# Application Note: Simulating Content Addressable Memory

Alan Silverstein, ajs@fc.hp.com,
September 18, 2001

## Introduction

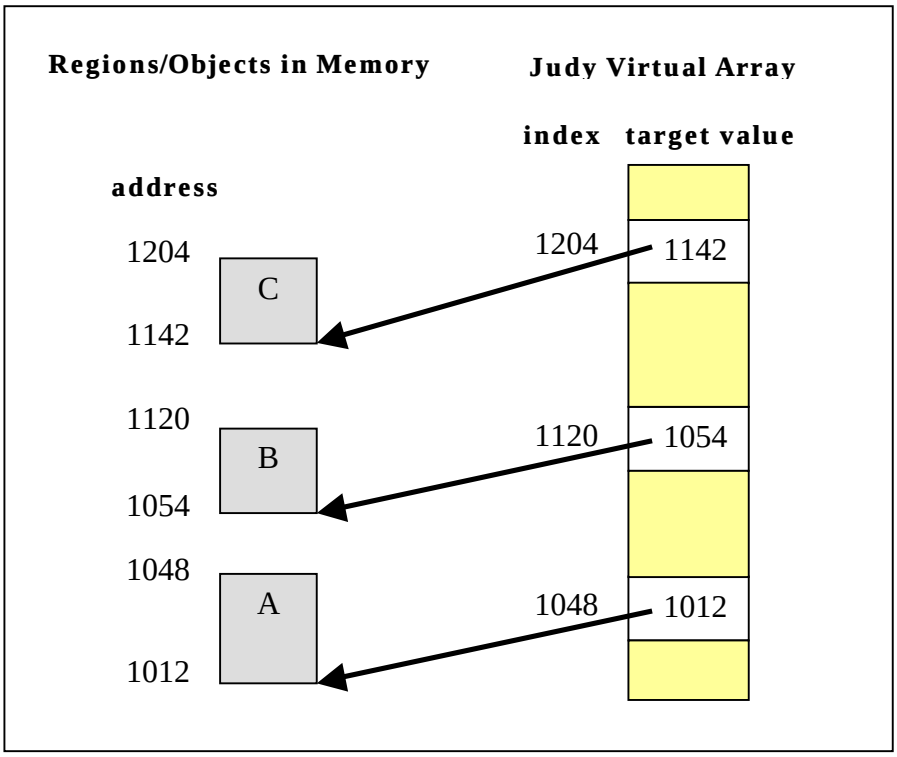
Judy, specifically JudyL, can be used to rapidly determine which of a set of variable-size objects residing in memory contains a specific address, or a byte at a specific address -- a kind of content-addressable memory (CAM). More generally, given a one-dimensional expanse of integers, such as memory addresses, which is divided into regions that either touch (are contiguous) or have gaps but do not overlap, such as machine code for compiled higher-level-language function bodies, Judy allows rapid mapping of a random integer (index, address) to the region that contains it, if any.

The inventor of Judy used an earlier version of it in an HP-internal process performance measurement tool called Prospect. This application had a general need for something like Judy, and a specific need for the CAM-like lookups illustrated below, which drove the need for neighbor lookups. ("Necessity is the mother of invention.")

## Method

Here's how to use JudyL to simulate content-addressable memory. Initialize the JudyL array to map from each region's or object's end address + 1 (the index) to its own starting address (the target value). Then, given any memory address, find the region or object, if any, containing that address by using `JLN()` (the `JudyLNext()` macro) to locate the next (successor) valid end address. This address is mapped (in JudyL) to the starting address of the containing object, as shown below.




In this example, a lookup starting at the address (index) 1052 results in finding index 1120 as the next valid end address, which points to (target value) 1054 as the beginning address of object "B". This indicates 1052 is not an address in any known object.

A lookup starting at address (index) 1148 results in finding 1204 as the next valid end address, which points to 1142, which indicates object "C" includes the lookup address (index) of 1148.

## Notes

- Prospect used Judy in this way to identify the region (process address space) containing any virtual memory address sampled during profiling.

- In the preceding example, each target value can be stored as an offset rather than an absolute number or address. The example in this application note happens to be the way Judy was used in Prospect.

- The preceding example could also be "mirror reversed": Store as each index the starting address of a region or object, and set the index's corresponding value to the size of the object (effectively an offset to its end) or to the address at or just beyond its end. Search backward using `JLL()` or `JLP()` (`JudyLLast()` or `JudyLPrev()`) as appropriate, rather than forward with `JLF()` or `JLN()` (`JudyLFirst()` or `JudyLNext()`) to find the enclosing object (see the C language example below).

- This search method allows insertion of new objects between existing objects without modifying any mappings (valid indexes) already in the Judy array.

- The CAM technique can also be used with overlapping regions or objects, but in this case the application must do multiple forward or backward searches until all possible "containers" are checked. If no region is allowed to fully enclose another, this search can terminate upon locating a region wholly outside the index of interest; otherwise, all regions must be checked until reaching the last in the Judy array.

## Example Code

In the following example, Judy library calls are highlighted in red.

```
// Example of how to use Judy to do a form of content-addressable-memory.
//
// This program reads stdin for lines containing either one or two fields.
//
// * If two fields, the first is the start of a region (an index value)
//   and the second is the size of the region.  (A variation on the
//   illustration earlier in this application note.)
//
// * If one field, it is taken as an index value to be looked up; the
//   start of the surrounding region is printed, or else a string
//   indicating the index is not in any known region.
//
// Note:  This simple example assumes the stored regions do not overlap,
// although they can be contiguous.
//
// Note: To keep this example simple it does little error checking.

#include <stdio.h>

#define JUDYERROR_SAMPLE 1              // use default error handling.
#include <Judy.h>

main()
{
        char     line[BUFSIZ];  // read from input.
        Word_t   index;         // from input, to store or find.
        Word_t   size;          // of a region to store.
        Pvoid_t  Parray = (Pvoid_t) NULL;       // JudyL array.
        PPvoid_t PPvalue;       // one value area in Parray.
        int      items;         // from fscanf().

        while (fgets(line, BUFSIZ, stdin) != (char *) NULL)
        {
            items = sscanf(line, "%lu %lu", &index, &size);

// SAVE A NEW REGION:

            if (items >= 2)
            {
                JLI(PPvalue, Parray, index);
                *((Word_t *) PPvalue) = size;
                (void) printf("stored region starting at index %lu,
                                size %lu\n",
                                index, *((Word_t *) PPvalue));
            }
```

```
// SEARCH FOR A REGION CONTAINING THE INDEX:

            else
            {
                Word_t indexstart = index;

                JLL(PPvalue, Parray, indexstart);

                if ((PPvalue == (PPvoid_t) NULL)    // no previous start.
                 || (indexstart + *((Word_t *) PPvalue) <= index))
                {                                   // after previous.
                    (void) printf("%lu not in any known region\n", index);
                }
                else
                {
                    (void) printf("%lu is in region %lu, size %lu\n",
                                  index, indexstart,
                                  *((Word_t *) PPvalue));
                }
            }
        } // while read next input line

} // main
```