Politechnika Śląska
Wydział Automatyki, Elektroniki i Informatyki

Mateusz Forczmański

Narzędzie do wizualizacji działań w sieciach przepływowych

projekt inżynierski

kierujący pracą: dr inż. Agnieszka Debudaj-Grabysz

Gliwice, 22 grudnia 2015

Spis treści

Rozdział 1

Wstęp

Sieć przepływowa jest abstrakcyjnym modelem danych, który jest wykorzystywany do rozwiązywania problemów związanych z przepływem produktów. Podobnie jak w klasycznym problemie producenta i konsumenta, istnieje w systemie źródło, gdzie produkty są tworzone, oraz ujście, gdzie są konsumowane. W sieci przepływowej zakłada się, że produkty są generowane i użytkowane w tym samym tempie, więc problemem nie jest synchronizacja, ale ilość produktów jakie można maksymalnie przesłać.

Intuicyjnie można wyobrazić sobie sieć przepływową jako układ hydrauliczny, gdzie krawędziami grafu są rury o stałej średnicy, wierzchołki jako połączenia rur, a dwa z nich są wyszczególnione jako miejsce wlewu oraz wylewu cieczy. Średnica każdej rury określa jej przepustowość, czyli maksymalną ilość cieczy jaka może przez nią przepłynąć. Połączenia są jedynie rozgałęzieniami - nie gromadzą płynu, jedynie przekazują go dalej. Innymi słowy, ilość cieczy jaka wpływa do rozgałęzienia musi być równa ilości jaka z niej wypływa. Jest to ta sama zasada co pierwsze prawo Kirchhoffa dla przepływu prądu w obwodzie elektrycznym.

Z siecią przepływową związany jest *problem maksymalnego przepływu*, czyli największej wartości produktów, jaka może przepłynąć w danej sieci ze źródła do ujścia. W swojej pracy inżynierskiej zająłem się implementacją trzech algorytmów służących do znajdowania maksymalnego przepływu w sieci: Forda-Fulkersona, Dinica oraz MKM (Malhortra, Kumar i Mahaswari). Zaprojektowana przeze mnie aplikacja ma mieć charakter edukacyjny: student, który będzie korzystał z programu powinien móc utworzyć własną sieć (wedle swojego zamysłu bądź odwzorować przykład z książki) oraz wykonać na niej jeden z algorytmów. Aplikacja ma na celu umożliwienie mu zrozumienie działania algorytmu krok po kroku, pracę na wielu sieciach oraz zapisanie swojego postępu.

- krótkie wprowadzenie (ok. 2 strony)
- przewodnik po pracy
- wstęp (i zakończenie) najłatwiej jest pisać na końcu

Rozdział 2

Analiza tematu

2.1 Podstawowe pojęcia

2.1.1 Definicja grafu

Sieć przepływowa $G = (V, E)$ jest grafem prostym skierowanym; strukturą, która składa się z dwóch zbiorów:

- n-elementowego zbioru wierzchołków $V = \{v_1, v_2, ..., v_n\}$
- m-elementowwgo zbioru uporządkowanych łuków $E = \{e_1, e_2, ..., e_m\}$, gdzie $e_i = (v_j, v_k)$; $\quad i = 1, ..., m$; $\quad v_j, v_k \in V$. Innymi słowy, każdy łuk jest parą wierzchołków należących do zbioru V.

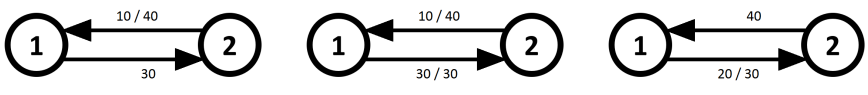
Grafem prostym nazywany taki graf w którym nie istnieją *pętle*. Pętlą nazywamy łuk (v_j, v_k) dla którego $v_j = v_k$. W sieci przepływowej wyróżnia się dwa wierzchołki: źródło $s \in V$ oraz ujście $t \in V$. W sieci przepływowej definiuje się ponadto *funkcję przepustowości* $c : E \to R_{\geq 0}$ odwzorowującą zbiór łuków w zbiór liczb rzeczywistych. Dodatkowo, w celu uproszczenia dziedziny, zakłada się, że z każdego wierzchołka u $v \in V \backslash \{s, t\}$ istnieje ścieżka prowadząca ze źródła do ujścia. Założenie to ma zapewnić, że w rozpatrywanych grafach będzie istniała co najmniej jedna ścieżka ze źródła do ujścia.

Przepływ w sieci G jest funkcją f, która odwzorowuje zbiór łuków w zbiór liczb rzeczywistych $f : E \to R_{\geq 0}$ i spełnia następujące założenia:

- Warunek przepustowości: $\forall_{v_i, v_j \in V} : f(v_i, v_j) \leq c(v_i, v_j)$.
  Przepływ w żadnym z łuków nie może przekroczyć jego przepustowości.
- Warunek skośnej symetryczności: $\forall_{v_i, v_j \in V} : f(v_i, v_j) = -f(v_j, v_i)$.
  Przepływ w przeciwnym kierunku traktujemy jakby posiadał wartość ujemną.
- Warunek zachowania przepływu: $\forall_{v \in V \backslash \{s,t\}} : \sum_{u \in V} f(v, u) = \sum_{u \in V} f(u, v)$.
  Suma przepływów wpływających do wierzchołka musi być równa sumie przepływów wypływających z niego.

2.1.2 Przepływ netto

Przepływem netto nazywamy wartość funkcji przepływu $f(v_i, v_j)$ pomiędzy wierzchołkami v_i oraz v_j. Zgodnie z warunkiem skośnej symetryczności, wartość może być dodatnia lub ujemna. Dodatnia oznacza przepływ z wierzchołka v_i do wierzchołka v_j, natomiast ujemna - w przeciwnym kierunku. Sieć przepływowa jest grafem skierowanym, więc dla dwóch wierzchołków może posiadać parę sąsiednich łuków. Poniższy przykład przedstawia istotę przepływu netto, jeżeli w obu sąsiadach istnieje niezerowy przepływ.



(a) (b) (c)

Rysunek 1: Zobrazowanie przepływu netto

Na rysunku 1a istnieje przepływ z wierzchołka 2 do wierzchołka 1 o wartości 10, czyli $f(2,1) = 10$. Zgodnie z warunkiem skośnej symetryczności $f(1,2) = f(2,1) = -10$. Można powiedzieć, że wierzchołek 2 opuszcza 10 jednostek materiału, które wpływają do wierzchołka 1. Na rysunku 1b pojawia się drugi przepływ, z wierzchołka 1 do wierzchołka 2 przepływa 30 jednostek. W tym przypadku wartość przepływu w istocie jest równa $f(1,2) = -10 + 30 = 20$. Jest to równoznaczne sytuacji, w której istnieje tylko przepływ 20 jednostek z wierzchołka 1 do wierzchołka 2. Przepływ z wierzchołka 2 do wierzchołka 1 zostaje zmniejszony do $f(2,1) = 10 - 30 = -20$, co jest zgodne z warunkiem zachowania skośności.

