VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

GAL Projekt

Artikulační body a mosty

Jiří Šotola (xsotol02),
Marek Dančo (xdanco00)

13. prosince 2024

Obsah

1 Úvod

Algoritmy jsou důležitou součástí jak v informatice, tak i v matematice, kde hrají zásadní roli při řešení problémů v teorii grafů. Algoritmus lze definovat jako systematický postup, který pomocí konečného počtu kroků vyřeší zadaný problém. V teorii grafů se tyto postupy používají k analýze vrcholů a hran, které modelují širokou škálu reálných situací, například počítačové sítě, dopravní infrastrukturu nebo sociální sítě.

V kontextu analýzy sítí jsou důležité pojmy artikulační body a mosty, které pomáhají určit zranitelné části grafu. Artikulační bod je takový vrchol, který při odstranění způsobí, že graf přestane být souvislý. Podobně most je hrana, která při odstranění rozdělí graf na více komponent. Tyto prvky jsou klíčové pro zajištění stability sítí, protože označují kritické části, které při selhání poruší integritu a můžou poškodit i celkovou funkčnost [2].

Využití těchto dvou konceptů je široké a používá se od návrhu odolných počítačových sítí přes analýzu spolehlivosti dopravních systémů až po detekci klíčových aktérů v sociálních sítích. Například v počítačových sítích mohou být mosty identifikovány jako kritické spoje, které by u výpadku rozdělil síť na nekomunikující části. V dopravě mohou artikulační body představovat klíčové uzly, u kterých by jejich uzavření by narušilo propojení mezi regiony. Tímto způsobem pomáhají algoritmy nejen analyzovat stávající systémy, ale také navrhovat odolnější struktury [6] [5].

2 Artikulační body a mosty

Artikulační body a mosty jsou důležité, protože označují prvky, které jsou nejvíce zranitelné. Při jejich odstranění se graf rozdělí na více souvislých komponent. Nalezení těchto prvků není však jisté, protože se v grafu nemusí vyskytnout. Grafy, které tyto body nemají, jsou tzv. 2-souvislé (obsahují 2-souvislé komponenty) [1]. Pro nalezení artikulační bodů a mostů je důležité použít efektivní algoritmus, který má dobrou časovou složitost. Všechny tyto algoritmy se zabývají neorientovanými grafy.

V následující části jsou vyobrazeny algoritmy pro hledání těchto prvků. Zobrazeny jsou jejich popis, efektivní algoritmus a jeho složitost. Algoritmus pro artikulační body je možné najít v sekci 2.1 a algoritmus pro mosty v sekci 2.2.

2.1 Artikulační body

Artikulační body (*articulation points*) jsou vrcholy v neorientovaném grafu, které při odstranění způsobí, že graf přestane být souvislý. Jinými slovy, artikulační bod je kritický uzel v grafu, který je nezbytný pro propojení jeho částí. Pokud graf obsahuje více souvislých komponent, může být odstraněním artikulačního bodu jedna z těchto částí oddělena od ostatních [2].

Algoritmus pro nalezení artikulační bodů se nazývá Tarjanův algoritmus pro hledání klíčových bodů (*Tarjan's Algorithm for Articulation Points*). Tento algoritmus je pojmenován po Robertu Tarjanovi, který ho vyvinul v roce 1972. Algoritmus je založen na prohledávání do hloubky (DFS). Algoritmus identifikuje artikulační body pomocí konceptů doby objevení a minimálního dosažitelného času. Následující odstavec zobrazuje jeho pseudokód [7].

```
AP(V, Adj):
for u in V do:
  color[u] = WHITE
  p[u] = null
  ap[u] = false
time = 0
for u in V do:
  if color[u] == WHITE then:
    AP-Visit(u)
```

```
AP-Visit(u):
color[u] = GRAY
children = 0
d[u] = low[u] = time = time + 1
for v in Adj[u] do:
  if color[v] == WHITE then:
    children = children + 1
    p[v] = u
    AP-Visit(v)
    low[u] = Min(low[u], low[v])
    if p[u] == null && children > 1 then:
      ap[u] = true
    if p[u] != null && low[v] >= d[u] then:
      ap[u] = true
  else if p[u] != v then:
    low[u] = Min(low[u], d[v])
color[u] = BLACK
```

Význam jednotlivých proměnných:

- d[u]: Čas první návštěvy uzlu u.
- low[u]: Nejmenší čas dosažitelný z uzlu u.
- ap[u]: Boolean hodnota označující, zda je uzel u artikulačním bodem.
- p[u]: Pole rodičů uzlů.
- color[u]: Stav uzlu (`WHITE`, `GRAY`, `BLACK`).

Algoritmus se skládá ze dvou hlavních částí: inicializační fáze `AP` a průchodu grafem pomocí DFS `AP-Visit`.

Inicializační fáze `AP` začíná inicializací, během kterého je každému vrcholu `WHITE` (nenavštíven). Pole rodičů ($p[u]$) je nastaveno na `null` (nemá předchůdce), a pole $ap[u]$ je inicializováno na `false`. Globální proměnná `time` je nastavena na nulu (čas průchodu). Následně algoritmus iteruje přes všechny vrcholy a spustí funkci `AP-Visit(u)` pro každý nenavštívený vrchol.

