1)
- Dodaj obecnie odwiedzany wierzchołek na koniec obecnie tworzonej ścieżki O(1)
- Sprawdź czy w grafie istnieje krawędź do wierzchołka startowego z "odwiedzanego"wierzchołka. O(1)
- Wybierz losowego nieodwiedzonego sąsiada odwiedzanego wierzchołka. O(n)
- Usuń krawędzie powiązane z odwiedzanym wierzchołkiem, jeśli odwiedzany wierzchołek nie jest wierzchołkiem startowym. O(n)
- ...

Podążanie losową ścieżką cd.

- Jeśli nie udało się wybrać losowego sąsiedniego i nieodwiedzonego wierzchołka to wybierany jest losowy wierzchołek spośród nieodwiedzonych, który ma najmniej krawędzi wchodzących z nieodwiedzonych wierzchołków. O(n)
- Jeśli nie udało się znaleść takiego wierzchołka to znaczy, że odwiedzono już wszyskie wierzchołki. Jeżeli brakuje krawędzi do początkowego wierzchołka to jest ona dodawana. Rekurencja się kończy. O(1)
- Jeśli udało się znaleść taki wierzchołek to dodawana jest krawędź od odwiedzanego wierzchołka do wybranego wierzchołka. O(1)
- Wywołana jest rekurencja dla wybranego w tym wywołaniu następnego wierzchołka. O(1)

Znajdowanie cykli hamiltona w rozszerzonym grafie $O(n^5)$

Pierwszy cykl Hamiltona był znaleziony podczas znajdowania mi minimalnego rozszerzenia grafu. Jeśli graf nie wymaga rozszerzenia by istniał w nim cykl Hamiltona to wykonana jest poniższa procedura.

- Wyznaczona jest liczba iteracji, będąca ilością krawędzi w grafie pomnożoną przez współczynnik retryFactor. O(n)
- ...

Znajdowanie cykli hamiltona w rozszerzonym grafie cd.

- Wykonanie poniższych kroków tyle razy ile wynosi liczba iteracji. $O(n^2)$
  - Wywołane jest podążanie losową ścieżką z wierzchołka o indeksie 0 w grafie. Różnica względem poprzedniego podążania losową ścieżką jest taka, że nie są usuwane krawędzie wychodzące z odwiedzonego wierzchołka oraz niepowodzenie przy wyborze sąsiedniego nieodwiedzonego wierzchołka kończy procedurę. $O(n^2)$
  - Jeśli wyznaczona ścieżka odwiedza wszystkie wierzchołki i istnieje krawędź w grafie z ostatniego do wierzchołka 0 to ścieżka ta jest dodawana do zbioru unikalnych cykli Hamiltona znalezionego rozszerzenia grafu. $O(1)$

Znajdowanie cykli hamiltona w rozszerzonym grafie cd.

Do szukania pozostałych cykli jeśli minimalne rozszerzenie zawiera krawędź wykorzystana jest poniższa procedura.

- Do grafu dodane są krawędzie ze znalezionego rozszerzenia. O(n)
- Dla każdej krawędzi z rozszerzenia wykonana jest poniższa logika. O(n)
  - Wyznaczona jest liczba iteracji będąca iloczynem retryFactor i ilości krawędzi w grafie. O(1)
  - ...

Znajdowanie cykli hamiltona w rozszerzonym grafie cd.

- Dla każdej iteracji wykonane są poniższe kroki. $O(n^2)$
  - Wykonane jest podążanie losową ścieżką w taki sposób jak dla przypadków, w których nie było dodatkowych krawędzi w rozszerzeniu grafu, z tym, że w pierwszym przejściu zagwarantowane jest przejście po krawędzi z rozszerzenia, a wierzchołkiem startowym jest wierzchołek z którego wychodzi krawędź z rozszerzenia grafu. $O(n^2)$
  - Jeśli wyznaczona ścieżka odwiedza wszystkie wierzchołki i istnieje krawędź w grafie z ostatniego do wierzchołka 0 to ścieżka ta jest dodawana do zbioru unikalnych cykli Hamiltona znalezionego rozszerzenia grafu. $O(1)$

Dowód - Znajdowanie minimalnego rozszerzenia grafu

Jako, że jest to algorytm aproksymacyjny to zamiast dowiedzenia, że algorytm znajduje minimalne rozszerzenie grafu zostanie pokazane, że algorytm znajduje rozszerzenie grafu do grafu zawierającego cykl Hamiltona.

Dowód - Znajdowanie minimalnego rozszerzenia grafu cd.

Przyjęto, że $G = (V, E)$, $|V| = n$, $|E| = m$.
Rozważmy pierwsze powtórzenie, wykonana jest wtedy procedura podążania losową ścieżką.
W trakcie wywołania tej procedury został wyznaczony zbiór krawędzi, który w pierwszej iteracji zostanie przypisany jako najmniejsze rozszerzenie. Każde następne wywołanie iteracji co najwyżej może podać mniejszy zbiór krawędzi, ale również mający tą własność, co poprzedni zbiór, czyli **będący rozszerzeniem grafu do grafu z cyklem Hamiltona**.

