Algebra a diskrétna matematika

doc. RNDr. Jana Šiagiová, PhD.

Prehľad zo 6. týždňa

Prehľadávacie algoritmy, minimálna kostra, eulerovské grafy, hamiltonovské kružnice

Prehľadávanie grafov

Často v grafoch chceme zistiť určitú vlastnosť (napr. súvislosť, priemer, maximálny stupeň, existenciu kružníc, atď). Zo zoznamu susedov alebo matice susednosti to nie je ľahké hneď identifikovať. Mnohé takéto úlohy si preto vyžadujú efektívne a systematické preskúmanie grafu, kedy postupne navštevujeme všetky vrcholy a hrany grafu.

Uvedieme neformálny opis dvoch najviac používaných metód podľa spôsobu prehľadávania.

Prehľadávanie grafu do hĺbky (depth-first search DFS) Hlavná myšlienka tohto prehľadávania je, že postupujeme cez susedov vrcholov tak hlboko, ako je možné.
Začneme prehľadávanie vo zvolenom vrchole, navštívime jedného jeho suseda, potom suseda tohto suseda, atď. Ak sa ďalej takto nedá pokračovať a ešte existujú nenavštívené vrcholy, tak postupujúc naspäť nájdeme prvý vrchol s ešte nenavštíveným susedom a opakujeme postup, až kým nepreskúmame všetky vrcholy.

Prehľadávanie grafu do šírky (breadth-first search BFS) V tomto prípade najprv skúmame zvolený vrchol a potom všetkých jeho susedov, potom všetkých susedov týchto susedov, atď, kým nenavštívime všetky vrcholy grafu (prípadne komponentu). Algoritmus prechádza vrcholy podľa vzrastajúcej vzdialenosti od zvoleného vrchola. Je preto vhodný na nájdenie najkratšej cesty medzi zvoleným vrcholom a ľubovoľným iným vrcholom.

V oboch algoritmoch každá hrana dostane šípku označujúcu smer jej prvého prechodu. Hrany rozdeľujeme na stromové, ak nás dovedú k novým vrcholom, a inak na spätné.

Výstupom oboch algoritmov je (DFS alebo BFS) kostra v každom komponente súvislosti a usporiadanie vrcholov v poradí, v akom boli navštívené.

Problém minimálnej kostry: Pre súvislý graf $G = (V, E)$ s kladným ohodnotením hrán w nájdite kostru $T = (V, E')$ grafu G s najmenšou možnou hodnotou $w(T)$.

Kruskalov algoritmus: Minimálnu kostru v grafe konštruujeme postupným pridávaním hrán s najmenšou váhou tak, aby sme nevytvorili žiaden cyklus.
Vstup: Súvislý hranovo ohodnotený graf $G = (V, E)$ s n vrcholmi

Výstup: Podmnožina hrán $A \subseteq E$ taká, že graf (V, A) je minimálna kostra grafu G.

Algoritmus

• Krok 1: Polož $A = \emptyset$.

• Krok 2: Polož $F = E$.

• Krok 3: Ak F=$\emptyset$ alebo graf (V, A) je strom, ukonči, inak choď na Krok 4.

• Krok 4: Odstráň z množiny F hranu e s minimálnym ohodnotením (ak ich je viac, začni ľubovoľnou z nich). Ak graf $(V, A \cup \{e\})$ neobsahuje kružnicu, pridaj hranu e do množiny A. Choď na Krok 3.

Eulerovské grafy

Úloha: Nakreslite daný graf jedným uzavretým ťahom bez zdvihnutia tužky z papiera, pričom žiadna hrana sa neobkreslí viackrát.

Uzavretý eulerovský ťah v grafe je uzavretý sled hrán a vrcholov, v ktorom sa každá hrana vyskytuje práve raz a každý vrchol aspoň raz.

Graf je **eulerovský** práve vtedy, keď má aspoň jeden uzavretý eulerovský ťah.

Charakterizácia eulerovských grafov: Graf je eulerovský práve vtedy, keď je súvislý a všetky jeho vrcholy sú párneho stupňa.

Mostom grafu $G = (V, E)$ je taká hrana $e \in E$, pre ktorú platí, že graf $G - e$ má väčší počet komponentov ako graf G.

Tvrdenie: Ak má graf všetky vrcholy párneho stupňa, tak neobsahuje most.

