Hash tables

Johan Montelius

KTH

HT23

Let's build a key-value store.

Let's build a key-value store.

- A linked list : ok, but $O(n)$ lookup operation.

Let's build a key-value store.

- A linked list : ok, but $O(n)$ lookup operation.
- A sorted tree : much better, $O(lg(n))$ operations (add and lookup).

Let's build a key-value store.

- A linked list : ok, but $O(n)$ lookup operation.
- A sorted tree : much better, $O(lg(n))$ operations (add and lookup).
- A sorted array : binary search gives us $O(lg(n))$ lookup, but ...

zip code data base

zip code data base

A file containing: zip code, name and population.

```
111 15,STOCKHOLM              ,3
111 20,STOCKHOLM              ,50
111 21,STOCKHOLM              ,344
111 22,STOCKHOLM              ,149
 :
 :
```

An array of nodes

```
pubic class Zip {
  Node[] data;

  private class Node {
    String code;
    String name;
    Integer pop;
    :
  }
  :
```

zip codes are ordered

```
public Zip(String file) {

  data = new Node[10000];
    :

  data[i++] = new Node(row[0], row[1], Integer.valueOf(row
    :
}
```

binary search

```
public String binary(String zip) {
   int mn = 0;
   int mx = max;

   while (true) {
     int index = (mn + mx)/2;

     int cmp zip.compareTo(data[index].code);

     if (cmp == 0) {
       return data[index].name;
     }
     :
   }
   return null;
 }
```

use zip code as index

```
public Zip(String file) {

   data = new Node[100000];
   :

   Integer key = Integer.valueOf(row[0].replaceAll("\\s","

   data[key] = new Node(key, row[1], Integer.valueOf(row[2
   :
 }
```

perfect ....

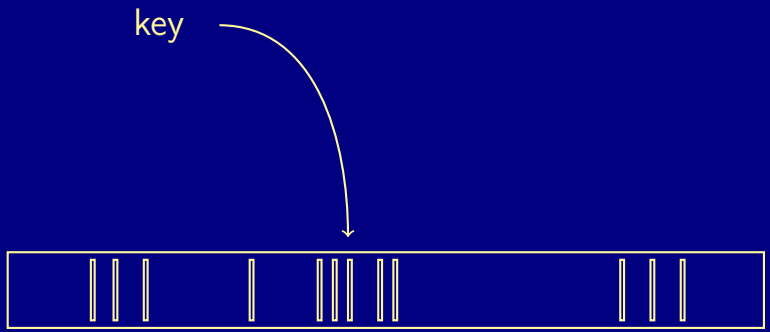
perfect ....

- $O(1)$ lookup

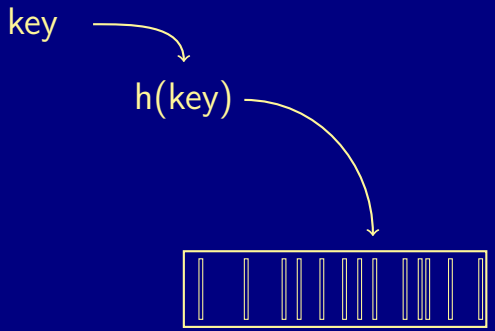
perfect ....

- $O(1)$ lookup
- what's the problem?

using key as index



the hash function



hash function

hash function

- A hash function h takes an key as input and generates an index: $0..k$

hash function

- A hash function h takes an key as input and generates an index: $0..k$
- Keys are evenly distributed over the range of indices.
- The range $0..k$ is *resonable* (?) small.

hash function

- A hash function h takes an key as input and generates an index: $0..k$
- Keys are evenly distributed over the range of indices.
- The range $0..k$ is *resonable* (?) small.
- *Few* (?) keys map to the same index.

example of hash function

```
public static int hash(Integer key, int M) {
  return key % M;
}
```

how about a string

how about a string

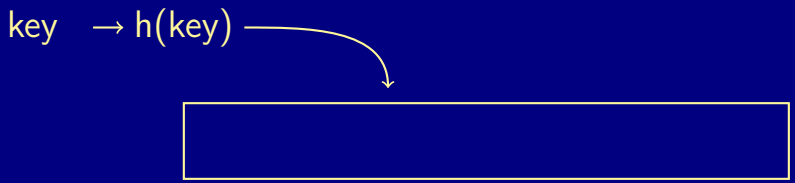
```
int R = 31;

public static Integer hash(String key, int M) {

    char[] chars = key.toCharArray();

    int value = 0;
    for (int i = 0; i < chars.length; i++) {
        value = (R * value + chars[i]) % M;
    }
    return value;
}
```

