Katedra informatiky
Přírodovědecká fakulta
Univerzita Palackého v Olomouci

BAKALÁŘSKÁ PRÁCE

Hledání klik v grafu

algoritmy a aplikace

2019

Adam Beneš

Vedoucí práce: Mgr. Petr Osička, Ph.D.

Studijní obor: Aplikovaná informatika, prezenční forma

Bibliografické údaje

Autor:	Adam Beneš
Název práce:	Hledání klik v grafu (algoritmy a aplikace)
Typ práce:	bakalářská práce
Pracoviště:	Katedra informatiky, Přírodovědecká fakulta, Univerzita Palackého v Olomouci
Rok obhajoby:	2019
Studijní obor:	Aplikovaná informatika, prezenční forma
Vedoucí práce:	Mgr. Petr Osička, Ph.D.
Počet stran:	53
Přílohy:	1 CD/DVD
Jazyk práce:	český

Bibliograhic info

Author:	Adam Beneš
Title:	Clique problems (algorithms and applications)
Thesis type:	bachelor thesis
Department:	Department of Computer Science, Faculty of Science, Palacký University Olomouc
Year of defense:	2019
Study field:	Applied Computer Science, full-time form
Supervisor:	Mgr. Petr Osička, Ph.D.
Page count:	53
Supplements:	1 CD/DVD
Thesis language:	Czech

Anotace

Cílem práce je pojednání o problematice hledání klik v grafech a příbuzných algoritmických problémech. Práce je rozdělena do několika kapitol. V první kapitole jsou popsány pojmy, které souvisí s teorií grafů. V druhé kapitole se rozebere problém klik. Ve třetí kapitole se probere problematika diverzifikovaných klik. Ve čtvrté kapitole se nachází příručka pro uživatele a v páté kapitole je programátorská část.

Synopsis

The aim of the work is to discuss the issue of searching for cliques in graphs and related algorithmic problems. The thesis is divided into several chapters. The first chapter describes the concepts related to graph theory. The second chapter discusses the problem of cliques. The third chapter discusses the issue of diversified cliques. In the fourth chapter there is a user manual and in the fifth chapter there is a programming part.

Klíčová slova: problém klik; diverzifikované kliky; nezávislá množina

Keywords: problem of cliques; diversified cliques; independent set

Rád bych poděkoval Mgr. Petru Osičkovi, Ph.D., za užitečné rady a připomínky při vedení této bakalářské práce. Dále bych chtěl poděkovat rodině za podporu a trpělivost.

Místopřísežně prohlašuji, že jsem celou práci včetně příloh vypracoval/a samostatně a za použití pouze zdrojů citovaných v textu práce a uvedených v seznamu literatury.

datum odevzdání práce podpis autora

Obsah

Seznam obrázků

1 Teorie

1.1 Graf

Neorientovaný graf Formálně chápeme neorientovaný graf G jako dvojici $G = (V, E)$, kde V je množina vrcholů a E je množina hran, které se reprezentují dvouprvkovou množinou uzlů. Množina hran je ve tvaru $\{\{a, b\} \in E : a, b \in V \wedge a \neq b\}$.

Orientovaný graf Orientovaný graf G je dvojice $G = (V, E)$, kde V je množina vrcholů a E je množina hran, které se reprezentují uspořádanou dvojicí uzlů. Množina hran je ve tvaru $\{\langle a, b\rangle \in E : a, b \in V\}$. V orientovaném grafu se mohou vyskytovat smyčky.

Ohodnocený graf V některých problémech si nevystačíme pouze s orientovaným nebo neorientovaným grafem. Proto se k hranám přidávají číselné hodnoty (ale mohou být i jiné) z množiny hodnot označené jako D, které značí například dobu trvání, délku silnice, pravděpodobnost a další. Grafu s takovými přidanými hodnotami se říká ohodnocený graf (nebo také síť). Zobrazení, které přiřazuje každé hraně (nebo vrcholu) určitou hodnotu nazýváme ohodnocením hrany (nebo vrcholu) a značíme ho ε.

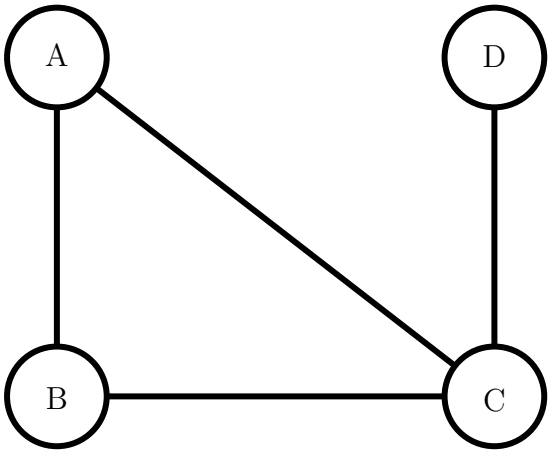
Ohodnocený graf je trojice ve tvaru $G = (V, E, \varepsilon)$, kde V a E je to samé jako u neorientovaného nebo orientovaného grafu. U neorientovaného grafu je ε zobrazení, které neuspořádané dvojici uzlů přiřadí hodnotu z D. A u orientovaného grafu je ε zobrazení, které uspořádané dvojici uzlů přiřadí hodnotu z D.

Stupeň vrcholu (u neorientovaného grafu) Značíme $deg(v)$ a znamená to počet hran vedoucích z/do uzlu v, nebo se to dá popsat jako počet sousedů vrcholu v.

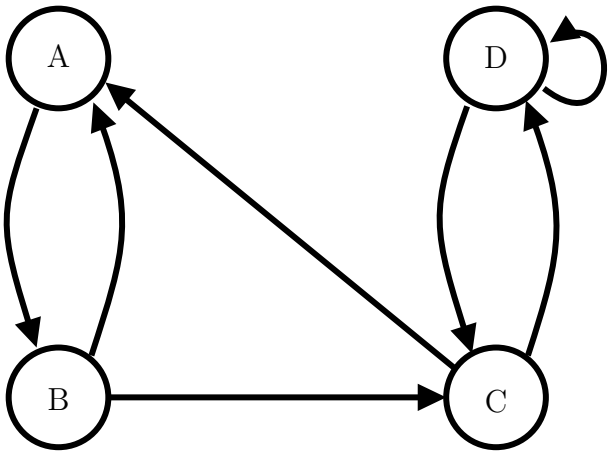
Úplný neorientovaný graf Je takový neorientovaný graf, kde jsou každé dva různé vrcholy spojeny hranou. U úplného grafu, kde číslo n značí počet vrcholů, se dá vyjádřit počet hran jako $\frac{n(n-1)}{2}$.

Prostý graf Mezi vrcholem v_1 a v_2 buďto nevede žádná hrana anebo vede právě jedna.

Komplementární (doplňkový) graf $G1 = (V, H1)$ a $G2 = (V, H2)$ jsou doplňkové jestliže platí: mají disjunktní množiny hran a $G = (V, H)$,kde $H = H1 \cup H2$, je úplný graf. Jinak řečeno, pokud v původním grafu G byla hrana mezi vrcholy a, b, tak v doplňkovém grafu hrana mezi vrcholy a, b není a naopak. V tomto textu se doplňkový graf ke grafu G bude značit jako G'.



Obrázek 1: Neorientovaný graf



Obrázek 2: Orientovaný graf

Podgraf grafu Podgraf grafu $G = (V, E)$ je graf $G_1 = (V_1, E_1)$, kde $V_1 \subseteq V$ a $E_1 \subseteq E$. To znamená, že podgraf grafu G vznikne vynecháním nějakých (nebo žádných) vrcholů a hran. Podgraf musí být také grafem. Je kladena podmínka, že spolu s hranou se v podgrafu musí vyskytovat i krajní body této hrany.

1.2 Znázornění grafu (kreslení grafu)

Grafy pro lepší názornost můžeme nakreslit, což je nesporná výhoda. Vrcholy kreslíme jako body (kolečka) a hrany kreslíme jako úsečky (čáry, oblouky), které spojují vrcholy. Pokud je hrana orientovaná, bývá doplněna šipkou, která směřuje od počátečního ke koncovému vrcholu.

1.3 Možnosti reprezentace grafu v počítači

1.3.1 Matice sousednosti

Mějme pevně zvolenou posloupnost vrcholů. V matici máme na řádku i ve sloupcích vrcholy z posloupnosti. Pokud mezi vrcholy V_i a V_j vede hrana, pak $m_{ij} = 1$, jinak $m_{ij} = 0$

Ukázka ke grafu, na obrázku 1, kde jsou uzly seřazeny podle abecedy tak, že na pozici $M_{0,1}$ je uložena hodnota o tom, jestli vede hrana mezi uzly A a B. Pokud je graf neorientovaný, tak je matice symetrická a to znamená, že stejná hodnota jako byla na pozici $M_{0,1}$ bude i na pozici $M_{1,0}$.

$$M = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \end{pmatrix}$$

1.3.2 Matice incidence

Mějme pevně zvolenou posloupnost vrcholů a hran. Matice incidence, budeme značit jako M, má ve sloupcích hrany a na řádcích vrcholy. Na pozici M_{ij} nalezneme záznam o vztahu vrcholu V_i k hraně H_j. U orientovaného grafu může M_{ij} nabývat těchto hodnot:

$$M_{ij} = \begin{cases} 1 & \text{jestliže je } V_i \text{ počátečním vrcholem hrany } H_j \\ -1 & \text{jestliže je } V_i \text{ koncovým vrcholem hrany } H_j \\ 2 & \text{jestliže jde o smyčku} \\ 0 & \text{v ostatních případech} \end{cases}$$

U neorientovaného grafu nastanou pouze dva případy. Buď je uzel incidentní s hranou nebo ne. Proto M_{ij} nabývá těchto hodnot:

$$M_{ij} = \begin{cases} 1 & \text{jestliže vrchol } V_i \text{ je incidentní s hranou } H_j \\ 0 & \text{v ostatních případech} \end{cases}$$

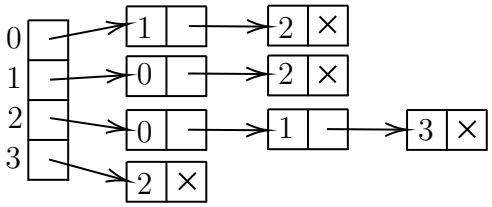
Příkad ukážeme na grafu 2, kde uzly jsou seřazeny podle abecedy, pak matice vypadá následovně:

$$M = \begin{pmatrix} 1 & -1 & 0 & -1 & 0 & 0 & 0 \\ -1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 1 & 1 & -1 & 0 \\ 0 & 0 & 0 & 0 & -1 & 1 & 2 \end{pmatrix}$$

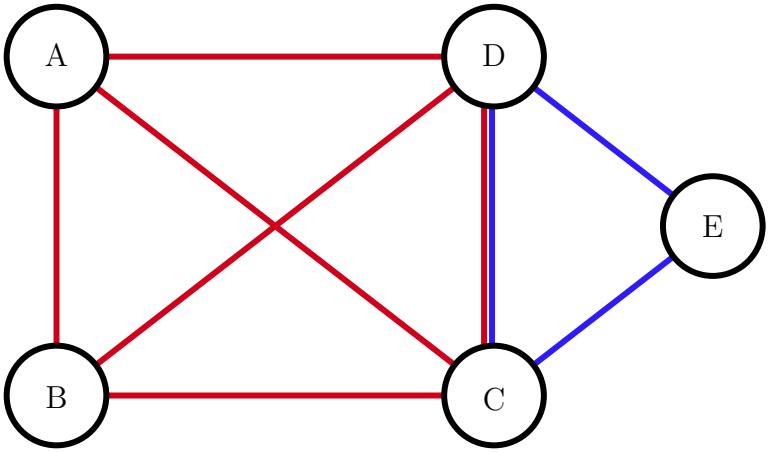
1.3.3 Seznam sousednosti

Graf se reprezentuje seznamem, kde pro každý uzel máme v seznamu uložené jeho sousedy, které jsou obecně uloženy v různém pořadí.

Příklad pro graf 1, kde jsou vrcholy seřazeny podle abecedy tak, že vrchol A je na indexu 0. Výsledný seznam je na obrázku 3.



Obrázek 3: Seznam sousednosti



Obrázek 4: Maximální klika

2 Problém klik

2.1 Klika

Klika grafu G je podgraf G', který je úplný, to znamená, že každé dva vrcholy z G' jsou spojeny hranou. Klikovost grafu je číslo, které udává počet uzlů v největší klice.

Maximální klika je klika, ke které již nelze přidat další vrchol, to znamená, že je to taková klika, která není součástí větší kliky.

