Algorytm Dijkstry

Dokumentacja projektu zaliczeniowego dla przedmiotu Integracja Sieci Komputerowych.

Przemysław Kleszcz

Styczeń 2018

Spis treści

1. Wprowadzenie

W zakresie projektu zaliczeniowego zrealizowana została aplikacja typu *Desktop* w technologii .NET. Wykorzystana została rama projektowa WPF (*Windows Presentation Foundation*). Wykonany program posiada graficzny interfejs użytkownika, przy pomocy którego użytkownik wprowadza i przetwarza dane oraz jest w stanie obserwować wyniki zaistniałych w czasie działania programu operacji.

Zadaniem aplikacji jest symulacja odnajdywania najkrótszej ścieżki w grafie, wykorzystując do tego celu algorytm Dijkstry. Interfejs zapewnia możliwość dodawania, usuwania, modyfikacji wierzchołków grafu i jego połączeń oraz graficzną reprezentację przebiegu poszukiwania najkrótszej drogi i obliczania bieżącego kosztu.

2. Wymagania techniczne

Program przeznaczony jest do uruchamiania na systemie Windows, wykorzystując do tego celu ramę projektową NET Framework w wersji 4.6.1.

3. Implementacja

Aplikacja składa się z jednego komponentu stworzonego w technologii WPF przy użyciu modelu *code behind.* Do obsługi wizualizacji grafów wykorzystana zostałą biblioteka GraphX.

3.1 Elementy składowe

Na kod źródłowy aplikacji składa się dwanaście plików. Są to pliki konfiguracyjne *xml*, pliki *cs* z kodem w języku C# oraz pliki *XAML* definiujące wygląd interfejsu użytkownika.

App.config – Plik konfiguracyjny aplikacji.

App.xaml – Punkt startowy każdej aplikacji WPF.

MainWindow.xaml – Główne okno aplikacji. Plik zawiera strukturę interfejsu użytkownika oraz powiązaną z nim logikę biznesową.

Dialog.xaml – Okno dialogowe umożliwiające pobranie ciągu znaków. Plik zawiera strukturę interfejsu okna dialogowego oraz powiązaną z nim logikę biznesową.

Dijkstra.cs – Plik zawiera implementację algorytmu wyznaczania najkrótszej ścieżki w grafie.

Edge.cs – Plik zawiera definicję struktury krawędzi grafu. W ramach wyznaczonej struktury wyszczególniamy identyfikator pierwszego i drugiego wierzchołka oraz koszt przejścia.

Vertex.cs – Plik zawiera definicję struktury wierzchołka grafu. W ramach wyznaczonej struktury wyszczególniamy identyfikator wierzchołka.

VertexTag.cs – Plik zawiera definicję struktury służącej do przechowywania dodatkowych informacji związanych ze strukturą wierzchołka. Np czy jego graficzna reprezentacja w interfejsie użytkownika została zaznaczona.

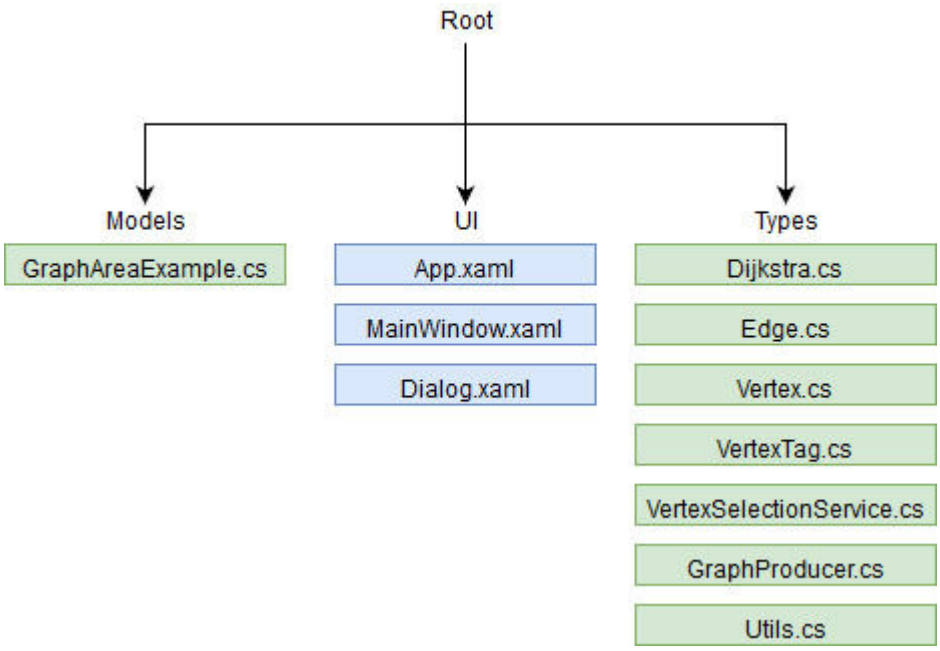
VertexSelectionService.cs – Plik zawiera logikę umożliwiającą manipulację graficzną reprezentacją wierzchołków grafu.

