UNIVERZITA PARDUBICE
Fakulta elektrotechniky a informatiky

Metodika výpočtu všech podgrafů síťového grafu s uplatněním omezujících podmínek

Štěpán Karták

Bakalářská práce
2011

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: **2010/2011**

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Štěpán KARTÁK**
Osobní číslo:	**I08077**
Studijní program:	**B2646 Informační technologie**
Studijní obor:	**Informační technologie**
Název tématu:	**Metodika výpočtu všech podgrafů síťového grafu s uplatněním omezujících podmínek**
Zadávající katedra:	**Katedra informačních technologií**

Zásady pro vypracování:

- Primárním cílem bakalářské práce je realizace softwarové podpory pro výpočet všech podgrafů daného bázového síťového grafu s omezující podmínkou výskytu vrcholu-zdroje a vrcholu-ústí v každém z podgrafů.
- Zmíněné podgrafy jsou graficky reprezentovány na podkladu bázového síťového grafu.
- Reprezentace grafu je postavena nad vhodnou abstraktní datovou strukturou umožňující efektivní implementaci výše zmíněného typu výpočtu.
- Pro testování cílové aplikace se použijí vybrané síťové grafy reprezentující specifické Petriho sítě (v rámci softwarového nástroje CPN Tools) datově popsané pomocí XML-souboru.

Rozsah grafických prací:

Rozsah pracovní zprávy:

Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam odborné literatury:

1. CENEK, P. , KLIMA, V., JANÁČEK, J. Optimalizace dopravních a spojových procesů, Žilina, Univerzita Žilina, 1994.

2. VOLEK, J. Operační výzkum I., skripta DFJP UPa, Pardubice 2002

3. CORMEN a kol.: Introduction to algorithms, MIT Press, Cambridge, 2001

4. KAVIČKA, A., ŽARNAY, M.: Application of coloured Petri net for agent control and communication in the ABAsim architecture. In Proceeding of 9th workshop and tutorial on practical use of coloured Petri nets and the CPN Tools. K. Jensen (Ed.). Aarhus: University of Aarhus (Denmark), 2008. pp. 47-62. ISSN 0105-8571.

Vedoucí bakalářské práce: **doc. Ing. Antonín Kavička, Ph.D.**
Katedra softwarových technologií

Datum zadání bakalářské práce: **17. prosince 2010**

Termín odevzdání bakalářské práce: **13. května 2011**

Univerzita Pardubice Česká republika

L.S.

prof. Ing. Simeon Karamazov, Dr.
děkan

Ing. Lukáš Čegan, Ph.D.
vedoucí katedry

V Pardubicích dne 31. března 2011

Prohlášení autora

Prohlašuji, že jsem tuto práci vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Souhlasím s prezenčním zpřístupněním své práce v Univerzitní knihovně.

V Pardubicích dne 11. 5. 2011 Štěpán Karták

Poděkování

Tímto bych chtěl poděkovat vedoucímu práce prof. Ing. Antonínu Kavičkovi, Ph.D. za všechny rady, připomínky, návrhy a čas, který mi věnoval.

Anotace

Práce se zabývá navržením algoritmu pro výpočet všech souvislých podgrafů daného bázového síťového grafu s omezující podmínkou výskytu *vrcholu-zdroje* a *vrcholu-ústí* v každém z podgrafů a následnou softwarovou realizací s vizuálním znázorněním podgrafů na zadaném grafu.

Klíčová slova

graf, podgraf, Petriho síť, cesty v grafu, datové struktury

Title

Methodic of calculating the subgraphs of a network graph with the application of restrictive conditions

Annotation

This work is dedicated to designing an algorithm for calculating all subgraphs of the network graph with restrictive condition. Each of subgraphs must contain *edge-source* and *edge-sink*. This part is followed by the creation of software for implementation this algorithm with graphic demonstration subgraphs on working graph.

Keywords

graph, subgraph, Petri net, paths in graph, data structures

Obsah

Seznam obrázků

Seznam tabulek

1 Cíl práce

Cílem práce je navrhnout algoritmus řešící problém hledání množiny souvislých podgrafů na zadaném síťovém grafu se specifickou omezující podmínkou – nutno vždy zahrnout *vrchol-zdroj* a *vrchol-ústí*.

Výstupem práce je implementace navrženého algoritmu v programu schopném zpracovat XML datový popis grafu z programu CPN Tools a následně generovat textový výstup s popisem všech nalezených podgrafů na zadaném bázovém grafu a vizuálně znázornit jednotlivé podgrafy na bázovém grafu. Software tak doplní nástroj CPN Tools, který tento problém neřeší.

2 Základní pojmy

V této části práce jsou uvedeny a vysvětleny vybrané pojmy převážně z oblastí datový struktur a teorie grafů, které se vyskytují v této práci. Vysvětlení těchto pojmů vychází z použité literatury [1, 2, 3 a 6].

2.1 Pojmy z teorie grafů

Kartézský součin

Kartézský součin je množinová operace.

Kartézský součin dvou množin $X = \{x_1, x_2\}$ a $Y = \{y_1, y_2\}$, je množina, zapsaná jako $X \times Y$, která obsahuje všechny uspořádané dvojice prvků množin X *a* Y – na první pozici bude prvek z množiny X a na druhé prvek z množiny Y. Výsledkem bude množina $M = \{[x_1, y_1], [x_1, y_2], [x_2, y_1], [x_2, y_2]\}$. Definice kartézského součinu dvou prvků:

$$X \times Y = \{[x, y]: x \in X \ \wedge y \ \in Y\}$$

Kartézský součin lze definovat pro n množin o m prvcích v každé množině. Kartézský součin n množin je množina všech uspořádaných n-tic, přičemž každá z n-tic obsahuje vždy jeden prvek z původních množin. Obecný kartézský součin definujeme:

$$X_1 \times X_2 \times \ldots \times X_n = \{[x_1, x_2, \ldots, x_m]: x_i \ \in \ X_i, 1 \leq i \leq n\}$$

Počet prvků (n-tic) P ve výsledné množině kartézského součinu odpovídá součinu mohutností množin $X_n$vstupujících do kartézského součinu.

$$P = |X_1| \ \cdot \ |X_2| \cdot \ldots \cdot |X_n|$$

Graf, vrchol, hrana

Graf je základním objektem teorie grafů. Máme dvě libovolné disjunktní množiny V, X a zobrazení $p: X \rightarrow V \times V$[1].

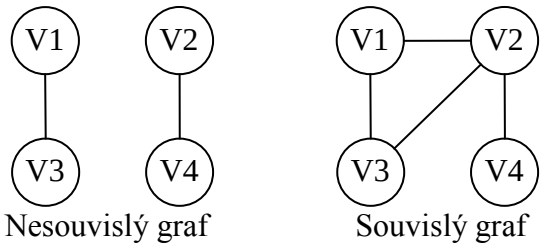
Neorientovaný **graf** je množina $G = (V, X, p)$. Prvky množiny *V* nazýváme **vrcholy** grafu *G*, prvky množiny X nazýváme **hranami** grafu G a **zobrazení** $p: V \rightarrow V \times X$ incidencí grafu *G*.

Příklad grafického znázornění grafu na obrázku 1.

[1] $V \times V$ je množina všech dvojic $(u, v); u, v \ \in V$.

Souvislý graf

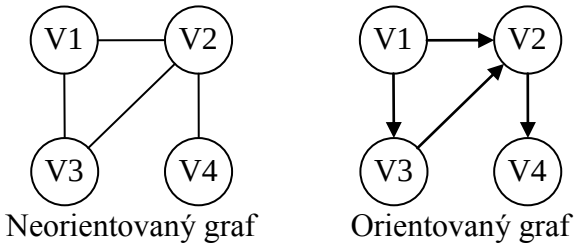
Souvislý graf je takový graf, ve kterém mezi každými dvěma vrcholy existuje alespoň jedna cesta[2].



Obrázek 1 – Nesouvislý a souvislý graf

Orientovaný graf

Orientovaným grafem $D = (V, Y, p)$ nazýváme uspořádanou dvojici množin V, Y a zobrazení $p: Y \rightarrow V \times Y$. Množina V jsou vrcholy, množina Y je tvořena uspořádanými dvojicemi $[u, v]$, kde $u \neq v$, $u \in V$, $v \in V$ a každá dvojice se vyskytuje v množině Y nejvýše jednou.

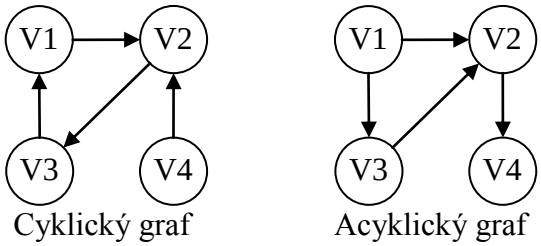


Obrázek 2 – Neorientovaný a orientovaný graf

Acyklický graf

Acyklickým grafem je nazýván takový orientovaný[3] graf, ve kterém nelze nalézt uzavřenou posloupnost propojených vrcholů (cyklus). (V opačném případě hovoříme o cyklickém grafu.)

[2] Cesta je posloupnost vrcholů a hran mezi dvěma vrcholy, ve které se neopakují žádné vrcholy a hrany.
[3] Na neorientovaném grafu nemá smysl hovořit o cyklickém či acyklickém grafu.

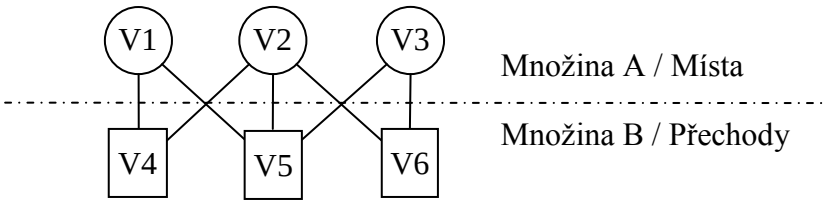


Obrázek 3 – Cyklický a acyklický graf

Bipartitní graf

Bipartitní graf je takový graf, jehož množina vrcholů obsahuje dvě disjunktní podmnožiny a vrcholy z jedné množiny jsou spojeny vždy pouze s vrcholy z druhé množiny a naopak.

V této práci budeme nazývat jednu množinu vrcholů místa a druhou přechody[4].



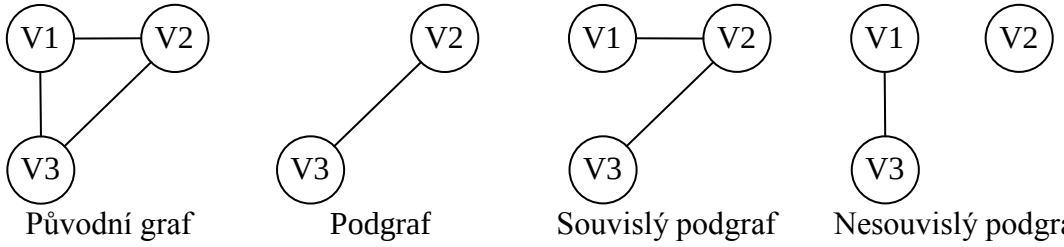
Obrázek 4 – Bipartitní graf

Podgraf

Podgrafem grafu $G = (V, X, p)$ rozumíme graf $\bar{G} = (\bar{V}, \bar{X}, \bar{p})$, pro který platí:

$$\bar{V} \subseteq V, \qquad \bar{X} \subseteq X$$

a pro každou hranu $h \in \bar{X}$ platí $\bar{p}(h) = p(h)$ (tj. $\bar{G} \subseteq G$) a dále musí být splněno, že každá hrana podgrafu inciduje se stejnou dvojicí vrcholů jako v původním grafu.



Obrázek 5 – Příklad podgrafů

[4] Jedná se o pojmy z Petriho sítí.

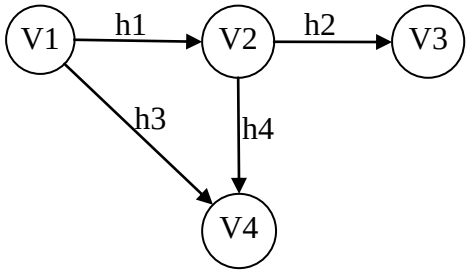
V této práci se budeme zabývat pouze souvislými orientovanými podgrafy.

Matice sousednosti

Matice sousednosti je jedním ze způsobů specifikace grafů. V této práci se budeme zabývat pouze maticí sousednosti na orientovaných grafech.

Matice sousednosti orientovaného grafu $G = (V, Y, p)$ je celočíselná matice $V = (v_{ij})_{i,j=1}^{m}$, kde $m = |V|$. Prvky matice v_{ij} nabývají hodnot:

- v_{ij} = 0 … pro i = j, nebo když $\nexists\, h \in Y: p(h) = [v_i, v_j]$,
- v_{ij} = 1 … když $\exists\, h \in Y: p(h) = [v_i, v_j]$.



Obrázek 6 – Příklad orientovaného grafu

Příklad matice sousednosti (podle grafu na obrázku 6):

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

Matice sousednosti budeme pro vyšší přehlednost zobrazovat v tabulce (viz tabulka 1).

Tabulka 1 - Matice sousednosti uvedená v tabulce (odpovídá grafu na obrázku 6)

	V1	**V2**	**V3**	**V4**
V1	0	1	0	1
V2	0	0	1	1
V3	0	0	0	0
V4	0	0	0	0

Mezi vlastnosti matice sousednosti patří:

- v řádcích čteme následníky vrcholu, tedy výstupní vrcholové okolí vrcholu (množina $\Gamma^{+}(v_i)$),

- ve sloupcích čteme předchůdce vrcholu, tedy vstupní vrcholové okolí vrcholu (množina $\Gamma^{-}(v_i)$).

Matice incidence

Matice incidence je jedním ze způsobů reprezentace grafů. V této práci se budeme zabývat pouze maticí incidence na orientovaných grafech.

Matice incidence orientovaného grafu $G = (V, Y, p)$ je celočíselná matice $A = (a_{ik})_{i,k=1}^{m,n}$, kde $m = |V|$ a $n = |Y|$. Prvky a_{ik} matice mohou nabývat hodnot:

- a_{ik} = +1, jestliže $p(h_k) = [v_i, v_j]$ pro nějaké $v_j \in V$,
- a_{ik} = −1, jestliže $p(h_k) = [v_j, v_i]$ pro nějaké $v_j \in V$,
- a_{ik} = 0 v ostatních případech $v_i \in p(h_k)$.

Příklad matice incidence (podle grafu na obrázku 6):

$$V = \begin{pmatrix} +1 & 0 & +1 & 0 \\ -1 & +1 & 0 & +1 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & -1 \end{pmatrix}$$

Matice incidence budeme pro vyšší přehlednost zobrazovat v tabulce (viz tabulka 2).

