PŘÍRODOVĚDECKÁ FAKULTA UNIVERZITY PALACKÉHO
KATEDRA INFORMATIKY

BAKALÁŘSKÁ PRÁCE

Toky v sítích

2014 Michaela Wehmhönerová

Anotace

Tato práce se zabývá sestavením aplikace pro výpočet maximálního toku sítě, nalezení minimálního řezu a zobrazení údajů ve výchozím grafu.

Děkuji RNDr. Arnoštu Večerkovi za vedení mé bakalářské práce, pomoc při shánění materiálů a přínosné konzultace.

Obsah

Seznam obrázků

1. Úvod

Téma této práce je z oblasti teorie grafů, zabývá se sestavením aplikace pro podporu řešení přepravních a dalších úloh, které lze modelovat pomocí toků v sítích. Aplikace umožňuje sestavení orientovaného grafu popisující strukturu sítě a vložení příslušných údajů (zdroj sítě, stok sítě, kapacita jednotlivých hran sítě, počáteční tok sítě). Aplikace následně vypočítá maximální tok sítě, zobrazí údaje o něm ve výchozím grafu. Aplikace rovněž najde a zobrazí minimální řez sítě.

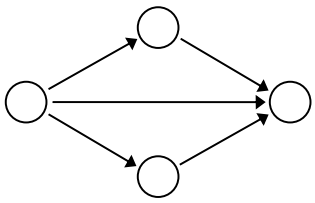
2. Teoretické zpracování

Tato kapitola slouží k vysvětlení základních pojmů. Jsou zde využity dříve používané definice a pojmy, jejichž zdroje jsou uvedeny v referenci.

2.1. Úvod do teorie grafů

Definice 1. ***Orientovaný graf*** *je dvojice* $\overrightarrow{G} = (U, H)$*, kde platí:*

- *U je neprázdná množina uzlů daného grafu*
- *H je neprázdná množina orientovaných hran*
- *Každá hrana* $h \in H(\overrightarrow{G})$ *je uspořádaná dvojice uzlů (u, v), hrana h vede z uzlu u do uzlu v*



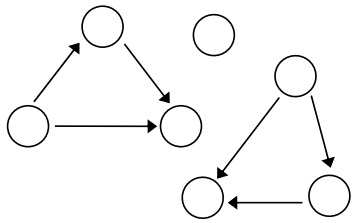
Obrázek 1. Příklad orientovaného grafu

Definice 2. ***Orientovaný sled*** *v grafu* $\overrightarrow{G} = (U, H)$*, je posloupnost uzlů a hran* $u_0, h_0, u_1, h_1, ..., u_i$*, kde všechny za sebou navazující hrany mají společný koncový uzel, který je uveden mezi nimi.*

Definice 3. ***Orientovaná cesta*** *v grafu* $\overrightarrow{G} = (U, H)$*, je takový sled, který obsahuje každý uzel nejvýše jednou.*

Definice 4. ***Souvislý graf*** *je takový graf, mezi jehož každými dvěma uzly existuje cesta.*

Definice 5. ***Komponenta grafu*** *je každý jeho maximální podgraf. Podgraf daného grafu je maximální, pokud jej už nelze zvětšit přidáním dalších hran, nebo uzlů z výchozího grafu tak, aby byl stále souvislý.*



Obrázek 2. Příklad komponenty grafu

2.2. Toky v sítích

Definice 1. ***Síť*** *je uspořádaná čtveřice* $S = (\overrightarrow{G}, z, s, k)$*, kde platí:*

- $\overrightarrow{G}$ *je orientovaný graf* $\overrightarrow{G} = (H, U)$
- z*,* s *jsou dva uzly sítě, nazývané zdroj a stok, pro které platí:* $z \neq s; z, s \in U$
- k *je funkce zvaná kapacita hrany, která přiřazuje každé hraně nezáporné reálné číslo* $(k : H(\overrightarrow{G}) \rightarrow \mathbb{R}_0+)$

Definice 2. ***Tok v síti*** $S = (\overrightarrow{G}, z, s, k)$ *je funkce* $t : H(\overrightarrow{G}) \rightarrow \mathbb{R}_0+$*, kde platí:*

- *Pro každou hranu* h *platí, že tok danou hranou nemůže být větší než je kapacita této hrany* $\forall h \in H : t(h) \leq k(h)$
- *Pro všechny uzly sítě, kromě zdroje a stoku platí, že součet toků na hranách vstupujících do uzlu se musí rovnat součtu toků na hranách vycházejících z uzlu.* $\forall u \in U, u \neq z, s : \sum_{h \rightarrow u} t(h) = \sum_{h \leftarrow u} t(h)$

Definice 3. ***Velikost toku*** t *nazýváme číslo* $|t| = \sum_{h \rightarrow z} t(h) - \sum_{h \leftarrow z} t(h)$*. Tok* t *se nazývá maximální, jestliže pro libovolný jiný tok* t_1 *v síti platí* $|t_1| \leq |t|$*.*

Poznámka 1. *Tok a kapacita hran v síti budou zjednodušeně zapisovány ve formátu T/K, kde T je hodnota toku na hraně a K je kapacita hrany.*

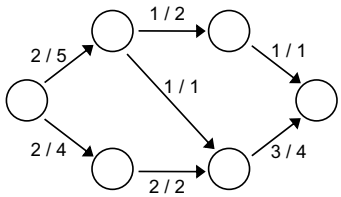
Definice 4. *Hrany jsou děleny na* ***nasycené a nenasycené****:*

- *Nasycená hrana je taková, pro kterou platí t(h)=k(h)*
- *Nenasycená je taková, pro kterou platí* $0 \leq t(h) < k(h)$

Definice 5. ***Řez v síti*** $S = (\overrightarrow{G}, z, s, k)$ *je podmnožina hran* $R \subseteq H(\overrightarrow{G})$ *taková, že po odebrání hran* R *z grafu* $\overrightarrow{G}$ *nezůstane žádná orientovaná cesta ze zdroje do stoku (zdroj i stok se nacházejí v různých komponentách grafu).*

- ***Velikost řezu*** R *je součet kapacit hran daného řezu* $|R| = \sum_{h \in H} k(h)$
- ***Minimální řez*** *má nejmenší velikost ze všech možných řezů.*

Věta 1. *Velikost maximálního toku je rovna velikosti minimálního řezu.*



Obrázek 3. Příklad toku v síti

3. Algoritmy

První algoritmus pro řešení problému maximálního toku v síti byl vytvořen v 2. polovině 20. století a nazývá se Ford-Fulkersonův algoritmus, podle jeho autorů. Algoritmus je založen na hledání vylepšujících cest a nezaručuje nalezení maximálního toku v případě iracionálních kapacit. Pokud se upraví Ford-Fulkersonův algoritmus tak, že se zvolí nejkratší vylepšující cesta (s nejmenším počtem hran), dojde ke zlepšení časové složitosti algoritmu. Tato myšlenka byla uvedena již Fordem a Fulkersonem jako heuristika.

