For Poll Ev

LECTURE-14

Open Addressing Hash Tables

Workings of Open Addressing Hash Tables

CS202: Data Structures (Fall 2025)

Dr Maryam Abdulghafur, Momina Khan

Department of Computer Science, SBASSE

Agenda

- Hash Tables Time complexities issues and remedies
- Collision management through – Open Addressing HashTables
- Complexity of the various functions of the Hash Tables
- Exploring issues with Open Addressing Hash Tables and their remedies

RECAP: MOTIVATION O(1) SEARCH

- Indexing is what lets us reach the O(1) goal.

  So obvious choice is an **array**

- Arrays are fixed size - more space needed! How?

  What steps are taken? What is the cost?

- What is a good hash function? On next slide

- Collisions cannot be avoided!! Handle them w/ **chaining!**

  Keep rate of collisions low!

Can there ever be a cause to shrink a Hash Table?

- What if a large number of items are deleted from the Hash Table?
- What will happen to the load factor when this happes?
- Is it good space utilization to have a very low load factor?
- In case you decide to shrink the Hash Table, what steps will it take?

    Declare an array half the size and rehash everything

Hash Function

Uniform spread

- Associated hash function must be **independent** and **deterministic** .

- **No** known hash function **eliminates collisions**. Minimize them!

- Should be efficient in time!

- Used for every **insert, delete and search** into the hash table.

Hash Code Base

- **A typical hash code base is a small prime.**
- **Why prime?**

Lower chance of resulting hash code having a bad relationship with the number of table slots

- **Why small?**

Lower cost to compute

- **A full treatment of good hash functions is well beyond the scope of our class**

Chained Hash Table - A mechanism to **handle collisions!**

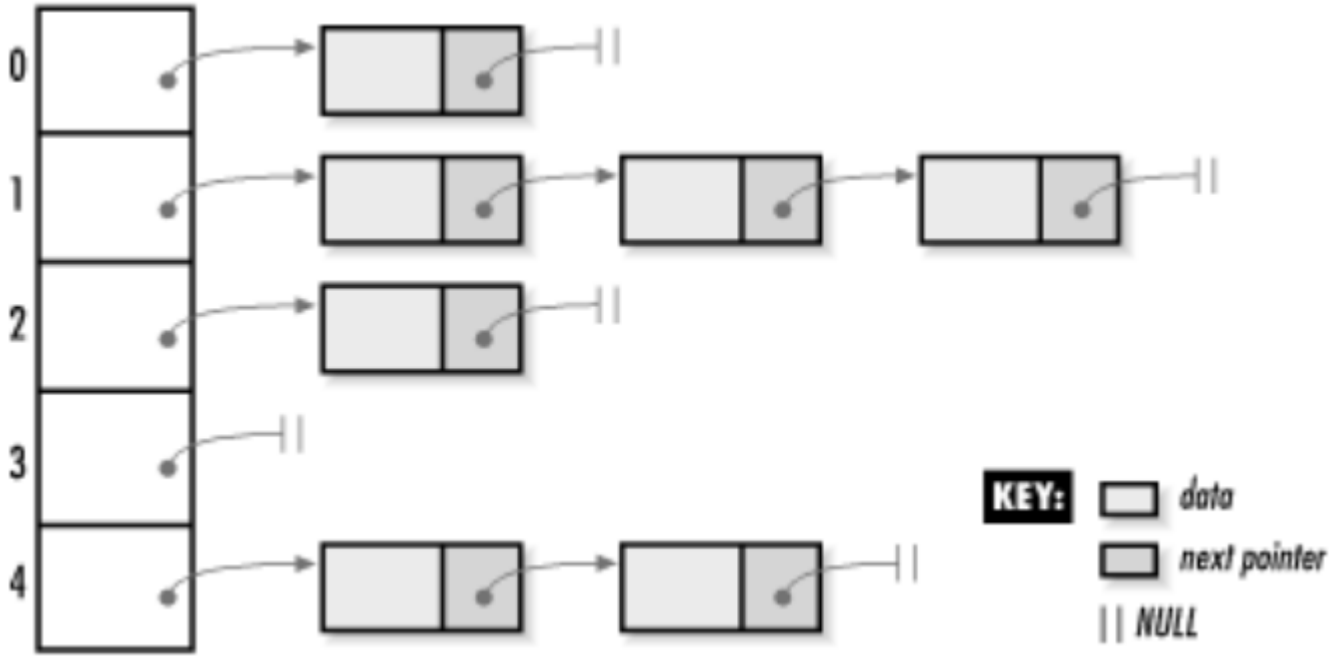
- What would be definition of the **class ChainedHashTable?**

  its data components …

  - **Array ** (Pointer to an array of pointers)**
  - **Size of array**
  - **Load factor (how many slots are occupied)**

- What is the definition of **Node** in the linked chain (linked list)?

