Hashing

size = 10

h(k) = k % SIZE

Keys Values

8, v1

3, v2

10, v3

4, v4

6, v5

13, v6

Collision

10, v3	0
	1
	2
3, v2	3
4, v4	4
	5
6, v5	6
	7
8, v1	8
	9

Hash Table

h(8) = 8 % 10 = 8
h(3) = 3 % 10 = 3
h(10) = 10 % 10 = 0
h(4) = 4 % 10 = 4
h(6) = 6 % 10 = 6
h(13) = 13 % 10 = 3
(collision)

store : O(1)
slot = k % size
arr[slot] = data;

retrieve : O(1)
slot = k % size
return arr[slot]

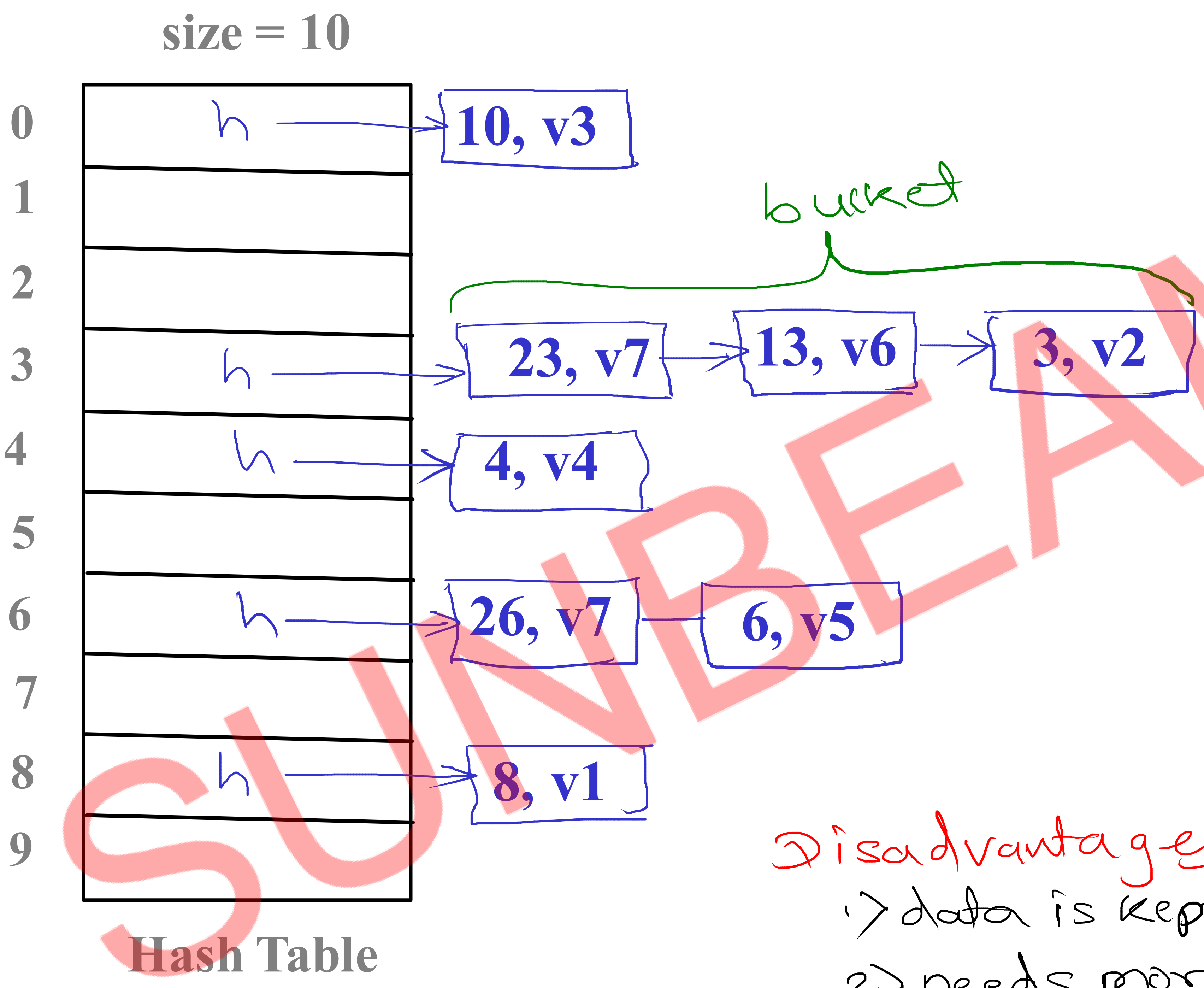
search : O(1)
slot = k % size
return arr[slot].data;