Tabulka 2 - Matice incidence uvedená v tabulce (odpovídá grafu na obrázku 6)

	h1	**h2**	**h3**	**h4**
V1	+1	0	+1	0
V2	–1	+1	0	+1
V3	0	−1	0	0
V4	0	0	−1	−1

Vlastnosti matice incidence jsou:

- počet hodnot +1 v řádcích matice vyjadřuje mohutnost množiny následníků $\Gamma^{+}(v_i)$,
- počet hodnot −1 v řádcích matice vyjadřuje mohutnost množiny předchůdců $\Gamma^{-}(v_i)$,
- ze sloupců lze vyčíst začátek a konec orientovaných hran.

2.2 Pojmy z datových struktur

Strom

Abstraktní datový typ Strom, odráží hierarchicky uspořádanou množinu prvků.

Výběr několika specifických prvků takovéto množiny:

- **kořen** je prvek, který nemá žádné předky a je vždy[5] jeden,
- **listy** jsou prvky, které nemají potomky.

V této práci se budeme zabývat pouze typem stromu **uspořádaný kořenový k-cestný strom**. Pro takovýto strom platí, že synové každého prvku představují lineárně uspořádanou množinu – tzn. že každý se synů daného prvku je jednoznačně označen jako první, druhý, ... n-tý.

Tabulka 3 – ADT Uspořádaný kořenový strom (k-cestný)

ADT Uspořádaný kořenový strom (k-cestný)	
Typy operací s ADT:	
A. Třída prvků	*Procedury pro práci s prvky stromu*
B. Les – třída konečných kořenových stromů	*Procedury pro práci s celými stromy*
Jednotlivé operace:	
A. Vytvoř	
Zruš	
JePrázdná(↑boolean)	
Mohutnost(↑PočetPrvků)	
Prohlídka(↓TypProhlídky,↓Akce)	*Do šířky, do hloubky, ...*
VložKořen(↓Prvek)	*Pouze do prázdné struktury*
VložList(↓Otec, ↓NaPořadí, ↓Prvek)	
JeListem(↓Prvek, ↑Boolean)	
OdeberList(↓Otec, ↓ZPořadí, ↑Prvek)	
ZpřístupniKořen(↑Prvek)	
ZpřístupniOtce(↓Syn, ↑Prvek)	
ZpřístupniSyna(↓Otec, ↓ZPořadí, ↑Prvek)	
ZpřístupniBratra(↓Koho, ↓ZPořadí, ↑Prvek)	
B. Vlož (↓KořenovýStrom, ↓Les)	
Odeber(↓KořenovýStrom, ↓Les)	

Graf

Abstraktní datový typ Graf, odráží binární relaci v množině a je to heterogenní (nehomogenní) bipartitní struktura pracující se dvěma odlišnými třídami prvku – vrcholy a hranami.

[5] Pokud není strom prázdný.

Tabulka 4 – Super ADT Graf

Super ADT Graf	
Typy operací s ADT:	
A. Třída prvků	*Procedury pro práci s prvky grafu*
B. Třída konečných grafů	*Procedury pro práci s jednotlivými grafy*
Jednotlivé operace:	
A. Vytvoř	
Zruš	
JePrázdný (↑Boolean)	
Mohutnost (↑PočetPrvků)	
Prohlídka (↓Typ, ↓Počátek, ↓Akce)	*Vrcholová/hranová, do hloubky/šířky*
VložVrchol (↓Vrchol)	
VložHranu (↓Hrana)	
OdeberVrchol (↓Klíc, ↑Vrchol)	
OdeberHranu (↓Klíc, ↑Hrana)	
NajdiVrchol (↓Klíc, ↑Vrchol)	
NajdiHranu (↓Klíc, ↑Hrana)	
ZpřístupniNásledníky (↓Koho, ↑Prvky)	
ZpřístupniPředchůdce (↓Koho, ↑Prvky)	
ZpřístupniIncidenčníPrvky (↓Koho, ↑Prvky)	
DefinujBránu (↓Prvek)	
AnulujBránu (↓Prvek)	
ZpřístupniBrány (↑Prvky)	
B. Sjednocení (↓GrafA, ↓GrafB, ↑GrafC)	

3 Problém k řešení

3.1 Popis hledaného algoritmu

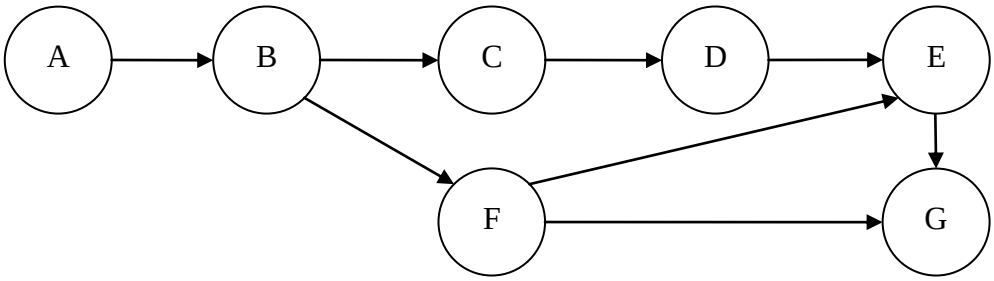
Naším úkolem je hledání množiny souvislých podgrafů na zadaném bázovém síťovém grafu. Tento problém není obecně[6] popsán, a proto je nutné navrhnout postup, jakým lze dosáhnout požadovaného výsledku.

Níže popsaná řešení jsou platná a otestovaná na síťovém bázovém grafu, splňujícím tyto podmínky:

- orientované hrany,
- souvislý graf,
- acyklický graf,
- obsahuje jeden *vrchol-zdroj* a jeden *vrchol-ústí*.

Při vytváření algoritmů bylo čerpáno z použité literatury [1, 2, 3, 5 a 6].

Všechny algoritmy uvažují za podgraf i bázový graf.



Obrázek 7 – Graf pro demonstraci algoritmů

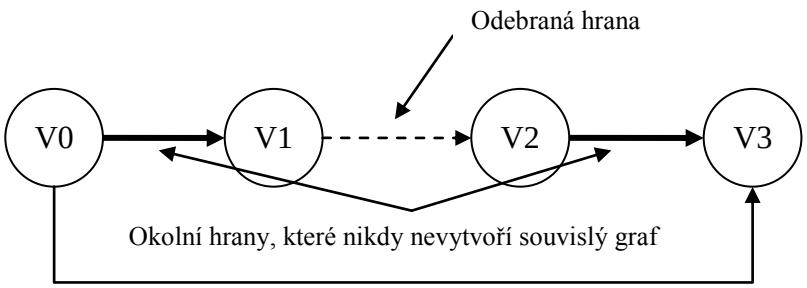
3.2 Supervrchol, superhrana a supergraf

Před přístupem k jednotlivým algoritmům zavedeme pojmy supervrchol, superhrana a supergraf.

Tyto pojmy zavádíme, protože nemá smysl pracovat se všemi sekvencemi hran, ale pouze s těmi, které nelze „rozdělit“ (viz obrázek 8 – tučné hrany nikdy nevytvoří souvislý graf). Tím je myšlena skutečnost, že pokud odebereme hranu spojující dva vrcholy, přičemž oba mají po jednom následovníku a jednom předchůdci[7] (tzn. ve vrcholu se nevytváří žádné větvení ani se do vrcholu nesbíhají hrany), nemá smysl zkoušet další

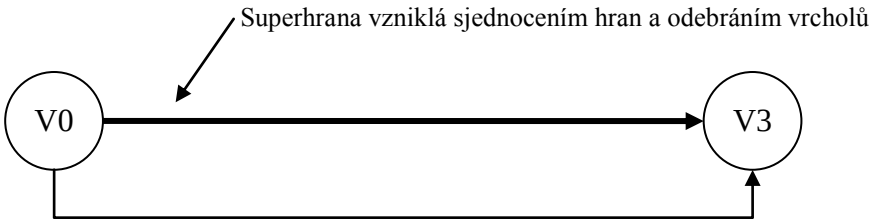
[6] Nebyly nalezeny žádné informace o konkrétním algoritmu
[7] Netýká se *vrcholu-zdroje* a *vrcholu-ústí*.

kombinace „okolních“ hran (viz obrázek 8), protože tyto hrany nikdy nevytvoří platný (souvislý) podgraf.



Obrázek 8 – Ukázka podmnožiny hran nemající smysl

Naopak, takovéto hrany spolu s vrcholy, které je spojují, lze sjednotit do jedné hrany, kterou nazveme **superhrana**. Sjednocujeme všechny hrany a vrcholy, které vytvářejí spojnici vrcholů, které mají více jak jednoho předchůdce, nebo více jak jednoho následovníka (viz obrázek 9 a jeho porovnání s obrázkem 8).



Obrázek 9 – Graf se zavedenou superhranou (odpovídá původnímu grafu z obrázku 8)

Vrcholy, které zůstanou po sjednocování hran v superhrany, nazveme **supervrcholy**.

Graf složený ze supervrcholů a superhran nazveme **supergraf**.

Algoritmus vytvoření supergrafu z bázového grafu je popsán v kapitole 3.5.1 – – krok 2 a algoritmus na převedení supergrafu na původní bázový graf v také v kapitole 3.5.1 – krok 5.

Všechny níže popsané algoritmy lze aplikovat na supergraf vzniklý z bázového grafu, na kterém hledáme podgrafy. Výsledné podgrafy se po rekonstrukci podle bázového grafu nebudou lišit, a zkrátíme tak průběh algoritmů.

3.3 Strom cest a jejich kombinace

Toto řešení (dále označované jako řešení 1) staví na myšlence vybudovat strom všech cest, které vzniknou při průchodu bázovým síťovým grafem a výsledné podgrafy získat kombinováním cest od kořene k jednotlivým listům.

3.3.1 Algoritmus

1. krok

Nejprve musíme vybudovat strom cest, se kterým budeme dále pracovat. Budeme pracovat s uspořádaným k-cestným kořenovým stromem.

V grafu nalezneme *vrchol-ústí* a tento vložíme do datové struktury typu zásobník[8]. Tento vrchol vložíme do stromu jak kořen.

2. krok

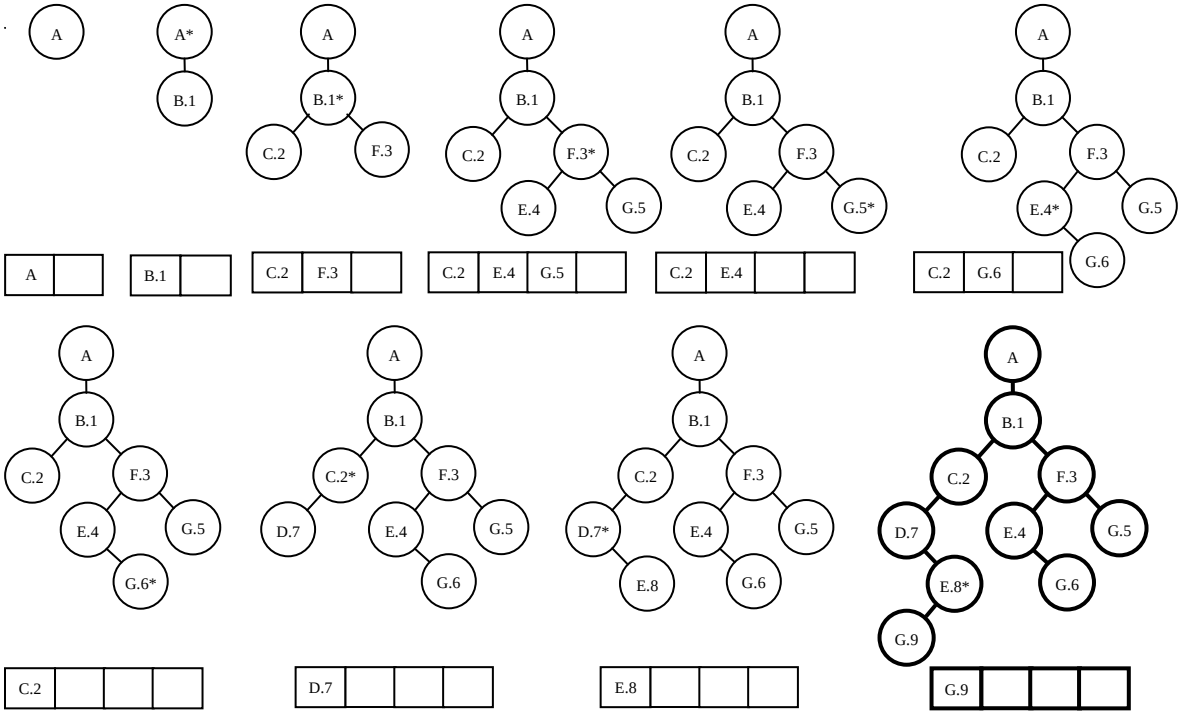
Ze zásobníku odebereme prvek/vrchol, tento vrchol nalezneme ve stromu a tento nalezený prvek stromu označíme za aktuální.

Vezmeme množinu následníků aktuálního vrcholu ($\Gamma^{+}(v_{aktualni})$) a jednotlivé prvky (tedy vrcholy) množiny následníků vložíme do stromu jako potomky aktuálního prvku a tyto nově vzniklé potomky také přidáme do zásobníku.

Zde je nutné zohlednit fakt, že jednotlivé vrcholy se velmi často budou opakovat, a proto je nutné zavést další označení vrcholu (např. vrcholy očíslovat podle pořadí zpracování následníků při vkládá do stromu).

Tento krok opakujeme, dokud v zásobníku jsou nějaké prvky/vrcholy. V opačném případě pokračujeme krokem 3.

[8] Je možné použít i datovou struktury typu frontu, na výsledek to nebude mít vliv.



Obrázek 10 – Vývoj budování stromu cest (řešení 1); poslední krok není zobrazen, na výsledku se ale nic nezmění, protože vrchol G (G.8) nemá žádné následníky → potomky

3. krok

Máme vybudovaný strom cest. Na stromu nalezneme množinu listů (množina L).

V našem příkladu (obrázky 7 a 10):

$$L = \{\text{G.9; G.6; G. 5}\}, \qquad |L| = 3$$

4. krok

Množina všech podgrafů bude odpovídat množině průchodů stromem ke všem možným kombinacím listů.

Počet P podgrafů, které takto dostaneme, můžeme spočítat podle vzorce:

$$P = K(1,n) + K(2,n) + \ldots K(n-1,n) + K(n,n), \text{ kde } n = |L|$$

Tato posloupnost kombinačních čísel odpovídá počtu n-člených variací s opakováním dvou prvků:

$$P = 2^{|L|} - 1$$

Pro ukázkový graf (obrázek 7) to je:

$$P = K(1,3) + K(2,3) + K(3,3) = 3 + 3 + 1 = 7$$

$$P = 2^3 - 1 = 7$$

Kroky 5, 6 a 7 opakujeme *P* krát.