Pred opisom algoritmu na generovanie eulerovských ťahov najprv opíšeme procedúru Proc(H, u), ktorá v súvislom eulerovskom grafe H vygeneruje *nejaký* uzavretý ťah T začínajúci a končiaci vo vrchole u grafu H:

Proc(H, u): Vyberieme ľubovoľnú hranu $e_1 = uv_1$ v H (taká musí existovať) a vytvoríme graf H_1 vynechaním hrany e_1 z grafu H; symbolicky, $H_1 = H - e_1$. Postup iterujeme, t.j. vyberieme ľubovoľnú hranu $e_2 = v_1v_2$ (taká musí existovať, pretože stupeň vrchola v_1 v novom grafe H_1 je nepárny – a zároveň $e_2 \neq e_1$, pretože e_1 už nie je v H_1) a vytvoríme graf $H_2 = H_1 - e_2$. Takto pokračujeme, až kým znova "narazíme" na vrchol u. To sa v niektorom kroku *musí* stať – t.j. niekde v i-tom kroku sa dostaneme do situácie, kedy v práve navštívenom vrchole v_{i-1} vyberieme hranu $e_i = v_{i-1}u$ "končiacu" vo vrchole u. Zostrojený *uzavretý* ťah $ue_1v_1e_2v_2 \dots v_{i-1}e_iu$ označíme symbolom T; ten bude tvorit' výstup našej procedúry.

Algoritmus vygenerovania eulerovského ťahu v eulerovskom grafe

Vstup: eulerovský graf $G = (V, E)$
Výstup: eulerovský ťah T
Algoritmus:

- Krok 1: Polož $H = G$.
- Krok 2: Polož $T = \emptyset$.
- Krok 3: Spusť Proc$(\tilde{H}, u)$ v nejakom súvislom komponente $\tilde{H}$ grafu H v nejakom vrchole $u \in H \cap T$ (ak $T = \emptyset$, $u \in H$), výstup označ T'
- Krok 4: Za T polož $T \cup T'$; t.j. ak $T = v_0e_1v_1 \dots e_rue_{r+1} \dots e_iv_0$ a $T' = uf_1 \dots f_su$, tak nové $T = v_0e_1v_1 \dots e_ruf_1 \dots f_sue_{r+1} \dots e_iv_0$.
- Krok 5: Za H polož $H - E(T')$.
- Krok 6: Ak $H = \emptyset$, ukonč, inak choď na Krok 3.

Hovoríme, že graf G s párnym počtom vrcholov $2n$ má **perfektné párovanie**, ak G obsahuje n nezávislých hrán (žiadne 2 nemajúce spoločný vrchol).

Veta (König, 1916): Každý pravidelný bipartitný graf s aspoň jednou hranou má perfektné párovanie.

Veta (Petersen, 1891) Každý pravidelný graf stupňa 3 bez mostov má perfektné párovanie. (Tu graf nemusí byť bipartitný!)

Problém obchodného cestujúceho
Nech v grafe G vrcholy reprezentujú mestá a hrany cesty medzi mestami. Problém obchodného cestujúceho spočíva v určení trasy začínajúcej a končiacej v tom istom meste, pričom cestujúci každé iné mesto navštívi práve raz a súčet prejdených vzdialeností (resp. nákladov, resp. časov) bude minimálny možný.

Kružnica v G je **hamiltonovská**, ak obsahuje všetky vrcholy grafu G.

Ekvivalentná formulácia problému obchodného cestujúceho: Nájsť v grafe G s kladne ohodnotenými hranami hamiltonovskú kružnicu s najmenším celkovým ohodnotením.

Algoritmy na hľadanie optimálnej hamiltonovskej kružnice sú veľmi komplikované. Nateraz sa uspokojíme s **heuristikou**, t.j. procedúrou, ktorá nájde hamiltonovskú kružnicu s "rozumne malým" ohodnotením.

Budeme uvažovať o ohodnotenom *úplnom* grafe G (aby sme sa vyhli problému existencie hamiltonovskej kružnice), pričom pre (kladné) ohodnotenie w hrán predpokladáme pre každú hranu uv **trojuholníkovú nerovnosť:**
$w(uv) \leq w(ux) + w(xv)$, pre každý vrchol x.

Heuristika zdvojenej kostry.

- Pomocou Kruskalovho algoritmu nájdeme v G najlacnejšiu kostru T.
- Na T vytvoríme uzavretý sled S, ktorý každú hranu T prejde 2-krát.
- Pomocou S budujeme kružnicu C z ľubovoľného vrchola u rekurzívne:

1. Vydáme sa z u v smere sledu S, až kým nie sme nútení prejsť nejakou hranou v opačnom smere, t.j. keď $S = u...vwv....$ Vezmeme dočasne $C = u...vw$.