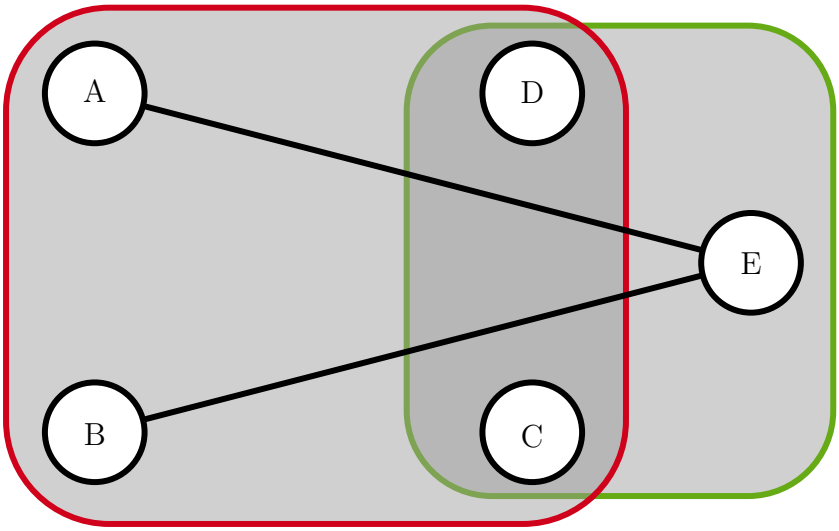
$G = (V, E)$, když $S \in V$, je maximální klika, pak platí: $\forall x, y \in S : (x, y) \in E$.

Neformálně je klika podmnožina uzlů, ve které je každý uzel spojen hranou se všemi dalšími uzly z množiny.

2.2 Nezávislá množina

Nezávislá množina je taková podmnožina vrcholů, kde žádné dva vrcholy nejsou spojeny hranou. Mějme graf $G = (V, E)$, pak podmnožina $S \subseteq V$ je nezávislá množina, pokud platí: $\forall x, y \in S : (x, y) \notin E$.

Nezávislost grafu (značeno $\alpha(G)$) je počet prvků největší nezávislé množiny v grafu, to je takové množiny, ke které už nelze přidat další vrchol, který by splňoval nezávislost vůči ostatním prvkům.



Obrázek 5: Nezávislá množina

Nezávislá množina úzce souvisí s klikou. Kliky v grafu G odpovídají vzájemně jednoznačně maximálním množinám v doplňkovém grafu G'.

2.3 Kliky v reálném světě

Problém hledání klik můžeme převést do reálného světa. Mějme neorientovaný graf, ve kterém vrcholy budou reprezentovat osoby. Hrana mezi vrcholy bude existovat, pokud se osoby mezi sebou navzájem znají. Kliku tedy chápeme jako skupinu lidí, ve které se každý člen skupiny zná s ostatními členy skupiny.

2.4 Problémy s klikami

V počítačových vědách je problém klik (anglicky *clique problem*) výpočetní problém pro hledání klik v grafu, který patří do skupiny tzv. NP-úplných problémů, to znamená, že neexistuje (a neví se, jestli vůbec bude existovat) algoritmus, který by jej řešil s polynomiální časovou složistí.

Algoritmy popsané níže mohou mít různé cíle:

- najít všechny maximální kliky
- najít největší maximální kliku

2.5 Algoritmy pro hledání všech maximálních klik

Jsou algoritmy, jejichž výsledkem jsou všechny maximální kliky.

2.5.1 Bron-Kerbosch algoritmus - základní

Základní Bron-Kerbosch je algoritmus založen na metodě rekurzivního backtackingu. Rekurzivní procedura má tři parametry (R, P, X). Z těchto parametrů najde maximální kliku, která obsahuje všechny vrcholy z R, některé vrcholy z P a neobsahuje žádný vrchol z X. Při každém volání rekurzivní procedury jsou množiny P a X disjunktní, to znamená, že nemají společné prvky. Sjednocení P a X jsou vrcholy, které jsou spojené s každým vrcholem z R. Pokud jsou obě množiny P a X prázdné, pak není vrchol, který by se dal přidat do R, takže algoritmus vrací R jako maximální kliku a backtrackingem se přesune do jiné větve výpočtu.

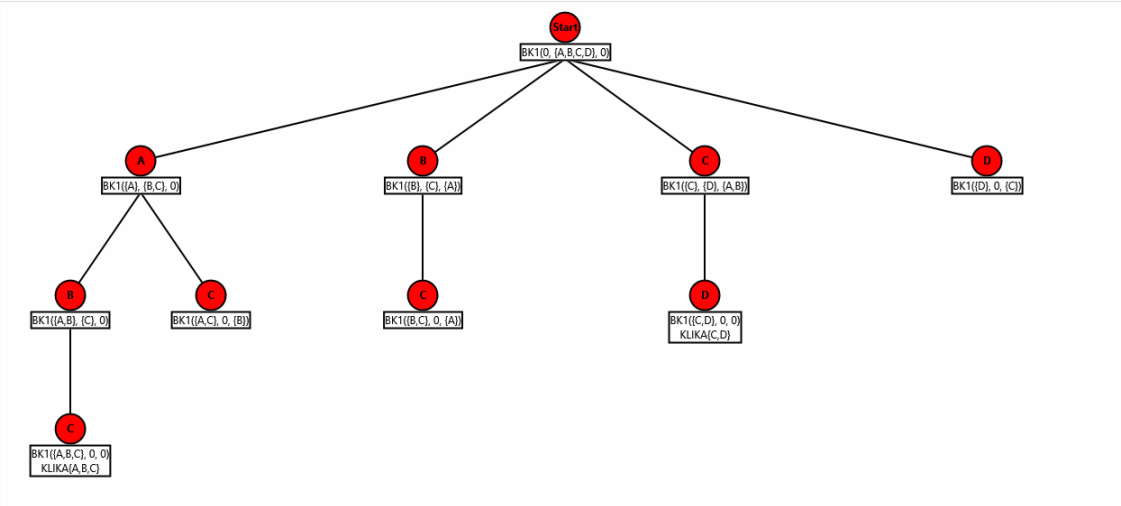
Algoritmus 1: BronKerbosch1(R,P,X)

```
if P = ∅ a X = ∅ then
    R je maximální klika
else
    foreach uzel v ∈ P do
        BronKerbosch1 (R ∪ {v}, P ∩ N(v), X ∩ N(v))
        P ← P \ {v}
        X ← X ∪ {v}
    end
end
```

Popis průchodu:

1. Při prvním volání procedury `BronKerbosch1` jsou množiny R a X prázdné a P je množina všech vrcholů v grafu.
2. V každém volání procedury `BronKerbosch1` se zkontroluje, zda jsou množiny P a X prázdné, pokud ano tak je R maximální klika.
3. Poté se pro každý vrchol $v \in P$:
   (a) zavolá rekurzivně `BronKerbosch1` s poupravenými parametry. K množině R se přídá vrchol v. U množin P a X se provede restrikce (zúžení) na sousedy vrcholu v, takto se najdou kliky, které obsahují vrchol v a sousedy vrcholu v.
   (b) poté se vrchol v přesune z množiny R do množiny X, aby se v budoucnu mohl vrchol v vyloučit z hledání, protože všechny kliky obsahující vrchol v již byly vygenerovány.

Příklad průběhu algoritmu na grafu 1:



Obrázek 6: Příklad algoritmus BronKerbosch1

```
BK1(0, {A,B,C,D}, 0)
  v:=A┌ BK1({A}, {B,C}, 0)
      │   v:=B┌ BK1({A,B}, {C}, 0)
      │       │   v:=C┌ BK1({A,B,C}, 0, 0)
      │       │       │   #KLIKA{A,B,C}
      │       │       └ P:=0; X:={C}
      │       └ P:={C}; X:={B}
      │   v:=C┌ BK1({A,C}, 0, {B})
      │       └ P:=0; X:={B,C}
      └ P:={B,C,D}; X:={A}
  v:=B┌ BK1({B}, {C}, {A})
      │   v:=C┌ BK1({B,C}, 0, {A})
      │       └ P:=0; X:={A,C}
      └ P:={C,D}; X:={A,B}
  v:=C┌ BK1({C}, {D}, {A,B})
      │   v:=D┌ BK1({C,D}, 0, 0)
      │       │   #KLIKA{C,D}
      │       └ P:=0; X:={A,B,D}
      └ P:={D}; X:={A,B,C}
  v:=D┌ BK1({D}, 0, {C})
      └ P:=0; X:={A,B,C,D}
```

2.5.2 Bron-Kerbosch algoritmus - s pivotem

Předchozí algoritmus `BronKerbosch1` prohledává každou kliku, ať už maximální nebo ne. Aby se ušetřil čas a algoritmus neprohledával zbytečně ty větve výpočtu, které neobsahují žádnou maximální kliku, tak se ze sjednocení množin P a X vybere vrchol u, kterého přezdíváme „pivot“. Využije se faktu, že každá

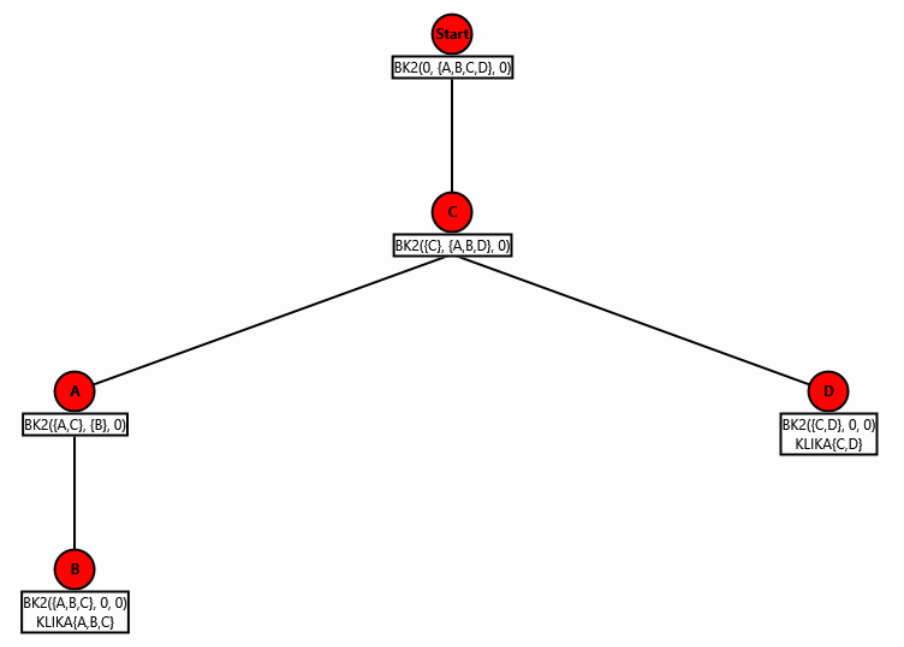
klika musí zahrnout buď vrchol u, nebo jeho nesousedy, protože jinak by se dala klika rozšířit přídáním vrcholu u. Proto pouze vrchol u, nebo jeho nesousedi, mohou být vybráni jako vrchol v.

Algoritmus 2: BronKerbosch2(R,P,X)

if $P = \emptyset \wedge X = \emptyset$ **then**
  R je maximální klika
else
  $u \leftarrow$ pivot z $P \cup X$
  foreach *uzel* $v \in P \setminus N(u)$ **do**
    `BronKerbosch2(`$R \cup \{v\}$, $P \cap N(v)$, $X \cap N(v)$`)`
    $P \leftarrow P \setminus \{v\}$
    $X \leftarrow X \cup \{v\}$
  end
end

Příklad průběhu algoritmu na grafu 1:

```
BK2(0, {A,B,C,D}, 0)
  pivot:=C; List->{C}
  v:=C┌ BK2({C}, {A,B,D}, 0)
      │   pivot:=A; List->{A,D}
      │   v:=A┌ BK2({A,C}, {B}, 0)
      │       │   pivot:=B; List->{B}
      │       │   v:=B┌ BK2({A,B,C}, 0, 0)
      │       │       │   #KLIKA{A,B,C}
      │       │       └ P:=0; X:={B}
      │       └ P:={B,D}; X:={A}
      │   v:=D┌ BK2({C,D}, 0, 0)
      │       │   #KLIKA{C,D}
      │       └ P:={B}; X:={A,D}
      └ P:={A,B,D}; X:={C}
```



Obrázek 7: Příklad algoritmus BronKerbosch2

2.5.3 Bron-Kerbosch algoritmus - vertex ordering

Další možnost, jak vylepšit základní `BronKerbosch1` algoritmus, je místo vybírání „pivota“ vybírat vrcholy ve vhodném pořadí, které dostaneme algoritmem `DegeneracyOrdering` (4). V první úrovni rekurze se pečlivě vybírají vrcholy tak, aby se minimalizovala velikost množiny P, protože čím větší je množina P, tím více volání procedury `BronKerbosch2` bude v další úrovni rekurze následovat.

