Contents

1 Základní pojmy

Zdroj Uzel, do kterého nevede žádná hrana

Úplný graf Graf, ve kterém pro libovolné různé dva vrcholy z množiny vrcholů existuje hrana.

Úplný bipartitní graf Graf, který lze rozdělit na dvě partity takové, že vrcholy jedné pratity nemají nemají vzájemně žádnou hranu, ale mají hranu s každým vrcholem druhé partity.

Úplný k-partitní graf Viz. úplný bipartitní graf.

Cesta $\mathbf{P_m}$ Posloupnost vrcholů s m hranami, pro kterou platí, že v grafu existuje hrana z daného vrcholu do následníka. Vrcholy ani hrany se nesmí opakovat. Délkou cesty se rozumí počet hran.

Stupeň vrcholu $\mathbf{deg_G(v)}$ Číslo označující počet hran při vrcholu v.

Otevřené okolí vrcholu $\boldsymbol{v}$ Množina všech sousedů vrcholu v.

Uzavřené okolí vrcholu $\boldsymbol{v}$ Množina všech sousedů vrcholu v včetně vrcholu v.

Regulární graf Graf je r-regulární, pokud pro všechny vrcholy platí, že mají stupeň r. Graf je regulární, pokud je r-regulární pro nějaké r.

Izolovaný vrchol Vrchol stupně 0.

2 Grafy

2.1 Vlastnosti grafů

Princip sudosti Pro každý graf platí, že součet stupňů všech vrcholů je roven dvojnásobku počtu hran. Z toho plyne, že počet vrcholů s lichého stupně sudý.

2.2 Podgrafy

Podgraf Libovolná podmnožina vrcholů a hran grafu G.

Indukovaný podgraf Libovolná množina vrcholů grafu G a všechny hrany, které navzájem spojují vrcholy ze zvolené podmnožiny.

Klika grafu Takový podgraf H grafu G, že H je úplný graf. Je tedy izomorfní s nějakým úplným grafem.

Nezávislá množina Indukovaný podgraf, který nemá žádné hrany.

Souvislý graf Graf, ve kterém pro každé dva vrcholy u a v platí, že existuje cesta z u do v. V opačném případě je nesouvislý.

Souvislá komponenta Indukovaný podgraf H grafu G, který je souvislý a zároveň neexistuje podgraf F grafu G takový, že H je podgraf F. Tedy je největší možný souvislý podgraf.

2.3 Souvislost grafu

2.3.1 DFS - Prohledávání do hloubky

- Z anglického *Depth First Search*, tedy prohledávání "nejdříve do hloubky".
- Algoritmus se stále zanořuje, ostatní sousedy prochází až po vynoření.
- Tento algoritmus je vhodný na vyhledávání souvislých komponent.

2.4 Orientované grafy

Orientovaný graf Uspořádaná dvojice *(V, E)*, kde V je neprázdná konečná množina vrcholů, E je množina orientovaných hran.

Orientovaná hrana Uspořádaná dvojice (u, v) je dvojice různých vrcholů u,v. Vrchol u je předchůdce, v je následník.

2.4.1 Stupeň vrcholu v orientovaném grafu

Vstupní stupeň Symbolem $\deg_G{}^+(v)$ označíme počet hran grafu G, končících ve vrcholu v.

Výstupní stupeň Symbolem $\deg_G{}^-(v)$ označíme počet hran grafu G, vycházejících z vrcholu v.

Stupeň vrcholu Jako stupeň vrcholu označíme $\deg_G(v) = \deg_G{}^+(v) + \deg_G{}^-(v)$

2.4.2 Okolí stupně v orientovaném grafu

Vstupní okolí Množina všech vrcholů $N_G{}^+(v)$, ze kterých vede hrana do vrcholu v (množina předchůdců).

Výstupní okolí Množina všech vrcholů $N_G{}^-(v)$, do kterých vede hrana z vrcholu v (množina následníků).

Okolí Sjednocení $N_G(v)$

2.4.3 Zdroje, stoky, izolované vrcholy

Izolovaný vrchol Stupeň vrcholu je 0.

Zdroj Vstupní stupeň vrcholu je 0.

Stok Výstupní stupeň vrcholu je 0.

2.4.4 Souvislost

Symetrizace Odstranění z grafu informace o orientaci hran.

