Seminář grafových algoritmů

2. ledna 2013

Obsah

1 Minimální kostry

Předpokládejme, že máme multigraf, váhy hran se dají porovnávat a jsou navzájem různé. Z toho plyne, že má právě jednu minimální kostru $mst(G)$. Dále předpokládejme, že je souvislý.

1.1 Značení

Nechť T je strom a u, v jsou vrcholy. Pak $T[u, v]$ je cesta stromem mezi nimi. Nechť $e = (u, v) \in E$, pak $T[e] = T[u, v]$. Hrany této cesty jsou pokryté hranou e.

Hrana e je T-***lehká*** $\Leftrightarrow \exists e' \in T[e]; w(e) > w(e')$. Hrana je T-***těžká*** $\Leftrightarrow \forall e' \in T[e]; w(e) < w(e')$.

Hrany $e \in T$ nejsou ani lehké ani těžké, zbylé musí být právě jedno z toho.

Pozorování 1 *Pokud je e T-lehká, pak $T \neq mst(G)$.*

Věta 1 *Kostra $T \subseteq G = mst(G) \Leftrightarrow \nexists$ T-lehká hrana.*

1.2 Základní poznatky

Lemma 1 (Modré) *Nechť C je řez grafu G. Nejlehčí hrana toho řezu $\in mst(G)$*

Lemma 2 (Červené) *Nechť C je cyklus v G. Nejtěžší hrana $e \in C \not\in mst(G)$.*

Silnější verze: je to ekvivalence.

Lemma 3 (Kontrakční) *Pokud najdu hranu ležící v minimální kostře, tu zkontrahuji najdu zbytek a rozkontrahuju, tak mám minimální kostru původního grafu.*

`TODO: Odvodit z předchozích`

Mějme $f_e : G-e \to g/e$ přirozenou projekci. Pokud $e \in mst(G) \to mst(G) = f^{-1}(mst(G/e)) + e$.

1.3 Základní algoritmy

Algoritmus 1 (Kontrahující borůvkův):

Najdu pro každý vrchol a najdu nejmenší hranu z každého. To mi vytvoří komponenty, ty zkontrahuji a pokračuji.

V každé fázi se počet vrcholů sníží alespoň 2·, dokážu fázi provést v čase $O(m_i)$. Vyházení paralelních hran stihnu v lineárním čase (přihrádkově je setřídím a jsou u sebe).

Z toho by šlo odhadnout $O(m \cdot \log n)$ nebo $O(n^2)$.

Pokud dokážeme omezit hustotu grafu i mezi kontrakcemi, tak je lineární.

☺

1.4 Verifikace minimální kostry

Chceme zjistit, jestli kostra je minimální a pokud ne, tak najít všechny T-lehké hrany.

Algoritmus 2 (Komlóšův):

Umí to na lineárně mnoho porovnání, sám lineární být nemusí.

Tedy, chceme najít všechny lehké hrany (z toho už se pozná, že je kostra minimální). Pokaždé nám stačí najít nejtěžší pokrytou hranu e a tu s tím porovnat. Tedy, chceme najít nejtěžší hrany těchto cest.

Při hledání by se hodilo, aby ten strom byl nějak rozumě vyvážený. Zkonstruujeme „borůvkový keř“ – spustí borůvkův algoritmus na kostru. Zajímá nás posloupnost mezivýsledků (doběhne to lineárně, je rovinný). Stavím strom, který popisuje, jak se spojují komponenty. Dolů dám původní vrcholy, vyšší patra jsou ty komponenty, hrany jsou ohodnoceny tím, kterou hranou se to spojovalo. Pokračuji, než mi zbude jeden vrchol.

Tento strom je hluboký maximálně $B(T) = \lceil \log n \rceil$. Každý vrchol stromu $v \in B(T)$ odpovídá nějaké komponentě $C(v)$ vzniklé během borůvkova algoritmu.

Lemma 4 *Váha nejtěžší hrany $T[x, y]$ je rovna váze nejtěžší hrany cesty $B(T)[x, y]$*

Důkaz:
$P := T[x, y], h = (a, b)$ nejtěžší na P. $P' := B(T)[x, y], h' = (a', b')$ nejtěžší na P'. Chceme dokázat, že $w(h) = w(h')$.

Nejdříve dokážu, že $\exists l' \in P'; w(l') = w(h) \Rightarrow w(h) \leq w(h')$. V borůvkovém keři jsou dole a, b. Existuje u nejnižší společný vrchol. Ten má syny v_a, v_b. $a \in C(v_a), b \in C(v_b)$. Alespoň jedna z hran z u dolů je původní h, tedy má stejnou váhu. BÚNO je to (u, v_a).

Nyní je třeba dokázat, že je na P'. Právě jeden z vrcholů x, y leží v komponentě $C(v_a)$. Dokážeme sporem. Kdyby to neplatilo – tak tam buď leží oba,

tak h není na této cestě, protože celá cesta je v $C(v_a)$. Nebo by tam neležel žádný, pak tamtudy cesta prochází a tato je nejtěžší. Pak jsem si ale musel vybrat tuhle hranu naposled z té cesty a tudíž jeden musel v té komponentě byl.

