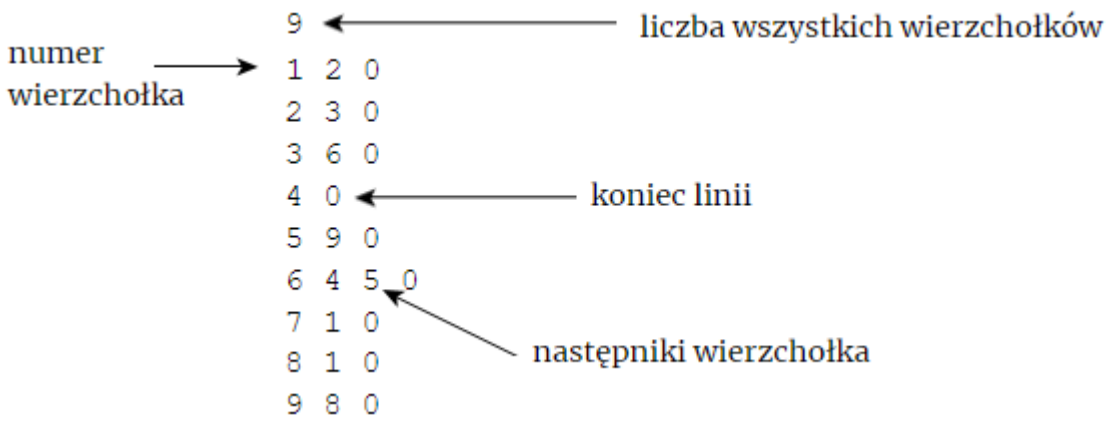
Sprawozdanie

Zadanie 2: Transformacja grafu
liniowego w jego graf oryginalny

I. Opis zastosowanego formatu zapisu grafu

Plik, zarówno wejściowy, jak i wyjściowy, jest zapisany w pliku tekstowym o formacie “.txt”. Liczba w pierwszej linii oznacza liczbę wierzchołków. Wszystkie następne linie są już listą następników, przy czym pierwsza liczba w linii zawsze oznacza numer wierzchołka, a wszystkie kolejne to jej następniki, oprócz liczby 0, która oznacza dla programu koniec wczytywania danych z obecnej linii i przejście do następnej. Numery wierzchołków w grafie zaczynają się od liczby 1, zatem nie dojdzie do żadnego konfliktu danych w programie.

Zawartość przykładowego pliku zapisanego w wyżej omówionym formacie:



II. Opis algorytmu

a) wczytanie dowolnego grafu skierowanego z pliku tekstowego

Plik tekstowy otwierany jest po uprzednim podaniu jego nazwy przez użytkownika. Informacje o grafie wczytywane są z pliku po jednej linii. Na początku do wcześniej utworzonej zmiennej przypisywana jest liczba wierzchołków. Lista następników przedstawiona w pliku tekstowym jest następnie zapisywana do wcześniej zadeklarowanego wektora wektorów, który reprezentuje graf G. Najpierw zapisywane są numer wierzchołka oraz jego następniki w wektorze pomocniczym dopóki program nie natrafi na liczbę 0 w pliku tekstowym. Wtedy cały wektor pomocniczy jest zapisywany w wektorze wektorów, przez co lista następników w algorytmie zaczyna reprezentować graf G. Proces ten powtarzany jest dla każdego wierzchołka w pliku tekstowym. W ten sposób powstaje dwuwymiarowa tablica dynamiczna. Plik tekstowy jest zamykany.

Wizualizacja listy następników dla wczytanego grafu:

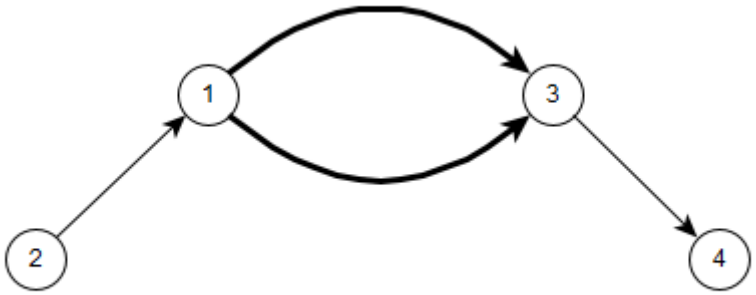
j↓ i→	0	1	2	3	4	5	6	7	8
0	1	2	3	4	5	6	7	8	9
1	2	3	6		9	4	1	1	8
2						5			

Litery *i* oraz *j* oznaczają przykładowe zmienne, których iterowanie pozwoli na przejście przez każdy element w wektorze. Dla dowolnego *i* oraz dla *j* = 0 wektor będzie przechowywał wartość oznaczającą numer wierzchołka, z kolei dla *j* > 0 wektor przechowuje już następniki tego wierzchołka, o ile one istnieją.

