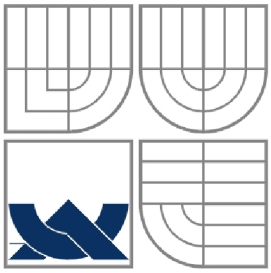
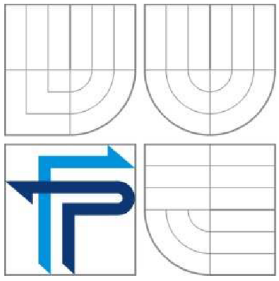


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA PODNIKATELSKÁ
ÚSTAV INFORMATIKY

FACULTY OF BUSINESS AND MANAGEMENT
INSTITUTE OF INFORMATICS

GRAFY A ALGORITMY PRO HLEDÁNÍ NEJKRATŠÍCH CEST

GRAPHS AND SHORTEST PATH ALGORITHMS

BAKALÁŘSKÁ PRÁCE
BACHELOR THESIS

AUTOR PRÁCE
AUTHOR
MICHAL HAMERNÍK

VEDOUCÍ PRÁCE
SUPERVISOR
Mgr. MARTINA BOBALOVÁ, Ph.D.

BRNO 2009

Anotace

Práce představuje učební text zaměřený na problematiku teorie grafů a grafových algoritmů. Teorie grafů pomáhá často řešit problémy a vztahy mezi částmi komplikovaných celků a grafové algoritmy pomáhají tyto problémy rychle a efektivně optimalizovat. V této práci jsou uvedeny základy teorie grafů, dále popis vybraných algoritmů a jejich případné praktické využití. Práce může být využita jako doplňující text při výuce předmětu Diskrétní matematika na Fakultě podnikatelské Vysokého učení technického v Brně.

Annotation

This bachelor thesis represents an educational text focused on graph theory and graph algorithms. The graph theory often helps to solve problems between parts of a complicated unit and graph algorithms are quick and effective in their optimization.
Basics of graph theory, samples of graph algorithms and practical examples of use are described in it. This thesis can be used as a supplementary text in Discrete Mathematics taught at Faculty of Business and Management in Brno University of Technology.

Klíčová slova

Orientovaný graf, neorientovaný graf, algoritmy pro hledání nejkratších cest, Dijkstrův algoritmus, Floyd-Warshallův algoritmus, Bellman-Fordův algoritmus, Johnsonův algoritmus

Keywords

Directed graph, non-directed graph, shortest path algorithms, Dijkstra´s algorithm, Floyd-Warshall algorithm, Bellman-Ford algorithm, Johnson´s algorithm

HAMERNÍK, M. *Grafy a algoritmy pro hledání nejkratších cest.* Brno: Vysoké učení technické v Brně, Fakulta podnikatelská, 2009. 47 s. Vedoucí bakalářské práce Mgr. Martina Bobalová, Ph.D.

Čestné prohlášení

Prohlašuji, že předložená bakalářská práce je původní a zpracoval jsem ji samostatně. Prohlašuji, že citace použitých pramenů je úplná, že jsem ve své práci neporušil autorská práva (ve smyslu Zákona č. 121/2000 Sb., o právu autorském a o právech souvisejících s právem autorským).

V Kopřivnici dne 21. května 2009

Podpis

Poděkování

Na tomto místě bych rád poděkoval paní Mgr. Martině Bobalové, Ph.D. za její pomoc, rady a připomínky v průběhu zpracování této bakalářské práce.

Obsah

1. Úvod

Naše jednadvacáté století je století dynamickým. Rychlý rozvoj a vývoj můžeme sledovat ve všech oblastech lidských činností. Motorem veškerého myšlení je mozek, který je zdrojem nápadů a myšlenek. Ale k jejich realizaci jsou zapotřebí stroje, přístroje a různá zařízení, kterými pak nápady uvedeme do praxe, aby nám už tak dost uspěchaný a komplikovaný život ulehčily.

Klasické obory jako jsou matematika, fyzika, chemie aj. jsou proto stále v popředí zájmu vědců a výzkumných pracovníků, protože tvoří pilíře dalších vědních oborů, bez kterých si náš život již těžko dokážeme představit – např. informatika.

Současný výzkum informatiky se věnuje zlepšení komunikace mezi člověkem a počítačem a zefektivnění způsobů tvorby, využití a hledání informací mimo jiné i za použití struktur zvané grafy, jejichž vlastnosti zkoumá teorie grafů.

Pod pojmem graf si většina lidí představuje graf statistický nebo graf funkce. Tato práce o zmíněných druzích grafů pojednávat nebude, nýbrž se zaměří na grafy reprezentované jako diskrétní struktury, které jsou tvořeny vrcholy, a ty propojeny hranami, sloužící jako abstrakce různých problémů, kde jsou důležitější topologické vlastnosti než například umístění v prostoru.

Takovéto struktury mohou pomáhat nejenom v informatice ale i v elektrotechnice, chemii, informatice, sociologii a dalších oborech.

Jedním z typických problémů teorie grafů je hledání nejkratší cesty a na tento problém je práce zaměřena.

2. Historický vývoj

2. 1 Celosvětový vývoj

Za zakladatele teorie grafů se považuje Leonhard Euler, který roku 1736 ve své publikaci Solutio problematis ad geometriam situs pertinentis řešil úlohu, zda je možné projít přes sedm mostů ve městě Königsbergu (dnešní Kaliningrad), kde podmínkou bylo, že po každém mostu je možné projít právě jednou.

Grafy a jejich využívání v úlohách pak nadlouho zůstávaly na okraji zájmu matematiků až do roku 1847, kdy Gustav Kirchhoff publikoval zákony, které platí v elektrických obvodech a slouží k výpočtu napětí a proudu v jednotlivých větvích obvodu.

V roce 1852 byl Francisem Guthriem představen problém „čtyř barev“. Byla položena jednoduchá otázka, zda je možné obarvit libovolnou mapu za pomoci nejvýše čtyř barev tak, aby každé dvě sousední země měly barvu odlišnou.

Teorie grafů zasáhla v roce 1857 i chemii, když Arthur Cayley za pomocí grafů zjišťoval počty různých uskupení molekul alkanů.

2. 2 Vývoj v českých zemích

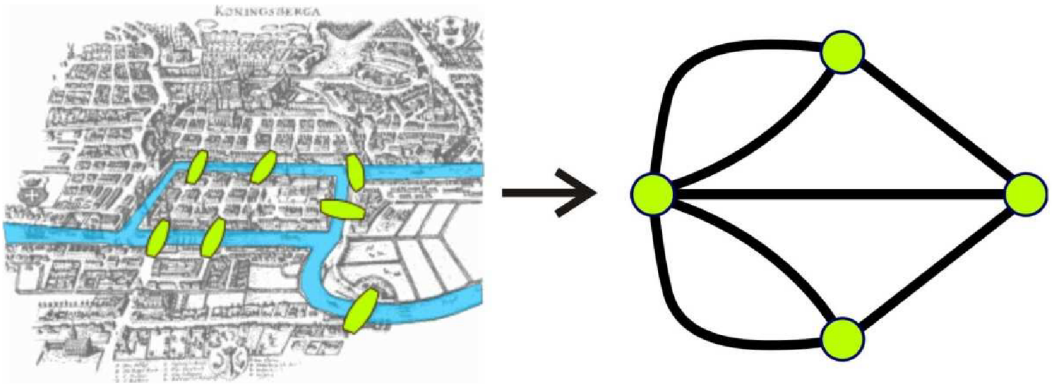
Ve 20. století docházelo k velkému rozvoji teorie grafů i za účasti českých matematiků. V roce 1926 publikoval Otakar Borůvka algoritmus pro nalezení minimální kostry grafu za účelem výstavby elektrických sítí. Stejný problém, avšak jiným algoritmem, vyřešil v roce 1930 také Vojtěch Jarník.

Jedním z novodobých významných českých matematiků v oblasti teorie grafů je Václav Chvátal z Concordia University in Montreal, který se mimo jiné, podílel na vývoji programu Concorde TSP Solver sloužící k řešení problému obchodního cestujícího (viz kapitola 3). Concorde je využíván konkrétněji k řešení problémů v genetice – při mapování genového fondu, k predikci funkcí proteinů, pro návrhy tras vozidlům, konverze bitmapových obrázků na malby kreslené nepřerušenými tahy, pro plánování tras lodí pro seismický výzkum atd., a to vše v rekordně krátkých časech.

3. Nejznámější grafové úlohy[1]

Úloha o sedmi mostech města Königsberg

Jak již bylo zmíněno, tak tato slavná úloha, jejímž úkolem bylo zjistit, zda bylo možné přejít sedm mostů ve městě takovým způsobem, aby ten, kdo se o to pokusí, vstoupil na každý most pouze jednou. Leonhard Euler jako první dokázal, že to možné není.



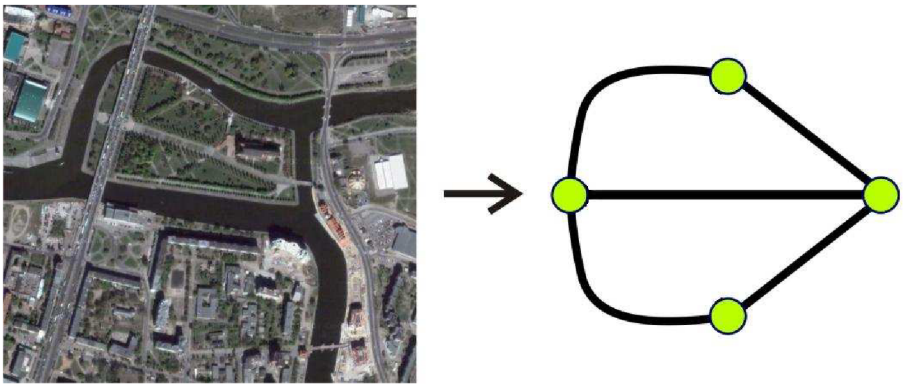
Obrázek 1: Úloha o sedmi mostech města Königsberg

Euler na základě mapy města Königsberg problém převedl na graf a dokázal, že eulerovský tah v grafu neexistuje (viz obrázek číslo 1). Eulerovské grafy mají totiž tu vlastnost, že je možné je „nakreslit jedním tahem“. Jednoduchá definice pro eulerovské grafy zní, že graf G je eulerovský, právě když je souvislý a každý vrchol má sudý stupeň.