Dowód - wykonanie procedury podążania losową ścieżką zwróci rozszerzenie grafu do grafu z cyklem Hamiltona

Jest to procedura rekurencyjna, która kończy się dopiero wtedy jeśli niemożliwe będzie wybranie losowego wierzchołka spośród nieodwiedzonych, który ma najmniej krawędzi wchodzących z nieodwiedzonych wierzchołków. Oznacza to, że w trakcie wykonania procedury został odwiedzony każdy wierzchołek.
Za każdym razem kiedy wybierany jest następny wierzchołek to jest on wybierany spośród nieodwiedzonych wierzchołków.
Z poprzednich obserwacji można zauważyć, że każdy wierzchołek jest odwiedzany dokładnie raz.

Dowód - wykonanie procedury podążania losową ścieżką zwróci rozszerzenie grafu do grafu z cyklem Hamiltona cd.

Kolejność odwiedzanych wierzchołków tworzona przy każdym wywołaniu rekurencyjnym poprzez zapisanie odwiedzanego wierzchołka na koniec "obecnie tworzonej ścieżki". Także kolejność odwiedzonych wierzchołków wraz z krawędzią od ostatniego do pierwszego wierchołka (krawędź ta jeśli nie istniała w grafie to tuż przed zakończeniem rekurencji jest ona dodawana do rozszerzenia grafu) tworzą cykl Hamiltona w grafie, do którego dodawano krawędzie za każdym razem kiedy nie udało się wybrać losowego sąsiedniego i nieodwiedzonego wierzchołka. Te dodane krawędzie są właśnie szukanym rozszerzeniem grafu.

Dowód - znajdowanie liczby cykli Hamiltona

Jako, że jest to algorytm aproksymacyjny to nie zostaną odnalezione wszystkie cykle Hamiltona, można natomiast pokazać, że znajdowane listy wierzchołków dodawane do zbioru cykli rzeczywiście reprezentują istniejące w grafie cykle Hamiltona.

Dowód - znajdowanie liczby cykli Hamiltona cd.

W przypadku wykonania tego algorytmu występują dwie sytuacje, szukanie cykli dla grafu, który miał cykl Hamiltona bez rozszerzania go oraz, szukanie cyklu dla grafu, do którego zostało dodane rozszerzenie zawierające przynajmniej jedną krawędź. Natomiast w obu przypadkach wykonywana jest procedura podążania losową ścieżką oraz dodanie tej ścieżki do zbioru unikalnych cykli Hamiltona. Różnica polega tylko na ilości prób szukania tych cykli oraz wyborze wierzchołka startowego (oraz w tym, że dla przypadku, gdy rozszeszenie grafu zawiera krawędzie drugi w kolejności odwiedzany wierzchołek będzie drugim wierzchołkiem do którego wchodzi rozważana krawędź z rozszerzenia grafu). Należy więc się przyjżeć procedurze podążania losową ścieżką, aby uwidocznić, że dodawane ścieżki do zbioru unikalnych cykli Hamiltona rzeczywiście reprezentują istniejący w grafie cykl Hamiltona.

Dowód - znajdowanie liczby cykli Hamiltona cd.

Procedura podążania losową ścieżką jest rekurencyjna, odwiedzane są tylko wcześniej nieodwiedzone wierzchołki, a jako następny w kolejności wierzchołek wybierany jest tylko wierzchołek sąsiadujący. Jeśli takiego sąsiadującego wierzchołka nie ma to rekurencja zostaje zakończona. Wyznaczona ścieżka reprezentuje cykl Hamiltona jeśli więc wszystkie wierzchołki zostały odwiedzone i istnieje krawędź od ostatniego do pierwszego wierzchołka (z tej ścieżki) w grafie. A dokładnie te warunki są dodatkowo sprawdzane po wywołaniu procedury podążania losową ścieżką przy próbie dodania ścieżki do zbioru unikalnych cykli Hamiltona. Tak więc rzeczywiście wszystkie ścieżki znajdujące się w zbiorze cykli Hamiltona poprawnie reprezentują cykle Hamiltona.

Znajdowanie maksymalnego cyklu w grafie oraz liczby maksymalnych cykli w grafie.

Inicjalizacja algorytmu rekurencyjnego

Algorytm polega na zbadaniu każdej możliwej ścieżki o różnych wierzchołkach w grafie, znalezieniu w ten sposób cykli oraz wyznaczeniu najdłuższego z nich

- Algorytm jest wywołany iteracyjnie dla każdego wierzchołka grafu, na początku i-tej iteracji do aktualnego ciągu wierzchołków zostaje dodany wierzchołek i.
- Następnie kolejno dla każdego sąsiada i-tego wierzchołka następuje dodanie go do aktualnej ścieżki i wywoływanie funkcji rekurencyjnej.

Funkcja rekurencyjna - kolejne wywołania

Funkcja rekurencyjna na wejściu otrzymuje aktualny ciąg wierzchołków - dotychczasową ścieżkę. Następnie sprawdzamy wszystkich sąsiadów ostatniego wierzchołka z ciągu. Dla każdego z nich, który nie znajduje się w aktualnym ciągu, następuje dodanie go do ciągu i wywoływanie funkcji rekurencyjnej.

