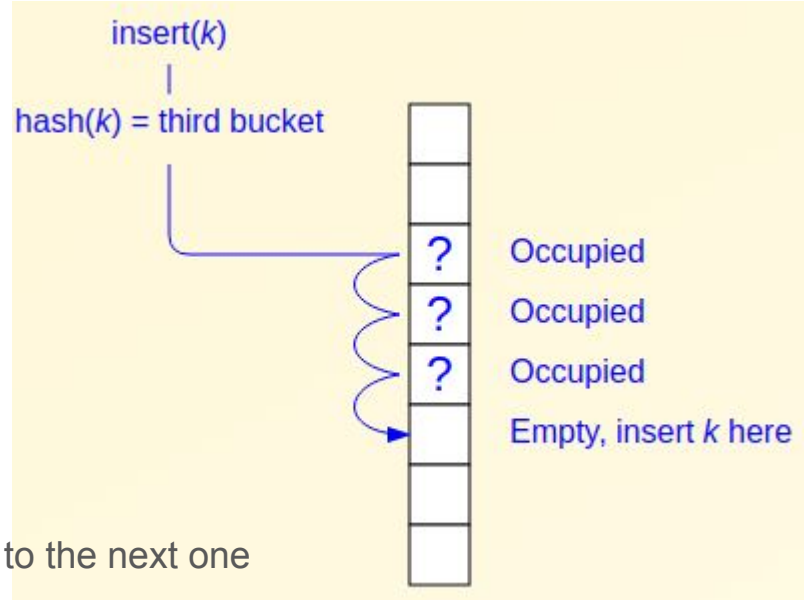
Data Structures

Linear Probing

Mostafa S. Ibrahim
Teaching, Training and Coaching since more than a decade!

Artificial Intelligence & Computer Vision Researcher
PhD from Simon Fraser University - Canada
Bachelor / Msc from Cairo University - Egypt
Ex-(Software Engineer / ICPC World Finalist)

Probing



- Another way to handle collisions
- Create only a single 1D array
- Given a hash index:
  - If it is empty, add the item
  - If not, move to the next element (**linear** probing)
    - Again, use it if it's empty. Otherwise, move to the next one
    - We may go circular in the array
    - In the worst case, all array content will be used
- Observe
  - Chaining technique: continually **expanding** in memory
  - Probing: **fixed** in memory, as it is limited to the initial array
  - In both, we can **rehash** if we have to

Img src

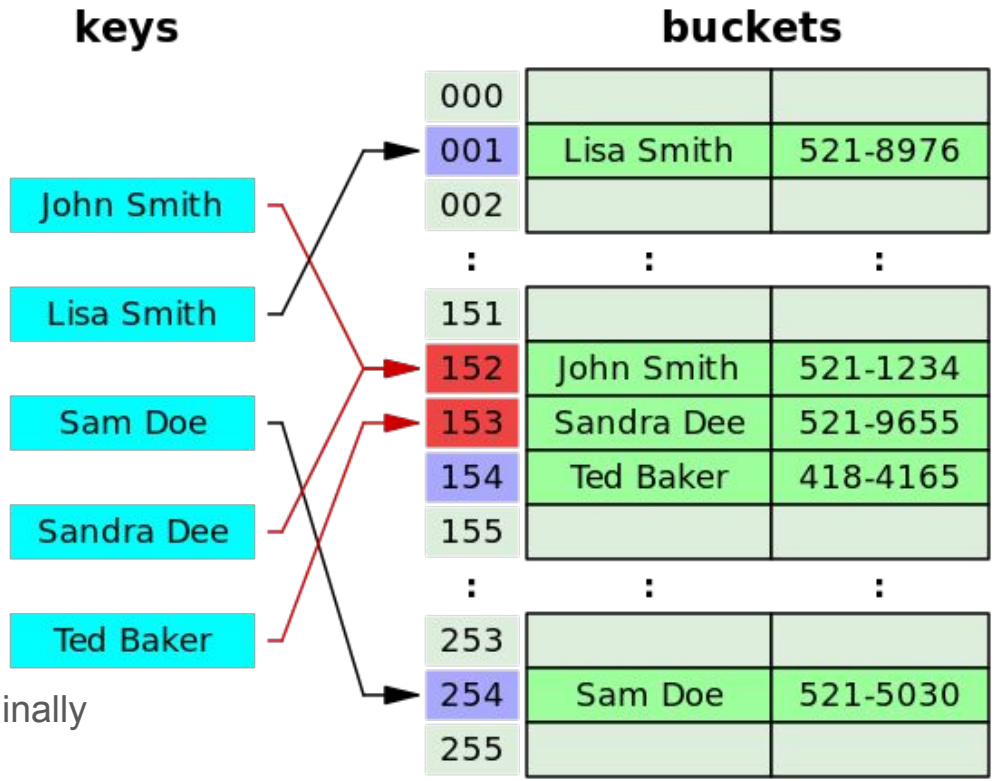
Recall the example

- Assume we have these entries, and already computed their final hash functions

Name (as search key)	Attached data (phone #)	Hash Function code
John Smith	5211234	152
Lisa Smith	5218976	1
Sam Doe	5215030	254
Sandata Dee	5219655	152 (collision)
Ted Baker	4184165	153

