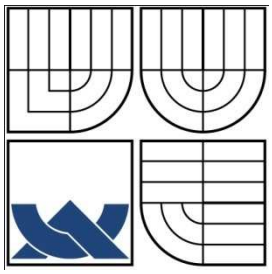
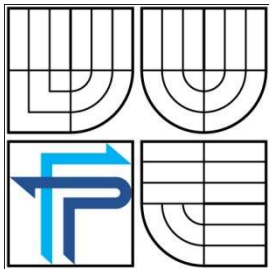


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA PODNIKATELSKÁ
ÚSTAV INFORMATIKY

FACULTY OF BUSINESS AND MANAGEMENT
INSTITUT OF INFORMATICS

GRAFY A GRAFOVÉ ALGORITMY
GRAPHS AND GRAPH ALGORITHMS

BAKALÁŘSKÁ PRÁCE
BACHELOR´S THESIS

AUTOR PRÁCE — ONDŘEJ PAVLÁSEK
AUTHOR

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

Brno 2010

Vysoké učení technické v Brně
Fakulta podnikatelská

Akademický rok: 2009/2010
Ústav informatiky

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Pavlásek Ondřej

Manažerská informatika (6209R021)

Ředitel ústavu Vám v souladu se zákonem č.111/1998 o vysokých školách, Studijním a zkušebním řádem VUT v Brně a Směrnicí děkana pro realizaci bakalářských a magisterských studijních programů zadává bakalářskou práci s názvem:

Grafy, grafové algoritmy a jejich využití

v anglickém jazyce:

Graphs, Graph Algorithms and their Application

Pokyny pro vypracování:

Úvod
Vymezení problému a cíle práce
Teoretická východiska práce
Analýza problému a současné situace
Vlastní návrhy řešení, přínos návrhů řešení
Závěr
Seznam použité literatury
Přílohy

Seznam odborné literatury:

DEMEL, Jiří. Grafy a jejich aplikace. 1. vyd. Praha : Academia, 2002. 258 s. ISBN 80-200-0990-6.
MATOUŠEK, Jiří, NEŠETŘIL, Jaroslav. Kapitoly z diskrétní matematiky. 3. upr. vyd. Praha : Karolinum, 2007. 424 s. ISBN 978-80-246-1411-3.
MEZNÍK, Ivan. Diskrétní matematika pro užitou informatiku. 1. vyd. Brno : Cerm, 2009. 104 s. ISBN 978-80-214-3948-1.
ROSEN, Kenneth H. Discrete Mathematics and Its Applications. 4th edition. New York : Mc Graw-Hill, 1999. 832 s. ISBN 0-07-289905-0.

Vedoucí bakalářské práce: Mgr. Martina Bobalová, Ph.D.

Termín odevzdání bakalářské práce je stanoven časovým plánem akademického roku 2009/2010.

L.S.

Ing. Jiří Kříž, Ph.D. Ředitel ústavu	doc. RNDr. Anna Putnová, Ph.D., MBA

V Brně, dne 03.06.2010

Abstrakt

Práce představuje text zaměřený na problematiku teorie grafů, popisuje jednotlivé grafové algoritmy a charakterizuje jejich praktické použití. Shrnuje výhody a nevýhody každého z nich a podává vysvětlení, který je vhodný použít za dané situace.

Abstract

This bachelor thesis represents text focused on graph theory, describes individual graph algorithms and distinguish practical examples of their use. Thesis summarizes advantages and disadvantages each of the algorithm and mention which one is suitable to use in certain situations.

Klíčová slova

Graf, neorientovaný graf, orientovaný graf, DFS, BFS, minimální kostra grafu, Dijkstrův algoritmus, Floyd-Warshallův algoritmus, Bellman-Fordův algoritmus, Johnsonův algoritmus.

Key words

Graph, undirected graph, directed graph, DFS, BFS, Minimum spanning tree, Dijkstra´s algorithm, Floyd-Warshall algorithm, Bellman-Ford algorithm, Johnson´s algorithm

Bibliografická citace práce:

PAVLÁSEK, O. *Grafy, grafové algoritmy a jejich využití* . Brno: Vysoké učení technické v Brně, Fakulta podnikatelská, 2010. 54 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 Brně dne 3. června 2010 …..…..…….

Ondřej Pavlásek

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

Jak již napovídá název bakalářské práce, předmětem zkoumání tohoto textu bude oblast matematiky zabývající se teorií grafů. Na začátků práce jsou vysvětleny základní pojmy z teorie grafů, aby bylo možné později snáze pochopit fungování jednotlivých algoritmů. Tato část práce čerpá převážně z odborné literatury, jejíž seznam je uveden na konci práce. Grafové algoritmy představují důležité nástroje v mnohých odvětvích lidského života, na což většina lidí zapomíná. Prostřednictvím grafových algoritmů se dají řešit například úlohy na výběr optimální cesty mezi body A a B (například mezi dvěma městy), úlohy spojené s nalezením optimálního dopravního spojení, optimální řízení provozu v počítačové síti, či plánování činností projektů tak, aby byl včas ukončený apod.

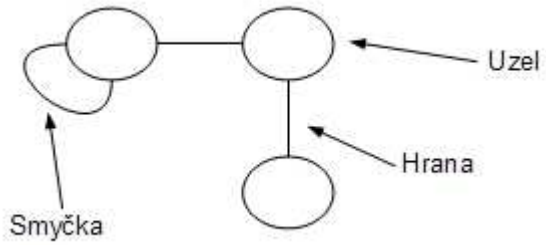
Samotný popis a charakteristika algoritmů je náplní druhé části práce. Postupně probereme algoritmy na procházení grafem, dále algoritmy k nalezení minimální kostry grafu a hlavní náplní práce jsou algoritmy na hledání nejkratší cesty v grafu. Ke každému z grafových algoritmů je na příkladu vysvětleno jeho chování a průběh, na konci každé kapitoly se snažím uvést příklad využití daného algoritmu v praktickém životě. Na závěr práce je podáno vzájemné porovnání z různých úhlů pohledu, na kterém je vysvětleno, jaký druh algoritmu je pro daný případ nejvhodnější použít.

2. Teoretická východiska

2.1. Základní druhy grafů

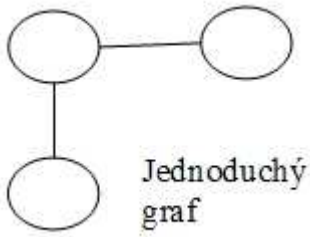
Grafem rozumíme grafické vyjádření vztahů mezi nějakými objekty. Tyto objekty jsou v grafech reprezentovány uzly (někdy též označovány jako vrcholy) a vztahy hranami. Skutečnost, že mezi dvěma objekty je určitý vztah, vyjádříme spojením dvou příslušných uzlů (vrcholů), jež tyto objekty v grafu reprezentují, hranou.

Hrana většinou začíná a končí v různých uzlech, pokud je však koncovým uzlem hrany uzel počáteční, pak takovou hranu nazýváme smyčkou.



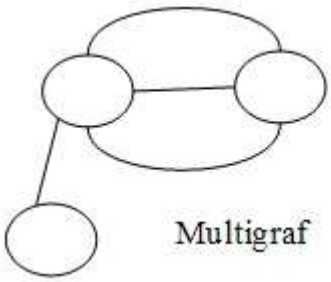
Obr. 2.1.1: Objekty grafu

Graf v němž hrany spojují vždy dva různé uzly a dva různé uzly spojuje nejvýše jedna hrana je graf jednoduchý.



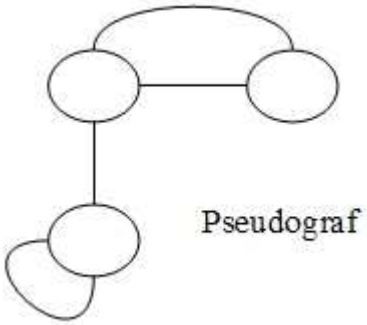
Obr. 2.1.2: Jednoduchý graf

Pro spojení dvou uzlů více než jednou hranou používáme označení násobná hrana používáme a takový graf nazýváme multigrafem. V těchto typech grafů nejsou povoleny smyčky.



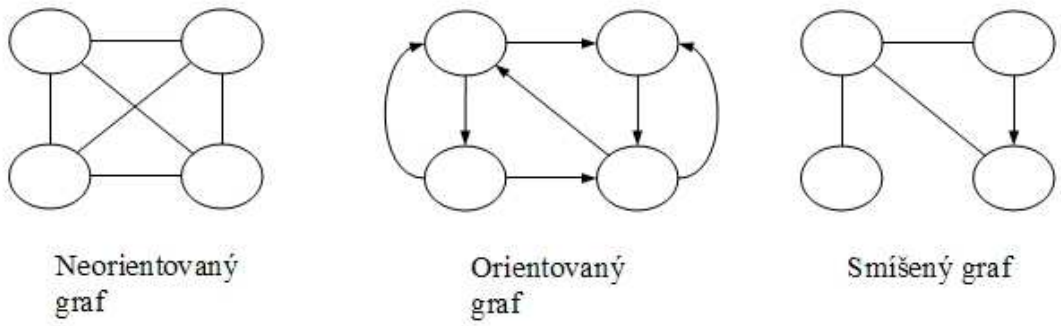
Obr. 2.1.3: Multigraf

Naproti tomu grafy charakteristické násobností hran a výskytem smyček se nazývají pseudografy.



Obr. 2.1.4: Pseudograf

Z hlediska orientace hran dělíme grafy na neorientované, orientované a smíšené. U neorientovaných grafů chápeme hrany jako symetrické spojení dvou uzlů, naproti tomu u orientovaného grafu je označením hrany šipkou na jejím konci zcela jasně určen jednosměrný vztah jednotlivých uzlů (který uzel je počáteční a který koncový). Smíšené grafy obsahují oba typy hran, ale v praxi se nepoužívají proto se jim již dále nebudeme věnovat. **(1, 5, 20)**



Obr. 2.1.5: Typy grafů

2.2. Neorientované grafy

Neorientovaný graf je uspořádaná trojice G = (U, H, ρ) tvořená neprázdnou konečnou množinou U, jejíž prvky nazýváme uzly, konečnou množinou H, jejíž prvky nazýváme neorientovanými hranami a zobrazení ρ, které nazýváme vztahem incidence, a které každé hraně $h \in H$ přiřazuje jedno- nebo dvouprvkovou množinu uzlů. Těmto uzlům říkáme krajní uzly hrany h nebo také, že uzly jsou incidentní s hranou *h*. Naopak hrana h je incidentní s těmito uzly (spojuje tyto uzly). Je- li hrana incidentní pouze s jedním uzlem (tj. jestliže je množina ρ(h) jednoprvková), označujeme hranu *h* (neorientovanou) smyčkou. Pokud několik hran spojuje stejné množiny vrcholů, tj. pro různé hrany h_1, h_2 platilo $\rho(h_1) = \rho(h_2)$, o takových hranách říkáme, že jsou násobné. Uzel, který neinciduje s žádnou hranou, je nazýván izolovaným. **(1, 4, 5, 20)**

Graf G zobrazený na obrázku dole lze definicí zapsat např. takto:

$U = \{u_1, u_2, u_3, u_4\}$

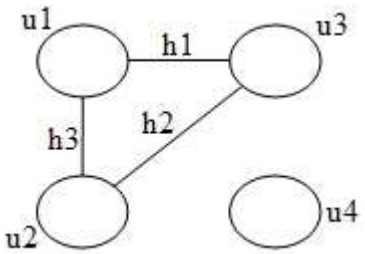
$H = \{h_1, h_2, h_3\}$

$\rho(h_1) = \{u_1, u_3\}$

$\rho(h_2) = \{u_2, u_3\}$

$\rho(h_3) = \{u_1, u_3\}$

kde neuspořádané dvojice uzlů jsme zapsali jako dvouprvkové množiny.