2. Pokračujeme v traverzovaní sledu S, tentoraz z vrchola w, až po prvý výskyt nejakého vrchola x (čiže teraz $S = u...vwv...x...$), ktorý nie je v našej dočasnej C; do C pridáme vrchol x a hranu wx. Ak taký x nie je, tak bola prejdená celá trasa S; do C pridáme hranu wu a skončíme.

3. Ak $S = u...vwv...xy...$ a $y \notin C$, budujeme ďalej C ako v 1 (x preberie rolu vrchola u v bode 1); ak $y \in C$ (a teda xy je prejdená druhýkrát), pokračujeme ako v 2 (x preberie rolu vrchola w v bode 2).

Z trojuholníkovej nerovnosti máme $w(C) \leq 2 \cdot w(T)$, čo považujeme za "rozumne malé".

Algoritmická zložitosť – neformálny výklad

Príklady algoritmických riešení problémov: Zostrojiť kostru daného grafu, alebo rozhodnúť, či daný graf je súvislý, alebo či je hamiltonovský.

V prvom prípade žiadame, aby výstupom algoritmu bol *objekt* – tu graf (kostra), zatiaľ čo v ďalších dvoch prípadoch výstupom algoritmu je len jedna z 2 možných odpovedí: ÁNO – NIE. Takýmto problémom hovoríme *rozhodovacie*.

Uvedomte si, že graf, v ktorom potrebujeme niečo nájsť alebo o ňom urobiť nejaké rozhodnutie, môže mať tisíce vrcholov, je na vstupe algoritmu v podobe napr. zoznamu vrcholov a ich susedov, a najmä to, že počítač ho nevidí!

Odteraz budeme uvažovať len rozhodovacie problémy. Opíšeme základný pojem zložitosti rozhodovacieho problému; na podanie presnej definície zatiaľ nemáme vybudovaný matematický aparát a dozviete sa ju neskôr v špeciálnych predmetoch.

Neformálne, pod **zložitosťou** algoritmického problému budeme rozumieť počet "krokov" $f(n)$, ktoré algoritmus v najhoršom prípade vykoná, aby na výstupe dal odpoveď ÁNO alebo NIE, ak "dĺžka vstupu je n".

"Krok"? Napríklad, v opise algoritmu na prehľadávanie grafu to môže byť inštrukcia 'vezmi ďalší vrchol (hranu) zo zoznamu". V podrobnejšom opise (pseudokóde) to môže byť séria pokynov typu:

(1) Pozri, či vrchol číslo i už bol navštívený.

(2) Ak áno, zvýš honotu i na $i + 1$ a vráť sa na (1).

(3) Ak nie, pridaj i do zoznamu prejdených vrcholov a choď na (5).

Atď...

"Krok" v opise algoritmu: $\leq$ *konštantne veľa* "krokov" v pseudokóde.

"Krok" v pseudokóde: $\leq$ *konštantne veľa* "krokov" v program. jazyku.

"Krok" v program. jazyku: $\leq$ *konštantne veľa* inštrukcií v stroj. kóde.

Inštrukcia v strojovom kóde: $\leq$ *konštantne veľa* taktov procesora.

Ak máme prehľadať napr. n-vrcholovú cestu a prejsť každý vrchol, tak:

v opise algoritmu musíme urobiť n "krokov";

v pseudokóde to bude viac, ale nie viac ako napr. $5 \cdot n$ krokov,

v strojovom kóde nie viac ako (povedzme) $4 \cdot 5 \cdot n$ krokov,

a celkove (povedzme) nie viac ako $5 \cdot 4 \cdot 5 \cdot n$ taktov.

Či už počet krokov meriame pomocou opisu algoritmu alebo pomocou taktov procesora, dostávame síce rôzne čísla – n a $100n$ – ale stále je to konštanta krát n, kde konštanta závisí od mašiny a implementácie.

V informatike túto situáciu zapíšeme symbolom $O(n)$. Ak teda $f(n)$ z opisu zložitosti je počet "krokov" nutných na prejdenie všetkých n vrcholov, tak by sme napísali $f(n) = O(n)$, čo formálne znamená, že $f(n) \leq cn$ pre nejakú konštantu c a pre všetky n. (V tej konštante je zahrnutá naša diskusia o počte krokov.)

