Verification of the Garsia–Wachs Algorithm

Makoto Kanazawa
Hosei University

Garsia–Wachs Algorithm in Dafny

- Dafny formalization of Kingston's (1988) correctness proof as presented by Knuth (1998).
- A simpler, easier-to-verify variant of the algorithm.

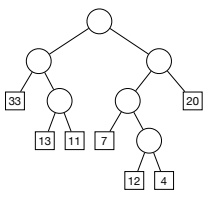
A first formalized proof of correctness of the Garsia–Wachs algorithm.

What I did in this work was to formalize in Dafny the correctness of the Garsia–Wachs algorithm that was given in Knuth's book, *The Art of Computer Programming*, Volume 3, which is originally due to Kingston.
In so doing, I came up with a variant of the algorithm that is both simpler and easier to prove correct.
To my knowledge, this is the first time that the Garsia–Wachs algorithm has been formalized.

Garsia–Wachs Algorithm

Input 33 13 11 7 12 4 20 weight sequence

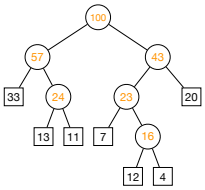
Output



optimum "alphabetic" binary tree

The Garsia–Wachs algorithm takes a sequence of nonnegative weights as input and outputs a binary tree where the leaves are labeled by weights such that the sequence of leaf labels exactly matches the input sequence.
Among such trees, it's supposed to output one whose "cost" is minimum.
Here, "alphabetic" means that the leaf labels are ordered in exactly the same way as the input sequence.

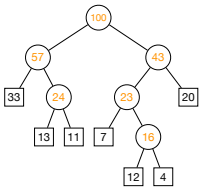
Cost



- Attach to each internal node the total weight of the subtree rooted at that node.
- The **cost** of the tree is the sum of all internal node values.

The cost of such a binary tree is defined as shown on the slide.
If this is a binary search tree where all data are located on the leaves and the weight of a leaf corresponds to the probability of reaching that leaf, then the cost of the tree corresponds to the expected number of nodes that are visited.

Cost



sum of the internal nodes

$$100 + 57 + 24 + 43 + 23 + 16 = 263$$

sum of the leaves multiplied by their levels

$$33 \times 2 + 13 \times 3 + 11 \times 3 + 7 \times 3 + 12 \times 4 + 4 \times 4 + 20 \times 2 = 263$$

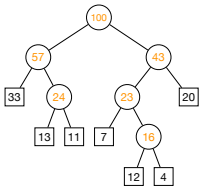
$$\text{Cost}(T) = \sum_{i=0}^{n} q_i l_i$$

q_i = weight of ith leaf

l_i = level of ith leaf

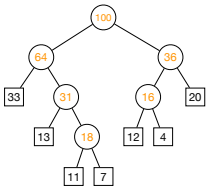
An alternative way of computing the cost is to multiply the weight of each leaf by its its distance from the root and sum the results.
The *n* in the formula is the length of the input sequence minus one.

Cost



$$57 + 24 + 16 + 23 + 43 + 100 = 263$$

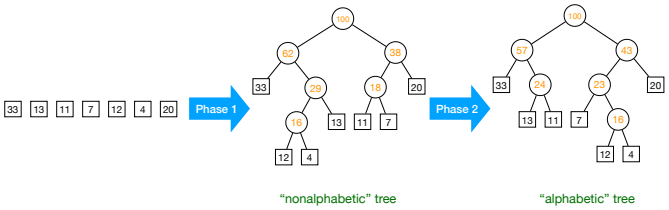
minimum cost



$$18 + 31 + 64 + 16 + 36 + 100 = 265$$

In this input sequence, (12, 4) is the pair of adjacent weights whose sum is the smallest. When (12, 4) is replaced by their sum, the minimum weight adjacent pair is (11, 7). For the purpose of building an optimum alphabetic tree, the strategy of repeatedly combining minimum weight adjacent pairs does not work.

Garsia–Wachs Algorithm

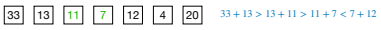


"nonalphabetic" tree "alphabetic" tree

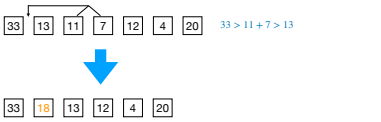
The algorithm consists of two phases.
The first phase constructs a tree whose leaf sequence is a rearrangement of the input weight sequence.
The second phase turns this tree into a minimum cost alphabetic tree for the input sequence.