2.1.3 Maksymalny przepływ w sieci

Wartość przepływu $|f|$ w sieci jest definiowana jako suma przepływów netto wychodzących ze źródła $\sum_{v \in V} f(s,v)$. Zgodnie z warunkiem zachowania przepływu, jest ona równa sumie wartości przepływów netto wpływających do ujścia $\sum_{v \in V} f(v,t)$. Problem maksymalnego przepływu polega na znalezieniu maksymalnej wartości $|f|$ w sieci przepływowej.[Agn12]

2.1.4 Sieć residualna

2.2 Algorytmy znajdowania maksymalnego przepływu

2.3 Wykorzystane technologie

Rozdział 3

Wymagania

- wymagania funkcjonalne i niefunkcjonalne
- przypadki użycia (diagramy UML)

Rozdział 4

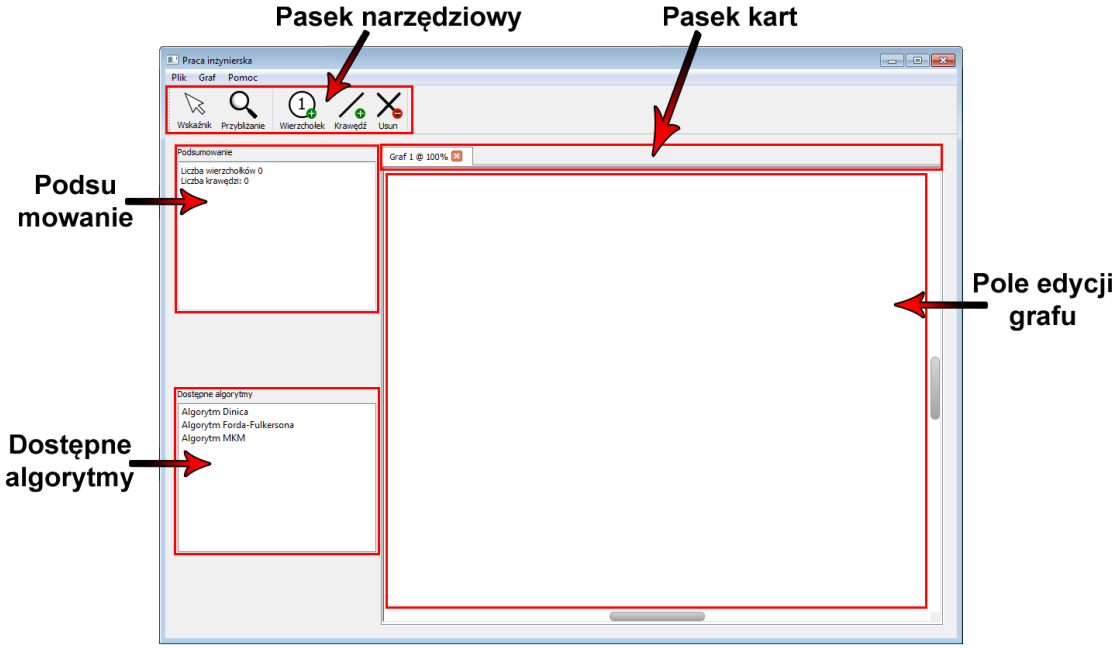
Specyfikacja zewnętrzna

4.1 Wymagania sprzętowe

Aplikacja działa pod systemami Windows 7 i nowszymi, na architekturze 64-bitowej. Instalacja nie jest wymagana, wystarczy otworzyć plik aplikacji .exe by móc z niej korzystać. Wszystkie biblioteki i pliki konfiguracyjne nie mogą zostać zmienione.

4.2 Korzystanie z aplikacji

Po uruchomieniu programu pojawia się okno z interfejsem graficznym złożonym z kilku podstawowych sekcji.



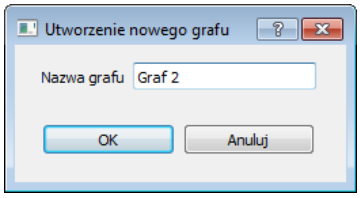
Rysunek 2: Interfejs graficzny

1. *Pasek narzędziowy* obejmuje pięć narzędzi do pracy z siecią przepływową: wskaźnik, lupę, dodanie grafu, dodanie krawędzi i usunięcie elementu.
2. *Podsumowanie* zawiera informacje na temat aktywnej w danej chwili sieci.
3. *Dostępne algorytmy* zawierają listę operacji, jakie można na sieci wykonać.
4. *Pasek kart* zawiera listę istniejących sieci w instancji aplikacji między którymi można się przełączać.

5. *Pole edycji grafu* to miejsce robocze, gdzie można umieszczać elementy grafu i pracować nad nim.

4.2.1 Tworzenie nowej sieci

Aby utworzyć nową sieć należy z paska menu wybrać *Plik* → *Nowy* lub skorzystać ze skrótu *CTRL + N*. Następnie pojawi się okienko w które należy wpisać roboczą nazwę sieci (można też pozostawić wygenerowaną domyślną nazwę). Po zatwierdzeniu zostanie dodana nowa karta z pustym polem edycyjnym.



Rysunek 3: Okienko z wyborem nazwy

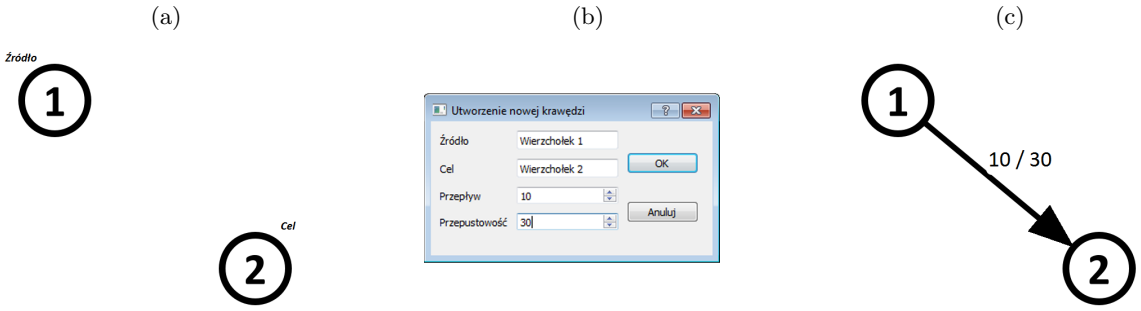
4.2.2 Rysowanie grafu

Dodanie wierzchołka

Aby dodać nowy wierzchołek należy wybrać narzędzie dodawania wierzchołków, a następnie kliknąć lewym przyciskiem myszy na puste miejsce w polu edycyjnym. Pojawi się nowy obiekt, miejsce kliknięcia będzie środkiem wierzchołka. Jeżeli klikniemy istniejący już obiekt, zostanie on zaznaczony.

Dodanie łuku

Aby móc dodać łuk na polu roboczym muszą istnieć co najmniej dwa wierzchołki. Należy wybrać narzędzie dodawania krawędzi i w pierwszej kolejności kliknąć na wierzchołek który będzie źródłem. Pojawi się nad nim etykieta *"Źródło"*. Potem należy kliknąć drugi wierzchołek, po najechaniu myszką pojawi się nad nim etykieta *"Ujście"*. Kliknięcie lewym przyciskiem myszy spowoduje potwierdzenie operacji i pojawienie okienka, gdzie należy wpisać przepływ i przepustowość tego łuku, domyślnie zero i jeden. Jeżeli między wierzchołkami już istnieje dane połączenie, operacja zostanie zignorowana. Łuk jest trwało przymocowany do wierzchołków i nie można zmienić jego pozycji.



