Politechnika Wrocławska
Wydział Informatyki i Telekomunikacji

Kierunek: **INF-PPT**
Specjalność: **-**

Praca Dyplomowa Inżynierska

Aplikacja do wizualizacji wybranych algorytmów grafowych

Błażej Wróbel

Opiekun pracy
dr Maciej Gębala

wizualizacja, algorytmy grafowe, algorytmy i struktury danych, teoria grafów

Wrocław 2022

Streszczenie

W pracy dyplomowej przedstawiono analizę porównawczą istniejących rozwiązań do wizualizacji wykonania wybranych algorytmów grafowych, analizę informatyzowanego zagadnienia, określono wymagania funkcjonalne oraz pozafunkcjonalne dla aplikacji, dostarczono projekt aplikacji wykonany za pomocą diagramów UML, opisano użyte algorytmy i technologie, dostarczono instrukcję obsługi aplikacji dla użytkownika, opisano proces instalacji i wdrożenia omawianej aplikacji, określono stan prac implementacyjnych i projektowych, wskazano możliwe drogi rozwoju dostarczonego rozwiązania oraz wymieniono nietrywialne algorytmy i technologie użyte w dostarczonej aplikacji.

Abstract

The bachelor thesis presents a comparative analysis of existing solutions for visualization of selected graph algorithms, an analysis of the computerized issue, the functional and non-functional requirements for the application were determined, the application design made with UML diagrams was provided, the algorithms and technologies used were described, user manual was provided, the process of installation and deployment of the discussed application was described, the state of implementation and design work was determined, possible paths of development of the delivered solution were indicated and non-trivial algorithms and technologies used in the delivered application were listed.

Spis treści

Spis rysunków

Wstęp

Cel pracy

Celem pracy było napisanie programu przedstawiającego graficznie działanie wybranych algorytmów grafowych (algorytm Prima, algorytm Kruskala, algorytm znajdowania silnie spójnych składowych w grafie skierowanym, algorytm Forda-Fulkersona, algorytm Edmondsa-Karpa) oraz stworzenie projektu aplikacji i kompletnej dokumentacji. Ponadto, celem dodatkowym samego programu, jest ułatwienie zrozumienia działania przedstawionych algorytmów grafowych.

Analiza i porównanie istniejących rozwiązań

Na rynku istnieje wiele aplikacji, darmowych albo płatnych, służących do wizualizacji grafów. W większości przypadków są to narzędzia analityczne tj. służą do przedstawienia danych i zależności między nimi jako graf, gdzie graficzna reprezentacja ułatwia zrozumienie takich danych i wspomaga np. procesy podejmowania decyzji w przedsiębiorstwach lub organizacjach. Jeśli chodzi o aplikacje służące do wizualizacji grafów i algorytmów na nich wykonywanych, to większość z nich jest projektami uniwersyteckimi lub otwartymi o charakterze edukacyjnym i zazwyczaj przyjmują postać aplikacji webowej, którą uruchamia się w przeglądarce. Pozwalają one na graficzne przedstawienie różnych algorytmów grafowych, niekoniecznie tych samych, które są wizualizowane w aplikacji będącej częścią pracy dyplomowej. Część z nich umożliwia wizualizacje nie tylko algorytmów grafowych, ale również algorytmów sortowania lub operacji na drzewach binarnych. Analizowane i porównywane rozwiązania można znaleźć na następujących stronach internetowych: [1] oraz [2].

Analizowane aplikacje umożliwiają różne sposoby tworzenia i manipulowania rysunkiem grafu: wczytywanie listy sąsiedztwa (wraz z wagami krawędzi) i wczytywanie współrzędnych 2D położenia wierzchołków, „ręczne" rysowanie grafu (wagi są dodawane automatycznie) lub generowanie grafu po stronie aplikacji (użytkownik nie ma wpływu na to, jaki graf zostanie wygenerowany). Wszystkie opisywane aplikacje umożliwiają uruchomienie wybranego algorytmu na przykładowym grafie, który pojawia się po uruchomieniu programu. Dodatkowo część z tych aplikacji umożliwia sprawdzenie tego, czy na danym grafie można wykonać wybrany algorytm (np. poprzez wyświetlanie odpowiednich komunikatów o błędzie).

W każdej z analizowanych aplikacji, szczegółowo opisywany jest krok wykonywanego algorytmu (np. poprzez wyświetlenie stosownego komunikatu lub zaznaczenie tego kroku w pseudokodzie algorytmu). Obsługa animacji nie różni się między aplikacjami – w każdej można rozpocząć animację, przewinąć animację w przód/tył, zatrzymać albo zakończyć animację oraz zmienić jej szybkość. Sposób graficznego przedstawienia algorytmu nie różni się istotnie między analizowanymi programami – zazwyczaj zaznaczane są (np. kolorem) przetwarzane wierzchołki oraz krawędzie i dodatkowo wyświetlane są inne informacje, które są wykorzystywane przez algorytm (np. koszt wierzchołka i jego przodek – dla algorytmu Prima). Po zakończeniu animacji, zazwyczaj jest wyświetlany efekt końcowy działania algorytmu np. znalezione drzewo MST lub najkrótsze ścieżki (również w tym przypadku rysunek grafu jest odpowiednio modyfikowany).

W każdej z analizowanych aplikacji, na proces tworzenia rysunku grafu nałożono pewne ograniczenia, które mogą utrudnić korzystanie z aplikacji np.: tworzenie rysunku grafu poprzez wczytywanie listy sąsiedztwa – w tym przypadku użytkownik nie ma żadnej kontroli nad tym, jak graf jest rysowany. Dla grafów gęstych rysunek grafu może być nieczytelny. Brak możliwości manualnego ustawienia wagi krawędzi, brak możliwości manipulowania położeniem wierzchołków – te ograniczenia również mogą spowodować,

że rysunek grafu będzie nieczytelny, również brak możliwości usuwania i dodawania wierzchołków oraz krawędzi w oczywisty sposób zmniejsza ”elastyczność” korzystania z aplikacji.

Z powyższej analizy wynika, że główne zadanie tych aplikacji (czyli graficzne przedstawienie wykonywanych algorytmów) jest dla wszystkich programów zrobione podobnie (tj. odpowiednio manipulują rysunkiem grafu). Dodatkowo na podstawie powyższej analizy można wskazać, czym się poszczególne aplikacje od siebie różnią: jakością wykonania (np. ładne GUI, prawidłowa obsługa błędów), sposobem współpracy z użytkownikiem (np. w jaki sposób jest tworzona i zmieniana graficzna reprezentacja grafu), prezentowanymi algorytmami, formą (aplikacja webowa lub działająca jako program na komputerze) oraz niektórymi dodatkowymi funkcjonalnościami (np. czy wygenerowany lub narysowany graf da się gdzieś zapisać). Podsumowując, analizowane aplikacje głównie różnią się realizowanymi założeniami pozafunkcjonalnymi, które mają wpływ na wygodę i ”przyjemność” korzystania z aplikacji.

Opis zawartości pracy

W rozdziale pierwszym pracy przedstawiono analizę informatyzowanego zagadnienia, określono założenia funkcjonalne oraz pozafunkcjonalne, które musi spełniać aplikacja. Przeprowadzono także analizę porównawczą już istniejących, podobnych rozwiązań z omawianą aplikacją oraz omówiono model matematyczny reprezentacji grafu w aplikacji. Ponadto zostały wymienione wszystkie algorytmy, które są w aplikacji wizualizowane.

W rozdziale drugim został przedstawiony projekt aplikacji wykonany za pomocą diagramów UML oraz scenariusze przypadków użycia.

W rozdziale trzecim został przedstawiony opis użytych i wizualizowanych algorytmów.

W rozdziale czwartym został przedstawiony opis użytych technologii, instrukcja obsługi aplikacji dla użytkownika oraz omówiono sposób instalacji i wdrożenia aplikacji.

W podsumowaniu przedstawiono stan zakończonych prac implementacyjnych oraz projektowych, wymieniono możliwe kierunki dalszego rozwoju aplikacji oraz wyszczególniono nietrywialne algorytmy i technologie stosowane w aplikacji.

W dodatku umieszczono krótki opis zawartości płyty CD dołączonej do pracy dyplomowej.

Rozdział 1

Analiza problemu

W tym rozdziale jest zawarta analiza informatyzowanego zagadnienia, wypunktowanie założeń funkcjonalnych oraz pozafunkcjonalnych dla projektowanej aplikacji, analiza porównawcza omawianego programu i innych dostępnych aplikacji tego typu, omówienie modelu matematycznego elementów aplikacji oraz wymienienie wizualizowanych algorytmów.

W pracy informatyzowanym zagadnieniem jest wizualizacja wykonania wybranych algorytmów grafowych. W szczególności, użytkownik będzie miał możliwość stworzenia rysunku grafu oraz manipulacji nim. Ponadto użytkownik będzie mógł wybrać algorytm i uruchomić animację, która przedstawia jego wykonanie (samo przedstawienie wybranego algorytmu odbywa się za pomocą odpowiedniej modyfikacji rysunku grafu tzn. zaznaczenie odpowiednim kolorem krawędzi i wierzchołków lub wyświetlenie odpowiednich informacji (istotnych dla wykonywanego algorytmu)). Aby to osiągnać aplikacja musi spełniać następujące założenia funkcjonalne:

1. Możliwość określenia rodzaju grafu (skierowany albo nieskierowany).
2. Możliwość tworzenia rysunku grafu.
3. Możliwość usuwania krawędzi i wierzchołków.
4. Możliwość ustawienia wagi krawędzi.
5. Możliwość zmiany położenia dowolnego wierzchołka.
6. Możliwość wybrania algorytmu do wizualizacji przez użytkownika.
7. Możliwość uruchomienia animacji przedstawiającej wybrany przez użytkownika algorytm.
8. Wyświetlanie odpowiednich komunikatów o aktualnym kroku (czynności) wykonywanego algorytmu i pokazanie go w wyświetlonym pseudokodzie.
9. Możliwość przywrócenia początkowej wersji grafu po animacji i usunięcia niepotrzebnego pseudokodu.
10. Możliwość ponownego uruchomienia animacji.
11. Możliwość zamknięcia kart z nieużywanymi rysunkami.

Oprócz powyższych założeń (które są niezbędne do osiągnięcia celu), aplikacja będzie spełniać dodatkowe założenia pozafunkcjonalne, które ułatwią użytkownikowi pracę z aplikacją i wpłyną pozytywnie na jakość animacji:

1. Przy braku wyboru rodzaju grafu, domyślnie jest tworzony graf nieskierowany.
2. Możliwość automatycznego ustawiania wag krawędzi.

3. Możliwość zmiany położenia wag krawędzi oraz danych związanych z wierzchołkiem (potrzebne do poprawienia jakości animacji).
4. Automatyczne poprawianie jakości rysunku.
5. Możliwość zmiany rozmiaru wierzchołków.
6. Możliwość całkowitego usunięcia rysunku grafu (bez usuwania pliku).
7. Możliwość wyłączenia animacji.
8. Możliwość wyłączenia animacji i wyświetlenia efektu końcowego działania algorytmu.
9. Możliwość uruchamiania wielu animacji naraz.
10. Możliwość usunięcia (wyczyszczenia) komunikatów generowanych przez aplikację.
11. Możliwość zamknięcia karty z uruchomioną animacją.
12. Możliwość zapisania rysunku grafu do pliku.
13. Możliwość otworzenia i pracy z wcześniej zapisanym rysunkiem grafu.
14. Możliwość usunięcia niepotrzebnego/nieużywanego pliku.
15. Możliwość otwierania/usuwania wielu plików naraz.
16. Możliwość usunięcia otwartego pliku (również z działającą animacją).
17. Możliwość swobodnego przechodzenia między kartami.
18. Oznaczanie kolorem wybranej krawędzi lub wybranego wierzchołka (krawędź lub wierzchołek wybiera użytkownik).
19. Sprawdzanie, czy na danym grafie można uruchomić wybrany algorytm (jeśli się nie da, to należy wypisać adekwatne komunikaty błędu).

W poprzednim rozdziale, zostały przeanalizowane inne aplikacje realizujące podobne funkcjonalności, co omawiana aplikacja. Podstawową różnicą między omawianą aplikacją, a innymi podobnymi programami jest to, że w omawianym programie nie ma możliwości sterowania szybkością animacji oraz nie ma możliwości przewijania animacji w przód/tył. Zamiast tego, w omawianym projekcie jest możliwość wyłączenia animacji (przydatne jeśli np. użytkownik przez pomyłkę wybrał niewłaściwą animację - po wykonaniu tej akcji, rysunek grafu wraca do stanu początkowego (sprzed animacji)) i możliwość wyłączenia prezentacji algorytmu, a potem wyświetlenia efektu jego działania (np. pokazanie znalezionego minimalnego drzewa rozpinającego (MST)). Ponadto, omawiana aplikacja daje większą możliwość manipulacji rysunkiem grafu - usuwanie krawędzi i wierzchołków, manualne oraz automatyczne dodawanie wag do krawędzi, manualne zmienianie położenia wierzchołków grafu, manualne zmienianie położenia wag krawędzi oraz położenia danych związanych z wierzchołkami. Wyżej wymienione możliwości, dają użytkownikowi dużą swobodę w manipulacji rysunkiem grafu oraz wpływ na czytelność animacji (w stosunku do innych dostępnych programów).

Dodatkowo aplikacja umożliwia: zapisywanie rysunków do pliku, otwieranie zapisanych rysunków oraz usuwanie niepotrzebnych/nieużywanych plików - to daje dużą swobodę w przygotowaniu animacji. Użytkownik może raz utworzyć rysunek grafu, a potem w każdej chwili otworzyć plik z tym rysunkiem i uruchomić dowolną animację. Oprócz tego, omawiana aplikacja umożliwia uruchamianie wielu różnych animacji (w oddzielnych zakładkach, na różnych grafach) oraz sprawdzenie, czy graf spełnia wymogi formalne wykonania na nim algorytmu (jeśli nie, to w terminalu aplikacji są wyświetlane stosowne komunikaty błędów).

Zatem omawiana aplikacja daje większe możliwości manipulacji rysunkiem grafu oraz ułatwia przygotowanie graficznej prezentacji wykonania algorytmu (poprzez możliwość zapisu i otwierania utworzonych

grafów). Z drugiej strony, omawiana aplikacja nie daje tylu możliwości manipulowania wykonaniem animacji (taka funkcjonalność może ułatwić i uprzyjemnić korzystanie z programu) oraz nie przedstawia innych algorytmów, które są wizualizowane w pozostałych programach.

Pracując z grafami, należy przyjąć odpowiednią reprezentację grafu w komputerze. Istnieją dwie struktury, którymi można reprezentować grafy: macierz incydencji oraz lista sąsiedztwa. W omawianej aplikacji wybrano listę sąsiedztwa, ze względu na to, że jest efektywniejsza pamięciowo niż macierz incydencji (mimo tego, że dostawanie się do elementów macierzy incydencji ma złożoność $\mathcal{O}(1)$, czyli lepszą niż w przypadku listy sąsiedztwa ($\mathcal{O}(|E| + |V|)$)). Do reprezentacji sąsiedniego wierzchołka, wagi krawędzi i danych dotyczących "fizycznej" krawędzi (tj. współrzędne początku i końca krawędzi, kolor), użyto krotki:

$$(u, w, [x_1, y_1, x_2, y_2], c, w_x, w_y), \tag{1.1}$$

gdzie u oznacza wierzchołek będący sąsiadem (dowolnego) wierzchołka v, (x_1, y_1) to współrzędne początku krawędzi, (x_2, y_2) to współrzędne końca krawędzi, c to liczba oznaczająca aktualny kolor krawędzi a (w_x, w_y) to współrzędne wagi krawędzi. Takie krotki są umieszczone w odpowiednim miejscu na liście sąsiedztwa (konkretnie znajdują się na liście sąsiadów danego wierzchołka), i na tej reprezentacji pracuje aplikacja. Oprócz reprezentacji grafu i krawędzi, trzeba jeszcze zapewnić reprezentację wierzchołków, która ma następującą postać:

$$(c_x, c_y, n, c, d, d_x, d_y), \tag{1.2}$$

gdzie (c_x, c_y) są współrzędnymi położenia wierzchołka (traktowanego jako okrąg o określonym promieniu), n jest numerem wierzchołka, c jest liczbą całkowitą oznaczającą jego kolor, d jest zmienną na dane związane z wierzchołkiem oraz (d_x, d_y) są współrzędnymi położenia tych danych. Mając te dane, aplikacja może swobodnie wpływać na reprezentację fizyczną grafu i dzięki temu, prezentować wykonanie wybranego algorytmu.