Collision :-
when two different keys yield same slot, it is called as collision

- Whenever collision will occur, next free slot will be find out by any one of the collision handling technique.

open probing → **Closed Addressing/ Seperate Chaining / Chaining**

8, v1
3, v2
10, v3
4, v4
6, v5
13, v6
23, v7
26, v7



h(k) = k % size

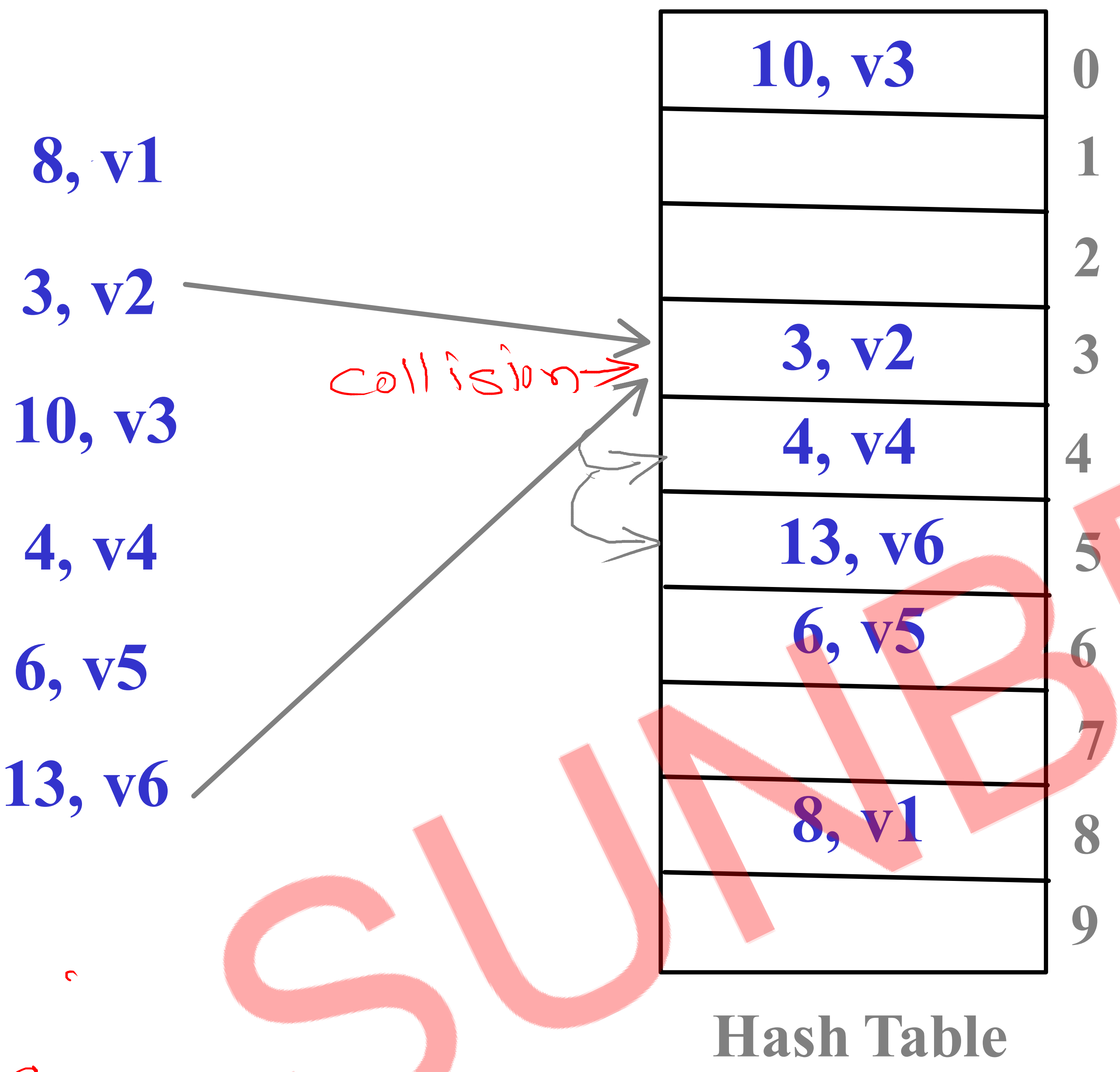
$h(8) = 8 \% 10 = 8$
$h(3) = 3 \% 10 = 3$
$h(10) = 10 \% 10 = 0$
$h(4) = 4 \% 10 = 4$
$h(6) = 6 \% 10 = 6$
$h(13) = 13 \% 10 = 3$
$h(23) = 23 \% 10 = 3$
$h(26) = 26 \% 10 = 6$