Algoritmus 3: BronKerbosch3(graf $G = (V, E)$)

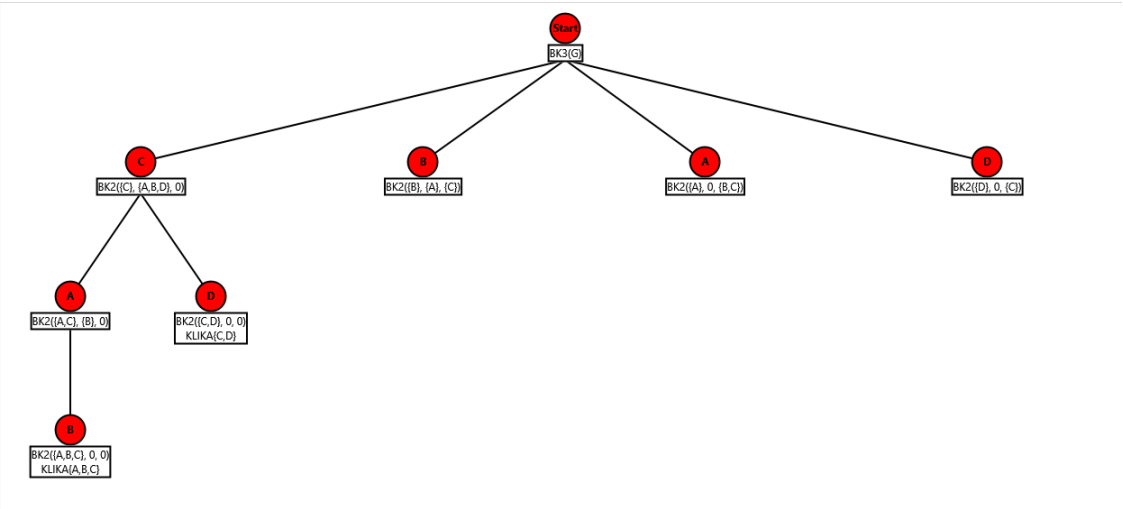
$P \leftarrow V(G)$
$R \leftarrow X \leftarrow \emptyset$
foreach *uzel* $v \in$ *uspořádání degeneracy ordering grafu* G **do**
    `BronKerbosch2` $(\{v\}, P \cap N(v), X \cap N(v))$
    $P \leftarrow P \setminus \{v\}$
    $X \leftarrow X \cup \{v\}$
end

Příklad průběhu algoritmu na grafu 1:

Algoritmus 4: DegeneracyOrdering(graf $G = (V, E)$)

```
/* Zdroj: Matula & Beck (1983)                                    */
L ← inicializace prázdného výstupního seznamu
foreach uzel v ∈ V do d(v) ← degreee(v)
D ← inicializace pole, kde ∀v ∈ V : v ∈ D[d(v)]
while ∃i ∈ {0, 1, ...} : D[i] ≠ ∅          /* i hledáme od 0 */
 do
    for jeden uzel v ∈ D[i] do
        přidej uzel v na začátek seznamu L
        odeber uzel v z D[i]
    end
    foreach w ∈ N(v) \ L do
        přesuň uzel w z D[d(w)] do D[d(w) − 1]
        d(w) ← d(w) − 1
    end
end
return L
```

```
BK3(0, {A,B,C,D}, 0)
  DegOrder:={C,B,A,D}
  v:=C┌ BK2({C}, {A,B,D}, 0)
      │   pivot:=A; List->{A,D}
      │   v:=A┌ BK2({A,C}, {B}, 0)
      │       │   pivot:=B; List->{B}
      │       │   v:=B┌ BK2({A,B,C}, 0, 0)
      │       │       │   #KLIKA{A,B,C}
      │       │       └ P:=0; X:={B}
      │       └ P:={B,D}; X:={A}
      │   v:=D┌ BK2({C,D}, 0, 0)
      │       │   #KLIKA{C,D}
      │       └ P:={B}; X:={A,D}
      └ P:={A,B,D}; X:={C}
  v:=B┌ BK2({B}, {A}, {C})
      │   pivot:=C; List->0
      └ P:={A,D}; X:={B,C}
  v:=A┌ BK2({A}, 0, {B,C})
      │   pivot:=C; List->0
      └ P:={D}; X:={A,B,C}
  v:=D┌ BK2({D}, 0, {C})
      │   pivot:=C; List->0
      └ P:=0; X:={A,B,C,D}
```



Obrázek 8: Příklad algoritmus BronKerbosch3

2.5.4 Tomita-Tanaka-Takahashi algoritmus

`Tomita` je algoritmus pro generování všech maximálních klik, který pomocí backtrakingu prohledává větve výpočtu do hloubky a zároveň tyto větve prořezává podobně jako v Bron-Kerbosch algoritmu. Časová složitost algoritmu je $O(3^{n/3})$, kde n je počet uzlů. Algoritmus se liší tím, že nezabírá tolik místa v paměti.

Procedura `EXPAND` má 2 parametry:

Q - Je globální proměnná, která reprezentuje množinu uzlů, tvořících právě konstruovanou kliku. Do této množiny se postupně přidávají další uzly, dokud nevznikne maximální klika. Na začátku je tato množina prázdná.

SUBG - Pokud jsou v Q uzly u_1, ... , u_n, pak SUBG $= V \cap N(u_1) \cap ... \cap N(u_n)$, SUBG je průnik všech uzlů z V a sousedů všech uzlů z Q. Ze začátku jsou v SUBG všechny uzly z V.

Aplikací `EXPAND` se postupně rozšiřuje Q. Pokud v SUBG již nejsou další uzly, tak se Q prohlásí za maximální kliku, jinak by pridáním $q \in$ SUBG vznikla větší klika. Rekurzivně aplikujeme `EXPAND` se SUBG $=$ SUBG $\cap\, N(q)$, kde $q \in$ SUBG, pro nalezení větší kliky obsahující uzel q.

Prohledávání můžeme znázornit v podobě lesa. Kořeny jsou uzly z V. Každý uzel $q \in$ SUBG má jako své potomky uzly ze SUBG. Cestou od kořene po list sestavíme kliku.

2 metody prořezávání:

1. CAND je množina uzlů, které jsou vhodní kandidáti pro rozšíření kliky Q a zároveň tyto uzly nebyly dosud zpracovány. CAND = SUBG \ FINI, kde FINI je množina již zpracovaných uzlů. Pouze uzly z CAND mohou rozšířit množinu Q, protože kliky obsahující uzly z FINI již byly vygenerovány.

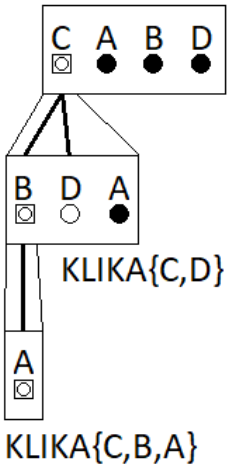
2. Předpokládejme, že pro určitý uzel $u \in$ SUBG byly všechny kliky obsahující uzel u, vygenerovány. Poté každá další klika musí obsahovat alespoň jeden uzel z množiny SUBG $\setminus N(u)$.

Algoritmus 5: Tomita(graf $G = (V, E)$))

```
Q ← ∅
EXPAND (V,V)
return Q

Function EXPAND (SUBG,CAND)
    if SUBG = ∅ then
        Q je maximální klika
    else
        u ← uzel v ∈ SUBG, pro který je |CAND ∩ N(v)| nevětší
        while (CAND \ N(u)) ≠ ∅ do
            q ← uzel z ∈ (CAND \ N(u))
            Q ← Q ∪ {q}
            SUBGq ← SUBG ∩ N(q)
            CANDq ← CAND ∩ N(q)
            EXPAND (SUBGq,CANDq)
            CAND ← CAND \ {q}
            Q ← Q \ {q}
        end
    end
return
```

Příklad průběhu algoritmu na grafu 1:



Obrázek 9: Příklad algoritmus Tomita

```
Tomita(G)
  EXPAND({A,B,C,D}, {A,B,C,D})
    u:=C; List->{C}
    q:=C┌ Q:={C}
        │ EXPAND({A,B,D}, {A,B,D})
        │   u:=B; List->{B,D}
        │   q:=B┌ Q:={B,C}
        │       │ EXPAND({A}, {A})
        │       │   u:=A; List->{A}
        │       │   q:=A┌ Q:={A,B,C}
        │       │       │ EXPAND(0, 0)
        │       │       │     #KLIKA{A,B,C}
        │       │       └ Q:={B,C}
        │       └ Q:={C}
        │   q:=D┌ Q:={C,D}
        │       │ EXPAND(0, 0)
        │       │     #KLIKA{C,D}
        │       └ Q:={C}
        └ Q:=0
```

2.6 Algoritmy pro hledání maximální kliky

Josu algoritmy, jejichž výsledkem je jedna maximální klika.

2.6.1 Maximální klika pomocí hledání maximální nezávislé množiny

Pro hledání maximální kliky můžeme využít faktu, že maximální nezávislá množina v grafu G je maximální klika v doplňkovém grafu G'.

V algoritmu se vyskytují dvě množiny:

M - Je množina uzlů, které tvoří rozpracovanou a ke konci hotovou maximální nezávislou množinu. Slouží jako akumulátor prvků.

Q - Je množina uzlů, které ještě nebyly zpracovány.

Algoritmus pro nalezení maximální nezávislé množiny v cyklu eliminuje z množiny Q uzly, dokud není množina Q prázdná. V cyklu se vybere uzel u, který má největší stupeň. Tento uzel se přidá do množiny M a odebere se z množiny Q. Poté se z Q odeberou i sousedi uzlu u a cyklus se opakuje. Když už je Q prázdná množina, tak se vrátí jako výsledek množina M, což je maximální nezávislá množina. Algoritmus pro nalezení maximální kliky pak pouze zavolá algoritmus pro nalezení maximální nezávislé množiny s doplňkovým grafem G'.

Algoritmus 6: MaximalniKlika(graf $G = (V, E)$)

```
H ← doplněk grafu G
return MaximalniNezavislaMnozina(H)

Function MaximalniNezavislaMnozina(H = (V, E))
    M ← ∅
    Q ← V
    while Q ≠ ∅ do
        u ← uzel z Q s maximálním stupněm
        M ← M ∪ {u}
        foreach v ∈ N(u) do Q ← Q \ {v}
        Q ← Q \ {u}
    end
return M
```

Příklad průběhu algoritmu na grafu 1:

```
MaxCliqueByIS(G)
  H:=Doplnek(G)
  MaxIndependentSet(H)
    N:=0; Q:={A,B,C,D}
    u:=C┌─N:={C}
        └─Q:={A,B,D}
    u:=A┌─N:={A,C}
        └─Q:={B}
    u:=B┌─N:={A,B,C}
        └─Q:=0
    #KLIKA{A,B,C}
```

3 Top-K Kliky

Algoritmus pro generování všech maximálních klik na větším grafu G vygeneruje někdy i několik stovek či tisíc klik, takže výsledek nemusí být přehledný a uchopitelný, protože někdy je potřeba mít pouze pár největších klik. Takový algoritmus má parametr k, což je počet klik, který má algoritmus vrátit.

Nabízela by se naivní verze takového algoritmu, který by použil algoritmus na vygenerování všech maximálních klik, sestupně by je seřadil podle počtu uzlů a vybral by prvních k největších klik. Takový algoritmus by měl velkou paměťovou složitost, protože by všechny kliky musel uchovávat v paměti.

Dále by se nabízelo vylepšení, kde by si algoritmus uchovával v paměti pouze k největších klik, zatímco by generoval nové kliky, ale výsledné kliky by nemusely pokrýt velký počet uzlů a tak by se mohlo stát, že výsledkem bude k klik, které se budou lišit pouze v pár uzlech.

Abych pouze nevylistoval k největších klik, což by nemělo až tak velkou vypovýdající hodnotu, protože by se kliky ve velké míře mohly překrývat, tak jsem se rozhodl implementovat algoritmus na hledání diverzifikovaných top-k klik, dále jen D-klik, který vrátí k klik pokrývajících největší počet uzlů v grafu G.

3.1 EnumAll

`EnumAll` je 1. verze algoritmu, který vrací k klik pokrývajících největší počet uzlů z grafu G. Jak můžeme vidět v pseudokódu 7, tak algoritmus nejprve vypočítá všechny kliky a uloží je do A. Pro výpočet klik používá algoritmus `CliqueAll`, který je vlastně stejný jako algoritmus `BronKerbosch2`. Poté se zavolá funkce `MaxCover` s argumenty V, A a k, která z množiny klik A vybere pouze k klik pokrývajících nejvíce uzlů z grafu G.

Tohle řešení sice nezlepšuje časovou ani paměťovou složitost, ale výsledek má větší vypovídající hodnotu než naivní verze z předchozího odstavce, která pouze vylistovala kliky s největším počtem uzlů, takže u klik hrozilo, že se budou ve velké míře navzájem překrývat.

