Projekt do předmětu PDS 2013L
Longest-Prefix Match

Jan Dušek
xdusek17@stud.fit.vutbr.cz

27.4.2013

1 Úvod

Úkolem projektu bylo naimplementovat algoritmus Longest-Prefix Match, což je algoritmus, který vrátí řetězec s nejdelším prefixem z množiny řetězců pro dotazovaný řetězec. V našem projektu jsme naimplementovali konkrétní použití tohoto algoritmu, při rozhodování do jaké počítačové sítě, patří daná adresa.

Takováto úloha, se obvykle implementuje přímo v hardwaru, u různých směrovačů, či dalších embeeded zařízení, nebo jako součást operačního systému, obsluhující směrování. Naším úkolem nicméně bylo tuto úlohu naimplementovat, jako obyčejný program pro příkazovou řádku v libovolném jazyce.

2 Analýza a návrh

V tomto oddíle se budeme zabývat analýzou problému a také návrhem konkrétního řešení, pro náš problém aplikace algoritmu Longest-Prefix Match, při rozhodování do které podsítě patří požadovaná adresa.

2.1 Longest-Prefix Match

[1] Tento algoritmus používají směrovače, pro zvolení správného záznamu, ze směrovací tabulky. Každý záznam ve směrovací tabulce obsahuje podsíť, tedy prefix a jeho délku, při směrování může IP adresa odpovídat více prefixům, správný záznam je pak ten, jehož prefix je nejdelší.

Práce s IP adresami a prefixy je nemožná v tradiční textové reprezentaci, pro verzi čtyři je použita desítková a pro verzi šest hexadecimální soustava, protože algoritmus pracuje s jednotlivými bity, a je tedy nutné používat binární soustavu.

Pro algoritmus je tedy potřeba nejdříve mít k dispozici samotnou směrovací tabulku, ze které budeme vybírat záznamy. Pro rychlé vyhledávání nejdelšího prefixu, je vynikající datová struktura *trie* a její modifikace, o této struktuře bude více řečeno v následujícím oddíle.

2.2 Trie

[4] Trie je stromová datová struktura, někdy nazývaná *prefix tree*. Používá se k ukládání množiny nebo asociativního pole, kdy klíčem jsou řetězce. Na rozdíl od binárního vyhledávacího stromu, není klíč uložen v každém uzlu, nýbrž poloha uzlu ve stromu určuje klíč patřící uzlu. Rodiče uzlů jsou totiž prefixem, jejich dětí, přičemž každý uzel reprezentuje jeden element řetězce.

V našem případě jako klíč nepoužijeme znakové řetězce, nýbrž bitové řetězce a tedy uzly v *trie* budou mít maximálně dva potomky. Klíčem do naší směrovací tabulky tedy bude bitový řetězec proměnlivé délky reprezentující prefix podsítě.

Strukturu *trie* je možno vylepšit, tak že uzly jež mají pouze jedno dítě, lze spojit s vlastním dítětem, uzly pak nereprezentují jen jeden element ale mohou reprezentovat i více elementů. Tato struktura se

nazývá *radix tree* či *patricia trie*[2]. Největšího zlepšení se dosáhne pro malý počet dlouhých řetězců, kdy je sloučeno velmi mnoho uzlů, strom se zmenší a prohledávání je pak rychlejší.

3 Implementace

Jako implementační jazyk byl zvolen C++, jelikož podporuje objektové programování, má rozsáhlou standardní knihovnu a zároveň je to jazyk kompilovaný do strojového kódu a tudíž i rychlý.

3.1 Bitové řetězce

Binární *trie* používá jako klíč bitové řetězce takže je potřeba použít něco co umožňuje jednoduchou práci s jednotlivými bity. Pro toto obsahuje standardní knihovna vhodné dvě třídy.

První je třída `bitset` ta reprezentuje množinu jednotlivých bitů, ale její velikost je daná parametrem šablony tedy staticky při kompilaci a jelikož my si do *trie* potřebujeme ukládat prefixy různých délek tak je tato třída nevhodná museli bychom si velikosti ukládat někam mimo, což by bylo nepříjemné.

Další třídou je specializace kontejneru `vector` pro typ bool, jež ukládá jednotlivé bity, tato třída neumožňuje nad bity provádět bitové operace jako celek, ale umožňuje ukládat bitové posloupnosti různých délek, bohužel třída `vector` je dynamické pole, což by bylo příliš výkonově náročné. Vhodnější by bylo použít staticky alokované pole, jako je třída `array` ze standardní knihovny. Jelikož my známe maximální velikost prefixu 32 bitů a 128 bitů pro IPv4 respektive IPv6 tak by statické pole nebyl problém. Pro tento účel byla vytvořena šablonová třída `BitArray`, která jako parametr bere velikost interního staticky alokovaného pole, kam se ukládají jednotlivé bity.