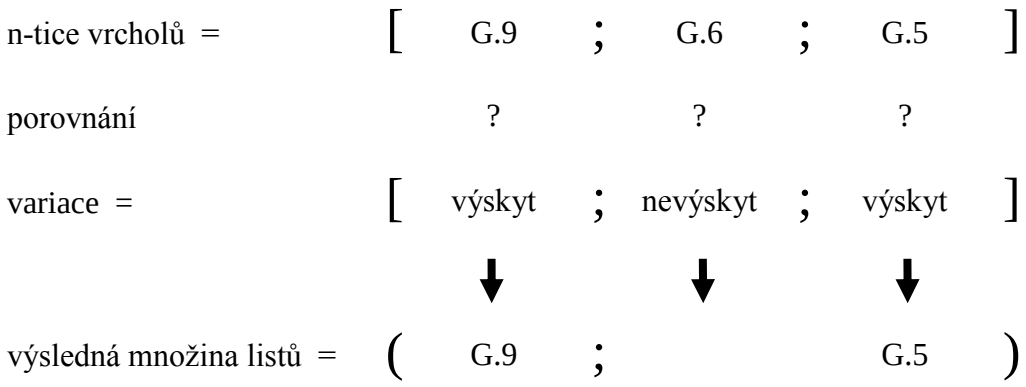
5. krok

Vytvoříme prázdnou množinu *C*.

Vzhledem ke spojitosti s variacemi (uvedeno v kroku 4) využijeme ke generování kombinací listů n-členné variace s opakováním dvou prvků („výskyt“ a „nevýskyt“), kde n = $|L|$.

Vytvoříme uspořádanou n-tici (n = $|L|$) prvků z množiny L.

Vytvoříme novou[9] (doposud originální) variaci s opakováním n-členů 2 prvků („výskyt“ a „nevýskyt“) a procházíme prvky variace, a pokud je nalezen prvek „výskyt“, pak přidáme do množiny C prvek z n-tice na stejné pozici. Pro lepší pochopení viz obrázek 11.



Obrázek 11 – Vysvětlení řešení 1 / krok 4

Získaly jsme množinu listů *C*. Platí, že:

$$C \subseteq L$$

Příklad:

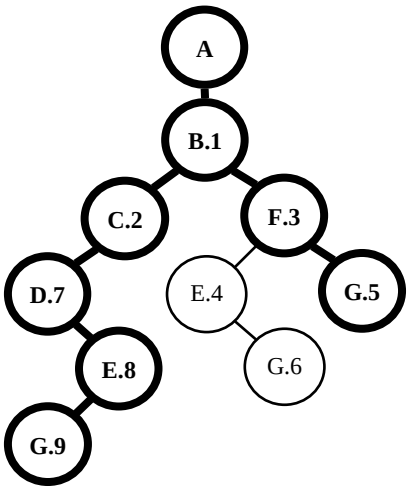
$$V_i = [\text{výskyt,nevýskyt,výskyt}]$$

$$C = (G.9, G.5)$$

[9] Algoritmus pro generování variací s vysvětlením – viz kapitola 4.2. Variace obsahující pouze prvky „nevýskyt“ nemá smysl, a takovou neuvažujeme.

6. krok

Projdeme strom cest z kroku 2 (obrázek 10) od listů v množině *C* po kořen a jednotlivé prvky, přes které projdeme, si označíme.



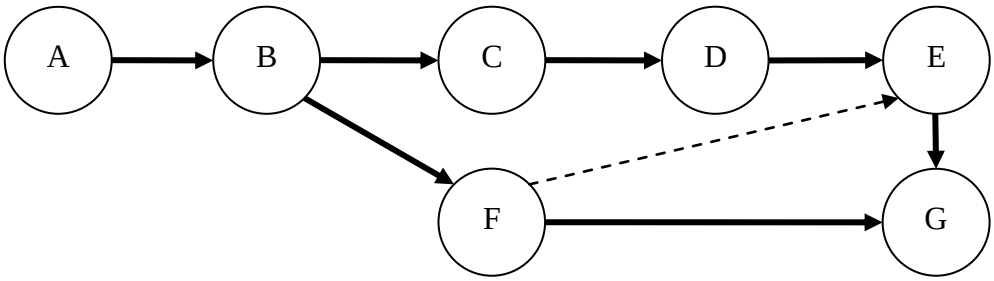
Obrázek 12 – Ukázka řešení 1 / krok 6

7. krok

Ve stromu cest máme nyní znázorněn podgraf. Posledním krokem je převést označené prvky/vrcholy (ale také to můžeme považovat za „podstrom“) na graf (podgraf původního grafu) označený *R*.

Procházíme strom od kořene (do hloubky či do šířky) a jednotlivé prvky/vrcholy V_i vkládáme do nového grafu *R* (pokud $V_i \notin R$), vkládáme i hrany (opět pokud $h_i \notin R$), a to podle závislosti otec – potomek ve stromu s orientací od otce k potomkovi.

Výsledkem je podgraf *R* (příklad na obrázku 13).



Obrázek 13 – Ukázkový podgraf na bázovém grafu

3.3.2 Výsledky

Tento algoritmus po úvodním experimentálním testování na jednoduchém grafu vytvářel očekávaný výsledek – počet reálných podgrafů odpovídal výsledkům algoritmu.

Po přistoupení k testování algoritmu na složitějších grafech (graf C – viz příloha C) bylo zjištěno, že algoritmus nevrací vždy unikátní podgrafy, tzn. že podgrafy se opakují.

V důsledku toho ani není možné přímo zjistit počet podgrafů podle uvedeného algoritmu (krok 1 až 4).

Po zjištění tohoto problému bylo zahájeno experimentální hledání problému se snahou upravit algoritmus.

Bylo zjištěno, že problém nastává, pokud se při rekonstrukci podgrafu (krok 7) opakuje „průchod" jednou hranou (ve výsledném podgrafu nevznikají vícenásobné hrany, kvůli aplikaci podmínky „pokud $h_i \notin S$"). Tento stav není problém odfiltrovat z množiny podgrafů v kroku 7 (pokud by se vyskytl stav při vkládání hrany h_i takový, že $h_i \in S$, potom takový podgraf nebude zahrnut do množiny řešení).

Dalším experimentálním testováním bylo zjištěno, že ani testování vícenásobných hran není řešením problému, v malém množství případů jsou některá řešení unikátní i pokud je některá hrana použita vícenásobně.

Detailní ukázku duplicit naleznete v příloze D.

Tento stav (problém s vícenásobným „zápisem" hran) se nepodařilo algoritmicky ošetřit, a proto je doporučeno při používání tohoto algoritmu testovat výskyt nového podgrafu proti množině již nalezených podgrafů a podle toho zařadit nový podgraf do množiny výsledných podgrafů nebo ho nezařazovat.

3.4 Vybudování úplného stromu řešeních

Tento algoritmus (dále označovaný jako řešení 2) počítá se záznamem průchodu jednotlivými vrcholy (k záznamu je použita datová struktura typu strom – uspořádaný kořenový k-cestný strom), pokud je možné z vrcholu sledovat více hran jak jednu, dojde k větvení průchodu na všechny možnosti, které se v dané pozici nacházejí.

3.4.1 Algoritmus

Algoritmus si vysvětlíme na grafu znázorněném na obrázku 7.

Budeme pracovat s jedním uspořádaným kořenovým k-cestným stromem. Každý prvek stromu bude množinou hran o předem neznámé mohutnosti.

1. krok

Nalezneme *vrchol-zdroj* v zadaném bázovém grafu a množinu jeho následníků $\Gamma^+(\mathrm{v_{zdroj}})$.

Jako kořenový prvek stromu vložíme množinu obsahující hrany $[v_{zdroj}, \gamma_i^+]$, $\gamma_i^+ \in \Gamma^+$.

2. krok

Ve stromu nalezneme množinu listů (tzn. množinu množin hran) a označíme ji množina *L*.

3. krok

Vezmeme prvek l_i množiny *L*, a označíme ho jako množina *N*, tedy platí: $l_i \in L$, $l_i = N$.

Znázornění kroků 3 až 6 viz tabulky 5 a 6 pro graf na obrázku 7.

4. krok

Tento krok provedeme pro všechny prvky n_j množiny *N*:

Vezmeme prvek n_j, tj. prvek, který představuje hranu. Nalezneme následníky koncového vrcholu hrany n_j (tedy $N_j' = \Gamma^+(\mathrm{v}_{\text{koncový n}_\mathrm{j}})$).

Pokud pro každé n_j platí, že $N_j' = \emptyset$, pokračujeme krokem 7.

Pokud $N_j' \neq \emptyset$, pak vytvoříme všechny možné kombinace prvků množiny N_j' a tyto kombinace (množiny vrcholů) uložíme do množiny N_j''. (Jedno z možných řešení je popsáno v kapitole 3.2.1 – kroky 4 a 5.)

5. krok

Provedeme kartézský součin všech prvků množin N'', čímž vznikne množina n-tic množin vrcholů, kterou nazveme *S*.

6. krok

Připravíme si množinu *H*.

Vezmeme n-tici s_k a označíme ji jako *U* (*U* obsahuje množiny), a pro všechny prvky ($u_q, u \in U$) množiny *U* provedeme:

Zavedeme $M = u_q$. Pro všechny prvky $m_r, m \in M$ vezmeme koncový vrchol hrany $n_q, n \in N$ – nazveme ho vrchol *K* a vytvoříme novou hranu $h_x' = [K, m_r]$ a tuto hranu zařadíme do množiny H_k', tedy $h_x' \in H_k'$.

Po aplikování postupu na všechny prvky v množině *U* vložíme H_k' do množiny H.

Tento krok provedeme pro každý prvek množiny *S* a potom prvky množiny *H* vložíme do stromu jako potomky množiny *N* ($N = l_i$).

7. krok

Krok 3 opakujeme pro všechny prvky množiny L. Poté přejdeme na krok 8.

8. krok

Krok 2 opakujeme, pokud v množině L existuje alespoň jedno l_i ($l_i = N$), kde $N_j' \neq \emptyset$.

9. krok

Nyní již máme výsledek. Počet listů stromu odpovídá počtu nalezených podgrafů, jednotlivé grafy můžeme zrekonstruovat čtením stromu do hloubky – jedna cesta od kořene k listu reprezentuje určitý průchod bázovým grafem a tedy jeden z hledaných podgrafů.

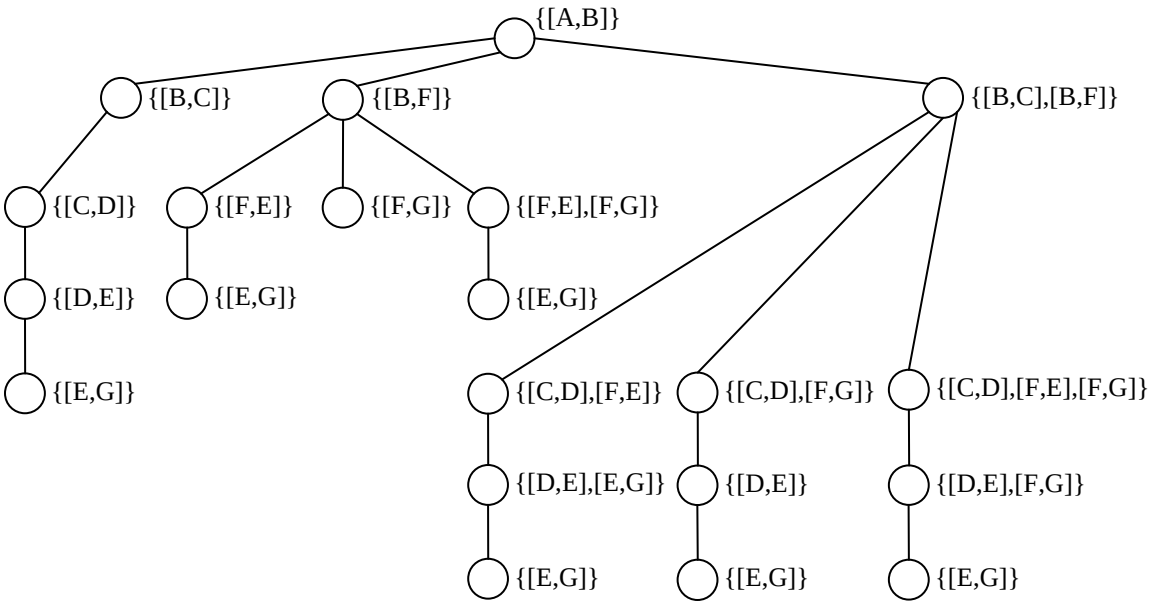
Výsledek algoritmu je vidět na obrázku 14.

Tabulka 5 – Ukázka řešení 2 / krok 3 až 6 přímo po kroku 1

$\boldsymbol{j}$	1
$\boldsymbol{n_j}$	$[A, B]$
$\boldsymbol{N_j'}$	$\{C, F\}$
$\boldsymbol{N_j''}$	$\begin{Bmatrix} \{C\}, \\ \{F\}, \\ \{C, F\} \end{Bmatrix}$
$\boldsymbol{S}$	$\begin{Bmatrix} [\{C\}], \\ [\{F\}], \\ [\{C, F\}] \end{Bmatrix}$
$\boldsymbol{H}$	$\begin{Bmatrix} \{[B, C]\}, \\ \{[B, F]\}, \\ \{[B, C], [B, F]\} \end{Bmatrix}$

Tabulka 6 – Ukázka řešení 2 / krok 3 až 6 pro prvek {[B,C],[B,F]} stromu cest

j	1	2
n_j	$[B,C]$	$[B,F]$
N'_j	$\{D\}$	$\{E,G\}$
N''_j	$\left\{\{D\}\right\}$	$\left\{\begin{matrix}\{E\},\\ \{G\},\\ \{E,G\}\end{matrix}\right\}$
S	$\left\{\begin{matrix}[\{D\},\{E\}],\\ [\{D\},\{G\}],\\ [\{D\},\{E,G\}]\end{matrix}\right\}$	
H	$\left\{\begin{matrix}\{[C,D],[F,E]\},\\ \{[C,D],[F,G]\},\\ \{[C,D],[F,E],[F,G]\}\end{matrix}\right\}$	



Obrázek 14 – Ukázka výsledku řešení 2

3.4.2 Výsledek

Tento algoritmus vykazuje stejný problém jako řešení 1, tzn. že generuje ne vždy unikátní řešení. Po testování a porovnání výsledků s výsledky řešení 1 bylo zjištěno, že se zde vyskytuje úplně stejný problém, a tím je vícenásobný „průchod“ hranou, případně hranami. Tato chyba je podrobněji popsána v kapitole 3.3.2. Stejně tak jako v řešení 1 se problém nepodařilo algoritmicky ošetřit a opět platí, že výsledky je nutné testovat navzájem a ponechat pouze unikátní podgrafy.

Duplicity ve výsledcích ale nejsou hlavním problémem toho algoritmu.