GW Algorithm, Phase 1

- Locate the leftmost locally minimal adjacent pair of weights.



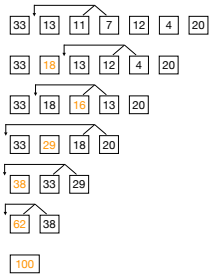
- Combine it into one weight and move it past smaller weights to its left.



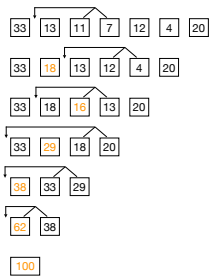
Let's look at how each of the two phases of the algorithm works.

GW Algorithm, Phase 1

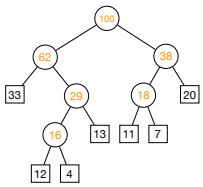
- Repeat until just one weight remains.



GW Algorithm, Phase 1

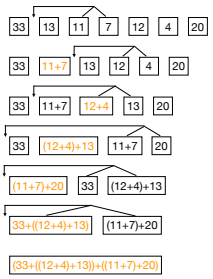


- Build a “nonalphabetic” tree recording which adjacent pairs were combined.

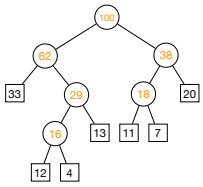


Construct an internal node for each adjacent pair that was combined.

GW Algorithm, Phase 1

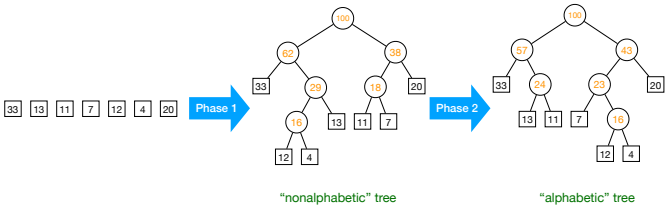


- Build a “nonalphabetic” tree recording which adjacent pairs were combined.



The tree is the expression for the final sum.

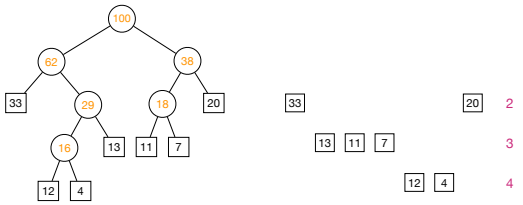
Garsia–Wachs Algorithm



That was the first phase of the algorithm.
Let’s now look at the second phase.

GW Algorithm, Phase 2

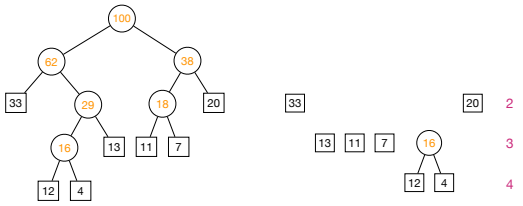
- Build an alphabetic tree where each weight appears at the same level.



Each element of the input sequence corresponds to a leaf of the tree output by Phase 1. Compute each leaf's level, i.e., its distance from the root.

GW Algorithm, Phase 2

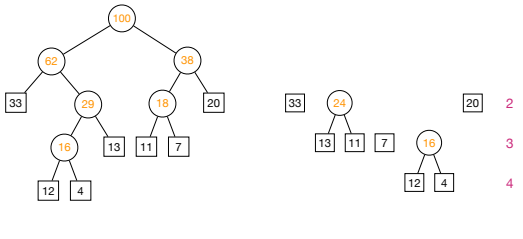
- Build an alphabetic tree where each weight appears at the same level.



There is exactly one alphabetic tree each of whose leaves is at the required level.

GW Algorithm, Phase 2

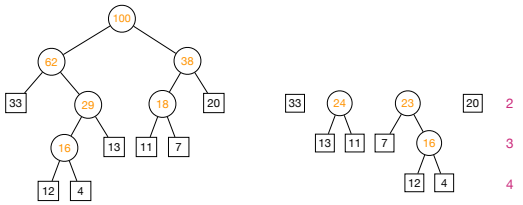
- Build an alphabetic tree where each weight appears at the same level.



