# Definice problému

Vyhledávání podřetězce v textu (Substring matching). Cílem je nalezení veškerých výskytů hledaného řetězce v zadaném textu.

## Naivní algoritmus

Datově nezávislý, se zaručenou složitostí Θ(m\*n). Spočívá v průchodu veškerých možností vzájemných pozic řetězců a jejich následném porovnání.

Předčasné ukončení vnitřního cyklu může v ideálním případě urychlit výpočet až na Ω(n), horní složitost zůstává Θ(m\*n).

## Knuth - Morris - Pratt algoritmus

Algoritmus používá předzpracování vzorku pro urychlené vyhledávání. Dobu vyhledávání zkracuje pomocí posunu o více než jednu pozici v závislosti na struktuře dat. Algoritmus nejprve vyplní tabulku posunů a v druhé fázi pak spustí samotné vyhledávání.

V případě neshody znaku se podívá do tabulky a provede předpočítaný posun v závislosti na struktuře vzorku.

Pro výpočet tabulky potřebujeme Θ(m) operací a pro samotné vyhledávání Θ(n). Paměťová náročnosti je Θ(m) = Ω(m) pro tabulku posunů.

# Měření

Měření času bylo provedeno na serveru star.fit.cvut.cz, pomocí knihovní funkce omp\_get\_wtime. Každé měření probíhalo jednotným způsobem, po načtení vstupních hodnot programem byl změřen počáteční čas, spuštěn algoritmus výpočtu a po jeho dokončení spočítán rozdílný čas v sekundách. Přibližná hodnota MIPS byla měřena během zvláštního běhu programu, aby nebyl ovlivněn čas měření.

# Optimalizace

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

# Výsledky

Po zapnutí optimalizací o3 se průměrný čas běhu zkrátil přibližně na polovinu v důsledku optimalizací 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 větší zrychlení, protože už původní verze měla hodnotu cache-miss velice nízkou. A to i přes značně veliké vstupní řetězce, zatím pouze odhaduji, že vícevláknová verze bude na výpadky trpět více.

Dalším vylepšením použitým na oba přístupy byla změna datového typu řetězce z uint8\_t na uint64\_t. Po té průměrný čas klesl v závislosti na vstupu přibližně 6 – 8 krát pro oba algoritmy.

Na zrychlenou verzi naivního algoritmu jsem aplikoval ruční unrolling nejvnitřnějšího cyklu. Pro některý vstupy se čas téměř nezměnil, u jiných nastalo cca 30% zrychlení.

Mimo tyto úpravy 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 násobně snížila 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%.