The Hash-Trie of Knuth & Liang
— Addendum —

Ştefan Vargyas

stvar@yahoo.com

Nov 28, 2015

Table of Contents

1 Introduction **1**

2 Mathematical Evaluations **1**

3 Applications to Hash-Trie **4**

A C++ Implementation Excerpts **5**

1 Introduction

The purpose of the following sections is to provide mathematical proofs for the claims made at the end of section 2.3, *The implementation of* `HashTrie<>` *Class Template*, on page 21 of the technical report *The Hash-Trie of Knuth & Liang: A C++11 Implementation*, **hash-trie-impl.pdf**:

> The replacement of expression "`h + tolerance - mod_x`" with the expression "`add(h, tolerance2) - mod_x`" had to be done, because, as it is easily provable, under cetain conditions, the subexpression "`h + tolerance`" is exceeding the upper bound `trie_size` of the boxed integer `pointer_t`. It is worthy of notice the readily provable fact that both assignments to `last_h` on lines 2523 and 2533 are correct: each of the expression on the right side of these assignments do not exceed upon evaluation the bounds of `pointer_t`.

The claim made by the first phrase is restated and proved by fact 17; the claim of the second phrase – by fact 19. Alongside these two facts, the section containing them establishes by proof a few more facts which concern the inner workings of the core algorithm of class `HashTrie<>`.

2 Mathematical Evaluations

1 Definition (Defining parameters).

(1) $trie\_size \in \mathbb{N}$,

(2) $max\_letter \in \mathbb{N}^*$.

(3) $tolerance \in \mathbb{N}$.

2 Definition (Dependent constants).

(4) $mod\_x \stackrel{\text{def}}{=} trie\_size - 2 \cdot max\_letter$,

(5) $alpha \stackrel{\text{def}}{=} \lceil 0.61803 \cdot mod\_x \rceil$,

(6) $max\_h \stackrel{\text{def}}{=} mod\_x + max\_letter$,

(7) $max\_x \stackrel{\text{def}}{=} mod\_x - alpha$.

3 Axioms (Initiating assertions)**.**

(8) $\quad alpha > 0,$

(9) $\quad tolerance > 0,$

(10) $mod\_x > 0,$

(11) $max\_h > tolerance,$

(12) $mod\_x > tolerance,$

(13) $mod\_x > alpha.$

4 Proposition.

(14) $trie\_size > 0,$

(15) $max\_h > mod\_x,$

(16) $(12) \implies (11),$

(17) $tolerance < trie\_size,$

(18) $max\_h = trie\_size - max\_letter,$

(19) $max\_h < trie\_size,$

(20) $(11) \iff max\_letter + tolerance < trie\_size.$

Proof. For (14): apply (10) and (4).
For (15): apply (6) and (2).
For (16): apply (15).
For (17): $tolerance \overset{(12)}{<} mod\_x \overset{(4)}{=} trie\_size - 2 \cdot max\_letter \overset{(2)}{<} trie\_size.$
For (18): apply (6) and (4).
For (19): apply (18) and (2).
For (20): (11) $\overset{(18)}{\iff} trie\_size - max\_letter > tolerance \iff max\_letter + tolerance < trie\_size.$ □

5 Definition (The *floor* function)**.**

(21) $\lfloor x \rfloor \overset{\text{def}}{=} \max \mathcal{M}(x) \in \mathbb{Z}$, for $x \in \mathbb{R}$,

where

(22) $\mathcal{M}(x) \overset{\text{def}}{=} \{k \in \mathbb{Z} \mid k \leq x\}$.