Funkce `AP-Visit(u)` označí vrchol u jako `GRAY` (právě zpracovávaný). Inicializuje počet dětí `children` na nulu a nastaví první návštěvu $d[u]$ i nejnižší dosažitelnou hodnotu low$[u]$ na aktuální hodnotu `time`, kterou následně inkrementuje. Poté algoritmus iteruje přes všechny sousedy v vrcholu u.

Pokud soused v nebyl dosud navštíven (`WHITE`), algoritmus zvýší počet dětí vrcholu u, nastaví $p[v] = u$, aby u bylo rodičem vrcholu v, a rekurzivně zavolá `AP-Visit(v)`. Po návratu z rekurze aktualizuje hodnotu low$[u]$ podle hodnoty low$[v]$. Následně se rozhodne, zda je vrchol u klíčovým bodem. Pokud u je kořen DFS stromu ($p[u] =$ null) a má více než jedno dítě (children > 1), pak je u klíčovým bodem. Pokud u není kořen ($p[u] \neq$ null) a low$[v] \geq d[u]$, pak u rovněž představuje klíčový bod, protože potomci v nejsou schopni dosáhnout žádného předka u mimo u samotného.

Pokud soused v byl již navštíven (`GRAY`) a není rodičem u, algoritmus aktualizuje low$[u]$ podle první návštěvy $d[v]$. Po zpracování všech sousedů označí vrchol u jako `BLACK` (plně zpracován).

Časová složitost tohoto algoritmu je $O(V + E)$, protože používá DFS a každý vrchol i hranu prochází jednou. Prostorová složitost je také $O(V + E)$, protože graf je obvykle reprezentován seznamem sousedů, a pomocné proměnné jako color, p, d, low a ap vyžadují prostor $O(V)$ [4].

2.2 Mosty

Mosty (*bridges*) v grafu jsou hrany, které při odstranění způsobí, že graf přestane být souvislý tzn. rozdělí graf na více komponent. Tyto hrany jsou kritické pro propojení částí grafu, a proto jejich identifikace může pomoci při analýze stability grafu [2].

Algoritmus pro nalezení mostů je také Tarjanův algoritmus, který byl vyvinut v roce 1972. Tento algoritmus je založen na prohledávání do hloubky (DFS). JJe velmi podobný předchozímu. Následující odstavec zobrazuje jeho pseudokód [7].

```
Bridges(V, Adj):
  for u in V do:
    color[u] = WHITE
    p[u] = null
  time = 0
  for u in V do:
    if color[u] == WHITE then:
      Bridge-Visit(u)

Bridge-Visit(u):
  color[u] = GRAY
  d[u] = low[u] = time = time + 1
  for v in Adj[u] do:
    if color[v] == WHITE then:
      p[v] = u
      Bridge-Visit(v)
      low[u] = Min(low[u], low[v])
      if low[v] > d[u] then:
        bridges.add((u,v))
    else if p[u] != v then:
      low[u] = Min(low[u], d[v])
  color[u] = BLACK
```

Význam jednotlivých proměnných:

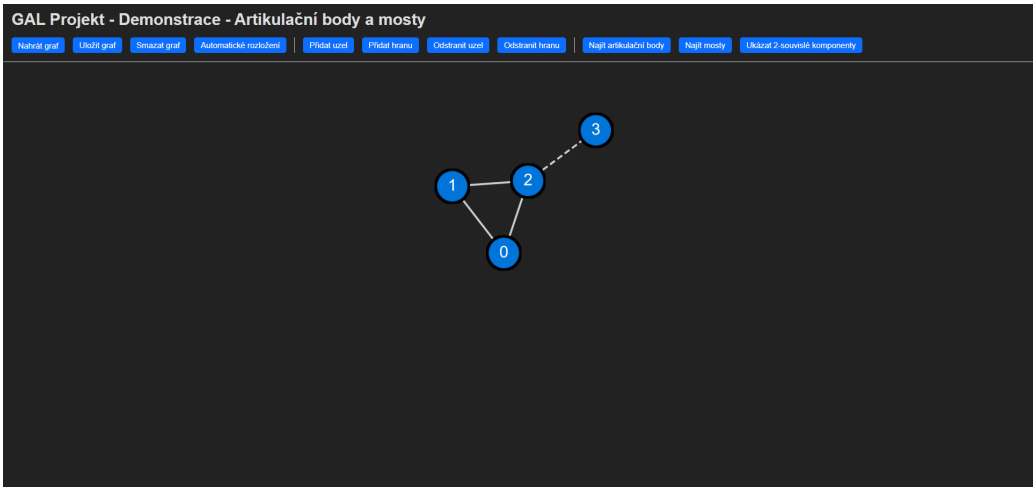
- d[u]: Čas první návštěvy uzlu u.
- low[u]: Nejmenší čas dosažitelný z u.
- p[u]: Pole rodičů uzlů.
- color[u]: Stav uzlu (`WHITE`, `GRAY`, `BLACK`).

Algoritmus pro hledání mostů začíná tím, že inicializuje všechny vrcholy grafu. Každý vrchol u je nastaven na `WHITE` (nenavštíven) a rodiče všech vrcholů jsou inicializováni na `null` (nemá předchůdce). Dále je nastaven časový údaj time, který bude sledovat časové značky během prohledávání grafu.

goritmus následně provádí iteraci přes všechny vrcholy. Pokud je vrchol u stále `WHITE`, spustí funkci `Bridge-Visit(u)`.