Heuristiku vyhledávání nejkratší vylepšující cesty analyzoval ruský matematik Dinic. Takto upravený algoritmus je nazýván Dinicův algoritmus a využívá vrstevnatou (čistou) síť a blokující tok. Nezávisle na tom provedli slabší analýzu Edmonds a Karp, proto bývá algoritmus někdy nazýván po nich.

S dalším vylepšením tohoto algoritmu přišli Indové Malhotra, Kumar a Maheshawari, kteří vymysleli efektivnější algoritmus, jak nalézt blokující tok v čisté síti a tak zlepšili čas Dinicova algoritmu. Tento algoritmus je nazýván algoritmem tří Indů. Později přišel s jinou metodou Goldberg. Jeho algoritmus je nazývaný push-relabel a funguje na opačném principu než algoritmy vylepšujících cest.

Zde jsou uvedeny pojmy se kterými se bude při představování jednotlivých algoritmů pracovat:

Definice 1. ***Rezerva hrany*** *je rozdíl její kapacity a aktuálníto toku touto hranou.*

Definice 2. ***Vylepšující cesta*** *je cesta ze zdroje do stoku, jejíž všechny hrany mají kladnou rezervu.*

Definice 3. ***Blokující tok*** *je takový, že každá orientovaná cesta ze zdroje do spotřebiče obsahuje alespoň jednu nasycenou hranu.*

Definice 4. ***Vrstevnatá síť*** *obsahuje jen ty hrany sítě, které leží na nejkratší cestě ze zdroje do stoku a uzly jsou rozdělené do vrstev podle vzdálenosti od zdroje.*

3.1. Algoritmy vylepšujících cest

Při realizaci algoritmů vylepšujících cest většinou začínáme s tokem o velikosti 0 a hledáme vylepšující cestu. Podle této cesty tok vylepšíme a opakujeme vyhledávání. Tok je maximální, když už nelze nalézt žádnou další vylepšující cestu ve zbytkové síti a algoritmus v tomto bodě končí.

3.1.1. Ford-Fulkersonův algoritmus

Ford-Fulkersonův algoritmus hledá jakoukoliv vylepšující cestu a nezabývá se způsobem, jak tuto cestu najít. Hledání cesty lze například implementovat prohledáváním do hloubky.

Pokud jsou kapacity hran celočíselné, v každém kroku stoupne velikost toku minimálně o 1. Pokud jsou kapacity hran racionální, vynásobením nejmenším společným jmenovatelem úlohu převedeme na předchozí případ. Algoritmus je v těchto případech konečný. Ovšem pro iracionální kapacity hran algoritmus konečný není a může dojít k zacyklení.

```
Ford-Fulkerson:
t=0;
while existuje vylepšující cesta C ze zdroje do stoku;
          vylepši tok t podél cesty C;
return t;
```

Časová složitost Ford-Fulkersonova algoritmu je závislá na způsobu implementace a její určení muže být složitá úloha. Asymptotická časová složitost algoritmu je O (NM2), paměťová O (N+M), kde N je počet vrcholu a M je počet hran.

3.1.2. Dinicův algoritmus

V případě hledání nejkratších vylepšujících cest dojde k dramatickému zlepšení časové složitosti Ford-Fulkersonova algoritmu. Tato myšlenka byla uvedena v práci Forda a Fulkersona jako heuristika. Analýzu této heuristiky provedl jako první ruský matematik **Dinic** v roce 1970. Nezávisle na tom byla publikována i analýza **Edmondse a Karpa** v roce 1972, tato analýza byla první anglicky publikovaná analýza, proto je často algoritmus označován jako Edmons-Karpův. V této práci bude algoritmus označován jako Dinicův, neboť Dinic přišel s další optimalizací algoritmu – využívá vrstevnatou síť a blokující tok.

Hledání vylepšující cesty a nejkratší vylepšující cesty lze provést průchodem do šířky a náročnost je tedy téměř srovnatelná.

Hledání nejkratší vylepšující cesty probíhá ve třech základních krocích:

- Pro každou hranu v síti s tokem f, přidáme hranu opačnou. Kapacita této hrany se bude rovnat aktuálnímu toku v původní hraně.
- Vytvoříme pomocnou **síť rezerv** a tím dojde ke zjednodušení hledání vylepšující cesty. Při vytváření sítě smažeme hrany v protisměru a násobné

orientované hrany. Dále se vypočítá rezerva hrany jako rozdíl kapacity a aktuálního toku na hraně. V případě, že existují hrany s nulovou kapacitou, odstraníme je ze sítě, protože jimi již nemůže nic protéct.

- Vytvoříme **čistou síť**, která bude obsahovat pouze hrany na nejkratších cestách ze zdroje do stoku. Nejčastější implementace je pomocí průchodu do šířky, kde je síť na konci průchodu rozdělena do vrstev podle vzdálenosti od zdroje – vrstevnatá síť.

Délka orientované cesty znamená počet hran, kterými musíme projít, abychom se dostali ze zdroje do stoku. Po vylepšení toku podél nejkratší vylepšující cesty nemůže nastat situace, kdy klesne délka nejkratší cesty. Při vylepšení toku podél této cesty dojde k nasycení některých hran, jejich rezerva bude nulová, a proto se odstraní ze sítě rezerv.