Hlavní problémem je nutnost budování stromu všech podgrafů, tj. všech řešení. To znamená u rozsáhlých grafů značné rozrůstání („košatění“) vytvářeného stromu, tomu se při softwarové implementaci řešení nelze vyhnout. Toto bude mít za následek rozsáhlé požadavky na operační paměť a s tím související snižování rychlosti práce nad datovou strukturou představující strom. Výsledkem může být strom o počtu listů v řádu sta milionů (viz kapitola 5 – graf C).

3.4.3 Zjednodušení

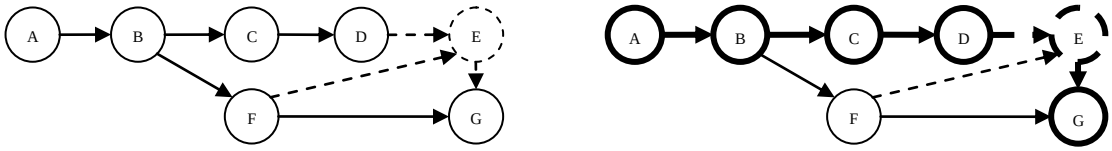
Tento algoritmus lze mírně zjednodušit. Nahradíme hrany ve formě uspořádané dvojice vrcholů za pojmenovanou hranu (tzn. že bude nutné vytvořit tabulku názvů hran s vrcholy, ke kterým sousedí), čímž zjednodušíme a zpřehledníme algoritmus, a podle toho jednotlivé kroky, samozřejmě, bude nutné mírně upravit.

Byl zmíněn problém s velikostí vytvořeného stromu. Tento problém lze z části řešit nahrazením vybraných hran bázového grafu superhranami (popis a vysvětlení pojmu naleznete v kapitole 3.2).

3.5 Kombinace hran

Tento algoritmus (dále označen jako řešení 3) je založen na faktu, že podgraf je oproti svému bázovému grafu tvořen pouze určitou podmnožinou vrcholů a hran bázového grafu. Vezmeme-li tento fakt v úvahu, zjistíme, že náš problém – nalezení podgrafů – je pouze vytvářením veškerých platných (tzn. splňujících omezující podmínky) podmnožin hran a vrcholů.

Algoritmus bude pracovat primárně s množinou hran, protože jeden vrchol může mít několik předchůdců a několik následníků (tzn. že jeden vrchol může být do grafu připojen více hranami). Není možné pracovat s množinou vrcholů a na ní hledat podmnožiny. Takovéto řešení by vyloučilo celou množinu hran, které vycházejí nebo vstupují do odebraného vrcholu, přestože takovéto hrany mohou vytvářet požadované podgrafy. Tento problém je znázorněn na obrázku 15. Tento stav nenastane, pokud budeme pracovat s množinou hran a vzniklé izolované vrcholy nebudeme uvažovat.



Odebrání vrcholu E a sousedních hran, které poté nemají smysl

Jeden z podgrafů bázového grafu, který nepůjde vytvořit bez odebraného vrcholu E

Obrázek 15 – Ukázka problémového vyloučení vrcholu pro vytvoření podgrafu z množiny vrcholů bázového grafu (obrázek 7)

3.5.1 Algoritmus

Algoritmus si uvedeme na grafu znázorněném na obrázku 7. Budeme přímo pracovat se supergrafem[10], protože pro tento algoritmus má redukce počtu hran na superhrany zásadní vliv na počet testovaných podgrafů.

1. krok

V prvním kroku vytvoříme matici sousednosti (tabulka 7) bázového grafu.

Tabulka 7 – Matice sousednosti pro řešení 3

	A	B	C	D	E	F	G
A	0	1	0	0	0	0	0
B	0	0	1	0	0	1	0
C	0	0	0	1	0	0	0
D	0	0	0	0	1	0	0
E	0	0	0	0	0	0	1
F	0	0	0	0	1	0	1
G	0	0	0	0	0	0	0

2. krok

V druhém kroku vytvoříme superhrany, k tomu využijeme matici sousednosti (tabulka 7).

Z tabulky vybereme ty řádky a sloupce vrcholů, které u svého vrcholu mají pouze jednu jedničku. Pro příklad uveďme, že nevybereme vrchol A, protože ve svém sloupci nemá žádnou jedničku, nevybereme ani vrchol B, protože má v řádku dvě jedničky, ale vybereme vrchol C, protože má v řádku i sloupci po jedné jedničce.

Tabulka 8 – Matice sousednosti pro řešení 3 / krok 2

	A	B	C	D	E	F	G
A	0	1	0	0	0	0	0
B	0	0	1	0	0	1	0
C	0	0	0	1	0	0	0
D	0	0	0	0	1	0	0
E	0	0	0	0	0	0	1
F	0	0	0	0	1	0	1
G	0	0	0	0	0	0	0

[10] Zavedení superhran není bezpodmínečně nutné, ale značně zvyšuje efektivitu algoritmu. Pokud nebudeme používat superhrany, krok 2 a 5 vynecháme.

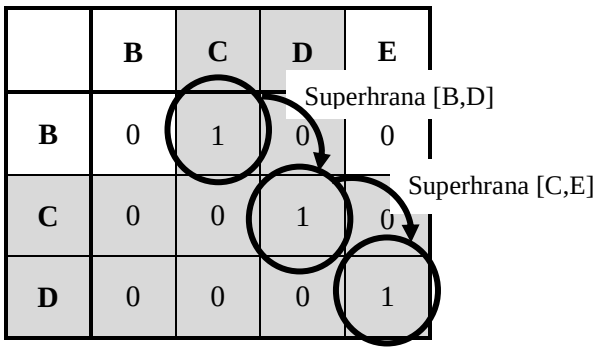
V tabulce 8 je prozatímní výsledek. Takto jsme vybrali vrcholy, které můžeme zanedbat. Nyní je ještě nutné vytvořit superhrany. Vybrané vrcholy leží na budoucí superhraně.

Pro zjednodušení a ukázku vytvoříme superhrany zvlášť pro vrchol C – superhrana bude odpovídat spojení [B,C] a [C,D] a pro vrchol D – superhrana bude odpovídat spojení [C,D] a [D,E].

Pro vrchol C vezmeme jedničku v příslušném sloupci a poznačíme si, ve kterém se nachází řádku – v řádku B – toto bude začátek nové superhrany. Nyní vezmeme řádek vrcholu C a vybereme sloupeček, ve kterém se nachází jednička – sloupec D – tento vrchol bude koncem nové superhrany. Výsledkem je sjednocení v superhranu [B,D].

Obdobně aplikujeme uvedený postup na vrchol D. Výsledkem je sjednocení v superhranu [C,E].

Detail grafického znázornění výsledku na obrázku 16.



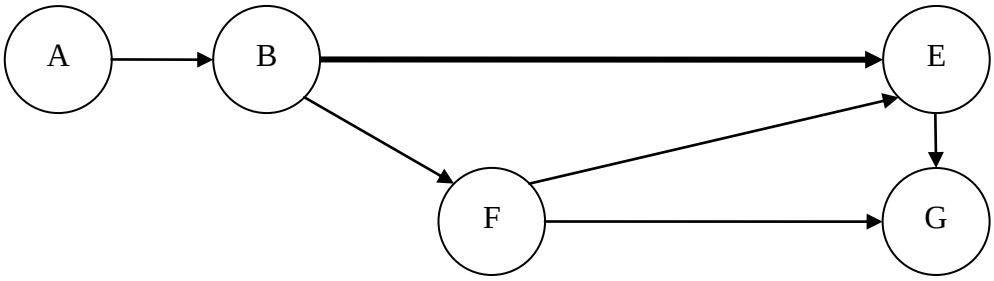
Obrázek 16 – Detail grafického znázornění superhran v matici sousednosti

Z obrázku 16 je patrné, že dvě vzniklé superhrany mají společnou buňku v tabulce sousednosti. To znamená, že se superhrany překrývají na hraně [C – řádek, D – sloupec], a my můžeme tuto hranu sjednotit. Výsledek bude tento:

$$[B, D] \cup [C, E] = [B, E]$$

Výsledkem je superhrana z vrcholu B do vrcholu E v původním grafu. Vrcholy B a E jsou nyní supervrcholy.

Pro ukázku viz obrázek 17 – byla vytvořena jedna superhrana, ostatní hrany splňují stejná kritéria, a proto je můžeme dále nazývat superhranami.



Obrázek 17 – Ukázkový supergraf

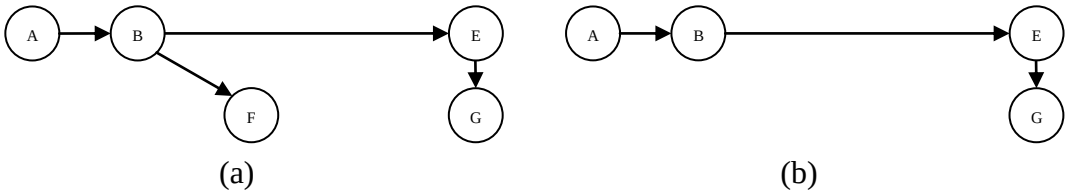
Obecně můžeme tento krok popsat tak, že nejprve vytvoříme pomocné superhrany pro vrcholy, které nemají žádné větvení cest (v řádku i sloupečku matice sousednosti mají po jedné jedničce) a to tak, že pro každý takovýto vrchol vybereme ze sloupce řádek s jedničkou (ve vrcholu tohoto řádku bude pomocná superhrana začínat), a z řádku vybereme sloupeček s jedničkou, jenž určuje ve kterém vrcholu bude pomocná superhrana končit. Následně provedeme sjednocení takto vzniklých pomocných superhran. Výsledkem bude jedna či více výsledných superhran, které použijeme pro další řešení.

3. krok

V tomto kroku vytvoříme novou množinu superhran tak, že vybereme určitou kombinaci superhran.

K tomu problému můžeme přistupovat stejně jako v řešení 1, a to využít variace s uspořádanou množinou superhran – viz kapitola 3.3.1 (kroky 4 a 5). Dva příklady výsledku toho kroku jsou znázorněny na obrázku 18.

Supervrcholy, které zůstanou izolovány v podgrafu neuvažujeme.



Obrázek 18 – Nově vzniklé podgrafy na supergrafu po aplikaci určité variace

4. krok

Ve čtvrtém kroku otestujeme, zda podgraf, který vznikne po aplikování vybrané variace, je platný, tzn. zda obsahuje jeden *vrchol-zdroj* shodný s původním bázovým grafem a jeden *vrchol-ústí* shodný s bázovým.

K řešení použijeme matici incidence bázového grafu se znázorněnými superhranami (tabulka 8) a pro praktickou ukázku podgrafy z obrázku 18.

Rozsah matice incidence musí odpovídat rozsahu bázového grafu superhran. Není možné pracovat s maticí incidence vzniklého podgrafu! Tuto matici vyplníme podle podgrafu a otestujeme platnost podgrafu podle pravidel:

- Pro řádky (vrcholy), které nejsou *vrchol-zdroj* nebo *vrchol-ústí* podle bázového grafu:
  - existuje-li v řádku matice hodnota 1, pak v řádku musí být minimálně jedna hodnota –1,
  - existuje-li v řádku matice hodnota –1, pak v řádku musí být minimálně jedna hodnota 1,
  - existují-li v řádku pouze nuly, pak je řádek platný.
- Pro řádek s *vrchol-zdroj* platí, že musí obsahovat minimálně jednu hodnotu 1.
- Pro řádek s *vrchol-ústím* platí, že musí obsahovat minimálně jednu hodnotu –1.

Platný podgraf je takový podgraf, pro který platí všechna pravidla. Praktická ukázka v tabulce 9 (podle obrázku 18a – neplatný podgraf) a v tabulce 10 (podle obrázku 18b – platný podgraf).

Pokud podgraf není platný, toto řešení neuvažujeme a postoupíme ke kroku 6.

Tabulka 9 – Matice incidence pro podgraf podle obrázku 18a

	[A,B]	[B,E]	[B,F]	[F,E]	[F,G]	[E,G]	Stav
A	1	0	0	0	0	0	**OK**
B	–1	1	1	0	0	0	**OK**
E	0	–1	0	0	0	1	**OK**
F	0	0	–1	0	0	0	**Chyba**
G	0	0	0	0	0	–1	**OK**

Tabulka 10 – Matice incidence pro podgraf podle obrázku 18b

	[A,B]	[B,E]	[B,F]	[F,E]	[F,G]	[E,G]	Stav
A	1	0	0	0	0	0	**OK**
B	–1	1	0	0	0	0	**OK**
E	0	–1	0	0	0	1	**OK**
F	0	0	0	0	0	0	**OK**
G	0	0	0	0	0	–1	**OK**

5. krok

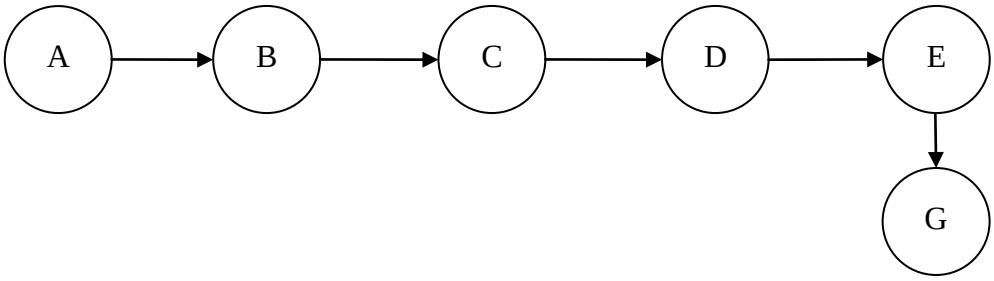
Pokud je podgraf platný, je nutné zrekonstruovat superhrany do původního stavu hran. K tomu využijeme matici sousednosti (tabulka 8) z kroku 2.

Superhrany jsme vytvořili buďto sjednocením hran (a zanedbáním vrcholů mezi nimy) nebo se jimi staly elementární hrany, které splňovaly požadavky na superhranu.

Bereme jednotlivé superhrany, a porovnáváme je s maticí sousednosti. Pokud existuje záznam, odpovídající superhraně, je vše v pořádku a máme hranu odpovídající bázovému grafu. Pokud neexistuje záznam v tabulce, musíme procházet tabulku po vrcholech od *vrcholu-začátku* superhrany až po *vrchol-konec* superhrany a tento „průchod" si zaznamenat, a poté nahradit superhranu těmito zaznamenanými daty (nalezenými vrcholy a hranami). Takto získáme jeden podgraf – jedno řešení.