Obr. 2.2.1: Graf G

Někdy se pro definici grafu používá pouze dvojice G = (U, H) kde U je množina uzlů a H množina hran. Hrany jsou v tomto případě definovány jako dvojice koncových uzlů hrany. Graf zobrazený na obrázku nahoře můžeme tímto stylem popsat např. takto:

U = {u1, u2, u3, u4}

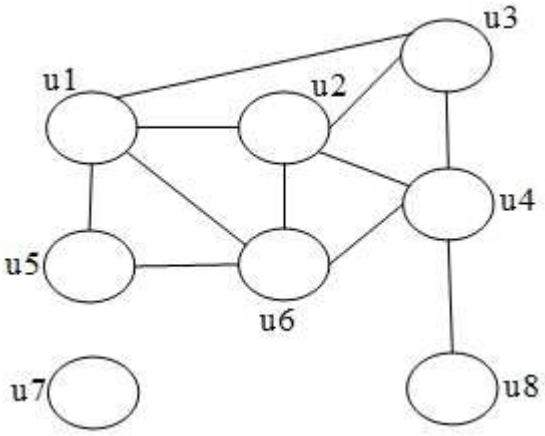
H = {{u1, u3}, {u2, u3}, {u1, u2}}

2.3. Stupeň uzlu

Stupeň uzlu je vyjádřen číslem, které značí počet hran, s nimiž uzel sousedí a značíme ho |x|. Je-li uzel izolovaný pak |x| = 0. **(5)**

Graf na obrázku 2.2.2 má stupně uzlů:

|u1| = 4, |u2| = 4, |u3| = 3, |u4| = 4, |u5| = 2, |u6| = 4, |u7| = 0, |u8| = 1



Obr. 2.3.1: Stupeň uzlu

Vlastnosti stupně uzlu v grafu:

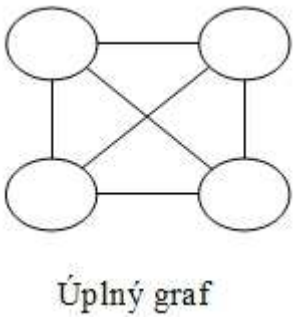
- Součet všech stupňů uzlů v grafu je sudé číslo

  Tato vlastnost vyplývá z toho, že každá hrana inciduje se dvěma uzly, součet stupňů všech uzlů je tedy roven dvojnásobku počtu hran, což je vždy sudé číslo.

- Počet vrcholů lichého stupně v grafu je vždy sudé číslo

  Kdybychom připustili, že počet uzlů lichého stupně je liché číslo, pak spolu se stupni uzlů sudého stupně by součet stupňů všech uzlů byl liché číslo, což je v rozporu s vlastností uvedenou výše.

Uzel, jehož stupeň uzlu je nulový, označujeme jako **diskrétní uzel**. Grafy jehož všechny uzly jsou diskrétní nazýváme diskrétní graf.
Graf jehož každé dva uzly jsou spojeny jednou hranou a neobsahuje smyčky je **graf úplný**. **(20)**



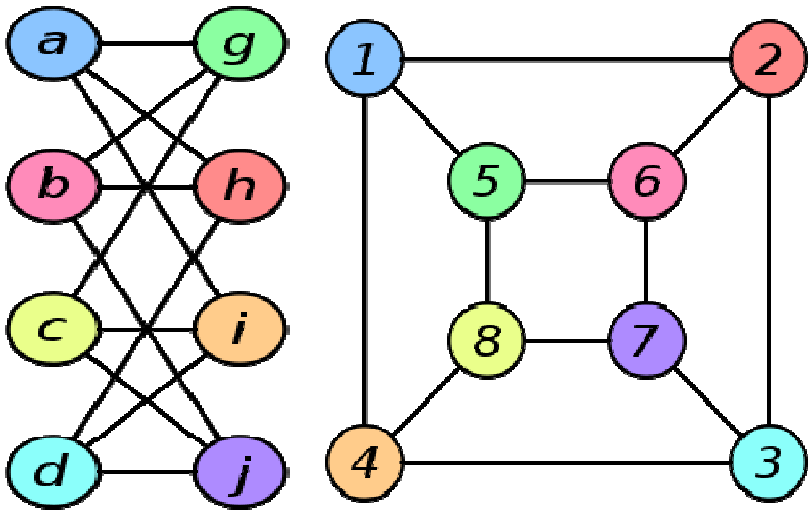
Obr. 2.3.2: Úplný graf

2.4. Izomorfismus

Grafy G, G´ se nazývají vzájemně izomorfní, když existují dvě vzájemně jednoznačná zobrazení f : U → U´ a g : H → H´ taková, že zachovávají vztahy incidence ρ a ρ´ a platí ρ(h) = (x,y) ↔ ρ´(g(h)) = (f(x)), (f(y)). Vztah izomorfismu značíme G ≅ G´. Zjednodušeně můžeme říct, že izomorfismus není nic jiného než přejmenování uzlů v grafu.

Při zjišťování izomorfismu lze často využít tří níže uvedených pozorování:

1. Izomorfní grafy musí mít stejné počty vrcholů i hran
2. Vrcholy stupně *k* lze izomorfismem přiřadit pouze vrchol stejného stupně *k*
3. Dvojici sousedních vrcholů může být izomorfismem přiřazena opět jen dvojice sousedních vrcholů **(1, 20)**



Obr. 2.4.1: Izomorfismus

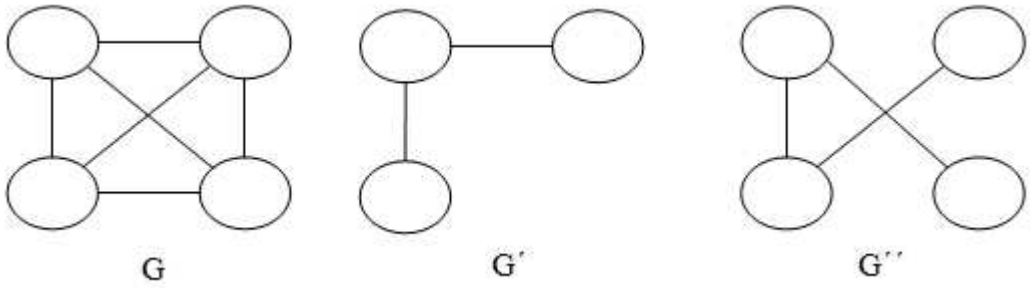
2.5. Podgraf

Graf G´ je **podgrafem** grafu G (značíme G´ ⊆ G) pokud v grafu vynecháme některé (žádné) hrany a některé uzly, včetně hran z nich vedoucích. Pro graf G´(U´, H´, ρ´) platí:

- U´⊆ U (množina uzlů grafu G´ je podmnožinou množiny uzlů grafu G)
- H´⊆ H (množina hran grafu G´ je podmnožinou množiny hran grafu G)
- a pro každou hranu h∈ H platí ρ´(h) = ρ(h)

Pokud zachováme množinu uzlů (U´ = U) podgraf nazveme faktorem grafu G. **(7, 20)**

Na obrázku 2.5.1 je zobrazen původní graf G, graf G´je jeho podgraf a graf G´´ je podgraf a zároveň i faktor grafu G.



Obr. 2.5.1: Podgraf, faktor

2.6. Sled

Posloupnost uzlů a hran u_0, h_1, u_1, h_2, u_2, … , h_k, u_k nazýváme sledem, jestliže každá hrana h_i z této posloupnosti spojuje uzly u_{i-1}, u_i. Vrchol u_0 v obou případech nazýváme počátečním a vrchol u_k koncovým vrcholem sledu. O sledu říkáme, že vede z uzlu u_0 do uzlu u_k, nebo také, že spojuje uzly v_0, v_k.

V orientovaném i neorientovaném sledu na sebe uzly hran navazují, u orientovaného sledu však navíc požadujeme, aby všechny hrany byly orientovány „vpřed ve směru sledu“, u neorientovaného sledu na orientaci nezáleží.

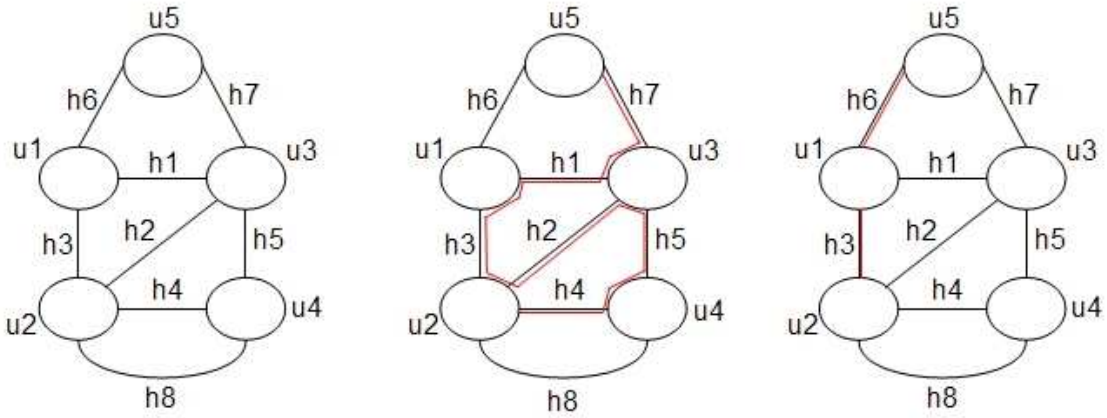
Prakticky si tyto poznatky můžeme vysvětlit na příkladu grafu silniční sítě s jednosměrnými ulicemi, kde smí auta jezdit jen podle orientovaných sledů, zatímco chodci smí chodit i podle sledů neorientovaných.

Triviální sled je sled, který obsahuje jediný uzel a žádnou hranu. Triviální sled lze pokládat za orientovaný i neorientovaný. **(1, 7)**

2.7. Tah, Cesta, Souvislý graf

Sled, v němž se žádná hrana neopakuje, nazýváme **tahem**, zatímco sled, v němž se neopakuje žádný uzel, nazýváme **cestou**. Z předpokladu, že se v cestě neopakují uzly, vyplývá, že se v ní neopakují ani hrany, a tím pádem můžeme konstatovat, že každá cesta je zároveň také tahem a sledem, zatímco tah je vždy sledem, ale nemusí být vždy cestou. Velmi důležitou vlastností grafů je propojitelnost různých dvojic uzlů pomocí sledů a cest. Pro vyjádření této vlastnosti zavedeme pojem souvislosti grafu,
a to tak, že graf mezi jehož každými dvěma uzly existuje cesta nazýváme **souvislý graf**. **(1, 20)**

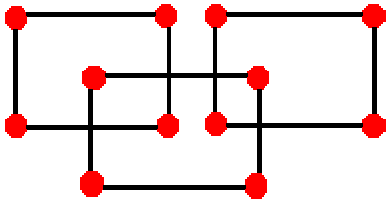
Na obrázku 2.7.1 vidíme graf G a vedle něj graf se zobrazeným tahem z uzlu u_2 do uzlu u_5 (u_2, h_4, u_4, h_5, u_3, h_2, u_2, h_3, u_1, h_1, u_3, h_7) a vedle je vyznačena cesta z uzlu u_2 do uzlu u_5 (u_2, h_3, u_1, h_6, u_5)



Obr. 2.7.1: Tah, cesta

2.8. Komponenta grafu

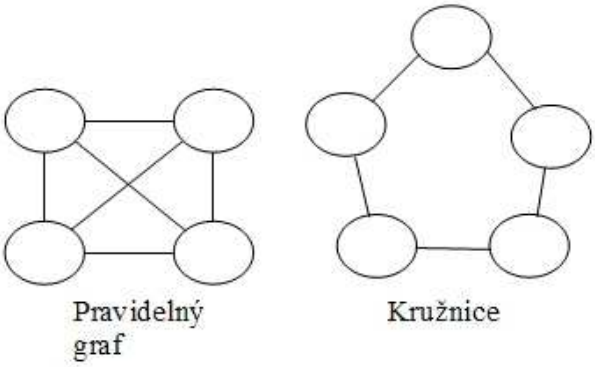
Pokud není graf souvislý, můžeme jej rozdělit na podgrafy, které již souvislé jsou. Takovému maximální podgrafu – tzn. nelze jej zvětšit přidáním dalších hran a uzlů tak, aby byl stále souvislý - říkáme komponenta grafu. Komponenty příslušné různým uzlům si jsou buď rovny, nebo jsou disjunktní a je-li graf souvislý, pak jsou si komponenty všech uzlů rovny a rovnají se grafu. **(5)**



