1 Minimální kostra grafu

Scéna: *Euklidovská kostra*

Scéna ukazuje množinu bodů v rovině, které budeme nazývat vrcholy. Naším cílem je propojit je nejlacinějším způsobem pomocí hran, které přímo spojují dvojice vrcholů. V této scéně je cena hrany dána Euklidovskou vzdáleností jejích vrcholů a cena propojení je rovna součtu cen použitých hran.

Poznamenávám, že Euklidovská metrika (označovaná také jako L_2-metrika) definuje vzdálenost d dvou vrcholů $s = [s_x, s_y]$ a $t = [t_x, t_y]$ takto:

$$d = \sqrt{(s_x - t_x)^2 + (s_y - t_y)^2}.$$

Po klepnutí na knoflík [**Krok**] Algovize ukáže nejlacinější propojení bodů tak, aby bylo možno se z každého vrcholu postupně dostat do libovolného jiného, takzvanou *minimální kostru*.

Změnou volby z [**Počítej**] na [**Edituj**] se přejde do editačního módu, kdy je možno stávající vrcholy přetahovat myší, klepnutím do prázdného místa přidávat nové, resp. po klepnutí na vrchol pravým knoflíkem myši a volbě z menu stávající vrchol vynechat. Po návratu do výpočetního režimu lze určit a zobrazit minimální kostru změněné množiny vrcholů.

Zkuste si měnit soubor vrcholů a sledovat, jak se mění jejich optimální propojení.

Scéna: *Kostra v metrickém grafu*

Je opět dána množina vrcholů, ale některé dvojice jsou propojeny modrými spoji, kterým říkáme hrany. Opět hledáme minimální propojení vrcholů (bude nakresleno zeleně), ale nyní je povoleno propojovat jen dvojice spojené modrými hranami.

Je zřejmé, že pro existenci řešení je nutné, aby výchozí graf byl souvislý. Pokud graf není souvislý, Algovize protestuje a je nutno přepnout do editačního módu a buď přidat hrany nebo vynechat izolované vrcholy. Kostra popsaná v předchozí scéně je speciální případ kostry grafu, kde zvolený graf je úplný.

Klikněte na [**Krok**] a Algovize ukáže řešení. Hrany grafu jsou úzké, hrany zvolené jako propojení jsou zdůrazněny podkladovým pásem zelené barvy. Knoflík [**Zpět**] řešení zase vymaže.

V editačním módu je možno přidávat, odebírat a pohybovat vrcholy jako v minulé scéně. Pokud je odebrán vrchol, odeberou se i všechny s ním incidentní hrany.

Je také možno přidat hranu tak, že se klikne na jeden z vrcholů a pak znovu klikne nad jiným (myš přitom tahne volný konec přidávané hrany). Tím se přidá nová hrana, ale současně se také zahájí proces přidávání další hrany vycházející z posledního vrcholu, který lze mnohokrát opakovat. Proces lze ukončit kliknutím do prázdného místa.

Lze také odebrat hranu tím, že se na ní klikne pravým knoflíkem myši; Algovize navrhne odebrání a provede jej, pokud je potvrzeno.

Scéna: *Kostra v ohodnoceném grafu*

Scéna je podobná předchozí, ale ceny hran grafu nejsou dány Euklidovskou vzdáleností koncových bodů, ale obecně kladnými čísly, připsanými u hran. Toto je nejobecnější případ; v praktických aplikacích znamená, že cena není dána prostou délkou spojení, ale i například charakterem terénu (např. telefonní kabel v rovné krajině, horách nebo na mořském dně) nebo způsobem provedení propojení.

Při přidávání nových hran je cena hrany zvolena náhodně mezi 1 až 99, ale dá se změnit (po klepnutí na hranu pravým knoflíkem myši a volbě ve vyskakovacím menu) v okénku, které se objeví na ovladači. V appletu musí být cena hrany celé číslo. Nulu a záporná čísla nepřipouštíme, protože by to změnilo charakter problému; omezení na celá čísla není na újmu obecnosti, ale z hlediska grafické úpravy je výhodné.

Hrajte si s množinou vrcholů, hranami a jejich cenami a sledujte, jak se tím mění minimální kostra grafu.

Scéna: *Obecný algoritmus*

Tato scéna ukazuje obecné schéma určování minimální kostry. Schéma připouští velkou volnost v lichých krocích výpočtu a konkrétní algoritmy pak tuto volnost omezují, vedeny obvykle snahou o jednoduchost nebo rychlost výpočtu. Libovolný výpočet spadající do schématu pochopitelně dává správný výsledek.