Poznámka:

Dnešní situace v Königsbergu, v současné době nazývanému Kaliningrad, je díky historickému dění v této oblasti jiná. Jeden most byl zničen již před druhou světovou válkou a pak byl znovu vybudován Němci v roce 1935, dva z mostů byly zničeny za britského náletu v roce 1944 a další dva byly později zničeny Sověty při stavbě dálnice.

[1] Zpracováno dle: *Graph theory - History, Wikipedie otevřená encyklopedie* (12)



Obrázek 2: Dnešní pohled na Kaliningrad[2]

Z mapy a obrázku číslo 2 je patrné, že po výše zmíněných historických událostech je možné projít mosty v Kaliningradu, propojující stejný ostrůvek s pevninou, takovým způsobem, že se každý most projde pouze jednou.

Úloha jezdce

V této úloze je úkolem, aby se šachový jezdec v souladu s šachovými pravidly pohyboval po prázdné šachovnici tak, aby každé pole navštívil právě jednou.

Tímto problémem se zabývali již středověcí arabští a indičtí učenci a první řešení jsou známá již z 9. století.

Tato úloha se řeší dále v různých modifikacích, kde se mění velikosti šachovnice nebo je podmínkou, že jezdec se musí pohybovat po šachovnici 8x8 polí tak, aby měl možnost se vrátit na výchozí místo, kde přesně ve 26 534 728 821 064 případech jezdec končí na poli, odkud opět ohrožuje své startovní pole.

Hamiltonova hra

V roce 1857 sir William Hamilton vymyslel hru, jejímž úkolem bylo pospojovat všechny vrcholy pravidelného dvanáctistěnu tak, aby byl každý vrchol použit právě jednou, a vrátit se do výchozího místa.

Problém čtyř barev

Problém čtyř barev je formulován tímto způsobem: „Stačí čtyři barvy na obarvení libovolné politické mapy tak, aby žádné dva sousedící státy nebyly obarveny stejnou

[2] Zpracováno dle: *Google maps – Kaliningrad* (13)

barvou?“ Obecněji se lze tázat na minimální potřebný počet barev. Podařilo se dokázat, že pět barev postačuje. Oproti tomu tvrzení, že čtyři barvy stačí, dlouhou dobu odolávalo všem pokusům o důkaz, nikdo však také nebyl schopen nalézt mapu, která by ho vyvrátila.

Důkaz představili až roku 1976 američtí matematici Kenneth Appel a Wolfgang Haken tím, že pomocí počítačového programu vymodelovali 1936 možných konfigurací. Následně pak dokázali, že tyto konfigurace pokrývají všechny možnosti (k tomu potřebovali 1200 hodin procesorového času). Tento důkaz však velká část matematiků odmítá akceptovat, protože jej žádný matematik není schopen přímo zkontrolovat.

Úloha obchodního cestujícího a čínského pošťáka

Tyto úlohy patří mezi tzv. NP-úplné problémy, což znamená, že v obecném případě není znám algoritmus jak nalézt řešení v rozumném čase, které by bylo v čase úměrném nějaké mocnině počtu vrcholů. V praxi se tyto úlohy řeší pouze přibližně heuristickými algoritmy.

Problém obchodního cestujícího je tedy obtížný diskrétní optimalizační problém, jehož úkolem je najít nejkratší možnou trasu, procházející všemi městy a vracející se nazpět do výchozího města.

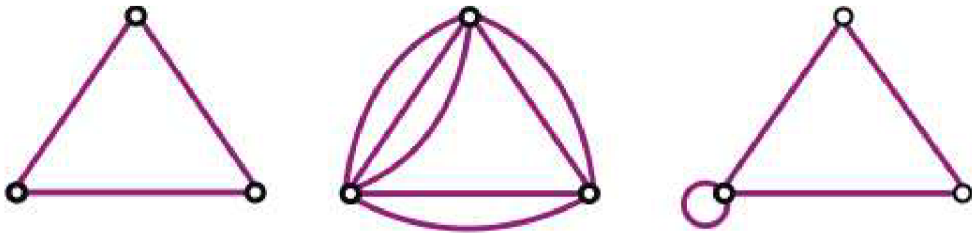
Problém čínského pošťáka je definován způsobem, že pošťák musí denně projít všechny ulice svého obvodu a vrátit se na místo, odkud vyšel. Jde mu o to, aby tato cesta byla co nejkratší, avšak na rozdíl od úlohy obchodního cestujícího může projít některými místy vícekrát.

4. Základní pojmy z teorie grafů

„Graf *G* je uspořádaná dvojice *(V,E)*, kde *V* je libovolná neprázdná množina a *E* je množina dvoubodových podmnožin množiny *V*. Prvky množiny *V* se jmenují vrcholy (uzly) grafu *G* a prvky množiny *E* hrany grafu.“[3]

Grafy se dle svých vlastností dělí na mnoho podskupin, kde základní rozdělení je na jednoduchý graf a multigraf.

Jednoduchý graf se skládá z vrcholů a vždy dva různé vrcholy spojuje právě jedna hrana. Naproti tomu takzvaný multigraf se skládá z vrcholů a hran, přičemž dva různé vrcholy může spojovat více různých hran. Občas potřebujeme v grafu využít i takzvané smyčky, které však v multigrafech nejsou povoleny. Takovým to grafům pak říkáme pseudografy (viz obrázek číslo 3).



Obrázek 3: Jednoduchý graf, multigraf a pseudograf

Konkrétnější, a pro tuto práci důležitější dělení grafů, je na grafy neorientované a orientované.

Neorientované grafy

„Neorientovaný graf G je uspořádaná dvojice (V,E), kde V je libovolná neprázdná množina a *E* je množina dvoubodových podmnožin množiny *V*. Prvky množiny *V* se jmenují vrcholy (uzly) grafu *G* a prvky množiny *E* hrany grafu. Hrany se zapisují způsobem e ={x,y}, tedy vrcholy x a y spolu incidují. Už z principu neorientovaných grafů navíc platí, že množiny {x,y}a {y,x} jsou si rovny. Další vlastností je, že jedna

[3] Zpracováno dle: MATOUŠEK, J. a NEŠETŘIL, J. *Kapitoly z diskrétní matematiky*, s. 95, (3)

hrana má společné maximálně dva vrcholy. Vrchol, který neinciduje s žádnou hranou je nazýván izolovaným.“ [4]

Stupeň vrcholu

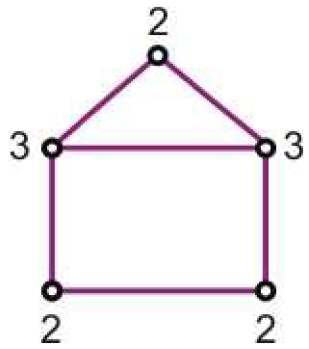
Stupeň vrcholu je vyjádřen číslem, které značí počet hran, se kterými vrchol inciduje (viz obrázek číslo 4).

Stupeň vrcholu značíme |x|.

Je-li vrchol izolovaný pak |x| = 0.

Vlastnosti vrcholů v grafu:

- Součet všech stupňů vrcholů v grafu je roven dvojnásobku počtu hran, tedy $\sum_{x \in V} |x| = 2|E|$
- Počet vrcholů lichého stupně v grafu je vždy sudé číslo

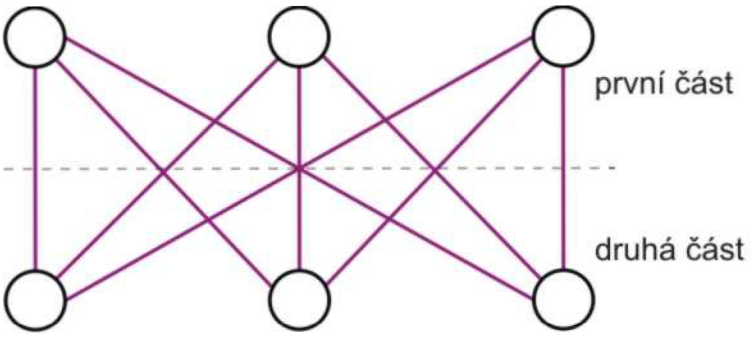


Obrázek 4: Stupně vrcholů v grafu

Bipartitní graf

Pojmem bipartitní graf (viz obrázek číslo 5) se označuje takový graf, jehož množinu vrcholů lze rozdělit na dvě části tak, že žádné dva vrcholy ze stejné množiny nejsou spojeny hranou. Pokud jsou všechny vrcholy množiny z první části propojeny s množinou vrcholů z části druhé, hovoříme o úplném bipartitním grafu.

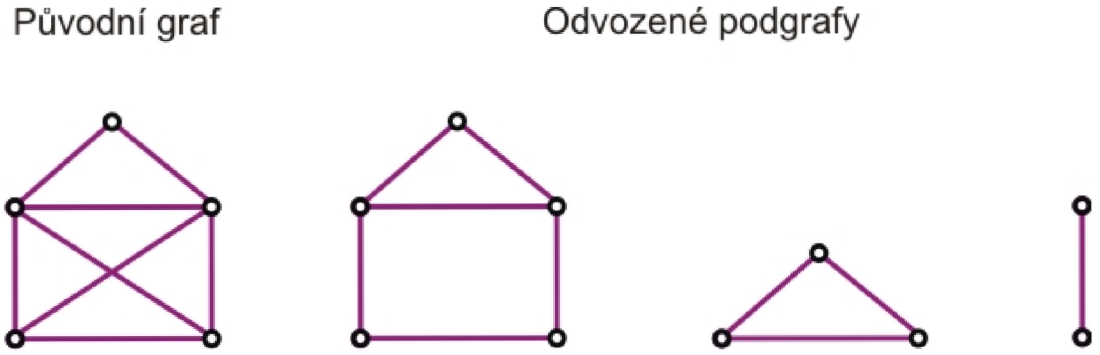
[4] Zpracováno dle: MEZNÍK, Ivan. *Diskrétní matematika*, s. 37 (4)



Obrázek 5: Bipartitní graf

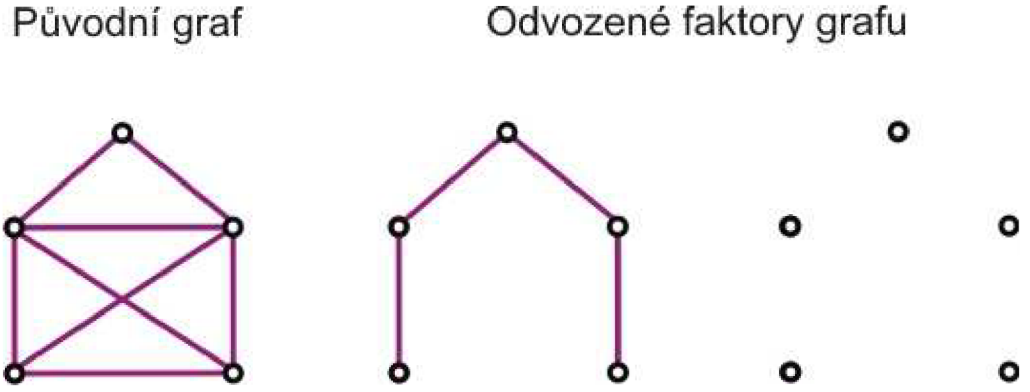
Podgraf, faktor grafu

Podgraf je část grafu, která vznikne vybráním podmnožiny hran spolu s vrcholy, které s danými hranami incidují (viz obrázek číslo 6).



