Application Note: Associative Array (a.k.a. JudySL)

6/22/2001

Problem

How did the Judy authors use JudyL to create an associative array (JudySL)?

Solution

Break a null-terminated string into a sequence of (32/64-bit) words and build a tree of JudyL arrays with those words as indexes to represent the unique prefix of every string. Each tree leaf node is a pointer to the unique string suffix (a string index can also end without a leaf node). This is how JudySL was implemented.

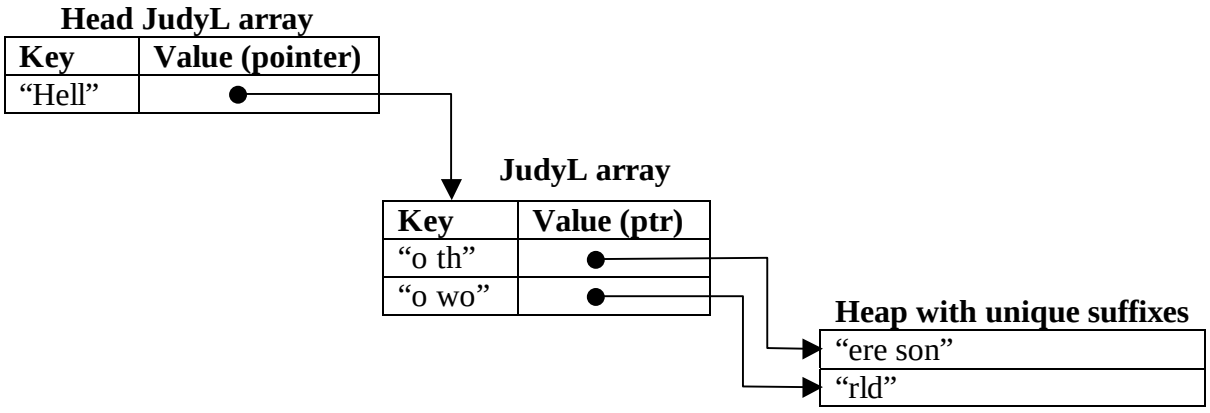
Example

Let's store two strings into a JudySL array. The strings are "Hello world" and "Hello there son".

First, think of these strings as if they were composed of a sequence of machine words rather than bytes. For this example, (assuming 32 bit words):

Word 1	Word 2	Word 3	Word 4
"Hell"	"o wo"	"rld"	
"Hell"	"o th"	"ere "	"son"

After inserting these two strings into a JudySL array, the array (actually a tree) looks like:



Critical to this method is the ability to tell the difference between a pointer to a JudyL array and a pointer to other objects like the unique suffix string. This algorithm makes use of the fact that a pointer to a Judy array contains extra information that encodes the type of Judy array being pointed to:

```
 if ((JudyArrayPointer & JLAP_MASK) == JLAP_INVALID)
      JudyArrayPointer does not point to a Judy array
else
      JudyArrayPointer does point to a Judy array
```

`JLAP_INVALID` and `JLAP_MASK` are defined in Judy.h. This little known feature only works for a pointer to a JudyL array (or a JudySL array which is a tree of JudyL arrays), not for a Judy1 array.

In addition to "normal" associative array operations (random insertions and deletions), JudySL can be used to sort by inserting strings into a JudySL array and then reading them back out with `JSLF()` and `JSLN()` (the `JudySLFirst()` and `JudySLNext()` macros). This is typically faster than using `qsort(3C)` or `sort(1)`.

