Data Structures and Algorithms

—

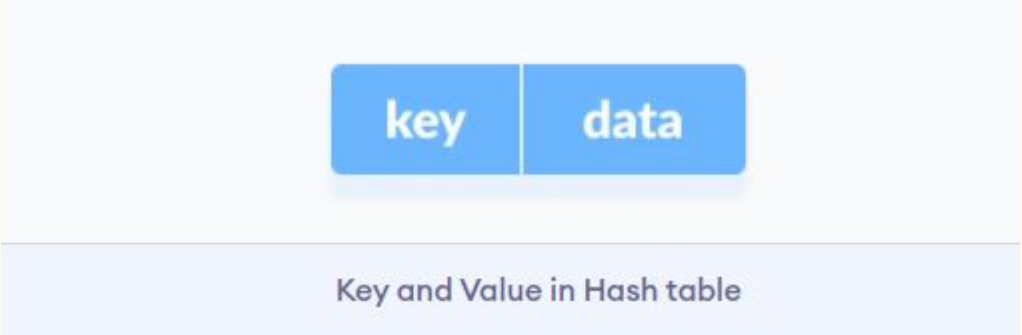
Lab 3
Implementation and application of Hashing

Agenda

- Recap
  - Map ADT
  - Hash functions
  - Hash table: collisions and handling
  - HashMap
- Usage of hashCode() and equals() Methods
- Live coding
- CodeForces Assignment

Map ADT

- Object that maps keys to values
- Cannot contain duplicate keys
- Each key can map to at most one value

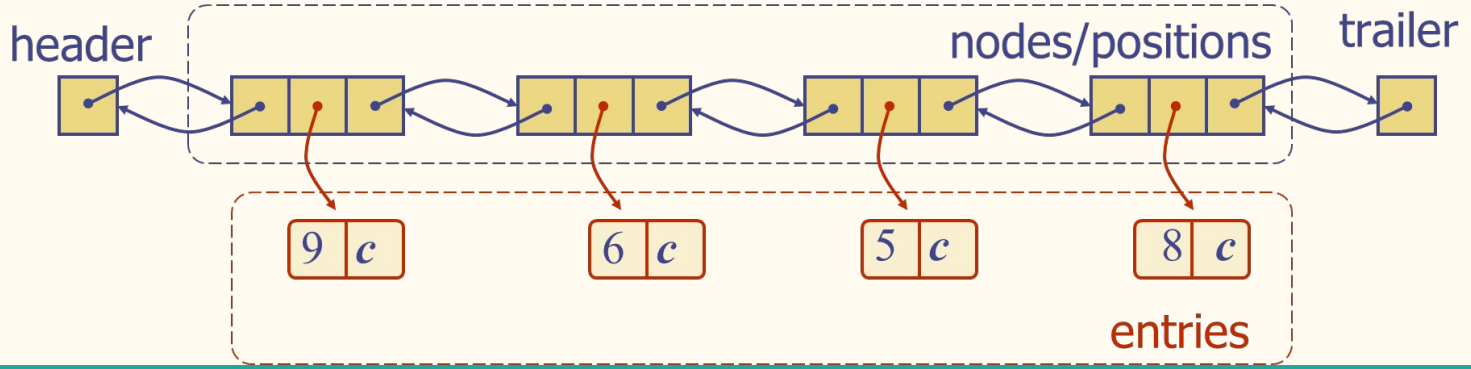


Map methods()

- get(k)
- put(k,v)
- remove(k)
- values()
- size()
- isEmpty()

Map implementations

1. Linked List based implementation
   - We can implement a map using an unsorted sequence
   - We store the items of the map in a list S (based on a doubly linked list), in arbitrary order

