Udemy

Algorithms and Data Structures in Java

Lecture: Very Simple HashMap

Instructors:
George Katsilidis
Nikos Katsilidis
Christos Topalidis

Very Siple HashMap

- In computing, a hash table (hash map) is a data structure which implements an associative array abstract data type, a structure that can map keys to values. A hash table uses a hash function to compute an index into an array of buckets or slots, from which the desired value can be found.
- Ideally, the hash function will assign each key to a unique bucket, but most hash table designs employ an imperfect hash function, which might cause hash collisions where the hash function generates the same index for more than one key. Such collisions must be accommodated in some way.

Very Siple HashMap

- In the current lecture we show the very simple hash table example. It uses simple hash function, collisions are resolved using linear probing (open addressing strategy) and hash table has constant size. This example clearly shows the basics of hashing technique.
- Underlying array has constant size to store 128 elements and each slot contains key-value pair. Key is stored to distinguish between key-value pairs, which have the same hash.
- Hash function to be used is the remainder of division by 128. In the view of implementation, this hash function can be encoded using remainder operator or using bitwise AND with 127.
- Note. Power of two sized tables are often used in practice . When used, there is a special hash function, which is applied in addition to the main one. This measure prevents collisions occuring for hash codes that do not differ in lower bits

Collision Resolution Strategy

- Linear probing is applied to resolve collisions. In case the slot, indicated by hash function, has already been occupied, algorithm tries to find an empty one by probing consequent slots in the array.
- Note. Linear probing is not the best techique to be used when table is of a constant size. When load factor exceeds particular value (appr. 0.7), hash table performance will decrease nonlinearly. Also, the number of stored key-value pairs is limited to the size of the table (128).
- Code snippets
- This implementation suffers one bug. When there is no more place in the table, the loop, searching for empty slot, will run infinitely. It won't happen in real hash table based on open addressing, because it is most likely dynamic-sized. Also the removal's implementation is omitted to maintain simplicity. See open addressing strategy for full implementation.

Hash node Code

```
public class HashEntry {
    private int key;
    private String value;
        HashEntry(int key, String value) {
          this.key = key;
          this.value = value;
    }
        public int getKey() {
          return key;
    }
        public String getValue() {
          return value;
    }
}
```

HashMap Code

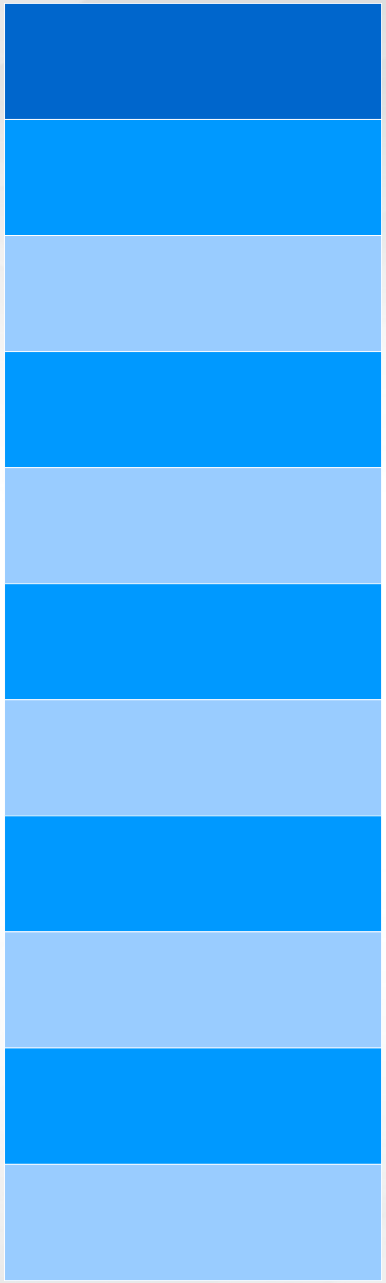
```
public class HashMap {
    private final static int TABLE_SIZE = 128;

    HashEntry[] table;

    HashMap() {
        table = new HashEntry[TABLE_SIZE];
        for (int i = 0; i < TABLE_SIZE; i++)
            table[i] = null;
    }
...
}
```

Linear Probing example

"George"