Disadvantages :-

1) data is kept out side the table
2) needs more space
3) one of the linked may grow heavily. in this case operations will not be performed in constant time.

closed probing → **Open Addressing - Linear Probing**

size = 10



h(k) = k % size

h(k,i) = [h(k) + f(i)] % size

f(i) = i

where i = 1, 2, 3,....

L probe number

$h(8) = 8 \% 10 = 8$

$h(3) = 3 \% 10 = 3$

$h(10) = 10 \% 10 = 0$

$h(4) = 4 \% 10 = 4$

$h(6) = 6 \% 10 = 6$

$h(13) = 13 \% 10 = 3$ (collision)

$h(13,1) = [3+1] \% 10$

$= 4$ (1^{st} probe)(collision)

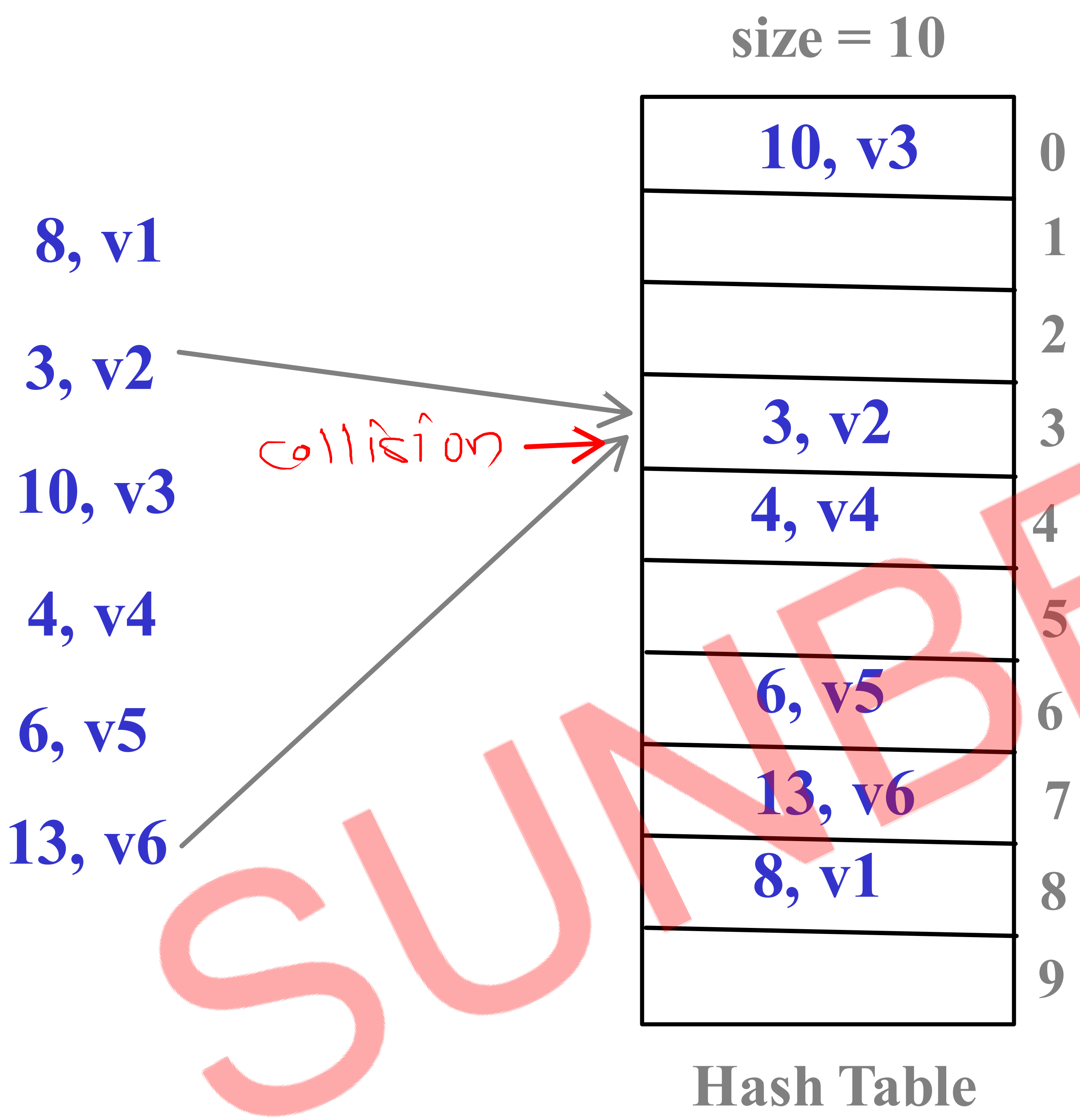
$h(13,2) = [3+2] \% 10$

$= 5$ (2^{nd} probe)

Probing :
– finding another free slot for a key when collision has occured

Primary clustering :
need long run of filled slots to find next empty slot "near" key positions

Open Addressing - Quadratic Probing



$h(k) = k \% size$

$h(k,i) = [h(k) + f(i)] \% size$

$f(i) = i^2$

where $i = 1, 2, 3, ...$

$h(13) = 13 \% 10 = 3$ c

$h(13,1) = [3+1] \% 10$

$= 4$ (1st probe) c

$h(13,2) = [3+4] \% 10$

$= 7$ (2nd probe)

- primary clustering is solved
- there is no garantee to get free slot

Open Addressing - Quadratic Probing

23, v7

33, v8

size = 10

	Hash Table
0	10, v3
1	
2	23, v7
3	3, v2
4	4, v4
5	
6	6, v5
7	13, v6
8	8, v1
9	33, v8