Put in order (linear probing)

- John Smith (152)
  - Empty: add
- Lisa Smith (001)
  - Empty: add
- Sam Doe (254)
  - Empty: add
- Sandata Dee (152)
  - 152 **not empty**. Move
  - 153 empty: add
- Ted Baker (153)
  - 153 **not empty**. Move
  - 154 empty: add
  - Notice how this wasn't a collision originally



Searching and deletion

- Search (same as put)
  - Compute the hash index. As long as our index is NOT empty, or if the entry doesn't represent the target key, move 1 step further
- Deletion is tricky!
  - Imagine we've inserted items that ended up as 5 consecutive elements in the table
  - E.g. A, B, C, D, E
  - Search(D) ⇒ Exist
  - Delete C ⇒ A, B, Empty, D, E
  - Search(D) ⇒ not exist!
  - A big problem is that search stops when it finds an empty item! However, this entry is empty due to deletion
  - Trivial solution: Mark a cell as deleted so that our search can continue if necessary

Example using integers

- Assume we have the following integers
  - [**18, 41, 22**, 44, 59, 32, 31, 73]
  - Their respective hash indices: [5, 2, 9, 5, 7, 6, 5, 8] and our array is 13 cells

	0	1	2	3	4	5	6	7	8	9	10	11	12
Insert 18 (5)						18							
Insert 41 (2)			41			18							
Insert 22 (9)			41			18				22			

Example using integers

- Assume we have the following integers
  - [18, 41, 22, **44**, 59, 32, 31, 73]
  - Their respective hash indices: [5, 2, 9, **5**, 7, 6, 5, 8] and our array is 13 cells

	0	1	2	3	4	5	6	7	8	9	10	11	12
Insert 44 (5)			41			18	**44**			22			

5? not empty
6? Empty. Use it

Example using integers

- Assume we have the following integers
  - [18, 41, 22, 44, **59**, 32, 31, 73]
  - Their respective hash indices: [5, 2, 9, 5, **7**, 6, 5, 8] and our array is 13 cells

	0	1	2	3	4	5	6	7	8	9	10	11	12
Insert 59 (7)			41			18	44	59		22			

Example using integers

- Assume we have the following integers
  - [18, 41, 22, 44, 59, **32**, 31, 73]
  - Their respective hash indices: [5, 2, 9, 5, 7, **6**, 5, 8] and our array is 13 cells

	0	1	2	3	4	5	6	7	8	9	10	11	12
Insert 32 (6)			41			18	44	59	**32**	22			

6? not empty
7? not empty
8? Empty. Use it

Example using integers

- Assume we have the following integers
  - [18, 41, 22, 44, 59, 32, **31**, 73]
  - Their respective hash indices: [5, 2, 9, 5, 7, 6, **5**, 8] and our array is 13 cells

	0	1	2	3	4	5	6	7	8	9	10	11	12
Insert 31 (5)			41			18	44	59	32	22	**31**		

5? not empty
6? not empty
7? not empty
8? not empty
9? not empty
10? Empty. Use it

Example using integers

- Assume we have the following integers
  - [18, 41, 22, 44, 59, 32, 31, **73**]
  - Their respective hash indices: [5, 2, 9, 5, 7, 6, 5, **8**] and our array is 13 cells

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

8? not empty
9? not empty
10? not empty
11? Empty. Use it

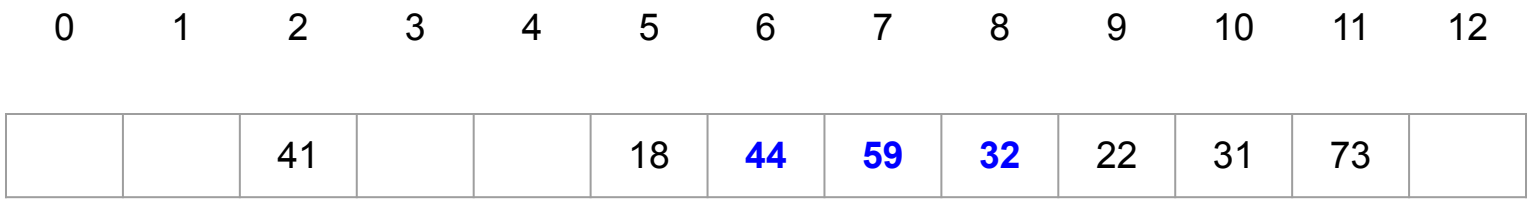
Example using integers

- Search 115 (hash = 8)
  - Idx = 8: 32 == 115? No. Move
  - Idx = 9: 22 == 115? No. Move
  - Idx = 10: 31 == 115? No. Move
  - Idx = 11: 73 == 115? No. Move
  - Idx = 12: None. **Not found**

