

## Direct Access Sets?

Have been assuming need to search for a key

- In an array sorted by key
- Better: in a tree sorted by key

Can data just be indexed by key?

<https://tutorcs.com>

### Questions

How could such indexing work?

- Want to use any type as a key

Assuming such indexing, how long would put and get take in a set containing  $N$  objects?

WeChat: cstutorcs

# Indexed Sets

Hash Tables index data (indirectly) by key

- A hash table  $T$  is (like) an array with  $m$  slots
- The key is converted to an integer index by a hash function  $h$
- So, an object with key  $k$  is stored at  $T[h(k)]$

<https://tutorcs.com>



WeChat: cstutorcs

The time taken by  $h$  depends only on  $k$

- New object added into  $N$  object set in  $\Theta(1)$  time (theoretical only!)

## Numerical Encoding

# Assignment Project Exam Help

Map any key object to a natural number

- Requirement: equal keys have same result
- Requirement: unequal keys have different result

### Exercise

Design a function to map every ASCII string to a different natural number

## Encoding Function

# Assignment Project Exam Help

One Possibility

The formula

$$k = s[0] + s[1] * 128 + s[2] * 128^2 + \dots$$

converts every ASCII string  $s$  to a different natural number.

- Treat each character as a digit
- Same principle can be applied (recursively) to any type

### Question

What is the problem with this as a practical solution?

# Collisions

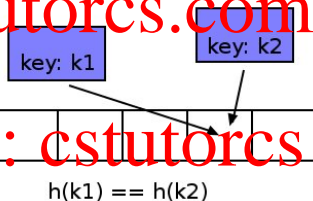
Assignment Project Exam Help

Impactical to store every key at a different index

- Very space inefficient, even if it's possible
- Result: collisions

<https://tutorcs.com>

WeChat: cstutorcs



Will need a way to resolve collisions (store both objects)

## A Hash Function Part 2

Map the numerical code  $k$  from Step 1 to a position in the table

Step 2

If the table has size  $m$ :

$$h(k) = k \bmod m$$

New requirement:

minimise collisions

- spread the keys as evenly as possible

Question

What happens to the ASCII string keys if  $m = 128$ ?

- All keys starting a.. hash to same slot
- If all keys start a... only one slot used
- Using a prime radix for  $k$  limits the problem

# Uniform Hashing

- Lots of ways to hash: universal, fingerprint, cryptographic, ...
- Best result is data dependent
- More **uniform**, often slower

## Definition (Simple Uniform Hashing Assumption)

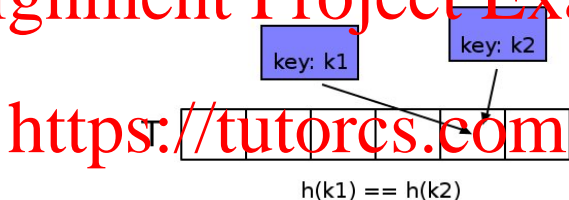
Given a hash table  $T$  with  $m$  slots, using hash function  $h$ , the *simple uniform hashing assumption* (SUHA) states that each new key  $k$  is equally likely to hash into any of the  $m$  slots. So, the probability that  $h(k) = i$ , for every slot  $1 \leq i \leq m$  is  $1/m$ .

- SUHA is an assumption about both  $h$  and input data
- Allows analysis to ignore details of both

## Hash Table Memory

Recall: need a way to **resolve** collisions (store both objects)

# Assignment Project Exam Help



### Exercise

## WeChat: cstutorcs

Design a way to resolve collisions

- Table has to store both objects somewhere
- What is the worst case time to add a new object?



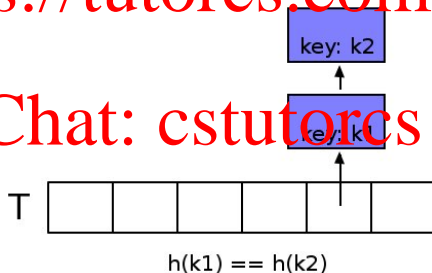
# Chaining

With collision resolution by Chaining

- All objects whose key hashes to  $i$  ( $k_i$ ) are placed into a linked list
- The table contains a pointer to the list
- So,  $T[i]$  contains a list of objects  $x$  where  $h(x.key) = i$

<https://tutorcs.com>

WeChat: cstutorcs



# Performance of Chaining

Add object  $x$  to table  $T$ :

Insert at head of list at  $T[h(x.key)]$

- takes  $\Theta(1)$  time

Search for an object with key  $k$

Search list at  $T[h(k)]$  for an object where  $x.key == k$

In a table containing  $N$  values

- Worst case is  $N$  elements in one chain:  $O(N)$  search
- Under SUHA, **expected time** is  $O(N/m)$
- $N/m$  is called the **load factor**

## Expected Time To Search

The **expected** time for an **unsuccessful** search for key  $k$ , in a hash table with  $m$  slots, containing  $N$  objects, assuming simple uniform hashing:

- By SUHA, expected length of each chain is  $N/m$
- $k$  equally likely to hash to all  $m$  positions
- Probability of searching chain at  $T[i]$  is  $1/m$

Expected number of comparisons is

$$\sum_{i=1}^m \frac{N}{m} \times \frac{1}{m} = \frac{N}{m}$$

If  $N$  is proportional to  $m$ , expected running time for Search is  $\Theta(1)$

- The design of the table needs to ensure  $N/m$  is  $\Theta(1)$
- Successful search reasoning is similar:  $O(N/m)$

# Probing

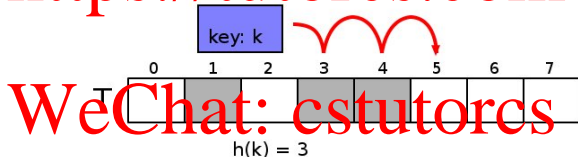
In an **open address** hash table objects are stored directly in the table

• We use **probing** to resolve collisions

• To insert an object we **probe** the table until we find a space

• The hash function generates a sequence  $\langle h(k, 0), \dots, h(k, m - 1) \rangle$

<https://tutorcs.com>



The simplest form (above) is **linear probing**

• Consecutive slots are probed, beginning with  $h(k)$ , up to  $h(k) - 1$

# Performance of Probing

## Definition (Uniform Hashing)

Given a hash table with  $m$  slots, a hash function produces uniform hashing if, for an unknown key  $k$ , the probability that the probe sequence of  $k$  is  $p$ , where  $p$  is a permutation of  $\langle 0, \dots, m-1 \rangle$  is the same for all such  $p$ .

- Uniform hashing first implies that every permutation is possible
- Linear probing does not produce uniform hashing

Assuming uniform hashing, the expected number of keys compared when inserting an object depends on the load factor  $N/m$

- Each probe is to a random slot, with probability  $N/m$  it is occupied

If  $N$  is proportional to  $m$ , expected time for insert (and search) is  $\Theta(1)$

## Limitations

# Assignment Project Exam Help

Hash tables do not support operations such as:

- In order iteration
- Next key / object
- Minimum key
- Maximum key

since objects are stored, by design, in random order.

<https://tutorcs.com>  
WeChat: cstutorcs