# Longest-Prefix Match

Dokumentace k projektu z předmětu PDS

Ondřej Fibich (xfibic01)

27. dubna 2013

# Úvod

Cílem projektu bylo vytvořit implementaci algoritmu Longest-Prefix Match (LPM) pro přiřazování IPv4 a IPv6 adres k autonomním sítím (AS) dle adres jejich sítí.

## 1 Postup tvorby programu

Samotný návrh algoritmu a jeho vývoj proběhl postupně v několika částech. Použité časové údaje byly naměřeny na serveru **merlin**.

### 1.1 Prototyp

Po krátké úvaze a prostudování problematiky jsem se rozhodl pro implementaci prototypu programu v jazyce Java. Zvolil jsem nejjednodušší algoritmus - trie s velikostí řetězce jedna. Tuto datovou strukturu lze chápat jako binární strom, kde v jednotlivých uzlech stromu mohou (ale nemusí, s výjimkou listů stromu) být vloženy identifikátory AS.

Celý algoritmus spočíval ve dvou částech. V první části byly sestaveny dva stromy pro uložení identifikátorů AS dle jejich adres sítí. Druhá část již načítala vstupní IP adresy a vyhledávala nejdelší odpovídající prefix a vracela příslušné AS. Dva stromy byly zvoleny pro rozdělení práce s IPv4 a IPv6. Algoritmy pro výstavbu stromu a vyhledání prefixu byly implementovány tak, aby pracovaly s binární reprezentací IP adres.

Po dokončení implementace proběhla fáze testování a měření výkonu. Výsledný program byl zhruba 8 krát rychlejší než minimální přípustná rychlost implementace, přičemž stavba stromů z dat daných referenčním souborem trvala cca $1,5$ sekundy.

### 1.2 Výsledný program

Po implementaci prototypu jsem se rozhodl přistoupit k optimalizacím programu, a proto jsem aplikaci důkladně profiloval nástroji dostupnými v NetBeans IDE. Výsledek profilování byl překvapivý, jelikož se ukázalo, že vylepšením samotného zvoleného algoritmu lze docílit jen minimálního navýšení výkonu. Hlavní zátěží programu se staly vstupně-výstupní operace, což není až tak překvapivé, protože velikost souboru s cca 18 miliony IP adres má velikost cca 280 MB.

Rozhodl jsem se tedy program přepsat do jazyka C, kde jsou vstupně-výstupní operace na nižší úrovni, a tedy rychlejší. Výsledný program, který algoritmicky odpovídal prototypu, se skutečně zrychlil až 2 krát. Výstavba samotného stromu a především alokace a uvolňování paměti pro uzly stromu ovšem nebyly příliš optimální, jelikož byly prováděny po jednotlivých uzlech. Rozhodl jsem se tedy opustit tento způsob výstavby stromu a vytvořit si vlastní strukturu pro alokaci paměti uzlů stromů. Struktura odpovídala seznamu, kde každá jeho položka obsahovala pole uzlů stromu. Po experimentování byla stanovena vhodná velikost pole uzlů. Tento způsob alokování snížil výstavbu samotného stromu na cca $0,2$ sekundy, což vzhledem k velikosti datového souboru byla přijatelná doba. Díky zjednodušení uvolňování paměti se podařilo významně urychlit celkový program. Výsledné měření je dostupné v kapitole 2.

Tato implementace splnila mé očekávání a do dalších optimalizací jsem se již nepouštěl. Na závěr bych jen dodal, že možnost optimalizace navržené v zadání (předzpracování vstupních souborů) nepokládám za skutečnou optimalizaci, a proto jsem tuto možnost ignoroval.

## 2 Statistiky běhu programu

Zadání projektu specifikovalo minimální přípustnou rychlost na 80000 přiřazených IP adres k AS za jednu sekundu. Pro testování této skutečnosti byl použit unixový nástroj `time` na různě velkých zdrojích dat (IP adres). Výsledky měření pořízených na školním serveru **merlin** jsou uvedeny v následující tabulce.

| **Počet AS** | **Počet IP adres** | **Doba běhu (s)** | **Rychlost (IP/s)** |
|---|---|---|---|
| 464178 | 1 | 0.224 | - |
| 464178 | 178220 | 0.336 | 530416 |
| 464178 | 1782200 | 1.312 | 1358384 |
| 464178 | 8911000 | 5.356 | 1663741 |
| 464178 | 17822000 | 10.453 | 1704965 |

Z výsledků tohoto měření je patrné, že aplikace splňuje minimální rychlost specifikovanou v zadání a dokonce je až 21 krát rychlejší, a proto považuji cíl projektu za splněný. První řádky mají nižší rychlost, a to z důvodu velkého vlivu inicializace stromů na celkovou rychlost.

## 3 Nesplněné body zadání

Žádné.