Handling Collisions in Dictionaries (Hash Tables)

In computer science, a dictionary (or hash table) is a data structure used to store key-value pairs. Hash tables use a hash function to compute an index into an array of buckets or slots, from which the desired value can be found. However, different keys may hash to the same index - this is called a collision.

What is a Collision?

A collision occurs when two different keys produce the same hash value and thus map to the same index in the hash table. Since a hash table must store both values, this situation must be resolved.

Collision Resolution Strategies:

1. **Chaining (Separate Chaining):**
- Each slot in the table holds a reference to a linked list (or other data structure) of entries that map to the same index.
- When a collision occurs, the new key-value pair is simply added to the list at that index.
- Pros: Simple to implement, efficient when load factor is low.
- Cons: Additional memory overhead, performance can degrade with many collisions.

2. **Open Addressing:**
- When a collision occurs, the algorithm searches for another slot using a probing sequence.
- Variants include:
- **Linear Probing**: Check the next slot (i+1, i+2, ...) until an empty slot is found.
- **Quadratic Probing**: Use a quadratic function to find the next slot (i+1^2, i+2^2, ...).
- **Double Hashing**: Use a second hash function to determine the step size.
- Pros: All data stored in the same array, good cache performance.
- Cons: Clustering can occur, resizing is complex.

3. **Resizing and Rehashing:**
- When the load factor (ratio of elements to size) exceeds a certain threshold, the table is resized

and all keys are rehashed.

- This reduces the number of collisions and maintains efficiency.

4. **Perfect Hashing (for static sets):**

- When the key set is fixed, perfect hashing can guarantee no collisions.
- Requires additional computation during setup but offers constant-time lookups without collisions.

Python Dictionaries:

Python dictionaries handle collisions using **open addressing with probing** internally, and are optimized for average-case O(1) access time. Python's implementation includes clever techniques such as:

- Keeping the load factor below a certain limit.
- Using a compact probing sequence to reduce clustering.
- Storing insertion order (from Python 3.7+).

Conclusion:

Collision handling is a crucial aspect of designing efficient hash-based data structures. The choice of method depends on the specific use case, data distribution, and performance needs. Modern languages and libraries, including Python, abstract these complexities to provide efficient and reliable dictionaries.