Nyní si vezměme libovolnou $(u, v) \in P'$. Nechť u je horní konec. Tato hrana leží na cestě mezi x, y v borůvkově keři, x leží pod v. Tedy, vede hrana q ven z $C(v)$ na cestě mezi x a y na T. A tuto hranu jsem si nevybral, vybral jsem si tuhle, tedy $w(u, v) \leq w(q)$.

☻

Budu počítat cesty jen zvrchu dolů a spojovat je dohromady v těchto vrcholech.

☺

Pro hranu e nadefinujeme Q_e, což je množina všech dotazů, které obsahují e (dotaz je na nejtěžší hranu té cesty). T_e jsou vršky cest z Q_e. Pamatuji si jako seznam, setříděný podle hloubek. Bez duplicit. P_e je seznam, jenž pro každý vršek $v \in T_e$ si pamatuje maximální váhu na cestě do dolního vrcholu e.

Tohle prohledávám do hloubky a pamatuji si, do kterého vrcholu jsme došli (y), poslední hranu (e) a T_e a P_e. Prostě spočítám ty maxima cest. Hodí se přimalovat šťopku nad kořen. Potom pro všechny hrany (yz) (potomky), zavolat rekurzivně. Potřebuji přepočítat T a P. Odečtu všechny co skončily nebo odbočily, přidám všechny, co mají horní hranu (yz). Každá další cesta je zkrácením některé předchozí, takže maxima se jen snižují. Proto mi stačí jednou projít P_e půlením intervalu a odtamtuď to přepíšu. Zavolám rekurzivně.

Lemma 5 *Pro předzpracovaný strom na n vrcholech a q dotazů algoritmus provede $O(n + q)$ porovnání.*

Důkaz:
Očíslujeme hladiny. Nechť n_i je počet vrcholů v i-té hladině. Strom je alespoň binární, tedy $n_i \leq \frac{n}{2^i}$. c_i bude počet porovnání při zpracování hran mezi $i+1$ a i-tou hladinou.

$$\begin{aligned} c_i &\leq \sum_e 1 + \log |T_e| \\ &\leq n_i + \sum_e \log |T_i| \\ &= n_i + n_i \cdot \frac{\sum_e \log |T_e|}{n_i} \end{aligned}$$

Logaritmus je konkávní, tedy můžu:

$$\begin{aligned} &\leq n_i + n_i \log \frac{q+n}{n_i} \\ &= n_i + n_i \cdot \log \frac{q+n}{+} n_i \cdot \log \frac{n}{n_i} \end{aligned}$$

Je potřeba to posčítat. Prý trochu analýzy a nehezkého počítání.

☻

V borůvkovi potřebujeme prořezávat hrany. Dosáhneme toho takto: občas si vybereme podgraf, tomu najdeme kostru. Hrany těžké k této určitě nebudou ve výsledku.

Lemma 6 (Karkerovo vzorkovací) *Je-li H podgraf G vzniklý vzorkováním hran (vezmu všechny vrcholy, u každé hrany si hodím mincí, jestli ji beru) s pravděpodobností p a T je minimální kostra H, pak počet hran z G, které nezahodíme bude malý vzhledem k počtu vrcholů – v průměru $\frac{n}{p}$.*

Důkaz:
Do kostry se mi dostane max. $n-1$ hran. Když už jsem $n-1$ přijal (házet můžu jen když přidávám do kostry), zbytek zahazuji. Neházím za hrany (kdybych bral třeba kruskala), které už zbyly na konci, tedy házím jen pro ty, které nejsou T-těžké.

☻

Algoritmus 3 (Karger-Klein-Tarjan):

- Odstraníme izolované vrcholy, pokud nic nezbylo, končíme.
- Provedeme 2 kroky kontrahujícího borůvky.
- Vybereme vzorkováním $H \subset G$.
- Vybereme T rekurzivně.
- Vyházeme T-těžké hrany.
- Zavoláme se rekurzivně.
- Nakonec doplníme to, co vyházel borůvka.

☺

Lemma 7 *Složitost v nejhorším případě je* $O(\min(n^2, m \cdot \log n))$.

TODO: Proof

Věta 2 *Průměrně je to lineární.*

Důkaz:
Rozdělím strom rekurze na „levé cesty“ – někde začnu a jdu jen doleva. Tím pokryju celý graf. Každá levá cesta má málo hran, u toho udokazuji, že to proběhne lineárně s velikostí vršku. Pravá za něco visí, tam mám vždycky dostatečně málo hran.

☻

2 Perzistence

Může být:

- Částečná – můžu přidávat na konec, ptát se na libovolnou verzi.
- Plná – můžu se vrátit do minulosti, dá se větvit.
- Stoková – dá se i slívat.

Využívá se např. ve funkcionálních jazycích, dvourozměrné geometrické úlohy.