Rysunek 4: Etapy dodawania łuku

Usuwanie elementów

W każdej chwili można usunąć wierzchołek lub łuk w sieci. W tym celu należy wybrać narzędzie usuwania oraz kliknąć wybrany element. Usunięcie łuku powoduje pozbycie się z sieci tylko niego, z kolei usunięcie wierzchołka powoduje zlikwidowanie wszystkich łuków, które wchodzą lub wychodzą z niego. Można usuwać wiele elementów naraz, narzędzie usuwa wszystko co jest zaznaczone.

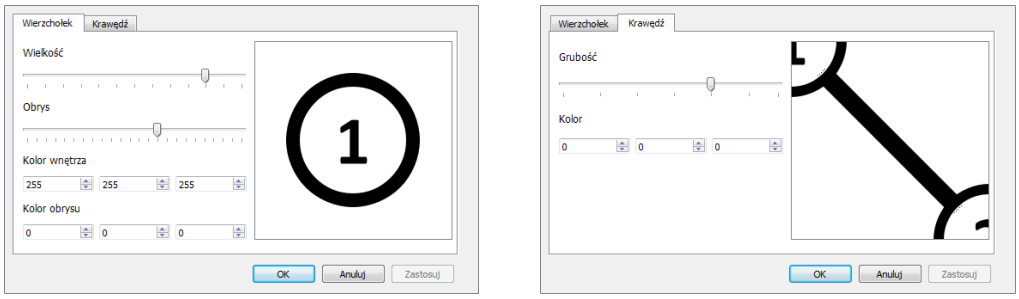
Poprawa estetyki sieci

Aby poprawić wygląd i czytelność sieci istnieje możliwość wyrównania pozycji wierzchołków. W tym celu należy zaznaczać dwa lub więcej wierzchołków, a następnie kliknąć *SHIFT* + jedną ze strzałek kierunkowych (↑, ↓, →, ←). Wszystkie wierzchołki w sieci posiadają te same rozmiary i komendy należy interpretować następująco:

- *SHIFT* + ↑ lub ↓ - ustawia pozycję pionową wszystkich wierzchołków równą pozycji wierzchołka najbardziej wysuniętego do góry lub na dół.
- *SHIFT* + → lub ← - ustawia pozycję poziomą wszystkich wierzchołków równą pozycji wierzchołka najbardziej wysuniętego na prawo lub na lewo.

Konfiguracja wyglądu grafu

Rozmiar i kolory wierzchołków oraz łuków można modyfikować w ustawieniach. W tym celu należy wybrać z paska menu *Graf* → *Kształt* lub skorzystać ze skrótu klawiszowego *CTRL+SHIFT+K*. Pojawi się okienko z konfiguracją lokalnego grafu.



(a) Konfiguracja wierzchołków (b) Konfiguracja łuków

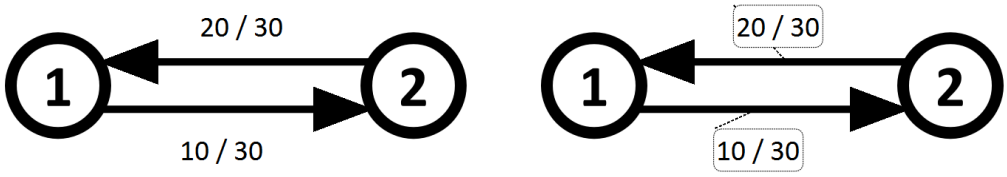
Rysunek 5: Możliwe zmiany w wyglądzie sieci

Oba okienka umożliwiają zmianę koloru i rozmiaru, w przypadku wierzchołka możliwa jest ponadto kontrola nad grubością obrysu oraz jego koloru. Kliknięcie przycisku *OK* lub *Zastosuj* skutkuje zapisaniem konfiguracji i aktualizacją wyglądu. Każda sieć posiada swoją lokalną konfigurację, w przypadku zapisywania jej do pliku, ona również jest zachowywana.

Operowanie na sieci

Wybierając narzędzie wskaźnika możliwe jest przesuwanie wierzchołków po polu edycyjnym oraz ich zaznaczanie. Można zaznaczyć wiele wierzchołków i łuk przytrzymując klawisz *CTRL* lub skorzystać z myszki - wystarczy kliknąć lewym przyciskiem na pustym polu, przytrzymać i przeciągnąć kursor by pojawił się prostokąt zaznaczający elementy sieci. Jeżeli jest wiele zaznaczonych elementów przesunięcie jednego z nich spowoduje przesunięcie wszystkich.

Zaznaczenie etykiety łuku, informującej o wartościach przepływu i przepustowości, spowoduje pojawienie się linii przerywanej prowadzącej do środka łuku. Ułatwia to pracę z siecią, gdy łuków i etykiet pojawia się coraz więcej.



(a) Niezaznaczone etykiety (b) Zaznaczone etykiety

Rysunek 6: Przedstawienie linii łączącej etykietę z łukiem

4.2.3 Otwieranie i zapisywanie

Aby móc zapisać sieć przepływową do pliku w sesji programu musi być otwarta co najmniej jedna karta. Klikając *CTRL+S* lub wybierając *Plik* → *Zapisz jako* można zapisać stan pracy do pliku XML. Program poprosi o wskazanie folderu oraz wpisanie nazwy pliku. Potwierdzenie utworzy nowy plik.

Aby otworzyć plik należy wybrać z paska menu *Plik* → *Otwórz* lub skorzystać ze skrótu klawiszowego *CTRL+O*. Program otworzy okno dialogowe i poprosi o wskazanie pliku XML jaki ma otworzyć. Jeżeli ma on poprawny format, sieć przepływowa zostanie wczytana i pokazana w nowej karcie.

4.2.4 Wykonanie algorytmów

W momencie utworzenia karty z siecią, nowym lub wczytanym z pliku, w lewym dolnym rogu pojawia się lista dostępnych algorytmów, jakie można wykonać w ramach programu: Forda-Fulkersona, Dinica oraz MKM.

