Raymond Lin

CS32 - Data Structures and Algorithms

Smallberg

30 January, 2018

Project 2 Report

1. I implemented a very simple doubly linked list. It did not have a dummy node, a tail pointer, and it was not circular. Nodes were named after the structure I created to store key-value pairs, KeyValuePair. Each KeyValuePair consisted of a key, a value, a pointer to the next item on the list called ‘next’, and a pointer to the previous item on the list called ‘prev’. The list also used a head pointer that points to the first item in the array, instead of a dummy node. The list was in no particular order, and the insert function would add to the end of the linked list.
2. bool Map::insert(const KeyType& key, const ValueType& value)

if no elements in linked list

Create new KeyValuePair and set appropriate pointers  
 Increment numElements and return true

else

Traverse through the list

if the key is already in the list, return false

if key is not in the list, create a new KeyValuePair and add it to end of list

bool Map::insertOrUpdate(const KeyType& key, const ValueType& value)  
 If there are no elements in the linked list, create a new one and set appropriate pointers. Then return true.  
 If key is in the linked list, change its value and return true.

If key is not in the linked list, create a new node, set appropriate pointers and return true.  
  
bool Map::erase(const KeyType& key)  
 Check to see if there are no elements in the list. If this is the case, return false.

If target element is in the first node, delete it and set appropriate pointers.

If target element is not in first node, find the node by traversing through list.

If found, break.

Delete the node and set appropriate pointers, and return true;

Else return false.

bool Map::update(const KeyType& key, const ValueType& value)

Traverse through the list to find the key.

If found, update the value and return true.

If not found return false.

bool Map::contains(const KeyType& key) const

Traverse through the list to find the key

If found, return true.

If not found, return false.

bool Map::get(const KeyType& key, ValueType& value) const

Traverse the list to find the key.

If found, change the value parameter and return true.

If not found return false.

bool Map::get(int i, KeyType& key, ValueType& value) const

Check to see if i is a valid index.

If not, return false.

Traverse the list to find the node at index i.

If found, change the key and value parameters and return true.

void Map::swap(Map& other)

swap the two private data members of Map class: head and numElements

Map::~Map()

Traverse the list and delete each node when reached.

Map::Map(const Map &other)

If other map is empty, head is nullptr

If not empty, head points to a new node that has key and value same as other head

Traverse the other list and this list so that this list constructs a new node that is the same as the corresponding node in other list

Map &Map::operator=(const Map &other)

Check if other’s address is the same as this

If it is, return a pointer to this

If other map is empty, head is nullptr

If not empty, head points to a new node that has key and value same as other head

Traverse the other list and this list so that this list constructs a new node that is the same as the corresponding node in other list

Return a pointer to this.

bool combine(const Map& m1, const Map& m2, Map& result)

Make a temp map of result and destruct it.

If m1 is empty, result is m2

If m2 is also empty, tempmap does not change.

Return true.

If m2 is empty, result is m1. Return true.

Else traverse m1 and add everything to tempmap

Loop through m2

if tempmap already contains the key and the key’s value is different from the other key’s value, erase it from tempmap. Set the returnValue to false.

If tempmap does not contain the key, insert it into tempmap.

swap tempmap with result and return the returnValue.

void subtract(const Map& m1, const Map& m2, Map& result)

Create a tempmap and assign m1 to it.

Loop through m2 and if found found any keys that also appear in m1, erase them

Swap tempmap and result.

1. Map m;

assert(!m.erase("abc")); //test false case of erase

assert(m.insert("abc", 123)); //test insert

assert(!m.insert("abc", 567)); //test false case of insert

assert(m.erase("abc") && m.size() == 0 && m.empty()); //test empty and size function

assert(m.insert("abc", 123) && m.insert("bcd", 321));

assert(m.contains("abc")); //test contains

assert(m.contains("bcd")); //make sure insert works right

assert(!m.contains("xyz")); //test false case of contains

ValueType v = 999;

assert(m.update("abc", v)); //test update

assert(!m.update("xyz", v)); //test false case of update

assert(m.get("bcd", v) && v == 321); //test get

assert(!m.get("xyz", v) && v == 321); //test false case of get

KeyType k = "";

assert(m.get(1, k, v) && k == "bcd" && v == 321); //test get (3 param)

assert(!m.get(2, k, v) && k == "bcd" && v == 321); //test false case of get (3 para)

Map m2;

assert(m2.insertOrUpdate("xyz", 456)); //test insertOrUpdate

m.swap(m2);

assert(m.contains("xyz") && m2.contains("abc") && m2.contains("bcd") && m.size() == 1 && m2.size() == 2); //test swap

Map m3 = m2;

assert(m3.size() == m2.size() && m3.contains("abc")); //test copy constructor

m = m2;

assert(m.size() == m2.size() && m.contains("abc") && m.contains("bcd")); //test assignment operator

m2.~Map();

m.~Map();

assert(m2.size() == 0 && m.size() == 0); //test destructor

Map test1;

Map test2;

Map result;

assert(test1.insert("abc", 123) && test1.insert("bcd", 234));

assert(test2.insert("abc", 123) && test2.insert("xyz", 890));

assert(combine(test1, test2, result) && result.size() == 3); //test combine

assert(test2.insert("bcd", 432) && test2.insert("ost", 000));

assert(!combine(test1, test2, result) && result.size() == 3); //test false case of combine

Map test3;

assert(combine(test3, test2, result) && result.size() == 4); //test empty case of combine

subtract(test2, test1, result);

assert(result.size() == 2); //test subtract

subtract(test2, test3, result);

assert(result.size() == 4); //test empty case of subtract