Slabá souvislost Graf je slabě souvislý, pokud je souvislá jego symetrizace (souvislý po odstranění orientace hran).

Silná souvislost Graf je silně souvislý, pokud pro každé vrcholy u a v existuje orientované cesta z u do v a současně existuje orientovaná cesta z v do u.

3 Stromy

Strom Graf, který je souvislý a zároveň neobsahuje cyklus.

Les Graf, který neobsahuje žádnou kružnici (jeho jednotlivé komponenty jsou stromy).

List Vrchol se stupněm 1.

3.1 Listy

- Každý strom obsahující alespoň dva dva vrcholy obsahuje alespoň dva listy.
- Je-li G graf o alespoň 2 vrcholech a v je list, pak $G - v$ je také strom.

3.2 Vlastnosti

- Pro každé dva vrcholy existuje právě jedna cesta mezi nimi.
- Strom je souvislý a vynecháním libolné hrany bude nesouvislý.
- Počet vrcholů je o 1 větší než počet hran, tedy $|V| = |E| + 1$

4 Vzdálenost v grafech, topologické uspořádání

4.1 Kostra grafu

- Podgraf souvislého grafu G, který obsahuje všechny vrcholy původního grafu a přitom je stromem.
- Nesouvislé grafy nemají kostru.
- Souvislý graf s kružnicemi má více koster.
- Nějakou kostru lze nalézt algoritmem DFS. Složitost nalezení je $O(|V| + |E|)$

4.2 BFS - Prohledání do šířky, nejkratší cesta

Algoritmus prochází graf pomocí iterativního výběru sousedů, které přidává do fronty. Poté co v uzlu přidá všechny potomky do fronty, vezme další uzel z fronty a přidá jeho potomky.

- Algoritmus při průchodu nalezne nejkratší cestu z počátku s do libovolného uzlu v.
- Pomocí BFS lze najít kostru grafu (tzv. kostru do šířky).

4.3 Topologické třídění

- Orientovaný graf G je acyklickým pokud neobsahuje orientovanou kružnici.
- Je-li G orientovaný a acyklický, pak obsahuje zdroj i stok.

Topologické uspořádání Uspořádání vrcholů grafu do řady $v_1, v_2, \ldots, v_n$ takové, kde pro všechny hrany platí, že hrany vedou pouze z vrcholů s menším indexem do vrcholu s menším.

Topologické uspořádání lze nalézt pomocí algoritmu DFS.

5 Řazení

5.1 Vlastnosti

5.1.1 Varianty řazení

Řazení probíhá buď offline (nejprve přijdou data, následně se řadí) nebo online (úkoly chodí průběžně během zpracování).

5.1.2 Paměťová náročnost

Algoritmy jsou buď in-place (využívají kromě počáteční paměti jen konstantní paměť navíc). Out-of-place algoritmy řadí prvky v nově naalokované paměti.

5.1.3 Stabilita

Určuje, jestli navzájem si rovné prvky jsou v seřazené podobě ve stejném pořadí (stabilní) nebo proházené (nestabilní).

5.1.4 Citlivost

Udává, zda algoritmus má stejnou složitost pro libovolný vstup (datově necitlivý) nebo se složitost mění v závislosti na vstupu (datově citlivý).

5.2 Algoritmy řazení

5.2.1 BubbleSort

BubbleSort funguje na principu prohazování dvojic, kde vždy po porovnání dvojci buď prohodí nebo ponechá ve stejném pořadí a posune se o prvek dál. Po prvním průchodu je na konci největší prvek napravo a dále se řadí $n - 1$ prvků. Algoritmus se zastaví, pokud v jednom běhu zleva doprava neproběhlo žádné prohození.

- Vhodný na částečně seřazenou posloupnost.
- Seřadí prvky v čase $O(n^2)$.
- Stabilní, in-place, datově citlivý.

5.2.2 ShakerSort

Nadstavba nad BubbleSort. Prochází oběma směry, tedy po první průchodu (zleva doprava) je vpravo největší prvek. Po druhém průchodu (zprava doleva) je nalevo nejmenší prvek.

- Řeší problém pomalého posunu malých prvů směrem doleva, snižuje tedy počet porovnání.
- Asymptoticky stále $O(n^2)$.
- Stabilní, in-place, datově citlivý.

