2018-03-27 Hashtables Part 2

Tuesday, March 27, 2018 8:55 AM

Recap from last lecture

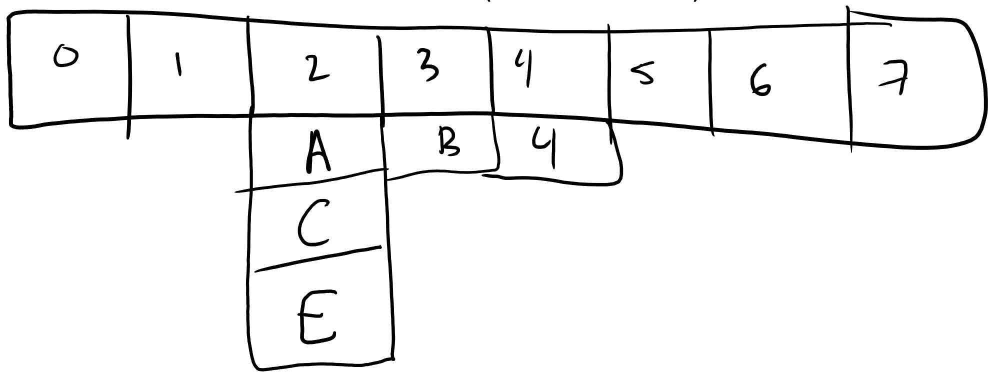
- Three factors that affect hashtable performance
  - Hashing function
  - Collision resolution mechanism (today's topic)
  - Load factor (% fullness, will talk about in lab)

Collison Resolution Mechanisms

- Collision - when two keys hash to the same integer box
- What do we do when this happens?
- Two "families" of approaches

Separate Chaining

- General idea: we have boxes of boxes (2D-like structure)



Insert Sequence: A -> 2, B -> 3, C-> 2, D->4, E->2

Basic "add" algorithm

- Hash key, find the data structure (LL or vector) at the particular box (e.g. 2)
- For each item in the box:
  - If the key matches, update the value
- If we never saw the key, do a push_back on that box

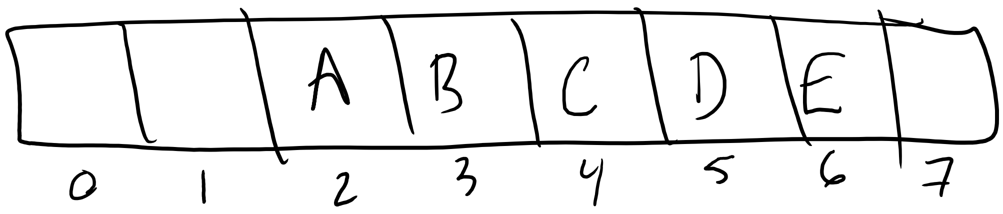
- Implications:
  - As more items are placed in each box, the time to add begins to look more like O(N) rather than O(1)
  - One advantage separate chaining has over collision resolution mechanisms is that a HT using separate chaining can be over 100% full

Open Addressing

- One item per box. If a box full, we must "probe" to find a new box
- "If my expected box is full, find a new box to go to."

Linear Probing

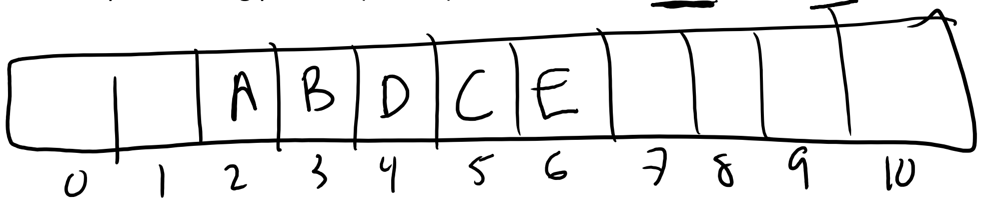
- If my box is full (e.g. box 2), try the next one over (e.g. box 3)
- E.g. on insert sequence: A -> 2, B -> 3, C-> 2, D->4, E->2



