Mbin

Matthew Temple

Overview

Mbin is a container data structure I invented in 2010. It's a twist on trees and hash tables.

Unlike containers that require the contained type to define a {less than, greater than, or equal to} compare function and/or a hash function, objects held by an mbin must define an {equal, not equal} compare function and a modulo function.

Reference Implementation

http://github.com/clownfysh/cf/blob/master/x/case/mbin.h
http://github.com/clownfysh/cf/blob/master/x/case/mbin.c

Data Structure

Tree

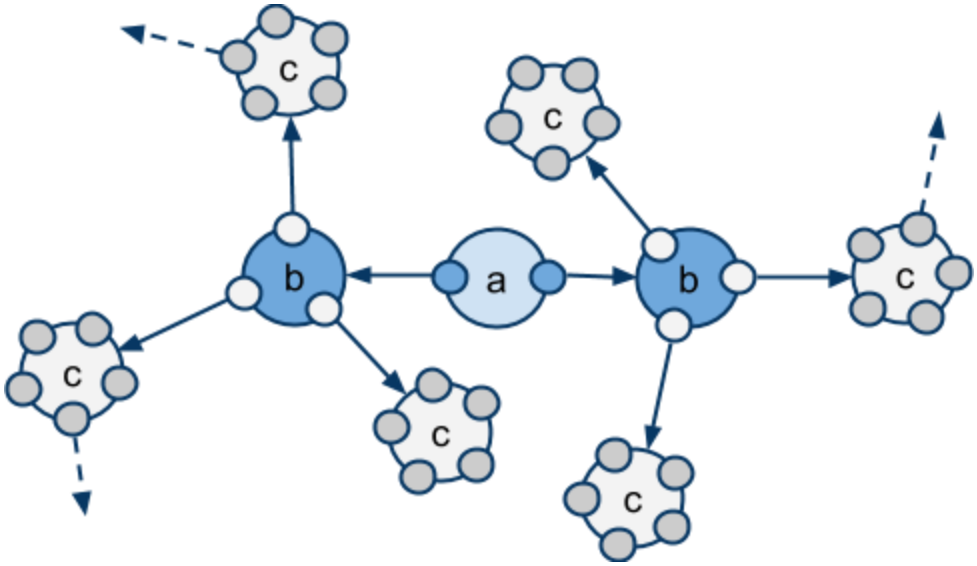
Mbin is a structured tree of bins.

Sometimes a bin doesn't have branches. When it does, the number of branches it has is a prime number. Each level of the tree is associated with a prime. Every bin that has branches, that resides on a particular level of the tree, has the same number of branches as every other branched bin on that level. That number of branches is the prime number associated with that level of the tree. The prime number associated with a tree level is unique to that level..it is not the same as the prime number used for any other level of the tree.

For simplicity, here, each level of the tree Q will be associated with the Qth prime number. The root bin, a—the bin on the first level—will have 2 branches. Each of its 2 child bins, when they have branches, will have 3 branches. Their child bins will each have either no branches, or 5 branches.

In actuality, the mbin starts with a higher prime and counts backwards through the primes toward 2. The starting number for the prime sequence is chosen based on the integer size returned by the **modulo function** (see details below) in a particular Mbin implementation. See my mbin implementation for a real-world example of the prime sequence—the simplified sequence used here is sufficient to explain the most essential properties of the mbin.

In this simplified description, the number of branches at each level is the sequence of prime numbers:



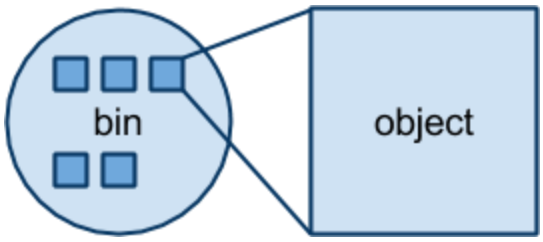
When an Mbin is created, it has only one bin, the root bin.

Bins

At a particular moment, each bin is either a simple bin or a container bin.

Simple

A simple bin contains only stored objects. It has no branches to other bins. There is a maximum number $max\_objects\_per\_bin$ of objects that a simple bin can contain. A simple bin can contain $[0, max\_objects\_per\_bin]$ objects. $max\_objects\_per\_bin = 8$ is recommended.