Funkcja rekurencyjna - warunek stopu

Jeśli jednym z sąsiadów ostatniego wierzchołka ciągu jest pierwszy wierzchołek ciągu, udało się znaleźć cykl. Wówczas jego długość jest porównywana z największą dotychczasową. Na tej podstawie odpowiednio aktualizowana jest długość najdłuższego cyklu oraz lista najdłuższych cykli.

Złożoność algorytmu dokładnego

Złożoność powyższego dokładnego algorytmu rekurencyjnego to O(n!). Związane jest to z faktem, że w pesymistycznym przypadku analizowana jest każda permutacja wierzchołków w grafie.

Wstępne przekształcenie grafu

- Przed rozpoczęciem przeszukiwania grafu w głąb usuwamy z grafu wszystkie liście, do momentu aż będzie się składał z samych cykli.
- Celem tego kroku jest pominięcie niepotrzebnych wywołań przeszukiwania oraz unikanie przeszukiwań ścieżek będących „ślepymi uliczkami”.

Struktury danych

Tworzymy pomocnicze tablice:

- **Numbers** - przechowuje odległość od początku przeszukiwania. Informacja ta pozwala wyznaczyć długość cyklu przy napotkaniu już odwiedzonego wierzchołka.
- **Parent** - przechowuje numer wierzchołka, z którego dotarliśmy do aktualnego wierzchołka. Początek przeszukiwania oraz nieodwiedzone wierzchołki mają przypisaną wartość -1.
- **Visited** - przechowuje informację o tym, czy dany wierzchołek został już odwiedzony podczas przeszukiwania.

Zmienne pomocnicze

Algorytm wykorzystuje także:

- **lastLargestCycle** - przechowuje długość najdłuższego znalezionego do tej pory cyklu.
- **Resulting_cycles** - przechowuje wszystkie dotychczas znalezione cykle maksymalnej długości.

Rozpoczęcie przeszukiwania

Dla każdego wierzchołka grafu:

- Resetujemy dane pomocnicze (oprócz danych o znalezionych dotychczas cyklach).
- Rozpoczynamy przeszukiwanie w głąb od wierzchołka startowego, przypisując mu wartość 1 w tablicy **Numbers**.
- Kolejne wierzchołki są wybierane w kolejności losowej, aby zwiększyć różnorodność przeszukiwanych tras.

Przeszukiwanie w głąb

- Każdemu nieodwiedzonemu sąsiadowi przyporządkowujemy bieżący wierzchołek jako rodzica w tablicy **Parent**.
- Aktualizujemy tablicę **Numbers**, przypisując bieżacemu wierzchołkowi numer o jeden większy niż jego własny rodzic (o ile nie jest wierzchołkiem startowym).
- Aktualizujemy tablicę **Visited**, aby oznaczyć bieżący wierzchołek jako odwiedzony.

Wykrywanie cyklu

- Gdy napotkamy odwiedzonego już sąsiada, sprawdzamy długość cyklu od niego do aktualnej pozycji na podstawie tablicy **Numbers**.
- Jeśli długość cyklu jest większa lub równa najdłuższemu dotychczas znalezionemu, weryfikujemy jego poprawność, cofając się po tablicy **Parent**.
- Poprawny cykl:
  - Jeśli jest większy, aktualizujemy długość **lastLargestCycle** i resetujemy kolekcję cykli.
  - Jeśli jest równy, dodajemy go do kolekcji, jeśli nie jest duplikatem.

Zakończenie przeszukiwania

- Po przeszukaniu grafu z każdego możliwego wierzchołka kończymy działanie algorytmu.
- Wynikiem jest długość najdłuższego znalezionego cyklu oraz kolekcja cykli o tej długości.

Pesymistyczna analiza złożoności

1. Algorytm wykonuje przeszukiwanie w głąb n razy.
2. Wewnątrz każdej iteracji DFS:
   - Każdy wierzchołek (n) oraz każda krawędź (m) są odwiedzane dokładnie raz.
   - Jeśli przy danej krawędzi sprawdzamy potencjalny cykl:
     - Sprawdzenie poprawności cyklu: $O(n)$.
     - Porównanie cyklu z kolekcją wynikową: $O(n \cdot m)$ w najgorszym przypadku.
3. W najgorszym przypadku dla każdej krawędzi wykonujemy $O(n + n \cdot m)$, co daje:

$$O(n + m \cdot (n + n \cdot m)) = O(n^2 \cdot m^2).$$

Średnia analiza złożoności

1. Przy pierwszych iteracjach DFS algorytm znajdzie duży cykl.
2. Kolejne znalezione cykle rzadko wymagają sprawdzenia poprawności lub porównania z kolekcją.
3. Dominującym kosztem jest n przeszukań w głąb, gdzie każda iteracja to $O(n+m)$, co daje:

$$O(n \cdot (n+m)) = O(n^2 + n \cdot m) = O(n^3).$$

Testy wydajności - porównanie

Czas wykonania [ms]	Ilość wierzchołków	2	3	4	5	6	7	8	10	15	20	30	50
Graf pełny	Algorytm dokładny	0,1	0,1	0,5	2,8	20,4	248,3	7195,8	-	-	-	-	-
	Algorytm aproksymacyjny	0,1	0,1	0,1	0,2	0,3	0,4	0,5	0,9	2,4	5,0	14,7	63,0
Graf losowy |V|/2-regularny	Algorytm dokładny	0,0	0,0	0,2	0,3	1,7	3,8	32,3	-	-	-	-	-
	Algorytm aproksymacyjny	0,1	0,1	0,1	0,2	0,2	0,3	0,4	0,6	1,4	3,1	8,4	35,2