Obrázek 6: Podgraf

Faktor grafu vyjadřuje, že graf má stejnou množinu vrcholů jako původní graf (viz obrázek číslo 7).

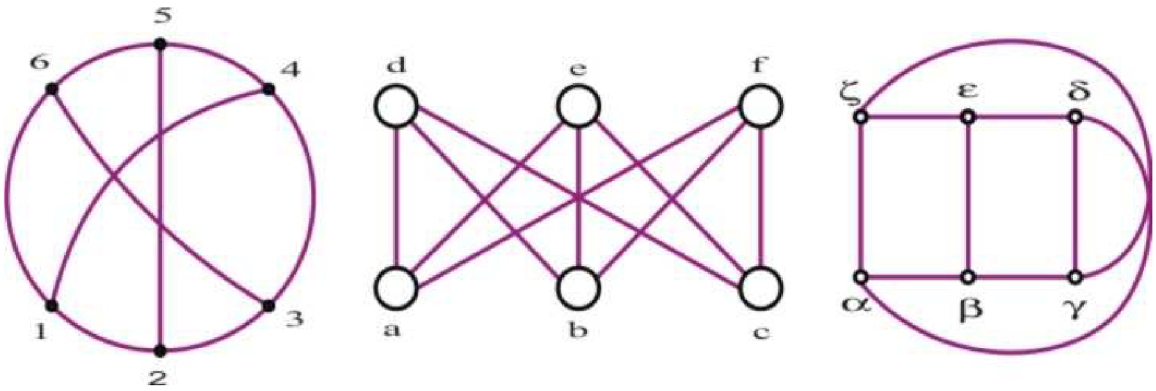


Obrázek 7: Faktor grafu

Isomorfismus

„Dva grafy *G=(V,E)* a *G´=(V´,E´)* nazýváme isomorfní, pokud existuje vzájemné jednoznačné zobrazení - bijektivní.“[5]

Isomorfismus ve své podstatě představuje „přejmenování vrcholů v grafu“, v matematické terminologii znamená ekvivalentní.
Při ověřování isomorfnosti grafů musíme hledat způsob, jak je možné vrcholy prvního grafu přejmenovat tak, aby odpovídaly vrcholům grafu druhého.



Obrázek 8: Isomorfismus

Na obrázku číslo 8 jsou všechny tři grafy isomorfní. Isomorfismus u prvních dvou grafů se dá znázornit například takovýmto zobrazením: 1 – a, 2 – d, 3 – b, 4 – e, 5 – c, 6 – f. Možností takovýchto zobrazení existuje mnohem více.

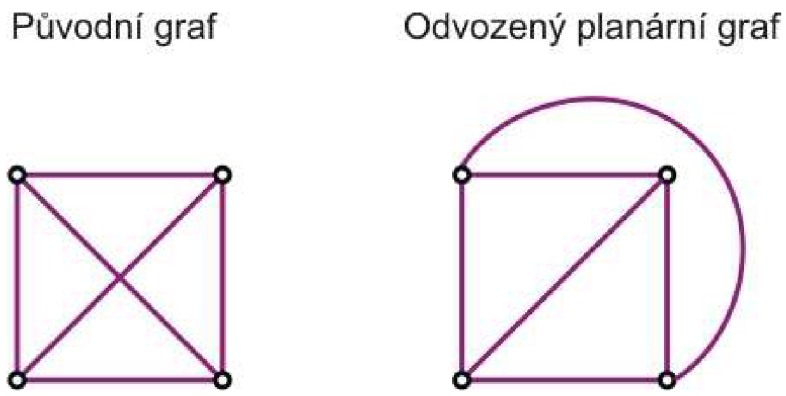
Poznámka:
Pro malé obrázky není zpravidla těžké rozhodnout, zda jsou isomorfní nebo ne, i když na příkladu z obrázku číslo 8 to nemusí být na první pohled patrné. Avšak rozhodování, zda jsou grafy isomorfní, je obecně obtížné a není znám žádný efektivní algoritmus fungující ve všech případech. Dokonce se soudí, že ani žádný efektivní algoritmus neexistuje.

[5] Zpracováno dle: MATOUŠEK, J. a NEŠETŘIL, J. *Kapitoly z diskrétní matematiky*, s. 98, (3)

Planární graf

„Planární (rovinný) graf je graf, pro který existuje takové rovinné zakreslení, že žádné dvě hrany se nekříží.“ [6]

Může se stát, že graf je nakreslen takovým způsobem, že se hrany protínají, ale přitom jinou grafickou interpretací může být nakreslen bez protínání hran. V takovém případě se jedná o planární reprezentaci grafu (viz obrázek číslo 9).



Obrázek 9. Planární graf

Cesta a souvislost v grafu

Cesta

Definice

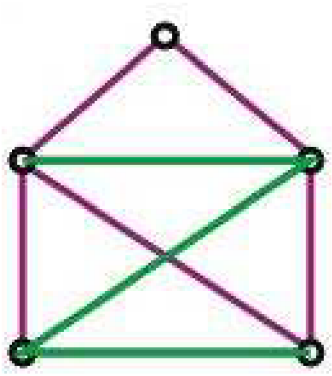
„Cestou v grafu je posloupnost vrcholů a hran (v_0,e_1, ... v_n,e_n), kde vrcholy v_0 ... v_n jsou navzájem různé vrcholy grafu *G* a pro každé $i= 1, 2 ... n$ je $e_i=\{ v_{i-1}, v_i\} \in E(G)$.“[7]

Cesta je tedy posloupnost vrcholů, pro kterou platí, že v grafu existuje hrana z daného vrcholu do jeho následníka (viz obrázek číslo 10). Žádné dva vrcholy (a tedy ani hrany) se přitom neopakují.

Poslední podmínka odlišuje cestu od dvou podobných pojmů – tah a sled:

- tah je posloupnost, kde se mohou opakovat vrcholy, ale ne hrany
- sled je posloupnost, kde se mohou opakovat i hrany

[6] Zpracováno dle: MEZNÍK, Ivan. *Diskrétní matematika*, s. 46 (4)
[7] Zpracováno dle: MATOUŠEK, J. a NEŠETŘIL, J. *Kapitoly z diskrétní matematiky*, s. 102, (3)



Obrázek 10: Cesta v grafu

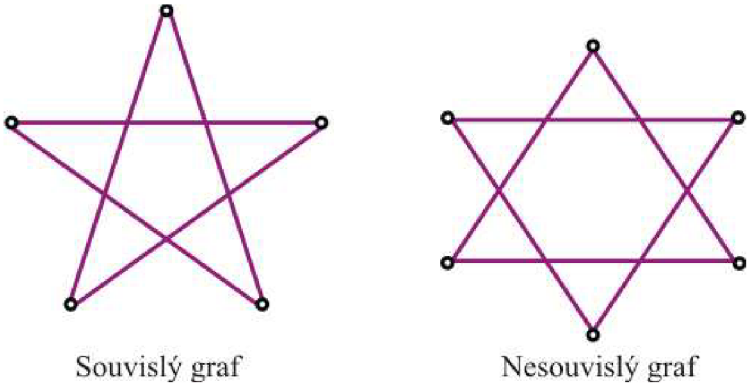
Souvislost

Definice

„Řekněme, že graf $G = (V, E)$ je souvislý, jestliže pro každé dva jeho vrcholy x a y existuje v G cesta z x do y.“[8]

Pokud graf není souvislý, tak části, ze kterých se skládá (a samy jsou souvislé), se nazývají komponenty grafu.

Rozdíl mezi grafem souvislým a nesouvislým je prezentován na obrázku číslo 11.



Obrázek 11: Ilustrace grafové souvislosti a nesouvislosti

Kružnice (cyklus) v grafu

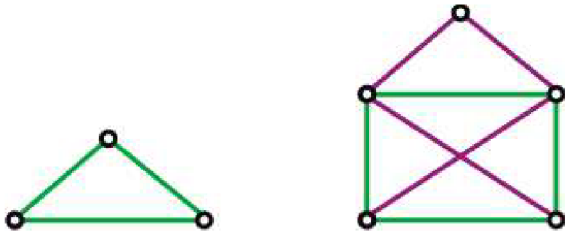
Definice

„Kružnicí v grafu rozumíme uzavřenou posloupnost propojených vrcholů, kde každý vrchol neorientované kružnice má stupeň 2.“[9]

[8] Zpracováno dle: MATOUŠEK, J. a NEŠETŘIL, J. *Kapitoly z diskrétní matematiky*, s. 103, (3)
[9] Zpracováno dle: MATOUŠEK, J. a NEŠETŘIL, J. *Kapitoly z diskrétní matematiky*, s. 102, (3)

Nejkratší možnou kružnicí v grafech je trojúhelník, tedy úplný graf se třemi vrcholy.

Definice cesty a kružnice jsou si velmi podobné. V obou dvou případech pracujeme s posloupností hran a vrcholů. Avšak na rozdíl od cesty je v kružnici první a poslední vrchol stejný a na navíc kružnice nepovoluje nulovou délku. Kružnice má výše zmíněnou minimální délku 3 (viz obrázek číslo 12).