Uveďme příklady:

- Superhrana [A,B] existuje v tabulce sousednosti (tabulka 8), můžeme ji považovat za hranu odpovídající bázovému grafu.
- Superhrana [B,E] neexistuje v matici sousednosti (v řádku vrcholu B není záznam o sloupci vrcholu E). Proto musíme projít matici od vrcholu B až po nalezení vrcholu E a cestu si zaznamenat: hrana z B do C, vrchol C, hrana z C do D, vrchol D, hrana z D do E, vrchol E (*vrchol-konec* superhrany), tzn. že jsme nalezli všechny hrany a vrcholy, které jsme sjednotili pod superhranu [B,E].
- Příklad platného podgrafu, který jsme brali za příklad při vysvětlování tohoto algoritmu, je uveden na obrázku 18b.



Obrázek 19 – Jeden z hledaných podgrafů na ukázkovém grafu u řešení 3

6. krok

Všechny podgrafy nalezneme opakováním kroků 3 až 5 do vyčerpání všech kombinací superhran.

3.5.2 Výsledky

Tento algoritmus poskytuje unikátní a neopakující se podgrafy. Počet výsledných podgrafů oproti počtu zkoušených kombinací (krok 4) je závislý na povaze grafu a nebyl nalezen způsob, jak tento počet přímo zjistit. U rozsáhlých grafů použití superhran značně zvyšuje efektivitu, i zde ovšem závisí na povaze grafu, jak značná redukce hran na superhrany bude a tedy jak vysokého snížení testovaných možností dosáhneme.

3.5.3 Možnosti implementace

Řešení 3 je možné implementovat, po vhodné úpravě, jako vícevláknovou úlohu, vzhledem komu, že zde odpadá nutnost kontrolovat podgrafy navzájem (jako tomu je u řešení 1, které by vyžadovalo synchronizaci vláken) je možné při použité vhodného generátoru variací rozdělit interval testovaných možností na několik (v závislosti na počtu vláken) a každý interval nechat zpracovat v jednom vláknu. Kritickou sekcí bude společné úložiště výsledných podgrafů. Je na zváženou, zda výsledky neukládat zvlášť a nesloučit je až na závěr, ale vzhledem k poměru platných a neplatných podgrafů (viz kapitola 5.2 a tabulka 15) to nebude nutné.

3.5.4 Optimalizace

Algoritmus lze ještě mírně optimalizovat dvěma způsoby:

1. Petriho sítě vždy začínají jednou hranou. Tato hrana musí být ve všech platných podgrafech, a proto můžeme tuto hranu (superhranu) zařadit do výsledného podgrafu automaticky a nezařazovat ji do množiny superhran, na které se použije variace. Vzhledem ke snížení o jeden člen variace 2 prvků docílíme snížení počtu testovaných možností na polovinu:

$$n \dots počet\ superhran$$

$$n_{opt} = n - 1$$

$$P_{opt} = K(1, n_{opt}) + K(2, n_{opt}) + \dots K(n_{opt} - 1, n_{opt}) + K(n_{opt}, n_{opt})$$

2. Nemá smysl zkoušet množinu superhran, která bude mít menší mohutnost než je počet superhran v nejkratší cestě. To znamená, že nalezneme nejkratší cestu (algoritmus řešící tento problém – viz použitá literatura [2]) a poté testujeme kombinace s minimálním počtem prvků rovném nebo vyšším počtu superhran v nejkratší cestě:

$$p \dots počet\ superhran\ v\ nejkratší\ cestě$$

$$P = K(p, n) + K(p + 1, n) + \dots K(n - 1, n) + K(n, n)$$

Pokud použijeme obě optimalizace, dostaneme tento počet testovaných možností:

$$P_{opt} = K(p, n_{opt}) + K(p+1, n_{opt}) + \ldots K(n_{opt}-1, n_{opt}) + K(n_{opt}, n_{opt})$$

4 Vybrané implementace

V této kapitole jsou popsány důležité a pro aplikaci navržených řešení stěžejní implementace.

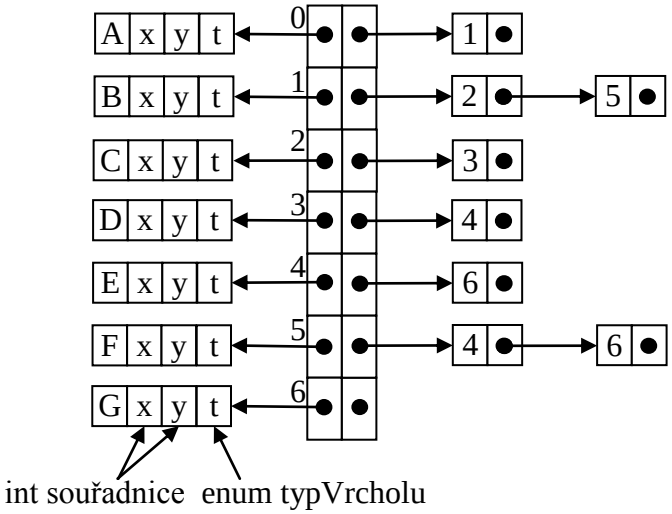
4.1 Použitá datová struktura pro graf

Byla použita datová struktura typu dopředná hvězda pole – tabulka(seznam následníků). Viz obrázek 20 (podle grafu na obrázku 7). Datová struktura je vytvořena podle informací v literatuře [1].

Je použita reprezentaci vrcholově-statická a hranově-dynamická. Tato struktura umožňuje rychlý přístup k jednotlivým vrcholům, udržování základních informací o vrcholu nutných k zobrazení celého grafu a také dynamický počet následníků vrcholu.

Složitost hlavních operací, kterou jsou využívány:

- NajdiVrchol – O(1),
- ZpřístupniNásledníky – O(s), kde $s = \max_{i=1,..,n} s_i$.



Obrázek 20 – Grafické znázornění použité datové struktury reprezentující graf

4.2 Generátor variací

Řešení 1 a řešení 3 vyžadují generování variací s opakováním, konkrétně se jedná o n-členné variace s opakováním 2 prvků („výskyt“ a „nevýskyt“). Navržený algoritmus řeší generování vybrané variace dvou prvků (odpovídajících datovému typu boolean) typu pravda (true) a nepravda (false) pro n-členů. Generátor tedy nelze použít obecně pro generování jakékoliv variace s opakováním.

Při vytváření algoritmu byla za vzor použita pravdivostní tabulka (tabulka 11), která se používá pro řešení výroků.

Tato část čerpá z použité literatury [5].

Tabulka 11 – Pravděpodobností tabulka řešení 3-členné variace s opakováním 2 prvků

Pořadí / Sloupce	1.	2.	3.
1.	0	0	0
2.	0	0	1
3.	0	1	0
4.	0	1	1
5.	1	0	0
6.	1	0	1
7.	1	1	0
8.	1	1	1

Na tabulce 11 vidíme, že splňuje naše požadavky na variace (v našem případě 3-členné variace 2 prvků). Na tabulce je také jasně patrný princip umísťování jedniček (pravda) a nul (nepravda).

Algoritmus vyplnění tabulky je tento:

1. Spočítáme počet (N) variaci pro n-členů:

$$N = V'(n, 2) = 2^n$$

2. Vytvoříme tabulku o rozměrech N řádků a n sloupců.

3. Určíme nové N:

$$N = \frac{N}{2}$$

4. Vyplníme od leva první nevyplněný sloupec podle postupu:

    a. Od prvního nevyplněného řádku v aktuálním sloupečku vyplníme do N řádků nulu.

    b. Do následujících N řádků vyplníme jedničku.

    c. Postup a až c opakujeme dokud nezaplníme všechny řádky.

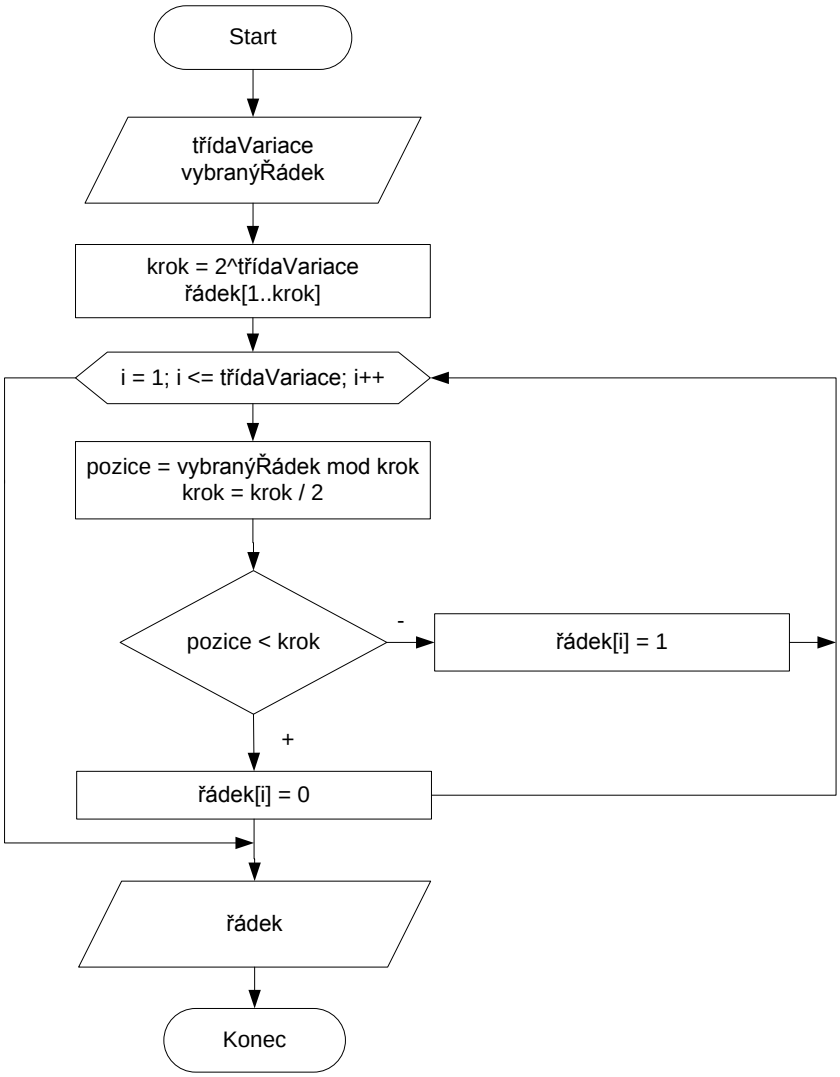
5. Opakujeme krok 3 až 5, dokud nezaplníme celou tabulku.

V takto vyplněné tabulce máme připravené veškeré variace, které mohou nastat. Problémem je rozsah tabulky u variací s vysokým počtem členů. Pro příklad uveďme, že pro třídu 30ti členů získáme tabulku o velikosti 1 073 741 824 řádků (odpovídá počtu

variací) a 30 sloupců, což je více jak 32 miliard (konkrétně 32 212 254 720) buněk, a to je z hlediska využití paměti zcela nepřijatelné. Z tohoto důvodu je nutné generovat jednotlivé variace (řádky tabulky) zvlášť.

I pro tento nově vzniklý problém použijeme pravděpodobností tabulku. Využijme toho, že jednotlivé sady jedniček a nul se periodicky opakují a to vždy v dvojnásobném počtu oproti předchozímu – viz tabulka 11 – v prvním sloupečku je série jedniček a nul jednou, v druhém 2x, ve třetím 4x).

Algoritmus lze nejlépe popsat pomocí vývojového diagramu (obrázek 21 – na vstupu je vybraný řádek a třída variace). Princip spočívá ve zjištění, ve které části periody nul a jedniček se vybraný řádek v jednotlivých sloupcích nachází. Implementace v jazyce Java je uvedena v příloze B.



Obrázek 21 – Diagram algoritmu získání vybrané variace s opakováním

4.3 Generátor kartézských součinů

Řešení 2 při každém postupu grafem musí vytvořit novou množinu listů (cest) – – kartézské součiny z libovolného počtu množin s libovolným počtem prvků v každé množině.

Tento problém by bylo možné řešit pomocí vnořených cyklů, pokud by byl zadán pevný počet množin. Pevný počet množin není zadán, a proto je nutné hledat řešení jinde.

Předem neznámý počet cyklů lze řešit pomocí rekurze, tento přístup byl použit i pro implementaci kartézské součinu.

Konkrétní použitý algoritmus viz příloha B. Rekurze spočívá v použití uspořádaného seznamu množin prvků. Kombinuje se každý prvek z množiny n_i s každým prvkem množiny n_{i+1} (pokud existuje) a toto se rekurzivně provádí pro každou množinu n_{i+1}.

5 Testování

V této části jsou popsány výsledky experimentálního testování algoritmů po jejich implementaci v jazyce Java.

5.1 Testované bázové grafy

Jednotlivé grafy jsou zobrazeny v příloze C.

Informace o grafech naleznete v tabulce 12.

Tabulka 12 – Informace o testovaných grafech

Testované grafy	Graf A	Graf B	Graf C
Počet hran	20	36	67
Počet superhran na grafu	9	14	28
Počet míst	9	16	27
Počet přechodů	8	15	26
Celkový počet vrcholů (= počet míst + počet přechodů)	17	31	53
Počet podgrafů	**39**	**159**	**200 703**

5.2 Výsledky

Testování probíhalo na nevytíženém stroji AMD Phenom™ II X4 955 (4 x 3,2 GHz), 8 GB RAM, Windows 7 Ultimate 64 bit, Java build 1.6.0_24-b07. Testovány byly algoritmy bez zápisu výsledků na disk.

Řešení 1

Toto řešení je u rozsáhlých grafů velmi pomalé, to je způsobeno porovnáváním výsledků při snaze zjistit, zda je nový podgraf unikátní. Na tento fakt navazuje další problém, kterým je nutnost vytvořit každý podgraf (alokace paměti – obecně pomalá operace), který je ve většině případů shledán jako duplicitní, a tedy nemá smysl. Vytvoření podgrafů ze stromu je navíc pomalejší než čtení matice, jako tomu je v řešení 3. Podrobné výsledky viz tabulka 13.

Tabulka 13 – Výsledky testování řešení 1

Testované grafy	Graf A	Graf B	Graf C
Počet testovaných podgrafů	63	255	1 073 741 823
Počet nalezených unikátních podgrafů	39	159	200 703
Počet nalezených duplicitních podgrafů	24	96	1 073 541 120
Počet unikátních podgrafů z celkového počtu testovaných podgrafů	64%	61%	0,018%
Doba výpočtu	**< 1 s**	**< 1 s**	**210 hodin***

*Odhad podle začátku běhu algoritmu

Řešení 2

Dobu potřebnou pro vytvoření stromu podgrafů u rozsáhlého grafu (graf C – viz příloha C) se nepodařilo zjistit. Testováním bylo zjištěno, že algoritmus se značně zpomalí po vytvoření přibližně 200 000 listu (tedy ještě ani ne hotových řešeních) oproti předchozímu průběhu. Podrobnosti viz tabulka 14.