W projekcie są przedstawiane graficznie następujące algorytmy:

1. Algorytm Prima i algorytm Kruskala - szukanie minimalnego drzewa rozpinającego.
2. Algorytm szukania silnie spójnych składowych w grafie skierowanym.
3. Algorytm Forda-Fulkersona i algorytm Edmondsa-Karpa - szukanie maksymalnego przepływu.

Zostały one wybrane, ponieważ nie są trywialne, a przedstawienie graficzne znacznie ułatwia zrozumienie ich działania. Dwa pierwsze algorytmy bazują na własności przekroju w grafach oraz podejściu zachłannym. Trzeci algorytm bazuje na algorytmie DFS oraz numerach "post" odwiedzanych wierzchołków (używa ich do szukania składowych ujściowych w grafach skierowanych). Dwa ostatnie algorytmy są podstawowymi metodami znajdowania maksymalnego przepływu w sieciach przepływowych, bazujących na koncepcji sieci residualnej indukowanej przez funkcję przepływu (należy wspomnieć o tym, że algorytm Edmondsa-Karpa jest modyfikacją algorytmu Forda-Fulkersona, polegającą na tym, że ścieżka ze źródła do ujścia jest zawsze najkrótsza (w sensie liczby krawędzi - do szukania ścieżek jest używany algorytm BFS)).

Rozdział 2

Projekt systemu

W tym rozdziale jest przedstawiony projekt aplikacji wykonany za pomocą następujących diagramów UML:

1. Diagram przypadków użycia.
2. Diagram komponentów.
3. Diagram klas.
4. Diagram aktywności.
5. Diagram sekwencji.

Ponadto, w poniższym podrozdziale, zostały zamieszczone scenariusze przypadków użycia.

2.1 Diagram przypadków użycia i scenariusze

Diagram przypadków użycia 2.1 przedstawia wszystkie istotne przypadki użycia. W przypadku tej aplikacji można wyszczególnić tylko jednego użytkownika (tego, który z tej aplikacji korzysta).

Dla wybranych przypadków użycia zostały zamieszczone ich scenariusze, które zawierają informacje dotyczące: priorytetu, aktora podstawowego, wyzwalacza, typu wyzwalacza, powiązań, zwykłego przepływu zdarzeń, alternatywnego przepływu zdarzeń oraz przepływów pobocznych. W ten sposób zostały opisane następujące przypadki użycia:

2.2 - scenariusz dla tworzenia pliku z rysunkiem grafu.

2.3 - scenariusz dla otwierania pliku z rysunkiem grafu.

2.4 - scenariusz dla zapisywania pliku z rysunkiem grafu.

2.5 - scenariusz dla usuwania pliku z rysunkiem grafu.

2.6 - scenariusz dla tworzenia rysunku grafu.

2.7 - scenariusz dla manipulacji rysunkiem grafu.

2.8 - scenariusz dla wyboru i uruchomienia animacji, przedstawiającej wybrany przez użytkownika algorytm grafowy.

2.9 - scenariusz dla wyświetlania instrukcji obsługi dla użytkownika.

2.10 - scenariusz dla przywracania początkowego rysunku grafu.

2.11 - scenariusz dla usuwania komunikatów generowanych przez aplikację z terminala.

Rysunek 2.1: Diagram przypadków użycia.

2.12 - scenariusz dla manipulacji jakością animacji.

2.13 - scenariusz dla wyłączania animacji.

We wszystkich scenariuszach został określony: priorytet, aktor podstawowy, udziałowcy i cele, wyzwalacz, typ wyzwalacza, powiązania oraz zwykły przepływ zdarzeń. Natomiast, przepływy poboczne oraz alternatywne, zostały określone tylko dla niektórych przypadków użycia. Jest to spowodowane tym, że część przypadków po prostu nie posiada przepływów pobocznych lub nie przewiduje żadnych sytuacji, których obsługa wymaga wykonania przepływu alternatywnego (np. przypadek użycia 2.2 lub 2.9).

2.2 Diagram komponentów

W tym podrozdziale przedstawiono diagram komponentów, który obrazuje podział aplikacji na współpracujące ze sobą moduły. Z diagramu 2.14 wynika, że system składa się z następujących komponentów:

`general` - zawierający klasy odpowiadające za obsługę najbardziej podstawowych funkcjonalności aplikacji (rysowanie grafu, operacje na plikach itd.).

`graph_checkers` - zawierający klasy odpowiadające za sprawdzanie poprawności grafu w kontekście założeń algorytmu.

`animators` - zawierający klasy odpowiadające za animowanie algorytmów.

`algorithms` - zawierający klasy odpowiadające za wykonanie algorytmu na grafie oraz przygotowanie listy kroków, które mają zostać pokazane w animacji.

`resources` - zawierający stałe wykorzystywane w obliczeniach i rysowaniu, kolory oraz komunikaty (które są wypisywane na terminal).

`dialogs` - zawierający klasy okienek dialogowych, służących do interakcji z użytkownikiem.

`data_structures` - zawierający implementacje kolejek priorytetowych, używanych w algorytmie Prima i Kruskala.

Z diagramu 2.14 wynika, że każdy komponent jest zależny od komponentu `resources` (oprócz niego samego) oraz, że komponent `general` jest zależny od reszty komponentów (komponent `general` realizuje część najbardziej podstawowych funkcjonalności, które muszą zostać uzupełnione o pozostałe, realizowane przez resztę komponentów).

2.3 Diagram klas

Diagram 2.15 przedstawia wszystkie klasy tworzące aplikację, oraz zależności między nimi. Na diagramie można zauważyć, że klasa `MainWindow` korzysta z pozostałych klas. Jest to spowodowane tym, że ta klasa odpowiada na działania użytkownika i deleguje jego akcje do odpowiednich klas, które mogą ją obsłużyć. Po obsłużeniu akcji przez odpowiednią klasę, wynik działania pojawia się w oknie głównym aplikacji (tj. w terminalu aplikacji lub w oknie do rysowania).

Za pomocą tego diagramu, można zauważyć, że (jedno) okno główne aplikacji posiada wiele instancji innych klas. Ma to na celu umożliwienie użytkownikowi tworzenia wielu nowych rysunków oraz uruchamiania animacji wybranych (różnych) algorytmów w jednym oknie. Ponadto widać, że każde ”płótno” do rysowania ma dokładnie jeden obiekt reprezentujący graf. Ma to na celu uniknięcie komplikacji takich jak np. rysowanie grafu nieskierowanego i skierowanego na jednym ”płótnie” (wtedy powstaje problem np. na którym grafie uruchomić wybraną animację).

2.4 Diagram aktywności

Diagramy 2.16 oraz 2.17 przedstawiają logikę stojącą za procesami rysowania grafu oraz animowania wybranego przez użytkownika algorytmu.

W przypadku pierwszego procesu użytkownik najpierw musi wybrać opcję tworzenia rysunku grafu oraz określić jego rodzaj (skierowany/nieskierowany). Potem pojawia się nowa karta, gdzie użytkownik może tworzyć rysunek grafu i manipulować nim (dodawać wierzchołki, krawędzie itd.). Na koniec użytkownik musi zdecydować, czy zapisać rysunek grafu. Jeśli tak, to aplikacja zapisuje stworzony przez użytkownika rysunek do pliku i proces rysowania kończy się. Jeśli nie, to program nic nie robi i od razu kończy proces rysowania.

W przypadku drugiego procesu użytkownik najpierw musi wybrać algorytm, którego wizualizację chce obejrzeć. Po wybraniu algorytmu, aplikacja sprawdza, czy graf narysowany przez użytkownika jest poprawny w kontekście założeń stawianych przez algorytm. Jeśli nie, to aplikacja wyświetla komunikaty błędów i proces animowania zostaje zakończony. Jeśli tak, to aplikacja rozpoczyna animowanie wybranego przez użytkownika algorytmu. Wtedy aplikacja przedstawia aktualny krok algorytmu na grafie, wyświetla odpowiedni komunikat na temat tego kroku oraz zaznacza aktualny krok algorytmu w pseudokodzie. Na koniec procesu animowania, jest przedstawiany efekt końcowy działania algorytmu. Potem proces animowania jest zakończony.

2.5 Diagramy sekwencji

Diagramy 2.18, 2.19, 2.20 oraz 2.21 przedstawiają interakcje poszczególnych klas w czasie oraz komunikaty, które są między nimi wymieniane. W ten sposób zostały zobrazowane następujące procesy (odpowiednio):

2.18 - proces zapisywania grafu do pliku.

2.19 - proces usuwania pliku z rysunkiem grafu.

2.20 - proces otwierania istniejącego pliku.

2.21 - proces rysowania.

Nazwa PU: *Tworzenie pliku z rysunkiem grafu*	**Numer PU:** *1*	**Priorytet:** *Wysoki*
Aktor podstawowy: *Użytkownik*	**Typ opisu:** *Szczegółowy*	
Udziałowcy i cele: *Utworzenie nowego pliku*		
Wyzwalacz: *Wciśnięcie przycisku "Nowy" lub kombinacji klawiszy "Ctrl+N"*	**Typ wyzwalacza:** *Zewnętrzny*	
Powiązania: **Asocjacja:** *Tworzenie rysunku grafu*		
Zwykły przepływ zdarzeń: 1. *Użytkownik wciska przycisk "Nowy" lub kombinację klawiszy "Ctrl+N".* 2. *Pojawia się okienko dialogowe, w którym użytkownik podaje nazwę pliku oraz wybiera rodzaj grafu (domyślnie jest tworzony graf nieskierowany).* 3. *Użytkownik wciska przycisk "Ok".* 4. *Otwiera się zakładka, w której użytkownik może tworzyć rysunek grafu.* 5. *W terminalu aplikacji pojawia się stosowna informacja o wykonanej operacji.*		
Przepływy poboczne: 1. *Użytkownik wciska przycisk "Nowy" lub kombinację klawiszy "Ctrl+N".* 2. *Użytkownik wciska przycisk "Cancel".* 3. *Proces tworzenia nowego pliku został przerwany. Okienko dialogowe zamyka się.*		

Rysunek 2.2: Scenariusz przypadku użycia dla tworzenia pliku.

Nazwa PU: *Otwieranie pliku z rysunkiem grafu*	**Numer PU:** *2*	**Priorytet:** *Wysoki*
Aktor podstawowy: *Użytkownik*	**Typ opisu:** *Szczegółowy*	
Udziałowcy i cele: *Otwieranie istniejącego pliku z rysunkiem grafu*		
Wyzwalacz: *Wciśnięcie przycisku "Otwórz" lub kombinacji "Ctrl+O"*	**Typ wyzwalacza:** *Zewnętrzny*	
Powiązania: **Asocjacja:** *Manipulacja rysunkiem grafu, wybór i uruchomienie animacji*		
Zwykły przepływ zdarzeń: 1. *Użytkownik wciska przycisk "Otwórz" lub kombinację klawiszy "Ctrl+O".* 2. *Otwiera się okienko dialogowe z listą istniejących plików do otworzenia .* 3. *Użytkownik wybiera z listy pliki, które chce otworzyć.* 4. *Użytkownik wciska przycisk "Ok".* 5. *Otwierają się zakładki z zapisanymi rysunkami grafów.* 6. *W terminalu aplikacji pojawia się stosowna informacja o wykonanej operacji.*		
Przepływy poboczne: 1. *Użytkownik wciska przycisk "Otwórz" lub kombinację klawiszy "Ctrl+O".* 2. *Pojawia się okienko dialogowe z listą istniejących plików do otworzenia.* 3. *Użytkownik wciska przycisk "Cancel".* 4. *Proces otwierania istniejących plików zostaje przerwany. Okienko dialogowe zamyka się.*		
Przepływy alternatywne: 1. *Użytkownik wciska przycisk "Otwórz" lub kombinację klawiszy "Ctrl+O".* 2. *Otwiera się okienko dialogowe z listą istniejących plików do otworzenia .* 3. *Użytkownik wybiera z listy pliki, które chce otworzyć.* 4. *Użytkownik wciska przycisk "Ok".* 5. *Otwieranie jednego z wybranych plików nie powiodło się.*		

Rysunek 2.3: Scenariusz przypadku użycia dla otwierania istniejącego pliku.

Nazwa PU: *Zapisywanie pliku z rysunkiem grafu*	**Numer PU:** *3*	**Priorytet:** *Wysoki*
Aktor podstawowy: *Użytkownik*	**Typ opisu:** *Szczegółowy*	
Udziałowcy i cele: *Zapisanie rysunku grafu do pliku*		
Wyzwalacz: *Wciśnięcie przycisku "Zapisz" lub kombinacji klawiszy "Ctrl+S"*	**Typ wyzwalacza:** *Zewnętrzny*	
Zwykły przepływ zdarzeń: 1. *Użytkownik wciska przycisk "Zapisz" lub kombinację klawiszy "Ctrl+S".* 2. *Graf zostaje zapisany do pliku.* 3. *W terminalu aplikacji pojawia się stosowna informacja o wykonanej operacji.*		
Przepływy alternatywne: 1. *Użytkownik wciska przycisk "Zapisz" lub kombinację klawiszy "Ctrl+S".* 2. *Zapisywanie grafu do pliku nie powiodło się.* 3. *W terminalu aplikacji pojawia się stosowny komunikat błędu.*		

Rysunek 2.4: Scenariusz przypadku użycia dla zapisywania pliku.

Nazwa PU: *Usuwanie pliku z rysunkiem grafu*	**Numer PU:** *4*	**Priorytet:** *Średni*
Aktor podstawowy: *Użytkownik*	**Typ opisu:** *Szczegółowy*	
Udziałowcy i cele: *Usunięcie wybranych przez użytkownika plików z rysunkami grafów*		
Wyzwalacz: *Wciśnięcie przycisku "Usuń" w menu "Plik"*	**Typ wyzwalacza:** *Zewnętrzny*	
Zwykły przepływ zdarzeń: 1. *Użytkownik wciska przycisk "Usuń" w menu "Plik".* 2. *Pojawia się okienko dialogowe z listą plików do wyboru.* 3. *Użytkownik wybiera plik do usunięcia.* 4. *Użytkownik zatwierdza wybór poprzez wciśnięcie przycisku "Ok".* 5. *Plik zostaje usunięty.* 6. *W terminalu aplikacji pojawia się stosowny komunikat o wykonanej operacji.*		
Przepływy alternatywne: 1. *Użytkownik wciska przycisk "Usuń" w menu "Plik".* 2. *Pojawia się okienko dialogowe z listą plików do wyboru.* 3. *Użytkownik wybiera plik do usunięcia.* 4. *Użytkownik zatwierdza wybór poprzez wciśnięcie przycisku "Ok".* 5. *Usunięcie wybranego pliku nie powiodło się.* 6. *W terminalu aplikacji pojawia się stosowny komunikat błędu.*		

Rysunek 2.5: Scenariusz przypadku użycia dla usuwania pliku.