- As load factor increases, search time begins to look more like O(N)
- Linear probing often creates a clustering effect.
    - Problems in one area tend to create problems in another area

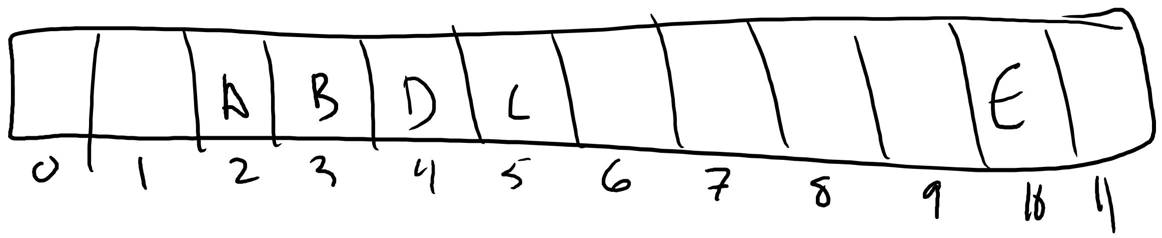
Quadratic Probing

- Rather than looking over by one, look over by some quadratic amount
    - e.g. next probe = (i^2 + 3i + 5)
- Tends to reduce clustering effect
- Most likely to get "stuck" in a cycle and be unable to find a new spot
    - Literature suggests that quadratic probe HTs be no more than 50% full
- Example HT using probe = (i^2 + 1) on A -> 2, B -> 3, C-> 2, D->4, E->2



Double Hashing

- If first hash creates a collisions, hash using a second algorithm plus a "salt"
- Ex: hash1(x) = x^2 * x + 3; hash2(y, salt) = 5y + 7 * 2salt + 1
- Example HT using hashes A-> 2,6; B-> 3,1; C->2,5; D->4,8; E->2,5,10



- Double hashing tends to avoid clustering
- As with quadratic probing, double hashing *can* create infinite loops
    - Implication: load factor should not get anywhere close to 100%
- In practice, double hashing tends to be the slowest of the 3 traditional open addressing schemes

Modern Open Addressing Technique #1: Cuckoo Probing

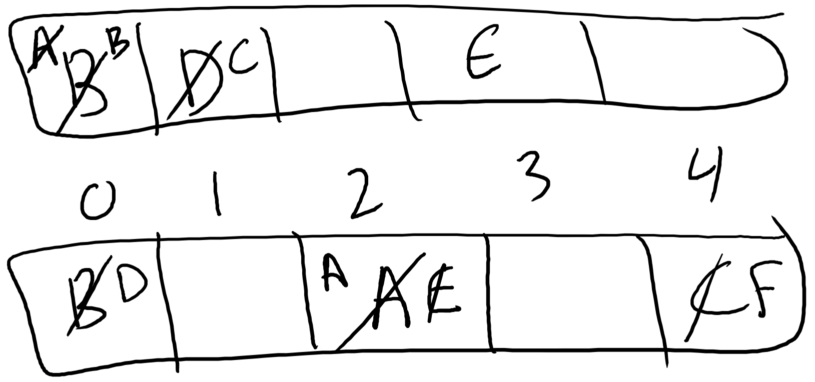
- Developed in 2001
- Came about as a practical implication of new probability theory
- Sort of like double hashing with two parallel arrays
- Each array has its own hashing function
- On insert, randomly pick the array to place the item in
- On collision, shove out the existing item, move it to its secondary location
    - Repeat process until a cycle is detected
- Example on the following
    -