Tabulka 14 – Výsledky testování řešení 2

Testované grafy	Graf A	Graf B	Graf C
Počet nalezených podgrafů	63	255	nezjištěno*
Počet nalezených unikátních podgrafů	39	159	nezjištěno*
Počet nalezených duplicitních podgrafů	24	96	nezjištěno*
Počet unikátních podgrafů z celkového počtu testovaných podgrafů	64%	61%	nezjištěno*
Doba výpočtu	**< 1 s**	**< 1 s**	**nezjištěno***

* Výpočet nebyl dokončen vzhledem k přílišné časové náročnosti

Řešení 3

Toto řešení dosahuje i u rozsáhlých bázových grafů požadovaných výsledků v rozumném čase, na rozdíl o řešeních 1 a 2. Testována byla jednovláknová optimalizovaná verze[11] implementace algoritmu. Podrobnosti viz tabulka 15.

Tabulka 15 – Výsledky testování řešení 3

Testované grafy	Graf A	Graf B	Graf C
Počet testovaných podgrafů	220	8101	134 217 350
Počet nalezených podgrafů	39	159	200 703
Počet neplatných podgrafů	217	8033	134 017 025
Počet nalezených podgrafů z celkového počtu testovaných podgrafů	18%	2%	0,15%
Doba výpočtu	**< 1 s**	**< 1 s**	**6 min 2 s**

5.3 Výběr vhodného řešení

Všechna tři řešení poskytují požadované výsledky, řešení 1 a 2 jsou v porovnání s řešením 3 velice pomalé, proto za vhodný algoritmus pro hledání podgrafů na bázovém grafu vybereme řešení 3.

[11] Optimalizace popsaná v kapitole 3.5.4.

6 Základní charakteristika softwarové aplikace

K otestování uvedených algoritmů byla vytvořena aplikace napsaná v jazyce Java.

Program zobrazuje zvolený bázový graf (vstupem je XML datový popis grafu z programu CPN Tools[12]) a vybraný podgraf.

Podgrafy jsou generovány podle popsaných algoritmů v kapitole 3. Uživatel může zvolit mezi řešením 1 a řešením 3, který je nastaveno jako výchozí. Bylo upuštěno od zpřístupnění řešení 2 pro uživatele (ve zdrojovém kódu programu je řešení 2 zahrnuto) vzhledem k jeho značným nevýhodám – viz kapitoly 3.4.2 a 5.

Jednou z hlavních funkcionalit je možnost exportovat výsledky (od bázového grafu po jednotlivé podgrafy) do textového souboru ve dvou formátech – řádkový výpis vrcholů a jejich následníků a matice sousednosti. Také je možné uložit zobrazení grafu a podgrafu jako obrázek v několika formátech.

Před zpřístupněním zobrazení podgrafů – jejich výběr spočívá zadáním pořadového čísla, podgrafy jsou očíslovány podle pořadí zpracování navrženými algoritmy – je nutné provést export podgrafů do souboru, z tohoto souboru bude program číst data a zobrazovat je. Externí uložení podgrafů je voleno vzhledem k:

- vysokému množství podgrafů – vysoké nároky na operační paměť počítače,
- časové náročnosti na generování podgrafů u rozsáhlých grafů a s tím spojenou znovupoužitelnost výsledků,
- nemožnosti identifikovat jednotlivé podgrafy bez znalosti všech.

Aplikace je napsaná v jazyce Java.

[12] Více viz http://cpntools.org.

7 Závěr

Byly navrženy 3 různé algoritmy řešení hledání podgrafů podle zadání práce. Algoritmy byly popsány, implementovány na vhodných datových strukturách a otestovány na několika grafech typu Petriho síť.

Po provedení testů bylo shledáno, že ne všechny navržené algoritmy jsou vhodné pro rozsáhlé grafy. Za optimální řešení, poskytující výsledky v přijatelném čase, bylo zvoleno řešení Kombinace hran a jeho následná optimalizace.

Byla vytvořena aplikace, ve které jsou použity dva z uvažovaných algoritmů. Aplikace doplňuje software CPN Tools, tak že používá její výstupy k výpočtům podgrafů. Umožňuje zobrazit výsledné podgrafy na vybraném bázovém grafu a exportovat výsledky ve dvou formátech do textových souborů pro další zpracování.

Výsledkem a výstupem práce je popsaný optimalizovaný algoritmus hledání podgrafů a jeho softwarová implementace.

Literatura

[1] **KAVIČKA, Antonín**. Datové struktury. Elektronické sylaby přednášek předmětu Datové struktury. 2009.

[2] **VOLEK, Josef**. Operační výzkum I. Pardubice: Univerzita Pardubice, 2008. 112 s. ISBN 978-80-7395-073-6.

[3] **CENEK, Petr; KLIMA, Valenta; JANÁČEK, Jaroslav**. Optimalizace dopravních a spojových procesů. Žilina: Vysoká škola dopravy a spojov v Žilině, 1994. 343 s. ISBN 80-7100–197-X.

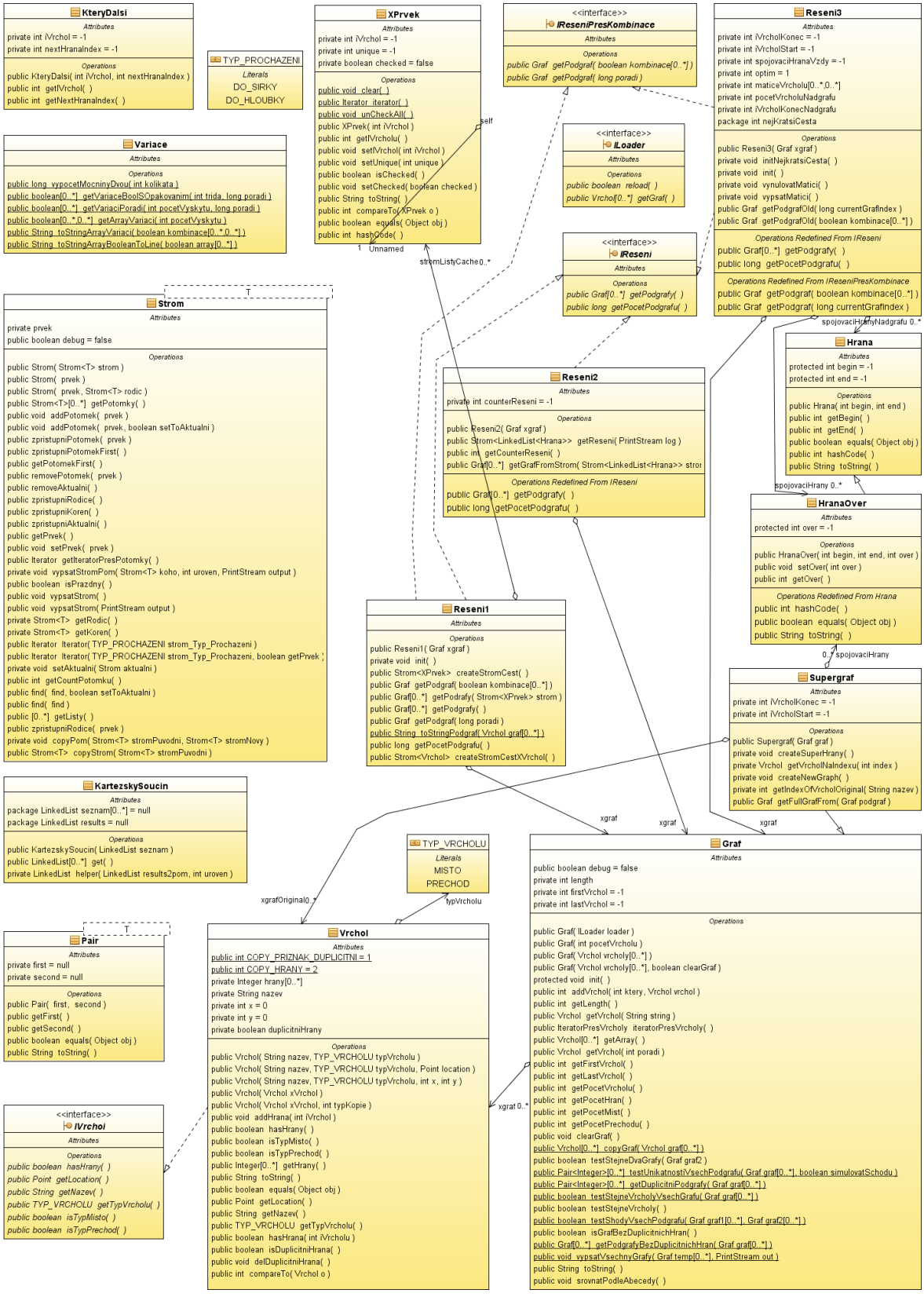
[4] **VESELÝ, Petr**. Počítačová grafika. Elektronické sylaby přednášek předmětu Počítačová grafika. 2008.

[5] **CALDA, Emil; DUPAČ, Václav**. Matematika pro gymnázia – Kombinatorika, pravděpodobnost a statistika. Praha: Prometheus, 2005. 172 s. ISBN 80-7196–147-7.

[6] **KRYNICKÝ, Martin.** Ucebnice.Krynicky.CZ [online]. 2009 [cit. 2011-05-02]. Kartézský součin. Dostupné z WWW: <http://ucebnice.krynicky.cz/Matematika/02_Funkce_a_rovnice/1_Linearni_funkce/2101_Kartezsky_soucin.pdf>.

[7] **Download.Oracle.com** [online]. 2011 [cit. 2011-05-01]. Java™ Platform, Standard Edition 6 API Specification. Dostupné z WWW: <http://download.oracle.com/javase/6/docs/api/overview-summary.html>.

Příloha A – UML diagram hlavních datových struktur programu



Obrázek 22 – UML diagram

Příloha B – Zdrojové kódy

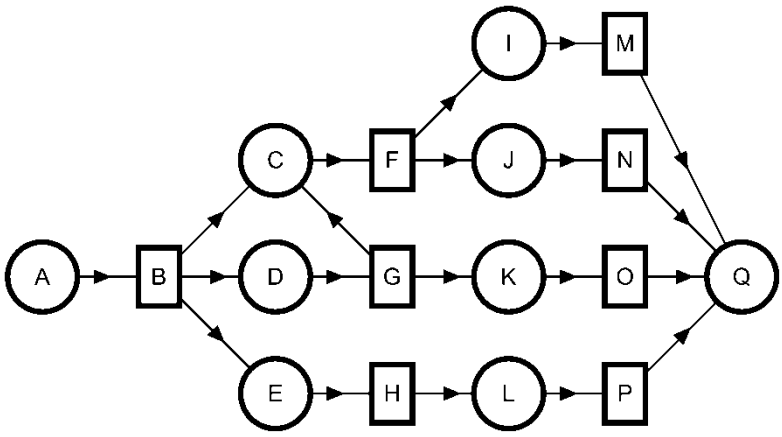
1. Generátor variací

```
// Implementace v jazyce Java
public static long vypocetMocninyDvou(int kolikata) {
	long res = 1;
	for (int i = 0; i < kolikata; i++) res *= 2;
	return res;
}


public static boolean[] getVariaceBoolSOpakovanim(int trida, long poradi)
{

	double pocetMoznosti = vypocetMocninyDvou(pocetVyskytu);
	double step = pocetMoznosti; // Pomocná, pro lepší orientaci
	boolean[] res = new boolean[pocetVyskytu];

	double stepMod;
	for (int i = 0; i < pocetVyskytu; i++) {
		stepMod = poradi % step;
		step = step/2;
		if ( stepMod < step ) {
			res[i] = false;
		}
		else {
			res[i] = true;
		}
	}
	return res;
}
```

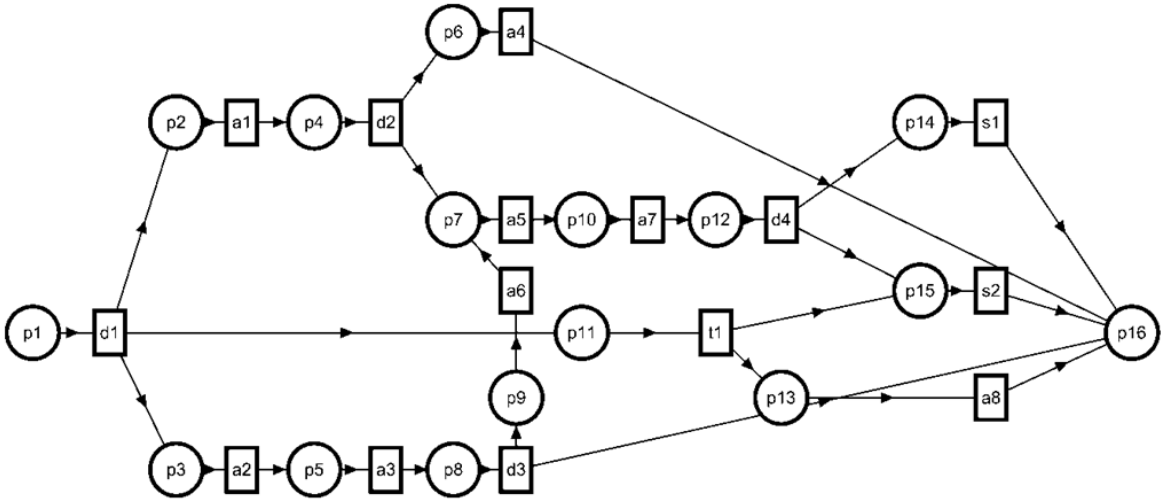
2. Realizace kartézského součinu

```
// Implementace v jazyce Java
import java.util.LinkedList;

/**
 * Algoritmus nelze použít, pokud je jedna ze vstupním množin prázdná
 */
public class KartezskySoucin {
	LinkedList<LinkedList> seznam = null;
	LinkedList results = null;

	/**
	 *
	 * @param seznam Spojový seznam, dvouúrovňový (množina množin),
kombinuje každého z jedné úrovně s každým z druhé úrovně, až do n úrovní
	 */
	public KartezskySoucin(LinkedList seznam) {
		this.seznam = seznam;
		this.results = new LinkedList();
	}

	/**
	 *
	 * @return Seznam všech variant
	 */
	public LinkedList<LinkedList> get() {
		LinkedList results2pom = new LinkedList();
		helper(results2pom, 0);
		return results;
	}

	/**
	 *
	 * @param results2pom
	 * @param uroven
	 * @return
	 */
	private LinkedList helper(LinkedList results2pom, int uroven) {
		if (uroven>=this.seznam.size()) {
			this.results.add(results2pom);
			return null;
		}

		for (int i = 0; i < this.seznam.get(uroven).size(); i++) {
			LinkedList x = new LinkedList();
			x.addAll(results2pom);
			x.add(this.seznam.get(uroven).get(i));
			helper(x, uroven+1);
		}
		return null;
	}
}
```

