PRL PROJEKT 2

Preorder Tree Traversal

Implementace prohledávání stromu do hloubky pomocí procesorů.

Autor: Monika Rosinská, xrosin00

Datum: 27. 4. 2022

Princip algoritmu

Principem převodu stromu (uvedeného v Breadh-First Search), je vytvoření adjacency listu (seznamu sousednosti), nad kterým je spočítána eulerova cesta. Následně jsou spočítány váhy hran (dopředná hrana má váhu jedna, zpětné hrany mají váhu 0), a nad těmito vahami je pomocí eulerovy cesty spočítána suma suffixů. Z té je poté možné určit preorder pořadí hran, a tím i vrcholů, do kterých hrany vedou.

Adjacency list

Adjacency list je vytvářen tak, že pro každý uzel se projdou všechny výstupní hrany v pořadí hrana k rodiči, hrana k levému potomkovi, hrana k pravému potomkovi. Jsou uloženy id hrany a id reversní hrany. Všechny hrany jsou poté uloženy do adjacency listu k odpovídajícímu vrcholu.

Eulerova cesta

Eulerova cesta se vytváří pomocí adjacency listu, kdy každý procesor provádí výpočet pro jednu hranu. Každý procesor se podívá, zda má v adjacency listu nějakého následníka. Pokud nějakého následníka má, přiřadíme do eulerovy cesty tohoto následníka. Pokud nemá, podívá se, z jakého uzlu hrana vede, a přiřadí do eulerovy cesty první položku z adjacency listu tohoto uzlu.

Váhy hran

Váhy hran jsou určeny tak, že dopředná hrana má váhu jedna, zpětné hrany mají váhu 0.

Suma suffixů

Pro vypočítání sumy suffixů je třeba zaslepit eulerovu cestu – hranu vedoucí ke kořeni od jeho pravého potomka je třeba “přeříznout”. Tím dosáhneme toho, že následníka této hrany v eulerově cestě nastavíme jako tuto hranu samotnou. Dále je třeba její hodnotu (zde váhu) nastavit na 0 – při paralelním výpočtu sumy suffixů se bude cyklit na posledním prvku, a je třeba, aby tento prvek byl neutrální (v případě sčítání, které se zde provádí, je neutrálním prvkem 0). Je ovšem potřeba původní hodnotu uložit, abychom později mohli provést korekci.

Poté se log(n) krát provádí následující cyklus:

i. ke své hodnotě přičteme hodnotu svého následníka

ii. jako svého následníka nastavíme následníka svého následníka

Cyklus se provádí log(n) krát, kde n je počet vrcholů, protože při každém dalším kroku se posuneme o 2x více pozic, tj. než se hrana nejvzdálenější od poslední hrany dostane k této poslední hraně, je potřeba provést log(n) iterací.

Na konci ke své hodnotě přičteme původní hodnotu posledního prvku.

Určení preorder pořadí vrcholů

Pro vypočítání preorder pořadí vrcholů je využito vypočítané sumy suffixů. V potaz se berou jen dopředné hrany (zpětné se ignorují). Předpokládejme dopřednou hranu *e(v, w)*, kde *v* je dopředná hrana a *w* je zpětná hrana. Pořadí vrcholu se poté určí pomocí vzorce *preorder(w) = n – suf_sum(e) + 1*, přičemž platí, že *preorder(kořenový uzel) = 1* (*n* reprezentuje počet vrcholů a *suf_sum* hodnotu sumy suffixů této hrany).

Analýza složitosti

Analýza složitosti je prováděna pro mou konkrétní implementaci, ale jsou uvedeny i ceny pro optimální (nebo lepší) algoritmy.

Počet potřebných procesorů

Jelikož při sumě suffixů každý procesor počítá hodnotu pro jednu hranu, je potřebný počet procesorů roven počtu hran. Hran v binárním stromě, počítáme-li i zpětné hrany, je 2*n – 2, kde n je počet vrcholů. Jelikož já využívám jeden proces čistě pro řízení, jje počet procesů 2*n – 1. Prostorová složitost je tedy *O(n)*.

