SUBJECT: How the trie works DATE: 2022/03/1 PAGE#:

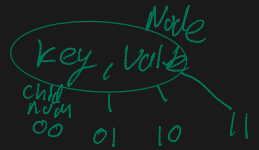
```python
class hamt:

    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)
            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))
```

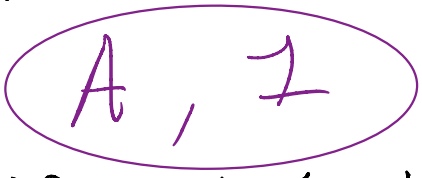


Point of view of current node

Point of view of child

key value

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

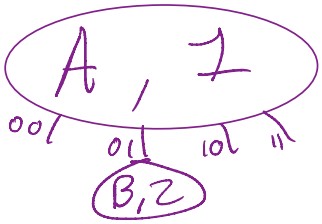


2. obj2 = obj1.set("B", 2)

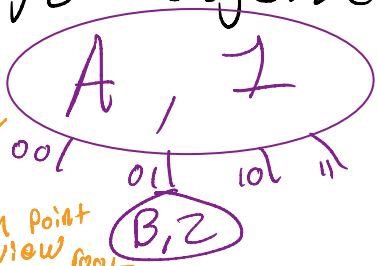
bin (hash("B")) = 0b0101

Mask = 0b11

0b0101
0b11
childnum = 01



3. obj3 = obj2.set("C",3)



From point of view root

original hash bits → bin (hash("C")) = 0b0101

Mask = 0b11

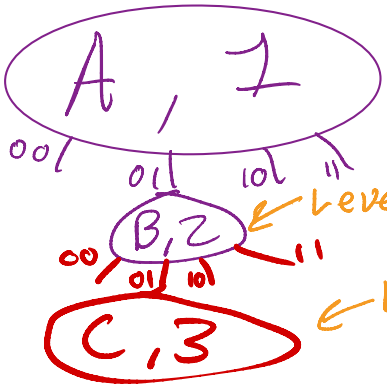
```python
else:
    # Continue by asking appropriate child to set key/value
    copy._children[child_num] = copy._children[child_num].     \
                        _set(key, value, hashbits >> hamt.BITS)
return copy
```