5.2.3 SelectSort

Posloupnost se rozdělí na seřazenou (zpočátku prázdná) a neseřazenou. Provádí se opakovaný výběr nejmenšího prvku, který se vymění s prvním prvkem neseřazené části.

- Asymptoticky O(n^2).
- Nestabilní, in-place, datově necitlivý.

5.2.4 InsertSort

Řazení vkládáním. Posloupnost rozdělíme na seřazenou (první prvek posloupnosti) a neseřazenou ($n-1$ prvků). V prvním kroce vybereme následující prvek a porovnáme ho s prvním a vložíme na správné místo. V dalším kroce bereme následující prvek a porovnáme ho s předchozími, vložíme na správné místo a zbytek podle potřeby posuneme. Takto postupujeme dokud není neseřazená část prázdná.

- Stabilní, in-place, datově citlivý.

5.2.5 HeapSort

Viz. HeapSort, kapitola Haldy.

6 Haldy

Zakořeněný strom Strom, kde je zvolený vrchol k jako kořen. Vrcholy se dělí podle vzdálenosti od kořene do hladin, kde v nulté hladině je kořen.

Binární strom Zakořeněný strom, kde každý vrchol má maximálně 2 syny. U synů se rozlišuje, který pravý a který levý.

Tvar haldy Strom má všechny hladiny kromě poslední plně obsazené. Poslední hladina se zaplňuje zleva.

Haldové uspořádání Pro každý vrchol v a jeho syna s platí, že $k(s) < k(v)$, tedy že každý syn má větší hodnotu než otec.

6.1 Binární minimová halda

Datová struktura tvaru binárního stromu, splňující haldové uspořádání a dodržující tvar haldy.

6.1.1 Vlastnosti

- Binární halda s n prvky má $\lfloor logn \rfloor + 1$ hladin.
- Nalezení minima v čase O(1).
- Odstranění minima v čase O($logn$).

6.1.2 Vložení prvku

1. Nový prvek se přidá na konec nejspodnější hladiny. Pokud je plná, založí se nová hladina.
2. Pokud platí haldové uspořádání mezi novým prvkem a jeho otcem, lze skončit.
3. Pokud neplatí, prohodíme otce a syna.
4. Po prohození mohlo být haldové pravidlo porušeno o hladinu výš, je tedy potřeba zkontrolovat pořadí s novým otcem, v případě nutnosti pak prvky prohodit.
5. Tímto způsobem pokračujeme až ke kořeni.

6.1.3 Odstranění minima

Odstranit minimum nelze udělat triviálně bez porušení haldového pravidla. Lze ale odstranit nejpravější list poslední hladiny. Pro odstranění minima lze prohodit minimum s tímto listem a následně probublat list na správné místo v haldě.

1. Prohození nejpravějšího listu z poslední hladiny s minimem.
2. Odstranění minima (nyní je na pozicu listu - stačí odpojit).
3. Probublání kořene na správné místo.
   (a) Výběr syna s nižším klíčem. Pokud tento syn je menší, prohoď jinak skonči.
   (b) Opakuj stejný postup o hladinu níž.

6.1.4 Reprezentace v paměti

- Haldu je možné reprezentovat pomocí pole, pokud očíslujeme postupně vrcholy od 1 do n.
  - Levý syn má index $2i$.
  - Pravý syn má index $2i + 1$.
  - Otec má index $\lfloor \frac{i}{2} \rfloor$.
  - Výraz $i \, mod \, 2$ udává, zda je vrchol levý či pravý syn.

6.1.5 Konstrukce haldy

Binární haldu lze pomocí operace HeapInsert vytvořit v čase $O(n \log n)$. Při reprezentaci polem to ale lze zvládnout i v linárním čase $O(n)$. Algoritmus *BuildHeap* staví na vlastnosti tvaru haldy, tedy že $\lceil \frac{n}{2} \rceil$ jsou listy haldy (sami o sobě validní haldy). Stačí tedy na vrcholy s indexy $\lfloor \frac{n}{2} \rfloor, \ldots, 1$ zavolat BubbleDown.

6.2 Řazení haldou

Halda díky rychlé konstrukci nabízí vynikající algoritmus na řazení *HeapSort*.

1. Vlož prvky do pole.
2. Pomocí algoritmu *BuildHeap* sestav haldu.
3. Opakovaným voláním algoritmu *ExtractMin* vygeneruj setříděnou posloupnost $O(n \log n)$.