There is exactly one alphabetic tree each of whose leaves is at the required level.

GW Algorithm, Phase 2

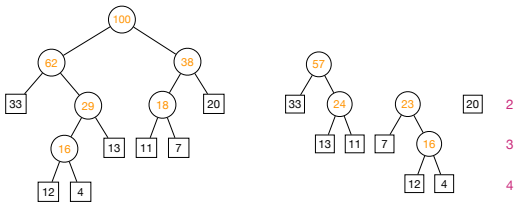
- Build an alphabetic tree where each weight appears at the same level.



There is exactly one alphabetic tree each of whose leaves is at the required level.

GW Algorithm, Phase 2

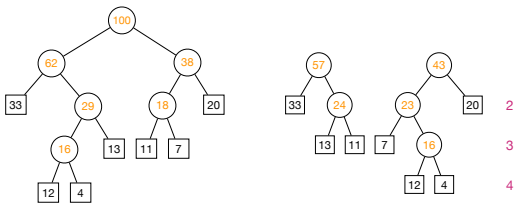
- Build an alphabetic tree where each weight appears at the same level.



There is exactly one alphabetic tree each of whose leaves is at the required level.

GW Algorithm, Phase 2

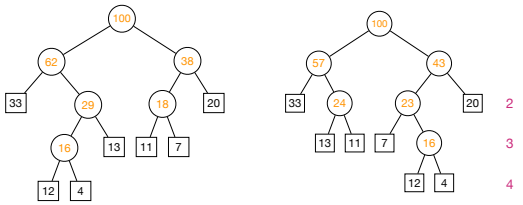
- Build an alphabetic tree where each weight appears at the same level.



There is exactly one alphabetic tree each of whose leaves is at the required level.

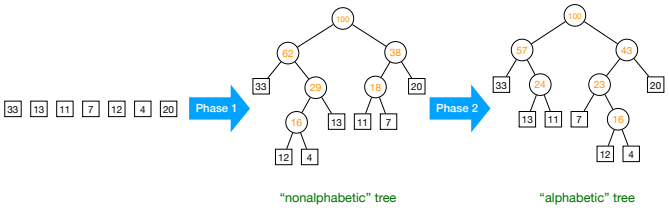
GW Algorithm, Phase 2

- Build an alphabetic tree where each weight appears at the same level.



There is exactly one alphabetic tree each of whose leaves is at the required level.

Garsia–Wachs Algorithm



The intermediate nonalphabetic tree and the final output tree are "level-equivalent" in the sense that each weight occurs at the same level. Other than that, it's difficult to exactly characterize the intermediate tree.

History

1959 • Gilbert and Moore $O(n^3)$ algorithm

1971 • Knuth $O(n^2)$ algorithm

1971 • Hu and Tucker $O(n^2)$ algorithm

"Professor D. E. Knuth informed us that a better implementation of our algorithm needs only $O(n \log n)$ operations when suitable data structures are employed."

1973 • Knuth, *The Art of Computer Programming*, Vol. 3, First Edition

"no simple proof [of the Hu–Tucker algorithm] is known, and it is quite possible that no simple proof will ever be found."

The problem that the Garsia–Wachs algorithm addresses was first considered by Gilbert and Moore in 1959.

History

1977 • Garsia and Wachs $O(n^2)$ algorithm
$O(n \log n)$ implementation due to Tarjan

"… closely related to the Hu-Tucker algorithm, but easier to verify".

1988 • Kingston

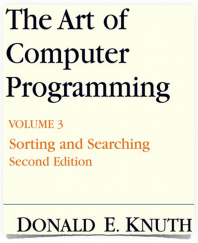
"… our proof of the Garsia–Wachs algorithm is the first to permit an intuitive understanding of why this class of algorithms is correct."

1998 • Knuth, *The Art of Computer Programming*, Vol. 3, Second Edition

"Simplifications [of the Hu-Tucker algorithm] … were found several years later by A. M. Garsia and M. L. Wachs, … although their proof was still rather complicated. Lemmas W, X, Y, and Z above are due to J. H. Kingston …"

The second edition of Knuth's book adopted the Garsia–Wachs algorithm together with Kingston's correctness proof.