Obr. 2.8.1: Komponenta grafu

2.9. Uzavřené sledy, pravidelný graf, kružnice

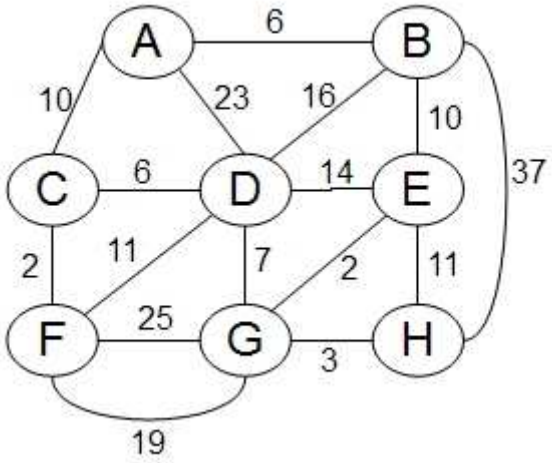
Sled, který má alespoň jednu hranu a jehož počáteční a koncový uzel splývají, nazýváme **uzavřeným sledem**, podobně mluvíme o uzavřeném tahu. Uzavřená cesta je uzavřený sled, v němž se neopakují uzly (kromě toho, že $u_0 = u_k$) a navíc se neopakují ani hrany. Pravidelným (též regulárním) grafem nazýváme graf G, jehož všechny uzly jsou stejného stupně. Jestliže všechny uzly mají stupeň k, pak tento graf označíme za k-pravidelný či k-regulární. Graf, jehož všechny vrcholy jsou stupně 2, nazýváme **kružnicí** (neorientovaná uzavřená cesta) a **cyklusem** (orientovaná uzavřená cesta). Opět platí, že cyklus je zároveň i kružnicí, ale naopak to neplatí. Kružnice, která má tři hrany, se nazývá trojúhelník. **(1, 4)**



Obr. 2.9.1: Pravidelný graf, Kružnice

2.10. Sledy v ohodnocených grafech

V ohodnocených grafech často hovoříme o součtu ohodnocení hran v nějakému sledu, cestě, cyklu apod. V praxi se používá ohodnocení hrany např. k vyjádření vzdálenosti, množství práce, času nebo peněz k průchodu hranou. Toto ohodnocení počítáme vždy tolikrát, kolikrát se hrana ve sledu vyskytuje. Pro tento součet se vžil termín délka sledu (cesty, tahu, kružnice…) a v tomto smyslu se hovoří o nejkratším nebo nejdelším sledu, cestě. **(1)**



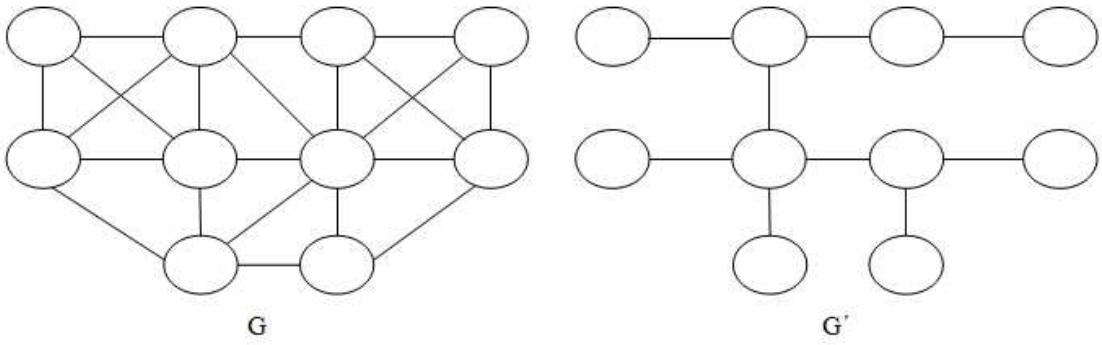
Obr. 2.10.1: Komponenta grafu

2.11. Strom

Stromem je každý souvislý graf bez kružnic a pro každé dva vrcholy stromu existuje právě jedna cesta, která je spojuje. V programování jsou stromy považovány za velmi důležitý druh datových struktur. Pomocí stromů můžeme také přehledně vyjádřit systematický postup probírání všech možností při řešení nějaké (grafové) úlohy. Např. strom všech maximálních cest grafu G , které vycházejí z uzlu *u*. **(1, 4)**

2.12. Kostra grafu

„Kostrou grafu G rozumíme podgraf v G, který obsahuje všechny vrcholy grafu G a je stromem.“ Kostra grafu je tedy libovolný podgraf, který hranami spojuje všechny vrcholy původního grafu a zároveň sám neobsahuje žádnou kružnici. [matoušek]
Na obrázku dole je graf G´ stromem grafu G a zároveň i jeho kostrou.



Obr. 2.12.1: Kostra grafu

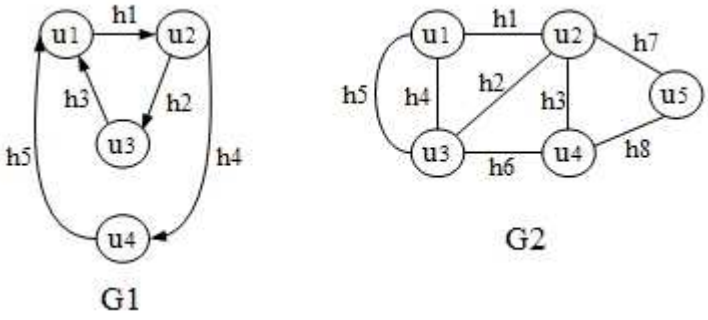
Reprezentace grafů

Nejčastějším a nejsrozumitelnějším způsobem reprezentace grafu, jak jsme poznali výše, je jeho popis pomocí diagramu (nakreslením), nebo přímo pomocí definice. Tato metoda je vhodná především pro grafy, které nejsou příliš složité a neobsahují především velké množství hran. Pokud však máme složitější graf a především provádíme -li výpočty v grafu pomocí počítače, zvolíme některý ze způsobů, které si uvedeme v následující kapitole.

2.13. Seznam uzlů a hran

Množinu uzlů popisuje prostým výčtem prvků, množinu hran popisujeme jako uspořádanou trojici tvořenou jménem hrany a jejím počátečním a koncovým uzlem. V neorientovaných grafech je jedno, jestli hranu $uv \in H$ prezentujeme jako dvojici *uv* nebo *vu*. Ale v případě orientovaných grafů už musíme dodržet správné pořadí uzlů, za správné pořadí považujeme to po směru šipky. Orientovaný graf může kromě šipky *uv* obsahovat totiž i opačnou šipku *vu*. Pokud nám nezáleží na jménu hrany, můžeme jej vypustit a hranu popsat dvojicí uzlů. **(1, 7)**

Orientovaný a orientovaný graf z obrázku dole můžeme popsat např. takto:



Obr.: 2.13.1 Graf G1, G2

Graf G_1:

Uzly: u1, u2, u3, u4

Hrany: (h1, u1, u2), (h2, u2, u3), (h3, u3, u1), (h4, u2, u4), (h5, u4, u1)

Graf G_2: pokud pro nás nejsou důležité jména uzlů a hran, můžeme je v seznamu vypustit.

Uzly: 1, 2, 3, 4, 5

Hrany: (1, 2), (2, 3), (2, 4), (3, 1), (3, 1), (3, 4), (2, 5), (4, 5)

Na závěr dodejme, že tento způsob reprezentace grafu je vhodný spíše pro zadávání grafu na vstupu nebo pro skladování grafu na pevném disku.

2.14. Seznam okolí vrcholů

Tento způsob reprezentace grafu je velmi podobný předchozímu případu. Množinu uzlů popisujeme opět výčtem, ale hrany jsou zapsány pouze svým jménem a koncovým uzlem, počáteční uzel je pro celou skupinu hran společný.

Graf G_1 popsaný tímto způsobem:

u_1: (h_1, u_2)

u_2: (h_2, u_3); (h_4, u_4)

u_3: (h_3, u_1)

u_4: (h_5, u_1)

K popisu neorientovaných grafů se tento způsob příliš nevyužívá, jelikož bychom museli uvést seznam množiny hran incidentních s jednotlivými uzly, což ovšem v tomto případě není příliš úsporné, protože každá hrana by byla vždy uvedena ve dvou seznamech jednotlivých uzlů. Možností jak tomuto jevu zamezit je nejdříve převést neorientovaný graf na orientovaný tak, že každou neorientovanou hranu $uv \in H$ nahradíme dvojicí hran s orientací $u\vec{v}$ a $v\vec{u}$ jdoucích proti sobě. **(1, 7)**

2.15. Matice sousednosti

Matice sousednosti zachycuje, které uzly spolu sousedí, respektive v matici si pro každou dvojici uzlů (u, v) pamatujeme, jestli z uzlu u vede hrana do uzlu v. Matice sousednosti o n uzlech je čtvercová matice S řádu n (a její prvky S_{jk} jsou dány předpisem:

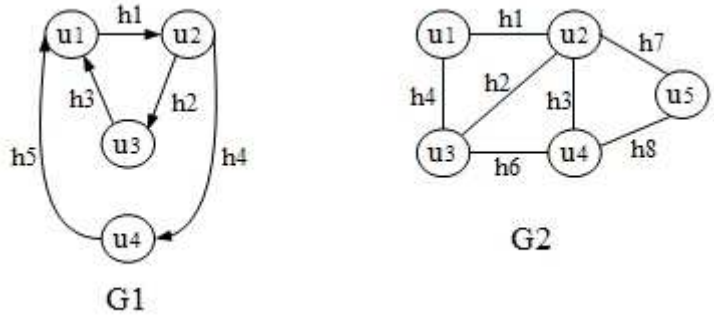
U neorientovaného grafu:

- $S_{jk} = 1$, pokud jsou uzly u_j a u_k sousední
- $S_{jk} = 0$, pokud uzly u_j a u_k sousední nejsou

U orientovaného grafu:

- $S_{jk} = 1$, pokud hrana vede z uzlu u_j do uzlu u_k
- $S_{jk} = 0$, pokud hrana z uzlu uj do uzlu u_k nevede

Matice sousednosti S pro grafy G1 a G2:



Obr. 2.15.1: Graf G1, G2 - matice sousednosti

$$S_{G1} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} \qquad S_{G2} = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \end{pmatrix}$$

Z příkladu si můžeme všimnout některých vlastností matice sousednosti:

- pokud uvažujeme grafy, které nemají smyčky, na hlavní diagonále jsou nuly
- u neorientovaného grafu je matice sousednosti symetrická podle hlavní diagonály
- počet jedniček v matici sousednosti u orientovaného grafu je rovna jeho počtu hran
- počet jedniček v matici sousednosti u neorientovaného grafu se rovná dvojnásobku počtu hran.

