Hashing

Part Two

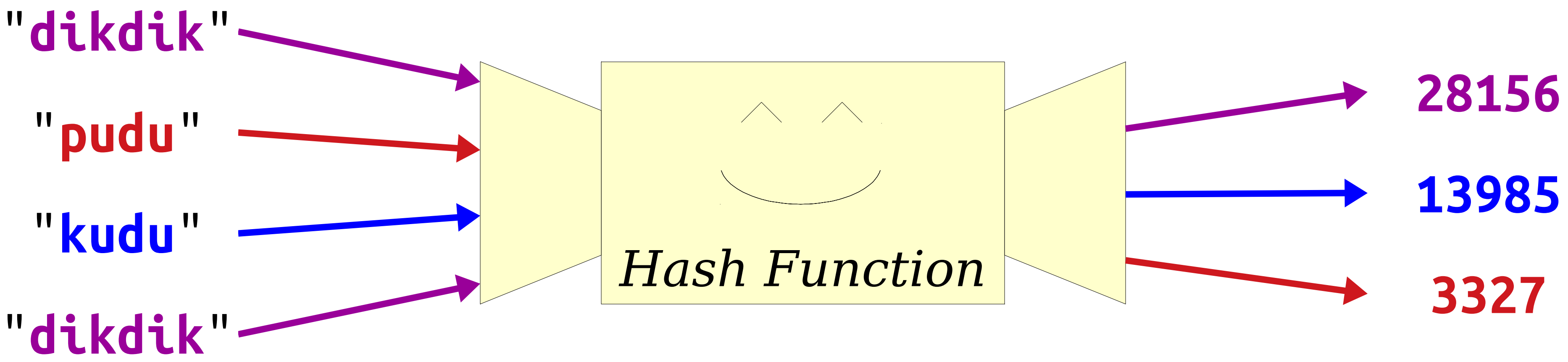
Outline for Today

- ***Recap from Last Time***
  - A quick refresher on hash functions.
- ***Hashing Variants***
  - We built a hash table last lecture. There are other strategies we could have used.
- ***Linear Probing***
  - A deceptively simple and fast hashing scheme.
- ***Robin Hood Hashing***
  - Moving items around in a hash table.

Recap from Last Time

Hash Functions

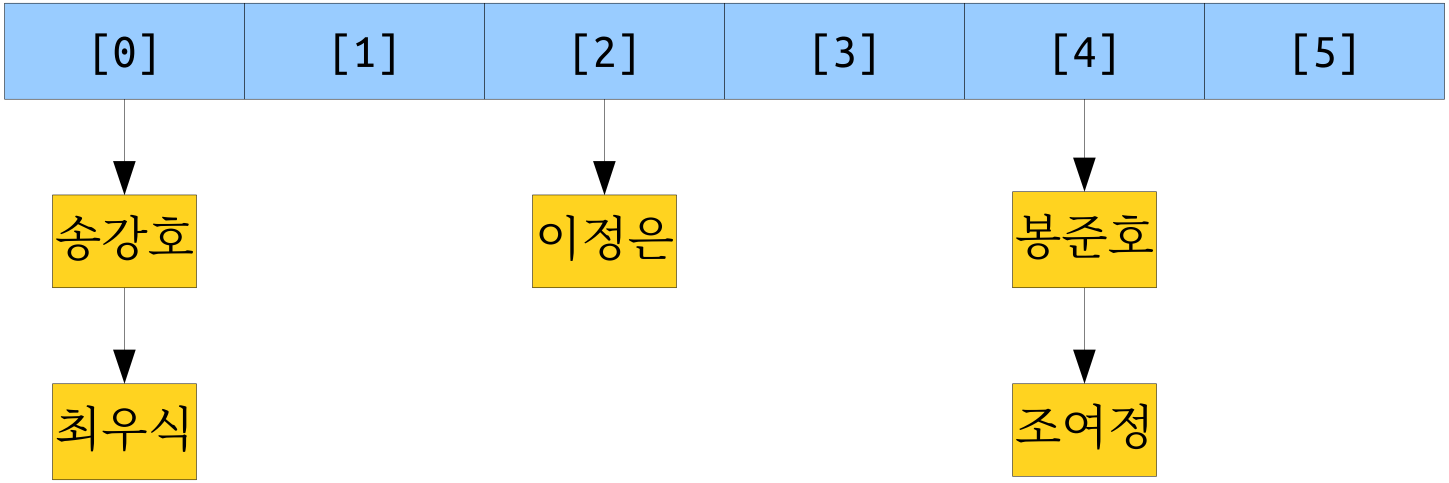
- A ***hash function*** is a function that takes an object as input and produces an integer called its ***hash code***.



- If you feed the same input to a hash function multiple times, it will always produce the same output.
- Aside from this, though, the outputs of hash functions should look more or less random.

Hash Tables

- A ***hash table*** is a data structure where items are positioned in an array based on their hash code.
- Last time, we saw ***chained hashing***, where all items with the same hash code are stored in the same slot.



New Stuff!

Hash Collisions

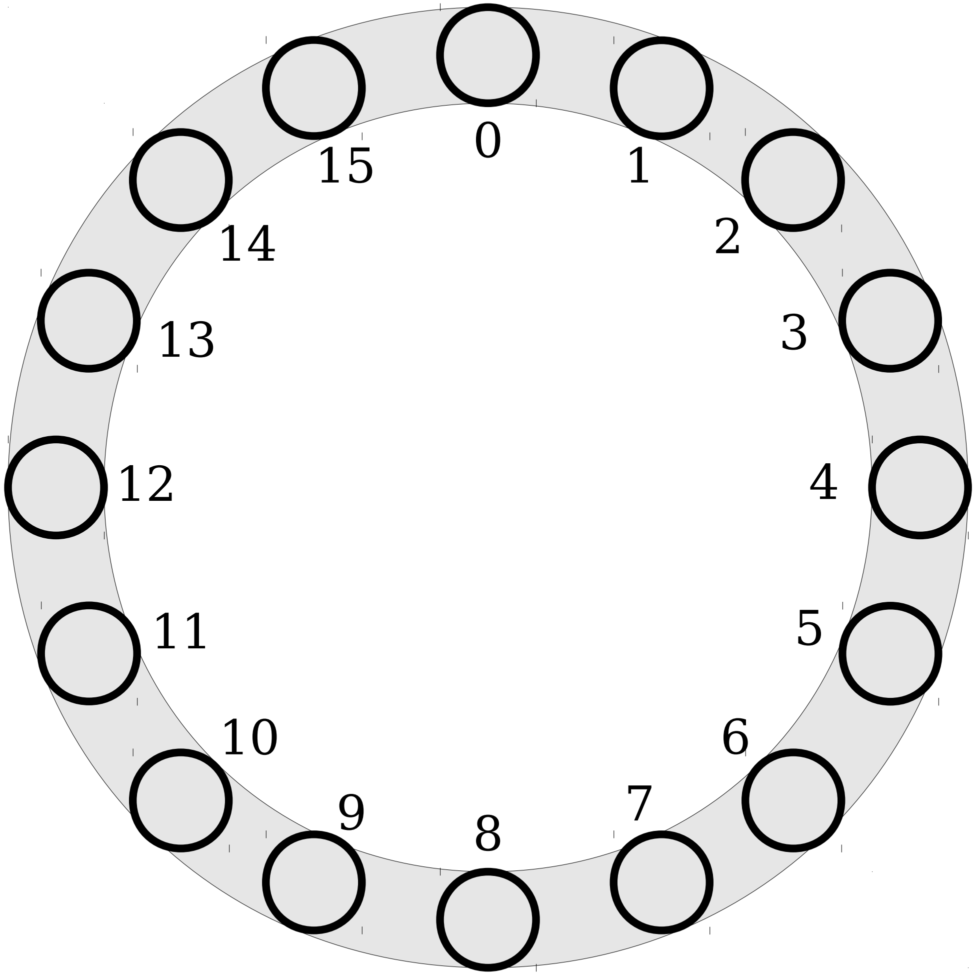
- There is a family of other hash tables that use an idea called ***open addressing***.
- In open addressing,

  ☞ ***each table slot holds at most one element***. ☜

- If multiple elements hash to the same slot, they “leak out” and spill over into other free slots.

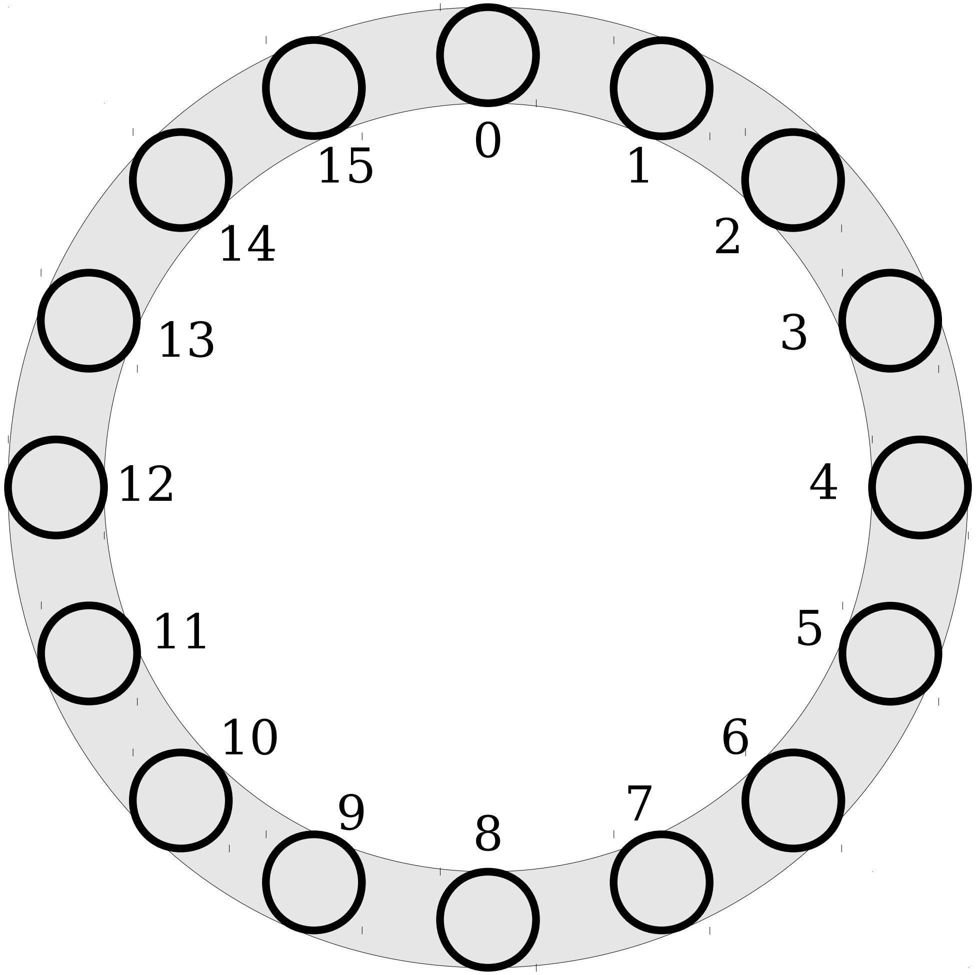
Linear Probing

- ***Linear probing*** is a simple open-addressing hashing strategy.
- We maintain an array of ***slots***, which we think of as forming a ring.



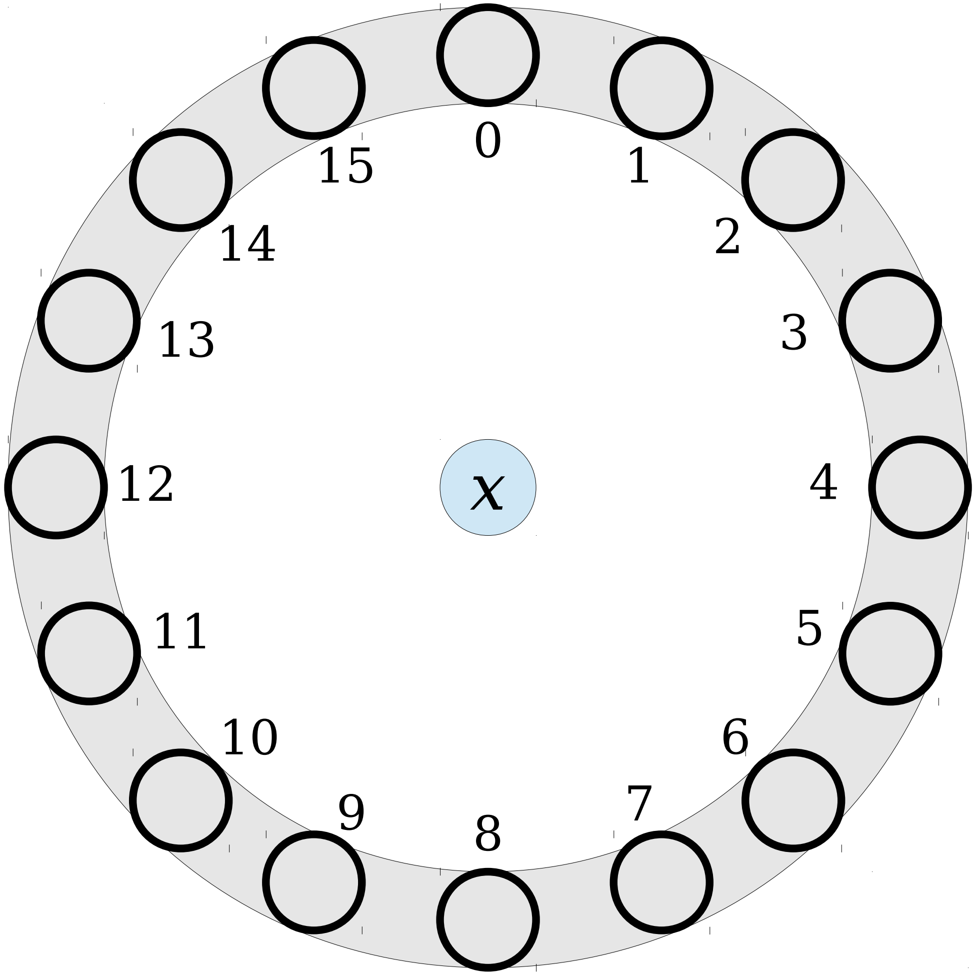
Linear Probing

- To insert an element, compute its hash code and try to place it at the slot with that number.



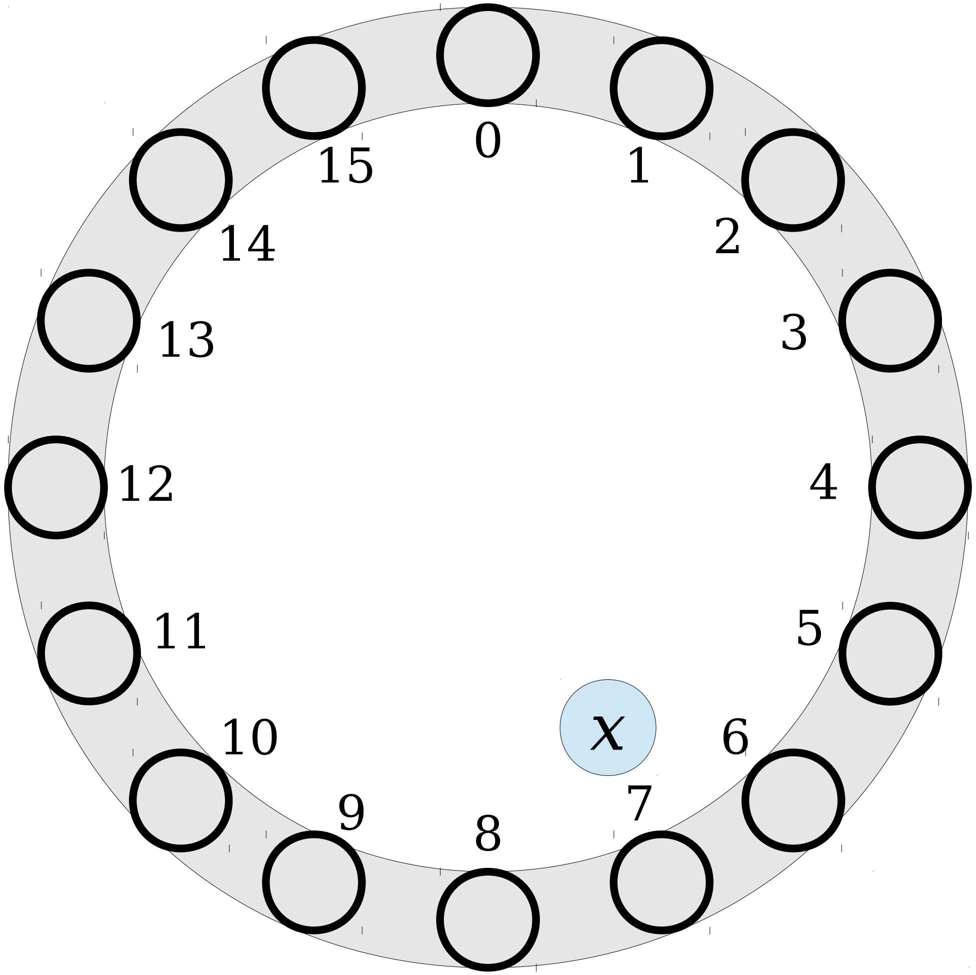
Linear Probing

- To insert an element, compute its hash code and try to place it at the slot with that number.



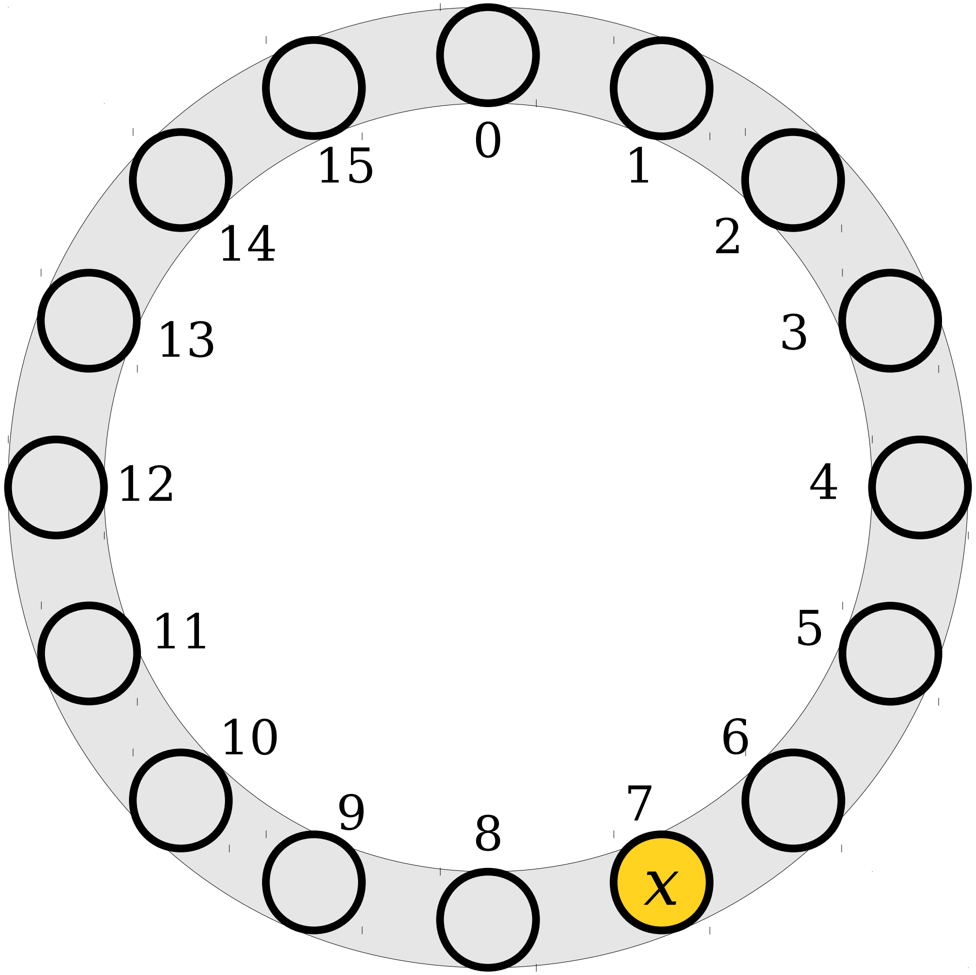
Linear Probing

- To insert an element, compute its hash code and try to place it at the slot with that number.



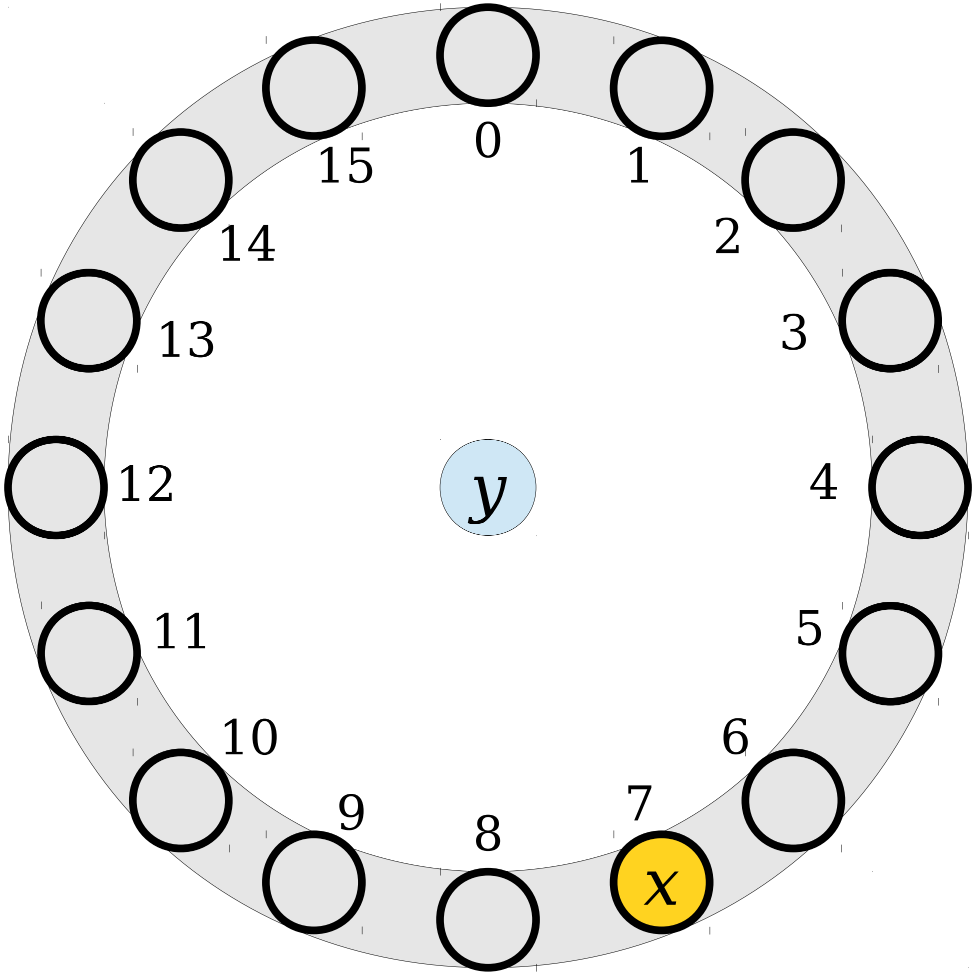
Linear Probing

- To insert an element, compute its hash code and try to place it at the slot with that number.



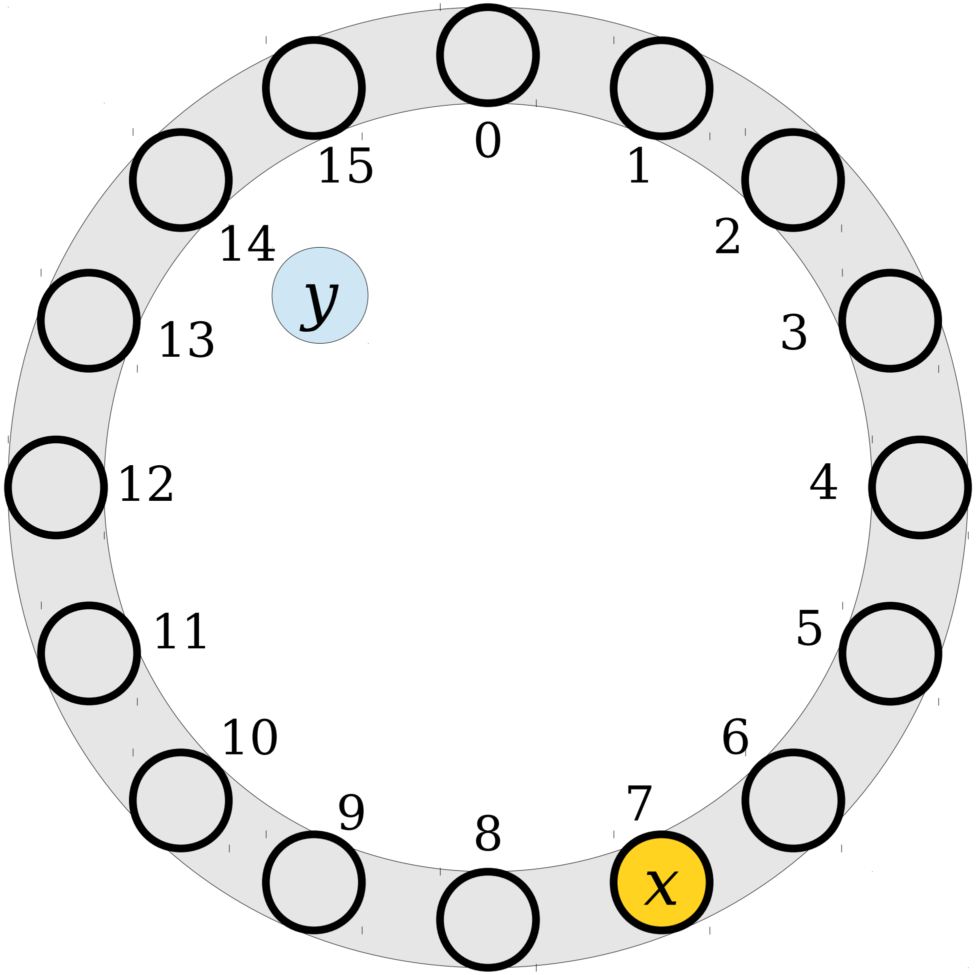
Linear Probing

- To insert an element, compute its hash code and try to place it at the slot with that number.



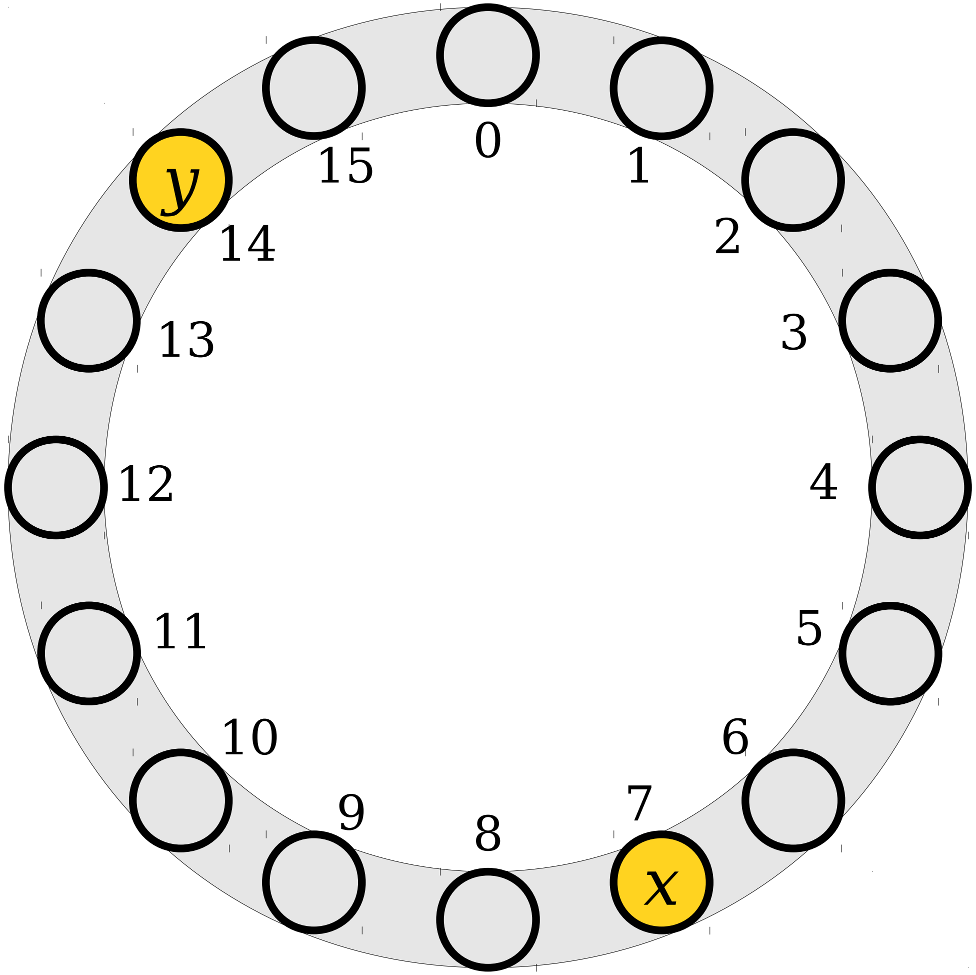
Linear Probing

- To insert an element, compute its hash code and try to place it at the slot with that number.



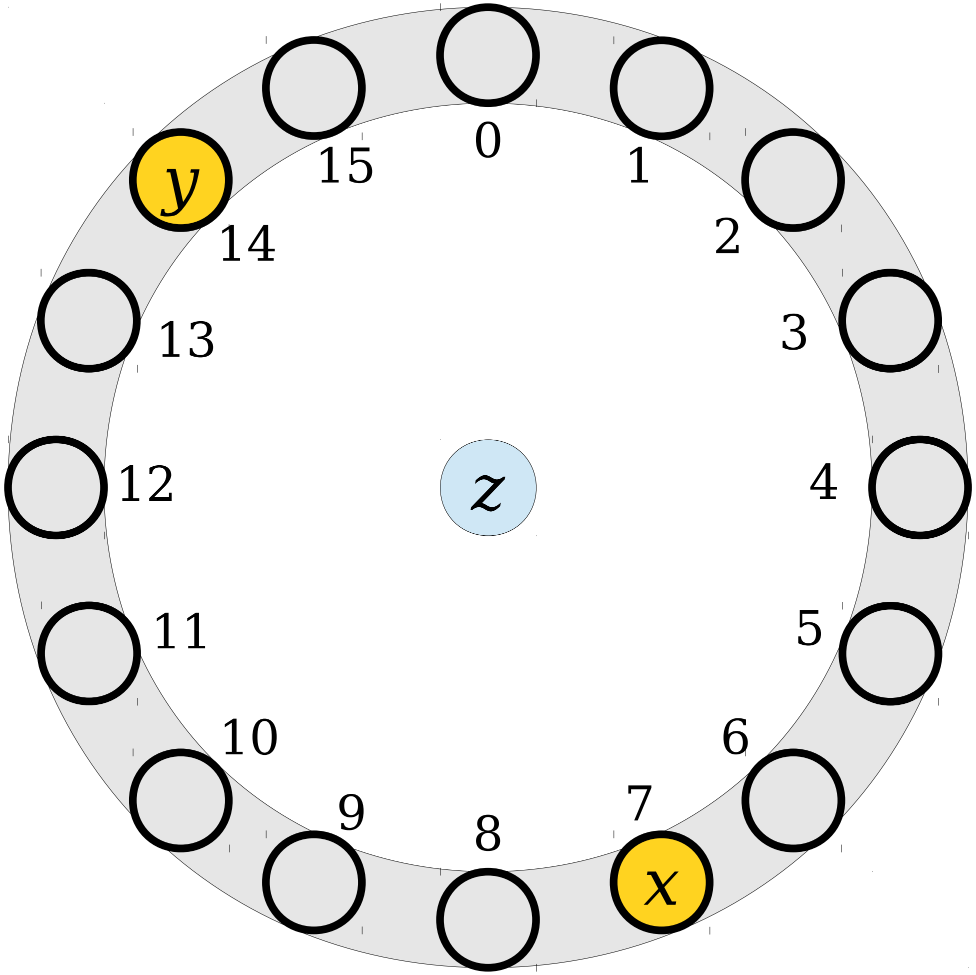
Linear Probing

- To insert an element, compute its hash code and try to place it at the slot with that number.



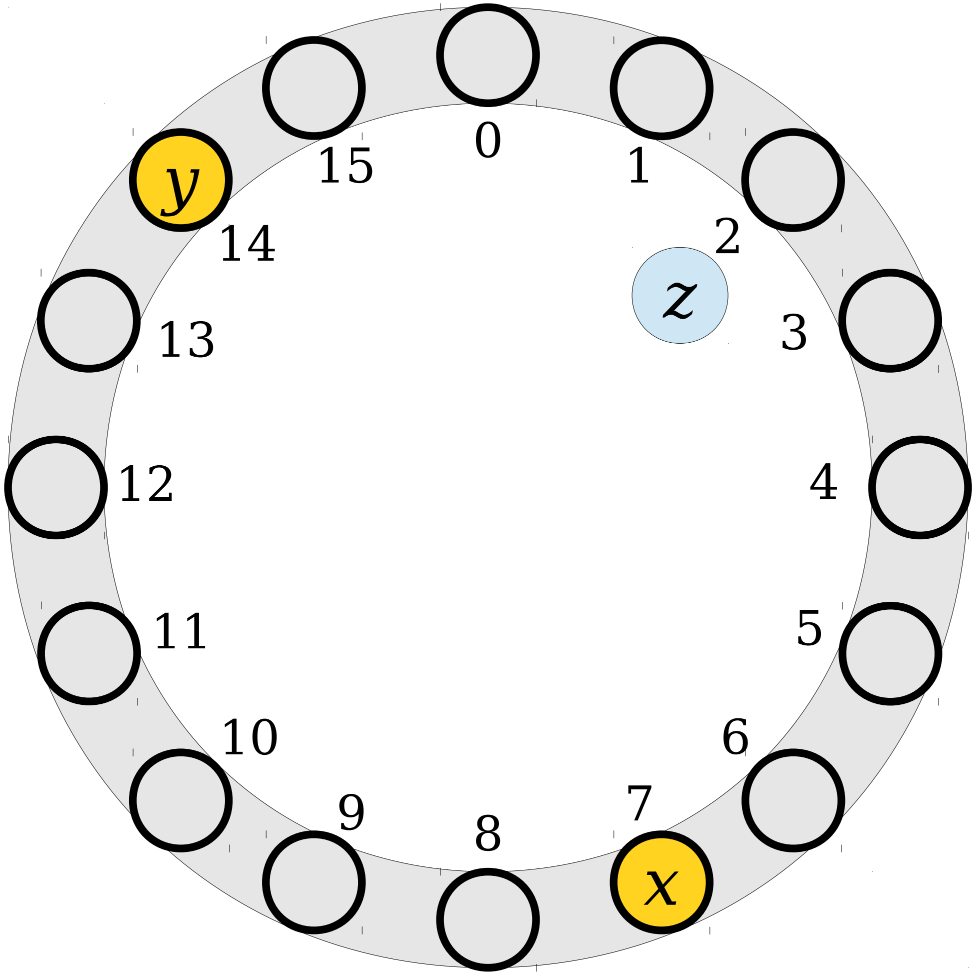
Linear Probing

- To insert an element, compute its hash code and try to place it at the slot with that number.



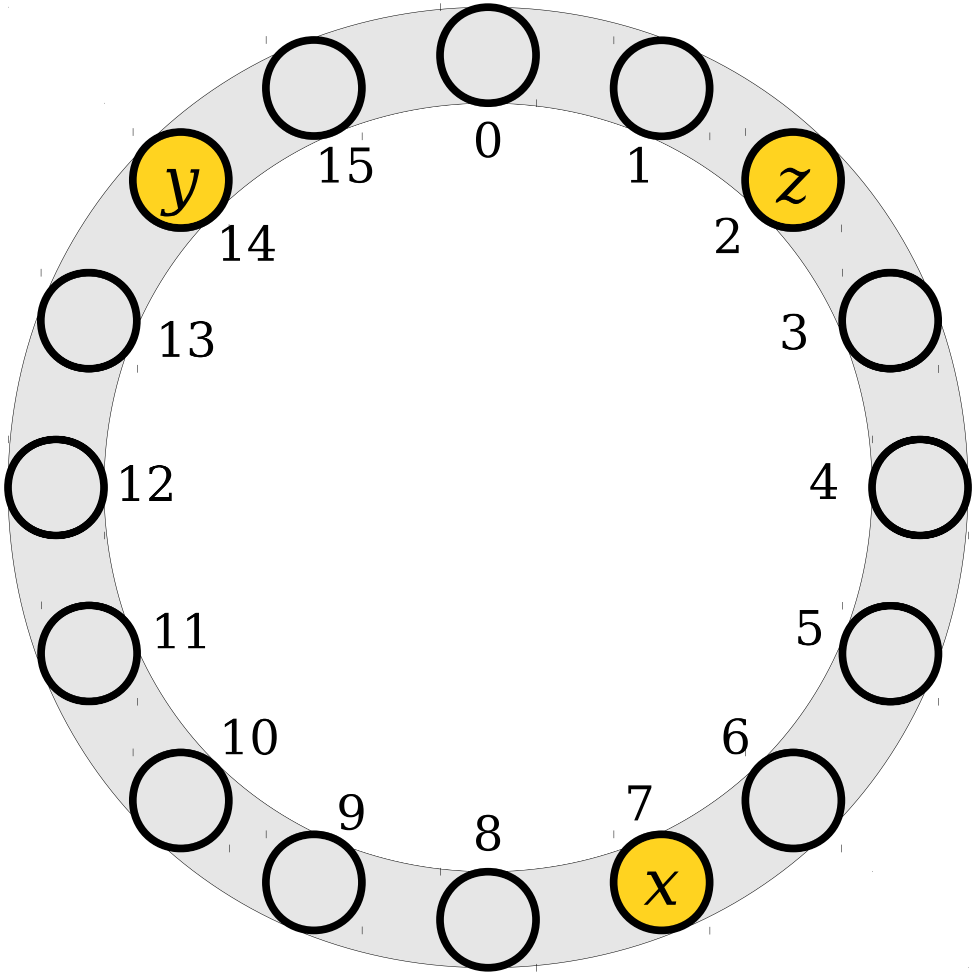
Linear Probing

- To insert an element, compute its hash code and try to place it at the slot with that number.



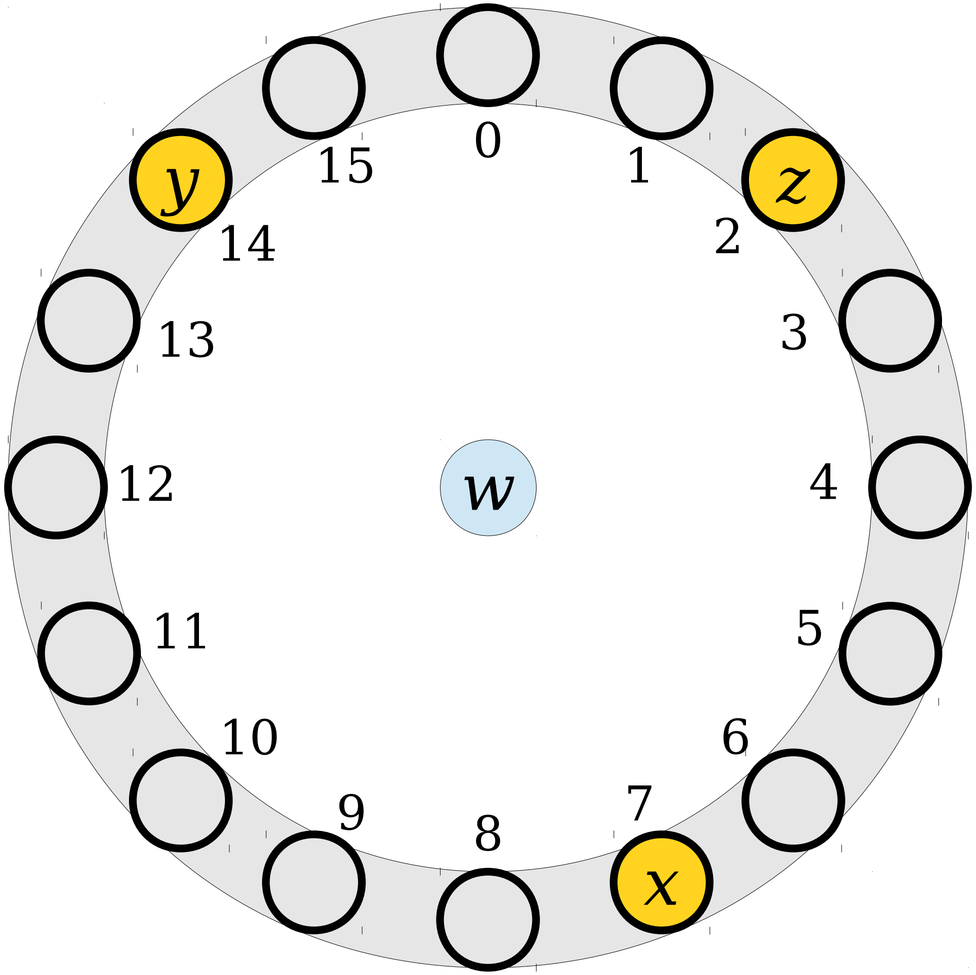
Linear Probing

- To insert an element, compute its hash code and try to place it at the slot with that number.



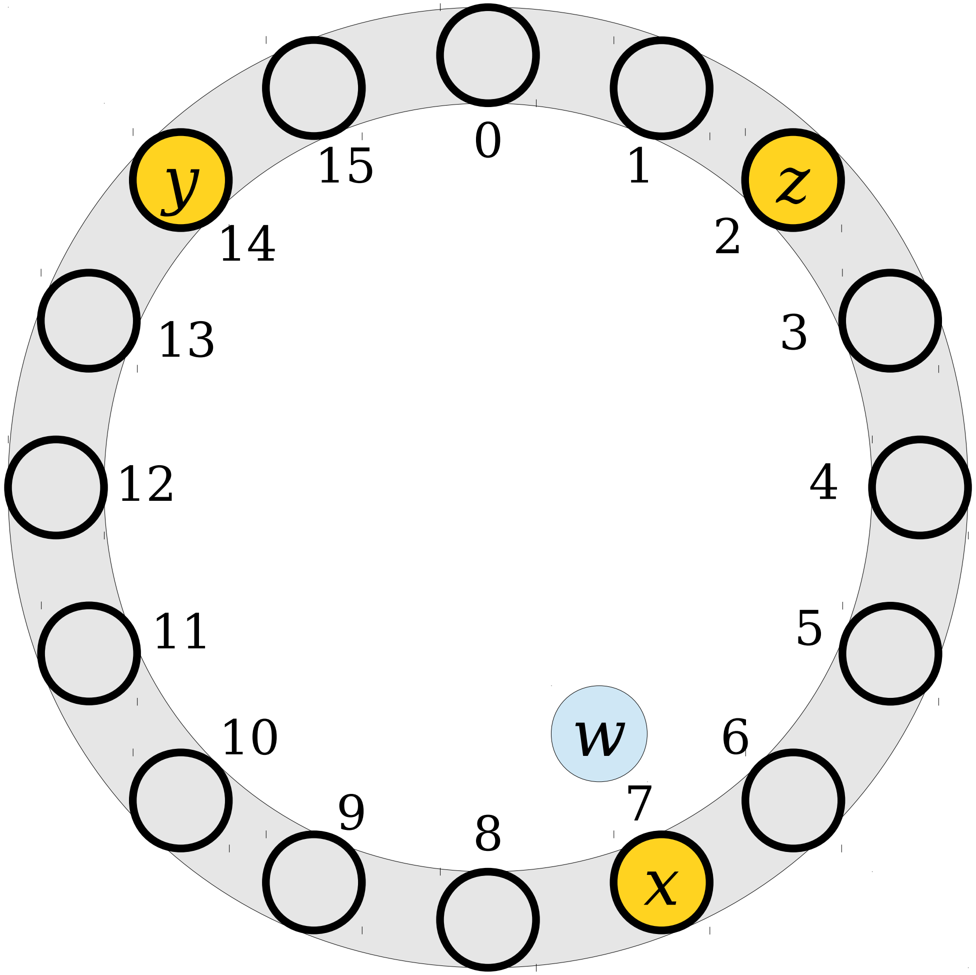
Linear Probing

- To insert an element, compute its hash code and try to place it at the slot with that number.



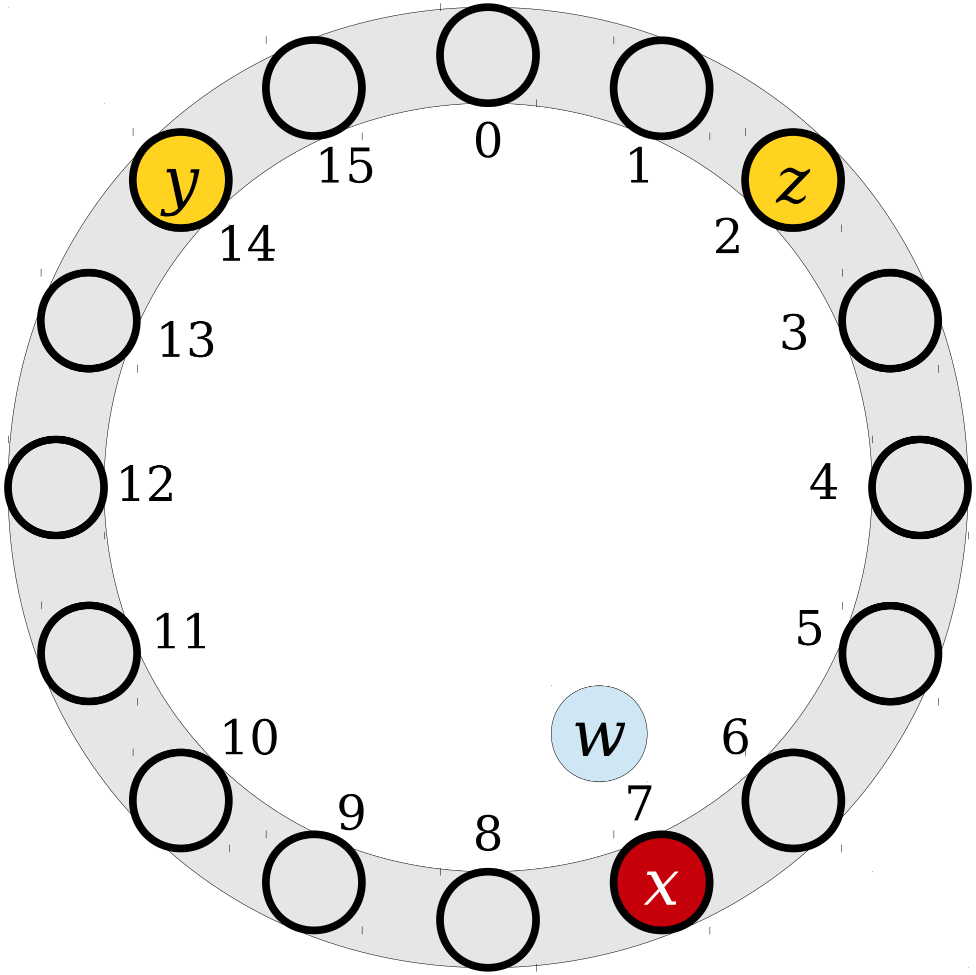
Linear Probing

- To insert an element, compute its hash code and try to place it at the slot with that number.



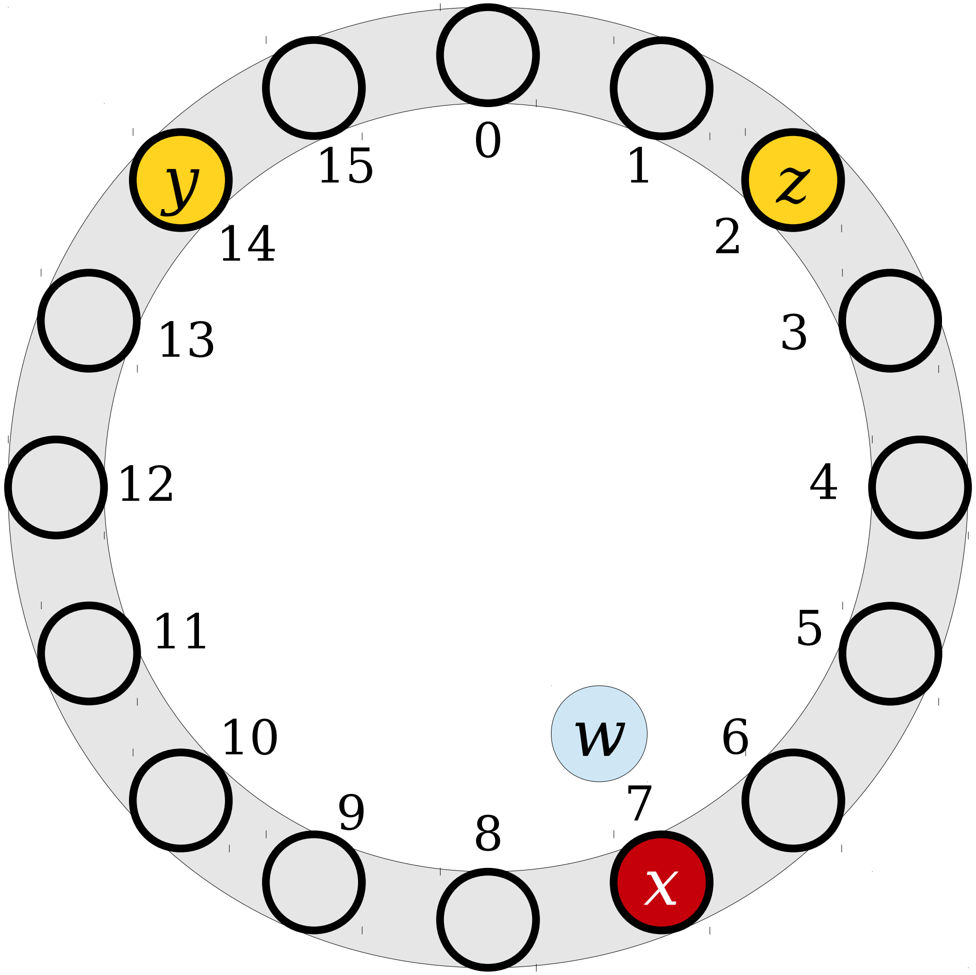
Linear Probing

- To insert an element, compute its hash code and try to place it at the slot with that number.



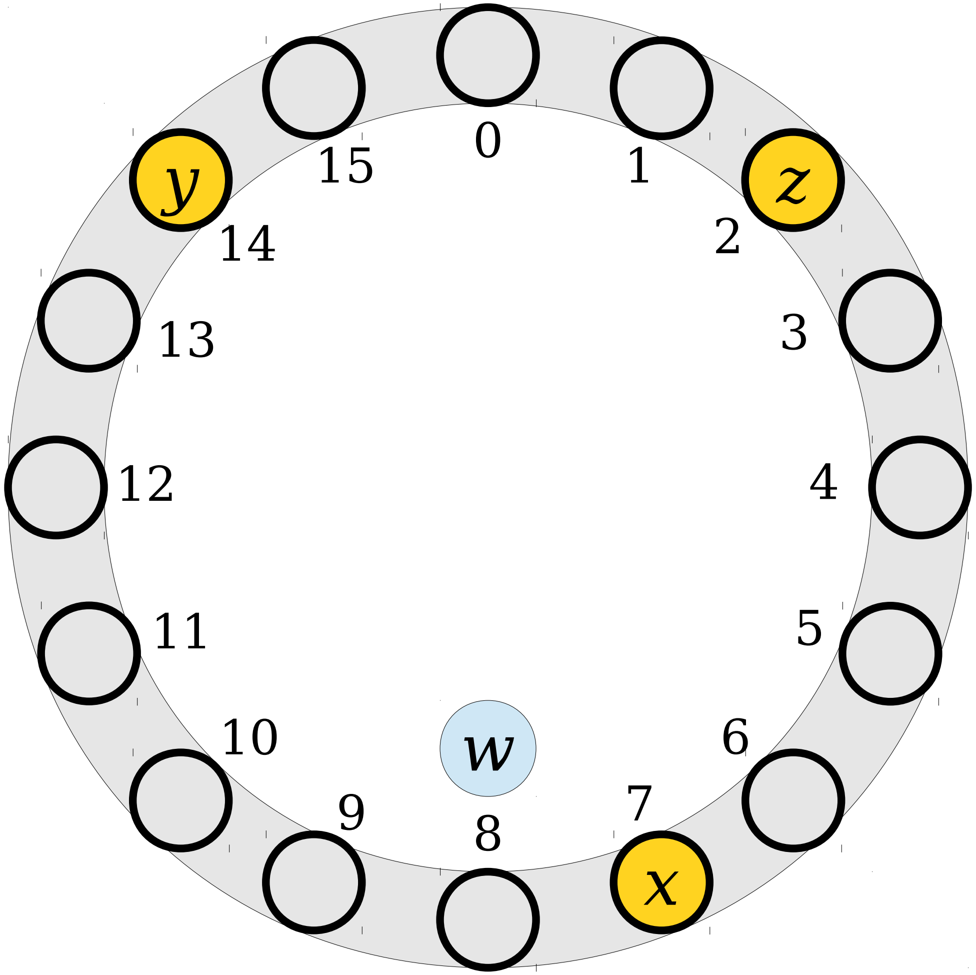
Linear Probing

- To insert an element, compute its hash code and try to place it at the slot with that number.
- If that spot is occupied, keep moving through the array, wrapping around at the end, until a free spot is found.



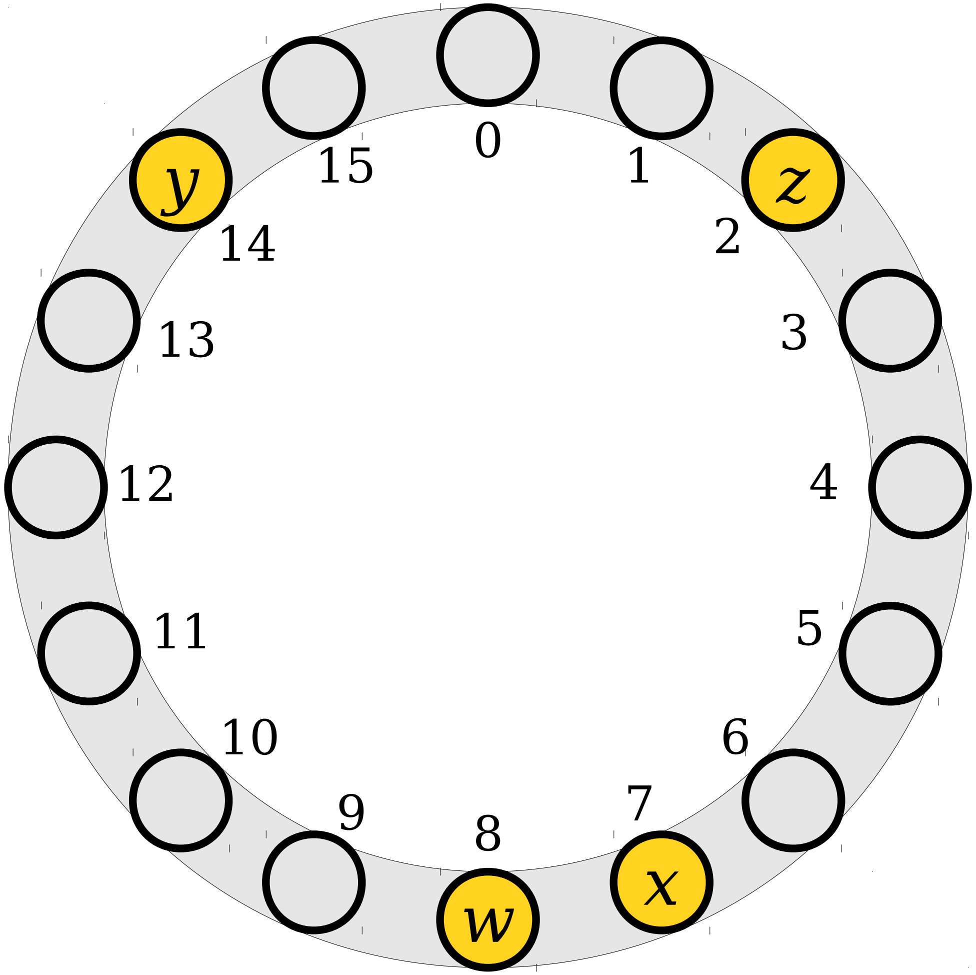
Linear Probing

- To insert an element, compute its hash code and try to place it at the slot with that number.
- If that spot is occupied, keep moving through the array, wrapping around at the end, until a free spot is found.



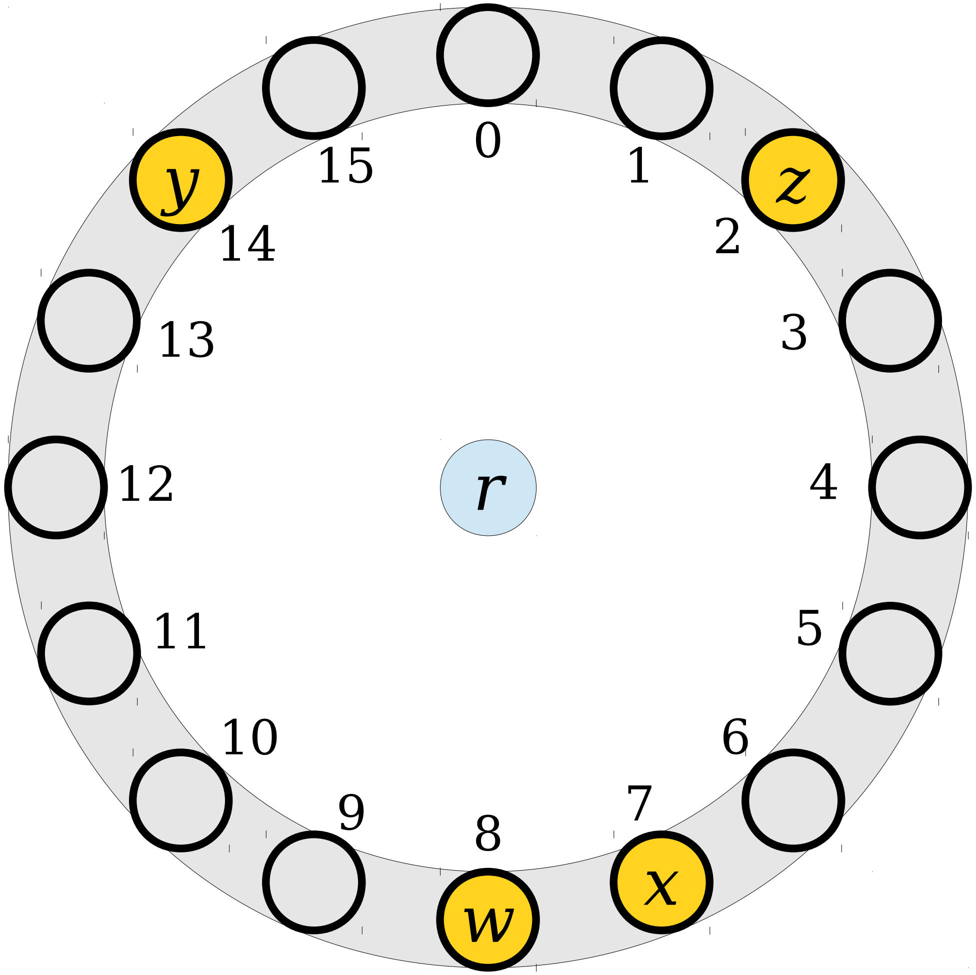
Linear Probing

- To insert an element, compute its hash code and try to place it at the slot with that number.
- If that spot is occupied, keep moving through the array, wrapping around at the end, until a free spot is found.



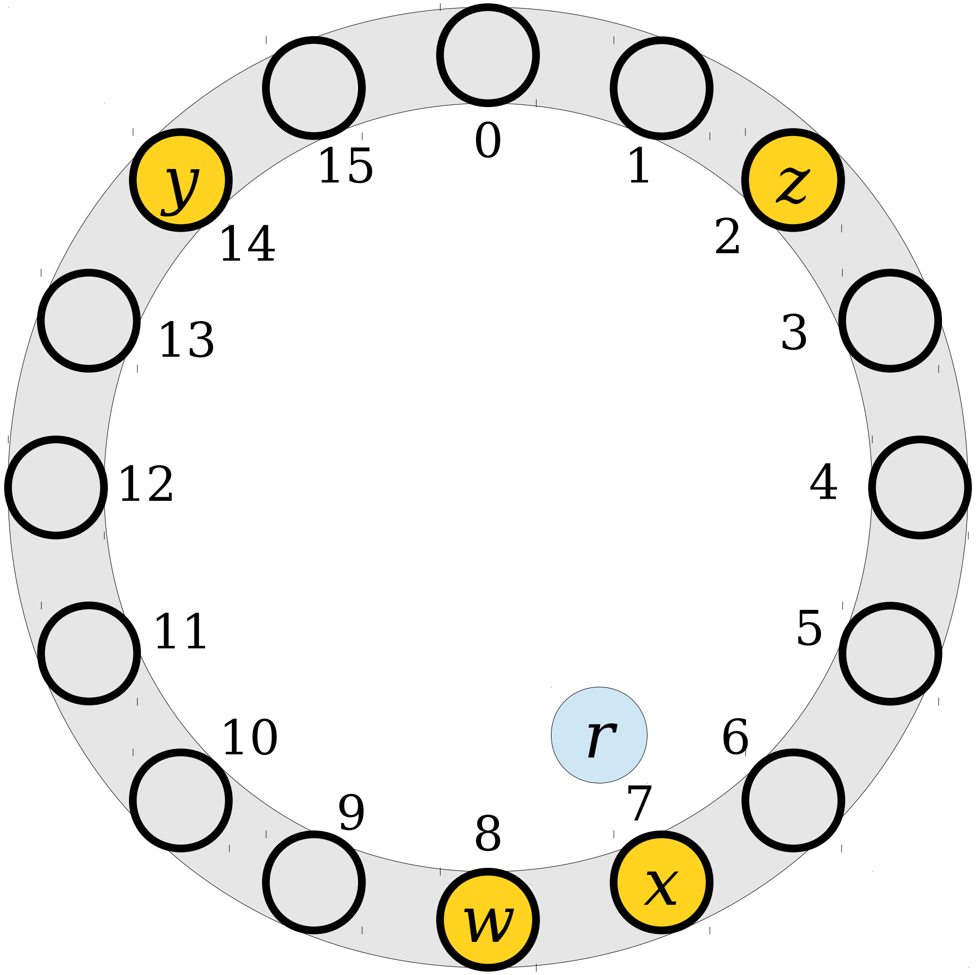
Linear Probing

- To insert an element, compute its hash code and try to place it at the slot with that number.
- If that spot is occupied, keep moving through the array, wrapping around at the end, until a free spot is found.