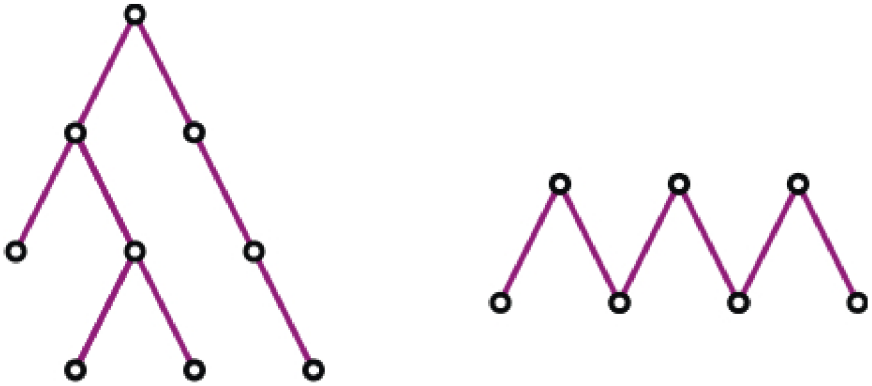
Obrázek 12: Kružnice v grafech

Strom grafu

Definice

Graf je stromem právě tehdy, když každé dva jeho různé vrcholy jsou spojeny jedinou cestou.

Strom je tedy souvislý graf neobsahující kružnici (acyklický) a mezi každými dvěma jeho vrcholy existuje právě jedna cesta (viz obrázek číslo 13).



Obrázek 13: Ukázky stromů

Pro stromy platí tzv. Eulerův vzorec pro stromy

$|V| = |E| + 1$

Poznámka:

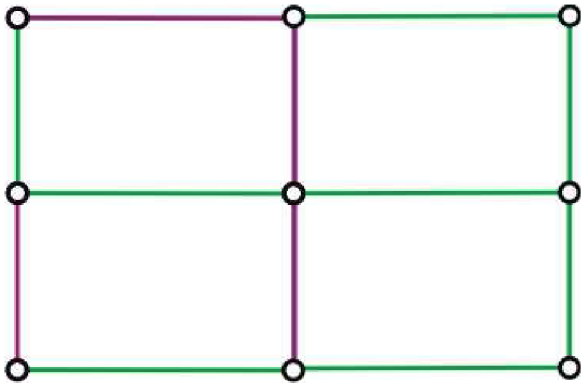
Stromy se často využívají v informatice k ukládání a vyhledávání dat jako datová struktura. Jejich hlavní výhodou je vysoká efektivita vyhledávání, ale lze je také využít k efektivnímu řazení hodnot.

Kostra grafu

Definice

„Kostrou grafu G rozumíme podgraf v G, který obsahuje všechny vrcholy grafu G a je stromem.“[10]

Kostra grafu je tedy libovolný podgraf, který hranami spojuje všechny vrcholy původního grafu a zároveň sám neobsahuje žádnou kružnici (viz obrázek číslo 14).



Obrázek 14: Kostra grafu

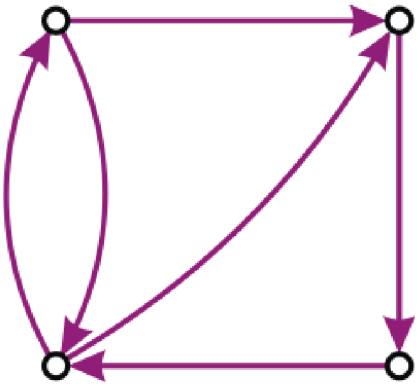
Orientované grafy

Pojmem orientovaný graf (viz obrázek číslo 15) se v teorii grafů označuje takový graf, jehož hrany jsou uspořádané dvojice. Naproti tomu hrany neorientovaných grafů jsou hrany reprezentovány dvouprvkovými množinami. Hrany orientovaného grafu mají tedy pevně danou orientaci a hrany vycházející z x do y a z y do x budou vyjadřovány různými hranami.

[10] Zpracováno dle: MATOUŠEK, J. a NEŠETŘIL, J. *Kapitoly z diskrétní matematiky*, s. 150, (3)

Definice

„Orientovaný graf G je dvojice (V, E), kde E je podmnožina kartézského součinu $V \times V$. Prvky E nazýváme orientované hrany. Orientovaná hrana e má tvar (x, y). Říkáme, že tato orientovaná hrana vychází z x a končí v y."[11]



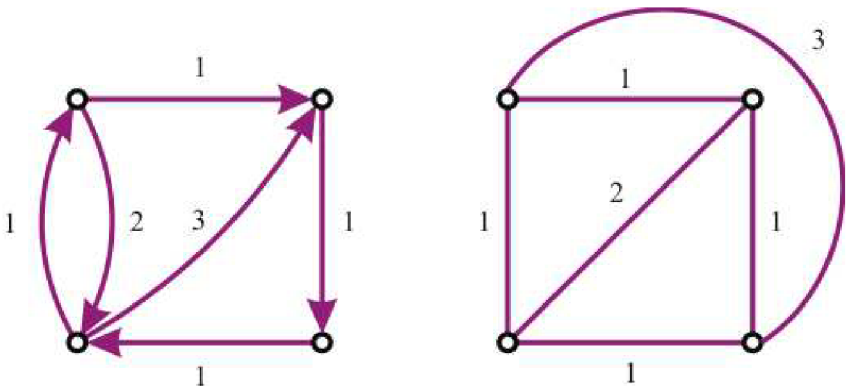
Obrázek 15: Orientovaný graf

Silně souvislý graf

Graf je silně souvislý, existuje-li pro každou dvojici uzlů x a y cesta z x do y i z y do x.

Ohodnocený graf

Pokud k hranám grafů, ať už neorientovaným nebo i orientovaným, přiřadíme čísla, pak je nazýváme ohodnocenými (viz obrázek číslo 16). Takovéto grafy mají poté velmi rozsáhlou možnost využití. Přiřazená čísla mohou reprezentovat vzdálenost mezi městy, cenu, kterou musíme zaplatit za průchod hranou, časovou náročnost úkonu apod.



Obrázek 16: Příklad ohodnocených grafů

[11] Zpracováno dle: MATOUŠEK, J. a NEŠETŘIL, J. *Kapitoly z diskrétní matematiky*, s. 128, (3)

V případě, že jsou v grafu hrany ohodnocené, tak součet hodnot z určitého výchozího uzlu do uzlu koncového vyjadřuje tzv. délku cesty.

V takových grafech poté existuje možnost, za pomocí algoritmů, hledat cesty nejkratší nebo v některých úlohách, typu časové analýzy u metody kritické cesty, naopak cesty nejdelší.

5. Matematická reprezentace grafů[12]

Doposud byly grafy popisovány pomocí různých typů diagramů, avšak grafy se dají popsat i jinými způsoby. Mezi další způsoby popisu grafů patří možnost pomocí seznamu sousedů již zmíněné datové struktury anebo pomocí matic. Jako nejdůležitější maticový popis grafů se využívá matice sousednosti.

Matice sousednosti

Definice:

Nechť je dán obyčejný graf G = (V,E) s množinou hran E = $\{e_1 .. e_n\}$ a množinou uzlů V = $\{v_1 .. v_n\}$. Matice sousednosti bude čtvercová matice řádu *n x n*, která ponese označení S a její prvky s_{ij} se budou řídit předpisy:

- U neorientovaných grafů

  s_{ij} = 1, pokud uzly v_i a v_j budou sousední

  s_{ij} = 0, pokud uzly v_i a v_j sousední nebudou

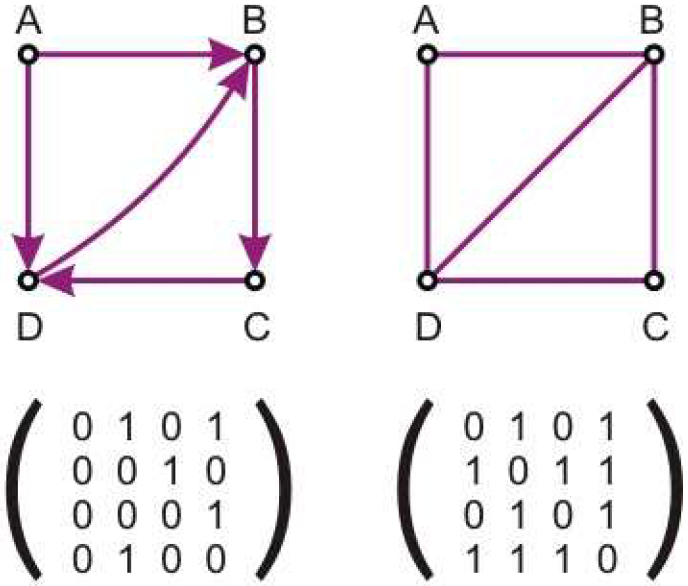
- U orientovaných grafů

  s_{ij} = 1, pokud hrana z uzlu v_i povede do uzlu v_j

  s_{ij} = 0, pokud hrana z uzlu v_i nepovede do uzlu v_j

Z těchto předpisů také vyplývá, že na hlavní diagonále matice, pokud budeme uvažovat o grafech bez smyček, budou vždy nuly, dále že u neorientovaných grafů bude matice symetrická podle hlavní diagonály. Počet jedniček v matici u neorientovaných grafů bude roven dvojnásobku počtu hran, zatímco počet jedniček v matici u orientovaných grafů bude stejný jako počet hran (viz obrázek číslo 17).

[12] Zpracováno dle: VEČERKA, A., *Grafy a grafové algoritmy*, s. 16, (5)



Obrázek 17: Matice sousednosti

U ohodnocených grafů stačí jen jedničky nahradit čísly reprezentující metriku v grafu a dostaneme tak matici sousednosti pro ohodnocené grafy.

Reprezentace grafů při programování

Jelikož se hojně při programování grafových struktur využívá matic sousednosti, tak si programátor vystačí s dvourozměrným polem řádu *n x n*.

Avšak matice sousednosti není vždy nejvhodnějším způsobem reprezentace grafu v paměti počítače. Obzvlášť když má graf málo hran. V těchto případech bývá vhodnější pro každý vrchol zadat seznam jeho sousedů. Takovou reprezentaci je možno udělat pomocí dvou jednorozměrných polí. První pole v tomto případě má stejný počet prvků jako je počet uzlů v grafu. Každému uzlu poté odpovídá jeden prvek pole, v němž je uložena hodnota indexu, od kterého v druhém poli začíná seznam uzlů, které jsou sousedy daného uzlu. V poli, ve kterém jsou uvedeny seznamy sousedů, musí existovat na konci seznamu sousedů jednotlivých uzlů druh oddělovače, který nám říká, kde seznamy sousedů končí. Jako příznak ukončení se dá využít například číslo -1.

Pro urychlení některých komplikovanějších algoritmů se využívají složitější reprezentace grafů. Taková reprezentace může být za pomocí dynamické datové struktury, kde uzel grafu bude reprezentován jako strukturovaný datový typ a každý uzel bude obsahovat pole (seznam) ukazatelů na sousední uzly.