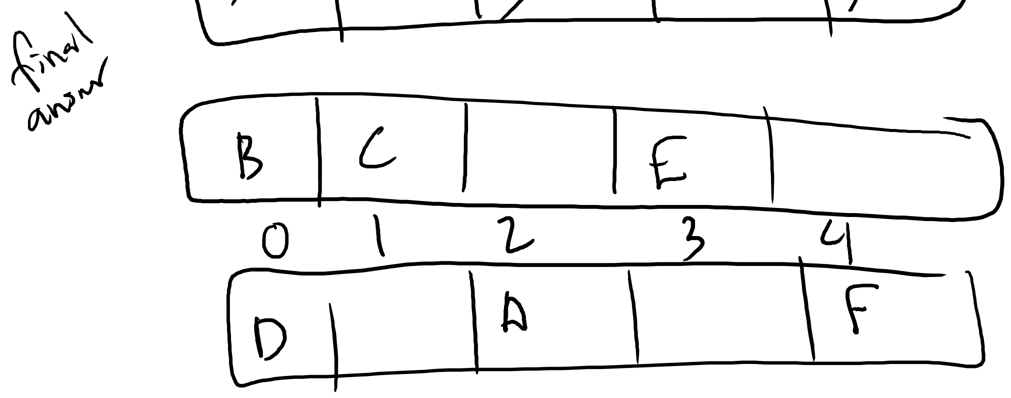
A: 0 2	B: 0 0	C: 1 4	D: 1 0	E: 3 2	F: 3 4

- Repeat process until a cycle is detected

- Example on the following

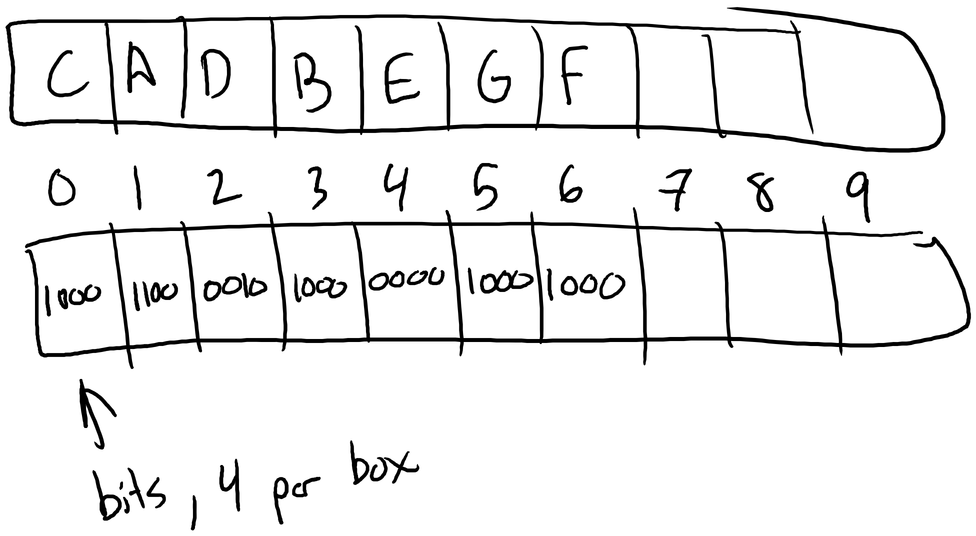
A: 0,2	B: 0,0	C: 1,4	D: 1,0	E: 3,2	F: 3,4





Hopscotch Hashing

- Published in 2009
- Based on linear probing, puts a hard limit on probe count
  - E.g. A given item cannot be more than 4 spaces away
  - Implication: guaranteed O(1) performance
  - Question: Is the overhead of Hopscotch Hashing worth the O(1) performance?
- Like cuckoo hashing, uses two arrays. 2nd array tracks distance from hash origin.