Čas potřebný pro výpočet

- zpracování vstupu je lineární, tj. *O(n)*

- výpočet adjacency listu je v případě mé implementace také lineární, přičemž každý vrchol může obsahovat maximálně 3 prvky, tj. *O(n)*3*, tj. *O(n);* optimální algoritmus by provedl vytváření pomocí více procesorů, a složistost by poté byla *O(c)*

- eulerova cesta je počítána paralelně všemi procesory, tedy je vypočítána v konstatním čase, tj. *O(c)*

- váhy hran jsou také počítány všemi procesy, a časová složitost je také *O(c)*

- suma suffixů je počítána všemi procesory, a každý ji provádí log(n) krát (důvod viz výše), a proto je časová složitost *O(log n)*

- určení preorder pořadí vrcholů provádí jen hlavní procesor, a časová složitost řešení je *O(n)*; optimální algoritmus by to provedl pomocí více procesů, tj. složitost *O(c)*, ale jejich výpis by ale stále prováděl hlavní proces

- celková časová složitost (pokud nepočítáme načtení vstupu) je *O(n) * O(c) * O(c) * O(log n) * O(n) =*

*= O(n^2 * log(n));* pokud bychom vše řešili optimálně, byla by časová složitost *O(log(n))*

Cena řešení

Cena řešení c(n) = p(n) * t (n), kde p(n) je počet potřebných procesorů a t(n) je čas potřebný pro seřazení n prvků.

Tedy: c(n) = $O(n) * O(n^2 * \log n) = O(n^3 * \log n)$. V případě využití optimálních algoritmů je cena $O(n) * O(\log n) = O(n * \log n)$. Ani jedna cena však není optimální, protože optimální cena je $O(n)$.

Implementace

Algoritmus byl implementován pomocí openMPI knihovny v jazyce C++. Bylo využito 2*n -1 procesorů, kde n představujue délku vstupu (počet vrcholů).

Za hlavní, takzvaný řídící procesor, je považován procesor s ID 0. Tento procesor načte vstup, zpracuje ho a uloží, vypočítá adjacency list. Následně ostatním procesům rozešle potřebná data. Ty vypočítají eulerovu cestu, následně suffix sum, a odešlou ji hlavnímu procesu. Ten mezitím zajišťuje výměnu dat mezi procesy. Následně hlavní proces vypočítá preorder pořadí, a vytiskne jej na výstup.

Načtení dat a jejich uložení

Uzly jsou uloženy ve třídě *Node*, která obsahuje id uzlu, jeho název a vektor hran, které z něj vedou. Všechny uzly jsou poté uloženy v *map<int, Node>*. Dále jsou uloženy hrany ve třídě *Edge*, která obsahuje id hrany, id uzlu, do kterého vede, a id uzlu, ze kterého vychází. Všechny hrany jsou poté uloženy ve *vector<Edge>*.

Pro uložení adjacency listu je využito struktur *Way* a *adjacency_list_line*. *Way* obsahuje vždy jednu položku uzlu v adjacency listu – hrana, reversní hrana, a navíc uzel mezi těmito hranami (v obou směrech). *adjacency_list_line* představuje celý seznam *Way* daného uzlu v adjacency listu, obsahuje tedy id uzlu a vektor *ways*.

Implementace jednotlivých algoritmů

Adjacency list je vytvářen tak, že pro každý uzel se projdou všechny výstupní hrany v pořadí hrana k rodiči, hrana k levému potomkovi, hrana k pravému potomkovi. Jsou uloženy id hrany a id reversní hrany, a navíc jsou pro zjednodušení výpočtu uloženy i id uzlu, které se nacházejí mezi nimi (v obou směrech). Všechny hrany jsou poté uloženy do adjacency listu na index odpovídající danému uzlu.

Váhy hran jsou určeny jednoduše - díky systému přiřazení hran mají všechny dopředné hrany liché id, a všechny zpetné hrany sudé id, a váha hrany tak může být určena čistě jako *id_hrany%2*.

Eulerova cesta a suma suffixů je implementována stejným způsobem, jako bylo popsáno výše. Jelikož je suma suffixů prováděna paralelně, a procesory spolu komunikují pomocí posílání zpráv (nemají společnou paměť), je potřeba po každé iteraci aktualizovat eulerovu cestu i sumu suffixů (právě pomocí posílání zpráv).

Závěr a zhodnocení

Tento projekt úspěšně implementuje preorder průchod stromem. Jeho cena není optimální, a šla by snížit paralelním výpočtem adjacency listu a preorder pořadí. Ale i při takové implementaci by cena nebyla optimální. Optimální ceny by mohlo být dosaženo mensím počtem procesorů, protože řada z nich provádí zbytečnou práci (např. při výpočtu sumy suffixů pracují zbytečně ty procesory, které jako svého následníka mají koncový uzel).