**Linear Probing in Hash Tables**

**Overview**:  
Linear probing is an open addressing method for resolving collisions in hash tables, meaning that all records are stored directly in the table itself (rather than using linked lists as in chaining). If a collision occurs (i.e., the target index is already occupied), linear probing looks for the next available index by checking subsequent positions in the table, wrapping around when the end of the table is reached.

**Key Concepts of Linear Probing**

1. **Open Addressing**:  
   Unlike closed addressing (such as chaining), open addressing places all elements directly in the hash table. In linear probing, if the hashed index for a key is occupied, the algorithm checks the next index in the sequence (i.e., index + 1, index + 2, etc.) until an empty slot is found.
2. **Wrap-Around**:  
   The hash table is treated as circular. If we reach the end of the table, probing wraps around to the beginning, continuing the search for an empty spot from index 0.
3. **Clustering**:  
   Linear probing has a tendency to form "clusters" of occupied slots. If collisions frequently occur, the number of adjacent filled slots increases, leading to more collisions (and thus longer probing sequences) when inserting new elements.
4. **Hash Function**:  
   The hash function generates an index for the key, but the key might not be inserted at that index if the spot is already occupied. Instead, the table is probed sequentially until an empty slot is found.

**Basic Operations**

**Insertion**

1. Use the hash function to find the index for the key.
2. If the spot is empty, insert the key there.
3. If the spot is occupied, probe the next index (index + 1), repeating until an empty spot is found.
4. If the end of the table is reached, wrap around to index 0 and continue probing.

**Search**

1. Use the hash function to find the index where the key should be.
2. If the key is not found at that index, probe the subsequent slots.
3. Continue probing until either the key is found or an empty slot is encountered. The search terminates upon finding an empty slot because this indicates that the key is not in the table.

**Removal**

1. Find the key using the search procedure.
2. Once found, remove the key by marking the spot as empty.
3. To maintain the cluster structure and prevent search failures, adjust the positions of any following records that belong to the same cluster. Shift records to fill the empty spot created by the removal to avoid breaking the probing sequence.

**Example of Linear Probing**

Consider a hash table with 10 slots (indices 0 to 9). Let the keys and their corresponding hash indices be:

| **Key** | **Hash Index** |
| --- | --- |
| k1 | 8 |
| k2 | 7 |
| k3 | 9 |
| k4 | 1 |
| k5 | 8 |
| k6 | 9 |
| k7 | 8 |

1. **Insert k1 to k4**:  
   Since there are no collisions, keys k1, k2, k3, and k4 are placed in their respective indices (8, 7, 9, and 1).
2. **Insert k5**:  
   The hash index of k5 is 8, but index 8 is already occupied by k1. Therefore, we probe to the next index, 9, which is also occupied by k3. Continuing the probe, index 0 is empty, so k5 is placed there.
3. **Search for k6**:  
   The hash index of k6 is 9. We start the search at index 9 and continue probing. After checking indices 9, 0, 1, and 2, we find that index 2 is empty, meaning k6 is not in the table.
4. **Search for k5**:  
   The hash index of k5 is 8. We start the search at index 8, but k5 is not found there (it's occupied by k1). Continuing the probe, we check index 9 (k3), and finally index 0, where we find k5.
5. **Remove k3**:  
   After removing k3 from index 9, the cluster from index 7 to 0 is split into two clusters. We check the keys after index 9 and adjust their positions if necessary. In this case, k5 moves from index 0 to fill the spot at index 9.

**Method 2: Tombstoning**

Tombstoning is an alternative removal strategy where, instead of marking a slot as empty after deletion, it is marked as "deleted" to indicate that a record was there but has been removed. The three statuses for a slot are:

* **Empty**: The slot has never been occupied.
* **Used/Occupied**: The slot contains a valid record.
* **Deleted**: A record was here, but it was deleted.

**Insertion with Tombstoning:**

* If a collision occurs, the algorithm probes the next available spot that is either empty or marked as deleted. This allows new elements to be inserted in places where records were previously removed.

**Search with Tombstoning:**

* The search continues past deleted slots, as a deleted slot does not indicate the end of a probe sequence.

**Removal with Tombstoning:**

* Instead of adjusting clusters, a deleted slot simply marks that an item has been removed, and subsequent insertions or searches continue as if the deleted spot is available.

**Other Probing Methods**

**Quadratic Probing**

* Instead of checking the next slot sequentially, quadratic probing uses a quadratic formula (e.g., i2i^2i2) to determine the next index in the probe sequence, spreading records out and reducing clustering.
* For example, if the hash index is 7, the probe sequence might be {7, 8, 1, 6, 3, 2...}.

**Double Hashing**

* Uses two different hash functions. If a collision occurs, the second hash function provides an offset to calculate the next probe index.
* This method is effective in reducing primary clustering and spreading records more uniformly.