Open Addressing Collision Handling technique in Hashing - GeeksforGeeks

Source: https://www.geeksforgeeks.org/open-addressing-collision-handling-technique-in-hashing/

Courses Tutorials Practice Jobs DSA Tutorial Interview Questions Quizzes Must Do Advanced DSA System Design Aptitude Puzzles Interview Corner DSA Python Technical Scripter 2026 Explore DSA Fundamentals Logic Building Problems Analysis of Algorithms Data Structures Array Data Structure String in Data Structure Hashing in Data Structure Linked List Data Structure Stack Data Structure Queue Data Structure Tree Data Structure Graph Data Structure Trie Data Structure Algorithms Searching Algorithms Sorting Algorithms Introduction to Recursion Greedy Algorithms Tutorial Graph Algorithms Dynamic Programming or DP Bitwise Algorithms Advanced Segment Tree Binary Indexed Tree or Fenwick Tree Square Root (Sqrt) Decomposition Algorithm Binary Lifting Geometry Interview Preparation Interview Corner GfG160 Practice Problem GeeksforGeeks Practice - Leading Online Coding Platform Problem of The Day - Develop the Habit of Coding DSA Course 90% Refund Open Addressing Collision Handling technique in Hashing Last Updated : 23 Jul, 2025 Open Addressing is a method for handling collisions. In Open Addressing, all elements are stored in the hash table itself. So at any point, the size of the table must be greater than or equal to the total number of keys (Note that we can increase table size by copying old data if needed). This approach is also known as closed hashing. This entire procedure is based upon probing. We will understand the types of probing ahead: Insert(k): Keep probing until an empty slot is found. Once an empty slot is found, insert k. Search(k): Keep probing until the slot's key doesn't become equal to k or an empty slot is reached. Delete(k) : Delete operation is interesting . If we simply delete a key, then the search may fail. So slots of deleted keys are marked specially as "deleted". The insert can insert an item in a deleted slot, but the search doesn't stop at a deleted slot. Different ways of Open Addressing: 1. Linear Probing: In linear probing, the hash table is searched sequentially that starts from the original location of the hash. If in case the location that we get is already occupied, then we check for the next location. The function used for rehashing is as follows: rehash(key) = (n+1)%table-size. For example, The typical gap between two probes is 1 as seen in the example below: Let hash(x) be the slot index computed using a hash function and S be the table size If slot hash(x) % S is full, then we try (hash(x) + 1) % S If (hash(x) + 1) % S is also full, then we try (hash(x) + 2) % S If (hash(x) + 2) % S is also full, then we try (hash(x) + 3) % S Example: Let us consider a simple hash function as “key mod 5” and a sequence of keys that are to be inserted are 50, 70, 76, 85, 93. Implementation : Please refer Program to implement Hash Table using Open Addressing 2. Quadratic Probing If you observe carefully, then you will understand that the interval between probes will increase proportionally to the hash value. Quadratic probing is a method with the help of which we can solve the problem of clustering that was discussed above. This method is also known as the mid-square method. In this method, we look for the i 2 'th slot in the i th iteration. We always start from the original hash location. If only the location is occupied then we check the other slots. let hash(x) be the slot index computed using hash function. If slot hash(x) % S is full, then we try (hash(x) + 1*1) % S If (hash(x) + 1*1) % S is also full, then we try (hash(x) + 2*2) % S If (hash(x) + 2*2) % S is also full, then we try (hash(x) + 3*3) % S Example: Let us consider table Size = 7, hash function as Hash(x) = x % 7 and collision resolution strategy to be f(i) = i 2 . Insert = 22, 30, and 50. Implementation : Please refer Program for Quadratic Probing in Hashing 3. Double Hashing The intervals that lie between probes are computed by another hash function. Double hashing is a technique that reduces clustering in an optimized way. In this technique, the increments for the probing sequence are computed by using another hash function. We use another hash function hash2(x) and look for the i*hash2(x) slot in the i th rotation. let hash(x) be the slot index computed using hash function. If slot hash(x) % S is full, then we try (hash(x) + 1*hash2(x)) % S If (hash(x) + 1*hash2(x)) % S is also full, then we try (hash(x) + 2*hash2(x)) % S If (hash(x) + 2*hash2(x)) % S is also full, then we try (hash(x) + 3*hash2(x)) % S Example: Insert the keys 27, 43, 692, 72 into the Hash Table of size 7. where first hash-function is h1 (k) = k mod 7 and second hash-function is h2(k) = 1 + (k mod 5) Comparison of the above three: Open addressing is a collision handling technique used in hashing where, when a collision occurs (i.e., when two or more keys map to the same slot), the algorithm looks for another empty slot in the hash table to store the collided key. In linear probing , the algorithm simply