V každém vrcholu (řekněme stromu) si potřebuji navíc pamatovat tabulku „změn“, občas se narodí nový vrchol, do něj nacpu změny z tabulky.

3 Softheap

Je to halda, která může podvádět. Občas smí některý prvek změnit. Na to má ale omezení:

- Hodnotu prvku smí jen zvětšit, nikoliv zmenšit. Při výstupu se dá určit, jestli byl poškozen a kolik byl původně (ale prostě prohazuje pořadí).
- Má parametr ϵ někde mezi 0 a 0.5 a zaručuje, že bude poškozených zároveň max. $\epsilon \cdot n$ prvků (ale n je počet prvků, které tam byly kdy vloženy, ne kolik jich je tam teď.

Každá operace poběží v amortizovaném čase $O(\log \frac{1}{\epsilon})$.

Operace:

- Insert
- Delete-Min
- Merge
- Explode ($O(n)$) – vybere všechno z haldy a řekne, které byly poničené

3.1 Aplikace

Algoritmus 4 (k-tý nejmenší):

Vytvořím softheap s $\epsilon = \frac{1}{3}$. Nacpu všechno, vyberu třetinu prvků. Poslední, který vytáhnu (jeho poničenou hodnotu) označím x. To je $\geq$ všem předchoze vytaženým původním hodnotám. Stejně tak je $\leq$ alespoň třetině prvků (max $\frac{1}{3}$ jich je poničených). To se zvládne v lineárním čase.

Použijeme x jako pivota a zjistíme, kolik jich je možné vyhodit. Vyhodí se alespoň $\frac{1}{3}$ prvků, zrekurzením dojdeme k výsledku v lineárním čase.

☺

3.2 Softtree

Binární strom (nemusí být úplný), každý vrchol má obousměrný spoják prvků, které bydlí v tomto vrcholu. V klidovém stavu jsou seznamy v levých synech prázdné.

Vrcholy jsou černé a bílé, bílé jsou leví.

Další hodnota je řídící klíč vrcholu. To je první prvek seznamu, pokud je neprázdný. Pokud je prázdný, tak je buď ∞, nebo zůstane nastavený z doby, kdy byl nastavený.

Má haldové uspořádání podle těchto klíčů.

Dále je $rank(v)$, který splňuje, že listy mají 0, otec má větší, než oba synové.

Ranky obou synů budou stejné.

Ve stromu ranku k má levá cesta minimálně $\frac{k}{2}$ vrcholů.

Každý vnitřní vrchol má právě dva syny.

Strom je ***úplný***, právě když rozdíly ranků je 1.

Pozorování 2 *Každý strom lze vnořit do nějakého úplného se stejným rankem v kořeni.*

Rank stromu bude rank jeho kořene. Obdobně s klíčem.

Pozorování 3 *Úplný strom ranku k má $2^{k+1} - 1$ vrcholů, z čehož 2^k je černých.*

3.2.1 Operace se stromy

- **Merge** – dostane dva stromy se stejným rankem a vytvoří jeden strom, jehož rank bude o 1 větší. Seznam z kořene jednoho stromu přesune do kořene nového, druhý jen zavěsí (ten s menším klíčem). Klíč novému bílému necháme stejný.
- **Dismantle** – pokud z kořene vyžereme všechny prvky, není potřeba. Proto zruší levou (zcela prázdnou) cestu. Vznikne $\leq k$ stromů různých ranků.

3.3 Struktura haldy

Halda obsahuje obousměrný spoják, ve kterém jsou zapojené stromy v pořadí ostře rostoucích ranků. Z toho plyne, že nejsou dva stromy stejného ranku. Nejmenší je hlava, největší ocas.

Dále si pamatuji suffixová minima klíčů. Tedy, každý si pamatuje nejmenšího z následníků.

Nakonec máme ještě nějaký parametr r, sudé celé číslo, závisí na ϵ.

Rank haldy je rank ocasu, klíč haldy také.

3.4 Operace

3.4.1 Merge

Smím slévat jen, pokud mám stejné r.

Slévání stromů udělám podobně jako u sčítání binárních čísel, procházením od nejmenších a slučování, pokud jsou stejné. Přepočítání sufixových minim zvládnu průchodem zpátky odtamtud, kde jsem skončil.

3.4.2 Insert

Vyrobí se jednoprvková halda a mergne se.

3.4.3 DeleteMin

Najde strom s minimálním klíčem a z kořene od konce prvek odebere.

Pokud se vyprázdnil, zavoláme naplnění. Ještě předtím zkontroluji invariant, že je strom dost hluboký.

Nakonec se přepočítají minima. Přepočítává se doleva od tohoto vrcholu. Na toto potřebuji, aby platil invariant o hloubce stromu.

Pokud invariant neplatil, udělám dismantle a mergnu to do haldy. Napřed potřebuji ještě přepočítat sufixová minima.