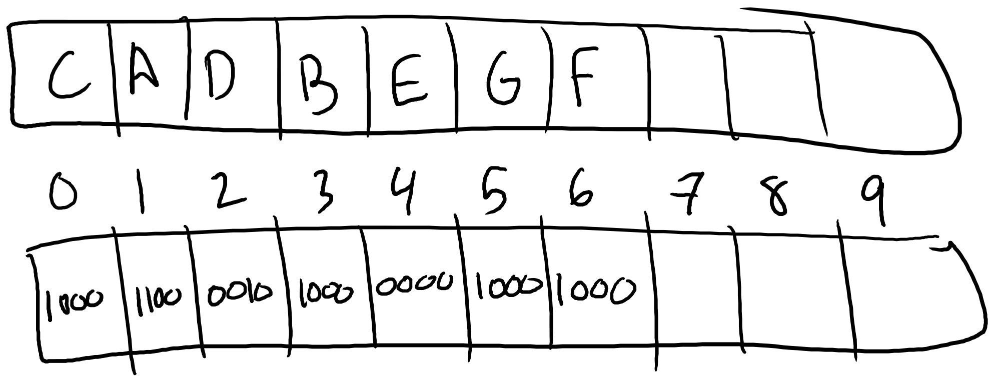


- Bits in 2nd array track the distance from original hashed value.

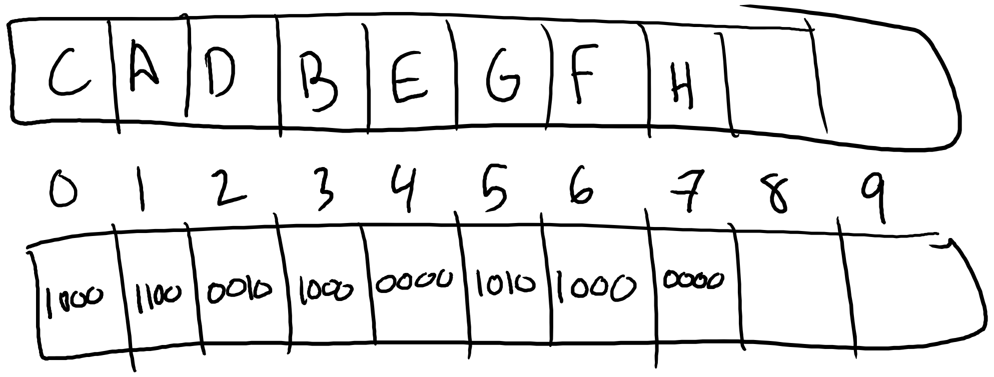
0/1	0/1	0/1	0/1
Something that is in the real box hashes to this value	One box over hashes	Two boxes over	Three boxes over

Algorithm (from wikipedia)

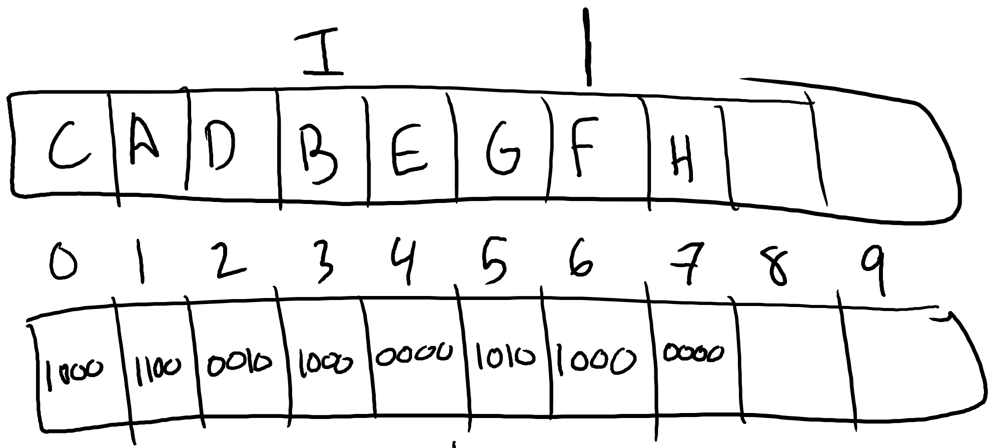
1. If original hashed box is empty, add to that box, update bits array. **Done.**
2. Starting at hashed value, try to find a box that is empty up to max distance (e.g. 4)
3. If none empty, starting at max distance away, try to move other boxes farther away from their hashed value. If this ultimately allows for us to place new value within max distance, we're good. Otherwise, stop and readjust HT.