Rysunek 2: Porównanie czasów algorytmu dokładnego i aproksymacyjnego (w milisekundach)

Testy wydajności - porównanie

Długość najdłuższego cyklu	Ilość wierzchołków	2	3	4	5	6	7	8	10	15	20	30	50
Graf pełny	Algorytm dokładny	2	3	4	5	6	7	8	-	-	-	-	-
	Algorytm aproksymacyjny	2	3	4	5	6	7	8	10	15	20	30	50
Graf losowy |V|/2-regularny	Algorytm dokładny	2	2	4	4	6	7	8	-	-	-	-	-
	Algorytm aproksymacyjny	2	2	4	4	6	7	8	10	15	20	30	50

Rysunek 3: Porównanie rozmiarów największego znalezionego cyklu

Dowód poprawności

Cel algorytmu

Jako, że jest to algorytm aproksymacyjny, zamiast dowiedzenia, że algorytm znajduje wszystkie maksymalne cykle w grafie, zostanie pokazane, że algorytm znajduje poprawne cykle w grafie o ile takie istnieją, oraz rozważa bardzo różne trasy, na których mogą istnieć potencjalne cykle.

Usuwanie liści z grafu

Proces transformacji grafu, w którym usuwane są krawędzie przylegające do liści, pozostawia graf zawierający jedynie cykle (jeśli graf początkowo zawierał cykle).
W takim grafie, przeszukiwanie DFS z dowolnego wierzchołka:

- Może zakończyć się jedynie napotkaniem już odwiedzonego wcześniej wierzchołka (nie ma "ślepych uliczek").
- Gwarantuje to znalezienie cyklu, o ile w grafie pozostały jakiekolwiek cykle po usunięciu liści.

Z tego wynika, że algorytm zawsze znajdzie przynajmniej jeden cykl, jeśli istnieje w grafie przekształconym przez funkcję usuwającą liście.

Losowy wybór sąsiadów

Losowe wybieranie sąsiadów zwiększa różnorodność tras rozpatrywanych w przeszukiwaniu.
Dzięki temu algorytm ma większe szanse przeanalizować różne potencjalne ścieżki w grafie, podczas gdy:

- Dobieranie na podstawie np. stopnia wierzchołka lub otoczenia wierzchołka powodowałoby, że dla każdego wywołania DFS przeszukiwanie grafu zachowywałoby podobną kolejność wyboru sąsiadów.

Dla każdego wierzchołka w grafie takich losowań będzie tyle, ile wywołań DFS, czyli tyle, ile wierzchołków w grafie.
Oznacza to, że dla większych grafów rozważane jest więcej możliwości na jeden wierzchołek. Zapewnia to rozważenie bardzo różnych tras przy kolejnych przeszukiwaniach w głąb grafu.

Testy wydajności - Algorytm 1

Ilość wierzchołków	3				4				5				6				7				8				16				30
Rozmiar grafu (po ilości wierzchołków)	**0**	**2**	**3**	**6**	**0**	**5**	**8**	**12**	**0**	**6**	**7**	**20**	**0**	**13**	**15**	**30**	**0**	**22**	**24**	**42**	**0**	**26**	**30**	**56**	**0**	**121**	**126**	**240**	**790**
Najmniejsze znalezione rozszerzenie	3	2	2	0	4	1	0	0	5	2	1	0	6	1	1	0	7	0	0	0	8	0	1	0	16	0	0	0	0
Czas szukania rozszerzenia	8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	4
Ilość znalezionych cykli Hamiltona	1	1	1	2	1	1	1	6	1	1	1	13	1	1	1	27	1	3	8	41	1	8	4	57	1	24	28	241	101
Czas szukania cykli Hamiltona	0	0	0	0	1	0	0	1	2	1	0	2	3	1	1	4	5	3	3	8	8	4	4	13	72	59	68	215	1636
Łączny czas wykonania algorytmu	9	0	0	0	1	0	0	1	2	1	1	2	4	1	2	5	5	3	4	8	8	4	5	13	72	62	68	216	1641

Rysunek 4: Dane wydajności dla retryFactor=1 (czas w milisekundach)