Výsledná složitost pro seřazení haldou je tedy $O(n \log n)$.

7 Amortizovaná složitost

Protože *worst-case* nepopisuje vždy skutečné chování programu, udává se u algoritmů, které se volají nad dynamicky se měnící strukturou tzv. amortizovaná složitost.

Amortizovaná složitost Udává průměrnou časovou složitost pro jedné operace pro k operací.

7.1 Příklad - Binární sčítačka

Uchovává bitovou reprezentaci čísla v paměťových buňkách. Sčítačka podporuje operace:

- $Inc(x)$ zvýší číslo o 1.
- $Add(x, y)$ přidá číslo y k číslu x.

Pomocí účetní metody lze dokázat, že složitost operací je nejvýše $O(2n)$. Tedy složitost jedné je amortizovaně $O(1)$.

8 Binomiální halda

Binomiální strom řádu k Uspořádaný zakořeněný strom, pro který platí:

- B_0 je tvořen pouze kořenem.
- B_k lze získat pomocí stromu B_{k-1}, kterému připojíme jako nejpravějšího syna strom B_k.

Binomiální halda Složení souboru binomiálních stromů. Každý strom dodržuje haldové uspořádání. V uspořádání se žádný strom se stejným řádem nevyskytuje dvakrát. Soubor je uspořádán vzestupně.

8.1 Vlastnosti binomiálních stromů

- Počet hladin stromu B_k je $k + 1$.
- Stupeň kořene stromu B_k je k.
- Počet vrcholů stromu B_k je 2^k.
- Binomiální strom s n vrcholy má hloubku $log\, n$ a počet synů kořene také $log\, n$.

8.2 Vlastnosti binomiální haldy

- Binomiální strom B_k se v souborů stromů vyskytuje tehdy, je li v dvojkovém zápisu čísla n nastavený k-tý nejnižší bit na 1.
- Halda se skládá z $log\, n$ binomiálních stromů.

8.3 Sloučení hald

Sloučení probíhá obdobně jako při sčítání v binární sčítačce, tedy slučuji stromy se stejným stupněm a udržuji si přenos (aby nevzniklo více stromů se stejným stupněm k).

8.4 Vložení prvku

Provádí se pomocí slučování hald (slučuji s jednoprvkovou haldou).

8.5 Odstranění minima

Minimum nalezneme průchedem kořenů jednotlivých stromů. Od minima odtrhneme všechny potomky (a jejich podstromy) a sloučíme je do nové haldy. Tuto novou haldu sloučíme s původní.

9 Binární vyhledávací strom

Binární vyhledávací strom Binární strom, ve kterém v každém uzlu platí, že levý syn je menší a pravý větší než otec.

9.1 Nalezení minima

1. Je-li levý syn prázdný, vrať sebe.
2. (nebo) Vrať minimum z levého podsyna.

9.2 Vložení prvku

1. Je-li kořen v prázdný, vytvoř nový uzel a vrať ukazatel na v.
2. (nebo) Je-li hodnota menší než kořen v, vlož hodnotu do levého podstromu a vrať výsledek.
3. (nebo) Je-li hodnota větší než kořen v, vlož hodnotu do pravého podstromu a vrať výsledek.
4. (nebo) Hodnota už ve stromu je, vrať ukazatel na v.

9.3 Odstranění prvku

Při odstranění můžou nastat 3 různé situace:

1. Vrchol nemá žádného syna, lze ho odebrat.
2. Vrchol má jednoho syna, lze ho synem nahradit.
3. Vrchol má oba syny, je nutné nalézt následníka a tím ho nahradit, následník má zcela jistě nejvýše jednoho syna.

9.4 Časová složitost, vyvážení

Časová složitost operací je rovna počtu hladin O($h(v)$). Přičemž hloubka stromu je minimálně ($log|T(v)|$), nejhůře pak O($|T(v)|$).

Dokonale vyvážený BVS Strom, pro který v každém jeho vrcholu v platí, že $||L(v)|-|R(v)|| <= 1$.

Je-li strom dokonale vyvážený, mají operace nad ním složitost O($log\, n$), vzhledem k hloubce $\lfloor log\, n \rfloor$.

10 AVL - Hloubkově vyvážené stromy

AVL strom Hloubkově vyvážený strom, ve kterém pro kavzdý vrchol v platí $|h((l(v)))-h(r(v)) <= 1|$.