Key("George") = 601

Hash (601) = 7

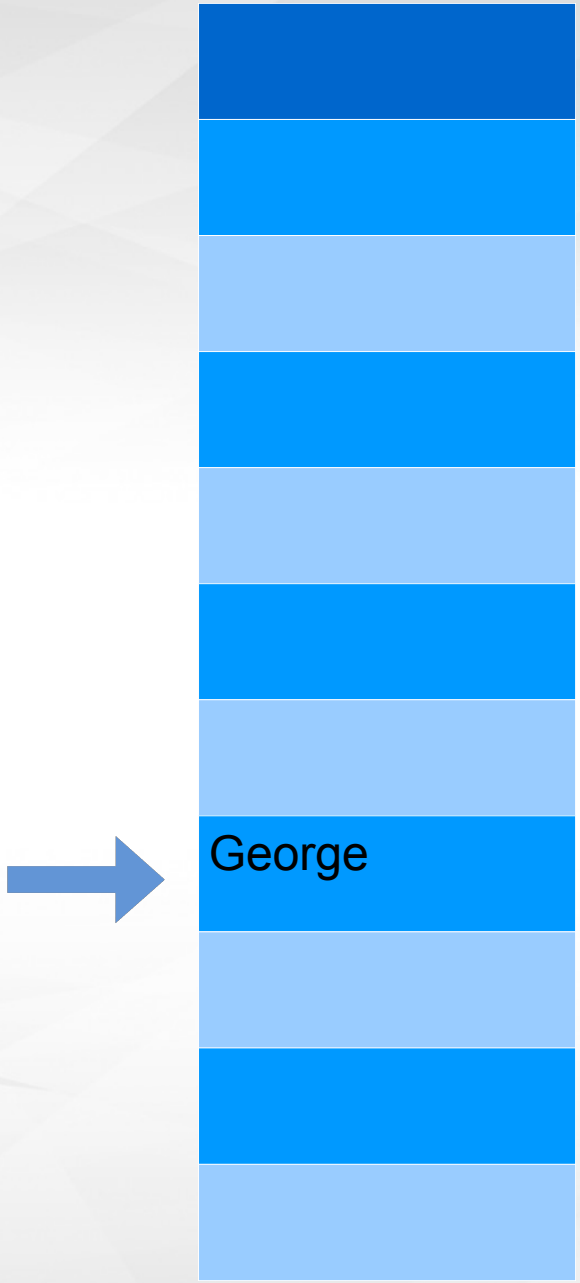


Linear Probing example

"George"

Key("George") = 601

Hash (601) = 7



Linear Probing example

"George"

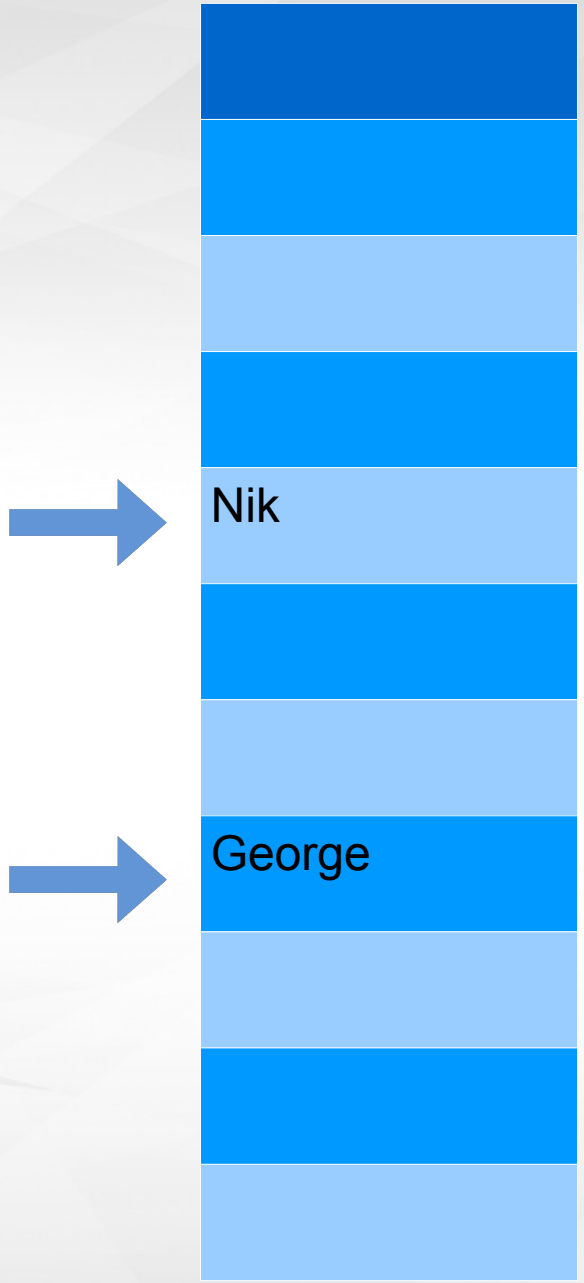
Key("George") = 601

Hash (601) = 7

"Nik"

Key("Nik") = 290

Hash (Key("Nik")) = 4



Linear Probing example

"George"

Key("George") = 601

Hash (601) = 7

"Nik"

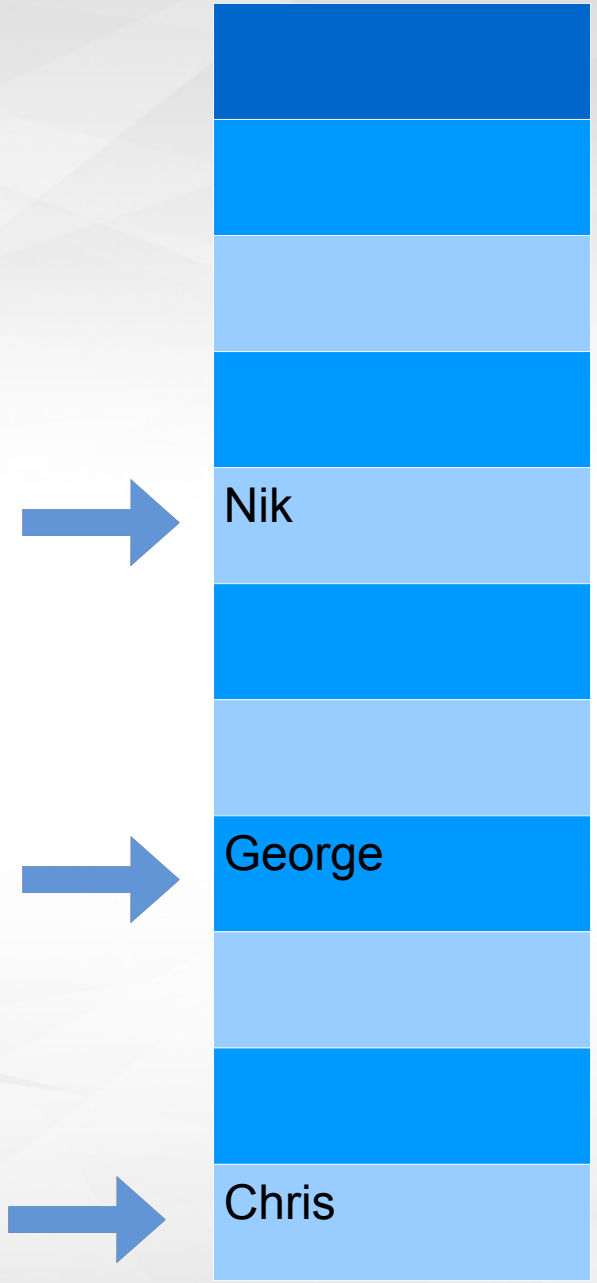
Key("Nik") = 290

Hash (Key("Nik")) = 4

"Chris"

