8. Hashing

Hashing

a[0]	
a[1]	
a[2]	j
a[3]	
a[4]	i
a[5]	
a[6]	
a[7]	sum
a[8]	
a[9]	

Symbol table

Hashing

- A hash table is an array of fixed size **m**, used to store input keys.
- Hashing is the mapping of the input keys to the indices of hash table using a hash function.

  e.g. **a[h(k)]** = **k**, where **a** is the hash table,
  k is an input key and
  h(k) is a hash function
- The goal of hashing is to perform the ff. operations:

  search, **insert and delete**

  in constant time on the average.

Hashing - Considerations

- Choosing a hash function
- Resolving a collision
  - Two distinct keys map/hash to the same table index
- Deciding the hash table size, **m**

Choosing a good hash function

- Simple Uniform Hashing (SUH)
  - Each key is equally likely to hash to any of the **m** slots.
- Example:
  - if keys are known to be random real numbers, uniformly distributed in the range [0,1)

    $hash(k) = \lfloor km \rfloor$

Choosing a good hash function

- If keys are character strings, they can be transformed to integers expressed in a suitable radix notation.
- Example:
  - string “pt” = (112, 116)
  - expressed as a radix-128 integer, (112*128)+116 = 14452

Hash functions

- Three popular methods (keys expressed as natural numbers)
  - Division method
    - hash(k) = k % m
  - Multiplication method
    - hash(k) = $\lfloor m(kA \bmod 1) \rfloor$, where $0<A<1$
  - Universal hashing
    - Using several hash functions

Three Implementations

- Direct addressing
  - Key **k** maps/hashes to table index **k**
- Open hashing(separate chaining)
  - Keys are not actually stored in the table, but in a separate data structure
- Closed hashing(open addressing)
  - Keys are stored directly on the table

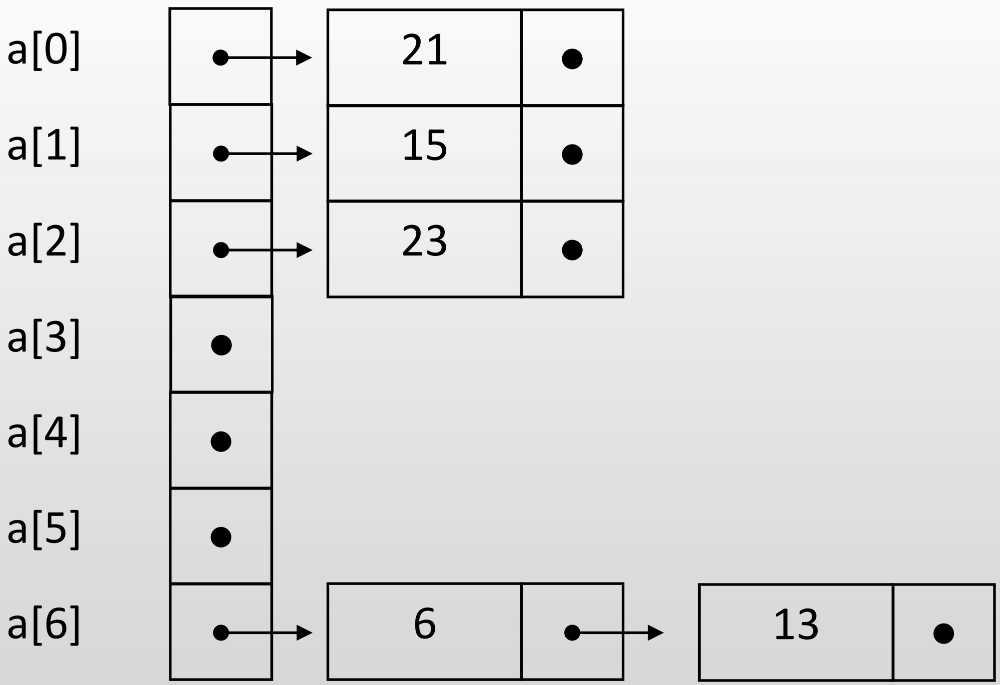
Direct Addressing

- Insert 6, 15, 23, 21, and 13 to a hash table of size **m** = 7

a[0]	
a[...]	
a[6]	6
a[13]	13
a[15]	15
a[21]	21
a[23]	23

Open Hashing

Insert 6, 15, 23, 21, and 13 to a hash table of size **m** = 7



Closed Hashing/Open Addressing

- Table stores the keys
- Collision resolution strategy via probing (alternate cells/slots to try in succession):
  - Linear probing
  - Quadratic probing
  - Double hashing

Formula

- Actual hash function (division method)
  hash(k) = k % m
- Probing formula in case of collision
  h(k, j) = (hash(k) + $\boxed{\textbf{f(j)}}$) % m
  – for j = 1, 2, 3, ...