Warunki poprawności sieci

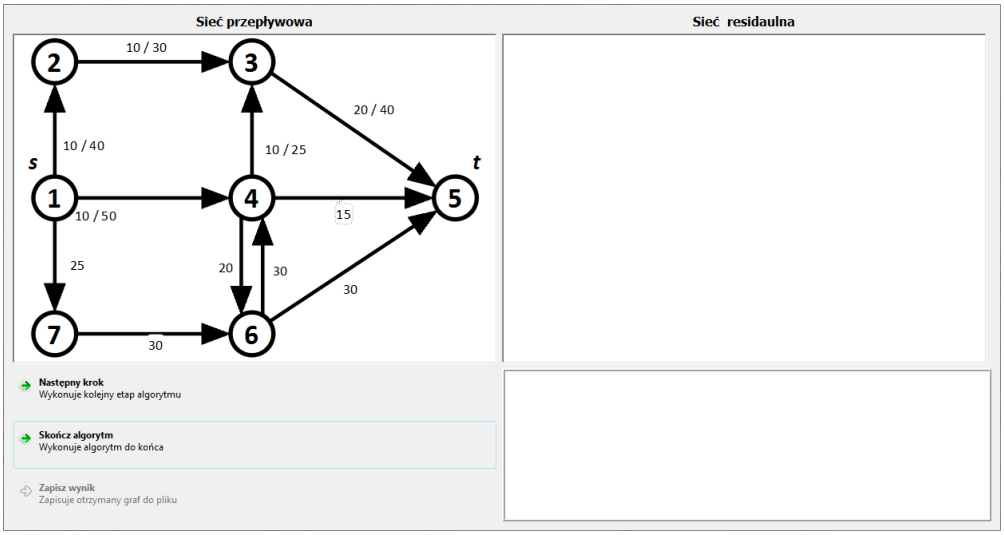
Aby uruchomić algorytm sieć musi być poprawnie zbudowana, tzn. spełniać następujące założenia:

- wierzchołek źródłowy oraz wierzchołek ujściowy muszą być oznaczone, by móc wyznać maksymalny przepływ. W tym celu należy kliknąć prawym przyciskiem myszki na wierzchołkach i wybrać z menu kontekstowego odpowiednio *oznacz jako źródło* oraz *oznacz jako ujście*. Nad wierzchołkiem źródłowym pojawi się etykieta ***s***, a nad wierzchołkiem ujściowym etykieta ***t***.
- Nie może istnieć żaden wierzchołek, inny niż źródło i ujście, przez który nie istnieje droga ze źródła do ujścia. Ma to na celu zapewnić, że w zbudowanym grafie istnieje co najmniej jedna ścieżka ze źródła do ujścia. Wystarczy, że dla każdego wierzchołka będą istniały co najmniej dwa łuki, wchodzący i wychodzący z niego.
- Zasada zachowania przepływu nie może zostać złamana. Program nie dopuszcza, żeby łuk mógł mieć większą wartość przepływu niż przepustowość, ale podczas budowania sieci nie można cały czas kontrolować tej zasady. Jest ona sprawdzana dopiero przy próbie uruchomienia algorytmu.

Jeżeli któryś z tych warunków nie zostanie spełniony, pojawi się okienko informujące dokładnie o rodzaju błędu oraz w którym miejscu grafu on wystąpił.

Okno wykonania

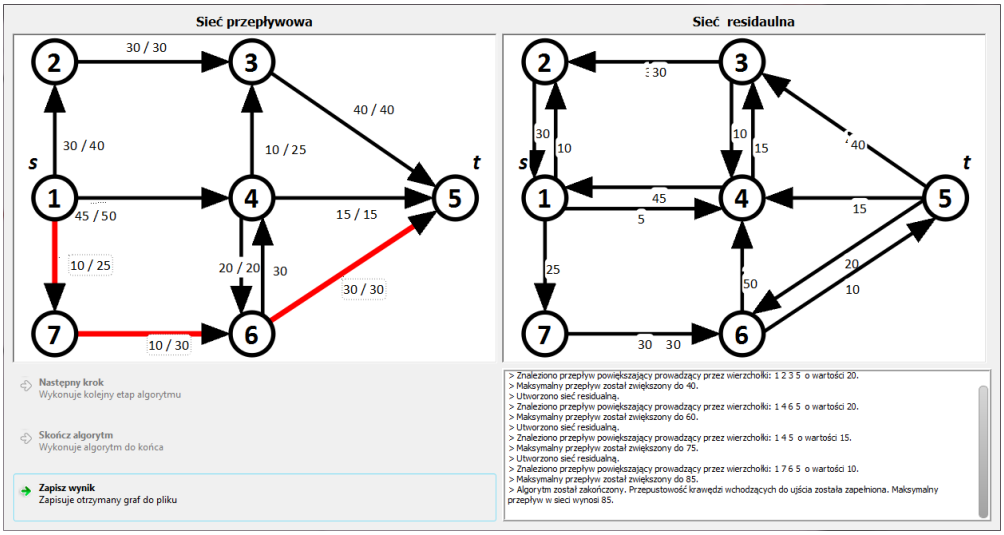
Jeżeli graf został poprawnie zbudowany, po kliknięciu w jeden z algorytmów pojawi się nowe okno w którym zostanie on wykonany.



Rysunek 7: Okno ilustrujące przebieg algorytmu Forda-Fulkersona

Okno zostało podzielone na 4 obszary:

- W górnym lewym rogu znajduje się pierwotna sieć przepływowa, której przepływ zostaje zwiększany wraz z kolejnymi krokami.
- W górnym prawym rogu jest podgląd tworzonej sieci residualnej oraz ścieżki powiększającej, jaka została w niej znaleziona. W oknie dla algorytmów Dinica i MKM znajduje się jeszcze dodatkowy, trzeci podgląd, obrazujący zmiany w warstwowej sieci residualnej.
- W dolnym lewym rogu znajdują się trzy przyciski:
  - *Następny krok*, jego naciśnięcie powoduje wykonanie istotnej instrukcji algorytmu i przedstawienie zaistniałych zmian w podglądach. *Skończ algorytm*, realizuje całość zadania, aż maksymalny przepływ zostanie znaleziony. Aplikacja wykonuje wówczas instrukcję *Następny krok* w odstępach równych 0.5 sekundy, aż do zakończenia obliczeń.
  - *Zapisz wynik*, przycisk staje się uaktywniony po zakończeniu zadania, umożliwia zapisanie sieci ze znalezionym maksymalnym przepływem do pliku XML.
- W dolnym prawym rogu znajduje się konsola w której pojawiają się informacje o przebiegu algorytmu. Za każdym razem, gdy instrukcja *Następny krok* zostanie wykonana, w tym polu pojawiają się szczegółowe informacje o utworzonych strukturach, znalezionych przepływach i innych zmianach w sieci.



Rysunek 8: Okno po zakończeniu wykonywania algorytmu

Przykładowe przedstawienie krok po kroku wyszukiwania maksymalnego przepływu znajduje się w dodatku C.

Rozdział 5

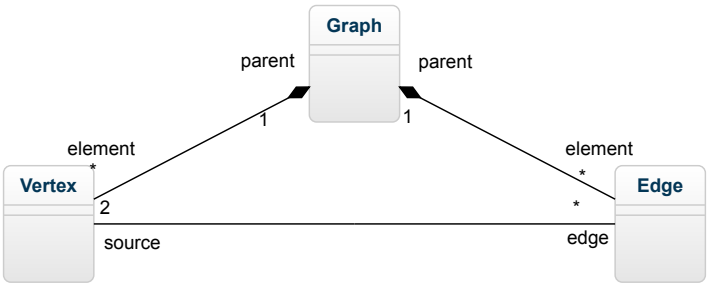
Specyfikacja wewnętrzna

5.1 Struktura danych

Reprezentację grafów w pamięci komputera można wykonać na dwa sposoby:

- *niskopoziomowy*: grafem jest macierz $n \times n$ liczb całkowitych, gdzie n to liczba wierzchołków.[Zbi10]
- *wysokopoziomowy*: wierzchołki i łuki grafu są zakapsułkowane do osobnych klas.[Agn12]

W swojej pracy wybrałem drugie podejście. Poprawne zdefiniowanie klas i zależności między nimi pozwala mi w intuicyjny sposób implementować algorytmy z pseudokodu: operacje rodzaju *"wykonaj daną instrukcję na wszystkich krawędziach"* można łatwo odwzorować pobierając kolekcję krawędzi i dodając im odpowiednią metodę. Nie ma potrzeby adaptować operacji na zupełnie inną strukturę.