3.4.4 Refill

Funguje rekurzivně na levých synech. Tam je taky prázdno, nechám ho sehnat data. Mám data v obou. Vyberu menší, vezmu si ho a zařídím, aby bylo nalevo prázdno.

Pokud nedostanu data, tak klíč je nekonečno. Taková větev je možná smazat. Tom můžu vymazat sebe a na své místo dát pravého.

V dolních patrech (nižších než r) už nic neslévám. V těch vyšších, pokud $rank(v) > r$ a $rank(v)$ lichý nebo $rank(v) > rank(p) + 1$, pak zavolám ještě jeden refill na levém a spojím si seznamy, klíč vezmu ten větší.

Nakonec, pokud nejsou žádná data, pak promazávám.

3.5 Analýza

Děláme odhad k r, což je parametr přesnosti a n, což je celkový počet insertů.

Lemma 8 $\forall v |list(v)| \leq \max(1, 2^{\lceil \frac{rank(v)}{2} \rceil - \frac{r}{2}})$

Důkaz:
Jediné, co dělá nepříjemnosti, je refill (3.4.4). Buď přesouvá do vrcholu s větším rankem, což je v pořádku, ale někdy volá dvakrát. V takovém případě je tam mezera alespoň 2, tedy se exponent zvětší max. o 1, tedy zdvojnásobí.

☻

Nyní chceme odhadnout celkovou délku seznamů v horních hladinách (tam, kde můžou být delší, než 1). Ale neznáme přesné tvary stromů, proto je vnoříme do úplných.

Lemma 9 *Úplné stromy ke všem stromům v haldě obsahují nejvýše n černých vrcholů.*

Důkaz:
Merge a insert jsou v pohodě.

Delete v pohodě, jediné, co něco dělá, je refill, ale on může jen černé vrcholy mazat.

☻

Lemma 10 *Úplný strom ranku k obsahuje nejvýše 2^{k-r+2} poškozených prvků.*

Důkaz:
Počet vrcholů ranku $i = 2^k - i$.

$$\begin{aligned}
\sum_{v,|list(v)|>1} |list(v)| &\leq \sum_{i=r+1}^{k} 2^{k-1} \cdot 2^{\lceil \frac{i}{2} \rceil - \frac{r}{2}} \\
&= 2^{k-\frac{r}{2}+\frac{1}{2a}} \cdot \sum_{i=r+1}^{k} 2^{-\frac{1}{2}} \\
&\leq 2^{k-\frac{r}{2}+\frac{1}{2}-\frac{r+1}{2}} \cdot \sum_{i\geq 0} 2^{-\frac{i}{2}} \\
&\leq 2^{k-r} \cdot 4
\end{aligned}$$

☻

Důsledek 1 *Úplný strom má 2^k černých vrcholů, tedy počet poškozených je $\leq 2^{2-r}$ černých.*

Celkem tedy přes všechny počet poškozených $\leq n \cdot 2^{2-r}$.

3.6 Nastavení r

$$\begin{aligned}
\epsilon &\geq 2^{2-r} \\
\log \epsilon &\geq 2 - r \\
-\log \epsilon &\leq r - 2 \\
r &\geq 2 + \log \frac{1}{\epsilon}
\end{aligned}$$

Dále jsme ho chtěli sudé.

3.6.1 Časová složitost

Napřed počítáme s tím, že insert stojí amortizovaně $O(r)$.

Zbytek se vždy nějak naúčtuje.

Napřed vezmeme **merge**. Ten jednak slévá, vyřizuje přenosy a nakonec počítá suffixová minima. Explicitní (např. při inzertu) bude trvat $O(1)$, indukovaný (z dismantle) $O(rank(T))$. U přenosu ubývají stromy, tedy se naúčtuje buď insertu nebo dismantlu. Slévání stojí $O(\min(rank(T_1), rank(T_2)))$. Kdyby všechny merge byly explicitní, tak si sestrojíme les mergů. Vždy se dvě haldy sloučí dohromady, slučují z jednoprvkových hald. Poprohazujeme syny tak, aby pravý měl menší rank. Tedy to stojí rank pravého syna, který je rovna jeho hloubce. Rozložíme přes všechny syny. Počet pravých synů s rankem $k \leq \frac{n}{2^k}$. Na pravých hranách se liší alespoň o 1, na levých se hnout nemusí.

Když mám víc vrcholů stejného ranku, tak ho zkomprimuji do jediného. Tímto způsobem už to určitě platí. Naúčtováno je $\sum \frac{n}{2^i} \cdot i = n \cdot \sum \frac{i}{2^i} = 2 \cdot n$.

Celková cena všech neindukovaných mergů je $O(n)$.

Nyní vrátíme indukované merge. Ale v tomto lese je nebudeme mít. Tím někdy může snížit ranky „nahoře“, ale to jen pomůže. Sami sebe zaplatí.

Refill v té přesné části stráví nejvíce $O(r)$. V horní části to celkově vyjde $O(n)$ na celou dobu. Při návštěvě vrcholu můžeme mazat (ale každý vrchol max. $1\times$), někdy rekurze $2\times$, někdy $1\times$. Slévám při tom seznamy, ale ty se rozeberou až nahoře. Slití dohromady je $O(n)$, neb strom slévání je binární. Když rekurzíme jen jednou, tak je jen jeden nad sebou.