6 Proposition (Basic properties of "$\lfloor \cdot \rfloor$")**.**

(23) $\lfloor x \rfloor \leq x$, *for* $x \in \mathbb{R}$,

(24) $n = \lfloor n \rfloor$, *for* $n \in \mathbb{Z}$,

(25) $\lfloor x \rfloor \leq \lfloor y \rfloor \iff \mathcal{M}(x) \subseteq \mathcal{M}(y)$, *for* $x, y \in \mathbb{R}$,

(26) $x \leq y \implies \lfloor x \rfloor \leq \lfloor y \rfloor$, *for* $x, y \in \mathbb{R}$,

(27) $n \leq x \iff n \leq \lfloor x \rfloor$, *for* $n \in \mathbb{Z}$ *and* $x \in \mathbb{R}$,

(28) $n \leq x < n + 1 \iff \lfloor x \rfloor = n$, *for* $n \in \mathbb{Z}$ *and* $x \in \mathbb{R}$,

(29) $a \bmod b = a - \lfloor a/b \rfloor \cdot b$, *for* $a \in \mathbb{Z}$ *and* $b \in \mathbb{N}^*$.

Proof. For (23): $x \in \mathbb{R} \overset{(21)}{\implies} \lfloor x \rfloor \in \mathcal{M}(x) \overset{(22)}{\implies} \lfloor x \rfloor \leq x$.
For (24): by (23) $\lfloor n \rfloor \leq n$; $n \leq n \in \mathbb{Z} \overset{(22)}{\implies} n \in \mathcal{M}(n) \implies n \leq \max \mathcal{M}(n) \overset{(21)}{=} \lfloor n \rfloor$.
For (25): "$\Rightarrow$": $k \in \mathcal{M}(x) \implies k \leq \max \mathcal{M}(x) \overset{(21)}{=} \lfloor x \rfloor \overset{\text{hyp}}{\leq} \lfloor y \rfloor \overset{(21)}{\in} \mathcal{M}(y) \overset{(22)}{\implies} k \in \mathcal{M}(y)$; "$\Leftarrow$": $\lfloor x \rfloor \overset{(21)}{\in} \mathcal{M}(x) \overset{\text{hyp}}{\subseteq} \mathcal{M}(y) \implies \lfloor x \rfloor \in \mathcal{M}(y) \overset{(21)}{\implies} \lfloor x \rfloor \leq \lfloor y \rfloor$.
For (26): $x \leq y \overset{(22)}{\implies} \mathcal{M}(x) \subseteq \mathcal{M}(y) \overset{(25)}{\implies} \lfloor x \rfloor \leq \lfloor y \rfloor$.
For (27): "$\Rightarrow$": $n \leq x \overset{(26)}{\implies} n \overset{(24)}{=} \lfloor n \rfloor \leq \lfloor x \rfloor$; "$\Leftarrow$": $n \leq \lfloor x \rfloor \overset{(21)}{\in} \mathcal{M}(x) \overset{(22)}{\implies} n \leq x$.

For (28): $n \le x \overset{(27)}{\iff} n \le \lfloor x \rfloor$; $x < n+1 \overset{(27)}{\iff} \lfloor x \rfloor < n+1 \iff \lfloor x \rfloor \le n$.
For (29): firstly remark the uniqueness part of the *Euclidean division theorem*: for $x, y \in \mathbb{Z}$, with $y > 0$: if exists q', q'', r', $r'' \in \mathbb{Z}$ such that $x = q' \cdot y + r'$, $x = q'' \cdot y + r''$, with $0 \le r'$, $r'' < y$, then $q' = q''$ and $r' = r''$. The proof of this goes easily: suppose $r' > r''$. Then $r' - r'' = y \cdot (q'' - q')$; follows that $q'' - q' > 0$; thus $q'' - q' \ge 1$; hence $r' - r'' \ge y$, which contradicts $r' - r'' < y \iff 0 \le r'$, $r'' < y$. Now, this uniqueness property leads to (29) if shown that $0 \le a - b \cdot \lfloor a/b \rfloor < b$. Indeed the relation holds true: $n = \lfloor a/b \rfloor \overset{(28)}{\iff} n \le a/b < n+1 \iff b \cdot n \le a < b \cdot (n+1) \iff 0 \le a - b \cdot n < b$. ☐

7 Proposition. *For* a, b, $x \in \mathbb{Z}$*:*

(30) $0 \le a < b \implies a \bmod b = a$,
(31) $0 < b \le a < 2 \cdot b \implies a \bmod b = a - b$,
(32) $0 \le x < max\_x \implies (x + alpha) \bmod mod\_x = x + alpha$,
(33) $max\_x \le x < mod\_x \implies (x + alpha) \bmod mod\_x = x - max\_x$.

