N-Dimensional Array
Ziggurat Indices

Contact info@adligo.com
Author Scott Morgan
Dates 2022-02-01 – 2022-02-20

Table of Contents

Abstract Summary

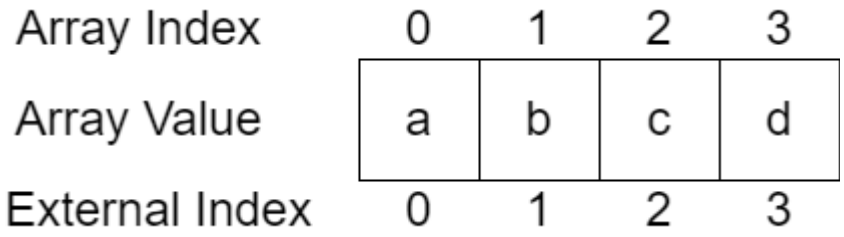
This paper explores the space time complexity of the use of a unbounded universe of items referenced in arrays to solve the problem of indexing. The basic structure involves starting with a simple array of slots for items. Then in order to grow the data structure past the initial array length an additional array is created with it's zero slot pointing to the original array, a pattern established with the migration to a Merkel Forrest from a Merkel Tree.

When this recursive growth structure is combined with bit masking and shifting techniques similar to those in use in Van Emde Boas trees interesting efficiencies can be achieved in the classic space time complexity trade offs.

Basic Ziggurat Structure

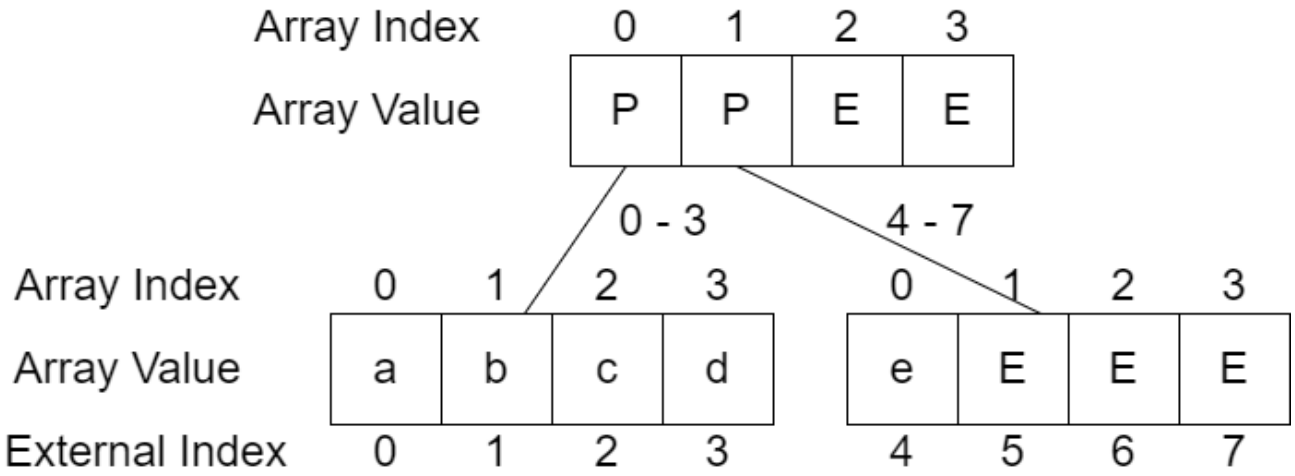
This basic structure starts with an array with a size of 2 ^ B. Where **B is the base,** and also allows for a configurable base, for example in this diagram the base is four. Also note the base four structure is used through out this paper to provide visual simplicitiy.

Figure 1:



Then once you need the ability to add a 4th item to the data structure a new array is created, which references the old array as one of it's elements. I have called this a ziggurat because of it's flat top shape which is different from a pyramid. Although this has been more commonly called a tree, the most common upside down whiteboard representation of CS trees look more pyramids to me. The following Figure 2 diagram illustrates this next stage in the data structure.