Pokud mají grafy stejnou matici sousednosti, pak jsou tyto grafy navzájem *izomorfní.* Naopak toto tvrzení však neplatí: změna pořadí vrcholů s sebou nese stejné změny v pořadí sloupců a řádků matice sousednosti. Vzájemně izomorfní grafy tedy mohou mít různé matice sousednosti. **(1, 7, 20)**

2.16. Matice incidence

Zatímco matice sousednosti popisuje vztahy mezi sousedícími uzly, matice incidence popisuje vztahy mezi hranou a jejím koncovým uzlem. Zvolíme - li libovolně pořadí uzlů $u_1 \ldots u_n$ a pořadí hran $h_1 \ldots h_n$, potom grafu G přiřadíme matici incidence I s prvky I_{jk}, které jsou dány předpisem:

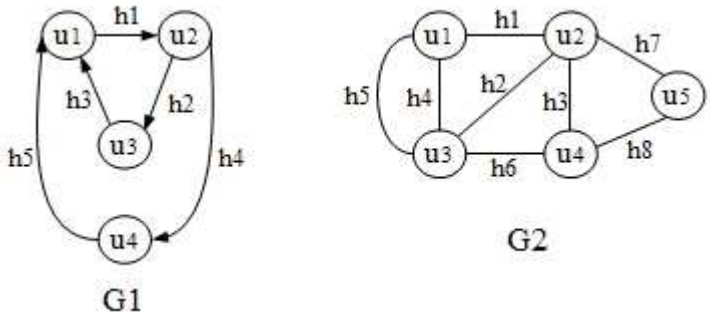
U neorientovaného grafu:

- $I_{jk} = 1$, pokud je uzel u_i incidentní s hranou h_k
- $I_{jk} = 0$, pokud uzel u_j s hranou h_k incidentní není

U orientovaného grafu:

- $I_{jk} = 1$, pokud uzel u_j je počátečním vrcholem hrany h_k
- $I_{jk} = -1$, pokud uzel u_j je koncovým vrcholem hrany h_k
- $I_{jk} = 0$ pokud uzel u_j není incidentní s hranou h_k

Matice incidence I pro grafy G1 a G2:



Obr. 2.16.1: Graf G1, G2 - matice incidence

$$I_{G1}=\begin{pmatrix} 1 & 0 & -1 & 0 & -1 \\ -1 & 1 & 0 & 1 & 0 \\ 0 & -1 & 1 & 0 & 0 \\ 0 & 0 & 0 & -1 & 1 \end{pmatrix} \qquad I_{G2}=\begin{pmatrix} 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{pmatrix}$$

Vlastnosti matice incidence:

- každý sloupec u matice sousednosti orientovaného grafu obsahuje vždy přesně jednu 1 a přesně jednu -1
- u matice incidence orientovaného grafu je součet hodnot v každém řádku roven počtu hran vycházejících z uzlu u_j – počet hran vstupujících do uzlu u_j
- každý sloupec u matice sousednosti neorientovaného grafu obsahuje vždy přesně dvě jedničky
- součet hodnot v *j*- tém řádku je roven stupni uzlu u_j **(1, 7, 20)**

3. Analýza problému a vlastní návrhy řešení

Grafové algoritmy

Algoritmus je schematický postup pro řešení určitého druhu problémů, který je prováděn pomocí konečného množství přesně definovaných kroků. Ačkoliv se dnes tento pojem používá především v informatice a přírodních vědách obecně, tak je jeho působnost daleko širší (kuchyňské recepty, návody a postupy...). Samotné slovo „algoritmus“ pochází ze jména perského matematika 9. století Abu Jafar Muhammada ibn Mūsā al-Chwārizmího, který ve svých dílech položil základy algebry (arabské číslice, řešení lineárních a kvadratických rovnic). **(3)**

3.1. Prohledávání do hloubky (Depth First Search - DFS)

Naše povídání o grafových algoritmech začneme dvěma základními způsoby procházení grafem. K tomu budeme potřebovat dvě podobné jednoduché datové struktury: *Fronta* je konečná posloupnost prvků, která má označený začátek a konec. Když do ní přidáváme nový prvek, přidáme ho na konec posloupnosti. Když z ní prvek odebíráme, odebereme ten na začátku. Proto se tato struktura anglicky nazývá *first in, first out*, zkráceně *FIFO*. *Zásobník* je také konečná posloupnost prvků se začátkem a koncem, ale prvky přidáváme a odebíráme z konce zásobníku. Anglický název této struktury označujeme jako *last in, first out*, čili *LIFO*.

Algoritmus prohledávání grafu do hloubky:

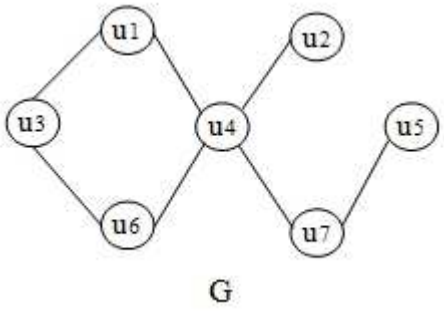
1. Na začátku máme v zásobníku pouze vstupní uzel u_0. Dále si u každého uzlu x pamatujeme značku x_z, která říká, zda jsme uzel již navštívili. Vstupní uzel je označený, ostatní nikoliv.
2. Odebereme uzel ze zásobníku, nazvěme ho u.
3. Každý neoznačený uzel, do kterého vede hrana z u, přidáme na zásobník a označíme.
4. Kroky 2 a 3 opakujeme, dokud není zásobník prázdný.

Na konci algoritmu budou označeny všechny uzly dosažitelné z uzlu u_0, tedy v případě neorientovaného grafu celá komponenta souvislosti obsahující u_0. To, že algoritmus někdy skončí, je zřejmé z kroku 3, kdy na zásobník přidáváme pouze uzly, které dosud nejsou označeny a hned je značíme. Proto se každý uzel může na zásobníku objevit nejvýše jednou, a jelikož ve 2. kroku pokaždé odebereme jeden uzel ze zásobníku, musí uzly někdy (konkrétně po nejvýše N opakováních cyklu) dojít. Ve 3. kroku probereme každou hranu grafu nejvýše dvakrát (v každém směru jednou). Časová složitost celého algoritmu je tedy lineární v počtu uzlů U a počtu hran H, tedy $O(U+H)$. Paměťová složitost je stejná, protože si musíme hrany a uzly pamatovat.

Použití:

Prohledávání do hloubky lze využít k nalezení kostry neorientovaného grafu, což je strom, který jsme prošli. Je stavebním kamenem algoritmů pro nalezení různých druhů komponent grafu, topologické třídění a řešení hádanek s jediným řešením jako jsou například bludiště. **(7, 14, 16, 19, 20)**

Mějme následující graf G a popišme průchod jeho jednotlivými uzly technikou prohledáváním do hloubky. Pro snazší orientaci jsme si uzly dopředu popsali. Vstupním uzlem je uzel *u1*.



Obr. 3.1.1: Původní graf

1. Počáteční podmínky: V zásobníku máme vstupní uzel u_1.
2. Odebereme vstupní uzel ze zásobníku, označíme ho příznakem u_{1z}, který říká, že uzel byl již navštíven a hledáme s ním incidentní uzly a ty vložíme do zásobníku.
   aktuální uzel: u_1, zásobník $\{u_3, u_4\}$

3. Ze zásobníku odebereme uzel u_3, učiníme ho uzlem aktuálním a označíme ho příznakem u_{3z}, který říká, že uzel byl již navštíven. Hledáme doposud nenavštívené uzly sousedící s uzlem u_3 a vložíme je do zásobníku. Uzel u_1 již do zásobníku vložit nemůžeme, protože byl již navštíven (to poznáme podle příznaku u_{1z})
   aktuální uzel: u_3, zásobník $\{u_4, u_6\}$

4. Ze zásobníku odebereme uzel u_6, učiníme ho uzlem aktuálním a označíme ho příznakem u_{6z}. Nenavštívené uzly (nemající příznak u_{nz}) sousedící s uzlem u_6 vkládáme do zásobníku. Uzel u_6 nemá žádného doposud nenavštíveného souseda, krom uzlu *u4*, který již v zásobníku je.
   aktuální uzel: u_6, zásobník $\{u_4\}$

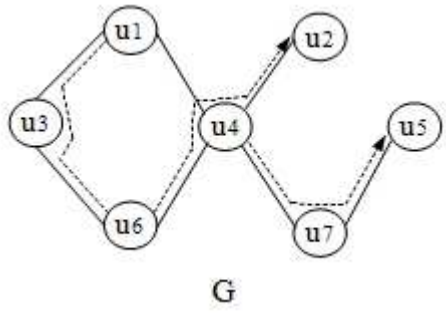
5. Ze zásobníku odebereme uzel u_4, učiníme ho uzlem aktuálním a označíme ho příznakem u_{4z}. Hledáme doposud nenavštívené uzly sousedící s uzlem u_4 a vložíme je do zásobníku.
   aktuální uzel: u_4, zásobník: $\{u_2, u_7\}$

6. Ze zásobníku odebereme uzel u_2, učiníme ho uzlem aktuálním a označíme ho příznakem u_{2z}. Doposud nenavštívené uzly sousedící s uzlem u_2 vkládáme do zásobníku. Uzel u_2 již nemá žádného doposud nenavštíveného souseda, proto se vracíme zpět a odebíráme ze zásobníku uzel u_7.
   aktuální uzel: u_2, zásobník: $\{u_7\}$

7. Ze zásobníku odebereme uzel u_7, učiníme ho uzlem aktuálním a označíme ho příznakem u_{7z}. Hledáme doposud nenavštívené uzly sousedící s uzlem u_7 a vložíme je do zásobníku.
   aktuální uzel: u_7, zásobník $\{u_5\}$

8. Ze zásobníku odebereme uzel u_5, učiníme ho uzlem aktuálním a označíme ho příznakem u_{5z}. Hledáme doposud nenavštívené uzly sousedící s uzlem u_5 a vložíme je do zásobníku. Tento uzel již žádného dosud nenavštíveného souseda nemá.
   aktuální uzel: u_5, zásobník $\{\ \}$

9. Zásobník je prázdný – průchod grafem je ukončen. Výsledný *DFS strom* ukazuje, v jakém pořadí byly jednotlivé uzly procházeny.



Obr. 3.1.2: DFS

3.2. Prohledávání do šířky (Breadth First Search – BFS)

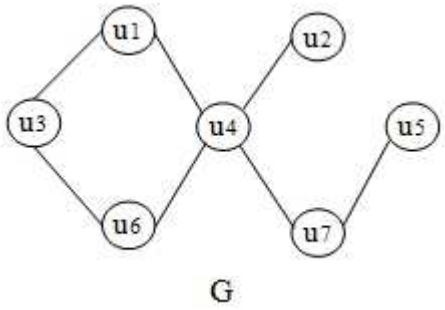
Prohledávání grafu do šířky je založeno na podobné myšlence jako prohledávání do hloubky, pouze místo zásobníku používá frontu. Na rozdíl od průchodu do hloubky se však jako další navštěvovaný uzel nevybírá soused aktuálního uzlu, ale vždy se bere uzel z fronty. Uzly jsou proto navštěvovány v tom pořadí, v jakém byly vloženy do fronty.

Prohledávání do šířky má také lineární časovou složitost s počtem hran a uzlů. Na každou hranu se ptáme dvakrát, stejně jako u prohledávání do hloubky, a paměťová složitost je rovněž *O(U + H)*. Tento algoritmus ovšem není vhodné použít v úlohách, ve kterých může mít i nejkratší řešení příliš mnoho tahů, neboť by vedl k nereálným paměťovým nárokům programu.

Použití:

Prohledávání do šířky lze použít na hledání komponent souvislosti, a stejně jako u předchozí techniky, také na hledání kostry grafu. Pokud nemáme ohodnocené grafy, lze jej také vhodně použít na najití nejkratší cesty mezi dvěma uzly. Dále lze technikou BFS testovat, zda- li je graf bipartitní („dvojdílný“) a v neposlední řadě také k výpočtu maximálního toku v grafu (realizace Ford-Fulkersonovou metodou) **(7, 14, 16, 19, 20)**

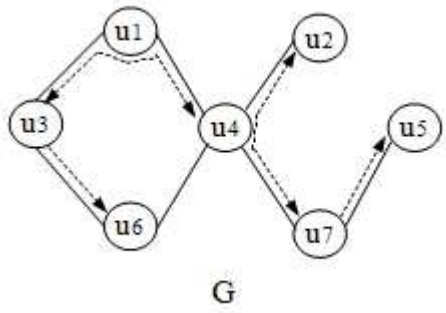
Mějme opět graf G a popišme si jeho průchod jednotlivými uzly technikou prohledáváním do šířky. Vstupním uzlem je opět uzel *u1*.