W tej wizualnej reprezentacji do odczytywania numeru wierzchołka oraz jego następników wykorzystywane są kolumny wyżej przedstawionej tabeli.

b) sprawdzenie, czy wczytany graf jest grafem sprzężonym

By sprawdzić, czy wczytany graf jest grafem sprzężonym najpierw należy sprawdzić, czy jest 1-grafem lub, alternatywnie, czy nie jest multigrafem, tj. czy nie posiada krawędzi wielokrotnych. We wczytywanym grafie następniki danego wierzchołka zawsze przedstawione są w kolejności rosnącej. W przypadku łuków wielokrotnych między dwoma wybranymi wierzchołkami numer wierzchołka, do którego wchodzą łuki będzie wymieniony dwa razy pod rząd w następnikach wierzchołka, z którego te łuki wychodzą. W ten sposób wystarczy, żeby algorytm porównał wartości dwóch sąsiadujących ze sobą następników dla aktualnie sprawdzanego wierzchołka - w przypadku, gdy będą się pokrywać, algorytm kończy działanie poprzez wypisanie na ekranie odpowiedniego komunikatu. Ten sposób będzie działał również w przypadku, gdy w grafie występuje wierzchołek posiadający więcej niż jedną pętlę własną, gdyż w zbiorze jego następników jego własny numer pojawi się więcej niż raz.



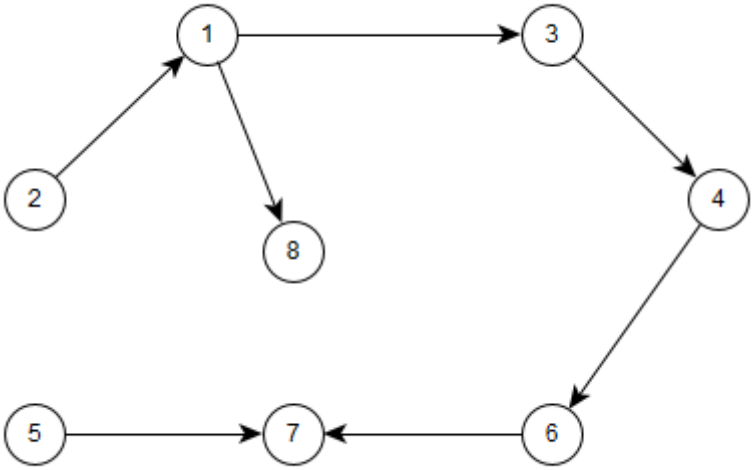
Pogrubioną linią zaznaczono przykład łuku wielokrotnego.

Kiedy jednak we wszystkich zbiorach wierzchołków nie zostanie wykryty ani jeden łuk wielokrotny, algorytm zaczyna sprawdzać, czy wczytany graf jest grafem sprzężonym. Graf jest sprzężony wtedy i tylko wtedy, gdy dla dowolnej pary wierzchołków w grafie spełniony jest poniższy warunek:

$$N^+(x) \cap N^+(y) \neq \emptyset \Rightarrow N^+(x) = N^+(y),$$

gdzie $N^+(x)$ to zbiór następników x,
a $N^+(y)$ to zbiór następników y.

Graf, będący przykładowym grafem sprzężonym, został zwizualizowany poniżej.

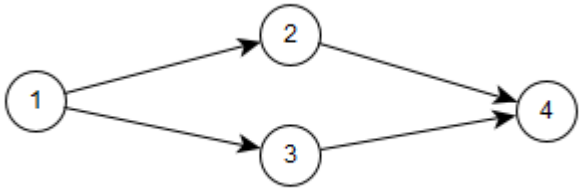


Zbiory następników dla wierzchołków 6 i 5 są identyczne, gdyż jest to tylko jeden wierzchołek o numerze 7. Każda inna dowolna para wierzchołków w tym grafie ma rozłączne zbiory następników, np. para 1 i 4: następnikami dla wierzchołka o numerze 1 są wierzchołki o numerach 3 i 8, natomiast dla wierzchołka o numerze 4 następnikiem jest tylko jeden wierzchołek, którego numer wynosi 6.

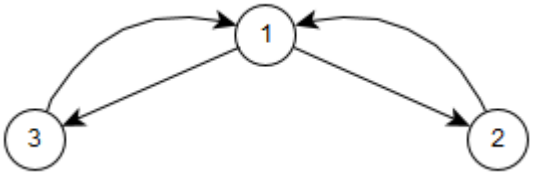
Dlatego też aby określić, czy wczytany graf jest grafem sprzężonym, algorytm musi sprawdzić, czy dla wszystkich wierzchołków zbiory ich następników albo są rozdzielne, albo całkowicie się pokrywają. W tym celu zbiór następników jednego wierzchołka porównywany jest ze zbiorem następników każdego następnego wierzchołka (o ile następny wierzchołek w ogóle posiada następniki, co również jest sprawdzane), porównywanie zaczynając od wierzchołka numer 1. Gdy algorytm znajdzie chociaż jeden identyczny wierzchołek w dwóch aktualnie porównywanych zbiorach następników, wywoływana jest funkcja *equal*, która sprawdza, czy te dwa zbiory następników są sobie równe. Jeżeli funkcja znajdzie chociaż jeden element niepasujący, zwraca wartość *false*, w innym przypadku zwraca wartość *true*. W obu sytuacjach na ekranie wypisywany jest odpowiedni komunikat.

c) sprawdzenie, czy graf jest grafem liniowym

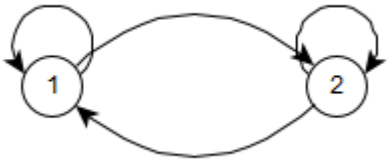
Aby algorytm uznał wczytany graf za graf liniowy, żaden z poniższych podgrafów nie mogą pojawić się we wczytanym grafie:



Struktura I



Struktura II



Struktura III

Sprawdzanie, czy któraś z dwóch pierwszych struktur znajduje się w grafie można przeprowadzić za jednym razem: różnią się one tylko tym, że w przypadku pierwszej struktury, następnikiem następników wierzchołka 1 jest wierzchołek 4, a w przypadku struktury drugiej jest to znowu wierzchołek 1. Program jednak może sprawdzić, czy w zbiorach następników wierzchołków będących następnikami wierzchołka 1 istnieje chociaż jeden powtarzający się element - w tym przypadku nie ma znaczenia, czy powtarzającym się elementem byłby wierzchołek 4 czy 1, gdyż liczy się fakt, że w ogóle występuje taki powtarzający się element, co oznacza, że graf zawiera strukturę I lub II, lub, jak w przypadku niektórych grafów, nawet obie te struktury na raz, a co za tym idzie - nie jest grafem liniowym.

Występowanie struktury trzeciej algorytm wykazuje poprzez sprawdzenie, czy wierzchołek posiada pętle własne, które w jego następnikach przedstawione są jako wierzchołki o tym samym numerze, co obecnie sprawdzany wierzchołek. Następnie algorytm sprawdza, czy którykolwiek z następników obecnego wierzchołka posiada pętlę własną, a gdy tak, to czy taki następnik zawiera numer obecnego wierzchołka w swoich następnikach. W sytuacji, kiedy te wszystkie warunki są spełnione, graf nie jest liniowy.

W przypadku wszystkich trzech struktur najpierw należy sprawdzić, czy obecnie sprawdzany wierzchołek ma co najmniej dwa następniki, w innej sytuacji wykonywanie tego sprawdzania nie miałoby sensu, bo żadna struktura nie mogłaby zaczynać się od takiego wierzchołka.

Gdy którakolwiek z tych struktur znajdzie się we wczytanym grafie, program wypisze odpowiedni komunikat. Kiedy jednak we wczytanym grafie nie będzie można wyodrębnić żadnego z wyżej przedstawionych podgrafów, wtedy algorytm uznaje graf za liniowy. W obu przypadkach jednak algorytm przechodzi do następnego kroku, czyli transformacji grafu sprzężonego w jego graf oryginalny.

d) przekształcenie grafu sprzężonego w jego graf oryginalny (H)

Grafem oryginalnym H grafu sprzężonego G jest taki graf, w którym łuk reprezentuje wierzchołek grafu G, jednak przy zachowaniu zasady, że jeżeli w grafie sprzężonym istniało przejście pomiędzy danymi wierzchołkami, to w grafie oryginalnym musi istnieć przejście pomiędzy odpowiadającymi tym wierzchołkom łukami.

Aby wykonać transformację grafu sprzężonego w jego graf oryginalny algorytm najpierw tworzy pomocniczy wektor wektorów, przy czym liczba kolumn zależna jest od liczby wierzchołków w grafie sprzężonym, natomiast liczba wierszy jest równa dwa. Wektor ten jest reprezentacją “pierwszej wersji” grafu H. Każda para liczb, a więc każdy łuk, reprezentuje jeden wierzchołek w grafie sprzężonym G. W tym celu wykorzystana zostaje następująca własność: dla wierzchołka m w grafie G numery wierzchołków tworzących odpowiadający mu łuk w grafie H tworzą parę $(2m-1, 2m)$. W ten sposób wektor pomocniczy wypełniany jest parami liczb reprezentującymi łuki rozłączne, czyli (1,2), (3,4), (5,6) itd. Po tym następuje odtworzenie

połączeń z grafu G 1-1 w taki sposób, że jeżeli para (**x**,**y**) reprezentuje wierzchołek *a*, a para (**q**,**z**) wierzchołek *b*, i w grafie G istnieje przejście pomiędzy wierzchołkiem *a* a wierzchołkiem *b*, to oznacza, że w grafie H trzeba skompresować wierzchołki **y** i **q** do jednego. Algorytm robi to poprzez najpierw wyszukanie wszystkich wierzchołków o wartości **q**, a następnie zmienienie ich na wartość **y**. W ten sposób powstają łuki (**x**,**y**) oraz (**y**,**z**). Przeindeksowanie takie wykonywane jest analogicznie dla reszty przejść między wierzchołkami w grafie G, tak by odtworzyć je w postaci przejść między łukami w grafie H.

Niestety, takie przeindeksowanie prowadzi do stworzenia nowego zbioru łuków, tym razem już nie rozłącznych, a numeracja wierzchołków w tych łukach nie jest z przedziału [1, *V*], gdzie *V* oznacza liczbę wszystkich wierzchołków w grafie H. Dlatego potrzebne jest kolejne przeindeksowanie tych wierzchołków tak, aby tym razem była wykorzystana każda liczba z wyżej wymienionego przedziału. W tym celu algorytm wykorzystuje hashmapę, która jest swoistym słownikiem translacji indeksów. Kluczem w mapie jest indeks wejściowy (przed przeindeksowaniem), a wartością - indeks wyjściowy. Najpierw deklarowana jest zmienna *index* dla kolejnych indeksów po przeindeksowaniu. Potem następuje iterowanie po wszystkich wartościach w wektorze wektorów reprezentującym graf H. Dla każdej wartości sprawdzane jest, czy taki klucz istnieje już w mapie translacji. W przypadku, gdy taki klucz istnieje, to zmieniana jest wartość w wektorze na to, co jest w mapie pod aktualną wartością w tablicy. Kiedy jednak taki klucz jeszcze nie istnieje, to aktualna wartość dodawana jest do mapy jako klucz, a następnie *index* zwiększa się o jeden. W ten sposób powstaje zbiór wierzchołków, który przyjmuje wartości z przedziału [1, *V*] oraz zachowuje łuki między wierzchołkami obecne przed wykorzystaniem hashmapy.

