SNS COLLEGE OF ENGINEERING

Kurumbapalayam (Po), Coimbatore – 641 107

An Autonomous Institution

Accredited by NBA – AICTE and Accredited by NAAC – UGC with 'A' Grade
Approved by AICTE, New Delhi & Affiliated to Anna University, Chennai

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING-IOT Including CS&BCT

COURSE NAME : **19CS307**- DATA STRUCTURES

II YEAR / III SEMESTER

Unit V- **SORTING AND SEARCHING**

Topic : ***Linear probing (Open Addressing or Closed Hashing)***

Linear probing (Open Addressing or Closed Hashing)

Linear probing (Open Addressing or Closed Hashing)

- In open addressing, instead of in linked lists, all entry records are stored in the array itself.
- He hash index of the hashed value is computed and then the array is examined (starting with the hashed index).
- If the slot at the hashed index is unoccupied, then the entry record is inserted in slot at the hashed index else it proceeds in some probe sequence until it finds an unoccupied slot.
- The name "open addressing" refers to the fact that the location or address of the item is not determined by its hash value.

Cont..

- Linear probing is when the interval between successive probes is fixed (usually to 1). Let's assume that the hashed index for a particular entry is **index**.
- The probing sequence for linear probing will be:

  index = index % hashTableSize
  index = (index + 1) % hashTableSize
  index = (index + 2) % hashTableSize
  index = (index + 3) % hashTableSize

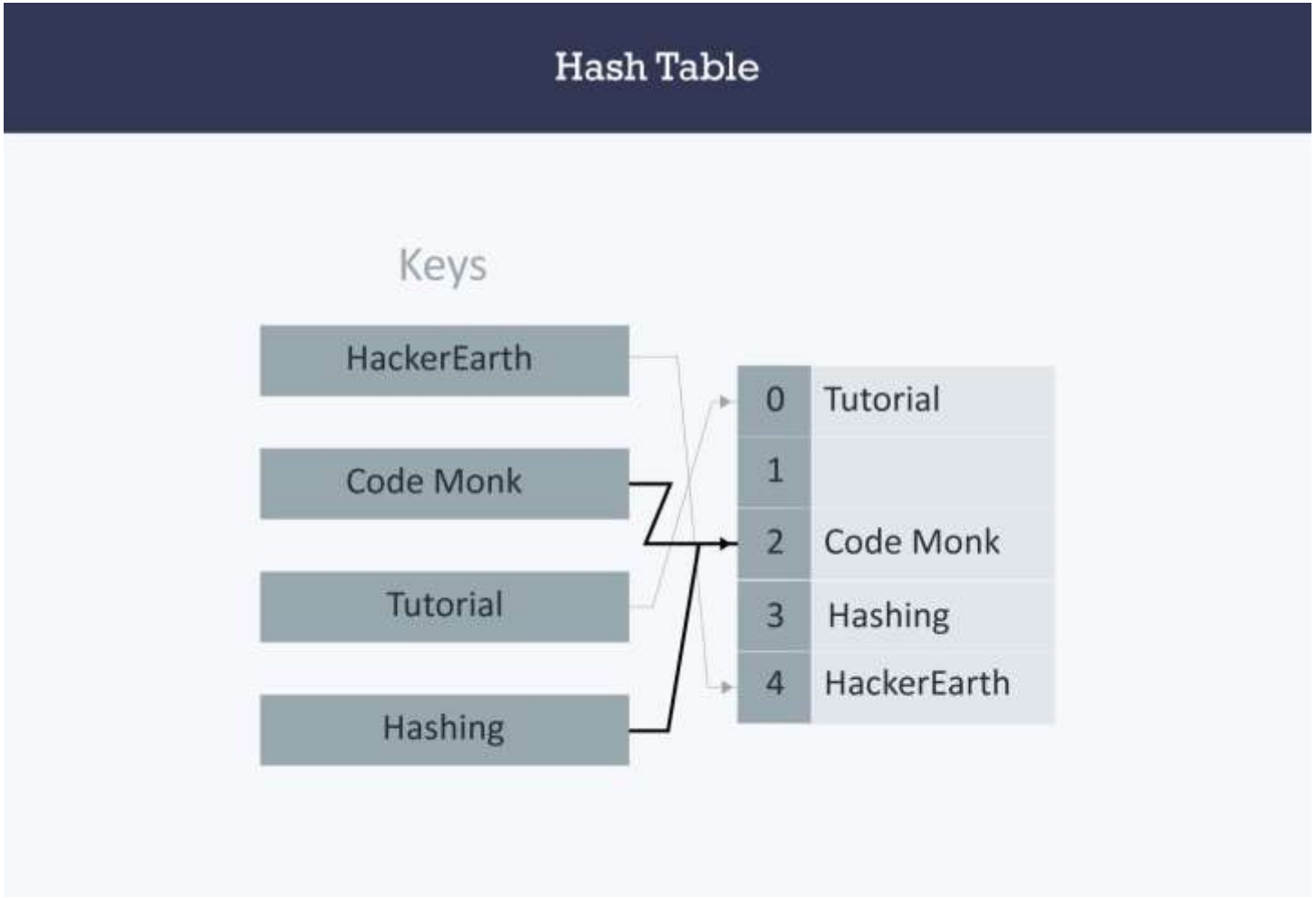
Implementation of hash table with linear probing (Syntax)

- string hashTable[21];
- int hashTableSize = 21;
- *Insert*
- void insert(string s)
- {
- //Compute the index using the hash function
- int index = hashFunc(s);
- //Search for an unused slot and if the index will exceed the hashTableSize then roll back
- while(hashTable[index] != "")
- index = (index + 1) % hashTableSize;
- hashTable[index] = s;
- }

Search

- void search(string s)
- {
- //Compute the index using the hash function int index = hashFunc(s);
- //Search for an unused slot and if the index will exceed the hashTableSize then roll back
- while(hashTable[index] != s and hashTable[index] != "")
- index = (index + 1) % hashTableSize;
- //Check if the element is present in the hash table
- if(hashTable[index] == s)
- cout << s << " is found!" << endl;
- else
- cout << s << " is not found!" << endl; }

Example


Hash Table
Keys
HackerEarth
Code Monk
Tutorial
Hashing
0
Tutorial
1
2
Code Monk
3
Hashing
4
HackerEarth

Thank you