Obr. 3.2.1: Původní graf G

1. Počáteční podmínky: Ve frontě máme vstupní uzel u_1.
2. Odebereme vstupní uzel z fronty, označíme ho příznakem u_{1z}, který říká, že uzel byl již navštíven. Aktuální uzel má dva nenavštívené sousedy u_3, u_4, které vložíme do fronty.
   aktuální uzel: u_1, fronta $\{u_3, u_4\}$
3. Z fronty vybereme uzel u_3, učiníme ho uzlem aktuálním a označíme ho příznakem u_{3z}. Aktuální uzel má jednoho nenavštíveného souseda u_3.
   aktuální uzel: u_3, fronta $\{u_4, u_6\}$
4. Z fronty vybereme uzel u_4, učiníme ho uzlem aktuálním a označíme ho příznakem u_{4z}. Aktuální uzel má tři nenavštívené u_2, u_6, u_7, ty vložíme do fronty.
   aktuální uzel: u_4, fronta $\{u_6, u_2, u_6, u_7\}$
5. Z fronty vybereme uzel u_6, učiníme ho uzlem aktuálním a označíme ho příznakem u_{6z}. Aktuální uzel již nemá žádného nenavštíveného souseda, proto se vracíme zpět a vybíráme další v pořadí uložený uzel ve frontě.
   aktuální uzel: u_6, fronta $\{u_2, u_6, u_7\}$
6. Odebereme uzel u_2 z fronty, učiníme ho uzlem aktuálním a označíme ho příznakem u_{2z}. Aktuální uzel nemá žádného nenavštíveného souseda, proto se vracíme zpět a vybíráme další v pořadí uložený uzel ve frontě.
   aktuální uzel: u_2, fronta $\{u_6, u_7\}$

7. Odebíráme uzel u_6 z fronty, uzel již byl navštíven, proto pokračujeme dále a vybíráme další v pořadí uložený uzel z fronty.
   aktuální uzel: u_6, fronta $\{u_7\}$
8. Odebereme uzel u_7 z fronty, učiníme ho uzlem aktuálním a označíme ho příznakem u_{7z}. Uzel má jednoho nenavštíveného souseda u_5, kterého vložíme do fronty.
   aktuální uzel: u_7, fronta $\{u_5\}$
9. Z fronty vybereme uzel u_5, učiníme ho uzlem aktuálním a označíme ho příznakem u_{5z}. Aktuální uzel již nemá žádného nenavštíveného souseda.
   aktuální uzel: u_5, fronta { }
10. Fronta je prázdná – průchod grafem je u konce. Výsledný *BFS strom* ukazuje, v jakém pořadí byly jednotlivé uzly procházeny.



Obr. 3.2.2 Graf G - BFS

Minimální kostra grafu

Nalezení minimální kostry grafu je jedním z klasických problémů v oblasti grafových algoritmů. Problém určení minimální kostry není samoúčelný, ale je motivován mnoha praktickými aplikacemi v reálném životě (hledání nejkratšího vlakového spoje, nalezení minimální silniční sítě, rozvody elektřiny mezi jednotlivými městy apod.).

Při generování všech koster grafu jsou různé kostry daného grafu v zásadě rovnocenné - všechny obsahují stejný počet hran. Jiná situace ovšem nastane, pokud každé hraně přiřadíme nějakou nezápornou reálnou délku. V takovém případě má smysl se zabývat hledáním takové kostry, která má nejmenší součet délek svých hran.

Problém určení minimální kostry má také svoji historii, do které se úspěšně zapsali i dva čeští matematici. Nejprve je formuloval a úspěšně vyřešil v polovině 20. let O. Borůvka v souvislosti s návrhem elektrické na Moravě. Efektivnější metodu tohoto řešení později navrhnul r.1930 V. Jarník. Používání prvních počítačů po r. 1950 přineslo oživení zájmu o algoritmické řešení problému minimální kostry, a tak byly algoritmy O. Borůvky a V. Jarníka znovu nezávisle objeveny a publikovány v USA americkými matematiky J. Kruskalem a R. C. Primem**,** čímž se posléze staly všeobecně známými. **(10, 11, 12)**

3.3. Borůvkův-Kruskalův algoritmus

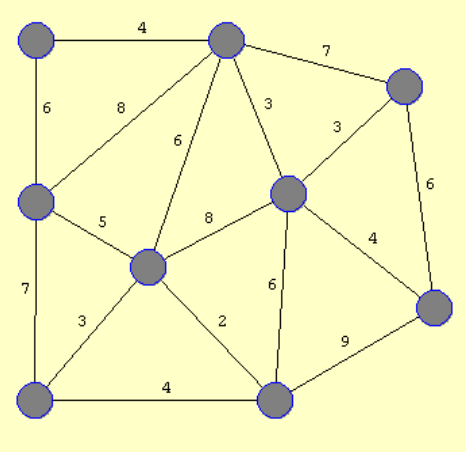
Základní myšlenkou Borůvkova - Kruskalova algoritmu je, že pro přidání hrany k postupně vznikající kostře vybírá takovou hranu, která má ze všech možných hran spojujících dva různé podstromy ve vytvářené kostře tu nejmenší váhu. Asymptotická časová složitost Borůvkova algoritmu je O(*H* log *U*), kde U je počet uzlů a H počet hran grafu. **(10, 11, 12)**

Použití:

Jednotlivé kroky algoritmu:

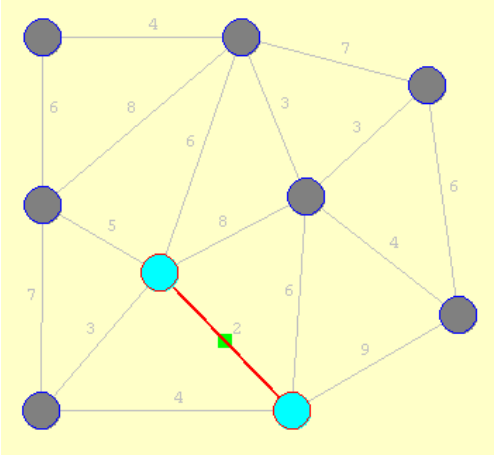
- Vytvoř seznam *L* hran vstupního grafu *G* a seřaď je do neklesajícího pořadí podle jejich ohodnocení.
- Vytvoř podgraf *S* grafu *G*, který je zpočátku prázdný.
- Pro každou hranu ze seznamu *L* proveď:
    - Nevznikne-li v grafu *S* přidáním této hrany kružnice, přidej ji do grafu *S* (včetně krajních uzlů).
- Opakujeme minulý krok, dokud vznikající kostra nespojí všechny vrcholy grafu.
- Graf *S* je minimální kostrou grafu *G*.

Využití algoritmu si předveďme na následujícím případu z praxe. Firma má záměr vybudovat kabelové rozvody pro televizi a další telekomunikační služby mezi sídlišti ve středně velkém městě. Rozložení sídlišť a jejich vzájemné vzdálenosti jsou uvedeny v grafu G. Stanovme optimální trasování rozvodů tak, aby délka výsledné sítě byla minimální.



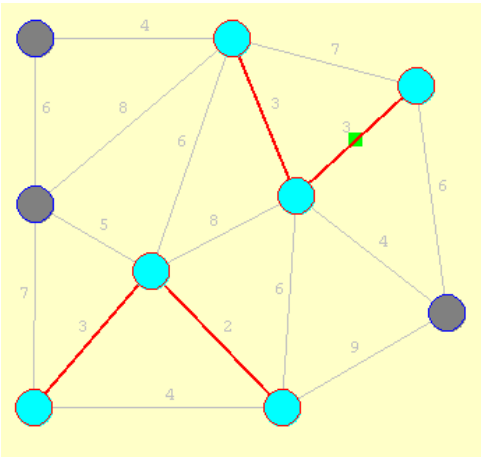
Obr. 3.3.1: Min. kostra grafu – původní graf G

Vybereme hranu s nejnižším ohodnocením (v tomto případě délky „2“) a zvýrazníme ji.



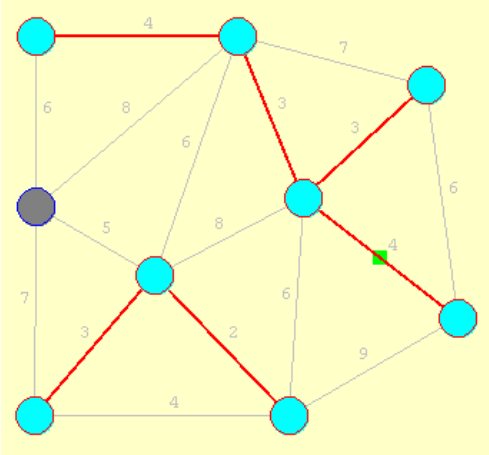
Obr. 3.3.2: Min. kostra grafu – krok 2

Nyní vybereme hrany s délkou „3“ a zvýrazníme je.



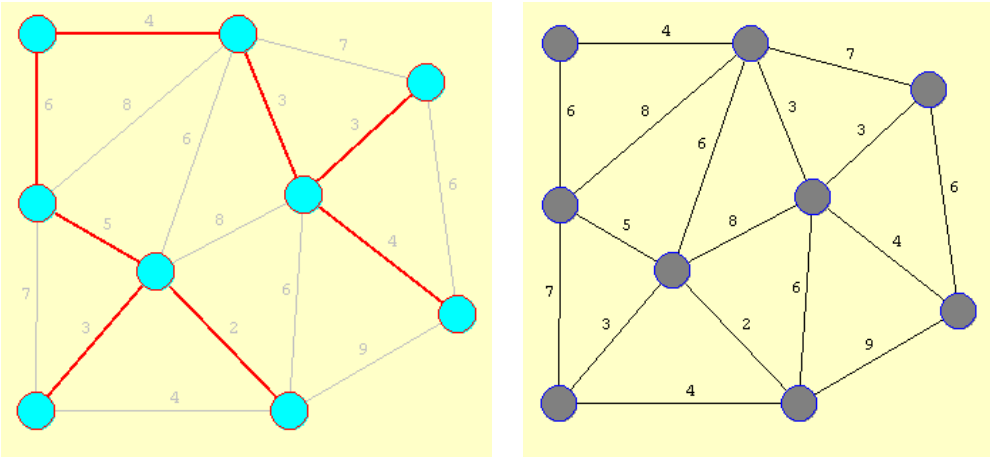
Obr. 3.3.3: Min. kostra grafu – krok 3

Dále vybereme hrany s délkou „4“ takové, které nevytvoří v grafu kružnici.



Obr. 3.3.4: Min. kostra grafu – krok 4

Tímto způsobem pokračujeme, dokud nám nevznikne výsledná minimální kostra grafu, která má celkové ohodnocení 19.



Obr. 3.3.5: Min. kostra grafu – výsledná min. kostra a orig. graf

pozn. zpracováno pomocí **(8)**.

3.4. Jarníkův - Primův algoritmus

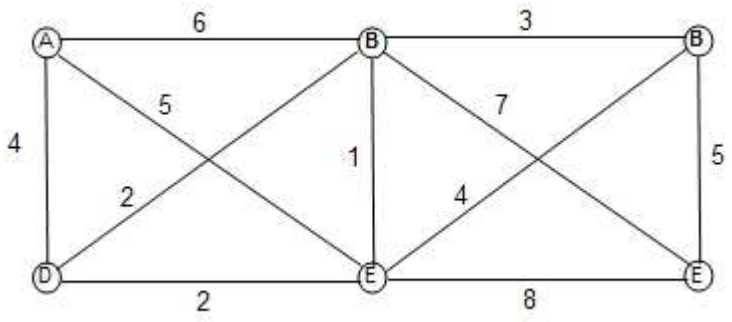
Jarník-Primova varianta algoritmu určení minimální kostry vychází z toho, že hrany vybrané do množiny hran reprezentující dosud vytvořenou část kostry tvoří stále jediný podstrom, k němuž se připojí přidanou hranou vždy pouze jeden nový uzel. Uzel, který je spolu s hranou přidáván, má k dosud vzniklé kostře ze všech zbývajících uzlů nejblíže. Protože hodnota "vzdálenosti od kostry" se přidáním uzlu může změnit, je po každém takovém kroku nutné provést přepočtení těchto údajů. Časová složitost Jarník-Primova algoritmu je $O(U^2)$. **(10, 11, 12,)**

Použití:

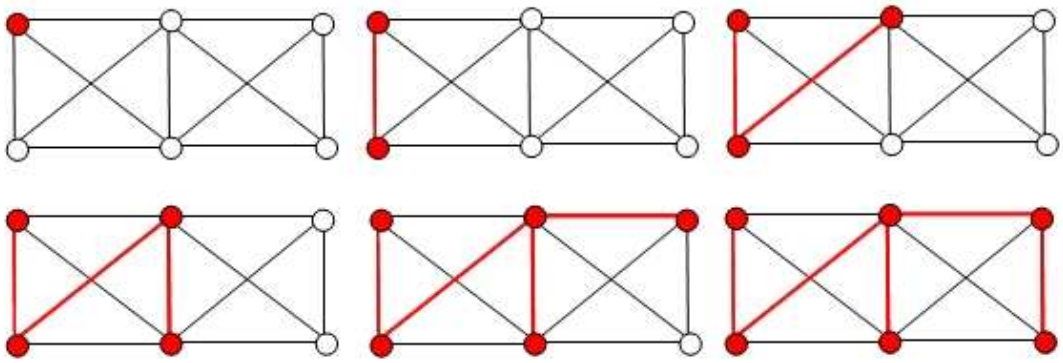
Algoritmus:

- Vytvoř prázdný graf *T*.
- Do grafu *T* vlož libovolný jeden uzel vstupního grafu *G*.
- Dokud má graf *T* méně uzlů než vstupní graf *G*, opakuj:
    - Najdi hranu s minimálním ohodnocením, spojující graf *T* s rozdílem grafů *G-T*
    - Přidej tuto hranu do grafu *T*.

