Hash-Tables

Introduction

Dictionary

Dictionary

- stores key-value pairs

	Find(k)	Insert(k, v)	Delete(k)
List	$\mathcal{O}(n)$	$\mathcal{O}(1)$	$\mathcal{O}(n)$
Sorted Array	$\mathcal{O}(\log n)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$
Balanced BST	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$

Dictionary implementations we know.

Goal

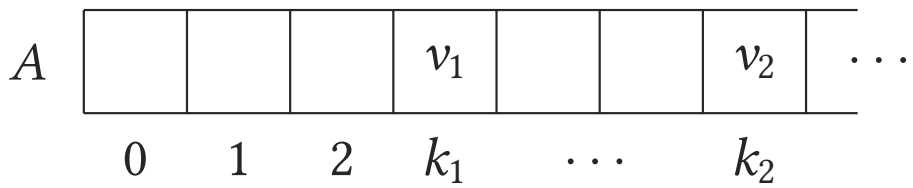
- All operations in $\mathcal{O}(1)$ time.

Hash Tables

Naive Approach

Direct Access Table

- One large array A.
- For each key value pair (k, v): $A[k] = v$.



Problems

1. Keys must be a non-negative integers.
2. Very large key range. Thus, huge amount of memory needed.

Problem 1: Getting Integer Keys

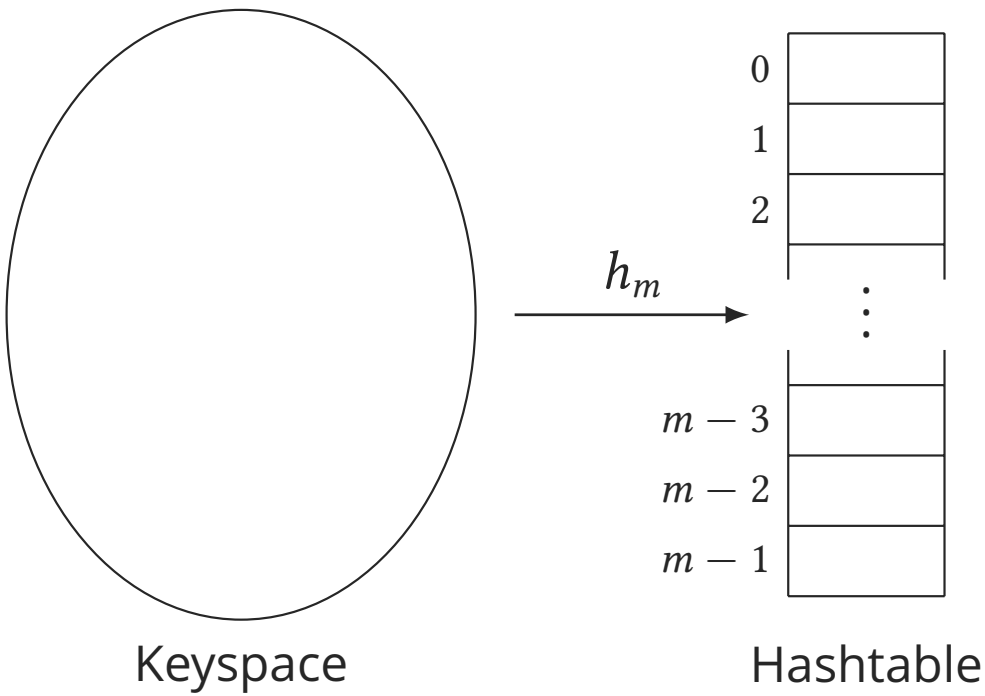
Prehashing

- Take a key k and map it on a non-negative integer k'.
- Easy in theory, because all finite data can be represented as integer.
- k' should not change when object changes.
- In ideal case: k' is unique for the object.

Problem 2: Getting Small Keys

Hashing

- U is *huge* universe of all possible (non-neg. int.) keys.
- Hash function $h_m \colon U \to \{0, 1, \ldots, m-1\}$ reduces keys to small range of integers. Ideally, $m \in \Theta(n)$ with a small constant $c \geq 1$.
- Computing h_m should require $\mathcal{O}(1)$ time with small constant.