Nazwa PU: *Tworzenie rysunku grafu*	**Numer PU:** *5*	**Priorytet:** *Wysoki*
Aktor podstawowy: *Użytkownik*	**Typ opisu:** *Szczegółowy*	
Udziałowcy i cele: *Utworzenie rysunku grafu*		
Wyzwalacz: *Utworzenie nowego pliku lub otwieranie już istniejącego pliku*	**Typ wyzwalacza:** *Zewnętrzny*	
Powiązania: **Asocjacja:** *Tworzenie pliku z rysunkiem grafu, otwieranie pliku z rysunkiem grafu* **Generalizacja:** *Manipulacja rysunkiem grafu*		
Zwykły przepływ zdarzeń: 1. *Użytkownik tworzy nowy plik lub otwiera już istniejący plik.* 2. *W otworzonej zakładce użytkownik dodaje wierzchołki, krawędzie, wagi krawędzi, zmienia położenie wierzchołków, usuwa wybrane krawędzie lub wierzchołki itd. .*		

Rysunek 2.6: Scenariusz przypadku użycia dla tworzenia rysunku grafu.

Nazwa PU: *Manipulacja rysunkiem grafu*	**Numer PU:** *6*	**Priorytet:** *Średni*
Aktor podstawowy: *Użytkownik*	**Typ opisu:** *Szczegółowy*	
Udziałowcy i cele: *Zmiana wyglądu rysunku grafu*		
Wyzwalacz: *Naciśnięcie prawego przycisku myszy nad krawędzią lub wierzchołkiem albo naciśnięcie lewym przyciskiem myszy na wierzchołek lub wagę krawędzi*	**Typ wyzwalacza:** *Zewnętrzny*	
Powiązania: **Asocjacja:** *Tworzenie rysunku grafu*		
Zwykły przepływ zdarzeń: 1. *Użytkownik naciska prawym przyciskiem myszy nad wierzchołkiem lub krawędzią.* 2. *Pojawia się menu kontekstowe z opcjami do wyboru.* 3. *Użytkownik wybiera jedną z dostępnych opcji.* 4. *Menu kontekstowe zamyka się.* 5. *Żądana przez użytkownika akcja zostaje wykonana.*		
Przepływy poboczne: 1. *Użytkownik naciska lewym przyciskiem myszy na wierzchołek lub wagę krawędzi.* 2. *Użytkownik przesuwa wierzchołek lub wagę krawędzi na wybrane przez niego miejsce.* 3. *Wierzchołek lub waga krawędzi jest teraz rysowana w nowym, wybranym przez użytkownika miejscu.*		

Rysunek 2.7: Scenariusz przypadku użycia dla manipulacji rysunkiem grafu.

Nazwa PU: *Wybór i uruchomienie animacji*	**Numer PU:** *7*	**Priorytet:** *Wysoki*
Aktor podstawowy: *Użytkownik*	**Typ opisu:** *Szczegółowy*	
Udziałowcy i cele: *Uruchomienie żądanej przez użytkownika animacji*		
Wyzwalacz: *Kliknięcie nazwy animacji żądanego algorytmu w menu "Algorytmy"*	**Typ wyzwalacza:** *Zewnętrzny*	
Powiązania: **Asocjacja:** *Tworzenie rysunku grafu, wyłączenie animacji, wyłączenie animacji i wyświetlenie efektu końcowego działania algorytmu, otwieranie istniejącego pliku z rysunkiem grafu* **Zawieranie:** *Wyświetlanie adekwatnych komunikatów odnośnie aktualnego kroku algorytmu, pokazanie aktualnego kroku w pseudokodzie, wyświetlanie odpowiednich komunikatów błędów,* *sprawdzenie poprawności grafu w kontekście założeń stawianych przez algorytm*		
Zwykły przepływ zdarzeń: 1. *Użytkownik naciska na nazwę żądanego algorytmu w menu "Algorytmy".* 2. *Aplikacja sprawdza, czy graf jest poprawny w kontekście założeń stawianych przez algorytm.* 3. *Aplikacja przedstawia graficznie wykonanie wybranego algorytmu.* 4. *Animacja kończy się. Wyświetlany jest efekt końcowy działania algorytmu.*		
Przepływy alternatywne: 1. *Użytkownik naciska na nazwę żądanego algorytmu w menu "Algorytmy".* 2. *Aplikacja sprawdza, czy graf jest poprawny w kontekście założeń stawianych przez algorytm.* 3. *Graf nie spełnia wymaganych przez algorytm założeń.* 4. *Aplikacja nie przedstawia wybranej animacji.*		

Rysunek 2.8: Scenariusz przypadku użycia dla wyboru i uruchamiania animacji.

Nazwa PU: *Możliwość wyświetlenia instrukcji obsługi aplikacji*	**Numer PU:** *8*	**Priorytet:** *Niski*
Aktor podstawowy: *Użytkownik*	**Typ opisu:** *Ogólny*	
Udziałowcy i cele: *Wyświetlenie instrukcji obsługi dla użytkownika*		
Wyzwalacz: *Wciśnięcie opcji "O programie" w menu "Pomoc"*	**Typ wyzwalacza:** *Zewnętrzny*	
Zwykły przepływ zdarzeń: 1. *Użytkownik wciska opcję "O programie" w menu "Pomoc".* 2. *Wyświetla się okno dialogowe z instrukcją obsługi dla użytkownika.*		

Rysunek 2.9: Scenariusz przypadku użycia dla wyświetlania instrukcji obsługi aplikacji.

Nazwa PU: *Możliwość przywrócenia początkowego rysunku grafu (po animacji)*	**Numer PU:** *9*	**Priorytet:** *Średni*
Aktor podstawowy: *Użytkownik*	**Typ opisu:** *Ogólny*	
Udziałowcy i cele: *Przywrócenie rysunku grafu po animacji*		
Wyzwalacz: *Kliknięcie opcji "Przywróć" w pasku narzędzi*	**Typ wyzwalacza:** *Zewnętrzny*	
Powiązania: **Asocjacja:** *Wybór i uruchomienie animacji, wyłączenie animacji i wyświetlenie efektu końcowego działania algorytmu*		
Zwykły przepływ zdarzeń: 1. *Użytkownik wciska opcję "Przywróć" w pasku narzędzi.* 2. *Graf zostaje przywrócony do pierwotnej postaci (sprzed animacji).*		

Rysunek 2.10: Scenariusz przypadku użycia dla przywracania pierwotnej postaci grafu (sprzed animacji).

Nazwa PU: *Możliwość wyczyszczenia komunikatów generowanych przez aplikację*	**Numer PU:** *10*	**Priorytet:** *Średni*
Aktor podstawowy: *Użytkownik*	**Typ opisu:** *Ogólny*	
Udziałowcy i cele: *Usunięcie z terminala aplikacji zbędnych komunikatów*		
Wyzwalacz: *Wciśnięcie opcji "Wyczyść terminal" w menu "Terminal"*	**Typ wyzwalacza:** *Zewnętrzny*	
Powiązania: **Asocjacja:** *Usuwanie pliku, otwieranie istniejącego pliku, tworzenie pliku z rysunkiem grafu, wybór i uruchomienie animacji, wyłączenie animacji, wyłączenie animacji i wyświetlenie efektu końcowego działania algorytmu*		
Zwykły przepływ zdarzeń: 1. *Użytkownik wciska opcję "Wyczyść terminal" W menu "Terminal".* 2. *Terminal aplikacji nie zawiera żadnych komunikatów.*		

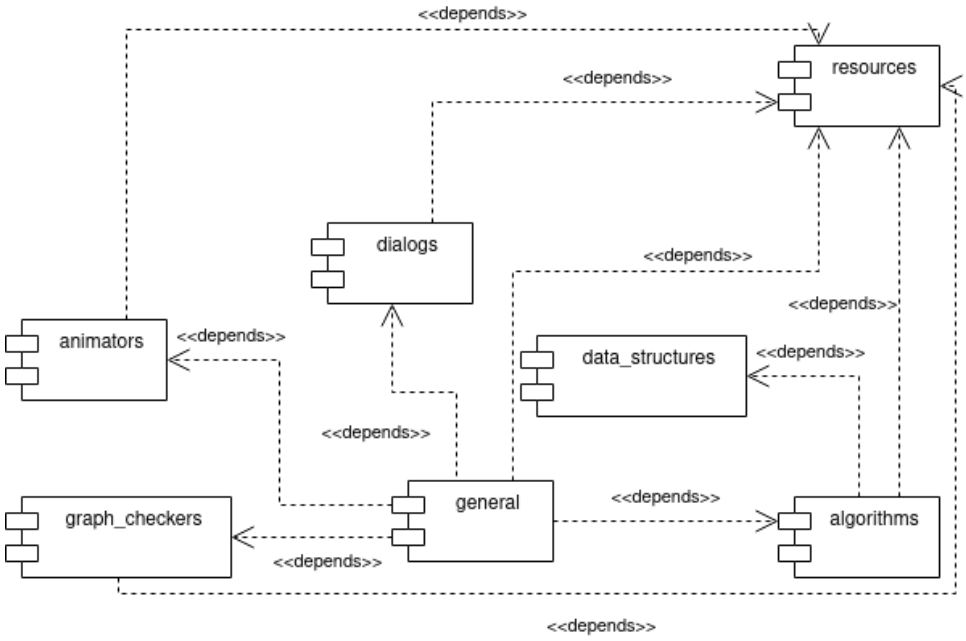
Rysunek 2.11: Scenariusz przypadku użycia dla czyszczenia terminala aplikacji.

Nazwa PU: *Manipulacja jakością animacji*	**Numer PU:** *11*	**Priorytet:** *Średni*
Aktor podstawowy: *Użytkownik*	**Typ opisu:** *Ogólny*	
Udziałowcy i cele: *Wpływanie na jakość (czytelność) animacji*		
Wyzwalacz: *Kliknięcie opcji "Wyświetl dane wierzchołków" lub kliknięcie na wagę krawędzi*	**Typ wyzwalacza:** *Zewnętrzny*	
Powiązania: **Asocjacja:** *Manipulacja rysunkiem grafu, tworzenie pliku z rysunkiem grafu, otwieranie pliku z rysunkiem grafu, tworzenie rysunku grafu*		
Zwykły przepływ zdarzeń: 1. *Użytkownik wybiera opcję "Wyświetl dane wierzchołków" (w pasku narzędzi) lub przyciska lewym przyciskiem myszy na wagę wybranej krawędzi.* 2. *Użytkownik manipuluje położeniem danych dotyczących wierzchołków lub krawędzi.*		

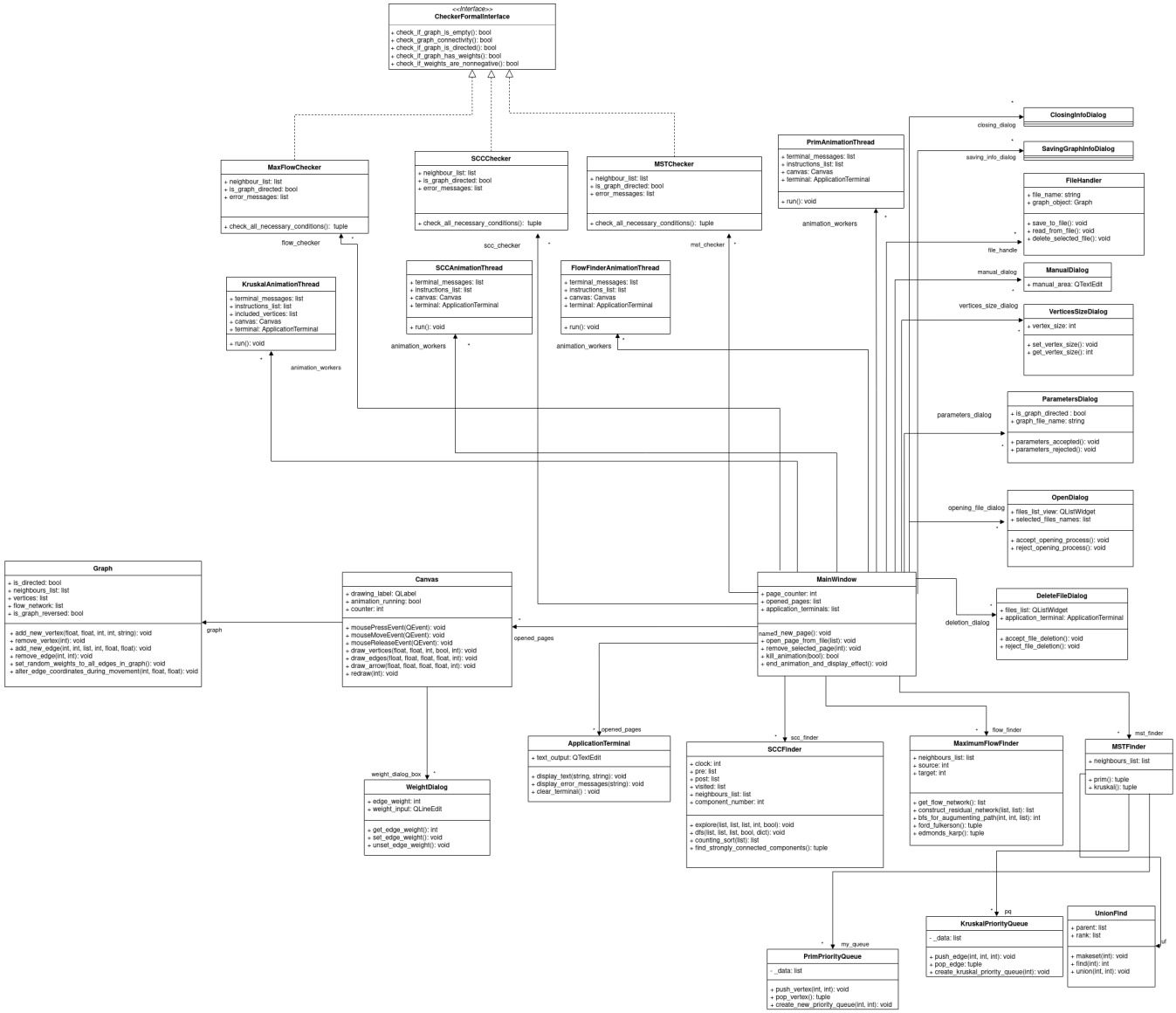
Rysunek 2.12: Scenariusz przypadku użycia dla manipulowania jakością animacji.

Nazwa PU: *Wyłączenie animacji*	**Numer PU:** *12*	**Priorytet:** *Średni*
Aktor podstawowy: *Użytkownik*	**Typ opisu:** *Ogólny*	
Udziałowcy i cele: *Wyłączenie bieżącej animacji*		
Wyzwalacz: *Wciśnięcie opcji "Zakończ" lub "Zakończ i wyświetl efekt końcowy" w menu "Animacja"*	**Typ wyzwalacza:** *Zewnętrzny*	
Powiązania: **Asocjacja:** *Wybór i uruchomienie animacji* **Rozszerzenie:** *Wyłączenie animacji i wyświetlenie efektu końcowego działania algorytmu*		
Zwykły przepływ zdarzeń: 1. *Użytkownik uruchamia animację przedstawiającą wybrany przez niego algorytm.* 2. *Użytkownik wybiera opcję "Zakończ".* 3. *Animacja wyłącza się. Narysowany przez użytkownika graf powraca do postaci początkowej.*		
Przepływy poboczne: 1. *Użytkownik uruchamia animację przedstawiającą wybrany przez niego algorytm.* 2. *Użytkownik wybiera opcję "Zakończ i wyświetl efekt końcowy".* 3. *Animacja wyłącza się. Zostaje wyświetlony efekt końcowy działania wybranego algorytmu.*		

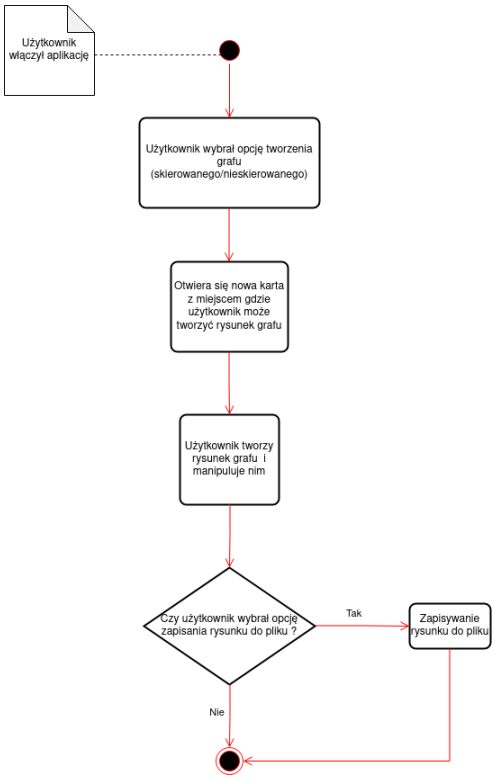
Rysunek 2.13: Scenariusz przypadku użycia dla wyłączania animacji.