Delete vyndává prvky ze seznamu, ale to je nezajímavé. Dále kontroluje invariant. Pokud platí, tak potom tam projde ještě refill, takže to nevadí. Neúspěšná se naúčtuje dismantlu.

Když došlo k dismantlu, tak má moc krátkou levou cestu. V horní polovině cesty alespoň jeden rank chybí (je jich tam moc málo). Tedy chybí strom ranku alespoň $\frac{k}{2}$. Tedy chybí dostatek prvků. Sice jsem mohl naúčtovat jednomu prvku více dismanlů, ale jen pro různé ranky, tedy to celkem vyjde jen $O(n)$.

Přepočítání suffixů je jen po cestě sem, s tím ale pomůže rankový invariant.

4 Pointer machine

Pointer machine má nějaké datové buňky, uvnitř je pevně stanovený počet položek na ukazatele a na data.

Dále má konstantní počet registrů na ukazatele a symboly.

Dá se přistupovat v registrech, jednak k hodnotách, na které ukazuji registry.

Umím kopírovat ukazatele a data, funkce na datech, podmíněné skoky.

V základní verzi je symbolů konečně mnoho (mnoho algoritmů porušuje, potřebují čísla). Rozšířená verze umí pracovat i s přirozenými čísly.

Další možnost je porovnávat ukazatele, vznikají stále větší a větší (ukazatel je timestamp).

Funkcionální verze nesmí modifikovat buňky, jen nastavit při vytváření.

4.1 Simulace

Pointer Machine lze simulovat na RAM s konstantním zpomalením.

RAM lze simulovat na Ponter Machine lze tak, že si pamatujeme čísla jako spojáky a paměť jako trii. Aritmetiku dokážu simulovat lineárně, přístup k paměti s logaritmickým zpomalením.

Šířka slova je nějaké w, takže složitost $O(T(n) \cdot \log T(n) \cdot w^2)$.

4.2 Algoritmy

Některé algoritmy jsou přímo stavěné na pointer machine, např. softheap. Jediné co, potřebuji čísla na ranky. Na ně utvořím spoják, první je nula, druhý jedna, etc. Porovnat lze porovnáním pointerů (pokud je máme, jinak lineárně). U softheapu si dokonce můžeme dovolit to zaplatit.

4.3 Přihrádkové třídění

Čísla nejsou čísla, ale spojáky, tak si každý pamatuje ukazatel, kam patří.

Případně si může každý pamatovat svý ukazatel (např. ve vrcholu).

U grafu bylo např. potřeba normovat dvojice vrcholů, aby $i < j$. To nejde udělat konstantně. Ale jde to udělat dávkově pro všechny v lineárním čase, přihrádkovým třídění.

Ne vždy je potřeba třídit, ale jen slučovat k sobě, pak není potřeba udržet je setříděné a stačí mi mít seznam „neprázdných“.

Věta 3 *Posloupnost n k-tic nad A-prvkovou abecedou lze unifikovat v čase $O(kn)$ po inicializaci v čase $O(A)$ (Inicializace je vyrobení spojáku klíčů).*

4.3.1 Řetězce

Roztřídí se napřed podle délek a poté jako k-tice.

Věta 4 *Posloupnost řetězců $s_1, \ldots, s_n$ nad A-prvkovou abecedou lze unifikovat v čase $O(\sum_i s_i + 1)$ po inicializaci v čase $O(A)$.*

4.4 Testování isomorfizmu časů

Napřed si všimněme, že stačí zakořeněné stromy. Potřebuji najít odpovídající kořeny – najdu centrum, což je jeden nebo dva vrcholy. Vyzkouším obě možnosti (v nejhorším případě).

Isomorfizmus zakořeněných stromů si zobecníme na les. Chceme najít ekvivalence na podstromech.

Vrcholům přiřadíme výšky. Budu je postupně trhat a přiřazovat fázi, kdy jsem je utrhnul.

Vím, že podstromy s různými výškami nejsou isomorfní. Ve výšce ≤ 1 jsou jen jedna možnost, tedy jsou ekvivalentní a přiřadím jim třídu.

Nechť tedy jsme v nějaké výšce a všechny menší jsou již spočítané. Každý si lze reprezentovat jako d-tici podstromů. Ty lze setřidit a potom zunifikovat.

Všechny unifikace zvládneme lineárně. Celková inicializace je také lineárně, neboť abeceda (třídy ekvivalence) je max. n stromů. Trochu problém je setřídit jednotlivé řetězce, podle dvojice hodnota-pozice, poté se znovu postaví.

4.5 Union-Find

Jediný problém je s ranky. Lze zařídit na porovnávání ukazatelů.

4.6 Cell-probe model

Je paměť s $\log n$ velkými slovy. Jediné věci, které se počítají do složitosti jsou přístupy do paměti.