Collisions

Problem with Hashing

- Because $|U| \gg m$, in some cases: $h_m(k_1) = h_m(k_2)$.
  This is called *collision*.

Questions

1. How to design h such that number of collisions is low?
2. How do we handle collisions?

For 1.

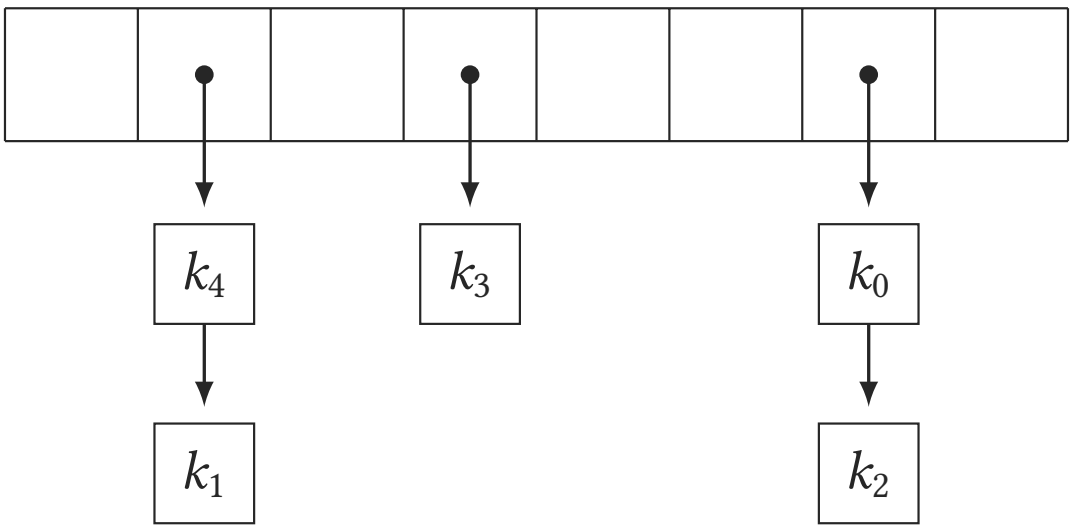
- For this class, assume h_m is given and has uniform distribution of hash values.
- Thus, expected size of sets with same hash value is $\frac{n}{m}$.
- $\alpha = \frac{n}{m}$ is called *load factor* of the table.

Chaining

Chaining

Idea

- Use a list (or other data structure) of colliding items in each slot of the table.



Find Operation

1. Use hash to determine slot in table.
2. Search in list for item.

Open Addressing

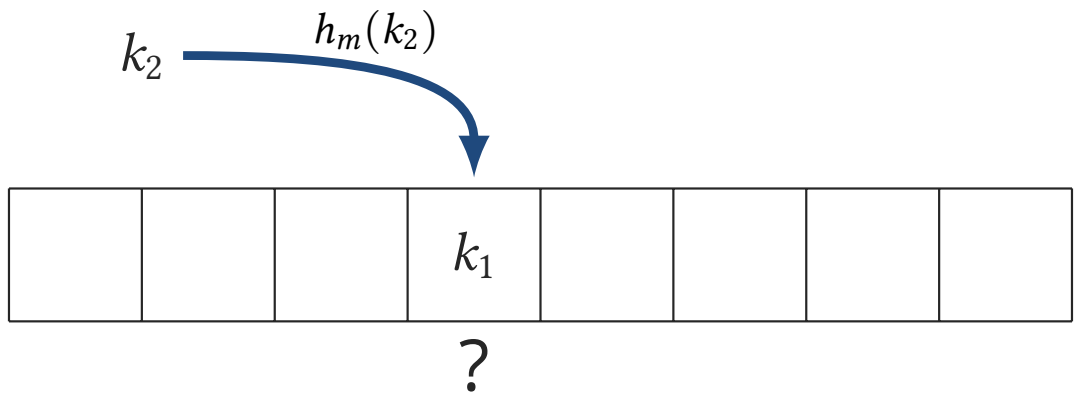
Open Addressing

Idea

- Store all items in the array (i. e., one item per slot).

Problem

- How do we handle collisions?