Key("Chris") = 505

Hash (Key("Chris")) = 10

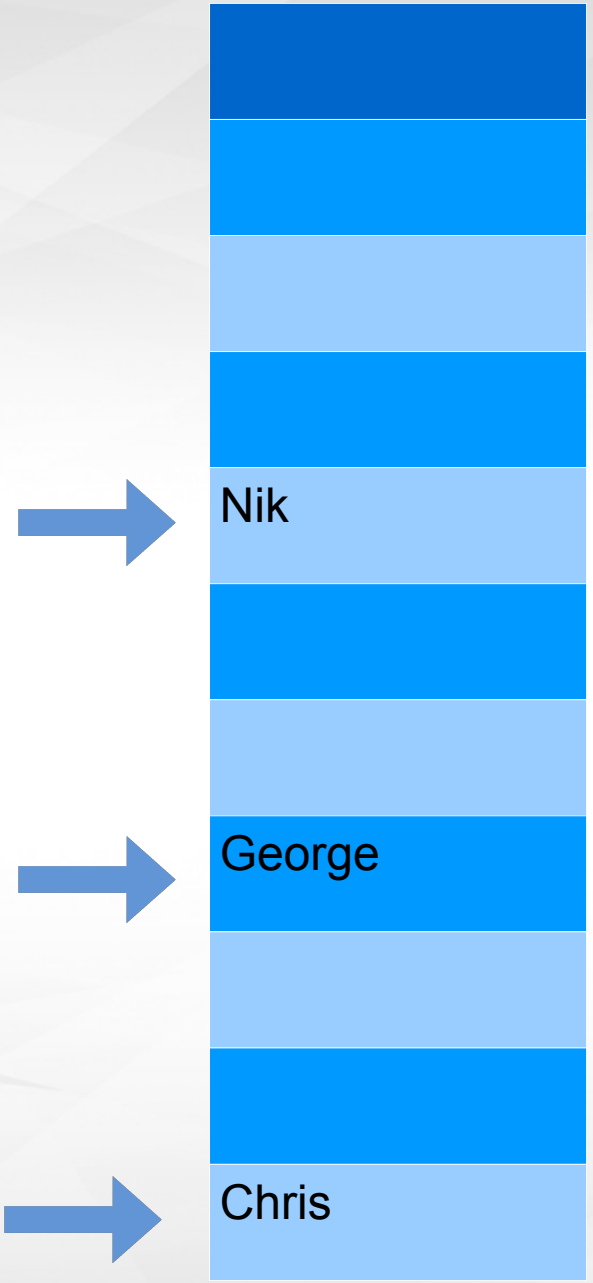


Linear Probing example

"George"

Key("George") = 601

Hash (601) = 7 collision !



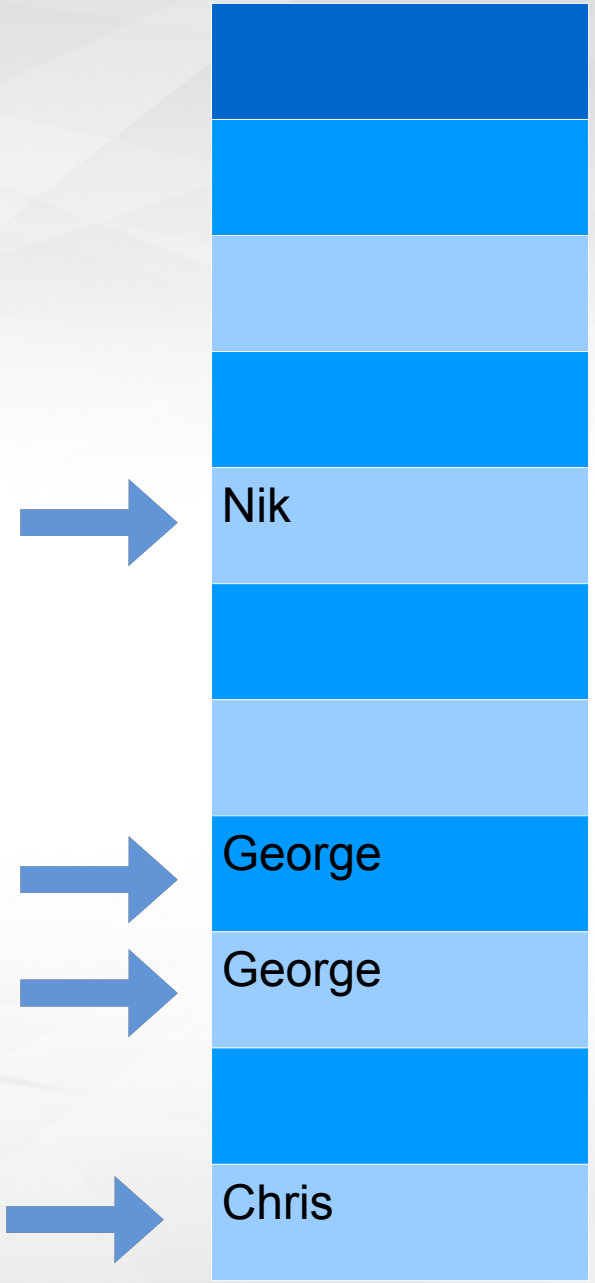
Linear Probing example

"George"

Key("George") = 601

Hash (601) = 7 collision !