Často se používá na dolní odhady časové složitosti.

4.7 Ackermanova funkce

$$\begin{aligned} A(0,y) &= 2y \\ A(x,0) &= 0 \\ A(x,1) &= 2 \\ A(x,y) &= A(x-1, A(x,y-1)) \end{aligned}$$

Roste rychle (velmi). Jednoparametrovou lze definovat jako diagonálu.

Možných inverzích je spoustu. První je nekonečná posloupnost funkcí, $\lambda_i(n)$ je inverze i-tého řádku. Tedy $\min\{y|A(i,y) > \log n\}$.

$\alpha(n)$ je diagonální inverze, tedy $\min\{x|A(x) > \log n\}$. Roste ještě pomaleji, než všechny λ.

$$\alpha(m,n) := \min\left\{x|A\left(x, 4\left\lceil\frac{m}{n}\right\rceil\right) > \log n\right\}$$

Pozorování 4 *Při zafixování n je největší pro $m = n$.*

Pozorování 5

$$\alpha(m,n) \leq \alpha(n+1)$$

Důkaz:

$$\begin{aligned} \alpha(m,n) &\leq \alpha(n,n) \\ A(x,4) = A(x-1, A(x,3) &\geq A(x-1, x-1) = A(x-1) \end{aligned}$$

☻

Pozorování 6 *Pro pevné i, když $m \geq n \cdot \lambda_i(n)$, pak $\alpha(m,n) \leq i$.*

Důkaz:
$A\left(x, 4 \cdot \left\lceil\frac{m}{n}\right\rceil\right) \geq A(x, 4 \cdot \lambda_i(n)) \geq A(x, \lambda_i(n)) > \log n$
To určitě nastane pro $x \geq i$.

☻

4.8 Nejnižší společný předek

Pro union-find platí, že $m \geq n$ operací na n-prvkové struktuře trvá $O(n + m \cdot \alpha(m,n))$.

Nyní, n bude velikost stromu (počet vrcholů) a $m := n + \#$dotazů. Já si tyto dotazy přidám jako hrany. BÚNO není multigraf (stejné dotazy dokážeme sjednotit na jednu hranu).

Algoritmus 5 (Pomalý):

Pustím na to DFS na původní strom. V každém vrcholu mám union-find ekvivalenční třídu na vrcholy „nalevo“ od cesty. Když se vracíme z vrcholu,

tak ekvivalenční třída obsahuje celý podstrom. Chystáme se sjednotit s otcem, ale napřed se podíváme na dotazoné hrany z tohoto vrcholu. Podíváme se, kam padne druhý konec, podle toho poznáme nejbližšího společného předchůdce (pokud už byl také navštívený – pokud ne, počkáme na druhý konec).

☺

Časová složitost tohoto algoritmu stojí na union-findu.

4.8.1 Dekompozice

Zvolíme nějaké k ($1 \leq k \leq n$). Mikrostromy zvolíme tak, že velikost je nejvýše k a kdybychom ho zavěsili výš, tak už k překročíme. Makrostrom bude ten zbytek.

Pozorování 7 *Má maximálně* $\frac{n}{k}$ *listů.*

Pokud se ptám na interně uvnitř mikrostromu, tak řeším lokálně. V makrostromu je to uvnitř, jinak nahrazuji listem makrostromu, pod kterým mikrostrom visí.

4.8.2 Upravený union-find

Máme n prvků, z nichž alespoň l je speciálních. Zakážeme union dvou komponent, které ani jedna nemá speciální vrchol. Potom posloupnost m operací trvá $O(n + m \cdot \alpha(m, l))$.

Vypadá stejně, jako minulá, budu si pamatovat místo počtu vrcholů jen počty speciálních vrcholů. Stromy vypadají tak, že ty obyčejné jsou vždycky listy. Operace nad obyčejnými přejdou po konstantě do množiny speciálních vrcholů.

Nyní můžu použít tuto union-find na spojování ve stromech, protože speciální budou listy.

Složitost algoritmu na makrostromu je tedy $O\left(n + m \cdot \alpha\left(m, \overline{n}k\right)\right)$. k nastavím na $\log^{\frac{1}{3}} n$. Tím jsme stlačili α pod 2, tedy složitost je $O(n + m)$.

4.8.3 Topologický grafový problém

Vstupem bude neorientovaný graf s vrcholy a hranami ohodnocenými symboly z konečné abecedy Σ.

Výstupem je ohodnocený neorientovaný graf a ukazatele do vstupu.

Např. by se tím dalo vyřešit perfektní párování – nechám jen párovací hrany a u vrcholu odkazy na původní ve vstupu.

Toto se dá vyřešit i s konektory (kvůli přenálepkovávání) – abych u velkého výstupu stihl vyměnit vstup.

4.8.4 Mikrostromy

Zakódujeme všechny mikrostromy do řetězců a pak setřídíme do přihrádek. Do mikroproblému zakódujeme nejen strom, ale i jaké se na nich dělají dotazy. Tedy, kódujeme hrany původního stromu a dotazové stromy.