Solution: *Probing*

- If slot is already used, compute new hash value. Repeat until free slot was found

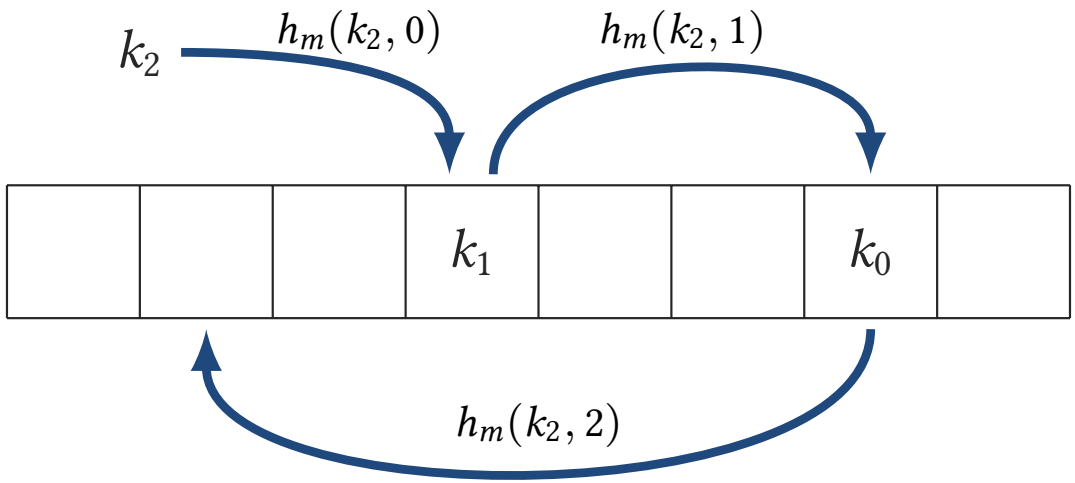
Probing

Hash function h specifies order of slots for a key k.

$$h_m \colon U \times \{0, 1, \ldots, m-1\} \to \{0, 1, \ldots, m-1\}$$

Resulting order: $\sigma(k) = \langle h_m(k, 0), h_m(k, 1), \ldots, h_m(k, m-1) \rangle$

In ideal case, $\sigma(k)$ is permutation of $\{0, 1, \ldots, m-1\}$.

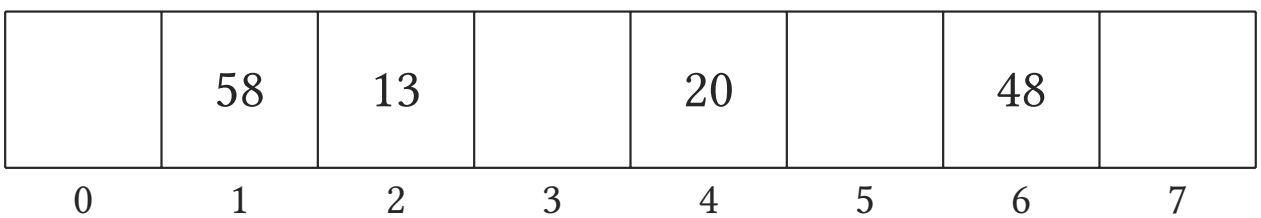


Example

Let $h_m(49, 0) = 4$, $h_m(49, 1) = 6$, $h_m(49, 2) = 1$, and $h_m(49, 3) = 5$.