Ilość wierzchołków	3				4				5				6				7				8				16				30
Rozmiar grafu (po ilości wierzchołków)	**0**	**2**	**3**	**6**	**0**	**5**	**8**	**12**	**0**	**6**	**7**	**20**	**0**	**13**	**15**	**30**	**0**	**22**	**24**	**42**	**0**	**26**	**30**	**56**	**0**	**121**	**126**	**240**	**790**
Najmniejsze znalezione rozszerzenie	3	2	1	0	4	1	0	0	5	2	1	0	6	1	1	0	7	0	0	0	8	0	0	0	16	0	0	0	0
Czas szukania rozszerzenia	0	0	0	0	0	0	0	1	0	0	1	2	0	1	2	4	0	3	3	7	0	3	0	12	1	21	23	78	297
Ilość znalezionych cykli Hamiltona	1	1	1	2	1	1	1	6	1	1	1	19	1	1	1	48	1	5	9	77	1	12	12	112	1	41	52	481	240
Czas szukania cykli Hamiltona	0	0	0	0	1	0	0	1	2	1	0	2	3	1	3	5	5	3	3	9	8	4	5	21	69	60	73	250	1719
Łączny czas wykonania algorytmu	0	0	0	1	1	1	1	2	2	2	1	5	3	3	5	10	6	6	7	17	9	8	5	33	71	82	97	329	2016

Rysunek 5: Dane wydajności dla retryFactor=5 (czas w milisekundach)

Testy wydajności - Algorytm 1

Ilość wierzchołków	3				4				5				6				7				8				16				30
Rozmiar grafu (po ilości wierzchołków)	**0**	**2**	**3**	**6**	**0**	**5**	**8**	**12**	**0**	**6**	**7**	**20**	**0**	**13**	**15**	**30**	**0**	**22**	**24**	**42**	**0**	**26**	**30**	**56**	**0**	**121**	**126**	**240**	**790**
Najmniejsze znalezione rozszerzenie	3	2	1	0	4	1	0	0	5	2	1	0	6	1	1	0	7	0	0	0	8	0	0	0	16	0	0	0	0
Czas szukania rozszerzenia	0	0	0	1	0	1	1	2	0	1	1	5	0	3	3	9	0	6	6	15	0	7	7	23	2	47	41	183	579
Ilość znalezionych cykli Hamiltona	1	1	1	2	1	1	1	6	1	1	1	22	1	1	1	79	1	8	12	148	1	14	23	219	1	83	111	961	474
Czas szukania cykli Hamiltona	0	0	0	0	2	0	1	2	4	2	1	5	7	2	3	12	11	6	7	23	17	9	11	41	138	129	140	639	3704
Łączny czas wykonania algorytmu	1	1	0	2	2	2	2	4	4	4	3	10	7	6	7	22	12	12	13	39	18	16	18	65	140	176	181	823	4283

Rysunek 6: Dane wydajności dla retryFactor=9 (czas w milisekundach)

Ilość wierzchołków	3				4				5				6				7				8				16				30
Rozmiar grafu (po ilości wierzchołków)	**0**	**2**	**3**	**6**	**0**	**5**	**8**	**12**	**0**	**6**	**7**	**20**	**0**	**13**	**15**	**30**	**0**	**22**	**24**	**42**	**0**	**26**	**30**	**56**	**0**	**121**	**126**	**240**	**790**
Najmniejsze znalezione rozszerzenie	3	2	1	0	4	1	0	0	5	2	1	0	6	1	1	0	7	0	0	0	8	0	0	0	16	0	0	0	0
Czas szukania rozszerzenia	0	0	0	1	0	2	2	3	0	2	2	8	1	5	5	16	1	8	9	29	1	10	11	46	2	63	67	397	931
Ilość znalezionych cykli Hamiltona	1	1	1	2	1	1	1	6	1	1	1	23	1	1	1	94	1	8	12	209	1	14	27	326	1	125	171	1441	719
Czas szukania cykli Hamiltona	0	0	0	0	2	0	1	2	4	2	1	5	7	2	3	14	11	6	8	30	17	9	11	54	139	135	154	932	4147
Łączny czas wykonania algorytmu	1	1	1	2	2	2	4	6	5	5	3	13	8	7	8	31	12	15	17	60	18	20	23	101	142	199	222	1330	5079

