#ASLI ENGINEERING

Implementing Hash Sets

SWIPE

BY

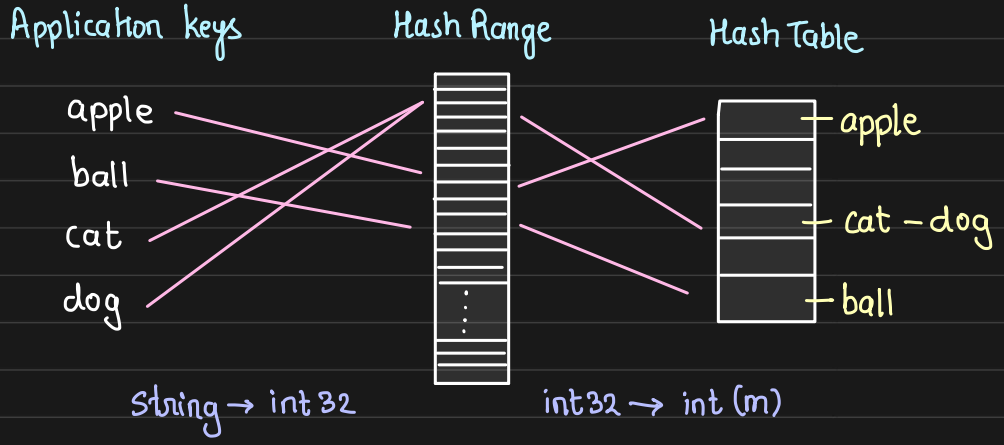
ARPIT BHAYANI

Implementing Hash Sets

Hash Set is a data structure that implements set over a hash table

{ apple, ball, cat, dog }

The idea is to allow only unique values in the data structure. but how do we implement it with Hash Tables



Different application keys map to same Hash Key

↓

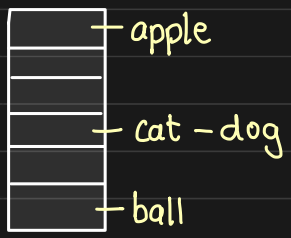
eg: cat and dog

From same hash key, they would be placed in two different slots of the hash table [collision handling]

Hence, while looking up we have to compare actual application key, and cannot just rely on hash key

Hence, we need to store

- the application keys in the Hash Table
- the hash keys along with the key
  ↳ avoid computing $fn(k) = h$




```
struct Key {
    void * key;      <-- hold key of any type
    int hash_key;    <-- hash stored to avoid re-computation
}
```

Implementation detail: if we support generic key (void *) how would we compare two such keys?

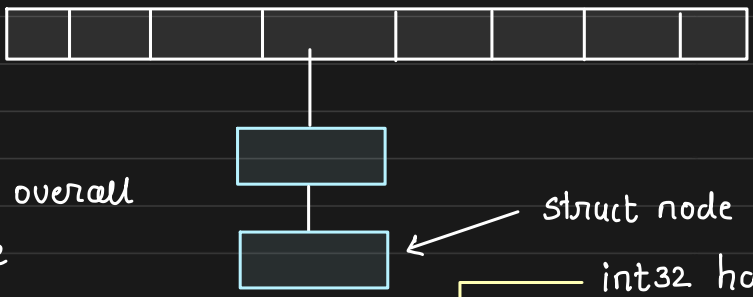
hence, we would need a custom comparator for the type

Implementation detail: when we delek a key from the hash table it may be our responsibility to clean them up

[manual memory management]

Hence, as part of robust implementation we would need comparator function and destructor function

Implementing Hash Set with Hash Table with Chaining



Hash Set overall will have

1. array
2. # size
3. # keys
4. comparator function
5. destructor function

```
struct node {
    int32 hash_key;
    void * key;
    struct node * next;
}
```

hash_key: hash of the key to avoid re-computation

key: application key

Implementation Detail

1. To have fast insert, prepend keys at head of the list
2. To have just unique keys, we have to check and insert

   a little slow on time but good on space

Implementing Hash set with Hash Table with Open Addressing

```
struct slot {
    bool is_empty;
    bool is_deleted;
    int32 hash_key;
    void * key;
}
```

marks if slot is empty

marks if key is soft deleted

application key

hash of the key to avoid re-computation

Hash Set overall will have

1. array
2. # keys
3. # used slots
4. comparator function
5. destructor function

Implementation Detail

During insert, lookup and delete when we find the matching hash key we need to explicitly compare the keys to check its existence.

Implementation Detail

Destructor should be invoked only when we are hard deleting the key