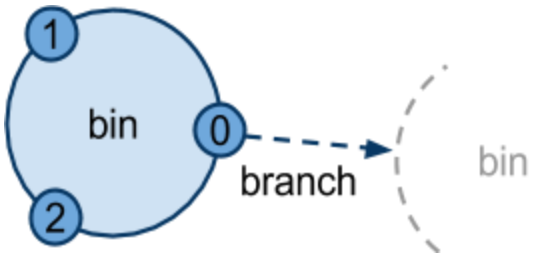


When an Mbin is created, its one bin, the root bin, is a simple bin containing zero objects.

Container

A container bin contains only branches to other bins. It contains no

objects. A container bin always has exactly the number of branches as the quantity of the prime number at that level. For instance, every container bin at the first level might have 2 branches; every container bin at the second level might have 3 branches; every container bin at the third level might have 5 branches; every container bin at the fourth level might have 7 branches; etc.



Branches are zero-indexed: this second-level bin's 3 branches are indexed $(0, 1, 2)$.

Contained object type

Objects stored in an mbin must define—must be operable on via—an {equal, not equal} compare function and a modulo function.

Compare function

The compare function determines whether or not two objects are equal.

In some contexts, this can be a function that compares only as much of two objects is necessary to determine that—in those cases—they are not equal.

Modulo function

The modulo function considers some aspect of an object to be a dividend. Given a divisor n that mbin supplies (n is the number of branches at a particular tree level), this function determines a modulo which is used to determine the path along branches of the mbin tree in which to store the object.

If your objects have integer indexes, you might simply define

this function as $index \bmod n$.

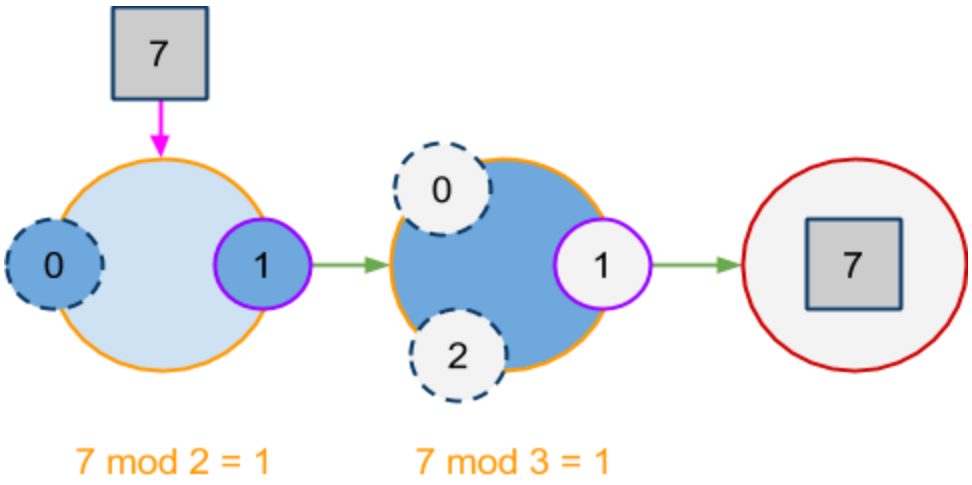
If your objects have UUIDs, you might consider part of the UUID an integer i and define this function as $i \bmod n$.

The modulo function must return numbers that are roughly evenly distributed across the range $[0, n-1]$ given by the divisor.

Add function

Here's how an object o, is added to an mbin:

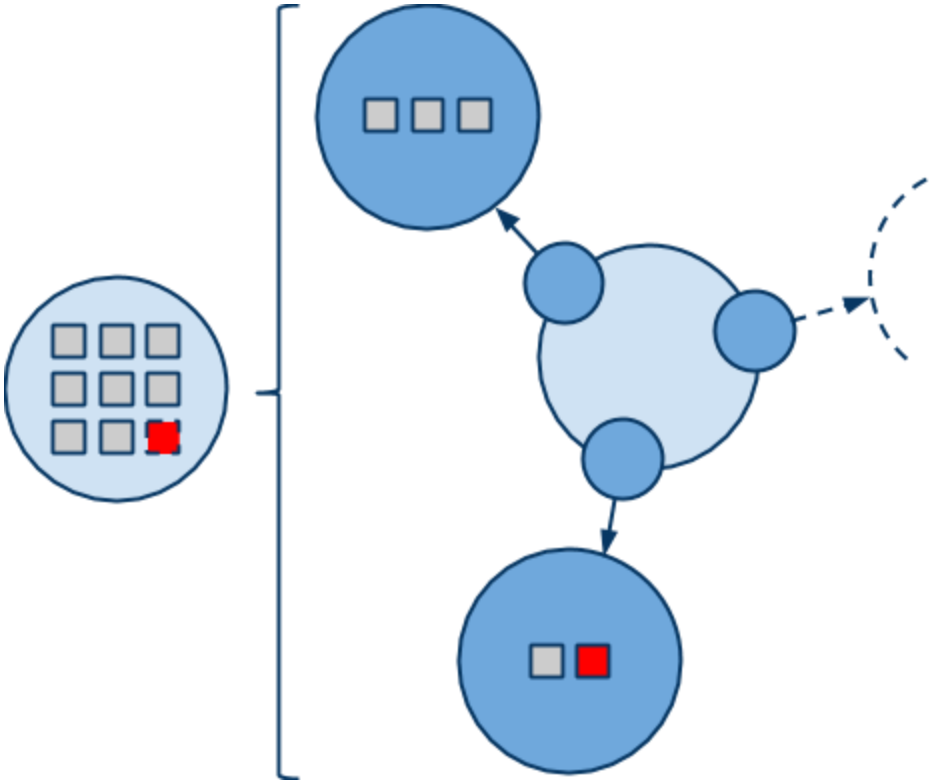
1. Start at the first level of the mbin tree, with the root bin.
2. Do one of three things:
    a. If this bin is a simple bin containing fewer than $max\_objects\_per\_bin$ objects, add o to the bin. Stop.
    b. If this bin is a simple bin containing $max\_objects\_per\_bin$ objects, **become a container** (see next section for details). Go to step 3.
    c. If this bin is a container bin, go to step 3.
3. For this Qth level of the mbin tree, determine the Qth prime number n. (For the first level, $n = 2$, for the second level, $n = 3$, for the third level, $n = 5$, etc.)
4. Find $o \bmod n$. Call that m.
5. Use m as a bin index. This bin has a branched bin b at index m. Now considering bin b, go to step 2.



Become a container

When you add an object to a simple bin that contains *max_objects_per_bin* objects, the bin first becomes a container bin.

It creates the number of branches suitable for that level, based on the quantity of the prime number at that level of the mbin tree. For each of those branches it creates a simple bin and points each of those branches to its new simple bin. Then it adds each object that was contained in the simple bin to the appropriate branched bin—the appropriate bin that is connected to this bin by a branch. Then it adds the object that pushed this once-simple bin over the limit of *max_objects_per_bin* objects, causing it to become a container bin.



Multiset

Mbin can be a set or a multiset. To use it as a set, for step 2a above, use the equal function to check whether the bin already contains this object, and if it does, don't add the object to the bin.

Find function

To find an object o in an mbin:

1. Start at the first level of the mbin tree, with the root bin.
2. Do one of two things:
    a. If this bin is a simple bin, then, using the equal function, start comparing o to each object in the bin. If you find an object equal to o, then the object is in the bin (and you've found it!), stop looking. If the equal function isn't true for any objects in the bin, then the objects isn't in the mbin.
    b. If this bin is a container bin, go to step 3.
3. For this Qth level of the mbin tree, determine the Qth prime number n. (For the first level, $n = 2$, for the second level, $n = 3$, for the third level, $n = 5$, etc.)
4. Find $o \bmod n$. Call that m.
5. Use m as a bin index. This bin has a branched bin b at index m. Now considering bin b, go to step 2.

