Grafové algoritmy

Peter „πtr“ Korcsok

Abstrakt. Viacero matematických problémov je možné previesť na niektorú z grafových úloh, pre ktorú už existuje mnoho spôsobov, ako ju úspešne vyriešiť. Tento príspevok predstavuje 4 základné grafové algoritmy v ich najzákladnejších podobách, aby boli zrozumiteľné aj pre ľudí, ktorí sa nepohybujú v informatickej oblasti.

Pri skúmaní matematiky môžeme naraziť na množstvo problémov z úplne odlišných oblastí. Často ale stačí trochu abstrakcie a vieme ich previesť na niektorú z grafových úloh, ktoré potom podstatne jednoduchšie vyriešime, ak vieme ako na to. Cieľom tohto príspevku je predviesť niekoľko základných metód, ako tieto zadania úspešne zdolať.

Na začiatok by asi bolo vhodné vysvetliť, čo vlastne slová z názvu znamenajú:

Definícia. *Grafom* G nazveme dvojicu (V, E), kde $E \subseteq \binom{V}{2}$.[1] Prvky množín V a E potom budeme nazývať *vrcholmi* prípadne *hranami* grafu G.

Definícia. *Algoritmus* je konečná postupnosť dobre definovaných inštrukcií na splnenie určitej úlohy.

Ďalej sa nám bude hodiť poznať niekoľko pojmov zo sveta grafov, preto si ich radšej hneď pripomeňme:

Definícia. *Cestou* v grafe $G = (V, E)$ rozumieme postupnosť v_1, e_1, v_2, …, v_{n-1}, e_{n-1}, v_n, kde $e_i = (v_i, v_{i+1}) \in E$ $(\forall i = 1, \ldots, n-1)$ a jednotlivé vrcholy v_i ani hrany e_i sa neopakujú.

Definícia. *Kružnicou* v grafe $G = (V, E)$ budeme rozumieť postupnosť v_1, e_1, v_2, …, v_{n-1}, e_{n-1}, v_n, e_n, v_1, kde $e_i = (v_i, v_{i+1}) \in E$ $(\forall i = 1, \ldots, n-1)$, $e_n = (v_n, v_1) \in E$ a jednotlivé vrcholy v_i ani hrany e_i sa neopakujú.

Definícia. Graf $G = (V, E)$ nazveme *súvislým*, pokiaľ medzi každými dvoma vrcholmi $u, v \in V$ vedie cesta.

Definícia. Pre súvislý graf $G = (V, E)$ definujeme jeho *kostru* ako súvislý graf $K = (V, E')$, ktorý neobsahuje žiadnu kružnicu a zároveň $E' \subseteq E$.

Kľúčové slová. graf, algoritmus, prehľadávanie do hĺbky, prehľadávanie do šírky, najkratšia cesta, hľadanie kružníc, Dijkstrov algoritmus, minimálna kostra, Jarníkov algoritmus

[1] Symbolom $\binom{X}{k}$ značíme množinu všetkých k-prvkových podmnožín množiny X.

Definícia. *Ohodnotenie* grafu $G = (V, E)$ je ľubovoľné zobrazenie $c\colon E \to \mathbb{R}$, takto ohodnotený graf budeme značiť (V, E, c).

Ešte si zaveďme niekoľko značiek, ktoré nám potom sprehľadnia a skrátia zápis algoritmov:

Značenie.

- Symbol $x := y$ označuje priradenie hodnoty y do premennej x.
- Symbolom $x \to F$ budeme značiť pridanie hodnoty x na koniec zoznamu F.
- A nakoniec $F \to x$ bude predstavovať výber prvej hodnoty zo zoznamu F a jej dosadenie do premennej x.

Pretože už vieme všetky potrebné pojmy, nič nám nestojí v ceste k našim algoritmom. Začnime dvoma spôsobmi, ako môžeme taký graf vôbec prejsť, aby sme zbytočne neprechádzali ten istý vrchol viackrát.

