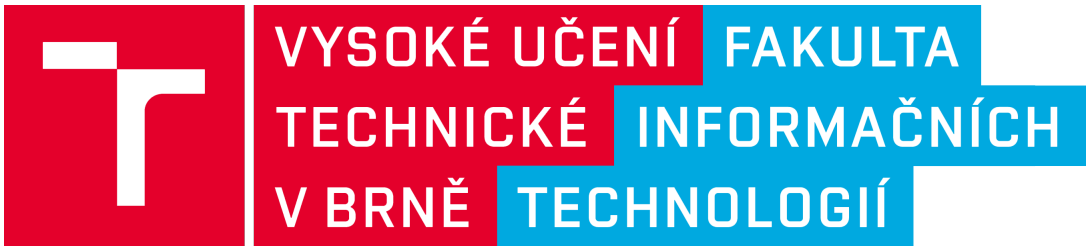


Dokumentace k projektu

Náhradní projekt z předmětu IAL

Varianta 5 – Rovinnost grafu

Závodský Ľubomír	xzavod14
Novák David	xnovak2r
Vráblik Matúš	xvrabl05
Mlynarič Daniel	xmlyna10

8. prosince 2021

Obsah

1 Úvod **3**

2 Vstupní grafy **3**
2.1 Korektnost grafů 3

3 Načítání grafů **4**

4 Zjednodušení grafů **4**

5 Prohledávací algoritmus **5**
5.1 Prohledávání na K(3,3) 5
5.2 Prohledávání na K(5) 5

6 Testování **6**
6.1 Závěr testování 7

1 Úvod

Cílem našeho projektu bylo vytvořit program, který dokáže určit, zda je graf (případně grafy) planární, či ne. Po seznámení se s problematikou a následnou konzultací jsme se jako tým rozhodli pro využití tvrzení pana Kuratowského. Ten pro určení, zda graf je, nebo není rovinný, využívá skutečnost, že žádný rovinný graf neobsahuje podgraf izomorfní s grafy K(5) nebo K(3,3).

2 Vstupní grafy

Pro testování jsme vytvořili řadu vstupů, které obsahují nejrůznější možné případy vstupů pro určení, zda program dokáže správně rozlišit, kdy je graf K(5) nebo K(3,3) v posuzovaném grafu podgrafem. Dále jsme takto testovali, jestli náš program dokáže správně zjednodušit vstupy a odstranit nepotřebné uzly, což slouží pro zrychlení našeho search algoritmu v grafech, které obsahují velké množství uzlů. Dále tento vstup obsahuje grafy, u kterých lze rovnou určit, že planární určitě jsou ať už z důvodu nedostatku vrcholů, hran, nebo dalších.

Pro následné dávkové testování jsme využili námi vytvořený program, který dokáže vytvořit zadané množství grafů, které mají předem zadaný maximální počet uzlů o maximálním počtem z nich vycházejících hran. Pozdější dvě skutečnosti jsou však ponechány náhodě a konkrétní grafy tedy mají svůj specifický počet uzlů, kdy jednotlivé uzly mají svůj specifický počet hran vedoucích do ostatních uzlů. Díky implementaci se však nejedná o *stupeň uzlu*, protože do uzlů může vést ničím neomezený počet hran.

Formát vstupních grafů:	
# Graph	*číslo grafu*
Vrchol grafu	Vrcholy do kterých vedou hrany
# Graph	1
1	1,3,5,2
2	2,3
3	3
4	4,1,5
5	5

Tabulka 1: Vstupní grafy

2.1 Korektnost grafů

Aby program fungoval správně, musí být grafy generovány podle určitých, námi zvolených pravidel.

- Vrcholy grafu jsou posloupnost čísel začínající jedničkou a zvyšující se vždy o jedna.
- Žádný vrchol nesmí být bez hran, které vedou z něj, i kdyby jedinou hranou vrcholu byla smyčka.
- Žádná hrana nesmí vést do vrcholu, který není definovaný ve sloupci **Vrchol grafu**

3 Načítání grafů

Ve zdrojovém souboru *graph.c* se pomocí standardní funkce jazyka c (fopen a následně fgetc)načte textový soubor který je uloženy do dynamického stringu do struktury, jejíž funkce jsou umístěny v souborech *array.c* a *array.h*. Tato struktura obsahuje pole dynamických stringů do kterých se uloží načtené data a následně se do proměnných ve struktuře uloží počet řádků struktury a alokovaná velkost pole na data.