Podobne budeme hľadieť na pojem "dĺžka vstupu". Či už je to zoznam vrcholov, ako ho máme na papieri, alebo prekódovaný na postupnosť núl a jednotiek, vždy v prípade napr. cesty na n vrcholoch môžme rovnako dobre povedať, že vstup má dĺžku $O(n)$.

Obvykle sa symbol O používa len na zložitosť, a nie na dĺžku vstupu (to je potom zahrnuté v "narábaní s multiplikatívnymi konštantami").

Matematika má na presné definície týchto pojmov prostriedky – tzv. Turingov stroj; opis jeho činnosti sa dozviete neskôr na špecializovaných prednáškach. Zatiaľ vystačíme s naznačenými intuitívnymi pojmami.

Príklad: Rozhodovací problém zistiť či graf s n vrcholmi a m hranami, zadaný na vstupe, je súvislý. Vzhľadom na našu diskusiu môžeme predpokladať, že dĺžka vstupu je $2m + n$. Počet krokov prehľadávacieho algoritmu je $O(2m + n)$, pretože každú hranu navštívime najviac dva razy.

Niekedy sa zložitosť udáva *len* v terminológii počtu vrcholov n grafu. Keďže $m \leq n(n-1)/2$, môžeme jednoducho povedať, že zložitosť prehľadávacieho algoritmu je nanajvýš $O(n^2)$. Podstatné je, že n^2 je *polynóm* v premennej n. Je to rastúca funkcia, ale nie "divoko" rastúca.

Algoritmy, ktoré na rozhodovacie problémy o grafoch s n vrcholmi dajú odpoveď ÁNO alebo NIE a spotrebujú pri tom najviac $O(n^k)$ krokov pre *konštantné* k, sa nazývajú **polynomiálne**, alebo **patriace do triedy** P.

Príklady: Určenie, či priemer grafu je $\leq d$ pre dané d, rozhodnutie, či graf má perfektné párovanie, alebo či daný graf je rovinný, atď.

Je však celý rad rozhodovacích problémov, na ktoré zatiaľ nepoznáme žiaden polynomiálny algoritmus.

Príklady: Rozhodnúť, či daný graf s n vrcholmi na vstupe je hamiltonovský, alebo či jeho chromatické číslo je $\leq \ell$ pre ľubovoľné *konštantné* $\ell \geq 3$.

Aj tie najlepšie známe algoritmy na tieto úlohy dokážu spracovať vstupný graf s n vrcholmi v najlepšom prípade až po $O(c^n)$ krokoch, kde $c > 1$

V informatike sa skúma celá trieda takýchto problémov, ktoré možno "rýchlo" – v polynomiálnom čase – pretransformovať jeden na druhý, a v konečnom dôsledku na rozhodnutie, či daný n-vrcholový graf na vstupe je hamiltonovský.

Tejto triede problémov sa hovorí **NP-úplné problémy**. Ich definícia vysoko presahuje túto prednášku (NP znamená *nedeterministické polynomiálne*).

Rozdiel medzi zložitosťou napr. $O(n^3)$ a $O(2^n)$: Ak napr. $n = 400$, tak (až na multiplikatívnu konštantu) porovnávame 400^3 s 2^{400}; to druhé je viac, ako fyzikmi odhadovaný počet elementárnych častíc vo vesmíre!

P-NP problém: Jeden z najslávnejších problémov v súčasnosti zo zoznamu 7 miléniových problémov Clayovho matematickéeho inštitútu, USA. Vyriešenie každého z nich je dotovaný miliónom USD! (Jeden z nich už je vyriešený.)

P-NP problém je otázka, či P=NP;
v ekvivalentnej formulácii, či existuje algoritmus polynomiálnej zložitosti na rozhodnutie, či ľubovoľný daný n-vrcholový graf je hamiltonovský. (Vzhľadom na ekvivalentnosť v triede NP-úplných problémov by kladná odpoveď znamenala existenciu polynomiálnych algoritmov pre *všetky* NP-úplné problémy.)

Verí sa, že P≠NP, ale nikto to nevie dokázať! Tu je významný problém *izomorfizmu* grafov: Rozhodnúť, či dva n-vrcholové grafy na vstupe sú izomorfné. Zatiaľ nepoznáme žiaden polynomiálny algoritmus na tento problém, ale na druhej strane ani nikto nevie dokázať, že je NP-úplný!

Predpokladá sa, že ide o problém, ktorý striktne medzi P a NP! Ak by to niekto dokázal, tak by zároveň vyriešil aj P-NP problém.