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

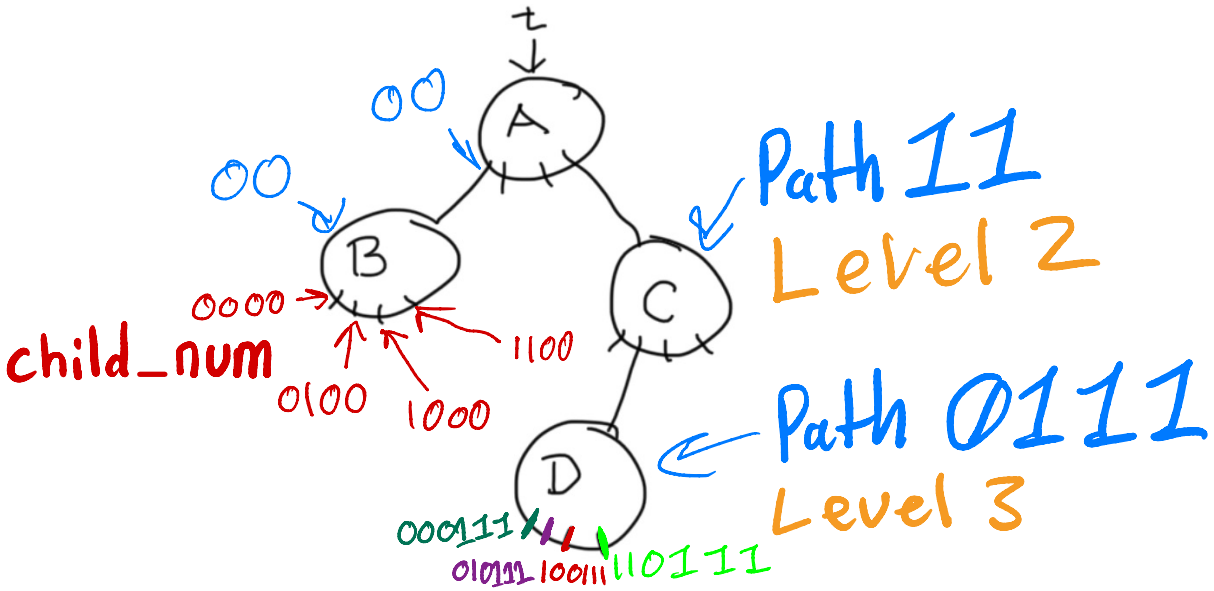
Mask = 0b11

childnum = 01

Bit shift 2



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

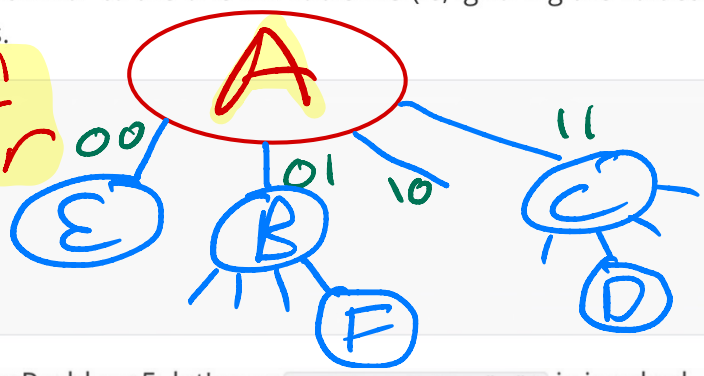


6. (ungraded) Draw the resulting HAMT when the values have been added with the keys A, B, C, D, E, F (in that order). Your drawing should look similar to the one in Problem 3 (ie, ignoring the values and beginning with root A). Use the following hashes.

```
h(A) = 10010000
h(B) = 10110001
h(C) = 11110011
h(D) = 10000111
h(E) = 10011100
h(F) = 01001101
```

Root A

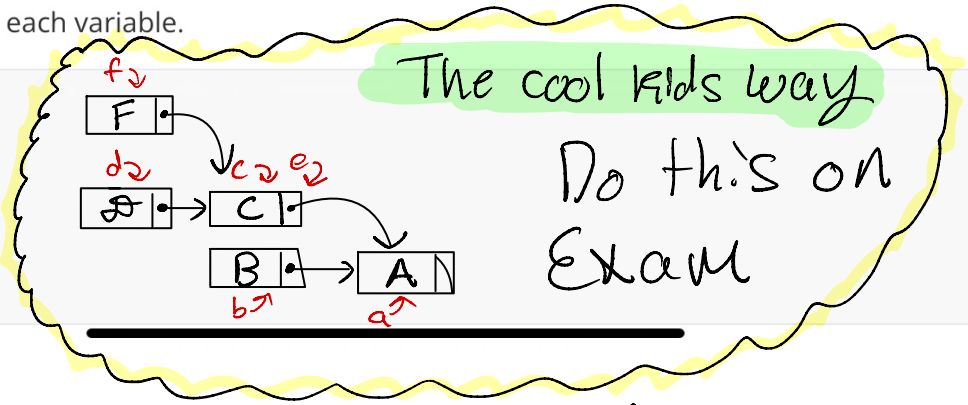
Root hash does Not Matter



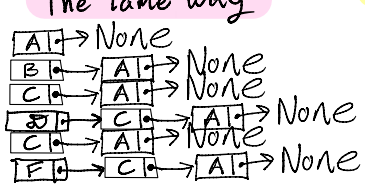
7. (ungraded) Begining with the trie in Problem 5, let's say `s = t.set(E, "e")` is invoked. Let `hash(E) = 00110011`. Draw the resulting tries s and t, including all nodes and links between them. (Don't include any values in your drawing.)

