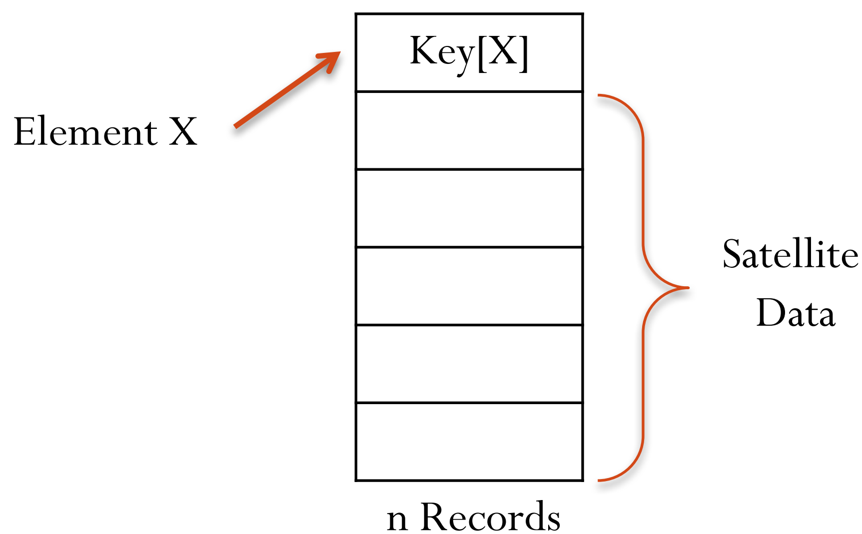
CME 2001
Data Structures and Algorithms

Zerrin Işık
zerrin@cs.deu.edu.tr

Hash Tables

Hash Tables

- Many applications require a dynamic set that supports only the ***dictionary operations*** INSERT, SEARCH, DELETE.
- E.g., A symbol table in a compiler.
- A hash table is effective structure to implement a dictionary.
- The expected search time is O(1), however, it could be Θ(n) in the worst-case.



Collision Resolution by Open Addressing

Alternative method for handling collisions.

Idea:

- Store all keys in the hash table itself (needs a larger table)
- If a collision occurs, successfully examine (*probe*) hash table until an empty cell is found.

Hash Function => $h : U \times \underbrace{\{0, 1, \ldots, m-1\}}_{\text{probe number}} \rightarrow \underbrace{\{0, 1, \ldots, m-1\}}_{\text{slot number}}.$

Probe Sequence => $\langle h(k, 0), h(k, 1), \ldots, h(k, m-1) \rangle$

Open Addressing Operations

```
HASH-SEARCH(T, k)
  i = 0
  repeat
      j = h(k, i)
      if T[j] == k
          return j
      i = i + 1
  until T[j] == NIL or i = m
  return NIL
```

```
HASH-INSERT(T, k)
  i = 0
  repeat
      j = h(k, i)
      if T[j] == NIL
          T[j] = k
          return j
      else i = i + 1
  until i == m
  error "hash table overflow"
```

DELETION:

- Marked removed key with "DELETED" flag instead of NIL
- "Search" should treat DELETED as though the slot holds a key that does not match the one being searched for.
- "Insertion" should treat DELETED as though the slot were empty, so that it can be reused.

Probing Strategies

- Linear Probing
- Quadratic Probing
- Double Hashing

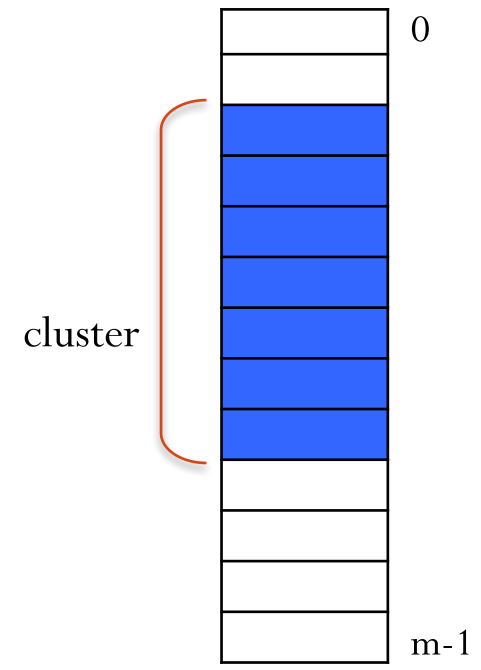
Linear Probing

$h(k,i) = (h`(k) + i) \bmod m$ (where $h`(k)$ is ordinary hash function)

i.e. the probe sequence starts at slot $h`(k)$ and continues sequentially through the table, wrapping after slot m-1 to slot 0.