- AVL strom na n vrcholech má hloubku právě $log\, n$.

Operace vložení a odstranění vrcholu mohou způsobit, že strom přestane být vyvážený. Z tohoto důvodu se v každém vrcholu udržuje znaménko vrcholu, které nabývá hodnot -1, 0, 1. Znaméko vrcholu lze spočítat jako $h(r(v)) - h(l(v))$, tedy hloubka pravého prodstromu ponížená o hloubku levého. Vyskytne-li se v nějakém vrcholu jiné znaménko, je potřeba strom opravit pomocí rotace.

10.1 Vložení prvku

- Nový vrchol se vkládá jako list se znaménkem 0.
- Pomocí rekurze je potřeba přepočítat znaménka předků na cestě ke kořeni. Může nastat více případů:
  1. Podmínka je porušena ve vrcholu, který je levým synem:
     (a) Jeho levý podstrom je vyšší - rotace LL.
     (b) Jeho pravý podstrom je vyšší - rotace LR (převede na problém LL).
  2. Podmínka je porušena ve vrcholu, který je pravým synem:
     (a) Jeho pravý podstrom je vyšší - rotace RR.
     (b) Jeho levý podstrom je vyšší - rotace RL (převede na problém RR).

11 Slovníky, tabulky a hashování

11.1 Hashování

Hashování Pro množinu klíčů zvolíme konečnou množinu umístění (hashovací tabulka) a hashovací funkci, která každému klíči přiřadí právě jednu přihrádku. Cílem je zvolit takovou velikost a takovou funkci, aby se počet kolizí minimalizoval.

Kolize Protože hashovací tabulka má menší kapacitu než je velikost množiny klíčů, může se stát, že několik klíčů se po použití hashovací funkce zobrazí na jedno umístění. Tento jev se nazývá kolize.

11.1.1 Kolize

- Metoda řetízků - každé umístění v tabulce je buď prázdné, nebo v něm začíná spojový seznam uložených prvků.

11.1.2 Hashovací funkce

Ideální hashovací funkci (bez kolizí) nelze sestrojit, proto se používají funkce které se chovají "náhodně".

- Lineární kongruence $x -> ax \, mod \, m$
  - Konstanta m bývá prvočíslo a a je dostatečně velká konstanta nesoudělná s m (typicky $0.618m$).
- Vyšší bity součinu
- Skalární součin
- Polynom

11.1.3 Složitost

Pro ideální průběh hashování vyždujeme následující vlastnosti

1. Výpočet hashovací funkce je rychlý O(1).
2. Funkce rozděluje univerzum rovnoměrně.
3. Vstupní data z univerza jsou na vstupu rozdělena rovnoměrně.

Uvažujeme-li hashovací tabulku velikosti m s n prvky a funkci h splňující nároky. Potom je střední hodnota hledání/mazání/vkládání $O(n/m)$.

11.1.4 Hashování s otevřenou adresací

Použije se pole s m přihrádkami, kde se do každé přihrádky vejde pouze jeden prvek. V případě, že je místo obsazené, zkouší se náhradní tak dlouho, dokud není nalezené volné místo. Hashovací funkce tedy ke každému prvku k z univerza vyhledávací posloupnost. Ta určuje v jakém pořadí se budou náhradní umístění hledat, funkce h je tedy $h(k, i)$.

Mazání Pro mazání se používá metoda náhrobků. Pokud je prvek vymazený, označí se umístění náhrobkem a jiný prvek už tam nemůže být vložen. Pokud by se označil jako prázdný, byl by problém s vyhledáváním: to by mohlo předčasně skončit. Jakmile je zaplněný určitý počet prvků (např. $m/4$), tabulka se celá přehashuje.

Funkce

Lineární přidávání Prohledávací posloupnost je dána funkcí $h(k,i) = (f(k)+i) \, mod \, m$ kde $f(k)$ je obyčená hashovací funkce. Využívá po sobě jdoucí přihrádky, ale tim vytváří souvislé bloky.

Dvojité hashování Posloupnost je dána funkcí $h(k,i) = (f(k) + i * g(k)) \, mod \, m$ kde $f(k)$ i $h(k)$ jsou hashovací funce, m je prvočíslo a i je počet neúspěšných pokusů.