collisions

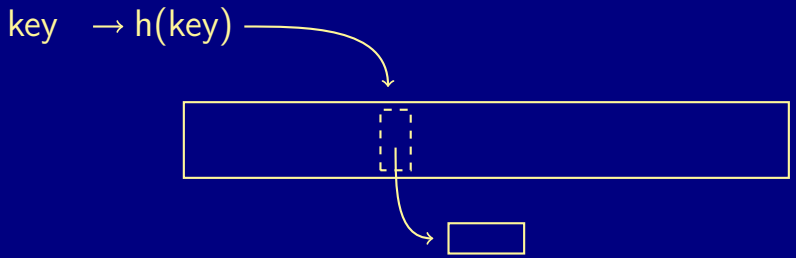
collisions

What should we do if we have collisions?

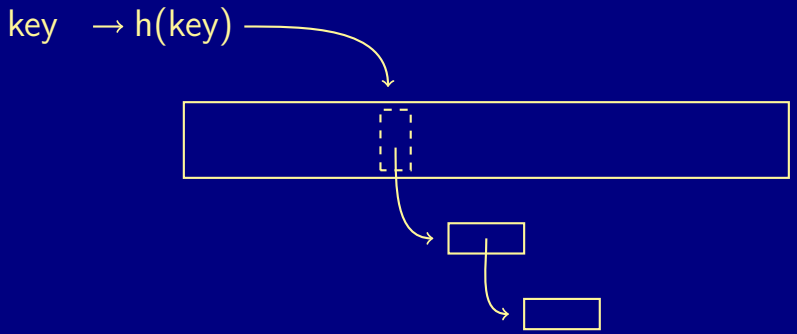
buckets



buckets



buckets



buckets

buckets

- Simple to implement, robust behaviour.

buckets

- Simple to implement, robust behaviour.
- Size of the table, m, can be half of n or less.

buckets

- Simple to implement, robust behaviour.
- Size of the table, m, can be half of n or less.
- Bucket as: linked list, array, ordered tree ....

buckets

- Simple to implement, robust behaviour.
- Size of the table, m, can be half of n or less.
- Bucket as: linked list, array, ordered tree ....

open adressing

open adressing

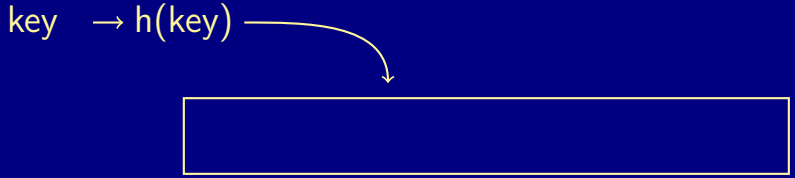
What if the hash function give us an index but we have options of where to place an item.

open adressing

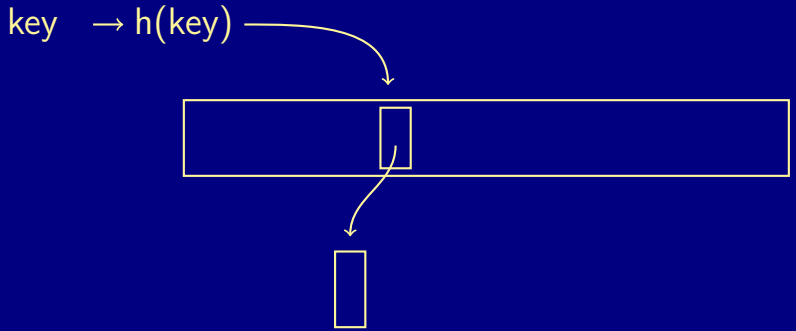
What if the hash function give us an index but we have options of where to place an item.

We need to search for the item.