- Suffers from ***primary clustering***:

  => long runs of occupied sequences build up.
- Avg. search and insertion times increase.



Quadratic Probing

$h(k,i) = (h`(k) + c_1 i + c_2 i^2) \bmod m$

where $c_1, c_2 \neq 0$ are constants, $i = 0, 1, \ldots, m-1$.

- Suffers from ***secondary clustering***:

  => if two distinct keys have the same h` value, then they have the same probe sequence.

Double Hashing

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

where $h_1(k)$ and $h_2(k)$ are ordinary hash functions.

E.g.:

$m=13$

$h_1(k) = k \bmod 13$

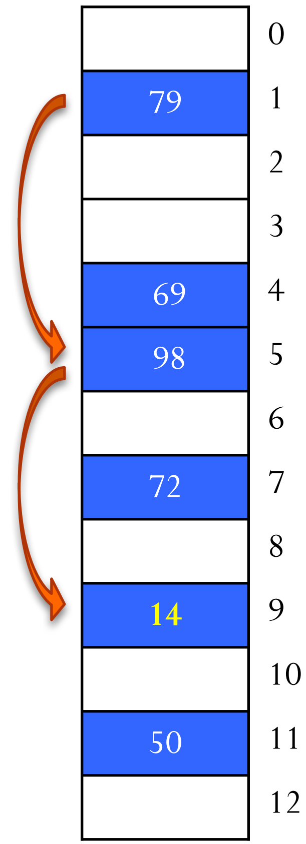
$h_2(k) = 1+(k \bmod 11)$

Insert key "14"

$h(k,0) = (h_1(14) + 0.h_2(14)) \bmod 13 = 1$ ✗

$h(k,1) = (h_1(14) + 1.h_2(14)) \bmod 13 = 5$ ✗

$h(k,2) = (h_1(14) + 2.h_2(14)) \bmod 13 = 9$ ✓



Rehashing

- The hash table will be inefficient, when it becomes full i.e., load factor gets larger, close to 1.
- What to do?
  - Replace hash table with a larger table, re-insert all items to new table with new hash function (i.e., rehashing).
  - New table size should be a *prime number*
- When rehashing should be applied?
  - If load factor > 0.5
  - Get an insertion fail

Rehashing example

- Insert 8, 25, 0, 13
- Linear probing: h(x) = x mod 5
- $\alpha = 4/5 = 0.8 \Rightarrow$ *Rehash*

25	0
0	1
	2
8	3
13	4

- New table size = 11
- h(x) = x mod 11
- Insert 8, 25, 0, 13

0	0
	1
13	2
25	3
	4
	5
	6
	7
8	8
	9
	10

Rehashing Cost

- Replace hash table with a larger table : O(1)
- Scan current table to fetch each item : O (1).n
- Re-insert all items to new table : O(n)

=> Total running time: O(n) + O(n) = O(n)

- It is acceptable cost, since rehashing does no occur frequently

Analysis of Open Address Hashing

<u>Assumptions:</u>

- Analysis is in terms of load factor $\alpha = n / m$.
- Assume that the table never completely fills, so we always have $0 \leq \alpha \leq 1$.
- Assume uniform hashing.
- No deletion.
- In a successful search, each key is equally likely to be searched for.

<u>Theorem:</u>

The expected number of probes in an unsuccessful search is at most $1/(1-\alpha)$.

Hash Table - Summary

- Used to implement the insert and find operations in *constant* average time.
  - it depends on the *load factor*
- It is important to have a prime table size, a correct choice of load factor and hash function.
- For separate chaining, the load factor should be close to 1.
- For open addressing, load factor should not exceed 0.5 unless this is completely unavoidable.
  - Rehashing can be implemented to grow (or shrink) the table.