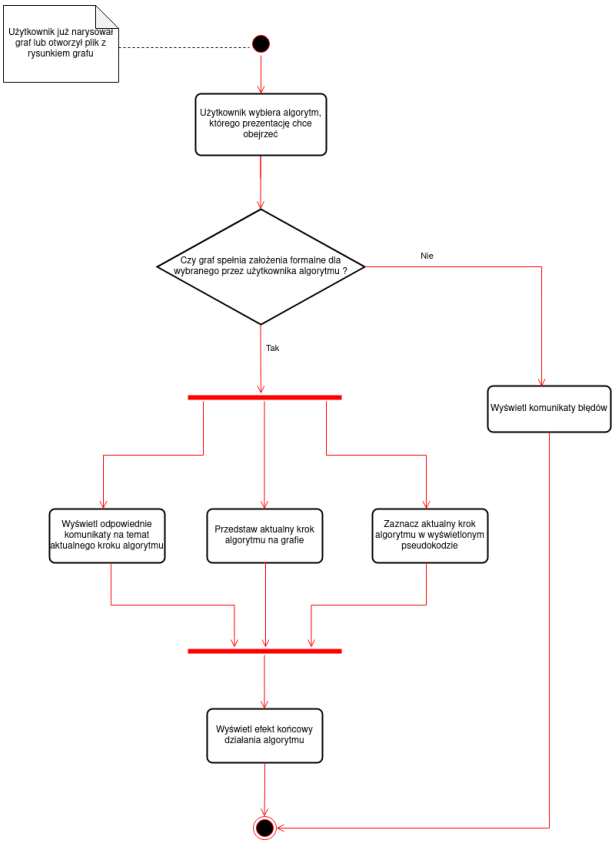
Rysunek 2.14: Diagram komponentów.



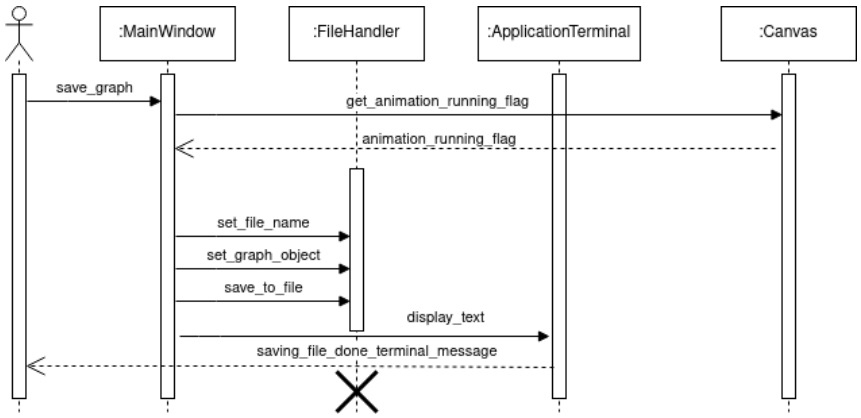
Rysunek 2.15: Diagram klas.



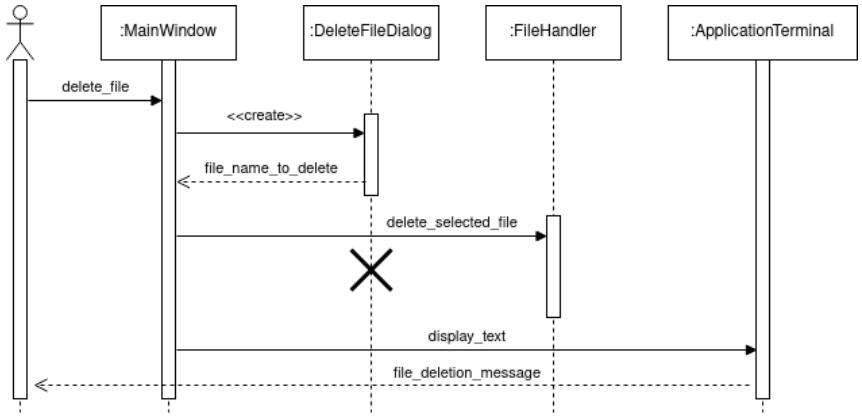
Rysunek 2.16: Diagram aktywności przedstawiający proces rysowania.



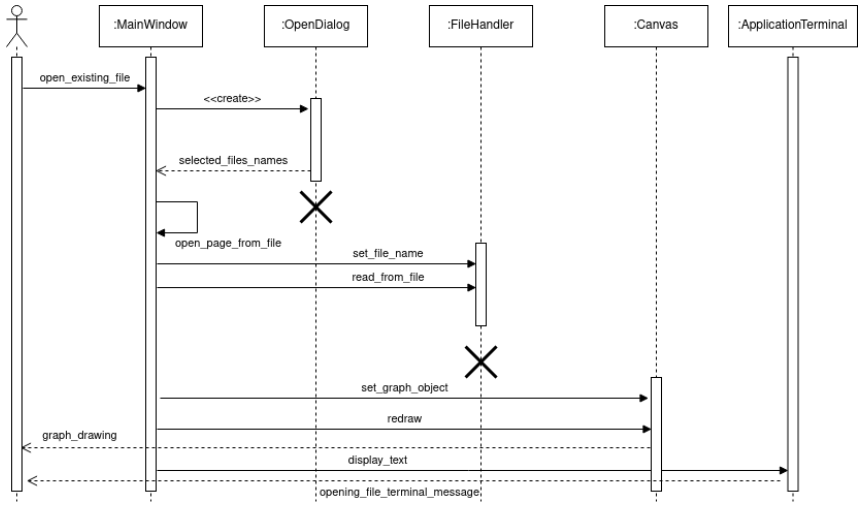
Rysunek 2.17: Diagram aktywności przedstawiający proces animowania wybranego algorytmu.



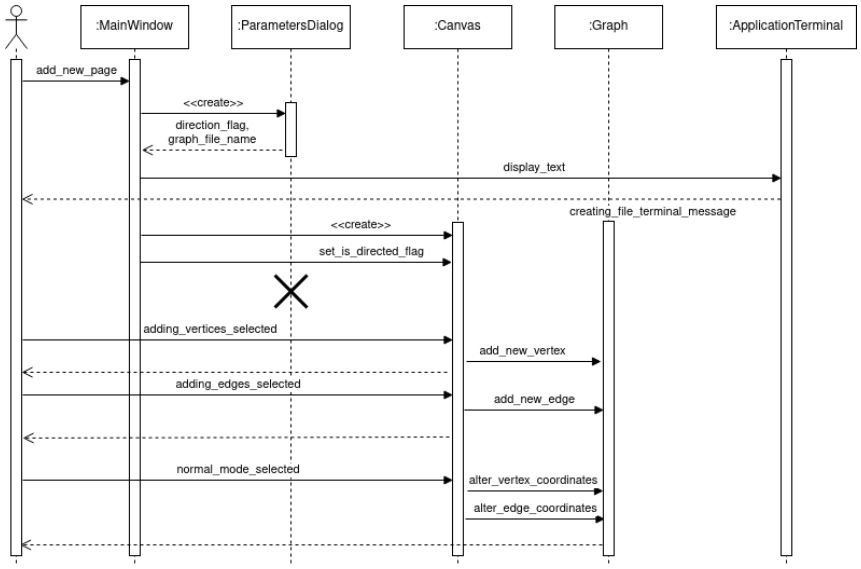
Rysunek 2.18: Diagram sekwencji przedstawiający proces zapisywania rysunku grafu.



Rysunek 2.19: Diagram sekwencji przedstawiający proces usuwania pliku z rysunkiem grafu.



Rysunek 2.20: Diagram sekwencji przedstawiający proces otwierania istniejącego pliku.



Rysunek 2.21: Diagram sekwencji przedstawiający proces rysowania.

Rozdział 3

Opis użytych algorytmów

W tym rozdziale zostały przedstawione i omówione, najważniejsze algorytmy i struktury danych użyte w aplikacji. Tematyka tego rozdziału jest obszernie opisana w następujących książkach: [6, rozdział 26] oraz [7, rozdziały 3 i 5].

3.1 Algorytm DFS

Algorytm DFS (pseudokod 3.2) jest wykorzystywany w algorytmie znajdowania silnie spójnych składowych w grafie skierowanym oraz w algorytmie Forda-Fulkersona (do szukania $s-t$ ścieżek). Algorytm działa następująco:

1. Oznaczyć wszystkie wierzchołki jako nieodwiedzone (linijki 2-3 w 3.2).
2. Dla każdego wierzchołka $v \in V$ jeśli jest nieodwiedzony, to wykonać na nim procedurę `explore` (linijki 4-6 w 3.2).

Algorytm DFS korzysta z procedury `explore` (pseudokod 3.1). Sposób jej działania polega na tym, że zaczyna w danym wierzchołku startowym. Jeśli ten wierzchołek startowy ma nieodwiedzonych sąsiadów, to procedura przetwarza jeden z sąsiednich wierzchołków, potem przetwarza nieodwiedzonych sąsiadów tego wierzchołka i tak dalej. Jeśli okaże się, że dany wierzchołek nie ma żadnych nieodwiedzonych sąsiadów, to wtedy procedura powraca do poprzednio przetwarzanego wierzchołka (i tam sprawdza, czy nie ma nieodwiedzonych wierzchołków sąsiednich). Jeśli podczas działania procedury zostały odwiedzone wszystkie wierzchołki osiągalne z wierzchołka startowego, to wtedy procedura powraca do tego wierzchołka startowego i kończy działanie (linijki 3-5 w 3.1). Krótko mówiąc, algorytm DFS dla danego wierzchołka startowego przetwarza wszystkie wierzchołki, które są z niego osiągalne (w szczególności jeśli graf jest nieskierowany i spójny, to odwiedzi wszystkie wierzchołki w grafie).

W procedurze `explore` można również rejestrować momenty pierwszego wejścia do wierzchołka oraz ostatniego wyjścia z niego (czyli tzw. numery *pre* i *post*). Polega to na inkrementowaniu zmiennej po zaznaczeniu, że wierzchołek został odwiedzony oraz w momencie, w którym procedura wycofuje się z wierzchołka (bo nie ma żadnych nieodwiedzonych sąsiadów). Numer *post* wierzchołka będzie wykorzystywany przy szukaniu silnie spójnych składowych grafu.

Złożoność obliczeniowa tego algorytmu to $\mathcal{O}(|V|+|E|)$, ponieważ łączna praca wykonana w pierwszej pętli pseudokodu 3.2 to $\mathcal{O}(|V|)$, a w procedurze `explore` każdą krawędź $\{u, v\}$ rozpatrujemy 2 razy - raz dla wierzchołka u i drugi raz dla wierzchołka v. Zatem złożoność obliczeniowa tej części wynosi $\mathcal{O}(|E|)$, więc złożoność obliczeniowa całego algorytmu DFS wynosi $\mathcal{O}(|V|+|E|)$. Ta złożoność jest możliwie najlepsza, ponieważ jest to złożoność potrzebna tylko do przeczytania całej listy sąsiedztwa grafu.

3.2 Algorytm sprawdzania spójności grafu nieskierowanego

Algorytm sprawdzania spójności grafu nieskierowanego (3.3) korzysta z procedury `explore` i własności wspomnianej w opisie algorytmu DFS - jeśli graf jest spójny, to procedura `explore` odwiedzi wszystkie

wierzchołki w grafie. Zatem działanie tego algorytmu polega na oznaczeniu wszystkich wierzchołków jako nieodwiedzone (linijki 3-4 w 3.3), potem wywołaniu procedury `explore` na wybranym wierzchołku początkowym (dowolnym) (linijki 5-6 w 3.3), a na koniec (po zakończeniu działania porcedury) sprawdzeniu, czy wszystkie wierzchołki zostały odwiedzone - jeśli nie, to graf nie był spójny (linijki 7-10 w 3.3) . W przeciwnym przypadku graf był spójny.

Złożoność tego algorytmu również wynosi $\mathcal{O}(|V| + |E|)$, ponieważ pierwsza i ostatnia pętla w pseudokodzie 3.3 ma złożoność $\mathcal{O}(|V|)$, a procedura `explore` ma złożoność $\mathcal{O}(|E|)$ (co zostało uzasadnione w poprzednim podrozdziale). Zatem całkowita złożoność obliczeniowa wynosi $\mathcal{O}(|V| + |E|)$.

3.3 Algorytm wspomagający rysowanie krawędzi

Zadaniem funkcji 3.4 jest takie zmienienie współrzędnych początku i końca krawędzi, aby końce krawędzi nie były rysowane w obszarze wierzchołków (zatem zasadniczym celem tej procedury jest poprawienie jakości rysunku). W tym celu, zostają obliczone współrzędne wektora o początku w środku pierwszego wierzchołka i końcu w środku drugiego wierzchołka oraz jego długość (linijki 4-6 w 3.4). Jeśli wierzchołki są zbyt blisko siebie (w tym przypadku nie ma sensu rysować krawędzi), to procedura kończy działanie (linijki 7-8 w 3.4). Następnie jest obliczany współczynnik, wykorzystywany do zmieniania współrzędnych początku i końca wektora (linijka 9 w 3.4). Na koniec są obliczane nowe współrzędne początku i końca krawędzi, poprzez dodanie do (lub odjęcie od) poprzednich współrzędnych odpowiedniego współczynnika (linijki 10-13 w 3.4).

Ze względu na założenie, że wektor ma początek w pierwszym wierzchołku, a koniec w drugim, nowe współrzędne początku krawędzi są obliczane poprzez dodanie współczynnika do poprzednich współrzędnych (to samo się dzieje przy obliczaniu współrzędnych końca krawędzi, tylko wtedy ten współczynnik jest odejmowany).

Ze względu na to, że w tej procedurze są wykonywane głównie podstawowe działania arytmetyczne, jej złożoność obliczeniowa wynosi $\mathcal{O}(1)$.

3.4 Struktura Union-Find

W pseudokodzie 3.5 są przedstawione podstawowe operacje na strukturze danych `Union-Find`, która służy do reprezentacji zbiorów rozłącznych. W tym przypadku dany zbiór reprezentujemy jako drzewo skierowane z korzeniem, w którym każdy węzeł (element zbioru) zawiera wskaźnik na swojego ojca (korzeń jest ojcem sam dla siebie i jest reprezentantem całego zbioru). Dodatkowo, każdy węzeł ma rangę, którą można traktować jako wysokość poddrzewa ukorzenionego w tym węźle.

Procedura `makeset` ma charakter inicjacyjny i służy do reprezentacji zbioru jednoelementowego (drzewo skierowane składające się tylko z korzenia, mające rangę 0). Złożoność obliczeniowa tej funkcji wynosi $\mathcal{O}(1)$.

