**Hash Table**

*Definition:* Hash Tables are used to store key-value pairs. They are like arrays, but the keys are not ordered.

*Dealing with Collisions:*

Separate Chaining: at each index in our array we store values using a more sophisticated data structure (e.g. an array or a linked list)

Linear Probing: when we find a collision, we search through the array to find the next empty slot. Unlike separate chaining, this allows us to store a single key-value at each index

*Pseudocode for Hash Table:*

set:

* Accept a key and a value
* Hash the key
* Store the key-value pair in the hash table array via separate chaining

get:

* Accept a key
* Hash the key
* Retrieve the key-value pair in the hash table
* If the key isn’t found, return undefined

key:

* Loop through the hash table array and returns an array of keys in the table

values:

* Loop through the hash table array and return an array of values in the table

*Solution for Hash Function:*

class HashTable{

constructor(size=53) {

this.keyMap = new Array(size);

}

hash(key) {

let total = 0;

let weird\_prime = 31;

for (var i = 0; i < Math.min(key.length, 100); i++) {

let char = key[i];

let value = char.charCodeAt(0) – 96;

total += (total\*weird\_prime + value) % this.keyMap.length;

}

return total;

}

set(key,value) {

var index = this.hash(key);

if (!this.keyMap[index]) {

this.keyMap[index] = [];

}

this.keyMap[index].push([key,value]);

}

get(key) {

var index = this.hash(key);

if (this.keyMap[index]) {

for (var i = 0; i < this.keyMap[index].length; i++) {

if (this.keyMap[index][i][0] === key) {

return this.keyMap[index][i][1];

}

}

}

return undefined;

}

values() {

var valuesArr = [];

for (var i = 0; i < this.keyMap.length; i++) {

if (this.keyMap[i]) {

for (var j = 0; j < this.keyMap[i].length; j++) {

if (!valuesArr.include(this.keyMap[i][j][1])) {

valuesArr.push(this.keyMap[i][j][1]);

}

}

console.log(this.keyMap[i]);

}

}

return valuesArr;

}

keys() {

var keysArr = [];

for (var i = 0; i < this.keyMap.length; i++) {

if (this.keyMap[i]) {

for (var j = 0; j < this.keyMap[i].length; j++) {

if (!keysArr.include(this.keyMap[i][j][0])) {

keysArr.push(this.keyMap[i][j][0]);

}

}

console.log(this.keyMap[i]);

}

}

return keysArr;

}

}

*Big O:*

Insert – O(1)

Deletion – O(1)

Access – O(1)