Rysunek 9: Diagram klas struktury sieci

Zgodnie z definicją w 2.1.1, graf (*Graph*) składa się z dwóch zbiorów, wierzchołków (*Vertex*) oraz łuków (*Edge*). Graf jest kompozytem, wierzchołki i łuki stanowią jego ciało, a usunięcie obiektu grafu powoduje usunięcie jego wszystkich podrzędnych elementów. Wierzchołek jest niezależnym elementem, z kolei łuk posiada referencje do dwóch obiektów klasy wierzchołka - łuk nie może istnieć bez pary wierzchołków, które go tworzą, a usunięcie wierzchołka powoduje usunięcie wszystkich łuków wychodzących i wchodzących do niego.

5.2 Konfiguracja wyglądu sieci

Każdy obiekt sieci przepływowej posiada swoją konfigurację, w której znajdują się informacje dotyczące rysowania sieci. Każda sieć musi posiadać istniejącą konfigurację, jest to wymuszone poprzez jej konstruktor.

```
explicit FlowNetwork(GraphConfig * config);
```

Wierzchołki i łuki sieci przepływowej podczas wydarzenia rysowania pobierają swoje ustawienia z rodzica. Klasa `GraphConfig` zawiera w sobie pięć składowych: nazwę sieci oraz cztery konteksty - sposób rysowania zwykłego elementu oraz zaznaczonego.

```
class GraphConfig
{
protected:
    QString _name;
    VertexContext * _normalVertexContext;
    EdgeContext * _normalEdgeContext;
    VertexContext * _selectedVertexContext;
    EdgeContext * _selectedEdgeContext;
}
```

Informacje o sposobie rysowania wierzchołka są zawarte w klasie `VertexContext`, a łuki w klasie `EdgeContext`. Idea oparta jest o wzorzec ***Pyłku***. Każdy obiekt wierzchołka i łuku posiada swój niezmienny stan wewnętrzny, jak identyfikator oraz pozycję, a także stan zewnętrzny, jakim jest kolor i kształt[Eri10]. Kontekst oznacza właśnie stan zewnętrzny, którymi grafy mogą się różnić. Każdy wierzchołek i łuk posiada swój wskaźnik na aktualny kontekst, a w momencie, gdy następuje kliknięcie w elementu graf i mechanizm *Qt* wykrywa zmianę zaznaczenia, wskaźnik wskazuje na przeciwny kontekst.

```
QVariant VertexImage::itemChange(GraphicsItemChange change, const QVariant &value)
{
    // ...
    else if (change == ItemSelectedChange)
    {
        if (value.toBool() == true)
        {
            _context = getParent()->getConfig()->SelectedVertexContext();
        }
        else
        {
            _context = getParent()->getConfig()->NormalVertexContext();
        }
    }
}
```

5.3 Serializacja sieci

Zapis oraz odczyt plików z grafami odbywa się w klasie `GraphSerializer`. Posiada ona dwie publiczne metody:

- `void serialize(GraphImage const & graph, std::string const & fileName);`
  Metoda *serialize* przyjmuje jako parametr referencję do graficznej reprezentacji grafu, który ma zapisać oraz nazwę pliku wyjściowego wraz ze ścieżką.
- `GraphImage * deserialize(std::string const & filePath);`
  Metoda *deserialize* przyjmuje jako parametr ścieżkę do pliku, który ma odczytać. Zwraca wskaźnik do graficznej reprezentacji grafu odczytanego z pliku.

Sieci przepływowe są zapisywane do formatu XML. W celu obsługi tego języka, klasa `GraphSerializer` korzysta z darmowej biblioteki do parsowania plików XML, *RapidXML*, autorstwa Marcina Kalicińskiego[1]

5.3.1 Format pliku XML

Plik zaczyna się od korzenia o nazwie `<Graph>`. Format pliku można podzielić na dwie części, konfiguracyjną (gałąź `<Config>`) i modelową (gałąź `<Model>`). W pierwszej znajdują się globalne ustawienia wyglądu, jak rozmiary i kolory wierzchołków i łuków. Druga część zawiera model sieci

[1] Strona domowa *RapidXML*: `http://rapidxml.sourceforge.net/`

przepływowej - zarówno pozycje i numery wierzchołków, jak i ich połączenia poprzez krawędzie, ich kierunek, przepustowość, przepływ oraz położenie napisu z informacją. Przykładowa sieć przepływowa oraz jej serializacja do pliku znajduje się w dodatku A.

5.4 Algorytmy

Wszystkie trzy algorytmy wykonane w tej pracy zostały zakapsułkowane do osobnych klas. Ich klasą bazową bazową jest `FlowNetworkAlgorithm`, która definiuje interfejs algorytmów oraz zawiera wspólną dla wszystkich funkcjonalność.

5.4.1 Tworzenie sieci residualnej

Algorytm służący do wygenerowania sieci residualnej jest niezmienny dla wszystkich klas i jest zawarty w metodzie klasy `FlowNetworkAlgorithm`.

```
int makeResidualNetwork(FlowNetwork * network, FlowNetwork *& outResidaulNetwork)
```

Funkcja przyjmuje dwa parametry:

- *network* jest siecią przepływową na podstawie której jest utworzona sieć przepływowa
- *outResidaulNetwork* jest referencją na wskaźnik do obiektu sieci, która będzie siecią residualną. W pierwszym kroku jest to głęboka kopia sieci przepływowej.

Wartością zwracaną jest najkrótsza odległość między źródłem, a ujściem, jeżeli tworzona sieć jest *warstwową siecią residualną*. Jeżeli nie jest, funkcja zwraca zero. W pierwszym kroku, algorytm usuwa wszystkie łuki z wyjściowej sieci residualnej zostawiając same wierzchołki. Następnie przechodzi przez wszystkie łuki sieci przepływowej i wykonuje poniższy algorytm. Utworzenie tablicy odwiedzonych

Algorithm 1 Tworzenie nowego łuku w sieci residualnej

procedure UTWÓRZ NOWY ŁUK(Łuk w sieci przepływowej)
  Oblicz przepustowość residualną
  if Istnieje łuk sąsiedni oraz łuk sąsiedni nie został odwiedzony **then**
    Oblicz przepływy między wierzchołkami zgodnie z zachowaniem przepływu netto w 2.1.2
    Dodaj łuk oraz łuk sąsiedni do odwiedzonych
  end if
  if Przepływ w łuku $\neq 0$ **then**
    Utwórz nowy łuk w sieci residualnej w tym samym kierunku o przepustowości równej przepływowi
  end if
  if Przepustowość residualna $\neq 0$ **then**
    Utwórz nowy łuk w sieci residualnej w przeciwnym kierunku o przepustowości równej przepustowości residualnej
  end if
end procedure

łuków ma zapewnić, że w sieci residualnej nie zostaną utworzone nadmiarowe łuki gdy pętla dojdzie do sąsiada. Pełny kod algorytmu znajduje się dodatku B.

