Hledání nejkratších cest v grafu

Tomáš Duda a Artemij Pozdňakov
dudatom2@fit.cvut.cz a pozdnart@fit.cvut.cz

14. listopadu 2014

Obsah

1 Řešený problém a základní implementace

1.1 Definice problému

Naším úkolem v rámci semestrálního projektu v předmětu BI-EIA je implementace a optimalizace dvou algoritmů pro hledání nejkratších cest. Vstupem programu je tedy graf zadaný výčtem hran grafu a výstupem délka nejkratší cesty pro každou dvojici uzlů.

Pro řešení problému jsme implementovali dva algoritmy, první je Dijkstrův, druhý Floyd-Warshallův. Jejich stručný popis a informace o základní implementaci jsou obsahem následujících částí.

1.2 Dijkstrův algoritmus

Dijkstrův algoritmus funguje obdobně jako prohledávání do šířky, jenom místo obyčejné fronty používá prioritní frontu. Do té jsou před během algoritmu přesunuty všechny uzlu, počáteční z nulovou, zbylé s ∞ vzdáleností. Poté se až do vyprázdnění fronty vybírá nejbližší uzel a pro všechny jeho sousedy se vyzkouší, zda byla nalezena zkracující cesta (relaxace).

Běžná verze Dijkstrova algoritmu je určena pro hledání nejkratších cest z jednoho uzlu do všech ostatních, proto ho musíme v našem řešení volat $|V|$-krát, tedy z každého uzlu. Asymptotická složitost Dijkstrova algoritmu závisí na dvou faktorech. Jednak jde o vnitřní reprezentaci grafu (seznam uzlů a jejich sousedů nebo matice sousednosti) a dále o způsob implementace prioritní fronty, kterou algoritmus využívá.

V rámci snahy o kompromis mezi rozumnou rychlostí a dostatečnou možností kód následně optimalizovat jsme použili kombinaci reprezentace grafu seznamem sousedů a prioritní fronty řešené pomocí binární haldy. Tato kombinace má při hledání nejkratších cest mezi všemi dvojicemi uzlů asymptotickou složitost $\mathcal{O}(|V|(|E| + |V|) \log |V|)$, kde V je množina uzlů a E je množina hran.

1.3 Floyd-Warshallův algoritmus

Floyd-Warshallův algoritmus hledá nejkratší cesty metodou postupné konstrukce. Skládá se ze tří do sebe vnořených cyklů, které iterují přes všechny uzly grafu. Vnější cyklus určuje prostředníka, přes které se algoritmus právě snaží nalézt zlepšující cestu, dva vnitřní cykly poté určují dvojici koncových uzlů. Jako vnitřní reprezentace grafu je použita matice sousednosti, která je postupem algoritmu přepsaná na výslednou distanční matici. Asymptotická složitost Floyd-Warshallova algoritmu je $\mathcal{O}(|V|^3)$

1.4 Porovnání algoritmů

Pro řídké grafy ($|E| \sim |V|$) má Dijkstrův algoritmus (s použitím binární haldy) nižší teoretickou složitost než Floyd-Warshallův. Dá se však očekávat, že při reálném použití u grafů, které jsme schopni v rozumném čase na poskytnutém HW upočítat (řádově tisíce uzlů), bude Floyd-Warshallův algoritmus rychlejší, jelikož potřebuje vykonat v nejvnitřnějším cyklu mnohem méně operací než Dijkstrův.

1.5 Popis souborů

- **main.cpp** obsahuje zpracování argumentů z příkazové řádky a volání algoritmů.
- **mygraph.cpp** obsahuje třídu MyGraph, která jednak slouží pro vnitřní reprezentaci grafu (podporuje jak matici sousednosti tak i reprezentaci pomocí seznamů uzlů a jejich sousedů). Dále realizuje zpracování vstupního grafu ze souboru.
- **node.cpp** a **edge.cpp** obsahují pomocné třídy pro uložení uzlu nebo hrany v grafu.
- **floydwarshall.cpp** implementuje Floyd-Warshallův algoritmus.
- **dijkstra.cpp** implementuje Dijkstrův algoritmus.
- **exception.cpp** definuje výjimku, která je použita při spuštění programu s chybnými argumenty.

2 Optimalizovaná verze sekvenčního algoritmu

Následující kapitola popisuje optimalizace provedené v sekvenčních verzích implementace a následné výsledky meření různých verzí.

2.1 Dijkstrův algoritmus

Jelikož nebylo Dijkstrův algoritmus možné optimalizovat klasickými metodami (vysoká datová provázanost, nemožnost rozbalit vnitřní cyklus kvůli komplikované datové struktuře), pokusili jsme se řešení optimalizovat dvěma jinými způsoby.

Prvním byla výměna původně použité prioritní fronty z STL za vlastní implementaci, která navíc podporuje operaci decreaseKey a tudíž není potřeba u každého uzlu vyňatého z fronty testovat, zda je jeho hodnota klíče aktuální (zkrátka není nutné používat reinserting).

Druhý pokus optimalizace proběhnul pomocí použití různých přepínačů při kompilaci v gcc. TODO.

2.2 Floyd-Warshallův algoritmus

Loop-tiling, loop unrolling, vektorizace? TODO.

2.3 Popis testovacích instancí

TODO.

2.4 Měření a porovnání výkonosti různých sekvenčních verzí

Uvádět výkonost v MFlops. TODO.

3 Vícevláknová implementace

4 Závěr

Reference

[1] KOLÁŘ, Josef. *Teoretická informatika*. Česká informatická společnost, Praha, 2004. 205s.