Mějme následující graf, který nám vyjadřuje například zjednodušenou silniční síť mezi šesti jednotlivými městečky. Stanovme optimální trasu pro jednotlivé složky integrovaného záchranného sytému v případě náhlé evakuace obyvatel z důvodu přírodní katastrofy tak, že najděme její minimální kostru pomocí Jarníkova – Primova algoritmu.



Obr. 3.4.1: Původní graf



Obr. 3.4.2: Průběh Jarník-Primova algoritmu

Jednotlivé kroky:

1. Počátečním uzlem zvolíme uzel A. Nyní algoritmus vyhledá hranu s nejmenším ohodnocením, kandidáty jsou hrany (A, D), (A, B).
2. Vybereme hranu (A, D), protože má menší ohodnocení. Nový graf má stále menší počet uzlů než graf původní, algoritmus tedy pokračuje. Dalšími kandidáty na rozšíření jsou hrany (D, B) a (D, E). Zde si můžeme povšimnou, že hrany jsou stejně ohodnoceny, je tedy na nás jako si zvolíme. Výsledkem tohoto faktu je, že graf může mít dvě minimální kostry.
3. My si vybereme hranu (D, B). Nyní máme na výběr hrany (B, E), (B, C) a (B, F).
4. Vybereme hranu (B, E), protože má nejnižší ohodnocení. Zbývá, „dostat se“ do dvou zbývajících uzlů C a F.
5. Do uzlu C se dostaneme hranou (B, C) protože má nižší ohodnocení než-li hrana (E, C). Do uzlu F se můžeme dostat buď hranou (C, F), nebo (E, F).
6. Vybereme hranu (C, F), protože má nižší ohodnocení.
7. Máme výslednou minimální kostru grafu.

Výsledná minimální kostra je vyznačena na obrázku vpravo dole a má ohodnocení 15.

3.5. Dijkstrův algoritmus

Dijkstrův algoritmus je grafový algoritmus vytvořený nizozemským vědcem Edsgerem Wybem Dijkstrou (1930–2002), který slouží k vyhledání nejkratší cesty z počátečního uzlu do koncového uzlu nebo z počátečního uzlu do všech ostatních uzlů ohodnoceného grafu. Na Dijkstrův algoritmus lze pohlížet jako na zobecněné procházení grafu do šířky, při kterém se vlna nešíří na základě vzdálenosti definované jako „počet hran od zdroje“, ale jako „vzdálenost od zdroje“. Tato vlna proto zpracovává jen ty uzly, k nimž již byla nalezena nejkratší cesta. Dijkstrův algoritmus je použitelný jen tehdy, obsahuje-li graf pouze nezáporně ohodnocené hrany, protože jinak by nebyl schopen garantovat, že při zpracování uzlu byla již nalezena nejkratší možná cesta.

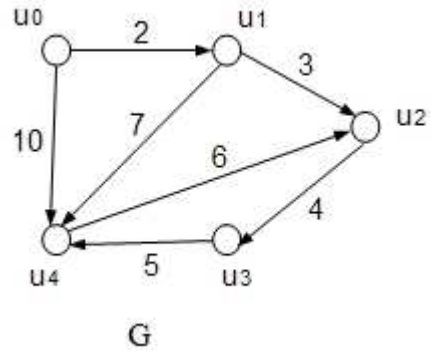
Při popisu algoritmu nám jako vstupní informace poslouží ohodnocený graf G (U,H) s počátečním uzlem u_0, na výstupu mějme pole D[u] udávající délku nejkratších cest mezi počátečním uzlem u_0 a uzly u. **(3, 10, 13, 14, 15, 16)**

Jednotlivé kroky Dijkstrova algoritmu:

- Postupně konstruujeme množinu S $\subseteq$ U takovou, že nejkratší cesty od u_0 do uzlů z S procházejí přes uzly z S.
- Do množiny S vložíme počáteční uzel u_0.
- Hodnoty pole D inicializujeme takto:
  - pro počáteční uzel $u_0 = 0$,
  - pro každý uzel u, který sousedí s počátečním uzlem u_0 = ohodnocení hrany (u_0, u),
  - pro ostatní uzly = ∞.
- Dokud S $\neq$ U opakujeme:
  - Najdeme uzel w s minimální hodnotou D[w].
  - Přidáme uzel w do množiny S.

- Pro každý uzel *u* sousedící s uzlem *w* který není v množině S provedeme:
  - hodnota D[u] je minimum ze stávající hodnoty a D[w] plus ohodnocení hrany (w,u).

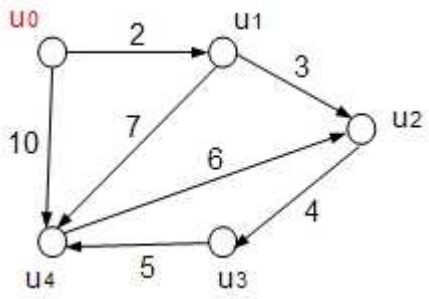
Mějme následující kladně ohodnocený orientovaný graf G a nejděme nejkratší cesty z jeho počátečního uzlu u_0 do všech ostatních uzlů.



Obr. 3.5.1: Původní graf

0. krok – inicializace:

Do množiny S přidáme počáteční uzel u_0 a hledáme jeho sousedy a hodnoty jejich délky cest zapíšeme do tabulky do hodnot polí D.

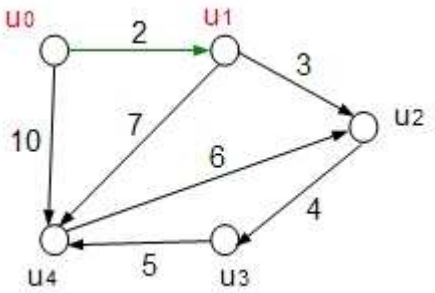


Obr. 3.5.2: Dijkstrův algoritmus - 0. krok

N	S	w	D[w]	$D[u_1]$	$D[u_2]$	$D[u_3]$	$D[u_4]$
0	u_0	-	-	2	∞	∞	10

1. krok

Nejkratší cesta z počátečního uzlu vede do uzlu u_1 a má délku rovnu hodnotě 2. Přídáme uzel u_1 do množiny S, hledáme jeho sousedy a hodnoty délky jejich nejkratších cest od počátečního uzlu u_1 zapíšeme do tabulky do hodnot polí D. Do uzlu u_1 již žádná kratší cesta nevede, tudíž jí označíme jako definitivní nejkratší cestu z počátečního uzlu u_0 do uzlu u_1 (v tabulce označeno červenou barvou). Stejným způsobem postupujeme i v dalších krocích dokud nenajdeme nejkratší cestu do všech uzlů.

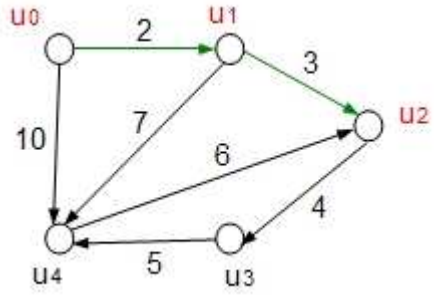


D[u] := min(D[u], D[w] + l(w, u))

Obr. 3.5.3: Dijkstrův algoritmus - 1. krok

N	S	w	D[w]	D[u_1]	D[u_2]	D[u_3]	D[u_4]
0	u_0	-	-	2	∞	∞	10
1	u_0, u_1	u_1	2	2	5	∞	9

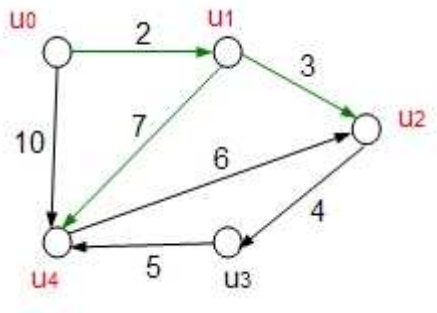
2. krok



Obr. 3.5.4: Dijkstrův algoritmus - 2. krok

N	S	w	D[w]	D[u_1]	D[u_2]	D[u_3]	D[u_4]
0	u_0	-	-	2	∞	∞	10
1	u_0, u_1	u_1	2	2	5	∞	9
2	u_0, u_1, u_2	u_2	5	2	5	9	9

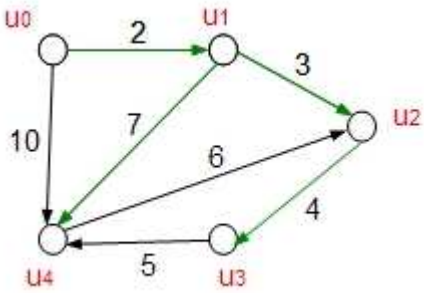
3. krok



Obr. 3.5.5: Dijkstrův algoritmus - 3. krok

N	S	w	D[w]	D[u_1]	D[u_2]	D[u_3]	D[u_4]
0	u_0	-	-	2	∞	∞	10
1	u_0, u_1	u_1	2	2	5	∞	9
2	u_0, u_1, u_2	u_2	5	2	5	9	9
3	u_0, u_1, u_2, u_4	u_4	9	2	5	9	9

4. krok



Obr. 3.5.6: Dijkstrův algoritmus - 4. krok

N	S	w	D[w]	D[u_1]	D[u_2]	D[u_3]	D[u_4]
0	u_0	-	-	2	∞	∞	10
1	u_0, u_1	u_1	2	2	5	∞	9
2	u_0, u_1, u_2	u_2	5	2	5	9	9
3	u_0, u_1, u_2, u_4	u_4	9	2	5	9	9
4	u_0, u_1, u_2, u_4, u_3	u_3	9	2	5	9	9

Vidíme, že množina S již obsahuje všechny uzly množiny U původního grafu G, algoritmus proto končí – našli jsme nejkratší cesty do všech uzlů grafu.

Časová složitost Dijkstrova algoritmu při použití lineárního pole je $O(U^2+H)$, kterou můžeme ještě upravit do podoby $O(U^2)$, jelikož H je nejvýše U^2. K uchovávání délek dosud nalezených nejkratších cest můžeme ovšem použít specializovanou datovou strukturu zvanou binární halda. Ta bude na začátku obsahovat U prvků a v každém kroku se počet jejích prvků sníží o jeden: Nalezneme a smažeme nejmenší prvek v čase $O(\log U)$ a případně upravíme délky nejkratších cest do sousedů právě zpracovávaného vrcholu. To pro každou hranu trvá rovněž $O(\log H)$, celkově za všechny hrany tedy $O(H \log U)$. Z toho vyjde celková časová složitost algoritmu $O((U+H) \log U)$, a proto je pro „řídké“ grafy (grafy s menším počtem) výrazně lepší použít tuto implementaci.

