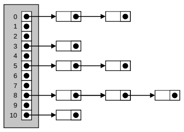


CSIS 3103

Ch 7: Hashing

Collision Resolution

Collisions are nearly inevitable

- How to deal with two keys that map to the same cell?
- We'll consider two ways to organize hash tables to handle collisions
  - Open addressing
  - Chaining

Open Addressing

- Requires:

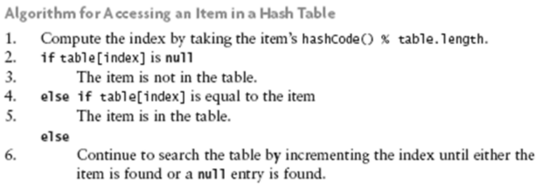
  A table with more cells than the expected number of items
- Put one item in each bucket
- Research has shown that using a prime number for table size gives a better distribution of indices

Collisions with Linear Probing

Resolving collisions in open-address hashing:

- *Linear probing*: If $h(key)$ produces a collision, try $h(k) + 1$, $h(k) + 2$, … until an empty cell (`null`) is found (wrap around at end)
- If the table gets close to being full, performance degrades
  - A solution is *rehashing*: making a bigger hash table and moving the entries into it

Open Addressing with Linear Probing



Algorithm for Accessing an Item in a Hash Table

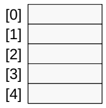
```
1.  Compute the index by taking the item's hashCode() % table.length.
2.  if table[index] is null
3.      The item is not in the table.
4.  else if table[index] is equal to the item
5.      The item is in the table.
    else
6.      Continue to search the table by incrementing the index until either the
        item is found or a null entry is found.
```

Hash Code Insertion Example 1

Values to insert:

`Tom Dick Harry Sam Pete`

Name	hashCode()	hashCode()%5
"Tom"	84274	4
"Dick"	2129869	4
"Harry"	69496448	3
"Sam"	82879	4
"Pete"	2484038	3



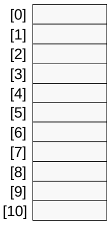
`table.length = 5`

Hash Code Insertion Example 2

Values to insert:

`Tom Dick Harry Sam Pete`

Name	hashCode()	hashCode()%11
"Tom"	84274	3
"Dick"	2129869	5
"Harry"	69496448	10
"Sam"	82879	5
"Pete"	2484038	7



`table.length = 11`

Deleting an Item Using Open Addressing

You cannot simply set a deleted table entry to null

- Think about searching for an item that may have collided with the deleted item

Instead, mark the location as available but previously occupied

- Deleted items waste storage space and reduce search efficiency

Collisions Handling Alternatives

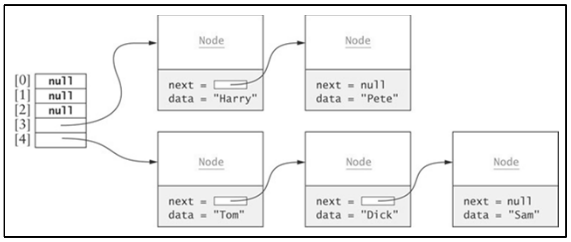
- Linear probing tends to form clusters of keys, causing longer search chains
- Other methods for handling collisions in open-address hashing, for example

  Quadratic probing – add square of increment values: $h(k) + 1^2$, $h(k) + 2^2$, $h(k) + 3^2$, …
- But now, the usual way to implement hash tables is *chaining*…

Chaining (Closed addressing)

- Create a table of m buckets
- Each bucket references a linked list () that contains all of the items that hash to the same table index
- Only items that have the same value for their hash codes will be examined when looking for an object

Chaining



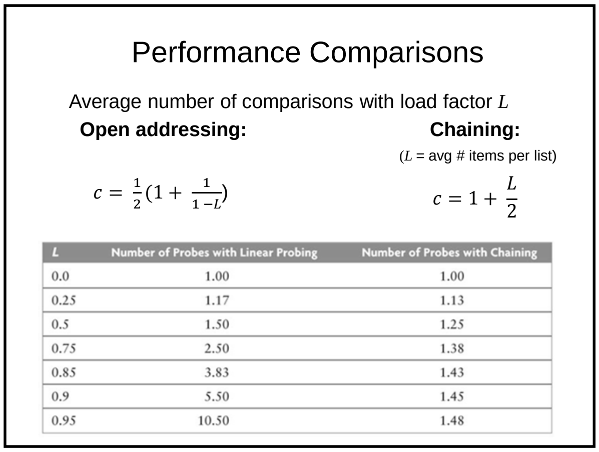
Performance of Hash Tables

*Load factor: T*he number of filled cells divided by the table size

The lower the load factor, the better the performance

- Smaller chance of collisions when a table is sparsely populated

If there are no collisions, performance for search and retrieval is O(1) regardless of table size



Performance Comparisons

Average number of comparisons with load factor L

Open addressing:

$$c = \frac{1}{2}(1 + \frac{1}{1 - L})$$

Chaining:

(L = avg # items per list)

$$c = 1 + \frac{L}{2}$$

L	Number of Probes with Linear Probing	Number of Probes with Chaining
0.0	1.00	1.00
0.25	1.17	1.13
0.5	1.50	1.25
0.75	2.50	1.38
0.85	3.83	1.43
0.9	5.50	1.45
0.95	10.50	1.48