Sparse Arrays

The `sparse_array.c` file contains an implementation of a sparse array that attempts to be both space and time efficient.

The sparse array is represented using a tree structure. Each node in the tree contains a block of pointers to either the user supplied leaf values or to another node.

There are a number of parameters used to define the block size:

```
OPENSSL_SA_BLOCK_BITS   Specifies the number of bits covered by each block
SA_BLOCK_MAX            Specifies the number of pointers in each block
SA_BLOCK_MASK           Specifies a bit mask to perform modulo block size
SA_BLOCK_MAX_LEVELS     Indicates the maximum possible height of the tree
```

These constants are inter-related:

```
SA_BLOCK_MAX        = 2 ^ OPENSSL_SA_BLOCK_BITS
SA_BLOCK_MASK       = SA_BLOCK_MAX - 1
SA_BLOCK_MAX_LEVELS = number of bits in size_t divided by
                      OPENSSL_SA_BLOCK_BITS rounded up to the next multiple
                      of OPENSSL_SA_BLOCK_BITS
```

`OPENSSL_SA_BLOCK_BITS` can be defined at compile time and this overrides the built in setting.

As a space and performance optimisation, the height of the tree is usually less than the maximum possible height. Only sufficient height is allocated to accommodate the largest index added to the data structure.

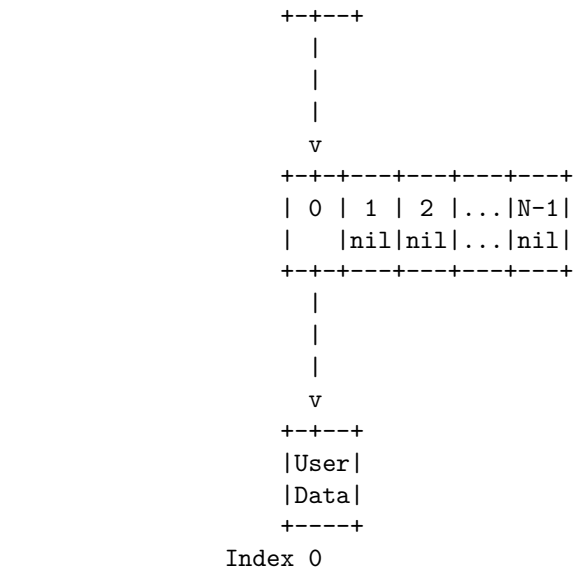
The largest index used to add a value to the array determines the tree height:

Largest Added Index	Height of Tree
SA_BLOCK_MAX - 1	1
SA_BLOCK_MAX ^ 2 - 1	2
SA_BLOCK_MAX ^ 3 - 1	3
...	...
size_t max	SA_BLOCK_MAX_LEVELS

The tree height is dynamically increased as needed based on additions.

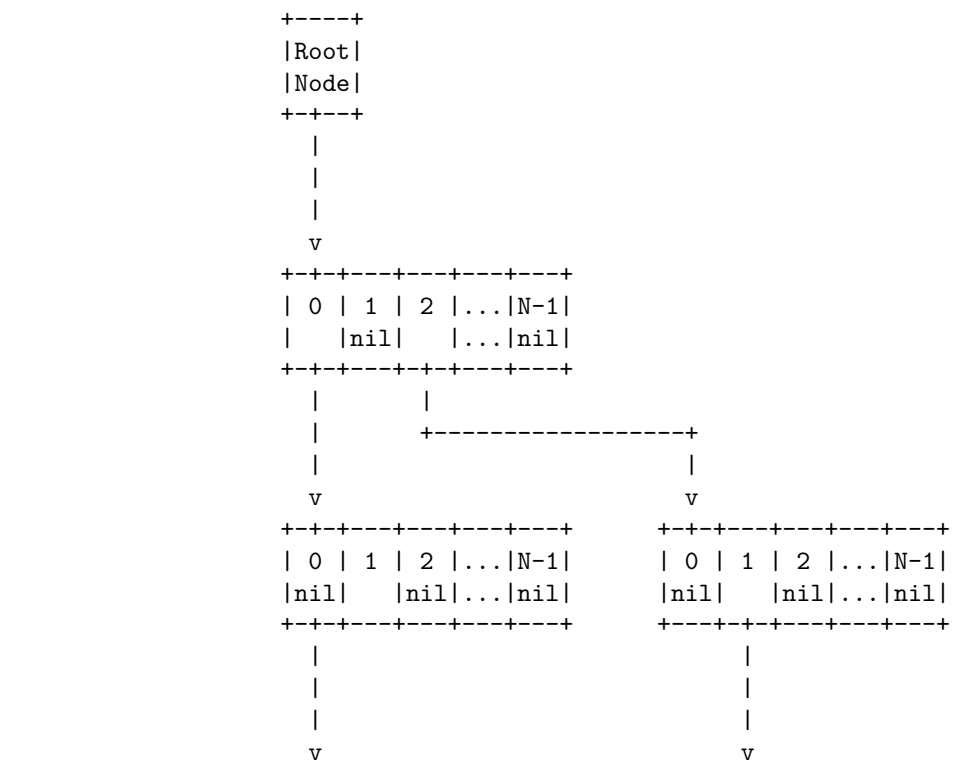
An empty tree is represented by a NULL root pointer. Inserting a value at index 0 results in the allocation of a top level node full of null pointers except for the single pointer to the user's data (N = SA_BLOCK_MAX for brevity):