Procedura `union` służy do łączenia drzew reprezentujących zbiory rozłączne. Dla dwóch węzłów x i y, najpierw jest sprawdzane, czy już są w jednym drzewie (linijki 11-12 w 3.5). Jeśli tak, to funkcja kończy działanie. Jeśli nie, to wtedy jest pobierana wysokość każdego z dwóch drzew i wskaźnik (na ojca) korzenia niższego drzewa jest ustawiany na korzeń wyższego drzewa (linijki 13-18 w 3.5). Zatem zawsze niższe drzewo jest "przyłączane" do wyższego. Takie podejście zapewnia, że wysokość drzew nie będzie niepotrzebnie rosła, co przełoży się na lepszą efektywność operacji wykonywanych na strukturze `Union-Find`. Jeśli łączone drzewa mają taką samą wysokość, to wtedy wzrost wysokości drzewa wynikowego jest nieuchronny i trzeba zwiększyć wysokość drzewa wynikowego o 1 (linijki 19-20 w 3.5). Złożoność obliczeniowa tej operacji zależy od złożoności operacji `find` (linijka 9-10 w 3.5), ponieważ reszta działań w tej funkcji ma złożoność $\mathcal{O}(1)$.

Procedura `find` służy do szukania korzenia drzewa, w którym znajduje się dany węzeł x. Oczywiście można to robić poprzez "cofanie" się po wskaźnikach na ojca dopóki korzeń nie zostanie osiągnięty. Jednak można tę metodę usprawnić. Przy każdym wywołaniu funkcji `find`, wskaźnik danego węzła x jest ustawiany na korzeń drzewa, co pozwala na zmniejszanie wysokości drzew. Pożytek z takiego podejścia pojawia się, gdy mamy ciąg operacji `union` oraz `find`, dlatego tutaj lepiej używać pojęcia kosztu zamortyzowanego niż "tradycyjnej" złożoności obliczeniowej. Koszt zamortyzowany operacji `union` oraz `find`

Pseudokod 3.1: Pseudokod funkcji *explore*.

```
Input: Graf G = (V, E) oraz wierzchołek startowy u.
Output:
1 Function explore(u):
2     visited[u] ← True;
3     forall {u, v} ∈ E do
4         if not visited[v] then
5             explore(v);
```

Pseudokod 3.2: Pseudokod funkcji *DFS*.

```
Input: Graf G = (V, E).
Output:
1 Function DFS(G):
2     forall v ∈ V do
3         visited(v) ← False;
4     forall v ∈ V do
5         if not visited(v) then
6             explore(v);
```

Pseudokod 3.3: Pseudokod funkcji *checkGraphConnectivity* służącej do sprawdzenia, czy graf nieskierowany jest spójny.

```
Input: Graf nieskierowany G = (V, E).
Output: True albo False, w zależności od tego, czy graf jest spójny.
1 Function checkConnectivity():
2     n ← |V|;
3     forall u ∈ V do
4         visited[u] ← False;
5     Wybrać wierzchołek początkowy u_0;
6     explore(u_0);
7     forall u ∈ V do
8         if not visited[u] then
9             return False;
10    return True;
```

jest większy niż $\mathcal{O}(1)$, ale mniejszy niż $\mathcal{O}(\log n)$.

3.5 Algorytm Prima

W algorytmie Prima 3.6 są użyte dwie tablice: *prev* służąca do przechowywania znalezionego MST oraz *cost*, w której są przechowywane koszty poszczególnych wierzchołków. Najpierw koszty wszystkich wierzchołków są ustawiane na nieskończoność, a tablica *prev* jest inicjowana wartościami *nil* (nie ma jeszcze żadnego MST - linijki 1-3 w 3.6). Następnie wybierany jest wierzchołek początkowy (dowolny) i jego koszt jest ustalany na 0. Potem zostaje utworzona kolejka priorytetowa z wierzchołkami, w której kluczami są koszty wierzchołków (linijki 4-6 w 3.6). Później, dopóki kolejka nie jest pusta (czyli są przeglądane wszystkie wierzchołki), wybierany jest wierzchołek o najniższym koszcie i przeglądani są wszyscy sąsiedzi tego wierzchołka. Jeśli koszt sąsiedniego wierzchołka z jest większy niż waga krawędzi między nim, a bieżącym wierzchołkiem v, to wtedy koszt wierzchołka z jest ustawiany na wartość wagi krawędzi, $prev(z)$ jest ustawiane na v i klucz wierzchołka z w kolejce priorytetowej H zostaje zmniejszony do nowej wartości (linijki 7-13 w 3.6).

Można zauważyć, że jeśli podzielimy wierzchołki na dwa zbiory S - wierzchołki już znajdujące się w MST oraz $V \setminus S$ - wierzchołki, których jeszcze nie ma w MST, to zawsze w *prev* umieszczamy krawędź o najmniejszej wadze, która łączy wierzchołki z tych dwóch zbiorów. To jest wykorzystanie tzw. własności przekroju dla grafów.

Złożoność algorytmu Prima jest zależna od struktury danych, na której jest zaimplementowana używana kolejka priorytetowa. Jeśli jest użyty zwykły kopiec binarny typu min, to wtedy złożoność obliczeniowa algorytmu Prima wynosi $\mathcal{O}((|V| + |E|) \log |V|)$, ponieważ dla takiej kolejki priorytetowej, złożoność obliczeniowa operacji `deletemin` oraz `decreasekey` wynosi $\mathcal{O}(\log |V|)$.

3.6 Algorytm Kruskala

W algorytmie Kruskala 3.7 jest wykorzystywana struktura `Union-Find`, która służy do sprawdzania, czy dodanie danej krawędzi do MST, nie wytworzy cyklu (jeśli wytworzy cykl, to wtedy nie będzie drzewa). Najpierw każdy wierzchołek tworzy jednoelementowy zbiór w strukturze `Union-Find` (bo żaden wierzchołek jeszcze nie należy do MST - linijki 1-2 w 3.7). Następnie jest tworzony pusty zbiór X, w którym będą trzymane krawędzie tworzące MST dla danego grafu oraz krawędzie są sortowane względem wagi (rosnąco - linijki 3-4 w 3.7). Krawędzie można posortować, poprzez umieszczenie ich w kolejce priorytetowej, gdzie kluczami będą ich wagi. Następnie jest rozpatrywana każda krawędź znajdująca się w kolejce priorytetowej. Jeśli końcowe wierzchołki krawędzi należą do różnych zbiorów, to dodanie ich do MST nie spowoduje utworzenia cyklu. Zatem ta krawędź jest dodawana do zbioru X, a wierzchołki tworzące jej końce są umieszczane w tym samym zbiorze. Ten proces jest powtarzany dopóki nie zostaną rozpatrzone wszystkie krawędzie w kolejce priorytetowej (linijki 5-8 w 3.7).

Omawiany i poprzedni algorytm są przykładami algorytmów zachłannych, w których w każdym kroku algorytmu jest wybierane aktualnie najlepsze rozwiązanie (w tym przypadku to wybór krawędzi o najmniejszej wadze).

Złożoność obliczeniowa tego algorytmu wynosi $\mathcal{O}(|E| \log |E|)$, gdyż tyle potrzeba do posortowania krawędzi względem wagi (a to dominuje nad resztą algorytmu).

3.7 Algorytm znajdowania silnie spójnych składowych w grafie skierowanym

Algorytm szukania silnie spójnych składowych 3.8 w grafie skierowanym korzysta z następujących własności:

Twierdzenie 3.1 *Jeśli procedura* `explore` *startuje w wierzchołku u, to zakończy swoje działanie po odwiedzeniu wszystkich wierzchołków osiągalnych z u.*

Twierdzenie 3.2 *Wierzchołek, który otrzymał największy numer post w przeszukiwaniu w głąb, musi leżeć w źródłowej silnie spójnej składowej grafu skierowanego.*

Pseudokod 3.4: Pseudokod funkcji *correction*.

Input: Współrzędne środka pierwszego (u_x, u_y) i drugiego (v_x, v_y) wierzchołka.
Output: Zmienione współrzędne początku i końca krawędzi: $x_{beg}, y_{beg}, x_{end}, y_{end}$.

Function `correction`(u_x, u_y, v_x, v_y)**:**
- $radius \leftarrow vertices\_size$;
- $parameter \leftarrow \frac{(radius+3)}{2}$;
- $dx \leftarrow v_x - u_x$;
- $dy \leftarrow v_y - u_y$;
- $length \leftarrow \sqrt{(dx)^2 + (dy)^2}$;
- **if** $|length - 0.0| \leqslant precision$ **then**
  - **return**;
- $factor \leftarrow \frac{parameter}{length}$;
- $x_{beg} \leftarrow u_x + factor * dx$;
- $y_{beg} \leftarrow u_y + factor * dy$;
- $x_{end} \leftarrow v_x - factor * dx$;
- $y_{end} \leftarrow v_y - factor * dy$;
- **return** $x_{beg}, y_{beg}, x_{end}, y_{end}$;

Pseudokod 3.5: Struktura Union-Find i związane z nią operacje.

Input:
Output:

Function `makeset`(x)**:**
- $parent[x] \leftarrow x$;
- $rank[x] \leftarrow 0$;

Function `find`(x)**:**
- **if** $x \neq parent[x]$ **then**
  - $parent[x] \leftarrow find(parent[x])$;
- **return** $parent[x]$;

Function `union`(x, y)**:**
- $x\_root \leftarrow$ `find`(x);
- $y\_root \leftarrow$ `find`(y);
- **if** $x\_root = y\_root$ **then**
  - **return**;
- $first\_tree\_height \leftarrow rank[x\_root]$;
- $second\_tree\_height \leftarrow rank[y\_root]$;
- **if** $first\_tree\_height > second\_tree\_height$ **then**
  - $parent[y\_root] \leftarrow x\_root$;
- **else**
  - $parent[x\_root] \leftarrow y\_root$;
  - **if** $first\_tree\_height = second\_tree\_height$ **then**
    - $rank[y\_root] \leftarrow rank[y\_root] + 1$;

Pseudokod 3.6: Algorytm Prima.

Input: Spójny graf nieskierowany $G = (V, E)$ z wagami krawędzi w_e.
Output: Minimalne drzewo rozpinające dla grafu G.

```
1  forall u ∈ V do
2      cost(u) ← ∞;
3      prev(u) ← nil;
4  Wybrać dowolny wierzchołek początkowy u_0;
5  cost(u_0) ← 0;
6  H ← makeheap(V) (jako klucz jest użyty koszt wierzchołków);
7  while H ≠ ∅ do
8      v ← deletemin(H);
9      foreach {v, z} ∈ E do
10         if cost(z) > w(v, z) then
11             cost(z) ← w(v, z);
12             prev(z) ← v;
13             decreasekey(H, z);
```

Pseudokod 3.7: Algorytm Kruskala.

Input: Spójny graf nieskierowany $G = (V, E)$ z wagami krawędzi w_e.
Output: Minimalne drzewo rozpinające dla grafu G.

```
1  forall u ∈ V do
2      makeset(u);
3  X ← {};
4  Posortować krawędzie w E (rosnąco względem wag krawędzi);
5  forall krawędź {u, v} ∈ E według rosnącej wagi do
6      if find(u) ≠ find(v) then
7          union (u, v);
8          Dodaj krawędź {u, v} do X;
```

W grafach skierowanych można wyróżnić źródłowe silnie spójne składowe (z których krawędzie wychodzą, ale do nich nie wchodzą) oraz ujściowe silnie spójne składowe (do których krawędzie wchodzą, ale z nich nie wychodzą). Uruchamiając procedurę `explore` na ujściowej silnie spójnej składowej, otrzymamy całą tą składową. Zatem algorytm szukania silnie spójnych składowych mógłby wyglądać następująco:

1. Znaleźć wierzchołek u leżący w ujściowej silnie spójnej składowej grafu.
2. Wykonać na znalezionym wierzchołku procedurę `explore`, w celu znalezienia całej silnie spójnej składowej (na mocy twierdzenia 3.1).
3. Usunąć silnie spójną składową, w której znajduje się wierzchołek u, z grafu.
4. Kontynuować proces, dopóki nie zostanie znaleziona ostatnia ujściowa silnie spójna składowa grafu.

Jednak, aby zastosować to podejście, należy rozwiązać dwa problemy:

1. W jaki sposób znaleźć wierzchołek, który na pewno leży w ujściowej silnie spójnej składowej grafu ?
2. W jaki sposób szukać następnych ujściowych silnie spójnych składowych w grafie ?

W rozwiązaniu pierwszego problemu, można wykorzystać twierdzenie 3.2. Jednak, to twierdzenie mówi o źródłowych składowych, a nie ujściowych. Ten problem można rozwiązać poprzez skonstruowanie grafu

odwrotnego G^R (w którym składowa ujściowa stanie się składową źródłową) i wykonanie na nim algorytmu DFS z wyznaczaniem numeru *post* wierzchołka. Zgodnie z twierdzeniem 3.2, wierzchołek z największym numerem *post*, będzie leżał w ujściowej składowej w grafie G. Wierzchołkiem w następnej ujściowej składowej będzie wierzchołek z drugim w kolejności numerem *post* (znowu na mocy twierdzenia 3.2), po usunięciu bieżącej składowej z grafu. Zatem, aby znaleźć ujściowe składowe, należy przeglądać wierzchołki w kolejności malejących numerów *post* (należy pamiętać, że po znalezieniu ujściowej składowej, usuwa się ją z grafu, razem z jej wszystkimi wierzchołkami). Po rozwiązaniu tych dwóch problemów można napisać następujący pseudokod algorytmu:

1. Wyznaczyć graf odwrotny G^R (linijka 1 w 3.8).
2. Wykonać algorytm DFS na grafie odwrotnym G^R (szukanie kolejnych składowych ujściowych) (linijka 2 w 3.8).
3. Posortować wierzchołki względem wyznaczonych numerów *post* (malejąco - linijka 3 w 3.8).
4. Wykonać DFS na pierwotnym grafie G (po wykonaniu DFS na składowej ujściowej, otrzyma się całą tą składową (twierdzenie 3.1) - linijka 4 w 3.8).

Złożoność obliczeniowa tego algorytmu wynosi $\mathcal{O}(|V|+|E|)$ (czyli jest liniowa), ponieważ wyznaczanie grafu odwrotnego G^R, DFS oraz sortowanie wierzchołków względem numerów *post* ma również złożoność $\mathcal{O}(|V|+|E|)$ (do sortowania wierzchołków można użyć sortowania przez zliczanie).

Pseudokod 3.8: Algorytm szukania silnie spójnych składowych w grafie skierowanym.

Input: Graf skierowany $G = (V, E)$.
Output: Silnie spójne składowe grafu G.
1 Wyznacz graf odwrotny do G (G^R);
2 Wykonaj algorytm DFS na grafie G^R i wyznacz numer *post* dla każdego wierzchołka;
3 Posortuj wierzchołki względem wyznaczonego numeru *post*;
4 Wykonaj algorytm DFS na pierwotnym grafie G, przeglądając wierzchołki względem malejącego numeru *post*;

3.8 Algorytm Forda-Fulkersona i algorytm Edmondsa-Karpa

Algorytm Forda-Fulkersona oraz algorytm Edmondsa-Karpa (pseudokod 3.9) są rozważane w tym podrozdziale razem, ponieważ drugi algorytm jest modyfikacją pierwszego algorytmu. Algorytm Forda-Fulkersona jest pewnym szablonem rozwiązania problemu szukania maksymalnego przepływu, gdzie można stosować różne algorytmy szukania $s-t$ ścieżki (czyli ścieżki od źródła do ujścia) w danej sieci przepływowej. Algorytm Edmondsa-Karpa tak naprawdę jest algorytmem Forda-Fulkersona, gdzie znajdowane $s-t$ ścieżki są najkrótsze w sensie liczby krawędzi (poprzez zastosowanie algorytmu BFS).

W celu zrozumienia działania algorytmu Forda-Fulkersona, należy znać pojęcie grafu residualnego. Graf residualny jest indukowany przez sieć przepływową i składa się z krawędzi z sieci przepływowej (z wartością przepływu będącą różnicą maksymalnego przepływu na krawędzi i aktualnego przepływu na krawędzi) oraz krawędzi do nich odwrotnych (z wartością przepływu równą aktualnemu przepływowi na danej krawędzi). Intuicyjnie, te nowe wartości przepływów dla krawędzi oznaczają ile można jeszcze "przepuścić" przez daną krawędź oraz ile przepływu można "wycofać" z krawędzi.