V této scéně předpokládáme cenu hran danou Euklidovskou délkou, případ obecných cen bude probrán dále.

Graf lze upravovat stejným způsobem jako bylo uvedeno výše, změna ale resetuje výpočet.

Algoritmické schéma je velmi jednoduché a pracuje tak, že do původně prázdného grafu přidává postupně hrany tak, aby nevznikl cyklus a výpočet se ukončí v okamžiku, kdy se vytvářený graf stane souvislým, neboli když se z něho stane strom.

V průběhu výpočtu je tedy neúplná kostra nesouvislý graf bez cyklů (les), který se rozpadá do řady komponent, které jsou stromy (souvislé grafy bez cyklů). Na začátku výpočtu jsou komponenty tvořeny jednotlivými vrcholy a přidáním každé hrany se dvě komponenty spojí v jednu a tedy se jejich počet sníží o 1. Množinu hran, které jsme do vytvářené neúplné kostry již přidali budeme nazývat *výpočetní les.*

Pro odlišení komponent se také používá barevné označení vrcholů. Na začátku mají všechny vrcholy (představující různé jednovrcholové komponenty) různé barvy. Při spojení dvou komponent vrcholy jedné komponenty přijmou barvu vrcholů druhé z nich, takže vrcholy vzniklé komponenty budou zase mít stejnou barvu.

Jelikož tedy je na začátku N jednovrcholových komponent a na konci vše splyne v jednu, spočívá výpočet v $N-1$ opakováních následujících dvou kroků:

Krok: *Volba komponenty*

Algoritmus zvolí *libovolnou* komponentu $\mathcal{C}$ výpočetního lesa.

Krok: *Volba hrany*

Mezi dvojicemi (v, w), kde v je vrchol ve zvolené komponentě $\mathcal{C}$ a w vrchol mimo ni, vybereme dvojici s nejmenší vzdáleností a přidáme ji do výpočetního lesa.

Volba komponenty tedy poskytuje úplnou volnost, volba propojované dvojice naopak je zcela vynucená. Pouze v případě, že by existovalo několik dvojic stejné minimální vzdálenosti, je povoleno zvolit libovolnou z nich.

Pro jednoduchost Algovize provádí volbu komponenty sama a to zcela náhodně.

Později ukážeme, že známé algoritmy pro hledání minimální kostry jsou speciálními implementacemi tohoto obecného schématu se speciálním způsobem volby komponenty. Kromě toho dokážeme, že *libovolný* postup podle schématu vede k nalezení optimální kostry.

V appletu algoritmus volí komponentu náhodně po stisknutí knoflíku [**Krok**] a její vrcholy zvýrazní červenou barvou. Následující 2 stisknutí knoflíku [**Krok**] ukazují volbu dvojice minimální vzdálenosti spojující vrcholy zvolené komponenty s oblastí mimo ni.

Pro znázornění určování dvojice s minimální cenou se nejprve po stisknutí knoflíku [**Krok**] kolem vrcholů zvolené komponenty rozšiřují okolí, která mají všechna stejný poloměr, který se s časem zvětšuje. Jakmile se první bod některého z těchto okolí dotkne vrcholu ležícího mimo zvolenou komponentu, je nalezena nejkratší spojnice zvolené komponenty se zbývajícími vrcholy grafu.

Po dalším stisknutí knoflíku [**Krok**] se nalezená dvojice propojí zelenou hranou.

Scéna: *Metrika L1*

Scéna opět ukazuje výpočet kostry, ale vzdálenost dvou bodů není měřena Euklidovskou metrikou $\mathcal{L}_2$, nýbrž metrikou $\mathcal{L}_1$ (neboli součtovou metrikou), ve které je cena propojení dvou bodů se souřadnicemi $[x_1, y_1]$ a $[x_2, y_2]$ rovna

$$|x_1 - x_2| + |y_1 - y_2|.$$

U této metriky nejsou okolí vrcholů kružnice, ale čtverce postavené na koso.