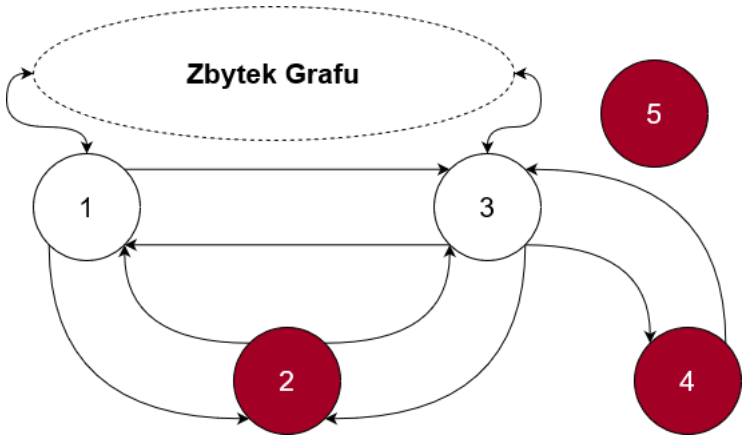
Po načtení celého souboru se soubor zavře funkcí fclose a následně se pošle struktura s daty (grafy) na vyhodnocení do funkce solvegraph umístěné v souboru *solution.c*. Tento způsob byl zvolený z důvodu, že pokud by během vyhodnocování grafů došlo k problému načítání souboru, nedokázali by jsme s jistotou zajistit, že se program ukonči správně. Proto jsme zvolili možnost prvně načíst všechny grafy a až následně vyhodnocovat jejich rovinnost.

4 Zjednodušení grafů

Grafy, které jsou do funkce *solution.c*, která zprostředkovává veškerou vnitřní logiku, předávány ve struktuře *array*. Z této struktury jsou postupně, po jednom grafu, přepsány grafy do *matic integerů*, díky čemuž jsou odstraněny jména grafů. Z *matic integerů* jsou následně jednotlivé grafy uloženy do struktur *graph*, poté zjednodušeny a následně vyřešeny. Po vyřešení problému rovinnosti grafu je smazána příslušná *matice integerů*.

Pro zjednodušení grafů přichází na řadu funkce *rmExcessEdges*, *checkEdgePoints* a *rmExcessPoints*.

- **rmExcessEdges** - Funkce nalezne a odstraní všechny násobné hrany a smyčky.
- **checkEdgePoints** - Funkce zajistí, že všechny body jsou propojené tam i zpět, protože z hlediska rovinnosti grafu je orientace hran nepodstatná.
- **rmExcessPoints** - Funkce odstraní body (a jejich hrany), které mají stupeň uzlu 0, 1, nebo 2. U uzlů se stupněm 2 dojde k jeho nahrazení hranou, v případě, že body, které by měla spojovat, ještě hranu nemají. Pokud dojde k odstranění uzlů, volá se funkce znovu od prvního bodu.



Obrázek 1: Odstraňované body

Vždy je tedy vybrán graf, který je zrovna na řadě, ve struktuře *array*. Ten je převeden do struktury *graph* a zjednodušen. Poté je rozhodnuto, zda je, či není planární a následně se přechází na další graf, dokud není zpracován poslední.

5 Prohledávací algoritmus

Prohledávání grafu proběhne pro graf postupně. Jako první proběhne testování na graf izomorfní s grafem K(3,3), následně, pokud nebyl graf K(3,3) nalezen proběhne testování na K(5). Na závěr proběhne vypsání výsledku prohledávání.

5.1 Prohledávání na K(3,3)

1. Algoritmus vezme první bod [b1] grafu a vezme první bod, do kterého z něj vede hrana [b2].
2. U tohoto bodu opět vezme první prvek [b3], který se však nesmí rovnat bodu [b1].
3. Zjistí, jestli body [b1] a [b3] mají alespoň 3 společné body, do kterých vedou z obou bodů hrany, včetně bodu [b2], dostaneme společné body [b1,3].
4. Následně se vybere bod [b4] z bodů [b1,3], kde [b2] a [b4] jsou rozdílné body.
5. Bod [b5] nalezneme ze společných bodů [b2,4], které nalezneme stejným postupem jako v kroku 3.
6. Bod [b6] nalezneme v množině společných bodů [b1,3,5], které dostaneme průnikem bodů [b1,3] a hran bodu [b5].
7. Zkontrolujeme, zda body [b1], [b3] a [b5] jsou v množině společných bodů [b2,4,6], kterou dostaneme stejným postupem jako v kroku 6.
8. Je nalezen graf K(3,3) a algoritmus končí a graf je vyřešený.

- Pokud v kterémkoliv kroku není nalezena hrana s požadovaným bodem, algoritmus se vrátí o 1 krok nazpět a pokračuje bodem s následující hranou.
- V případě, že jsou vyčerpány všechny hrany bodu [b1], smaže tento bod v případě, že je stupeň bodu menší, než 4. Toto nijak neovlivní ani prohledávání na K(5), jelikož body grafu K(5) mají minimální stupeň bodu roven 4. Následující bod bodu [b1] se tedy stává novým bodem [b1].
- Pokud neexistuje následující bod bodu [b1], nebo je počet bodů grafu menší, než 6, algoritmus končí a následuje prohledávání na K(5).