Algorytm Forda-Fulkersona rozpoczyna swoje działanie od ustawienia przepływu na każdej krawędzi na 0 (linijki 1-2 w 3.9). Następnie rozpatrywane są wszystkie ścieżki ze źródła do ujścia w grafie residualnym. Dla każdej takiej ścieżki, wyznaczany jest najmniejszy dopuszczalny przepływ na krawędzi znajdującej się na bieżącej $s-t$ ścieżce - c_f. Następnie dla każdej krawędzi na znalezionej ścieżce, jeśli krawędź jest w sieci przepływowej, to przepływ na tej krawędzi jest zwiększany o c_f. Jeśli danej krawędzi (u, v) nie ma w sieci przepływowej, to oznacza, że jest krawędź (v, u) i na niej przepływ zostanie zmniejszony o c_f. Ta procedura jest powtarzana, dopóki istnieje $s-t$ ścieżka w grafie residualnym (linijki 3-9 w 3.9).

Idea stojąca za tym algorytmem jest prosta: powiększać przepływ, dopóki można "przesłać" ileś jednostek przepływu ze źródła s do ujścia t.

Złożoność obliczeniowa szukania ścieżki w linijce 3 w 3.9 wynosi $\mathcal{O}(|E|)$ oraz pętla `while` wykona się co najwyżej $\mathcal{O}(f_{max})$ razy (ponieważ w każdej iteracji, przepływ zwiększa się o jedną jednostkę), zatem złożoność obliczeniowa algorytmu Forda-Fulkersona wynosi $\mathcal{O}(f_{max}|E|)$ (gdzie f_{max} oznacza wartość maksymalnego przepływu w danej sieci przepływowej). Złożoność obliczeniowa algorytmu Edmondsa-Karpa wynosi $\mathcal{O}(|V||E|^2)$.

Pseudokod 3.9: Algorytm Forda-Fulkersona i algorytm Edmondsa-Karpa.

Input: Sieć przepływowa $G = (V, E)$, źródło s i ujście t.
Output: Maksymalny przepływ w sieci $G = (V, E)$.

```
1 forall (u, v) ∈ E do
2     (u, v).f ← 0
3 while istnieje ścieżka p ze źródła s do ujścia t w grafie residualnym G_f do
4     c_f(p) ← min{c_f(u, v) : (u, v) ∈ p};
5     foreach krawędź (u, v) ∈ p do
6         if (u, v) ∈ E then
7             (u, v).f ← (u, v).f + c_f(p)
8         else
9             (v, u).f ← (v, u).f − c_f(p)
```

Rozdział 4

Implementacja systemu, instrukcja obsługi aplikacji oraz czynności instalacyjne i wdrożeniowe

W tym rozdziale jest zawarty krótki opis użytych technologii i języków programowania, instrukcja obsługi aplikacji dla użytkownika oraz opis sposobu instalacji i wdrożenia aplikacji.

4.1 Opis użytych technologii

Do zaimplementowania omawianej aplikacji użyto języka Python3 oraz biblioteki Qt, a konkretnie jej wersji dla Pythona - PyQt5. Szczegółowy opis użytych technologii można znaleźć w [3], [4] oraz [5]. Przy tworzeniu aplikacji wykorzystano mechanizm slotów i sygnałów, który jest charakterystycznym elementem biblioteki Qt. W celu zapewnienia płynności działania aplikacji i animacji, wykorzystano programowanie współbieżne (wątki, muteksy - skorzystano z API udostępnionego przez bilbiotekę PyQt5).

4.2 Instrukcja obsługi aplikacji

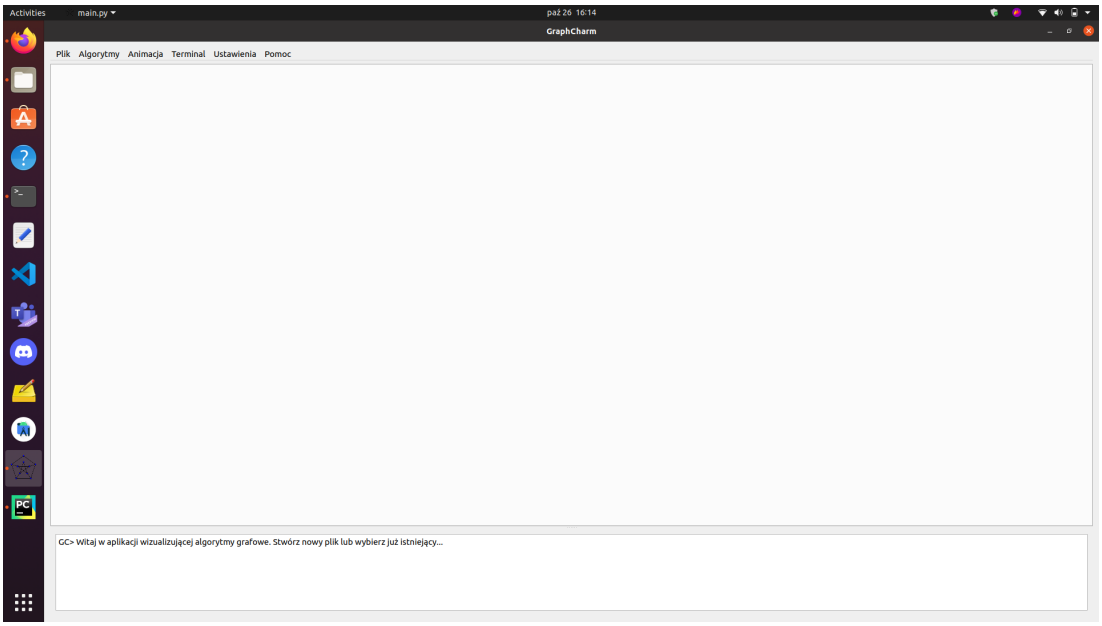
1. Po uruchomieniu programu przez użytkownika, pojawia się okno główne aplikacji, w którym użytkownik może wybrać żądaną przez niego akcję (rysunek 4.1).

2. Jeśli użytkownik chce utworzyć nowy plik z rysunkiem grafu, to musi wybrać opcję `Nowy` w menu `Plik` (rysunek 4.2).

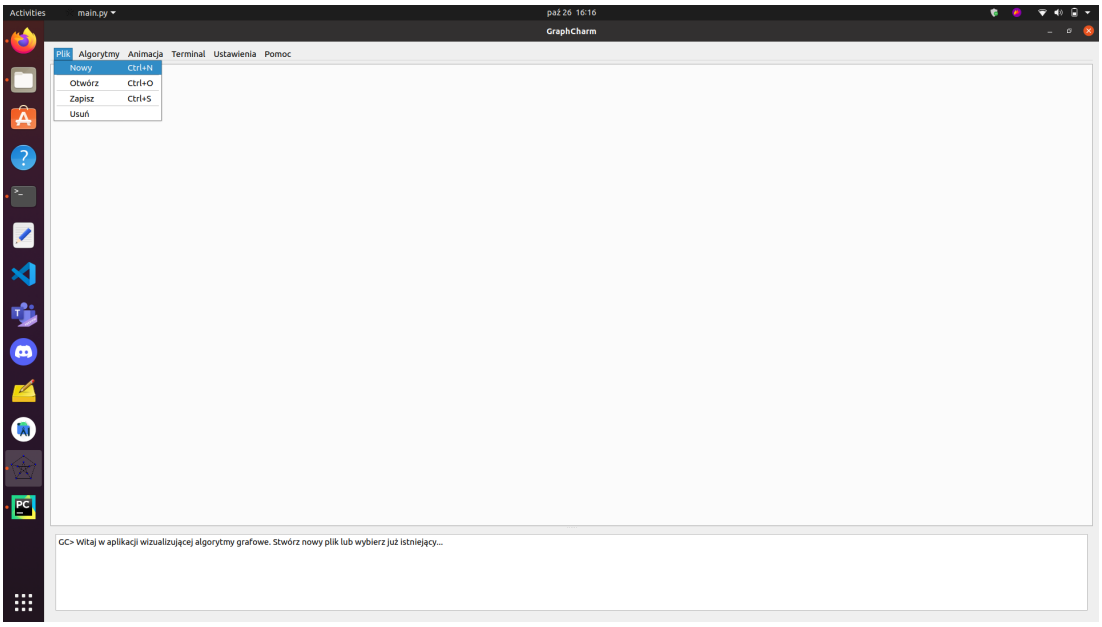
3. Po wybraniu tej opcji, otworzy się okienko dialogowe, w którym użytkownik będzie musiał podać nazwę nowego pliku oraz rodzaj grafu, który będzie się w nim znajdował (skierowany albo nieskierowany - domyślnie jest tworzony graf nieskierowany). Omawiane operacje są przedstawione na rysunkach 4.3 oraz 4.4.

4. Po wciśnięciu przycisku `Ok` (jeśli użytkownik wciśnie przycisk `Cancel`, to zostanie przerwany proces tworzenia nowego pliku), otworzy się zakładka, w której jest: miejsce do rysowania grafu, terminal aplikacji, miejsce na wyświetlanie pseudokodu oraz pasek narzędzi. Użytkownik może dodać nowy wierzchołek do grafu, poprzez wybranie opcji `Dodaj wierzchołek` w pasku narzędzi oraz naciśnięcie lewym przyciskiem myszy na "płótno" (rysunek 4.5).

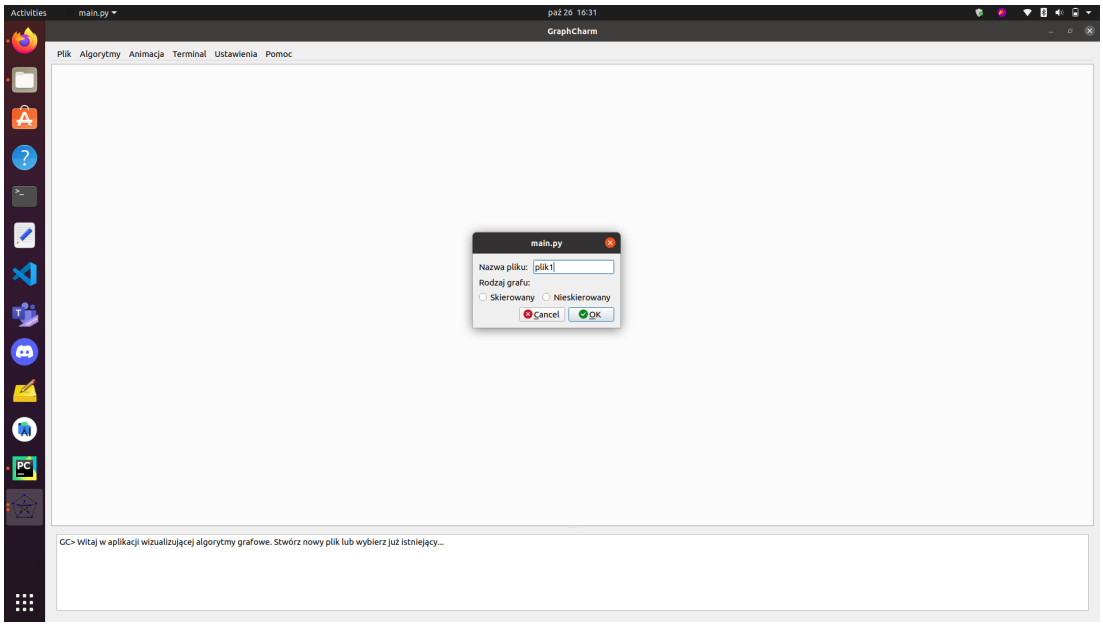
5. Jeśli użytkownik chce dodać nową krawędź do grafu, to powinien wybrać opcję `Dodaj krawędź` w pasku narzędzi, a następnie kliknąć lewym przyciskiem myszy na wybrany wierzchołek, przeciągnąć kursor do wybranego przez użytkownika wierzchołka docelowego i puścić lewy przycisk myszy. Jeśli użytkownik puści przycisk poza jakimkolwiek wierzchołkiem, wtedy krawędź nie zostanie narysowana



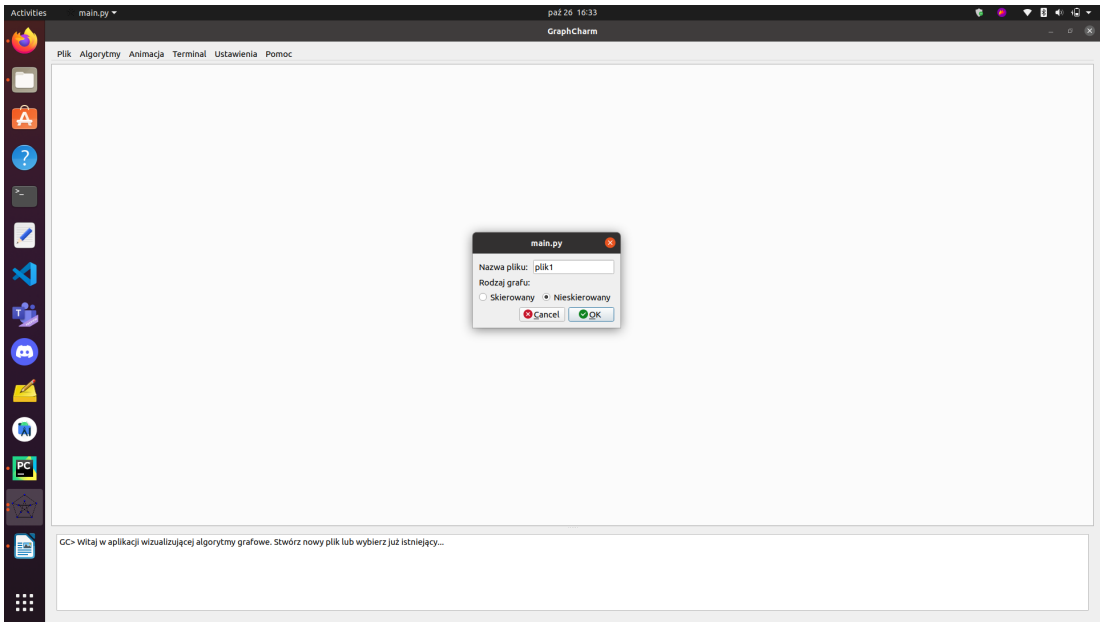
Rysunek 4.1: Okno główne aplikacji.



Rysunek 4.2: Wybór opcji tworzenia nowego pliku.



Rysunek 4.3: Podanie nazwy nowego pliku w oknie dialogowym.



Rysunek 4.4: Wybór rodzaju tworzonego grafu.

(ma to na celu zapewnienie, że krawędź zawsze będzie łączyła jakieś wierzchołki). Efekt dodania nowej krawędzi do grafu jest przedstawiony na rysunku 4.6.

6. Po dodaniu wierzchołków i krawędzi (czyli narysowaniu grafu), użytkownik może zmieniać położenie wierzchołków grafu za pomocą opcji `Tryb normalny` (dostępnej w pasku narzędzi). Po wybraniu tej opcji, użytkownik naciska lewym przyciskiem myszy na wybrany przez niego wierzchołek i przesuwa kursor na wybrane przez niego miejsce. Wybrany wierzchołek przesuwa się razem z kursorem i jeśli użytkownik puści przycisk myszy w danym miejscu, to tam zostanie ustalona nowa pozycja dla wybranego przez niego wierzchołka.

7. Jeśli użytkownik chce zmienić pozycję wyświetlania danych związanych z wierzchołkami, to może to zrobić poprzez wybranie opcji `Wyświetl dane wierzchołków` w pasku narzędzi. Po wybraniu tej opcji obok wierzchołków wyświetlą się przykładowe dane związane z wierzchołkami. Następnie należy kliknąć za pomocą lewego przycisku myszy na wybraną liczbę (oznaczającą przykładowe dane dla danego wierzchołka), przesunąć na żądane przez użytkownika miejsce i puścić lewy przycisk myszy - wówczas zostanie ustalona nowa pozycja wyświetlania danych związanych z wybranym wierzchołkiem (rysunek 4.7). Celem tej funkcjonalności jest umożliwienie użytkownikowi wpływania na jakość animacji.