Čistá síť je acyklický orientovaný graf, kde jsou kapacity hran nahrazeny jejich rezervou, a lze v ní hledat tok. V čisté síti je blokující tok, pokud každá orientovaná cesta ze zdroje do stoku obsahuje alespoň jednu nasycenou hranu. Blokující tok nemusí být maximální a je roven hodnotě toku, o který zvětšíme aktuální tok během jedné fáze algoritmu.

```
Dinic:
počáteční tok t = libovolný tok;
while (true);
        sestrojení sítě rezerv;
        nalezení nejkratší cesty ze zdroje do stoku;
        if (neexistuje nejkratší cesta) break;
        pročištění sítě;
        nalezení blokujícího toku v čisté síti;
        tok tb = 0;
        while (existuje cesta ze zdroje do stoku);
                naleznutí cesty ze zdroje do stoku;
                tb += nejmenší rezerva na cestě;
                dočištění sítě;
        vylepšení toku t += tb;
return t;
```

3.1.3. Algoritmus tří Indů

V roce 1978 vymysleli Indové **Malhotra, Kumar a Maheshawari** efektivnější algoritmus pro hledání blokujícího toku v síti, jejich algoritmus je nazýván algoritmus tří Indů.

Algoritmus upravuje hledání blokujícího toku v čisté síti a vychází z Dinicova algoritmu. V tomto algoritmu se pracuje s tzv. **rezervou uzlu**. Tato kapacita se vypočítá tak, že sečteme kapacity hran vstupujících do uzlu, poté sečteme

kapacity hran vystupujících z uzlu a určíme jejich minimum. U zdroje uvažujeme pouze hrany vystupující a u stoku hrany vstupující.

V každé iteraci algoritmu nalezneme uzel s nejnižší rezervou a zvýšíme tok tak, aby se tato rezerva vynulovala. U každého uzlu využíváme tzv. plán, což je velikost toku, který potřebujeme dostat ze zdroje do daného vrcholu.

Takto upravený algoritmus ukazuje na náročnějších případech výrazné zlepšení oproti předchozím algoritmům.

3.2. Push – relabel algoritmus

Push-Relabel algoritmus vymyslel v roce 1985 **Goldberg**, proto je občas algoritmus označován jako Goldbergův. V algoritmech vylepšujících cest se tok postupně vypočítává podél vylepšujících cest, tento algoritmus je založen na myšlence, zda lze poslat tok podél hrany naráz. Algoritmus používá dvě základní operace:

- Protlačení toku po hraně (push)
- Zvýšení výšky uzlu (relabel)

Základní myšlenka algoritmu je, že na začátku protlačíme ze zdroje co největší tok do sousedních uzlů, dále se budeme snažit protlačit přebytek toku v uzlu do sousedních uzlů směrem ke stoku. Při protlačování nesmí být překročena kapacita hran a může probíhat pouze po hranách s nenulovou rezervou. V případě, že tok nelze protlačit dál, vracíme ho zpět do zdroje.

```
Push-Relabel:
poslat co nejvyšší tok ze zdroje;
seznam uzlů = všechny uzly sítě, kromě zdroje a stoku;
foreach (uzel v seznamu uzlů);
     if (přebytek uzlu > 0)
          if (existuje soused pro přemístění přebytku)
               přemístit přebytek k sousedovi;
          else:
               zvednout uzel;
     if se změnila výška uzlu;
          přemístit uzel na začátek seznamu;
          nové prohledávání seznamu od začátku;
```

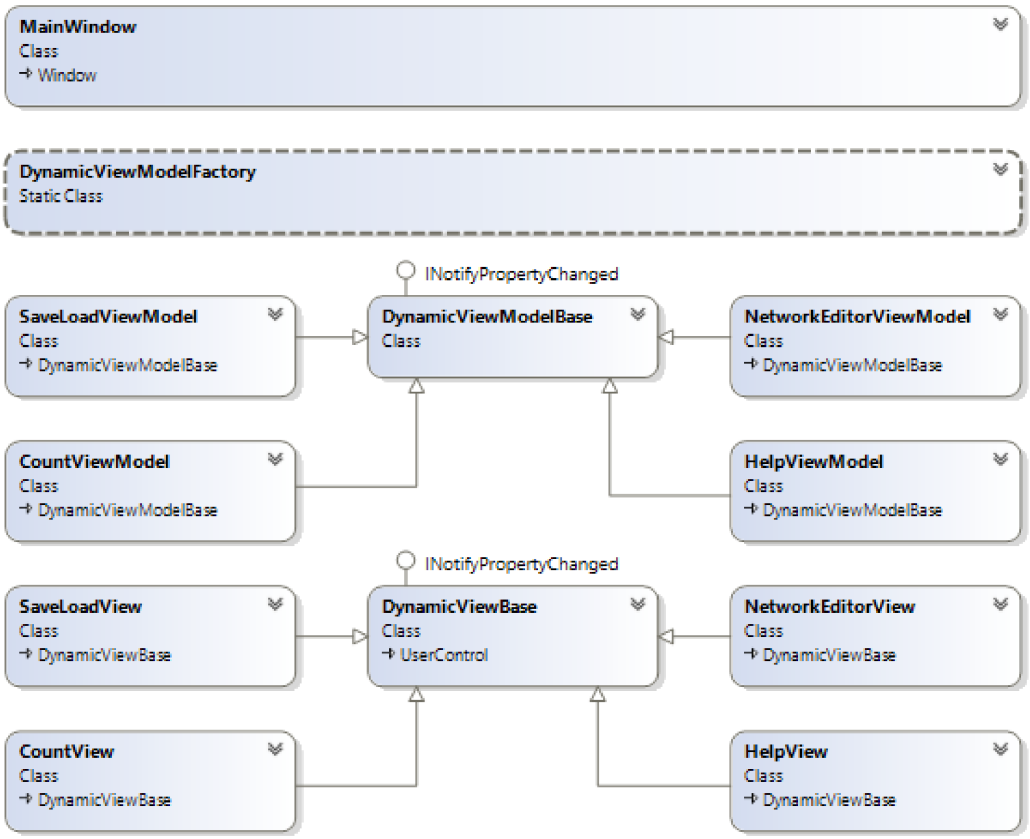
Existuje několik různých implementací push-relabel algoritmu. Jedna z implementací využívá síť rezerv, stejně jako Dinicův algoritmus. Tento algoritmus je vhodný k použití u rozsáhlých sítí, u menších sítí může být rychlejší Dinicův algoritmus.