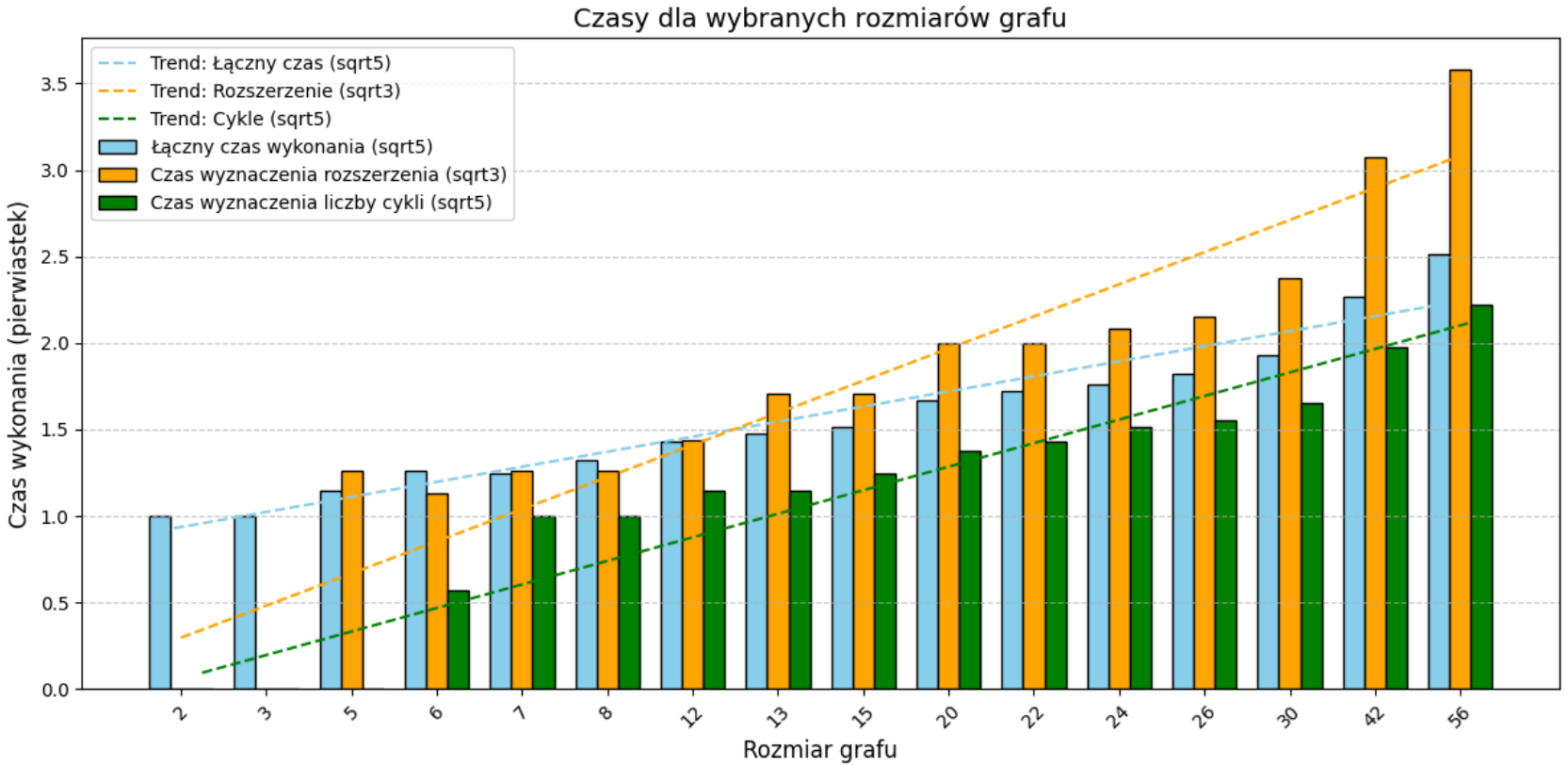
Rysunek 7: Dane wydajności dla retryFactor=13 (czas w milisekundach)

Testy wydajności - Algorytm 1

Ilość wierzchołków	3				4				5				6				7				8				16				30
Rozmiar grafu (po ilości wierzchołków)	**0**	**2**	**3**	**6**	**0**	**5**	**8**	**12**	**0**	**6**	**7**	**20**	**0**	**13**	**15**	**30**	**0**	**22**	**24**	**42**	**0**	**26**	**30**	**56**	**0**	**121**	**126**	**240**	**790**
Najmniejsze znalezione rozszerzenie	3	2	1	0	4	1	0	0	5	2	1	0	6	1	1	0	7	0	0	0	8	0	0	0	16	0	0	0	0
Czas szukania rozszerzenia	0	1	1	2	1	3	3	5	1	3	3	10	1	6	6	28	1	13	12	48	1	13	15	76	3	90	97	679	1594
Ilość znalezionych cykli Hamiltona	1	1	1	2	1	1	1	6	1	1	1	24	1	1	1	109	1	8	12	272	1	14	29	430	1	178	217	1921	959
Czas szukania cykli Hamiltona	0	0	0	0	2	0	1	2	4	2	1	5	7	2	3	16	11	6	7	37	17	9	12	72	140	152	181	1207	4627
Łączny czas wykonania algorytmu	1	1	1	3	3	4	4	7	5	6	5	16	8	9	10	44	13	19	20	85	18	23	27	149	144	242	279	1887	6221