Figure 2:

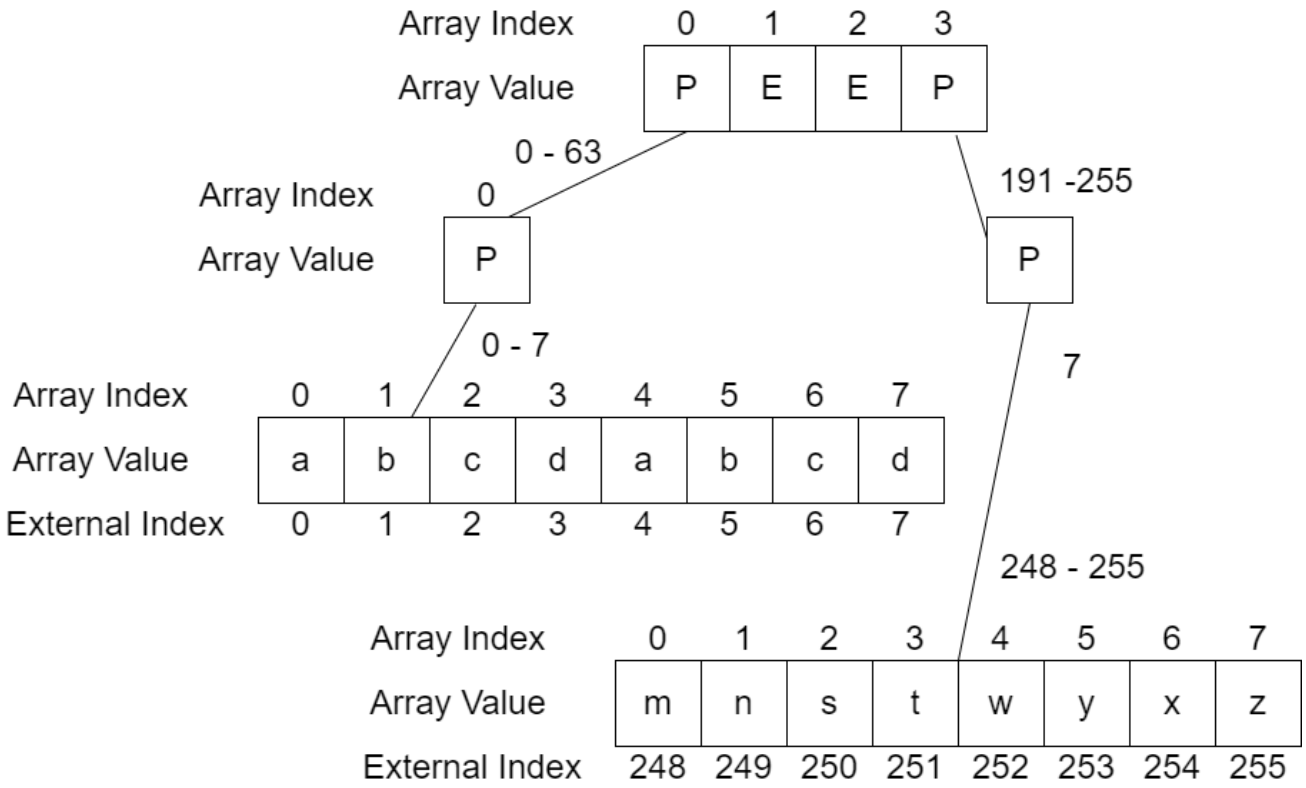


Note in the above diagram the capital P represents a present reference and the capital E represents a empty reference.

Initial Space Complexity / Cost of a Ziggurat Index

The space cost is complex to calculate with something like Asymptotic Analysis for a number of reasons. With the simple structure it is definitely greater than O(n) noting the empty slots and slots used for present references. In addition having a variable base, which also plays into the typical base 2 of Asymptotic Analysis causes issues. Finally the space usage is quite different for sparse and dense usage. Noting these challenges I believe the space to be between O(n^2) for sparse usage and O(log n) for dense usage.

Additionally, in the spirit of configuration if we compress the size of the arrays at or near the bottom (often called leaves in tree structures). We have the ability to trade time efficiency for space efficiency. The following Figure 3 diagram illustrates how this compression could occur.



Note the two 0-7 arrays of size 8 are on the same tier of the ziggurat, they were offset to make the diagram fit on the page better. Also note how the configurable nature of this data structure provides the ability for the user of the data structure to optimize a structure already in use with simple configuration.

Comparison of the Ziggurat with more common Data Structures

Also note the following interesting properties of this data structure when compared with other common data structures;

Operation	Ziggurat	ArrayList	Hashtable
Add			

While multiple dimensional arrays are NOT a new topic in programming, computer science or software engineering However, I have found the use of

Citations

Merkel Forrests
https://www.youtube.com/watch?v=Bqs_LzBjQyk

Van Embe Boas trees
https://en.wikipedia.org/wiki/Van_Emde_Boas_tree