4. Programové zpracování

4.1. Architektura aplikace

V této části bude vysvětlena struktura aplikace a popis řešení jednotlivých problémů. Struktura aplikace je rozdělena do dvou vrstev, které slouží k oddělení prezentační logiky od věcné a datové. Tímto je zvýšena udržovatelnost a zlepšena možnost zapracování změn.

K vyvoření aplikace je využit jazyk C#, .NET framework 4.5 a pro grafické zpracování technologie Windows Presentation Foundation (WPF). Aplikace byla vytvořena na operačním systému Windows 8.1 za použití integrovaného prostředí Visual Studio 2012, Visual Studio 2013 a pro grafické zpracování byl použit Blend for Visual Studio 2012.



Obrázek 4. Class diagram - struktura aplikace

4.1.1. Třída DynamicViewBase

Třída DynamicViewBase představuje prezentační vrstvu aplikace, která slouží k vytvoření grafického rozhraní pro interakci s uživatelem. Je vytvořena pomocí technologie WPF, která je součásti .NET frameworku. Tato technologie využívá značkovací jazyk XAML, který umožňuje oddělení funkčnosti a vzhledu aplikace. Nejdůležitější metody této třídy jsou:

- `protected void RaisePropertyChanged(string name)` - vyvolá změněnu vlastnosti
- `public void SetController<T>(DynamicViewModelBase viewModel) where T : DynamicViewModelBase` - nastavení ViewModelu pro dané View

Třída CountView je potomkem třídy DynamicViewBase a slouží k obsluze výpočtu. Volá metody reágující na interakci uživatele. Obsahuje dvě základní metody:

- `private void StartCountOnClick(object sender, RoutedEventArgs e)` - vyvolá metodu zahajující výpočet maximálního toku a minimálního řezu v síti
- `private void CheckStartFlowClick(object sender, RoutedEventArgs e)` - vyvolá metodu pro nastavení a kontrolu počátečního toku v síti

Aby bylo možno při zadání počátečního toku vložit pouze číselné hodnoty do TextBoxu je zde využita metoda `private void NumberValidationTextBox(object sender, TextCompositionEventArgs e)`. Tato metoda obsahuje regulární výraz, který tento problém ošetřuje.

Třída HelpView je potomkem třídy DynamicViewBase a nejsou zde obsaženy žádné důležité metody ani vlastnosti. Slouží pouze k zobrazení textové nápovědy k aplikaci.

Třída NetworkEditorView je potomkem třídy DynamicViewBase a slouží k vytvoření a editaci sítě. Umožňuje vkládat, editovat a mazat uzly i hrany, zadávat název a typ uzlu, zadávat kapacitu hran a zvolit zdroj a stok sítě. Pro všechny tyto akce volá metody na základě událostí vyvolaných uživatelem. Využívá následující události:

- `private new void MouseMove(object sender, MouseEventArgs e)` - volá metodu obsluhující drag and drop, konkétně část pohybování objektem

- `private void DesigningCanvasOnClick(object sender, MouseButtonEventArgs e)` - volá metodu obsluhující událost kliknutí do canvasu
- `private void SaveButtonOnClick(object sender, RoutedEventArgs e)` - zobrazí obrazovku pro uložení sítě
- `private void LoadButtonOnClick(object sender, RoutedEventArgs e)` - zobrazí obrazovku pro načtení sítě
- `private void ButtonOnClick(object sender, RoutedEventArgs e)` - nastaví aktuální zvolené tlačítko
- `private void DeleteAllButtonOnClick(object sender, RoutedEventArgs e)` - vymaže obsah canvasu

Stejně jako v CountView je zde využita validace vstupu při zadávání kapacity hrany.

Třída SaveLoadView je potomkem třídy DynamicViewBase a slouží k uložení a načtení sítě, popřípadě její odstranění z databáze. Události obsluhující tuto třídu jsou:

- `private void SaveOnClick(object sender, RoutedEventArgs e)` - volá metodu pro uložení sítě do databáze pod zvoleným názvem
- `private void SaveAsOnClick(object sender, RoutedEventArgs e)` - volá metodu přepisující zvolenou síť v seznamu na aktuální síť a ukládá ji do databáze
- `private void DeleteOnClick(object sender, RoutedEventArgs e)` - volá metodu odstraňující zvolenou síť z databáze
- `private void LoadOnClick(object sender, RoutedEventArgs e)` - volá metodu načítající zvolenou síť z databáze

4.1.2. Třída DynamicViewModelBase

Tato vrstva slučuje věcnou a datovou logiku aplikace. Reaguje na události pocházející z prezentační vrstvy aplikace a obsahuje metody pro jejich obsluhu. Třída samotná obsahuje nástedující metody:

- `protected void RaisePropertyChanged(string name)`, `protected void RaisePropertyChanged<T>(Expression<Func<T>> propertyExpression)` - vyvolá změnu vlastnosti
- `public void ShowView(MainWindow mainWindow)` - zobrazení daného View v hlavním okně

Třída CountViewModel je potomkem třídy DynamicViewBaseModel a vytváří instanci třídy Dinic, obsahující algoritmus pro výpočet maximálního toku a minimálního řezu. Obsahuje dvě základní metody:

- `public void StartCount()` - metoda zahajující výpočet maximálního toku a minimálního řezu v síti
- `public void CheckStartFlow(double startFlow)` - metoda pro nastavení a kontrolu počátečního toku v síti

Třída NetworkEditorView je potomkem třídy DynamicViewBaseModel a slouží k vytvoření a editaci sítě. Na základě metod volaných ze třídy NetworkEditorView vytváří, mění a maže objekty sítě a mění jejich vlastnosti. Obsahuje následující metody pro vytvoření a smazání objektů:

- `private void CrateNewNode(Point point)` - vytvoří uzel na základě předaného bodu
- `public bool CreateNewEdge(Node node)` - vytvoří hranu vedoucí z aktuálního uzlu v síti do zvoleného uzlu
- `public void DeleteAll()` - vymaže všechny uzly a hrany z aktuální sítě

