CS 225 Spring 2019 :: TA Lecture Notes
3/15 Hashing

By Wenjie

- ## Collision

A Perfect Hash Function is a **one-to-one** function, where for each **data / input**, the hash function produces a **unique output**.

There are 3 components of Hash Tables:

1. The hash function: f(h) -> Integers
   - Choose a good hash function (it could be tricky to define a good hash function - but try not use self created hash function)
2. The compression: Integer -> array
3. What happened in chaos/collision
   - One of big problem that hash table would have is collisions.

- **Separate chaining**

We could resolve collisions using separate chaining which is an instance of open hashing, which we use another linked list to save all collisions.

- Running time:
  - Insert: O(1)
  - Remove/find:
    - worst case: o(n)
    - SUHA: O(N/n) = load factor

(*Open hashing : data stored outside of the array

- ## Linear probing

Another way to handle collisions is to use closed hashing. When using closed hashing, we keep all our data inside of our hash table. We will cover two strategies for closed hashing:
Probing- based hashing

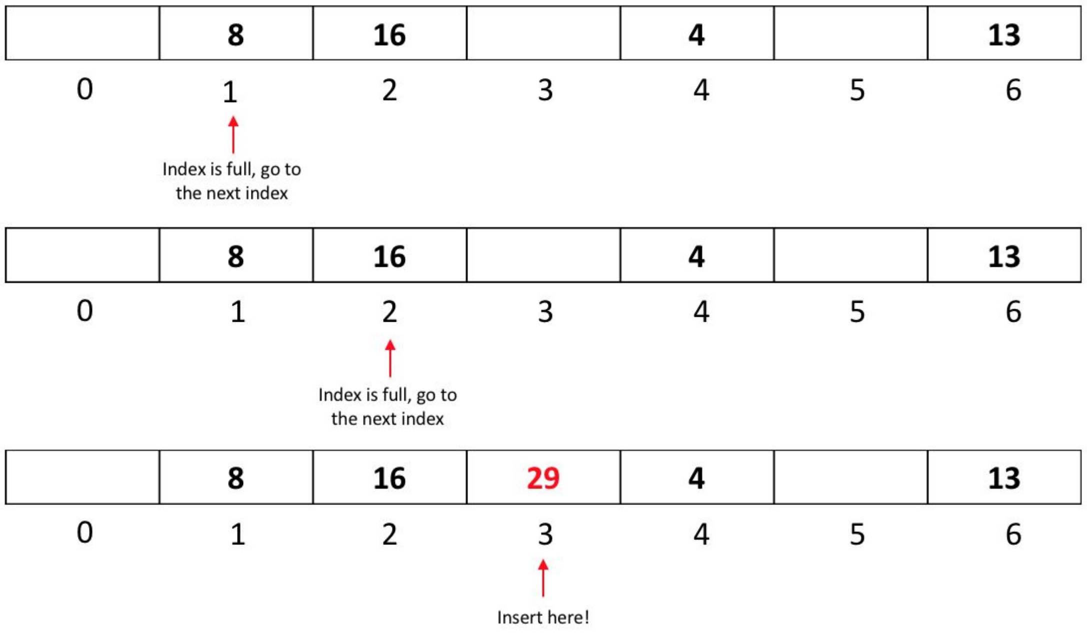
- Hash the key and try to insert. If the spot is filled, linearly loop down the array to find the next open spot to place the data. Assume we have the following hash table:

	8	**16**		**4**		**13**
0	1	2	3	4	5	6

Insert 29: 29 % 7 = 1

CS 225 Spring 2019 :: TA Lecture Notes
3/15 Hashing

By Wenjie


8
16
4
13
0
1
2
3
4
5
6
Index is full, go to the next index
8
16
4
13
0
1
2
3
4
5
6
Index is full, go to the next index
8
16
29
4
13
0
1
2
3
4
5
6
Insert here!