History



- Knuth's pseudocode description of the Garsia–Wachs algorithm initially contained a serious bug, which was corrected in the 17th (2004) printing.
- Corrected $O(n^2)$ implementation in C (2004).
- A number of substantial corrections, both to the correctness proof and to the pseudocode description of the algorithm, were made between the first (1998) and the 2009 printing.

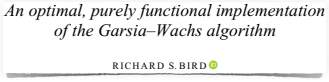
Knuth presented a more sophisticated version of the Garsia–Wachs algorithm (along the lines of Tarjan), but its time complexity was still quadratic. A more efficient O(*n* log *n*)-time implementation was left as an exercise.

History

2020 •

$O(n \log n)$ implementation in Haskell

"[The Garsia–Wachs algorithm] is fairly easy to describe … but even the best current proof of its correctness has some tricky details …"



"It is difficult to give a convincing intuition as to why this process works …"

Phase 1

Bird and Gibbons gave a purely functional implementation of the Garsia–Wachs algorithm whose time complexity is *O*(*n* log *n*), but did not prove its correctness (or its time complexity).

Kingston-Knuth Correctness Proof

Suppose $q_{i-1} > q_{i+1} \iff q_{i-1} + q_i > q_i + q_{i+1}$

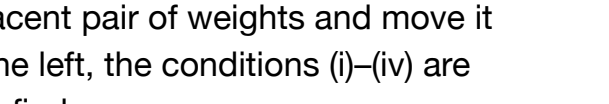
(i) $q_{i-1} > q_{i+1}$ for $1 \le i < k$,

(ii) $k = n$ or $q_{k-1} \le q_{k+1}$,

(iii) $q_i < q_{k-1} + q_k$ for $j \le i < k-1$,

(iv) $j = 0$ or $q_{j-1} \ge q_{k-1} + q_k$.

$(q_0, \ldots, q_n)$ input weight sequence



$(q'_0, \ldots, q'_{n-1}) =$
$(q_0, \ldots, q_{j-1}, q_{k-1} + q_k, q_j, \ldots, q_{k-2}, q_{k+1}, \ldots, q_n)$

Lemma X.
There is an optimum tree in which $l_0 \le \ldots \le l_{k-1} = l_k$ and either

(a) $l_j = l_k - 1$ or

(b) $l_j = l_k$ and $\boxed{j}$ is a left child.

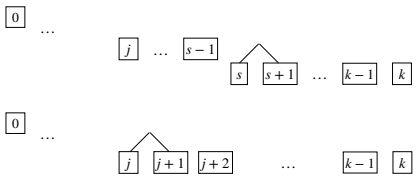
l_i level of ith leaf

$\boxed{i}$ ith leaf

Lemma X.
There is an optimum tree in which $l_0 \le \ldots \le l_{k-1} = l_k$ and either

(a) $l_j = l_k - 1$ or

(b) $l_j = l_k$ and $\boxed{j}$ is a left child.



$(q'_0, \ldots, q'_{n-1}) = (q_0, \ldots, q_{j-1}, q_{k-1} + q_k, q_j, \ldots, q_{k-2}, q_{k+1}, \ldots, q_n)$

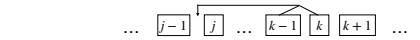
Lemma Y.

$$C(q'_0, \ldots, q'_{n-1}) \le C(q_0, \ldots, q_n) - (q_{k-1} + q_k)$$

minimum cost

Take an optimum tree T for $(q_0, \ldots, q_n)$ given by Lemma X, and let c be its cost.
T can be turned into a tree for $(q'_0, \ldots, q'_{n-1})$ with cost $c - (q_{k-1} + q_k)$.

Let's look at the Kingston–Knuth proof.
In Phase 1, when you combine an adjacent pair of weights and move it to the left, the conditions (i)–(iv) are satisfied.



$$(q'_0, \ldots, q'_{n-1}) = (q_0, \ldots, q_{j-1}, q_{k-1} + q_k, q_j, \ldots, q_{k-2}, q_{k+1}, \ldots, q_n)$$

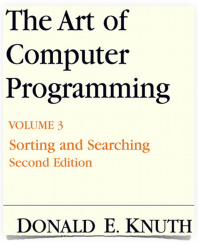
Lemma Z.

$$C(q'_0, \ldots, q'_{n-1}) = C(q_0, \ldots, q_n) - (q_{k-1} + q_k)$$

minimum cost

Any optimum tree U for $(q'_0, \ldots, q'_{n-1})$ (with cost c') can be turned into a tree T for $(q_0, \ldots, q_n)$ with cost $c \le c' + (q_{k-1} + q_k)$.