Využití

Dijkstrova algoritmu využíváme například v routovacích protokolech, konkrétně třeba v algoritmu OSPF (Open Shortest Path First), který se používá pro interní routování uvnitř autonomního systému (AS). Routery, používající tento protokol, si v pravidelných krátkých intervalech zvláštními zprávami (ECHO) kontrolují spojení se svými sousedními routery. Při zjištění jakékoliv změny zasílá oznámení všem routerům v síti, ty si pak podle nové informace přepočítají nové cesty síti právě pomocí dijkstrova algoritmu a podle toho upraví routovací tabulky. Různé modifikace tohoto algoritmu se používají pro hledání spojení dopravních prostředků, při trasování v navigačních softwarech apod. Algoritmus se modifikuje podle toho, jestli chceme trasu nejrychlejší, nejkratší, nejekonomičtější apod. **(16, 18, 19)**

3.6. Bellman – Fordův algoritmus

Bellmanův-Fordův algoritmus (pojmenovaný po amerických matematicích Richardu Bellmanovi a Lesteru Fordovi, Jr.) počítá, stejně jako algoritmus Dijkstrův, nejkratší cestu v ohodnoceném grafu z jednoho uzlu do všech ostatních uzlů ovšem s tím rozdílem, že připouští výskyt hran se záporným ohodnocením.

Bellmanův-Fordův algoritmus využívá, podobně jako algoritmus Dijkstra, metodu *relaxace* hran. Do této operace vstupují dva uzly a hrana, která mezi nimi vede. Pokud je vzdálenost zdrojového uzlu sečtená s délkou hrany menší než aktuální vzdálenost cílového uzlu, tak se za předchůdce cílového uzlu na nejkratší cestě označí zdrojový uzel. V případě nesplnění nerovnosti tato hrana cestu nezkracuje a neprovádí se proto žádné změny. Zatímco Dijkstrův algoritmus relaxoval vždy jen hrany vycházející z vybraného uzlu, v tomto případě se opakovaně relaxují *systematicky* všechny hrany; jinak řečeno v Dijkstrově algoritmu pokud projdeme v grafu všemi následovníky daného uzlu, tak tento uzel považujeme za definitivní a později jej už neupravujeme. Zatímco Bellmanův-Fordův algoritmus prochází všechny uzly grafu opakovaně a přepočítává a upravuje hodnoty nejkratší cest.

Detekování záporných cyklů provádíme tak, že v posledním cyklu algoritmu provedeme ještě jednou relaxaci přes všechny hrany, ale jen zkoumáme, zda nějaká z rozběhnutých relaxací bude úspěšná, zda-li se takovou relaxací mohlo docílit zmenšení vzdálenost některého z uzlů D[*u*]. Pokud toto nastane, pak graf obsahuje cyklus záporné délky, pokud k žádné takové relaxaci nedojde, algoritmus může vrátit výsledek.

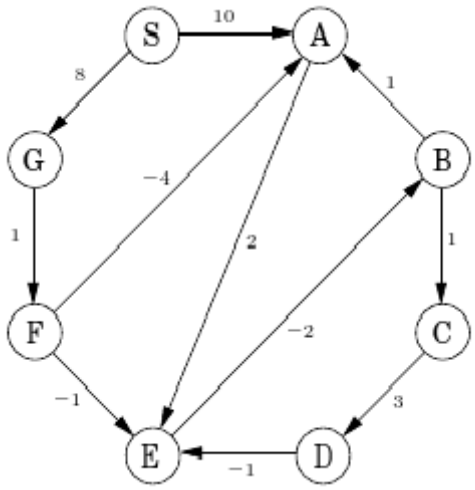
Asymptotická časová složitost Bellmanova-Fordova algoritmu je *O*(U*H), neboť relaxace hrany má v tomto případě konstantní složitost (na rozdíl od Dijkstrova algoritmu, nepotřebuje přesouvat prvky v prioritní frontě) a provádí se *U*H*- krát.
K použití BFA přistupujeme většinou pouze tehdy, je-li reálný předpoklad výskytu záporného ohodnocení hran. **(1, 3, 7, 15)**

Použití:

V praxi se využívá Bellman-Fordova algoritmu např. v implementaci směrovacího protokolu Routing Information Protocol (RIP), který si v současné době

stále uchovává svou popularitu díky jeho jednoduchosti a široké podpoře. RI protokol je součástí skupiny Distance Vector Routing Protocols (volným překladem směrování podle délky vektoru), které typicky používají Bellman-Fordův algoritmus k určení nejlepší cesty sítí. **(17)**

V příkladu máme následující ohodnocený graf a máme najít nejkratší cesty z uzlu A do všech ostatních uzlů. Průběh Bellman-Fordova algoritmu je zachycen tabulkou, která pro každý uzel *u* obsahuje hodnoty délky nejkratší cesty do daného uzlu D[*u*] v jednotlivých iteracích.



Obr. 3.6.1: Bellman - Fordův algoritmus - původní graf

Uzel	Iterace							
	0	1	2	3	4	5	6	7
S	0	0	0	0	0	0	0	0
A	∞	10	10	5	5	5	5	5
B	∞	∞	∞	10	6	5	5	5
C	∞	∞	∞	∞	11	7	6	6
D	∞	∞	∞	∞	∞	14	10	9
E	∞	∞	12	8	7	7	7	7
F	∞	∞	9	9	9	9	9	9
G	∞	8	8	8	8	8	8	8

Na začátku algoritmu nastavíme všem uzlům hodnotu pole D[u] na nekonečno, kromě počátečního uzlu S, ten má hodnotu pole vzdálenosti D[s] rovnu 0.

V první iteraci vede cesta do uzlů A a G, hodnoty délky nejkratší cesty zapíšeme do tabulky.

Ve druhé iteraci pokračujeme do uzlů E a F. Relaxujeme všechny hrany a zjišťujeme, zda- li do dostupných uzlů nevede nějaká kratší cesta. Pokud ne, pokračujeme další iterací.

Ve třetí iteraci se již můžeme dostat do uzlu B, do něhož se dostaneme přes uzel A a E cestou délky 10. Relaxací všech dostupných hran zjistíme, že do uzlu A vede kratší cesta než délky 10 a to cesta přes uzly G, F o délce 5. Do uzlu E se rovněž dostaneme přes uzly G, F. Délka cesty má hodnotu 8.

Ve čtvrté iteraci nalezneme cestu do uzlu C, cesta vede přes uzel B a její délka je 1. Jelikož víme, že do uzlu B jsme se dostali v minulé iteraci délkou cesty 10, nejkratší zatím známa cesta do uzlu C, bude mít tudíž hodnotu 11. Procházíme znovu všechny hrany a zjišťujeme, zda- li se nedá dostat do uzlů kratší cestou. Do uzlu B se lze dostat přes uzly G, F, E o celkové délce 6. Hodnotu zapíšeme do tabulky. Do uzlu E existuje kratší cesta přes uzly G, F, A než- li cesta stávající a má celkovou délku 7.

V páté iteraci dosáhneme uzlu D, dostaneme se do něj přes uzel C – délka nejkratší cesty je rovna 14. Opět relaxuje všechny hrany a zjišťujeme, zda se do nějakého uzlu nedá dostat kratší cestou, než tou kterou máme uloženou v poli D příslušného uzlu. Do uzlu B se dostaneme cestou přes uzly A a E, délka cesty je 5, do uzlu C lze jít přes uzly G, F, E a B o délce 7.

V předchozí iteraci jsme našli cesty do všech uzlů grafu, proto nyní již jen budeme relaxovat jednotlivé hrany a zjišťovat, zda- li jsou délky nejkratších cest do všech uzlů definitivní, nebo jestli se nedá do jednotlivých uzlů dostat i kratší cestou. V minulé iteraci jsme našli nejkratší cestu do C o délce 7, hodnotu D[$_D$] tedy přepisujeme na (7+3) 10. Dále jsme našli nejkratší cestu do uzlu B o délce 5. Víme, že do uzlu C vede cesta z B o délce jedna, proto nejkratší cesta do C bude mít nyní hodnotu 6 a vede přes uzly G, F, A, E a B.

V poslední iteraci ještě naposledy testujeme přepočítáváme jednotlivé hrany

a jediným uzlem do kterého lze najít kratší cestu, než je zadaná v tabulce je cesta do uzlu D vedoucí přes uzly G, F, A, E, B a C délky 9. Nalezli jsme nejkratší cesty do všech uzlů z počátečního uzlu S.

3.7. Floyd-Warshallův algoritmus.

Floydův–Warshallův algoritmus (poprvé popsaný matematiky Robertem Floydem a Stephenem Warshallem) pracuje na orientovaném grafu, který neobsahuje záporné cykly a najde délky nejkratších orientovaných cest mezi každou dvojicí uzlů. Navíc ze všech cest stejné délky vybere tu s nejmenším počtem hran. Pokud chceme spočítat vzdálenost každé dvojice uzlů, tak můžeme n-krát použít Dijkstruv algoritmus (na každý uzel). Lepší možností je však použít Floyd - Warshalluv algoritmus, který počítá všechny vzdálenosti přímo, proběhne rychleji než n-krát použitý Dijkstrův algoritmus a ještě se snadněji implementuje. Pokud nám nezáleží na časové složitosti, ale jen na rychlosti naprogramování, tak je lepší volbou než algoritmus Dijkstra.

Floyd-Warshallův algoritmus má na vstupu matici sousednosti délek D^0. Pokud mezi dvěma uzly (u_0, u_1) vede hrana délky l, tak tato matice obsahuje na indexu (u_0, u_1) právě tuto hodnotu. Na diagonále má tato matice samé nuly a na ostatních indexech, které neodpovídají hraně nekonečno. Jinými slovy tato matice obsahuje vzdálenosti uzlů, které nevedou skrze žádného „prostředníka“.

V každé iteraci Floyd-Washallova algoritmu se tato matice přepočítá tak, aby vyjadřovala vzdálenost všech dvojic uzlů skrze postupně se zvětšující množinu přípustných prostředníků. Jednoduše řečeno matice D^1 bude vyjadřovat vzdálenost všech uzlů s možností využití jednoho (daného) prostředníka, D^2 vzdálenost při možném využití dvou prostředníků, D^k při možnosti využití k prostředníků.

Výstupem Floyd-Warshallova algoritmu je matice nejkratších cest mezi jednotlivými uzly. Například pokud hledáme nejkratší cestu mezi uzly u_0, u_1, pak se podíváme na záznam na řádku u_0, sloupec u_1. Pokud je pole nulové, pak cesta neexistuje, v opačném případě udává délku nejkratší cesty mezi těmito dvěma uzly.

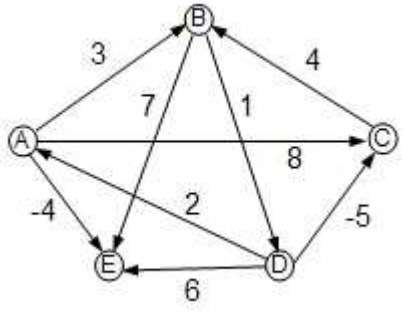
Nevýhodou Floyd-Warshallova algoritmu je vysoká asymptotická časová složitost algoritmu, která je rovna $O(U^3)$, a proto hlavně v případě velkého množství uzlů je použití tohoto algoritmu nevhodné.

Použití:

Floyd-Warshallova algoritmu můžeme využít v mnoha oblastech teorie grafů. Jeho modifikace se používá se například k nalezení regulárního výrazu označujícího

regulární jazyk přijímaný konečným automatem, dále k inverzi matic reálných čísel (Gauss-Jordan algoritmus). Využití nachází také k nalezení cest s maximálním tokem v aplikaci optimálního směrování mezi dvěma uzly, maximálního využití šířky pásma v tocích v síti apod. Dále jej používáme na testování neorientovaného grafu, zda - li je graf bipartijní a v neposlední řadě také k rychlým výpočtům v sítích Pathfinder. **(3, 7, 9)**

Mějme zadaný následující orientovaný graf a chtějme najít nejkratší cesty mezi všemi jeho uzly. Řešení provedeme pomocí Floyd-Warshallova algoritmu.



Obr. 3.7.1: Floyd-Warshallův algoritmus - původní graf

Nejprve sestavíme matici sousednosti obsahující délky cest mezi jednotlivými uzly. Hodnotou ∞ je vyjádřena skutečnost, že mezi dvěma danými uzly neexistuje spojení.