12 Rekurze, metoda rozděl a panuj

Rekurzivní algoritmus Postup řešení problému, kdy se nad vstupními daty opakuje stejný postup pro určité části dat. Staví na stejném řešení stejného podproblému s menším rozsahem.

12.1 MergeSort

Pomocí rekurze půlí intervaly, které má řadit dokud intervaly nejsou jednoprvkové. Při návratu z rekurze tyto jednoprvkové seřazené posloupnosti "slévá".

12.2 Rekurze

Koncová Rekurzivní volání je posledním příkazem algoritmu, po kterém se už neprovádějí žádné další operace.

Lineární V těle algoritmu je jen jedno volání rekurze (nebo dvě ale disjunktní - provede se pouze jedno).

Stromová V těle algoritmu jsou alespoň dvě rekurzivní volání. Obvykle algoritmy rozděl a panuj: $F(n) = F(n-1) + F(n-2)$.

12.3 QuickSort

QuickSort je algoritmus, který k řazení využívá takzvaného pivota. Pomocí pivota rozdělí posloupnost na tři části:

1. Levá - Prvky menší než pivot
2. Střední - Prvky rovné pivotu
3. Pravá - Prvky větší než pivot

Následně algoritmus pustí sám sebe na Levou a Pravou část. Výsledky se na konci spojí a vznikne seřazená posloupnost.

12.4 CountingSort

Algoritmus, u kterého je předem nutné znát minimum, maximum a celkový počet prvků:

1. Vytvoř dvě pole o velikosti n.
2. Projdi vstupní data a spočítej kolikrát se každý prvek v poli nachází.
3. Počáteční pozici nejmenšího prvku nastav na 1, následně každou další pozici dopočítej jako pozice předchozího prvku + počet předchozího prvku.
4. Vypiš prvky tak, že projdeš vstupní pole a do výstupního pole vypíšeš prvek na pozici, kterou udává tabulka pozic, následně pozici v tabulce pozic inkrementuj.

13 Dynamické programování

Dynamické programování Technika založená na rekurzivním rozkladu problému. Na rozdíl od techniky *Rozděl a panuj* využívá DP toho, že se některé problémy opakují. Podmínkou DP je, že každý podproblém se vyřeší právě jednou. Toho se docílí pomocí memorizace.

13.1 Nejdelší rostoucí podposloupnost

Pole se prochází odzadu, pro každý prvek se projdou odzadu přechůdci (resp. následníci) a hledá se takový, který je větší a má největší roustocí posloupnost. K její délce se přičte jednička a přiřadí se hodnotě zkomaného čísla, pokračuje se o číslo vlevo.

14 Minimální kostra

Minimální kostra Kostra, kteréa má mezi všemi kostrami nejmenší součet vah hran.

Na vyhledávání se používá Jarníkův (Primův) algoritmus.

15 Řezy grafu

Řez grafu Pokud rozdělíme vrcholy grafu na dvě disjunktní množiny A a B, hranám majícím jeden vrchol v A a jeden v B říkáme elementární řez grafu.

15.1 Jarníkův algoritmus

1. Začíná se s jedním vrcholem a žádnou hranou.
2. Ze všech již objevených vrcholů se vezmou v potaz všechny hrany, které vedou do neobjeveného vrcholu a z nich se vybere ta s nejmenší vahou.
3. Nyní se algoritmus opakuje.

15.2 Kruskalův algoritmus

Založený na stejném principu jako Jarníkův algoritmus, tedy vybírání hran s nejmenší váhou. Oproti Jarníkovi ale používá metodu spojování stromů. Pokud přidává hranu, musí se vždy podívat, jestli spojuje dvě různé komponenty.

15.2.1 Implementace

1. Polem (ke každému prvku si držíme, v jaké komponentě se nachází)
2. Pomocné stromy (keříky) - v poli se drží keřík tak, že každý vrchol si pamatuje svého předka. Při vyhledávání se porovnají kořeny keříků, pokud jsou stejné tak se jedná o jednu komponentu. V opačném případě se mělčí připojí pod hlubší.

16 Využití algoritmů

- Existence grafu
- Počet automorfismů
- Hledání silných komponent
- Nalezení cyklů

- Kostra grafu
    1. Kostra
    2. Minimální kostra - Jarníkův algoritmus
- Nejkratší cesta z u do v
- Topologické upořádání grafu