5.4.2 Szukanie ścieżki powiększającej

5.4.3 Zwiększenie przepływu w sieci

5.4.4 Warunki stopu

5.4.5 Algorytm Forda-Fulkersona

5.4.6 Algorytm Dinica

5.4.7 Algorytm MKM

Rozdział 6

Testowanie i uruchamianie

Sposób testowania w ramach pracy (organizacja eksperymentów, przypadki testowe, wyniki, zakres testowania – pełny/niepełny)

Rozdział 7

Uwagi o przebiegu i wynikach prac

- uzyskane wyniki w świetle postawionych celów
- kierunki ewentualnych danych prac (rozbudowa funkcjonalna ...)
- problemy napotkane w trakcie pracy

Rozdział 8

Podsumowanie

Bibliografia

[Agn12] Agnieszka Debudaj-Grabysz, Sebastian Deorowicz, Jacek Widuch. *Algorytmy i struktury danych. Wybór zaawansowanych metod*. Wydawnictwo Politechniki Śląskiej, Gliwice 2012.

[Eri10] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides. "Wzorce projektowe. Elementy oprogramowania wielokrotnego użytku". W: Wydawnictwo HELION, 2010. Rozd. PYŁEK (FLYWEIGHT).

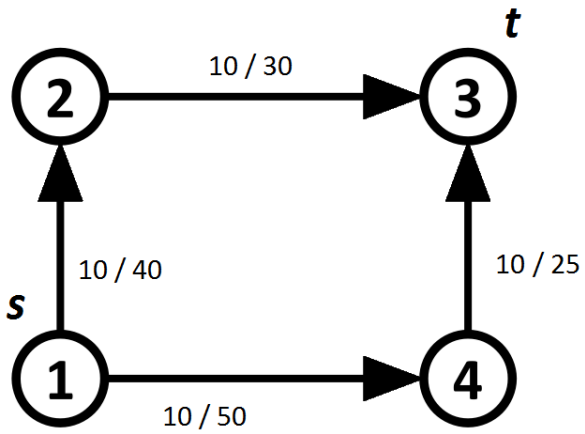
[Tho09] Thomas H. Corman, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein. *Introduction to Algorithms*. Third. The MIT Press, 2009.

[Zbi10] Zbigniew J. Czech, Sebastian Deorowicz, Piotr Fabian. "Algorytmy i struktury danych. Wybrane zagadnienia". W: Wydawnictwo Politechniki Śląskiej, Gliwice 2010, s. 133.

Załączniki

Dodatek A

Przykład serializacji




```
<?xml version="1.0" encoding="UTF-8"?>
<Graph type="flow_network" source="1" target="3" weighted="true">
   <Config>
      <Name>Graf 2</Name>
      <VertexContext type="normal" size="59" strokeSize="12">
         <Color type="color" r="255" g="255" b="255"/>
         <Color type="strokeColor" r="0" g="0" b="0"/>
         <Font bold="true" size="60" family="Calibri"/>
      </VertexContext>
      <VertexContext type="selected" size="59" strokeSize="12">
         <Color type="color" r="0" g="0" b="255"/>
         <Color type="strokeColor" r="0" g="0" b="80"/>
         <Font bold="true" size="60" family="Calibri"/>
      </VertexContext>
      <EdgeContext type="normal" size="10">
         <Color type="color" r="0" g="0" b="0"/>
      </EdgeContext>
      <EdgeContext type="selected" size="15">
         <Color type="color" r="255" g="0" b="0"/>
      </EdgeContext>
   </Config>
```

```
22    <Model>
23       <Vertex id="1">
24          <Position x="-244.531250" y="72.376556"/>
25          <Point id="1" x="-185" y="72"/>
26          <Point id="2" x="-244" y="13"/>
27          <Point id="6" x="-244" y="131"/>
28       </Vertex>
29       <Vertex id="2">
30          <Position x="-244.531250" y="-302.734344"/>
31          <Point id="2" x="-244" y="-243"/>
32          <Point id="3" x="-185" y="-302"/>
33          <Point id="4" x="-185" y="-302"/>
34       </Vertex>
35       <Vertex id="3">
36          <Position x="298.898376" y="-302.734344"/>
37          <Point id="3" x="298" y="-243"/>
38          <Point id="4" x="239" y="-302"/>
39       </Vertex>
40       <Vertex id="4">
41          <Position x="298.898376" y="72.376556"/>
42          <Point id="1" x="239" y="72"/>
43          <Point id="3" x="298" y="13"/>
44          <Point id="4" x="298" y="13"/>
45          <Point id="8" x="298" y="131"/>
46          <Point id="10" x="293" y="13"/>
47       </Vertex>
48       <Edge type="straight" id="2" vertexFrom="1" vertexTo="2" capacity="40"
             flow="10" offsetType="false" offsetValue="0.000000">
49          <Position x="-244.531250" y="13.376556"/>
50          <EdgeTextItem>
51             <Position x="20.013672" y="-115.109367"/>
52          </EdgeTextItem>
53       </Edge>
54       <Edge type="straight" id="1" vertexFrom="1" vertexTo="4" capacity="50"
             flow="10" offsetType="false" offsetValue="0.000000">
55          <Position x="-185.531250" y="72.376556"/>
56          <EdgeTextItem>
57             <Position x="72.968758" y="20.110939"/>
58          </EdgeTextItem>
59       </Edge>
60       <Edge type="straight" id="4" vertexFrom="2" vertexTo="3" capacity="30"
             flow="10" offsetType="false" offsetValue="0.000000">
61          <Position x="-185.531250" y="-302.734344"/>
62          <EdgeTextItem>
63             <Position x="134.121063" y="-70.312492"/>
64          </EdgeTextItem>
65       </Edge>
66       <Edge type="straight" id="3" vertexFrom="4" vertexTo="3" capacity="25"
             flow="10" offsetType="false" offsetValue="0.000000">
67          <Position x="298.898376" y="13.376556"/>
68          <EdgeTextItem>
69             <Position x="33.179668" y="-123.274200"/>
70          </EdgeTextItem>
71       </Edge>
72    </Model>
73 </Graph>
```