Hash = 8



Linear Probing example

"George"

Key("George") = 601

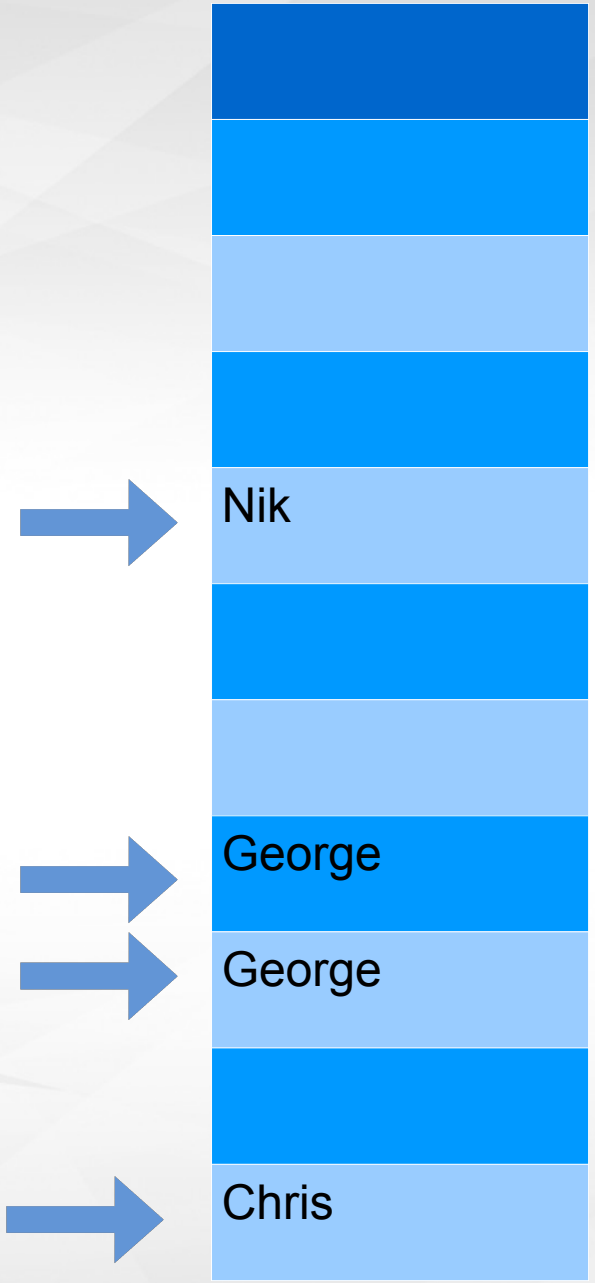
Hash (601) = 7 collision !

Hash = 8

"Nik"

Key("Nik") = 290

Hash (Key("Nik")) = 4collision !



Linear Probing example

"George"

Key("George") = 601

Hash (601) = 7 collision !

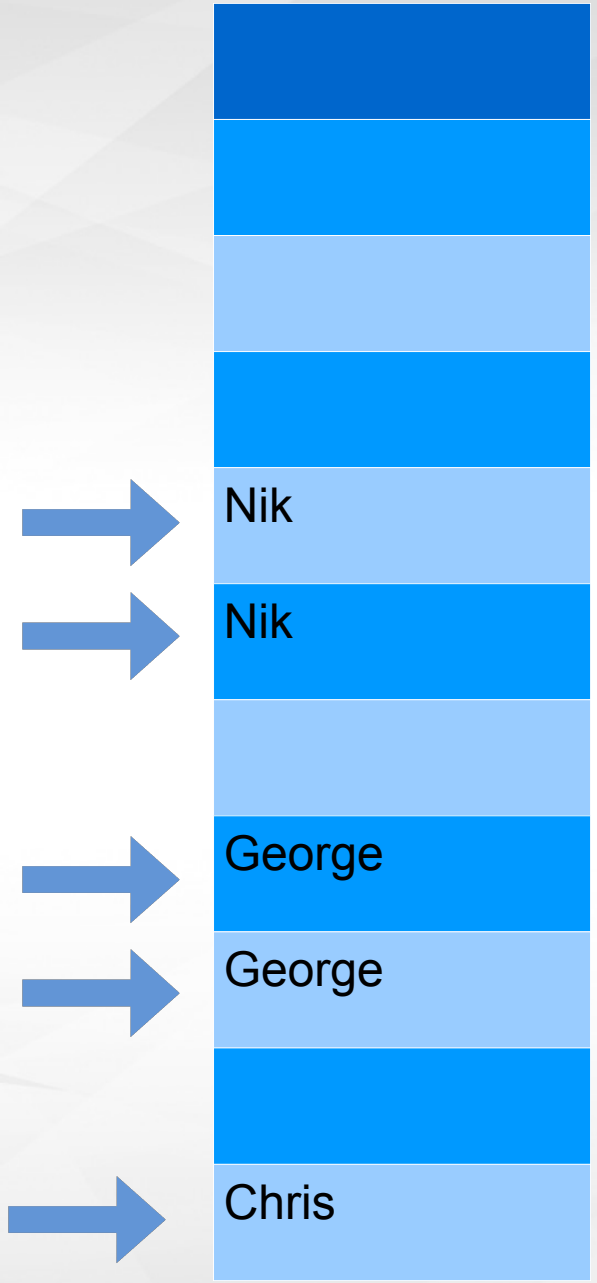
Hash = 8

"Nik"

Key("Nik") = 290

Hash (Key("Nik")) = 4collision !