Hash Table

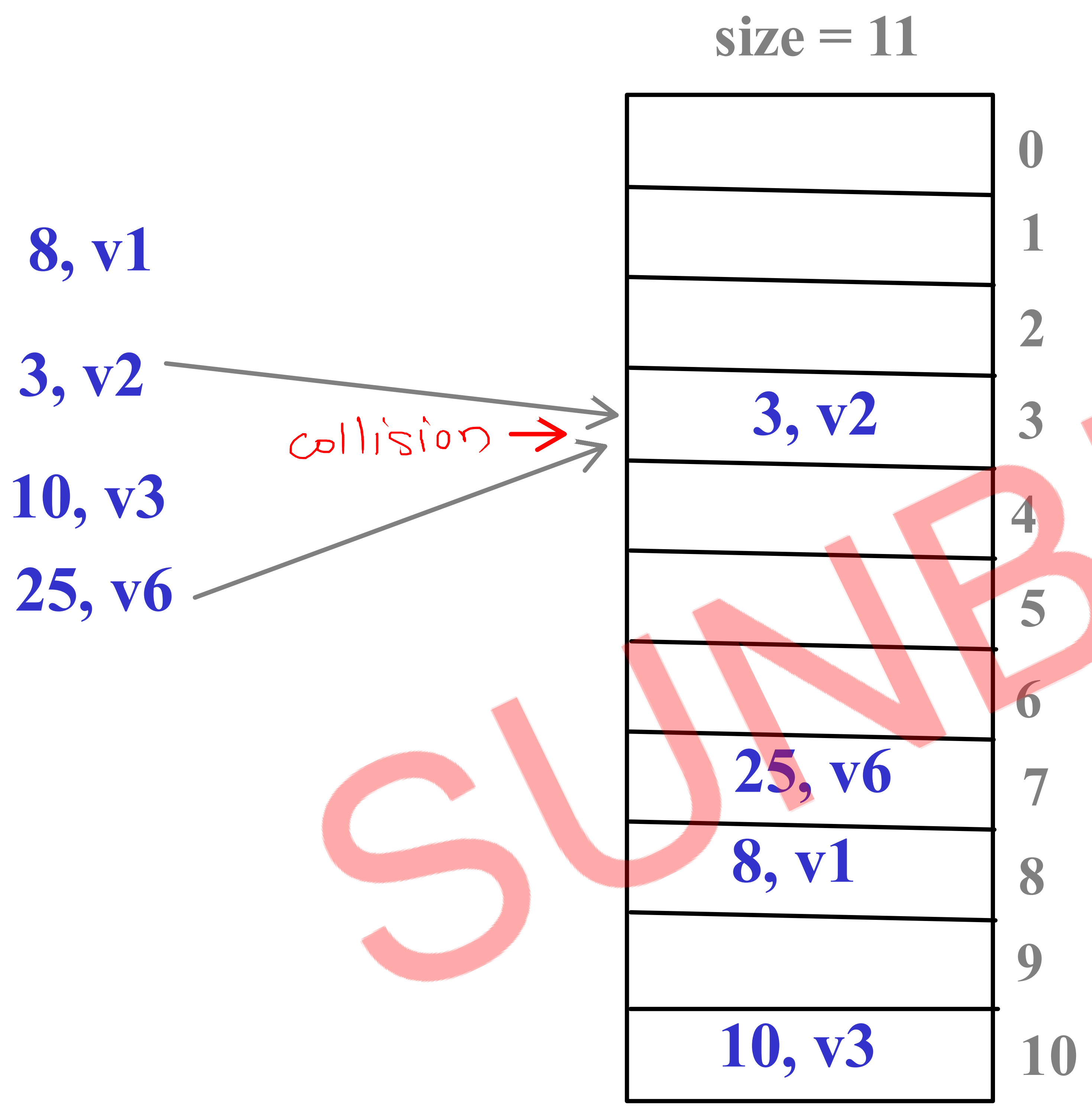
h(k) = k % size
h(k,i) = [h(k) + f(i)] % size
f(i) = i^2
where i = 1, 2, 3,....

$h(23) = 23 \% 10 = 3$ c
$h(23,1) = [3+1] \% 10 = 4$ (1^{st}) c
$h(23,2) = [3+4] \% 10 = 7$ (2^{nd}) c
$h(23,3) = [3+9] \% 10 = 2$ (3^{rd})

$h(33) = 33 \% 10 = 3$ c
$h(33,1) = [3+1] \% 10 = 4$ (1^{st}) c
$h(33,2) = [3+4] \% 10 = 7$ (2^{nd}) c
$h(33,3) = [3+9] \% 10 = 2$ (3^{rd}) c
$h(33,4) = [3+16] \% 10 = 9$ (4^{th})

Secondary clustering :
- need long run of filled slots to find empty slot "away" key position

Hashing - Double Hashing



Hash Table

$h1(k) = k \% size$
$h2(k) = 7 - (key \% 7)$

$h(k, i) = [\ h1(k) + i * h2(k)\ ] \% size$

$h_1(8) = 8 \% 11 = 8$
$h_1(3) = 3 \% 11 = 3$
$h_1(10) = 10 \% 11 = 10$
$h_1(25) = 25 \% 11 = 3$ c
$h_2(25) = 7 - (25 \% 7) = 4$
$h(25, 1) = [3 + 1 * 4] \% 11$
$= 7$ (1st probe)

$h_1(14) = 14 \% 11 = 3$ c
$h_2(14) = 7 - (14 \% 7) = 7$
$h(14, 1) = [3 + 1 * 7] \% 11 = 10$ c
$h(14, 2) = [3 + 2 * 7] \% 11 = 7$