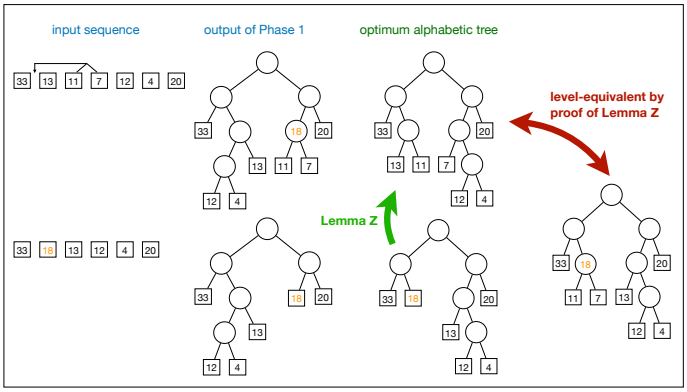
Since U is optimum, $c \ge c' + (q_{k-1} + q_k)$ and T is optimum by Lemma Y.



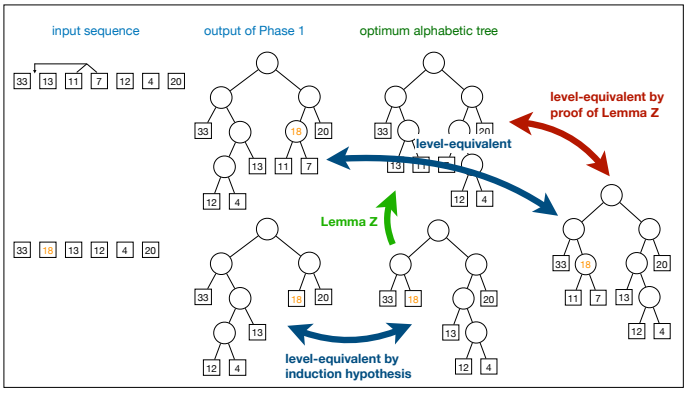
"These lemmas show that the problem for $n + 1$ weights $q_0, q_1, \ldots, q_n$ can be **reduced** to an n-weight problem … The proofs also show that any optimum tree T' that is obtained from the new weights $(q'_0, \ldots, q'_{n-1})$ can be rearranged into a tree T that has the original weights $(q_0, \ldots, q_n)$ in the correct left-to-right order; moreover, each weight will appear at the same level in both T and T'."
[Emphasis mine.]

But the algorithm does **not** construct an optimum tree for $(q'_0, \ldots, q'_{n-1})$!!

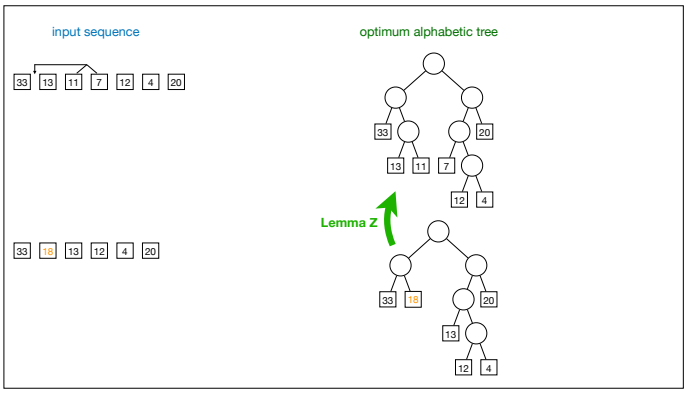
Here's what Knuth wrote after the proof of Lemma Z.



Additional reasoning necessary to verify Phase 1.



In order to show that the output of Phase 1 is level-equivalent to an optimum alphabetic tree, you have to use the induction hypothesis applied to the shorter weight sequence. You have to refer to trees that the algorithm never builds. Neither Kingston nor Knuth spells out this inductive argument.