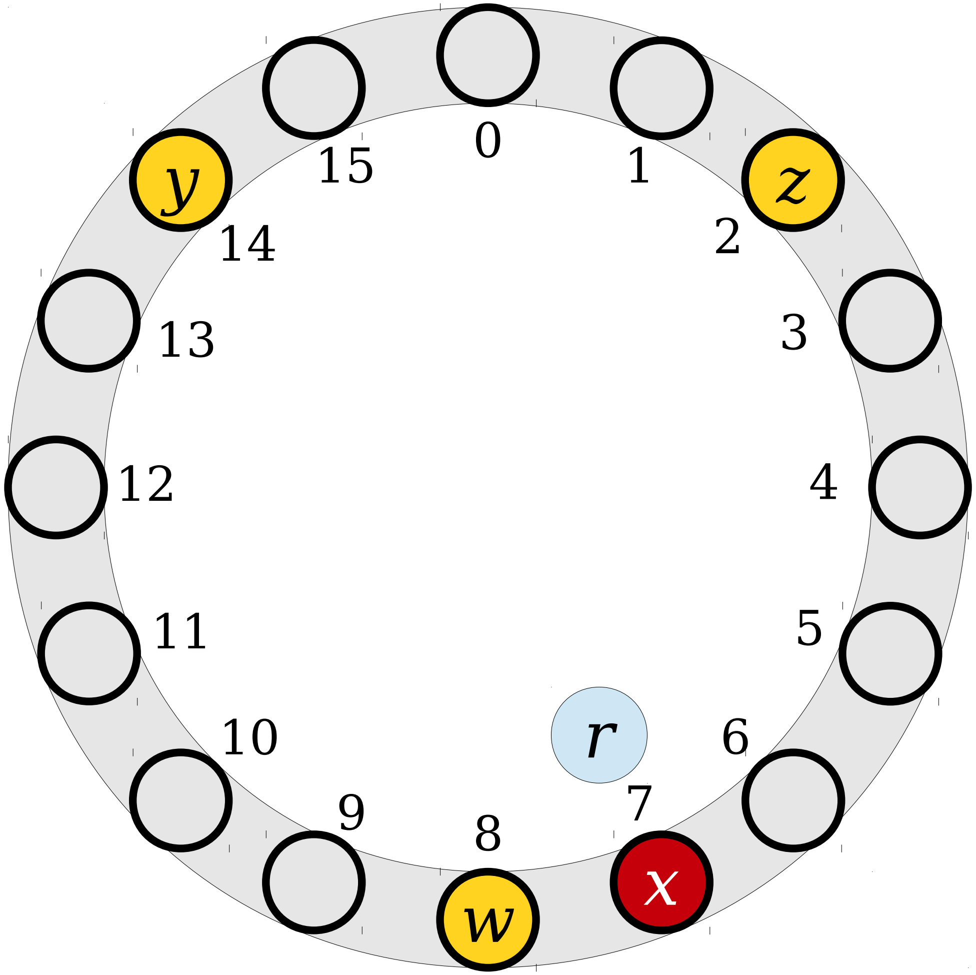
Linear Probing

- To insert an element, compute its hash code and try to place it at the slot with that number.
- If that spot is occupied, keep moving through the array, wrapping around at the end, until a free spot is found.



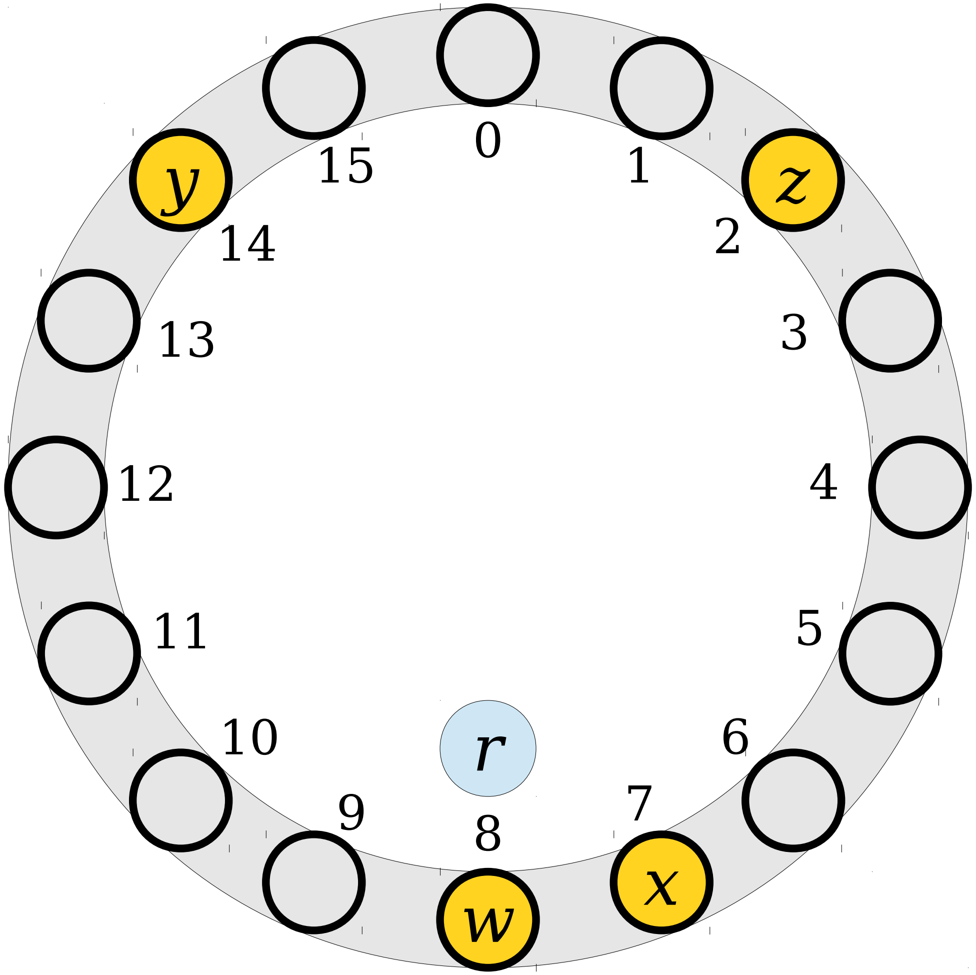
Linear Probing

- To insert an element, compute its hash code and try to place it at the slot with that number.
- If that spot is occupied, keep moving through the array, wrapping around at the end, until a free spot is found.



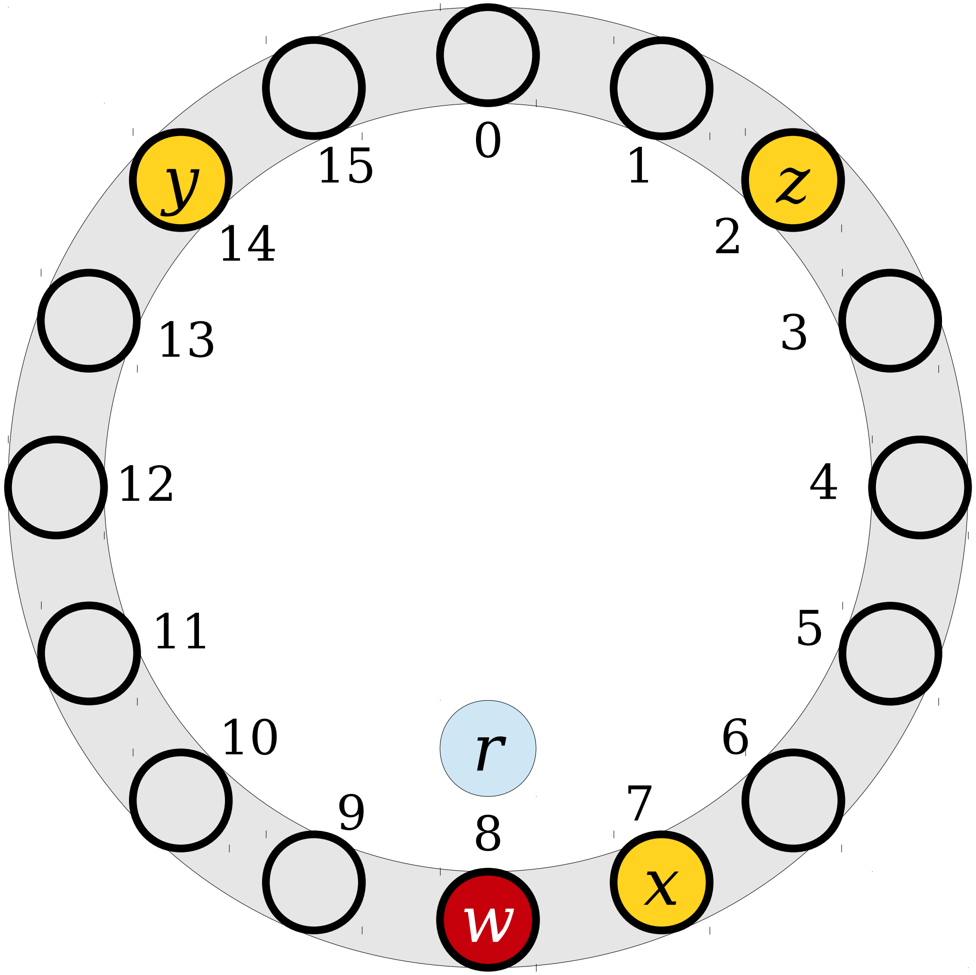
Linear Probing

- To insert an element, compute its hash code and try to place it at the slot with that number.
- If that spot is occupied, keep moving through the array, wrapping around at the end, until a free spot is found.



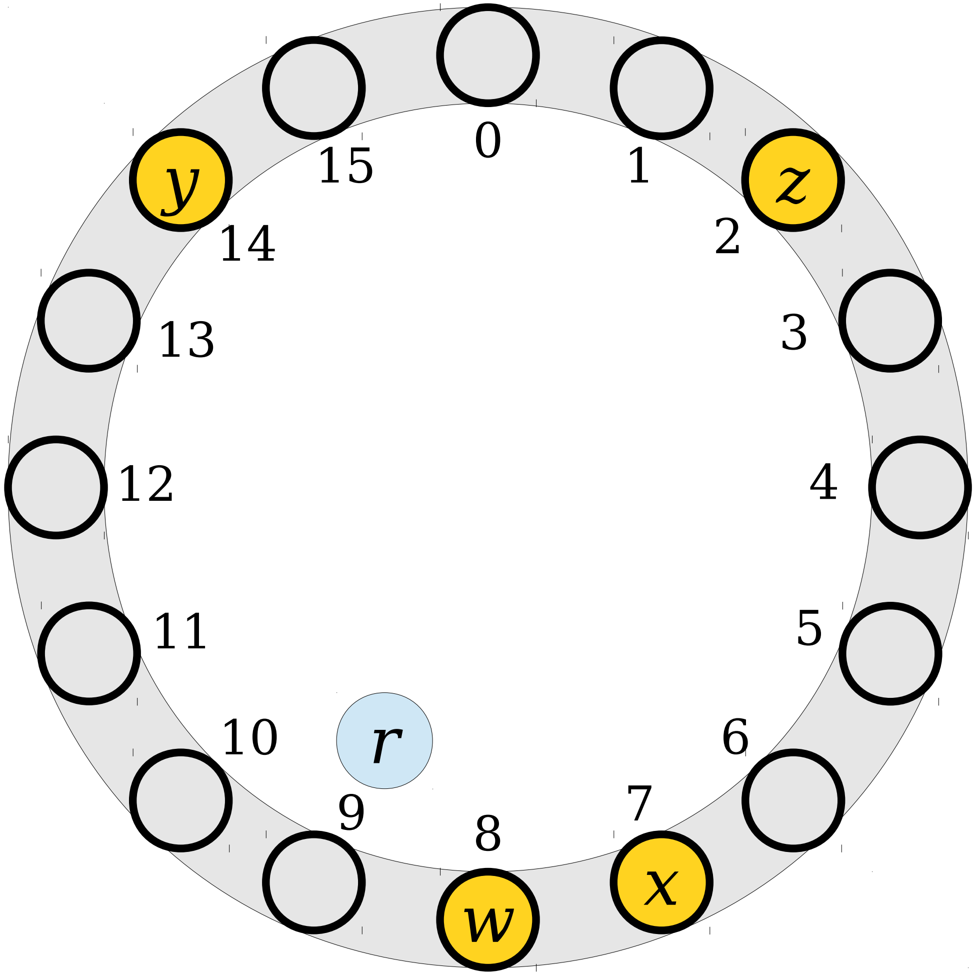
Linear Probing

- To insert an element, compute its hash code and try to place it at the slot with that number.
- If that spot is occupied, keep moving through the array, wrapping around at the end, until a free spot is found.



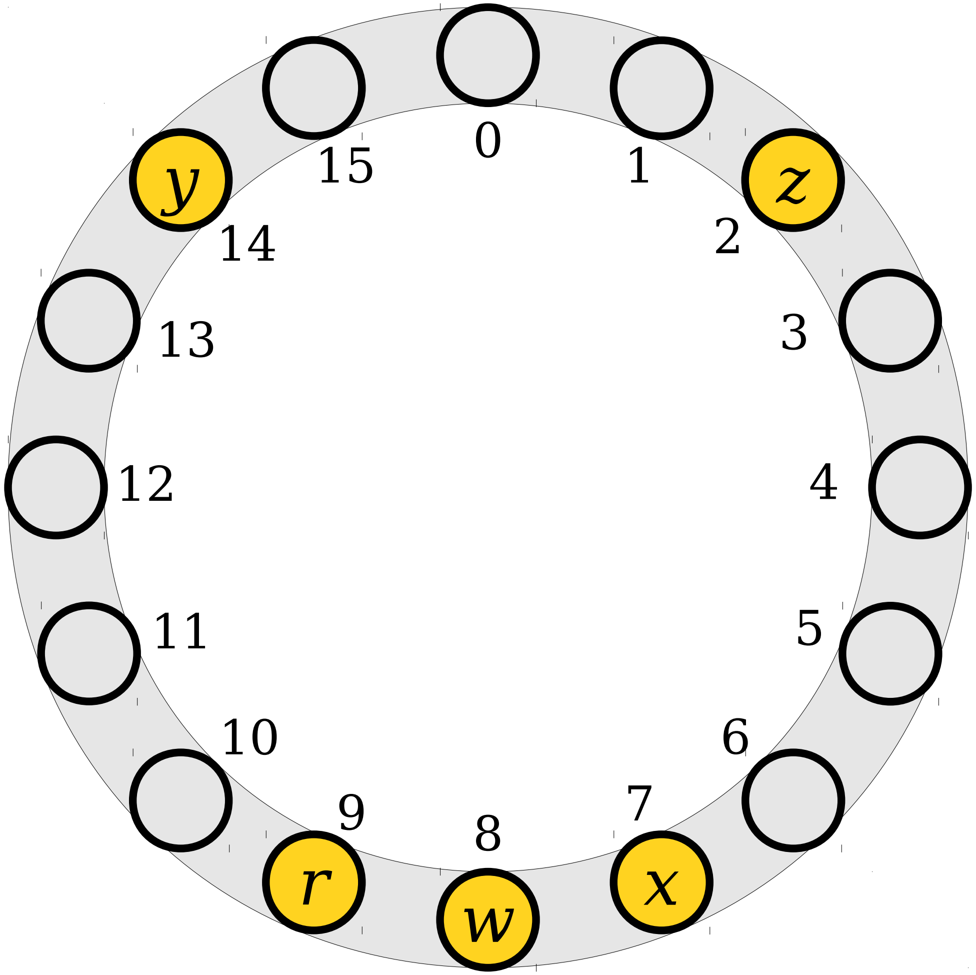
Linear Probing

- To insert an element, compute its hash code and try to place it at the slot with that number.
- If that spot is occupied, keep moving through the array, wrapping around at the end, until a free spot is found.



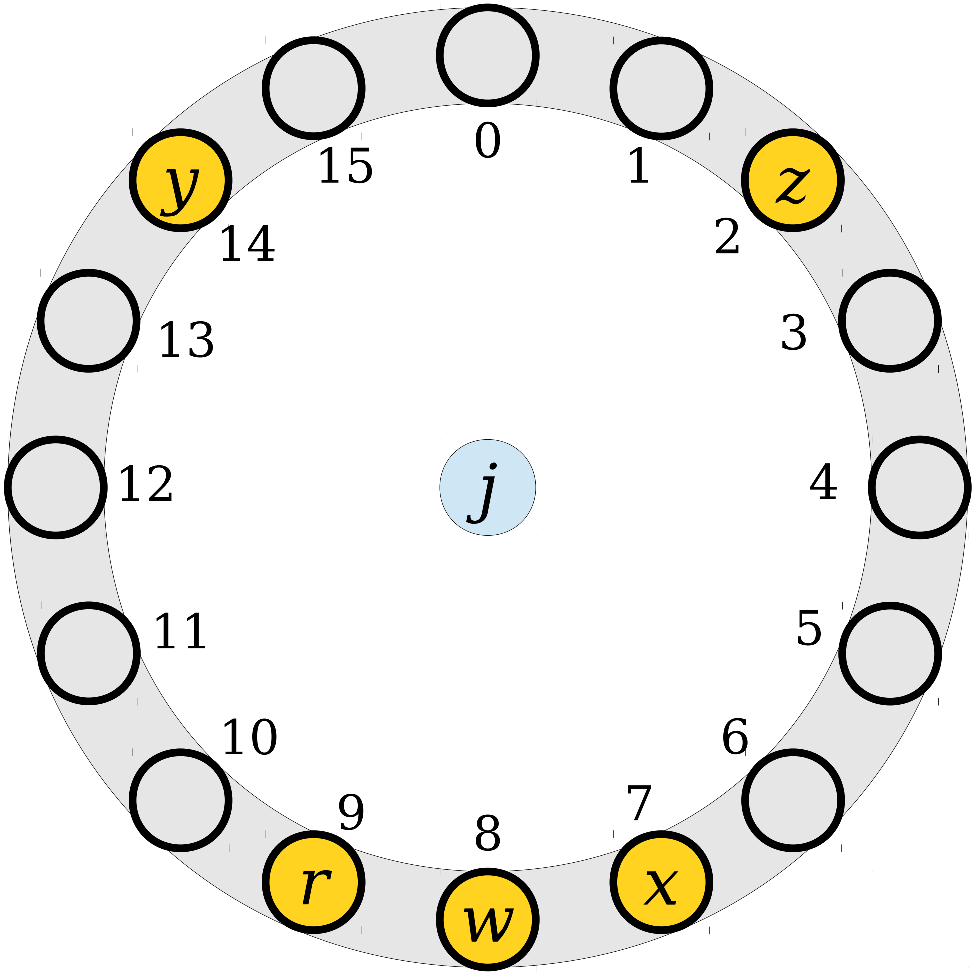
Linear Probing

- To insert an element, compute its hash code and try to place it at the slot with that number.
- If that spot is occupied, keep moving through the array, wrapping around at the end, until a free spot is found.



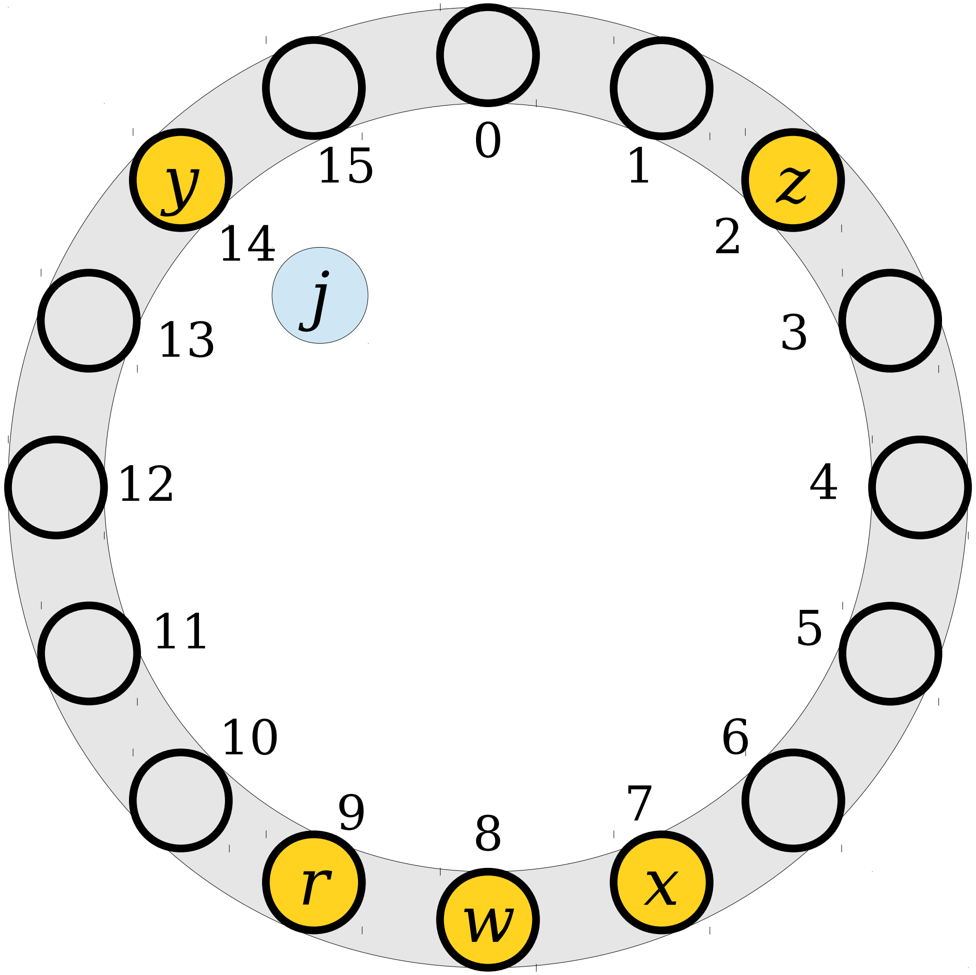
Linear Probing

- To insert an element, compute its hash code and try to place it at the slot with that number.
- If that spot is occupied, keep moving through the array, wrapping around at the end, until a free spot is found.



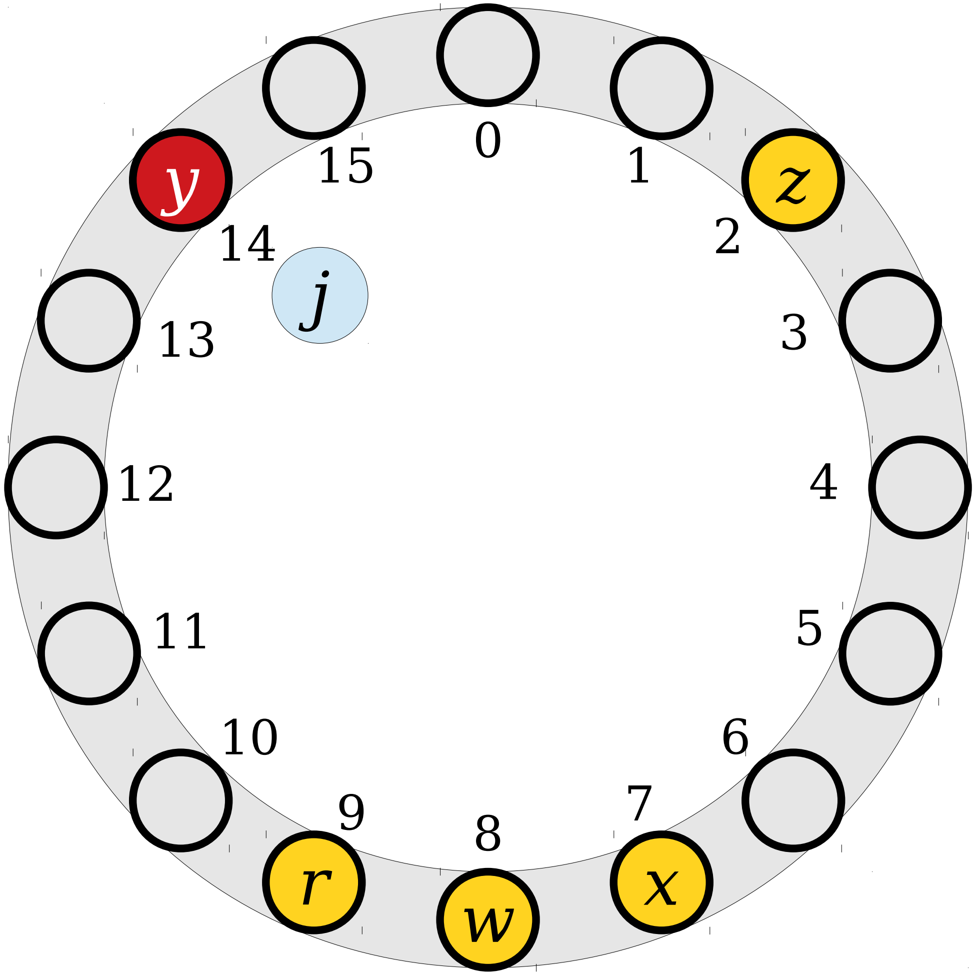
Linear Probing

- To insert an element, compute its hash code and try to place it at the slot with that number.
- If that spot is occupied, keep moving through the array, wrapping around at the end, until a free spot is found.



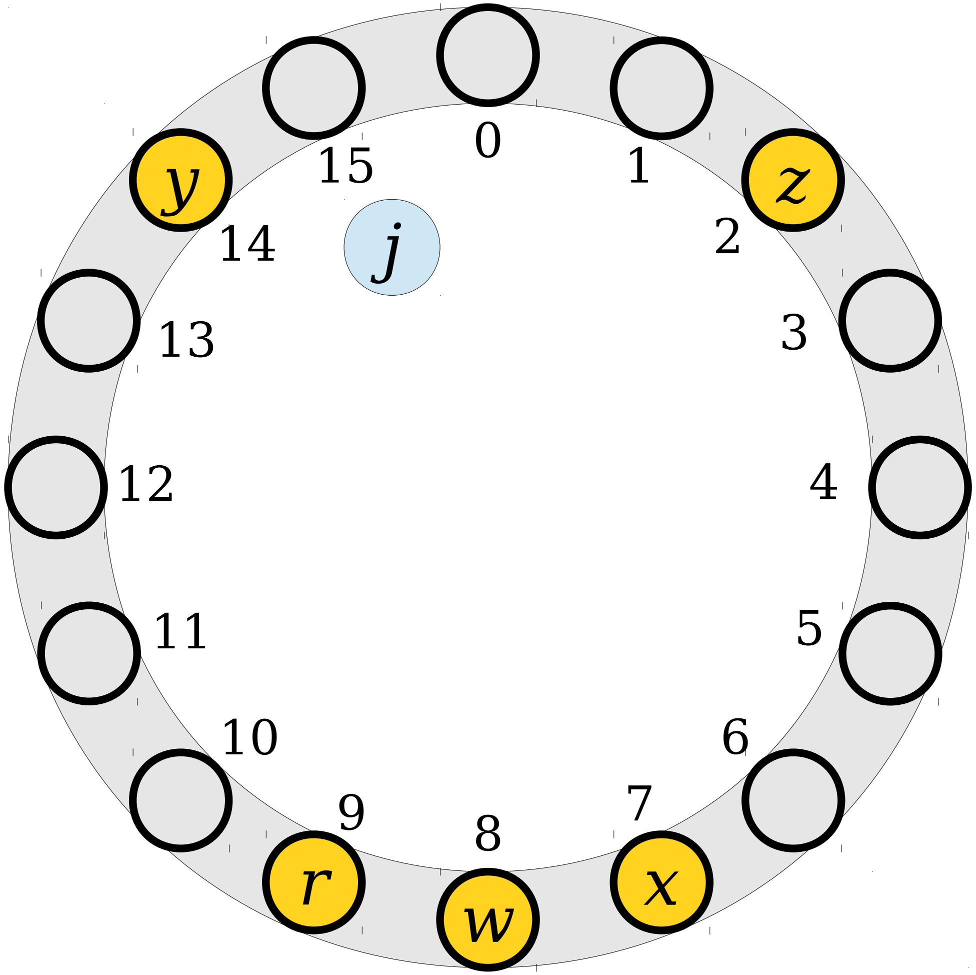
Linear Probing

- To insert an element, compute its hash code and try to place it at the slot with that number.
- If that spot is occupied, keep moving through the array, wrapping around at the end, until a free spot is found.



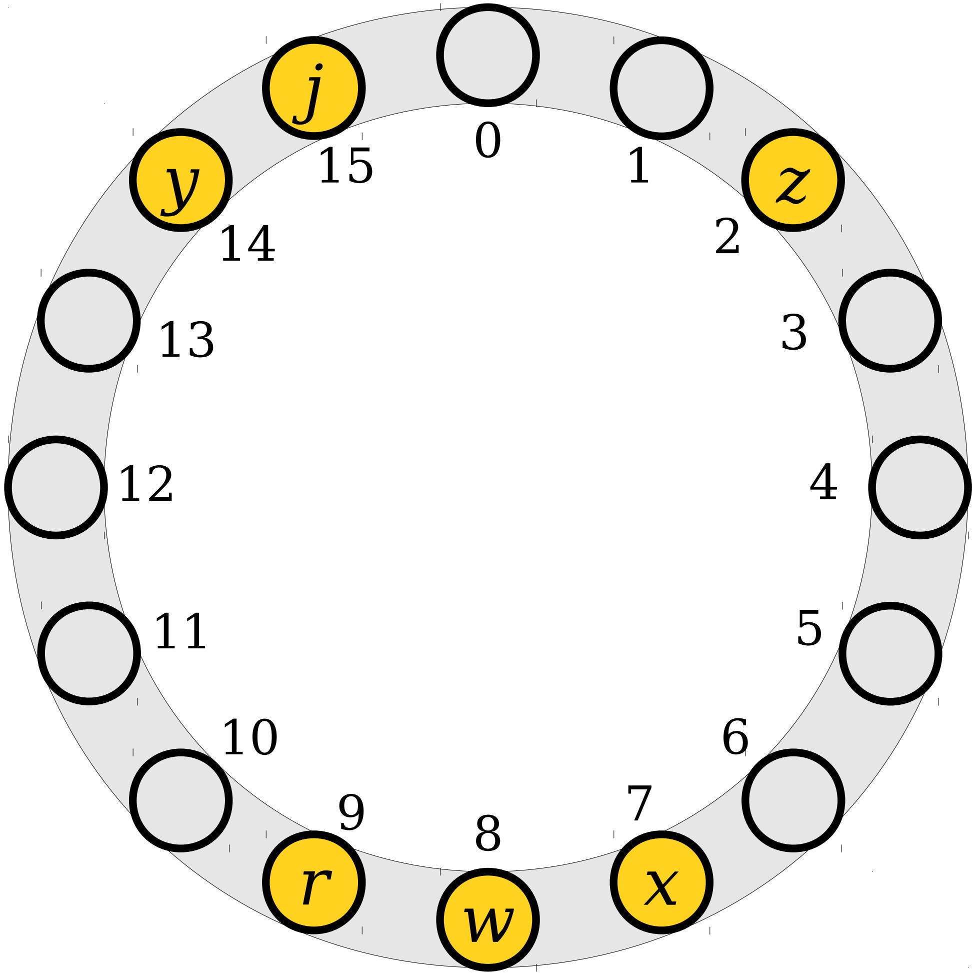
Linear Probing

- To insert an element, compute its hash code and try to place it at the slot with that number.
- If that spot is occupied, keep moving through the array, wrapping around at the end, until a free spot is found.



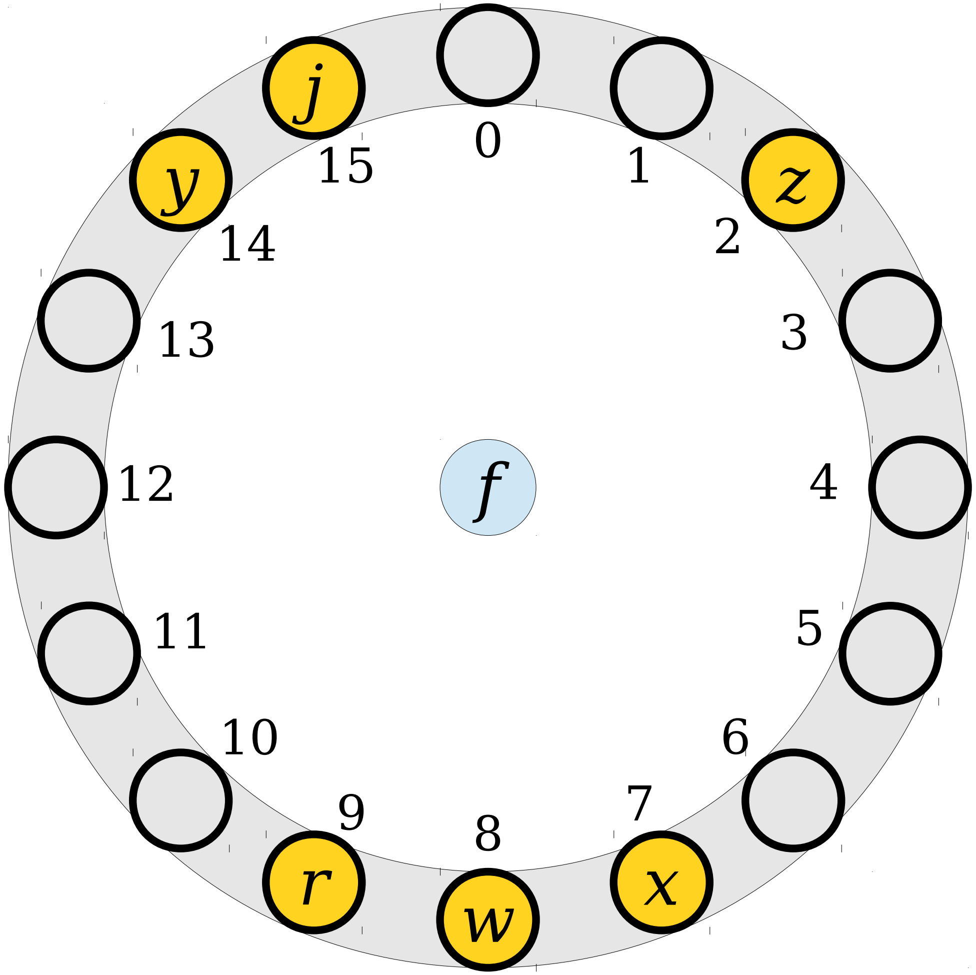
Linear Probing

- To insert an element, compute its hash code and try to place it at the slot with that number.
- If that spot is occupied, keep moving through the array, wrapping around at the end, until a free spot is found.



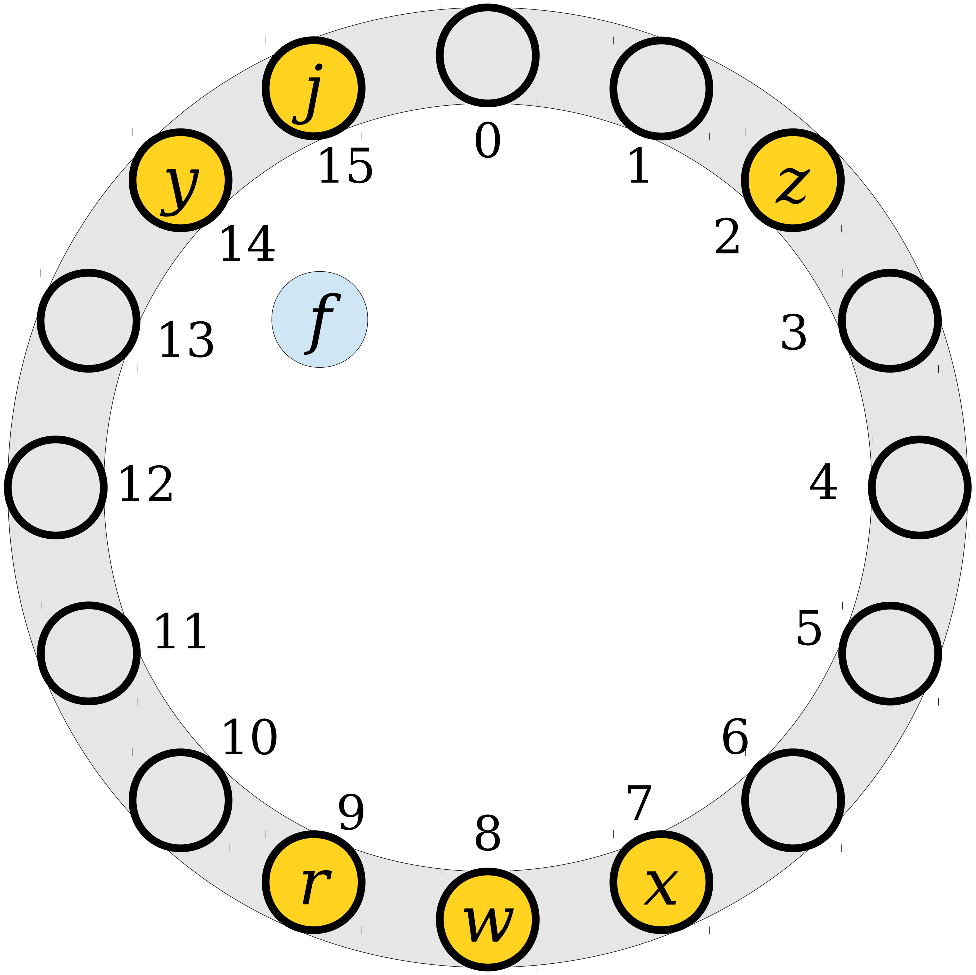
Linear Probing

- To insert an element, compute its hash code and try to place it at the slot with that number.
- If that spot is occupied, keep moving through the array, wrapping around at the end, until a free spot is found.



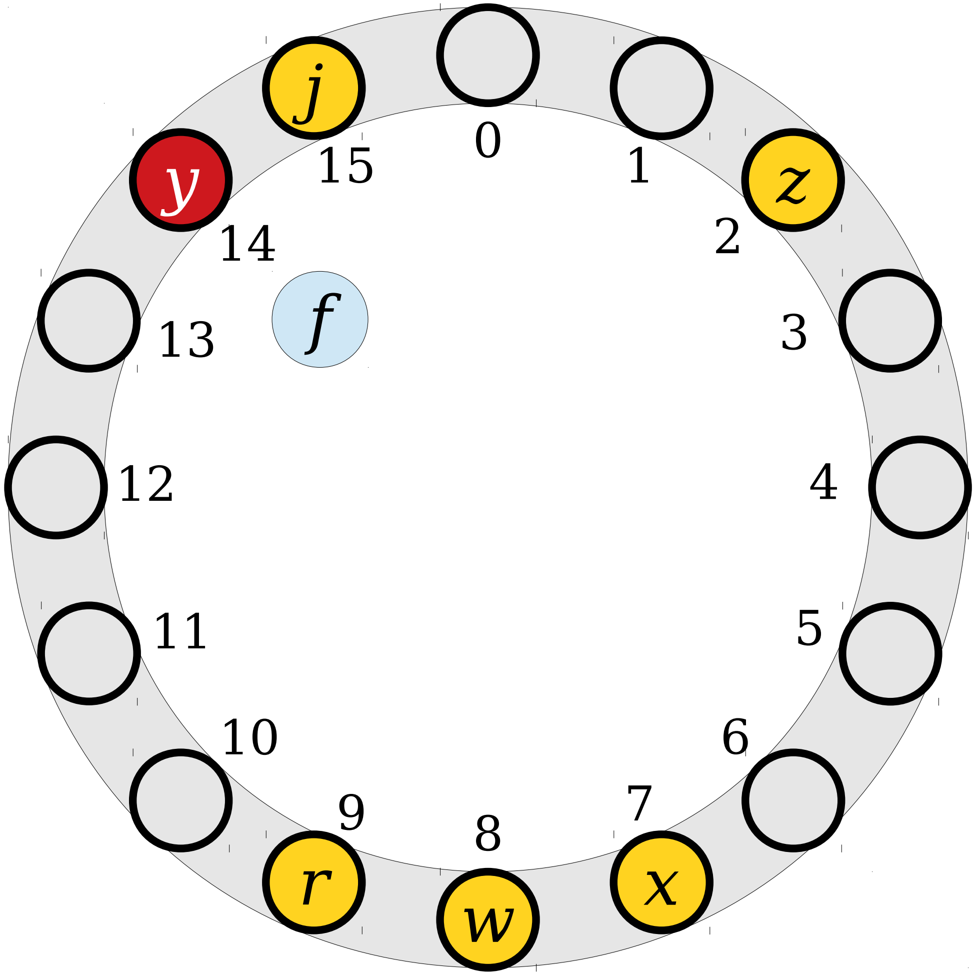
Linear Probing

- To insert an element, compute its hash code and try to place it at the slot with that number.
- If that spot is occupied, keep moving through the array, wrapping around at the end, until a free spot is found.



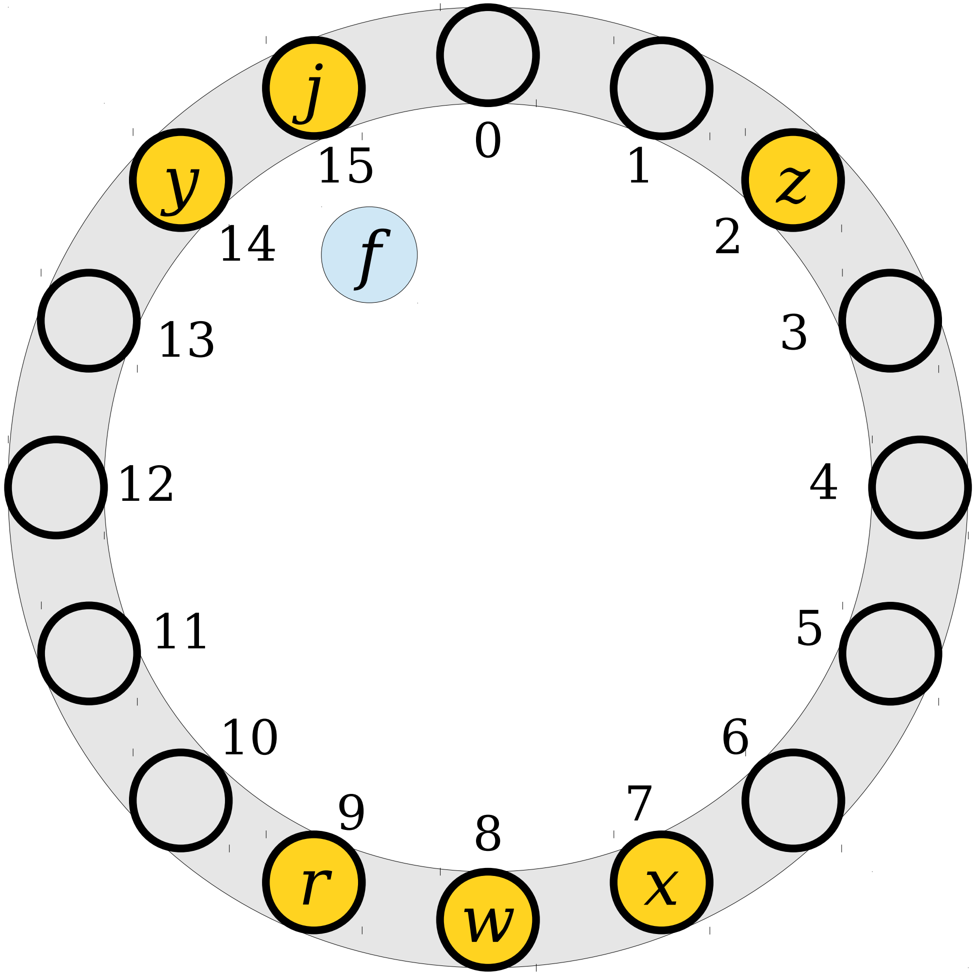
Linear Probing

- To insert an element, compute its hash code and try to place it at the slot with that number.
- If that spot is occupied, keep moving through the array, wrapping around at the end, until a free spot is found.



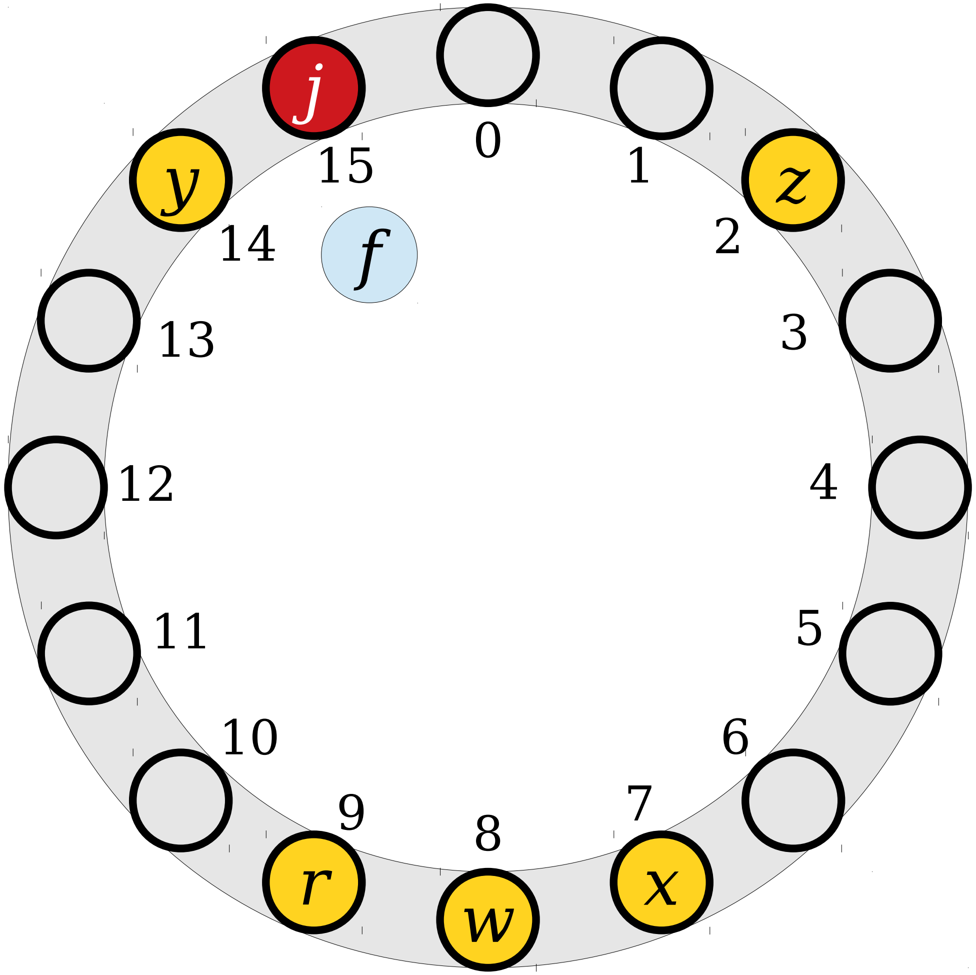
Linear Probing

- To insert an element, compute its hash code and try to place it at the slot with that number.
- If that spot is occupied, keep moving through the array, wrapping around at the end, until a free spot is found.



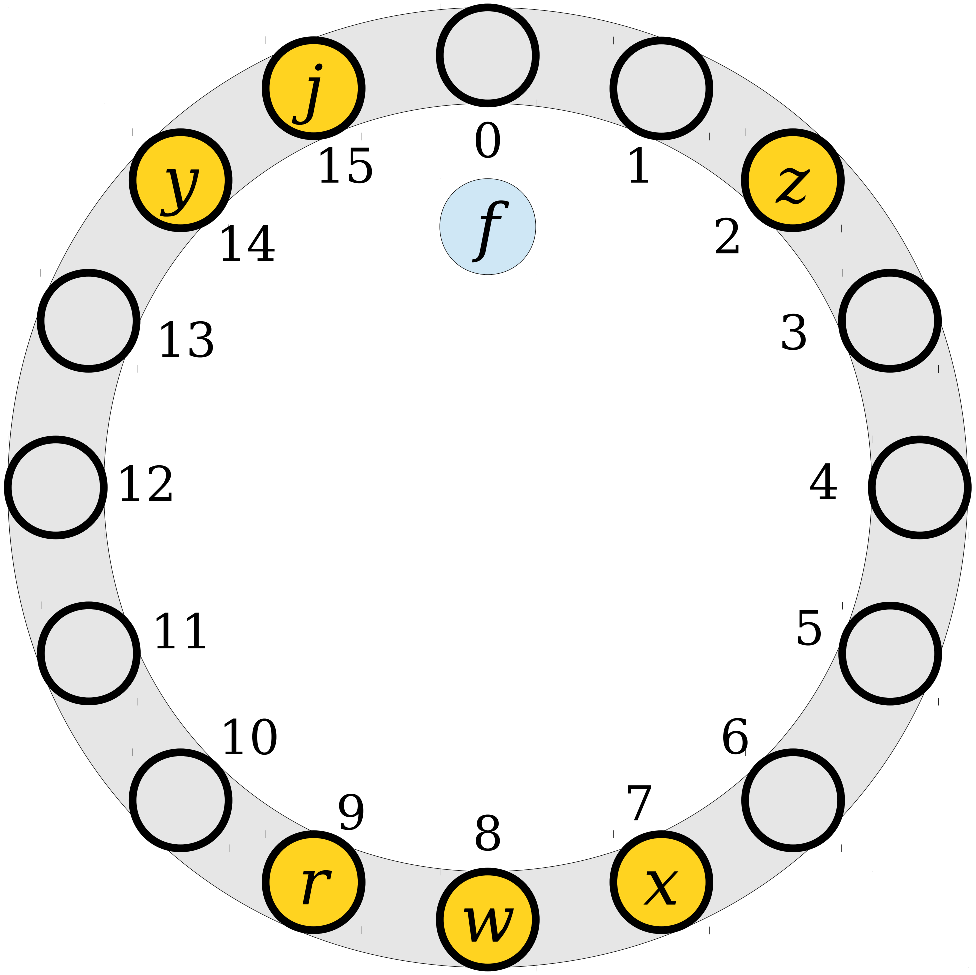
Linear Probing

- To insert an element, compute its hash code and try to place it at the slot with that number.
- If that spot is occupied, keep moving through the array, wrapping around at the end, until a free spot is found.



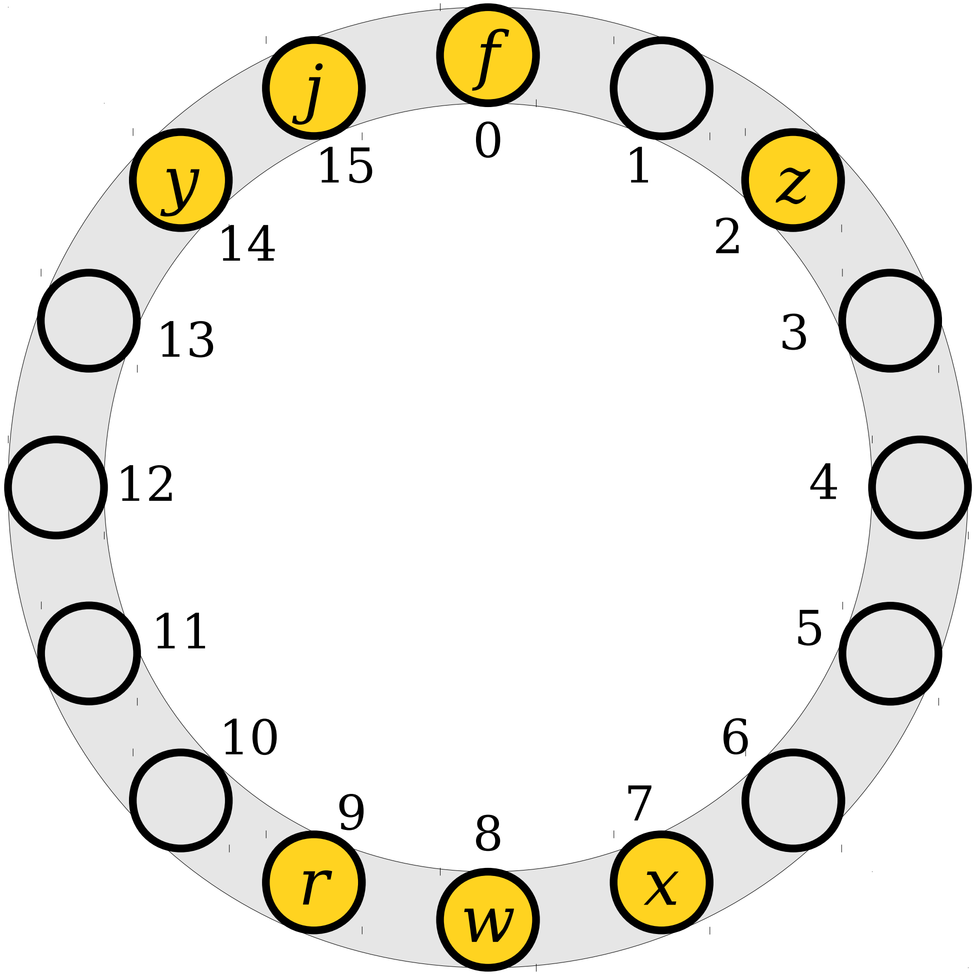
Linear Probing

- To insert an element, compute its hash code and try to place it at the slot with that number.
- If that spot is occupied, keep moving through the array, wrapping around at the end, until a free spot is found.



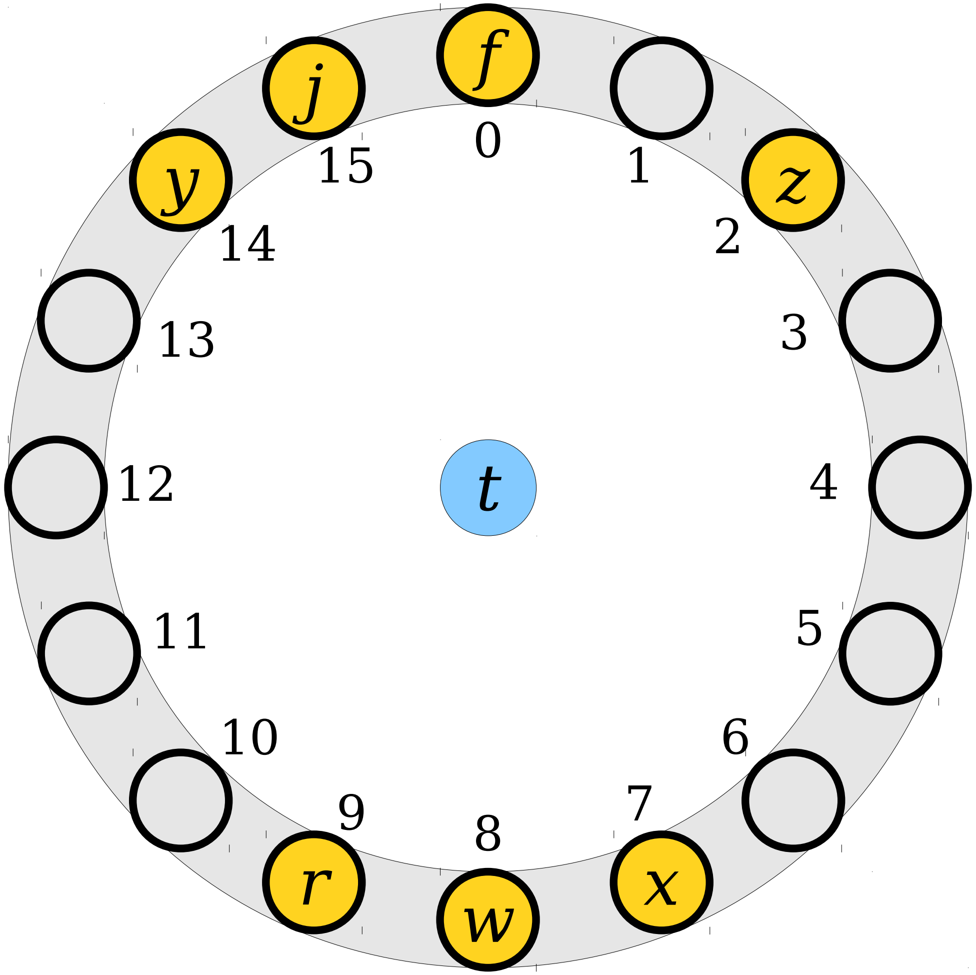
Linear Probing

- To insert an element, compute its hash code and try to place it at the slot with that number.
- If that spot is occupied, keep moving through the array, wrapping around at the end, until a free spot is found.



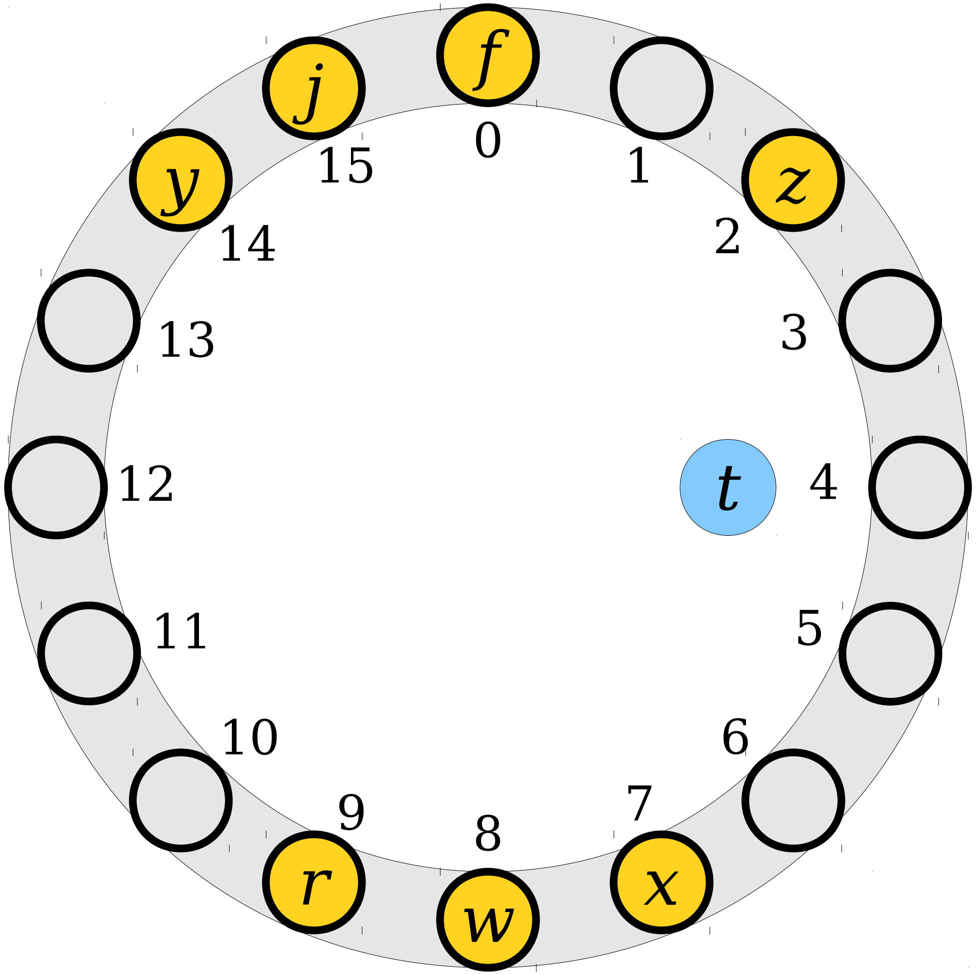
Linear Probing

- To look up an element, compute its hash code and start looking there.
- Move around the ring until either the element is found or a blank spot is detected.
- (If every single slot is full, stop looking after you've tried them all.)



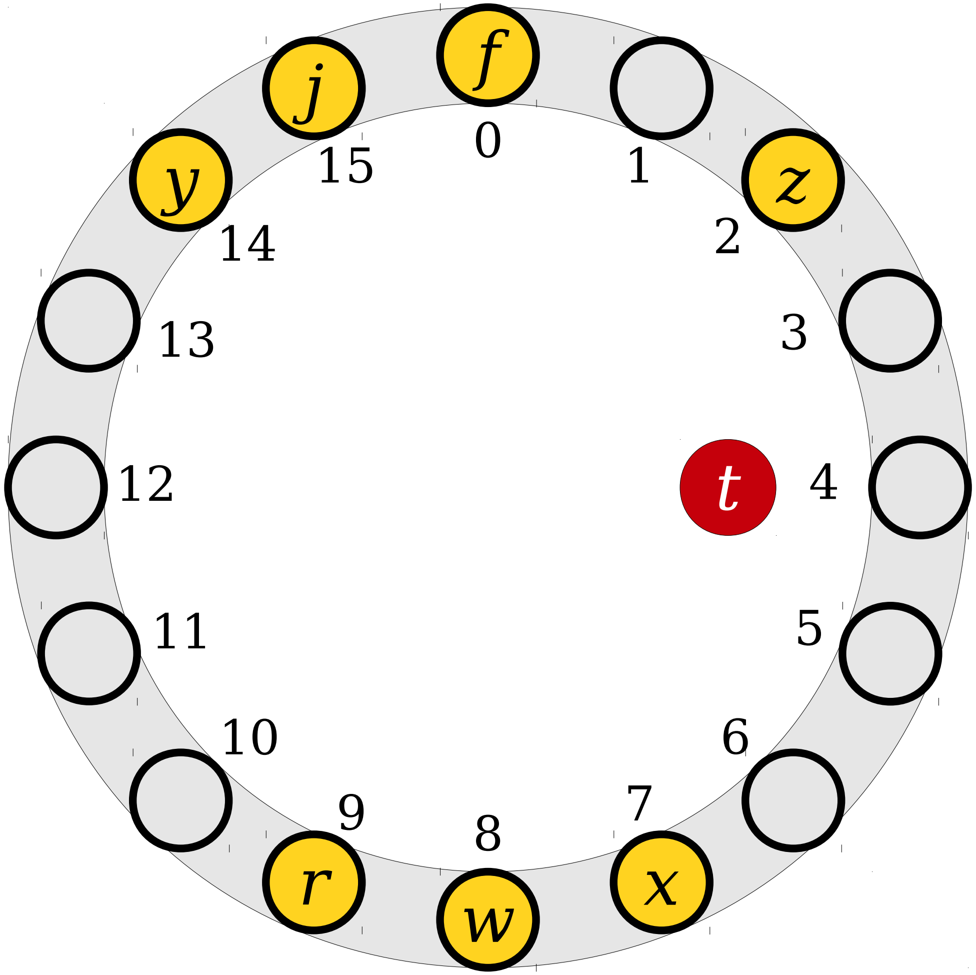
Linear Probing

- To look up an element, compute its hash code and start looking there.
- Move around the ring until either the element is found or a blank spot is detected.
- (If every single slot is full, stop looking after you've tried them all.)