Rehashing

e.g $\lambda = \frac{6}{10} = 0.6 \rightarrow$ hash table is 60% filled

$$\text{Load factor } (\lambda) = \frac{n}{N}$$

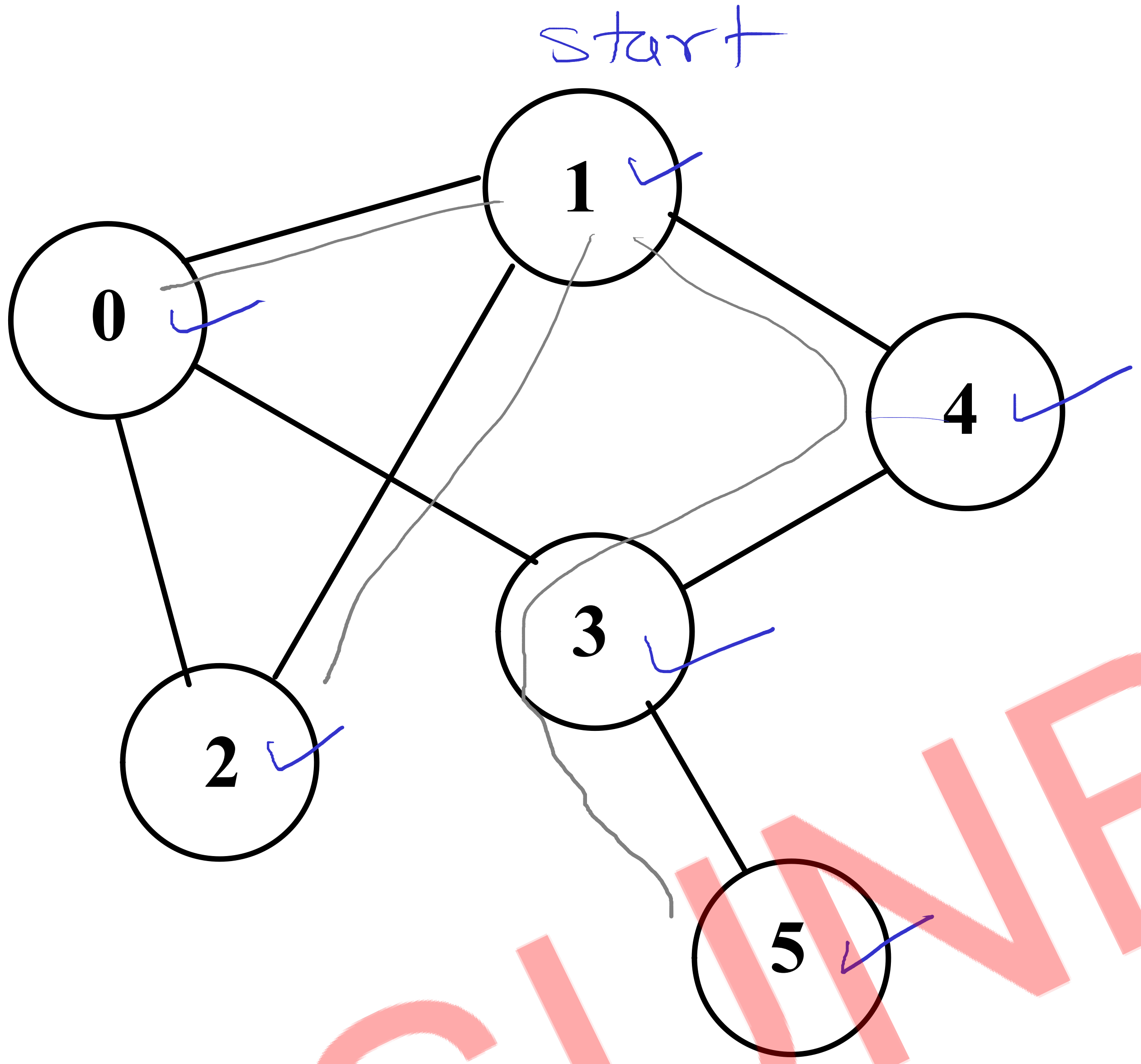
n - number of elements (key value pairs) in hash table
N - Number of slots in hash table