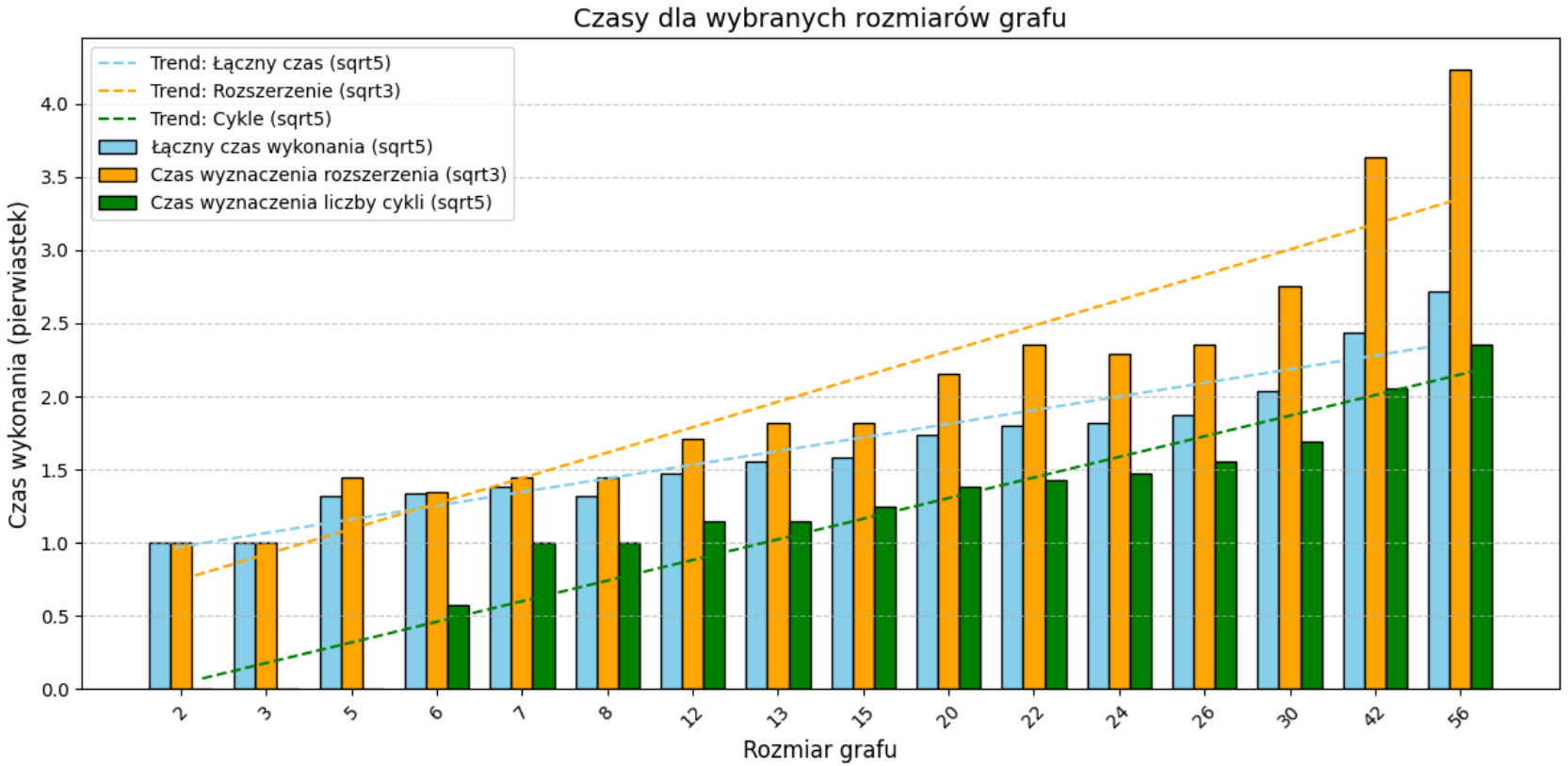
Rysunek 8: Dane wydajności dla retryFactor=17 (czas w milisekundach)

Testy wydajności - Algorytm 1



Rysunek 9: Zestawienie przekształconych o pierwiastek czasów wykonania dla retryFactor=13

Testy wydajności - Algorytm 1



Rysunek 10: Zestawienie przekształconych o pierwiastek czasów wykonania dla retryFactor=17

Wnioski

- Czas szukania rozszerzenia jest dla większych grafów znacząco krótkszy od czasu poświęconego na szukanie liczby cykli Hamiltona. Z tego powodu warto rozważyć zmniejszenie ilości wywołań szukania ścieżek dobrze rezeprezentujacych cykle Hamiltona.
- Dla przedstawionych danych widać, że dla retryFactor, algorytm znalazł większe najmniejsze rozszerzenie niż dla pozostałych wartości retryFactor. Dla pozostałych sprawdzonych wartości nie znaleziono podobnych odchyleń. Czasy wykonania dla retryFactor nie są znacząco większe niż dla retryFactor 1, dlatego też lepiej jest nie korzystać z retryFactor 1.

Wnioski

- Jeżeli użytkownik chce znaleść więcej cykli Hamiltona, to zwiększenie retryFactor skutenicze daje taką możliwość w przypadku większych grafów. Dotyczy to głównie grafów, dla których udało się znaleść cykl Hamiltona bez rozszerzania tych grafów.

Algorytm odległości grafów - dokładny algorytm

Dokładny algorytm odległości

Dokładny algorytm będzie polegał na policzeniu odległości edycyjnej między grafami dla wszystkich permutacji jednego z grafów. Bierzemy minimum z obliczonych odległości edycyjnych (jeśli która kolwiek z nich to 0, to kończymy obliczenia bez sprawdzania kolejnych permutacji).

Złożoność algorytmu dokładnego

Złożoność

Algorytm dokładny wykonuje permutacje wierzchołków grafu czyli ma złożoność wykładniczą. Zatem przygotowaliśmy także algorytm aproksymacyjny.