Scéna: *Metrika Lmax*

Tato scéna je zase jako předchozí dvě, ale metrika, určující vzdálenost dvou bodů je tak zvaná maximová metrika $\mathcal{L}_{max}$ nebo $\mathcal{L}_\infty$, ve které je cena propojení dvou bodů se souřadnicemi $[x_1, y_1]$ a $[x_2, y_2]$ rovna

$$\max(|x_1 - x_2|, |y_1 - y_2|).$$

U této metriky nejsou okolí vrcholů kružnice, ale čtverce sedící na jedné své straně.

Scéna: *Metrický graf*

Tato scéna opakuje předchozí, cena hrany je opět její Euklidovská délka, ale propojující strom musí být zvolen z modrých hran výchozího grafu stejně jako v druhé scéně tohoto appletu.

Okolí vrcholů nejsou ukázána úplně, ale je jen znázorněno, kam až by dosahovala v nakreslených hranách. Nejbližší soused z jiné komponenty je nalezen, když první růžová část některé hrany dorazí k vrcholu, který nepatří do zvolené komponenty.

Scéna: *Ohodnocený graf*

Tato scéna opakuje scénu předchozí, cena hrany je ale dána číslem zapsaným u hrany stejně jako v třetí scéně tohoto appletu.

Pro znázornění okolí se opět používají růžové segmenty. Jejich délka ale roste v různých hranách různou rychlostí, která je nepřímo úměrná ceně hrany. Znamená to, že délka růžového segmentu hrany s cenou c je k/c, kde hodnota k je stejná pro všechny hrany a roste s časem.

Nejbližší soused z jiné komponenty, nyní vzhledem k obecným cenám hran, je opět nalezen, když první růžová část některé hrany dorazí k vrcholu, který nepatří do zvolené komponenty.

Scéna: *Jarník - Prim*

Tato scéna ukazuje algoritmus popsaný Jarníkem v roce 1930 a znovu objevený Primem v roce 1957, který je jednoduchou variantou obecného schématu. Na začátku se zvolí zcela libovolně vrchol v_0 a za komponentu $\mathcal{C}$ se v obecném schématu volí vždy komponenta obsahující vrchol v_0.

V průběhu výpočtu proto neustále narůstá jedna komponenta, zatímco ostatní zůstávají jednovrcholové a jsou "požírány" komponentou obsahující zvolený vrchol.

Scéna je plně funkční co do volby výchozího grafu i metriky a povolených hran. Existují tři základní volby: [**Ohodnocený graf**] přináší omezení v podobě povinných modrých hran s obecným číselným ohodnocením, [**Metrický graf**] má také povinné hrany, ale s metrickým ohodnocením a [**Úplný metrický graf**] nemá žádná omezení volby hran a jejich cena je určena také metrikou. Ve druhém a třetím případe je možno zvolit libovolnou z metrik L_2, L_1 a L_{max}. Je také pochopitelně možno měnit množinu vrcholů a, pokud to přichází v úvahu, i hrany a jejich ceny.

Efektivní implementace Jarník - Primova algoritmu vyžaduje vhodný způsob volby minimální hrany vycházející z narůstající komponenty obsahující vrchol v_0. Výhodně se dá provést pomocí prioritní fronty, do které umístíme hrany vycházející z této komponenty s uvážením jejich ohodnocení. Když je do velké komponenty přidán nový vrchol v, hrany vedoucí z v do vrcholů, které již v komponentě jsou, se z prioritní fronty vynechají a hrany z v do vrcholů, které ještě v komponentě nejsou, se naopak přidají.

Scéna: *Kruskal*

Kruskalův algoritmus na první pohled nespadá pod obecné schéma; je i poněkud odlišně vizualizován. Později však zdůvodníme, že i ten je implementací obecného schématu. Nyní ale si algoritmus vysvětlíme a to pro případ grafu s hranami ohodnocenými jejich Euklidovskou délkou (počáteční nastavení) a nebo obecným ohodnocením.

Na počátku výpočetní les neobsahuje žádné hrany. Algoritmus probírá hrany v pořadí narůstajících vah od nejmenší po největší. Pro každou hranu určí, zda by její přidání do výpočetního lesa vytvořilo cyklus. Pokud nevytvoří, hrana se do výpočetního lesa přidá, jinak se vyhodí.

Pokud je na ovladači zvolen malý krok, probíraná hrana se vždy označí červeně; pokud je přidána do výpočetního lesa, dostane v dalším kroku zpět svoji původní barvu a navíc zelený podklad, zatímco pokud by uzavírala cyklus, v následujícím kroku zažloutne, aby byla vizuálně potlačena. Při delším kroku se zčervenání přeskakuje.