Isomorfizmus budeme dělat tak, že procházíme do hloubky, číslujeme vrcholy a u zpětných hran zapisujeme, co bylo na druhém konci (a label). Toto je lineární ve velikosti grafu. Přesněji, maximálně $n \cdot (n+4)$. Abeceda obsahuje čísla $1 \dots n$, labelů a závorek. Abeceda je velká $n + s$. Počet různých kódů je max. $(n + s)^{n \cdot (n+4)}$.

Vstupem bude $\mathcal{C}$ kolekce grafů. Postavíme $\mathcal{G}$ kolekci generických grafů. Vyřešíme pro generické grafy, získáme $\mathcal{O}$ kolekci generických řešení. Pak roztřídíme $\mathcal{C}$ podle kódů a přiřádíme řešením a propojujeme konektory.

Třídění běží dostatečně rychle. Vyrobit $\mathcal{G}$ za $O(k+s)^{(k+4)^2}$. Spočítat $|G|(T(k)+O(k^2)$.

To se pro T polynomiální stihne v lineárním čase.

Ověření minimality kostry udělám tak, že si pomocí LCA najdu předchůdce a každou hranu nahradím hranami „nahoru“. Potom ještě potřebuji hledat maxima cest. Použiji strukturu link-eval (podobná union-findu).

Máme les zakořeněných stromů s ohodnocenými hranami. Máme operaci $\star$ (to je eval). Link přivěsí strom někam. Eval spočítá cestu z v do kořene.

Kdybych v union-findu nedělal path-compression, je to OK. Když dělám, ukládám do hran.

S tímhle mám hledání nejtěžší hrany na cestě.

Problém je, že minimalitu kostry nemůžeme prohlásit za topologický problém. Použijeme rozhodovací stromy – bude odpovídat ne na dotazy, ale rozhodovacími stromy.

Rozhodovací strom spočítám v $2^{O(k^2)}$, po dosazení to ještě vyjde.

5 Nejkratší cesty

Hledat nejkratší cesty nejde rychle, umíme nejkratší sledy. Dá se to obejít, když nejsou záporné cykly, splývá s nejkratším sledem.

Budeme pracovat na ohodnocených grafech.

Chceme dostat graf, dva vrcholy, chceme najít nejkratší cestu mezi nimi. To se také moc neumí, umí se ze zdrojového najít nejkratší sledy do všech ostatních. Přímo konstruovat cesty je nepraktické, obvykle se konstruují buď vzdálenosti, nebo strom nejkratších cest. Strom lze převádět na vzdálenosti, opačně to také jde. Tomuhle se říká SSSP.

Často se hledá nezáporná verze, kde nesmějí být žádné záporné hrany.

Chceme sestrojit všechny dvojice nejkratších cest. Tomu se říká APSP.

5.1 SSSP

Obvykle začneme s nějakým ohodnocením, postupně zlepšujeme, až dokonvergujeme. Tři druhy vrcholů – neviděné, načaté, hotové.

Poté se nějakým způsobem vybere rozdělaný vrchol, relaxuje podél hran, co vedou ven a prohlásí za hotový. Cílový, pokud upraví, nastaví na rozdělaný (i když ten byl i hotový). Této operaci budeme říkat scan. Algoritmus končí ve chvíli kdy jsou všechny hotové.

Tohle nikdy neskončí, pokud tam je záporný cyklus. Libovolné ohodnocení vždy odpovídá nějakému sledu ze zdrojového vrcholu sem. Kdyby sled nebyl cestou, tak vznikl cyklus, který musel být záporný.

Na pořadí scanů záleží. Někdy to může trvat až exponenciálně dlouho.

Pokud očísluji vrcholy a scanuji v cyklickém pořadí, potom je v i-tém průběhu, $d(v)$ je omezené na cykly délky alespoň i.

Bellman-Ford je zlepšená verze tohoto. Tomu nevadí záporné hrany. Lze stihnout v $\Theta(m \cdot n)$.

Byly různé vylepšovací heuristiky, ale kazí složitost v nejhorším případě.

Dijxtra zavírá ten z otevřených, který má nejmenší ohodnocení. Takto formulované to funguje i se zápornými hranami, jen bude ničit složitost. Spousta algoritmů vychází z tohoto, jen vyměňuje haldu a využívá např. celočíselnosti.

5.1.1 Redukce na nezáporné

Mějme potenciál $\Phi : V(G) \to \mathbb{R}$. Kdykoliv máme $l : E(G) \to \mathbb{R}$, pak lze udělat úpravu l tak, že $l_\Phi(u, v) := l(u, v) + \Phi(u) - \Phi(v)$. Potom se to poodčítá na každém sledu tak, že je tam jen první a poslední. Cykly to neničí.