Prehľadávanie grafu do hĺbky

Prvá možnosť je tzv. *prehľadávanie do hĺbky* (angl. *Depth First Search*, skrátene *DFS*). Hlavnou ideou tohto algoritmu je začať v jednom vrchole, z neho prejsť na nenavštíveného suseda, odtiaľ na ďalšieho suseda, ... a to celé až dovtedy, kým sa nám neprejdení susedia „neminú“. Vtedy sa vrátime k niektorému z predchádzajúcich vrcholov a skúšame ďalej.

Vstupom do tohto algoritmu by mal byť súvislý graf (V, E) a nejaký jeho vrchol $v \in V$, z ktorého chceme prehľadávanie spustiť. Výstupom potom bude napríklad usporiadanie množiny V v poradí, v akom boli prvky navštívené.

Algoritmus. (DFS)

1. označ v za navštívený, ostatné vrcholy za nenavštívené, $R := \{v\}$
2. ak $|R| = |V|$, vráť R a skonči
3. ak existuje nenavštívený sused v, označ ho ako u a navštívený, $u \to R$, $r(u) := v$, $v := u$ a prejdi na 2. krok
4. ak všetci susedia v sú už navštívení, $v := r(v)$ a prejdi na 3. krok

Pomocou DFS vieme napríklad nájsť nejakú kostru zadaného hrafu, nájsť v ňom kružnice a ak vynecháme podmienku súvislosti vstupného grafu (a mierne upravíme 2. krok), preskúmané vrcholy budú tvoriť práve tzv. *komponentu súvislosti* obsahujúcu vrchol v, teda najväčšiu súvislú časť grafu, v ktorej sa vrchol v nachádza.

Miernymi úpravami tohto algoritmu dostaneme pomerne efektívny spôsob na hľadanie cesty v labyrinte, prípadne dokonca vytváranie rôzne komplikovaných bludísk.

Prehľadávanie grafu do šírky

Druhou možnosťou je potom tzv. *prehľadávanie do šírky* (angl. *Breadth First Search*, skrátene *BFS*). Podobne ako pri DFS, aj tu začíname v jednom vrchole, aku-

rát z neho prejdeme najprv všetkých jeho susedov, potom susedov susedov a takto pokračujeme, kým nemáme prejdený celý graf (alebo jeho komponentu).

Vstupom algoritmu je opäť súvislý graf (V, E) a štartovací bod $v \in V$, výstupom je znovu poradie prechodu vrcholov.

Algoritmus. (BFS)

1. označ v za navštívený, ostatné vrcholy za nenavštívené, $F := \{v\}$, $R := \emptyset$
2. ak $|R| = |V|$, vráť R a skonči
3. ak F nie je prázdne, $F \to v$, $v \to R$
4. ak existuje nenavštívený sused v, označ ho ako u a navštívený, $u \to F$ a prejdi znovu na 4. krok, inak prejdi na 3. krok

BFS prechádza vrcholy podľa vzrastajúcej vzdialenosti (na počet hrán) od počiatku v, takže ho môžeme použiť napríklad, keď potrebujeme nájsť najkratšiu cestu medzi dvoma bodmi. Podobne ako DFS, aj tento algoritmus môžeme upraviť tak, aby bol schopný bežať na nesúvislých grafoch – opäť preskúma práve vrcholy z jednej komponenty.

Hľadanie najkratšej cesty

Základnú myšlienku BFS môžeme rozšíriť aj na ohodnotené grafy a vzdialenosť počítanú ako minimálny súčet hodnôt hrán na ceste medzi dvoma bodmi. Na nájdenie tej najkratšej máme hneď niekoľko možných spôsobov, najjednoduchší z nich je asi *Dijkstrov algoritmus*. Znovu vyjdeme z jedného vrcholu v, ktorého vzdialenosť od seba samého je 0, vzdialenosti ostatných bodov ale zatiaľ nepoznáme. V každom kroku si potom zvolíme najbližší nespracovaný vrchol a všetkým jeho susedom upravíme vzdialenosť od v.