6. Grafové algoritmy pro hledání nejkratších cest

Jednou ze základních úloh v teorii grafů je hledání nejkratší cesty v grafech mezi dvěma zadanými vrcholy. Takový požadavek vzniká v mnoha praktických příkladech např. při hledání nejkratšího dopravního spojení. Dosud objevené algoritmy řešící tento typ úlohy většinou počítají mnohem více, než je od nich požadováno - zpravidla nalézají nejkratší cesty z výchozího vrcholu do všech (nebo mnoha) vrcholů.

V teorii grafů existuje velké množství algoritmů, které korektně fungují při splnění některých grafových podmínek. Některé typy algoritmů fungují například jen tehdy, když se jedná o neorientovaný graf, nebo právě tehdy, když se jedná o graf orientovaný. Také se musí brát v potaz ohodnocení hran, neboť některé algoritmy nedokážou pracovat se záporně ohodnocenými hranami v grafech.

6. 1 Dijkstrův algoritmus

Dijkstrův algoritmus (čti „Dajkstrův“, holandské jméno) je algoritmus, který byl roku 1959 představen informatikem Edsgerem Wybe Dijkstrou, sloužící k nalezení nejkratší cesty v grafu. Je konečný (pro jakýkoliv konečný vstup algoritmus skončí), protože v každém průchodu cyklu se do množiny navštívených uzlů přidá právě jeden uzel. Průchodů cyklem je tedy nejvýše tolik, kolik má graf vrcholů. Funguje nad hranově kladně ohodnoceným grafem a nezáleží na tom, zda je graf orientovaný či nikoliv.

Popis algoritmu

Mějme graf G, v němž hledáme nejkratší cestu. Řekněme, že V je množina všech vrcholů grafu G a množina E obsahuje všechny hrany grafu G. Algoritmus pracuje tak, že si pro každý vrchol v z množiny V pamatuje délku nejkratší cesty, kterou se k němu dá dostat. Označme tuto hodnotu jako $d[v]$. Na začátku mají všechny vrcholy v hodnotu $d[v]=\infty$, kromě počátečního vrcholu s, který má $d[s]=0$. Nekonečno symbolizuje, že neznáme cestu k vrcholu.

Dále si algoritmus udržuje množiny Z a N, kde Z obsahuje už navštívené vrcholy a N dosud nenavštívené. Algoritmus pracuje v cyklu tak dlouho, dokud N není prázdná nebo je navštíven koncový bod nejkratší možnou cestou. V každém průchodu cyklu se přidá jeden vrchol $vmin$ z N do Z, a to takový, který má nejmenší hodnotu $d[v]$ ze všech vrcholů v z N.

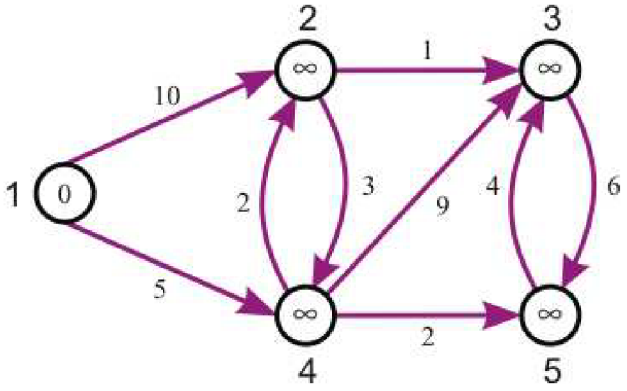
Pro každý vrchol *u*, do kterého vede hrana (označme její délku jako *l(vmin,u)*) z *vmin*, se provede následující operace.

Pokud (*d[vmin]* + *l(vmin,u)*) < *d[u]*, pak do *d[u]* přiřaď hodnotu *d[vmin]* + *l(vmin,u)*, jinak neprováděj nic.

Až algoritmus skončí, potom pro každý vrchol *v* z *V* je délka jeho nejkratší cesty od počátečního vrcholu uložena v *d[v]*.

Příklad

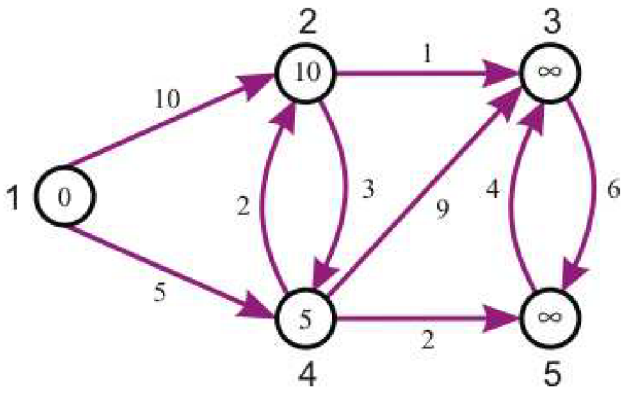
Na začátku algoritmu všem vrcholům, kromě počátečního, nastavíme hodnotu nekonečno symbolizující fakt, že neznáme cestu k vrcholu. Počátečnímu vrcholu nastavíme jako výchozí hodnotu nulu (viz obrázek číslo 18). Koncovým vrcholem je uzel číslo pět.



Obrázek 18: První (inicializační) krok Dijkstrova algoritmu

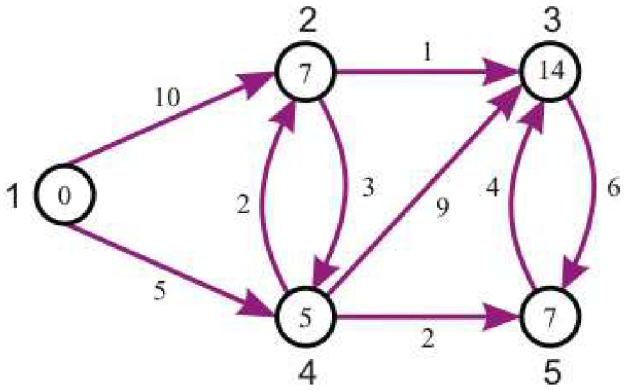
Je patrné, že graf nemá záporně ohodnocené hrany, tedy nejkratší možnou nalezenou cestou je cesta o délce nula, proto je tedy i přiřazena počátečnímu vrcholu.

V každém následujícím kroku algoritmu (obrázky číslo 19 - 24) bude vybrán uzel, který nemusí být celkově optimální, ale prozatím bude nejblíže počátečnímu vrcholu. Poté přepočítáme hodnoty každému uzlu, který je s nově výchozím propojen hranou.



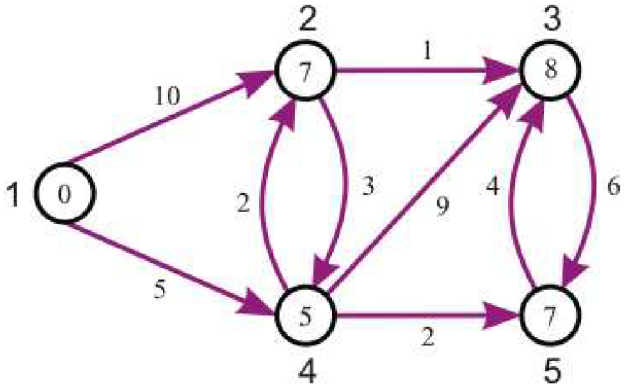
Obrázek 19: Druhý krok Dijkstrova algoritmu

V tomto okamžiku jediný navštívený (optimální) uzel je uzel počáteční. Jako s následujícím (nově výchozím) uzlem budeme počítat s uzlem číslo čtyři (díky jeho menší vzdálenosti od počátku), ze kterého se přepočítají další následovníci.



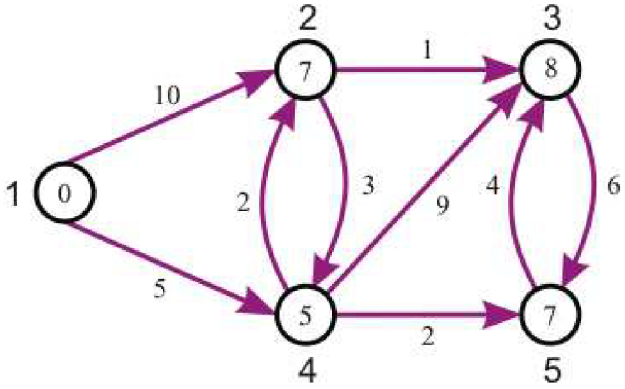
Obrázek 20: Třetí krok Dijkstrova algoritmu

Nyní docházíme k důležitému zjištění, a to takovému, že z počátečního uzlu do uzlu číslo dva jsme nalezli kratší cestu než původní přímou. Přesuneme se tedy do uzlu číslo dva (nově výchozí) a přepočítáme hodnotu v třetím uzlu. Následuje návrat zpět do uzlu číslo čtyři z důvodu nižší metriky než je u uzlu dva.



Obrázek 21: Čtvrtý krok Dijkstrova algoritmu

Když se podíváme na uzly číslo pět a tři, zjistíme jednoduchým porovnáním jejich délkových hodnot, že jsme již nejkratší možnou cestu našli a algoritmus ukončíme.



Obrázek 22: Výsledek Dijkstrova algoritmu

Poznámka

Při programování a použití řídkých grafů (grafy s počtem hran mnohem menším než $|V|^2$) může být Dijkstrův algoritmus implementován mnohem efektivněji tím, že se graf zapisuje pomocí seznamu ukazatelů na sousední uzly.

Modifikace Dijkstrova algoritmu [13]

Dijkstrův algoritmus, ačkoliv je poměrně rychlým algoritmem, existuje v několika modifikacích, které hledání nejkratší cesty urychlují a zefektivňují.

[13] Zpracováno dle: ČERNÝ, J., *Základní grafové algoritmy*, s. 101, (8)

Mezi základní modifikace patří:

- Obousměrný Dijkstrův algoritmus
- Dijkstrův algoritmus s heuristikou
- Silniční hierarchie (zefektivnění hledání)

Obousměrný Dijkstrův algoritmus

Jedná se o modifikaci algoritmu, kde během průběhu algoritmu se střídají dvě jednosměrná vyhledávání. Jedno je vedeno ze startovní pozice *s* a druhé pak z pozice cílové *d*. Obě vyhledávání vytváří strom nejkratších cest ze startovního bodu *S*, respektive cílového bodu *D*. Algoritmus je ukončen ve chvíli, kdy je průnik stromů *D* a *S* neprázdný. Nalezená cesta je pak kombinace cest $(s - \{D \cap S\}) \cup (\{D \cap S\} - d)$.
Tato modifikace se využívá ke směrování v městských dopravních sítích.