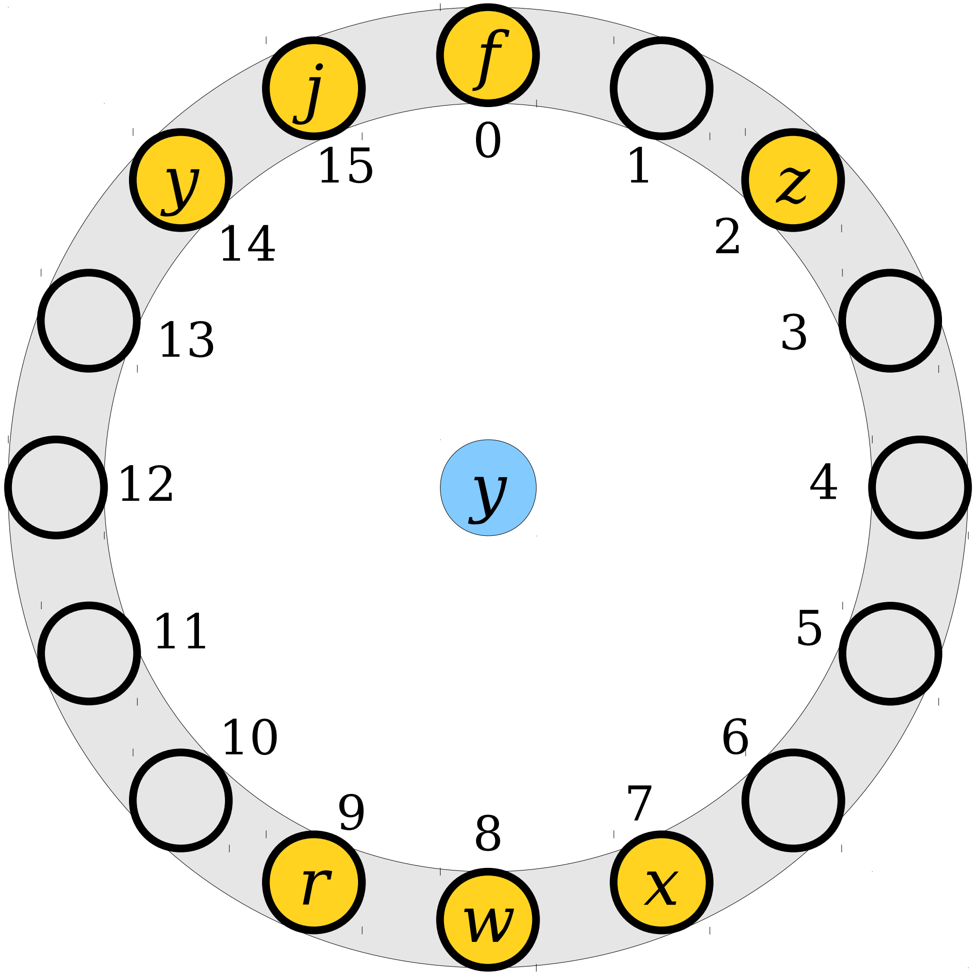
Linear Probing

- To look up an element, compute its hash code and start looking there.
- Move around the ring until either the element is found or a blank spot is detected.
- (If every single slot is full, stop looking after you've tried them all.)



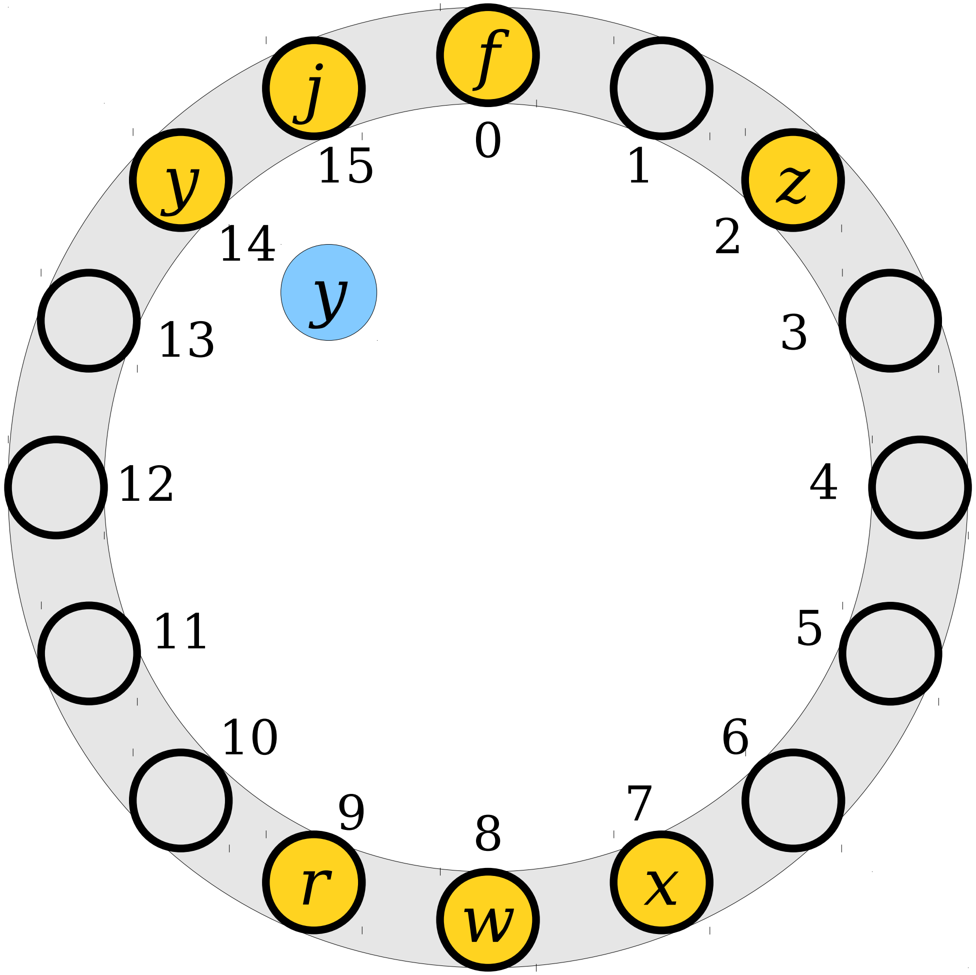
Linear Probing

- To look up an element, compute its hash code and start looking there.
- Move around the ring until either the element is found or a blank spot is detected.
- (If every single slot is full, stop looking after you've tried them all.)



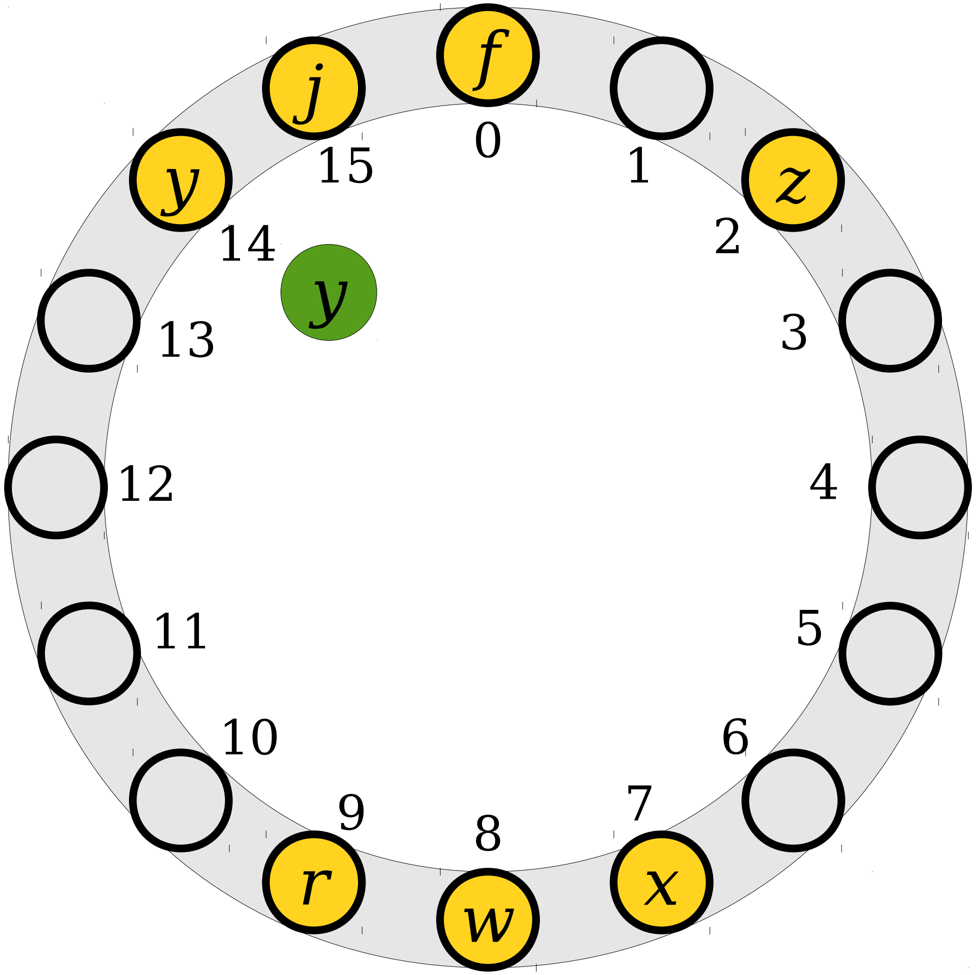
Linear Probing

- To look up an element, compute its hash code and start looking there.
- Move around the ring until either the element is found or a blank spot is detected.
- (If every single slot is full, stop looking after you've tried them all.)



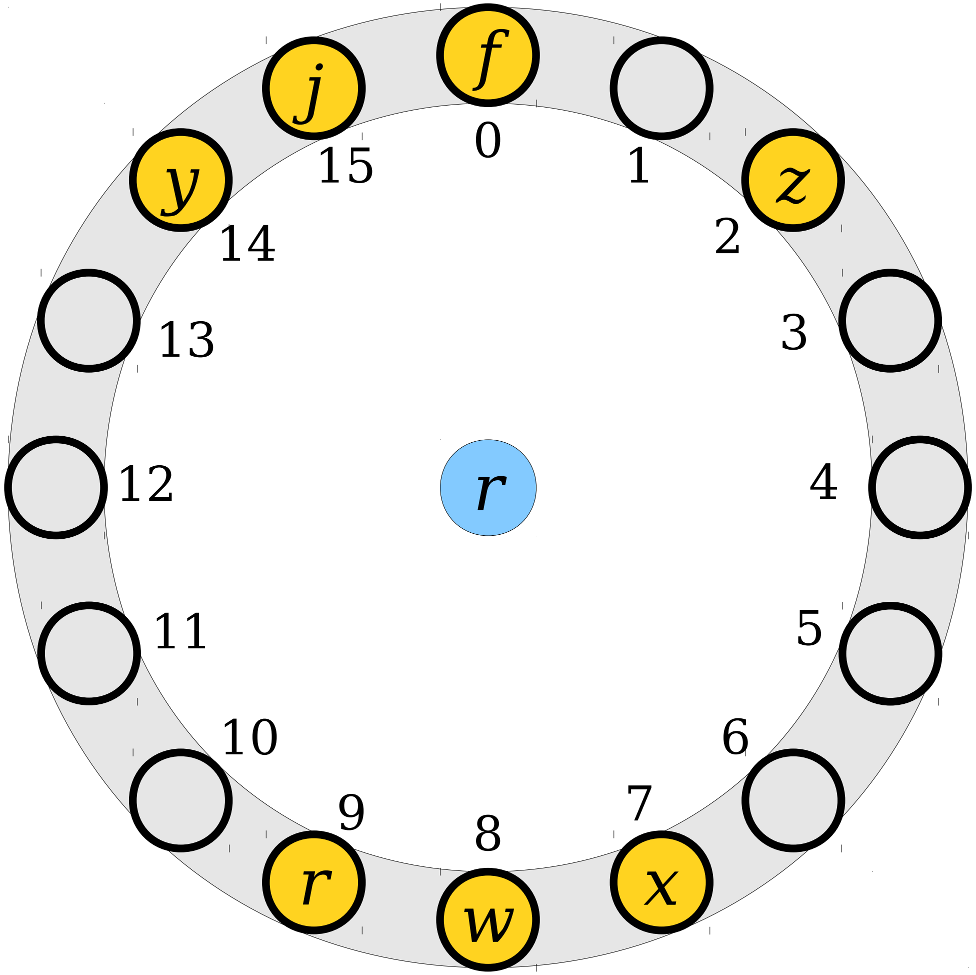
Linear Probing

- To look up an element, compute its hash code and start looking there.
- Move around the ring until either the element is found or a blank spot is detected.
- (If every single slot is full, stop looking after you've tried them all.)



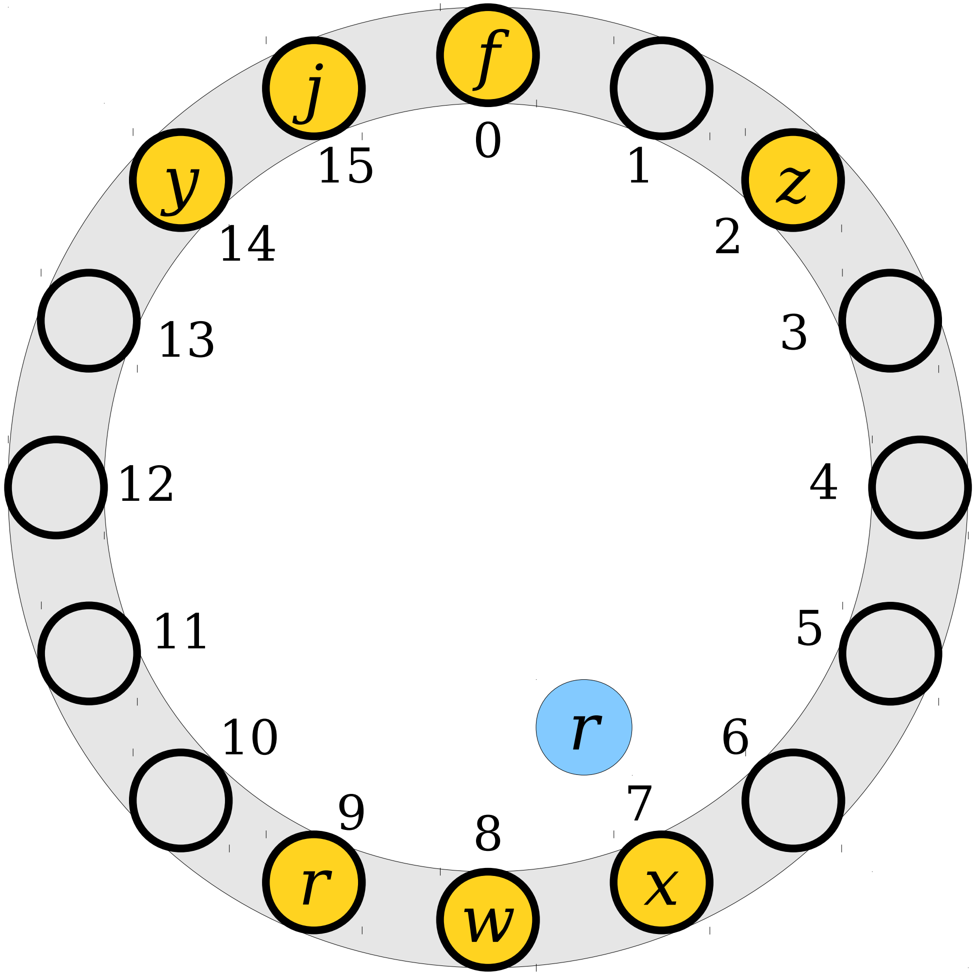
Linear Probing

- To look up an element, compute its hash code and start looking there.
- Move around the ring until either the element is found or a blank spot is detected.
- (If every single slot is full, stop looking after you've tried them all.)



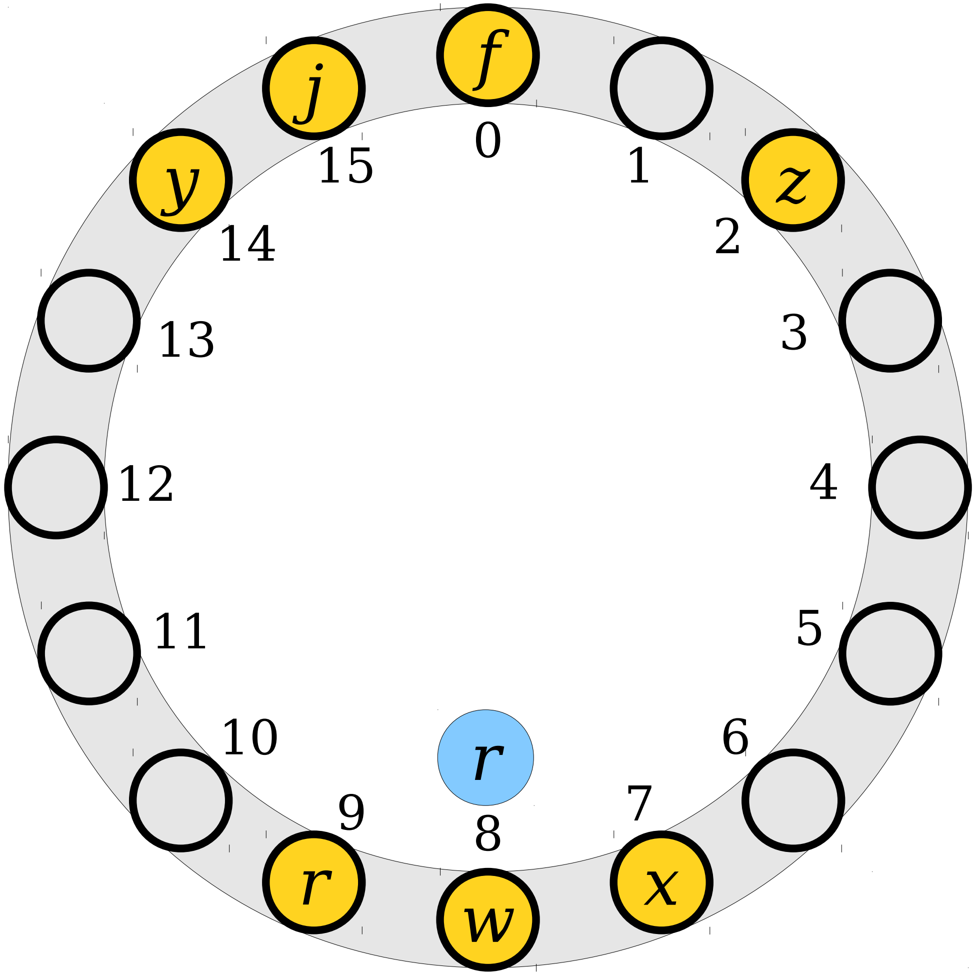
Linear Probing

- To look up an element, compute its hash code and start looking there.
- Move around the ring until either the element is found or a blank spot is detected.
- (If every single slot is full, stop looking after you've tried them all.)



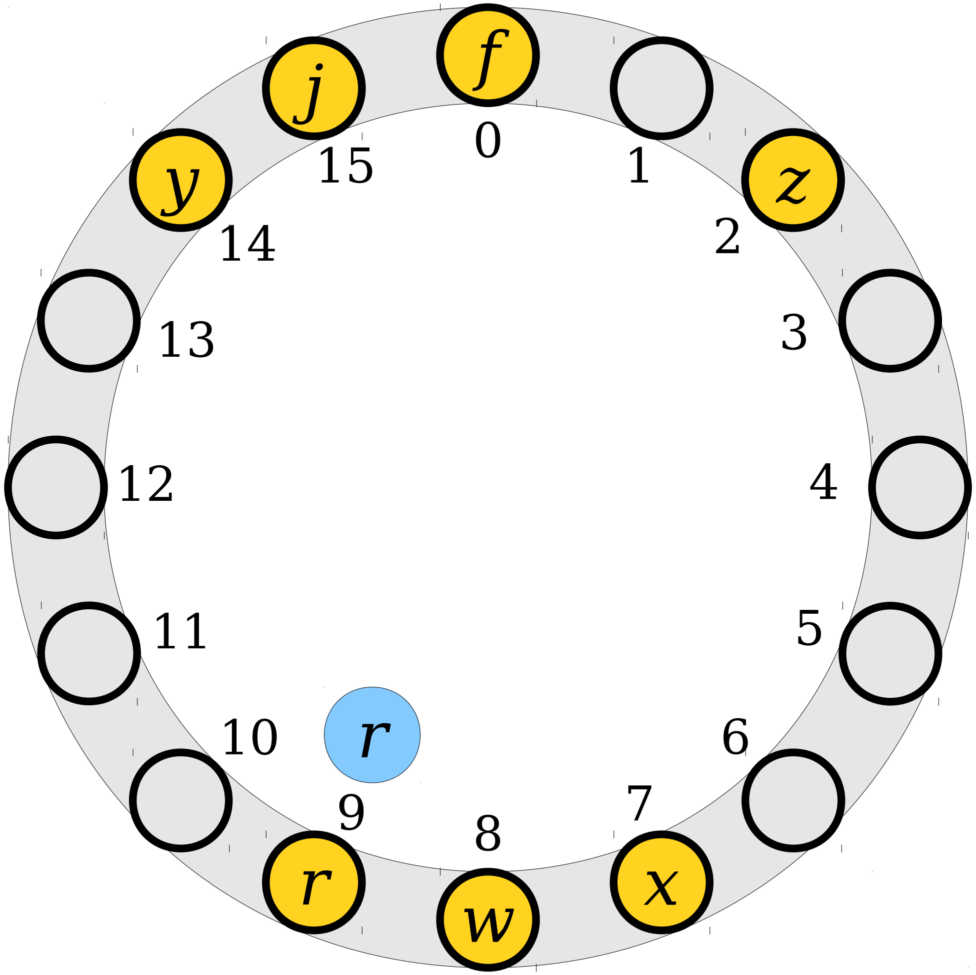
Linear Probing

- To look up an element, compute its hash code and start looking there.
- Move around the ring until either the element is found or a blank spot is detected.
- (If every single slot is full, stop looking after you've tried them all.)



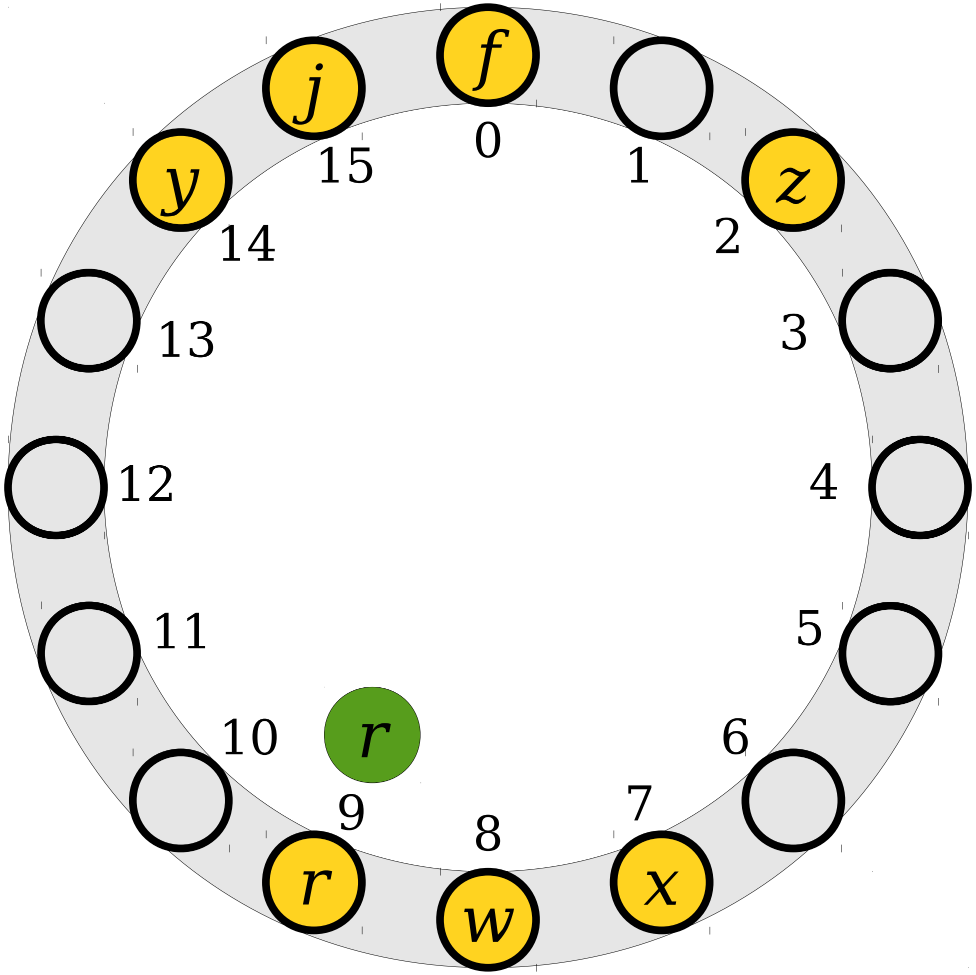
Linear Probing

- To look up an element, compute its hash code and start looking there.
- Move around the ring until either the element is found or a blank spot is detected.
- (If every single slot is full, stop looking after you've tried them all.)



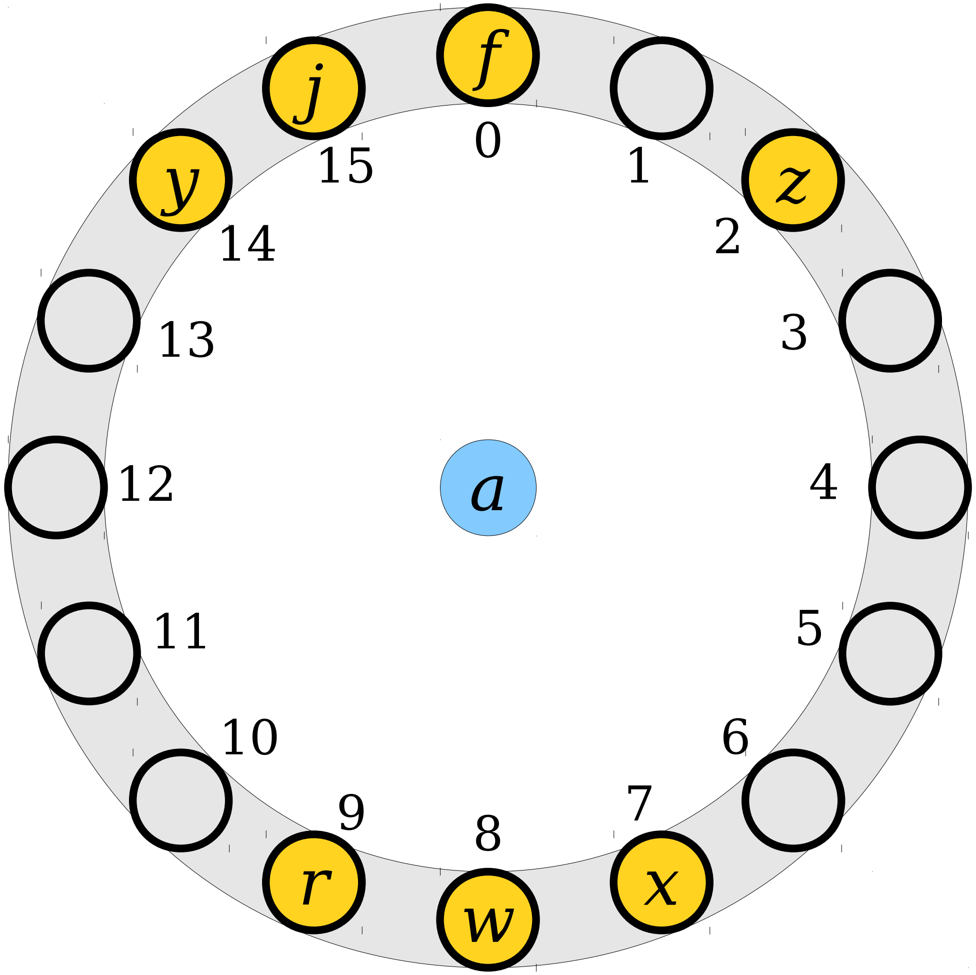
Linear Probing

- To look up an element, compute its hash code and start looking there.
- Move around the ring until either the element is found or a blank spot is detected.
- (If every single slot is full, stop looking after you've tried them all.)



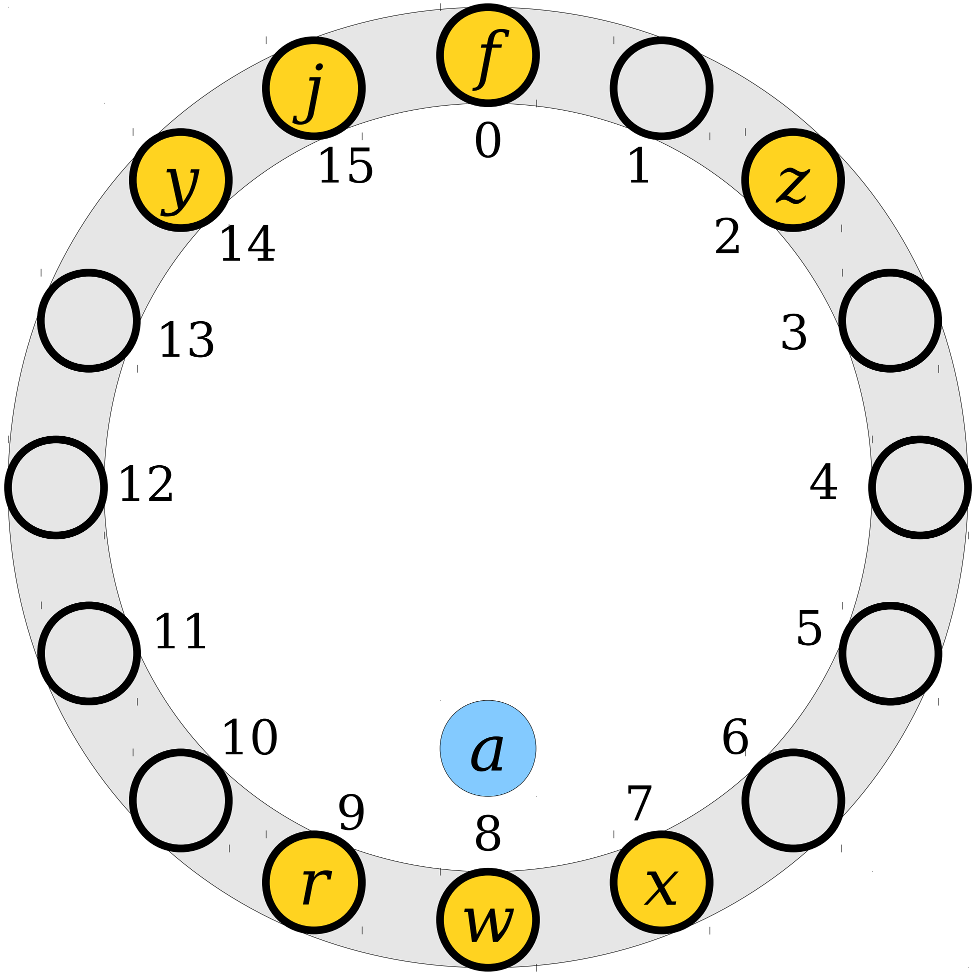
Linear Probing

- To look up an element, compute its hash code and start looking there.
- Move around the ring until either the element is found or a blank spot is detected.
- (If every single slot is full, stop looking after you've tried them all.)



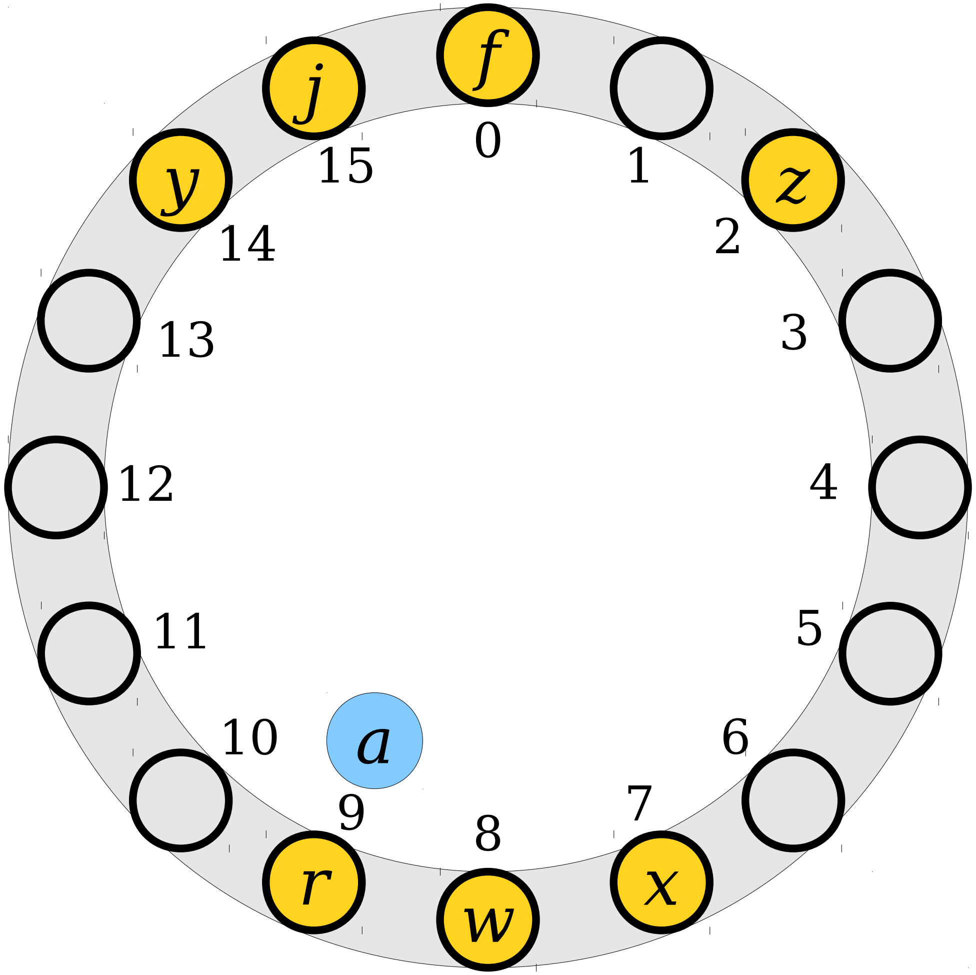
Linear Probing

- To look up an element, compute its hash code and start looking there.
- Move around the ring until either the element is found or a blank spot is detected.
- (If every single slot is full, stop looking after you've tried them all.)



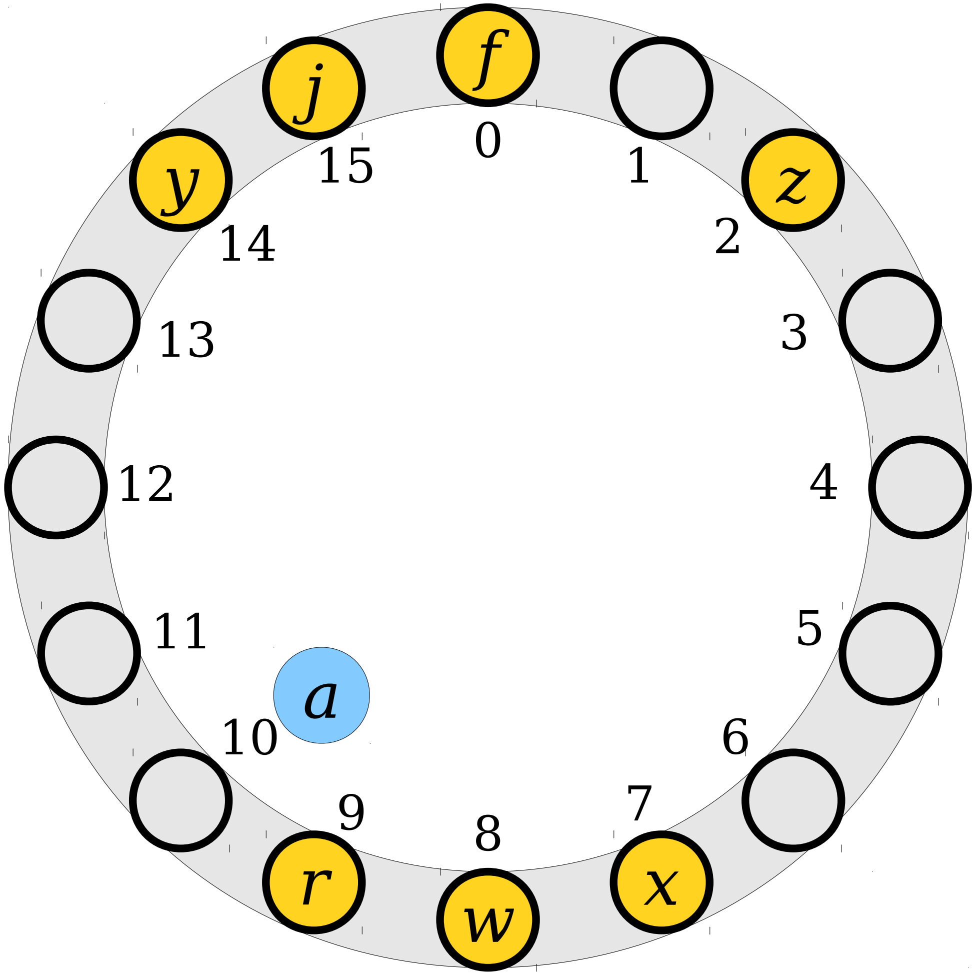
Linear Probing

- To look up an element, compute its hash code and start looking there.
- Move around the ring until either the element is found or a blank spot is detected.
- (If every single slot is full, stop looking after you've tried them all.)



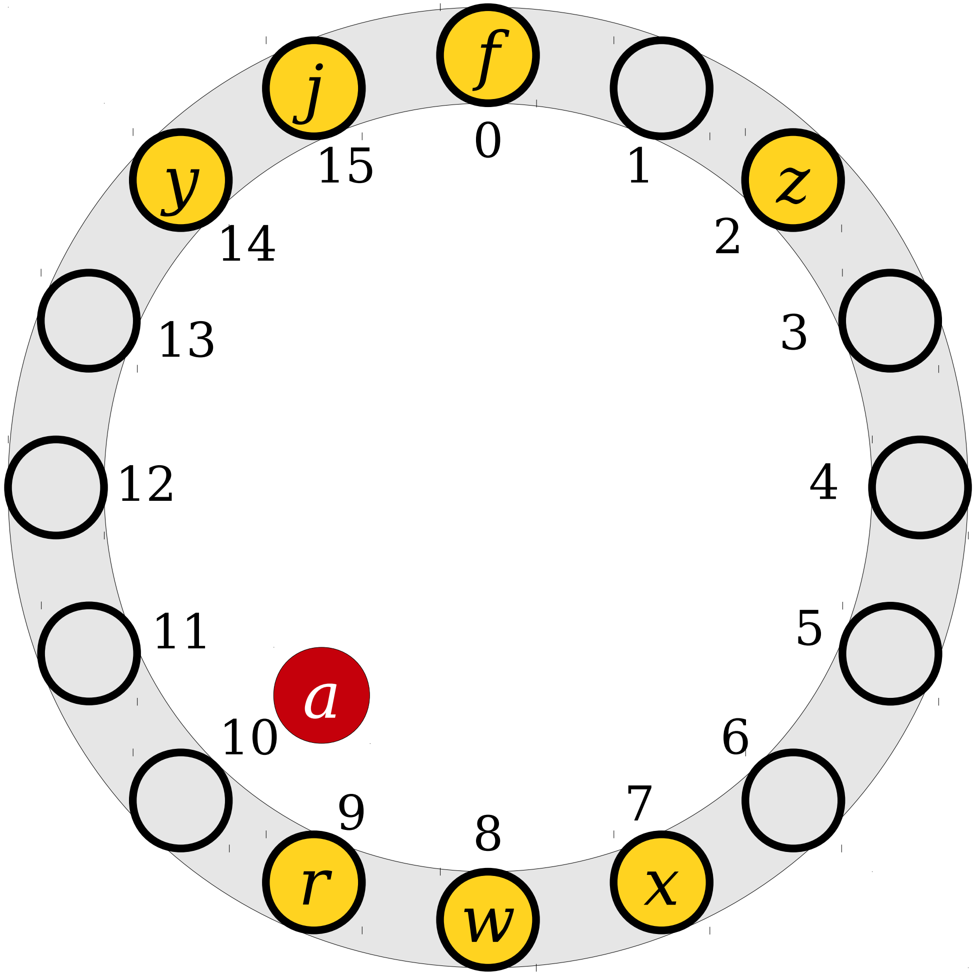
Linear Probing

- To look up an element, compute its hash code and start looking there.
- Move around the ring until either the element is found or a blank spot is detected.
- (If every single slot is full, stop looking after you've tried them all.)



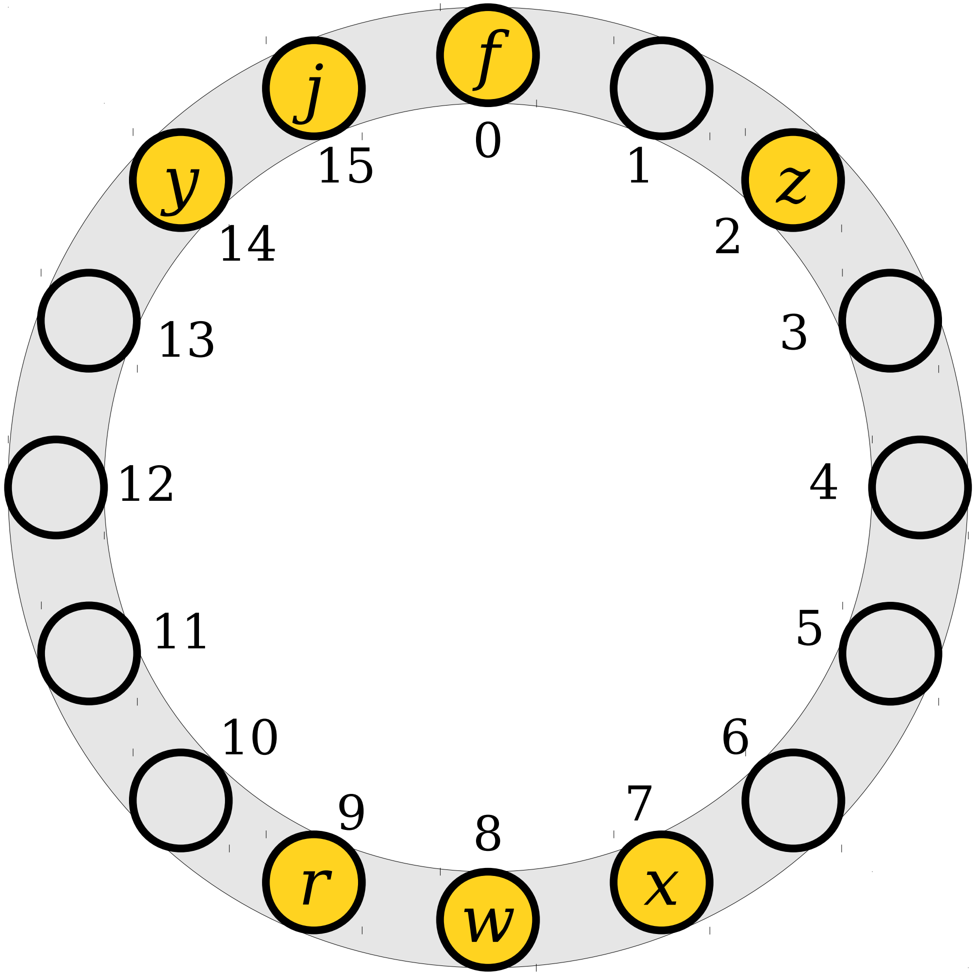
Linear Probing

- To look up an element, compute its hash code and start looking there.
- Move around the ring until either the element is found or a blank spot is detected.
- (If every single slot is full, stop looking after you've tried them all.)



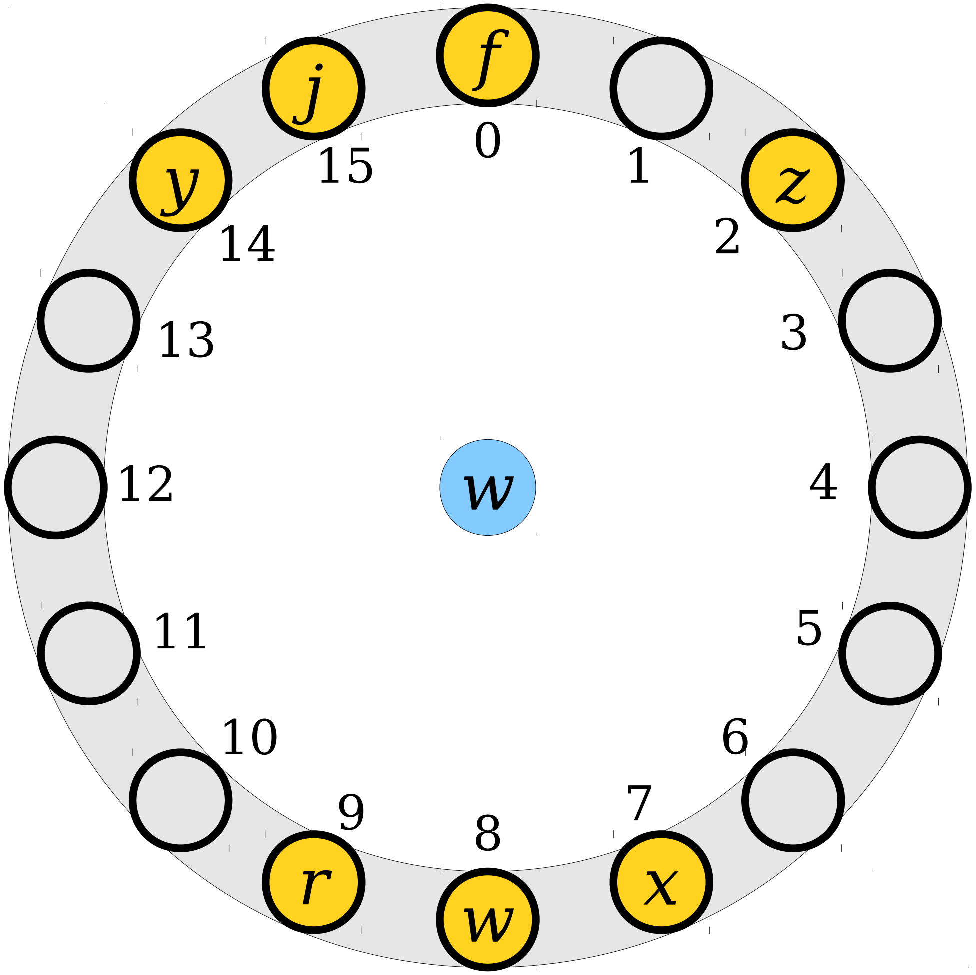
Linear Probing

- Deletions are a bit trickier than in chained hashing.
- We cannot just do a search and remove the element where we find it.
- Why?



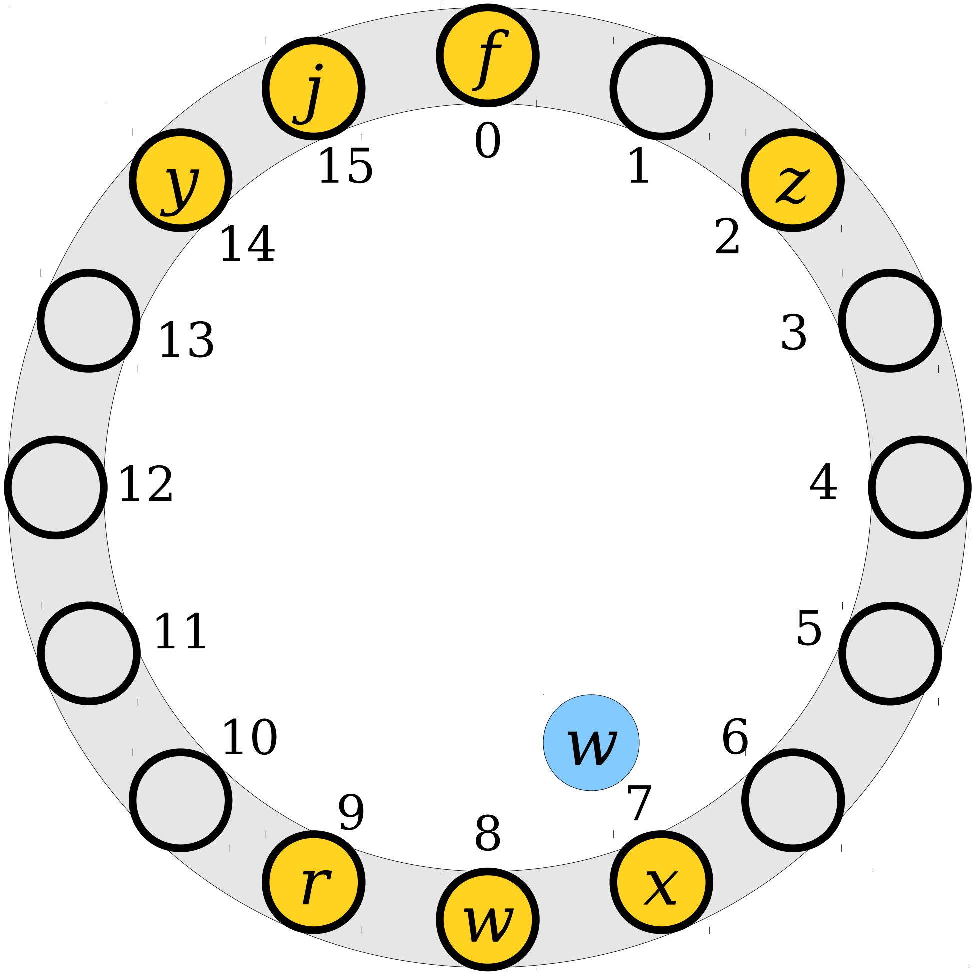
Linear Probing

- Deletions are a bit trickier than in chained hashing.
- We cannot just do a search and remove the element where we find it.
- Why?



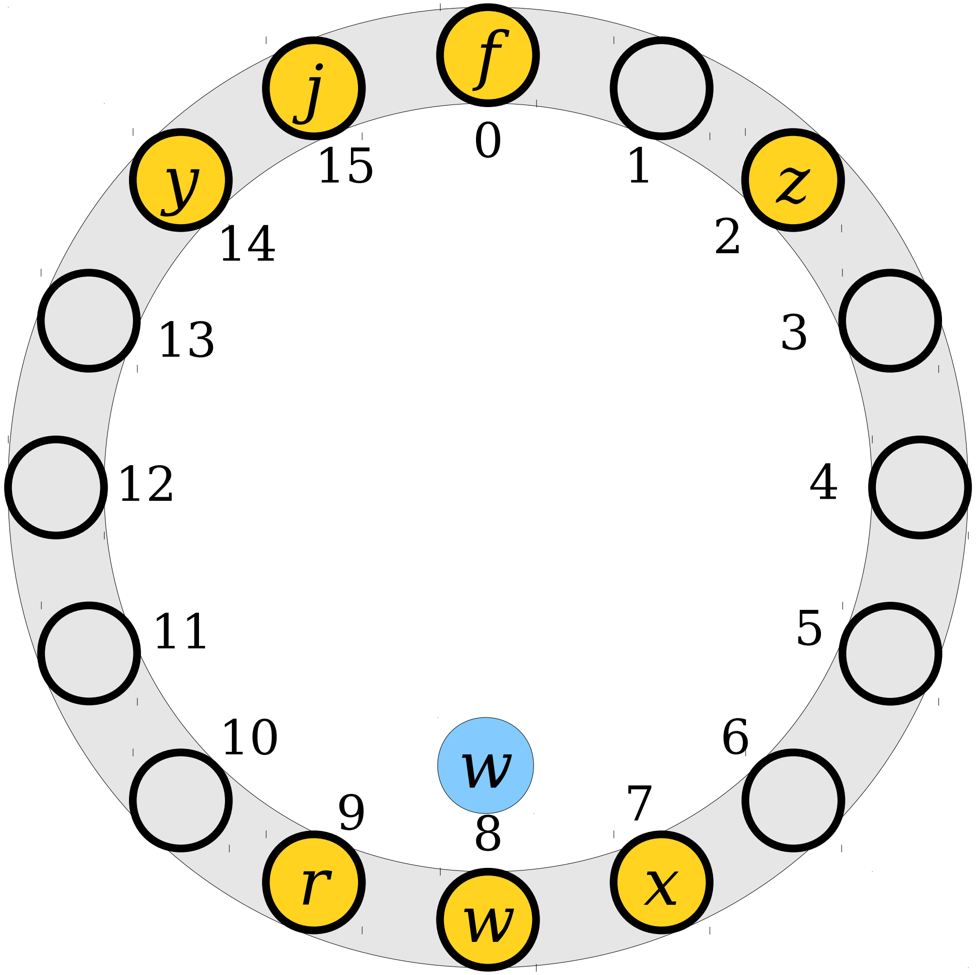
Linear Probing

- Deletions are a bit trickier than in chained hashing.
- We cannot just do a search and remove the element where we find it.
- Why?



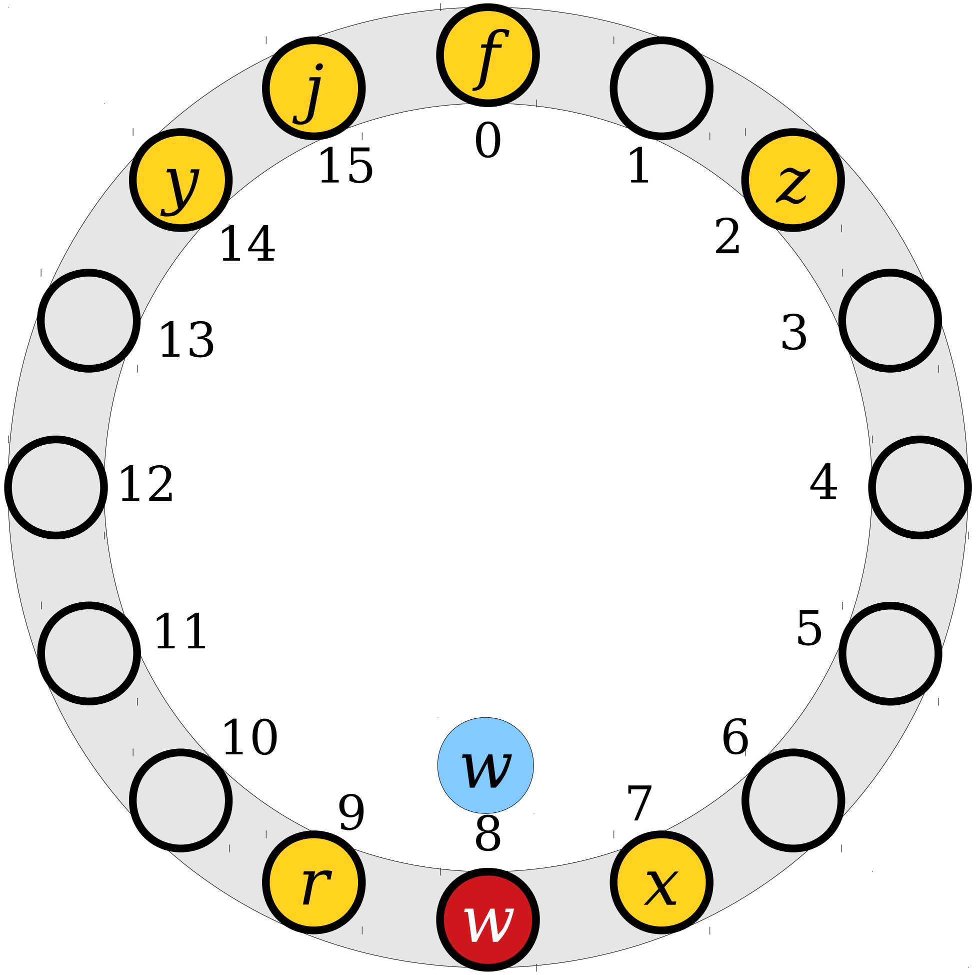
Linear Probing

- Deletions are a bit trickier than in chained hashing.
- We cannot just do a search and remove the element where we find it.
- Why?



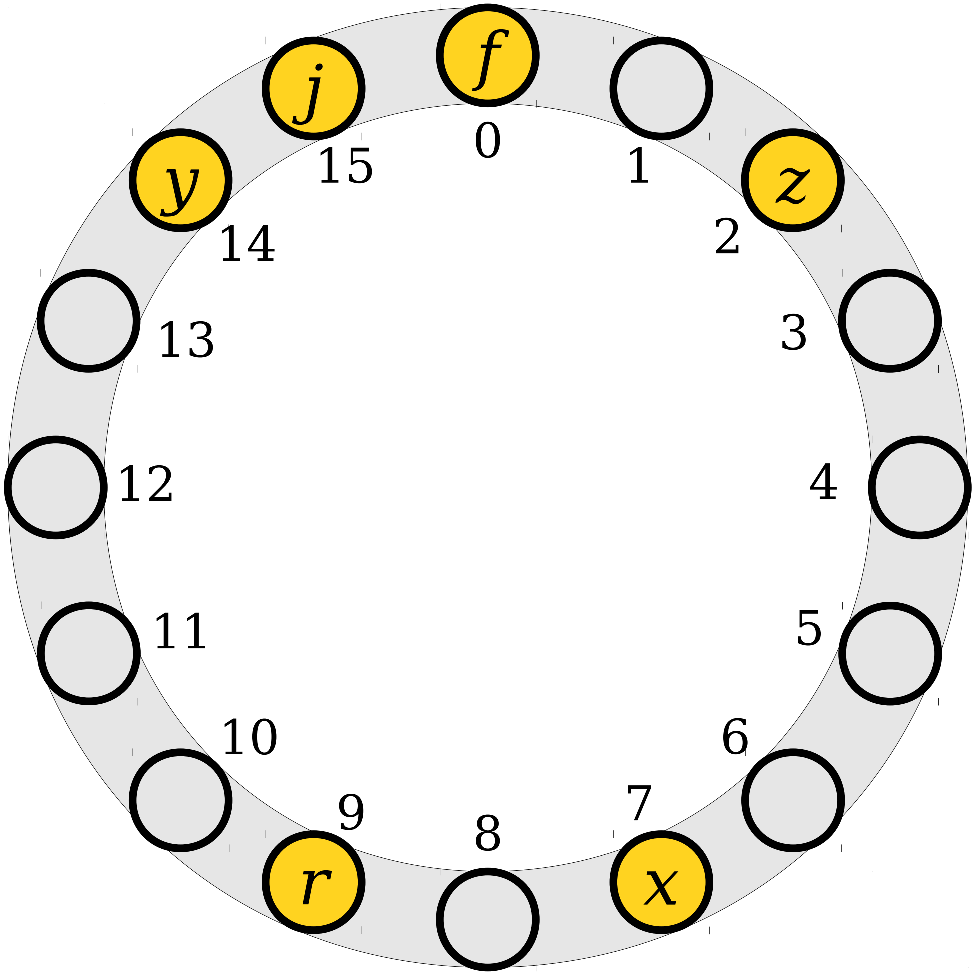
Linear Probing

- Deletions are a bit trickier than in chained hashing.
- We cannot just do a search and remove the element where we find it.
- Why?



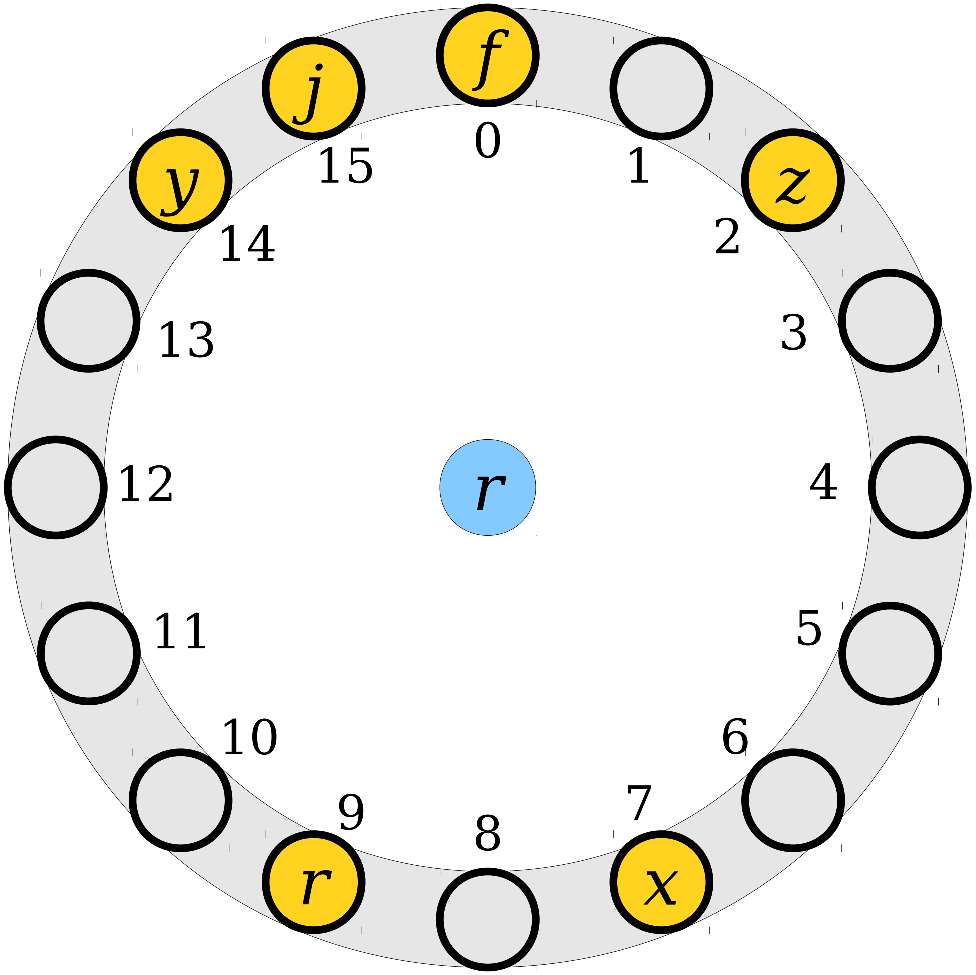
Linear Probing

- Deletions are a bit trickier than in chained hashing.
- We cannot just do a search and remove the element where we find it.
- Why?