Perform

1. Insert(49)
2. Delete(58)
3. Find(49)

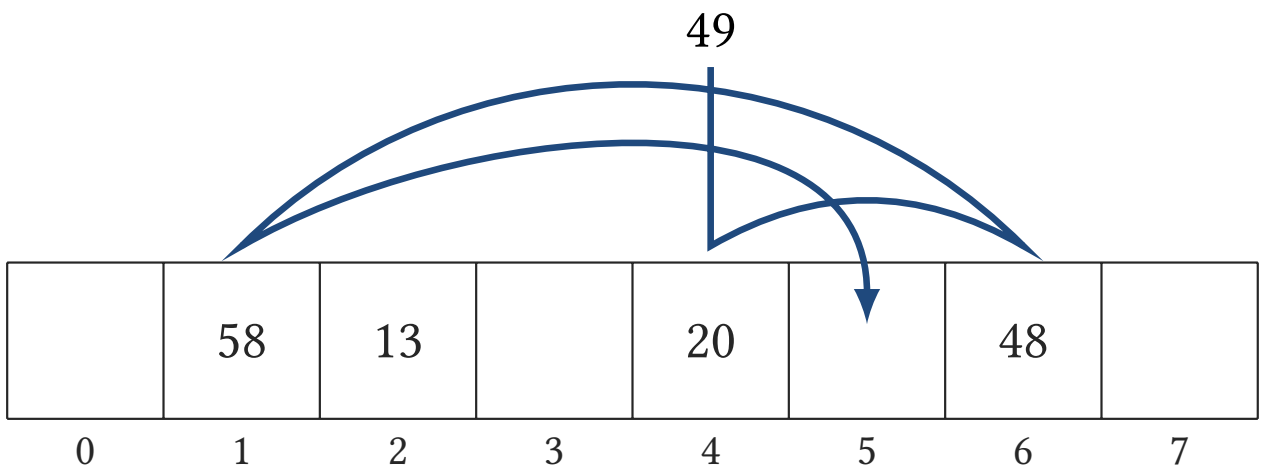


Example

Let $h_m(49, 0) = 4$, $h_m(49, 1) = 6$, $h_m(49, 2) = 1$, and $h_m(49, 3) = 5$.

Perform

1. Insert(49)
2. Delete(58)
3. Find(49)

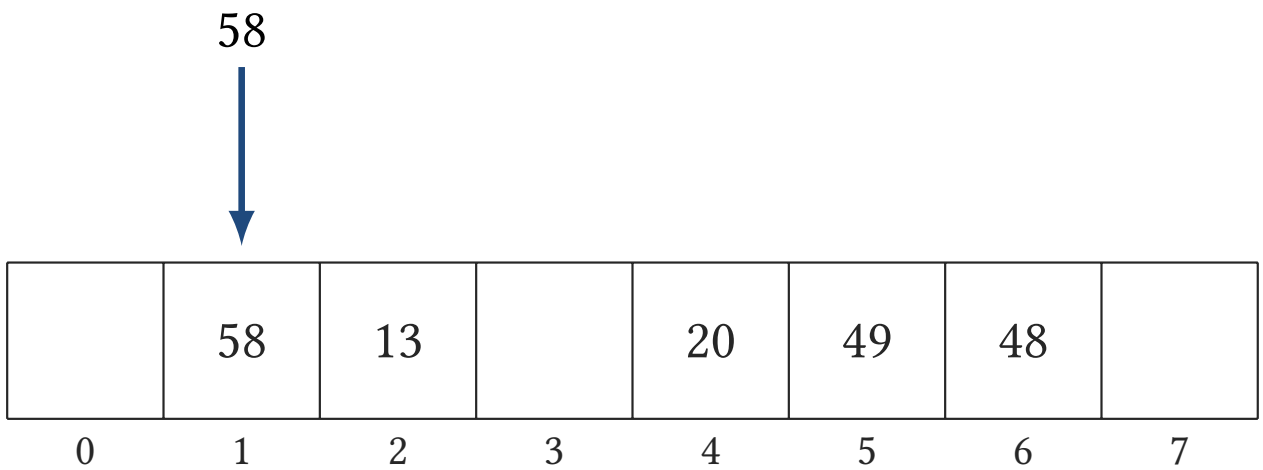


Example

Let $h_m(49, 0) = 4$, $h_m(49, 1) = 6$, $h_m(49, 2) = 1$, and $h_m(49, 3) = 5$.