Dijkstrův algoritmus s heuristikou

Dijkstrův algoritmus hledá nejkratší cestu rovnoměrně na všechny strany – prochází tedy všechny vrcholy v pořadí určeném vzdáleností od počátku. Když budeme hledat cestu například z Prahy do Brna tak při běhu algoritmu spočítáme nejkratší cesty také do Jihlavy, ale i Karlových Varů. Přitom Karlovy Vary leží na opačné straně od Prahy než Brno.
Heuristika u tohoto algoritmu spočívá v tom, že místo vzdálenosti *d[v]* budeme počítat s $d'[v] \coloneqq d[v] +$ vzdálenost z *v* do cíle vzdušnou čarou. To nám zaručí, že cestu do Jihlavy spočítáme dříve než do Karlových Varů díky tomu, že vrcholy poblíž cíle budou zvýhodněny.

Silniční hierarchie

Představme si, že graf *G* odpovídá silniční síti celé Evropy (přibližně milión vrcholů). Pokud bychom chtěli nalézt nejkratší cestu z Helsinek do Lisabonu tak využijeme Dijkstrův algoritmus.
Kdybychom chtěli realizovat nějaký plánovač tras, kde existuje možnost změny cesty a tedy i velké množství výpočtů, které by měly probíhat v reálném čase, tak samostatný Dijkstrův algoritmus se prováděl několik desítek minut.

Když tedy budeme hledat nejkratší cestu z Tampere do Lisabonu, tak ji nebudeme hledat v celé silniční síti Evropy (jedná se o příliš velký graf), ale ve třech fázích a v mnohem menších grafech. V první fázi nalezneme nejkratší cesty z Tampere na hraniční přechody Finska. Ve druhé fázi nalezneme nejkratší cesty z hraničního přechodu Finska do Portugalska. Cesty navíc hledáme ve zjednodušeném grafu evropské silniční sítě, který jako vrcholy obsahuje pouze hraniční přechody. Ve třetí fázi nalezneme nejkratší cesty z hraničních přechodů Portugalska do Lisabonu.
Z výsledku všech tří fází poskládáme acyklický graf, který obsahuje pouze Tampere, Lisabon a hraniční přechody Finska a Portugalska. Takovýto graf má už jen pár vrcholů a i díky tomu, že je acyklický můžeme v něm najít cestu v lineárním čase.
Hierarchie může mít i více úrovní. Celou Evropu můžeme rozdělit na oblasti podle států, státy na oblasti podle krajů. Oblasti vůbec nemusí odpovídat právnímu rozdělení. Jde jen o to, aby každá oblast měla málo vstupních míst. Čím méně jich bude graf mít, tím bude jednodušší a algoritmus rychlejší.
Pro řešení problému jak nalézt ty správné silnice se poté využívají různě modifikované heuristiky.

Využití

Tento algoritmus je často používán u síťových prvků jako součást směrovacích protokolů typu OSPF (Open Shortest Path First), což je adaptivní hierarchický distribuovaný routovací protokol, provádějící změny v routovacích tabulkách na základě změny stavu v síti a jedná se o nejpoužívanější routovací protokol uvnitř autonomních systémů.
Taktéž modifikace tohoto algoritmu se používají například pro hledání spojení vlaků nebo tras v navigačních systémech. Modifikace spočívají v tom, zda hledáme nejkratší nebo nejrychlejší trasu.

6. 2 Floyd - Warshallův algoritmus

Floyd-Warshallův algoritmus hledá nejkratší vzdálenosti mezi všemi páry uzlů. Docílí toho tak, že porovnává všechny možné cesty mezi každými dvěma uzly a algoritmus postupně zlepšuje odhad nejlepší cesty až do okamžiku, kdy už ví, že nalezená cesta je opravdu ta nejkratší.

Algoritmus je zejména vhodný pro husté grafy, kde je rychlejší než Dijkstrův, opakovaný pro všechny vrcholy. Algoritmus je založen na práci s maticí sousednosti.

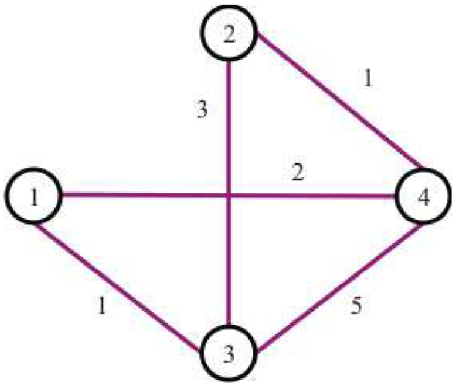
Popis algoritmu

Máme graf G o vrcholech 1 až N. Čtvercová matice sousednosti D[i][j] o rozměrech N x N nese ohodnocení hran grafu. Algoritmus se skládá z N fází. Na konci každé k-té fáze bude algoritmus porovnávat délku cesty v D[i][j], která může obsahovat vrcholy 1 až k – 1, s délkou cesty, která může navíc obsahovat vrchol k. Porovná se tedy hodnota D[i][j] s hodnotou D[i][k] + D[k][j] a menší z nich bude uložena do prvku D[i][j]. Po absolvování všech N fází je algoritmus u konce a můžeme z matice sousednosti zjistit délky nejkratších cest mezi všemi dvojicemi vrcholů v grafu.

S rekonstrukcí cest algoritmus nepočítá, ale o tuto funkci jej lze rozšířit.

Příklad

Stejně jako při provádění algoritmu počítačem, tak i při početní metodě je vhodnější příklad propočítávat za pomocí matic sousednosti. Jelikož je příklad velmi jednoduchý, ale obsahuje velký počet kroků, tak některé budou záměrně vynechány.



Obrázek 23: Graf pro demonstraci Floyd - Warshallova algoritmu

Z příkladového grafu (viz obrázek číslo 23) si sestrojíme matici sousednosti, ve které postupně budeme porovnávat v jednotlivých krocích hodnoty vzdáleností.

	1	2	3	4
1	0	N	1	2
2	N	0	3	1
3	1	3	0	5
4	2	1	5	0

→

	1	2	3	4
1	0	N	1	2
2	N	0	3	1
3	1	3	0	5
4	2	1	5	0

→→

(k,i,j)=(1,1,2)
0 + N < N ?
NENÍ

(k,i,j)=(1,1,3)
0 + 1 < 1 ?
NENÍ

	1	2	3	4
1	0	N	1	2
2	N	0	3	1
3	1	3	0	5
4	2	1	5	0

→

	1	2	3	4
1	0	N	1	2
2	N	0	3	1
3	1	3	0	5
4	2	1	5	0

→→

(k,i,j)=(2,2,2)
N + N < 0 ?
NENÍ

(k,i,j)=(1,2,3)
N + 1 < 3 ?
NENÍ

	1	2	3	4
1	0	N	1	2
2	N	0	3	1
3	1	3	0	5
4	2	1	5	0

→

	1	2	3	4
1	0	N	1	2
2	N	0	3	1
3	1	3	0	5
4	2	1	5	0

(k,i,j)=(1,3,3)
1 + 1 < 0 ?
NENÍ

(k,i,j)=(1,3,4)
1 + 2 < 5 ?
ANO JE

Jako výslednou matici získáme matici, ve které budou uloženy vzdálenosti mezi vzájemně propojenými uzly (viz obrázek číslo 24).

	1	2	3	4
1	0	N	1	2
2	N	0	3	1
3	1	3	0	3
4	2	1	3	0

Obrázek 24: Výsledná matice Floyd - Warshallova algoritmu

Poznámka

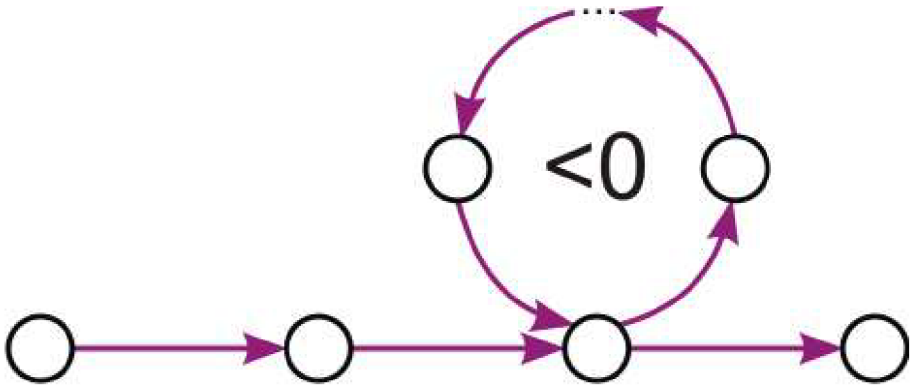
Velkou nevýhodou tohoto algoritmu je jeho výpočetní složitost s rostoucím počtem vrcholů, mezi kterými nejkratší vzdálenost počítáme, jelikož počet kroků algoritmu roste podobně rychle jako graf funkce $y= x^3$.

Využití

Tento algoritmus se využívá více v matematických disciplínách, než v reálných případech. Lze jej využít například pro inverze regulárních matic (Gauss-Jordanova eliminace) nebo testování zda je neorientovaný graf bipartitní.

6. 3 Bellman – Fordův algoritmus

Bellman – Fordův algoritmus hledá nejkratší cestu v orientovaném grafu s libovolným ohodnocením hran (tedy i se záporným ohodnocením). Pro správný průběh algoritmu se nesmí v případě záporně ohodnocených hran objevit v grafu záporný cyklus, který by automaticky probíhal do nekonečna (viz obrázek číslo 25).



Obrázek 25: Příklad záporného cyklu v grafu

Popis algoritmu

Bellman-Fordův algoritmus také, stejně jako Dijkstrův algoritmus, využívá metodu relaxace hran, která zajišťuje aktuálně nastavenou hodnotu nejkratší vzdálenosti od počátečního uzlu.

