Hashing

- Dictionaries
- Chained Hashing
- Linear Probing
- Hash Functions

Philip Bille

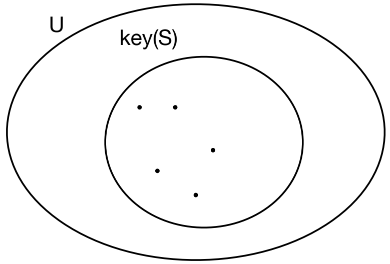
Hashing

- Dictionaries
- Chained Hashing
- Linear Probing
- Hash Functions

Dictionaries

- Dictionaries. Maintain dynamic set S of elements supporting the following operations. Each element x has a key x.key from a universe U and satellite data x.data.
  - SEARCH(k): determine if element with key k exists. If so, return it.
  - INSERT(x): add x to S (we assume x is not already in S)
  - DELETE(x): remove x from S.

- U = {0,..,99}
- key(S) = {1, 13, 16, 41, 54, 66, 96}



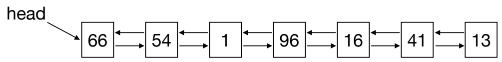
Dictionaries

- Applications.
  - Basic data structures for representing a set.
  - Used in numerous algorithms and data structures.

- Challenge. How can we solve problem with current techniques?

Dictionaries

- Solution 1: linked-list. Maintain S as a linked list.



- Search(k): linear search for key k.
- Insert(x): insert x in the front of the list.
- Delete(x): remove x from list.

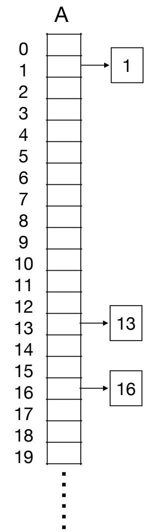
- Time.
  - Search in O(n) time.
  - Insert and Delete in O(1) tine.
- Space.
  - O(n).

Dictionaries

- Solution 2: direct addressing.
  - Maintain S in array A of size |U|.
  - Store element x at A[x.key].

- Search(k): return A[x.key].
- Insert(x): Set A[x.key] = x.
- Delete(x): Set A[x.key] = null.

- Time.
  - Search, Insert and Delete in O(1) time.
- Space.
  - O(|U|)



Dictionaries

Data structure	Search	Insert	Delete	space
linked list	O(n)	O(1)	O(1)	O(n)
direct addressing	O(1)	O(1)	O(1)	O(|U|)

- Challenge. Can we do significantly better?

Hashing

- Dictionaries
- Chained Hashing
- Linear Probing
- Hash Functions

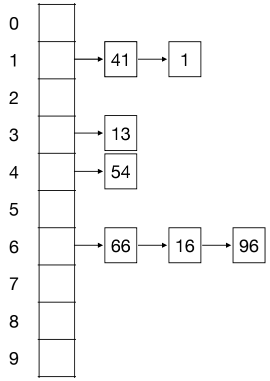
Chained Hashing

- Idea. Find a hash function h : U → {0, ... , m-1}, where m = Θ(n). Hash function should spread keys from S approximately evenly over {0, ..., m-1}.

- Chained hashing.
  - Maintain array A[0..m-1] of linked lists.
  - Store element x in linked list at A[h(x.key)].

- Collision.
  - x and y collides if h(x.key) = h(y.key).

- SEARCH(k): linear search in A[h(k)] for key k.
- INSERT(x): insert x in front of list A[h(x.key)].
- DELETE(x): remove x from list A[h(x.key)].



U = {0,..,99}

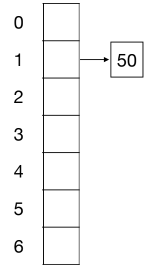
key(S) = {1, 13, 16, 41, 54, 66, 96}

m = 10

h(k) = k mod 10

---

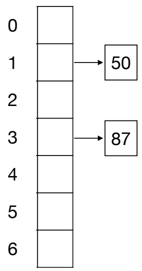
k = 50



h(k) = k mod 7

---

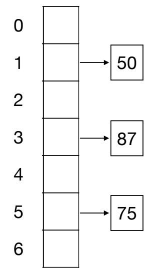
k = 87



h(k) = k mod 7

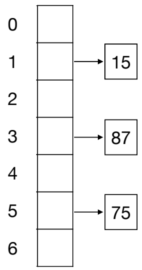
---

k = 75



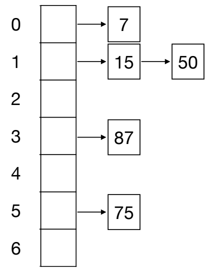
h(k) = k mod 7

k = 15



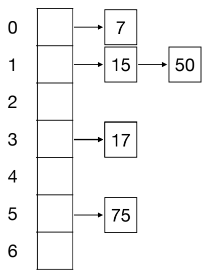
h(k) = k mod 7

k = 7



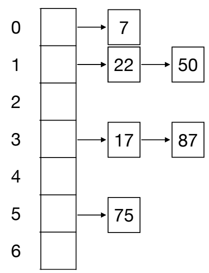
h(k) = k mod 7

k = 17



h(k) = k mod 7

k = 22



h(k) = k mod 7

Chained Hashing

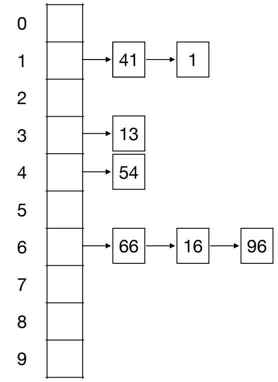
- SEARCH(k): linear search in A[h(k)] for key k.
- INSERT(x): insert x in front of list A[h(x.key)].
- DELETE(x): remove x from list A[h(x.key)].