Následující metody mění vlastnost objektů v síti:

- `public void ChangeNodeSize()` - změna velikosti uzlu
- `public void ChangeEdgeThickness()` - změna tloušťky hrany

Nejdůležitější vlastnosti této třídy jsou:

- `public Node CurrentNode` - reprezentuje aktuální zvolený uzel v síti
- `public Edge CurrentEdge` - reprezentuje aktuální zvolenou hranu v síti
- `public double NodeSize` - vrací aktuální velikost uzlu
- `public double EdgeThickness` - vrací aktuální tlošťku hrany

Třída SaveLoadView je potomkem třídy DynamicViewBaseModel a obsluhuje ukládání a načítání dat z databáze. Obsahuje tyto metody:

- `public void Save(string text)` - uložení sítě do databáze pod zvoleným názvem
- `public void SaveAs()` - přepsání zvolené sítě v seznamu na aktuální síť a uložení do databáze
- `public void Delete()` - odstranění zvolené sítě z databáze
- `public void Load()` - načtení zvolené sítě z databáze

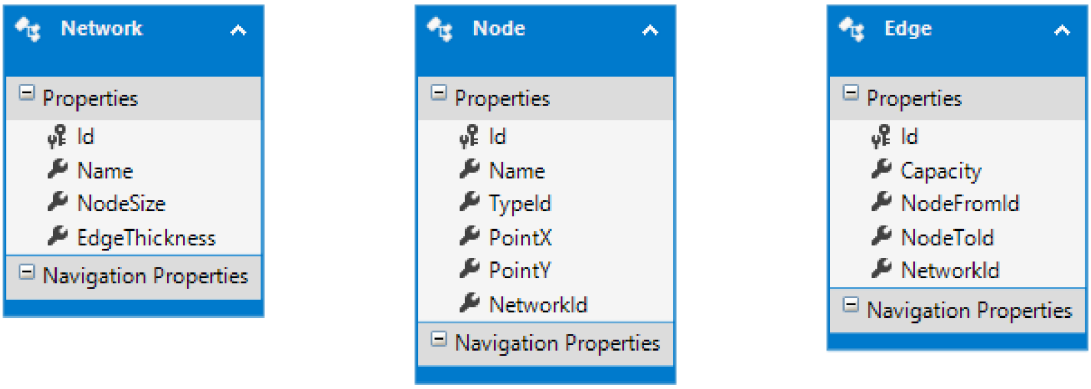
4.1.3. Třída DynamicViewModelFactory

Každý objekt v aplikaci je vytvářen pomocí **Factory Method** vzoru. Tento vzor umožňuje hlídat množství vytvořených objektů a ošetřuje přístup k nim. Při vytváření objektu se propojí zvolené View typu DynamicViewBase a ViewModel typu DynamicViewBaseModel. Tato třída obsahuje jednu základní metodu:

- `public static ControllerType CreateViewController<ControllerType, DynamicViewType>() where DynamicViewType : DynamicViewBase where ControllerType : DynamicViewModelBase` - metoda zajišťující vytvoření objektu

4.1.4. Datová vrstva

Data aplikace jsou uloženy v lokální SQL databázi. V aplikaci je využit **Entity Framework**, který umožňuje pracovat s relačními daty pomocí objektů. ADO.NET Entity Data Model umožňuje vytvořit koncepční model z existující databáze a graficky jej zobrazit a upravit.



Obrázek 5. Struktura databáze

Tabulky obsažené v databázi jsou zobrazeny na obrázku 2. V databázi jsou dále vytvořeny procedury zajišťující vkládání, editaci a smazání řádku v dané tabulce. Obsluha těchto procedur v aplikaci je vygenerována z databáze pomocí Entity Data Modelu.
Procedury pro vložení nového řádku:

- Insert_Network
- Insert_Node
- Insert_Edge

Procedury pro editaci řádku:

- Update_Network
- Update_Node
- Update_Edge

Procedury pro smazání řádku:

- Delete_Network
- Delete_Node
- Delete_Edge

4.2. Popis řešení

4.2.1. Třída Node

Třída Node je logickou i grafickou reprezentací uzlu. V následující části budou popsány nejdůležitější metody a parametry v této třídě.
Metody pro práci s databází:

- `public int Insert(int networkId)` - vložení uzlu
- `public void Update(Node node, int networkId)`, `public void Update()` - editace uzlu
- `public void Delete()` - smazání uzlu

Metody pro obsluhu Drag and Drop:

- `private void MouseLeftButtonDown(object sender, MouseButtonEventArgs e)` - inicializace uzlu pro drag and drop
- `private void PreviewMouseLeftButtonUp(object sender, MouseButtonEventArgs e)` - ukončení drag and drop

Ostatní metody:

- `private void NewNodeOnMouseUp(object sender, MouseButtonEventArgs mouseButtonEventArgs)` - slouží k odstranění uzlu a nastavení typu uzlu na základě aktuálního stavu aplikace a k označení / odoznačení aktuálního uzlu v síti
- `public void SetType(NodeType type)` - slouží k nastavení typu uzlu a zajišťuje aby v síti byl obsažen pouze jeden zdroj a jeden stok.

Základní vlastnosti třídy, ukládané do databáze:

- `public string Name` - jméno uzlu
- `public NodeType Type` - typ uzlu
- `public double PoinX, public double PointY` - body pro vytvoření grafické reprezentace uzlu

Ostatní vlastnosti třídy:

- `public NodeControl NodeControl` - grafická reprezentace uzlu typu UserControl
- `public int Distance` - vlastnost sloužící k sestrojení vrstevnaté sítě

4.2.2. Třída Edge

Třída Edge je logickou i grafickou reprezentací hrany. V následující části budou popsány nejdůležitější metody a parametry v této třídě.
Metody pro práci s databází:

- `public int Insert(int networkId)` - vložení hrany
- `public void Update(Edge edge, int networkId)`, `public void Update()` - editace hrany
- `public void Delete()` - smazání hrany

Metody pro vykreslení hrany:

- `public void GetEdge(Point from, Point to, double thickness)` - slouží k získání grafické reprezentace hrany
- `private void SetEdgeColor(string color)` - změna barvy hrany
- `public Point GetPointFrom(Point from, Point to, double size)`, `public Point GetPointTo(Point from, Point to, double size)` - získá koncový bod hrany na základě šířky hrany
- `public Polygon GetArrow(Point from, Point to, double size)` - metoda pro vykreslení šipky na konci hrany, která využívá následující metody:
  - `public Point GetPointVector(Point p1, Point centerPoint, int rot, double size)` - získání bodu polygonu
  - `public Point GetCenterPoint(Point p1, Point p2)` - získání středového bodu vektoru

Ostatní metody:

- `private void EdgePathOnMouseUp(object sender, MouseButtonEventArgs mouseButtonEventArgs)` - slouží k odstranění hrany a k označení / odoznačení aktuální hrany v síti
- `private void EdgePathOnMouseEnter(object sender, MouseEventArgs mouseEventArgs)`, `EdgePathOnMouseLeave(object sender, MouseEventArgs mouseEventArgs)` - události pro zavolání metody na přebarvení hrany
- `SetEdgeColor(string color)` - metoda pro nastavení barvy hrany

Základní vlastnosti třídy, ukládané do databáze:

- `public Node NodeFrom, public Node NodeTo` - koncové uzly hrany
- `public double Capacity` - kapacita hrany

Ostatní vlastnosti třídy:

- `public Grid EdgePath` - grafická reprezentace hrany typu Grid
- `public double Flow` - vlastnost sloužící k výpočtu maximálního toku v síti
- `public string LabelText` - text zobrazen u dané hrany reprezentující tok / kapacita

4.2.3. Třída Network

Třída Network je logickou reprezentací sítě. V následující části budou popsány nejdůležitější metody a parametry v této třídě.
Metody pro práci s databází:

- `public int Insert()` - vložení sítě
- `public void Update()` - editace sítě
- `public void Delete()` - smazání sítě

Základní vlastnosti třídy, ukládané do databáze:

- `public string Name` - jméno sítě
- `public Nullable<double> NodeSize` - velikost uzlu
- `public Nullable<double> EdgeThickness` - šířka hrany

4.2.4. Třída Dinic

Třída Dinic obsahuje Dinicův algoritmus pro výpočet maximálního toku a minimálního řezu v síti. Slouží také pro nastavení a kontrolu počátečního toku v síti. Základní metody pro obsluhu třídy jsou:

- `public double DinicAlgorithm(Network network, double startFlow)` - spuštění výpočtu maximálního toku v síti
- `public bool SetStartFlow(Network network, double startFlow)` - nastavení a kontrola počátečního toku
- `public void GetMinimumCut(Network network)` - nalezení minimálního řezu v síti

Jednotlivé části algoritmu:

- `private double FindBlockFlow(Network network)` - nalezení blokujícího toku v čisté síti
- `private Network MakeResidualGraph(Network network)` - vyvoření reziduálního grafu
- `private void CleanNetwork(Network network)` - očištění sítě
- `private void SetNodesDistance(Network network)`, `private void GetNodesDistance(Network network, Node node, int distance)` - nastavení vzdálenosti uzlu od zdroje

5. Uživatelská příručka

Tato část popisuje funkce a chování aplikace z pohledu uživatele.

5.1. Instalace a spuštění aplikace

Předpokladem pro správné nainstalování aplikace je operační systém Windows 8 s .NET frameworkem 4.5.

Doporučená konfigurace počítače:

- **Procesor**: Intel® Core® i5 nebo Intel® Core® i7
- **Operační systém**: Windows 8 a Windows 8.1
- **RAM paměť**: minimálně 2GB, doporučeno 4GB
- **.NET framework**: verze 4.5
- v počítači **nesmí** být nainstalován SQL Server 2008

Po spuštění aplikace se zobrazí úvodní okno s editorem sítě.

5.2. Vytvoření sítě

5.2.1. Přidání a editace uzlu

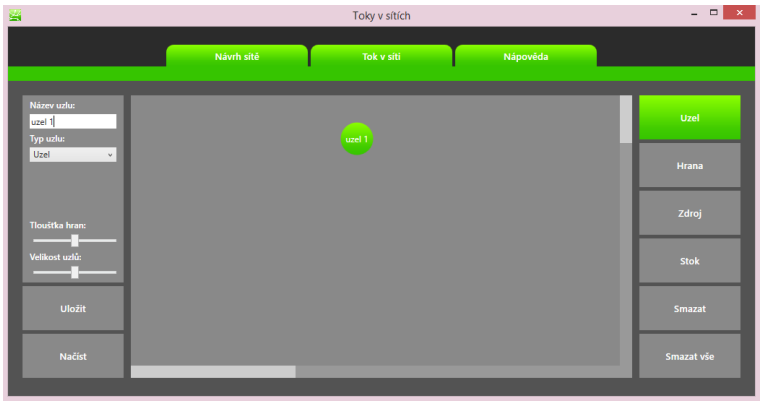
Pro **přidání uzlu** se zvolí v právém panelu tlačítko **"Uzel"**. Kliknutím na tlačítko dojde k jeho aktivaci a kliknutím do editoru se přidá uzel. Po přidání zůstává tlačítko "Uzel"aktivní a lze proto přidat libovolný počet uzlů. V případě, že chceme ukončit zadávání uzlů, klikneme na tlačítko "Uzel". Tím dojde k jeho deaktivaci. Druhým způsobem, pro ukončení zadávání uzlů, je zvolení jiného tlačítka v pravém panelu (mimo tlačítka "Smazat vše").

Přejmenování uzlu lze provést kliknutím na uzel, tím dojde k jeho označení, a zadáním libovolného názvu do textového pole "Název uzlu". **Změnu typu uzlu** lze provést více způsoby:

- označením uzlu a změnou typu v levém panelu v číselníku "Typ uzlu"
- zvolením tlačítka "Zdroj"/ "Stok"a kliknutím na uzel, u kterého chceme typ změnit na zdroj / stok

Při změně typu uzlu na zdroj / stok se prohledají všechny uzly v síti a v případě, že je nějaký uzel již zvolen jako zdroj / stok, dojde ke změně jeho typu na uzel a typ zvoleného uzlu se změní na zdroj / stok.