Proof. For (30): $0 \le a < b \implies 0 \le a/b < 1 \overset{(28)}{\implies} \lfloor a/b \rfloor = 0 \overset{(29)}{\implies} a \bmod b = a$.
For (31): $0 < b \le a < 2 \cdot b \implies 1 \le a/b < 2 \overset{(28)}{\implies} \lfloor a/b \rfloor = 1 \overset{(29)}{\implies} a \bmod b = a - b$.
For (32): $x < max\_x \overset{(7)}{\iff} x + alpha < mod\_x$; $x \ge 0 \overset{(8)}{\implies} x + alpha > 0$; then (30) implies (32).
For (33): $x \ge max\_x \overset{(7)}{\iff} x + alpha \ge mod\_x$; $alpha \overset{(13)}{<} mod\_x$ and $x \overset{\text{hyp}}{<} mod\_x \implies x + alpha < 2 \cdot mod\_x$; the latter two consequents along with (10) conclude to $(x + alpha) \bmod mod\_x \overset{(31)}{=} x + alpha - mod\_x \overset{(7)}{=} x - max\_x$. ☐

8 Notation. For $h \in \mathbb{N}^*$ let $h' \overset{\text{def}}{=} h + tolerance \in \mathbb{N}^*$ and $h'' \overset{\text{def}}{=} h' - mod\_x \in \mathbb{Z}$.

9 Proposition.

(34) $h \le max\_h - tolerance \iff h' \le max\_h$,
(35) $h \ge max\_letter + 1 \implies h' > max\_letter + 1$,
(36) $h \le max\_h \iff h'' \le max\_letter + tolerance$,
(37) $h > max\_h - tolerance \iff h'' > max\_letter$,
(38) $max\_h - tolerance < h \le max\_h \iff max\_letter < h'' \le max\_letter + tolerance$,
(39) $max\_letter + tolerance \le max\_h$,
(40) $max\_letter < h \le max\_h \iff 0 \le max\_h - h < mod\_x$,
(41) $max\_h + tolerance > trie\_size \iff tolerance > max\_letter$.

Proof. For (34): $h \le max\_h - tolerance \iff h' = h + tolerance \le max\_h$.
For (35): $h' = h + tolerance \overset{\text{hyp}}{\ge} max\_letter + 1 + tolerance \overset{(9)}{>} max\_letter + 1$.
For (36): $h \le max\_h \iff h'' = h + tolerance - mod\_x \le max\_h - mod\_x + tolerance \overset{(6)}{=} max\_letter + tolerance$.
For (37): $h > max\_h - tolerance \iff h'' = h + tolerance - mod\_x > max\_h - mod\_x \overset{(6)}{=} max\_letter$.
For (38): apply (36) and (37).
For (39): $max\_letter + tolerance \overset{(12)}{\le} max\_letter + mod\_x \overset{(6)}{=} max\_h$.
For (40): $max\_letter < h \le max\_h \iff 0 \le max\_h - h < max\_h - max\_letter \overset{(6)}{=} mod\_x$.
For (41): $max\_h + tolerance > trie\_size \iff tolerance > trie\_size - max\_h \overset{(18)}{=} max\_letter$. ☐

10 Proposition.

(42) $max\_letter + 1 \le h \le max\_h - tolerance \implies max\_letter + 1 \le h' \le max\_h$,
(43) $max\_h - tolerance < h \le max\_h \implies max\_letter + 1 \le h'' \le max\_h$.