Algoritmus 7: EnumAll($G = (V, E)$, k)

```
A ← CliqueAll(∅, V, ∅)
return MaxCover(V, A, k)

Function CliqueAll(R, P, X)
    if P = ∅ ∧ X = ∅ then
        R je maximální klika
    else
        u ← pivot z P ∪ X
        foreach uzel v ∈ P ∪ X do
            CliqueAll(R ∪ {v}, P ∩ N(v), X ∩ N(v))
            P ← P \ {v}
            X ← X ∪ {v}
        end
    end
return

Function MaxCover(V, S, k)
    D ← ∅
    V' ← V
    for i ← 1 to k do
        C ← C' ∈ (S \ D), pro kterou platí, že |C' ∩ V'| je největší
        V' ← V' \ C
        D ← D ∪ {C}
    end
return D
```

3.2 EnumK

2. verze algoritmu používá funkci `CliqueAll`, ve které je pozměněn 2. řádek. Namísto předchozího přidání kliky do množiny všech klik se v tomhle algoritmu v paměti uchovává pouze k klik. V pseudokódu 8 jsou uchovávané kliky v množině D. Každá další vyhledaná klika C se podrobí zkoušce, jestli by nebylo výhodnější kliku C vyměnit za jinou kliku C' nacházející se v množině D. Pokud je výhodnější klika C, tak se z D vyjme klika C' (funkce `Delete` (C')) a přidá se do ní klika C (funkce `Insert` (C)).

Tento algoritmus opět prochází všechny kliky, ale narozdíl od algoritmu EnumAll (z předchozí kapitoly) má menší pamětovou složitost.

3.2.1 Pojmy

Pojmy vyskytující se v následujících algoritmech:

cov(D): *(Coverage)* vrací množinu všech uzlů vyskytujících se v některé z klik $C \in D$.

priv(C,D): *(Private-Node-Set)* kde $C \in D$, vrací množinu uzlů, které jsou v C a nejsou v ostatních klikách z D. Uzly, které náleží pouze jedné klice, budeme nazývat privátní uzly.

C_{min}(D): *(Min-Cover-Clique)* vrací kliku $C \in D$, která má nejmenší počet privátních uzlů.

rcov(v,D): *(Reverse-Coverage)* kde v je uzel z grafu G, vrací množinu klik z D, které obsahují uzel v.

rpriv(i,D): *(Reverse-Private-Node-Set)* kde $0 \leq i \leq |V|$, vrací množinu klik $C \in D$, které mají počet privátních uzlů rovno i.

3.2.2 PNP-Index

PNP-Index je struktura, která je postavená na množině klik D a grafu G, obsahující následující komponenty:

- $|priv(C)|$: počet privátních uzlů pro každou kliku z D.
- $rcov(v)$: pro každý uzel $v \in G$ si uchovává množinu klik, ve kterých se vyskytuje uzel v.
- $|cov|$: počet uzlů, které pokrývá D.
- C_{min}: kliku z D, která má nejmenší počet privátních uzlů
- $rpriv(i)$: pro $0 \leq i \leq |V|$, si uchovává množinu klik majících počet privátních uzlů rovno i.

Algoritmus 8: EnumK(graf $G = (V, E)$, int k)

$D \leftarrow \emptyset$; $|cov| \leftarrow 0$; $C_{min} \leftarrow \emptyset$
foreach $v \in V$ **do** $rcov(v) \leftarrow \emptyset$
for $i \leftarrow 0$ **to** $|V|$ **do** $rpriv(i) \leftarrow \emptyset$
`CliqueAll(`$\emptyset$, V, $\emptyset$`)/* 2. řádek (R je maximální klika)`
`   nahradíme funkcí CandMaintain(`R`).  */`
return D

Function `CandMaintain(`*klika* C`)`
  if $|D| < k$ **then**
    `Insert(`C`)`
  else
    $p_{new} \leftarrow |\{v \in C : |rcov| = 0$ nebo $(v \in C_{min}$ a $|rcov(v)| = 1)\}|$
    if $p_{new} > \left|priv(C_{min}) + \alpha * \frac{|cov|}{D}\right|$ **then**
      `Delete(`C_{min}`)`
      `Insert(`C`)`
    end
  end
return

Function `Delete(`*klika* C`)`
  $D \leftarrow D \setminus \{C\}$
  remove C from $rpriv(|priv(C)|)$
  foreach $v \in C$ **do**
    $rcov(v) \leftarrow rcov(v) \setminus \{C\}$
    if $|rcov(v)| = 0$ **then** $|cov| \leftarrow |cov| - 1$
    if $|rcov(v)| = 1$ **then**
      $C' \leftarrow$ maximální klika z $rcov(v)$
      $|priv(C')| \leftarrow |priv(C')| + 1$
      přesuň C' z $rpriv(|priv(C')| - 1)$ do $rpriv(|priv(C')|)$
    end
  end
return

Algoritmus 9: Pokračování EnumK

```
Function Insert(klika C)
    D ← D ∪ {C}
    |priv(C)| ← 0
    foreach v ∈ C do
        rcov(v) ← rcov(v) ∪ {C}
        if |rcov(v)| = 1 then
            |priv(C)| ← |priv(C)| + 1
            |cov| ← |cov| + 1
        end
        if |rcov(v)| = 2 then
            C′ ← klika z rcov(v) \ {C}
            |priv(C′)| ← |priv(C′)| − 1
            přesuň C′ z rpriv(|priv(C′)| + 1) do rpriv(|priv(C′)|)
        end
    end
    rpriv(|priv(C)|) ← rpriv(|priv(C)|) ∪ {C}
    for i ← 0 to |priv(C)| do
        if priv(i) ≠ ∅ then
            C_min ← libovolná klika z rpriv(i)
            break
        end
    end
return
```

3.3 EnumKOpt

Narozdíl od předchozích algoritmů se v tomhle algoritmu při výpočtu neprochází všechny kliky. Pro zredukování klik, se kterými se má počítat se využívají 3 strategie:

1. **Globální prořezávání** - V globálním prořezávání se pro každý uzel vypočítá hodnota $score(v)$, reprezentující potenciální maximální velikost kliky, obsahující uzel v. $score(v)$ (výpočet popsán níže) se poté dosadí do podmínky pro globální prořezávání, která je ve tvaru: $score(v) \leq |priv(C_{min})| + \alpha * \frac{|cov(D)|}{|D|}$. Doporučená hodnota pro α je 0.5.
2. **Lokální prořezávání** - V lokálním prořezávání potřebujeme stanovit podmínku, která zastaví expandování dané kliky. Používáme proto informace z množin R a P, kde R je sestavovaná klika a P je množina kandidátů, které mohou rozšířit kliku. Podmínka pro lokální prořezávání je ve tvaru $score(P_a \cup P_b) + |R_a \cup R_b| \leq |priv(C_{min})| + \alpha * \frac{|cov(D)|}{|D|}$.
3. **Počáteční výpočet kandidátů** - Dobrá počáteční množina kandidátů, může výrazně zlepšit chod algoritmu a naplnění podmínek pro globální a lokální prořezávání. Pro výpočet se používá hladový algoritmus `InitK`, který naplní D klikami tak, že velikost $|C| : \forall C \in D$ by měla být co největší a velikost $|C_i \cap C_j|$: pro každý pár, kde $C_i \in D$ a $C_j \in D$, by měla být co nejmenší. V `InitK` se vygeneruje $\eta * k$ klik ($\eta \geq 1$), které se nepřekrývají a vybere se z nich k největších.

Pojmy

V následujícím algoritmu se vyskytují pojmy z předchozího algoritmu a tyto pojmy:

colour(S) - Kde S je množina uzlů z V. Pro každý uzel se na začátku vypočítá algoritmem `Welsh-Powell-Colour-Graph` číslo, které značí barvu. $colour(S)$ vrací počet různých barev, které mají uzly z S.

core(v) - Kde v je uzel z V. Jádro uzlu se vypočítá pro všechny uzly algoritmem `Batagelj-Zaveršnik-Cores`.

score(v) - Kde v je uzel z V, využívá se v globálním prožezávání. $score(v)$ se vypočítá tak, že se vezme minimum z $core(v)$ a $color(\{v \cup N(v)\})$. Takže $score(v) = \texttt{min}\,(core(v), color(\{v \cup N(v)\}))$.

Algoritmus 10: *EnumKOpt*(graf $G = (V, E)$, int k)

```
D ← ∅;  |cov| ← 0;  C_min ← ∅
foreach v ∈ V do rcov(v) ← ∅
for i ← 0 to |V| do rpriv(i) ← ∅
D ← InitK(G, k)
P ← V;  R ← ∅;  X ← ∅;  u ← uzel z V s největším stupněm
foreach v ∈ V, v sestupném pořadí podle score(v) do
    if |D| = k a GlobalPruning(v) then break
    CliqueK(P ∩ N(v),R ∪ {v},X ∩ N(v))
    P ← P \ {v};  X ← X ∪ {v}
end
Function CliqueK(P, R, X)
    if P ∪ X = ∅ then CandMaintain(R)
    if LocalPruning(P, R) then return
    u ← uzel v ∈ P ∪ X, pro který je |P ∩ N(v)| největší
    foreach v ∈ P \ N(u) do
        CliqueK(P ∩ N(v), R ∪ {v}, X ∩ N(v))
        P ← P \ {v}; X ← X ∪ {v}
    end
return

Function InitK(graf G = (V, E), int k)
    S ← ∅; U ← ∅
    foreach v ∈ V, v sestupném pořadí podle score(v) do
        if |S| = η * k then break
        if v ∉ U then
            C ← CliqueGreedy(N(v), {v})
            S ← S ∪ {C}
            foreach u ∈ C do U ← U ∪ {u} ∪ N(u)
        end
    end
    M ← k klik s největším počtem uzlů
return M

Function CliqueGreedy(P, R)
    if P = ∅ then return R
    u ← uzel v ∈ P, pro který je min(|P ∩ N(v)|, score(v)) největší
return CliqueGreedy(P ∩ N(u), R ∪ {u})
```

Algoritmus 11: GlobalPruning a LocalPruning

Function `GlobalPruning(`v`)`

$result \leftarrow score(v) \leq |priv(C_{min})| + \alpha * \frac{|cov(D)|}{|D|}$

return *result*

Function `LocalPruning(`P, R`)`

$P_a \leftarrow P \setminus cov(D)$
$P_b \leftarrow P \cap priv(C_{min})$
$R_a \leftarrow R \setminus cov(D)$
$R_b \leftarrow R \cap priv(C_{min})$
$result \leftarrow score(P_a \cup P_b) + |R_a \cup R_b| \leq |priv(C_{min})| + \alpha * \frac{|cov(D)|}{|D|}$

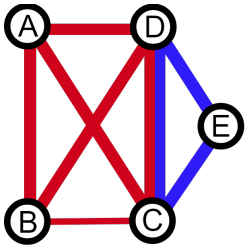
return *result*

Function `Welsh-Powell-Colour-Graph(`*graf* $G = (V, E)$`)`

$S \leftarrow V$; $i \leftarrow 1$
while $S \neq \emptyset$ **do**
  $v \leftarrow$ uzel z S s největším stupněm
  $S \leftarrow S \setminus \{v\}$
  $colour(v) \leftarrow i$
  foreach $u \in S \setminus N(v)$ **do** $colour(u) \leftarrow i$
  $S \leftarrow S \setminus N(v)$
  $i \leftarrow i + 1$
end
return

Function `Batagelj-Zaveršnik-Cores(`*graf* $G = (V, E)$`)`

$V' \leftarrow V$; $deg \leftarrow \emptyset$
foreach $v \in V$ **do** $deg(v) \leftarrow degree(v)$
while $deg \neq \emptyset$ **do**
  $v \leftarrow$ uzel $u \in deg$, který má $deg(v)$ největší
  $core(v) \leftarrow deg(v)$
  foreach $u \in N(v) \cap deg$ **do**
    if $deg(u) > deg(v)$ **then** $deg(u) \leftarrow deg(u) - 1$
  end
  $deg \leftarrow deg \setminus \{v\}$
end
return



Obrázek 10: Logo aplikace Kliky

4 Uživatelská příručka

Příručka je sepsaná pro uživatele a je v ní popsána struktura a funkčnost programu.

4.1 Logo Aplikace

Protože se v aplikaci hledají kliky, tak je tento fakt promítnut i v logu, které se skládá z grafu obsahujícího pět uzlů. V grafu se vyskytují dvě kliky, které jsou vyznačené červenou a modrou barvou.