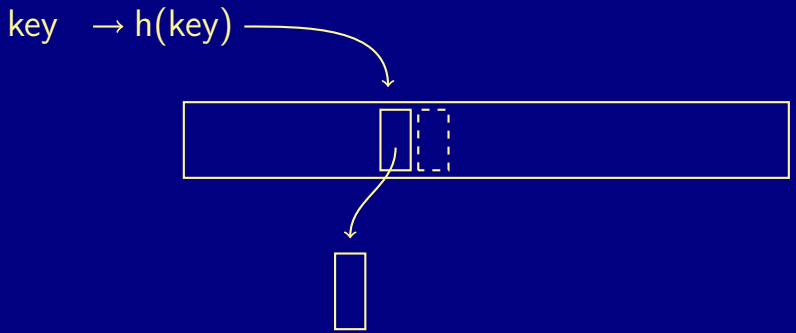
linear probing



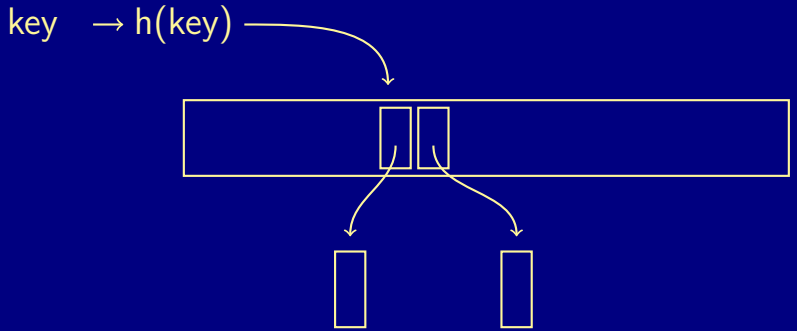
linear probing



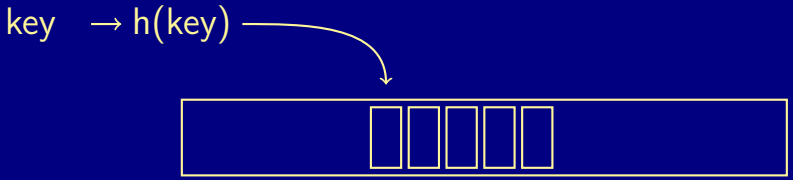
linear probing



linear probing



clustering



linear probing

linear probing

- Simple to implement.

linear probing

- Simple to implement.
- Growing *clusters* a problem.

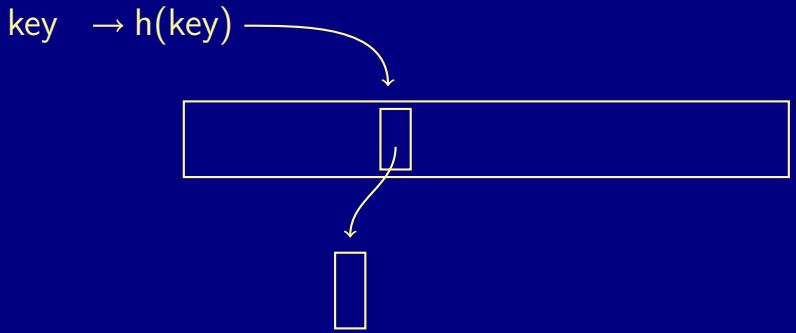
linear probing

- Simple to implement.
- Growing *clusters* a problem.
- Size of the table, m, should be at least $2 \times n$.

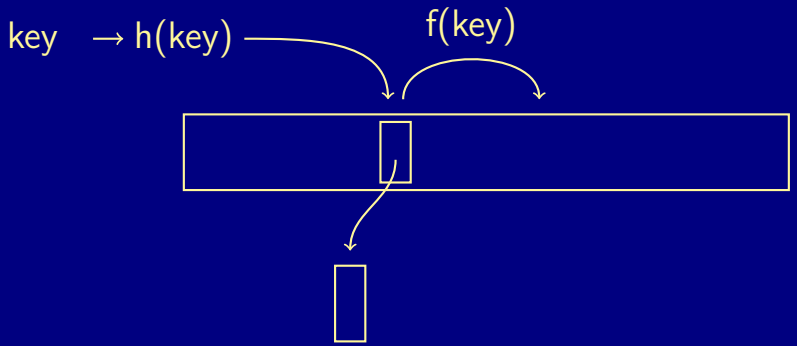
linear probing

- Simple to implement.
- Growing *clusters* a problem.
- Size of the table, m, should be at least $2 \times n$.