Funkce `Bridge-Visit(u)` provádí prohledávání do hloubky (DFS) z vrcholu u. Nejprve označí vrchol u jako `GRAY`. Nastaví první návštěvu $d[u]$ a hodnotu low$[u]$ na aktuální hodnotu time, kterou následně inkrementuje.

Pro každý sousední vrchol v vrcholu u algoritmus vykoná následující kroky:



Obrázek 1: Vzhled 1. obrazovky.

- Pokud je vrchol v stále `WHITE` (nenavštíven), algoritmus nastaví $p[v] = u$ (rodič vrcholu v), zavolá rekurzivně `Bridge-Visit(v)` a po návratu aktualizuje hodnotu low$[u]$ podle low$[v]$. Pokud je low$[v] > d[u]$, znamená to, že hrana (u, v) je most a algoritmus ji přidá do seznamu mostů.
- Pokud je v již navštíven (`GRAY`) a není rodičem u ($p[u] \neq v$), algoritmus aktualizuje hodnotu low$[u]$ podle hodnoty $d[v]$.

Po zpracování všech sousedů vrcholu u je tento vrchol označen jako `BLACK` (plně zpracován).

Časová složitost tohoto algoritmu je $O(V + E)$, protože každý vrchol a každá hrana jsou zpracovány právě jednou. Prostorová složitost je také $O(V + E)$, protože graf je reprezentován seznamem sousedů a pomocné struktury (color, p, d, low) vyžadují prostor $O(V)$.

3 Návrh

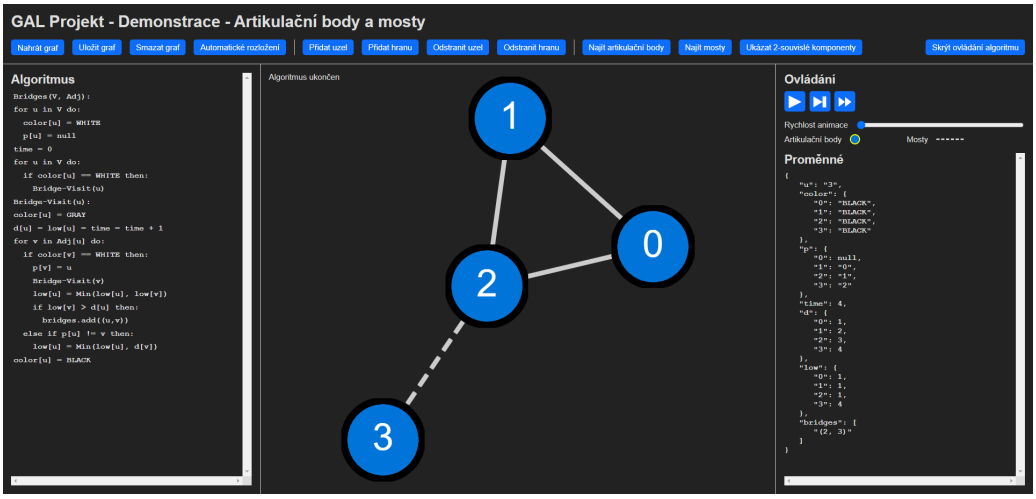
Vyvinutá aplikace se má soustředit na demonstraci algoritmů pro nalezení artikulačních bodů a mostů. K tomu se musí přizpůsobit vzhled a funkčnost aplikace, aby nejen zobrazovala hezký výstup, ale také poskytovala funkční prostředí se základními ovládacími prvky.

Návrh se soustředí na vzhled aplikace a její funkčnost na frontendu. Prvně je představen vzhled a rozložení v sekci Uživatelské rozhraní 3.1 a poté je popsána funkčnost jednotlivých tlačítek a kláves.

3.1 Uživatelské rozhraní

V této části je primárně představen vzhled aplikace a změny obrazovek podle jejího stavu. Aplikace má primárně dva stavy, které mezi sebou přechází. První obrazovka se stará o nahrávání a ukládání grafů, jednoduché úpravy grafu a spouštění algoritmů. Druhá obrazovka se věnuje obsluze algoritmů, kde je zobrazen algoritmus a jeho stav, proměnné a jejich hodnoty, ovládání a nakonec legenda.

Výsledný vzhled první obrazovky vypadá podle obrázku 1. Tato obrazovka je výchozí a po každém spuštění se uživateli vždy zobrazí prázdná. Obrazovka je na nejvyšší úrovni rozdělena do dvou částí. První část obsahuje nadpis a pod ním jsou sekce s úpravami, které lze provést se grafem. První sekce se zaměřuje na ukládání a nahrávání grafů. Dále obsahuje globální úpravy jako smazání celého grafu a automatické rozložení, které přizpůsobí graf obrazovce. Druhá sekce se týká jednoduchých úprav, které přidávají nebo odstraňují uzly či hrany. Poslední sekce je na spouštění algoritmů pro artikulační body a mosty. Dále tam je zobrazené 2-souvislé komponenty pro snadnější reprezentaci a zpřehlednění výsledků algoritmu.



Obrázek 2: Vzhled 2. obrazovky.