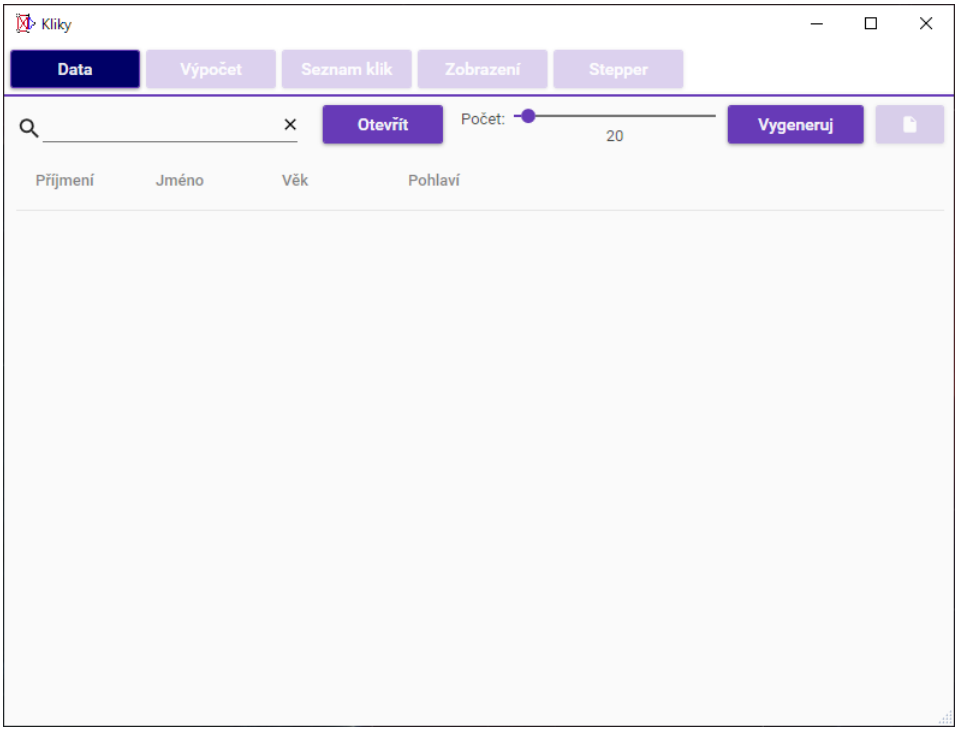
4.2 Struktura programu

Program se skládá z hlavního okna, to obsahuje v horní části ovládací tlačítka, kterými se přepíná mezi jednotlivými stránkami aplikace. Dále hlavní okno obsahuje prostor zabírající většinu plochy, ve které se vyskytuje obsah jednotlivých stránek.

4.2.1 Stránka Data

Na stránce *Data* máme možnost prohlédnout si s jakými osobami se aktuálně pracuje. Největší plochu stránky zaujímá seznam s informacemi o osobách. V levém horním rohu stránky se vyskytuje vyhledávací pole. Po zapsání vyhledávaného jména nebo příjmení se vypíší hledané osoby. Po vymazání vyhledávacího pole se seznam obnoví do původního stavu před hledáním, pro rychlou obnovu do původního stavu se využije tlačítko s křížkem, které smaže veškerý obsah ve vyhledávacím poli.

Vedle vyhledávacího pole se nachází tlačítko *Otevřít* a po jeho stisknutí se objeví vyskakovací okno *Otevřít*, jehož funkčnost je popsána níže. Vedle tlačítka *Otevřít* se nachází ovládací prvky, pomocí kterých si může uživatel vygenerovat testovací soubory s osobami a vztahy, které se následně otevřou v aplikaci. Slidebarem se vybere počet osob, které se mají vygenerovat a stistkne se tlačítko *Vygeneruj*, které vygeneruje daný počet osob a vztahy mezi nimi. Zároveň



Obrázek 11: Aplikace po spuštění

se zpřístupní tlačítko s ikonkou souboru, kterým se vygenerované soubory mohou otevřít (v defaultní aplikaci pro práci s csv soubory) a případně uložit na uživatelem zvolené místo pro pozdější použití.

4.2.2 Okno Otevřít

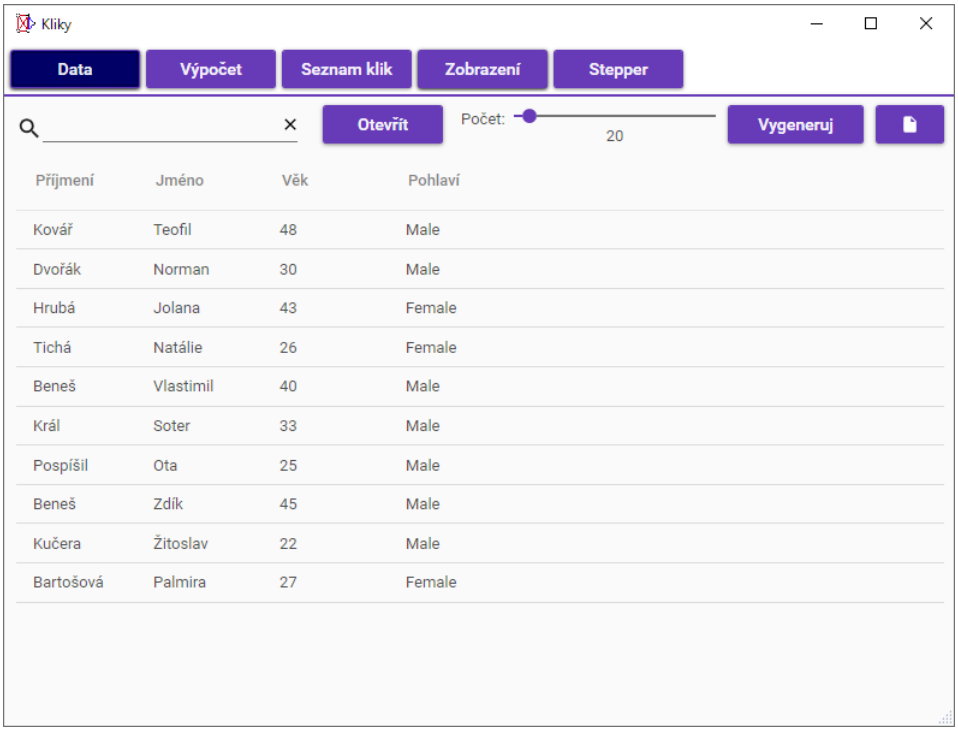
V okně *Otevřít* se vybere cesta k souborům (ve formátu csv) s informacemi o osobách a vztazích. Po kliknutí na tlačítka *Najít...* se vybere v adresáři vždy jeden soubor. Prvním tlačítkem *Najít...* se vybírá soubor s osobami a druhým tlačítkem *Najít...* se vybírá soubor se vztahy. Pokud oba soubory obsahují hlavičku, je nutné zatrhnout políčko *Soubory s hlavičkou*, jinak aplikace nebude fungovat správně. Po zadání obou cest se akce otevření potvrdí tlačítkem *Potvrdit*. Po stisknutí tlačítka *Storno* se okno zavře a současné osoby nebudou změněny.

Pokud se v některém souboru objeví chyba, díky které se nepodaří data správně načíst, tak je do souboru s chybami zapsán řádek, který tuto chybu zavinil. Soubor se po dokončení načítání zobrazí.

4.2.3 Stránka Výpočet

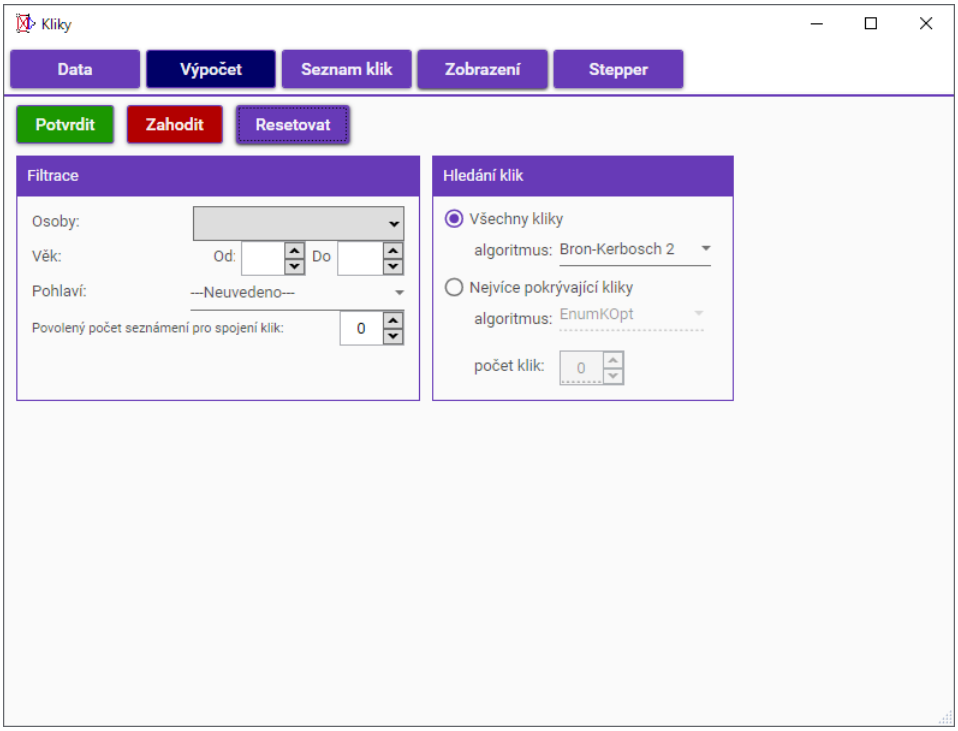
V této stránce se filtrují osoby a volí se algoritmus, kterým se kliky hledají.

Sekce *Filtrace* obsahuje prvky pro výběr osob, omezení věku, pohlaví a povolený počet seznámení pro spojení klik. Zaškrtnuté osoby se budou vyskytovat



Obrázek 12: Stránka Data

Obrázek 13: Okno Otevřít



Obrázek 14: Stránka Výpočet

spolu v jedné klice.

V sekci *Hledání klik* je na výběr ze dvouch možností:

- *Výběr všech klik* - Vylistují se všechny maximální kliky. Lze vybrat, kterým algoritmem se kliky budou hledat.
- *Nejvíce pokrývající kliky* - Vylistují se jen ty kliky, které pokrývají nejvíce osob. Zde lze vybrat algoritmus a maximální počet vylistovaných klik.

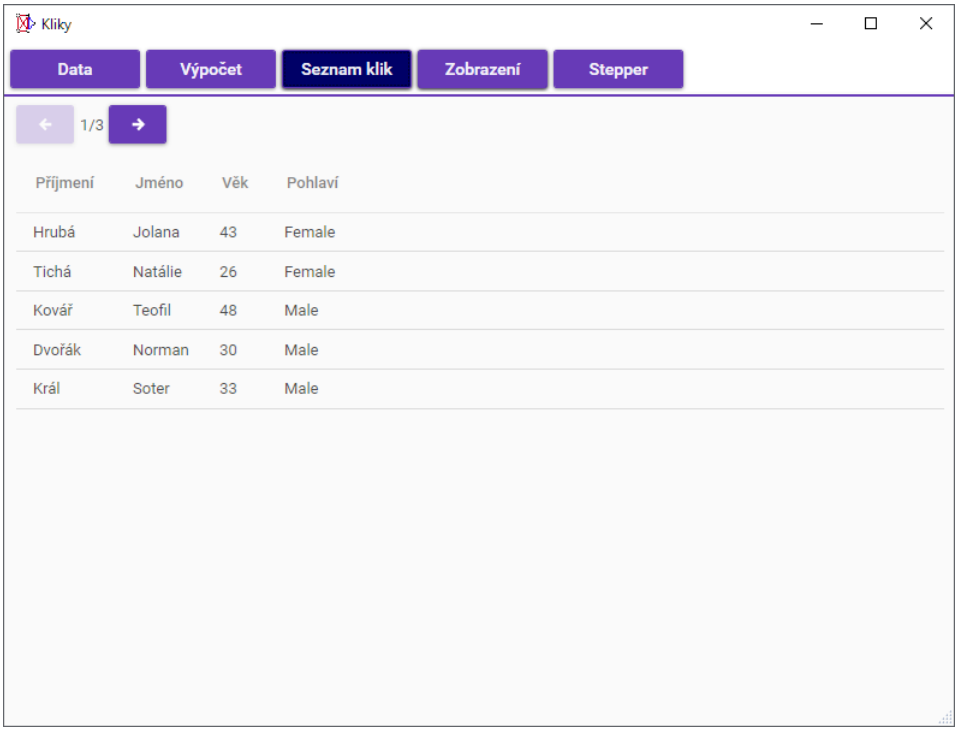
Tlačítkem *Potvrdit* se uloží změny. Tlačítkem *Zahodit* se změny vrátí do stavu, v jakém byly při otevření této stránky. Tlačítem *Resetovat* se nastavení na této stránce vrátí do počátečního nastavení.

4.2.4 Stránka Seznam klik

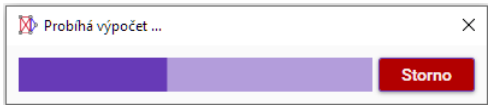
Na této stránce se prochází nalezenými klikami. V seznamu klik se pohybujeme tlačítky s šipkami, mezi kterými se vyskytuje před lomítkem číslo udávající kolikátou kliku prohlížíme a za lomítkem se vyskytuje číslo s celkovým počtem všech nalezených klik.