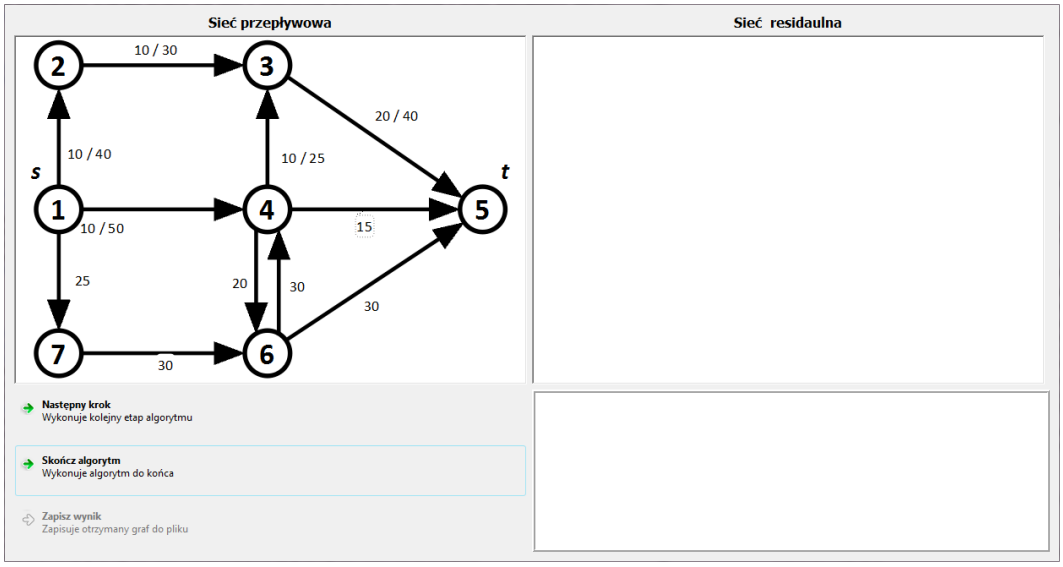
Dodatek B

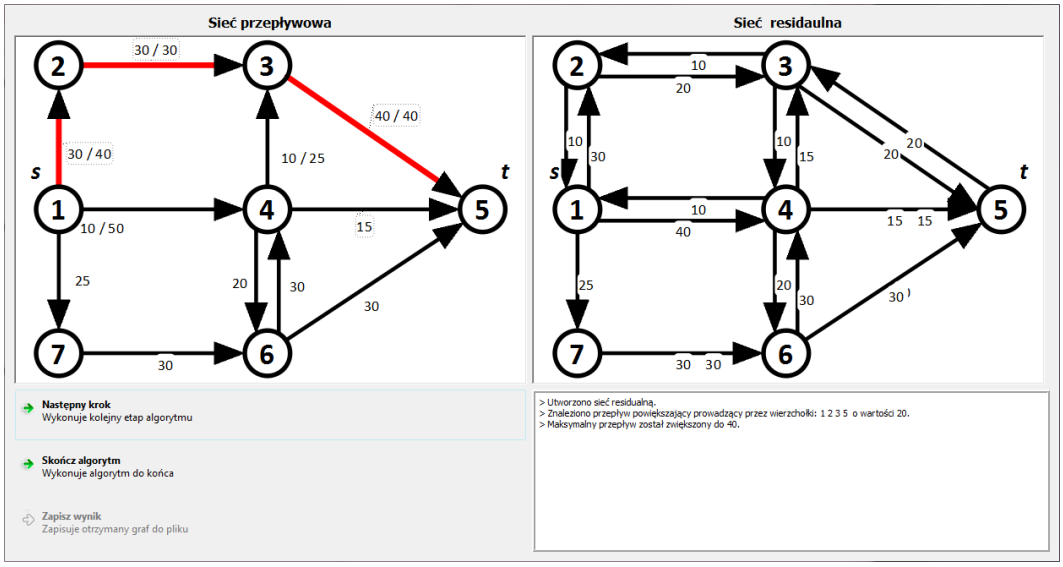
Tworzenie sieci residualnej

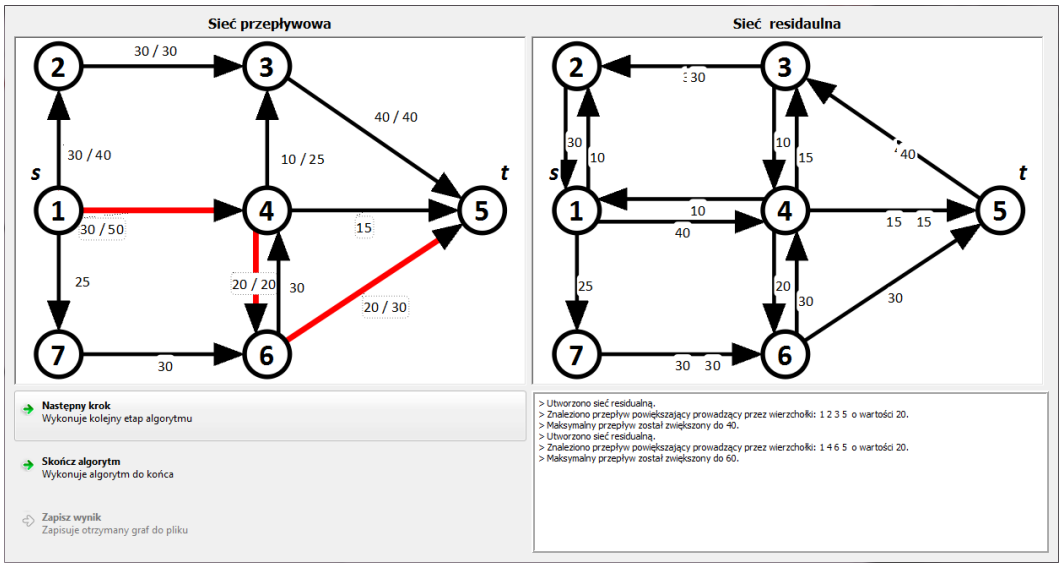
```
1  int FlowNetworkAlgorithm::makeResidualNetwork(FlowNetwork * network, FlowNetwork *& outResidaulNetwork)
2  {
3      // usunięcie starych krawędzi
4      auto oldEdges = outResidaulNetwork->getEdges();
5      for (auto it = oldEdges.begin(); it != oldEdges.end(); ++it)
6      {
7          outResidaulNetwork->removeEdge(*it);
8      }
9      // analiza krawędzi i utworzenie sieci residualnej
10     auto edges = network->getEdges();
11     QList<EdgeImage*> visitedNeighbours;
12     EdgeImage *neighbor;
13     for (EdgeImage * edge : edges)
14     {
15         int capacity = edge->getCapacity();
16         int flow = edge->getFlow();
17         int vertexFromId = edge->VertexFrom()->getId();
18         int vertexToId = edge->VertexTo()->getId();
19         int residualCapacity = capacity - flow;
20         if (edge->hasNeighbor() && !visitedNeighbours.contains(neighbor = network->edgeAt(vertexToId,
       ↪  vertexFromId)))
21         {
22             visitedNeighbours.push_back(neighbor);;
23             visitedNeighbours.push_back(edge);
24             int neighborFlow = neighbor->getFlow();
25             int neighborCapacity = neighbor->getCapacity();
26             residualCapacity = capacity - flow + neighborFlow;
27             int neighborResidualCapacity = neighborCapacity - neighborFlow + flow;
28             if (residualCapacity != 0)
29                 outResidaulNetwork->addEdge(vertexFromId, vertexToId, residualCapacity);
30             if (neighborResidualCapacity != 0)
31                 outResidaulNetwork->addEdge(vertexToId, vertexFromId, neighborResidualCapacity);
32         }
33         else
34         {
35             if (flow != 0)
36                 outResidaulNetwork->addEdge(vertexToId, vertexFromId, flow);
37             if (residualCapacity != 0)
38                 outResidaulNetwork->addEdge(vertexFromId, vertexToId, residualCapacity);
39         }
40     }
41     return 0;
42 }
```