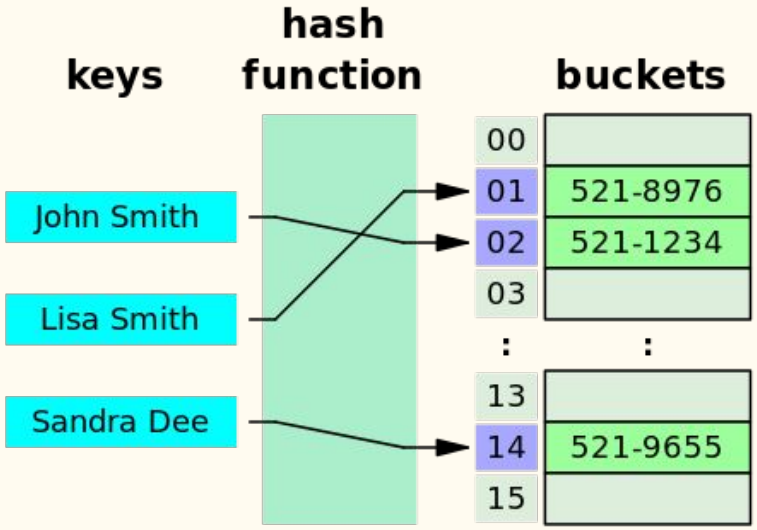


Performance of a List-Based Map

- Performance:
  - put takes O(1) time since we can insert the new item at the beginning or at the end of the sequence
  - get and remove take O(n) time since in the worst case (the item is not found) we traverse the entire sequence to look for an item with the given key

Map implementations

1. List based implementation
2. Array based implementation
   - We need mapping function
     - Maps input data to array index
     - One-way function
     - Hash function
       - Input : map key
       - Output : hash code
   - Hash map/table



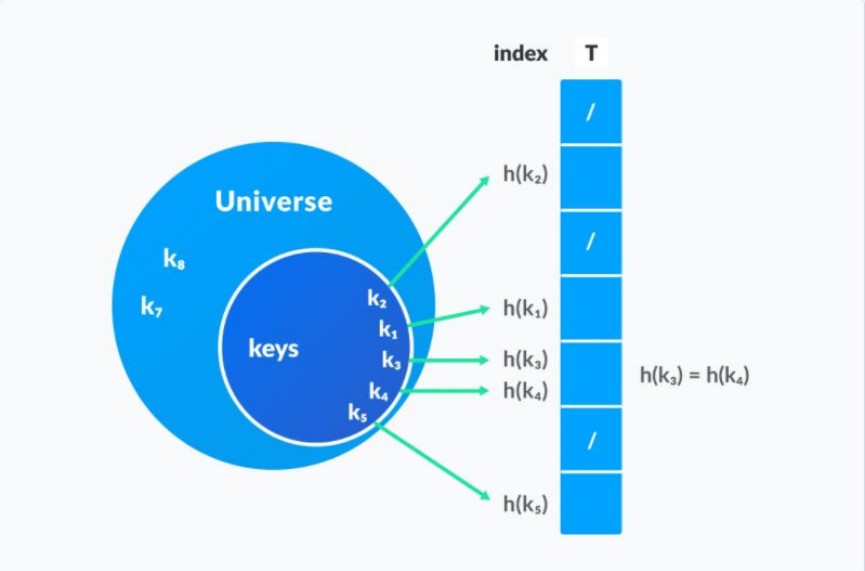
A small phone book as a hash table

Hash Table

- A ***hash table*** for a given key type consists of:
  - 1. Hash function h (a mathematical way of mapping an arbitrary key to an index in the array)
  - 2. Array (which is called table) of size N or m
- the goal is to store item (k, o) at index i = h(k), where k is the key, o is the data objectHash Function

Hash function

- In a hash table, a new index is processed using the keys. And, the element corresponding to that key is stored in the index. This process is called hashing.
- Let k be a key and h(x) be a hash function.
- Here, h(k) will give us a new index to store the element linked with k.