GraphProducer.cs – Plik zawiera logikę niezbędną do pełnej konfiguracji struktury grafu w celu odwzorowania jej graficznej reprezentacji na ekranie głównym.

GraphAreaExample.cs – Zawiera model, którego implementacja jest konieczna do prawidłowego działania biblioteki GraphX.

Utils.cs – Wszelkie niezbędne metody pomocnicze.

3.2 Struktura projektu



3.3 Opis wybranych funkcji

void GenerateGraph_OnClick – Metoda obsługi zdarzenia kliknięcia w przycisk *Wygeneruj*. Struktura grafu zostaje poddana konfiguracji, wizualna reprezentacja grafu zostaje odwzorowana na ekranie.

void RandomizeGraph_OnClick – Metoda obsługi zdarzenia kliknięcia w przycisk *Wylosuj graf. Generuje* zbiór przypadkowych wierzchołków i krawędzi po czym uzupełnia nimi powiązane tablice.

void BtnAddVertex_OnClick- Metoda obsługi zdarzenia kliknięcia w przycisk *Dodaj wierzchołek.* Pobiera od użytkownika nazwę wierzchołka. Sprawdza czy wierzchołek o podanym identyfikatorze istnieje w systemie. Jeśli tak – wyświetlone zostaje odpowiednie powiadomienie, w przeciwnym wypadku – dodaje wierzchołek do powiązanej tablicy.

void BtnAddEdge_OnClick – Metoda obsługi zdarzenia kliknięcia w przycisk *Dodaj krawędź.* Sprawdza czy dwa niezależne wierzchołki są zaznaczone na ekranie. Jeśli nie – wyświetla stosowny komunikat, w przeciwnym wypadku pobiera koszt nowej krawędzi. Podany koszt musi być liczbą całkowitą, w przeciwnym wypadku zostanie wyświetlony stosowny komunikat a funkcja zakończy pracę. Jeśli powyższe warunki zostaną spełnione – dodawana jest nowa krawędź do powiązanej tablicy.

void BtnCalculatePath_OnClick – Metoda obsługi zdarzenia kliknięcia w przycisk *Oblicz ścieżkę.* Pobiera identyfikatory dwóch wierzchołków. Jeśli choć jedna z podanych wartości jest pusta – wyświetlony zostanie stosowny komunikat i funkcja zakończy działanie. Oblicza najkrótszą ścieżkę łączącą dwa punkty i jej koszt. Zaznacza na ekranie graficzną reprezentację drogi od punktu początkowego do punktu końcowego. W przypadku niemożliwości określenia ścieżki – wyświetlany jest stosowny komunikat.

BtnNextCost_OnClick – Metoda obsługi zdarzenia kliknięcia w przycisk *Dalej*. W przypadku gdy koszt przejścia pomiędzy dowolnymi dwoma dowolnymi punktami grafu został już wyznaczony – uwzględnia następną krawędź w kolejce do sumy całkowitego kosztu oraz zaznacza ją na ekranie.

void BtnPreviousCost_OnClick – Metoda obsługi zdarzenia kliknięcia w przycisk *Wstecz.* W przypadku gdy koszt przejścia pomiędzy dwoma dowolnymi punktami grafu został już wyznaczony – pomija poprzednią krawędź w kolejce w sumie całkowitego kosztu oraz odznacza ją na ekranie.