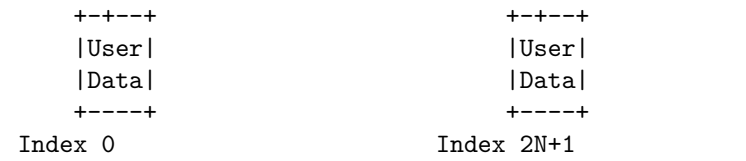
```
+----+
|Root|
|Node|
```



Index 0

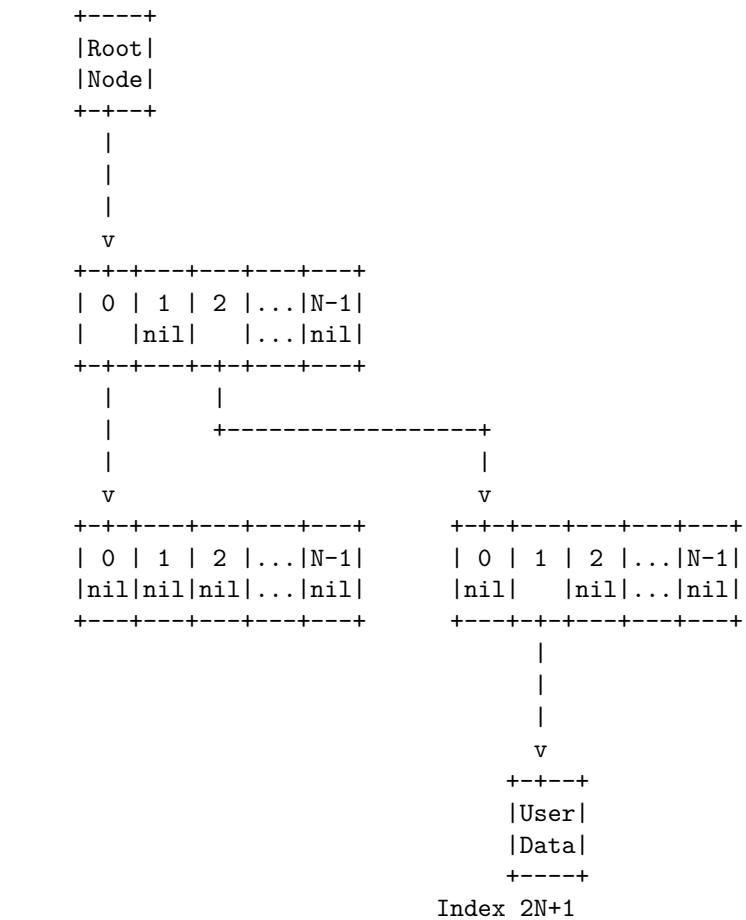
Inserting at element 2N+1 creates a new root node and pushes down the old root node. It then creates a second second level node to hold the pointer to the user's new data:





The nodes themselves are allocated in a sparse manner. Only nodes which exist along a path from the root of the tree to an added leaf will be allocated. The complexity is hidden and nodes are allocated on an as needed basis. Because the data is expected to be sparse this doesn't result in a large waste of space.

Values can be removed from the sparse array by setting their index position to NULL. The data structure does not attempt to reclaim nodes or reduce the height of the tree on removal. For example, now setting index 0 to NULL would result in:



Accesses to elements in the sparse array take O(log n) time where n is the largest element. The base of the logarithm is `SA_BLOCK_MAX`, so for moderately small

indices (e.g. NIDs), single level (constant time) access is achievable. Space usage is O(minimum(m, n log(n)) where m is the number of elements in the array.

Note: sparse arrays only include pointers to types. Thus, `SPARSE_ARRAY_OF(char)` can be used to store a string.