Hash table Representation

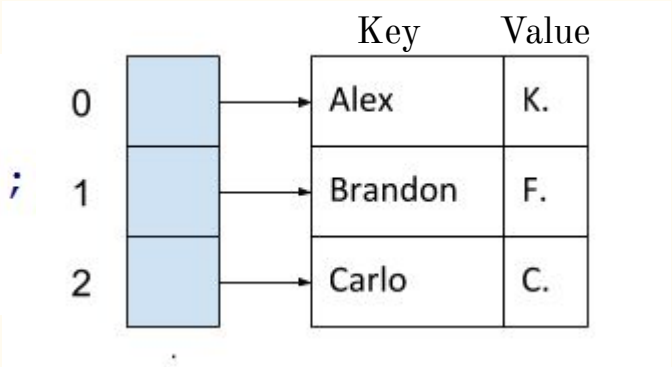
Hash function

- Allows mapping arbitrary data into fixed interval of values
- It is relatively safe to store hash values because these functions are one-way
- You cannot restore input data from output data, at least in a short time

Hash function: Simple example

- A hash function returns a specific integer for a string key

```
public class String {
    public int hashCode() {
        return (int)(this.charAt(0) - 'A');
    }
}
```



- What are the advantages of hashing for implementing Map ADT compared to list-based implementation?

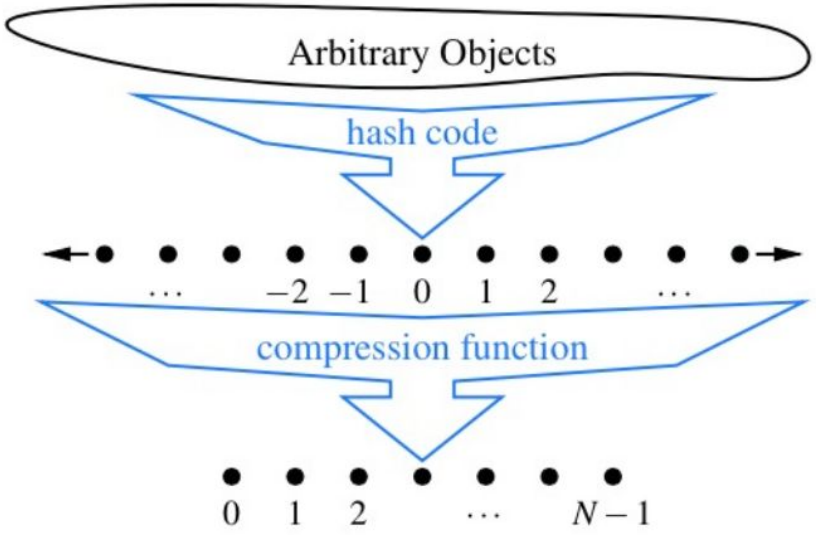
Hash function properties

- **Determinism**(For a given input value it must always generate the same hash value)
- **Uniformity** (Inputs as evenly as possible over its output range)
- **Defined range** (The output of a hash function have fixed size)
- **Data normalization**(Any two inputs that are considered equivalent must yield the same hash value. This can be accomplished by normalizing the input before hashing it, as by upper-casing all letters)
- **Continuity** (map all the values very close to one another)
- **Non-invertible** (For cryptographic hashing and a consequence of hashing being used to convert a large set into a smaller one)

Hash function applications

- Document Integrity (Checksum)
- Storing Passwords
- Generate Unique ID
- Pseudorandom Number Generation