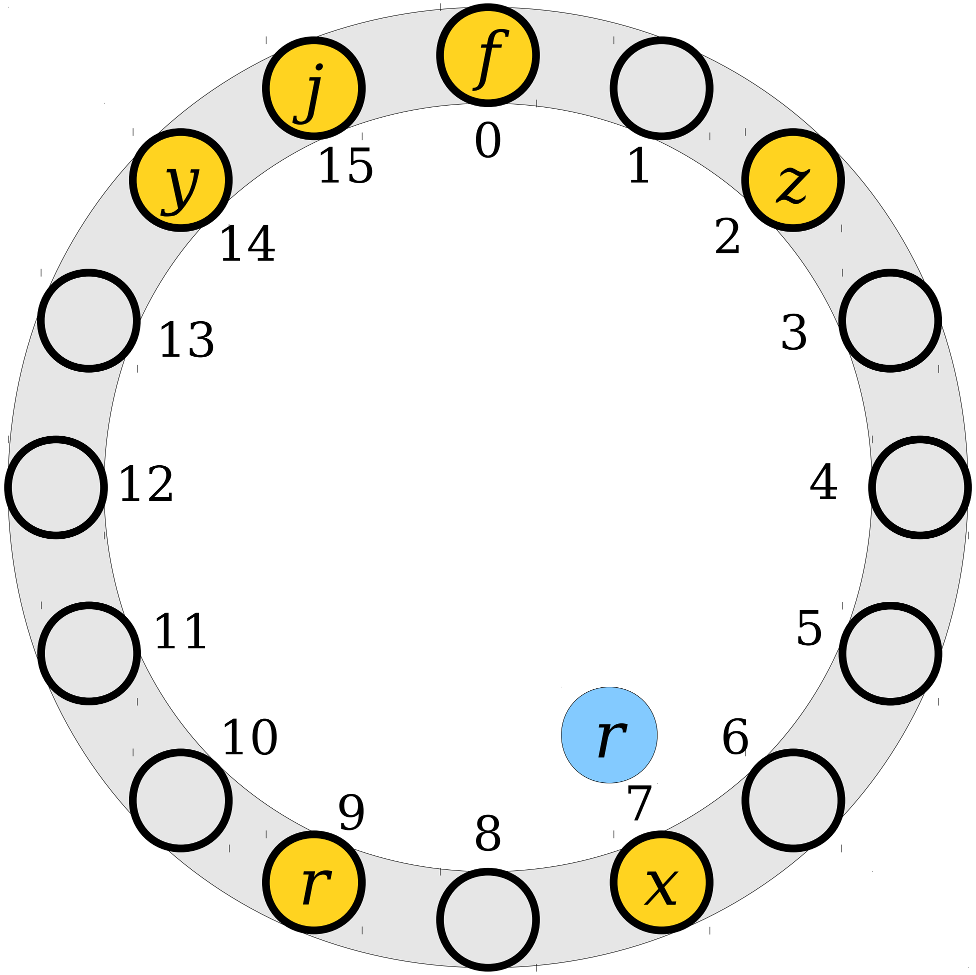
Linear Probing

- Deletions are a bit trickier than in chained hashing.
- We cannot just do a search and remove the element where we find it.
- Why?



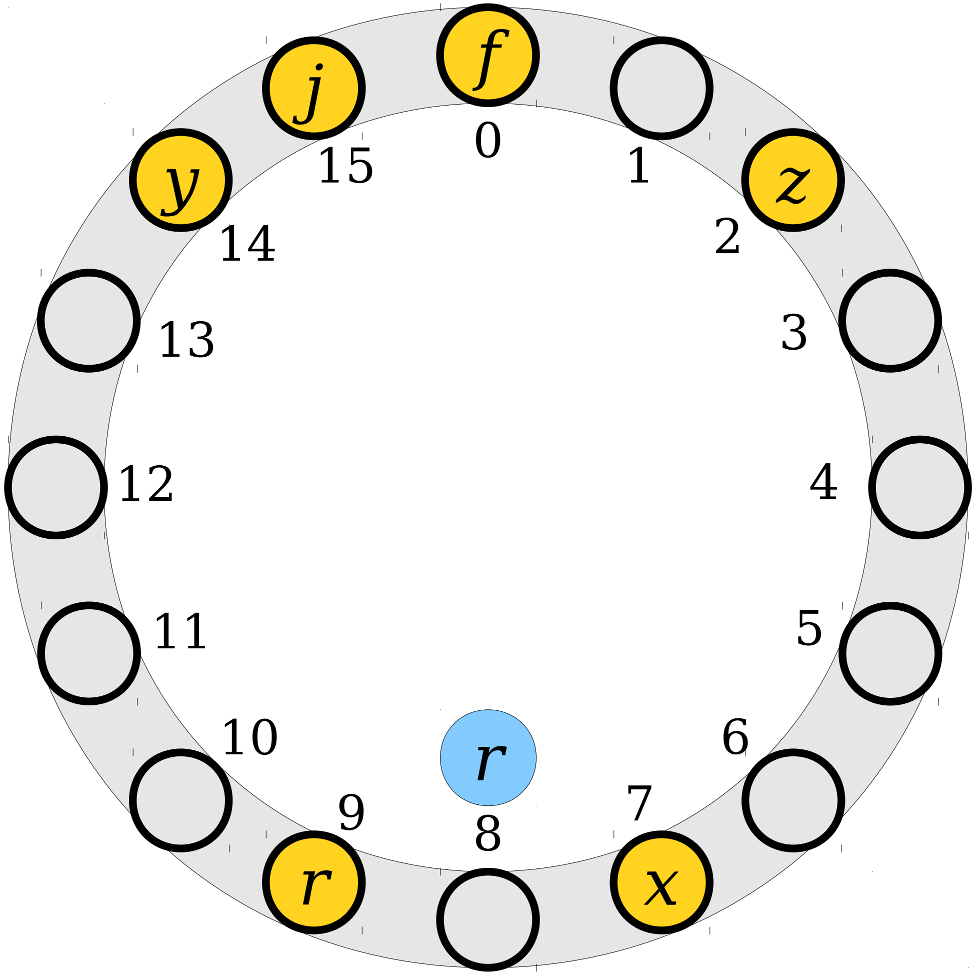
Linear Probing

- Deletions are a bit trickier than in chained hashing.
- We cannot just do a search and remove the element where we find it.
- Why?



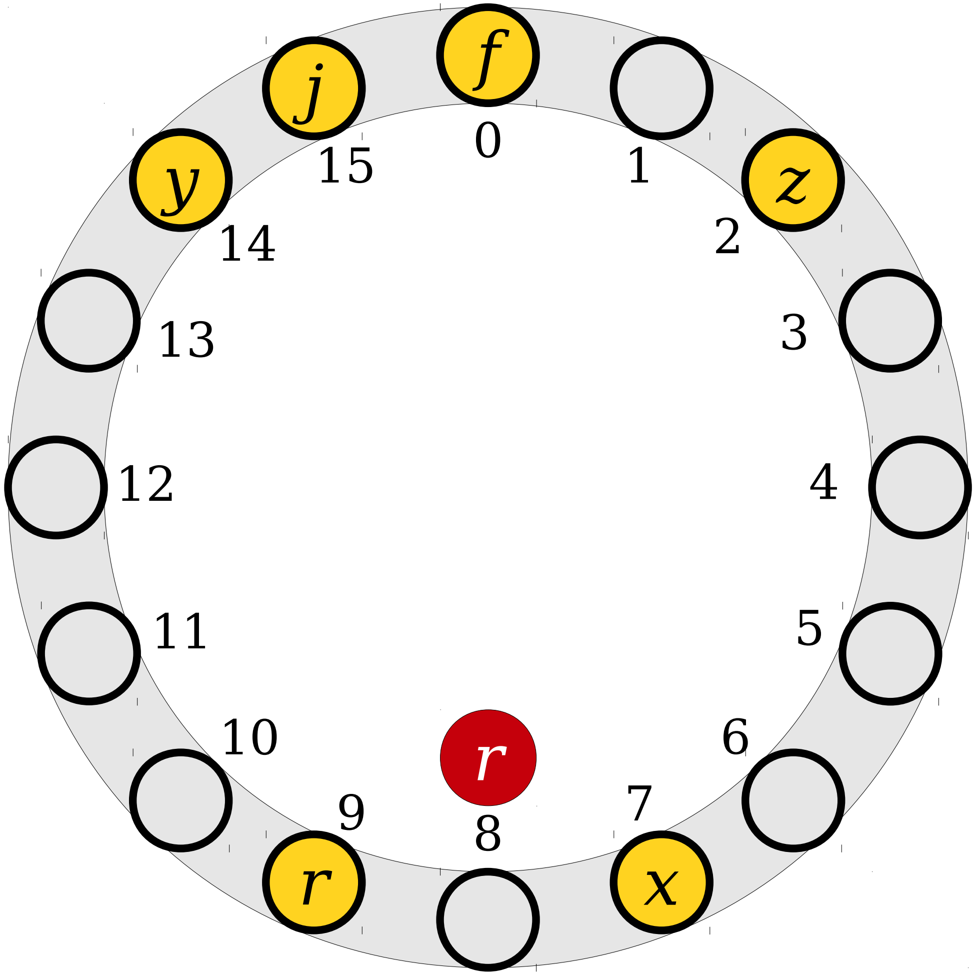
Linear Probing

- Deletions are a bit trickier than in chained hashing.
- We cannot just do a search and remove the element where we find it.
- Why?



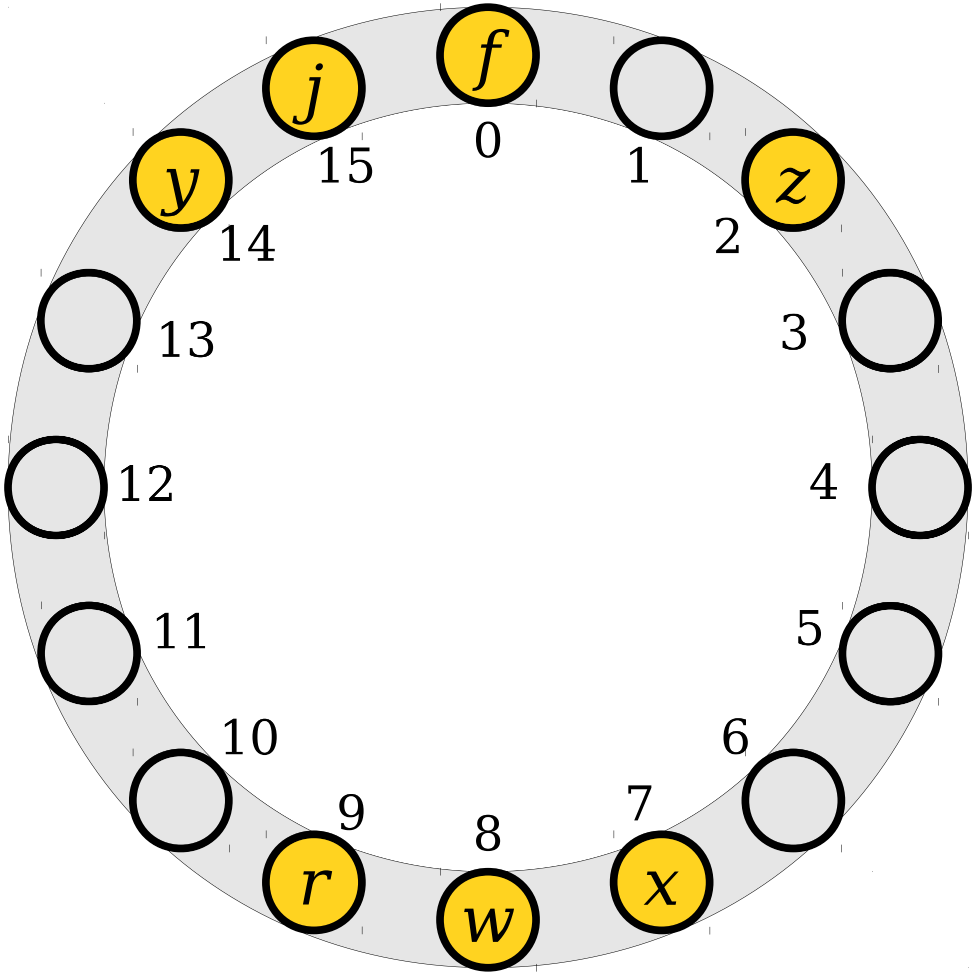
Linear Probing

- Deletions are a bit trickier than in chained hashing.
- We cannot just do a search and remove the element where we find it.
- Why?



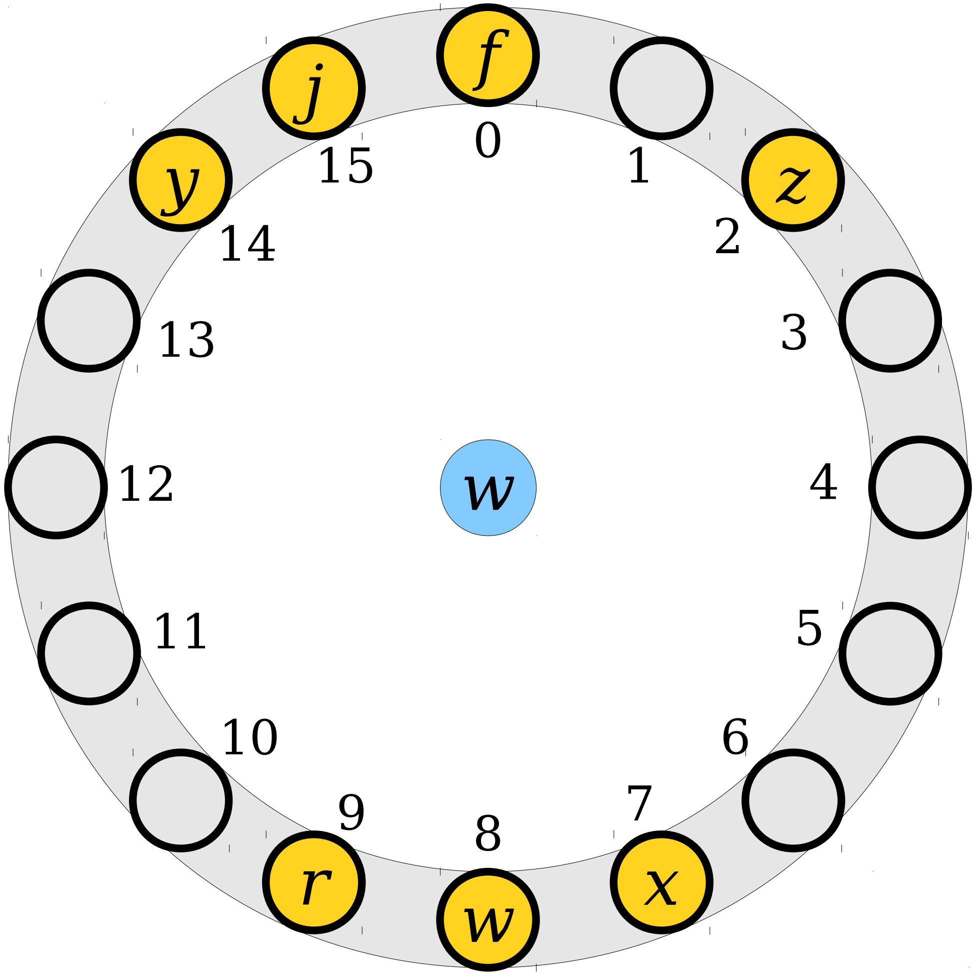
Linear Probing

- Deletions are often implemented using ***tombstones***.
- When removing an element, mark that the cell is empty and was previously occupied.
- When doing a lookup, don't stop at a tombstone. Instead, keep the search going.



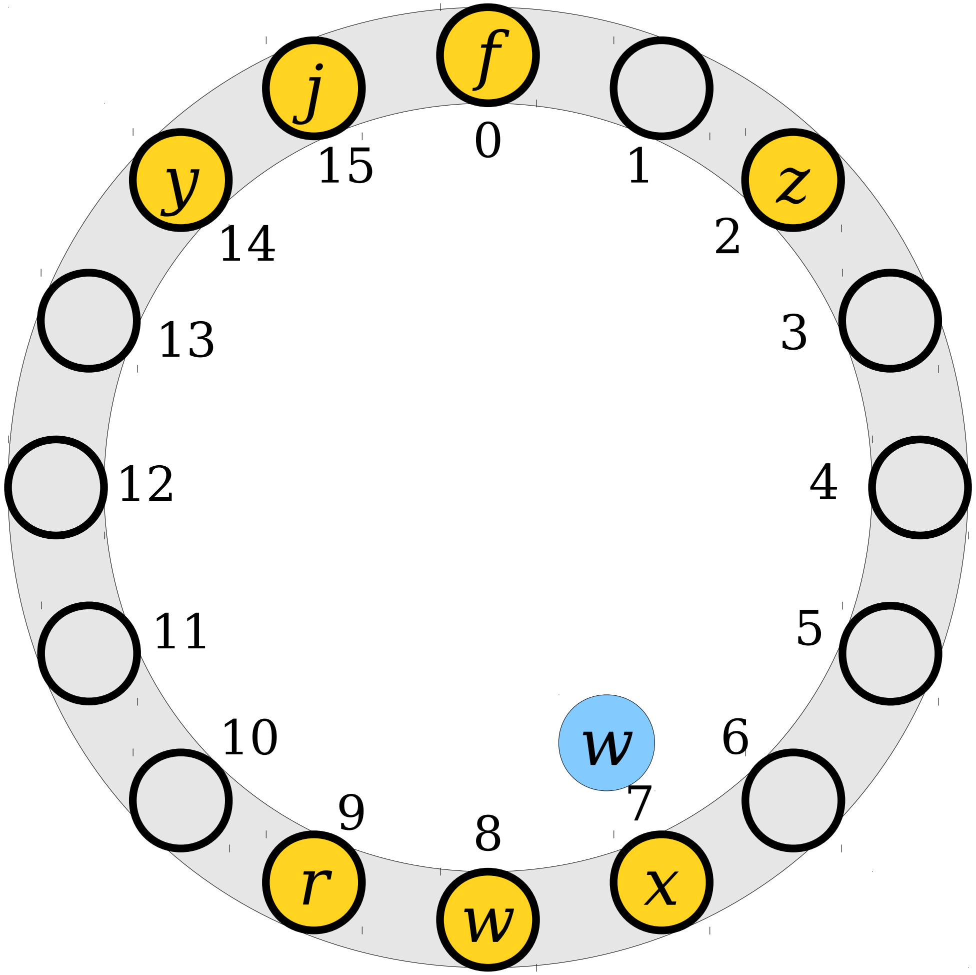
Linear Probing

- Deletions are often implemented using ***tombstones***.
- When removing an element, mark that the cell is empty and was previously occupied.
- When doing a lookup, don't stop at a tombstone. Instead, keep the search going.



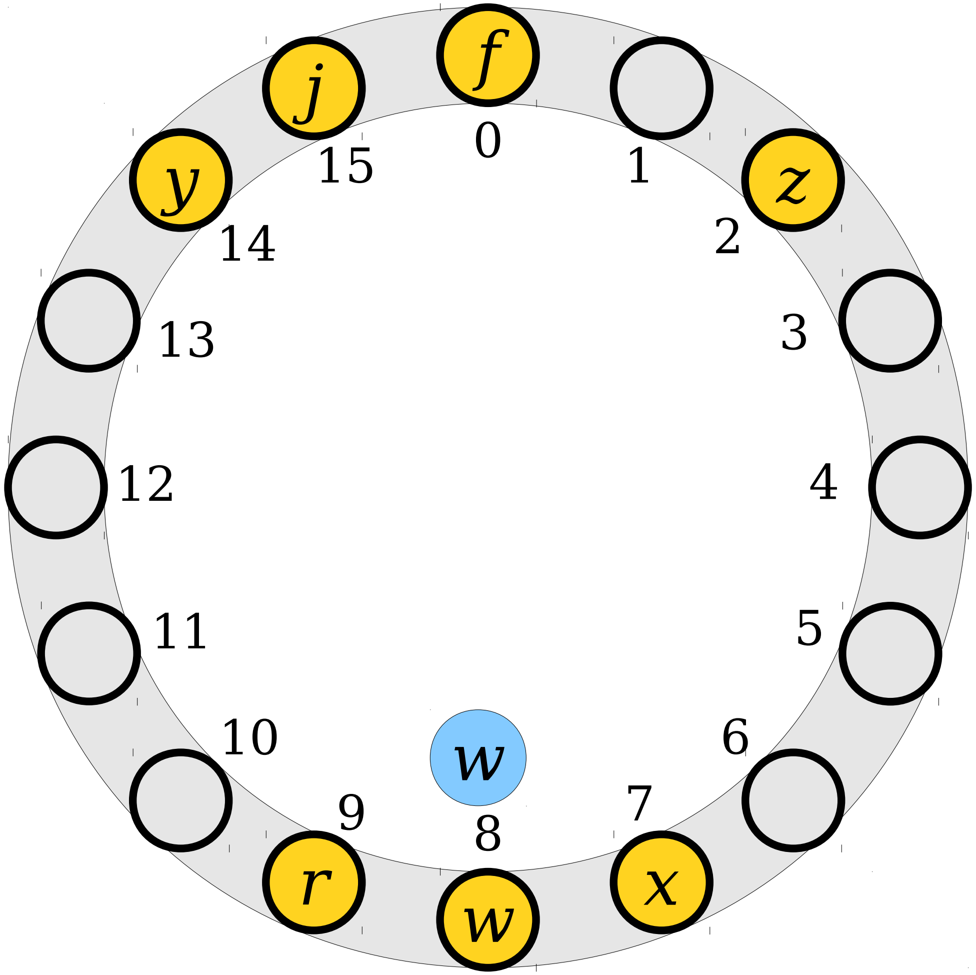
Linear Probing

- Deletions are often implemented using ***tombstones***.
- When removing an element, mark that the cell is empty and was previously occupied.
- When doing a lookup, don't stop at a tombstone. Instead, keep the search going.



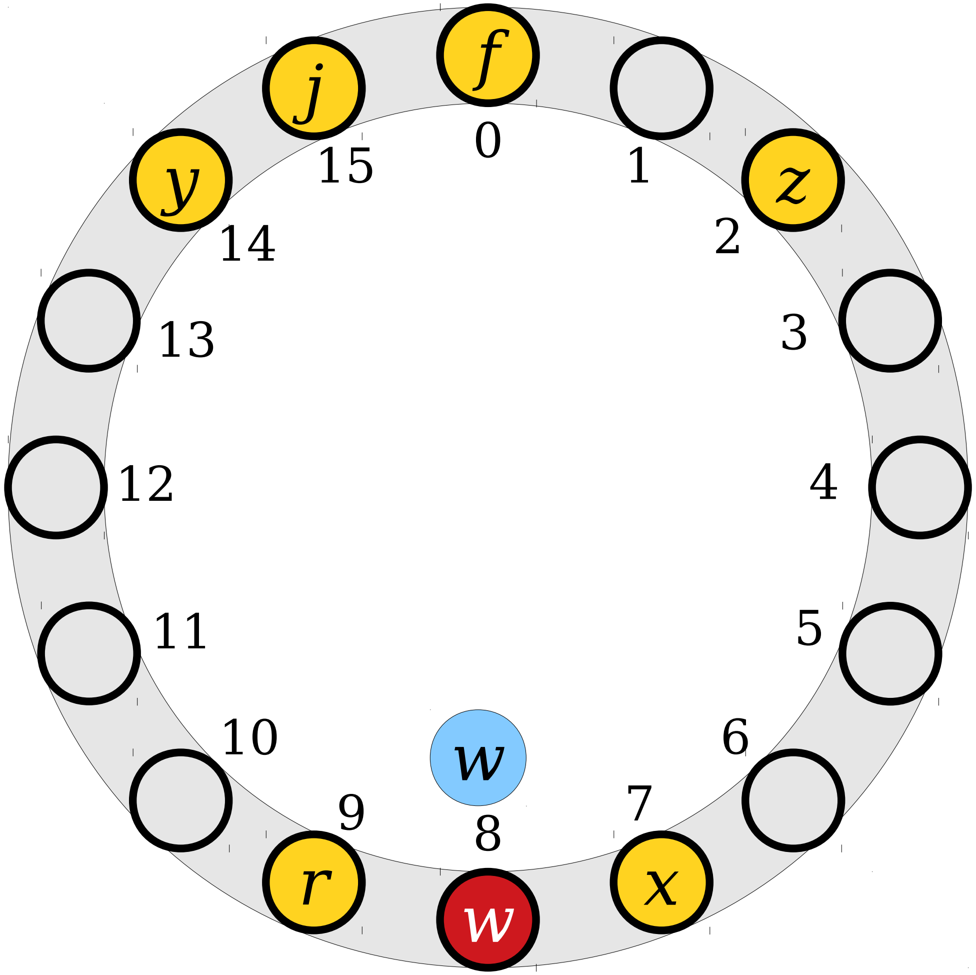
Linear Probing

- Deletions are often implemented using ***tombstones***.
- When removing an element, mark that the cell is empty and was previously occupied.
- When doing a lookup, don't stop at a tombstone. Instead, keep the search going.



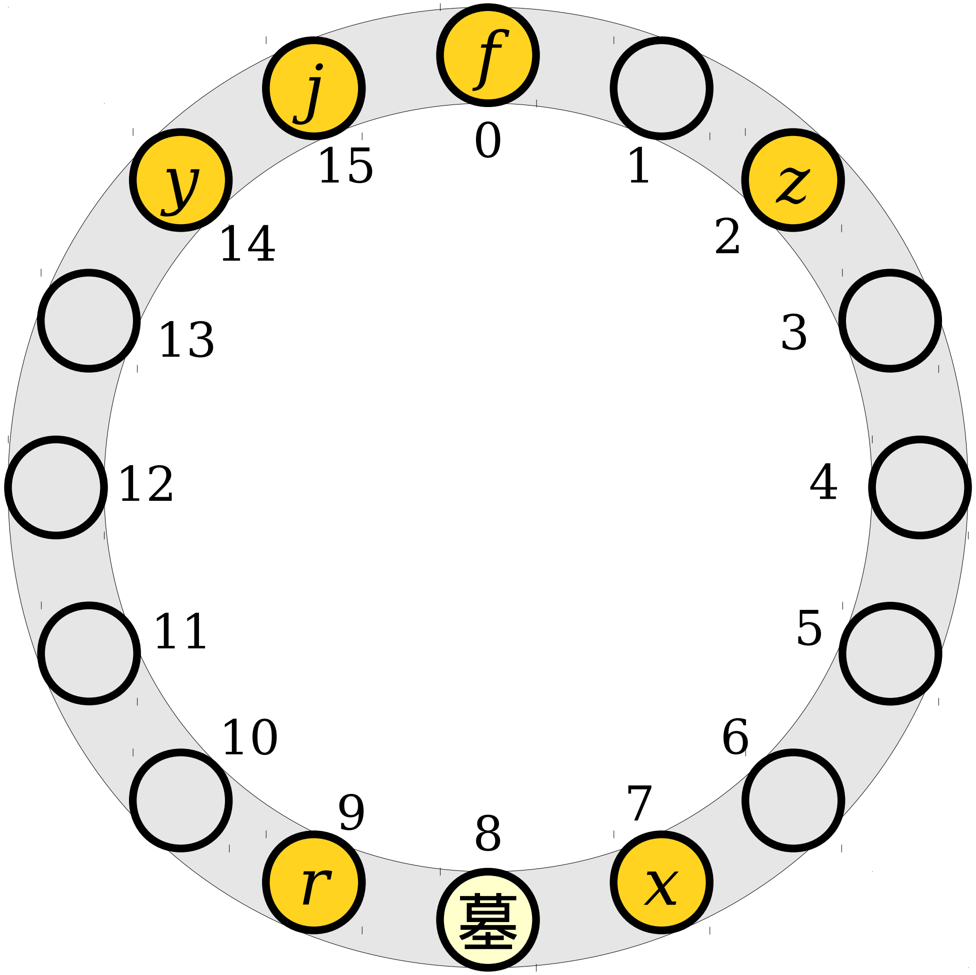
Linear Probing

- Deletions are often implemented using ***tombstones***.
- When removing an element, mark that the cell is empty and was previously occupied.
- When doing a lookup, don't stop at a tombstone. Instead, keep the search going.



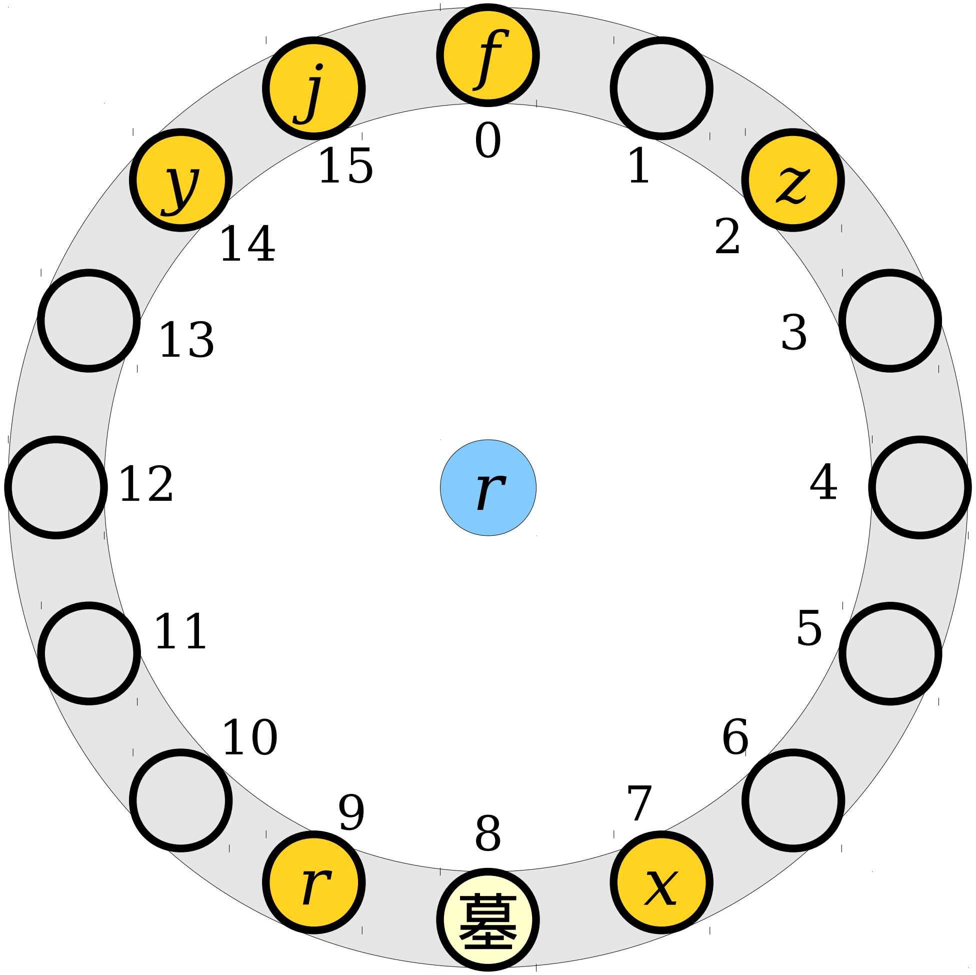
Linear Probing

- Deletions are often implemented using ***tombstones***.
- When removing an element, mark that the cell is empty and was previously occupied.
- When doing a lookup, don't stop at a tombstone. Instead, keep the search going.



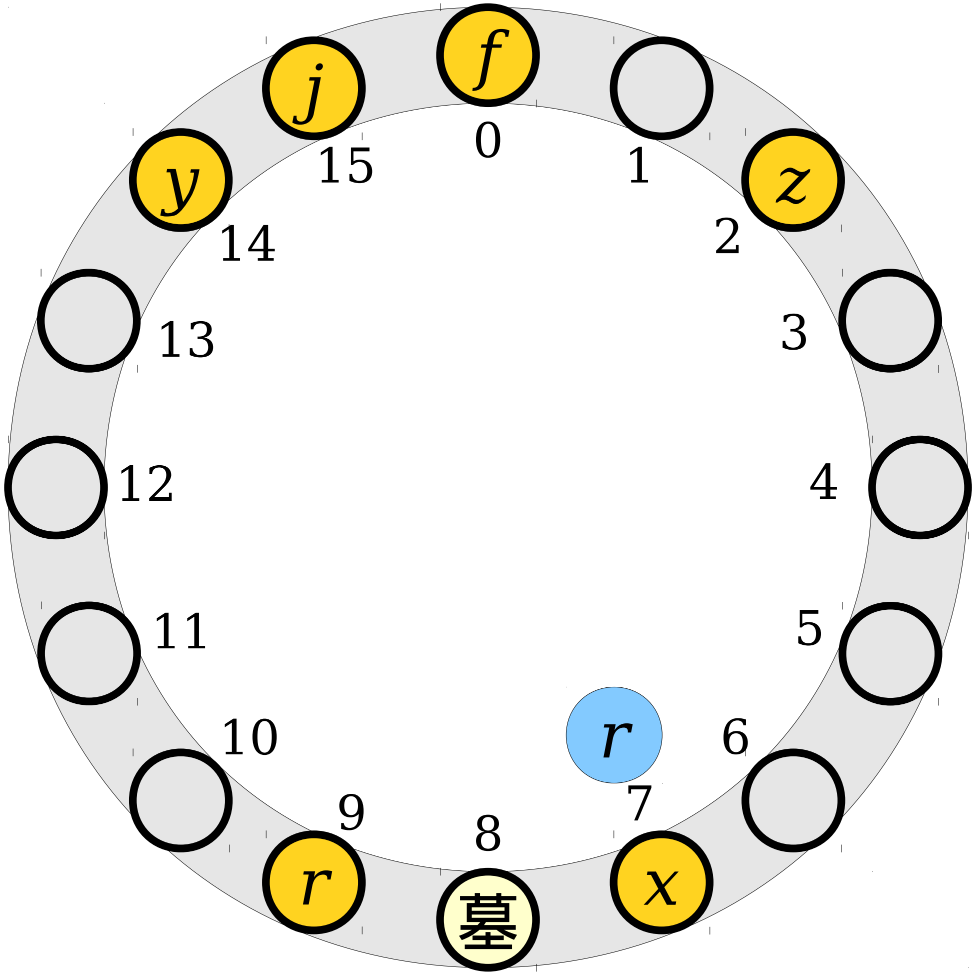
Linear Probing

- Deletions are often implemented using ***tombstones***.
- When removing an element, mark that the cell is empty and was previously occupied.
- When doing a lookup, don't stop at a tombstone. Instead, keep the search going.



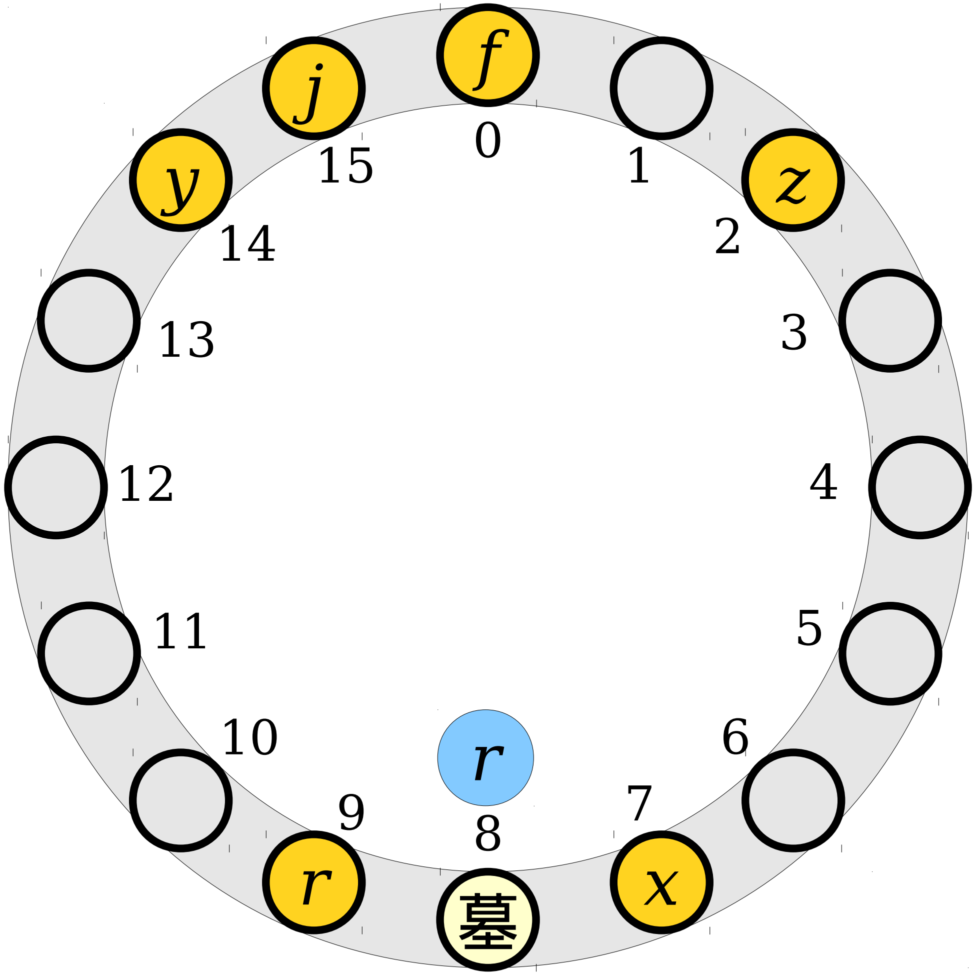
Linear Probing

- Deletions are often implemented using ***tombstones***.
- When removing an element, mark that the cell is empty and was previously occupied.
- When doing a lookup, don't stop at a tombstone. Instead, keep the search going.



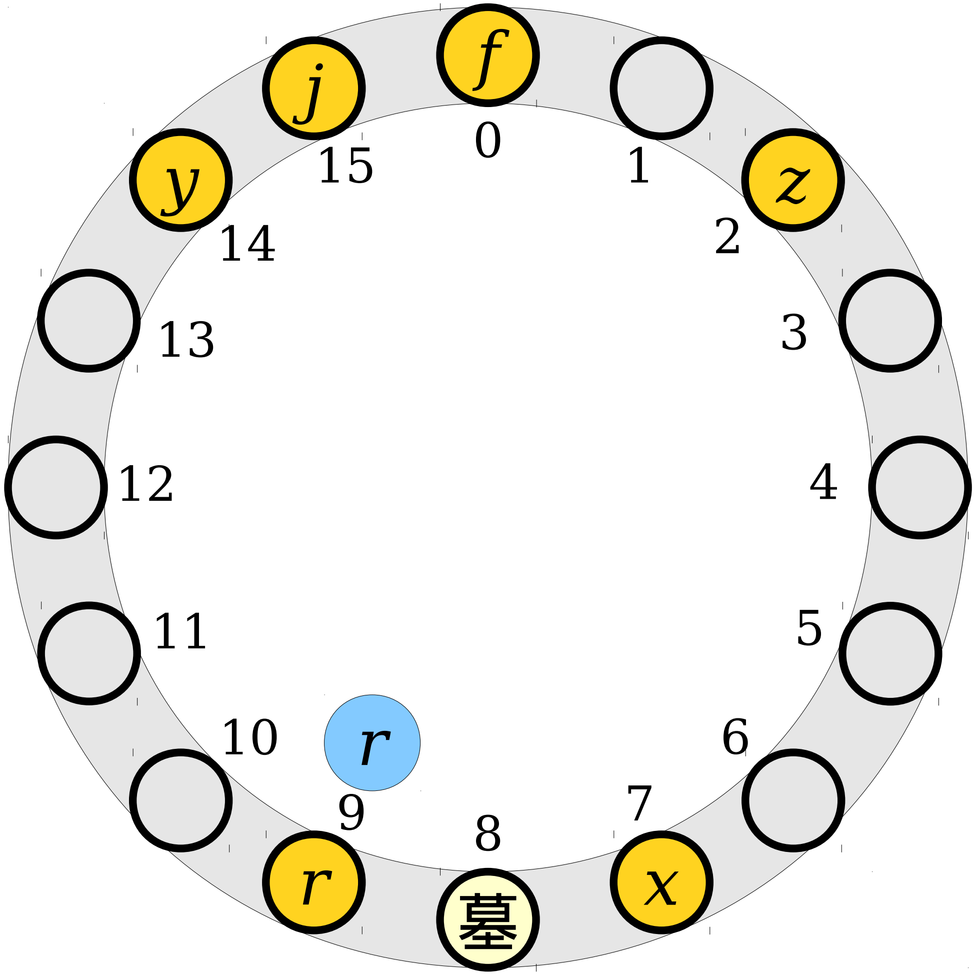
Linear Probing

- Deletions are often implemented using ***tombstones***.
- When removing an element, mark that the cell is empty and was previously occupied.
- When doing a lookup, don't stop at a tombstone. Instead, keep the search going.



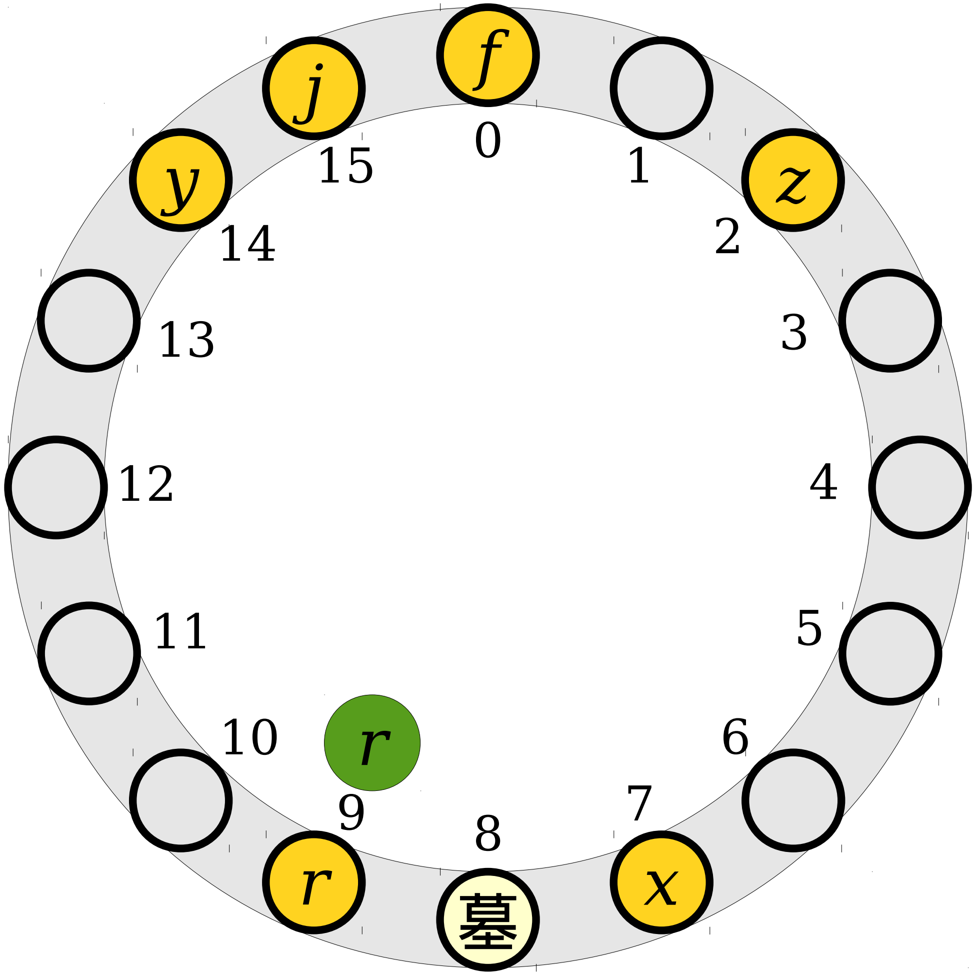
Linear Probing

- Deletions are often implemented using ***tombstones***.
- When removing an element, mark that the cell is empty and was previously occupied.
- When doing a lookup, don't stop at a tombstone. Instead, keep the search going.



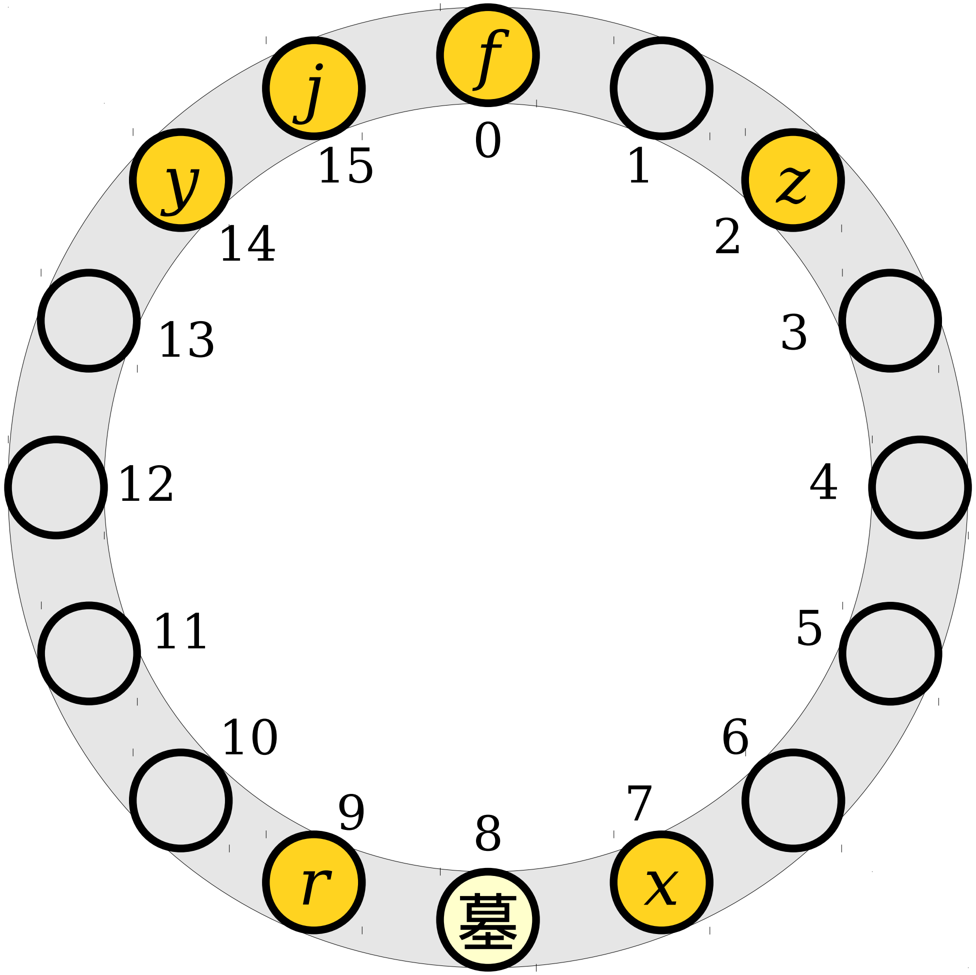
Linear Probing

- Deletions are often implemented using ***tombstones***.
- When removing an element, mark that the cell is empty and was previously occupied.
- When doing a lookup, don't stop at a tombstone. Instead, keep the search going.



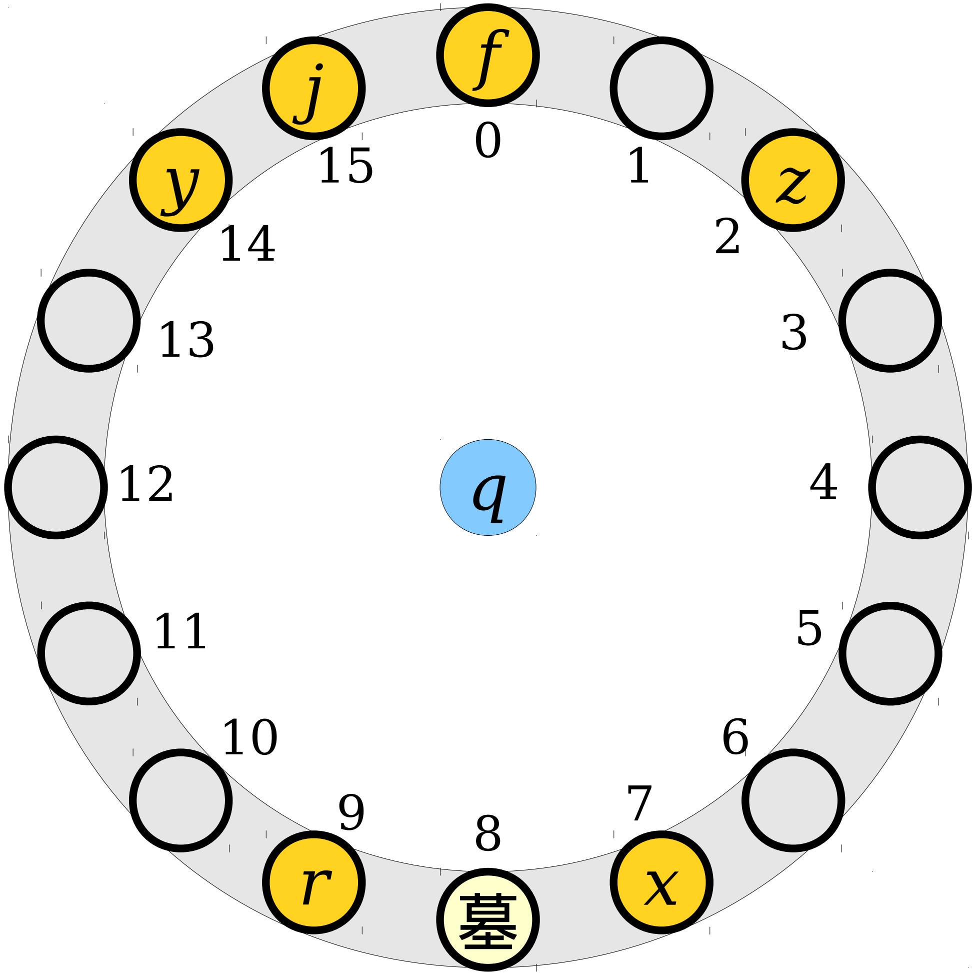
Linear Probing

- Having too many tombstones in a table can slow down lookups, since we have to scan past them.
- Tombstones should be overwritten when new elements are inserted.



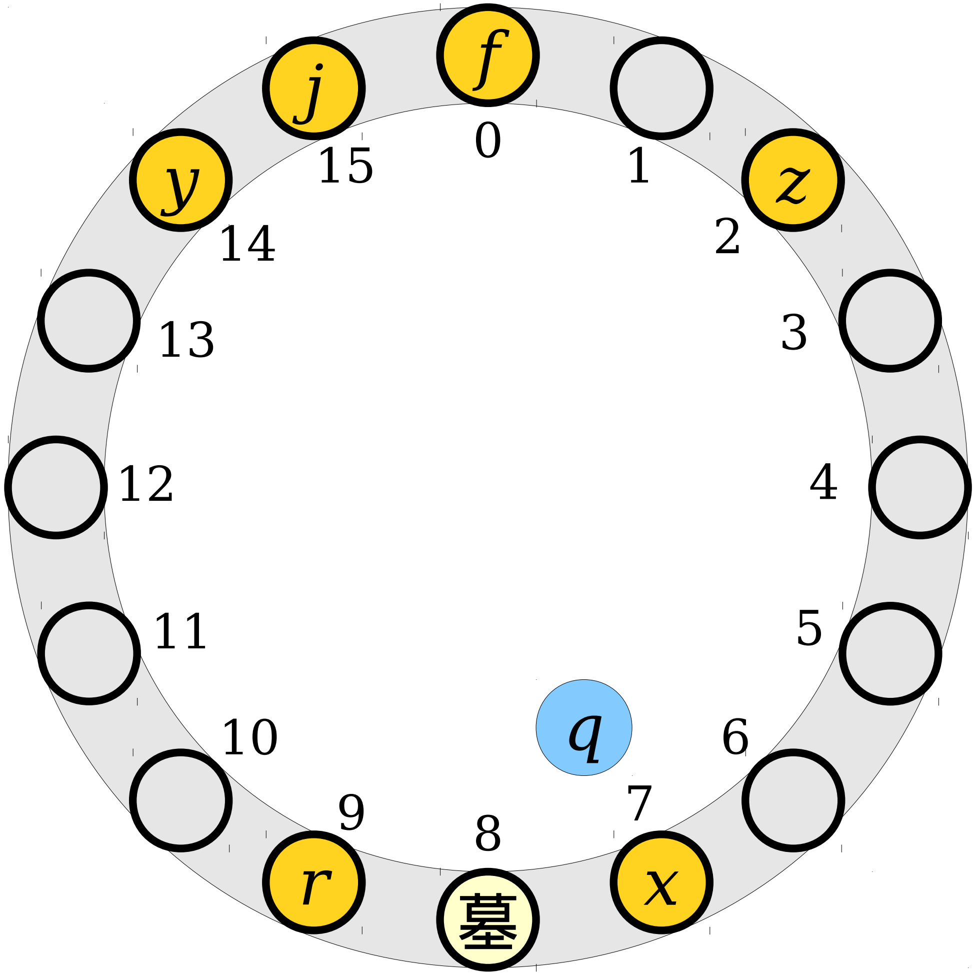
Linear Probing

- Having too many tombstones in a table can slow down lookups, since we have to scan past them.
- Tombstones should be overwritten when new elements are inserted.



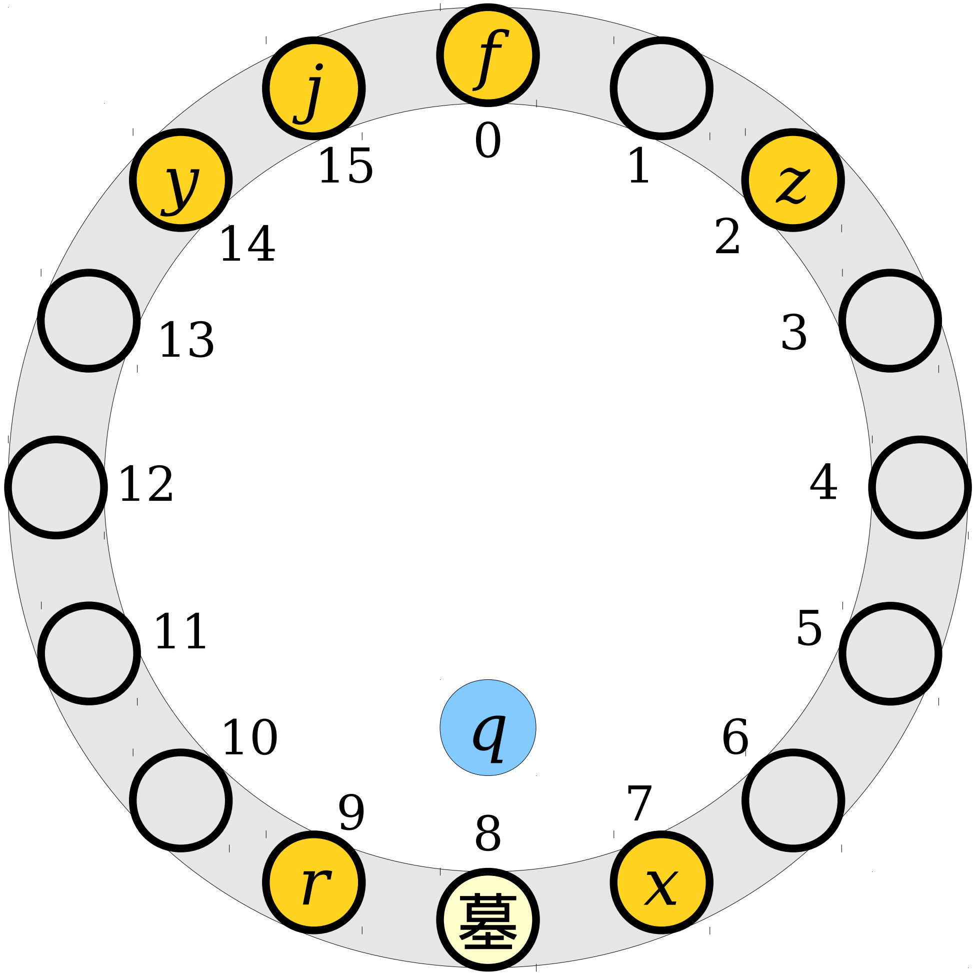
Linear Probing

- Having too many tombstones in a table can slow down lookups, since we have to scan past them.
- Tombstones should be overwritten when new elements are inserted.



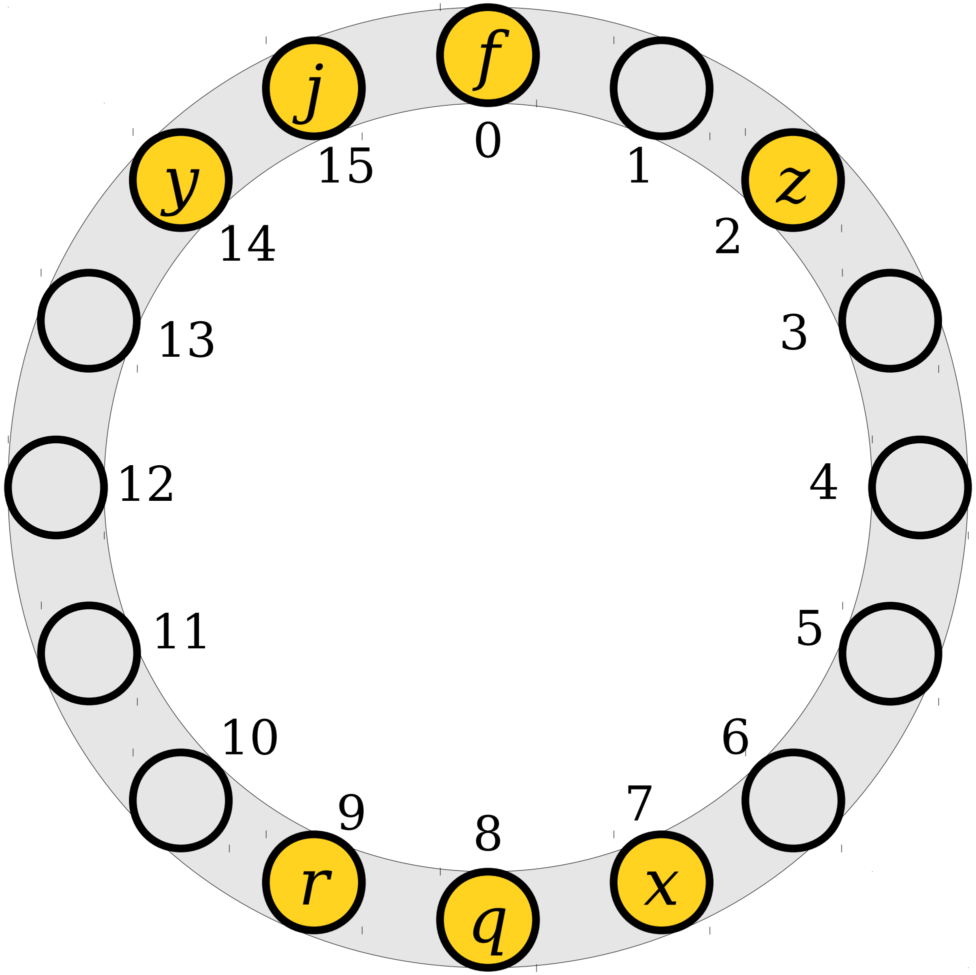
Linear Probing

- Having too many tombstones in a table can slow down lookups, since we have to scan past them.
- Tombstones should be overwritten when new elements are inserted.



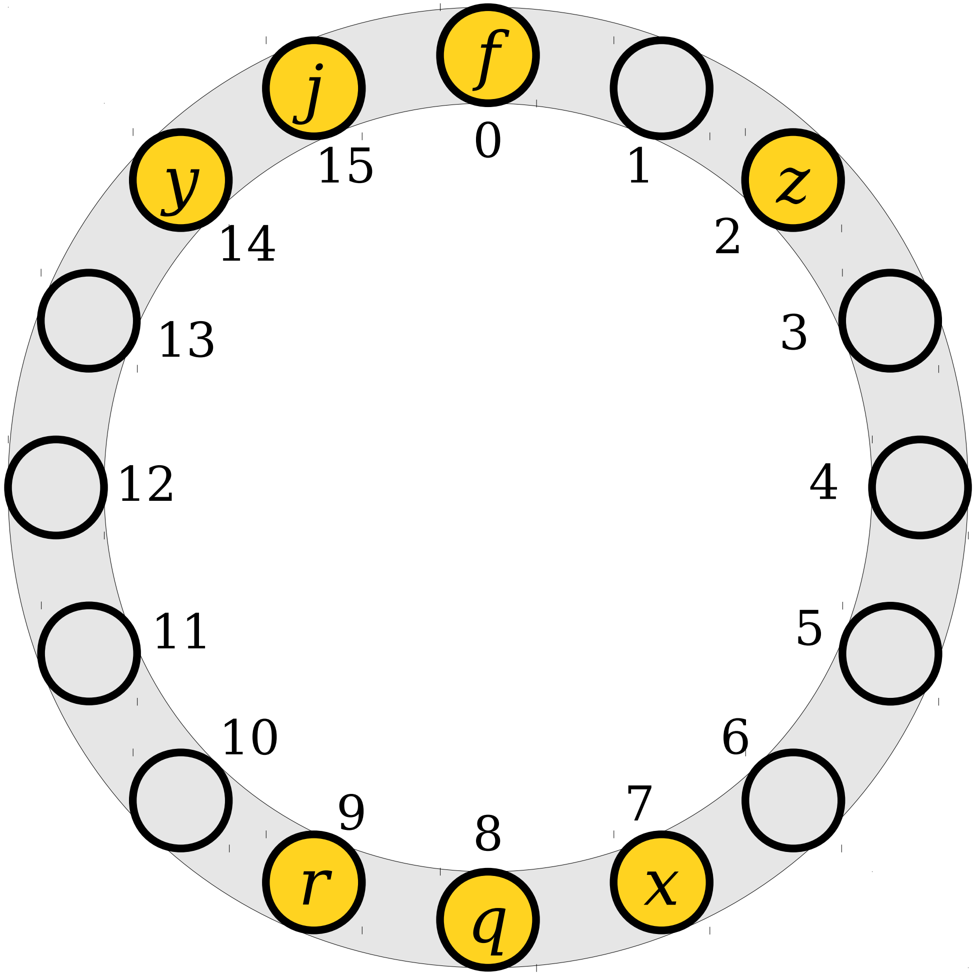
Linear Probing

- Having too many tombstones in a table can slow down lookups, since we have to scan past them.
- Tombstones should be overwritten when new elements are inserted.