Změna umístění uzlu se provádí metodou "chytni a táhni"- klikneme na uzel a za držení levého tlačítka myši a pohybu se objekt přesunuje, zároveň se přesunují i všechny hrany obsahující tento uzel.



Obrázek 6. Vytvoření sítě - přidání uzlu

5.2.2. Přidání a editace hrany

Pro **přidání hrany** se zvolí v právém panelu tlačítko **"Hrana"**. Kliknutím na tlačítko dojde k jeho aktivaci stejně jako u tlačítka "Uzel". Přidání hrany je složeno ze dvou kroků:

- zvolíme uzel odkud hrana vede - v případě, že je označen jiný uzel, než který chceme zvolit, kliknutím na něj jej deaktivujeme a můžeme zvolit požadovaný uzel
- zvolíme uzel kam hrana vede

Tím dojde k vytvoření hrany. Po přidání zůstává tlačítko "Hrana"aktivní a jako aktuální uzel je označen uzel, kde nově vytvořená hrana končí, můžeme tedy pokračovat v přidávání hran. Ukončení zadávání hran se provede stejně jako u zadávání uzlů.

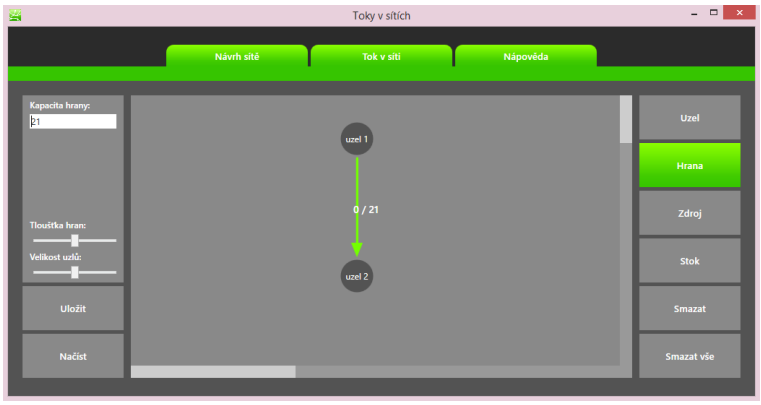
Změnu kapacity hrany lze provést kliknutím na hranu, tím dojde k jejímu označení, a zadáním hodnoty do textového pole "Kapacita hrany". Zadávaný text musí být celé číslo, jakýkoliv jiný vstup je ignorován.

5.2.3. Velikost uzlu a šířka hrany

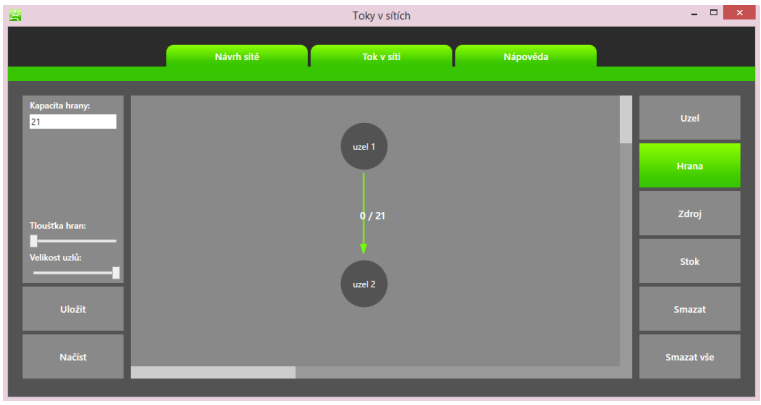
V aplikaci lze změnit velikost uzlu i šířku hrany pomocí posuvníků v levém panelu. Táhnutím doleva se velikost / šířka zmenšuje a doprava zvětšuje. Pro **změnu velikosti uzlu** nastavujeme hodnotu posuvníku **"Velikost uzlů"**, pro **změnu šířky hrany** nastavujeme hodnotu posuvníku **"Tloušťka hran"**.

5.2.4. Mazání objektů z editoru

Ke **smazání objektů** lze požít dvě metody:



Obrázek 7. Vytvoření sítě - přidání hrany



Obrázek 8. Vytvoření sítě - změna velikosti uzlu a šířky hran

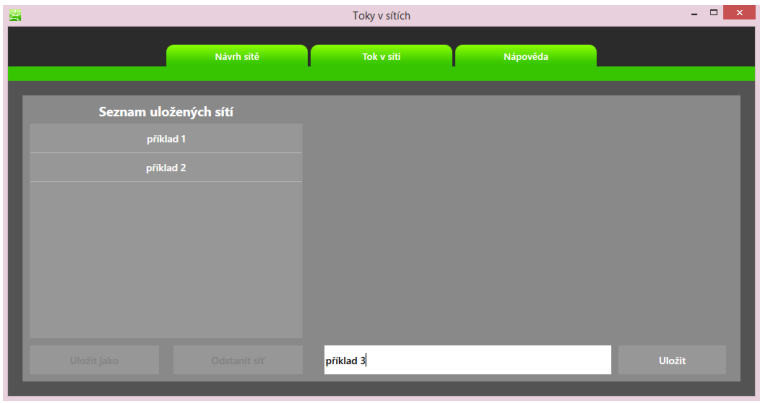
- zvolíme tlačítko **"Smazat"**, a následně klikneme na objekt, který chceme smazat
- zvolíme tlačítko **"Smazat vše"**, v tomto případě dojde ke smazání všech objektů v editoru

V případě, že mažeme uzel, dojde i k odstranění všech hran, obsahujících tento uzel.