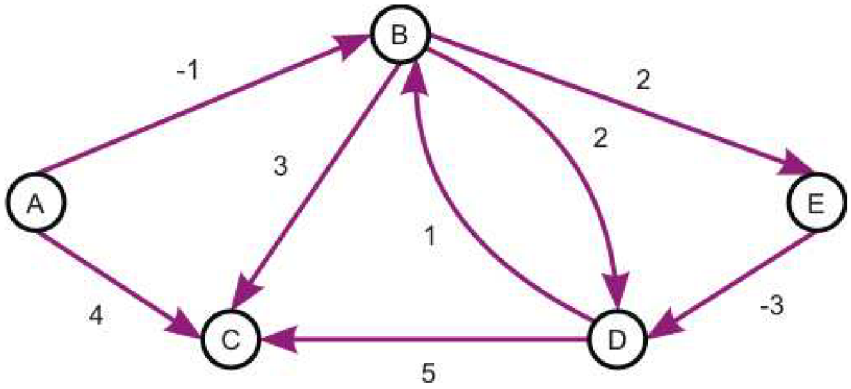
Jestliže je algoritmem zjištěno, že hodnota v uzlu je vyšší než hodnota z nynějšího uzlu plus ohodnocení hrany z nynějšího uzlu do uzlu, v kterém bychom chtěli změnit jeho hodnotu, tak tuto hodnotu změníme, respektive snížíme.

Hlavní rozdíl oproti Dijkstrovu algoritmu spočívá v průchodu grafu. U Dijkstrova algoritmu, jestliže projdeme všechny následníky jednoho uzlu tak tento uzel uzavře a poté ho už neupravuje. V Bellman-Fordovu algoritmu se toto neděje, jelikož všechny uzly v grafu se několikrát projíždí a postupně se upravují hodnoty vzdáleností nejkratších cest.

Algoritmus je tedy oproti Dijkstrovému algoritmu jednodušší, ale pomalejší.

Příklad

Na následujícím příkladě bude demonstrován chod Bellman-Fordova algoritmu (viz obrázek 26). Pro vhodnější znázornění jednotlivých kroků, se využije tabulka, ve které budou uváděny a přepočítávány hodnoty vzdáleností v jednotlivých iteracích.



Obrázek 26: Graf pro demonstraci Bellman - Fordova algoritmu

Na začátku algoritmu, stejně jako u Dijkstrova algoritmu, všem vrcholům, kromě počátečního, nastavíme hodnotu nekonečno symbolizující fakt, že neznáme cestu k vrcholu. Počátečnímu vrcholu nastavíme jako výchozí hodnotu nulu.

Poté začneme prošetřovat jednotlivé hrany, kde zjistíme cesty do uzlu B a následně do uzlu C. Ke konci první iterace zjišťujeme, že do vrcholu C vede kratší cesta než přímá. Poslední řádek tabulky tedy ještě zaktualizujeme (viz tabulka číslo 1).

iterace	A	B	C	D	E
1.	0	∞	∞	∞	∞
	0	-1	∞	∞	∞
	0	-1	4	∞	∞
	0	-1	2	∞	∞

Tabulka 1: První iterace Bellman - Fordova algoritmu

V druhé iteraci algoritmu se posouváme směrem k dalším uzlům a nacházíme vzdálenost do uzlu E. Při šetření uzlu B ještě nacházíme nejkratší (prozatím) cestu do uzlu D. V posledním kroku druhé iterace dochází ke zjištění, že do uzlu D existuje kratší cesta než přímá a to přes uzel E. Hodnoty opět aktualizujeme.

iterace	A	B	C	D	E
2.	0	-1	2	∞	1
	0	-1	2	1	1
	0	-1	2	-2	1

Tabulka 2: Druhá iterace Bellman - Fordova algoritmu

Poslední řádek tabulky číslo 2 u druhé iterace, je zároveň i řešením nejkratších cest z uzlu A do všech ostatních. Samotný neoptimalizovaný algoritmus by však druhou iteraci nekončil. Celkově se provedou iterace čtyři (n-1 iterací, kde n je počet uzlů), avšak v těch dvou následujících se už tabulka nezaktualizuje. Tento počet iterací je navržen kvůli zajištění správnosti algoritmu.

Poznámka

Pokud bychom chtěli mít plně funkční a optimalizovaný Bellman – Fordův algoritmus, tak je nutné do něj zahrnout zjišťování aktualizací metriky, kde po splnění podmínek (neprovede se žádná další aktualizace některé hrany) další iterace se nebudou muset

provádět a algoritmus může být ukončen dříve. Vhodné je také zavést testování výskytu záporných cyklů, aby algoritmus neběžel do nekonečna.

Využití

Jako praktický příklad využití tohoto algoritmu můžeme uvést implementaci ve směrovacím protokolu RIP (Routing Information Protocol), který slouží ke komunikaci routerů propojených v síti. Bellman-Fordův algoritmus je využit k nalezení nejkratších cest mezi prvky sítě a přibližně do 30ti sekund po změně topologie sítě je schopen aktualizovat směrovací tabulku routerů. Velikost sítí je však omezena. Největší možná vzdálenost mezi routery je 15 skoků.

Přestože patří tento protokol mezi nejstarší doposud používané směrovací protokoly v sítích, má stále své uplatnění v menších sítích, a to především pro svou nenáročnou konfiguraci a jednoduchost. Existují různé verze tohoto protokolu a nejnovější RIPng byl definován pro použití v IPv6.

6. 4 Johnsonův algoritmus

Základní myšlenkou Johnsonova algoritmu je n-násobné použití Dijkstrova algoritmu, kde n je počet vrcholů grafu. Algoritmus nám slouží k hledání nejkratších cest mezi všemi páry vrcholů v řídkých orientovaných grafech. Připouští záporné hrany, avšak v grafu nesmí existovat žádný záporný cyklus.

Samotný algoritmus využívá jak Dijkstrova algoritmu tak i Bellman-Fordova algoritmu. Bellman-Fordův algoritmus je využíván k výpočtu transformace vstupního grafu, která odstraní záporné váhy hran, což splní podmínky pro běh Dijkstrova algoritmu z každého uzlu.

Algoritmus je pojmenován po Donaldu Johnsonovi, který jej zveřejnil v roce 1977.

Popis algoritmu

Johnsonův algoritmus se skládá z následujících kroků:

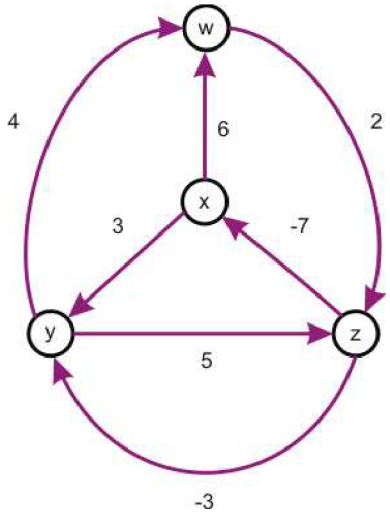
1. Prvně je přidán nový uzel q s nulovou váhou do všech uzlů grafu
2. Bellman-Fordův algoritmus je využit tak, že z nového vrcholu q vyhledává pro každý vrchol v nejkratší cestu, s hodností $h(v)$, z q do v. Pokud v tomto kroku bude zjištěn negativní cyklus, algoritmus se ukončí.

3. Hrany z původního grafu se přepočítají za pomocí hodnot spočtených podle Bellman-Fordova algoritmu: hraně z *u* do *v*, která má délku *w(u,v)* je přiřazena nová délka *w(u, v) + h(u)-h(v)*.
4. Pro každý uzel se spustí Dijkstrův algoritmus pro nalezení nekratší cesty v grafu s přetransformovanými hranovými hodnotami.

Nalezené cesty v transformovaném grafu jsou ve výsledku totožné s cestami jako v grafu původním. Avšak vzhledem tomu, jakým způsobem jsou přepočítány jednotlivé hrany, které jsou navíc, pro chod Dijkstrova algoritmu, nezáporné, tak délky cest neodpovídají skutečným hodnotám. Vzdálenosti mohou být ale přepočítány zpětnou transformací z hodnot Dijkstrova algoritmu.

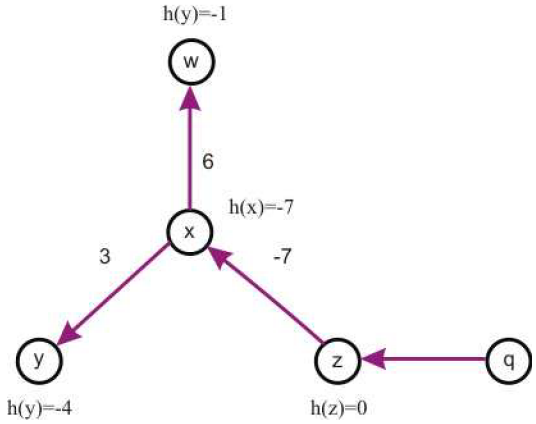
Příklad

V následujícím příkladě jsou popsány první tři etapy Johnsonova algoritmu. Z obrázku číslo 27 je patrné, že v grafu není žádný záporný cyklus.



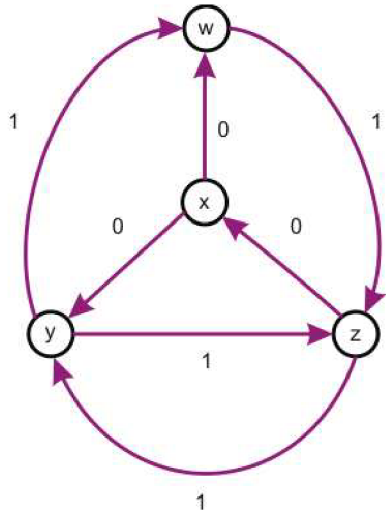
Obrázek 27: Původní graf

Strom znázorňující nejkratší cesty do jednotlivých uzlů je počítán Bellman-Fordovým algoritmem, kde *q* je počáteční uzel a hodnoty cest do ostatních uzlů jsou počítány jako nejkratší cesta z uzlu *q* do daného uzlu. Za zmínku stojí fakt, že všechny hodnoty jsou negativní, a jelikož má hrana z *q* nulovou délku, tak velikost nejkratší cesty nemůže být větší než hodnota té hrany (viz obrázek číslo 28).



Obrázek 28: Strom nalezený Bellman-Fordovým algoritmem

Na obrázku číslo 29 je již transformovaný graf, kde každá hranová hodnota je přepočtena podle výše zmíněného vzorce. Žádná hrana tak už není ohodnocena záporně, ale i přes tento fakt je nejkratší cesta mezi dvěma libovolnými uzly v takto přetransformovaném grafu shodná s cestou v grafu původním. Algoritmus bude ukončen hned po provedení Dijkstrova algoritmu ze všech uzlů.