Parts of a Hash Function



Hash codes

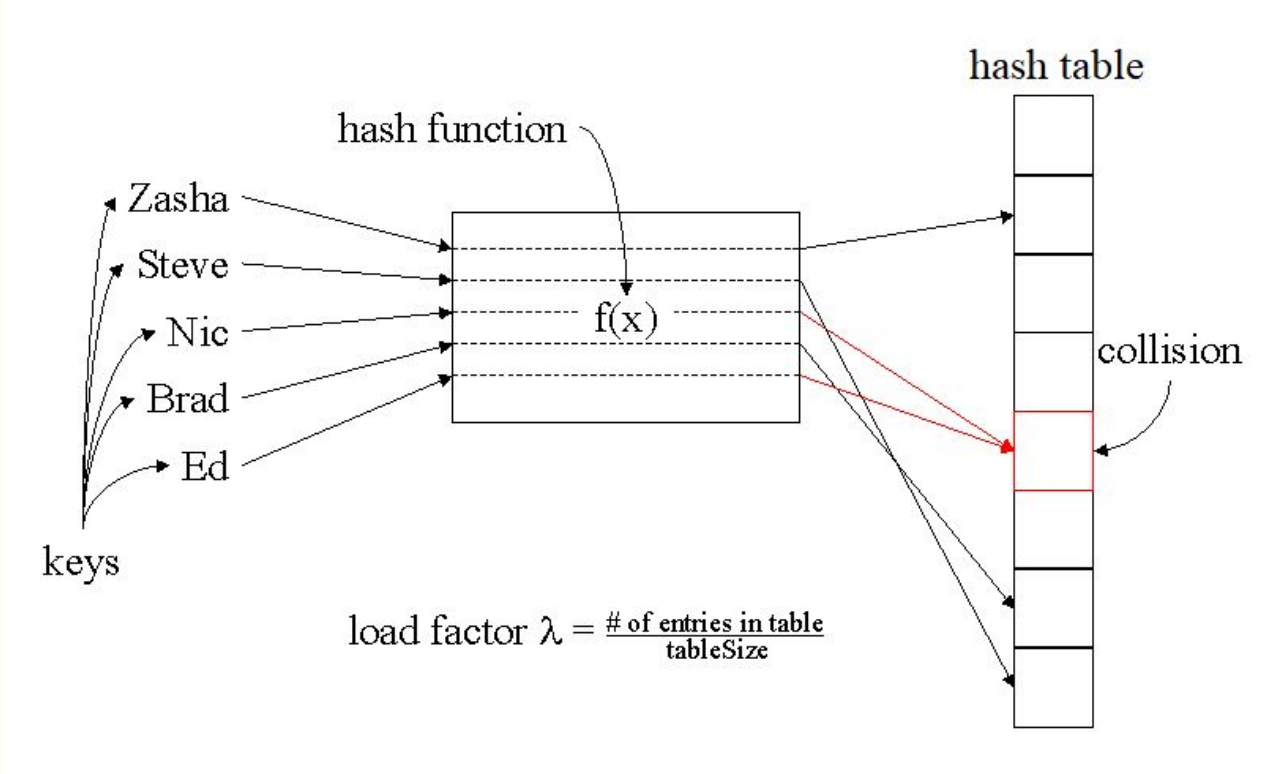
- Memory address (not repeatable)
- Integer cast (Suitable for keys of length less than or equal to the number of bits of the integer type)
- Component sum (fails to treat permutations differently)
- Polynomial accumulation

Compression functions

- **Division:**
  h2 (y) = y mod N where (N is the size of the array)
  we choose N to be prime to helps “spread out” the distribution of hashed values

- **Multiply, Add and Divide:**
  h2 (y) = [(ay + b) mod p] mod N

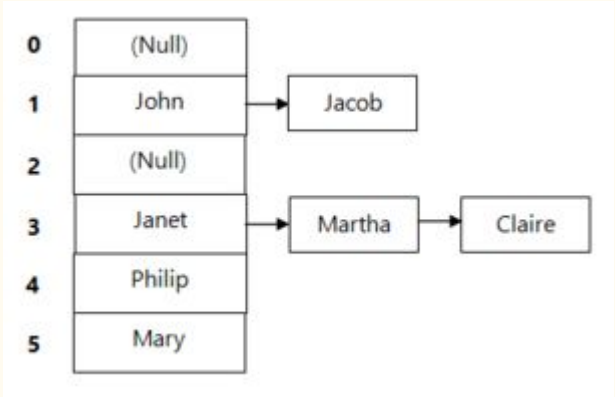
Hash table Collision



Hash table collisions handling

- Let n be the number of items inserted into a hash table of size m

- Open addressing (m much larger than n)
  - Linear probing
  - Double hashing
- Separate chaining (m much smaller than n)



