Domácí úkol 5:

Při porovnávání třech základních algoritmů na kostry si můžeme všimnout, různých detailů chování algoritmů. Kontrahující verze Borůvkova algoritmu umí velmi rychle redukovat počet vrcholů vstupního grafu, ale v obecném případě se jeho pracovní graf zahustí a hrany ho zpomalí. Na druhou stranu Jarníkův algoritmus s vhodnou haldou (třeba Fibbonacciho) běží nejrychleji na hustých grafech, protože jeho složitost závisí na počtu hran pouze lineárně.

Jak těchto nuancí zneužít? Můžeme zkusit použít Borůvku na snížení počtu vrcholů vstupního grafu, a než se příliš zpomalí předat slovo Jarníkovi. Jarníkovi nebude vadit velké množství hran, ale menší množství vrcholů ho potěší.

Analyzujte časovou složitost následujícího algoritmu a ukažte, že je asymptoticky rychlejší než $\mathcal{O}(m \log(n))$

Function *Chimera(G, w váhy hran)*
  Proveď $\log \log n$ fází Borůvkova algoritmu
  $F_B \leftarrow$ hrany vybrané Borůvkou
  $G' \leftarrow G/F_B$ (kontrakce lesa nalezeného Borůvkou)
  $T' \leftarrow$ JarnikSFibHaldou(G')
  return $T' + F_B$
end

Upřesnění: Hrany chápeme jako objekty, které si zachovávají svou identitu i poté co jsou jejich koncové vrcholy změněny (např. každá hrana má svoje id). Všechny hrany G' tedy vědí ze kterých hran vstupního G vznikly. (a tedy dává smysl kostru G stavět z hran G')

Jak je zvykem u minimálních koster, předpokládáme, že G je souvislý a váhy hran jsou unikátní.

Pro správnou časovou analýzu je potřeba znát časovou složitost Jarníkova alg. s Fib. haldou, složitost fáze Borůvkova algoritmu, a správně odhadnout počet vrcholů v grafu G' (počet hran stačí odhadnout triviálně jako $\mathcal{O}(m)$). Všechny potřebné ingredience se řešily na cvičení, případně najdete spoustu detailů v sekcích 7.2 a 7.3 průvodce.

Pro úspěšné řešení je vyžadována pouze časová analýza časové složitosti se zdůvodněním.

Analýza Borůvkové části

Jedna fáze Borůvkova algoritmu trvá $\mathcal{O}(n+m)$, prohledávání pro detekci komponent a projití hran pro hledání minimálních incidentních hran. Z předpokladu souvislosti můžeme psát pouze $\mathcal{O}(m)$. Celková složitost $\log\log n$ fází tedy bude $\mathcal{O}(m \log\log n)$.

Analýza počtu vrcholů

Každá fáze Borůvkova algoritmu sníží počet komponent alespoň o polovinu. Po k fázích tedy zbývá nejvýše $n/2^k$ komponent. Použijeme standardní předpoklad, že použitý logaritmus je dvojkový, nebo si všimneme, že volba dvojkového logaritmu je optimální. Logaritmus je inverzní k exponenciální funkci, obecně tedy platí $2^{\log f(n)} = f(n)$. Dostáváme, že po $\log\log(n)$ iteracích a následné kontrakci komponent počet vrcholů je nejvýše

$$\frac{n}{2^{\log\log(n)}} = \frac{n}{\log(n)}$$

Analýza Jarníkové části

Použitím Fibonacciho haldy dosahuje Jarník složitost $\mathcal{O}(m' + n'\log(n'))$. Kde n', m' jsou hodnoty n, m pro graf vstupující do Jarníkové části. Počet hran odhadneme jednoduše jako $\mathcal{O}(m)$ a dosadíme počet vrcholů po kontrakci $\frac{n}{\log(n)}$. Podíváme-li se na člen závislý na počtu vrcholů, vidíme, že $1/\log n$ klesá rychleji než $\log(n/\log n)$ roste, obě funkce se tedy vyruší do multiplikativního členu, kteý můžeme odhadnout jako $o(1)$ (asymptoticky menší než konstanta). Protože část s Borůvkou již dává superlineární složitost, tento odhad stačí a dostáváme, že Jarníková část trvá nejvýše lineárně dlouho.

$$\mathcal{O}\left(m + \frac{n}{\log n}\log\left(\frac{n}{\log n}\right)\right) = \mathcal{O}\left(m + n \cdot \frac{\log\left(\frac{n}{\log n}\right)}{\log n}\right) = \mathcal{O}(m+n) = \mathcal{O}(m)$$

Závěr

Součet složitostí obou částí (a kontrakce s linerání složitostí) získáváme složitost $\mathcal{O}(m \log\log n)$, což je zjevně lepší než $\mathcal{O}(m \log n)$.

Poznámky

Všimněme si hlavně závěru, že Jarníkova část proběhne v lineráním čase. Protože Jarníka s Fib. haldou zpomalují hlavně vrcholy, stačílo pouze mírně snížit jejich počet. K tomu se využilo pozorování, že Borůvkův algoritmus v kontrahující verzi umí velmi rychle vrcholy redukovat.

To není náhoda nebo vlastnost minimálních koster. Podobné kombinace algortimů lze najít u mnoha problémů. Například u třídění lze najít spoustu algoritmů s dobrými praktickými vlastnostmi založených na kombinaci Insert-sortu (pomalý, stabilní, malá paměť) a Merge-sortu (rychlý, nestabilní, velká paměť), např. Timsort, Strandsort a jejich zajímavější varianty. Obdobně existují různé hybridní konstrukce pro hledání nejkratších cest, konstrukce datových struktur, a veškeré další základní problémy.