Q: For the Chained Hash Table shown will future inserts have a greater chance of increasing chain lengths? Yes, but why?



Q. For a hash function with a uniform distribution what is the probability or chances that a future insert will get an index with a chain rather than one without a chain? 4:1

Chained Hash Table - A mechanism to **handle collisions!**

- What would be Big O complexities for **class ChainedHashTable?**
  - **Search**
  - **Insert**
  - **Delete**

Q. How are complexities related to the length of chains in the HT?

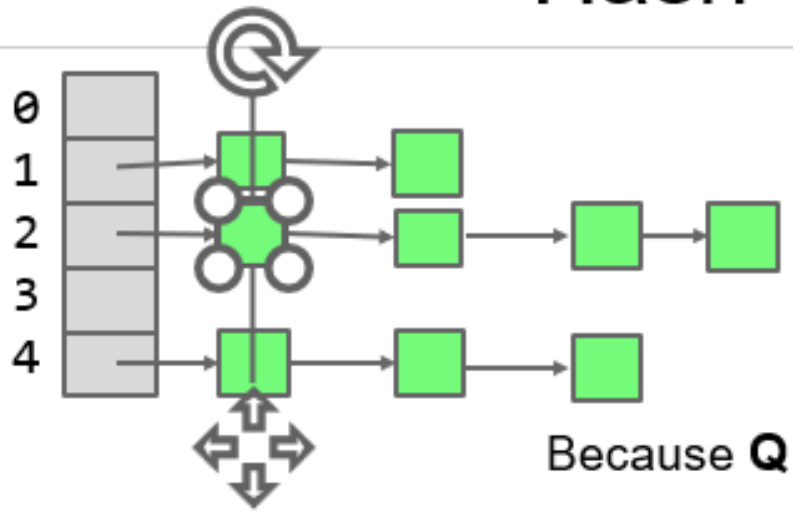
Q. Can we keep chains short? How?

Chained Hash Table - A mechanism to **handle collisions!**

- What would be Big O complexities for **class ChainedHashTable?**

  - **Search**
  - **Insert – where in the chain should I insert the new item? How does it matter?**
  - **Delete**

Hash Table Runtime



Because **Q** = Θ(n)

Because **Q** ≈ Θ(1)

Worst case runtimes

	search(key)	insert(key)
AVL Trees	Θ(log n)	Θ(log n)
Linked List	Θ(n)	Θ(1) ☼
Separate Chaining Hash Table With No Resizing	Θ(n)	Θ(n)
… With Resizing	Θ(1)†	Θ(1)*†

Hash table operations are on average constant time if:

- We monitor load factor to ensure constant average length of linked lists
- Items are evenly distributed

*: Indicates "on average"
†: Assuming items are evenly spread
☼: Depends on where you insert

Open Addressing (OA) Hash Tables

This is the other scheme for collision handling employed in an

Open Addressing Hash Table

The **motivation** is simple: better space utilization!

When we already have a lot of empty slots in the HT array why not try and accommodate collisions in these empty slots!

Linear Probing (LP)

With LP, all collisions are resolved by placing the colliding item in the next (circularly) available table cell

Each table cell inspected is called a "probe"

0	Item-1
1	Item-2
	..
	..
m-1	Item-n

e.g., h(key) = [hc(key) + i] mod 13, where the initial value of i is 0 and is incremented every time a collision occurs

Example

- **Hash function: h(key) = [hc(key) + i] mod 13**
- **Insert keys 18, 41, 22, 44, 59, 32, 49, 73 and 31(left to right order) in a table of size m=13**

0	1	2	3	4	5	6	7	8	9	10	11	12

Example

- **Hash function: h(key) = [hc(key) + i] mod 13**
- **Insert keys 18, 41, 22, 44, 59, 32, 49, 73 and 31 (left to right order) in a table of size m=13**

0	1	2	3	4	5	6	7	8	9	10	11	12
		41			18	44	59	32	22	49	73	31

How do we do search on this hash table?

How do we search with our new collision detection scheme?

Hash the item and then do linear probing to locate element!
When do we stop linear probing?

Downside is clustering?

Clustering means that we might be traversing through the chain of another key group that happens to cluster with our colliding elements. Chains are interleaved or clustered

Example

- **Hash function: h(key) = [hc(key) + i] mod 13**

Q. How will we search 31 in the Hash Table?

0	1	2	3	4	5	6	7	8	9	10	11	12
		41			18	44	59	32	22	49	73	31

Example

- **Hash function: h(key) = [hc(key) + i] mod 13**

Q. When we delete 44 from this HashTable, can you identify a problem?

A. The chain will break and we stop LP when we encounter and empty slot!

0	1	2	3	4	5	6	7	8	9	10	11	12
		41			18	44	59	32	22	49	73	31

Sol: Mark a slot with an isDeleted flag whenever a value from that slot is deleted. Initially isDeleted is false!