ECE 250
Project 2 - Hashing
May Toyingsirikul, student ID: mtoyings

Overview of Classes

Class 1: StudentRecords (both method)
Description: A class that has variable student number and student last name for storing data in a hash table.
Member Variables:

Public Variables:

`studentNumber`: store student number field of the given data
`studentName`: store student last name of the given data

Class 2: Node (only chaining)
Description: A node class where each node stores the address of the data, and data is stored of type studentRecords. Each node contains reference to the next node.
Member Variables:

Public Variables:

`value`: the value stored in the node as type `studentRecords`
`next`: stores the next pointer node of the current node

Class 3: LinkedList (only chaining)
Description: Generic singly linked list class with some basic operations of linked list, where each element is stored of type `Node` class. It provides reference to the head node of the linked list, and contains functions as follows: insert a node at the beginning of the list, remove a node at the given key, and find an element in the list from the given key. It has `hashTable` class as a friend class.
Member Variables:

Public Variables:

`p_head`: a pointer for the first element of the list of a Node type

Member Functions:

`insert_front`: insert a node to the front of the linked list
`extract`: remove the a node from given the given key (student number) from the linked list
`isFound`: find element from a given key in the hash table if it is not empty

Class 4: hashTable (both methods)
Description: A hash table implemented using vectors with given capacity, entries are stored as type `studentRecords` for double hashing and `LinkedList` for chaining. Contains functions as follows: create a hash table of given capacity, insert data to the table using appropriate open addressing method (chaining or double hashing), search whether the given data if the given key is in the table, remove element of given key from hash table, and print all the keys (student number) within the given index (only for chaining method).
Member Variables:

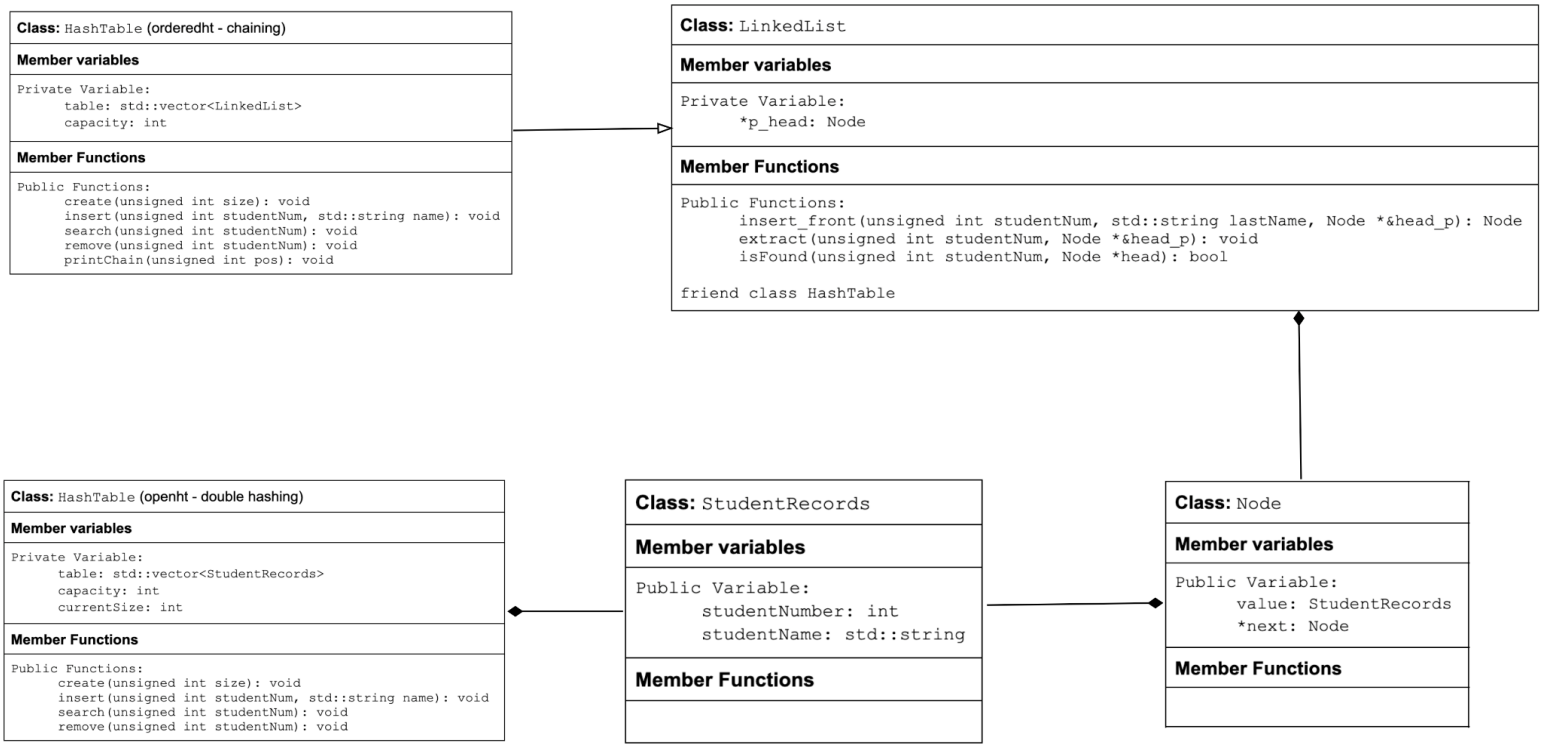
Private Variables:

`table`: a vector that stores entries of the hash table
`capacity`: maximum capacity of index in the hash table
`currentSize`: current size of the hash map (only for double hashing)

Member Functions:

`create`: set the maximum capacity of the hash table
`insert`: insert a node (student record) to the hash table using appropriate open addressing method
`search`: traverse through the hash table (for double hashing) or linked list at the given index (for chaining), and find whether the key is in the list when the list if not empty
`remove`: remove the given key (and value) from the hash table if in the table and table is not empty
`printChain:` if the list is not empty, print all entries in linked list of the given index from from front to back (only for chaining method of open addressing)

UML Class Diagram



Design Decision Details

Class 1: StudentRecords

Constructors: initialise variable `studentNumber` at `0`, and `studentName` at "”

Class 2: Node

Constructors: initialise node `next` as `nullptr`, and `value` at initial value of class `StudentRecords`

Destructors: set `next` Node to `nullptr`

Class 3: LinkedList

Constructors: initialise `p_head` pointer as `nullptr`

Destructors: traverse the list and delete each element in it, then delete `p_head` node

Const Parameters: All parameters, except the Node parameter, of the function `insert_front`, `extract`, and `isFound` can be constant since no manipulation is made to the value.

Class 4: HashTable

Constructors: default constructor

Destructors: traverse through every index in the hash table, and traverse through the linked list of that index to delete all the entries, then delete `p_head` node of the index linked list.

Const Parameters: Every parameter of all the functions could be constant since it does not need to be manipulated. The functions are `create`, `insert`, `search`, `remove`, `printChain`.

Test Cases

- Provided Test Cases

Edge cases insertion:

- Inserting key that does not exist with last name that already exist, should return success
- Inserting key and last name that already exist, should return failure

Edge cases search:

- Search key in the linked list that is empty, return failure (chaining)
- Search key in the hash table that is empty, return failure (double hashing)

Edge cases deletion:

- Delete given key from a hash table that is empty, return failure (both)
- Delete the key that was previously deleted, return failure (both)
- Attempt to delete given key from the list that is empty, return failure (chaining)

Edge cases print:

- Print linked list of the index beyond the capacity, return chain is empty

Performance Considerations

For `create`, both methods have **O(1)** time complexity because creating a hash table is resizing the size of the vector to the given size

Calculating hash functions takes a constant. For double hashing, checking size and checking whether the key is already in the table take a constant. Hence `insert` for double hashing is a constant **O(1)**. Let m be the capacity of the hash table, and n be the number of entries inserted to the hash table. Where n/m is the expected length of the linked list, given that the hash function is uniformly distributed. Traversing through a linked list of that index takes O(n/m), but n/m is considered a constant. Hence, `insert` for chaining takes **O(1)** time complexity.

Similarly, `search` and `delete` for double hashing and chaining takes a constant time **O(1)**. Removing for chaining is traversing the linked list (O(1)) and deleting the node if found (O(1)). Removing for double hashing is assigning “0” to the slot of that key (if exist)

`print`, like other functions for chaining, is **O(1)** since it is traversing through a linked list of the given index and print.

Reference

https://programming.guide/hash-tables-complexity.html
https://iq.opengenus.org/time-complexity-of-hash-table/
https://www.educative.io/answers/singly-linked-list-in-cpp
https://www.geeksforgeeks.org/c-program-hashing-chaining/
https://www.tutorialspoint.com/ceil-and-floor-functions-in-cplusplus