Proof. For (42): by (35) $h' > max\_letter + 1$, therefore $h' \ge max\_letter + 1$. By (34) $h' \le max\_h$.
For (43): by (37) $h'' > max\_letter$, therefore $h'' \ge max\_letter + 1$. By (36) and (39) $h'' \le max\_h$. ☐

3 Applications to Hash-Trie

11 Fact. *The definitions (1), (2) and (3) correspond to lines #2042, #2045 and #2043 respectively of the* **C++** *implementation.*

12 Fact. *The definitions (4), (5), (6) and (7) correspond to lines #2267, #2266, #2268 and #2269 respectively of the* **C++** *implementation.*

13 Fact. *The axioms (8)–(13) correspond one by one to the* `CXX_ASSERT` *lines #2271–#2276.*

14 Fact. *The assignment on line #2445 is correct.*

Proof. Due to the definition of `pointer_t`, the statement on line #2445 is correct if and only if `tolerance` is greater or equal than `0` and less or equal than `trie_size` (these are the limiting bounds of `pointer_t`). From (9) and (17) results that the line #2445 is indeed correct. □

15 Fact. *The variable* `x` *declared, initialized and maintained on lines #2231, #2318 and respectively #2477–#2480 is iterating correctly the elements of the sequence* $(x_n)_{n\in\mathbb{N}}$*, where* $x_n \stackrel{\text{def}}{=} (alpha \cdot n) \bmod mod\_x$ *for* $n \in \mathbb{N}$*.*

Proof. Proceed by induction on $n \in \mathbb{N}$. For $n = 0$ the statement made above holds, since the line #2318 is showing that the initial value of `x` is `0`. Now suppose that prior to executing line #2477 `x` has the value of x_n for some $n \in \mathbb{N}$. In view of the relations (32) and (33), upon the execution of lines #2477–#2480, `x` becomes `(x + alpha)` *mod* `mod_x`. Taking into account that, by the definition of sequence $(x_n)_{n\in\mathbb{N}}$, $x_{n+1} = (x_n + alpha) \bmod mod\_x$, indeed `x` is x_{n+1} after the line #2480. □

16 Fact. *The assertion stated within the comment on line #2482 is correct.*

Proof. Need to prove that upon executing line #2482, $max\_letter < h \leq trie\_size - max\_letter \stackrel{(18)}{=} max\_h$: Fact 15 $\Longrightarrow 0 \leq x < mod\_x \stackrel{(6)}{=} max\_h - max\_letter \iff -1 < x \leq max\_h - max\_letter - 1 \iff max\_letter < h \stackrel{\#2482}{=} x + max\_letter + 1 \leq max\_h$. □

17 Fact. *Under certain conditions, the result of evaluating the expression* `add(h, tolerance2)` *on line #2523 exceeds the value of* `trie_size` *for some* `h`*.*

Proof. By fact 16: $max\_letter < h \leq max\_h$. If let $h \stackrel{\text{def}}{=} max\_h$, then, under the condition that $tolerance > max\_letter$, (41) shows that $h + tolerance > trie\_size$ indeed. □

18 Remark. The fact above indicates that the expression `h + tolerance` wouldn't have been a proper choice of coding the line #2523: in the case of $h + tolerance > trie\_size$, the evaluation of the expression `h + tolerance` would have caused the program to halt abruptly (assuming that the configuration parameter `CONFIG_HASH_TRIE_STRICT_TYPES` was `#define`d at compile-time).