8. Jeśli użytkownik wciśnie prawy przycisk myszy, to pojawi się menu kontekstowe, w którym będą następujące opcje do wyboru:

   Usuń wierzchołek - usuwa wybrany wierzchołek i wszystkie krawędzie z nim incydentne.

   Usuń krawędź - usuwa wybraną krawędź.

   Dodaj wagę - dodaje wagę do wybranej krawędzi (pojawia się okno dialogowe, gdzie użytkownik podaje wagę).

   Dodaj wagi automatycznie - dodaje losowe wagi do wszystkich krawędzi w grafie.

   Anuluj - wyłącza menu kontekstowe.

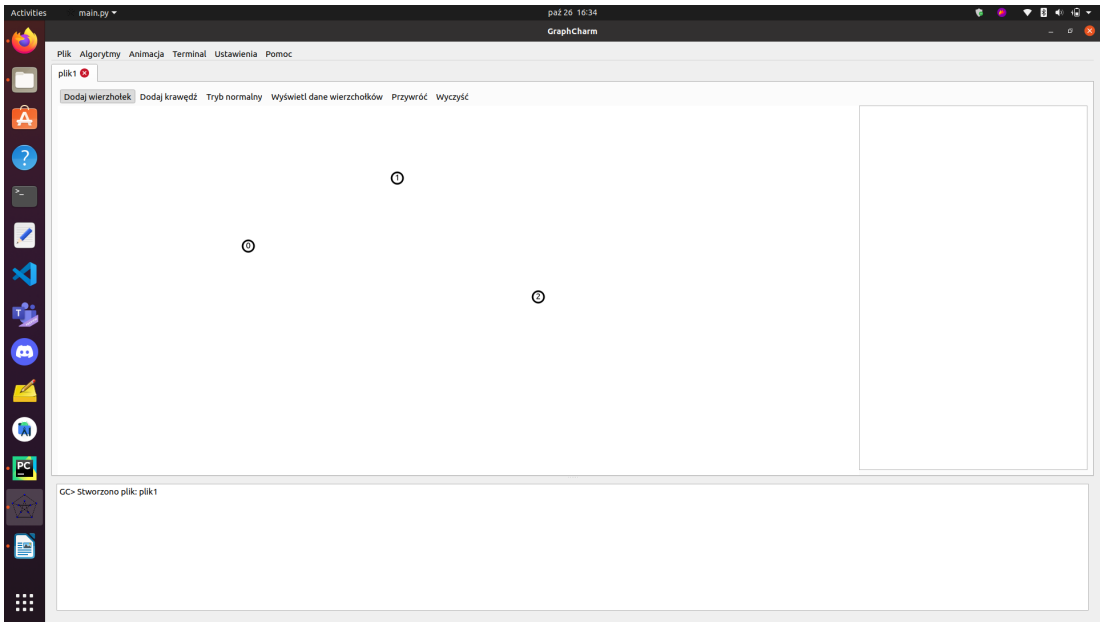
   Jeśli kursor myszy znajduje się nad krawędzią albo wierzchołkiem, to po wciśnięciu prawego przycisku, kolor krawędzi albo wierzchołka zmieni się na fioletowy (oznaczenie, że został wybrany) i pojawi się menu kontekstowe. Jeśli użytkownik wybrał wierzchołek i wybierze opcję ściśle związaną z krawędzią, to aplikacja nie wykona żadnej akcji (to działa tak samo, gdy użytkownik wybrał krawędź i opcję ściśle związaną z wierzchołkiem). Po wybraniu opcji `Anuluj`, menu kontekstowe zostaje wyłączone, a wybrany obiekt zmienia kolor z fioletowego na czarny (rysunek 4.8).

9. W trybie normalnym, użytkownik ma możliwość zmieniania pozycji wyświetlania wagi krawędzi. Odbywa się to w taki sam sposób jak zmiana pozycji wyświetlania danych związanych z wierzchołkami.

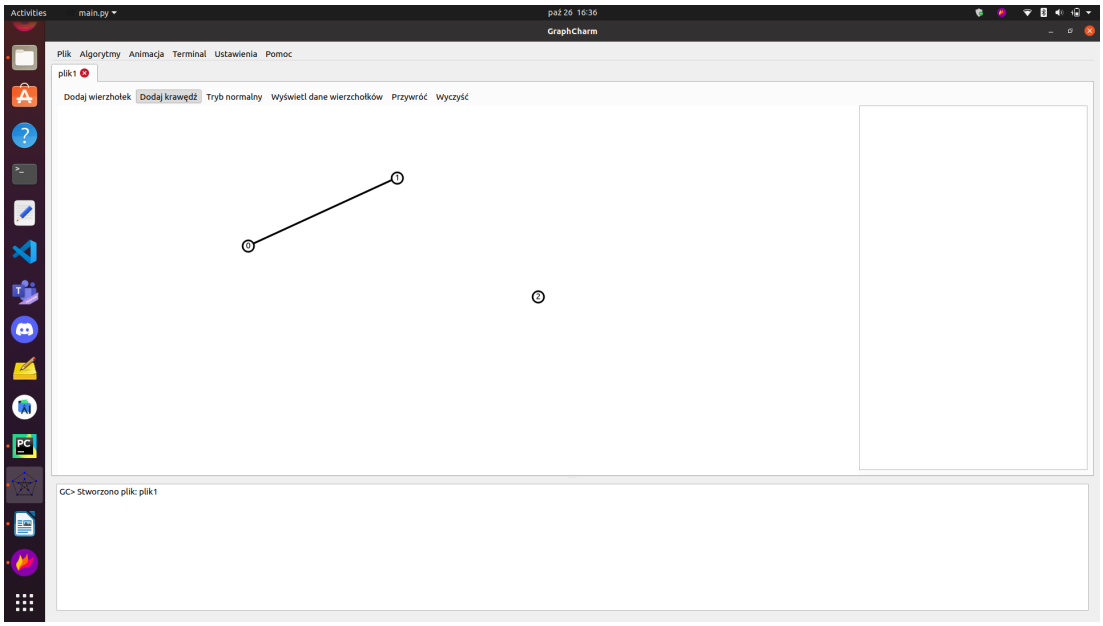
10. Jeśli użytkownik już narysował graf, to może uruchomić animację przedstawiającą wykonanie wybranego przez niego algorytmu na aktualnym grafie - musi wybrać nazwę żądanego algorytmu z menu `Algorytmy`. Po wybraniu algorytmu aplikacja sprawdza, czy dany graf spełnia założenia formalne istotne dla algorytmu. Jeśli tak, to uruchamiana jest animacja. W przeciwnym przypadku, wyświetlane są odpowiednie komunikaty błędów w terminalu aplikacji. Po zakończonej animacji, jest wyświetlany efekt końcowy działania algorytmu (rysunki 4.9, 4.10 oraz 4.11). Podczas animowania algorytmu (oraz po animacji) nie ma możliwości manipulowania rysunkiem grafu.

11. Podczas animowania algorytmu w danej zakładce nie można zapisywać rysunku grafu do pliku, usuwać plików z rysunkami grafów, uruchamiać innych animacji, usuwać komunikatów generowanych przez aplikację oraz zmieniać rozmiaru wierzchołków grafu - opcje odpowiadające tym czynnościom nie działają. Te ograniczenie dotyczy tylko zakładek, w których jest uruchomiona animacja - w przypadku zakładek, w których nie jest uruchomiona żadna animacja, wyżej wymienione opcje działają normalnie.

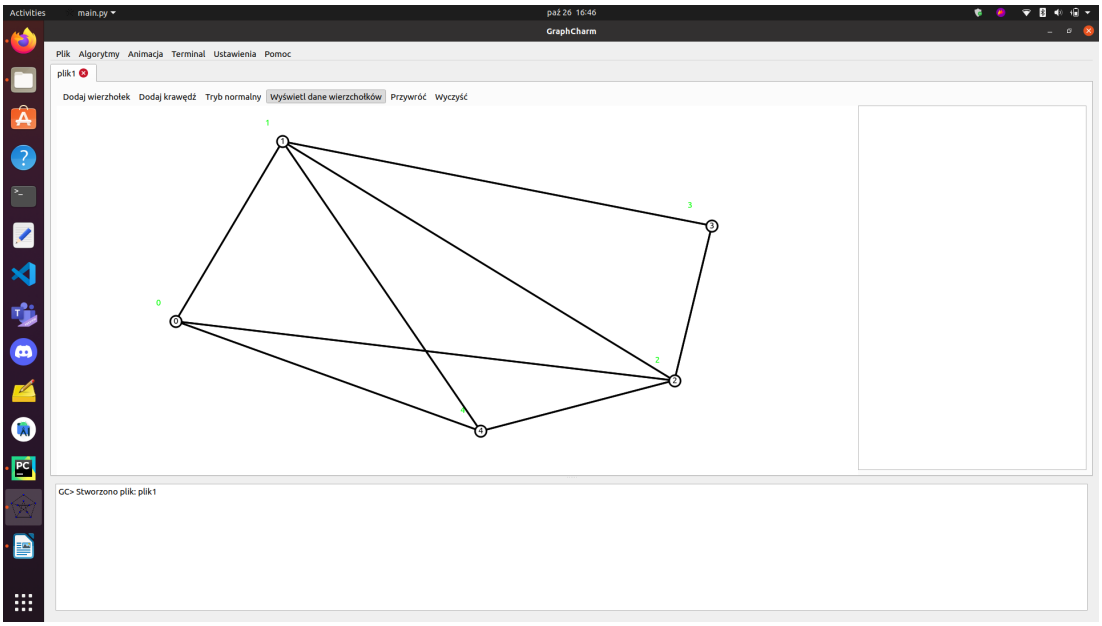
12. Po zakończonej animacji, można przywrócić pierwotną postać grafu, poprzez wybranie opcji `Przywróć` w pasku narzędzi (wówczas manipulowanie rysunkiem grafu staje się możliwe).



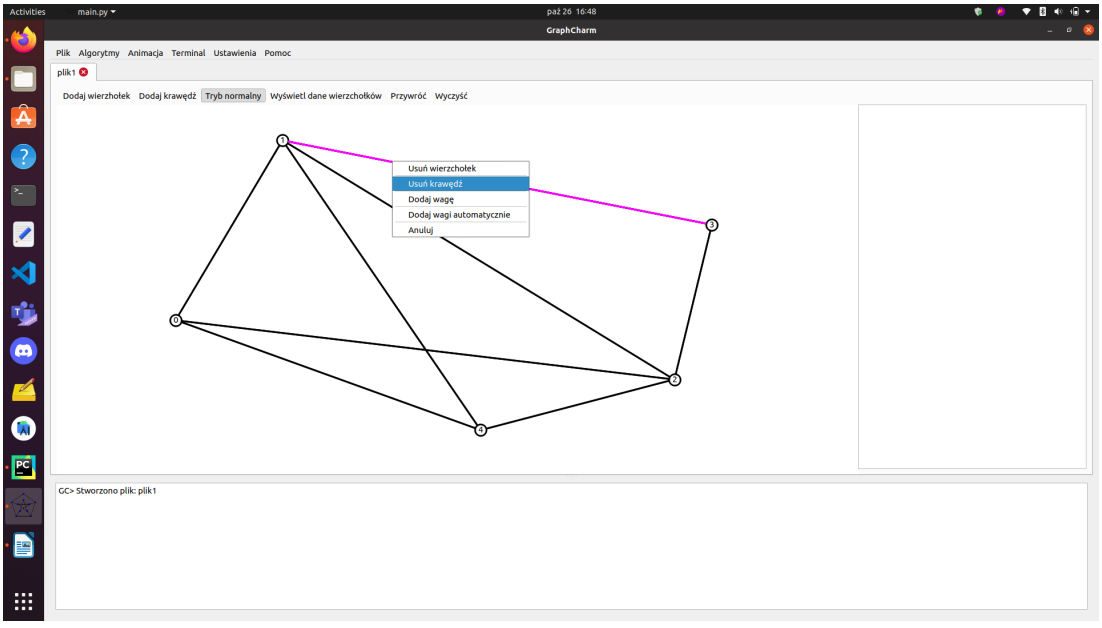
Rysunek 4.5: Dodawanie nowego wierzchołka do grafu.



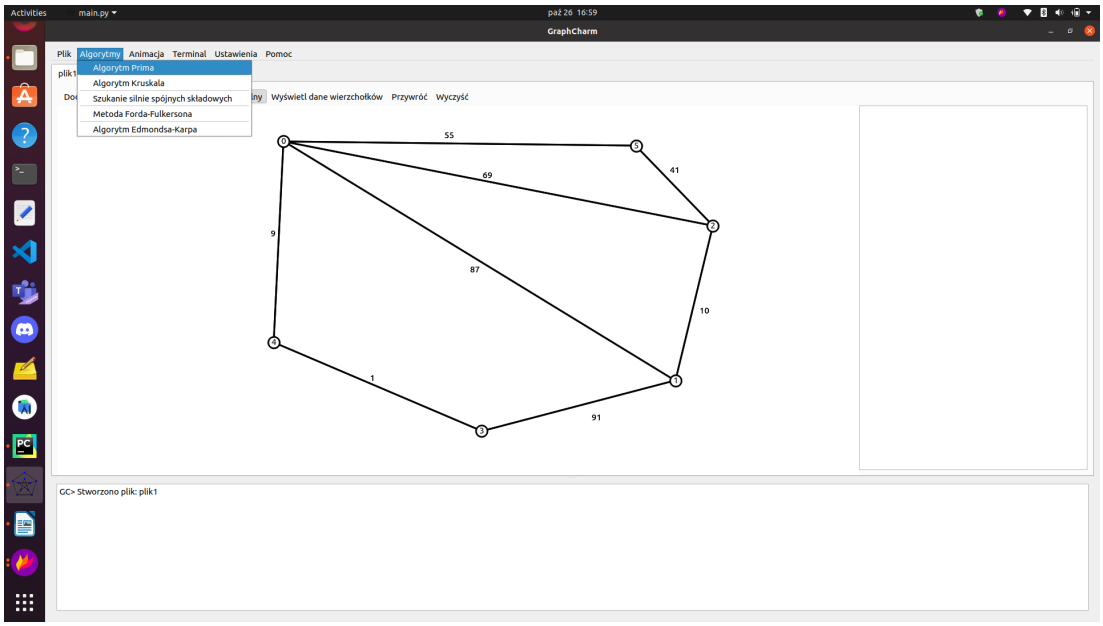
Rysunek 4.6: Dodawanie nowej krawędzi do grafu.



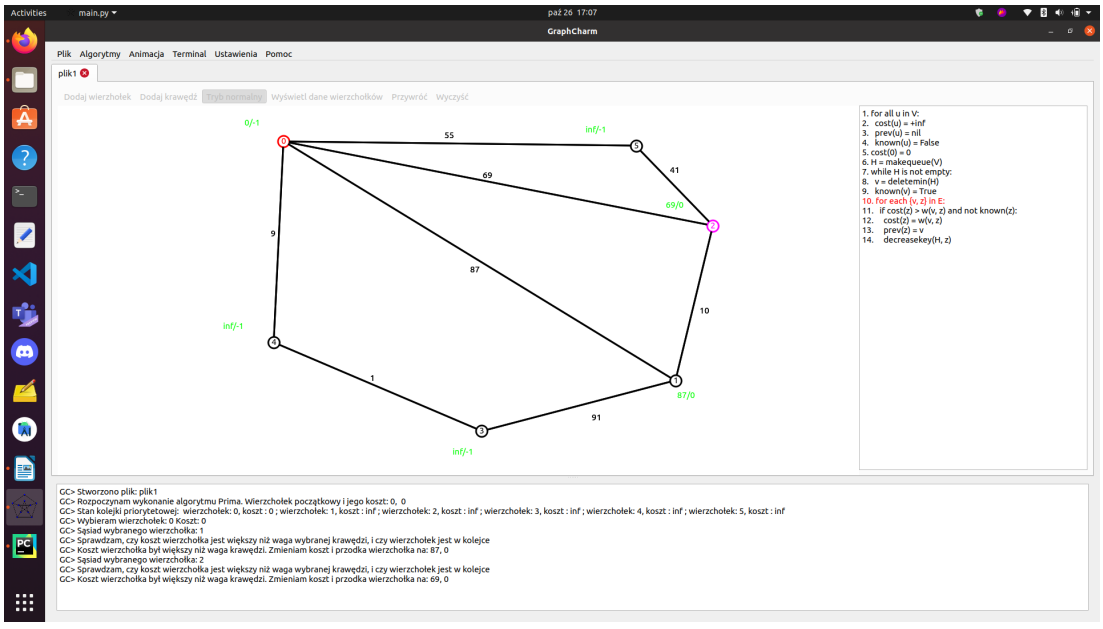
Rysunek 4.7: Działanie opcji wyświetlania danych związanych z wierzchołkami.