Po všech sekcích je zobrazena druhá část, kde je volná plocha na kreslení grafů. Tato plocha je interaktivní, takže pokud se na ni klikne, tak se může posunout a nebo pokud se posune kolečkem ,tak se zvětší nebo zmenší. Pro přesunutí uzlu ho stačí vzít a posunou myší. V této oblasti se také zobrazuje výstup nebo nápověda z tlačítek vlevo nahoře. Například po kliknutí na tlačítko "Přidat uzel", na výstupu se zobrazí "Klikněte pro přidání uzlu".

Druhá obrazovka je zobrazena na obrázku 2. Tato obrazovka má k dispozici stejné funkce jako první, ale pouze za očitých podmínek. Přepnutí do této obrazovky je možné po spuštění jednoho ze dvou algoritmů. Spuštěný algoritmus se zastaví na prvním kroku. Na levé části obrazovky je zobrazen algoritmus, přičemž krok je zvýrazněn ve žlutém okně, které má žlutý rámeček a pozadí. Na pravé části obrazovky, od spodního okraje, jsou zobrazeny hodnoty proměnných ve formátu JSOU. Jsou zde takto ze dvou důvodů. První je z důvodu přehlednosti. Pokud by byly hodnoty zobrazeny v tabulce, při větším počtu proměnných by se nevešly na obrazovku a bylo by nutné je posouvat. Takže to je nutné mít v vertikálním formátu. Druhým důvodem je možnost uložit stav proměnných do externího souboru pro kontrolu chyb nebo pozdější zpracování. Nad tímto se vykytuje malá legenda, kde proužkované hrany jsou mosty a artikulační body jsou uzly se žlutém okrajem. Jako poslední sekce je ovládání. Prvním tlačítkem lze spustit pomalou reprezentaci kroků s ovladatelnou rychlostí. Druhý znak přeskočí jeden krok a poslední vykoná celý algoritmus najednou. Algoritmus nastavuje různé barvy uzlům, které jsou reprezentované v přehledu proměnných nebo v samotném grafu. Například pokud je uzel `WHITE`, tak jeho okraj je bílý.

Jak jsem zmínil na začátku předchozího odstavce, tlačítka z předchozí obrazovky lze používat, ale za určitých podmínek. Tyto podmínky jsou splněné pouze, pokud je algoritmus dodělaný. Existují však výjimky pro tlačítka spuštění jiných algoritmů nebo skrytí algoritmu.

4 Implementace

Z návrhu je nyní definována aplikace, jak má vypadat a jaké má funkce. Tyto údaje je nutné implementovat ve správném prostředí se správnými funkcemi. Tímto se zabývá sekce Technologie 4.1. Dále je samotná implementace návrhu, kde jsou popisuje různé funkce, zachycené události a hlavně celý backend, které jsou podrobně popsány v sekci Implementace návrhu 4.2.

4.1 Technologie

Pro aplikaci bylo zvoleno prostředí JavaScript pro vytvoření webového rozhraní. Je to z důvodu, snadného spuštění (stačí pouze webový prohlížeč), lehké přenosnosti mezi systémy a obsahu podpůrných knihoven.

Jelikož se tento projekt zaměřuje na demonstraci algoritmů, tak toto řešení má nejlepší vlastnosti z dostupných prostředí.

Celá aplikace je postavena na knihovně Cytoscape. Cytoscape je open-source software navržený pro vizualizaci, analýzu a interpretaci sítí, které jsou modelovány jako grafy s uzly a hranami. Hlavní předností Cytoscape je schopnost zpracovávat komplexní sítě a zobrazovat je pomocí pokročilých vizualizačních nástrojů. Uživatelé mohou upravovat vzhled grafů, měnit barvy, velikosti a tvary uzlů, stejně jako styly hran, což umožňuje lepší přehlednost a srozumitelnost dat [3].

Cytoscape podporuje většinu moderních prohlížečů, které implementují standardy HTML5 a mají funkční JavaScriptové API s podporou ECMAScript 5 a vyšší. Podporované prohlížeče jsou například Google Chrome, Mozilla Firefox, Microsoft Edge, Safari a jejich ekvivalenty. Starší verze Internet Exploreru nejsou podporovány. Jelikož cytoscape běží přímo v webovém prohlížeči, může mít vyšší nároky na paměť, zejména při práci s většími projekty [3].

4.2 Implementace návrhu

Aplikace je rozdělená do tří souborů. První je `index.html`, který má uložený přibližné rozložení plocha tlačítek. Druhý je soubor `style.css`, který obsahuje styly jak vypadá naformátovaný první soubor. Poslední je `app.js`, který se stará o události, backend tlačítek a definuje proměnné.

Soubor `app.js` obsahuje globální proměnné a inicializace, konfiguraci Cytoscape.js, manipulaci s grafem, algoritmy pro grafy a události aplikace.

Glomanipulace s grafem, algoritmy pro grafy, události aplikacebální proměnné

Proměnné pro ovládání 1. obrazovky:

- `statusLine, newNodeId, selectedNode, graphChanged`: Sledují aktuální stav aplikace (např. přidávání uzlu/hrany, změna grafu apod.). Obecně sledují vše, co nelze udělat jedním krokem nebo potřebují v mezikroku uživatele.
- `currentBccs, nodeBccs`: Proměnné uchovávající informace o 2-souvislých komponentách grafu (Biconnected Components, BCC).
- `biconnectedOn`: Příznak, který určuje, zda jsou 2-souvislé komponenty aktuálně zobrazeny.
- `downloadLink, fileUpload`: HTML prvky, které slouží k možnosti stahování a nahrávání grafy.
- `currentAlgo`: Odkaz na aktuálně běžící algoritmus, nebo `null`, pokud žádný neběží. Slouží také jako detekci aktuální obrazovky.