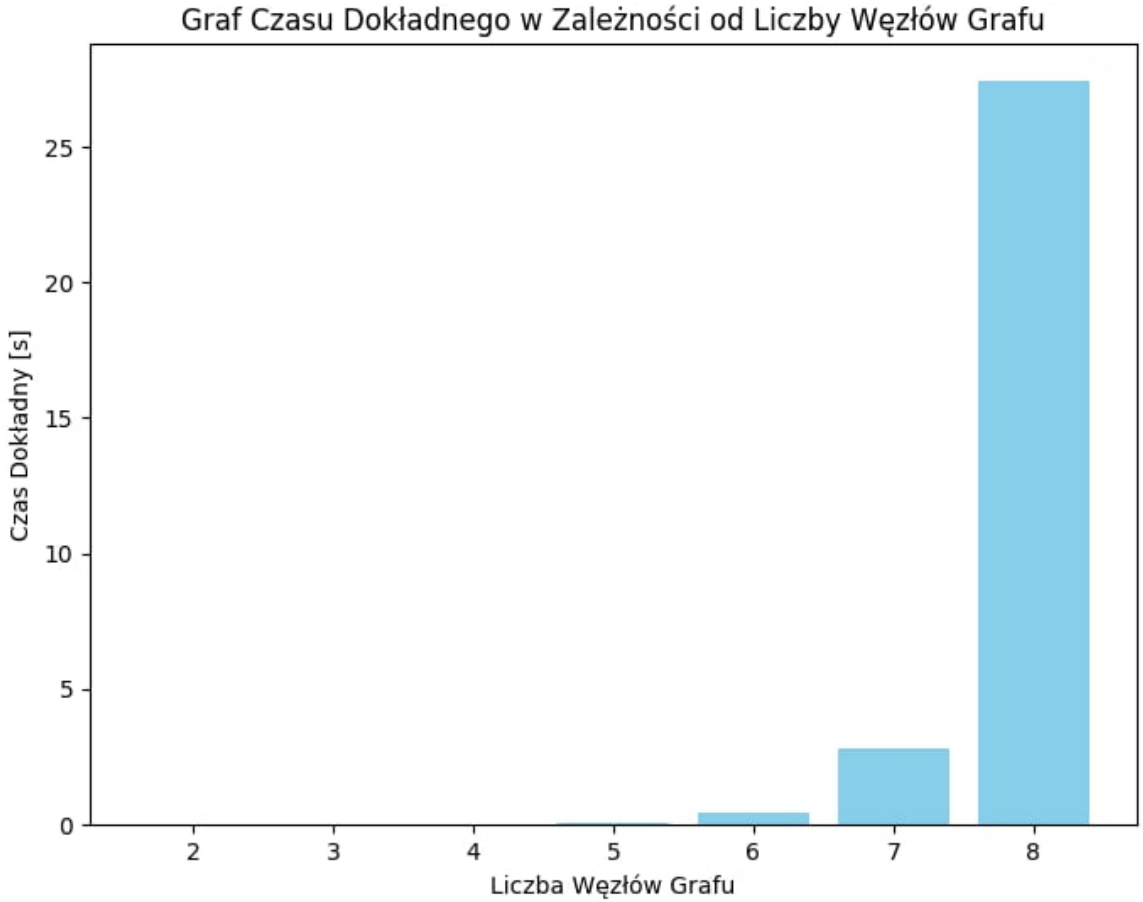
Dokładny algorytm odległości

W algorytmie aproksymacyjnym wykonujemy sortowanie wierchołków grafów najpierw stopniem wierzchołka krawędzi wychodzących. Sortowanie jest stabilne tzn. nie tracimy informacji i kolejności wierzchołków po pierwszym sortowaniu. Następnie algorytm wykonuje sprawdzenie odległości edycyjnej na posortowanych wersjach grafów.

Dokładny algorytm odległości

Sortowanie wierzchołków grafu jest $O(n \log^2 n)$. Obliczenie odległości edycyjnej jest w czasie liniowym O(n). Czyli ostatecznie algorytm jest $O(n \log^2 n)$.

Test wydajności obliczania odległości grafów



Rysunek 11: Czasy obliczenia odległości dokładnej dla różnych rozmiarów grafów

Test wydajności - Porównanie obliczania odległości

Dokładny wynik	Aproksymacja	Rozmiar grafu	Błąd absolutny
0	0	4	0
2	3	3	1
0	0	2	0
2	2	3	0
2	6	4	4
4	10	5	6
6	18	6	12
6	16	7	10
12	36	8	24

Tabela 1: Test wydajności - Wyniki ekspertymentów obliczania odległości grafów

Test wydajności - wnioski dla algorytmów liczenia odległości grafów

Wnioski

Algorytm aproksymacyjny bardzo szybko oblicza swój wynik dla każdego z podanych rozmiarów grafu. Niestety jednak szybko zaczyna on tracić na dokładności dla coraz to większych grafów. Algorytm dokładny za to bardzo szybko osiąga czasy obliczeń niekaceptowalne w testach, dla grafu o rozmiarze 8 węzłów było to ponad 25s, a większe grafy wychodziły po za rozważane ramy czasowe przeprowadzanych testów (kilka minut)