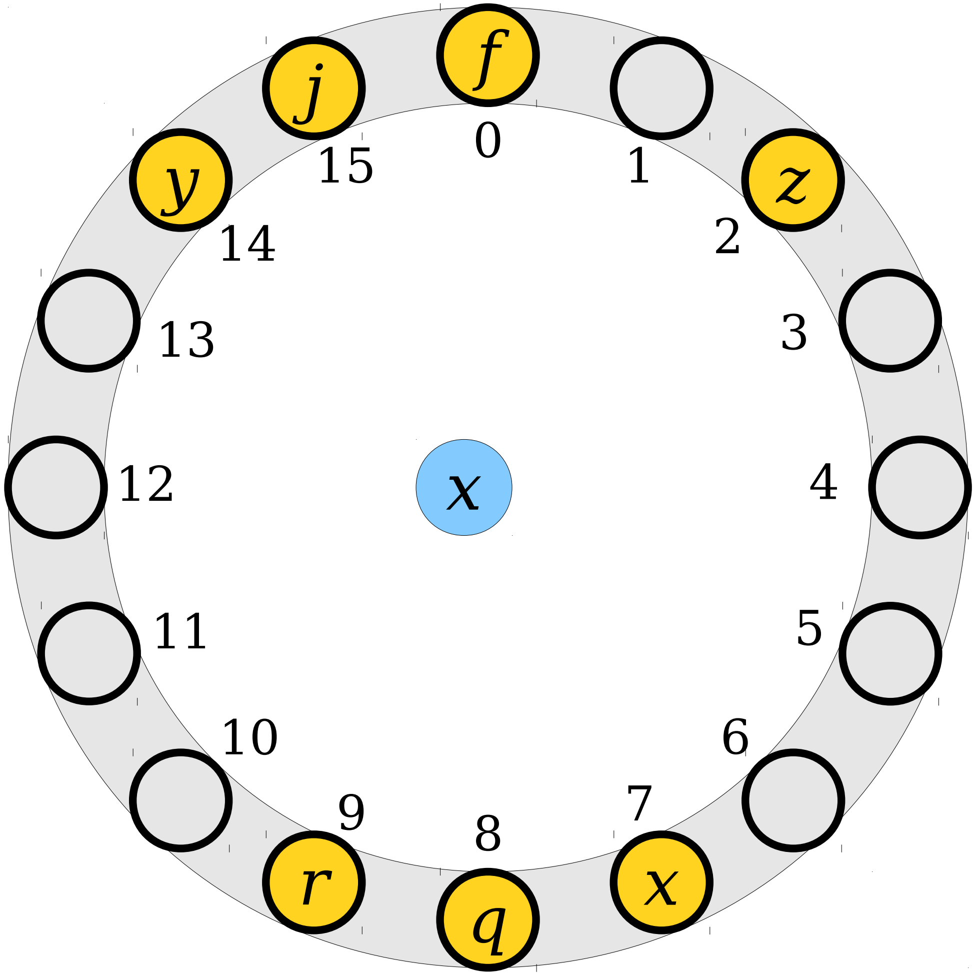
Linear Probing

- Be careful, though, to make sure you don’t allow for duplicates in your table.



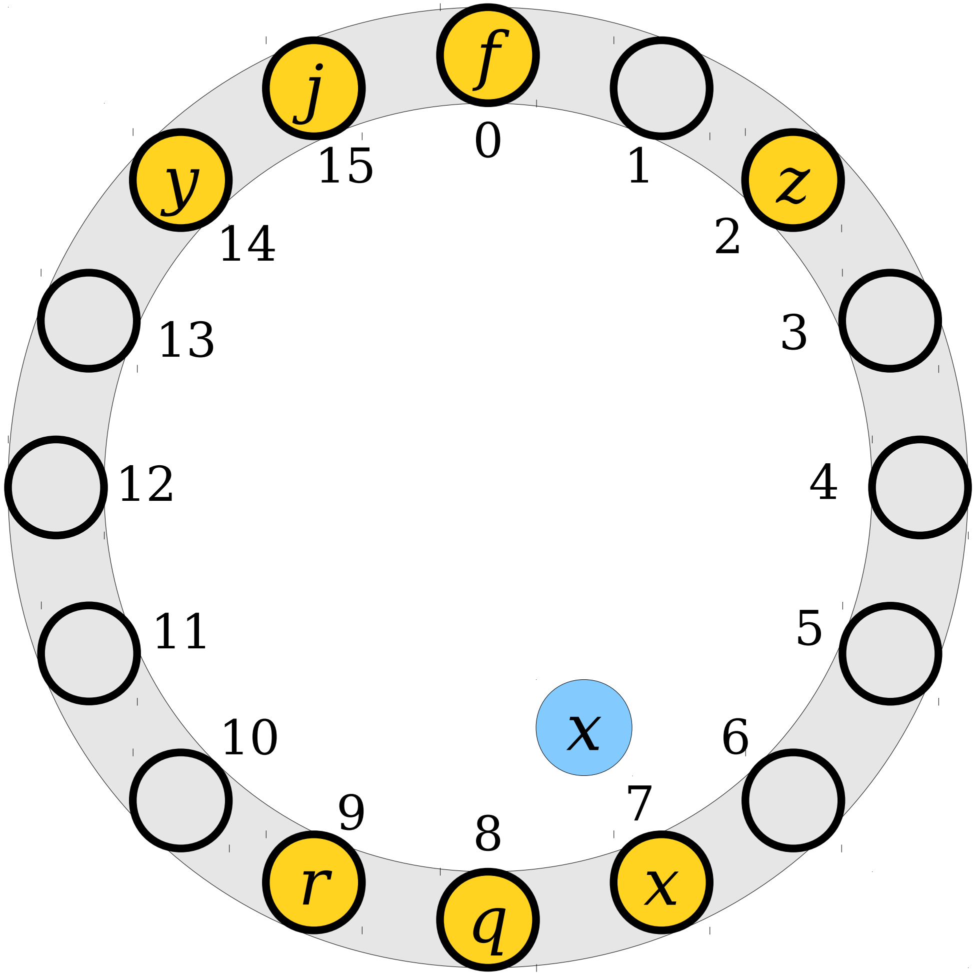
Linear Probing

- Be careful, though, to make sure you don't allow for duplicates in your table.



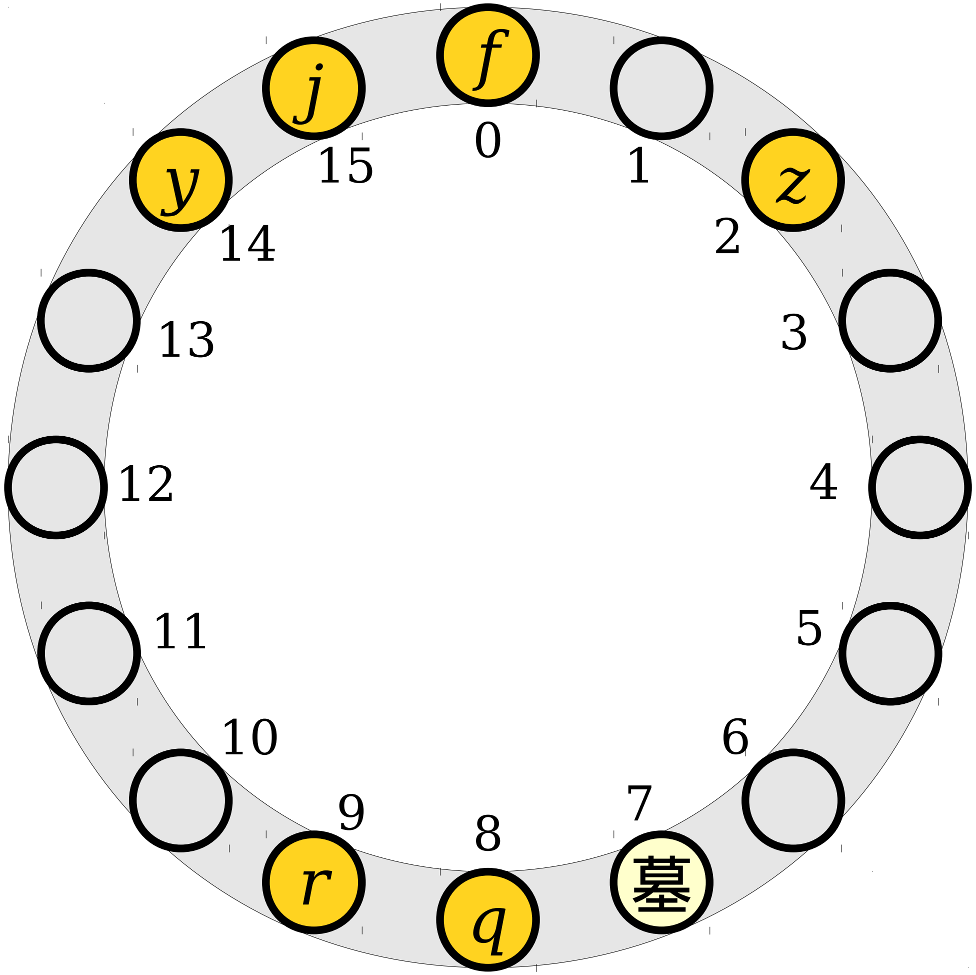
Linear Probing

- Be careful, though, to make sure you don't allow for duplicates in your table.



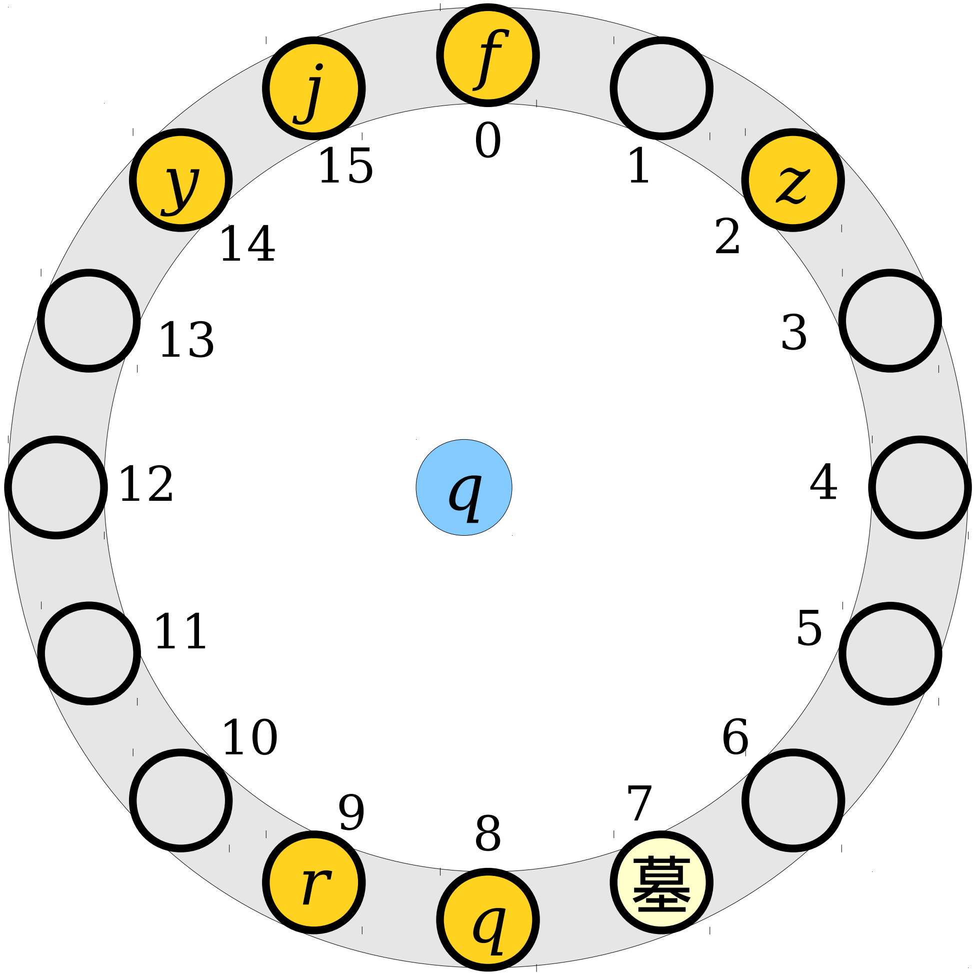
Linear Probing

- Be careful, though, to make sure you don’t allow for duplicates in your table.



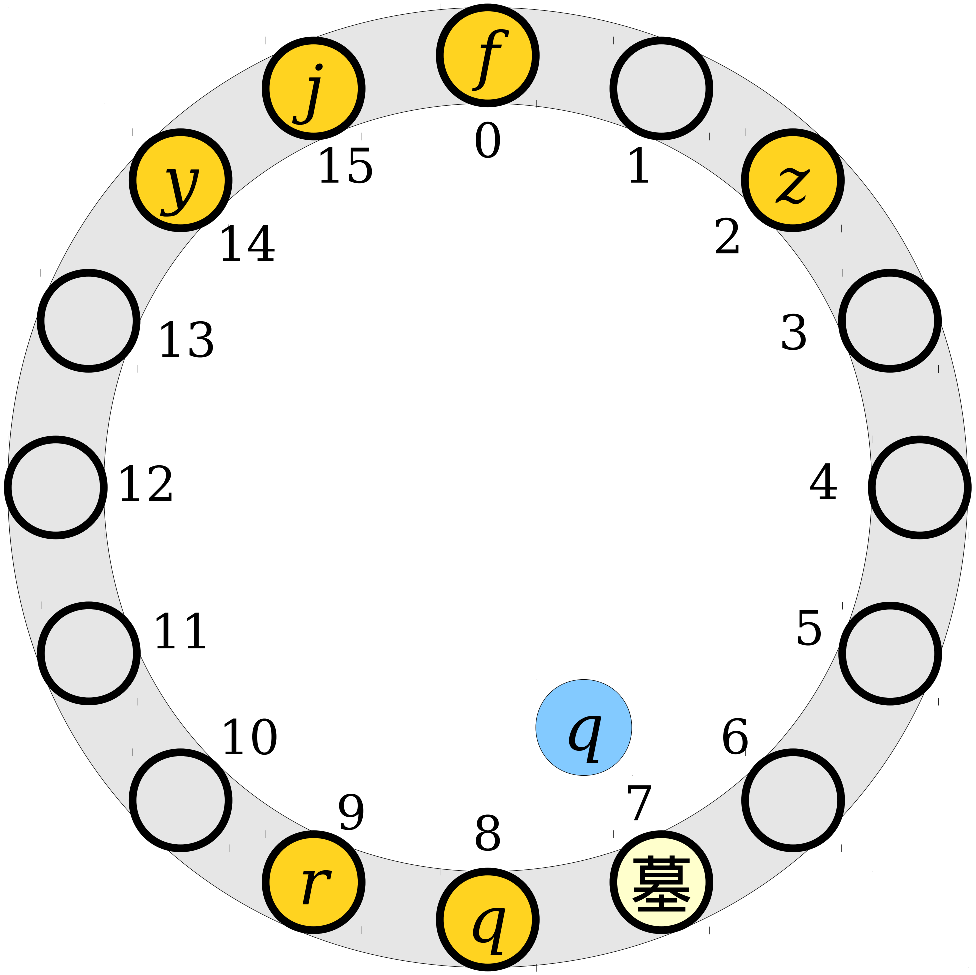
Linear Probing

- Be careful, though, to make sure you don’t allow for duplicates in your table.



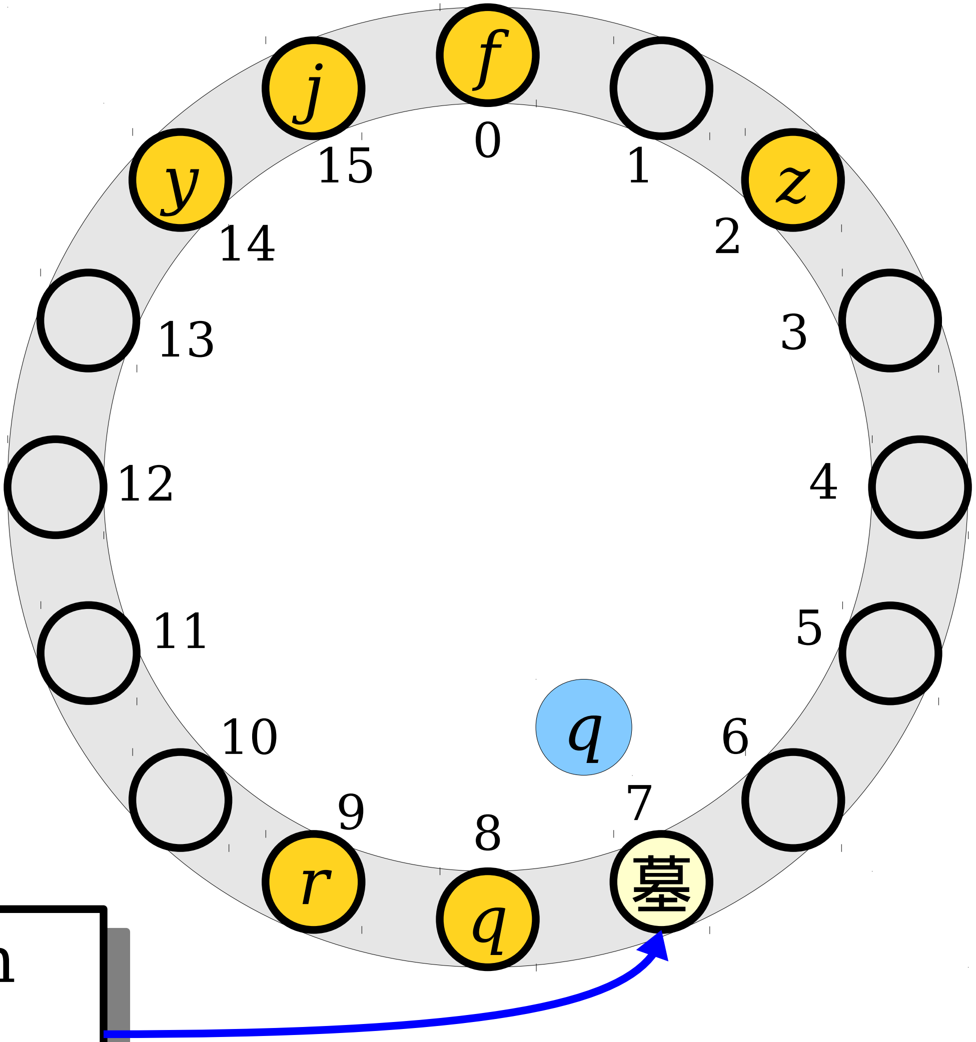
Linear Probing

- Be careful, though, to make sure you don’t allow for duplicates in your table.



Linear Probing

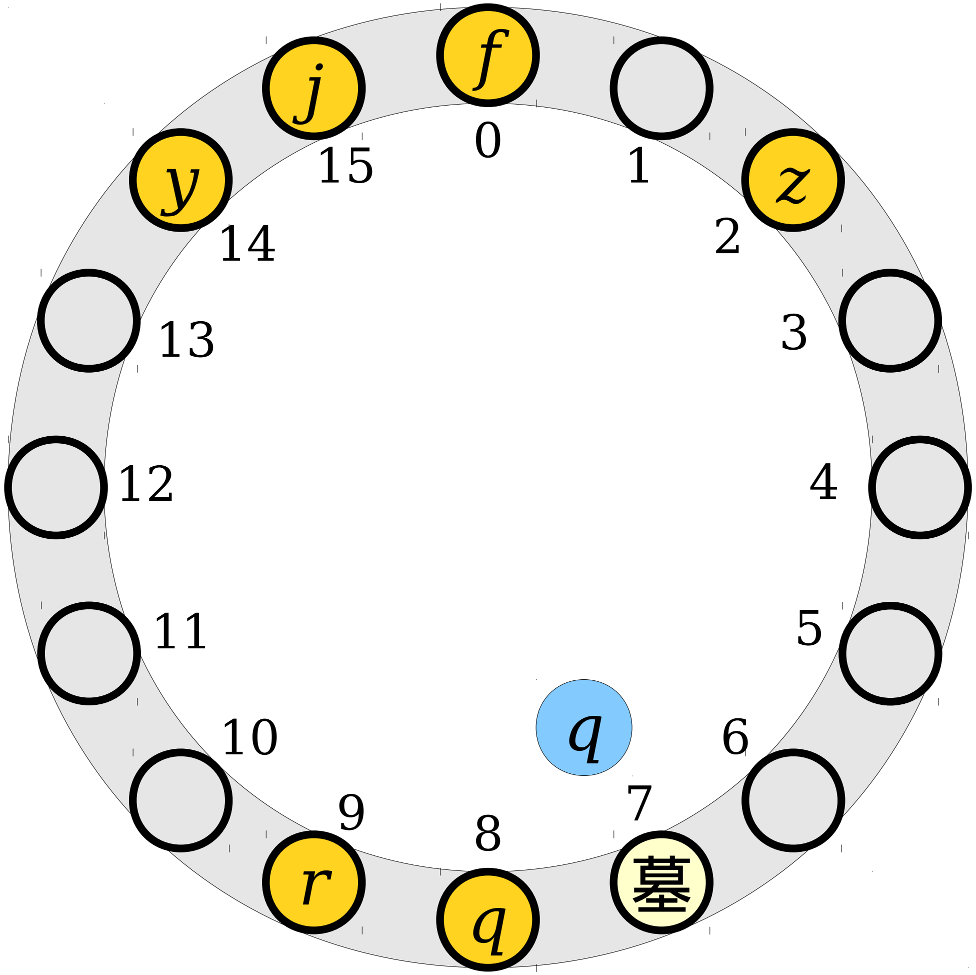
- Be careful, though, to make sure you don’t allow for duplicates in your table.



Don’t put *q* in this slot – it already exists!

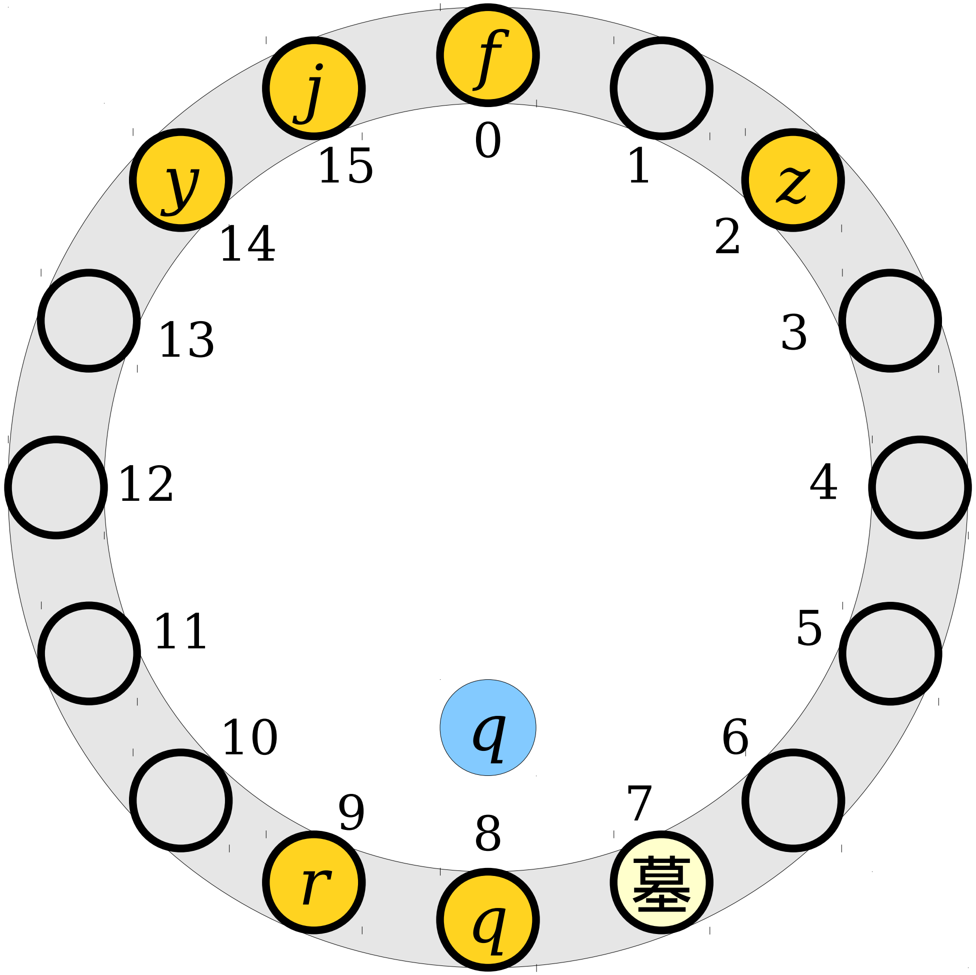
Linear Probing

- Be careful, though, to make sure you don't allow for duplicates in your table.



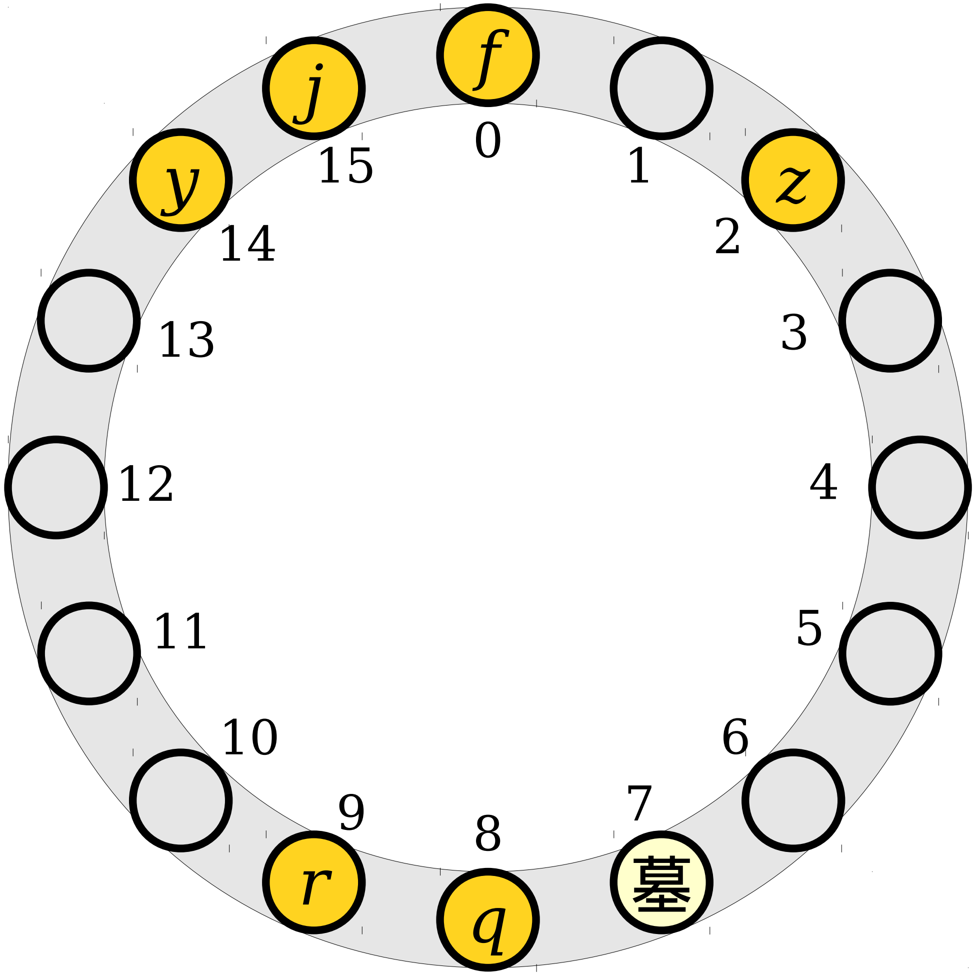
Linear Probing

- Be careful, though, to make sure you don't allow for duplicates in your table.



Linear Probing

- Be careful, though, to make sure you don't allow for duplicates in your table.



Linear Probing

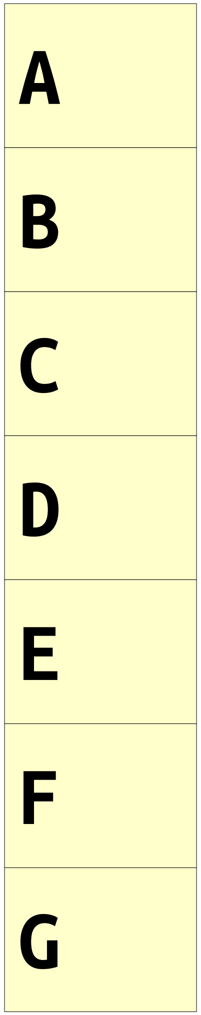
- To search for an item, jump to the slot given by its hash code, then keep moving to the next slot, wrapping around if necessary, until you find the item or a blank slot.
- To insert an item that doesn’t already exist in the table, start at the slot for its hash code and move until you find a blank spot or tombstone.
- To remove an item, search for it and replace it with a tombstone.

How Fast is Linear Probing?

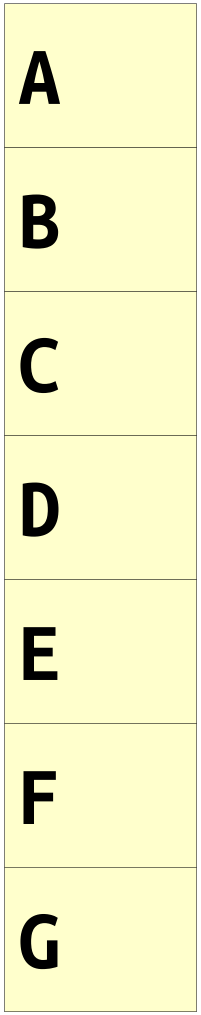
- ***Recall:*** The load factor of a hash table, denoted α, is the ratio of the number of items in the table to the number of slots.
- For any fixed value $\alpha < 1$, the expected cost of a lookup in a linear probing table is $O(1)$, assuming you have a good hash function.
- This is the same big-O cost as a chained hash table, though with a totally different strategy!

- We have seven elements to insert into a linear probing table. They’re all shown to the right.

- We have seven elements to insert into a linear probing table. They’re all shown to the right.



- We have seven elements to insert into a linear probing table. They’re all shown to the right.
- Each number indicates the hash code for the element.



A

B

C

D

E

F

G

- We have seven elements to insert into a linear probing table. They’re all shown to the right.
- Each number indicates the hash code for the element.

A (5)

B (5)

C (5)

D (8)

E (7)

F (6)

G (5)

- We have seven elements to insert into a linear probing table. They’re all shown to the right.
- Each number indicates the hash code for the element.
- If we insert them in alphabetical order, what does the final table look like?

A (5)

B (5)

C (5)

D (8)

E (7)

F (6)

G (5)

- We have seven elements to insert into a linear probing table. They’re all shown to the right.
- Each number indicates the hash code for the element.
- If we insert them in alphabetical order, what does the final table look like?

A (5)

B (5)

C (5)

D (8)

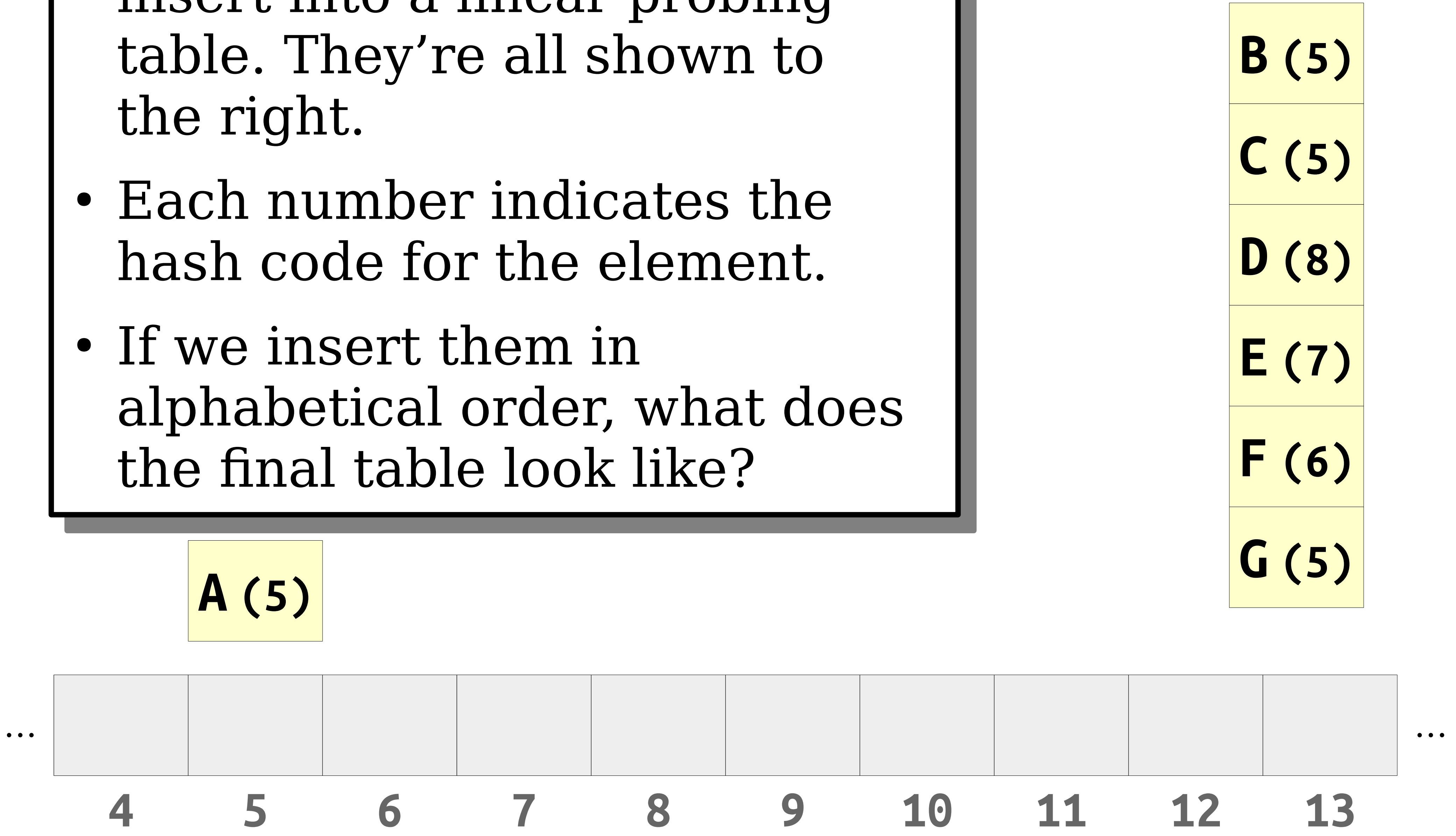
E (7)

F (6)

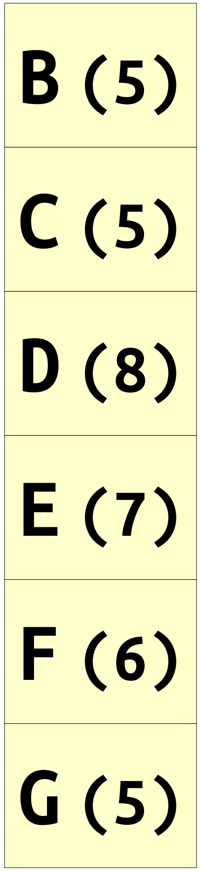
G (5)

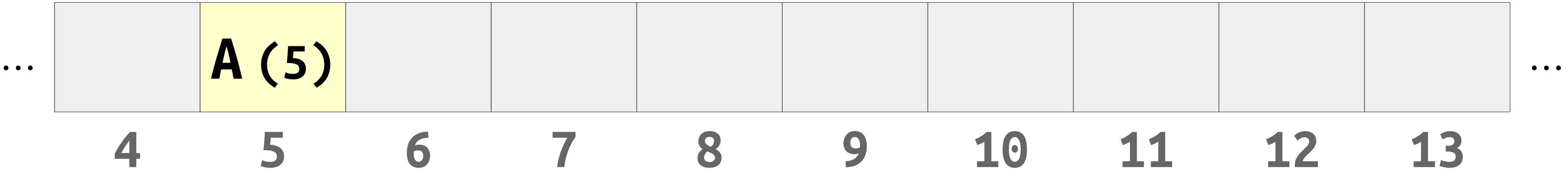
…											…
	4	5	6	7	8	9	10	11	12	13	

- We have seven elements to insert into a linear probing table. They’re all shown to the right.
- Each number indicates the hash code for the element.
- If we insert them in alphabetical order, what does the final table look like?

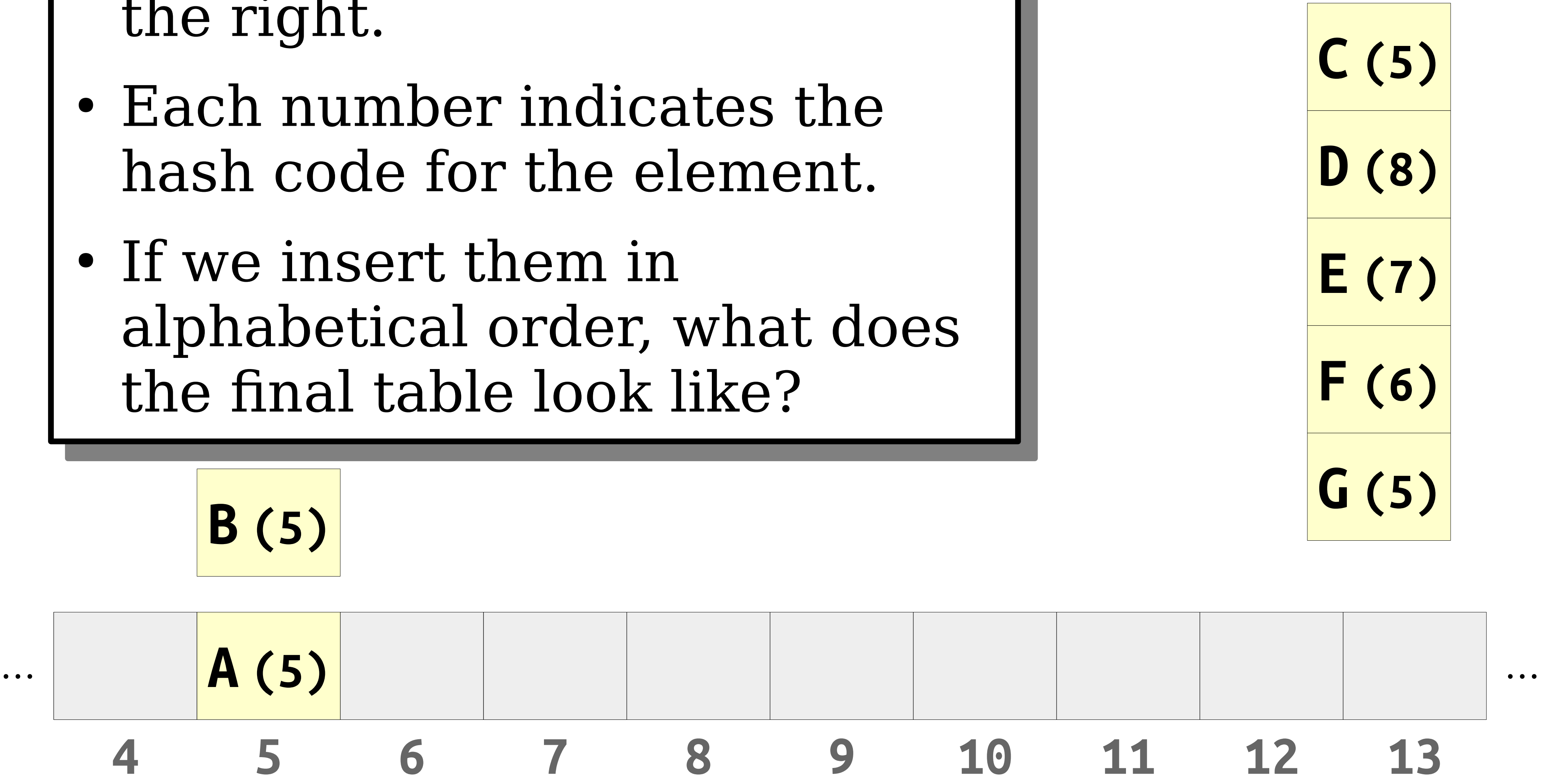


- We have seven elements to insert into a linear probing table. They’re all shown to the right.
- Each number indicates the hash code for the element.
- If we insert them in alphabetical order, what does the final table look like?





- We have seven elements to insert into a linear probing table. They're all shown to the right.
- Each number indicates the hash code for the element.
- If we insert them in alphabetical order, what does the final table look like?



C (5)

D (8)

E (7)

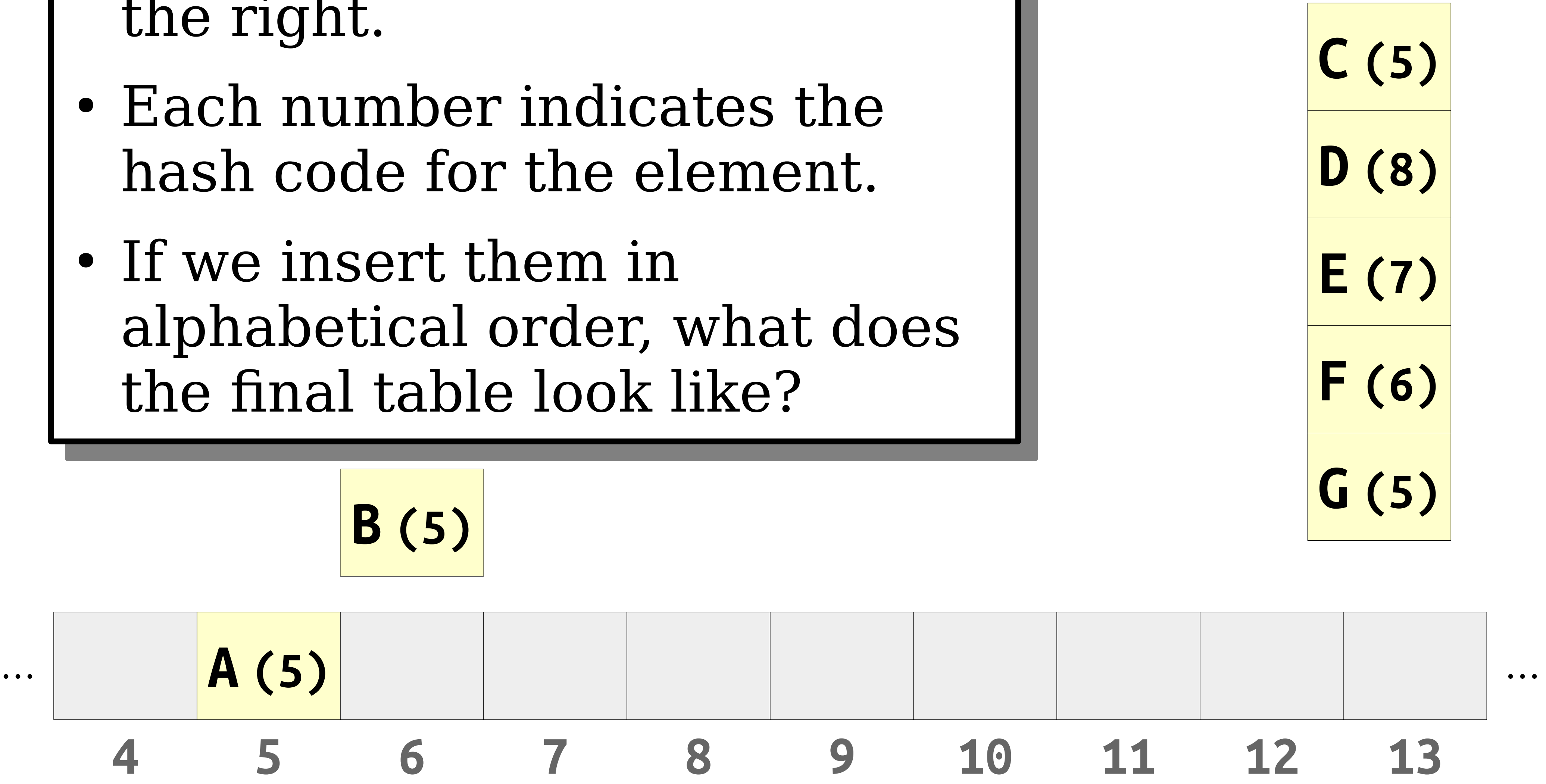
F (6)

G (5)

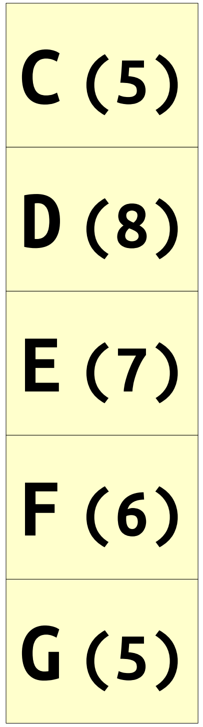
B (5)

…		A (5)									…
	4	5	6	7	8	9	10	11	12	13	

- We have seven elements to insert into a linear probing table. They’re all shown to the right.
- Each number indicates the hash code for the element.
- If we insert them in alphabetical order, what does the final table look like?



- We have seven elements to insert into a linear probing table. They're all shown to the right.
- Each number indicates the hash code for the element.
- If we insert them in alphabetical order, what does the final table look like?



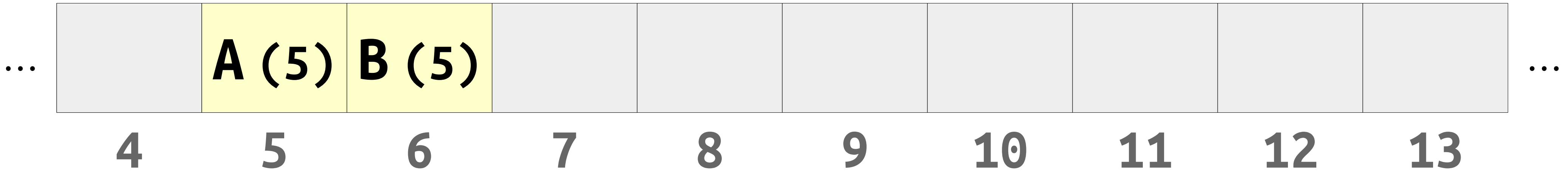
C (5)

D (8)

E (7)

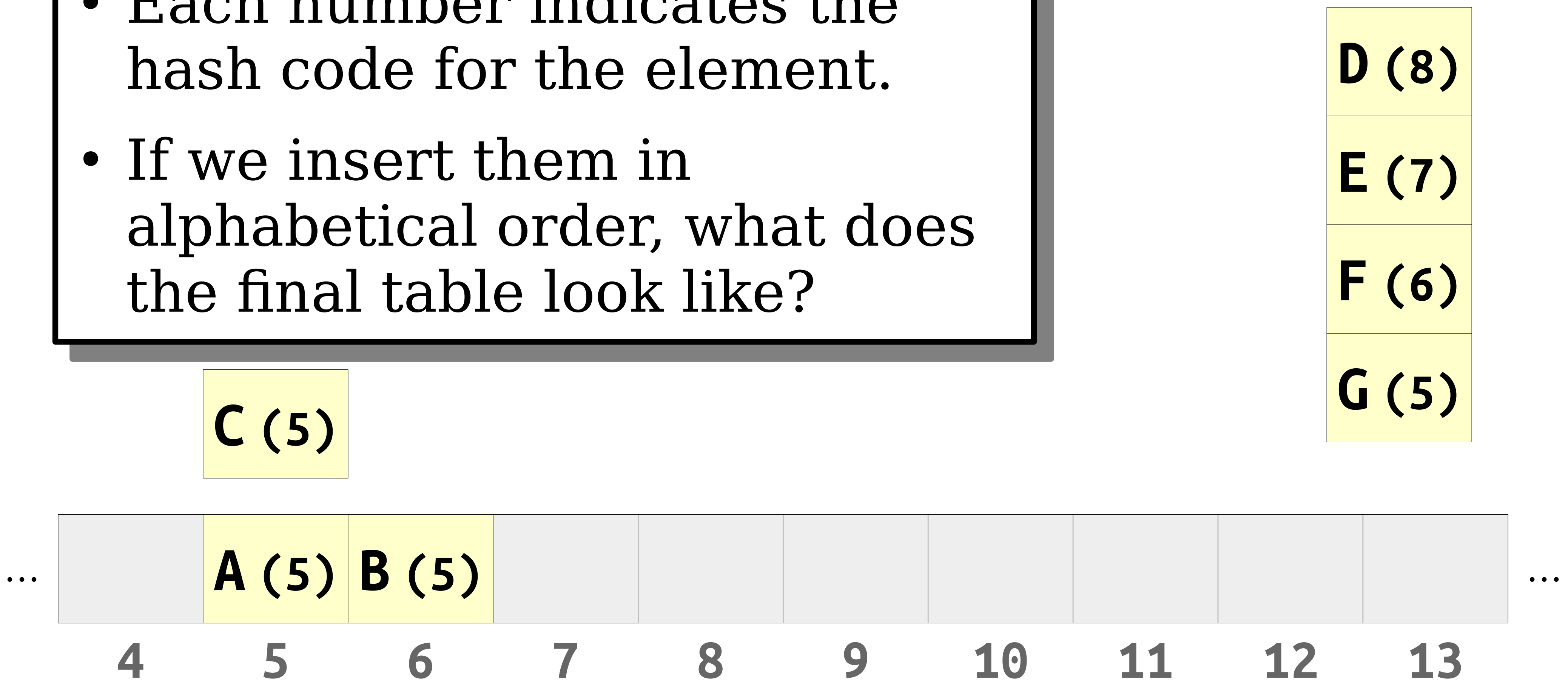
F (6)

G (5)

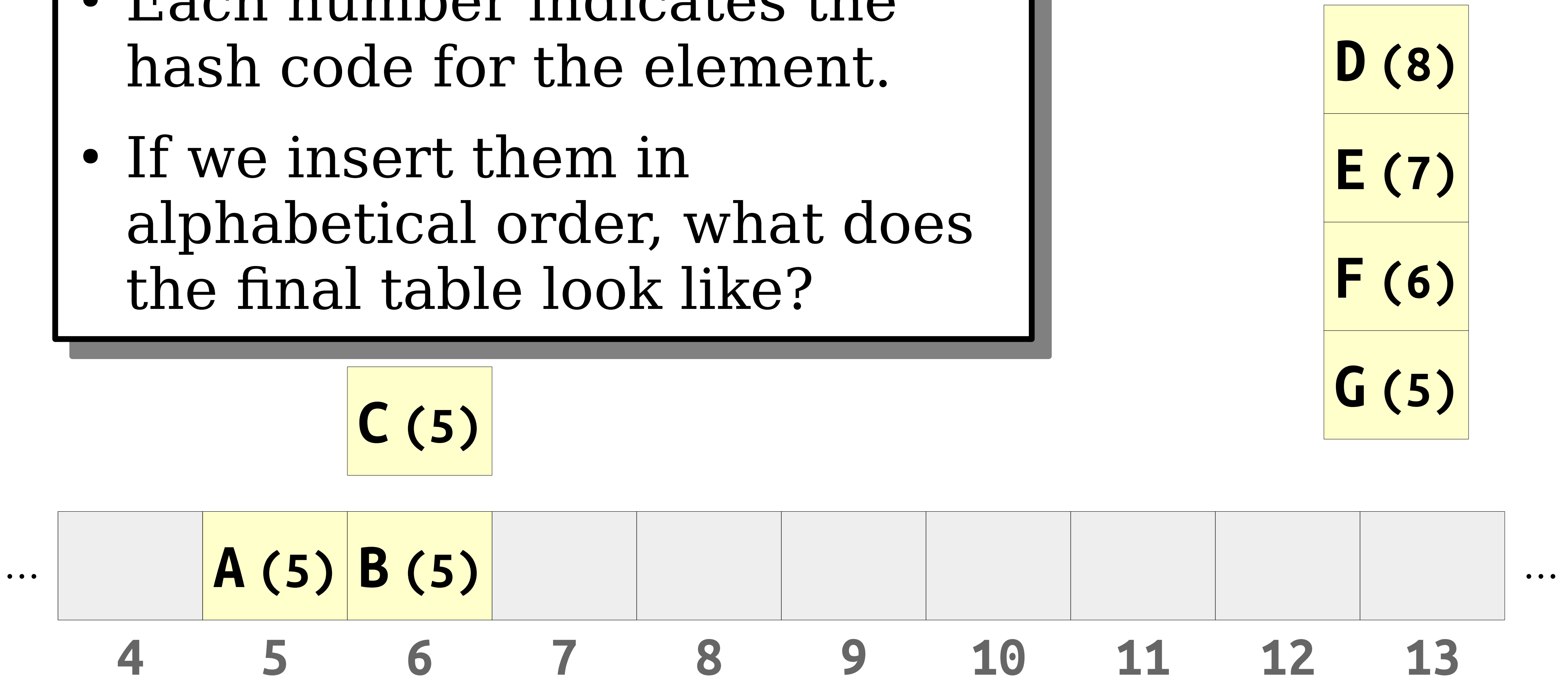


…		A (5)	B (5)								…
	4	5	6	7	8	9	10	11	12	13	

- We have seven elements to insert into a linear probing table. They're all shown to the right.
- Each number indicates the hash code for the element.
- If we insert them in alphabetical order, what does the final table look like?



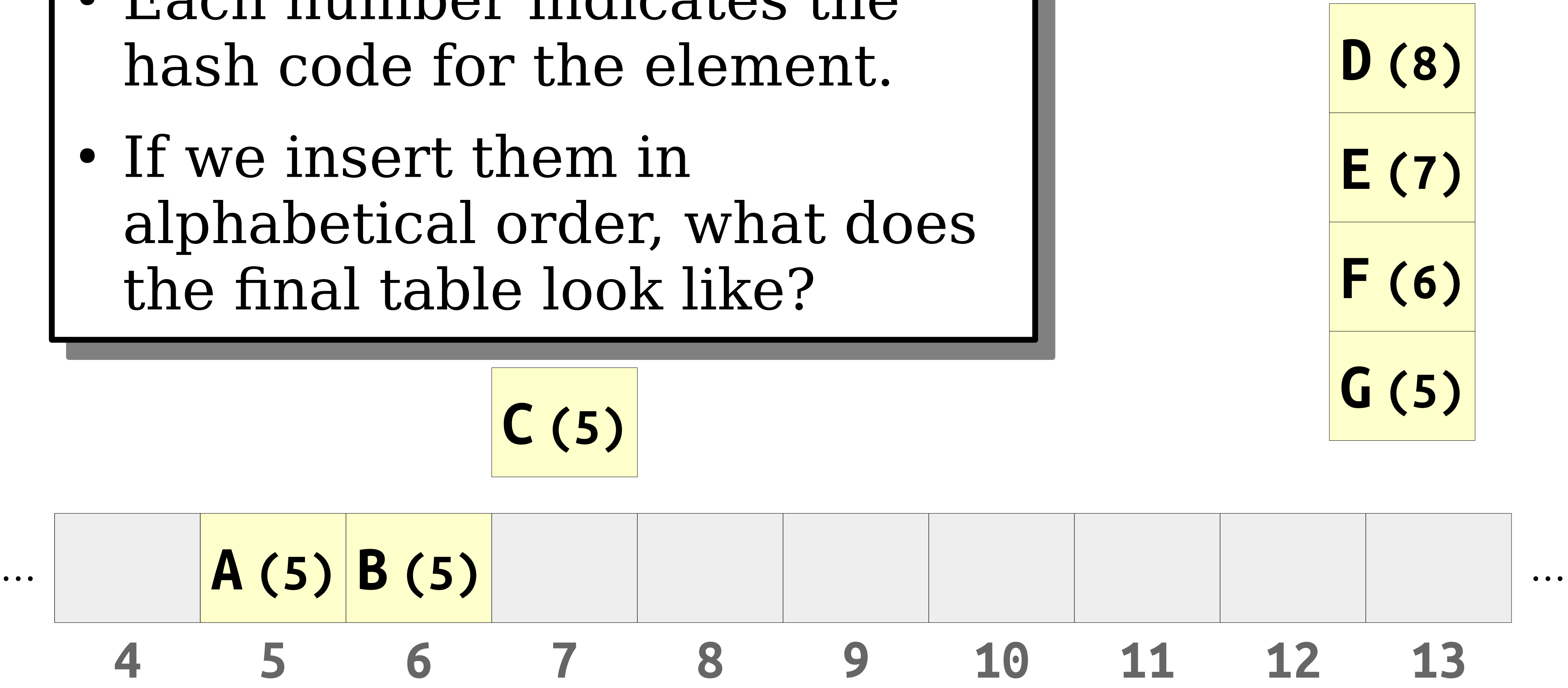
- We have seven elements to insert into a linear probing table. They're all shown to the right.
- Each number indicates the hash code for the element.
- If we insert them in alphabetical order, what does the final table look like?



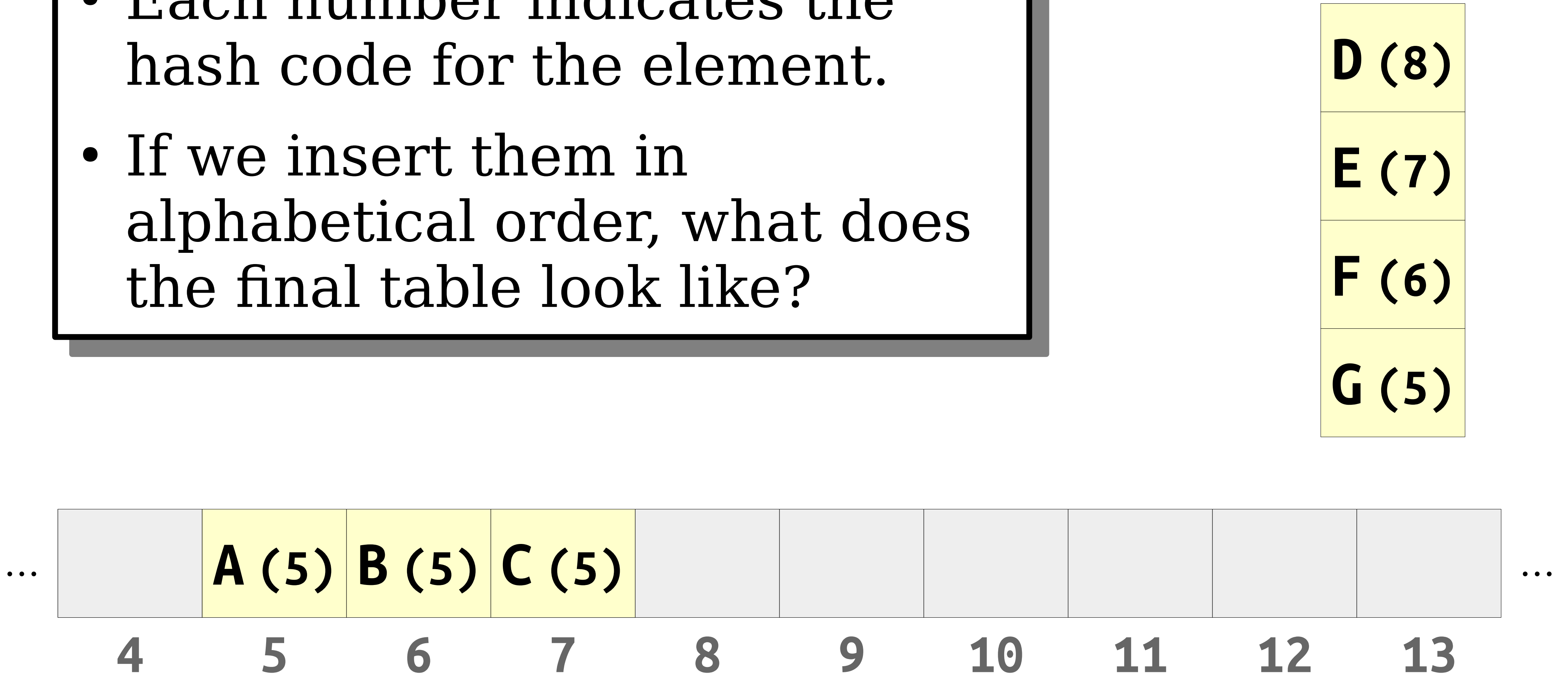
D (8)
E (7)
F (6)
G (5)
C (5)

…	4	5	6	7	8	9	10	11	12	13	…
		A (5)	B (5)								

- We have seven elements to insert into a linear probing table. They're all shown to the right.
- Each number indicates the hash code for the element.
- If we insert them in alphabetical order, what does the final table look like?



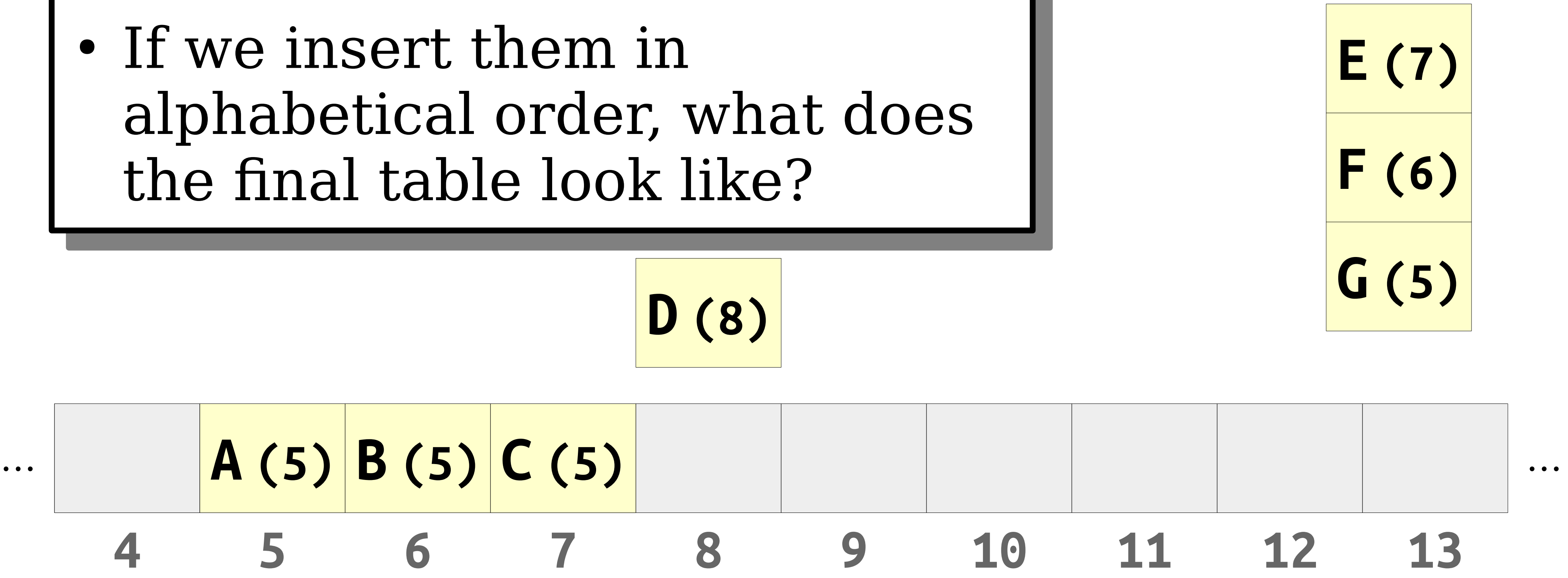
- We have seven elements to insert into a linear probing table. They’re all shown to the right.
- Each number indicates the hash code for the element.
- If we insert them in alphabetical order, what does the final table look like?