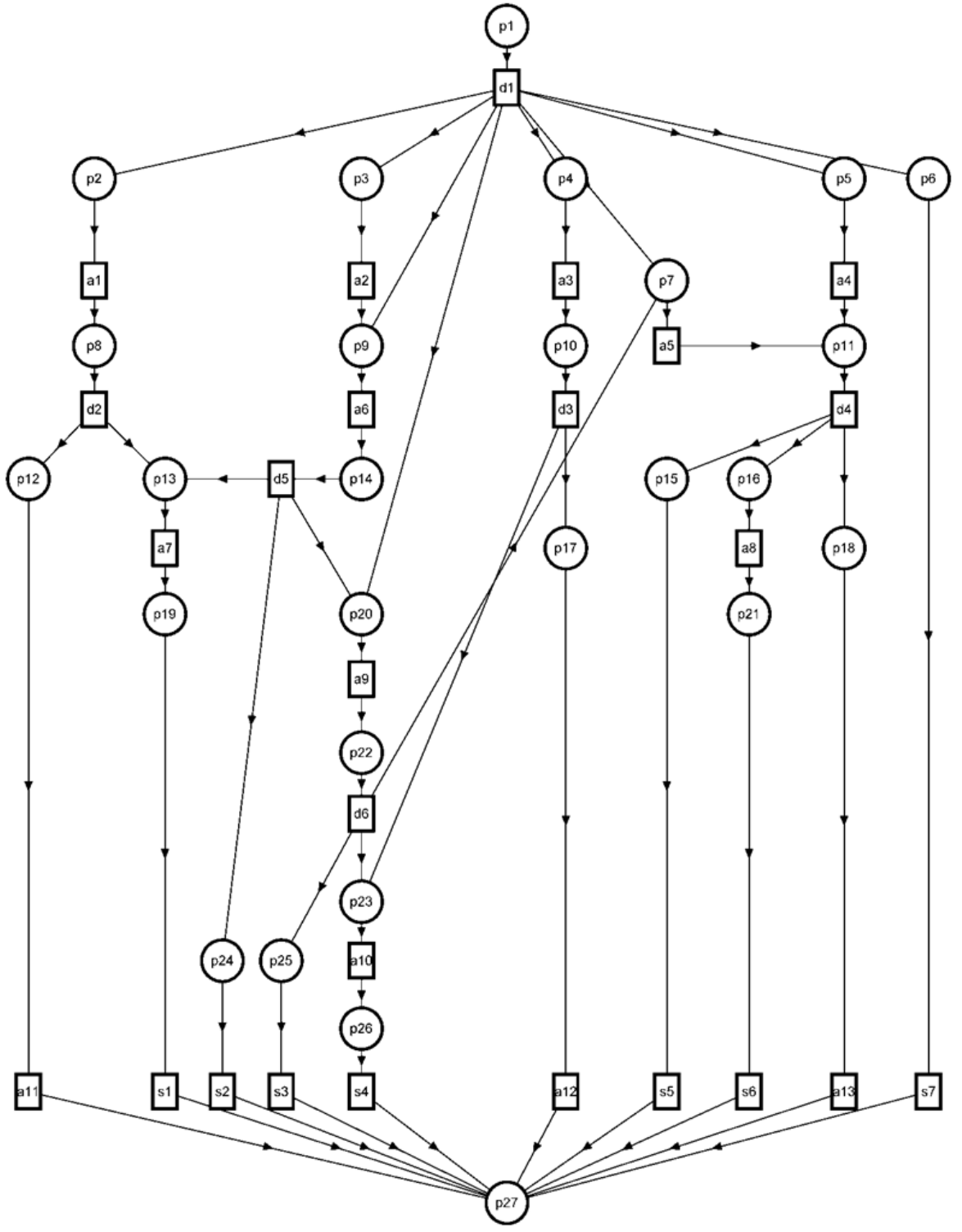
Příloha C – Testované grafy



Obrázek 23 – Graf A



Obrázek 24 – Graf B

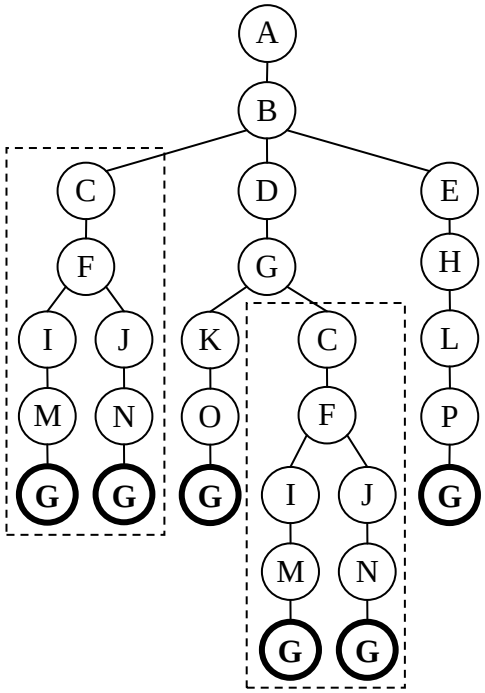


Obrázek 25 – Graf C

Příloha D – Ukázka duplicit podgrafů pro řešení Strom cest a jejich kombinace

Duplicity si ukážeme na grafu A. Pro graf A řešení 1 zkouší 63 podgrafů. Unikátních podgrafů je 39. To znamená, že se zde vyskytují duplicitní podgrafy. Strom cest toho grafu je znázorněn na obrázku 26. Na obrázku jsou vyznačeny opakující se průchody vrcholy – vznikají duplicitní hrany.

Počet duplicitních grafů (celkem 24) je rozdíl testovaných a unikátních podgrafů – – podrobnosti v tabulce 16, grafické znázornění stromů cest pro duplicitní grafy shodné s podgrafem 20 (viz tabulka 16) jsou znázorněny v obrázku 27 a zobrazení podgrafu 20 (a tedy i těch duplicitních) na obrázku 28. Ostatní duplicitní grafy jsou na obrázku 29.

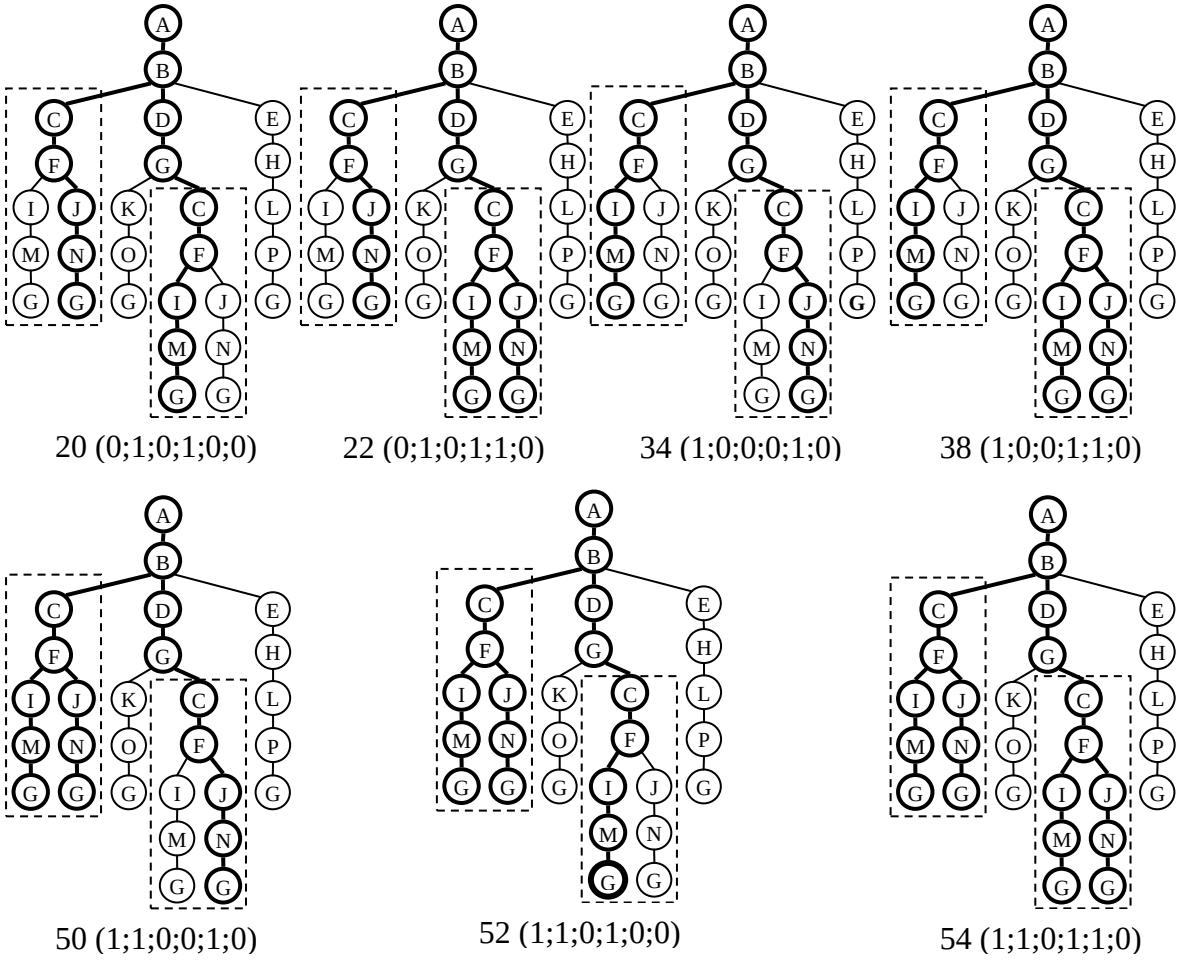


Obrázek 26 – Strom cest pro graf A s vyznačenými listy a opakující se skupinou

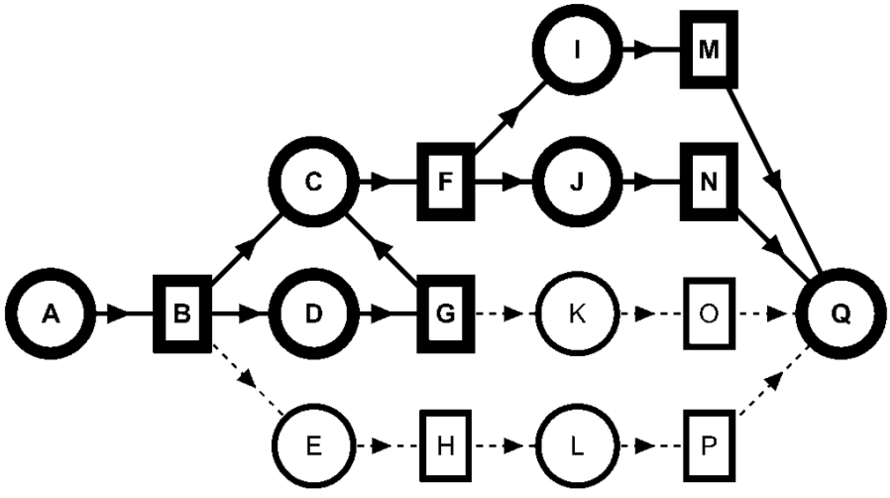
Tabulka 16 – Duplicitní podgrafy (pořadí podle průběhu algoritmu) na grafu A s variacemi, podle kterých vznikly

Vybraný podgraf	**Podgraf 20 (0;1;0;1;0;0)**	**Podgraf 21 (0;1;0;1;0;1)**	**Podgraf 28 (0;1;1;1;0;0)**	**Podgraf 29* (0;1;1;1;0;1)**
Duplicitní grafy	22 (0;1;0;1;1;0) 34 (1;0;0;0;1;0) 38 (1;0;0;1;1;0) 50 (1;1;0;0;1;0) 52 (1;1;0;1;0;0) 54 (1;1;0;1;1;0)	23 (0;1;0;1;1;1) 35 (1;0;0;0;1;1) 39 (1;0;0;1;1;1) 51 (1;1;0;0;1;1) 53 (1;1;0;1;0;1) 55 (1;1;0;1;1;1)	30 (0;1;1;1;1;0) 42 (1;0;1;0;1;0) 46 (1;0;1;1;1;0) 58 (1;1;1;0;1;0) 60 (1;1;1;1;0;0) 62 (1;1;1;1;1;0)	31 (0;1;1;1;1;1) 43 (1;0;1;0;1;1) 47 (1;0;1;1;1;1) 59 (1;1;1;0;1;1) 61 (1;1;1;1;0;1) 63 (1;1;1;1;1;1)
Ukázka na obrázku	27 a 28	29a	29b	29c

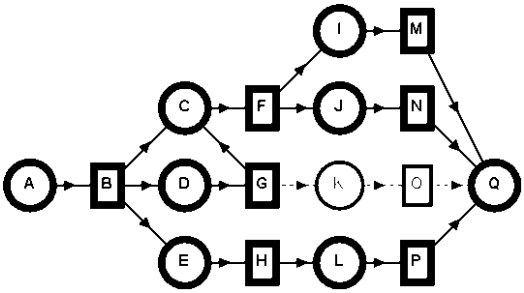
* Je shodný s bázovým podgrafem – viz duplicitní graf s číslem 63 (takovýchto grafů je náhodná, u jiných grafů se nemusí vyskytovat).