Hash = 5



Linear Probing example

"George"

Key("George") = 601

Hash (601) = 7 collision !

Hash = 8

"Nik"

Key("Nik") = 290

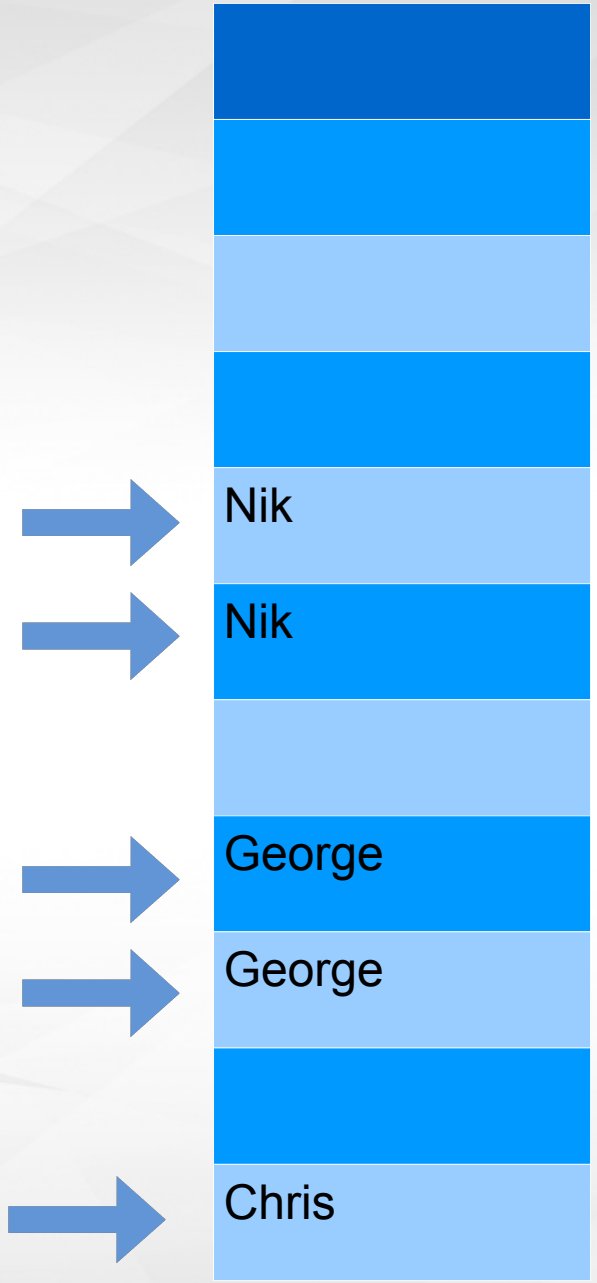
Hash (Key("Nik")) = 4collision !

Hash = 5

"Chris"

Key("Chris") = 505

Hash (Key("Chris")) =10collision !



Linear Probing example

"George"

Kay("George") = 601

Hash (601) = 7 collision !

Hash = 8

"Nik"

Key("Nik") = 290

Hash (Key("Nik")) = 4collision !

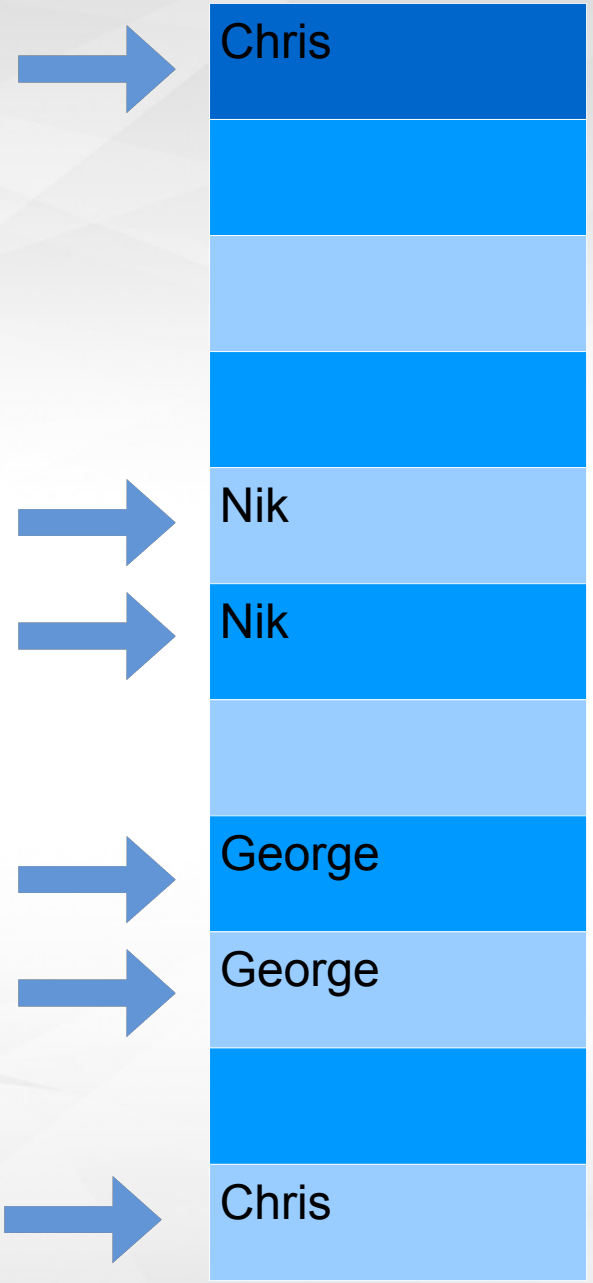
Hash = 5

"Chris"

Key("Chris") = 505

Hash (Key("Chris")) =10collision !