Ostatnim etapem jest “przepisanie” danych z wektora pomocniczego do wektora, który będzie ostateczną wersją reprezentującą graf H, toteż musi posiadać w sobie listę następników w takim samym formacie, co graf G. W tym celu najpierw znajdowany jest numer ostatniego wierzchołka z przedziału [1, *V*], który oznacza również liczbę wszystkich wierzchołków w grafie H. Następnie tworzony jest kolejny wektor wektorów, do którego będą przepisywane dane z wektora pomocniczego. Tworzona jest też zmienna *vertex*, której wartość oznacza numer wierzchołka, dla którego obecnie przepisywane będą następniki. Następnie przeszukiwany jest zbiór łuków w wektorze pomocniczym tak, by pierwsza liczba z pary oznaczająca początek łuku, pokrywała się z numerem wierzchołka odpowiadającym wartości

zmiennej *vertex*. Wtedy druga liczba z takiej pary zapisywana jest jako następnik wierzchołka o wartości *vertex*. Zapisywanie tych danych do wektora następuje w sposób analogiczny, co zapisywanie danych do wektora z pliku wejściowego, które zostało omówione w podpunkcie a). W przypadku, kiedy dany wierzchołek nie ma żadnych następników, do wektora zapisywany jest tylko numer takiego wierzchołka. Gdy cały zbiór zostanie przeszukany dla danego wierzchołka, następuje zwiększenie wartości *vertex* o jeden. Proces ten powtarzany jest dla wszystkich wierzchołków. W ten sposób powstaje wektor wektorów, który jest - tak samo jak wektor reprezentujący graf G - w formie listy następników.

e) zapisanie grafu wynikowego H do pliku tekstowego innego niż wejściowy, lecz w tym samym formacie

Ostatnim krokiem jest zapisanie grafu wynikowego H do pliku tekstowego, który jest plikiem innym niż wejściowy. Zostaje on automatycznie nazwany poprzez dodanie do nazwy pliku wejściowego sufiksu "*Oryginalny.txt*". Następnie program najpierw wpisuje do pliku tekstowego liczbę wszystkich wierzchołków, a w kolejnej linii zapisuje numer wierzchołka oraz jego następniki. Po wczytaniu wszystkich następników dla danego wierzchołka dopisywana jest liczba 0 oznaczająca koniec danej linii. Kiedy wszystkie wierzchołki oraz ich następniki zostaną zapisane w pliku, zostaje on zamknięty.

III. Oszacowanie złożoności algorytmu

Biorąc pod uwagę fakt, że w swoim programie korzystałam głównie z pętli *for* oraz *while*, które wykorzystywałam do iterowania po wierzchołkach, oraz fakt, że pętle te najczęściej były zagnieżdżone jedna w drugiej, a także fakt, że korzystałam z funkcji *equal*, złożoność swojego algorytmu oszacowałam na $O(n^5)$, gdzie n to liczba wierzchołków we wczytanym grafie. Jest to złożoność wielomianowa.

IV. Opis przeprowadzonych testów

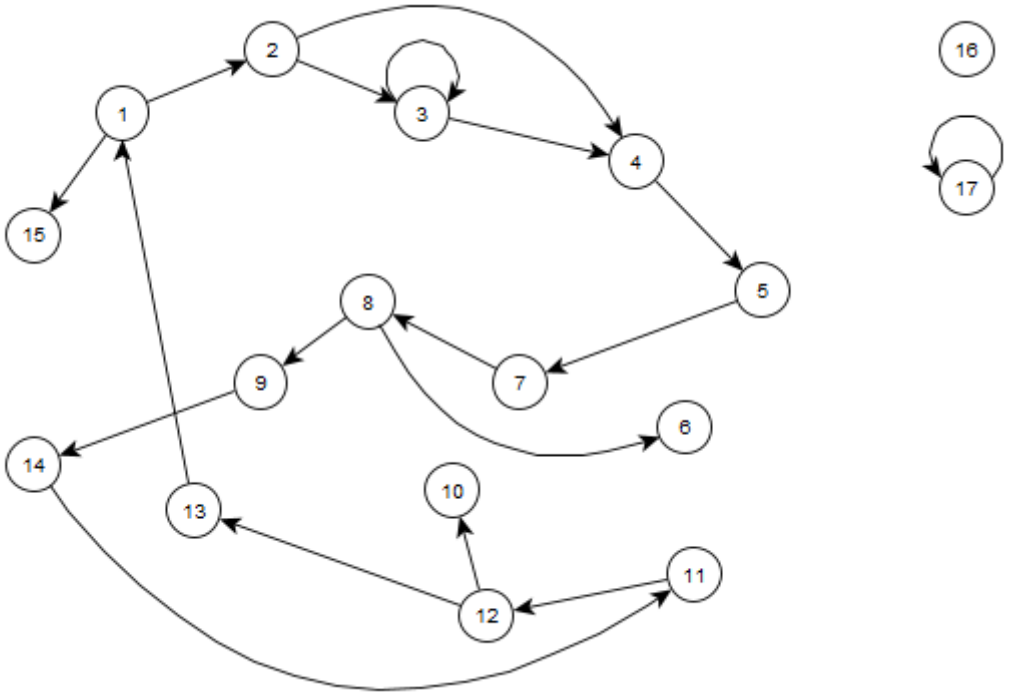
Celem przeprowadzonych testów było najpierw sprawdzenie, czy zaimplementowane przeze mnie funkcje spełniają swoje role w poprawny sposób. By to potwierdzić, każdy wczytywany graf wcześniej rozrysowywałam ręcznie na kartce i ustalałam, czy jest 1-grafem, a gdy tak, to czy tylko grafem sprzężonym, czy może też grafem liniowym. Następnie porównywałam moje ustalenia z komunikatami wyświetlanymi przez mój program. Do tego testowania skonstruowałam 6 różnych grafów o różnej liczbie wierzchołków: od 7 do 15. Jeden z nich nie był 1-grafem, kolejny był 1-grafem, ale nie był grafem sprzężonym. Dwa następne były grafami sprzężonymi, jednak zawierały jedną z trzech struktur, które nie mogą znajdować się w grafie liniowym. Ostatnie dwa grafy były natomiast grafami liniowymi.