3.2 Binární trie

Samotné binární *trie* bylo implementováno jako šablonová třída `BinaryTrie`, která po vzory standardních kontejnerů, umožňuje do uzlů ukládat data jakéhokoliv typu.

Trie bylo implementováno jako *patricia trie* tedy uzly, které neobsahují data a mají pouze jedno dítě jsou spojeny s vlastním dítětem. *Trie* je implementováno jako binární strom pomocí ukazatelů, kdy každý uzel obsahuje ukazatel na rodiče a dvě děti, dále obsahuje počet bitů klíče platných pro tento uzel, samotný klíč je uložen pouze u datových uzlů, to je optimalizace, díky které není při procházení stromu u každého uzlu třeba testovat přímo bity, což je náročné, ale stačí pouze porovnávat počet platných bitů.

Samotný Longest-Prefix Match, je implementován tak, že se prochází *trie*, dle příslušného bitu daného klíče jdeme doprava respektive doleva, dokud nenalezneme uzel jehož počet platných bitů je větší jak počet bitů v daném klíči, cestou si datové uzly ukládáme do zásobníku. Až prohledávání skončí, jednoduše porovnáme klíče uložených datových uzlů s daným uzlem a první, který se shoduje tak je uzel s nejdelším prefixem.

4 Profilování

Součástí hodnocení projektu je i rychlost implementace, takže bylo potřeba se alespoň pokusit o optimalizaci. Pro identifikaci míst v programu, které je nejvýhodnější optimalizovat jsou použil program *callgrind*, což je součást rodiny programů *valgrind*, pro profilování programů. Jako vizualizační nástroj výstupu *callgrindu* byl použit nástroj *KCacheGrind.*

Z výsledků profilování, vyplynulo, že největším „žroutem" času není samotný algoritmus Longest-Prefix Match, ale funkce `getaddrinfo`, která byla použita pro převod IP adres z textového formátu do binární podoby, proto byla tato funkce nahrazena jednodušší funkcí `inet_pton`, která sice neumí rozhodnout zda je IP adresa verze čtyři nebo šest, ale to bylo vyřešeno tak, že se nejprve adresa pokusí převést jako verze čtyři pokud funkce selže tak adresa převede jako verze šest.

Dále byli velmi náročné vstupně/výstupní operace pomocí standardních C++ streamů, takže bylo použito volání `ios_base::sync_with_stdio(false)`, které způsobí, že operace nad streamy nejsou synchronizovány se standardními streamy jazyka C, takže je pak nelze kombinovat se streamy z C++ jsou pak stejně rychlé jako ty z C.

5 Testování

Pro testování byli použity soubory poskytnuté v rámci zadání. Pro ověření správnosti implementace byla vytvořena referenční implementace projektu v jazyce *python* pomocí knihovny *py-radix*[3], kdy výsledek této referenční implementace byl pak porovnáván s výsledky vlastní implementace v C++.

Rychlost byla pak měřena na školním serveru *merlin* pomocí utility *time*. Měření probíholo od začátku programu do jeho konce tedy včetně inicializace, a bylo spuštěno na datech ze zadání tedy 178 220 prefixů, v tabulce 1 jsou vidět výsledky.

Tabulka 1: Měření rychlosti implementace

měř. 1 [s]	měř. 2 [s]	měř. 3 [s]	měř. 4 [s]	měř. 5 [s]	průměr [s]	prefixů/s
0.731	0.694	0.758	0.697	0.704	0.7168	248 632,8125

6 Závěr

V projektu se podařilo implementovat algoritmus Longest-Prefix Match pro IP adresy, s použitím vlastní implementace binární *patricia trie*.

Reference

[1] *Longest prefix match - Wikipedia, the free encyclopedia* [online]. [cit. 28.2.2013]. Dostupné na: <`http://en.wikipedia.org/wiki/Longest_prefix_match`>.

[2] *Radix tree - Wikipedia, the free encyclopedia* [online]. [cit. 28.2.2013]. Dostupné na: <`http://en.wikipedia.org/wiki/Radix_tree`>.

[3] *Radix tree data structure for network lookups* [online]. [cit. 27.4. 2013]. Dostupné na: <`https://code.google.com/p/py-radix/`>.

[4] *Trie - Wikipedia, the free encyclopedia* [online]. [cit. 28.2.2013]. Dostupné na: <`http://en.wikipedia.org/wiki/Trie`>.