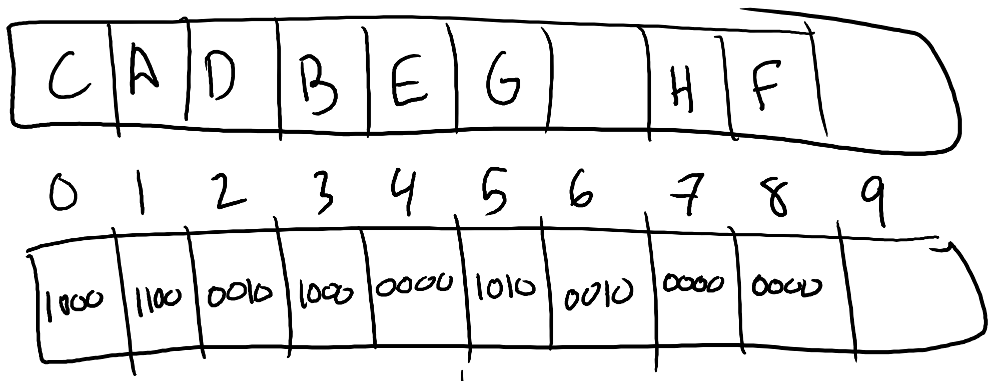
Ex: H → 5



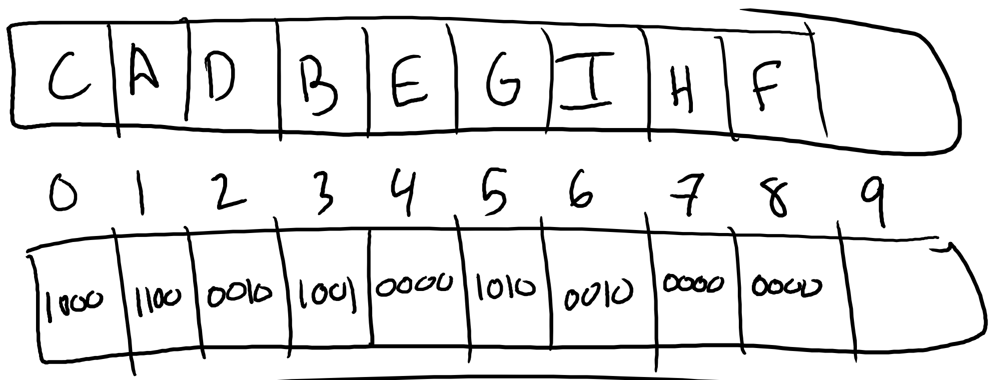
Ex: I → 3



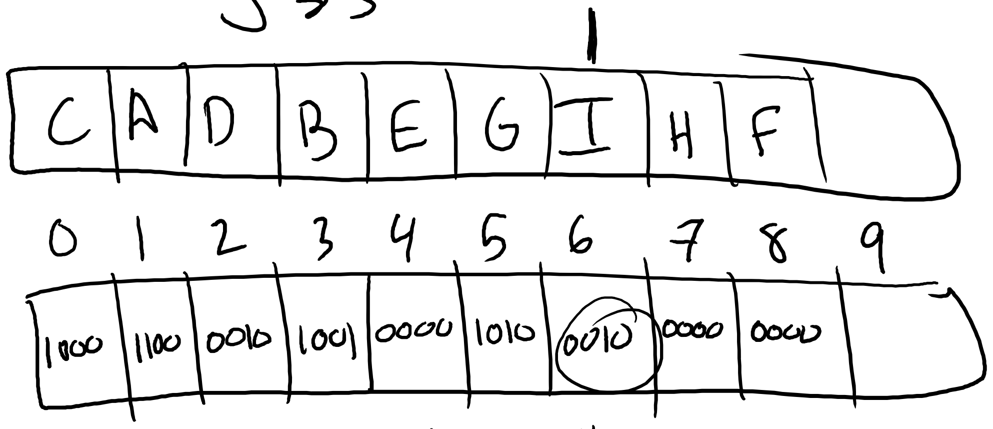
Step 1: Create room ↓



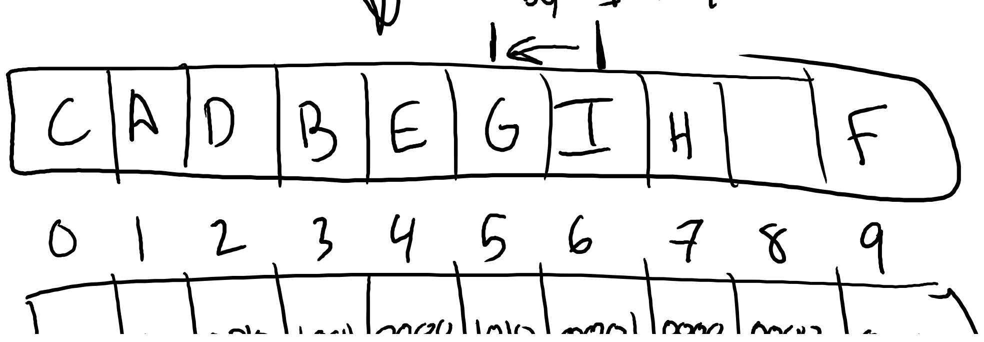
step #2
add I

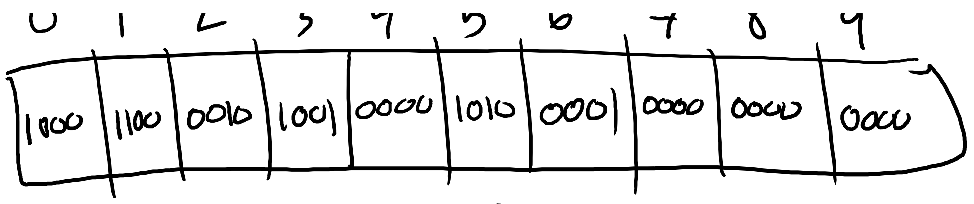


J → 3

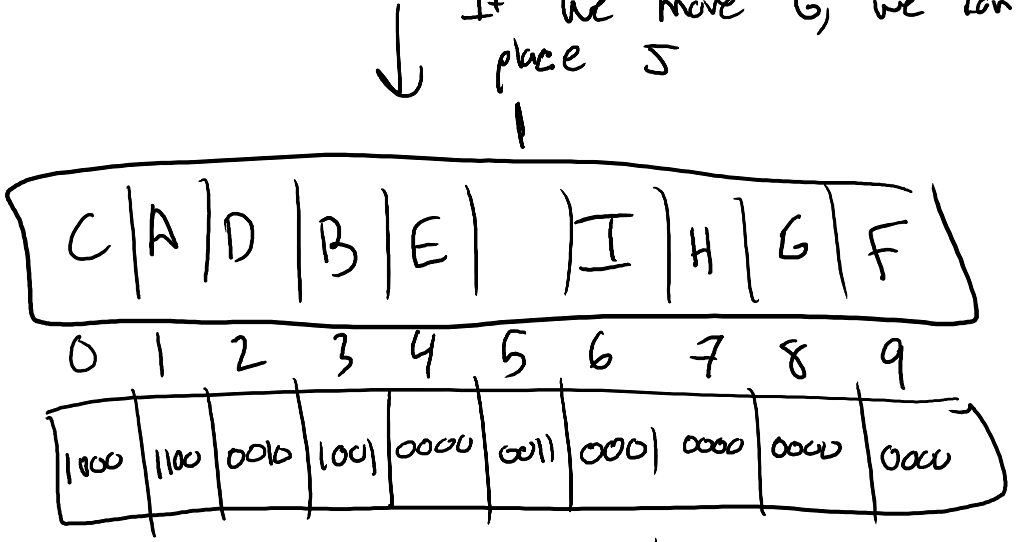


tells us
F can move
by 1 if possible

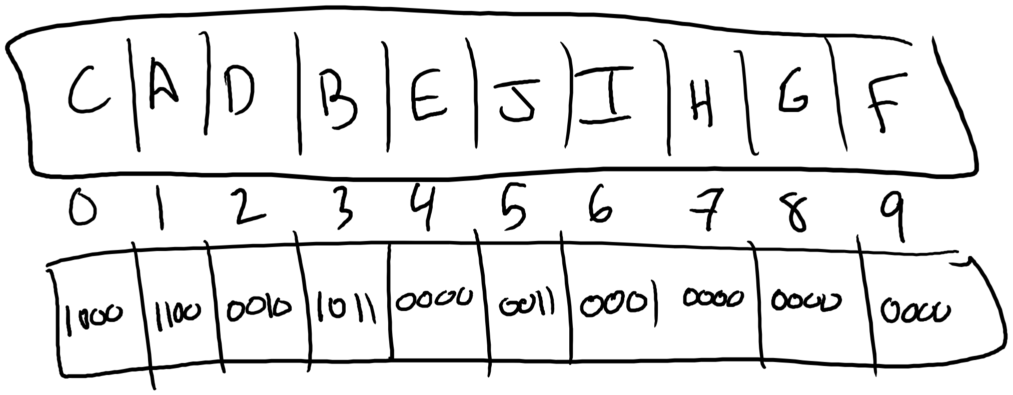




If we move G, we can place J

