Separate Chaining Collision Handling Technique in Hashing - GeeksforGeeks

Source: https://www.geeksforgeeks.org/separate-chaining-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 Separate Chaining Collision Handling Technique in Hashing Last Updated : 24 Jul, 2025 Separate Chaining is a collision handling technique . Separate chaining is one of the most popular and commonly used techniques in order to handle collisions. In this article, we will discuss about what is Separate Chain collision handling technique, its advantages, disadvantages, etc. There are mainly two methods to handle collision: Separate Chaining Open Addressing In this article, only separate chaining is discussed. We will be discussing Open addressing in the next post Separate Chaining: The idea behind separate chaining is to implement the array as a linked list called a chain. Linked List (or a Dynamic Sized Array) is used to implement this technique. So what happens is, when multiple elements are hashed into the same slot index, then these elements are inserted into a singly-linked list which is known as a chain. Here, all those elements that hash into the same slot index are inserted into a linked list. Now, we can use a key K to search in the linked list by just linearly traversing. If the intrinsic key for any entry is equal to K then it means that we have found our entry. If we have reached the end of the linked list and yet we haven't found our entry then it means that the entry does not exist. Hence, the conclusion is that in separate chaining, if two different elements have the same hash value then we store both the elements in the same linked list one after the other. Example: Let us consider a simple hash function as " key mod 5 " and a sequence of keys as 12, 22, 15, 25 Implementation Please refer Program for hashing with chaining for implementation. Advantages: Simple to implement. Hash table never fills up, we can always add more elements to the chain. Less sensitive to the hash function or load factors. It is mostly used when it is unknown how many and how frequently keys may be inserted or deleted. Disadvantages: The cache performance of chaining is not good as keys are stored using a linked list. Open addressing provides better cache performance as everything is stored in the same table. Wastage of Space (Some Parts of the hash table are never used) If the chain becomes long, then search time can become O(n) in the worst case Uses extra space for links Performance of Chaining: 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 hash table n = Number of keys to be inserted in hash table Load factor α = n/m Expected time to search = O(1 + α) Expected time to delete = O(1 + α) Time to insert = O(1) Time complexity of search insert and delete is O(1) if α is O(1) Data Structures For Storing Chains: Below are different options to create chains. The advantage of linked list implementation is insert is O(1) in the worst case. The advantage of array is cache friendliness, but the insert operation can be O(1) in cases when we have to resize the array. The advantage of Self Balancing BST is the worst case is bounded by O(Log (len)) for all operations 1. Linked lists Search: O(len) where len = length of chain Delete: O(len) Insert: O(1) Not cache friendly 2. Dynamic Sized Arrays (Vectors in C++, ArrayList in Java, list in Python) Search: O(len) where len = length of chain Delete: O(len) Insert: O(1) Cache friendly 3. Self Balancing BST (AVL Trees, Red-Black Trees) Search: O(log(len)) where len = length of chain Delete: O(log(len)) Insert: O(log(len)) Not cache friendly Java 8 onward versions use this for HashMap Related Posts: Open Addressing for Collision Handling Hashing | Set 1 (Introduction) Comment Article Tags: Article Tags: Hash DSA