3.4 Opis algorytmu

Aplikacja w celu wyznaczania najkrótszej ścieżki wykorzystuje algorytm Dijkstry. Algorytm ten pozwala znaleźć koszty dojścia od wierzchołka startowego v do każdego innego wierzchołka w grafie (o ile istnieje odpowiednia ścieżka). Dodatkowo wyznacza on poszczególne ścieżki.

Na samym początku w celu określenia najkrótszej ścieżki, tworzony jest słownik, którego wartości klucza są początkowymi wierzchołkami *v1* skierowanych krawędzi. Wartości tych słowników to kolejne słowniki, których wartości klucza są końcowymi wierzchołkami *v2* skierowanych krawędzi. Wartości słowników wewnętrznych do koszty przejścia od *v1* do *v2*.

Po zakończonej fazie przygotowawczej, deklarowane są dwa kolejne słowniki i lista:

- Słownik poprzedników

- Słownik kosztów

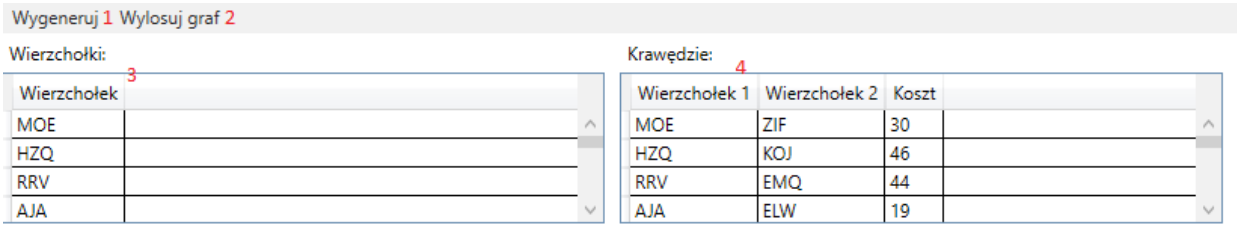
- Lista węzłów

Do listy węzłów dodawane są kolejno wszystkie wierzchołki *v1*. Do słownika kosztów dodawane są kolejno pozycje o wartości klucza *v1* oraz wartości maksymalnej. Wyjątkiem jest tutaj wierzchołek startowy, dla którego wartość słownika kosztów wynosi 0.

Teraz w pętli dopóki lista węzłów nie jest pusta, wyznaczany jest wierzchołek o najmniejszym koszcie dojścia. Wyznaczony wierzchołek usuwany jest z listy węzłów. Na podstawie początkowego słownika, pobierana jest lista sąsiadów bieżącego wierzchołka. Dla każdego ze sąsiadów obliczany jest współczynnik równy sumie wartości ze słownika kosztów dla bieżącego wierzchołka oraz wagi krawędzi (*bieżący wierzchołek → sąsiad*). Jeśli współczynnik jest mniejszy od wartości ze słownika kosztów dla sąsiada, wartość słownika zastępuje współczynnik. Jednocześnie do słownika poprzedników dodawana jest nowa pozycja, której kluczem jest wierzchołek sąsiada a wartością bieżący wierzchołek.

Jeśli wierzchołek o najmniejszym koszcie dojścia po *n* iteracji będzie tożsamy z wierzchołkiem końcowym, odczytywana jest ścieżka ze słownika poprzedników.

4. Interfejs użytkownika

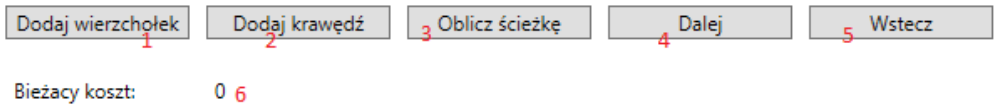


1. Przycisk *Wygeneruj*. Po kliknięciu, struktura grafu zawarta w tabelach zostaje odwzorowana na ekranie.

2. Przycisk *Wylosuj graf*. Uruchamia funkcjonalność generowania przypadkowych wierzchołków i krawędzi. Wygenerowane dane są dodawane do powiązanych tabel.