looks for the next available slot in the hash table and places the collided key there. If that slot is also occupied, the algorithm continues searching for the next available slot until an empty slot is found. This process is repeated until all collided keys have been stored. Linear probing has the best cache performance but suffers from clustering. One more advantage of Linear probing is easy to compute. In quadratic probing , the algorithm searches for slots in a more spaced-out manner. When a collision occurs, the algorithm looks for the next slot using an equation that involves the original hash value and a quadratic function. If that slot is also occupied, the algorithm increments the value of the quadratic function and tries again. This process is repeated until an empty slot is found. Quadratic probing lies between the two in terms of cache performance and clustering. In double hashing , the algorithm uses a second hash function to determine the next slot to check when a collision occurs. The algorithm calculates a hash value using the original hash function, then uses the second hash function to calculate an offset. The algorithm then checks the slot that is the sum of the original hash value and the offset. If that slot is occupied, the algorithm increments the offset and tries again. This process is repeated until an empty slot is found. Double hashing has poor cache performance but no clustering. Double hashing requires more computation time as two hash functions need to be computed. The choice of collision handling technique can have a significant impact on the performance of a hash table. Linear probing is simple and fast, but it can lead to clustering (i.e., a situation where keys are stored in long contiguous runs) and can degrade performance. Quadratic probing is more spaced out, but it can also lead to clustering and can result in a situation where some slots are never checked. Double hashing is more complex, but it can lead to more even distribution of keys and can provide better performance in some cases. S.No. Separate Chaining Open Addressing 1. Chaining is Simpler to implement. Open Addressing requires more computation. 2. In chaining, Hash table never fills up, we can always add more elements to chain. In open addressing, table may become full. 3. Chaining is Less sensitive to the hash function or load factors. Open addressing requires extra care to avoid clustering and load factor. 4. Chaining is mostly used when it is unknown how many and how frequently keys may be inserted or deleted. Open addressing is used when the frequency and number of keys is known. 5. Cache performance of chaining is not good as keys are stored using linked list. Open addressing provides better cache performance as everything is stored in the same table. 6. Wastage of Space (Some Parts of hash table in chaining are never used). In Open addressing, a slot can be used even if an input doesn't map to it. 7. Chaining uses extra space for links. No links in Open addressing Note: Cache performance of chaining is not good because when we traverse a Linked List, we are basically jumping from one node to another, all across the computer's memory. For this reason, the CPU cannot cache the nodes which aren't visited yet, this doesn't help us. But with Open Addressing, data isn't spread, so if the CPU detects that a segment of memory is constantly being accessed, it gets cached for quick access. Performance of Open Addressing: Like Chaining, the performance of hashing can be evaluated under the assumption that each key is equally likely to be hashed to any slot of the table (simple uniform hashing) m = Number of slots in the hash table n = Number of keys to be inserted in the hash table Load factor α = n/m (< 1) Expected time to search/insert/delete < 1/(1 - α) So Search, Insert and Delete take (1/(1 - α)) time Related Articles: Hashing | Set 1 (Introduction) Hashing | Set 2 (Separate Chaining) Comment Article Tags: Article Tags: Hash DSA