Po wykonaniu tych wstępnych testów przeszłam do sprawdzania, czy mój algorytm wykonuje prawidłowe transformacje. W tym celu testowałam go na 13 różnych grafach, które różniły się liczbą wierzchołków (od 10 do 20) oraz liczbą łuków. Znaczna część z nich była grafami liniowymi, jednak testy przeprowadzałam też na grafach, które nie były liniowe i zawierały którąś z trzech struktur przedstawionych wcześniej. Każdy z tych trzynastu grafów sprzężonych rozrysowałam na początku na kartce, a następnie ręcznie przeprowadzałam ich transformacje w grafy oryginalne. Następnie nanosiłam na wierzchołki przetransformowanych grafów indeksy zgodnie z listą następników, którą wygenerował mój program. W przypadku, gdy wszystkie następniki się zgadzały, transformację danego grafu uznawałam za poprawną.

Poniżej zaprezentowana została wizualizacja transformacji 4 spośród 13 testowanych przeze mnie grafów.

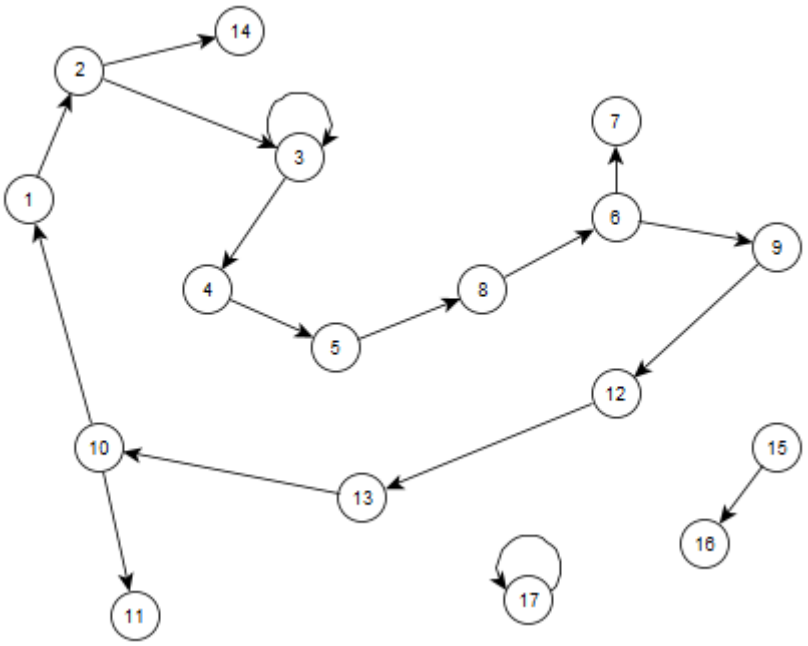
Graf 1

Graf liniowy posiadający 17 wierzchołków oraz 18 łuków, w tym 2 pętle własne.