Proměnné pro ovládání 2. obrazovky s algoritmy:

- `algoStepTime`: Doba (v milisekundách) mezi jednotlivými automatickými kroky algoritmu.
- `algoVars`: Veřejné proměnné algoritmu (například uzly, které se zpracovávají).
- `algoVarsInternal`: Interní proměnné algoritmu, nepřístupné pro uživatele.
- `algoStep`: Indikuje aktuální krok algoritmu (index v poli `currentAlgo`).
- `algoInterval`: Intervalová proměnná pro automatické kroky algoritmu (`setInterval`).
- `algoStepsContent`: HTML element zobrazující kroky algoritmu (např. pseudokód).
- `algoVarsContent`: HTML element zobrazující hodnoty proměnných algoritmu (např. `low`, `disc`).
- `animSpeed`: Ovládací prvek pro nastavení rychlosti animace.
- `playToggle`: Tlačítko pro spuštění/pozastavení algoritmu.

Inicializace událostí, cytoscape.js konfigurace a ovládání

Na začátku je také nastavení `cytoscape.js`, kde se element `cy` obsahuje konfiguraci Cytoscape.js (např. styl uzlů/hran a rozložení pomocí algoritmu "cose").

Dále jsou nastavené události grafu podle stavů proměnné `Status`, které se týkají práce s plochou a uživatelem. Například při kliknutí na plochu (tap) při statusu `addingNode`, se přidá uzel. Nastavují se všechny základní funkce, které jsou popsané v sekci Uživatelské rozhraní 3.1 (Uložení / Odstranění uzlu a hrany).

Funkce pro tlačítka 1. obrazovky

Následující jsou nadefinované složitější funkce pro práci přímo s tlačítky a jsou pojmenovány anglickými ekvivalenty. Všechny tyto funkce prvně zkontroluje obrazovka, a až pak se spustí. První je stahování a nahrávání, kde je podporován formát GraphML. Při nahrávání se prvně zpracuje korektnost vstupu a následně se všechny hrany a uzly vloží do grafu pomocí seznamu uzlů a hran. Následně nastaví vlastnosti a rozložení. Při stahování se projdou všechny elementy pomocí seznamu uzlů a hran a uloží je do formátu GraphML, který se nakonec stáhne.

Druhé jsou funkce pro tlačítka na odstranění všeho, změnu rozložení, spuštění uložení / odstranění uzlu a hrany a spuštění algoritmů. Funkce, které měli jednostavovou funkci, pouze vykonali tuto funkce skrze cytoscape a ukončili se, ale dvoustavovou nastavily status, který poté zpracovala událost. Základní události byly popsané o dva odstavce výše. Poslední zbylo spuštění algoritmů, které nastavil obrazovku s algoritmem a status.

Poslední funkce pro tlačítka je zobrazení a skrytí 2-souvislých komponent. Tato funkce nalezne a obarví 2-souvislé komponenty. Algoritmus je stejný jako pro hledání artikulačních bodů a mostů, ale s úpravou, že nehledá tyto prvky, ale zaznamenává si množiny prvků než naleznou artikulační bod nebo most.

Funkce pro 2. obrazovku

Následují funkce jsou pro ovládání animace, nastavení zobrazení proměnných a stav algoritmu. První funkce `setAlgoStep` nastavuje zvýraznění řádku algoritmu. Další funkce `rerenderAlgoVars` zobrazí proměnné použité v algoritmu. Funkce `changeAnimSpeed` nastaví novou rychlost animace. Funkce `setCurrentAlgo` slouží k nastavení algoritmu pro hledání bodů nebo mostů. Následují funkce `toggleAlgoControls` je ke skrytí nebo zobrazení ovládacích prvků algoritmu.

Další funkce jsou navázané s ovládávám algoritmu, kde `toggleAlgoPlay` spustí nebo zastaví animaci. Funkce `algoFinish` dokončí celý algoritmus po krocích a `execAlgoStep` vykoná jeden krok v algoritmu tzn. vyobrazí proměnné a posune algoritmus. Na závěr jsou dvě proměnné `articulationPoints` a `bridges`, které mají uložený algoritmus v textové a funkční formě.

Natavení událostí a klávesových zkratek

Jako poslední je je natavení událostí příslušným tlačítkům a klávesovým zkratkám skrze předem nadefinovaným funkcím, které byli v předchozích dvou podsekcím. Každá funkce má plnoklávesovou zkratku zobrazitelnou po nejetí na element. Tyto informace jsou uložené v `eventConfigs`.

5 Demonstrace použití

V této sekci se zabývá spuštění jednotlivých příkladů, které jsou od vytvoření nových uzlů až po ovládání algoritmů. Tyto příklady slouží jenom pro demonstraci všech funkcí a zdokumentování správného použití.

Pro upřesnění všechny kliknutí v této sekci se dělají levým tlačítkem.