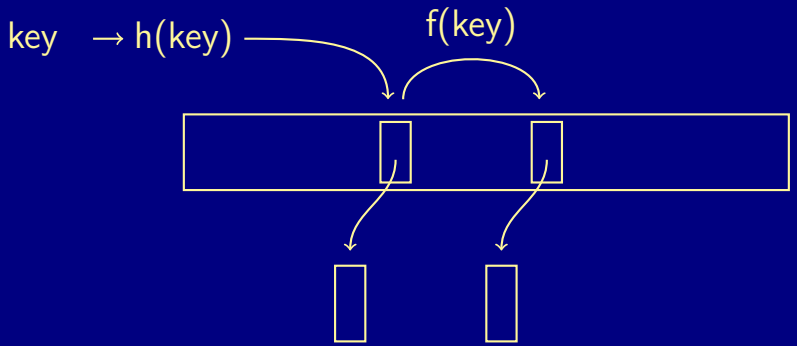
double hashing



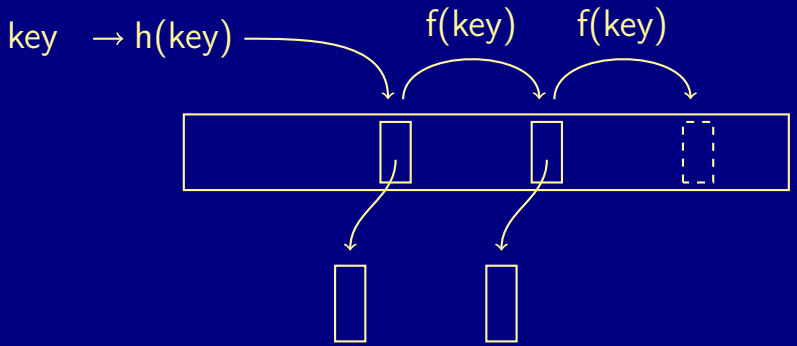
double hashing



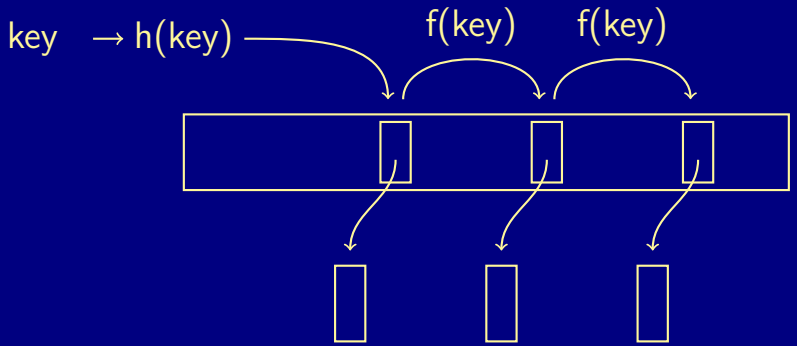
double hashing



double hashing



double hashing



double hashing

double hashing

- Slightly more complicated.

double hashing

- Slightly more complicated.
- The second hash function should give us an offset 1..d where d is relative prime to m (choose $m - 1$).

double hashing

- Slightly more complicated.
- The second hash function should give us an offset 1..d where d is relative prime to m (choose $m - 1$).
- Growing clusters less of a problem.

double hashing

- Slightly more complicated.
- The second hash function should give us an offset 1..d where d is relative prime to m (choose $m - 1$).
- Growing clusters less of a problem.
- Size of the table, m, could approach n in more advanced schemes.

double hashing

- Slightly more complicated.
- The second hash function should give us an offset 1..d where d is relative prime to m (choose $m - 1$).
- Growing clusters less of a problem.
- Size of the table, m, could approach n in more advanced schemes.

what about remove

what about remove

- buckets: not a problem

what about remove

- buckets: not a problem
- open adressing: .... problem

R.I.P

R.I.P

Replace removed key/values with a tombstone.

increase the size

increase the size

In a dynamic array we simply copied everything to a larger array ... problem?

cryptographic hash function

cryptographic hash function

The hash functions that we have used are quite simple.
Cryptographic hash functions have more requirements:

cryptographic hash function

The hash functions that we have used are quite simple.
Cryptographic hash functions have more requirements:

- extremly unlikely that two key have the same hash value

cryptographic hash function

The hash functions that we have used are quite simple.
Cryptographic hash functions have more requirements:

- extremly unlikely that two key have the same hash value
- no efficient way of finding the key given a hash value

cryptographic hash function

The hash functions that we have used are quite simple.
Cryptographic hash functions have more requirements:

- extremly unlikely that two key have the same hash value
- no efficient way of finding the key given a hash value

cryptographic hash function

The hash functions that we have used are quite simple.
Cryptographic hash functions have more requirements:

- extremly unlikely that two key have the same hash value
- no efficient way of finding the key given a hash value

Computing a chryptographic hash is more expensive.