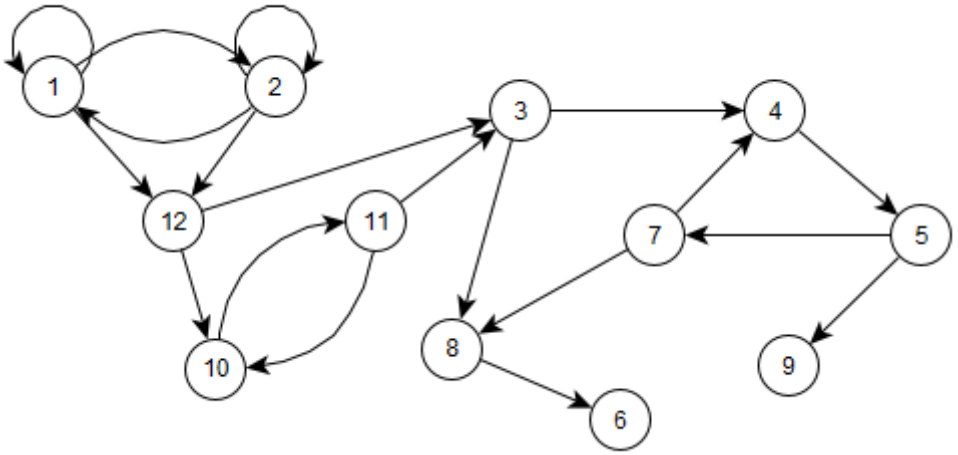
transformacja

graf sprzężony G



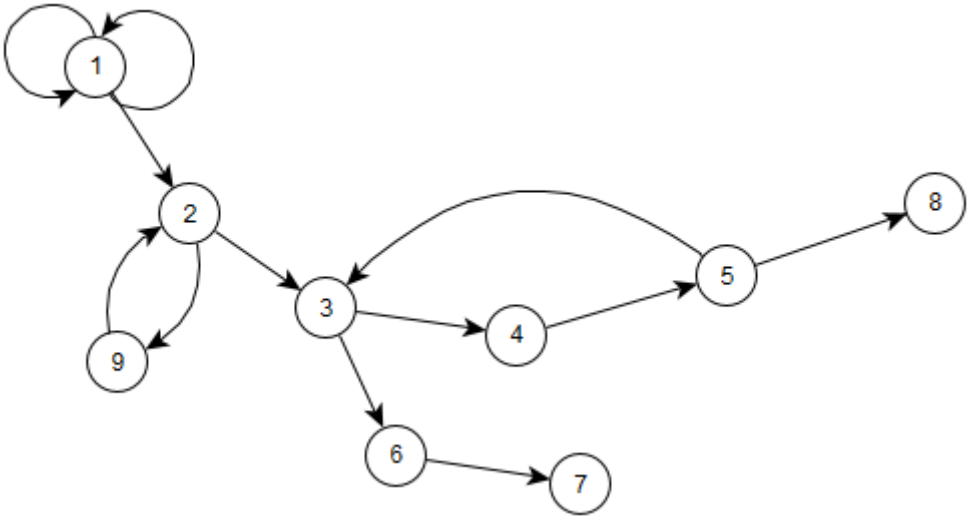
graf oryginalny H

Graf 2
Graf nieliniowy posiadający 12 wierzchołków oraz 19 łuków, w tym 2 pętle własne.