19 Fact. *The assignments to variable* `last_h` *on lines #2523 and #2533 are both correct. Upon the execution of either of them,* $max\_letter + 1 \leq last\_h \leq max\_h$*.*

Proof. The statements on lines #2523 and #2533 are correct if and only if each of the expression on the right side of the respective assignments evaluates to an integer not exceeding the bounds of type `pointer_t`. By the fact 16, before executing each of the two lines: $max\_letter < h \leq max\_h$. Now, for the case of line #2523 apply (43) (from line #2506: $h > max\_h - tolerance$) and, respectively, for the case of line #2533 apply (42) (from line #2506: $h \leq max\_h - tolerance$). Both give that $max\_letter + 1 \leq last\_h \leq max\_h$. Consequently, the bounds of `pointer_t` are respected: by (2) and (19), the previous double inequality yields: $0 < last\_h < trie\_size$. □

20 Fact. *The inner loops of method* `HashTrie<>::find` *(not displayed by the listing below) that are based on* `h` *computed by lines #2545–#2551 are finite.*

Proof. By the fact 16, each of these loops start iterating with an `h` satisfying $max\_letter + 1 \leq h \leq max\_h$. The lines #2545–#2551 show that `h` is incremented circularly within the boundaries $max\_letter + 1$ and max_h. By the fact 19, $max\_letter + 1 \leq last\_h \leq max\_h$ on each execution of lines #2545–#2551, i.e. `last_h` lies between the same boundaries as `h`. The implementation code also shows (not seen below, though) that `last_h` is an invariant of each of these loops. Consequently, `h` has to meet `last_h` upon a finitely many succesive calls of the lambda function `compute_the_next_trial_header_location`. This leads the lambda function to return `false` – thus terminating the iterations. □

A C++ Implementation Excerpts

```cpp
template<
    typename C = char,
    template<typename> class T = char_traits_t,
    typename S = size_traits_t>
class HashTrie :
    private T<C>,
    private S
{
public:
    typedef S size_traits_t;
    typedef T<C> char_traits_t;
...
private:
...
    using size_traits_t::trie_size;
    using size_traits_t::tolerance;
...
    using char_traits_t::max_letter;
...
    // x_n = (alpha * n) % mod_x
    pointer_t x;
...
    pointer_t find(const char_t*);
...
    static constexpr size_t make_alpha(size_t trie_size, size_t max_letter)
...
    { return std::ceil(0.61803 * (trie_size - 2 * max_letter)); }
...
    static constexpr size_t alpha = make_alpha(trie_size, max_letter);
    static constexpr size_t mod_x = trie_size - 2 * max_letter;
    static constexpr size_t max_h = mod_x + max_letter;
    static constexpr size_t max_x = mod_x - alpha;

    CXX_ASSERT(alpha > 0);
    CXX_ASSERT(tolerance > 0);
    CXX_ASSERT(trie_size > 2 * max_letter);
    CXX_ASSERT(max_h > tolerance);
    CXX_ASSERT(mod_x > tolerance);
    CXX_ASSERT(mod_x > alpha);
...
};

template<
    typename C,
    template<typename> class T,
    typename S>
HashTrie<C, T, S>::HashTrie()
{
...
    x = 0;
}

template<
    typename C,
    template<typename> class T,
    typename S>
typename
    HashTrie<C, T, S>::pointer_t
    HashTrie<C, T, S>::find(const char_t* str)
{
...
    const pointer_t tolerance2 = tolerance;
    // trial header location
    pointer_t h;
    // the final one to try
    pointer_t last_h; // INT: int last_h;

    const auto get_set_for_computing_header_locations = [&]() {
        // 24. Get set for computing header locations
...
        if (x >= max_x)
            x -= max_x;
        else
            x += alpha;

        h = x + max_letter + 1; // now max_letter < h <= trie_size - max_letter
...
```

```
        if (h > max_h - tolerance) {
...
            last_h = add(h, tolerance2) - mod_x;
        }
        else {
...
            last_h = h + tolerance;
        }
    };

    const auto compute_the_next_trial_header_location = [&]() {
        // 25. Compute the next trial header location h, or abort find
...
        if (h == last_h)
            return false;
        if (h == max_h)
            h = max_letter + 1;
        else
            h ++;
        return true;
    };
...
}
```