4.2.5 Stránka Zobrazení

Na této stránce si můžeme prohlédnout grafickou reprezentaci grafu. V levém horním rohu stránky se nachází přepínač, který dává na výběr mezi zobrazením



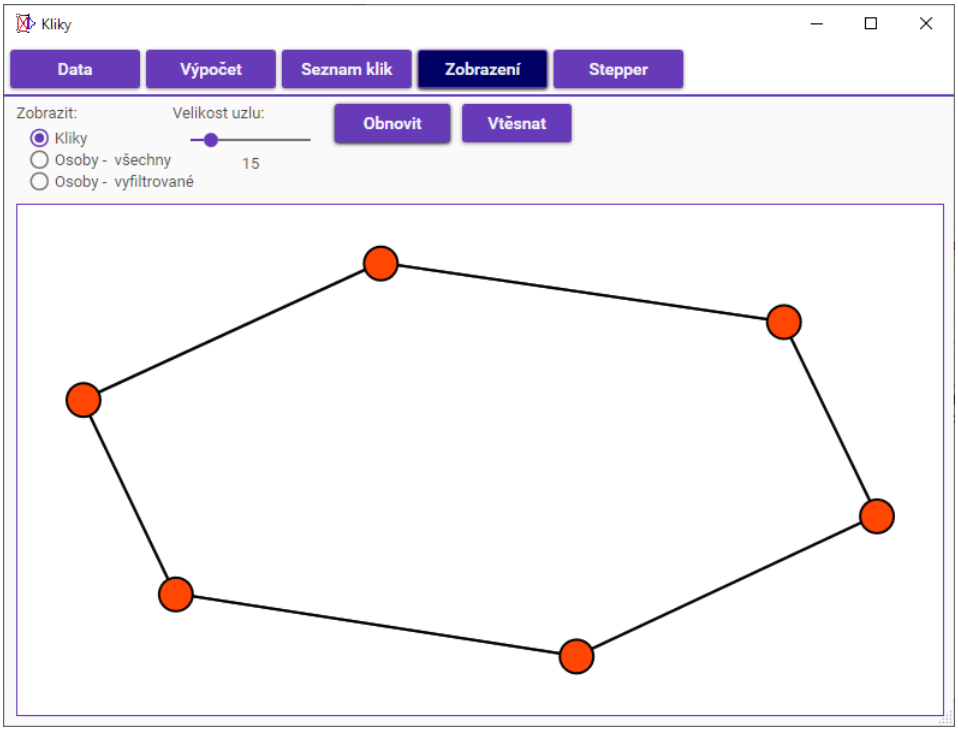
Obrázek 15: Stránka Seznam klik



Obrázek 16: Okno Probíhá výpočet...

nalezených klik, zobrazením všech osob a zobrazením všech osob, které byly vyfiltrovány. Varianta s vyfiltrovanými osobami je vykreslena s doplněnými vztahy, které jsou značeny hranou modré barvy. Graf s osobami uvnitř uzlu obsahuje jméno a příjmení osoby. Pokud se změnily data, tak se budou muset překreslit všechny varianty. Pokud se od posledního prohlížení grafu změnily nalezené kliky, pak se bude muset překreslit graf s klikami a graf s filtrovanými osobami. Při velkém počtu klik nebo osob se objeví vyskakovací okno s otázkou, zdali se má vykreslit graf. Při kliknutí na *Ne* se bude muset znovu zvolit, kterou variantu zobrazit. Pokud počet uzlů není velký, zobrazí se rovnou vyskakovací okno *Probíhá výpočet*, ve kterém máme možnost ukončit výpočet stisknutím tlačítka *Storno* nebo křížkem v pravém horním rohu.

Vedle ovládacích prvků pro zvolení varianty grafu se nachází ovládací prvkek, kterým se mění velikost uzlu. Počáteční hodnota je 15. Vedle se nachází tlačítko *Obnovit*, po jehož stisknutí se vypočítají nové polohy uzlů a vše se vycentruje na střed.



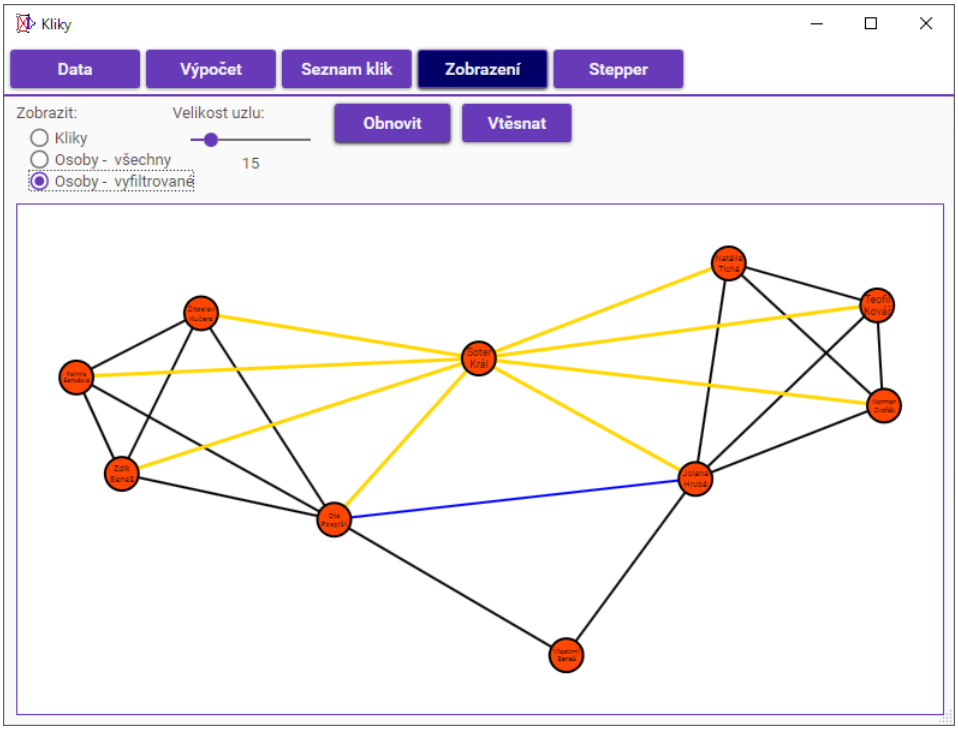
Obrázek 17: Stránka Zobrazení - kliky

Pokud by při interakci s grafem došlo k posunutí mimo plátno, namísto tlačítka *Obnovit* je možnost stisknout tlačítko *Vtěsnat*, které graf také vycentruje na střed, ale nevypočítávají se nové polohy uzlů.

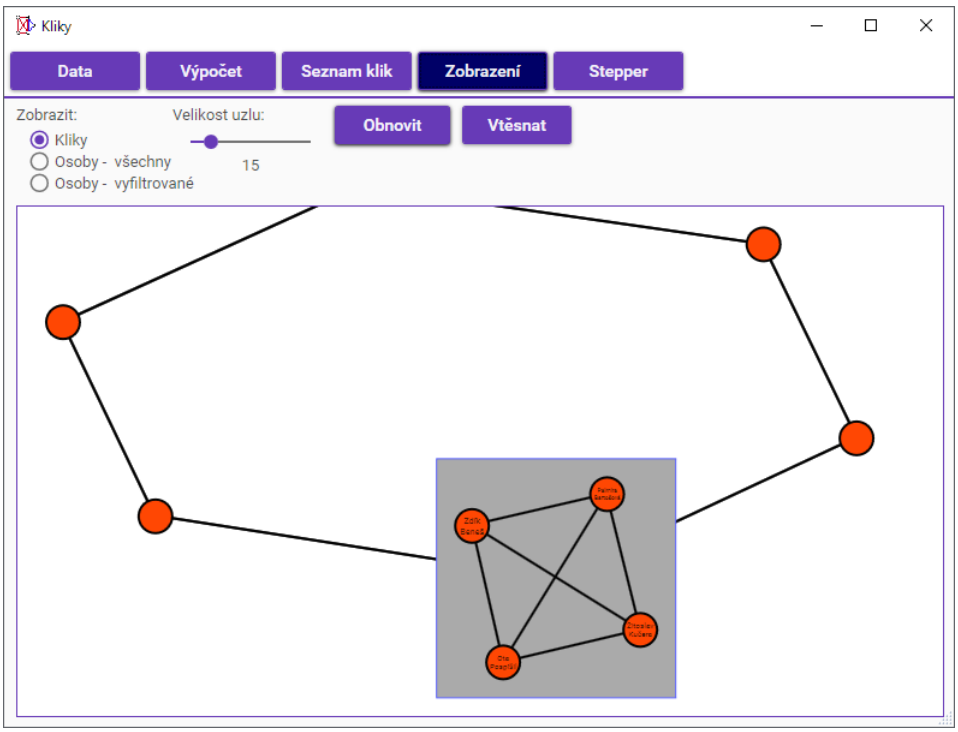
Interakce s plátnem

Pro prohlížení grafu slouží interakce s plátnem, které se liší v závislosti na zvolené variantě.

1. Ve **všech variantách** lze grafem posouvat do stran, přibližovat nebo oddalovat graf a zvýrazňovat hrany vedoucí do uzlu.
   - Posouvání se provede podržením levého tlačítka kdekoliv na plátně a posunutím myši.
   - Přiblížení nebo oddálení grafu se provede kolečkem myši.
   - Zvýraznění hran vedoucích z daného uzlu se provede tak, že se klikne pravým tlačítkem myši na uzel. Při najetí myši z uzlu zpátky mimo uzel se hrany opět vrátí do původního stavu.
2. Ve variantách pro **vykreslení osob** se při najetí myší nad uzel změní kurzor myši na šipku s otazníkem a objeví se okénko s informacemi o dané osobě.



Obrázek 18: Stránka Zobrazení - zvýraznění hran



Obrázek 19: Stránka Zobrazení - informační okno

3. Ve variantě pro **vykreslení klik** máme více interakcí:

   - Při najetí myši na hranu se změní kurzor a zobrazí se okénko s informací o společném jádře, to jsou osoby, které obsahují obě kliky spojené danou hranou.
   - Při kliknutí levým tlačítkem myši na uzel se uzel překryje informačním oknem, ve kterém jsou vykresleny osoby v klice, která představovala daný uzel. Toto okno umožňuje stejné interakce jako varianta pro vykreslení osob. Těchto oken můžeme otevřít i více klikáním na další uzly.
   - Při kliknutí levým tlačítkem myši na informační okno se toto okno zobrazí nad ostatními informačními okny, které ho předtím překrývaly.
   - Pokud se klikne levým tlačítkem myši mimo uzel a mimo informační okno, tak zmizí všechny informační okna.

4.2.6 Stránka Stepper

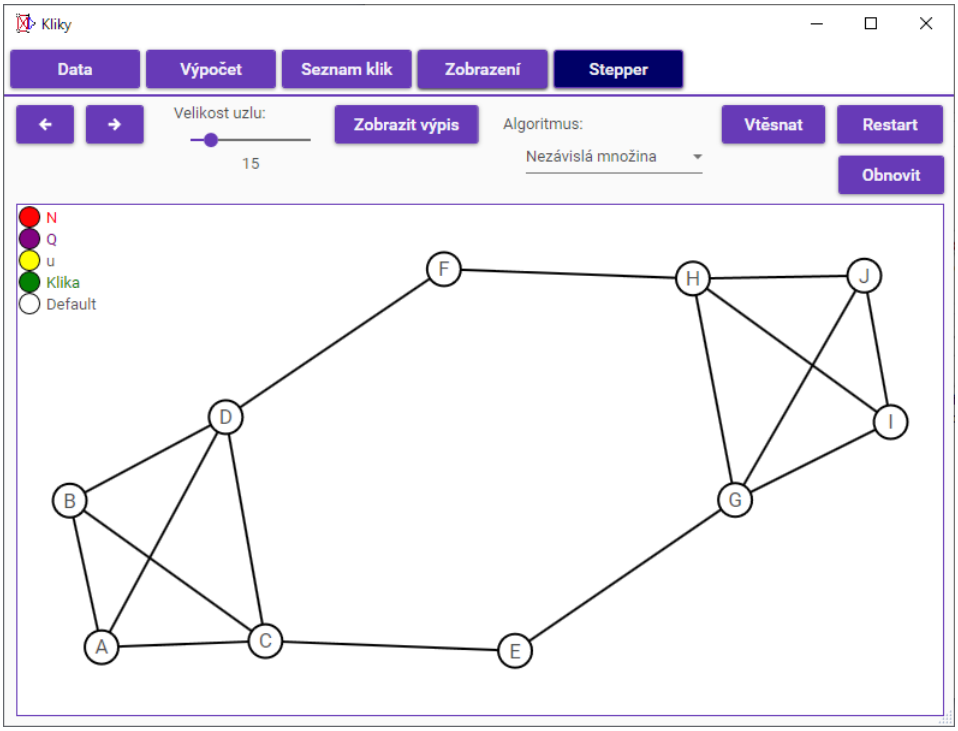
Jednotlivé algoritmy se skládají z cyklů a rekurzivních volání. Když procházíme algoritmem na určitém příkladu, značíme si stavy proměnných s tím, jak algoritmus prochází cykly nebo provádí rekurzivní volání. Stránka *Stepper* umožňuje zobrazit průchod algoritmem pro hledání klik.