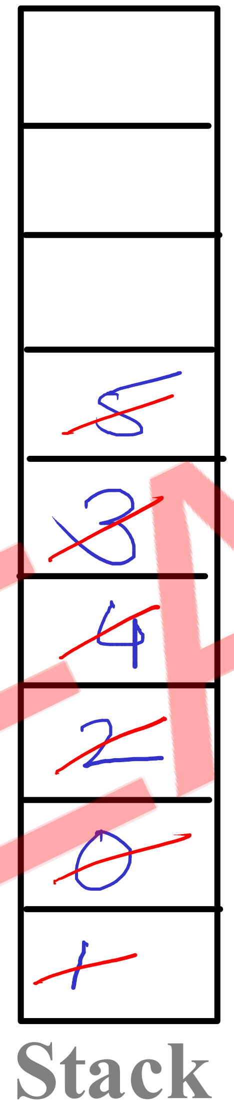
if $n < N$	load factor < 1	- free slots are aviable
if $n = N$	load factor = 1	- no free slots
if $n > N$	load factor > 1	- can not insert at all

- Rehashing is making the hash table size twice of existing size if hash table is 60 to 70 % full

- In rehashing existing keys are again mapped according to new size of table

DFS Traversal

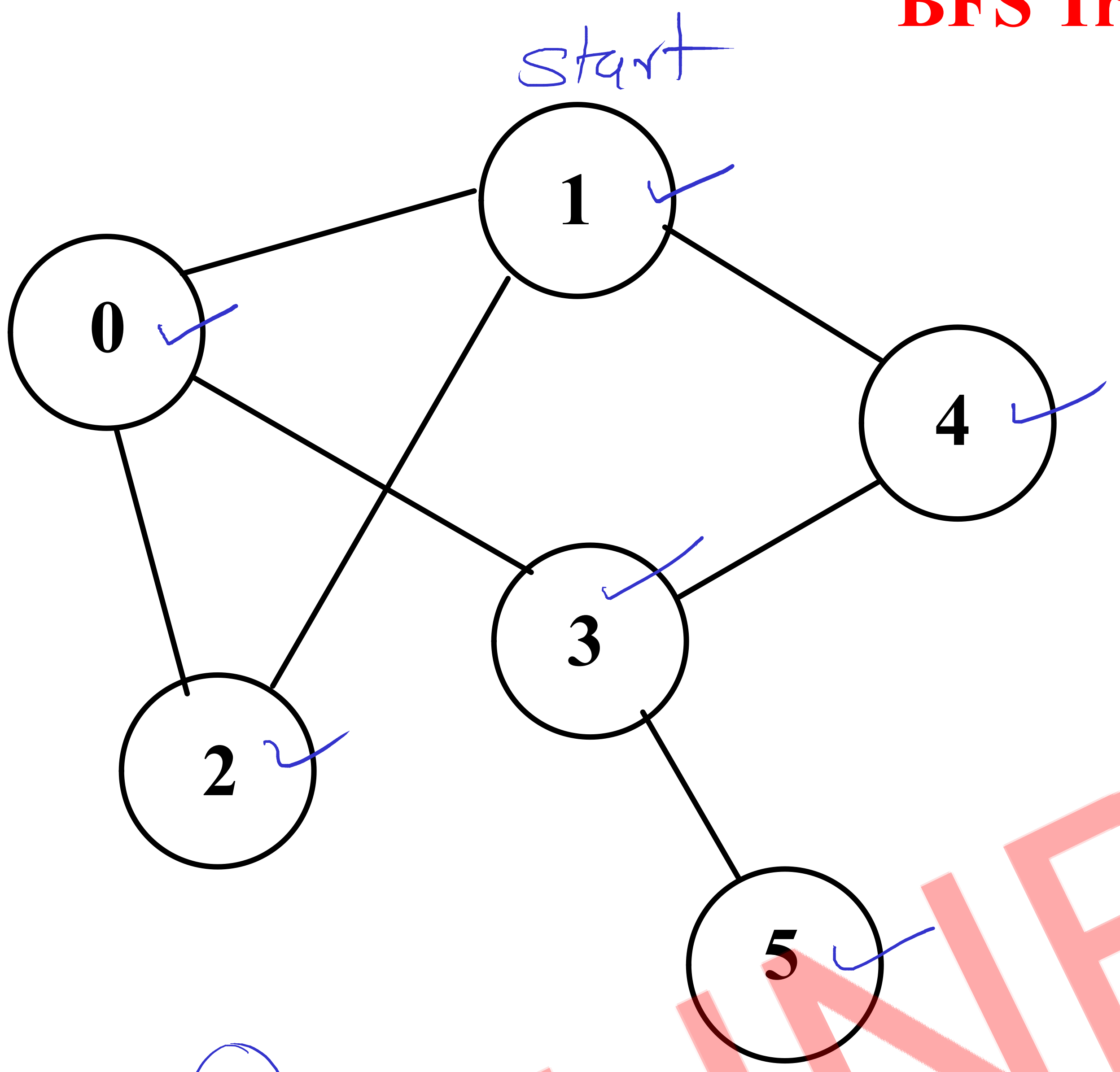


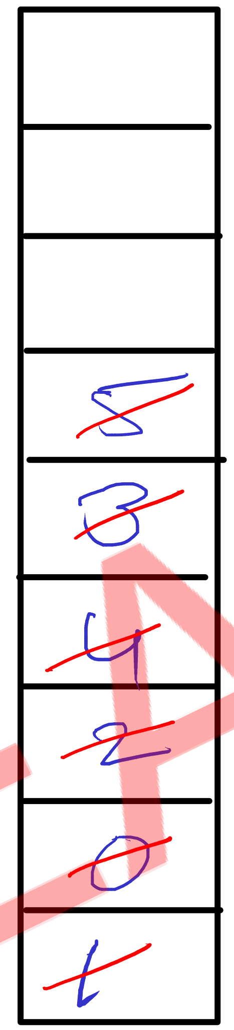


Traversal :
1, 4, 3, 5, 2, 0

//1. Choose a vertex as start vertex.