Uzel	A	B	C	D	E
A	0	3	8	∞	-4
B	∞	0	∞	1	7
C	∞	4	0	∞	∞
D	2	∞	-5	0	∞
E	∞	∞	∞	6	0

Nyní provádíme jednotlivé kroky Floyd-Warshallova algoritmu. Během jednotlivých iterací je matice sousednosti nahrazena maticí délek nejkratších cest a hodnoty jsou v každém kroku průběžně aktualizovány (změny v matici jsou barevně vyznačeny).

První iterace (využití uzlu A jako prostředníka):

Uzel	A	B	C	D	E
A	0	3	8	∞	-4
B	∞	0	∞	1	7
C	∞	4	0	∞	∞
D	2	-5	-5	0	-5
E	∞	∞	∞	6	0

Druhá iterace (využití uzlů A, B):

Uzel	A	B	C	D	E
A	0	3	8	4	-4
B	∞	0	∞	1	7
C	∞	4	0	5	11
D	2	-5	-5	0	-5
E	∞	∞	∞	6	0

Třetí iterace (využití uzlů A, B, C):

Uzel	A	B	C	D	E
A	0	3	8	4	-4
B	∞	0	∞	1	7
C	∞	4	0	5	11
D	2	-1	-5	0	-5
E	∞	∞	∞	6	0

Čtvrtá iterace (využití uzlů A, B, C, D):

Uzel	A	B	C	D	E
A	0	3	-1	4	-4
B	3	0	-4	1	-1
C	7	4	0	5	3
D	2	-1	-5	0	-5
E	8	5	1	6	0

Pátá iterace (využití všech uzlů):

Uzel	A	B	C	D	E
A	0	1	-3	2	-4
B	3	0	-4	1	-1
C	7	4	0	5	3
D	2	-1	-5	0	-5
E	8	5	1	6	0

Výpočet pomocí Floyd-Warshallova algoritmu je u konce, na výstupu jsme dostali matici délek nejkratších cest mezi jednotlivými uzly grafu. Každá hodnota udává délku nejkratší cesty mezi párem uzlů. Například pokud hodnota „5“ leží na průniku třetího řádku a čtvrtého sloupce, vyjadřuje délku nejkratší cesty z uzlu C do uzlu D.

3.8. Johnsonův algoritmus

Základní ideou Johnsonova algoritmu je n-násobné použití Dijkstrova algoritmu, kde n je počet vrcholů grafu. Aby bylo možné Dijkstrův algoritmus použít, je nutné graf, který může být obecně záporně ohodnocený, přehodnotit. Získání nového ohodnocení c‘ z původního c se nazývá přehodnocením grafu G a musí splňovat následující dvě podmínky:

- Pro každou dvojici uzlů *u, v* je nejkratší cesta pro ohodnocení *c* shodná s nejkratší cestou pro ohodnocení *c‘*.
- Pro každou hranu *(u, v)* je ohodnocení *c‘(u, v)* nezáporné.

Přehodnocení lze provést v čase $O(U*H)$. Jako další podprogram se v Johnsonově algoritmu používá Bellman-Fordova algoritmu. Výsledkem je buď matice vzdáleností nebo detekce záporného cyklu a ukončení algoritmu. Efektivnějšího řešení při použití nad řídkými grafy dosahuje Johnsonův algoritmus také ve srovnání s Floyd-Warshallovým algoritmem. Pro určení složitosti Johnsonova algoritmu je rozhodující *U* - násobné použití Dijkstrova algoritmu. Odhad složitosti Johnsonova algoritmu má hodnotu $O(U^2 log\ U + H*\ U)$, což je stále pro řídké grafy rychlejší než Floyd-Warshallův algoritmus se složitostí $O(n^3)$. **(7, 15)**

4. Závěrečné zhodnocení

Práce rozšiřuje základní poznatky z teorie grafů a zaměřila se především na popis a využití jednotlivých grafových algoritmů. Oblast teorie grafů se v dnešním světě nadále velmi dynamicky rozvíjí, vznikají nové algoritmy, nebo dochází k modifikacím algoritmů stávajících s cílem řešit stále složitější problémy v co nejkratším čase, s co nejnižší paměťovou složitostí apod. O všech těchto algoritmech a jejich modifikacích si lze najít mnoho informací v literatuře i na internetu, jejich popis a použití přesahuje rámec této práce.

Pro porovnání základních grafových algoritmů v mé práci je zapotřebí si klást otázku z jakého pohledu je chceme vůbec porovnávat. Algoritmy prohledávání do hloubky a do šířky jsou jakousi vstupní branou do grafových algoritmů, používají se k nalezení koster grafu, komponent grafu, zjišťování zda- li je graf bipartitní apod. Velkou úlohu hrají jako dílčí nebo podpůrné algoritmy při řešení složitějších výpočtů v grafech. Použití algoritmů hledání minimální kostry, ve kterých zanechali významnou stopu i čeští matematici, je vysvětleno v textu na velmi praktickém příkladu rozvodů kabeláže a najde jistě uplatnění v praxi.

Algoritmy pro hledání nejkratší cesty v grafu lze posuzovat z hlediska podmínek použití, časové složitosti a složitostí implementace. Dijkstův algoritmus a jeho modifikace je možno označit za nejrozšířenější, hlavně díky jeho poměrně rychlému zpracování a nepříliš složité implementaci. Nevýhodou je podmínka použití pouze kladně ohodnoceného grafu. Nejčastějším případem využití Dijkstrova algoritmu je nalezení nejkratší cesty z daného počátečního uzlu do uzlu koncového. S tímto problémem se v reálném životě setkáváme dnes a denně, ať již tohoto algoritmu využívají routery v počítačových sítích, nebo navigační software v našich mobilech, nebo PDA přístrojích, či systémy na hledání dopravních spojení apod.

Pokud připustíme existenci záporného ohodnocení v grafu je potřeba využít algoritmu Floyd - Warshallova, Bellman - Fordova nebo algoritmu Johnsonova. Výhodou Floyd - Warshallova algoritmu je jeho velmi jednoduchá implementace a přímočarost. Před samotným použitím algoritmu je ovšem potřeba sestavit z daného grafu matici sousednosti. Nevýhodou Floyd - Warshallova algoritmu je, že počítá sice vzdálenosti mezi všemi jednotlivými uzly, ale nevíme kudy samotné cesty vedou, pokud

algoritmus vhodně nemodifikujeme. Bellman - Fordův algoritmus je nejuniverzálnějším ze všech uvedených algoritmů, avšak z hlediska časového nejnáročnější, protože je závislý na počtu hran v grafu, a proto není vhodné jeho použití v hustých grafech s velkým množstvím hran. Johnsonův algoritmus je z hlediska implementačního nejnáročnější, protože kombinuje postupy Dijkstrova a Bellman - Fordova algoritmu. Tyto nevýhodu smazává nejnižší časovou složitostí při hledání nejkratších cest mezi všemi uzly.

Jak lze vidět, výběr vhodného algoritmu není tak jednoznačný, jak by se mohlo na začátku zdát a je třeba se vždy před začátkem řešení dané úlohy rozhodnout, který z uvedených faktorů je pro náš výběr určující a přednostně je nezbytně nutné se přesně seznámit se specifikací zadaného problému.

5. Odborná literatura

Knižní zdroje

1. DEMEL, J. *Grafy a jejich aplikace*. 2002. ISBN: 80-200-0990-6.
2. HOLOUBEK, J. *Ekonomicko - matematické metody*. 2006. ISBN 978-80-7157-970-0.
3. KOLÁŘ, Josef. *Teoretická informatika*. 2. vyd. Praha : Česká informatická společnost, 2004. 205 s. ISBN 80-900853-8-5.
4. MATOUŠEK, Jiří a NEŠETŘIL, Jaroslav. *Kapitoly z diskrétní matematiky*. 2002. 374 s. ISBN 80-246-0084-6.
5. MEZNÍK, Ivan. *Diskrétní matematika*, 2004, 132 s. ISBN 80-214-2754-X

Elektronické zdroje

6. *Algoritmy na grafech* [online]. 2005 [cit. 2010-06-03]. Dostupné z WWW: <http://www.fd.cvut.cz/projects/k611x1p/lide/hudy/ang.doc>.
7. ČERNÝ, Jakub. *Základní grafové algoritmy*, 2008, 175 s. [online] [cit. 2009–12-15]. Dostupné z: <http://kam.mff.cuni.cz/~kuba/ka/>.
8. DRLÍK, Radovan. *Kruskalův algoritmus* [online]. 2006 [cit. 2010-06-03]. Dostupné z WWW: <http://www.drlik.com/kruskal/>.
9. GOSPER, Jeffrey J. *Floyd-Warshall All-Pairs Shortest Pairs Algorithm* [online]. 1998 [cit. 2010-06-03]. Dostupné z WWW: <http://www.pms.ifi.lmu.de/lehre/compgeometry/Gosper/shortest_path/shortest_path.html#visualization>.
10. HORDEJČUK, Vojtěch. *Algoritmus* [online]. 2009 [cit. 2010-06-03]. Dostupné z WWW: <http://voho.cz/wiki/stitek/algoritmus/>.
11. *Hledaní minimální kostry neorientovaného grafu* [online]. 2000 [cit. 2010-06-03]. Dostupné z WWW: <http://www.kasan.net/fel/pjw/>.

12. JIROVSKÝ, Lukáš. *Teorie grafů* [online]. 2008-05-20 [cit. 2010-06-03]. Hledání minimální kostry v grafu. Dostupné z WWW: <http://teorie-grafu.elfineer.cz/vybrane-problemy/minimalni-kostra.php#kostra4Menu>.
13. KOVÁŘ, Petr. *Diskrétní matematika* [online]. 2009 [cit. 2010-06-03]. Dostupné z WWW: <http://homel.vsb.cz/~kov16/files/dim_prednaska08.pdf>.
14. KOVÁŘ, Petr. *Grafové algoritmy* [online]. 2005, Aktualizace: Sat, 07 May 2005 [cit. 2010-06-02]. Dostupné z WWW: <http://homel.vsb.cz/~kov16/talks/algoritmy/obsah.html>
15. KRAUTER, Michal. *Nejkratší cesty v grafu*. [s.l.], 2009. 65 s. FIT VUT v Brně. Diplomová práce. Dostupný z WWW: <http://www.fit.vutbr.cz/study/DP/rpfile.php?id=7245>.
16. MAREŠ, Martin; MATOUŠEK, David; ŠKODA, Petr. *Korespondenční seminář z programování* [online]. 2005-12-12 [cit. 2010-06-03]. Grafy. Dostupné z WWW: <http://ksp.mff.cuni.cz/tasks/18/cook2.html>.
17. MIKULEC, Martin. *RIPv1 - Úvod* [online]. 2010 [cit. 2010-06-03]. Dostupné z WWW: <http://owebu.bloger.cz/PC-site/RIPv1-Uvod-1-dil?km=c>.
18. PERFILIEVA, Irina. *Dijkstrův algoritmus* [online]. 2006-11-16 [cit. 2010-06-03]. Dostupné z WWW: <http://nofak.meep.cz/download/_VYSL1_/1639_Lecture7.pdf>.
19. TEUBELOVÁ, Dana. *Základní grafové algoritmy* [online]. 2008-06-05 [cit. 2010-06-03]. Základní grafové algoritmy. Dostupné z WWW: <http://statnice.obrys.cz/index.php?title=Z%C3%A1kladn%C3%AD_grafov%C3%A9_algoritmy_-_prohled%C3%A1v%C3%A1n%C3%AD_graf%C5%AF_(do_hloubky,_do_%C5%A1%C3%AD%C5%99ky),_hled%C3%A1n%C3%AD_nejkrat%C5%A1%C3%AD_cesty,_minim%C3%A1ln%C3%AD_kostra_grafu,_toky_v_s%C3%ADt%C3%ADch.#Floyd-Warshall.C5.AFv_algoritmus>.
20. VEČERKA, Arnošt. *Grafy a grafové algortimy* [online]. 2007 [cit. 2010-06-03]. Dostupné z WWW: <http://www.cs.vsb.cz/ochodkova/courses/gra/Grafy_a_grafove_algoritmy.pdf>.

6. Seznam obrázků