D (8)
E (7)
F (6)
G (5)

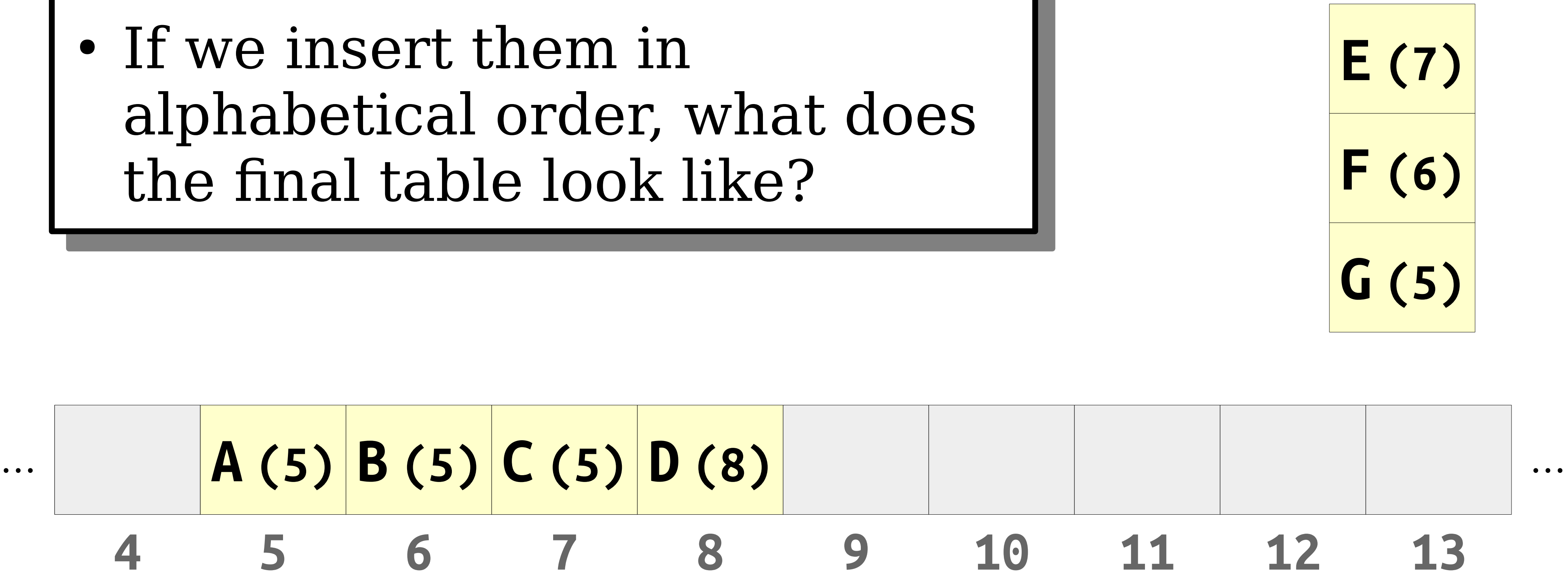
…	4	5	6	7	8	9	10	11	12	13	…
		A (5)	B (5)	C (5)							

- We have seven elements to insert into a linear probing table. They're all shown to the right.
- Each number indicates the hash code for the element.
- If we insert them in alphabetical order, what does the final table look like?



…		A (5)	B (5)	C (5)							…
	4	5	6	7	8	9	10	11	12	13	

- We have seven elements to insert into a linear probing table. They’re all shown to the right.
- Each number indicates the hash code for the element.
- If we insert them in alphabetical order, what does the final table look like?



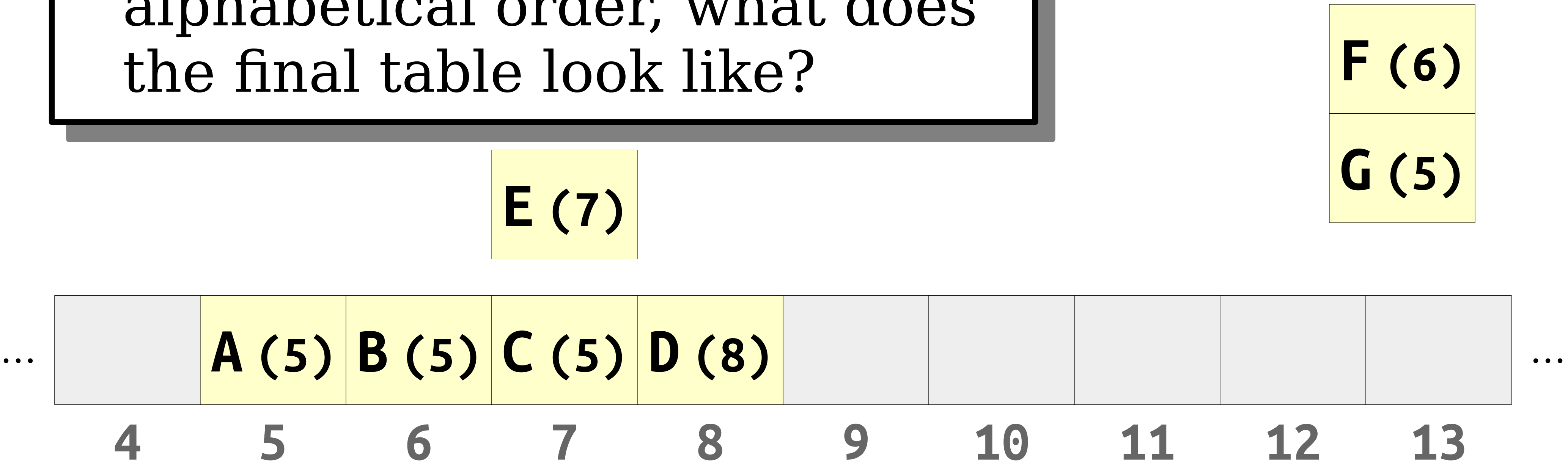
E (7)

F (6)

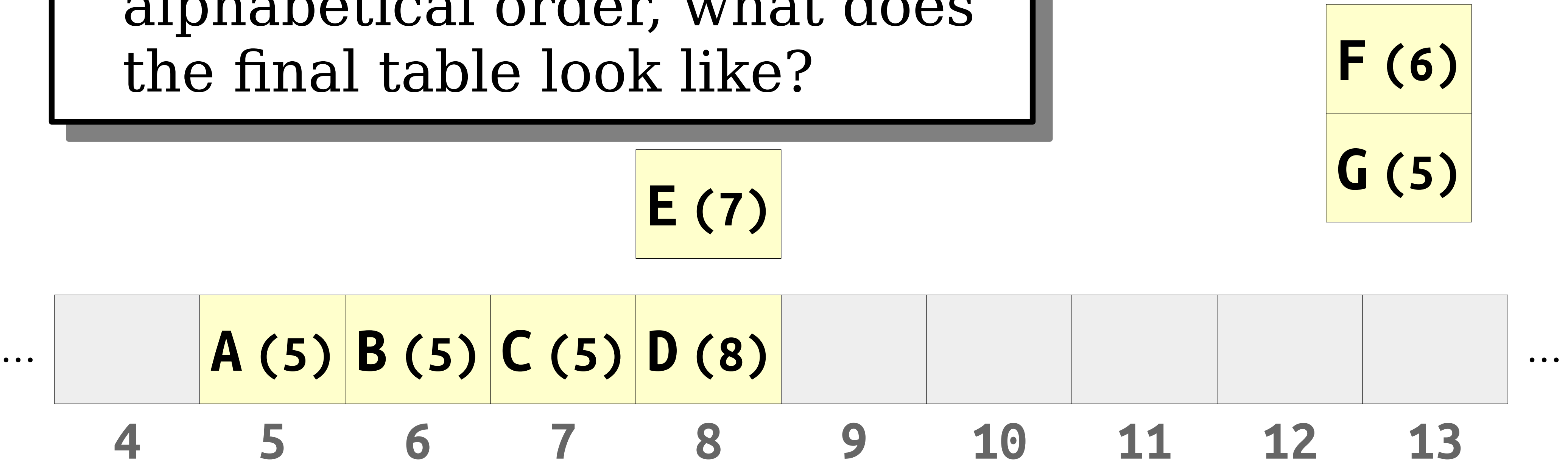
G (5)

…		A (5)	B (5)	C (5)	D (8)						…
	4	5	6	7	8	9	10	11	12	13	

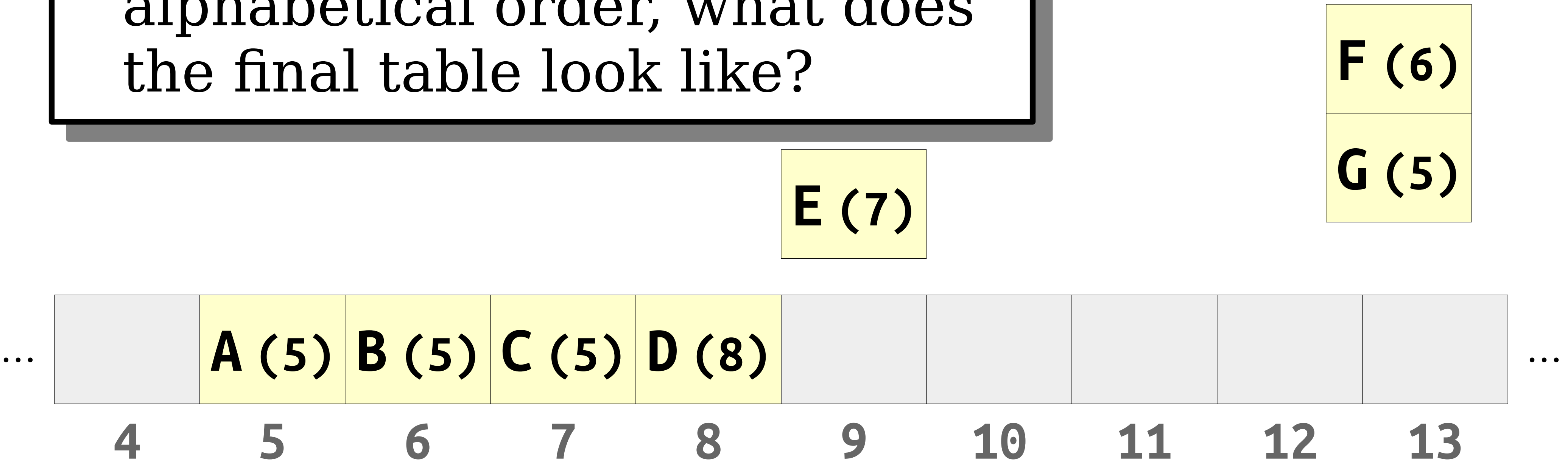
- We have seven elements to insert into a linear probing table. They’re all shown to the right.
- Each number indicates the hash code for the element.
- If we insert them in alphabetical order, what does the final table look like?



- We have seven elements to insert into a linear probing table. They’re all shown to the right.
- Each number indicates the hash code for the element.
- If we insert them in alphabetical order, what does the final table look like?



- We have seven elements to insert into a linear probing table. They’re all shown to the right.
- Each number indicates the hash code for the element.
- If we insert them in alphabetical order, what does the final table look like?



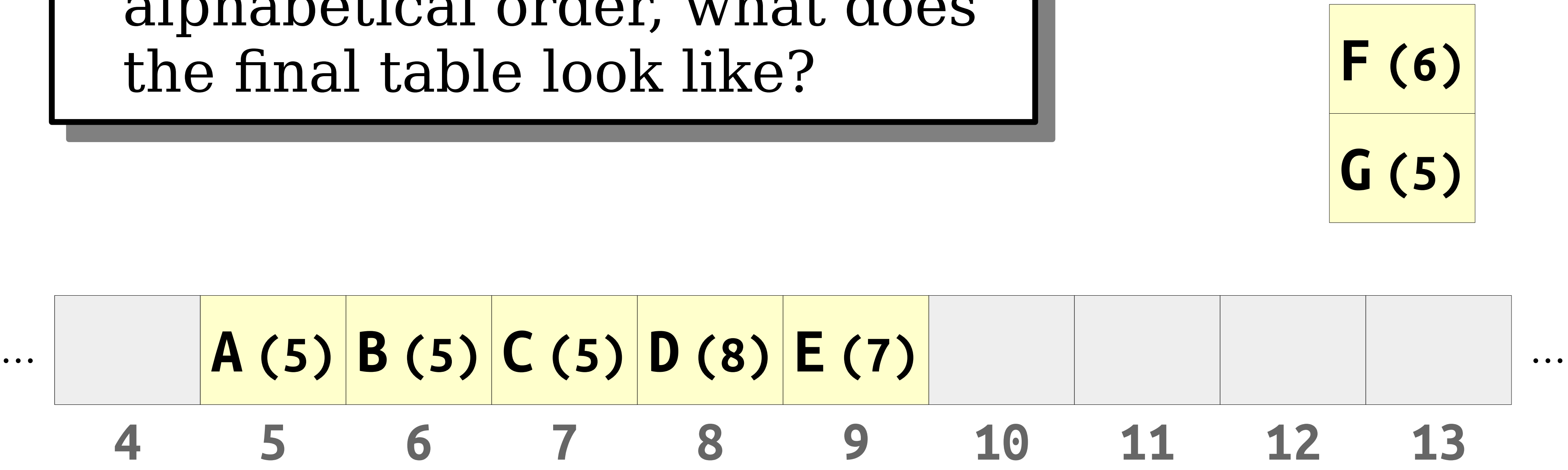
F (6)

G (5)

E (7)

…		A (5)	B (5)	C (5)	D (8)						…
	4	5	6	7	8	9	10	11	12	13	

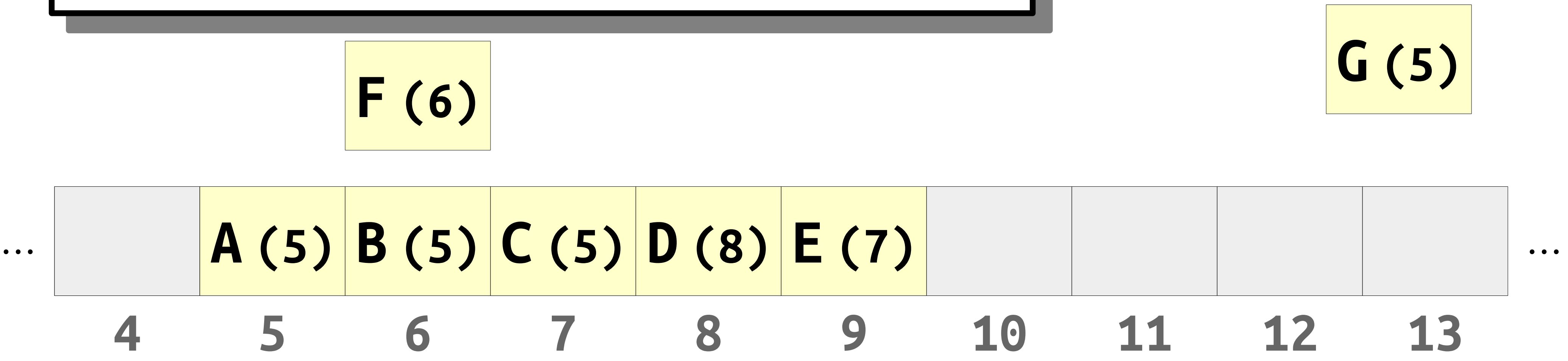
- We have seven elements to insert into a linear probing table. They’re all shown to the right.
- Each number indicates the hash code for the element.
- If we insert them in alphabetical order, what does the final table look like?



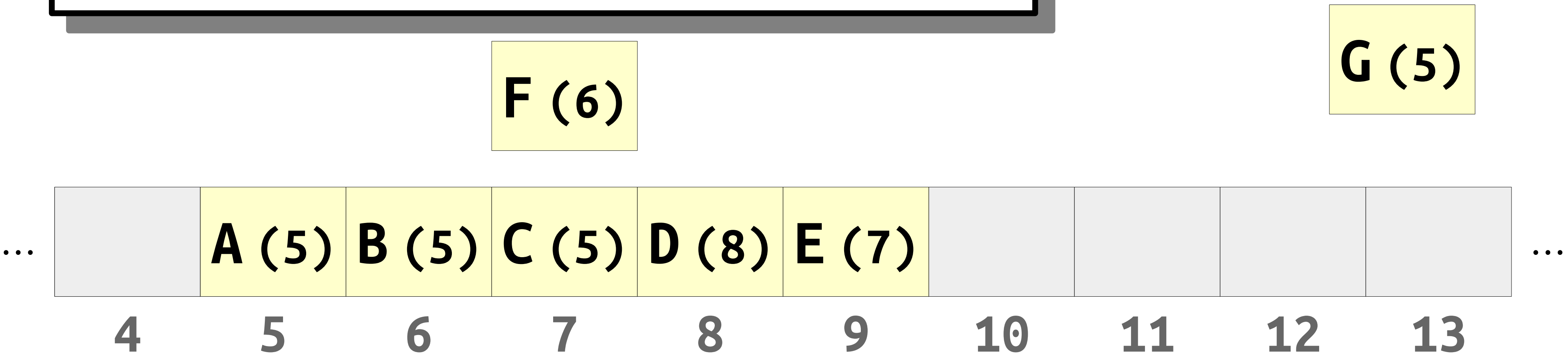
F (6)
G (5)

…	4	5	6	7	8	9	10	11	12	13	…
…		A (5)	B (5)	C (5)	D (8)	E (7)					…

- We have seven elements to insert into a linear probing table. They’re all shown to the right.
- Each number indicates the hash code for the element.
- If we insert them in alphabetical order, what does the final table look like?



- We have seven elements to insert into a linear probing table. They’re all shown to the right.
- Each number indicates the hash code for the element.
- If we insert them in alphabetical order, what does the final table look like?

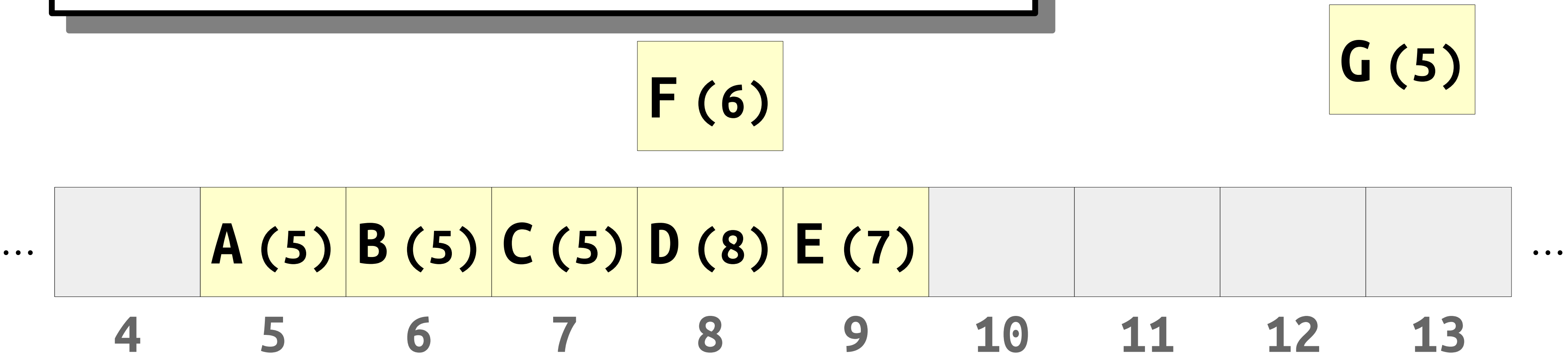


F (6)

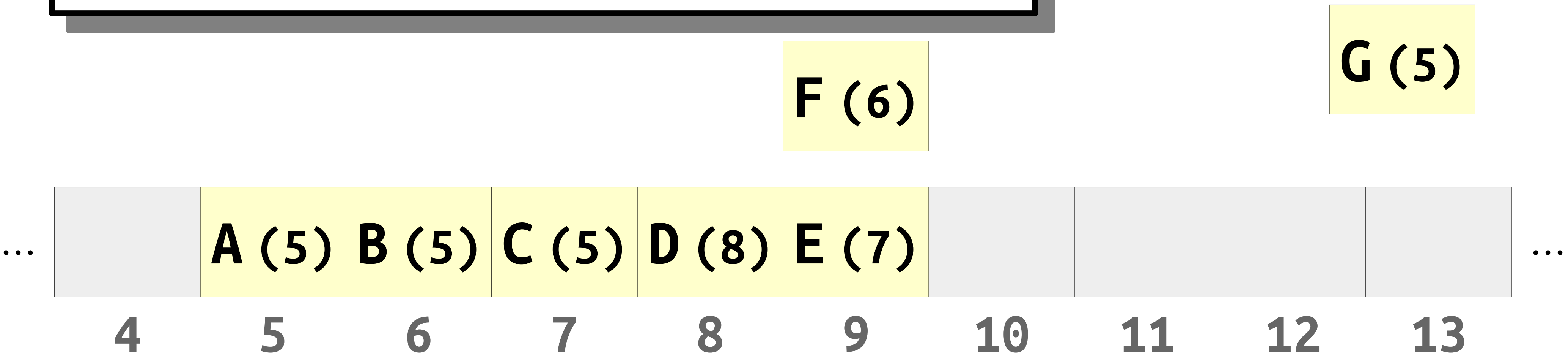
G (5)

…		A (5)	B (5)	C (5)	D (8)	E (7)					…
	4	5	6	7	8	9	10	11	12	13	

- We have seven elements to insert into a linear probing table. They're all shown to the right.
- Each number indicates the hash code for the element.
- If we insert them in alphabetical order, what does the final table look like?



- We have seven elements to insert into a linear probing table. They’re all shown to the right.
- Each number indicates the hash code for the element.
- If we insert them in alphabetical order, what does the final table look like?

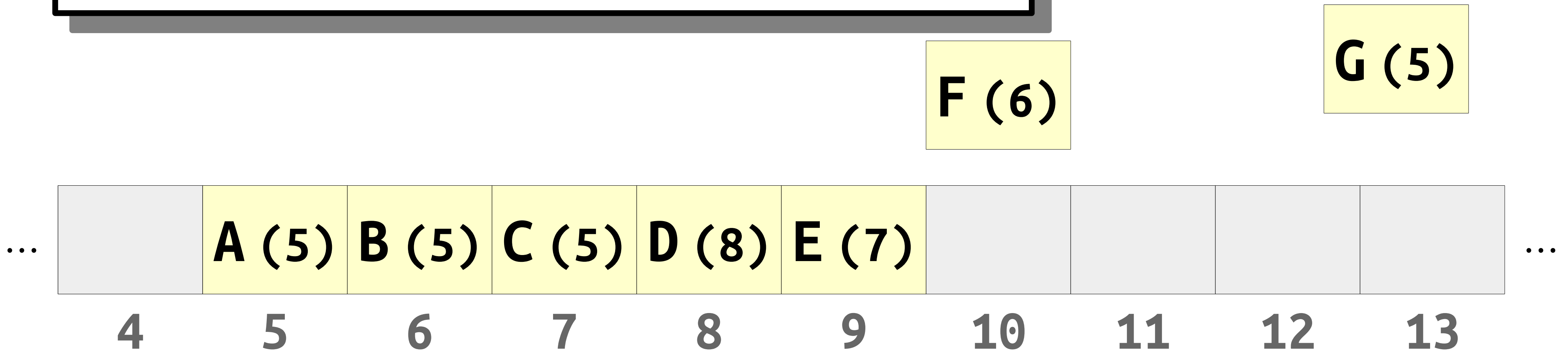


F (6)

G (5)

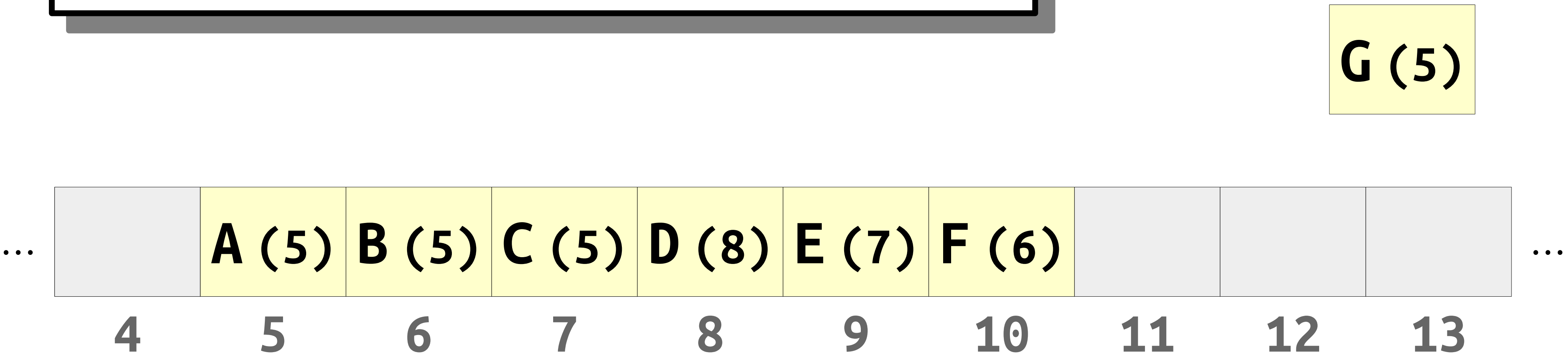
…		A (5)	B (5)	C (5)	D (8)	E (7)					…
	4	5	6	7	8	9	10	11	12	13	

- We have seven elements to insert into a linear probing table. They’re all shown to the right.
- Each number indicates the hash code for the element.
- If we insert them in alphabetical order, what does the final table look like?



…		A (5)	B (5)	C (5)	D (8)	E (7)					…
	4	5	6	7	8	9	10	11	12	13	

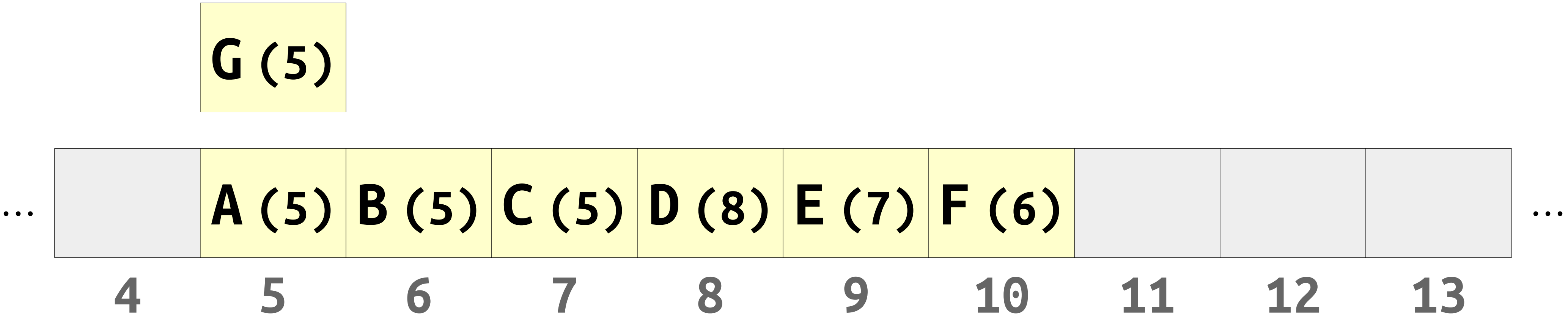
- We have seven elements to insert into a linear probing table. They’re all shown to the right.
- Each number indicates the hash code for the element.
- If we insert them in alphabetical order, what does the final table look like?



G (5)

…		A (5)	B (5)	C (5)	D (8)	E (7)	F (6)				…
	4	5	6	7	8	9	10	11	12	13	

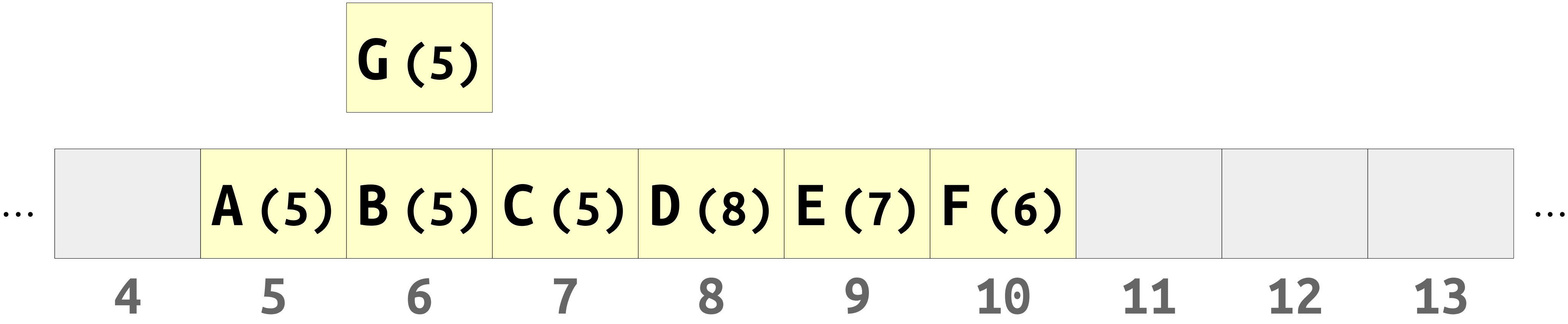
- We have seven elements to insert into a linear probing table. They’re all shown to the right.
- Each number indicates the hash code for the element.
- If we insert them in alphabetical order, what does the final table look like?



G (5)

…	4	5	6	7	8	9	10	11	12	13	…
…		A (5)	B (5)	C (5)	D (8)	E (7)	F (6)				…

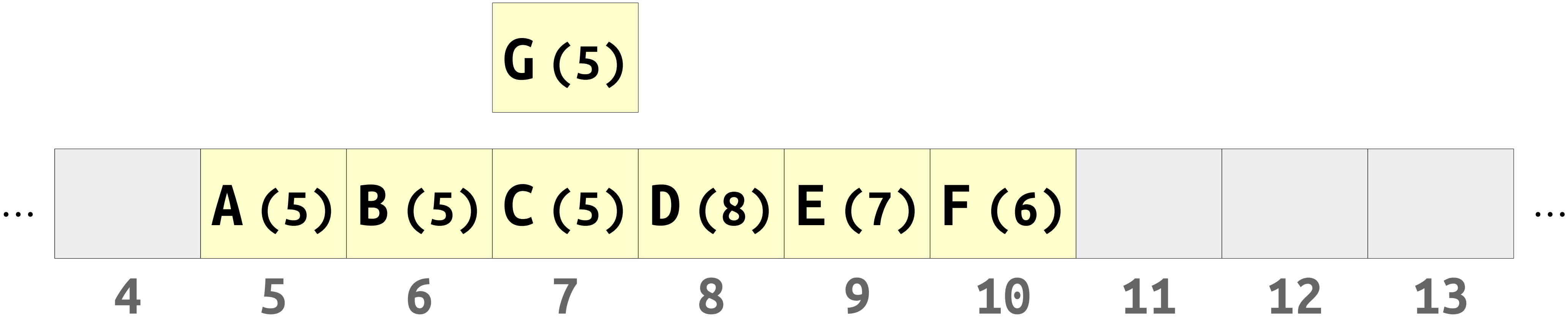
- We have seven elements to insert into a linear probing table. They’re all shown to the right.
- Each number indicates the hash code for the element.
- If we insert them in alphabetical order, what does the final table look like?



G (5)

…		A (5)	B (5)	C (5)	D (8)	E (7)	F (6)				…
	4	5	6	7	8	9	10	11	12	13	

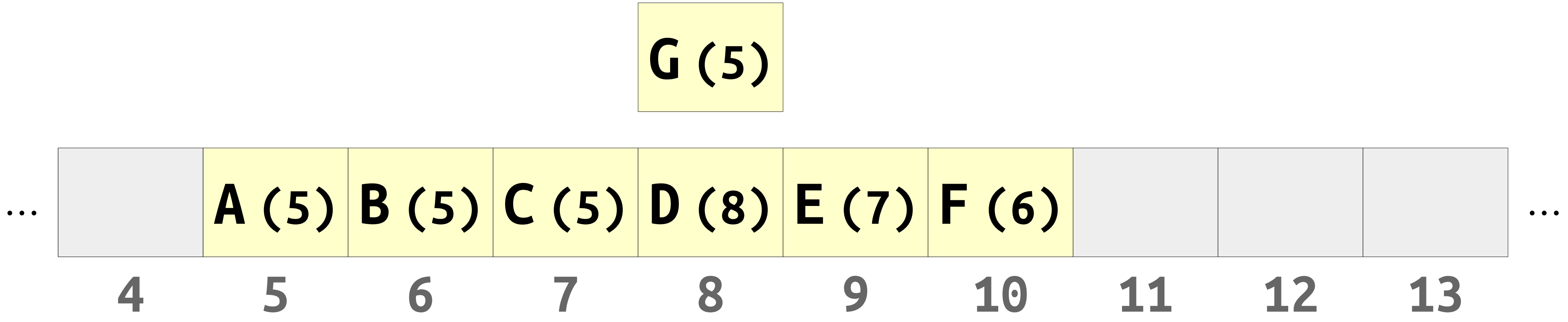
- We have seven elements to insert into a linear probing table. They’re all shown to the right.
- Each number indicates the hash code for the element.
- If we insert them in alphabetical order, what does the final table look like?



G (5)

…		A (5)	B (5)	C (5)	D (8)	E (7)	F (6)				…
	4	5	6	7	8	9	10	11	12	13	

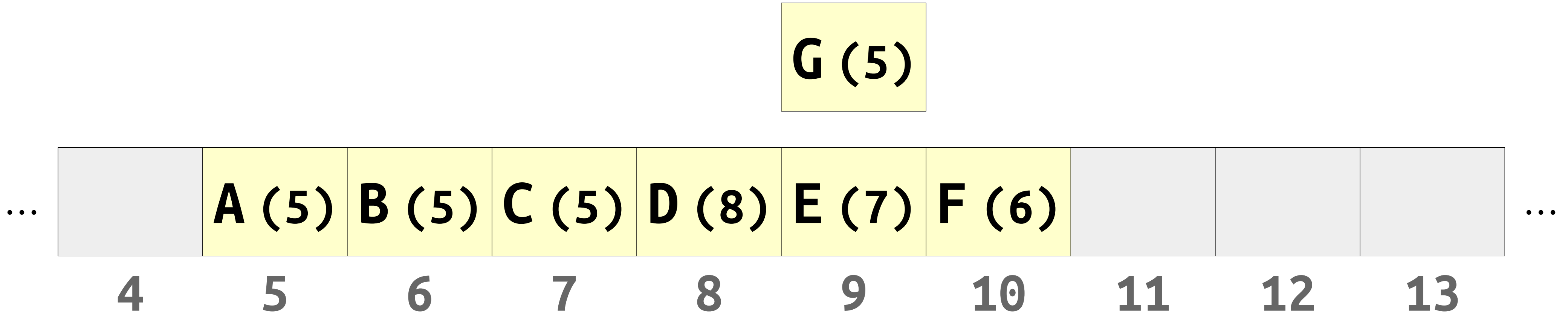
- We have seven elements to insert into a linear probing table. They’re all shown to the right.
- Each number indicates the hash code for the element.
- If we insert them in alphabetical order, what does the final table look like?



G (5)

…										…
		A (5)	B (5)	C (5)	D (8)	E (7)	F (6)			
	4	5	6	7	8	9	10	11	12	13

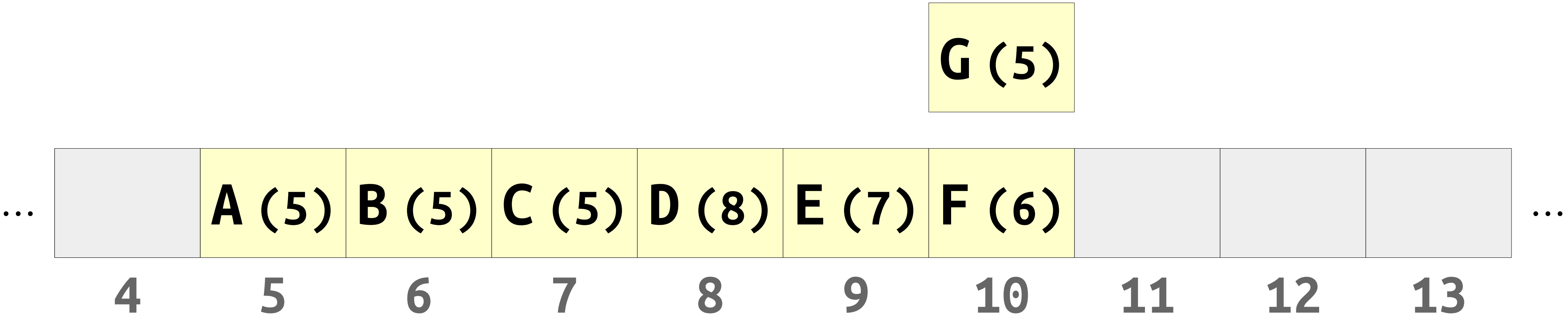
- We have seven elements to insert into a linear probing table. They’re all shown to the right.
- Each number indicates the hash code for the element.
- If we insert them in alphabetical order, what does the final table look like?



G (5)

…		A (5)	B (5)	C (5)	D (8)	E (7)	F (6)				…
	4	5	6	7	8	9	10	11	12	13	

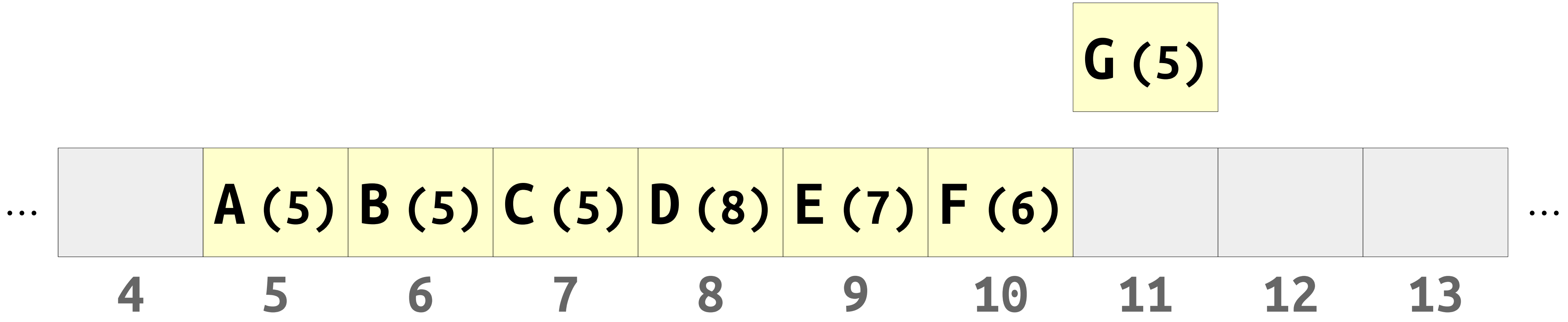
- We have seven elements to insert into a linear probing table. They’re all shown to the right.
- Each number indicates the hash code for the element.
- If we insert them in alphabetical order, what does the final table look like?



G (5)

	4	5	6	7	8	9	10	11	12	13	
…		A (5)	B (5)	C (5)	D (8)	E (7)	F (6)				…

- We have seven elements to insert into a linear probing table. They’re all shown to the right.
- Each number indicates the hash code for the element.
- If we insert them in alphabetical order, what does the final table look like?



G (5)

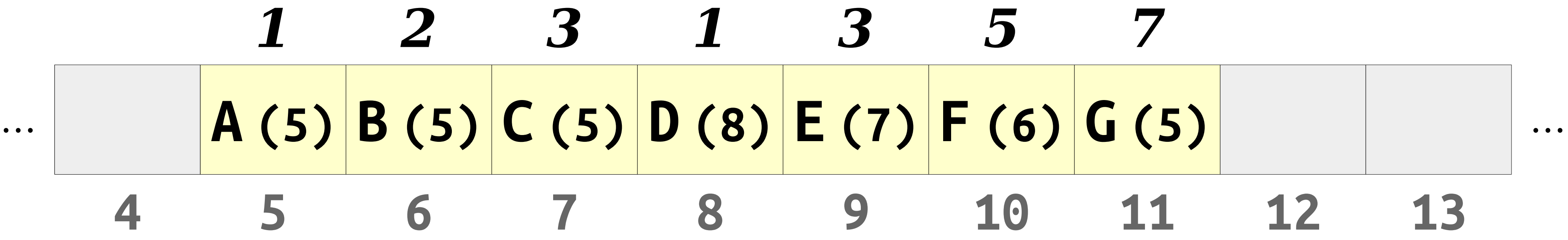
…		A (5)	B (5)	C (5)	D (8)	E (7)	F (6)				…
	4	5	6	7	8	9	10	11	12	13	

- We have seven elements to insert into a linear probing table. They’re all shown to the right.
- Each number indicates the hash code for the element.
- If we insert them in alphabetical order, what does the final table look like?

…		A (5)	B (5)	C (5)	D (8)	E (7)	F (6)	G (5)			…
	4	5	6	7	8	9	10	11	12	13	

A Question of Fairness

- Suppose we look up each of these elements. How many slots would we need to look at to find each of them?



		1	***2***	***3***	***1***	***3***	***5***	***7***			
…		A (5)	B (5)	C (5)	D (8)	E (7)	F (6)	G (5)			…
	4	5	6	7	8	9	10	11	12	13	

A Question of Fairness

- Suppose we look up each of these elements. How many slots would we need to look at to find each of them?
- There's a large *variance* in how long it's going to take to find things.
- How can we fix this?

		1	***2***	***3***	***1***	***3***	***5***	***7***			
…		**A (5)**	**B (5)**	**C (5)**	**D (8)**	**E (7)**	**F (6)**	**G (5)**			…
	4	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	

- ***Robin Hood hashing*** is a slight modification to linear probing.
- When we insert an element, if the element we're inserting is further from home than the current element, we displace that element to make room for the new one.

A (5)

B (5)

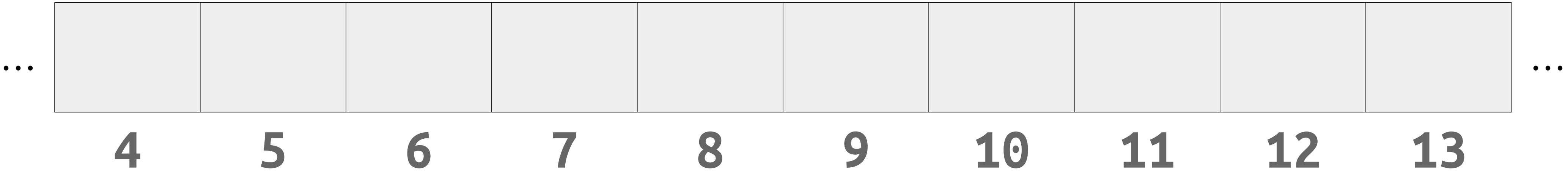
C (5)

D (8)

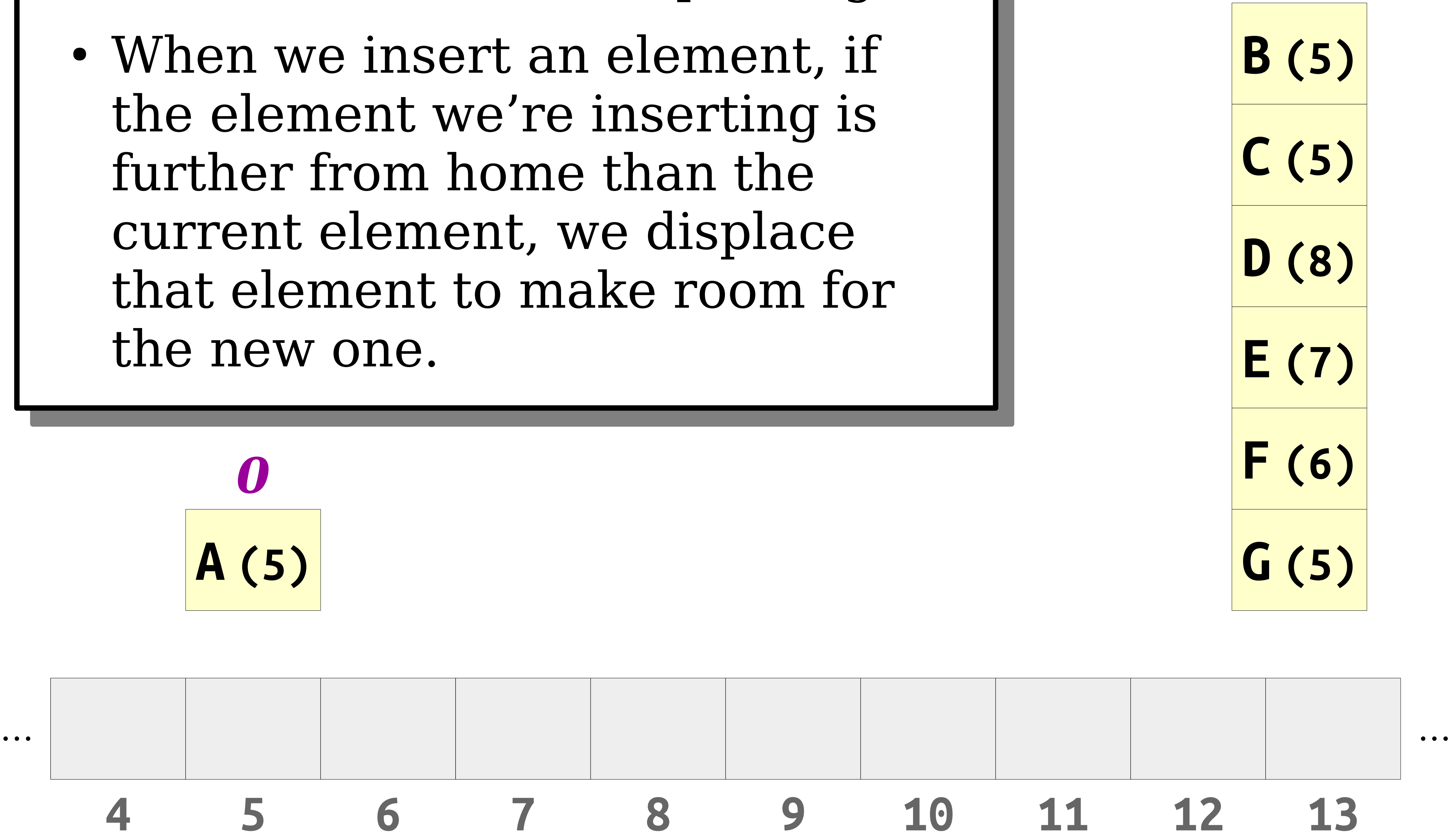
E (7)

F (6)

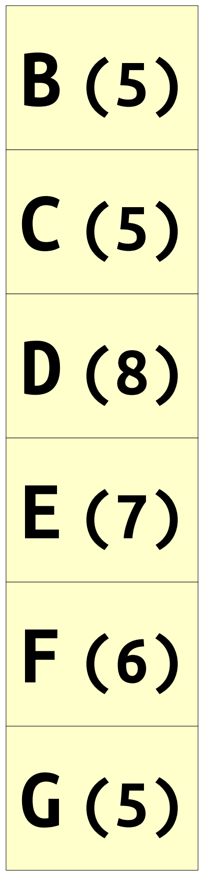
G (5)

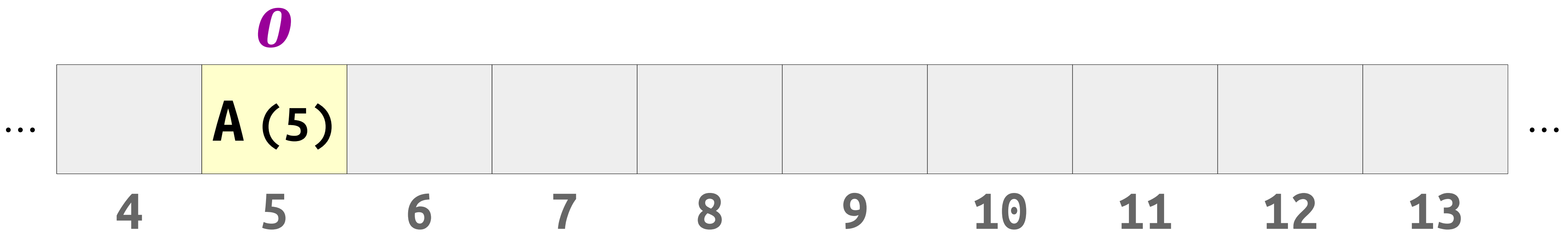


- ***Robin Hood hashing*** is a slight modification to linear probing.
- When we insert an element, if the element we're inserting is further from home than the current element, we displace that element to make room for the new one.

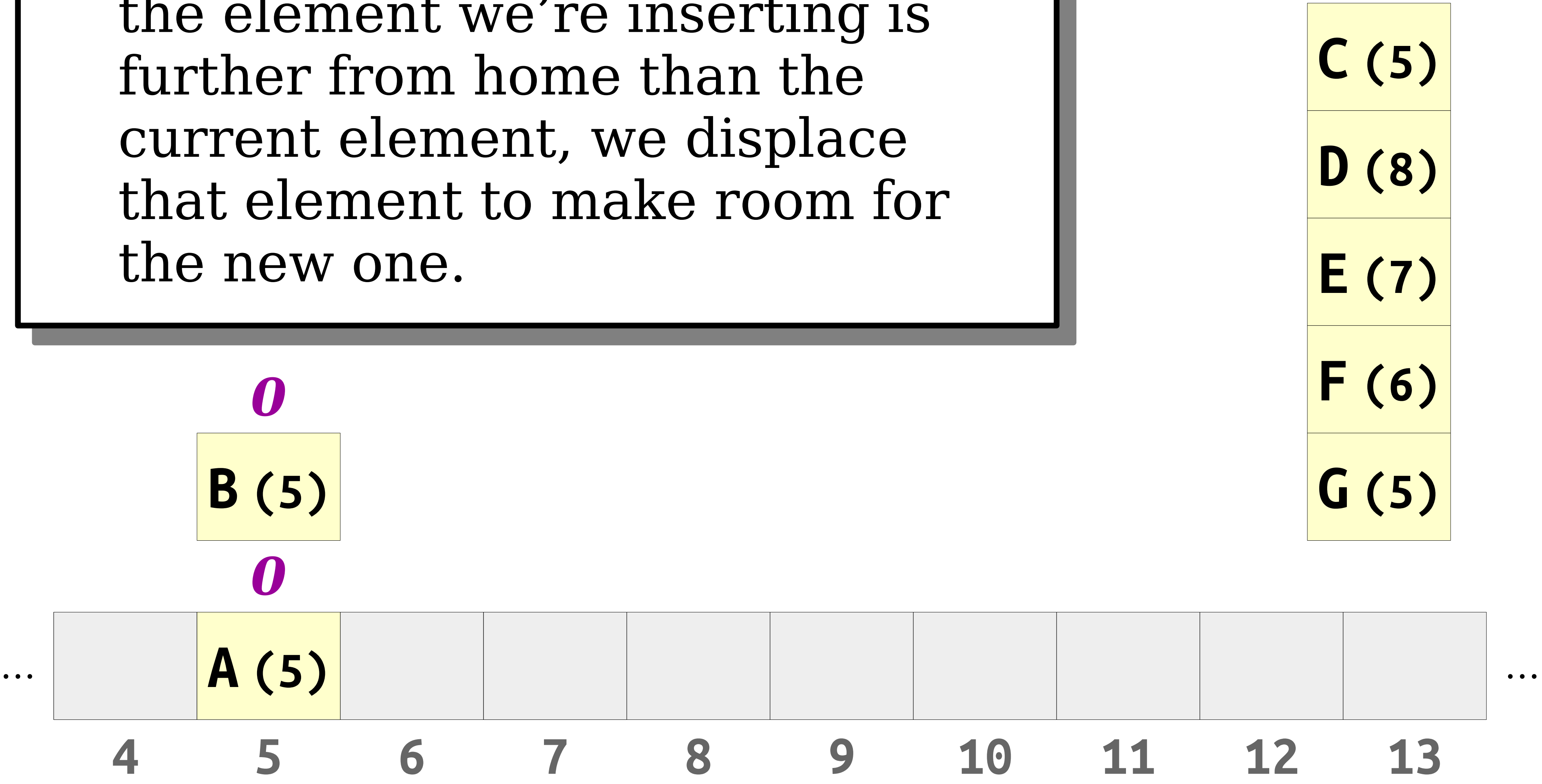


- ***Robin Hood hashing*** is a slight modification to linear probing.
- When we insert an element, if the element we're inserting is further from home than the current element, we displace that element to make room for the new one.

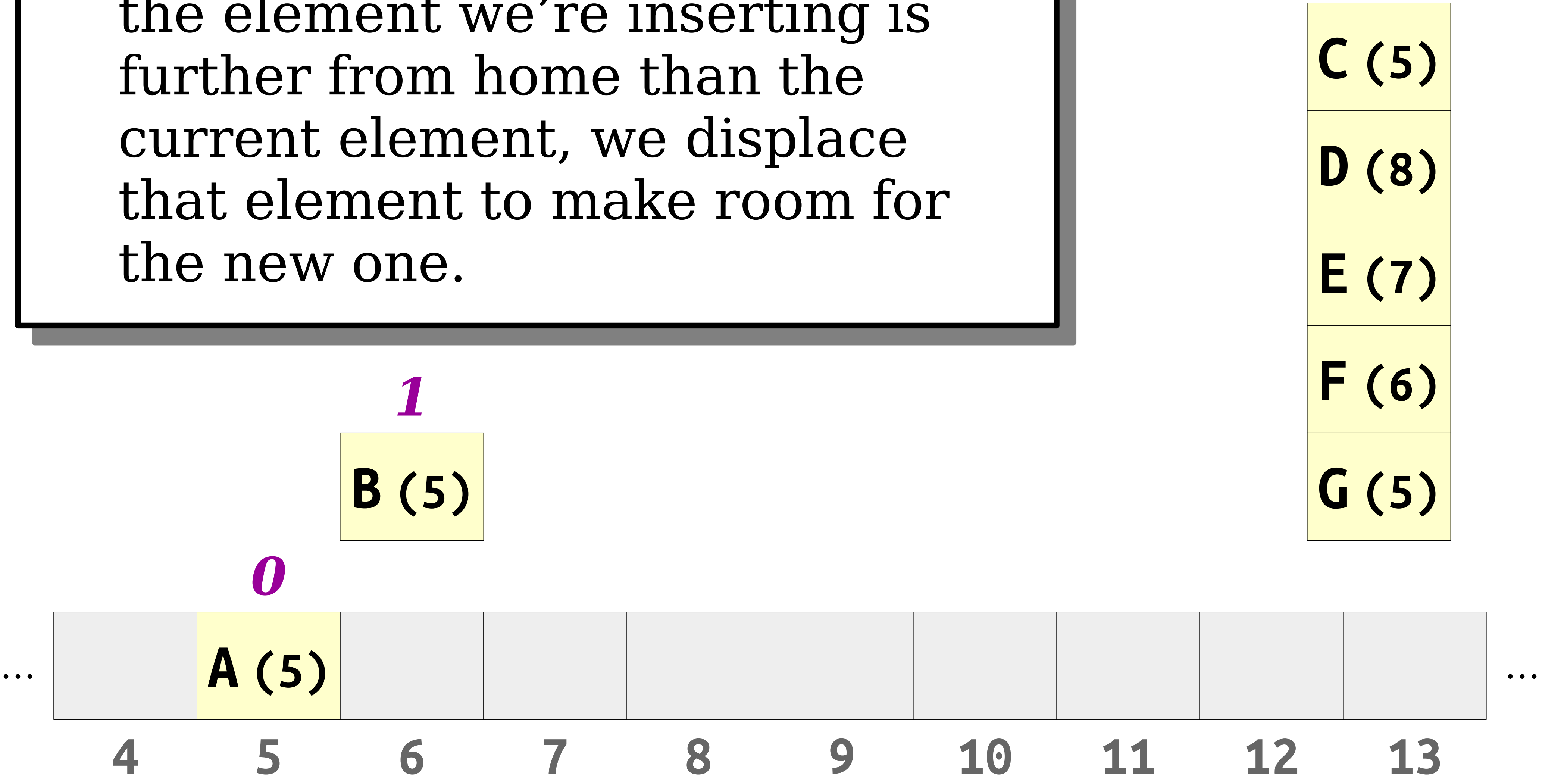




- ***Robin Hood hashing*** is a slight modification to linear probing.
- When we insert an element, if the element we're inserting is further from home than the current element, we displace that element to make room for the new one.



- ***Robin Hood hashing*** is a slight modification to linear probing.
- When we insert an element, if the element we're inserting is further from home than the current element, we displace that element to make room for the new one.



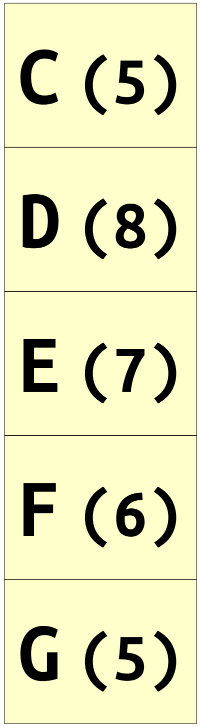
C (5)
D (8)
E (7)
F (6)
G (5)

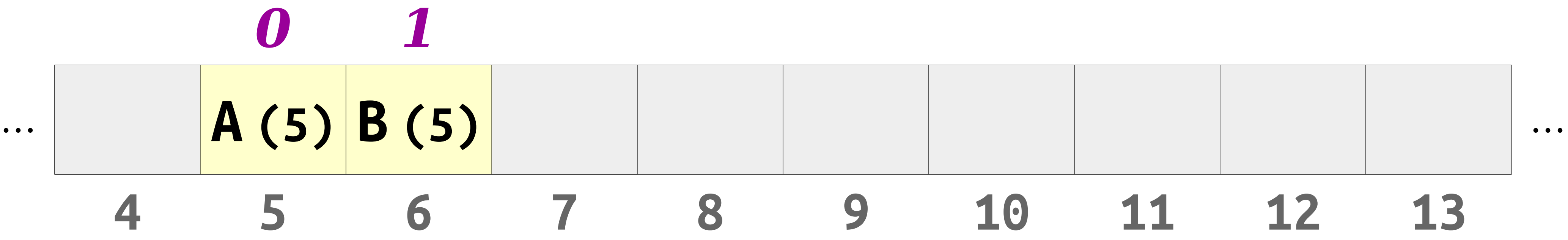
1
B (5)

0
… A (5) …

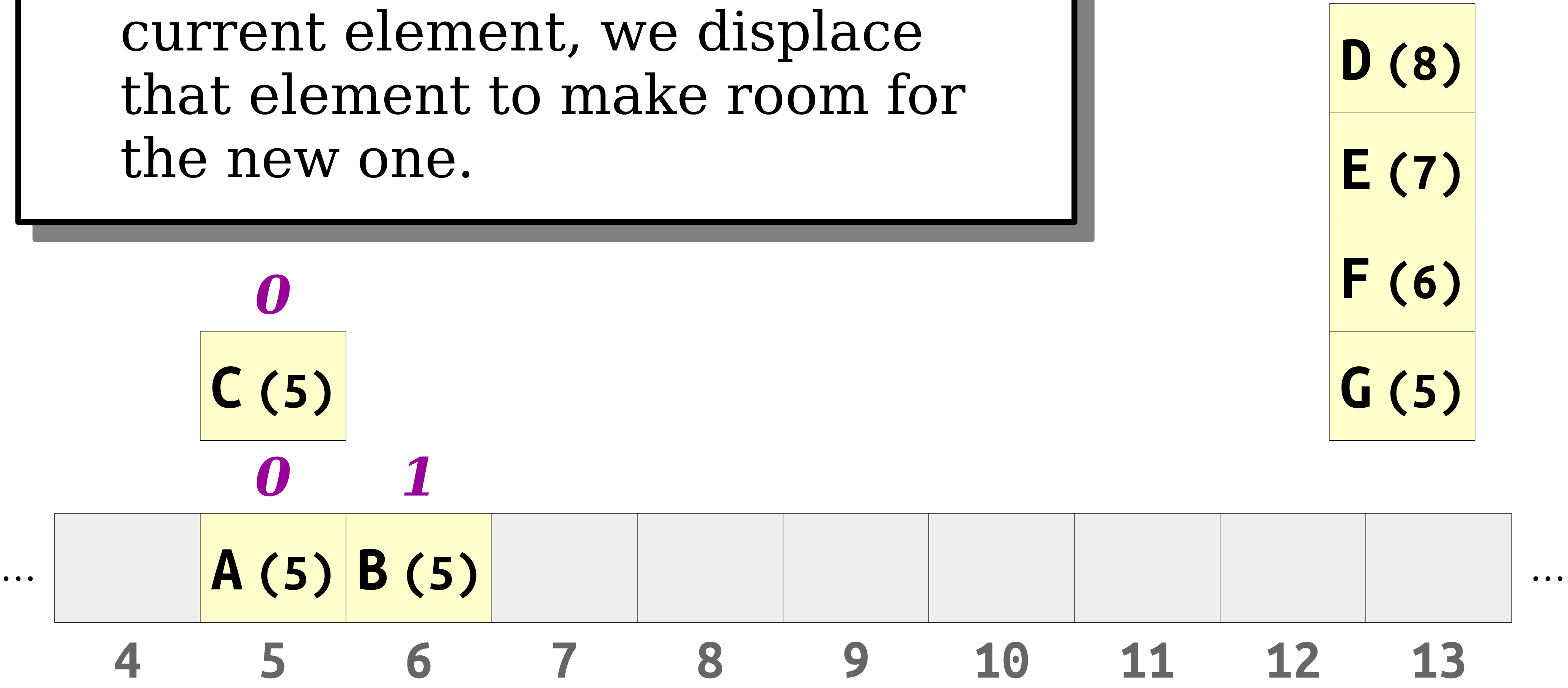
4 5 6 7 8 9 10 11 12 13

- ***Robin Hood hashing*** is a slight modification to linear probing.
- When we insert an element, if the element we're inserting is further from home than the current element, we displace that element to make room for the new one.