Tu si už s obyčajným grafom nevystačíme, budeme potrebovať, aby bol ohodnotený a dokonca všetky hrany musia mať nezápornú hodnotu (inak si v grafe narobíme neporiadok a to predsa nechceme :-)), po úspešnom priebehu ale budeme poznať najkratšiu vzdialenosť z počiatočného vrcholu v do všetkých ostatných.

Algoritmus. (Dijkstrov)

1. označ všetky vrcholy za nenavštívené, $d(v) := 0$, $d(u) := \infty$ $(\forall u \neq v)$
2. ak neexistuje vrchol v s $d(v) < \infty$, vráť d a skonči
3. za v označ nenavštívený vrchol s minimálnou hodnotou $d(v)$, označ v za navštívený
4. všetkým nenavštíveným susedom u vrcholu v nastav $d(u) := \min\{d(v) + c(v, u),\ d(u)\}$ a prejdi na 2. krok

Pomocou Dijkstrovho algoritmu vieme riešiť množstvo úloh, ktoré je možné previesť na problém najkratšej cesty (konkrétne príklady si spomenieme na prednáške). Na riešenie tohto problému samozrejme existujú aj iné algoritmy, viaceré si dokonca vedia poradiť aj so zápornými hranami, vo väčšine prípadov nám ale ten Dijkstrov dostatočne poslúži.

Minimálna kostra

Poslednou oblasťou grafových úloh, ktorej sa budeme venovať, je problematika kostier. Už sme si uviedli jeden spôsob, ako ľubovoľnú z nich nájsť – DFS. Pretože ale v dnešnom svete je všetko príliš drahé, skúsme rovno nájsť kostru, ktorej celková cena za všetky vybrané hrany bude najnižšia možná. Aj na toto máme viacero možných algoritmov – napríklad *Jarníkov*[2], *Borůvkov* alebo *Kruskalov*, všetky tri sa radia medzi tzv. „pažravé“[3] algoritmy.

Pozrime sa teraz podrobnejšie na Jarníkov postup: jeho základnou ideou je budovanie jednej komponenty súvislosti (definícia je pri DFS), ku ktorej v každom kroku pridáme jeden vrchol a niektorú hranu z tohto bodu do už vytvorenej komponenty, pričom sa snažíme, aby pridaná hrana mala vždy najmenšiu váhu. Vstupom je opäť ohodnotený súvislý graf (V, E, c), výstupom niektorá z jeho minimálnych kostier.

Algoritmus. (Jarníkov)

1. zvoľ si niektorý vrchol v, $V' := \{v\}$, $E' := \emptyset$
2. ak $|V'| = |V|$, vráť (V', E') a skonči
3. z množiny hrán $\{(u, v); u \in V', v \in V \setminus V'\}$ vyber hranu $e = (u, v)$, že hodnota $c(e)$ je minimálna
4. $v \to V'$, $e \to E'$ a prejdi na 2. krok

Podmienku súvislosti vstupného grafu potrebujeme k tomu, aby sme nakoniec do kostry mohli pridať všetky vrcholy, v opačnom prípade by sme našli iba kostru komponenty, do ktorej patrí náhodne zvolený prvý vrchol. Tento algoritmus síce inde ako pri hľadaní kostry minimálnej váhy veľmi nevyužijeme, ale už len tým nám dokáže značne uľahčiť ďalšiu prácu na pôvodnom probléme. Obecne sú totiž stromy[4] (čo kostra je) na skúmanie jednoduchšie ako grafy s kružnicami.

Tým sa pomaly dostávame na koniec príspevku. Spomenuté algoritmy samozrejme nie sú jediné, často sa v praxi upravujú a kombinujú s ďalšími, aby sa prispôsobili špecifickým podmienkam zadania. Keby sme chceli aspoň okrajovo popísať všetky, tento text by musel byť podstatne dlhší.

Literatúra a zdroje

[1] Knižnica PraSiatka, *http://mks.mff.cuni.cz/library/*

[2] V anglickej literatúre je známy skôr ako *Primov*, ale pretože Jarník ho popísal 27 rokov pred Primom, myslím, že toto označenie si zaslúži.

[3] Po česky „hladové“, po anglicky „greedy“.

[4] *Strom* je súvislý graf bez kružníc.