IVS profiling - Fitutubies

Výstup měření

Kalkulačka tráví 85-95% času mazáním prvků vektoru. Důvodem je, že po smazání prvku z vektoru se musí ostatní prvky přesunout doleva aby opět vznikla nepřerušovaná pamět. Protože prvky vektoru jsou v programu často mazány od začátku dochází k tomu že každý prvek se bude přesunovat až tolikrát na kolikáté pozici leží.

```
+   99.74%     0.00%  fcalc_profiling  fcalc_profiling      [.] main
-   99.72%     0.00%  fcalc_profiling  fcalc_profiling      [.] calcLib::solveEquation[abi:cxx11]
   - 99.72% calcLib::solveEquation[abi:cxx11]
      - 96.94% calcLib::solveBinaryOperation
         - 96.51% std::vector<Token, std::allocator<Token> >::erase
            + 96.47% std::vector<Token, std::allocator<Token> >::_M_erase
      + 1.77% calcLib::parseEquation
+   96.94%     0.02%  fcalc_profiling  fcalc_profiling      [.] calcLib::solveBinaryOperation
+   96.58%     0.01%  fcalc_profiling  fcalc_profiling      [.] std::vector<Token, std::allocator<Token> >::erase
+   96.53%     0.00%  fcalc_profiling  fcalc_profiling      [.] std::vector<Token, std::allocator<Token> >::_M_erase
```

Možná řešení

Ideálním řešením by bylo přepsat knihovnu aby používala bottom-up parser. To byl také původní plán ale z časových (a znalostních) důvodu jsem byl nucen přejít na jednoduší variantu.

Existující kód by měl jít přepsat tak aby používal std::list který má O(1) časovou komplexitu při mazání prvku uprostřed seznamu.

Profile guided optimization

Rozhodl jsem se nahradit std::vector za std::list. Jak je z tabulky níže patrné časová komplexita nové implementace roste v O(N) narozdíl od vektoru který rostl v O(N^2). Vyrovnanost prvních tří řadků je způsobena geniálností moderních překladačů. Profiling byl překládán s `-O3`. Pokud by jsme optimalizátor vypnuly tak se v levém sloupci nedopočítáme už u počtu 10000 prvků.

Tabulka časů běhu profilingu

Počet prvků	std::vector	std::list
10	4ms	4ms
100	5ms	5ms
1000	31ms	10ms
10000	1s 663ms	75ms
100000	3m 20s	723ms
1000000	N/A	7s 556ms