5.1 Spuštění aplikace

Jako první krok pro práci s aplikací je její spuštění. Jelikož to je webová aplikace, tak je nutné pouze pustit webový prohlížeč se souborem `index.html`. Po tomto kroku by se mělo zobrazit tmavé okno, které je na obrázku 1.

5.2 Úpravy uzlů a hran

Nejdůležitější na začátku je si vytvořit uzly pomocí tlačítka ”Přidat uzel”. kliknutím na toto tlačítko se spustí proces vkládání uzlů a při kliknutí na prázdnou plochu níže se vytvoří uzel v místě ukazatele. Klávesová zkratka pro nový uzel je písmeno ”n”.

Pro přidání hrany to funguje podobně. Nejprve se klikne na ”Přidat hranu”. Poté se klikne na dva různé uzly a následně se vytvoří hrana mezi vybranými uzly. Klávesová zkratka pro novou hranu je písmeno ”e”.

Některé uzly nebo hrany překážejí a jsou nutné občas odstranit. Tuto funkčnost zajišťují tlačítka ”Odstranit uzel”nebo ”Odstranit hranu”. Po kliknutí na chtěné tlačítko se poté klikne na hranu nebo uzel pro odstranění a hrana nebo uzel se všemi navazujícími hranami zmizí. Pokud je nutné odstranění celého grafu. Je nutné kliknout na tlačítko ”Smazat graf”.

5.3 Pohybování se na ploše s uzly a hranami

Pro úpravu, posunování a škálování plochy se používá myš. Pokud je potřeba posunout plochu i se všemi uzly, klikne do volného místa, podrží se a posunutím se posune celá plocha.

Pro ztenčení nebo zvětšení uzlů a plochy se posunuje kolečkem nahoru a dolu.

Pro posunutí jednoho uzlu je nutné na něj kliknout, podržet a posunout do místa učení.

Aplikace má také funkci automatického rozložení, kde se všechny uzly posunou a neškálují přesně na obrazovku. Je možné, tak udělat pomocí kliknutí na tlačítko ”Automatické rozložení”.

5.4 Nahrávání a ukládání grafů

Aplikace podporuje nahrávání a ukládání grafů ve formátu GraphML. Tato soubory končí s příponou `.graphml`.

Pro nahrání je potřeba kliknout na ”Nahrát graf”. Poté vyběhne nové okno k výběru souboru, které je potřeba ukončit otevřením. Následně se vše na ploše smaže a přidá se graf ze souboru. Pokud dojde k chybě, zobrazí se upozornění.

Stahování prohýbá po kliknutí na ”Stáhnout graf”, která se uloží na výchozí ukládací místo. Většina prohlížečů má nastavené Stažené soubory.

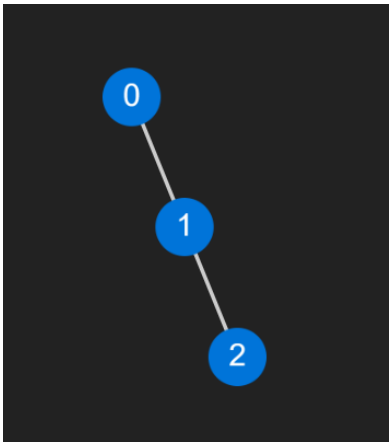
5.5 Ovládání algoritmů

Pokud je na ploše nějaká graf, může se pustit algoritmus pro nalehnutí artikulačních bodů nebo mostů, kliknutím na tlačítka ”Najít artikulační body”nebo ”Najít mosty”.

Tím to krokem se zobrazí příslušný algoritmus na levé části obrazovky. Ve stření čísti se automaticky rozloží graf, který se bude vyvíjet podle stavů algoritmu. V pravé části se vyskytnou ovládací prvoky, kde první spustí animaci algoritmu, druhý udělá jeden krok při kliknutí a poslední projde okamžitě celý alegorismus. Pod těmito prvky je ještě posuvník, který určuje rychlost animace. Pokud půjdeme dále, tak tam je miniaturní legenda, kde artikulačních body jsou zobrazeny žlutým okrajem a mosty přerušovanou čárou. Jako poslední jsou zobrazené proměnné a jejich hodnoty použité v algoritmu.

Na zvýraznění funkčnosti algoritmu se zde vyskytuje ještě jedno tlačítko ”2-souvislé komponenty”, které vypočítá a zobrazí 2-souvislé komponenty. Jejich odlišnost lze poznat barevným odlišením. Poté stejným tlačítkem se mohou i skrýt.

Poslední tlačítko je ”Skrýt ovládání algoritmu”, který skryje ovládání při spuštěném algoritmu a při ukončení ho vypne. Pokud je skryté, tak se opět může zobrazit tlačítkem ”Zobrazení ovládání algoritmu”.



Obrázek 3: Příklad 1. grafu.

5.6 Příklady použití algoritmů

Pro demonstraci použití jsou vytvořeny tři grafy, které předvedou použití algoritmů.