Remove function

To remove an object o from an mbin:

1. Start at the first level of the mbin tree, with the root bin.
2. Do one of two things:
    a. If this bin is a simple bin: Using the equal function, start comparing o to each object in the bin.
        i. If you find an object equal to o, then stop looking and remove that object from the bin. If the number of objects in the bin is less than $\frac{max\_objects\_per\_bin}{become\_simple\_divisor}$, **become simple** (see next section for details).
        ii. If the equal function isn't true for any objects in the bin, then the objects isn't in the mbin and you can't remove it.
    b. If this bin is a container bin, go to step 3.
3. For this Qth level of the mbin tree, determine the Qth prime number n. (For the first level, $n = 2$, for the second level,

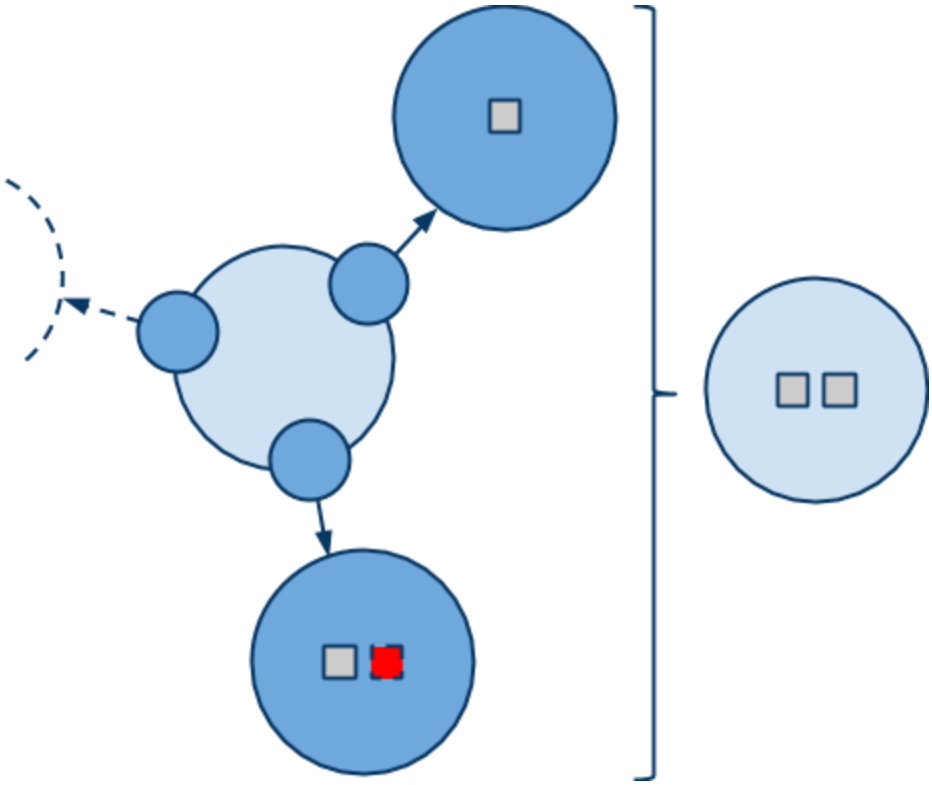
$n = 3$, for the third level, $n = 5$, etc.)

4. Find $o \bmod n$. Call that m.
5. Use m as a bin index. This bin has a branched bin b at index m. Now considering bin b, go to step 2.

Become simple

When you remove an object from a container bin and that causes the number of objects recursively contained within that bin—within all the bins contained by that bin and all the bins contained by them, etc—when that number of recursively contained objects falls below $\frac{max\_objects\_per\_bin}{become\_simple\_divisor}$ (with $become\_simple\_divisor = 2$ recommended)..in that case, the container bin becomes a simple bin.

It iterates through all the objects recursively contained within its bins. It puts those objects in this bin. It destroys this bin's branches and all the container bins they point to, recursively.



Iterate function

To iterate through an mbin:

1. Start with the root bin.
2. Do one of two things:
    a. If this bin is a simple bin, then iterate through each object in this bin.
    b. If this bin is a container bin, then this bin has a number of contained bins. For each of them, do step 2.

Poetry reference

This algorithm reminds me of a poem I wrote that contains the line "prime numbers / hidden / deeper and deeper / in handsome cracks" (perhaps that poem was the impetus for this algorithm). Of course prime numbers *are* hidden deeper and deeper in a certain type of combinatorial crack. This algorithm takes advantage of that.