- Exercise. Insert sequence of keys K = 5, 28, 19, 15, 20, 33, 12, 17, 10 in an initially empty hash table of size 9 using chained hashing with hash function h(k) = k mod 9.

Chained Hashing

- SEARCH(k): linear search in A[h(k)] for key k.
- INSERT(x): insert x in front of list A[h(x.key)].
- DELETE(x): remove x from list A[h(x.key)].

- Time.
  - SEARCH in O(length of list) time.
  - INSERT and DELETE in O(1) time.
  - Length of list depends on hash function.
- Space.
  - O(m + n) = O(n).



U = {0,..,99}

key(S) = {1, 13, 16, 41, 54, 66, 96}

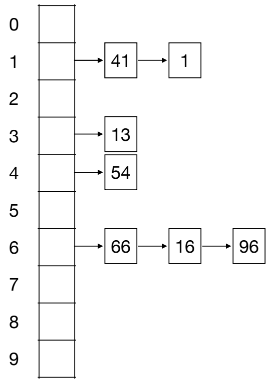
m = 10

h(k) = k mod 10

Chained Hashing

- Def. Load factor α = average length of lists = n/m = O(1)

- Simple uniform hashing. Assume that every key is mapped uniformly at random to {0, .., m-1}.
  - Expected length of list = α.
  - ⇒ expected time for SEARCH is O(1).
- Time (assuming simple uniform hashing).
  - SEARCH in O(1) expected time.
  - INSERT and DELETE in O(1) time.



U = {0,..,99}

key(S) = {1, 13, 16, 41, 54, 66, 96}

m = 10

h(k) = k mod 10

Dictionaries

Data structure	SEARCH	INSERT	DELETE	space
linked list	O(n)	O(1)	O(1)	O(n)
direct addressing	O(1)	O(1)	O(1)	O(|U|)
chained hashing	O(1)†	O(1)	O(1)	O(n)

† = expected time assuming simple uniform hashing

Hashing

- Dictionaries
- Chained Hashing
- **Linear Probing**
- Hash Functions

Linear Probing

- Linear probing.
  - Maintain S in array A of size m.
  - Element x stored in A[h(x.key)] or in cluster to the right of A[h(x.key)].
  - Cluster = consecutive (cyclic) sequence of non-empty entries.

	41	1	11	13	54			98	
0	1	2	3	4	5	6	7	8	9

h(k) = k mod 10

- SEARCH(k): linear search from A[k] in cluster to the right of A[k].
- INSERT(x): insert x on A[h(x.key)]. If non-empty, insert on next empty entry to the right of x (cyclically).
- DELETE(x): remove x from A[h(x.key)]. Re-insert all elements to the right of x in the cluster.

5 1 27 32 54 11 19

0	1	2	3	4	5	6	7	8	9	10

h(k) = k mod 11

Linear Probing

- Theorem. Simple uniform hashing $\Longrightarrow$ expected O(1) time for linear probing operations.
- Caching. Linear probing is cache-efficient.

	41	1	11	13	54			98	
0	1	2	3	4	5	6	7	8	9

h(k) = k mod 10

- Variants.
  - Quadratic probing
  - Double hashing.

Dictionaries

Data structure	SEARCH	INSERT	DELETE	space
linked list	O(n)	O(1)	O(1)	O(n)
direct addressing	O(1)	O(1)	O(1)	O(|U|)
chained hashing	O(1)†	O(1)	O(1)	O(n)
linear probing	O(1)†	O(1)†	O(1)†	O(n)

† = expected time assuming simple uniform hashing

Hashing

- Dictionaries
- Chained Hashing
- Linear Probing
- Hash Functions

Hash Functions

- Simple hash functions.
  - h(k) = k mod m. Typically, m is prime.
  - $h(k) = \lfloor m(kZ - \lfloor kZ \rfloor) \rfloor$, for constant Z, 0 < Z < 1.
- Universal hash functions.
  - Choose hash functions randomly from family of hash functions.
  - Designed to have strong guarantees on collision probabilities.
  - ⇒ Dictionaries with constant expected time performance.
  - Expectation on random choice of hash function. Independent of input set.
- Other hash functions.
  - Tabulation hashing, MurmurHash, SHA-xxx, FNV, ...
- Applications.
  - Cryptography, similarity, coding, ...

Hashing

- Dictionaries
- Chained Hashing
- Linear Probing
- Hash Functions