graf sprzężony G

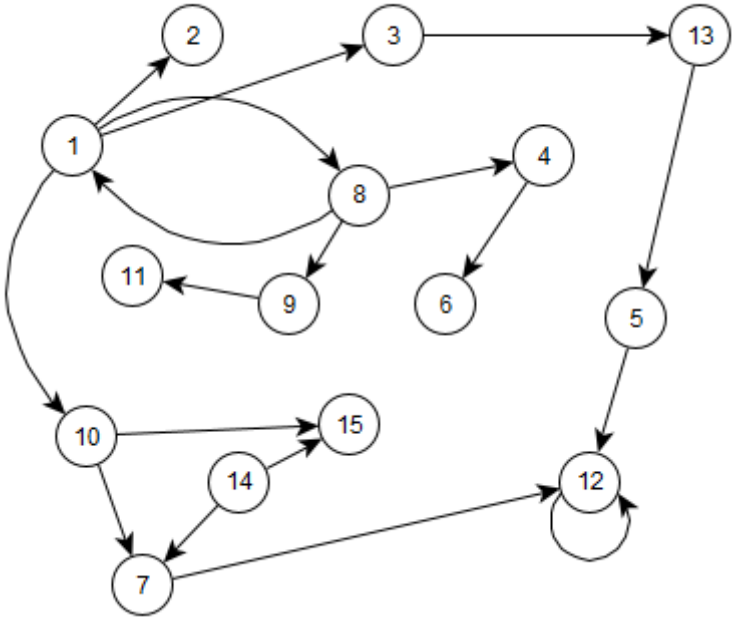
transformacja



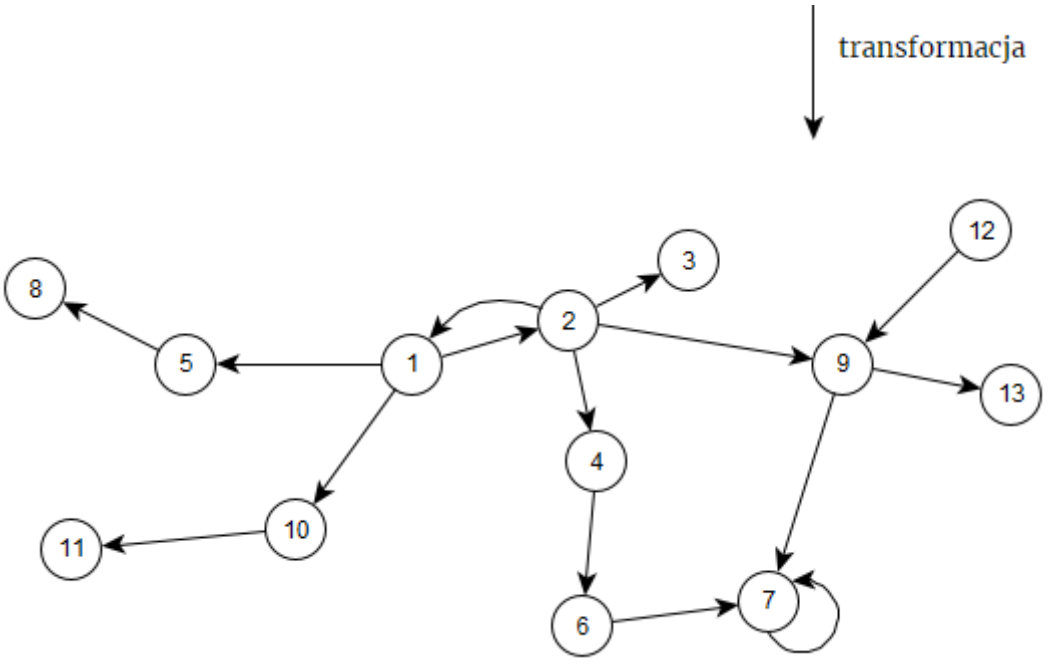
graf oryginalny H

Graf 3

Graf liniowy posiadający 15 wierzchołków oraz 18 łuków, w tym 1 pętlę własną.