//2. Push start vertex on stack & mark it.
//3. Pop vertex from stack.
//4. Print the vertex.
//5. Put all non-visited neighbours of the vertex
//on the stack and mark them.
//6. Repeat 3-5 until stack is empty.

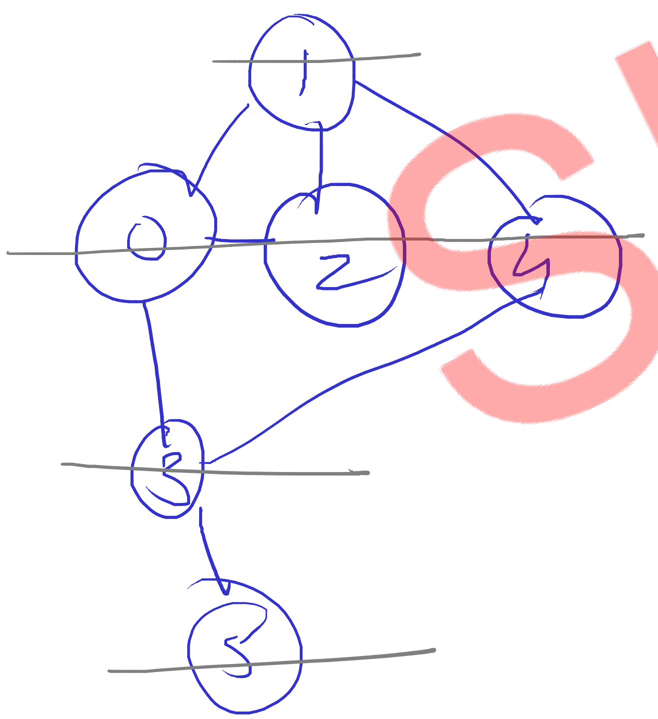
BFS Traversal





Queue

Traversal :
1, 0, 2, 4, 3, 5



//1. Choose a vertex as start vertex.
//2. Push start vertex on queue & mark it
//3. Pop vertex from queue.
//4. Print the vertex.
//5. Put all non-visited neighbours of the vertex
//on the queue and mark them.
//6. Repeat 3-5 until queue is empty.