Hash = 0



Linear Probing example

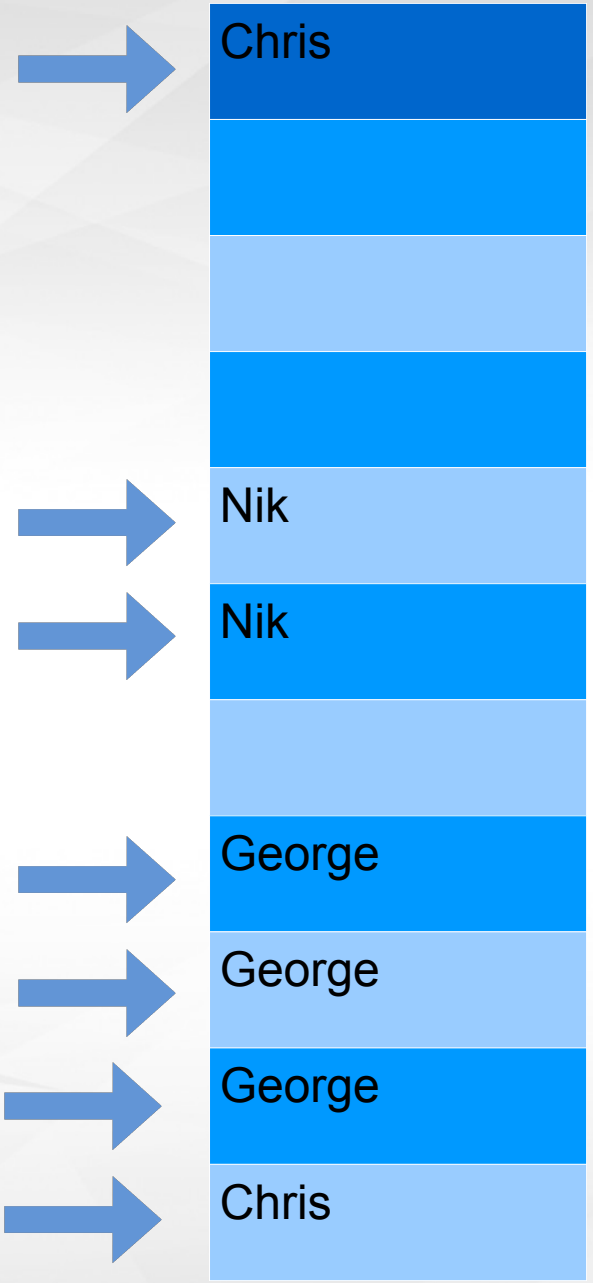
"George"

Key("George") = 601

Hash (601) = 7 collision !

Hash = 8 collision !

Hash = 9



Linear Probing example

"George"

Key("George") = 601

Hash (601) = 7 collision !

Hash = 8 collision !

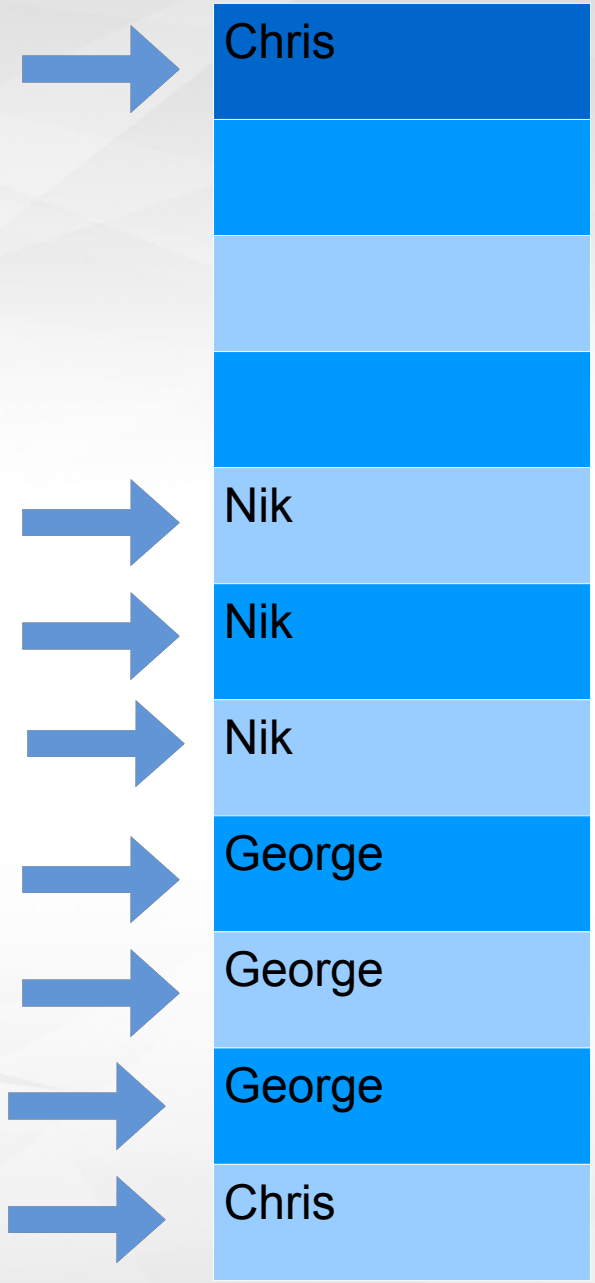
Hash = 9

Key("Nik") = 290

Hash (Key("Nik")) = 4 collision !

Hash = 5 collision !

