ECE 250 Project 2: Hashing

Name: Kyle Lee
Student ID: 20892255
Date: Nov 7, 2022

1. Overview of Classes

a. Class Student

This class is composed of two private member variables called number and name that store the student number and student's last name, respectively. HashTable classes use this class to store student information. Once it is initialized, you can't alter the value of the number or name since there are no setters.

b. Template Class HashTable

This class represents a generic hash table that has a user-specified maximum capacity. It is designed as a template class to store the data of any data type. By implementing this class as a template class, it can be a parent class of both OpenHashTable and OrderedHashTable. This class is composed of basic member variables and functions that both OpenHashTable and OrderedHashTable have in common. The bucket of HashTable is an array of a template class. Since the size of the bucket never changes throughout the execution, I decided to implement it using a dynamically allocated array.

Member variables/functions:

- # size (dtype: int): Stores the number of entries in the bucket
- # max_size (dtype: int): Stores the maximum size of the bucket
- # array (dtype: T[ ]): Acts as a bucket of the table; stores data of type T
- # int hash_1 (unsigned int k): Primary hash function; returns (k mod max_size)

c. Class OpenHashTable

This class represents a hash table that resolves collisions using a technique called open addressing using double hashing. It is implemented by inheriting from the HashTable class. Since the data stored in this class is a Student class, its mother class is HashTable<Student>.

Member variables/functions:

- **+** void insert (unsigned int student_id, const string student_name):
  1. Check if the table is full; if it is full, print out "failure"
  2. Using hash functions ((h1(k) + i.h2(k)) mod max_size), find the appropriate index to insert the key
  3. If the key is already in the table, print out "failure"
  4. Else, insert the key and value and print out "success"
- **+** void search (unsigned int student_id):
  1. Using hash functions, search for the key in the table
  2. If the key was found in the position p of the table, print out "found LN in p"
  3. Else, print out "not found"
- **+** void remove (unsigned int student_id):
  1. Using hash functions, search for the key in the table
  2. If the key was found in the position p of the table, remove the key from the array and print out "success"
  3. Else, print out "failure"
- **-** int find (unsigned int student_id):
  1. Helper function for functions search and remove
  2. Using hash functions ((h1(k) + i.h2(k)) mod max_size), find the key in the table
  3. If the key was found in the positions p of the table, return p
  4. Else, return -1
- **-** int hash_2 (unsigned int k): Secondary hash function; returns ($\left\lfloor \frac{k}{\text{max\_size}} \right\rfloor$ mod max_size)