Rozložení stránky

Největší část stránky zabírá plátno, ve kterém je v levém horním rohu legenda a ve zbytku graf. Interakce s plátnem je stejná jako na stránce *Zobrazení* s variantou vykreslení osob. Legenda obsahuje kružnice vyplněné barvou a vysvětlivky, o které proměnné se jedná. Uzly jsou obarveny podle toho, v jaké proměnné se aktuálně vyskytují.

V horním pásmu stránky se vyskytují ovládací prvky. Tlačítky s šipkami se prochází algoritmem. Posuvníkem se upravuje velikost uzlů. Po stisknutí tlačítka *Zobrazit výpis* se zobrazí nové okno s textovou reprezentací průchodu algoritmem, kde je jeden řádek značí jeden stav průchodu. Zvýrazněný řádek v okně s textem značí aktuální pozici průchodu. V jednom řádku je napsáno, která funkce se volá, popřípadě které uzly jsou v jaké proměnné. Například pokud je v textu $u := A$, znamená to, že v proměnné u se vyskytuje uzel A. S klikáním na tlačítka s šipkami se aktuální řádek mění. Vedle tlačítka *Zobrazit výpis* se vyskytuje prvek umožňující vybrat algoritmus, u kterého se bude reprezentovat průchod.

Tlačítkem *Vtěsnat* se graf pouze vycentruje na střed. Tlačítkem *Restart* se graf vycentruje na střed a průchod začne od začátku. Tlačítko *Obnovit* vypočítá nové polohy uzlů a hran a průchod začne od začátku.



Obrázek 20: Stránka Stepper

Vysvětlení ke spojení klik

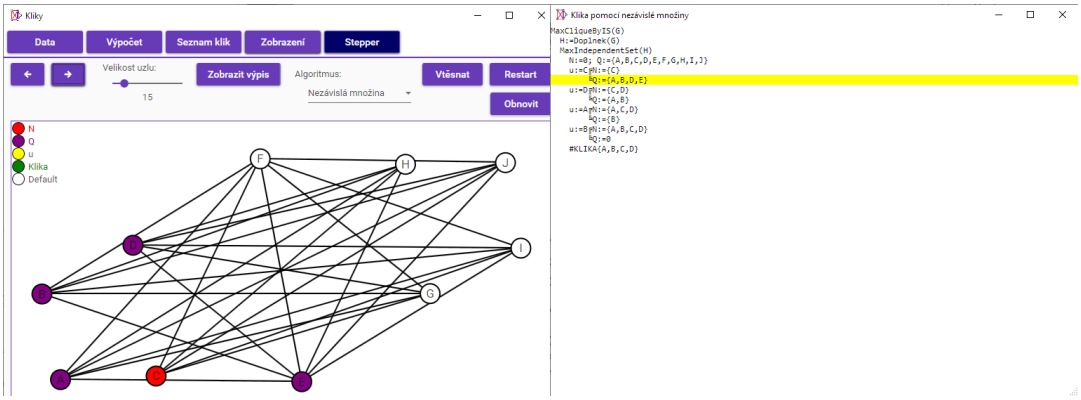
Pokud se na stránce *Výpočet* zvolil počet seznámení pro spojení klik, počet nalezených klik ve *Stepperu* se nebude shodovat s počtem vylistovaných klik na stránce *Seznam klik* nebo s počtem vykreslených klik na stránce *Zobrazení* (při zvolené variantě pro vykreslení klik). Počty se liší, protože se nejdříve naleznou všechny kliky a až poté se vyfiltrují kliky, které by se mohly spojit, spojí se a doplní se vztahy. Na stránce *Stepper* se nám tudíž prezentuje pouze hledání všech klik, nikoliv filtrování.

4.3 Vstup (import)

V okně *Otevřít* máme možnost vybrat dva soubory ve formátu csv:

- soubor s osobami,
- soubor se vztahy.

Formát csv (*Comma Separated Values*, česky čárkou oddělené údaje) CSV je formát, který jednoduše reprezentuje tabulková data. Každý řádek v dokumentu reprezentuje jeden řádek v tabulce. Sloupce jsou v dokumentu odděleny oddělovačem, v našem případě je to znak ; (středník). Dokument má koncovku



Obrázek 21: Stránka Stepper s oknem pro zobrazení výpisu

.csv a může být upravován v poznámkovém bloku, ale i v sofistikovanějším nástroji (například Microsoft Excel), který reprezentuje data jako tabulku, tudíž umožňuje jednodušší orientaci v datech a pohodlnější manipulaci s nimi.

Formát souboru s osobami Tabulka s osobami má 5 sloupců, zleva:

1. **ID** - Identifikační číslo osoby, pod kterým se reprezentuje osoba v tabulce se vztahy. Číslo musí být pro každou osobu unikátní, jinak hrozí špatné načtení dat.
2. **Jméno** - Jméno osoby.
3. **Příjmení** - Příjmení osoby.
4. **Věk** - Věk osoby, kladné číslo.
5. **Pohlaví** - Pohlaví osoby. Políčko musí obsahovat hodnotu Muž nebo Žena.

Jméno a příjmení osob se nedoporučuje psát příliš dlouhé, protože by při vykreslení grafu nemuselo být jméno a příjmení viditelné.

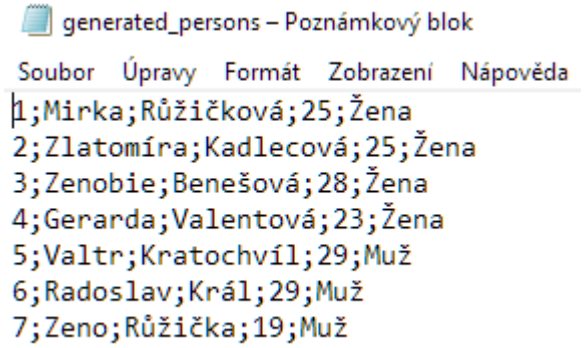
Nepovinná hlavička je ve tvaru: ID;Jméno;Příjmení;Věk;Pohlaví .

Formát souboru se vztahy Tabulka se vztahy má 2 sloupce, zleva:

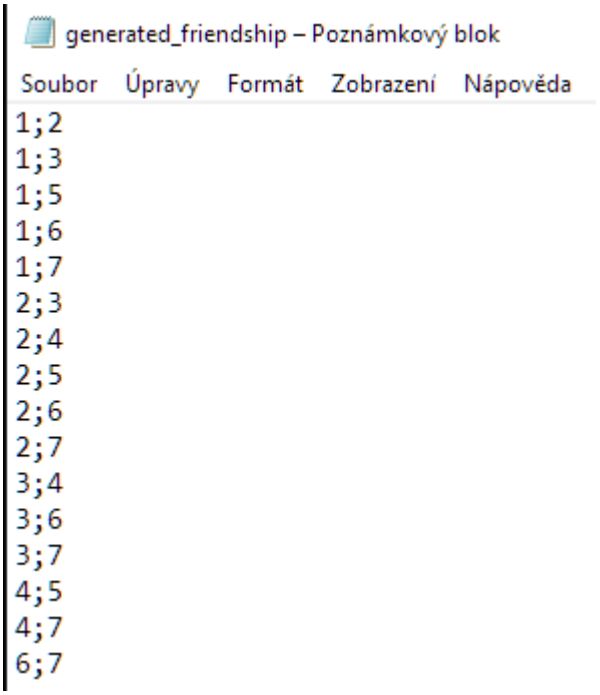
1. **ID1** - Identifikační číslo první osoby.
2. **ID2** - Identifikační číslo druhé osoby.

Pokud popíšeme první vztah jako „2;3“, tak tím dáváme najevo, že se osoba s ID 2 zná s osobu s ID 3. Řádek „3;2“ již do souboru se vztahy nemusíme uvádět, protože vztah těchto dvou osob už je zaznamenaný, ale pro přehlednost ho tam můžeme nechat.

Nepovinná hlavička je ve tvaru: ID1;ID2 .



Obrázek 22: Příklad souboru s osobami



Obrázek 23: Příklad souboru se vztahy

4.4 Instalace aplikace

Po spustění souboru *setup.exe* se zobrazí průvodce instalací. Pro úspěšnou instalaci je nutné projít jednotlivými fázemi instalace:

1. fáze Uvítání průvodcem, po přečtení stisknout tlačítko *Další*.

2. fáze Výběr instalační položky.

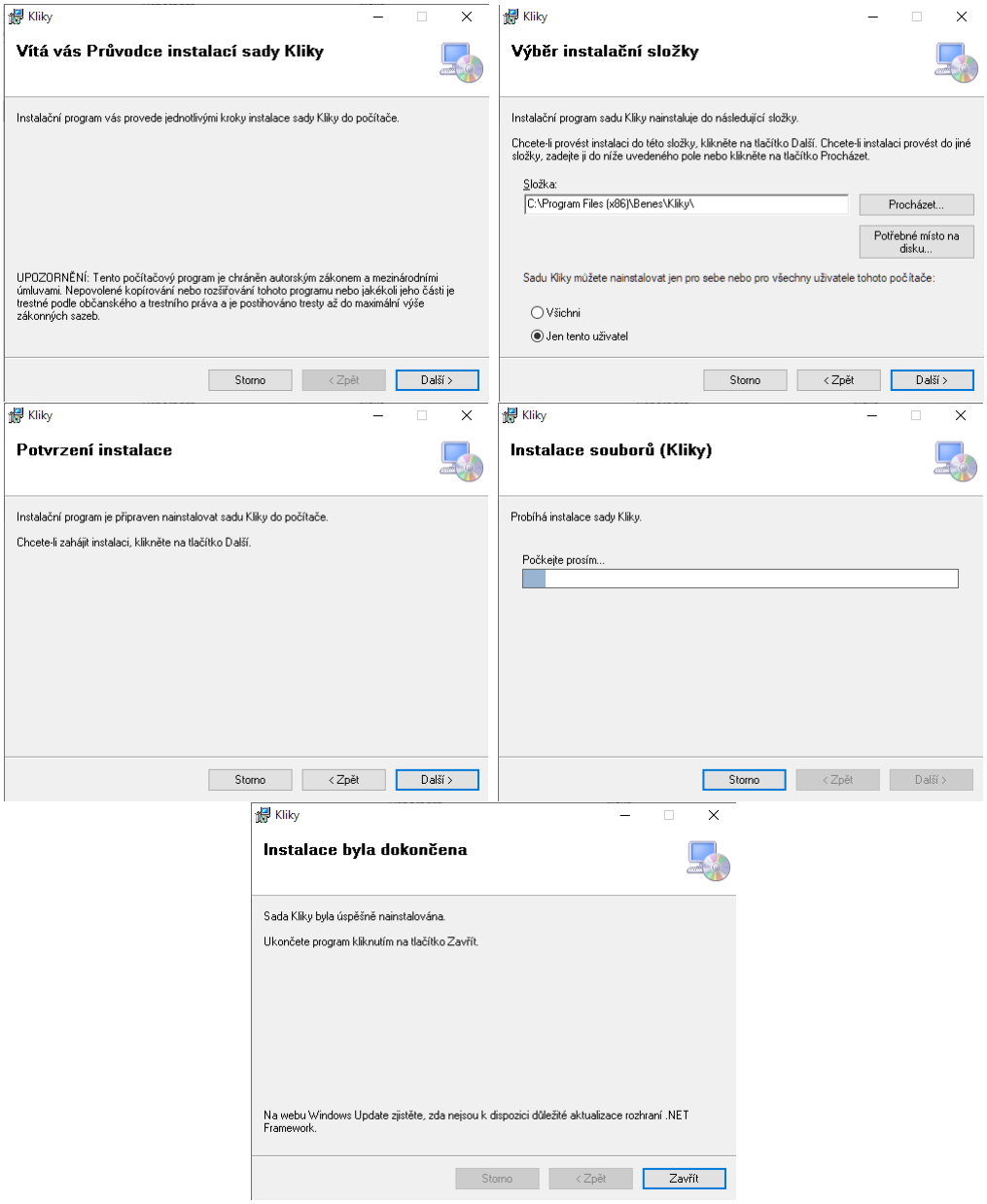
- Tlačítkem *Procházet...* vybrat složku, kam se má aplikace nainstalovat. Pokud chceme aplikaci nainstalovat do defaultní složky, výběr složky přeskočíme.
- Vybereme, jestli aplikaci zpřístupnit pouze pro aktuálního uživatele, nebo pro všechny uživatele.

3. fáze Potvrzení instalace - po přečtení a souhlasu stisknout tlačítko *Další*.

4. fáze Instalace aplikace

5. fáze Instalace byla dokončena, stisknutím tlačítka *Zavřít* ukončíme instalátor aplikace.

V některých bodech instalace se můžeme vrátit o krok zpět (tlačítko *Zpět*) nebo instalaci zrušit (tlačtko *Storno*).



Obrázek 24: Kroky instalace

5 Programátorská část

5.1 Architektura programu

Sekce obsahující popis hlavních tříd aplikace.

5.1.1 Sekce výpočtu klik

Třída Cliques_algorithm

Je třída, která obsahuje implementace algoritmů, pro hledání klik.