Obrázek 29: Transformovaný graf bez záporně ohodnocených hran

Využití

Tento algoritmus, se stejně jako Floyd-Warshallův algoritmus, v praktických případech moc nevyužívá a zůstává součástí matematických disciplín, kde díky své rychlosti může právě Floyd-Warshallův algoritmus nahrazovat.

7. Asymptotická složitost algoritmů[14]

Při řešení algoritmických úloh je vhodné mít nástroj, kterým dokážeme porovnat efektivitu a rychlost vykonávání jednotlivých algoritmů. Pro tento účel byly zavedeny pojmy asymptotická složitost a operační náročnost algoritmu.

Asymptotická složitost je způsob klasifikace počítačových algoritmů. Určuje operační náročnost algoritmu tak, že zjišťuje, jakým způsobem se bude chování algoritmu měnit v závislosti na změně počtu vstupních dat.

Při vyjadřování asymptotické složitosti zanedbáváme operace nezávislé na datech, dílčí jednodušší části algoritmu a multiplikační konstanty.

Příklady časové složitosti

- O(1) - indexování prvku v poli (konstantní složitost)
- $O(\log_2 N)$ - vyhledání prvku v seřazeném poli metodou půlení intervalu (logaritmická složitost)
- O(N) - vyhledání prvku v neseřazeném poli lineárním vyhledáváním (lineární složitost)
- O(N log N) - seřazení pole reálných čísel podle velikosti např. algoritmem Mergesort (lineárně logaritmická složitost)
- $O(N^2)$ - diskrétní Fourierova transformace (kvadratická složitost)
- $O(2^N)$ - řešení problému obchodního cestujícího za pomoci dynamického programování (exponencionální složitost)
- O(n!) - přesné řešení problému obchodního cestujícího pomocí hrubé síly (faktoriálová složitost)

Poznámka

Tyto příklady jsou řazeny od nejrychlejších k nejpomalejším.

[14] Zpracováno dle: *Asymptotická složitost – Wikipedie, otevřená encyklopedie*, (7)

Časová složitost algoritmů pro hledání nejkratších cest

- Dijkstrův algoritmus

  Při nejjednodušší implementaci Dijkstrova algoritmu je asymptotická časová složitost $O(|V|^2+|E|)$, kde $|V|$ je počet vrcholů a $|E|$ počet hran.

  Při použití binární haldy (stromová datová struktura) bude složitost $O((|E|+|V|)log|V|)$.

  Fibonacciho halda čas dokonce zlepší na $O(|E|+|V| \log |V|)$.
- Floyd-Warshallův algoritmus

  Tento algoritmus, jelikož využívá kvadratické množství paměti vůči počtu vrcholů graf, má asymptotickou složitost rovnu $O(V^3)$.
- Bellman-Fordův algoritmus

  Složitost tohoto algoritmu je rovna $O(V*E)$.
- Johnsonův algoritmus

  Složitost tohoto algoritmu je rovna $O(V^2 log\ V + VE)$.

V	E	Dijkstrův algoritmus	Dijkstrův algoritmus (binární halda)	Dijkstrův algoritmus (Fibonacciho halda)	Floyd-Warshallův algoritmus	Bellman-Fordův algoritmus	Johnsonův algoritmus
1	0	1	0	0	1	0	0
1	1	2	0	1	1	1	1
2	2	6	1,20	2,60	8	4	5,20
4	4	20	4,82	6,41	64	16	25,63
4	6	22	6,02	8,41	64	24	33,63
10	10	110	20	20	1000	100	200
10	15	115	25	25	1000	150	250
100	100	10100	400	300	1000000	10000	30000
100	120	10120	440	320	1000000	12000	32000

Tabulka 3: Časová složitost vybraných algoritmů

Tabulka číslo 3 názorně demonstruje rychlosti algoritmů. Avšak přímo srovnávat je nemůžeme, jelikož hodnoty, které algoritmy vrátí, nejsou stejného charakteru.

8. Zhodnocení a závěr

Teorie grafů je moderní a rychle se rozvíjející oblast matematiky. Jejich poznatků a aplikací se užívá v mnoha dalších disciplínách od informatiky až po řešení různých hříček a hlavolamů.

Práce je zaměřena na hledání nejkratších cest v grafech, jelikož s těmito optimalizačními úlohami se člověk setkává každý den a ani si toho nemusí být vědom.

Na úlohy pro hledání nejkratších cest existuje několik druhů algoritmů, ale každý z nich pracuje jiným způsobem a od toho se odvíjejí i výsledky nalezených cest. Nutné je taky podotknout, že následníci Edsgera Dijkstry se jeho algoritmem nechávali inspirovat, výjimku tvoří jen Floyd-Warshallův algoritmus. Všechny čtyři představené algoritmy mají významné uplatnění v matematických disciplínách a informatice.

Dijkstrův algoritmus je poměrně rychlý algoritmus, který při jeho modifikacích pomocí různých stromových datových struktur nebo heuristik dosahuje velmi nízkých časů pro vyhodnocení výsledků. Výsledkem tohoto algoritmu je nalezení nejkratší cesty z počátečního bodu do bodu koncového, která je složena z menších, avšak také nejkratších cest do jednotlivých uzlů, které jsou součástí výsledné cesty.

Floyd-Warshallův algoritmus je ze všech čtyř uvedených algoritmů algoritmem nejjednodušším, avšak časově nejnáročnějším. Počítá vzdálenosti mezi všemi dvojicemi uzlů, avšak u výsledných cest známe pouze vzdálenost. Pokud bychom chtěli znát i konkrétní cesty z jednotlivých uzlů, tak by bylo nutno algoritmus upravit.

Bellman-Fordův algoritmus dokáže spočítat nejkratší cesty i v záporně ohodnocených grafech, ale za podmínky, že v grafu není obsažen záporný cyklus.

Johnsonův algoritmus je ze všech čtyř algoritmů nejsložitějším. Dokáže pracovat s negativně ohodnocenými hranami, a i přes skutečnost, že je složen jak z Bellman-Fordova algoritmu (pro detekci záporně ohodnocených hran) a Dijkstrova algoritmu (pro počítání nejkratších cest z jednotlivých uzlů) je rychlejší než Floyd-Warshallův algoritmus řešící tentýž optimalizační problém.

Pokud jako hlavní kritérium pro srovnání algoritmů bereme jednoduchost zápisu, je na tom Floyd-Warshallův algoritmus nejlépe. Je jednodušší na výpočet i programátorský

zápis. Dijkstrův i Bellman-Fordův algoritmus díky své podobnosti jsou na stejné úrovni složitosti. Johnsonův algoritmus je ze všech čtyř nejsložitější hlavně díky tomu, že je spojen z dvou jiných algoritmů.

Co se týče asymptotické časové složitosti při hledání nejkratších cest mezi všemi vrcholy tak je nejrychlejší Johnsonův algoritmus. Dokonce i Floyd-Warshallův algoritmus je rychlejší než pro všechny vrcholy opakovaný Dijkstrův algoritmus bez modifikací.

Pro hledání cest mezi dvěma vrcholy je nejvhodnější algoritmus Dijkstrův.

9. Seznam informačních zdrojů

Odborná literatura

1) ČADA, Roman, KAISER, Tomáš a RYJÁČEK, Zdeněk. *Diskrétní matematika*, 2004, 232 s. ISBN 80-7082-939-7
2) DEMEL, Jiří. *Grafy a jejich aplikace*, 2002, 180 s. ISBN 80-200-0990-6
3) MATOUŠEK, Jiří a NEŠETŘIL, Jaroslav. *Kapitoly z diskrétní matematiky.* 2002. 374 s. ISBN 80-246-0084-6.
4) MEZNÍK, Ivan. *Diskrétní matematika,* 2004, 132 s. ISBN 80-214-2754-X
5) VEČERKA, Arnošt. *Grafy a grafové algoritmy,* 2007, 112 s. Katedra informatiky univerzity Palackého v Olomouci

Elektronické zdroje

6) *All Sources Shortest Path: The Floyd-Warshall Algorithm*[online] [cit. 2009–04-10]. Dostupné z: <http://compprog.wordpress.com/2007/11/15/all-sources-shortest-path-the-floyd-warshall-algorithm/>
7) *Asymptotická složitost* [online] [cit. 2009–04-23]. Dostupné z: <http://cs.wikipedia.org/wiki/Asymptotická_složitost>
8) ČERNÝ, Jakub. *Základní grafové algoritmy*, 2008, 175 s. [online][cit. 2009–05-18]. Dostupné z: <http://kam.mff.cuni.cz/~kuba/ka/>
9) DEMAINE, Erik. *Shortest Paths II:Bellman-Ford, Linear Programming, Difference Constraints* [online] [cit. 2009–04-15]. Dostupné z: <http://videolectures.net/mit6046jf05_demaine_lec18/>
10) DEMAINE, Erik. *Shortest Paths III:All-pairs Shortest Paths, Matrix Multiplication, Floyd-Warshall, Johnson* [online] [cit. 2009–05-18]. Dostupné z: <http://videolectures.net/mit6046jf05_demaine_lec19/>
11) *Grafové algoritmy* [online] [cit. 2009–04-07]. Dostupné z: <http://fav.q-e-e.net/PT/prednasky/grafove_algoritmy.ppt>
12) *Graph theory – History* [online] [cit. 2009–03-09]. Dostupné z: <http://en.wikipedia.org/wiki/Graph_theory#History>

13) *Kaliningrad* [online] [cit. 2009-03-09]. Dostupné z: <http://maps.google.com/maps?f=q&source=s_q&hl=cs&geocode=&q=kaliningrad&sll=37.0625,95.677068&sspn=35.273162,78.75&ie=UTF8&ll=54.704813,20.509586&spn=0.012572,0.05476&t=h&z=15&iwloc=A>

14) *Konigsberg bridges* [online] [cit. 2009-03-09]. Dostupné z: < http://en.wikipedia.org/w/index.php?title=File:Konigsberg_bridges.png>

15) *One Source Shortes Path: Dijkstra's Algorithm* [online] [cit. 2009–04-07]. Dostupné z: <http://compprog.wordpress.com/2007/12/01/one-source-shortest-path-dijkstras-algorithm/>

16) *Teorie grafů* [online] [cit. 2009-03-10]. Dostupné z: <http://cs.wikipedia.org/wiki/Kategorie:Teorie_grafů>

17) *The Traveling Salesman Problem* [online] [cit. 2009-03-10]. Dostupné z: < http://www.tsp.gatech.edu/index.html >

10. Seznam obrázků a tabulek

11. Rejstřík