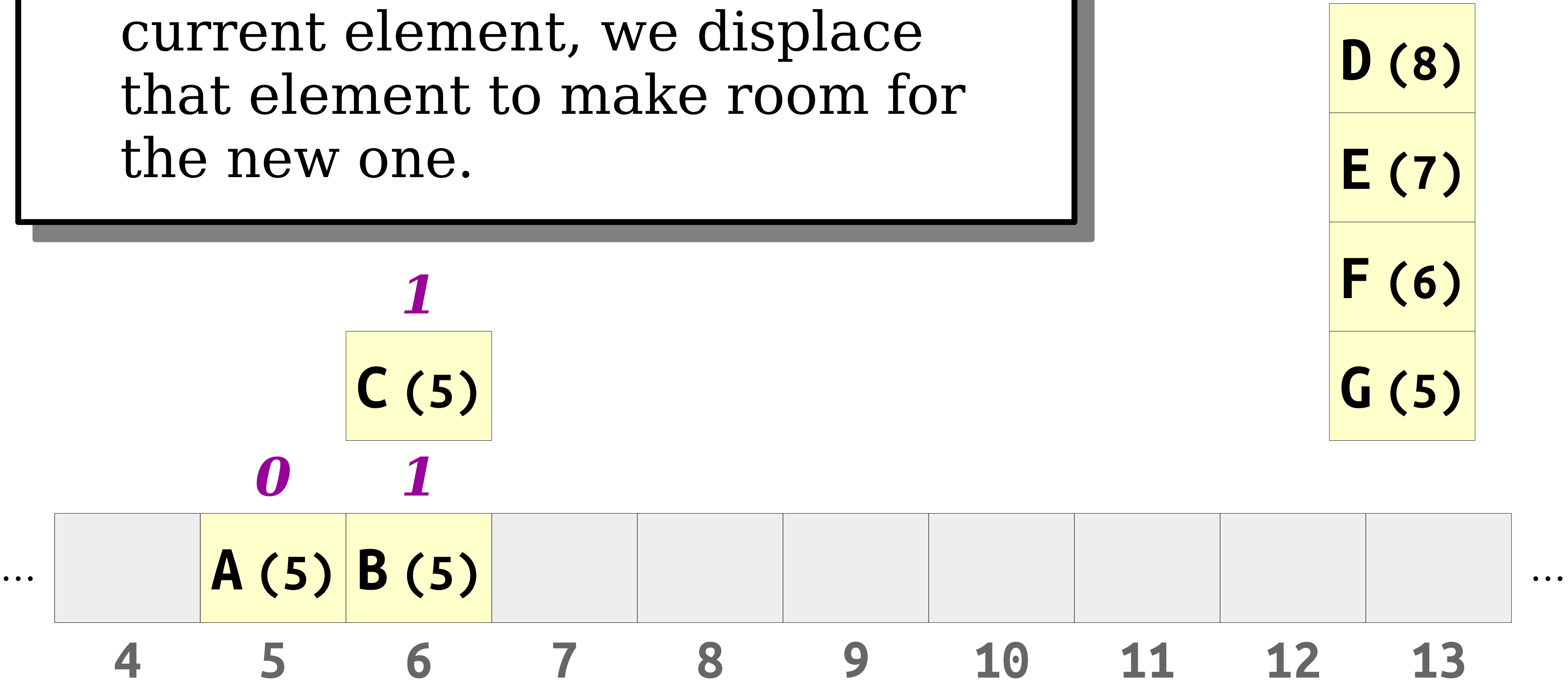




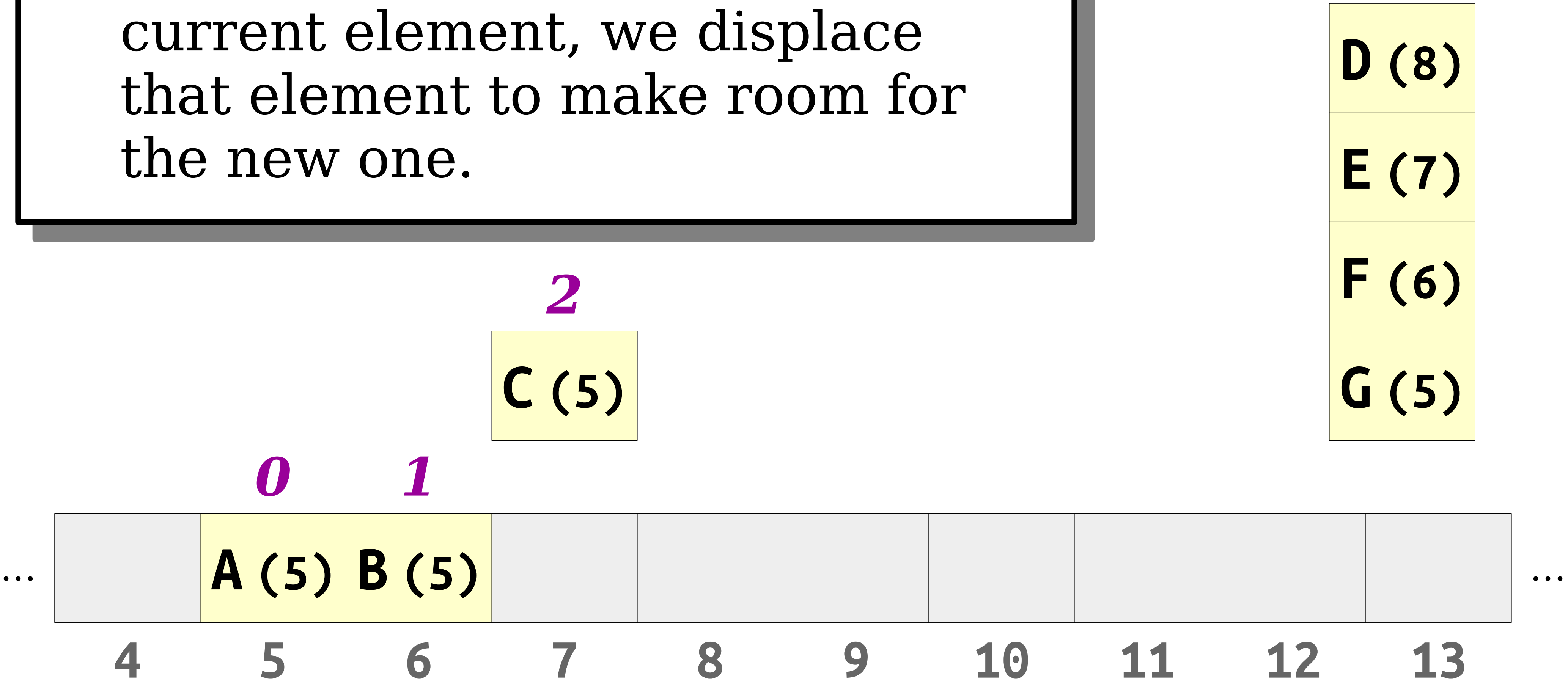
- ***Robin Hood hashing*** is a slight modification to linear probing.
- When we insert an element, if the element we're inserting is further from home than the current element, we displace that element to make room for the new one.



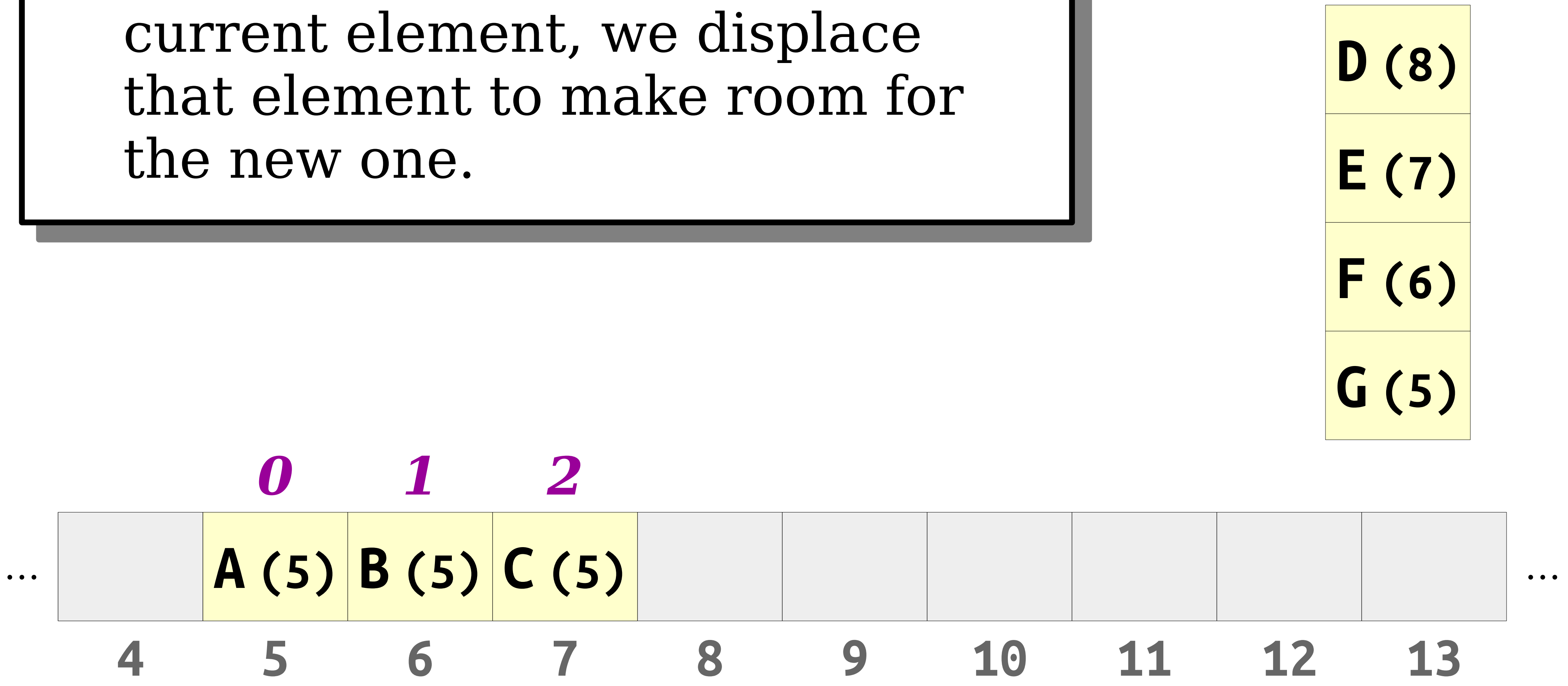
- ***Robin Hood hashing*** is a slight modification to linear probing.
- When we insert an element, if the element we're inserting is further from home than the current element, we displace that element to make room for the new one.



- ***Robin Hood hashing*** is a slight modification to linear probing.
- When we insert an element, if the element we're inserting is further from home than the current element, we displace that element to make room for the new one.



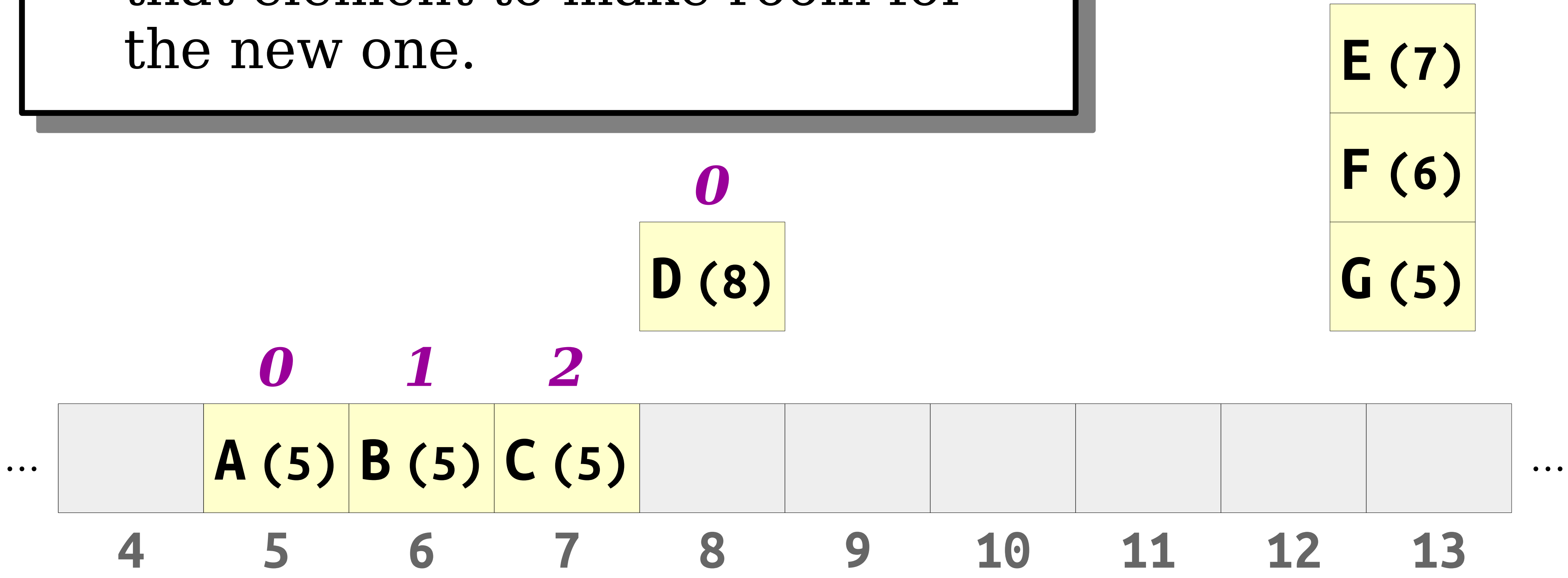
- ***Robin Hood hashing*** is a slight modification to linear probing.
- When we insert an element, if the element we're inserting is further from home than the current element, we displace that element to make room for the new one.



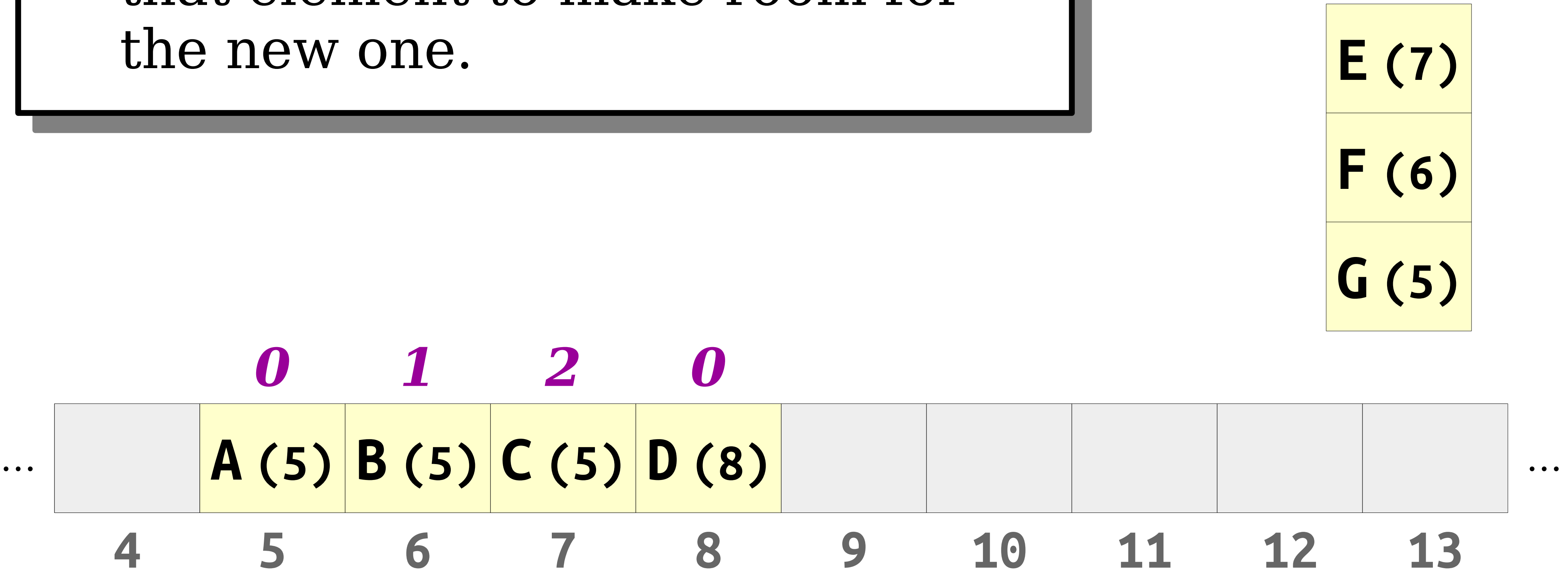
D (8)
E (7)
F (6)
G (5)

		0	*1*	*2*							
…		A (5)	B (5)	C (5)							…
	4	5	6	7	8	9	10	11	12	13	

- ***Robin Hood hashing*** is a slight modification to linear probing.
- When we insert an element, if the element we're inserting is further from home than the current element, we displace that element to make room for the new one.



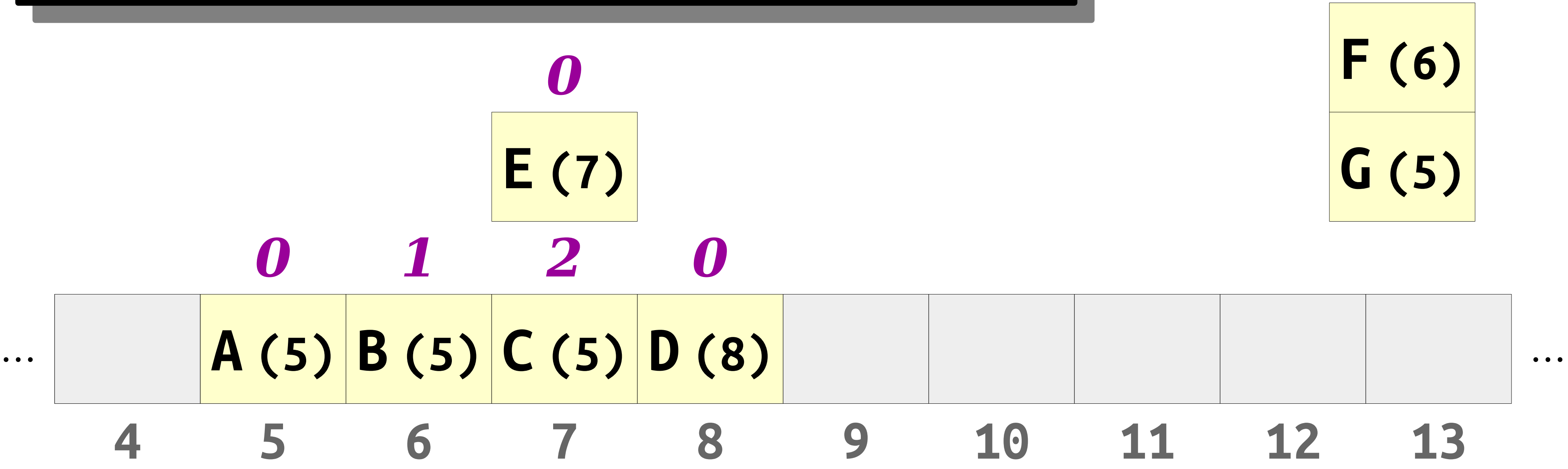
- ***Robin Hood hashing*** is a slight modification to linear probing.
- When we insert an element, if the element we're inserting is further from home than the current element, we displace that element to make room for the new one.



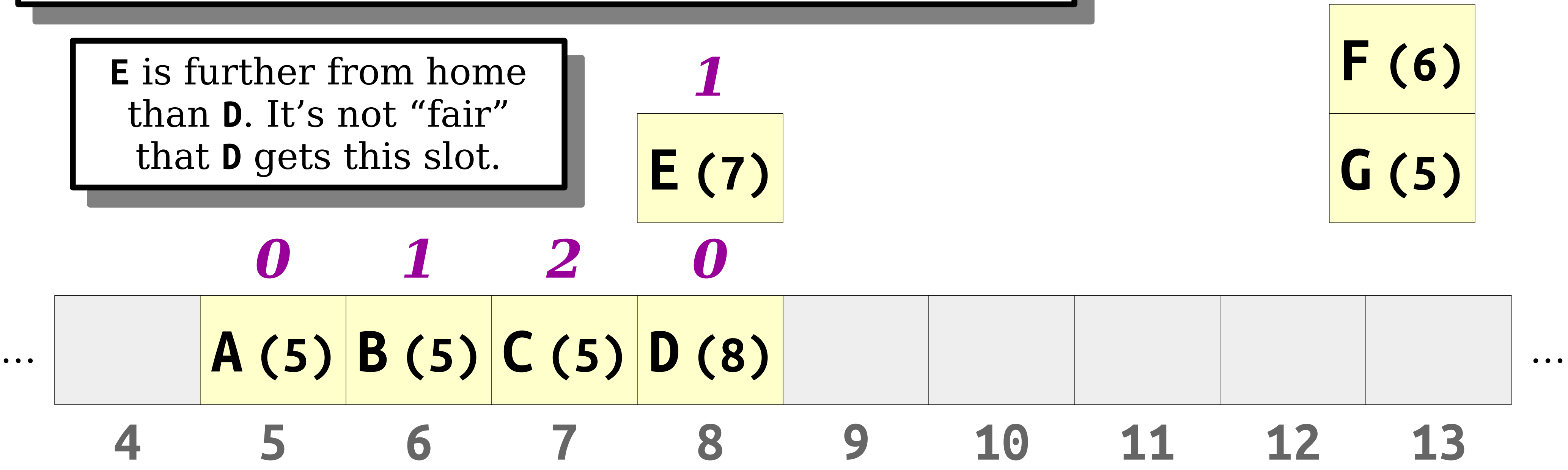
E (7)
F (6)
G (5)

		0	*1*	*2*	*0*						
…		A (5)	B (5)	C (5)	D (8)						…
	4	5	6	7	8	9	10	11	12	13	

- ***Robin Hood hashing*** is a slight modification to linear probing.
- When we insert an element, if the element we're inserting is further from home than the current element, we displace that element to make room for the new one.

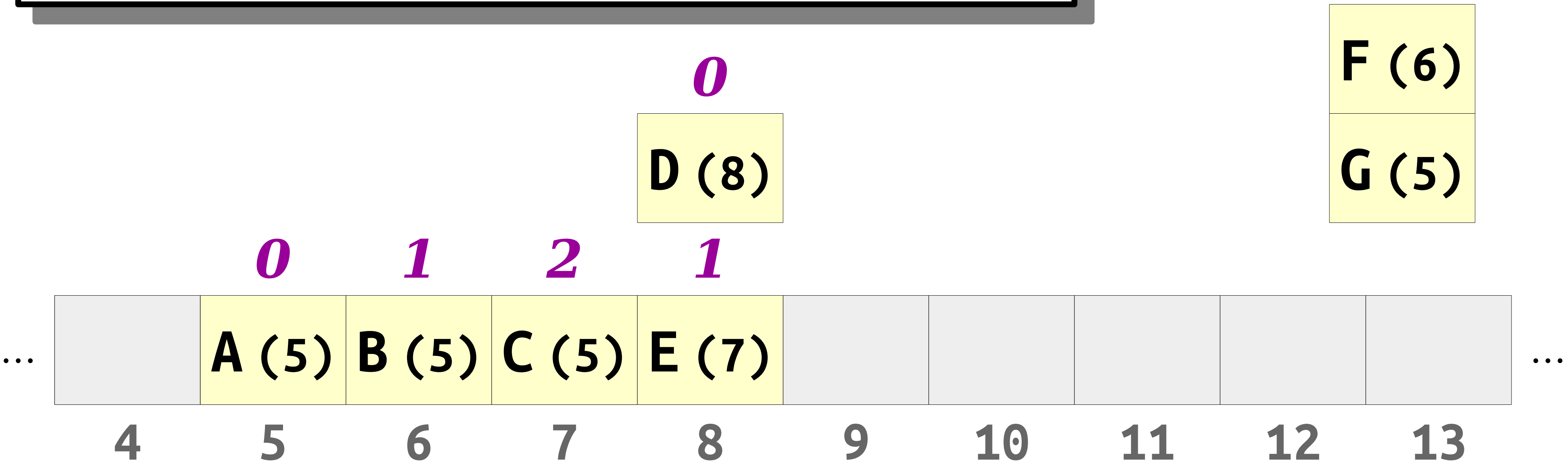


- ***Robin Hood hashing*** is a slight modification to linear probing.
- When we insert an element, if the element we're inserting is further from home than the current element, we displace that element to make room for the new one.

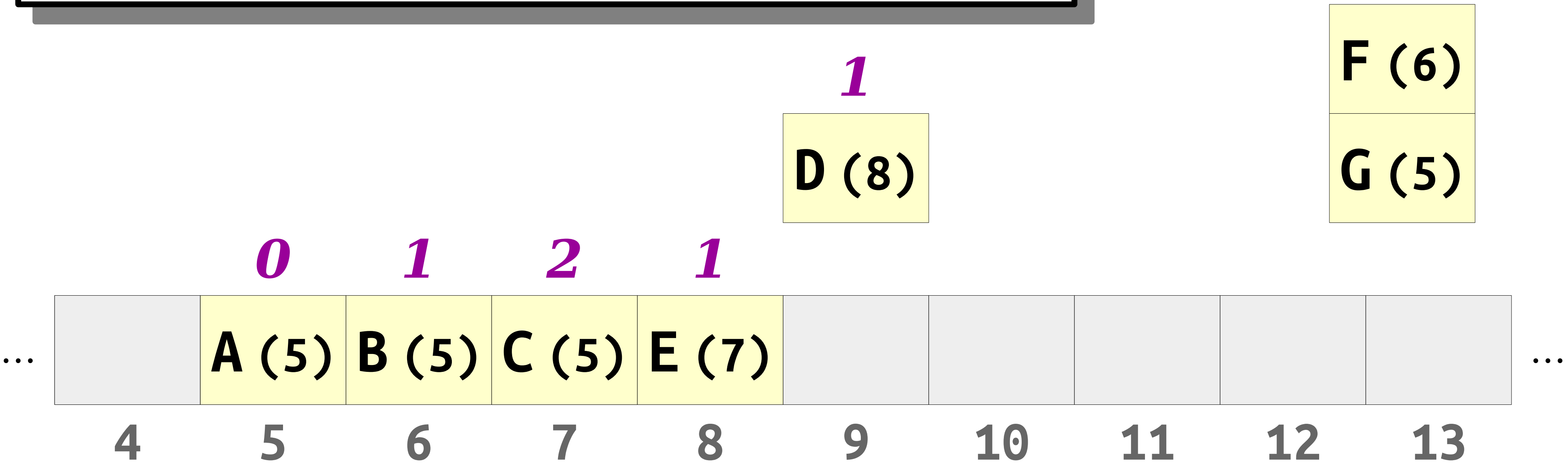


E is further from home than **D**. It's not "fair" that **D** gets this slot.

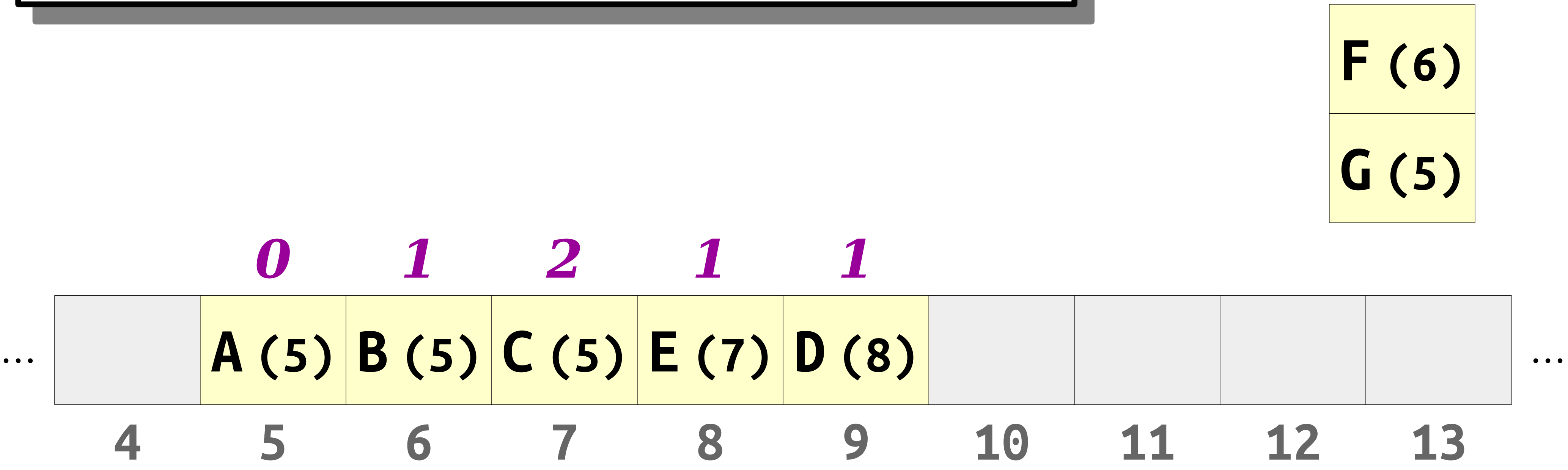
- ***Robin Hood hashing*** is a slight modification to linear probing.
- When we insert an element, if the element we're inserting is further from home than the current element, we displace that element to make room for the new one.



- ***Robin Hood hashing*** is a slight modification to linear probing.
- When we insert an element, if the element we're inserting is further from home than the current element, we displace that element to make room for the new one.



- ***Robin Hood hashing*** is a slight modification to linear probing.
- When we insert an element, if the element we're inserting is further from home than the current element, we displace that element to make room for the new one.

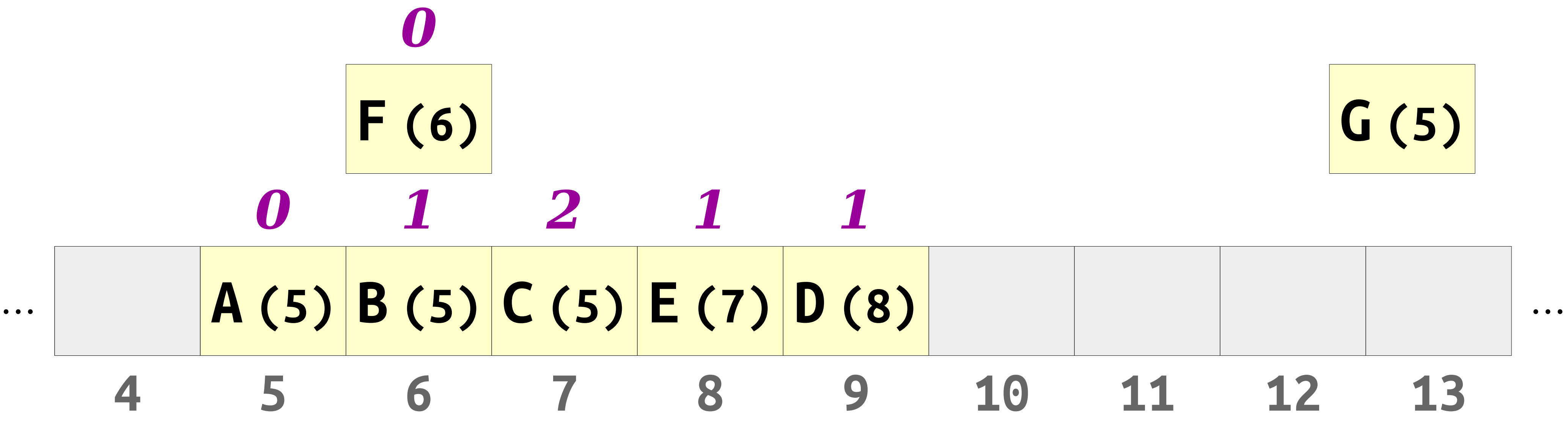


F (6)

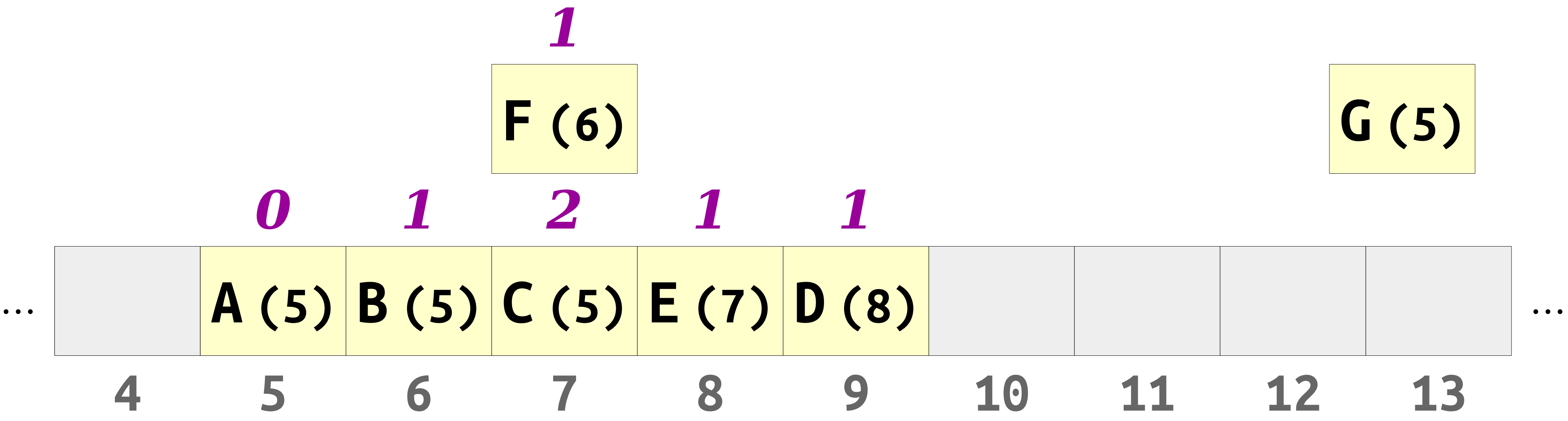
G (5)

…											…
		0	*1*	*2*	*1*	*1*					
…		A (5)	B (5)	C (5)	E (7)	D (8)					…
	4	5	6	7	8	9	10	11	12	13	

- ***Robin Hood hashing*** is a slight modification to linear probing.
- When we insert an element, if the element we're inserting is further from home than the current element, we displace that element to make room for the new one.

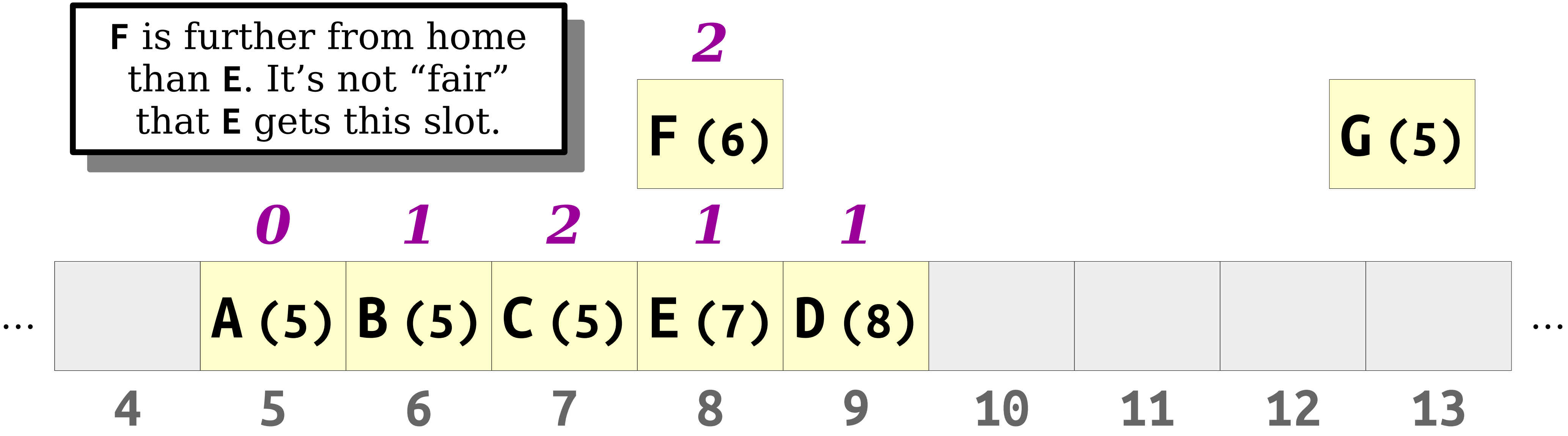


- ***Robin Hood hashing*** is a slight modification to linear probing.
- When we insert an element, if the element we're inserting is further from home than the current element, we displace that element to make room for the new one.



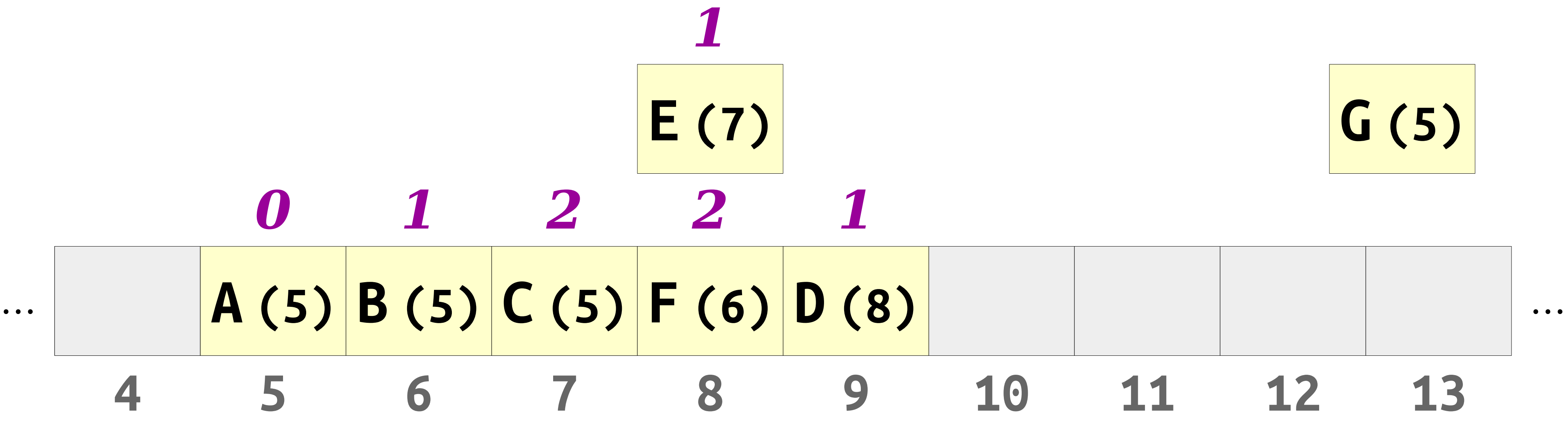
- ***Robin Hood hashing*** is a slight modification to linear probing.
- When we insert an element, if the element we're inserting is further from home than the current element, we displace that element to make room for the new one.

F is further from home than **E**. It's not "fair" that **E** gets this slot.

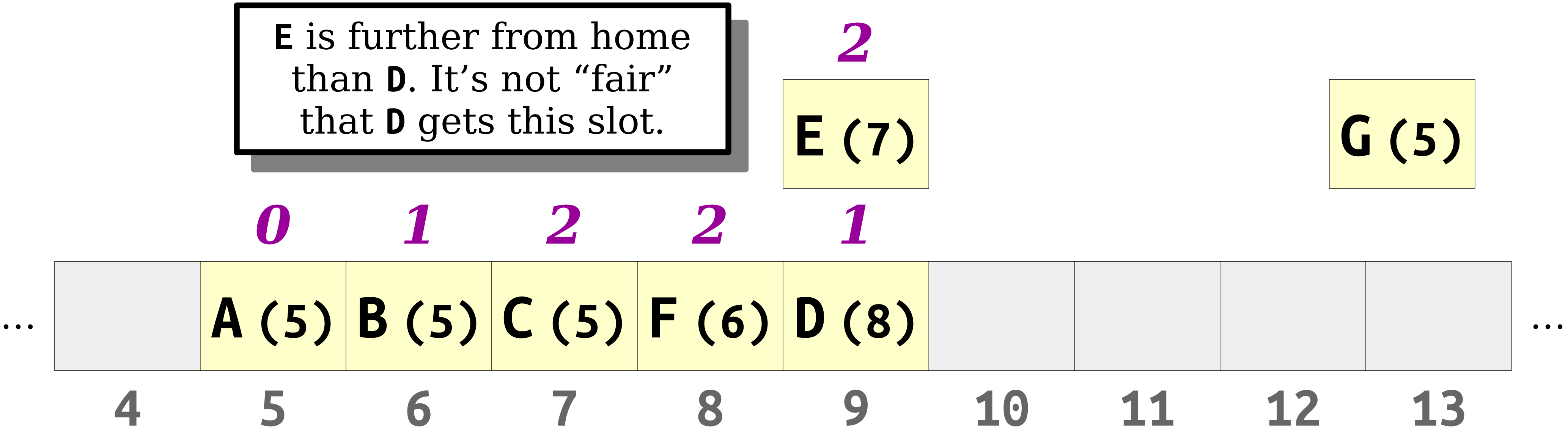


…		*0*	*1*	*2*	*1*	*1*					…
…		A (5)	B (5)	C (5)	E (7)	D (8)					…
	4	5	6	7	8	9	10	11	12	13	

- ***Robin Hood hashing*** is a slight modification to linear probing.
- When we insert an element, if the element we're inserting is further from home than the current element, we displace that element to make room for the new one.



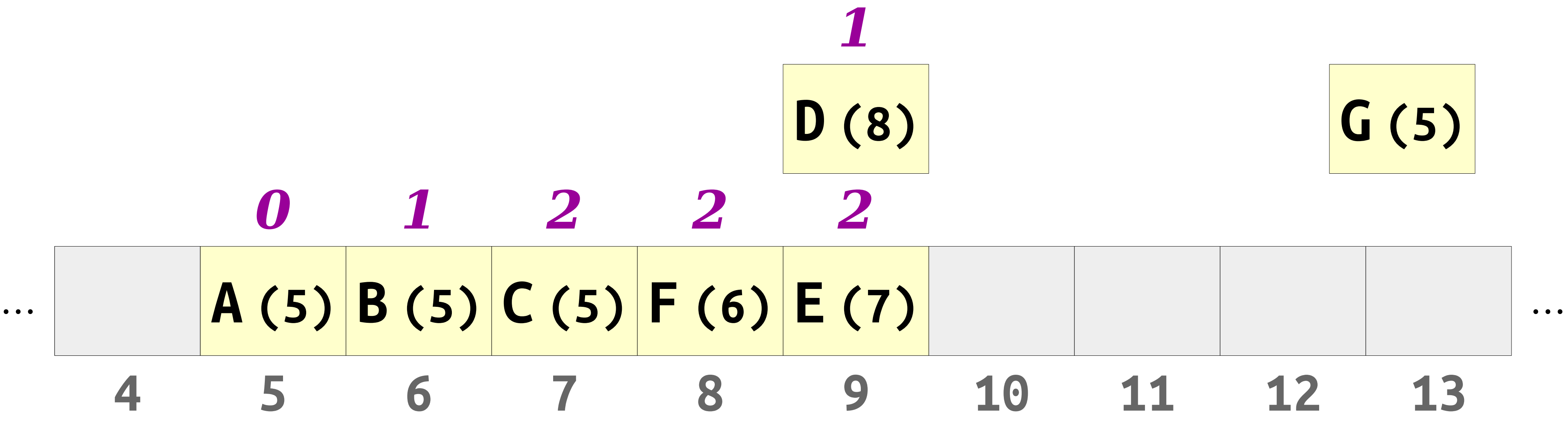
- ***Robin Hood hashing*** is a slight modification to linear probing.
- When we insert an element, if the element we're inserting is further from home than the current element, we displace that element to make room for the new one.



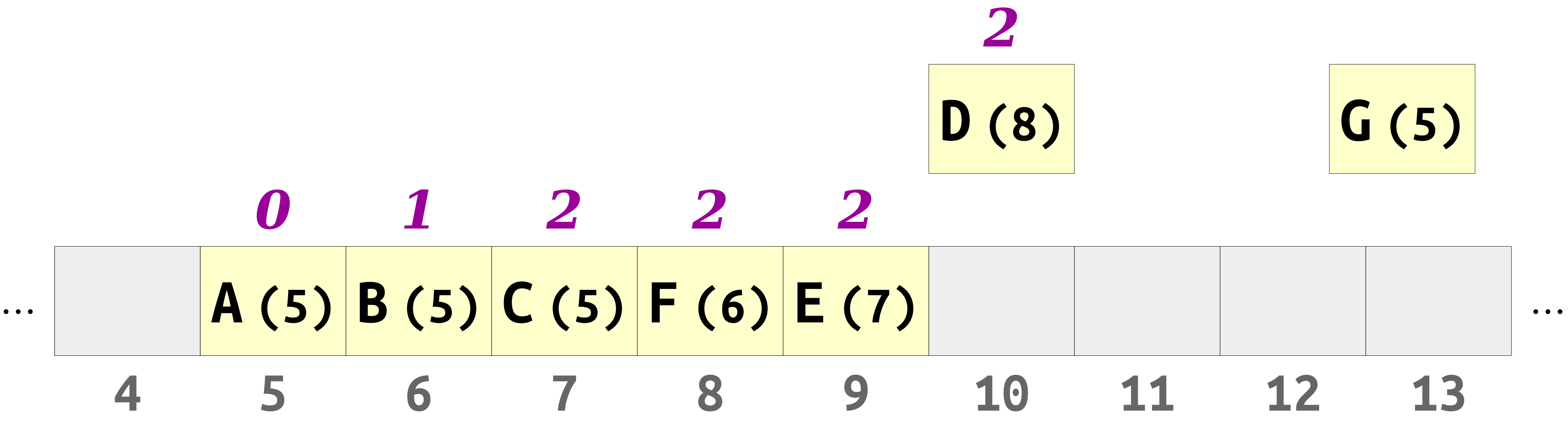
E is further from home than **D**. It's not "fair" that **D** gets this slot.

						2					
						E (7)				G (5)	
		0	*1*	*2*	*2*	*1*					
…		A (5)	B (5)	C (5)	F (6)	D (8)					…
	4	5	6	7	8	9	10	11	12	13	

- ***Robin Hood hashing*** is a slight modification to linear probing.
- When we insert an element, if the element we're inserting is further from home than the current element, we displace that element to make room for the new one.



- ***Robin Hood hashing*** is a slight modification to linear probing.
- When we insert an element, if the element we're inserting is further from home than the current element, we displace that element to make room for the new one.



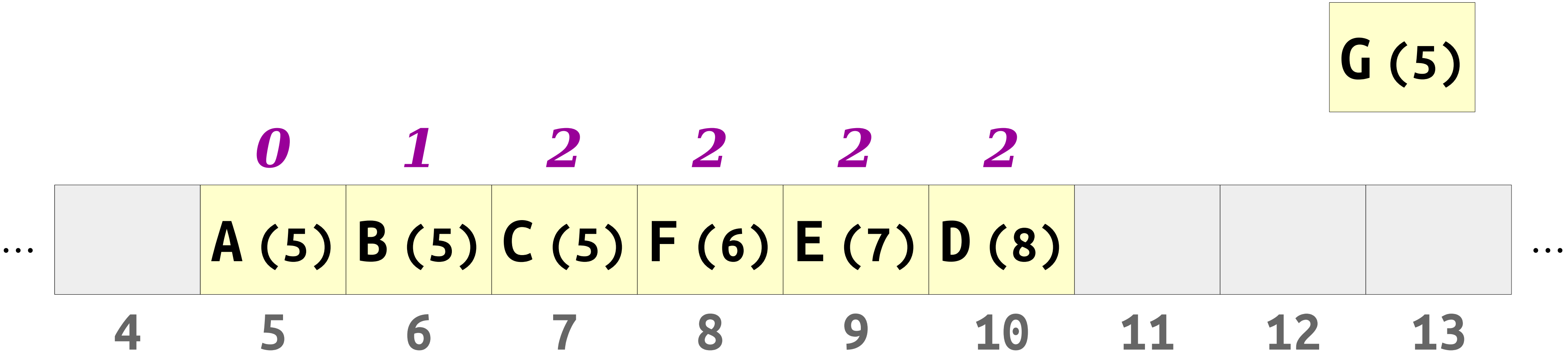
2

D (8)

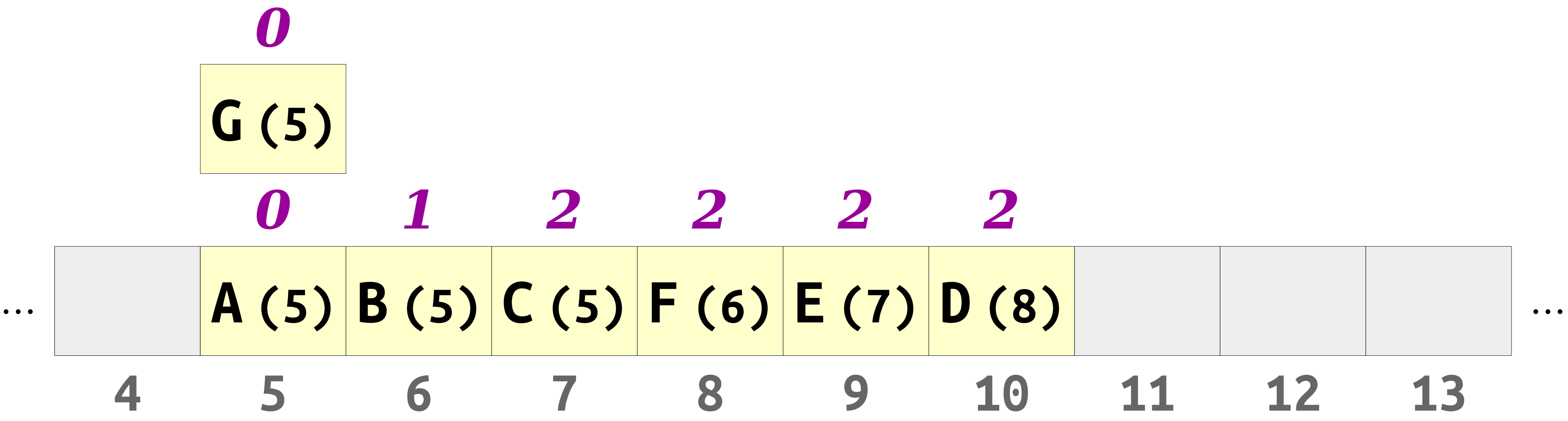
G (5)

		0	1	2	2	2					
…		A (5)	B (5)	C (5)	F (6)	E (7)					…
	4	5	6	7	8	9	10	11	12	13	

- ***Robin Hood hashing*** is a slight modification to linear probing.
- When we insert an element, if the element we're inserting is further from home than the current element, we displace that element to make room for the new one.

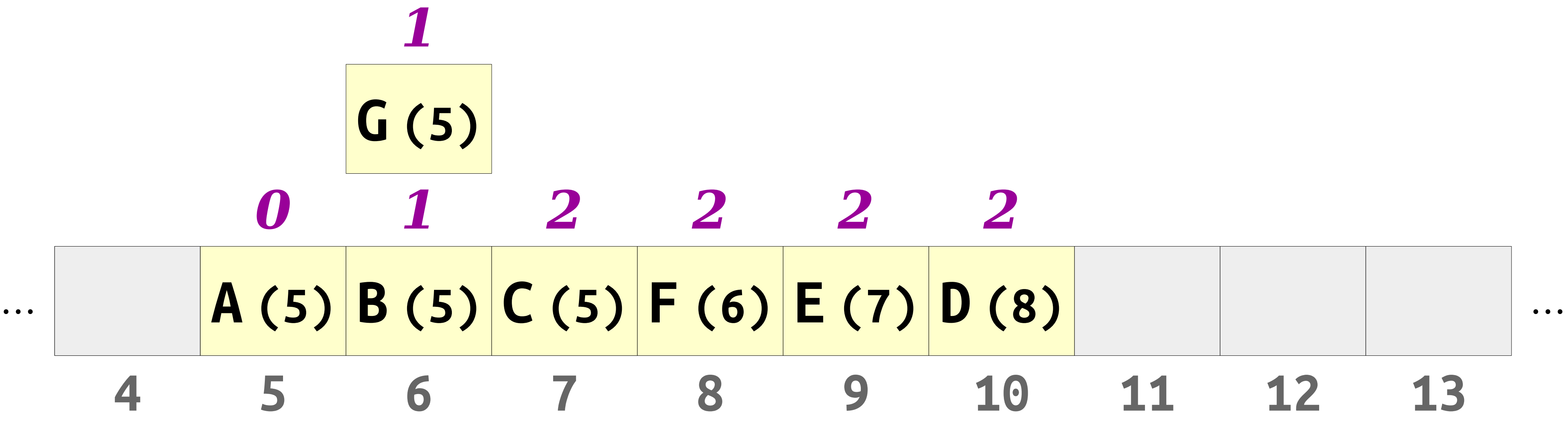


- ***Robin Hood hashing*** is a slight modification to linear probing.
- When we insert an element, if the element we're inserting is further from home than the current element, we displace that element to make room for the new one.

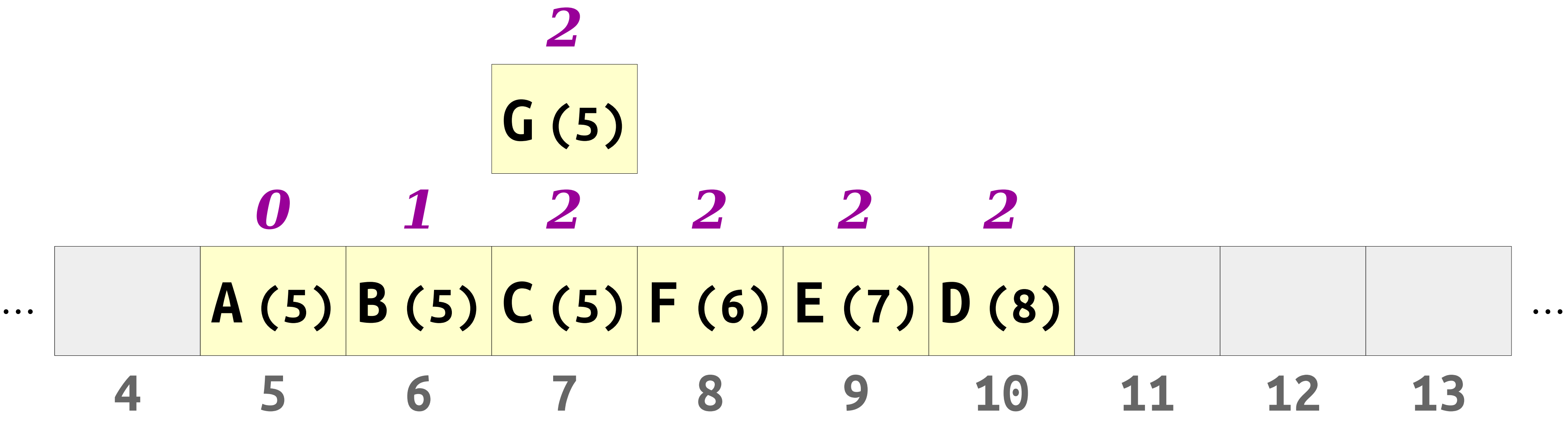


		0									
		G (5)									
		0	1	2	2	2	2				
…		A (5)	B (5)	C (5)	F (6)	E (7)	D (8)				…
	4	5	6	7	8	9	10	11	12	13	

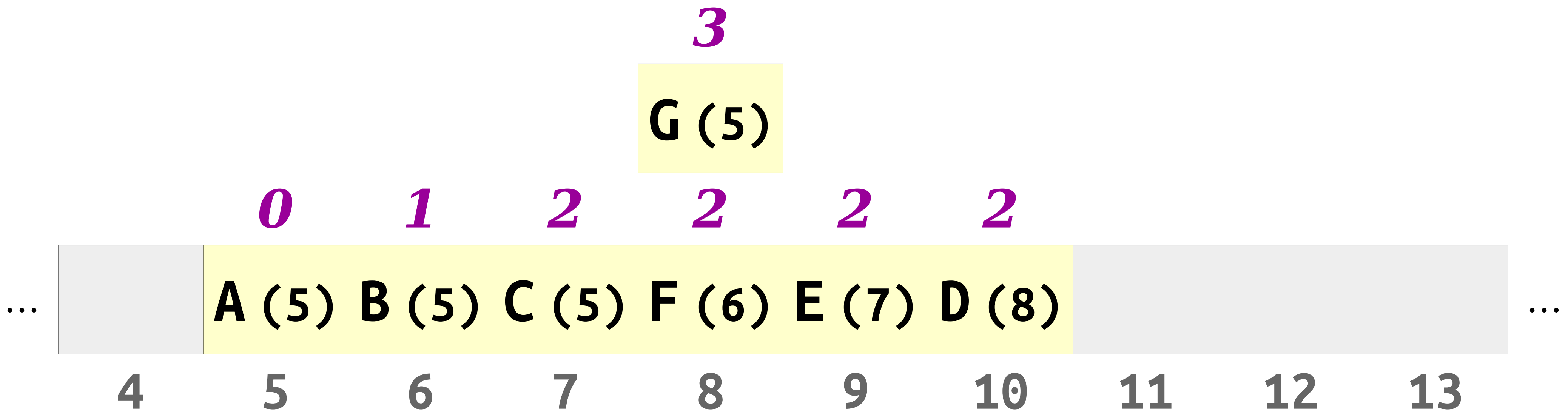
- ***Robin Hood hashing*** is a slight modification to linear probing.
- When we insert an element, if the element we're inserting is further from home than the current element, we displace that element to make room for the new one.



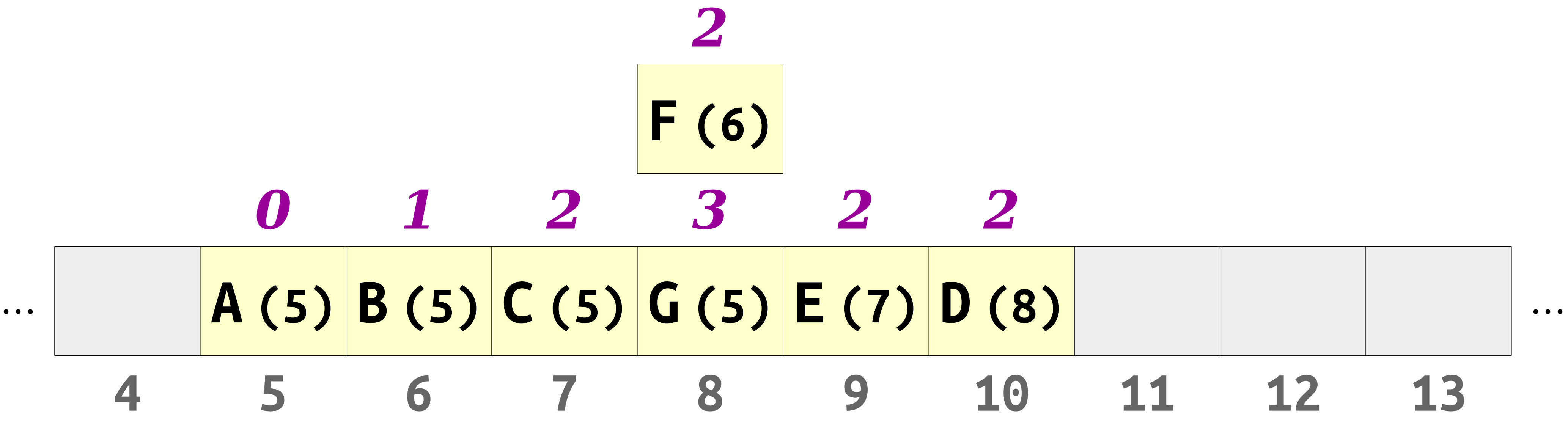
- ***Robin Hood hashing*** is a slight modification to linear probing.
- When we insert an element, if the element we're inserting is further from home than the current element, we displace that element to make room for the new one.



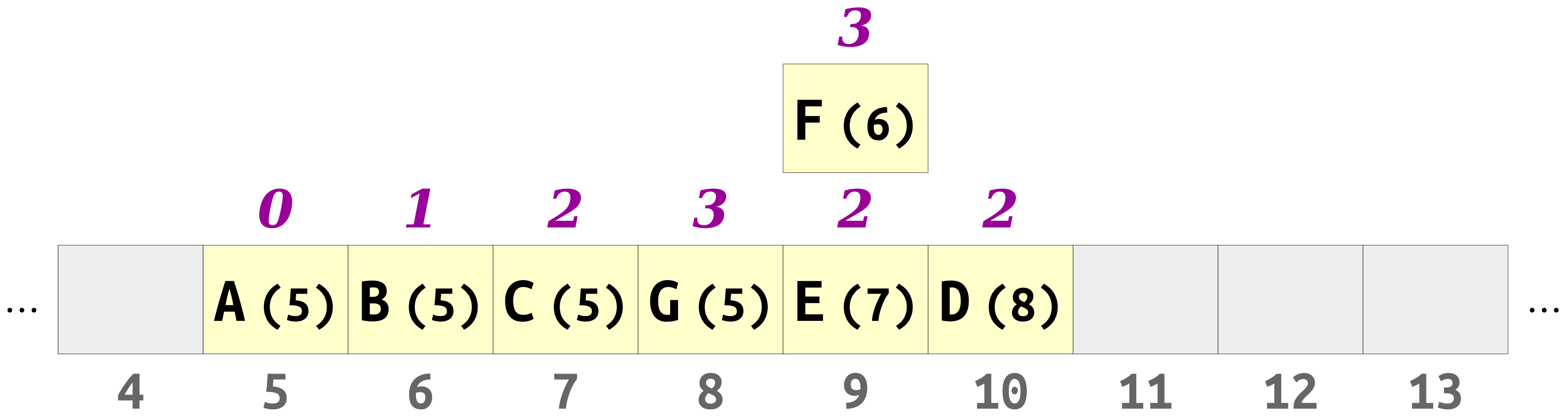
- ***Robin Hood hashing*** is a slight modification to linear probing.
- When we insert an element, if the element we're inserting is further from home than the current element, we displace that element to make room for the new one.



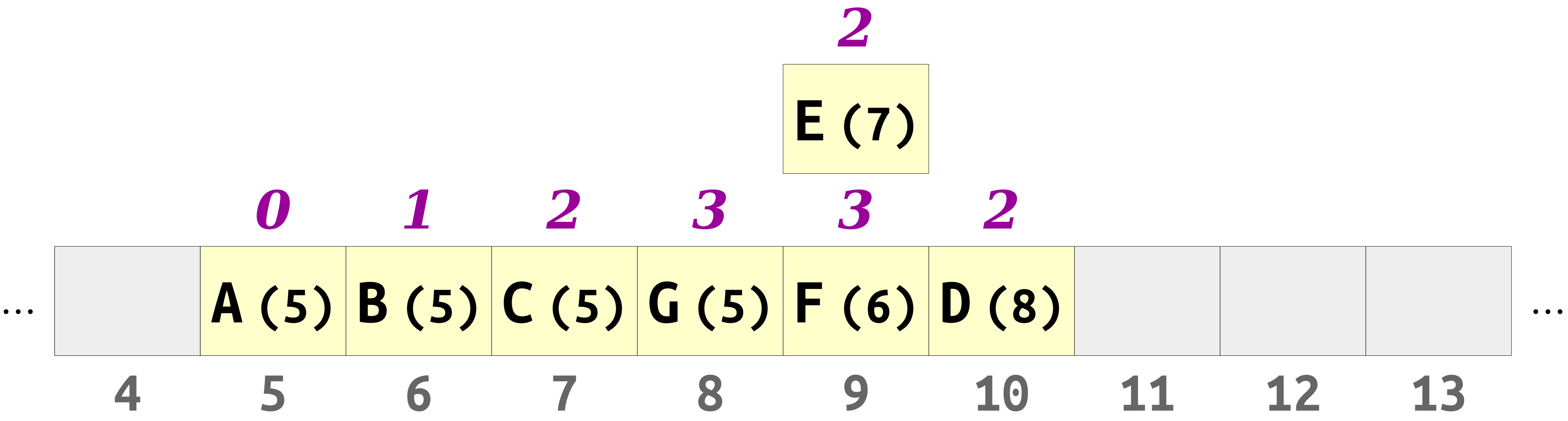
- ***Robin Hood hashing*** is a slight modification to linear probing.
- When we insert an element, if the element we're inserting is further from home than the current element, we displace that element to make room for the new one.