Scéna: *Kruskal ještě jednou*

Kruskalův algoritmus je možno přeformulovat takto: vychází se z grafu bez hran a dokud výpočetní les není souvislý (neboli dokud nemá $N-1$ hran, kde N je počet vrcholů), provádí se opakovaně provádí následující akce:

- mezi hranami spojující dvě *různé* komponenty výpočetního lesa se najde hrana s *minimální* vahou, a přidá se do výpočetního lesa.

V této scéně zkoumáme pro jednoduchost případ metrické konfigurace bez zakázaných hran. Zde se dvojice vrcholů z různých komponent s minimální vahou (t.j. délkou) nalezne tak, že ze *všech* vrcholů všech komponent začnou rozšiřovat okolí stejného s časem se zvětšujícího poloměru. (Tvar okolí je opět dán zvolenou metrikou). Okolí mají barvu odvozenou od barvy svých určujících vrcholů, což umožní snadno odlišit okolí dvou vrcholů z různých komponent.

Dotek nebo překrývání okolí vrcholů ze stejné komponenty ponecháváme bez povšimnutí, ale animace se zastaví v okamžiku, kdy se dotknou dvě okolí z různých komponent, tedy dvě okolí různé barvy, protože v tomto případě spojnice vrcholů v jejich středech je hledaná minimální spojnice dvou různých komponent.

Zkuste si výpočet podle Kruskalova algoritmu pro různé množiny vrcholů a metriky. Kruskalův algoritmus nedává uživateli žádnou volnost (s vyjímkou případu, kdy existují dvě nebo více hran spojujících různé komponenty, které mají *stejné* minimální váhy - v tomto případě lze volit libovolnou z nich; naše animace by to provedla náhodně sama, aniž by se dotazovala uživatele).

I Kruskalův algoritmus je implementací výše uvedeného obecného schématu: krok volby komponenty se provede tak, že se nalezne spojnice h dvou různých komponent s minimální vahou a za komponentu $\mathcal{C}$ se zvolí komponenta obsahující jeden (libovolně vybraný) ze dvou konců hrany h. Jelikož h je hrana s minimální vahou mezi hranami spojujícími různé komponenty, je i hranou s minimální vahou mezi hranami spojujícími komponentu $\mathcal{C}$ s jinou komponentou a proto musí (nebo může - v případě existence více minimálních hran tohoto typu) být v následujícím kroku volby hrany obecným schématem zvolena jako propojení zvolené komponenty $\mathcal{C}$ s jinou komponentou.

Efektivní implementace Kruskalova algoritmu vyžaduje počáteční setřídění hran podle jejich vah a dále rozhodnutí, zda dva zvolené vrcholy (konce probírané hrany) leží nebo neleží ve stejné komponentě výpočetního lesa. První problém se snadno vyřeší volbou některého z algoritmů probíraných v kapitole o třídění, pro druhý se výhodně použije faktorová množina, popisovaná v kapitole o datových strukturách.

Scéna: *Správnost*

V této scéně ukážeme, že obecné schéma dává *vždy* správné řešení, tedy minimální kostru zvoleného hranově ohodnoceného grafu. Klíčem je dokázat, že po celou dobu výpočtu platí následující podmínka:

I: existuje alespoň jedna minimální kostra grafu, která v sobě obsahuje všechny hrany výpočetního lesa (tedy hrany označené v appletu zelenou barvou).

Tvrzení (I) zjevně platí z triviálních důvodů na začátku výpočtu, kdy výpočetní les neobsahuje žádné hrany. Pokud tvrzení (I) platí i na konci výpočtu, pak je výpočetní les minimální kostrou. Na konci výpočtu je totiž výpočetní les souvislý, obsahuje proto alespoň $N-1$ hran, ale zároveň je obsažen v nějaké minimální kostře $\mathcal{K}$, a ta má také $N-1$ hran. Proto výpočetní les musí být *roven* minimální kostře $\mathcal{K}$.

Stačí tedy ukázat, že přidání hrany, zvolené obecným schématem, do vytvářené kostry výše uvedenou podmínku (I) neporuší. Abychom ukázali, proč tomu tak je, přidáme do průběhu krokování vždy za volbu hrany několik dalších kroků, které sice vytvářenou kostru nemění, ale objasňují logické vazby v grafu.