Lemma Z give a way of turning an optimum tree for the shorter sequence into an optimum tree for the original, longer sequence.
If you make the entire algorithm recursive, this is all you need.

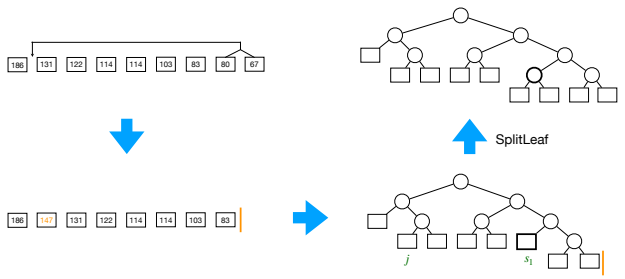
A Closer Look at Lemma Z

$$(q'_0, \ldots, q'_{n-1}) = (q_0, \ldots, q_{j-1}, q_{k-1} + q_k, q_j, \ldots, q_{k-2}, q_{k+1}, \ldots, q_n)$$

Lemma Z.
Any optimum tree T' for $(q'_0, \ldots, q'_{n-1})$ (with cost c') can be turned into a tree T for $(q_0, \ldots, q_n)$ with cost $c = c' + (q_{k-1} + q_k)$.

T' $\qquad$ $T = \text{SplitLeaf}(T, s_1)$

T is the result of splitting the leaf at index s_1 of T'.

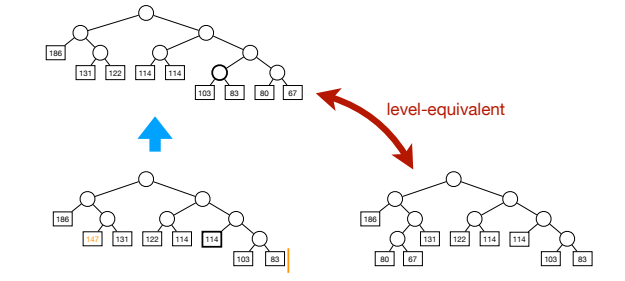


s_1 is the greatest index $\leq k-1$ such that $l'_j = l'_{s_1}$.

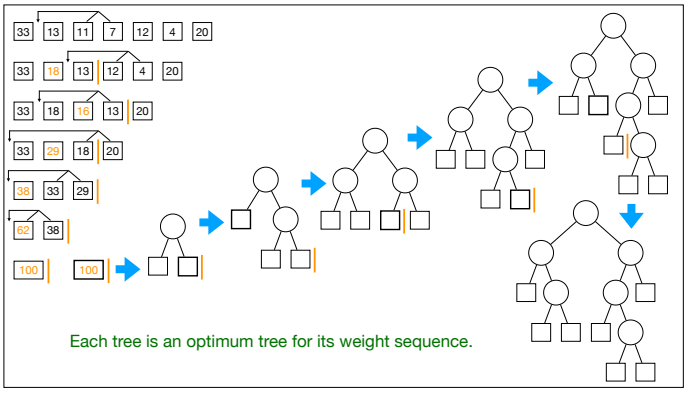
Between the moved item and its original position, you split the **rightmost leaf** that is at the same level as the moved item.

A One-Phase Algorithm

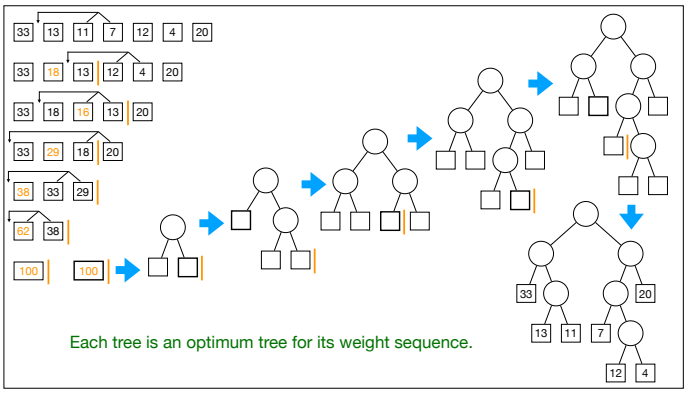
1. Given a weight sequence $(q_0, \ldots, q_n)$, return a single node tree if $n = 0$. Otherwise, find the k such that
   (a) $q_{i-1} > q_{i+1}$ for $1 \leq i < k$, and
   (b) either $k = n$ or $q_{k-1} \leq q_{k+1}$.
2. Find the largest $j < k$ such that either $j = 0$ or $q_{j-1} \geq q_{k-1} + q_k$, and let $(q'_0, \ldots, q'_{n-1}) = (q_0, \ldots, q_{j-1}, q_{k-1} + q_k, q_j, \ldots, q_{k-2}, q_{k+1}, \ldots, q_n)$.
3. Recursively call this procedure with the new weight sequence $(q'_0, \ldots, q'_{n-1})$, obtaining a tree T' with level sequence $(l'_0, \ldots, l'_{n-1})$.
4. Let s_1 be the greatest index $\leq k-1$ such that $l'_j = l'_{s_1}$. Return SplitLeaf(T', s_1).