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

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



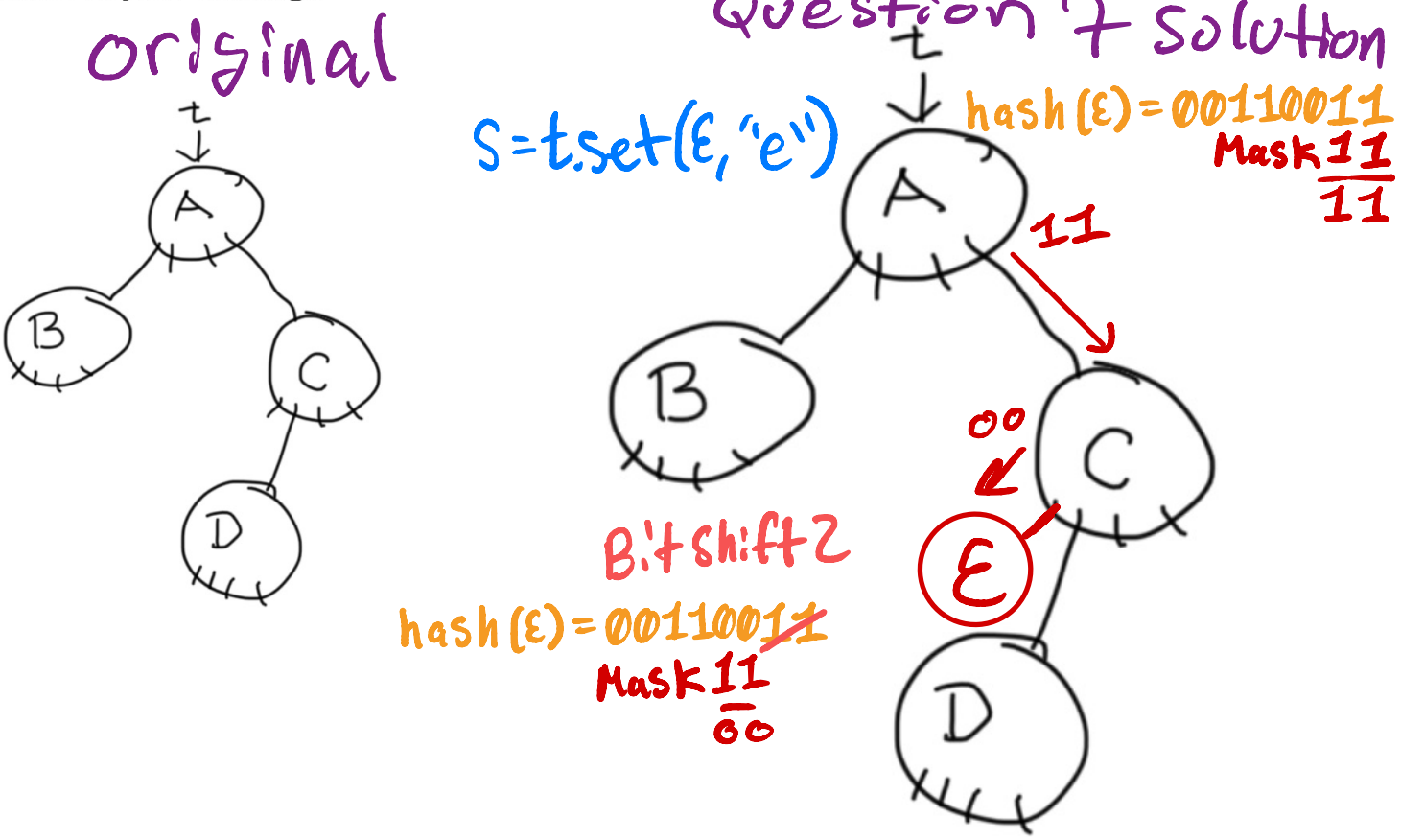
The lame way



← LAME Don't do on exam

SUBJECT: DATE: / / PAGE#:

7. (ungraded) Begining with the trie in Problem 5, 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.)



SUBJECT: DATE: / / PAGE#:

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