Usage of hashCode() and equals() Methods

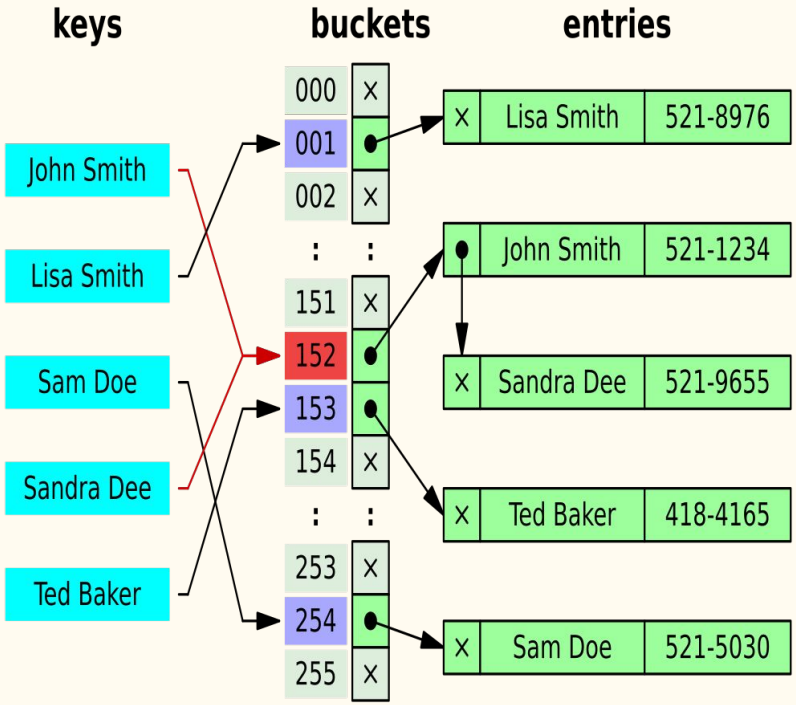
- equals(Object otherObject) – It is used to verify the equality of two objects.
- hashcode() – Returns a unique integer value for the object in runtime.

```
int hash = hash(k) //calculates the hashcode e.g. - 87673459
int index = hash & (n - 1)  //n is the capacity or size of HashMap. In a defualt implementat
ion it is 16.
```

- It is generally necessary to override the hashCode() method whenever equals() method is overridden to maintain the general contract for the hashCode() method, which states that equal objects must have equal hash codes.

Hash Map Implementation (live coding)

- Map Item/Entry consists of a key, a value and a pointer to next node.
- Keys are unique.
- Buckets are stored in a fixed array structure (fixed hash table).
- Each bucket is a linked list of Map entries.
  - Map size is the number of entries.
  - Map capacity is the number of buckets or array size.
- Use **separate chaining** method to handle the collisions.



A small phone book as a hash table

Hash Map Implementation (live coding)

```java
interface ILinkedHashMap<K,V> {
    int size();
    boolean isEmpty();
    void put(K key, V value);
    void remove(K key);
    V get(K key);
}
```

Java

```cpp
template <class K, class V>
class ILinkedHashMap{

public:
    virtual int size() = 0;
    virtual bool isEmpty() = 0;
    virtual void put(K key, V value) = 0;
    virtual void remove(K key) = 0;
    virtual V get(K key) = 0;
};
```

C++

https://codeforces.com/group/M5kRwzPJlU/contest/366311

- You should have your own implementation of Map ADS

See You next week!