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

Example using integers

- Search 32 (hash = 6)
  - Idx = 6: 44 == 32? No. Move
  - Idx = 7: 59 == 32? No. Move
  - Idx = 8: 32 == 32? Found



Example using integers

- Delete 59 (hash = 7)
  - Idx = 7: 59 == 59? Yes, delete (mark as deleted, but leave it showing something was here)

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

Example using integers

- Search 32 (hash = 6)
  - Idx = 6: 44 == 32? No. Move
  - Idx = 7: X == 32? No. Move [without an indicator (deleted), we fail here!]
  - Idx = 8: 32 == 32? Found

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

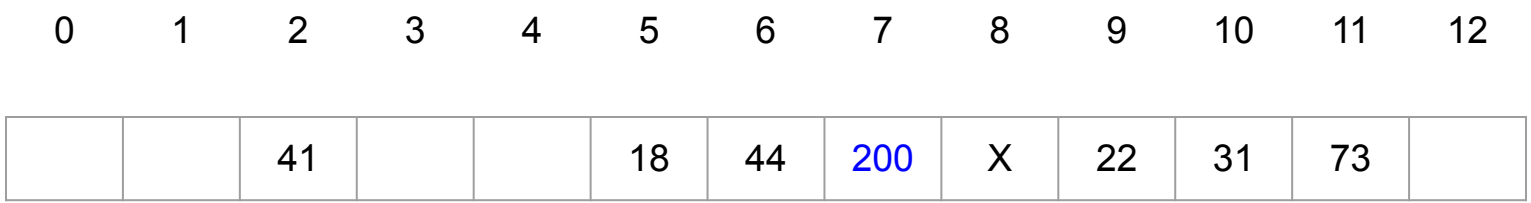
Example using integers

- Delete 32 (hash = 6)
  - Idx = 6: 44 == 32? No. Move
  - Idx = 7: X == 32? No. Move
  - Idx = 8: 32 == 32? Mark as deleted

0	1	2	3	4	5	6	7	8	9	10	11	12
		41			18	**44**	**X**	**X**	22	31	73	

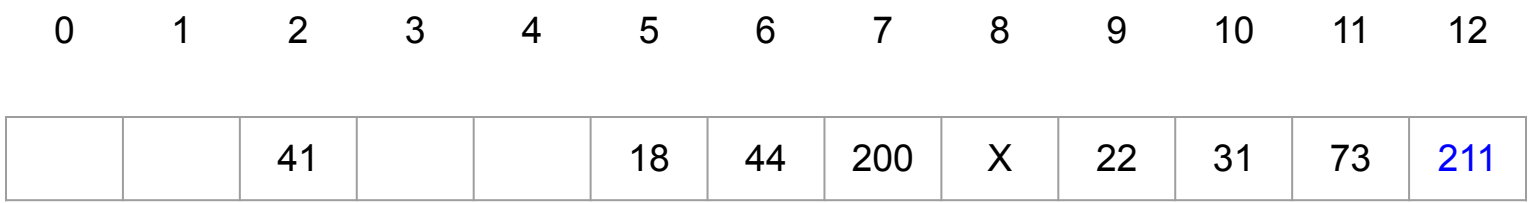
Example using integers

- Insert 200 (hash = 6)
  - Idx 6: empty? No. move
  - Idx 7: is marked as deleted. Just use it



Example using integers

- Insert 211 (hash = 12)
  - Idx 12: empty? Use it



Example using integers

- Insert 315 (hash = 12)
  - Idx 12: empty? No move. Go circular to 0
  - Idx 0: empty? Use it
- Note: there are only 5 elements remaining in this table (4 + X)

0	1	2	3	4	5	6	7	8	9	10	11	12
315		41			18	44	200	X	22	31		211

Rehashing

- Just as with chaining, we can rehash if the load factor limit is reached
- There is another reason to use the probing technique
- Imagine we have several deletions. The array will have multiple 'X' marks
- But this means our search will take a long time
  - It's best to rehash at this stage
- With a good hash function, it can be shown that the expected number of insertions is approximately 1 / (1 - load_factor) steps
  - E.g. for a load factor of 0.75, we take 4 steps ⇒ O(1)

Probing Techniques

- Linear probing (today)
  - Index = (initial_hash + i)%table_size for i = {0, 1, 2, 3, 4} ⇒ move to next position
  - If initial_hash = 10 ⇒ try as long as it's not empty {10, 11, 12, 13, 14, 15….}
  - Issue: creates blocks of consecutive values ⇒ More time for search
- Quadratic Probing
  - Index = (initial_hash + **i * i**)%table_size
  - We jump to square positions. E.g. 10 + 0, then 10+1, then 10 + 4, then 10+9, then 10+16
- Double Hashing
  - Use 2 independent hash functions
  - Index = (initial_hash1 + initial_hash2 *** i**)%table_size
  - The 2nd hash function should have several characteristics

"Acquire knowledge and impart it to the people."

"Seek knowledge from the Cradle to the Grave."