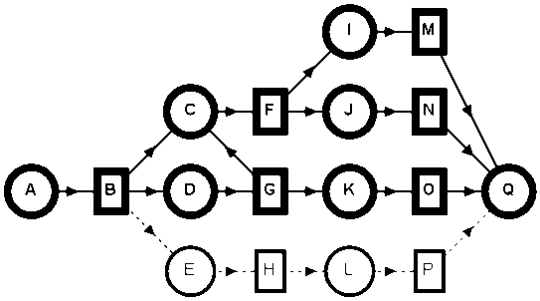
Obrázek 27 – Duplicitní podgrafy na stromu cest



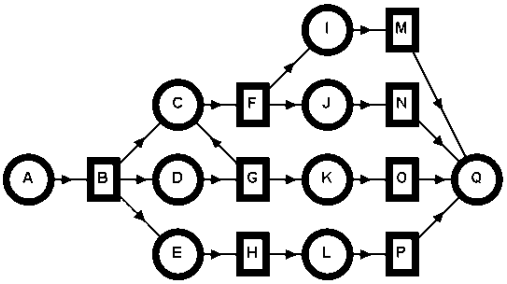
Obrázek 28 – Graf A se znázorněným duplicitním podgrafem 20



(a) podgraf 21



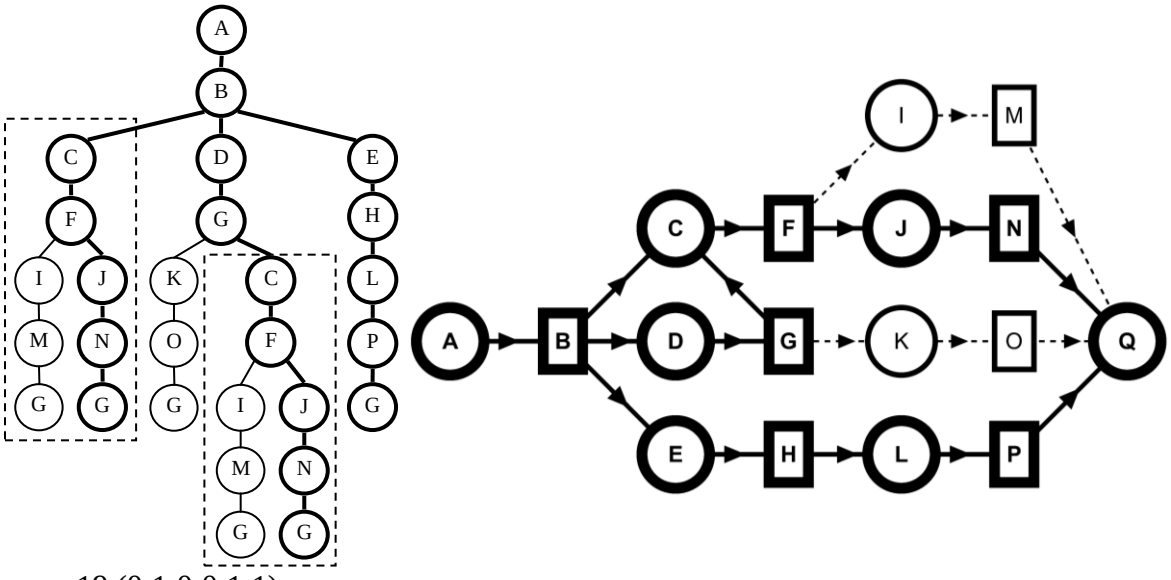
(b) podgraf 28



(c) podgraf 29

Obrázek 29 – Graf A se znázorněnými duplicitními podgrafy 21, 28 a 29

V popisu výsledků řešení 1 (kapitola 3.3.2) je zmíněno, že ne všechny grafy s vícenásobným použitím určité hrany jsou duplicitní, za příklad si vezměme podgraf 19 na obrázkem 30 (zde je vícenásobný průchod hranou [C,E]).



19 (0;1;0;0;1;1)

Obrázek 30 – Unikátní podgraf 19 s vícenásobným průchodem jednou hranou, který je unikátní

Příloha E – Uživatelská příručka

1. Základní charakteristika

Program zobrazuje zvolený bázový graf (vstupem je XML datový popis grafu z programu CPN Tools[13]) a vybraný podgraf.

Podgrafy jsou generovány podle popsaných algoritmů. Uživatel může zvolit mezi řešením 1 a řešením 3, který je nastaveno jako výchozí. Bylo upuštěno od zpřístupnění řešení 2 pro uživatele (ve zdrojovém kódu programu je řešení 2 zahrnuto) vzhledem k jeho značným nevýhodám.

Před zpřístupněním zobrazení podgrafů – jejich výběr spočívá zadáním pořadového čísla, podgrafy jsou očíslovány podle pořadí zpracování algoritmem – je nutné provést export podgrafů do souboru, z tohoto souboru bude program číst data a zobrazovat je. Externí uložení podgrafů je voleno vzhledem k:

- vysokému množství podgrafů – vysoké nároky na operační paměť počítače,
- časové náročnosti na generování podgrafů u rozsáhlých grafů a s tím spojenou znovupoužitelnost výsledků,
- nemožnosti identifikovat jednotlivé podgrafy bez znalosti všech.

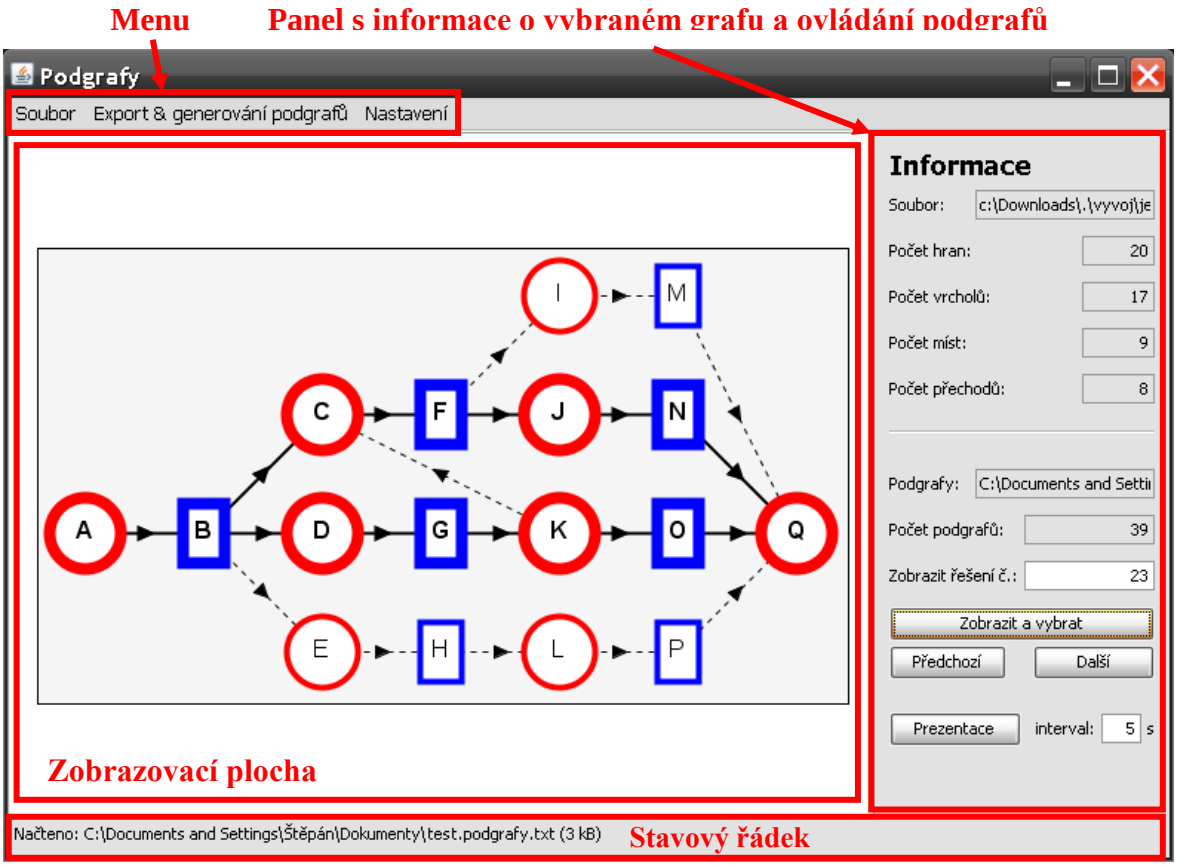
2. Instalace

Program se nemusí instalovat, jedná se jeden spustitelný EXE soubor, který umožňuje spuštění programu na jakémkoliv počítači (OS Windows) vybaveném Java Virtual Machine s J2SE (verze Java SE 1.4.2 JDK nebo vyšší).

3. Rozložení aplikace

Aplikace je rozdělena na dvě hlavní části – Zobrazovací plochu pro vizuální znázornění grafu a část Informace, ve které jsou informace o vybraném grafu, informace o podgrafech a hlavně ovládání zobrazení podgrafů. Ukázka viz obrázek 31.

[13] Více viz http://cpntools.org.



Obrázek 31 – Rozložení aplikace

4. Získání podgrafů / ukázka ovládání

1. Vyberete soubor .cpn pomocí Soubor / Otevřít ...

2. Nabídka Export & generování podgrafů.

3. Volba Všechny podgrafy / Řádkový zápis.

4. Vyberete soubor, do kterého ho zapíší podgrafy.

5. Zpřístupní se ovládání v pravé části aplikace.

6. Zadáním pořadového čísla podgrafu (pořadí je určeno pořadím exportování podgrafu programem) a použitím tlačítka „Zobrazit a vybrat“ zobrazíte zvolený podgraf.

7. Další možnosti a ovládání programu jsou rozepsány v následujících kapitolách.

5. Zobrazení grafu a podgrafů

Vždy je zobrazen bázový graf, a pokud je vybrán podgraf pak i podgraf.

Pokud je vybrán podgraf, jsou hrany bázového grafu zobrazeny přerušovanou čarou a hrany a vrcholy podgrafu jsou zvýrazněny.

Vrcholy typu místo jsou znázorněny červenou kružnicí a vrcholy typu přechod jsou znázorněny modrým obdélníkem.

Se zobrazovací plochou lze pohybovat úchopem myši. Dvojklik způsobí vycentrování obrazu. Přiblížení a oddálení plochy je řešeno kolečkem myši. Kliknutí kolečkem způsobí nastavení do výchozího přiblížení.

6. Menu a ovládání aplikace

Nabídka Soubor

Otevřít XML...

Vyvolá dialogové okno pro výběr souboru s XML strukturou podle softwaru CPN Tools (.xml nebo .cpn) bázového grafu, nad kterým chceme hledat podgrafy.

Aplikace načte XML strukturu, a vytvoří bázový graf, se kterým bude dále pracovat. Program může provést některé úpravy:

- doplnění všem přechodů bez následníka *vrchol-ústí* jako následníka,
- doplnění všem přechodům s názvem začínajícím na „d“ následníka *vrchol-ústí*, pokud má takovýto přechod pouze jednoho následníka.

Pokud existuje ve stejné složce soubor „{název vybraného souboru}.podgrafy.txt“[14], pak je automaticky načten a zpřístupní se funkce pro zobrazení podgrafů v pravé části aplikace.

Otevřít soubor s podgrafy...

Vyvolá dialogové okno pro výběr souboru s exportem podgrafů (je podporován řádkový formát – viz Export & generování podgrafů / Všechny podgrafy / Řádkový zápis). Pokud je takový soubor načten, zpřístupní se funkce pro zobrazení podgrafů v pravé části aplikace.

Konec

Ukončí aplikaci.

[14] Příklad: graf.cpn a graf.podgrafy.txt.

Nabídka Export & generování podgrafů

Bázový graf

- **Řádkový zápis...** – Vyvolá dialogové okno pro výběr názvu souboru, do kterého se provede export bázového grafu.
- **Matice sousednosti...** – Vyvolá dialogové okno pro výběr názvu souboru, do kterého se provede export bázového grafu.
- **Strom cest...** – Vyvolá dialogové okno pro výběr názvu souboru, do kterého se provede export bázového grafu. Výstupem je grafické textové stromu cest použitém v řešení 1.

Všechny podgrafy

- **Řádkový zápis...** – Vyvolá dialogové okno pro výběr názvu souboru, do kterého se provede export bázového grafu. Nabízí soubor „{název souboru bázového grafu}.podgrafy.txt“, tedy takový, který se automaticky načte při otevření bázového grafu. Tento soubor, pokud je export úspěšný bude automaticky načten aplikací jako zdrojový soubor podgrafů.
- **Matice sousednosti...** – Vyvolá dialogové okno pro výběr názvu souboru, do kterého se provede export bázového grafu.

Aktuální podgraf

- **Řádkový zápis...** – Vyvolá dialogové okno pro výběr názvu souboru, do kterého se provede export aktuálního podgrafu.
- **Matice sousednosti...** – Vyvolá dialogové okno pro výběr názvu souboru, do kterého se provede export aktuálního podgrafu.
- **Strom cest...** – Vyvolá dialogové okno pro výběr názvu souboru, do kterého se provede export aktuálního podgrafu. Výstupem je textové znázornění stromu cest aktuálního podgrafu podle řešení 1.

Obrázek

- **Formát JPG...** – Vyvolá dialogové okno pro výběr názvu souboru, do kterého chcete uložit aktuální stav grafického znázornění programu ve formátu JPG.
- **Formát PNG...** – Vyvolá dialogové okno pro výběr názvu souboru, do kterého chcete uložit aktuální stav grafického znázornění programu ve formátu PNG.

- **Formát GIF...** – Vyvolá dialogové okno pro výběr názvu souboru, do kterého chcete uložit aktuální stav grafického znázornění programu ve formátu GIF.

Nabídka Nastavení

Generování podgrafů

Možnost volby mezi **Řešení 1 (pomalejší)** a **Řešení 3 (rychlejší)**. Výchozí stav je Řešení 3. Podle tohoto nastavení volí program algoritmus pro export všech podgrafů.

7. Pravá – Informační část aplikace

Tato část aplikace je v pravé části a poskytuje základní informace o vybraném bázovém grafu a podgrafech. Z této části lze ovládat zobrazení a výběr podgrafů.

Spodní část se zpřístupní pouze, pokud bude vybrán soubor s podgrafy. Je možné zobraz podgraf specifikovaný pořadím pomocí tlačítka „Zobrazit a vybrat“.

Také lze spustit funkci prezentaci podgrafů, ve které budou zobrazeny podgrafy od aktuálního po poslední ve volitelném časovém intervalu.

8. Export dat

Výsledkem exportu je textový soubor. Lze exportovat bázový graf, všechny podgrafy nebo aktuální podgraf. K dispozici jsou dva formáty exportu:

Řádkový zápis

V tomto formátu jsou znázorněné vrcholy a v závorce jsou uvedeny jejich následníci.

Příklad:

```
1:A(B);B(E);E(H);H(L);L(P);P(Q);Q
```

Matice sousednosti

Matice sousednosti se oproti standardní verzi liší. Je zde navíc prvek –1. Ten označuje nepoužité hrany bázového grafu.

Příklad:

```
=================== PODGRAF #1 ====================
   a  B  C  D  E  F  G  H  I  J  K  L  M  N  O  P  Q
A  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
B  0  0 -1 -1  1  0  0  0  0  0  0  0  0  0  0  0  0
C  0  0  0  0  0 -1  0  0  0  0  0  0  0  0  0  0  0
D  0  0  0  0  0  0 -1  0  0  0  0  0  0  0  0  0  0
E  0  0  0  0  0  0  0  1  0  0  0  0  0  0  0  0  0
F  0  0  0  0  0  0  0  0 -1 -1  0  0  0  0  0  0  0
G  0  0 -1  0  0  0  0  0  0  0 -1  0  0  0  0  0  0
H  0  0  0  0  0  0  0  0  0  0  0  1  0  0  0  0  0
I  0  0  0  0  0  0  0  0  0  0  0  0 -1  0  0  0  0
J  0  0  0  0  0  0  0  0  0  0  0  0  0 -1  0  0  0
K  0  0  0  0  0  0  0  0  0  0  0  0  0  0 -1  0  0
L  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1  0
M  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 -1
N  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 -1
O  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 -1
P  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1
Q  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
```

Formát souboru

Jedná se o standardní textový soubor, mezi jednotlivými záznamy je použit oddělovač „nový řádek“. (Při exportu všech podgrafů / řádkový výpis je na prvním řádku uložen bázový graf, podle kterého se kontroluje příslušnost souboru k bázovému grafu).