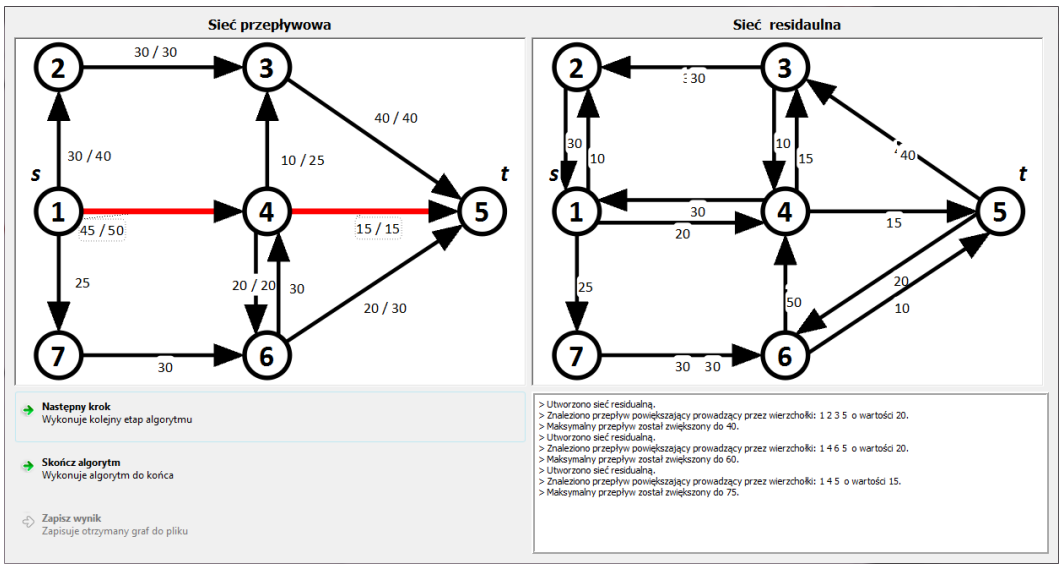
Dodatek C

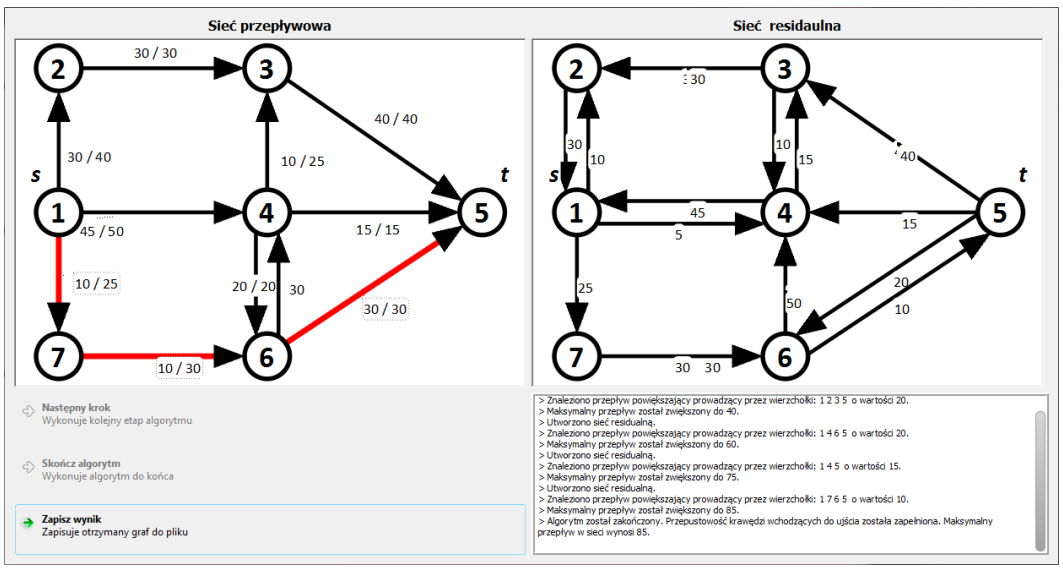
Realizacja algorytmu Forda-Fulkersona






Sieć przepływowa
Sieć residualna
30 / 30
40 / 40
30 / 40
10 / 25
30 / 50
15
20 / 20
30
20 / 30
25
30
Następny krok
Wykonuje kolejny etap algorytmu
Skończ algorytm
Wykonuje algorytm do końca
Zapisz wynik
Zapisuje otrzymany graf do pliku
> Utworzono sieć residualną.
> Znaleziono przepływ powiększający prowadzący przez wierzchołki: 1 2 3 5 o wartości 20.
> Maksymalny przepływ został zwiększony do 40.
> Utworzono sieć residualną.
> Znaleziono przepływ powiększający prowadzący przez wierzchołki: 1 4 6 5 o wartości 20.
> Maksymalny przepływ został zwiększony do 60.


Sieć przepływowa
Sieć residualna
30 / 30
40 / 40
30 / 40
10 / 25
45 / 50
15 / 15
20 / 20
30
20 / 30
25
30
Następny krok
Wykonuje kolejny etap algorytmu
Skończ algorytm
Wykonuje algorytm do końca
Zapisz wynik
Zapisuje otrzymany graf do pliku
> Utworzono sieć residualną.
> Znaleziono przepływ powiększający prowadzący przez wierzchołki: 1 2 3 5 o wartości 20.
> Maksymalny przepływ został zwiększony do 40.
> Utworzono sieć residualną.
> Znaleziono przepływ powiększający prowadzący przez wierzchołki: 1 4 6 5 o wartości 20.
> Maksymalny przepływ został zwiększony do 60.
> Utworzono sieć residualną.
> Znaleziono przepływ powiększający prowadzący przez wierzchołki: 1 4 5 o wartości 15.
> Maksymalny przepływ został zwiększony do 75.


Sieć przepływowa
Sieć residualna
30 / 30
40 / 40
30 / 40
10 / 25
45 / 50
15 / 15
10 / 25
20 / 20
30
30 / 30
10 / 30
Następny krok
Wykonuje kolejny etap algorytmu
Skończ algorytm
Wykonuje algorytm do końca
Zapisz wynik
Zapisuje otrzymany graf do pliku
> Znaleziono przepływ powiększający prowadzący przez wierzchołki: 1 2 3 5 o wartości 20.
> Maksymalny przepływ został zwiększony do 40.
> Utworzono sieć residualną.
> Znaleziono przepływ powiększający prowadzący przez wierzchołki: 1 4 6 5 o wartości 20.
> Maksymalny przepływ został zwiększony do 60.
> Utworzono sieć residualną.
> Znaleziono przepływ powiększający prowadzący przez wierzchołki: 1 4 5 o wartości 15.
> Maksymalny przepływ został zwiększony do 75.
> Utworzono sieć residualną.
> Znaleziono przepływ powiększający prowadzący przez wierzchołki: 1 7 6 5 o wartości 10.
> Maksymalny przepływ został zwiększony do 85.
> Algorytm został zakończony. Przepustowość krawędzi wchodzących do ujścia została zapełniona. Maksymalny przepływ w sieci wynosi 85.