1. graf

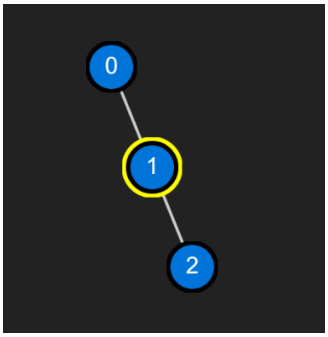
Jako první graf byl zvolen graf o třech uzlech a dvou hranách, který ze zobrazen na obrázku 3. Průběh algoritmu vypadá následně:

1. **Inicializace**:

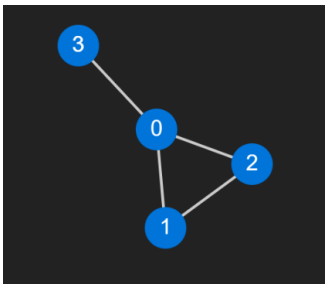
- Pro každý vrchol $u \in \{0, 1, 2\}$ nastavíme:
  - color$[u]$ = WHITE (neprozkoumaný vrchol),
  - p$[u]$ = null (vrchol nemá rodiče),
  - ap$[u]$ = false (počátečně žádný vrchol není artikulační bod),
  - time $= 0$.

2. **Průchod grafem (DFS)**: Algoritmus začíná prohledávat graf od vrcholu 0, protože je stále **WHITE**.

- **Začínáme od vrcholu 0**:
  - color$[0]$ = GRAY (vrchol 0 je nyní prozkoumáván),
  - d$[0]$ = low$[0] = 1$, čas se zvyšuje,
  - Prohlížíme sousedy vrcholu 0, což je vrchol 1.
- **Pokračujeme na vrchol 1**:
  - color$[1]$ = GRAY,
  - d$[1]$ = low$[1] = 2$, čas se zvyšuje,
  - Prohlížíme sousedy vrcholu 1, což je vrchol 2.
- **Pokračujeme na vrchol 2**:
  - color$[2]$ = GRAY,
  - d$[2]$ = low$[2] = 3$, čas se zvyšuje,
  - Vrchol 2 nemá žádné další sousedy, proto se označí jako **BLACK**.



Obrázek 4: Výsledek 1. příkladu



Obrázek 5: Příklad 2. grafu

3. **Návrat na vrchol 1**:

- Po návratu na vrchol 1, aktualizujeme $\text{low}[1] = \min(\text{low}[1], \text{low}[2]) = 2$,
- Zkontrolujeme podmínky pro artikulační bod:
  - Vrchol 1 není kořen (má rodiče),
  - $\text{low}[2] \geq \text{d}[1]$, což znamená, že vrchol 1 je artikulační bod, protože odstranění vrcholu 1 by rozdělilo graf na dvě komponenty.

4. **Návrat na vrchol 0**:

- Po návratu na vrchol 0, aktualizujeme $\text{low}[0] = \min(\text{low}[0], \text{low}[1]) = 1$,
- Vrchol 0 nemá žádné další sousedy a označí se jako **BLACK**.

V tomto grafu je **vrchol 1** artikulačním bodem, protože jeho odstraněním by graf byl rozdělen na dvě části (jednu obsahující vrchol 0 a druhou obsahující vrchol 2). Vrcholy 0 a 2 nejsou artikulačními body, protože jejich odstranění by nezpůsobilo rozdělení grafu na více komponent. Výsledek je vidět na na obrázku 4.

2. graf

Mějme graf na obázku 5 se 4 vrcholy $V = \{0, 1, 2, 3\}$ a následujícími hranami:

$$\text{Hrany: } \{(0, 1), (0, 2), (0, 3), (1, 2)\}.$$

Použijeme algoritmus pro nalezení mostů (Bridges) pomocí DFS.

1. Inicializujeme všechny vrcholy jako nenavštívené (`color[u] = WHITE`) a nastavíme rodiče vrcholů (`p[u] = null`).

2. Každý vrchol u, který ještě nebyl navštíven, procházíme pomocí funkce `Bridge-Visit(u)`.
3. Sledujeme:
    - `d[u]`: čas objevení vrcholu u,
    - `low[u]`: nejmenší dosažitelný čas z u.

DFS z vrcholu 0:

- Nastavíme color[0] = GRAY, d[0] = low[0] = 1, time = 1.
- Zpracujeme hranu $(0, 1)$: protože color[1] = WHITE, voláme `Bridge-Visit(1)`.

DFS z vrcholu 1:

- Nastavíme color[1] = GRAY, d[1] = low[1] = 2, time = 2.
- Zpracujeme hranu $(1, 2)$: protože color[2] = WHITE, voláme `Bridge-Visit(2)`.

DFS z vrcholu 2:

- Nastavíme color[2] = GRAY, d[2] = low[2] = 3, time = 3.
- Hrana $(2, 0)$: zpětná hrana k předkovi, nastavíme low[2] = min(low[2], d[0]) = 1.
- Hrana $(1, 2)$: zpětná hrana k předkovi, ignoruje se.

Po návratu do vrcholu 1 nastavíme:

$$\text{low[1] = min(low[1], low[2]) = 1.}$$

Po návratu do vrcholu 0, zpracujeme hranu $(0, 3)$:

- color[3] = WHITE, voláme `Bridge-Visit(3)`.

DFS z vrcholu 3:

- Nastavíme color[3] = GRAY, d[3] = low[3] = 4, time = 4.
- Žádné další sousedy. Návrat zpět.