This property is not necessary for the correctness proof of the one-phase algorithm!



The entire run of the one-phase algorithm on the sequence (33, 13, 11, 7, 12, 4, 20).



You don’t need to look at the weights during the top-down construction of trees.

Two-Phase vs. One-Phase

Two-phase algorithm	One-phase algorithm
× Bottom-up construction of nonalphabetic tree followed by reconstruction. × Some additional reasoning over and above Knuth’s lemmas is necessary to prove correctness. × Need to relate the levels of nonalphabetic and alphabetic trees.	✓ Direct top-down construction of alphabetic tree. ✓ Knuth’s lemmas suffice. ✓ No need to relate the levels of two trees.
all previous works	this paper

Dafny

- Syntax looks like an ordinary (imperative) programming language.
- Specification of
  - pre- and post-conditions of methods and functions,
  - termination metrics,
  - loop invariants.
- Lemmas are methods of a special kind.
- Automatic proof via SMT solver Z3.

Sequences in Dafny

- `seq<T>` is the type of sequences consisting of elements of type `T`.
- The length of a sequence `s` is `|s|`.
- `s[i]` is the element of `s` at position `i`.
- `s[lo .. hi]`
- The last two operations are **partial**.

Sequences in Dafny are quite convenient to work with in formalizing the proof of correctness of the Garsia–Wachs algorithm.

Some Formalization Decisions

- Use unlabelled trees.

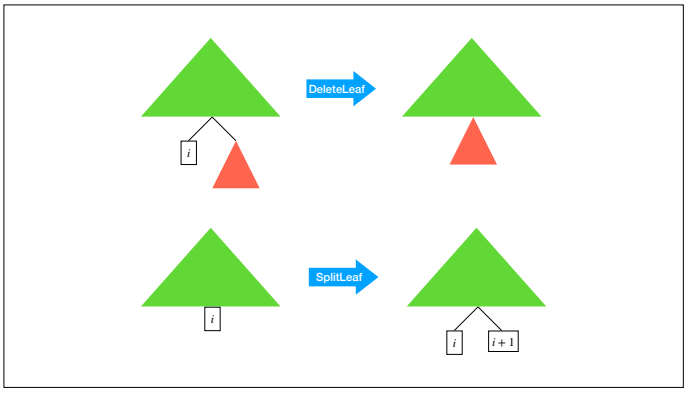
```
datatype Tree = Leaf | Node(left: Tree, right: Tree)

function Cost(t: Tree, qs: seq<int>): int
  requires |qs| == LeafCount(t)
```

- Avoid complicated operations on trees in the correctness proof.

```
function DeleteLeaf(t: Tree, i: nat): (t': Tree)
  requires t.Node?
  requires i < LeafCount(t)
  ensures LeafCount(t') == LeafCount(t) - 1

function SplitLeaf(t: Tree, i: nat): (t': Tree)
  requires i < LeafCount(t)
  ensures LeafCount(t') == LeafCount(t) + 1
```



Conclusion

- Verified the correctness of the Garsia–Wachs algorithm for building minimum cost alphabetic tree in Dafny.
- The alphabetic tree can be constructed directly, without detour through a nonalphabetic tree; the method is implicit in the proof of Lemma Z. This leads to a **more natural**, **easier-to-verify** variant of the algorithm.
- A one-phase implementation of the optimized version of the algorithm (running in time $O(n \log n)$) is also possible.