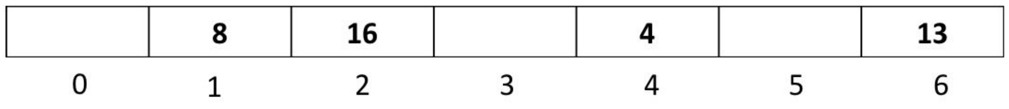
  - We stop when we find an empty spot to insert our data.
    - In the worst case scenario, it will run in O(n) time.

- **Primary Clustering**
  - Primary cluster is the largest sequential block of data in a hash table.
  - It is the source of the worst case running time in linear probing.
  - It takes the largest chunk of space; therefore data points have the highest probability of landing exactly there, which makes the cluster even bigger.
  - Possible solutions:
    - Could we jump by 2 steps instead of 1? - We will have the same problem on odd/even parity.
    - Could we jump by some random number? - This is the best solution to the primary cluster problem and it is explored as our second strategy to handling collisions → Double Hashing.

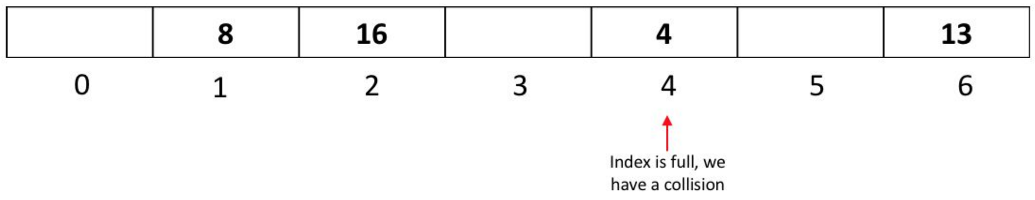
- **Double Hashing**
  - The idea behind double hashing is to add a second hash function, which is called a step function, where a good step function is:

        - Has to return values between 1 and (size - 1).
        - Needs to cover all elements of the array.
            - We need to make array size a prime or a relative prime (will not be divisible by the step size) to the range of the second hash function.
            - Ex: if the size was 6 and the step was 2, we would visit 0, 2, 4 in a cycle.

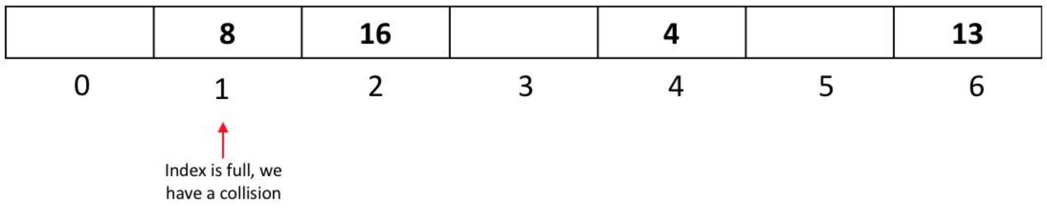
Double hashing example:



Insert 11: 11 % 7 = 4 → we have a collision.



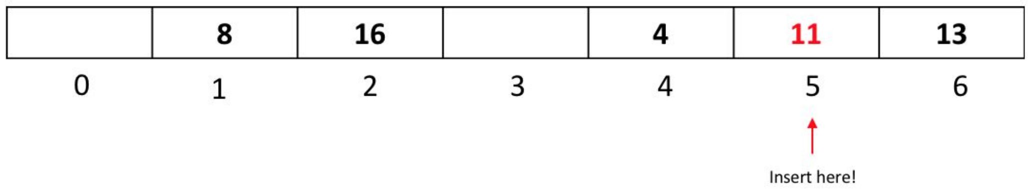
Let step function be: 5 - (k % 5) → 5 - (11 % 5) = 5 - 1 = 4; the step is 4.
We move by 4:



Index 1 is still full, so we move another 4 steps and since index 5 is free, we add:

CS 225 Spring 2019 :: TA Lecture Notes 3/15 Hashing

By Wenjie



- **Running Time:**
    - Separate chaining
        - Successful 1 + α/2
        - Unsuccessful 1 + α
    - Linear probing:
        - Successful ½(1+1/(1-α))
        - Unsuccessful ½(1+1/(1-α))^2
    - Double hashing:
        - Successful 1/α * iN(1/(1-α))
        - Unsuccessful 1/(1-α)