Po návratu do vrcholu 0:

$$\text{low[3] ¿ d[0]} \implies \text{hrana } (0, 3) \text{ je most.}$$

Mosty v grafu jsou:

$$(0, 3)$$

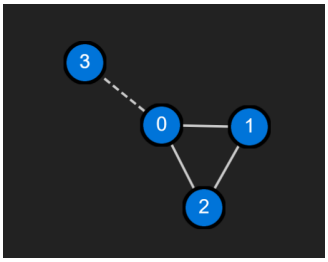
, které jsou vidět na obrázku 6.

3. graf

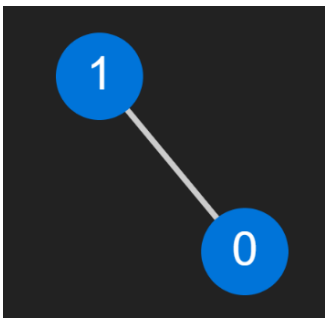
Mějme graf na obrázku 7 se dvěma vrcholy $V = \{0, 1\}$ a jedinou hranou:

$$\text{Hrany: } \{(0, 1)\}.$$

Použije se algoritmus pro nalezení artikulačních bodů pomocí DFS.



Obrázek 6: Výsledek 2. příkladu



Obrázek 7: Příklad 3. grafu

Inicializace:

- Všechny vrcholy nastavíme jako nenavštívené (color$[u]$ = WHITE).
- Nastavíme p$[u]$ = null a ap$[u]$ = false.
- Spustíme DFS od vrcholu 0.

DFS z vrcholu 0**:**

- Nastavíme color$[0]$ = GRAY, d$[0]$ = low$[0] = 1$.
- Zpracujeme souseda $v = 1$: voláme `AP-Visit(1)`.

DFS z vrcholu 1**:**

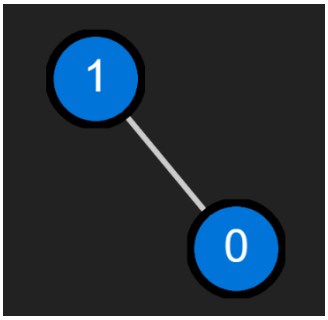
- Nastavíme color$[1]$ = GRAY, d$[1]$ = low$[1] = 2$.
- Vrchol 1 má souseda $v = 0$, což je jeho předek, takže low$[1] = \min(\text{low}[1], \text{d}[0]) = 1$.
- Barvíme color$[1]$ = BLACK a vracíme se zpět do vrcholu 0.

Návrat do vrcholu 0**:**

- Po návratu nastavíme low$[0] = \min(\text{low}[0], \text{low}[1]) = 1$.
- Vrchol 0 má children $= 1$, a protože children ≤ 1, není artikulačním bodem.

Kontrola vrcholu 1**:**

- Vrchol 1 nemá žádné děti a nesplňuje podmínky pro klíčový vrchol.



Obrázek 8: Výsledek 3. příkladu

Výsledek

V tomto grafu neexistují klíčové vrcholy. Odstranění žádného vrcholu nerozdělí graf na více komponent jak je vidět na obrázku 8.

6 Závěr

V tomto projektu jsme měli vytvořit aplikaci pro demonstraci nalezení artikulačních bodů a mostů, což se povedlo skrze Tarjanův algoritmus. Tento algoritmu s má lineární složitost a tudíž je dobře použitelný ve širém množství případů. Tento algoritmu byl zanořen do knihovny cytoscape v programovacím prostředí JavaScript pro jeho jednoduchou použitelnost a přenositelnost. Dále bylo přidáno jednoduché ovládáni grafu a poté i algoritmu.

Ve výsledku se může ohodnotí, že aplikace splňuje požadavky na nalezení důležitých prvků a jejich reprezentaci. Avšak aplikace má stále možnost nějakého rozvoje, například ve vzhledu nebo přidání různých dalších algoritmů.

Reference

[1] Blažej, V.: An Overview of Graph Theory. 2020, navštíveno: 3.12.2024. Dostupné z: https://vaclavblazej.github.io/post/2020/graph_theory/

[2] Cormen, T. H.; Leiserson, C. E.; Rivest, R. L.; aj.: *Introduction to Algorithms*. MIT Press, třetí vydání, 2009, 748–759 s.

[3] Cytoscape Consortium: Cytoscape: An Open Source Platform for Complex Network Analysis and Visualization. 2024, navštíveno: 8.12.2024. Dostupné z: https://cytoscape.org/index.html

[4] Farina, G.: The Impact of Linear Time Algorithms on the Computational Complexity of Problems in Graph Theory. 2015, navštíveno: 3.12.2024. Dostupné z: https://www.mit.edu/~gfarina/2016/impact-linear-icalp-yr-2015/impact_linear_icalp_yr_2015.pdf

[5] GeeksforGeeks: Articulation Points (or Cut Vertices) in a Graph. 2024, navštíveno: 1.12.2024. Dostupné z: https://www.geeksforgeeks.org/articulation-points-or-cut-vertices-in-a-graph/

[6] GeeksforGeeks: Introduction to Graphs - Data Structure and Algorithm Tutorials. 2024, navštíveno: 1.12.2024. Dostupné z: https://www.geeksforgeeks.org/introduction-to-graphs-data-structure-and-algorithm-tutorials/?ref=lbp

[7] Tarjan, R.: Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, ročník 1, č. 2, 1972: s. 146–160, doi:10.1137/S0097539700000526.