

```

DEG = 4      # Children per node (must be power of 2)
BITS = 2      # log2(DEG), bits needed to select child
MASK = 0b11  # Bitmask for extracting low BITS bits (DEG - 1)

def __init__(self, key, value, children=None):
    self._key = key
    self._value = value
    if children == None:
        self._children = [None] * hamt.DEG
    else:
        self._children = children

def _set(self, key, value, hashbits):
    # Each node encountered during search will get altered.
    # To maintain persistence, each is duplicated, updated, returned.
    if (self._key == key):
        # This node matches key. Return duplicate with new value
        return hamt(self._key, value, self._children)
    else:
        # Continue search using hashbits to pick direction
        child_num = hashbits & hamt.MASK
        # Copy can reuse key/value. Child list gets updated, so duplicate
        copy = hamt(self._key, self._value, list(self._children))
        if (copy._children[child_num] == None):
            # End of search, key not found, add new node
            copy._children[child_num] = hamt(key, value) ← Point O.
        else:
            # Continue by asking appropriate child to set key/value
            copy._children[child_num] = copy._children[child_num]. \
                _set(key, value, hashbits >> hamt.BITS)
    return copy

def set(self, key, value):
    # Pass key/value and hashbits to recursive helper
    return self._set(key, value, hash(key))

```

```
    list(self._children))
):
    add new node
    key, value) ← Point
    child to set key/value
    _children[child_num]. \
        key, hashbits >> hamt.BITS)
```

↑ point of view
of child

1. obj1 = ham("A", 1)

A, 1

2. `obj2 = obj1.set("B", 2)`

bin(hash("B")) = 0b0101

Mask = 0b11

$$\begin{array}{r} 0101 \\ \times 0101 \\ \hline 01 \end{array}$$

$$\text{Child norm} = \frac{\text{obj}}{\sigma}$$

3. $\text{obj3} = \underline{\text{obj2.set("C",3)}}$

A, I

✓ From point of view foot B, Z

original \rightarrow bin (hash("C")) = 0b0101
hash bits
Mask = 0b11

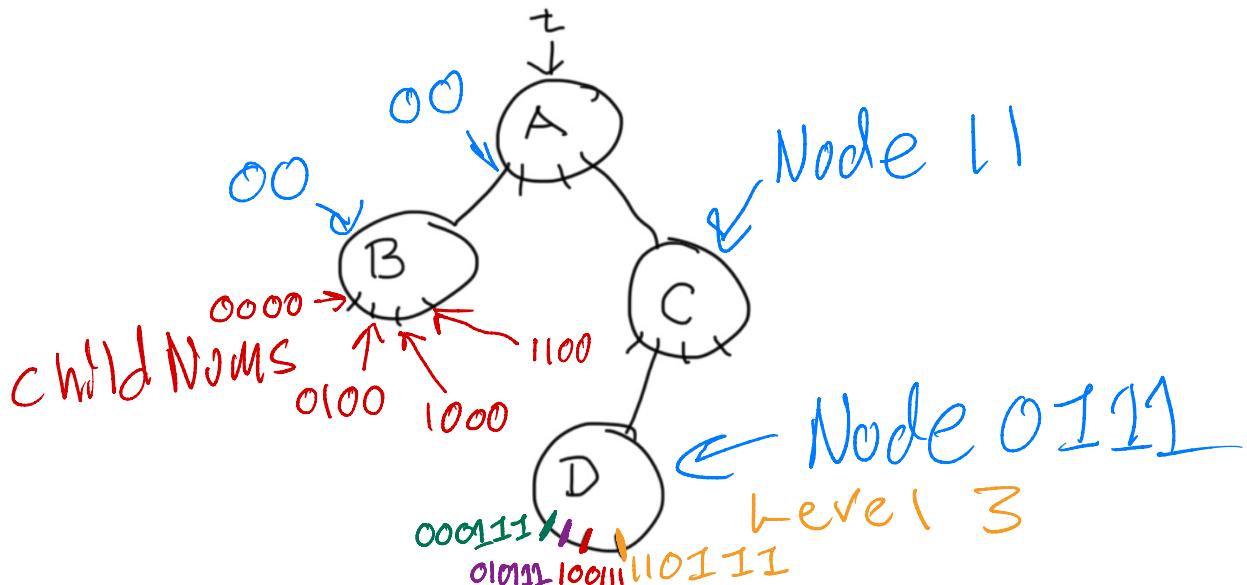
bin(hash("C")) = 0b0101

$\text{MASK} = \text{0b11}$ Bit shift
 $\text{wid num} = 01$ ↙ 2

```

graph TD
    A((A, 1)) --> B00((00))
    A((A, 1)) --> B01((01))
    A((A, 1)) --> C101((101))
    B00((00)) --> B0000((00))
    B00((00)) --> B0001((01))
    B01((01)) --> B0111((11))
    B01((01)) --> B0110((10))
    C101((101)) --> C10111((11))
    C101((101)) --> C10110((10))
    B0000((00)) --> B000011((11))
    B0000((00)) --> B000010((10))
    B0001((01)) --> B000111((11))
    B0001((01)) --> B000110((10))
    C10111((11)) --> C1011111((11))
    C10111((11)) --> C1011110((10))
    C10110((10)) --> C1011011((11))
    C10110((10)) --> C1011010((10))
    C1011110((10)) --> C101111011((11))
    C1011110((10)) --> C101111010((10))
    C1011010((10)) --> C101101011((11))
    C1011010((10)) --> C101101010((10))
  
```

5. (ungraded) The following figure is an HAMT with four keys (their associated values are not shown). What trailing bits can you deduce for the hashes of A, B, C, and D?



6. (ungraded) Draw the resulting HAMT when the values have been added with the keys A, B, C, D, E, F (in that order). Your drawing should look similar to the one in Problem 3 (ie, ignoring the values and beginning with root A). Use the following hashes.



7. (ungraded) Beginning with the trie in Problem 3, let's say `s = t.set(E, "e")` is invoked. Let `hash(E) = 00110011`. Draw the resulting tries s and t, including all nodes and links between them. (Don't include any values in your drawing.)

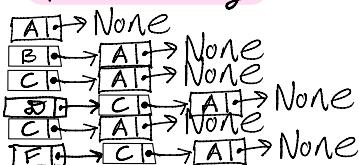
8. (ungraded) Draw all nodes in the resulting structure (including their data element) and indicate which nodes are pointed at by each variable.

```

a = list135("A")
b = a.cons("B")
c = a.cons("C")
d = c.cons("D")
e = d.rest()
f = e.cons("F")

```

The lame way



The cool kids way

Do this on
Exam

LAME Don't do on exam