Linear probing
Quadratic probing
Double hashing

Linear Probing

- Use of a linear function

  $f(j) = j$

- disadvantage:

  Primary clustering

Closed Hashing and Linear Probing

$h(k) = (k\%7 + i) \% 7$, i = number of collision(s)

primary clustering

a[0]	21
a[1]	15
a[2]	23
a[3]	13
a[4]	
a[5]	
a[6]	6

inserting 13:

$(13\%7 + 0) \%7 = 6$

$(6 + 1)\%7 = 0$

$(6 + 2)\%7 = 1$

$(6 + 3)\%7 = 2$

$(6 + 4)\%7 = 3$

Quadratic Probing

- Use of a quadratic function

  f(j) = (c1 * j) + (c2 * j^2),

  - where c1 and c2 are given constants, e.g. 2 & 3 respectively

  f(j) = j^2
- disadvantage:

  Secondary clustering

Quadratic Probing

$h(k) = (k\%7 + i^2)\ \%\ m$

secondary clustering

a[0]	21
a[1]	15
a[2]	23
a[3]	13
a[4]	
a[5]	
a[6]	6

inserting 13:

$(13\%7 + 0^2)\ \%7 = 6$

$(6 + 1^2)\%7 = 0$

$(6 + 2^2)\%7 = 3$

Double hashing

- Use of a second hash function

  f(j) = j * **hash_2(k)**,

  e.g.

  hash_2(k) = 1 + (k % m)

  hash_2(k) = k % (m – 1)

  hash_2(k) = R – (k % R)

Double Hashing

h(k) = (k%7 + i*h2(k)) % m, where h2(k) is another hash function e.g. h2(k) = 5 – k%5

a[0]	21
a[1]	15
a[2]	23
a[3]	13
a[4]	
a[5]	
a[6]	6

inserting 13:

(13%7 + 0) %7 = 6

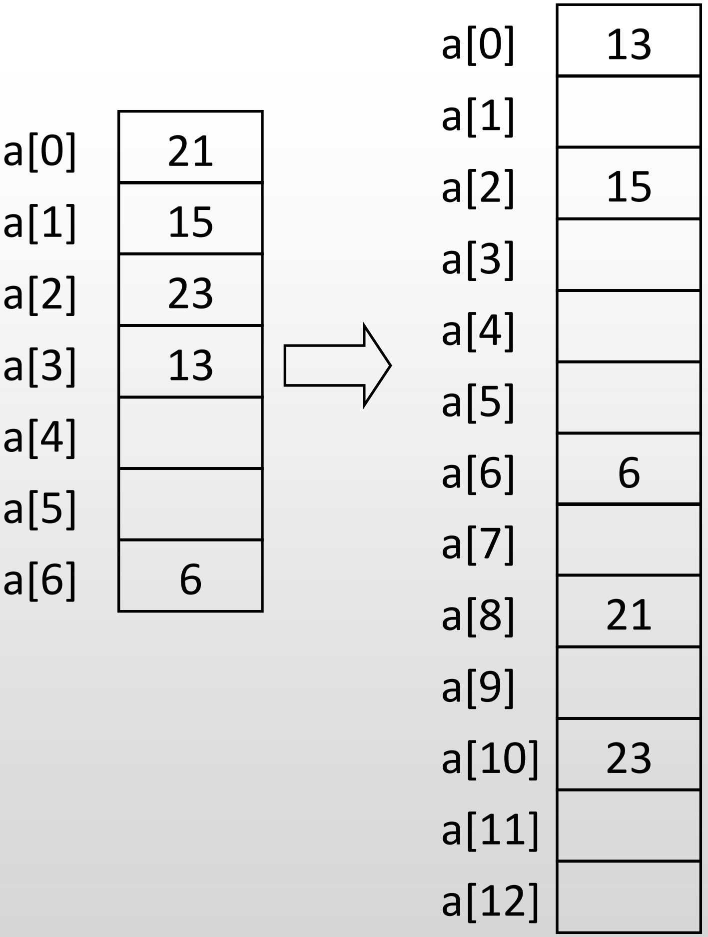
(6 + 1*(5-13%5))%7

= (6 + 1*(2))%7 = 1

(6 + 2*(2))%7 = 3

Rehashing

Increase the size of the hash table, e.g. from $m = 7$ to 13, then rehash all entries using $m = 13$.



Quiz

- Insert 14, 6, 16, 22, and 27 to a hash table of size **m** = 7 using h(k) = (k%m + i*h2(k)) % m, where h2(k) = k%(m – 1)

a[0]	
a[1]	
a[2]	
a[3]	
a[4]	
a[5]	
a[6]