Rysunek 4.8: Użytkownik ma możliwość uruchomienia menu kontekstowego z różnymi opcjami.



Rysunek 4.9: Uruchamianie animacji wybranego algorytmu przez użytkownika.



Rysunek 4.10: Animowanie wybranego przez użytkownika algorytmu.

13. Za pomocą opcji `Zapisz` dostępnej w menu `Plik`, użytkownik może zapisać rysunek grafu. Rysunki grafów są zapisywane w katalogu `graph_visualizer_projects`, który znajduje się (jest tworzony) wewnątrz katalogu głównego aplikacji (tzn. wewnątrz katalogu o nazwie `PracaDyplomowa`).

14. Za pomocą opcji `Wyczyść terminal` dostępnej w menu `Terminal`, można usunąć komunikaty z terminala aplikacji.

15. W aplikacji istnieje możliwość wyłączenia wybranej, działającej animacji albo jej wyłączenia i wyświetlenia efektu końcowego działania algorytmu - należy wybrać opcję (odpowiednio) :

    (a) `Zakończ`.

    (b) `Zakończ i wyświetl efekt końcowy`.

    Obie opcje są dostępne w menu `Animacja`.

16. Jeśli dany graf nie spełnia założeń formalnych istotnych dla wybranego przez użytkownika algorytmu, to wyświetlane są odpowiednie komunikaty błędów.

17. Obsługa aplikacji dla grafu skierowanego jest niemal identyczna jak dla grafu nieskierowanego. Różnica polega na tym, że w grafie skierowanym można również przesuwać krawędzie (przy założeniu, że są krawędziami między tymi samymi wierzchołkami i są przeciwnie skierowane) - ten proces odbywa się tak samo jak zmiana pozycji wyświetlania danych związanych z wierzchołkami. Przy zmianie pozycji wierzchołków krawędzie są ponownie złączane, w celu prawidłowego wyliczenia współrzędnych ich początku i końca (rysunki 4.12 oraz 4.13).

18. Istnieje możliwość otworzenia wcześniej przygotowanych, przykładowych rysunków grafów, na których można uruchamiać wybrane animacje. Wystarczy wybrać opcję `Otwórz` znajdującą się w menu `Plik`.

4.3 Czynności instalacyjne i wdrożeniowe

Pakiet instalacyjny jest umieszczony na płycie CD dołączonej do części pisemnej pracy dyplomowej i składa się z katalogu `PracaDyplomowa`, w którym są zawarte wszystkie pliki i katalogi tworzące aplikację. Program powstawał i był testowany pod systemem operacyjnym Ubuntu (wersja 20.04). Aplikację należy uruchomić na jednej z nowszych wersji ($\geqslant$ 18.04) systemu operacyjnego Ubuntu (program można również uruchamiać na innych dystrybucjach systemu operacyjnego Linux, jednak nie ma gwarancji, że aplikacja będzie działać). Przed uruchomieniem aplikacji należy upewnić się, że interpreter języka Python jest zainstalowany na używanej platformie (najlepsza wersja $\geqslant$ Python 3.8).

W celu skonfigurowania i uruchomienia aplikacji, należy wykonać następujące czynności:

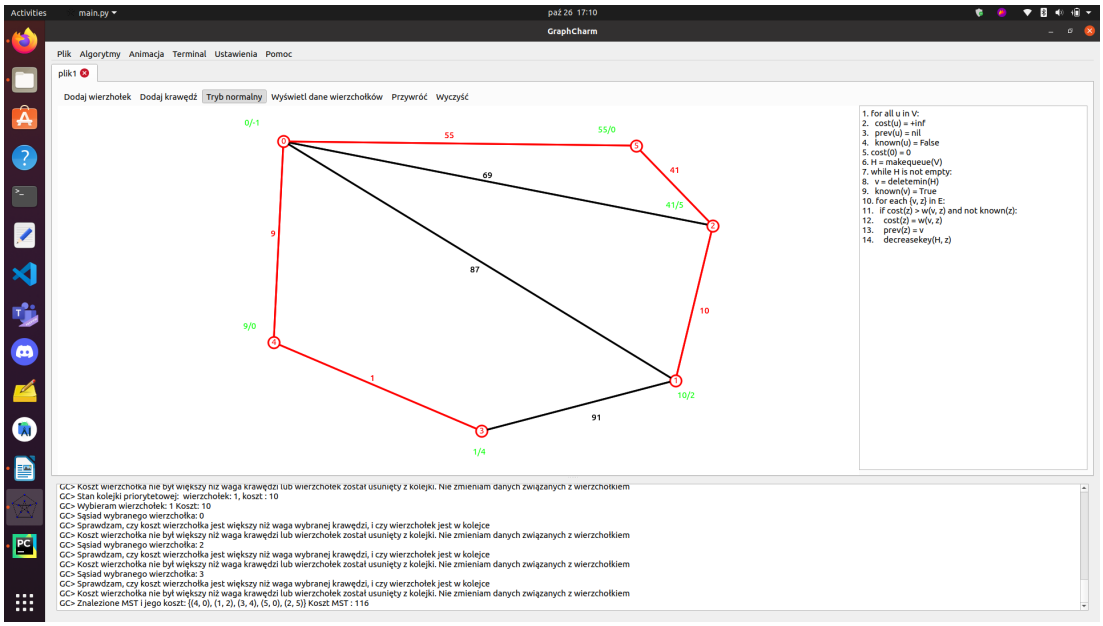
1. Zainstalować bibliotekę PyQt5 np. za pomocą komendy (w terminalu): `pip install PyQt5`.

2. Zainstalowć bibliotekę PyQt5-stubs (można zainstalować dowolną, nowszą wersję tej biblioteki) np. za pomocą komendy (w terminalu): `pip install PyQt5-stubs==5.15.2.0`.

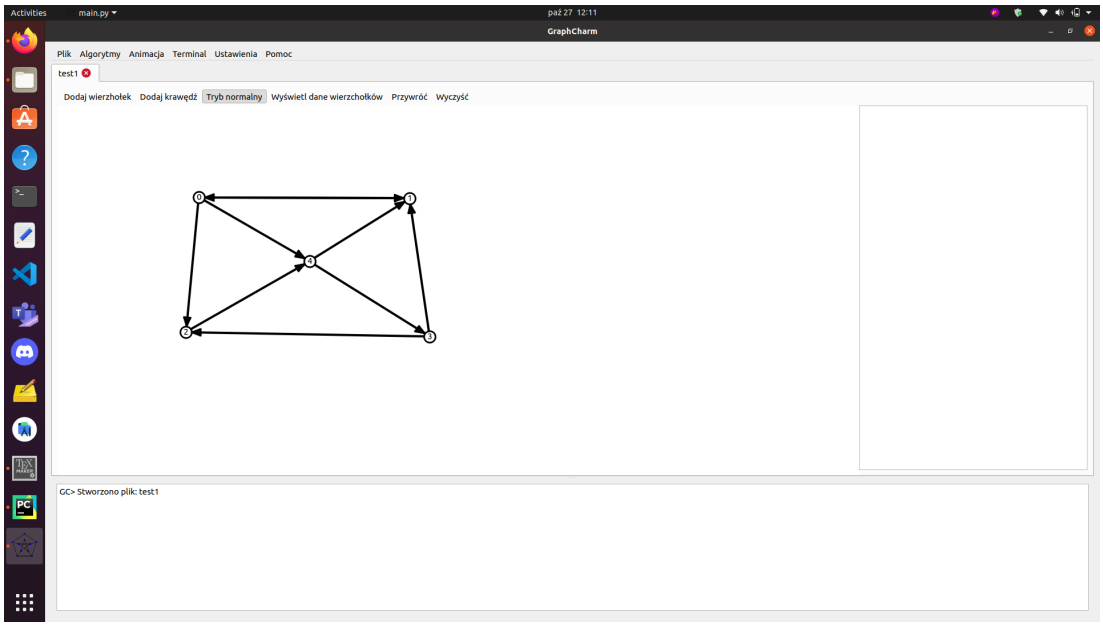
3. Po zainstalowaniu potrzebnych bibliotek, przejść do katalogu `PracaDyplomowa` i uruchomić skrytp `main.py`. Te czynności można wykonać za pomocą następujących komend (w terminalu): `cd PracaDyplomowa` oraz `python3 main.py`.

4. Po uruchomieniu skryptu `main.py`, na ekranie powinno pojawić się okno główne aplikacji.

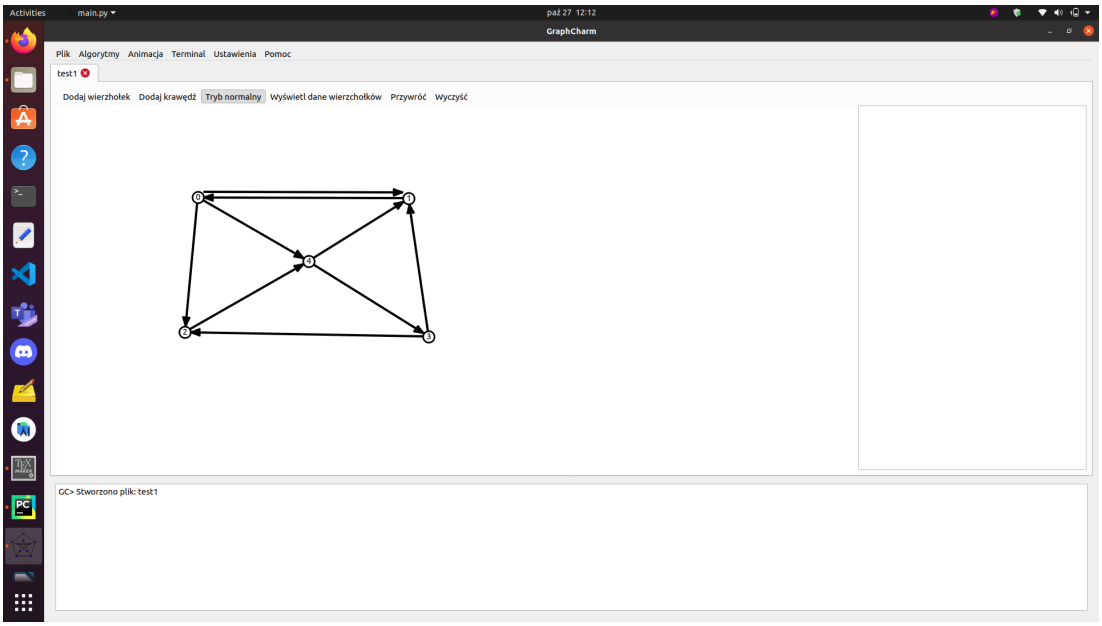
Po wykonaniu powyższych czynności, aplikacja działa i oczekuje na akcje typowego użytkownika.



Rysunek 4.11: Wyświetlenie efektu końcowego działania algorytmu na grafie.



Rysunek 4.12: Przykładowy graf skierowany.



Rysunek 4.13: Podwójne krawędzie skierowane można przesuwać. W przypadku pojedynczych krawędzi skierowanych taka operacja nie jest możliwa.

Podsumowanie

W części pisemnej pracy dyplomowej zamieszczono projekt aplikacji wykonany za pomocą diagramów UML, scenariusze przypadków użycia, opis wykorzystywanych w aplikacji algorytmów oraz instrukcję obsługi aplikacji dla użytkownika.

Dostarczona aplikacja spełnia wszystkie wymagania funkcjonalne oraz pozafunkcjonalne, które zostały dla niej określone.

W omawianej aplikacji wykorzystano wiele nietrywialnych algorytmów grafowych (wizualizowanych oraz służących do sprawdzania poprawności grafów) oraz algorytmów bazujących na geometrii analitycznej (służących do rysowania określonych obiektów lub wspomagania procesu rysowania). W celu zapewnienia płynności działania całej aplikacji po uruchomieniu animacji, wykorzystano programowanie współbieżne (wątki, muteksy).

Omawiana aplikacja może być rozwijana na wiele sposobów:

1. Dodanie wizualizacji innych algorytmów grafowych (np. A*, algorytm Dynica, algorytm Floyda-Warshalla).
2. Umożliwienie użytkownikowi przewijania animacji w przód i w tył oraz sterowania jej szybkością.
3. Umożliwienie użytkownikowi rysowania dowolnie dużych grafów (miejsce do rysowania nie będzie ograniczone), wraz z możliwością powiększania/zmniejszania narysowanego grafu (skalowanie).
4. Stworzenie własnego języka do opisu operacji na grafach, który umożliwi wizualizowanie algorytmów napisanych przez użytkownika.
5. Umożliwienie użytkownikowi tworzenia grafu za pomocą wczytywania listy sąsiedztwa (np. z pliku).

Bibliografia

[1] Strona: https://visualgo.net/en. Data wstępu: 12.11.21, godzina: 12:25.

[2] Strona: https://www.cs.usfca.edu/~galles/visualization/Algorithms.html. Data wstępu: 12.11.21, godzina: 12:26.

[3] Python 3.9.7 documentation. Strona: https://docs.python.org/3/.

[4] Qt documentation. Strona: https://doc.qt.io/.

[5] Qt for python. Strona: https://doc.qt.io/qtforpython/.

[6] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein. *Wprowadzenie do algorytmów*. Wydawnictwo Naukowe PWN, Warszawa, 2012.

[7] S. Dasgupta, C. Papadimitriou, U. Vazirani. *Algorytmy*. Wydawnictwo Naukowe PWN, Warszawa, 2010.

Załącznik A

Zawartość płyty CD

Na płycie CD, dołączonej do pracy dyplomowej, znajdują się kody źródłowe aplikacji (plik zip o nazwie `W04N_250070_2021_aplikacja_dyplomowa_kody_źródłowe.zip`), część pisemna pracy dyplomowej (plik pdf o nazwie `W04N_250070_2021_praca inżynierska.pdf`) oraz dokumentacja aplikacji (plik zip o nazwie `W04N_250070_2021_aplikacja_dyplomowa_dokumentacja.zip`).

W pliku `W04N_250070_2021_aplikacja_dyplomowa_kody_źródłowe.zip` znajduje się katalog o nazwie `PracaDyplomowa`, który jest głównym katalogiem całego projektu i w nim znajdują się wszystkie kody źródłowe aplikacji.

W pliku `W04N_250070_2021_aplikacja_dyplomowa_dokumentacja.zip` znajduje się katalog o nazwie `dokumentacja`, w którym są umieszczone pliki html, będące dokumentacją poszczególnych modułów używanych w aplikacji. Nazwy plików z dokumentacją odpowiadają (tzn. są identyczne) nazwom modułów aplikacji, które dokumentują. Dokumentację wybranego modułu można otworzyć w przeglądarce - wtedy będzie widoczny spis i opis klas oraz metod (obecnych w module) wraz z ich sygnaturami. Ponadto, istnieje możliwość podglądu kodu źródłowego dowolnej metody, która znajduje się w wybranym module.