Last Lectures

- We saw how to ...
  - describe collision in hashing
  - present collision resolution strategies
  - discuss tradeoffs of these collision resolution strategies
Learning Outcomes

- At the end of these lectures, a student will be able to:
  - define hashing as well as chained and open addressed hash table
  - discuss tradeoffs in designing hash functions and between collision resolution strategies
  - demonstrate and trace operations on hash table
Today’s menu

- Our goal in this lecture is to
  - describe another collision resolution strategy: **chaining**
  - discuss efficiency related to hashing
Collision resolution strategies so far

- Using **linear probing hashing**, experiments have shown that when a hash table is ...
  - Up to 50% full -> $O(1)$
  - $> 50\%$ full -> clusters start to form (primary clusters appear)
  - $\sim 70\%$ full -> clusters start growing into each other and performance is reduced from $O(1)$ to $O(n)$

- How can this be improved?
One more open addressing collision resolution strategy

- Let’s increase the size of our hash table
  1. Estimate capacity required (# of elements)
  2. Double it and use this number as size of hash table

- This way, hash table is never more than 50% full

- Using this solution, time efficiency gained from hashing comes at the cost of space efficiency
Introducing the load factor

\[ \sigma = \frac{n}{B} \]

example: \( \sigma = 1 \rightarrow \text{full hash table} \)

\[ \sigma = 0.5 \rightarrow \text{hash table that is 50\% full} \]
Hashing so far...

- 1 element stored @ each hash index
Another strategy to resolve collisions

- Why don’t we resolve collisions not by producing a sequence of probing locations, but ...
  - ... by constructing our hash table differently
    - By allowing more than 1 element to be stored in at a given hash index ...
Chaining

- Elements do not have to be stored in the hash table itself
- Hash table structure
  - mixture of bounded and unbounded data structure
  - Section 1 of hash table is bounded
    - Hash table is a bounded data structure -> array allowing indexing (direct access)
    - Each cell of the hash table is associated with a chain and no longer contains an element
  - Section 2 of hash table is unbounded
    - Chain is unbounded (expandable) data structure such as a linked list, etc...
    - This is the section containing the elements
Chaining

- **INSERT**
  - \( H(K) = \text{index} \rightarrow h_0 \)
  - ADD ELEMENT TO CHAIN (insertFirst(s))

- **SEARCH**
  - \( H(K) = \text{index} \rightarrow h_0 \)
  - TRAVERSE CHAIN \( \rightarrow \) LOOKING FOR "KEY"
  - SEARCH - IF "KEY" NOT IN AN ELEMENT OF CHAIN
  - THEN ELEMENT NOT IN HASH TABLE

**Adv:** LENGTH OF CHAIN \(<\) LENGTH OF CLUSTER \(<\) 17
Example: Chaining

Insert the following elements with key value:
32, 47, 26, 34, 87, 39, 78, 61, 48, 66

\[ h(key) = key \mod 10 \]

Hash index:

# of probes:

Hash Table
Observation: length of chain < length of cluster
Chaining - We still need a good hash function

if Hash fn is "good":

IDEAL

CHAINS +/- EQUAL!

if Hash fn is "poor":

CHAINS UNEVEN
Chaining - We still need a good estimate of number of elements.

\[
\sigma = \frac{100}{100} = 1
\]

In reality \( \sigma = \frac{400}{100} = 4 \)
Chaining – Conclusion – 1

\[ \sigma = \frac{n}{B} \]

\( \sigma \): average length of each chain

\# of elements

\# of chains
(Size of hash table)

\[ n = B \]

Time Efficiency of insertion: \( O(1) \)

Time Efficiency of search: \( O(1 + \sigma) \)

Since we want constant time:

\( \therefore \) Common to set \( \sigma \approx 1 \) \( (n \approx B) \)

\[ \Rightarrow O(1) \]

Chain \( \Rightarrow \) only 1 element or very few

If have a good estimate of \# of elements to be stored in hash table

\[ \Rightarrow n \]
Chaining – Conclusion – 2

IF # of ELEMENTS ↑
(if σ ↑ i.e. CHAINS now have more THAN 1 ELEMENTS.)

THE TIME EFFICIENCY ↓
(now O(1+σ) i.e. time to search chain)

But not as much as with other collision resolution strategies
(Because length of chain < length of cluster)

:: CHAIN HASHING GOOD even when cannot PREDICT n EXACTLY
Chaining – Conclusion – 3

- For short chains, this is a very fast method, but increasing their length can significantly degrade insertion and retrieval time efficiency.
Terms related to hashing

- **Open addressing (closed hashing)**
  - **Open addressing**: Because we can probe hash table at any address (index) when resolving a collision
  - **Closed hashing**: Because all elements stored in the hash table (array)

- **Chaining (open hashing)**
  - **Chaining**: Because elements stored in 2nd data collection which is “chained” to hash table
  - **Open hashing**: Because all elements stored “outside” the hash table (array)
In general, hashing is ... 

\[ \text{As long as } O(1) \text{ (see next slide)} \]

---

Effective data structure for:

- Insertion
- Deletion
- Search
- Retrieval

\[ \Rightarrow O(n) \]

Worst case when hashtable becomes a long linear sequence
As long as ...

- hash fn is "good"
- distribute keys uniformly throughout hash table
- if we use:
  - linear probing hashing
  - quadratic hashing
  - random hashing
  - double hashing (rehashing)
  - chain hashing

REMEmBER ...

\[
\sigma = \frac{\text{# of elements}}{\text{size of hash table}}
\]
Complexity of hashing

All strategies give comparable results up to $\sigma = 0.5$.
Then chaining hashing wins.
However ...

- What if client wants to frequently find largest/smallest element in hash table?

- Or print elements in order?

- Disadvantage of hashing in general.

- Sort order retrieval is expensive!

  (Not efficient)
✓ Learning Check

- We can now ...
  - describe another collision resolution strategy: chaining
  - discuss efficiency related to hashing
Next Lectures

- Dictionary data collection ADT class
- External Storage