Perform

1. Insert(49)
2. Delete(58)
3. Find(49)

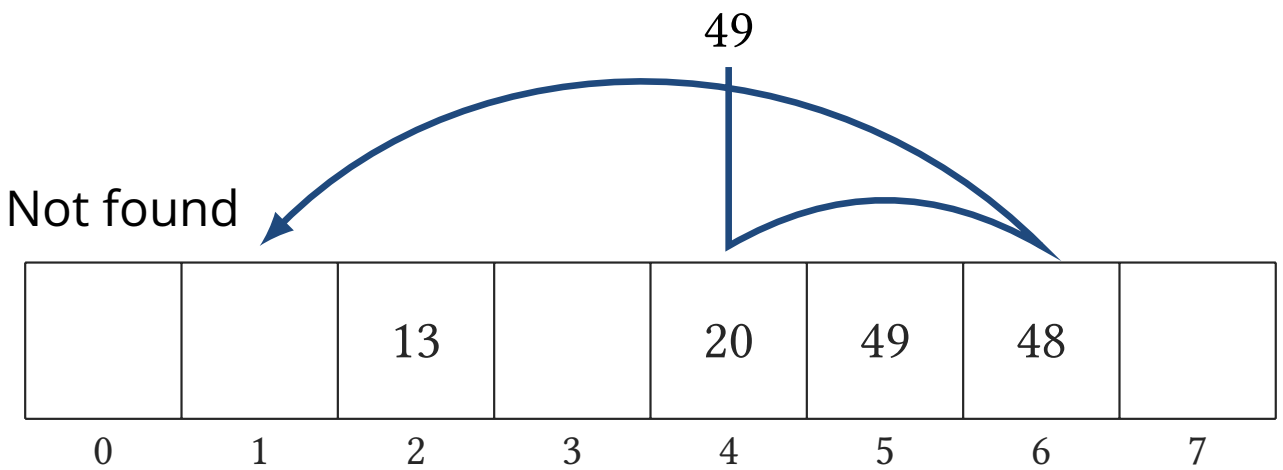


Example

Let $h_m(49, 0) = 4$, $h_m(49, 1) = 6$, $h_m(49, 2) = 1$, and $h_m(49, 3) = 5$.

Perform

1. Insert(49)
2. Delete(58)
3. Find(49)



Open Addressing – Delete

Delete

- Simple deletion can lead to failure of Insert/Find.
- Flag slot with deleted item as ‘Deleted’.
- Use flag for Insert/Find.

Question

- What if k is already in the table, but Insert encounters a field flagged as ‘Deleted’?

Probing Strategies – Linear Probing

Idea

- Slightly increase index

  $h_m(k, i) = (h_m(k) + i) \bmod m$

Good

- Gives a permutation (i. e., no index checked twice)

Problem

- *Clustering*: consecutive groups of occupied slots.
- For $0.01 < \alpha < 0.99$, there are clusters of size $\Theta(\log n)$ even if h_m is perfect.

Probing Strategies – Double Hashing

$$h(k, i) = (h_1(k) + i \cdot h_2(k)) \bmod m$$

h_1 and h_2 should be independent, i. e., probability for $h_1(x) = h_1(y)$ and $h_2(x) = h_2(y)$ is $\frac{1}{m^2}$.

Hits all slots if $h_2(k)$ and m have no common divisor (e. g. $m = 2^r$ and $h_2(k)$ is always odd).

Assuming ideal hash function h, the expected cost for an operation is $\leq \frac{1}{1-\alpha}$.

Using Hash Tables

Dictionaries

Dictionary

- Stores key-value pairs
- The *key* is an identifier.
- The *value* is an information associated with the key.

Operations

- ***Insert.*** Inserts or overrides a given key-value pair into the dictionary.
- ***Find.*** Returns the value associated with the given key.
  (often implemented as two function: *Contains* and *GetValue*)
- ***Delete.*** Deletes the key-value pair with the key.

Example: Counting

You are given an array A of integers. Determine the most frequent number.

Example: Counting

You are given an array A of integers. Determine the most frequent number.

Idea

- Numbers in A are keys.
- Value in dictionary is counter for associated key.
- Key with largest associated value is answer.

Example: Counting

Input: An array A of integers.

Output: The most frequent number in A.

1 Create empty dictionary D.

2 **For** $i = 0$ **To** $|A| - 1$

3 **If** D contains key $A[i]$ **Then**

4 Insert key-value pair $(A[i], 0)$.

5 Increase the value of key $A[i]$ by 1.

6 Let k be the key in D with the largest associated value.

7 **Return** k

Exercises

You are given an array A of integers. Find the last (i. e., with the highest index) non-repeating integer in A in linear time.

You are given an array A of integers and an integer k. Determine whether there are two distinct indices i and j such that $A[i] = A[j]$ and $|i - j| \leq k$.

Given two strings S and T (only lowercase letters). T is generated by shuffling S and then adding one more letter at a random position. Determine the letter that was added into T.

You are given an array A of integers. Determine the longest connected subsequence without repeating characters.