Tím se dá zbavit záporných hran. Zařídíme, aby to byl dostatečný rozdíl. Vzdálenosti od nějakého daného vrcholu, tak to platí. Tohle se vyplatí, pokud se něco má iterovat mnohokrát (např. jednou pustit bellman-forda, potom mnohokrát dijxstru, pokud chceme všechny dvojice).

5.2 Heuristiky pro hledání k cílovému vrcholu

Existují různé, které se snaží vylepšit dijxtru, aby nemusel prohledávat všechno.

Například A^* je vylepšení dijxtry, který obecně nefunguje. Máme hint $h(v)$, potom vybíráme $\min d(v) + h(v)$. Kdybych koukal na $h(v)$ jako náš upravovací potenciál (potenciál je $-h(v)$). Např. na mapě euklidovská vzdálenost funguje.

Další trik je, že se dají pustit dva dijxtrové „proti sobě“. Tohle ne vždy funguje, ale když počítám nějak přes spojovací hranu, tak už to fungovat bude.

5.3 Trik s majáčky

Rozmístím majáčky (dobře vybrané vrcholy) a rozpočítat si vzdálenosti k nim odněkud. Minimum, maximum a průměr je korektní potenciál. Průběžným přepínáním se dá zařídit, aby scanoval „přibližně správným směrem“ přednostně.

Dijxtrově algoritmu stačí monotónní halda – minima tvoří monotónní posloupnost. Lze použít nějaké přihrádky, pokud jsou malé celočíselné délky hran. Dá se to různě nakombinovat.

5.4 Kvantování

Zvolím si kvantum Q, nasekám délky na toto (tím jsou celočíselné). Rekurzivně spočítám v tom, co jsem dostal. Výsledek se liší maximálně o $Q \cdot l$. Definujeme potenciál jako $Q\cdot$ to co vyšlo.

Délka ze zdroje do libovolného vrcholu je max. $Q \cdot n$.

Pozorování 8 *Vzdálenosti v grafu nakvantovaném jsou maximálně $Q \cdot n$. Nalezení této cesty lze tedy stihnout (díky přihrádkám) v $O(m + Q \cdot n)$. Pro $Q = \frac{m}{n}$ vyjde čas celého $O(m \cdot \log_{\frac{m}{n}} \cdot L)$.*

Věta 5 *Je-li G rovinný na n vrcholech, potom existuje separátor velikosti maximálné $O(\sqrt{n})$ a zbylé komponenty mají max. $\frac{2}{3}$ vrcholů.*

Když seřadím napřed komponenty a potom až nakonec separátor, pak když pustím Floyd-Warshalla, potom můžu napřed vyřešit jednotlivé komponenty zvlášť, potom vyřešit separátor – dopočítání je tedy v $O(n^{2.5})$. Postupně práce ubývá, tedy dokážeme všechny dvojice v $n^{2.5}$.

Pokud máme matici sousednosti, pak nám násobením vypadne, co je s čím spojené. Pokusíme se to upravit na to, aby počítalo délky cest, resp. sledů. To

místo násobení budeme sčítat, místo sčítání budeme brát minimum. Tohle bohužel nefunguje s těmi zrychlovacími triky, ty potřebují inverzi (odčítání). Ale jde to převést na maticový součin na polynomech, ale to nepomáhá (leda by se počítalo na fourierových obrazech, fuj).

Dá se to uutlouct na na něco jako $6 +, \min$ součinů a dva normální.

5.5 Všechny vzdálenosti v neohodnocených neorientovaných grafech

Uděláme si G^2 (má hranu kde byla původně hrana, nebo sled délky 2, ale ne smyčky). Z toho se dají spočítat vzdálenosti δ^2. V původním jsou dvojnásobné. Je potřeba zjistit, kde odečíst jedničku od toho dvojnásobku. Dá se vykoukat z průměru vzdáleností sousedů.

Redukce se zastaví, když máme úplňák.

6 Isomorfismus stromů

Zakořeníme si to. Každý strom má centrum nebo bicentrum, vezmu je (pokud jsou bicentra, vyzkouším obě možnosti).

Začnu počítat kódování od listů. List má nějaký triviální, při počítání otce zkombinuji všechny syny a seřadím. Problém je, že tohle trvá dlouho.

Přihrádkové třídění můžu udělat i na pointer machine (můžu indexovat přímo pointery na vrcholy).

6.1 Binární stromy

Budu počítat třídy isomorfismů na všech zakořeněných podstromech. Budu průběžně přiřazovat kódy. Máme nějaké stromy hloubky n, začneme tím, že budeme pojmenovávat od prázdných stromů.

Napřed si je chceme uspořádat (ty dvojice synů). To by bylo pomalé, ale můžu si udržovat přihrádky, které už jsou použité. Na pořadí nezáleží, nám jde jen o to, aby byly u sebe. Poté zunifikuji stejné dvojice, stejné budou nové ekvivalenční skupiny.

Když jsou více než binární, tak napřed setřídíme podle stupňů, poté můžeme třídit už normálně přihrádkově.

Je potřeba ještě umět uspořádávat syny. Stejný trik s přidáváním přihrádek.