- **BronKerbosch1** - Základní verze algoritmu.
- **BronKerbosch2** - Algoritmus s využitím pivota.
- **BronKerbosch3** - Algoritmus využívající uspořádání uzlů.
- **Tomita** - Algoritmus od Tomita, Tanaka a Takahashi.
- **MaxCliqueByIS** - Vypočtení maximální kliky převodem na výpočet maximální nezávislé množiny v doplňkovém grafu.
- **EnumK** - Základní algoritmus pro výpočet top-k diversifikovaných klik.
- **EnumKOpt** - Optimalozovaný algoritmus pro výpočet top-k diversifikovaných klik.

Třída Graf

Je třída pro reprezentaci grafu. Uchovává v sobě seznam uzlů a matici sousednosti. Umí vrátit doplněk grafu, seznam uzlů v degeneracy uspořádání, přiřadit uzlům color number a core number.

Třída Uzel

Třída reprezentující uzel v grafu. Obsahuje id uzlu, stupeň uzlu, color number, core number a seznam sousedících uzlů.

5.1.2 Sekce pro práci s daty

Třída Person

Třída reprezentující osobu, která uchovává následující informace: ID (číslované od 0 kvůli přístupu do matice sousednosti), jméno, příjmení, věk a pohlaví. Dále třída obsahuje metodu pro vygenerování instance třídy Person z řádku v csv souboru.

Třída Clique

Tato třída reprezentuje kliku a uchovává informace o osobách obsažených v klice.

Třída Methods

Má atributy Persons (seznam osob) a Friendships (matice sousednosti). Třída obsahuje metody, které mají podobný charakter jako databáze a metody pro hledání v klikách.

coreOfCliques - Metoda bere seznam klik a vrací seznam osob, které mají kliky společné.

additionCliques - Metoda má jako parametr seznam klik a vrací dvojice osob, které by se měly seznámit, aby se kliky mohly spojit v jednu kliku.

CliquesWithMaxCore - Metoda bere index i a seznam klik. Ke klice na indexu i vrací seznam, ve kterém jsou seznamy klik, které mají stejné jádro a toto jádro je největší.

NumberOfSeznameni - Metoda bere dvě kliky a vrací číslo, které udává počet seznámení nutných pro spojení klik v jednu kliku.

CliqueListGroup - CliqueGroup ze seznamu klik a čísla k rozdělí kliky na skupiny, kde v každé skupině je seznam klik, pro které je potřeba méně nebo stejně seznámení jako k.

CliqueGroup - Ze seznamu klik a čísla k rozdělí kliky na skupiny pomocí metody CliqueListGroup a poté kliky v každé skupině sjednotí v jednu kliku. Vrací seznam sjednocených klik.

Třída Data

Třída Data v sobě uchovává informace, které mohou být potřeba z různých míst. Jsou tu uloženy vypočtené kliky, aby se nemuseli znovu počítat, když například uživatel přepne ze stránky se seznamem klik na stránku s vykreslením klik. Dále je tu uložena informace o použitém algoritmu, kterým se mají kliky počítat a instance třídy Methods.

Obsahuje metody na dotazování, zdali se má vykreslit daný počet osob nebo klik.

Třída GenerateData

Třída, která poskytuje funkcionalitu pro vygenerování náhodných osob a vztahů.

Konstruktor bere jako parametry cesty k souborům, kam se mají uložit vygenerované osoby a vztahy. Dále konstruktor bere parametr cesty ke složce, ve které musí být uloženy soubory:

- *mJmena.txt* - Soubor s mužskými jmény. Jména jsou oddělena čárkou.
- *zJmena.txt* - Soubor s ženskými jmény. Jména jsou oddělena čárkou.
- *prijmeni.txt* - Soubor s příjmeními. Na každém řádku je mužský tvar příjmení a ženský tvar příjmení oddělené od sebe čárkou.

V konstruktoru se načtou jména a příjmení ze souborů a metoda Generate, která bere počet osob a pravděpodobnost, s jakou se osoby znají, vygeneruje nebo přepíše soubory s osobami a vztahy zadané v konstruktoru.

5.1.3 Sekce kreslení grafu

Třída Vertex

Třída pro uchování informací pro uzel, který má být vykreslen. Uchovává v sobě:

- informace potřebné pro výpočet,
- pozici uzlu určenou výpočtem,
- typ - klika, osoba,
- objekty, které reprezentují graf na plátně (kružnice, popisek),
- seznam hran vedoucích z uzlu.

Třída má metody:

Draw - Metoda pro vykreslení uzlu na plátně.

DrawWithText - Metoda pro vykreslení uzlu na plátně s popiskem.

Třída Spring

Třída určená k uchování dat o hraně. Uchovává v sobě:

- reference na koncové uzly,
- informace určené pro vykreslení - barva, tloušťka,
- objekt přímky, která reprezentuje hranu na plátně.

Třída má metody:

Draw - Metoda pro vykreslení hrany na plátně.

Update - Metoda zavolaná po změnění poloh jednoho z uzlů.

Třída DrawGraph

Třída pro vykreslení grafu, která má dva konstruktory. Oba mají parametry výšku a šířku plátna a liší se v tom, že jeden vykresluje kliky a druhý vykresluje osoby. V obou konstruktorech se inicializují uzly a hrany.

Třída má metody:

Calculate - Metoda, která každému uzlu vypočítá polohu.

getComplementSprings - Metoda, která k uzlům a hranám vrátí seznam hran, které se vyskytují v dopňkovém grafu.

Třída CanvasMethods

Třída, ve které se plátnu přiřadí funkcionalita pro interakci s grafem jako například:

- posunutí grafu,
- přiblížení a oddálení grafu,
- změna velikosti uzlů,
- zvýraznění hran vedoucích z daného uzlu,
- posunutí grafu na střed.

5.1.4 Sekce Stepperu

Třída Stepper

Třída, která na plátně reprezentuje průchod algoritmem. Obsahuje informace o plátně, na kterém se má kreslit a metody Next a Previous, kterými se přepíná na následující nebo předchozí stav. Z této třídy dědí třídy StepperBK1, StepperBK2, StepperBK3, StepperIS a StepperTomita, které implementují reprezentaci průchodem algoritmu pro algoritmy Bron-Kerbosch verze 1-3, algoritmus pro vyhledání maximální kliky za pomocí nezávislých množin a algoritmus Tomita-Tanaka-Takahashi.

Okno StepperText

Okno, ve kterém se vypíše průchod algoritmem podobně jako v příkladě průchodem algoritmu Bron-Kerbosch1 2.5.1. Z tohoto okna jsou odvozena okna StepperTextBK1, StepperTextBK2, StepperTextBK3, StepperTextIS, StepperTextTomita, které v sobě implementují metodu pro vygenerování průchodu daným algoritmem. Okno navíc obsahuje metodu `UpdateLine(int index)`, která zvýrazní řádek na daném indexu.

Třídy StepperText*

Jsou to třídy, které pro zadaný graf vygenerují text o průchodu algoritmem (algoritmy jsou stejné jako ve třídě Stepper). Tento text se vypíše v okně StepperText. Konkrétně jsou to třídy:

- StepperTextBK1,
- StepperTextBK2,
- StepperTextBK3,
- StepperTextIS,
- StepperTextTomita.

5.2 Použité technologie

Sekce obsahující informace o použitých technologiích.

5.2.1 .NET Framework

Je zastřešující název pro soubor technologií v softwarových produktech. Tato softwarová platforma je určena pro vývoj aplikací na Windows, webových aplikací, aplikací pro mobilní zařízení a jiné. Pro vývoj jsou nejpoužívanější programovací jazyky:

- C#,
- Visual Basic .NET,
- F#,
- J#,
- IronPython,
- Object Pascal,
- Boo.

Kód v daném programovacím jazyce se přeloží do mezijazyka Common Intermediate Language a následně je spuštěn v samostatném běhovém prostředí CLR (Common Language Runtime).

.NET Framework poskytuje sadu základních knihoven, které poskytují například základní třídy a funkce, datové struktury a kolekce, práce se souborovým systémem, funkce pro práci s vlákny a další.

5.2.2 Jazyk C#

C# je moderní, mnohoúčelový objektově orientovaný jazyk vyvinutý firmou Microsoft s platformou .NET Framework. Je založen na jazycích C++ a Java a lze ho použít k tvorbě databázových programů, webových aplikací a stránek, formulářových aplikací ve Windows, softwaru pro mobilní zařízení a další.

5.2.3 WPF

Je knihovna tříd, kterými se v .NET frameworku od verze 3.0 tvoří grafické rozhraní. Pro vytváření grafického rozhraní využívá značkovací jazyk XAML, který umožňuje oddělit funkčnost od vzhledu aplikace.

5.2.4 XAML

Je značkovací jazyk (obdova HTML), který se využívá k popisu grafického rozhraní v aplikacích společnosti Microsoft a je založený na XML. Vše, co se popíše jazykem XAML se dá popsat i pomocí standartních .NET jazyků. Výhodou XAML je ale velká jednoduchost.

Závěr

Cílem práce bylo pojednat o problematice hledání klik v grafu a příbuzných problémech. Práci jsem se rozhodl programovat v jazyce C#, protože s ním mám nejvíce zkušeností a dobře se mi v něm programuje.

Nejprve jsem začal budovat knihovnu s algoritmy pro hledání klik v grafech. Poté jsem se začal pracovat na aplikaci, ve které bych tuto knihovnu použil. Pouhé vypsání vyhledaných klik bylo nepřehledné a nic nevypovýdající, proto jsem do aplikace zapracoval grafickou reprezentaci grafu, aby si uživatel mohl prohlédnout graf osob i vyhledaných klik. Při větších datech začalo být klik mnoho, proto jsem do aplikace doprogramoval algoritmy související s hledáním diverzifikovaných klik.

Díky této práci jsem nabral spoustu zkušeností do budoucna a jsem za to nesmírně rád.

Conclusions

The aim of the work was to discuss the issue of finding cliques in the graph and related problems. I decided to program my work in C# language because I have the most experience with it and I have good programming in it.

First I started to build a library with algorithms for finding cliques in graphs. Then I went to the application where I would use this library. Simply listing the search clicks was confusing and unpredictable, so I incorporated into the application graphical representation of the graph, so that the user can view the graph of people and search cliques. With larger data, the cliques became too many, so I programmed algorithms related to diversified cliques search.

Thanks to this work I have gained a lot of experience for the future and I am extremely glad for it.

A Obsah přiloženého CD/DVD

bin/
: Instalátor SETUP.EXE programu KLIKY.

data/
: Ukázková a testovací data.

doc/
: Text práce ve formátu PDF a všechny soubory potřebné pro vygenerování PDF dokumentu.

src/
: Kompletní zdrojové texty programu KLIKY.

readme.txt
: Instrukce pro instalaci a spuštění programu KLIKY.

Literatura

[1] DEMEL, Jiří. Grafy a jejich aplikace. Vyd. 2., (Vlastním nákladem 1.). Libčice nad Vltavou: J. Demel, 2015.

ISBN 978-80-260-7684-1.

[2] MATOUŠEK Jiří a Jaroslav NEŠETŘIL. Kapitoly z diskrétní matematiky. 3., upr. a dopl. vyd. V Praze: Karolinum, 2007.

ISBN 978-80-246-1411-3.

[3] VEČERKA, Arnošt. Grafy a grafové algoritmy.

Dostupné z: https://1url.cz/LMg1B

[4] DIESTEL, Reinhard. Graph theory.

Dostupné z: http://www.esi2.us.es/ mbilbao/pdffiles/DiestelGT.pdf

[5] TOMITA Etsuji, TANAKA Akira a TAKAHASHI Haruhisa. The worst-case time complexity for generating all maximal cliques and computational experiments.

Dostupné z: https://1url.cz/xMg1p

[6] WEBOVÁ STRÁNKA WIKIPEDIA. Mnohojazyčná webová encyklopedie se svobodným (otevřeným) obsahem.

Dostupné z: https://cs.wikipedia.org/

[7] YUAN Long, QIN Lu, LIN Xuemin, CHANG Lijun, ZHANG Wenjie. Diversified Top-K Clique Search.

Dostupné z: https://1url.cz/rMg1G

[8] WELSH-POWELL. Algoritmus na barvení grafu.

Dostupné z: https://1url.cz/MMg1l

[9] BATAGELJ Vladimir a ZAVERŠNIK Matjaž. An $O(m)$ Algorithm for Cores Decomposition of Networks.

Dostupné z: http://vlado.fmf.uni-lj.si/pub/networks/doc/cores/cores.pdf

[10] FRICK Arne, LUDWIG Andreas a HEIKO Mehldau. A Fast Adaptive Layout Algorithm for Undirected Graphs.

Dostupné z: https://1url.cz/BMg1F

[11] OTEVŘENÁ DATA V ČR. Formát csv.

Dostupné z: https://opendata.gov.cz/standardy:csv

[12] DOTNETPORTAL. Webový magazín zaměřený na vývoj aplikací.

Dostupné z: https://1url.cz/3Mg1R