Hash = 6



Linear Probing example

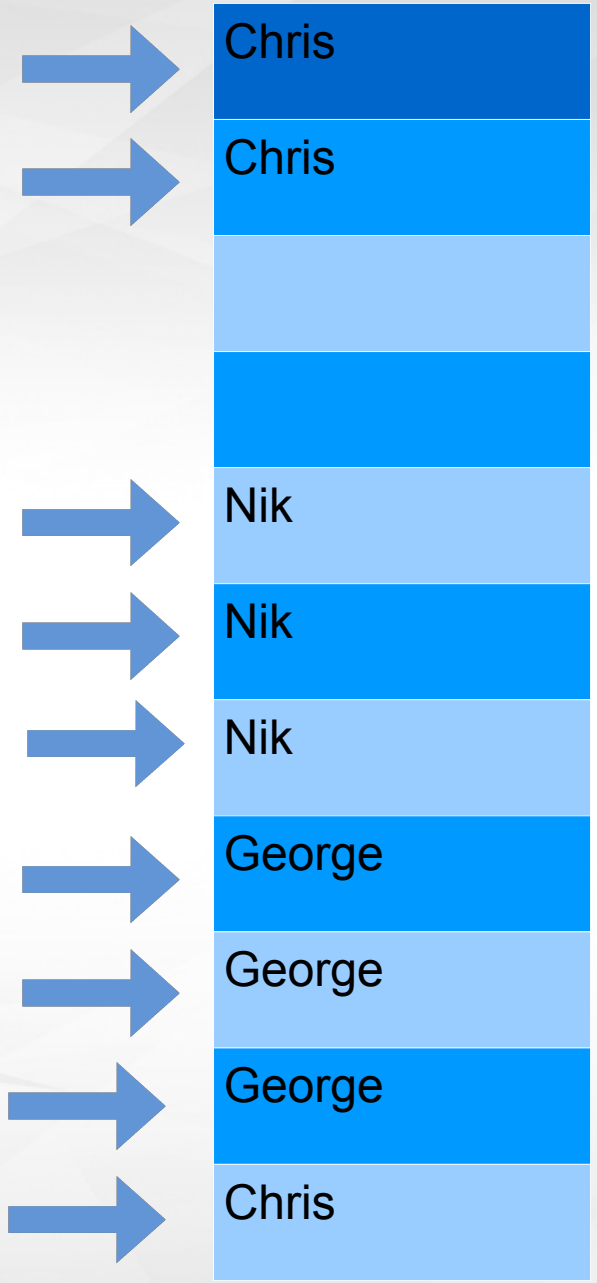
"Chris"

Key("Chris") = 505

Hash (505) = 10 collision !

Hash = 0 collision !

Hash = 1



Linear Probing example

"George"

Key("George") = 601

Hash (601) = 7 collision !

Hash = 8 collision !

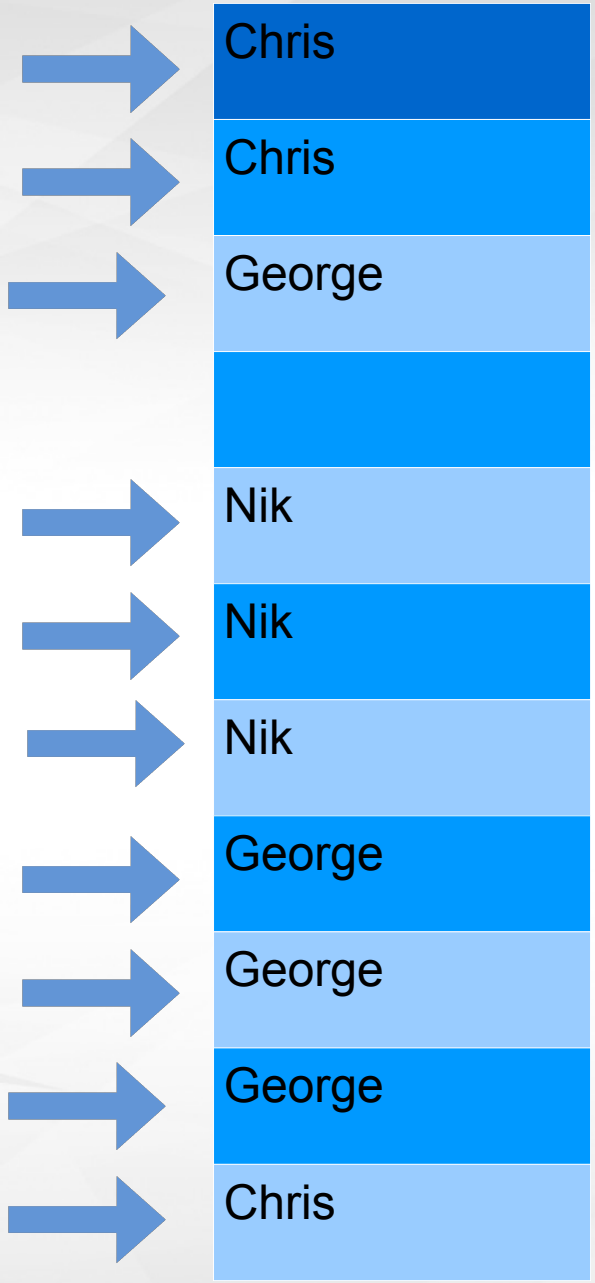
Hash = 9 collision !

Hash = 10 collision !

Hash = 0 collision !

Hash = 1 collision !

Hash = 2



Linear Probing example

Key("Nik") = 290

Hash (Key("Nik")) = 4 collision !

Hash = 5 collision !

Hash = 6 collision !

Hash = 7 collision !

Hash = 8 collision !

Hash = 9 collision !