Přidaná hrana zůstává po dobu přidaných kroků zbarvena červeně, tedy odlišně od starších hran ve vytvářené kostře (které jsou zelené) a budeme ukazovat, co by se stalo, kdyby její přidání porušilo podmínku (I), tedy pokud by žádná minimální kostra obsahující staré zelené hrany vytvářené kostry neobsahovala nově přidanou červenou hranu.

Také se dočasně ponechává původní barevné označení vrcholů komponent, které nově přidaná hrana spojuje. Jejich barvy se sjednotí až po provedení kroků důkazu.

Krok: *Označení zvolené komponenty*

Komponentu, která byla naposledy zvolena v kroku volby komponenty budeme dále označovat jako $\mathcal{C}$ a její vrcholy zvýrazníme bílým podkladem.

Krok: *Alternativní kostra*

Jelikož předpokládáme, že podmínka (I) platila, ale po přidání červené hrany neplatí, měla by existovat minimální kostra, obsahující všechny zelené hrany, ale ne hranu červenou.

V druhém vloženém kroku se vykreslí kostra, která obsahuje všechny starší (zelené) hrany vytvářené kostry, další hrany nakreslené žlutě, ale ne nově přidanou červenou hranu. [1] Ukážeme, že předpoklad, že kostra tvořená zelenými a žlutými hranami je menší než libovolná kostra obsahující v daném okamžiku nový výpočetní les (tedy zelené hrany plus červenou hranu) vede ke sporu a tím dokážeme, že ve skutečnosti žádný krok obecného algoritmu nemůže podmínku (I) porušit.

Krok: *Alternativní cesta*

Konce červené hrany musejí být v žluto-zelené kostře spojeny cestou, kterou označíme π; pochopitelně nejsou spojeny přímo, protože červená hrana do žluto-zelené kostry nepatří. V tomto kroku je cesta π znázorněna tak, že její hrany jsou ovroubeny světle fialovou barvou.

Krok: *Nalezení můstku*

Jeden z koncových vrcholů červené hrany leží v komponentě $\mathcal{C}$, tedy je označen bílým podkladem a druhý konec červené hrany v $\mathcal{C}$ neleží. Barevný podklad hran původně světle fialově zvýrazněné cesty v žluté kostře, spojující konce červené hrany, se přebarví následujícím způsobem:
Barva první hrany, která má ještě jeden konec v $\mathcal{C}$, ale druhý již mimo $\mathcal{C}$, se změní na černou. Takováto hrana, které budeme říkat *můstek*, zjevně musí existovat. Pokud jsou v cestě hrany předcházející můstek, změní se navíc jejich barva ze světle na tmavě zelenou. Barva ostatních hran cesty se nezmění.

Krok: *Vynechání můstku*

Černý můstek i červená hrana jsou dvě hrany, které mají jeden konec v komponentě $\mathcal{C}$ (jejíž vrcholy jsou označeny bílým podkladovým kolečkem) a druhý konec mimo $\mathcal{C}$. Nově přidaná červená hrana byla volena tak, že má minimální ohodnocení mezi hranami spojujícími $\mathcal{C}$ s jinou komponentou. Proto musí být cena černého můstku větší nebo rovna ceně nově přidané červené hrany.

V tomto kroku se černý můstek ze žluto-zelené kostry vynechá a přidá se do ní červená hrana, čímž vznikne nová kostra grafu. Nově vzniklý soubor skutečně kostru vytváří: přidáním červené hrany se vytvoří jediný cykl, tvořený červenou hranou a hranami cesty π, který se ale přeruší vynecháním černého můstku. Vynechání černého můstku neporuší souvislost - co bylo spojeno přes černý můstek, je nyní propojeno červenou hranou spolu s hranami cesty π, které v kostře zůstaly.

Je jasné, že součet vah hran nové kostry s červenou hranou je buď menší nebo roven součtu vah hran původní žluté kostry s černým můstkem. V prvním případě tedy původní červená kostra nebyla minimální (nalezli jsme kostru, která je lepší) a v druhém případě jsme určili kostru, která je stejně dobrá jako původní žluto-zelená kostra, ale obsahuje červenou hranu. V obou případech dostáváme spor s původním předpokladem neplatnosti podmínky (I).

[1] Algoritmus vykreslí nejmenší kostru, obsahující zelené hrany předchozího výpočetního lesa, ale neobsahující červenou hranu, která byla nově do výpočetního lesa přidána