5.3. Uložení a načtení sítě

Pro **uložení** slouží v návrhu sítě tlačítko **"Uložit"**. Po jeho stisknutí dojde k navigaci na obrazovku se seznamem uložených sítí. Jsou dvě možnosti, jak síť uložit:

- vytvoření nové sítě v databázi - zadáme název sítě do textového pole a stiskneme **"Uložit"**
- přepsání existující sítě v databázi - zvolíme síť v seznamu uložených sítí a stiskneme **"Uložit jako"**, tím dojde k přepsání zvolené sítě na aktuální síť



Obrázek 9. Uložení sítě

Při ukládání mohou nastat tyto stavy:

- uložení proběhne bez problémů
- v databázi existuje zadaný název sítě - uložení neproběhne
- není zadaný název sítě - uložení neproběhne

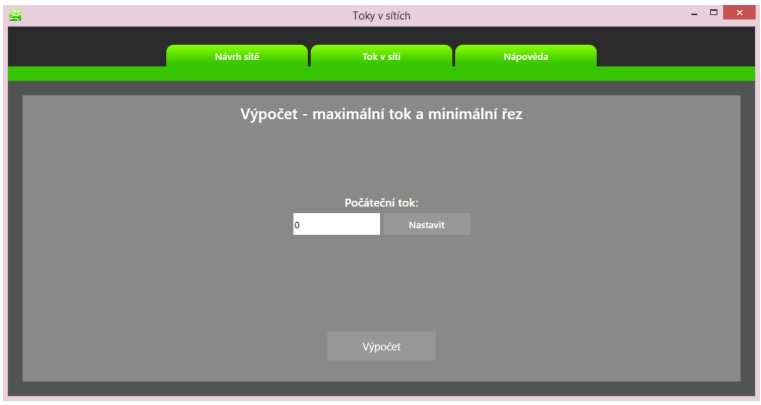
Pro **načtení** slouží v návrhu sítě tlačítko **"Načíst"**. Po jeho stisknutí dojde k navigaci na obrazovku se seznamem uložených sítí. Zde zvolíme požadovanou síť a stiskneme "Načíst".

K **odstranění sítě z databáze** slouží tlačítko **"Odstranit síť"**, které je umístěno pod seznamem uložených sítí. Nejdříve zvolíme síť, kterou chceme odstranit a stiskneme tlačítko "Odstranit síť".

5.4. Výpočet

Pro výpočet maximálního toku a minimálního řezu zvolíme v horním menu položku **"Tok v síti"**. **Počáteční tok** v síti se nastaví vložením hodnoty do textového pole "Počáteční tok"a stisknutím tlačítka nastavit. Zde může aplikace přejít do těchto stavů:

- počáteční tok je v pořádku
- počáteční tok nelze pro síť nastavit

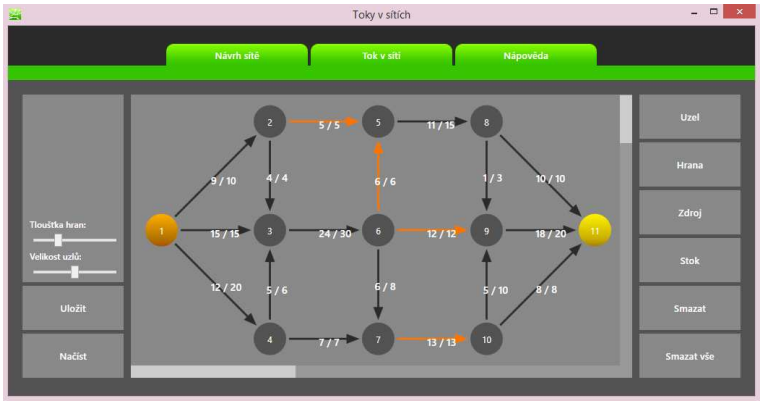


Obrázek 10. Výpočet

- síť nemá nastaven zdroj nebo stok

Pro **výpočet maximálního toku a minimálního řezu** slouží tlačítko "Výpočet". Po jeho stisknutí se může aplikace dostat do těchto stavů:

- výpočet proběhl v pořádku, zobrazí se dialog s výsledkem
- síť nemá nastaven zdroj nebo stok



Obrázek 11. Výpočet - zobrazení v síti

Pro **zobrazení maximálního toku a minimálního řezu na síti** se po provedení výpočtu stačí přepnout v hlavním menu na obrazovku "Návrh sítě".

Závěr

Cílem bakalářské práce bylo vytvořit aplikaci pro podporu řešení přepravních a dalších úloh, které lze modelovat pomocí toků v sítích za pomoci vyhledání maximálního toku a minimálního řezu v dané síti. K vytvoření aplikace bylo využito vývojové prostředí Microsoft Visual Studio 2012, Visual Studio 2013, Blend for Visual Studio 2012 a znalosti objektově orientovaného programování. Výslednou aplikaci lze využít k výukovým účelům.

Conclusions

The purpose of this thesis was to develop an application for support solving transport solutions and other tasks that can be modeled using network flow with maximum flow and minimum cut support. Application was created under development environment Microsoft Visual Studio 2012, Visual Studio 2013 Blend for Visual Studio 2012 and knowledge of object-oriented programming. This application can be used for learning purposes.

Reference

[1] Josef Kolář, Olga Štěpánková, Michal Chytil. *Logika, algebra a grafy.* SNTL, 1989.

[2] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein. *Introduction to Algorithms.* The MIT Press, 2009.

[3] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *LNetwork Flows: Theory, Algorithms, and Applications.* Prentice Hall, 1993.

[4] M. Mareš. *Krajinou grafových algoritmů.* ITI Series, Prague, 2008. http://mj.ucw.cz/vyuka/ga/

[5] Network Flow http://www.cs.arizona.edu/classes/cs445/spring05/NetworkFlow.pdf.

[6] Microsoft MSDN http://msdn.microsoft.com.

[7] Wikipedia http://www.wikipedia.org/.

A. Obsah přiloženého CD

`bin/`
: Instalátor SETUP.EXE programu a program samotný, spustitelný přímo z CD/DVD. Adresář obsahuje i všechny potřebné knihovny a další soubory pro bezproblémové spuštění programu.

`doc/`
: Dokumentace práce ve formátu PDF, vytvořená dle závazného stylu KI PřF pro diplomové práce, včetně všech příloh, a všechny soubory nutné pro bezproblémové vygenerování PDF souboru dokumentace (v ZIP archivu), tj. zdrojový text dokumentace, vložené obrázky, apod.

`src/`
: Kompletní zdrojové texty programu se všemi potřebnými zdrojovými texty, knihovnami a dalšími soubory pro bezproblémové vytvoření spustitelných verzí programu (v ZIP archivu).

`readme.txt`
: Instrukce pro instalaci a spuštění programu, včetně požadavků pro jeho provoz.