Definice problØmu

VyhledÆvÆn podřetězce v textu (Substring matching). C lem je nalezen vešker ch v skytů hledanØho řetězce v zadanØm textu.

Naivn algoritmus

Datově nezÆvisl , se zaručenou složitost Θ(m*n). Spoč vÆv průchodu vešker ch možnost vzÆjemn ch pozic řetězců a jejich nÆsledn Øm porovnÆn .

PředčasnØukončen vnitřn ho cyklu může v ideÆln m př padě urychlit v počet až na Ω(n), horn složitost zůstÆvÆΘ(m*n).

Knuth - Morris - Pratt algoritmus

Algoritmus použ vÆpředzpracovÆn vzorku pro urychlenØvyhledÆvÆn . Dobu vyhledÆvÆn zkracuje pomoc posunu o v ce než jednu pozici v zÆvislosti na struktuře dat. Algoritmus nejprve vypln tabulku posunů a v druhØfÆzi pak spust samotnØ vyhledÆvÆn .

V př padě neshody znaku se pod vÆdo tabulky a provede předpoč tan posun v zÆvislosti na struktuře vzorku.

Pro v počet tabulky potřebujeme Θ(m) operac a pro samotnØvyhledÆvÆn Θ(n). PaměťovÆnÆročnosti je Θ(m) = Ω(m) pro tabulku posunů.

Měřen

Měřen času bylo provedeno na serveru star.fit.cvut.cz, pomoc knihovn funkce omp_get_wtime. KaždØměřen prob halo jednotn m způsobem, po načten vstupn ch hodnot programem byl změřen počÆteč n čas, spuštěn algoritmus v počtu a po jeho dokončen spoč tÆn rozd ln čas v sekundÆch. PřibližnÆhodnota MIPS byla měřena během zvlÆštn ho běhu programu, aby nebyl ovlivněn čas měřen .

Optimalizace

Protože oba algoritmy jsou silně datově závislá a oba mají různou složitost, měřil jsem jejich časy odděleně a na jiných instancích problému. Pro algoritmus KMP jsem použ val přibližně 16x větš vstupní řetězce než pro naivní implementaci.

Výsledky

Po zapnutí optimalizace o3 se průměrný čas běhu zkrátil přibližně na polovinu v důsledku optimalizace kompilátoru. Do srovnávacích cyklů jsem musel vložit nadbytečnou operaci, aby nedošlo k jejich odstranění v důsledku agresivní optimalizace.

Vylepšením, aplikovaným na oba algoritmy bylo rozdělení vstupního textu na části, které se vejdou do cache procesoru. Po změření jsem nezaznamenal větš zrychlení, protože už původní verze měla hodnotu cache-miss velice nízkou. A to i přes značně velikého vstupní řetězce, zatím pouze odhaduji, že v cevlášťnové verze bude na výpadky trpět výkon.

Dalším vylepšením použitím na oba přístupy byla změna datového typu řetězce z uint8_t na uint64_t. Po tómto průměrný čas klesl v závislosti na vstupu přibližně 6 – 8 krát pro oba algoritmy.

Na zrychlenou verzi naivního algoritmu jsem aplikoval ručně unrolling nejvnitřnějšího cyklu. Pro některé vstupy se čas tímto měř nezměnil, u jiných nastalo cca 30% zrychlení.

Mimo tyto opravy jsem změnil vnitřní cyklus naivního postupu kvůli vektorizaci.
Nevýhodou je nalezení všech rozdílů vůči textu místo prvního, kvůli nemožnosti větvení ve vektorizovaném cyklu. Tím se průměrná složitost dostala až na Ω(m^2), urychlení je dosaženo pouze díky vektorizaci, která následně sníží počet průchodů cyklem.
Protože ale došlo k logické změně algoritmu, nedá se výsledek porovnávat s ostatními.

Výsledky jsou vyneseny v přiložených grafech.

Navíc jsem vyhodnotil přibližné zkreslení doby výpočtu během výpočtu MIPS. Měřením došlo k průměrnému nárůstu času o 10 – 15%.