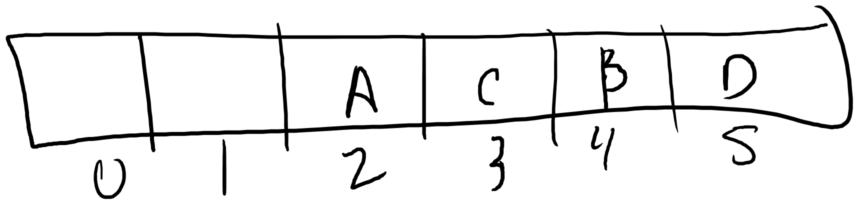


now, place J

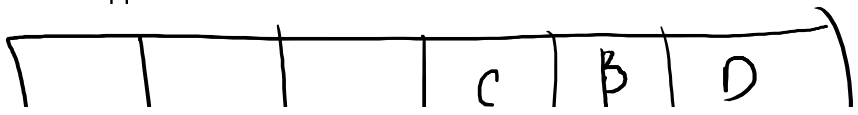


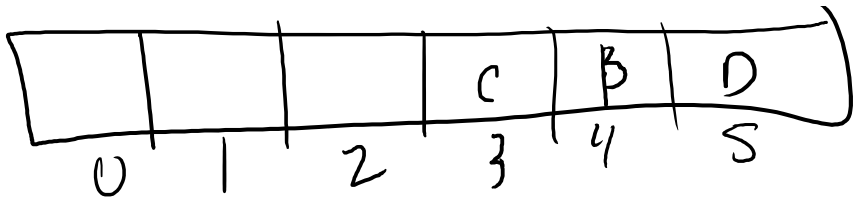
Deleting Items in a hash table

- Not so "obvious"
- Consider the following insert sequence for linear probing
    - A->2, B->4, C->2, D->4

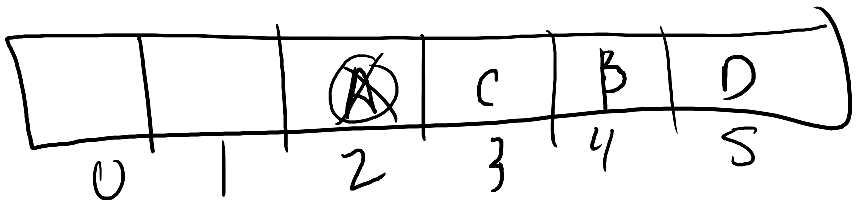


What happens when we delete "A" from the HT?





- What if I wanted to alter C's value?
- Removing A caused all items that are farther down the line that also hash to 2 to disappear from the HT
- As a workaround, we perform a "soft delete". Don't actually delete value, just mark it for deletion



- Actual deletes only occur on resizing.
- Resizing is also interesting.
  - Hash values are determined by array size. Altering array size means that it is highly likely that a value's hash will change when we resize.
  - Implication: For every resize, we must rehash all elements in our array