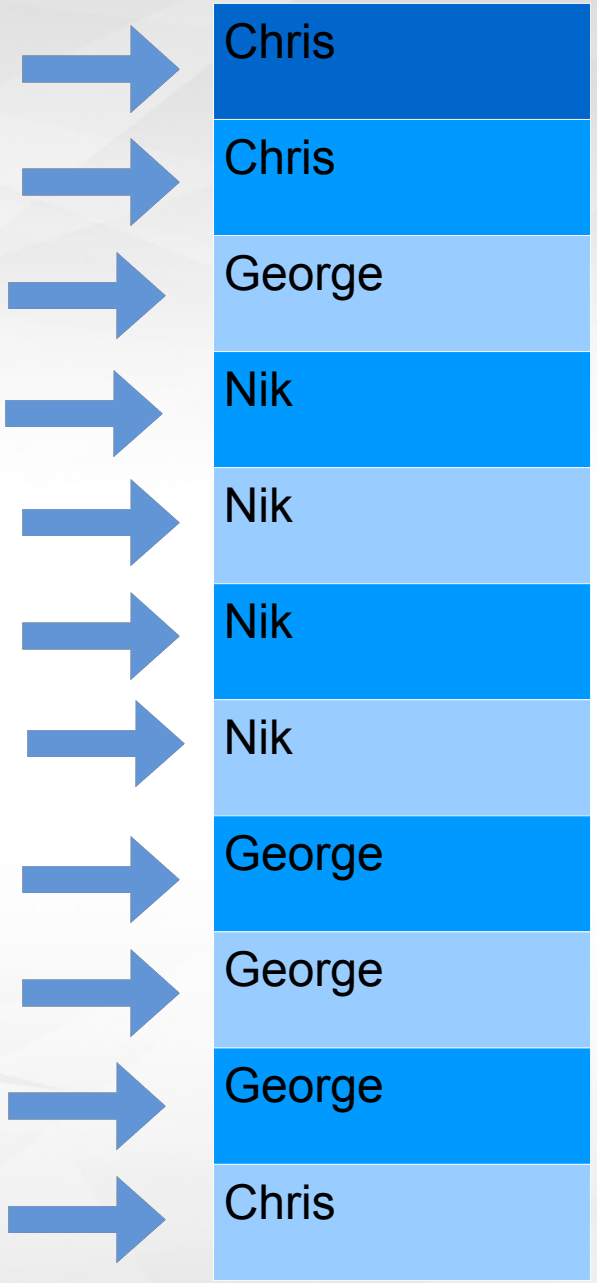
Hash = 10 collision !

Hash = 0 collision !

Hash = 1 collision !

Hash = 2 collision !

Hash = 3



Linear Probing example

"Chris"

Key("Chris") = 505

Hash (505) = 10 collision !

Hash = 0 collision !

Hash = 1 collision !

Hash = 2 collision !

Hash = 3 collision !

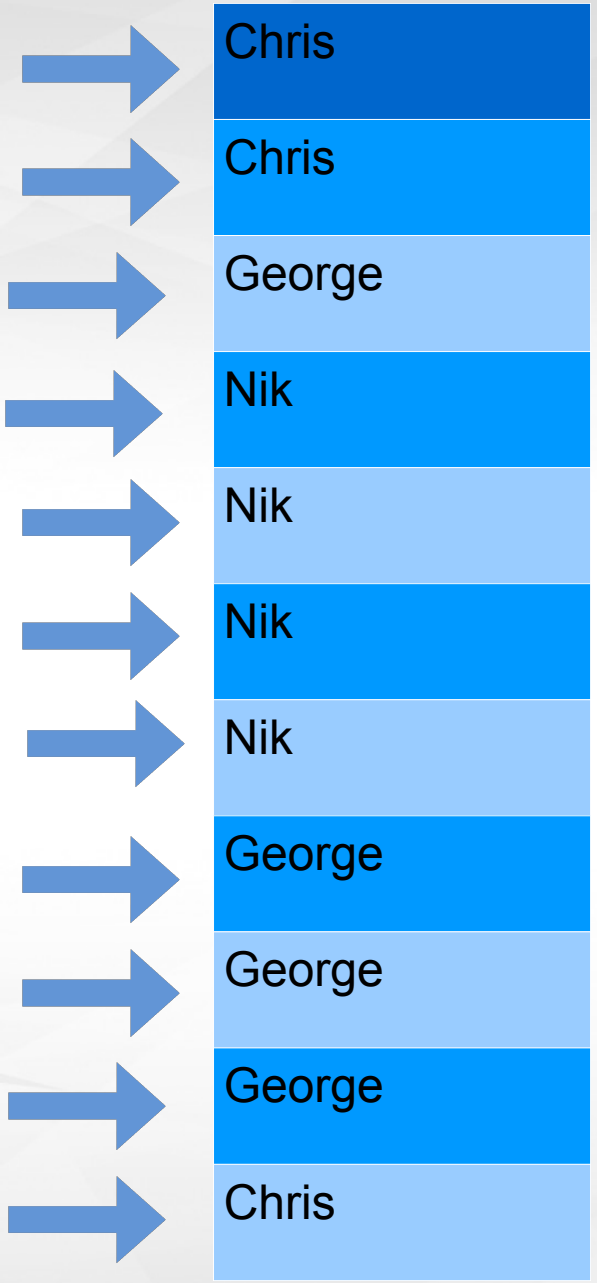
Hash = 4 collision !

Hash = 5 collision !

Hash = 6 collision !

Hash = 7 collision !

Hash = 8 collision ! ....



Key method

```
int getKey(String username){
      int i,j=0;
      for(i=0;i<username.length();i++)
          j = j+ username.charAt(i);
     return j;
   }
```

Put method

```
public void put(int key, String value) {
        int hash = (key % TABLE_SIZE);
        while (table[hash] != null && table[hash].getKey() != key)
            hash = (hash + 1) % TABLE_SIZE;

        table[hash] = new HashEntry(key, value);
    }
```