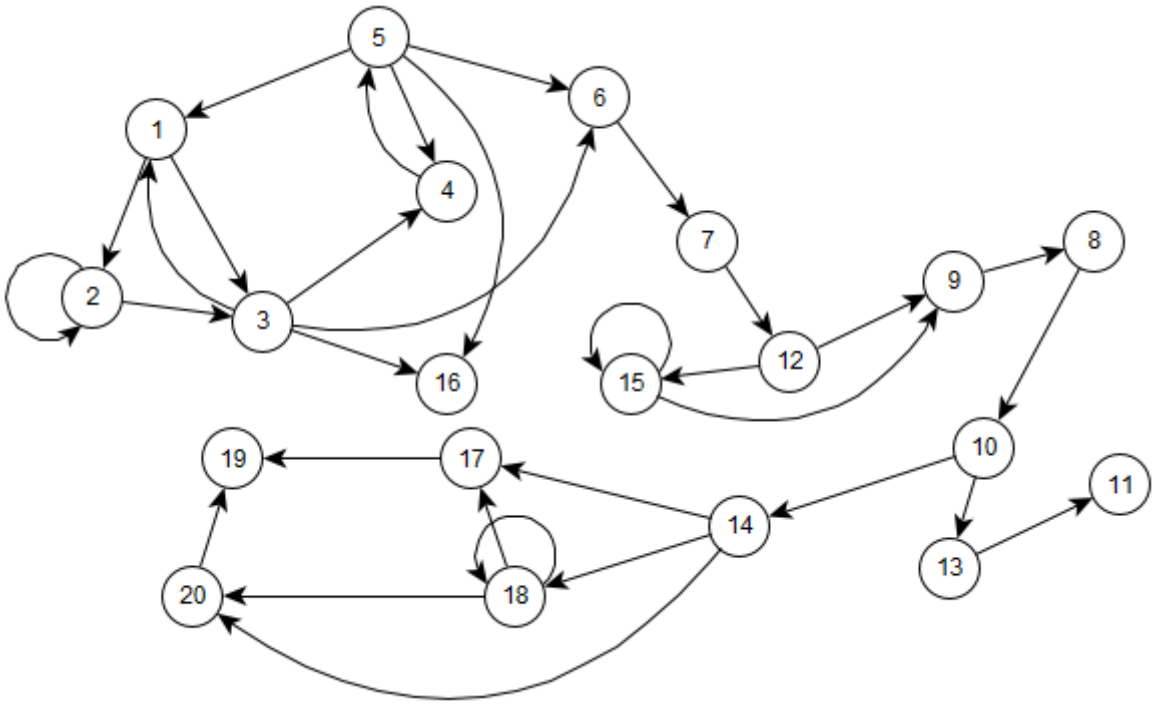
graf sprzężony G



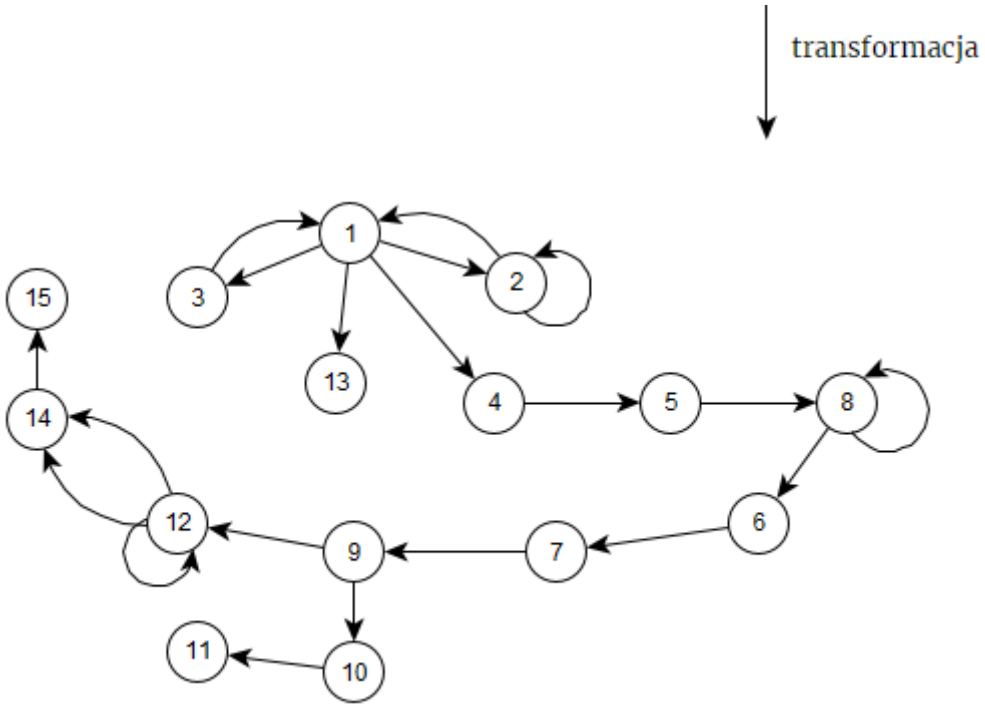
graf oryginalny H

Graf 4

Graf nieliniowy posiadający 20 wierzchołków oraz 32 łuki, w tym 3 pętle własne.



graf sprzężony G



graf oryginalny H

V. Wnioski

Już podczas testowania różnych grafów zauważyłam kilka własności, które posiadają grafy sprzężone oraz ich grafy oryginalne. Wraz z wykonywaniem kolejnych testów moje spostrzeżenia wydawały się potwierdzać. Jedną z takich obserwacji jest fakt, że po transformacji w grafie oryginalnym było dokładnie tyle samo pętli własnych, co w grafie sprzężonym. Innym spostrzeżeniem jest to, że gdy w grafie sprzężonym występował wierzchołek izolowany niebędący następnikiem żadnego innego wierzchołka, to z kolei taki wierzchołek izolowany nie pojawiał się już w grafie oryginalnym. Jest to raczej logiczne, biorąc pod uwagę podstawowe założenia i zasady transformacji grafu sprzężonego w jego graf oryginalny, czyli że łuk w grafie oryginalnym odpowiada konkretnemu wierzchołkowi w grafie sprzężonym. Jednak dopiero testowanie pozwoliło mi świadomie zauważyć tę zależność, na którą potem często zwracałam uwagę. Pozwoliła mi ona także wykryć nieznaczne błędy w działaniu mojego algorytmu. Kolejną zależnością dotyczącą grafów sprzężonych i ich grafów oryginalnych jest to, że rzeczywiście w przypadku stworzenia nieliniowego grafu sprzężonego jego graf oryginalny nie był 1-grafem, lecz multigrafem, a więc taki graf oryginalny nie zaliczał się już do grafów sprzężonych i nie można było przeprowadzać jego dalszych transformacji. Z początku wydawało mi się też, że wszystkie grafy oryginalne będą miały taką samą lub - co przeważnie miało miejsce - mniejszą liczbę wierzchołków niż grafy sprzężone, z których powstały. Takiej zależności jednak nie zauważyłam w przypadku niektórych grafów, które testowałam. Posiadały one mniej łuków niż grafy, które zwizualizowałam wyżej, toteż zaczęłam się zastanawiać, czy liczba wierzchołków w grafie oryginalnym może zależeć od liczby łuków w grafie sprzężonym. Zależność ta wydawała się być odwrotnie proporcjonalna, jednak nie przeprowadziłam wystarczająco dużo testów na grafach o niedużej liczbie łuków, by wysnuć inne wnioski z tej rzekomej zależności lub by uznać ją za prawdziwą lub nie, więc pozostawiam ją tylko jako moje przypuszczenie.

Zarówno zależność związana z wierzchołkami izolowanymi w grafie sprzężonym, jak i inne zależności, które nie były tylko i wyłącznie moimi przypuszczeniami, pozwoliły mi na dopracowanie niektórych wad i błędów, które ujawniły się w moim algorytmie podczas testowania go. Spostrzeżenie tych zależności pozwoliło mi również na szybkie wykrycie niewielkich błędów, które przez przypadek zdarzało mi się popełniać podczas ręcznej transformacji grafów na kartce. Dzięki temu udało mi się uniknąć

niepotrzebnego poprawiania niektórych rzeczy w moim kodzie, które działały poprawnie.

Nie zaobserwowałam żadnej różnicy między czasami transformacji grafów nieliniowych a czasami transformacji grafów liniowych, dlatego też nie jestem w stanie stwierdzić, czy w ogóle istnieje zależność pomiędzy faktem, że graf jest liniowy (bądź nie) a czasem jego transformacji. Instancje, na których testowałam swój algorytm nie były duże i ograniczały się tylko do 20 wierzchołków, toteż by dalej przebadać ewentualne istnienie takiej zależności, potrzebowałabym testować swój algorytm na większych instancjach.

Źródła

C. Berge, *Graphs and Hypergraphs*, North-Holland Publishing Company, London, 1973.
J. Błażewicz, A. Hertz, D. Kobler, D. de Werra, *On some properties of DNA graphs, Discrete Applied Mathematics* 98, 1-19, 1999.
W. Lipski, *Kombinatoryka dla programistów*, Wydawnictwa Naukowo-Techniczne, 2007.
https://en.wikipedia.org/wiki/Line_graph