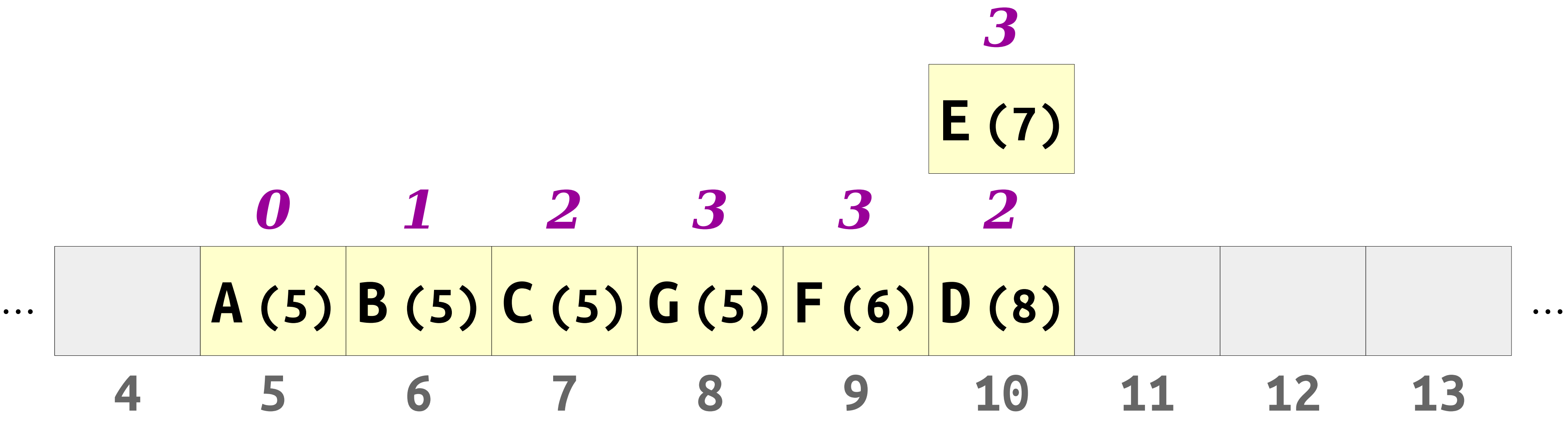
- ***Robin Hood hashing*** is a slight modification to linear probing.
- When we insert an element, if the element we're inserting is further from home than the current element, we displace that element to make room for the new one.



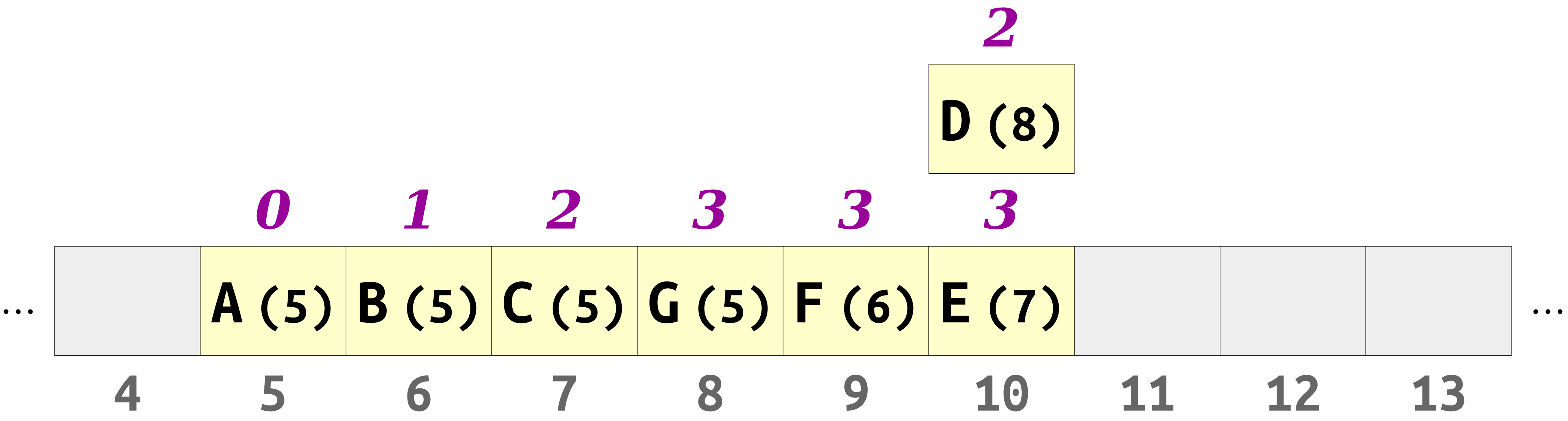
- ***Robin Hood hashing*** is a slight modification to linear probing.
- When we insert an element, if the element we're inserting is further from home than the current element, we displace that element to make room for the new one.



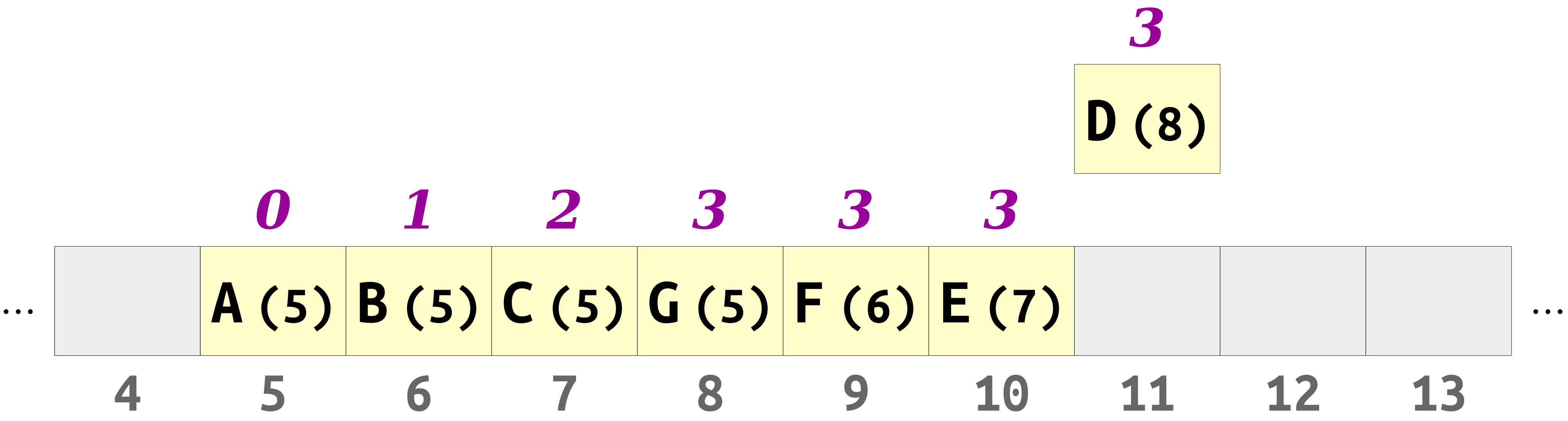
- ***Robin Hood hashing*** is a slight modification to linear probing.
- When we insert an element, if the element we're inserting is further from home than the current element, we displace that element to make room for the new one.



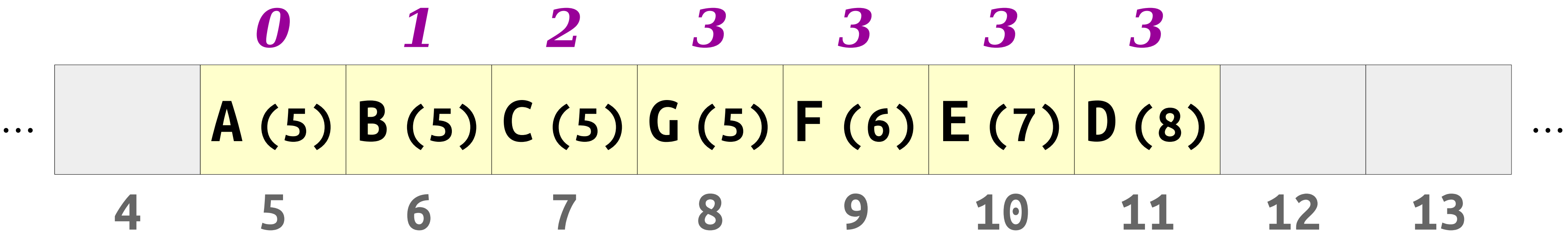
- ***Robin Hood hashing*** is a slight modification to linear probing.
- When we insert an element, if the element we're inserting is further from home than the current element, we displace that element to make room for the new one.



- ***Robin Hood hashing*** is a slight modification to linear probing.
- When we insert an element, if the element we're inserting is further from home than the current element, we displace that element to make room for the new one.

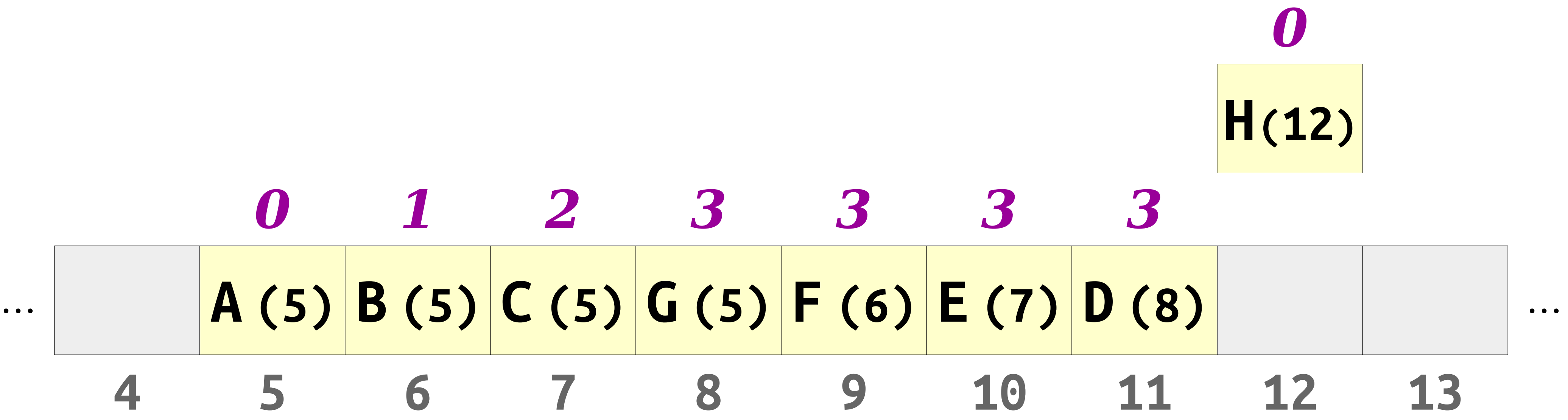


- ***Robin Hood hashing*** is a slight modification to linear probing.
- When we insert an element, if the element we're inserting is further from home than the current element, we displace that element to make room for the new one.

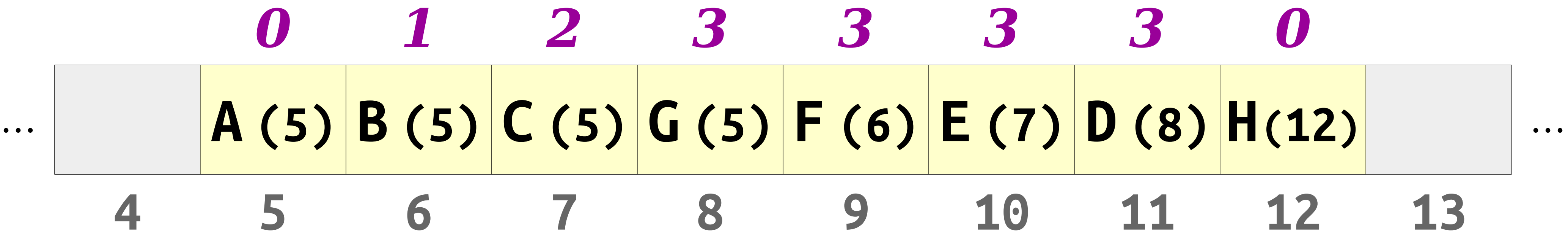


	4	5	6	7	8	9	10	11	12	13	
		0	*1*	*2*	*3*	*3*	*3*	*3*			
…		A (5)	B (5)	C (5)	G (5)	F (6)	E (7)	D (8)			…

- ***Robin Hood hashing*** is a slight modification to linear probing.
- When we insert an element, if the element we're inserting is further from home than the current element, we displace that element to make room for the new one.

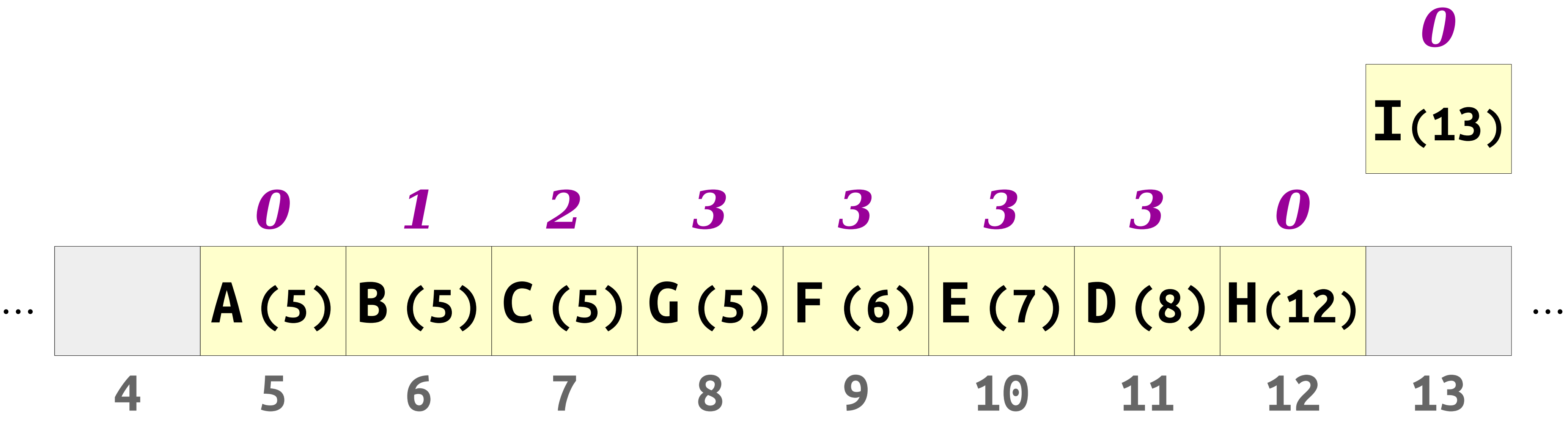


- ***Robin Hood hashing*** is a slight modification to linear probing.
- When we insert an element, if the element we're inserting is further from home than the current element, we displace that element to make room for the new one.



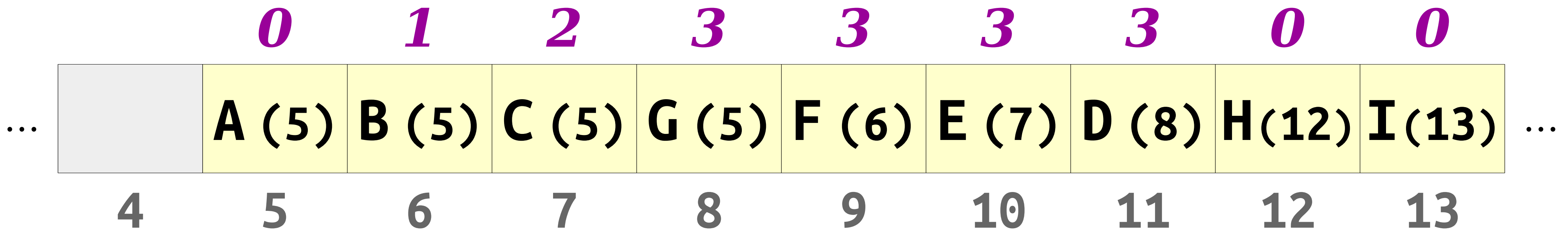
		0	*1*	*2*	*3*	*3*	*3*	*3*	*0*		
…		A (5)	B (5)	C (5)	G (5)	F (6)	E (7)	D (8)	H(12)		…
	4	5	6	7	8	9	10	11	12	13	

- ***Robin Hood hashing*** is a slight modification to linear probing.
- When we insert an element, if the element we're inserting is further from home than the current element, we displace that element to make room for the new one.



		0	1	2	3	3	3	3	0		
…		A (5)	B (5)	C (5)	G (5)	F (6)	E (7)	D (8)	H(12)		…
	4	5	6	7	8	9	10	11	12	13	

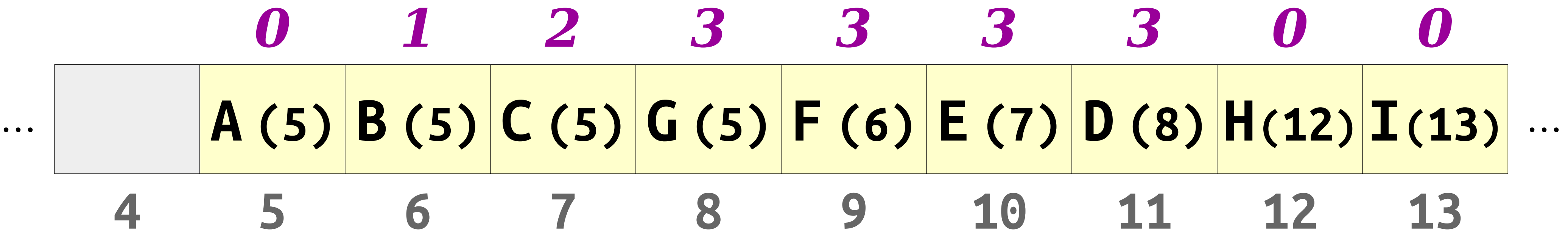
- ***Robin Hood hashing*** is a slight modification to linear probing.
- When we insert an element, if the element we're inserting is further from home than the current element, we displace that element to make room for the new one.



		0	1	2	3	3	3	3	0	0	
…		A (5)	B (5)	C (5)	G (5)	F (6)	E (7)	D (8)	H(12)	I(13)	…
	4	5	6	7	8	9	10	11	12	13	

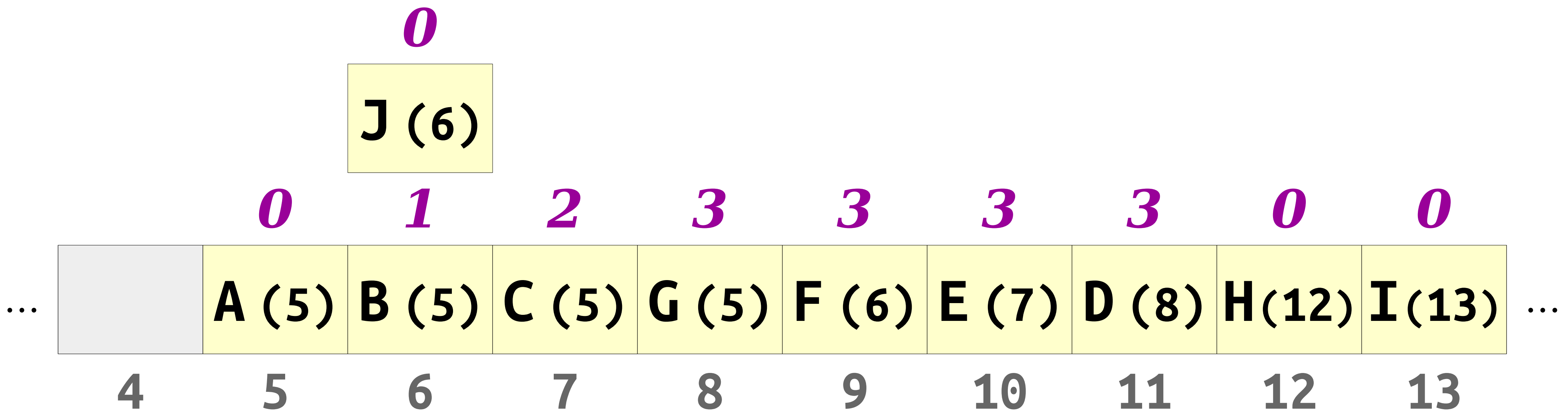
- ***Neat trick:*** We can make unsuccessful lookups in a Robin Hood hashing table faster than in a linear probing table.
- ***Idea:*** Compare the distances of the item to insert and the item being looked up.

J (6)



	0	*1*	*2*	*3*	*3*	*3*	*3*	*0*	*0*	
…		A (5)	B (5)	C (5)	G (5)	F (6)	E (7)	D (8)	H(12)	I(13)	…
	4	5	6	7	8	9	10	11	12	13	

- ***Neat trick:*** We can make unsuccessful lookups in a Robin Hood hashing table faster than in a linear probing table.
- ***Idea:*** Compare the distances of the item to insert and the item being looked up.

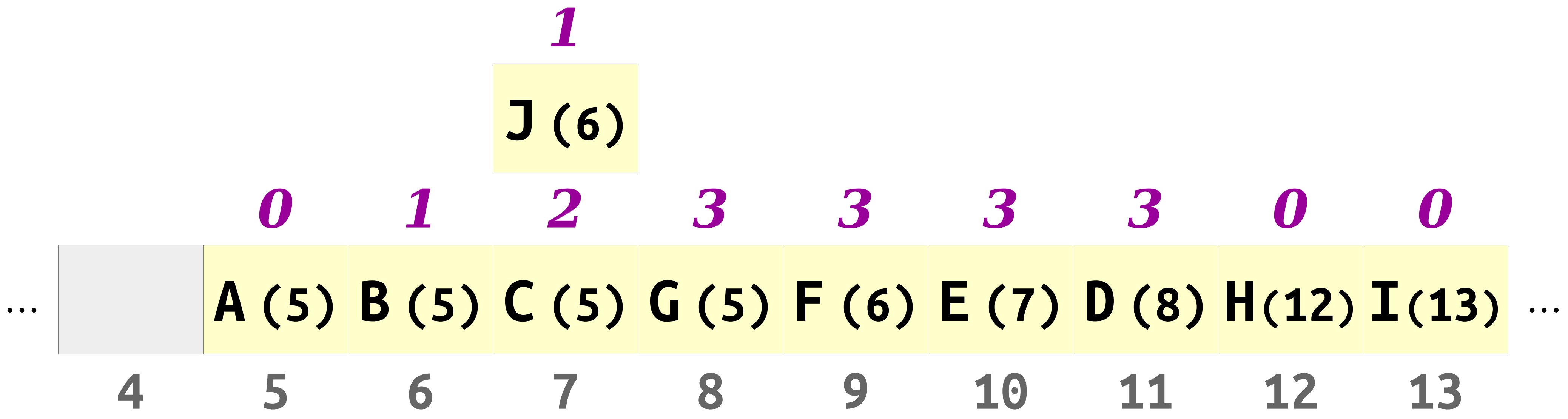


0

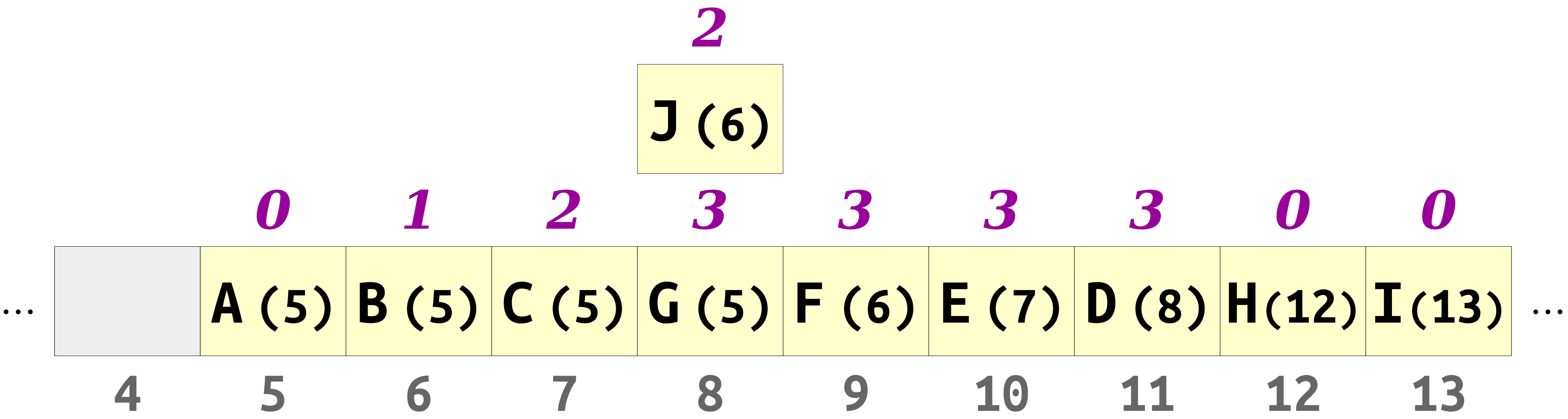
J (6)

		0	*1*	*2*	*3*	*3*	*3*	*3*	*0*	*0*	
…		A (5)	B (5)	C (5)	G (5)	F (6)	E (7)	D (8)	H (12)	I (13)	…
	4	5	6	7	8	9	10	11	12	13	

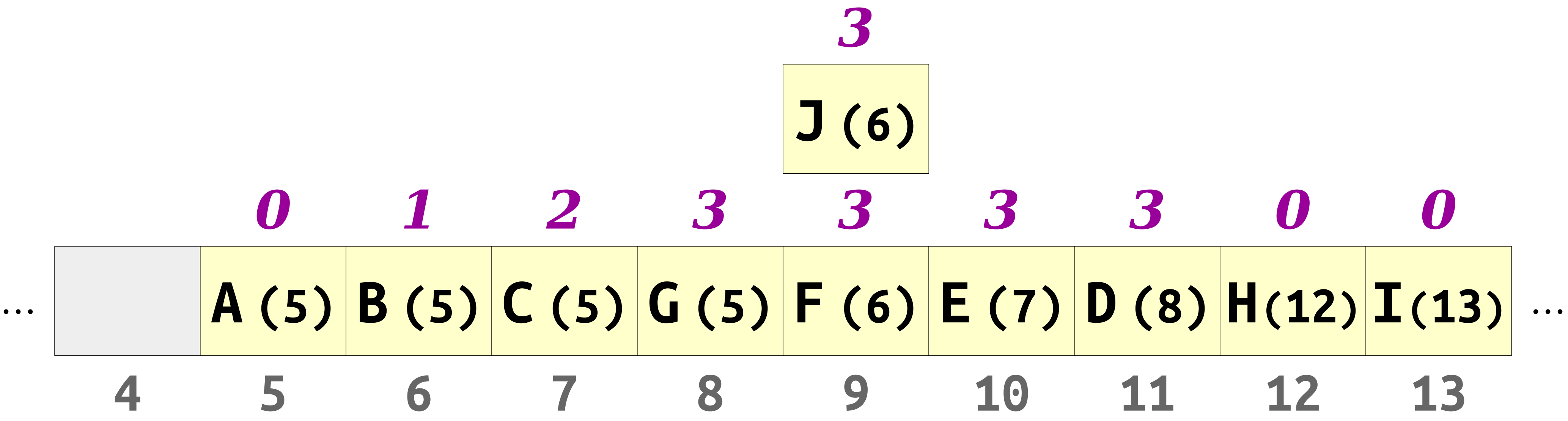
- ***Neat trick:*** We can make unsuccessful lookups in a Robin Hood hashing table faster than in a linear probing table.
- ***Idea:*** Compare the distances of the item to insert and the item being looked up.



- ***Neat trick:*** We can make unsuccessful lookups in a Robin Hood hashing table faster than in a linear probing table.
- ***Idea:*** Compare the distances of the item to insert and the item being looked up.

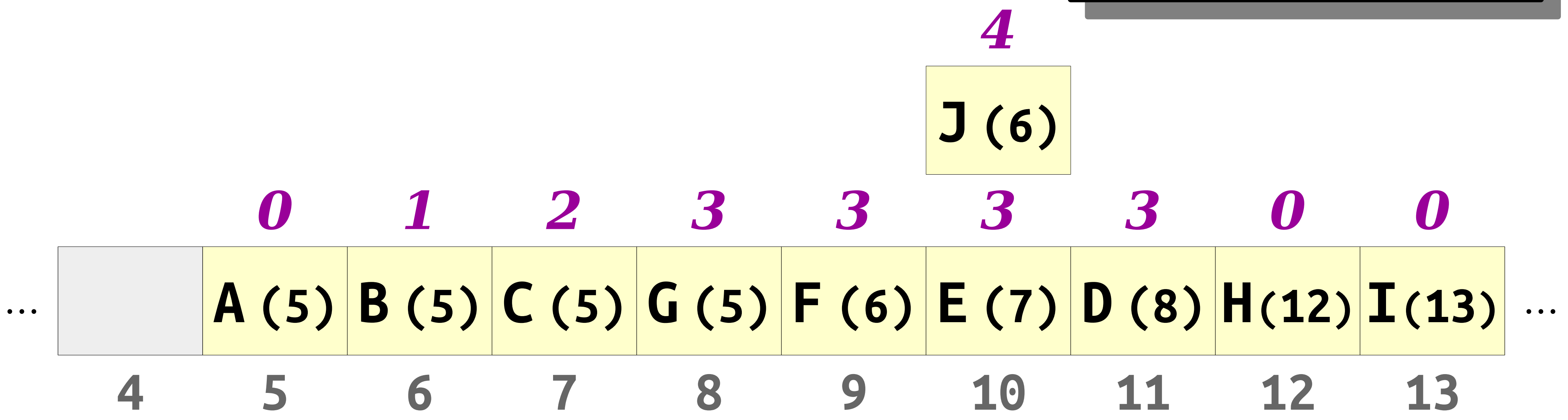


- ***Neat trick:*** We can make unsuccessful lookups in a Robin Hood hashing table faster than in a linear probing table.
- ***Idea:*** Compare the distances of the item to insert and the item being looked up.



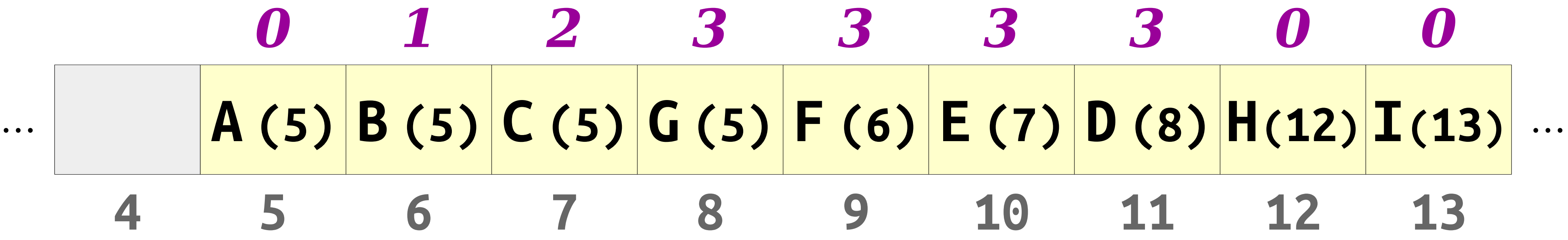
- ***Neat trick:*** We can make unsuccessful lookups in a Robin Hood hashing table faster than in a linear probing table.
- ***Idea:*** Compare the distances of the item to insert and the item being looked up.

If **J** were in this table, it would have displaced the **E**. So **J** can't be in the table!



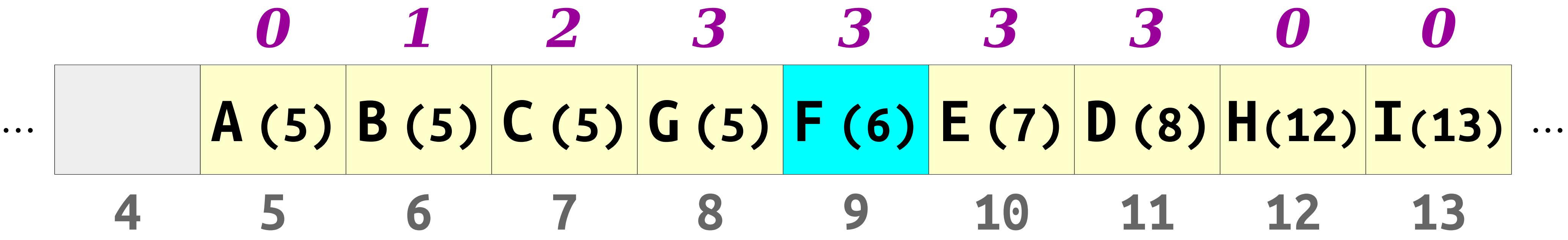
…	4	5	6	7	8	9	10	11	12	13	…
		0	1	2	3	3	3	3	0	0	
		A (5)	B (5)	C (5)	G (5)	F (6)	E (7)	D (8)	H (12)	I (13)	

- ***Neat trick:*** Robin Hood hashing doesn't need tombstones.
- We can use a technique called ***backward-shift deletion*** instead.



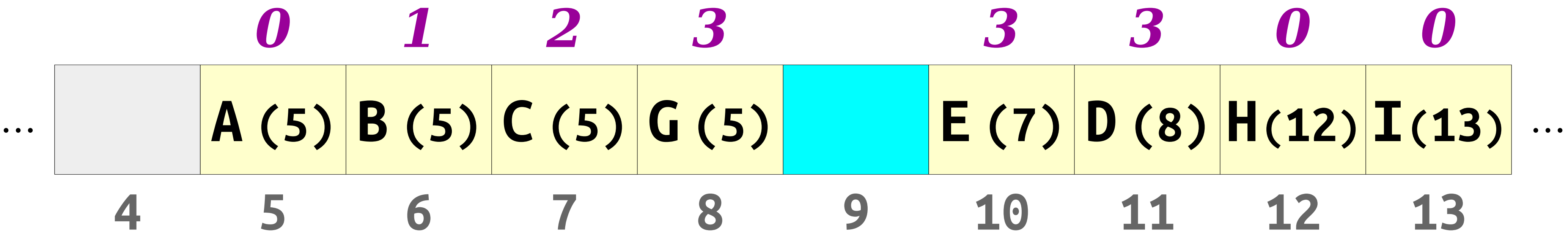
		0	*1*	*2*	*3*	*3*	*3*	*3*	*0*	*0*	
…		A (5)	B (5)	C (5)	G (5)	F (6)	E (7)	D (8)	H(12)	I(13)	…
	4	5	6	7	8	9	10	11	12	13	

- ***Neat trick:*** Robin Hood hashing doesn’t need tombstones.
- We can use a technique called ***backward-shift deletion*** instead.



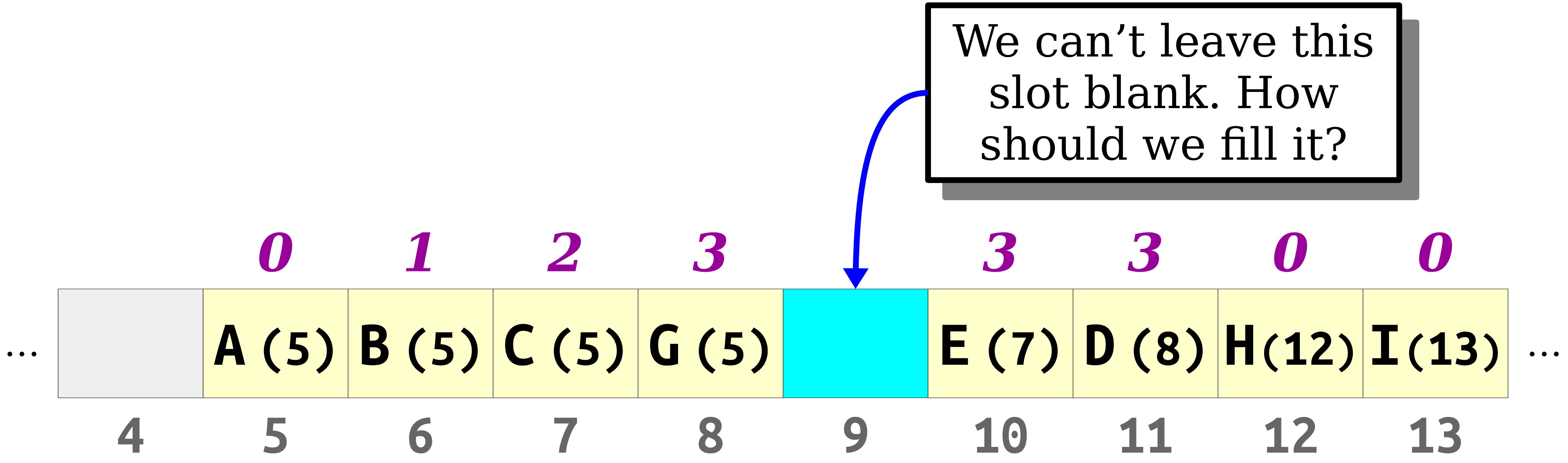
		0	*1*	*2*	*3*	*3*	*3*	*3*	*0*	*0*	
…		A (5)	B (5)	C (5)	G (5)	F (6)	E (7)	D (8)	H(12)	I(13)	…
	4	5	6	7	8	9	10	11	12	13	

- ***Neat trick:*** Robin Hood hashing doesn’t need tombstones.
- We can use a technique called ***backward-shift deletion*** instead.



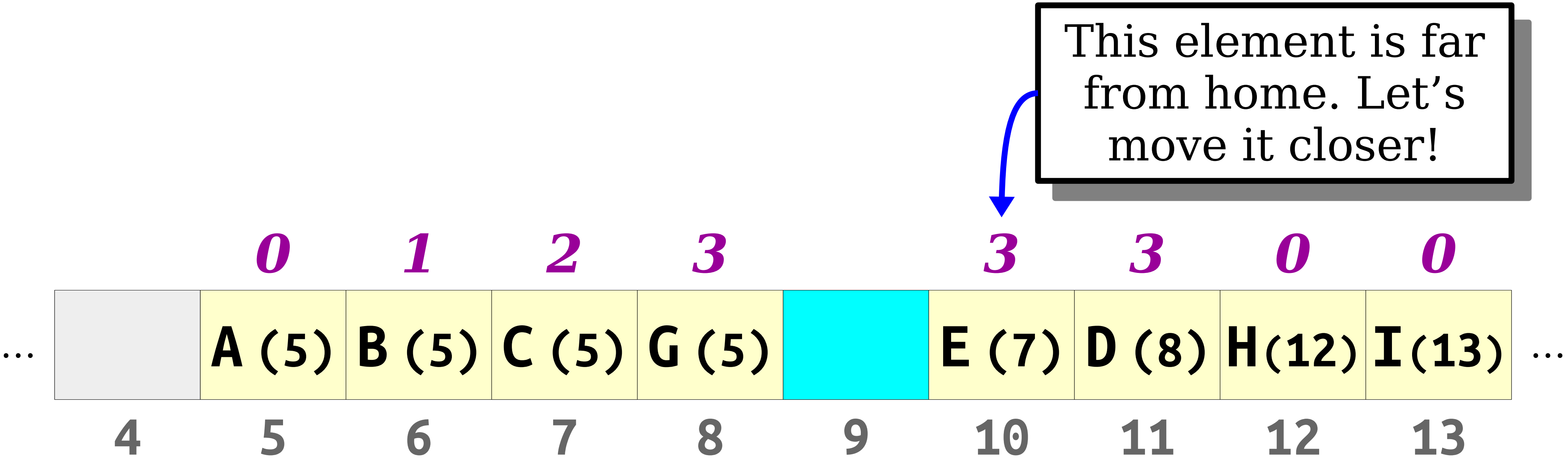
	4	5	6	7	8	9	10	11	12	13	
		0	*1*	*2*	*3*		*3*	*3*	*0*	*0*	
…		A (5)	B (5)	C (5)	G (5)		E (7)	D (8)	H(12)	I(13)	…

- ***Neat trick:*** Robin Hood hashing doesn’t need tombstones.
- We can use a technique called ***backward-shift deletion*** instead.



We can’t leave this slot blank. How should we fill it?

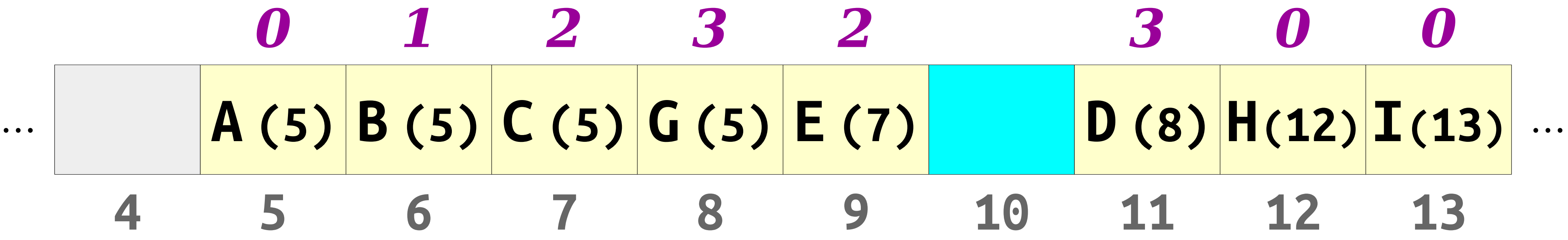
- ***Neat trick:*** Robin Hood hashing doesn't need tombstones.
- We can use a technique called ***backward-shift deletion*** instead.



This element is far from home. Let's move it closer!

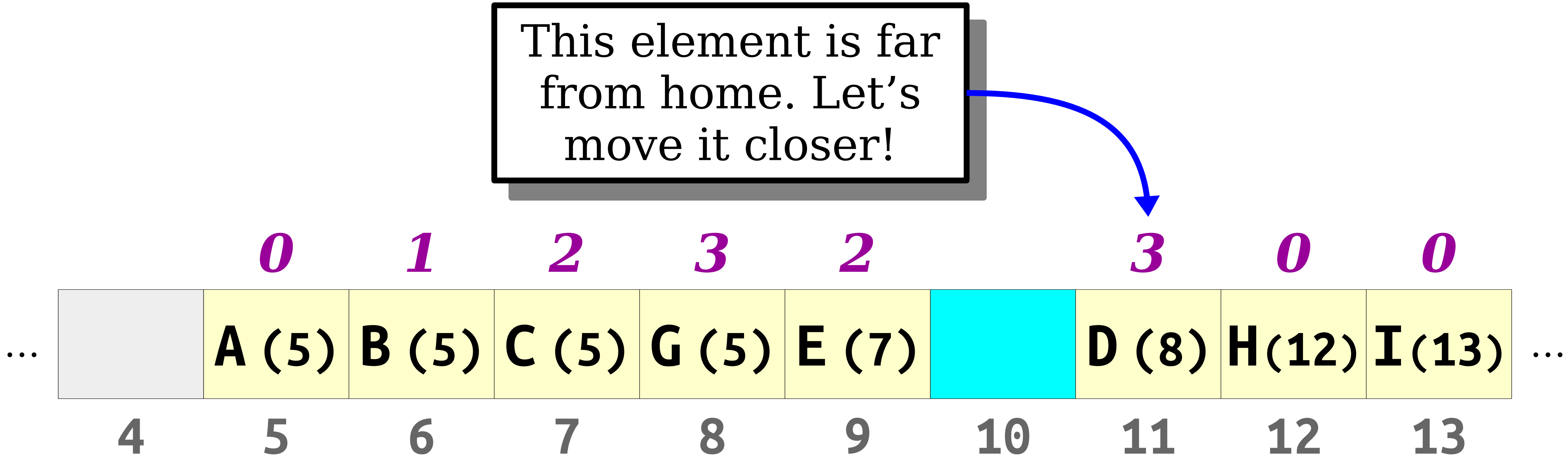
		0	*1*	*2*	*3*		*3*	*3*	*0*	*0*	
…		A (5)	B (5)	C (5)	G (5)		E (7)	D (8)	H(12)	I(13)	…
	4	5	6	7	8	9	10	11	12	13	

- ***Neat trick:*** Robin Hood hashing doesn’t need tombstones.
- We can use a technique called ***backward-shift deletion*** instead.



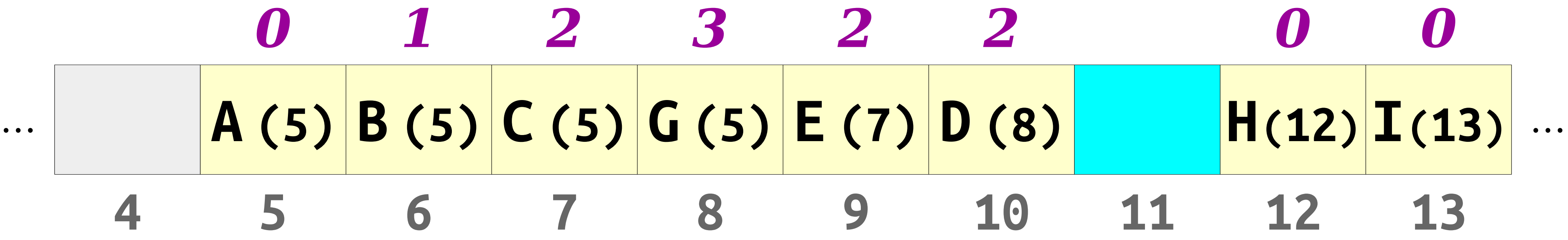
		0	*1*	*2*	*3*	*2*		*3*	*0*	*0*	
…		A (5)	B (5)	C (5)	G (5)	E (7)		D (8)	H(12)	I(13)	…
	4	5	6	7	8	9	10	11	12	13	

- ***Neat trick:*** Robin Hood hashing doesn't need tombstones.
- We can use a technique called ***backward-shift deletion*** instead.



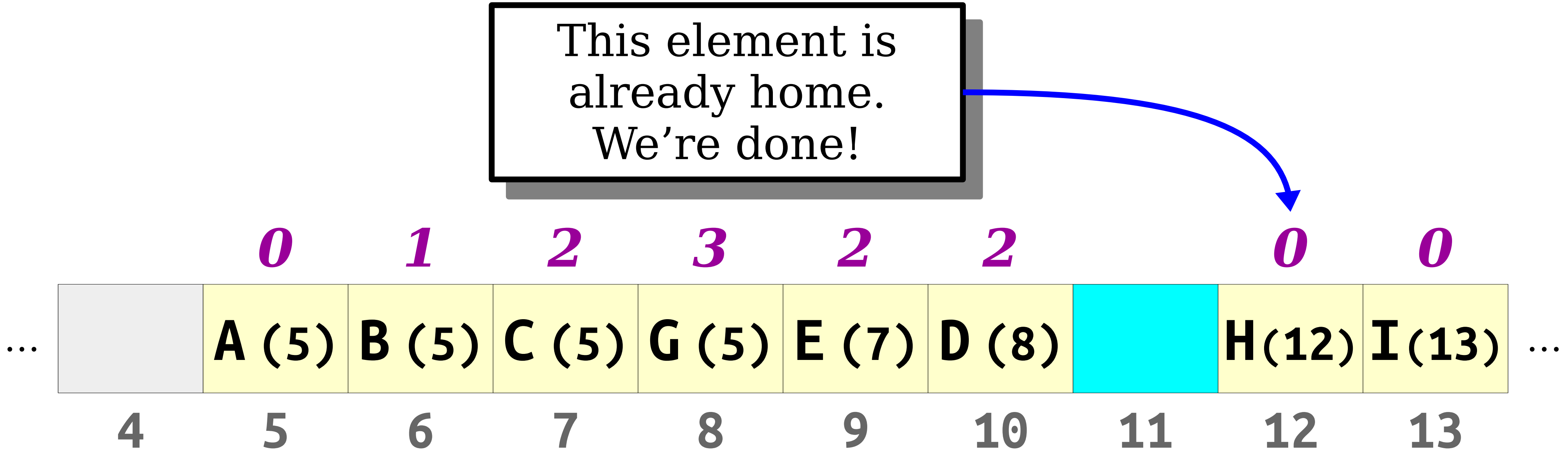
This element is far from home. Let's move it closer!

- ***Neat trick:*** Robin Hood hashing doesn't need tombstones.
- We can use a technique called ***backward-shift deletion*** instead.



		0	*1*	*2*	*3*	*2*	*2*		*0*	*0*	
…		A (5)	B (5)	C (5)	G (5)	E (7)	D (8)		H(12)	I(13)	…
	4	5	6	7	8	9	10	11	12	13	

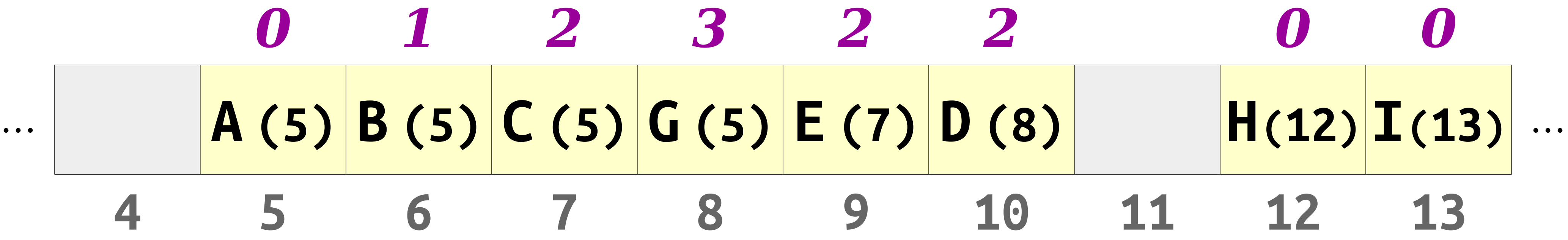
- ***Neat trick:*** Robin Hood hashing doesn’t need tombstones.
- We can use a technique called ***backward-shift deletion*** instead.



This element is already home. We’re done!

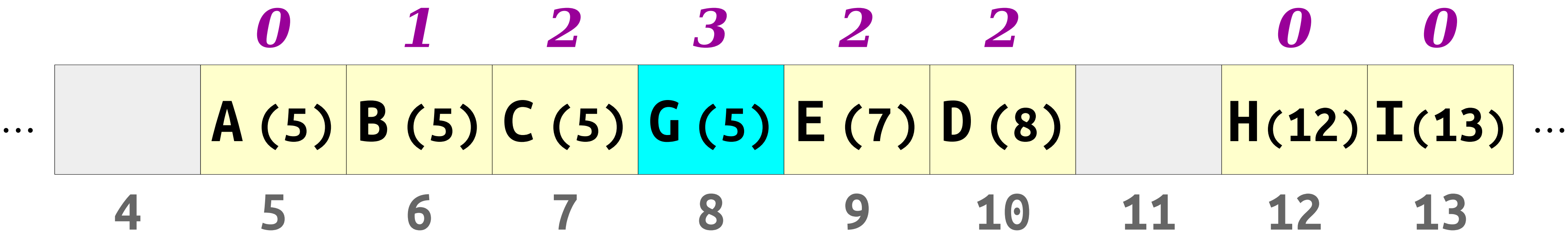
		0	*1*	*2*	*3*	*2*	*2*		*0*	*0*	
…		A (5)	B (5)	C (5)	G (5)	E (7)	D (8)		H(12)	I(13)	…
	4	5	6	7	8	9	10	11	12	13	

- ***Neat trick:*** Robin Hood hashing doesn't need tombstones.
- We can use a technique called ***backward-shift deletion*** instead.

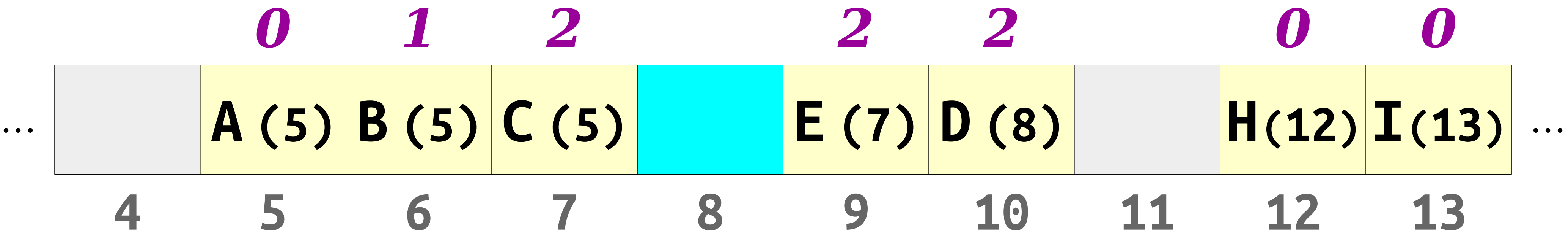


		0	*1*	*2*	*3*	*2*	*2*		*0*	*0*	
…		A (5)	B (5)	C (5)	G (5)	E (7)	D (8)		H(12)	I(13)	…
	4	5	6	7	8	9	10	11	12	13	

- ***Neat trick:*** Robin Hood hashing doesn't need tombstones.
- We can use a technique called ***backward-shift deletion*** instead.

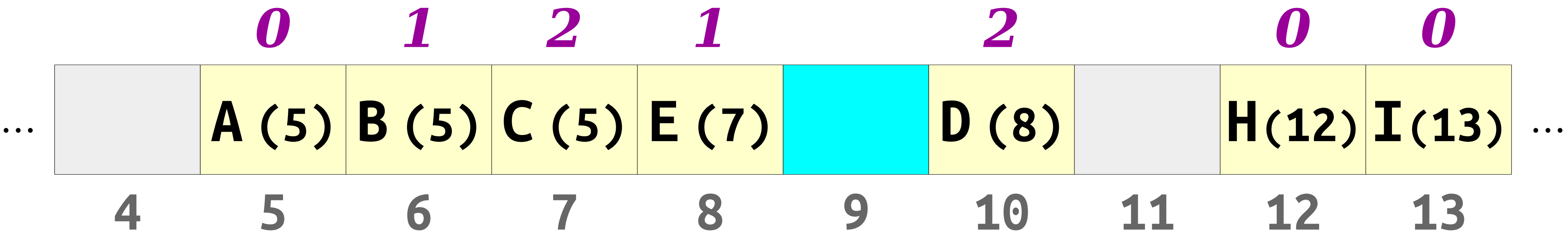


- ***Neat trick:*** Robin Hood hashing doesn't need tombstones.
- We can use a technique called ***backward-shift deletion*** instead.

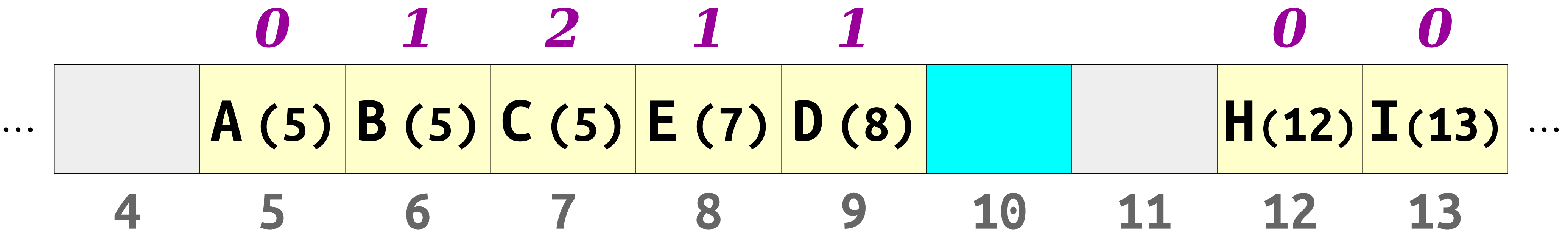


		0	*1*	*2*		*2*	*2*		*0*	*0*	
…		A (5)	B (5)	C (5)		E (7)	D (8)		H(12)	I(13)	…
	4	5	6	7	8	9	10	11	12	13	

- ***Neat trick:*** Robin Hood hashing doesn't need tombstones.
- We can use a technique called ***backward-shift deletion*** instead.

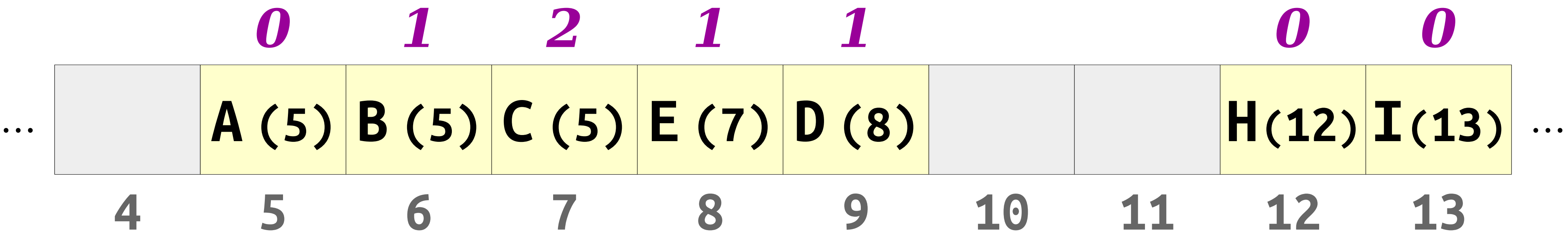


- ***Neat trick:*** Robin Hood hashing doesn't need tombstones.
- We can use a technique called ***backward-shift deletion*** instead.



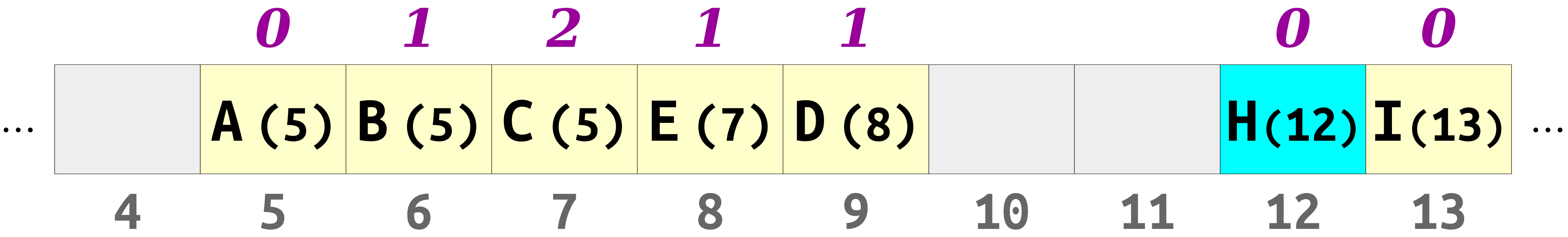
	4	5	6	7	8	9	10	11	12	13	
		0	*1*	*2*	*1*	*1*			*0*	*0*	
…		A (5)	B (5)	C (5)	E (7)	D (8)			H(12)	I(13)	…

- ***Neat trick:*** Robin Hood hashing doesn’t need tombstones.
- We can use a technique called ***backward-shift deletion*** instead.



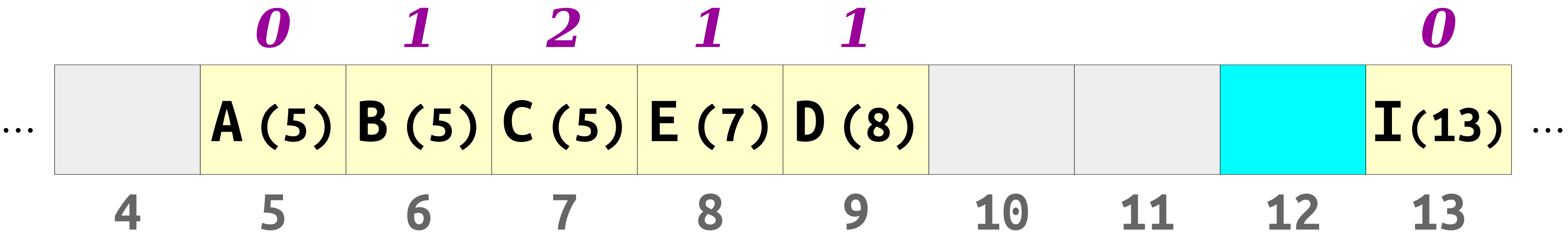
		0	*1*	*2*	*1*	*1*			*0*	*0*	
…		A (5)	B (5)	C (5)	E (7)	D (8)			H(12)	I(13)	…
	4	5	6	7	8	9	10	11	12	13	

- ***Neat trick:*** Robin Hood hashing doesn't need tombstones.
- We can use a technique called ***backward-shift deletion*** instead.

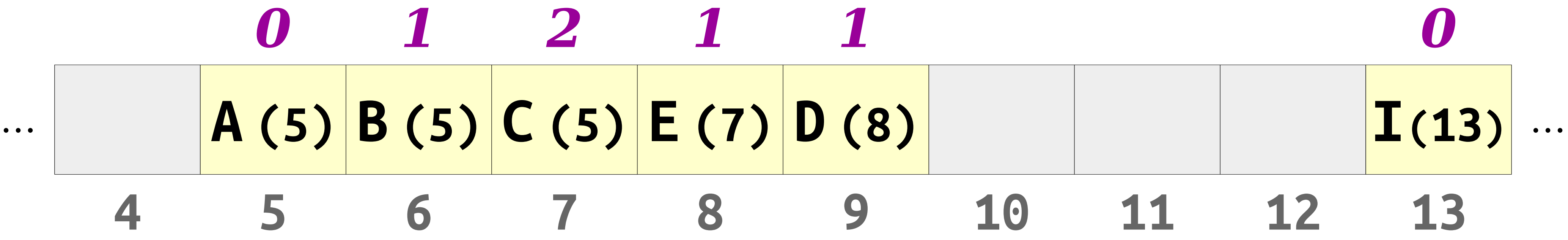


		0	*1*	*2*	*1*	*1*			*0*	*0*	
…		A (5)	B (5)	C (5)	E (7)	D (8)			H(12)	I(13)	…
	4	5	6	7	8	9	10	11	12	13	

- ***Neat trick:*** Robin Hood hashing doesn't need tombstones.
- We can use a technique called ***backward-shift deletion*** instead.



- ***Neat trick:*** Robin Hood hashing doesn't need tombstones.
- We can use a technique called ***backward-shift deletion*** instead.



		0	*1*	*2*	*1*	*1*				*0*	
…		A (5)	B (5)	C (5)	E (7)	D (8)				I (13)	…
	4	5	6	7	8	9	10	11	12	13	

Robin Hood Hashing

- Annotate each table slot with its distance from home.
- During insertions, if the element to insert is further from home than the current table element, “displace” the element in the table and insert that element instead.
- During lookups, stop searching if the element to search for is further from home than the current element.
- During deletions, pull elements backward until we hit an empty slot or find an element that’s already home.

Robin Hood Hashing

- Like linear probing, with a good hash function, the expected cost of a lookup in a Robin Hood hash table is O(1).
- Is Robin Hood hashing worth it?
- How do these approaches compare to chained hashing?
- ***Find out in Assignment 6!***

Your Action Items

- ***Start Assignment 6***
  - You have plenty of time to complete this one if you start early. Aim to be mostly done with Linear Probing by the time we return on Monday.

Next Time

- ***Linked Lists***
  - Chaining things together!
- ***Recursive Data Types***
  - Data types defined in terms of themselves.