5.2 Prohledávání na K(5)

1. Algoritmus vezme první bod [b1] grafu a spočítá společné body s každým bodem, do kterého z něj vede hrana. Pokud má bod [b1] alespoň 4 takové body, u kterých existují alespoň 3 hrany do společných bodů, pokračuje prohledávání na K(5). V opačném případě je tento bod smazán a novým bodem [b1] se stává následující bod po [b1].
2. První z těchto bodů vezme jako bod [b2] a najde množinu společných bodů [b1,2].
3. Další z těchto bodů vezme jako bod [b3], najde množinu společných bodů [b1,3] a zkontroluje, zda množina [b1,3] obsahuje bod [b2]. Pokud ano, vytvoří novou množinu [b1,2,3] průnikem množin [b1,2] a [b1,3]. Pokud ne, opakuji krok 3 s následujícím bodem.
4. Algoritmus a zkontroluje zda je počet prvků této množiny větší, než 3. Pokud není splněna podmínka, algoritmus se vrací do kroku 3 a vezme následující bod jako nový [b3].

5. Zopakuje krok 3 s bodem [b4] a množinou [b1,2,3], nalezne množinu společných bodů [b1,2,3,4], která musí obsahovat body [b2] a [b3]. Pokud ne, opakuji krok 5 s následujícím bodem.

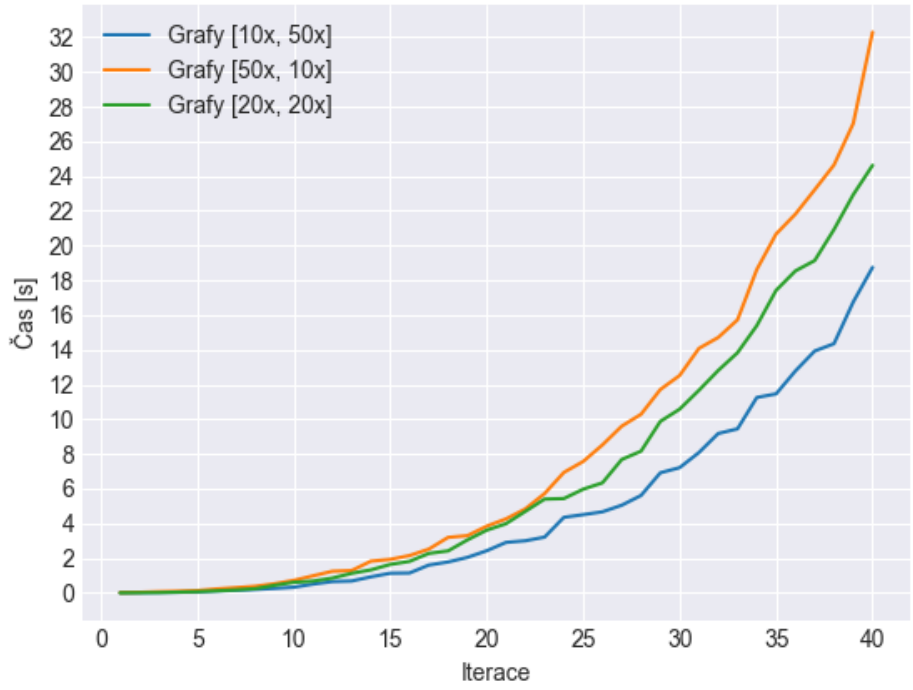
6. Zopakuje krok 3 s bodem [b5] a množinou bodů [b1,2,3,4] nalezne množinu společných bodů [b1,2,3,4,5], která musí obsahovat body [b2], [b3] a [b4]. Pokud ne, opakuji krok 6 s následujícím bodem.

- Pokud v bodě neexistuje další hrana s bodem, který by mohl být testován, algoritmus se o krok vrací a bere příslušný následující bod.
- Pokud v grafu neexistuje další bod, který by mohl být testován, algoritmus končí, graf je planární.

6 Testování

Testování probíhalo na našich počítačích a na školním serveru Eva. K testování byly využity již dříve zmiňovaný ručně tvořený soubor s řadou vstupů a vstupy generované generátorem náhodných grafů. Pro automatizaci generování grafů a spouštění programu byly vytvořeny jednoduché bash skripty a pro získání dat o čase byl použit linuxový příkaz *time*.

Testování probíhalo ve 40 iteracích na 100 grafech s rozdílným počtem bodů a hran. V legendě grafu jsou počty bodů na prvním místě dvojice bodů a počty hran na místě druhém. X je číslo iterace. Pokud tedy máme [10x, 50x], při první iteraci máme 100 grafů, které mají až 10 bodů a až 50 hran, které z těchto bodů vycházejí, při druhé iteraci 100 grafů s 20 body a 100 stranami atd.



Obrázek 2: Graf výsledků testování

6.1 Závěr testování

Z výsledků testování můžeme vyvodit, že program má exponenciální časovou složitost, která je spíše ovlivněna počtem hran, než počtem bodů. Bylo testováno i spouštění programu se stoupajícím počtem grafů, ale ten ovlivňuje rychlost programu více méně lineárně.