3. Tabela zawierająca dane wierzchołków. Program umożliwia bezpośrednie dodawanie/usuwanie/modyfikowanie wierzchołków w tabeli bądź dodawanie wierzchołków za pomocą funkcjonalności *Dodaj wierzchołek.*

4. Tabela zawierająca dane krawędzi. Program umożliwia bezpośrednie dodawanie/usuwanie/modyfikowanie krawędzi w tabeli bądź dodawanie krawędzi za pomocą funkcjonalności *Dodaj krawędź.*



1. Przycisk *Dodaj wierzchołek.* Dodaje nowy wierzchołek do powiązanej tabeli wierzchołków i automatycznie odwzorowuje go na ekranie.

2. Przycisk *Dodaj krawędź.* Dodaje nową krawędź do powiązanej tabeli krawędzi i automatycznie odwzorowuje ją na ekranie. W celu dodania krawędzi, dokładnie dwa wierzchołki muszą być zaznaczone poprzez dwukrotne kliknięcie.

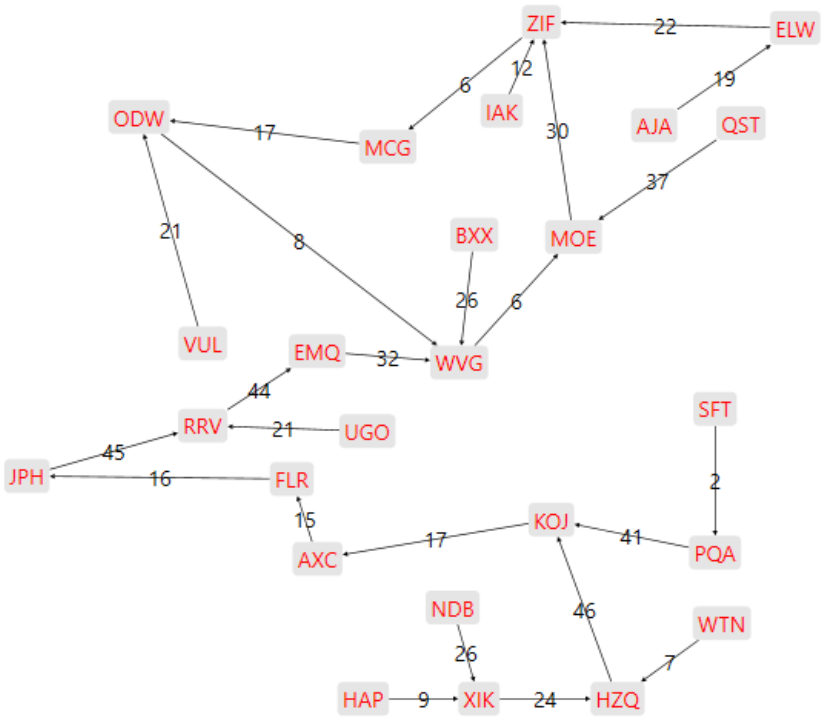
3. Przycisk *Oblicz ścieżkę*. Oblicza najkrótszą ścieżkę pomiędzy dwoma podanymi wierzchołkami. Automatycznie zaznacza reprezentację ścieżki na ekranie i wyświetla koszt zaznaczonych krawędzi.

4. Przycisk *Dalej*. Pozwala uwzględnić następną krawędź w kolejce poprzez zaznaczenie jej na ekranie i powiększeniu całkowitego kosztu ścieżki o jej koszt.

5. Przycisk Wstecz. Pozwala pominąć poprzednią krawędź w kolejce poprzez odznaczenie jej na ekranie i pomniejszeniu całkowitego kosztu ścieżki o jej koszt.

6. Bieżący koszt zaznaczonej na grafie ścieżki.

Odwzorowanie utworzonego grafu w głównym oknie aplikacji:



5. Podsumowanie

Główne zadanie postawione przed aplikacją zostało zrealizowane. Wyznaczanie najkrótszej ścieżki w grafie odbywa się w sposób poprawny za pomocą algorytmu Dijkstry a interfejs użytkownika w czytelny sposób prezentuje odnalezioną ścieżkę oraz bieżący obliczony koszt przejścia.